



HAL
open science

Versatile machine learning for neurodevelopmental imaging

Seongbin Lim

► **To cite this version:**

Seongbin Lim. Versatile machine learning for neurodevelopmental imaging. Artificial Intelligence [cs.AI]. Institut Polytechnique de Paris, 2023. English. NNT : 2023IPPAX006 . tel-04482954

HAL Id: tel-04482954

<https://theses.hal.science/tel-04482954v1>

Submitted on 28 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Versatile machine learning for neurodevelopmental imaging

Thèse de doctorat de l'Institut polytechnique de Paris
préparée à École polytechnique

École doctorale n°626 : École doctorale
de l'Institut polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 13 octobre 2023, par

Seongbin LIM

Composition du Jury :

Elsa ANGELINI Professeure associée, Telecom Paris Palaiseau	Présidente du jury
Thomas WALTER Professeur, Mines Paris Paris	Rapporteur
David ROUSSEAU Professeur, Université d'Angers Angers	Rapporteur
Thierry PÉCOT Ingénieur de recherche, Université de Rennes 1 Rennes	Examineur
Emmanuel BEAUREPAIRE Directeur de recherche, École Polytechnique Palaiseau	Co-directeur de thèse
Anatole CHESSEL Professeur assistant, École Polytechnique Palaiseau	Co-directeur de thèse

CONTENTS

Preface	1
Acknowledgement	3
Résumé de la thèse en français	5
Abbreviations and Symbols	7
1 Introduction	9
1 Microscopy and Bioimage Data	11
1.1 Neurodevelopmental Imaging Techniques	11
1.2 Bioimage Informatics and Computer Vision	17
1.3 Deep Learning Approaches for Bioimages	20
1.4 Limitations and Challenges of Current Solutions	29
2 General Trends of Computer Vision Solutions	34
2.1 Supervised Learning Dominates	34
2.2 Unsupervised Learning is Pursued	35
2.3 Self-supervised Learning Arrives	37
2.4 Self-supervised Learning in Computer vision	38
3 Self-supervised Learning in This Thesis	43
3.1 Generative Model	43
3.2 Representation Learning	47
4 Objectives of The Thesis	53
2 Supervised learning and Versatility	57
1 How powerful and versatile SOTA supervised models?	58
1.1 Existing solutions	59
1.2 Why they failed and What to do next	60
1.3 Supervised learning in practice	60
2 Bioimagerloader: facilitate machine learning for bioimages	64
2.1 Why bioimagerloader?	65
2.2 Overview	65
2.3 Technical details	71
3 Versatile supervised learning	75
3.1 Clustering	75
3.2 Versatile supervised instance segmentation models	80
4 Conclusions	87
5 Discussions and Perspectives	87

3	NU-Net: Self-supervised versatile CNN filter	89
1	Introduction	90
1.1	Background	90
1.2	Related works	92
2	Methods	94
2.1	Perceptual loss	94
2.2	Morphological loss	95
2.3	Data	96
2.4	Training	98
2.5	Early NU-NETS	100
3	Results	103
3.1	Perceiving styles	103
3.2	Loss curves	104
3.3	Contrast enhancement	104
3.4	Controlling filtering magnitude	106
3.5	Side effect: Long objects	107
3.6	Application: Napari plugin	108
3.7	Artifacts and Limitations	110
3.8	More figures	111
4	Discussions and Perspectives	113
5	Conclusions	118
4	Applications	119
1	Segmenting fibroblasts on human skins	119
2	Workflow integration	123
2.1	Napari plugin: Proofreading ChromS Brainbow	123
2.2	Napari plugin: BigAnnotator	125
3	Segmentation in ChromS's pipeline	128
5	Conclusions	131
1	General conclusions and Perspectives	131
2	Wider Perspectives	133
	About me	139
A	Short Introduction to Machine Learning	141
1	Machine learning with Examples	141
1.1	Population: Regression	142
1.2	Checkers: Memorization vs. Learning	143
1.3	Chatbot: Natural language processing and Self-supervised learning	144
1.4	Handwritten digits: Computer vision and Convolutional layer	146
2	Generative model	151
2.1	Probability Density Distribution	151
2.2	Generation Process	152
2.3	Distance between Distributions	152
3	Artificial neural network	153
3.1	AI or AGI? Call it simply Deep learning	153
3.2	Perceptron	154
3.3	Optimization and Training	156

4	Deep learning in Computer vision	165
4.1	Deep neural network and Deep learning	165
4.2	Deep convolutional neural networks	166
B	List of Datasets	173
C	Metric	177
D	[Publication] NU-Net	181
E	[Publication] bioimagerloader	193

PREFACE

I targeted my readers to be not only computer scientists but also scientists in other domains as well as whomever simply want to know about the topic. I made this decision mainly for two reasons. (i) Deep learning, which is the main interest of this thesis, is fairly new and moving so fast. What used to be the state of the arts when I started my PhD are hardly competitive to the current ones, and new papers have been pouring out every moment. I thought that it was imperative to review how deep learning have evolved to talk about it. (ii) I have mostly worked with non-computer scientists and found that my colleagues had hard time understanding my words during my PhD thesis. After all, I worked as a computer scientist in the advanced microscopy team of Laboratoire d'Optique et Biosciences (LOB; laboratory for optics and biosciences), where my colleagues were microscopists, microscope engineers, physicists, biologists, chemical biologists, and those who were not necessarily familiar with computer science nor computer engineering. I realized that this decision should also benefit friends of mine who kindly showed their interest in my work, and my family who are mostly not scientists. Consequently, I spent a fair amount of time in the appendix to give background knowledge of machine learning as well as deep learning and to walk you through with examples, which some of you might think too basic. Additionally, I sometimes used the first-person narrative when I expressed my own opinions, in discussion with my supervisors, which is not typical in scientific writing but makes texts more approachable to wider audiences. For those who have prior knowledge about the topic, I hope that this appendix would be a refresh and tell you my perspective for the topic, so that you can better understand in a plain language where my ideas come from and thus my work. I tried my best to clarify definitions of each terminology I used, to put as many figures and graphics as possible, to explain equations with concrete examples, and to use plain and simple languages. You will also find many footnotes to add more explanation or simply to give interesting facts. I must say that I enjoyed the process. Yet, despite all my efforts, you may find this book still boring and too long, because, after all, it is a PhD thesis, a three-year-long scientific diary (and sorry, it is not a science fiction). Last but not least, as a disclaimer, this book lies in the domain of computer science and machine learning engineering. Thus, it covers mainly machine learning and computational infrastructure of microscopes for neurodevelopmental imaging, but not much of neurodevelopmental imaging techniques in details.

CHAPTER OVERVIEW

This thesis has 5 chapters. They are organized as followed:

Chapter 1 is an introduction. It covers computer vision and machine learning, and reviews current states in general. Then, self-supervised learning follows as an emerging trend as well as the main topic of the manuscript. Lastly, going more specific, I state challenges that bioimages have been facing in computer vision with machine learning technology and set problems.

Chapter 2 is about seeking versatility in machine learning using supervised learning in the context of bioimages, in particular for segmenting instances of cells and nuclei. I introduce *bioimageloader*, a programming library, to address many practical issues that bioimages have been facing and to facilitate machine learning researches.

Chapter 3 is the main focus of this thesis, where I explored self-supervised learning for more versatility. The chapter is dedicated to a novel self-supervised learning method and the resulted model, called *NU-Net* which is a versatile filter for cells or nuclei. I will describe its ideas, and show trials and failures, successes, and its potentials.

Chapter 4 covers other projects that I was involved in. They were helping analysis of assessing skin damage to UV light with image segmentation, developing a tool to make it easy to proofread automatic segmentation of brain cells, and creating an interface to facilitate and accelerate segmentation annotation.

Chapter 5 summarizes and concludes my works. Additionally, it opens discussion for potentials of modern machine learning technologies and their concerns.

Reach out to me As a reader of my thesis, you are always welcome to reach out to me. I really appreciate that you picked it up. If you have any questions, just want to have a chat, or say a word, please write me an email or drop me a message through social media. I also have a homepage where I occasionally make updates.

- Email: sungbin246@gmail.com
- GitHub: <https://github.com/sbinnee>
- LinkedIn: <https://www.linkedin.com/in/seongbin-lim-696666159>
- Twitter: [@TheSbinnee](https://twitter.com/TheSbinnee)
- Homepage: <https://sbinnee.github.io/>

ACKNOWLEDGEMENT

I thank my two supervisors, Emmanuel Beaurepaire and Anatole Chessel. Emmanuel gave me wider views outside computer science while I talked a lot about computers with Anatole. I especially thank Anatole not only because he helped me a lot to progress my projects but also because he was patient, listened to me and respected my opinions. I thank Lien Ung and Chiara Stringari for their collaboration that resulted in my first publication during my thesis. I thank Hugo Blanc for giving access to the ChroMS data, and Josephine Morizet for the THG data. I thank collaborators Jean Livet and Gabriel Kaddour in Institut de la Vision and Ignacio Arganda-Carreras for working together on the ChroMS project.

I supervised three interns from Master's programs with Anatole and thank each of them. Slimane Baamara worked on a plugin software **bigannotator** to improve an annotation pipeline. Xingjian Zhang contributed to **bioimagerloader** by implementing new collections and improving documentation. Tanguy Rolland helped me to develop a plugin of NU-NET for Napari viewer and also contributed to analyzing collections of **bioimagerloader**. I cannot thank enough my colleagues who welcomed me and spent time together in and out of the laboratory. I specially thank my lunch gang: Anastasia, Júlia, Sophie, Maëlle, and Robin. Food has become such an important part of my life ever since I became vegan, and they were with me for every lunch. It was my favorite moment of the day. Particularly, Ana, Júlia, and Sophie literally fed me during my writing period of this very thesis when I had no time to cook. I appreciate all the helps so much that kept me alive and helped me go through hard time. For that matter, I also thank Chiara for introducing me a bowl of awesome Italian soup.

I spent so much time with Ana. I enjoyed time with her a lot, drinking teas, sharing series, introducing new recipes and ingredients, and hanging out in Paris. Also, it was my pleasure to get to know sweet and amazing Ada the dog. I thank the hiking gang: Arthur, Maëlle, and Clément. I will not forget our experience in Normandie. Arthur made me a lot of good memories: invited me to Bretagne, played board games a lot together, and invited me many times to his place for barbecues. I wish I had more time with Clément both for working and having fun. With Karsten and Arthur, I talked about and played guitars.

I thank both my Korean and Slovak family for supporting my study abroad. More than anyone, I thank Janka, my madam baby, my ex-girlfriend and now my partner and wife. Without her support, I may not have had decided to come to France and could not have had this amazing experience. I truly appreciate her devotion that let me start this journey in the first place. Even though we were in distance for the most period of my PhD, I relied on her every time I had hard time.

RÉSUMÉ DE LA THÈSE EN FRANÇAIS

Le présent travail de thèse s'intéresse au développement et l'application de modèles polyvalents d'apprentissage automatique pour l'imagerie neurodéveloppementale.

Dans l'introduction sont exposé les types de données abordés, à savoir les bioimages, caractérisées par leur grande taille et leur résolution allant jusqu'aux noyaux cellulaires et présenté les techniques de microscopie permettant d'acquérir de telles données. La bioinformatique des bioimages et la vision par ordinateur dans ce contexte sont ensuite abordées, soulignant les tâches telles que le stockage, le transfert, la visualisation et la préparation des données et la segmentation d'image, une tâche cruciale de vision par ordinateur qui est au cœur de cette thèse.

Sont discuté ensuite les avantages des approches d'apprentissage profond pour l'analyse des bioimages. Leur généralisabilité sans précédent dans les tâches de vision par ordinateur ont conduit à la création de nombreux modèles et ensembles de données pour segmenter, restaurer ou débruiter les bioimages, soulignant également l'importance des outils logiciels pour distribuer ces modèles. Les limitations actuelles des solutions d'apprentissage profond pour les bioimages sont ensuite discutées, principalement liées à l'apprentissage supervisé nécessitant des ensembles de données annoté entièrement manuellement. Cela a motivé l'exploration, dans la lignée d'une tendances générales actuel en vision par ordinateur, d'un passage de l'apprentissage supervisé à d'autres schémas tels que l'apprentissage non- ou auto- supervisé.

L'apprentissage auto-supervisé est d'abord apparu dans le domaine du traitement du langage naturel. Malgré les défis d'application directe de l'auto-attention (self attention) aux données image, d'autres méthodes basées sur des idées similaires ont été développées, telles que l'apprentissage par contraste (contrastive learning) et la perte InfoNCE. Les concepts fondamentaux de l'apprentissage auto-supervisé, tels que la modélisation générative et l'apprentissage de représentation, ont été identifiés comme essentiels pour développer un nouveau cadre adapté aux bioimages. L'accent a été mis sur les progrès dans l'apprentissage auto-supervisé en vision par ordinateur, malgré les défis liés à l'application de l'auto-attention aux données d'image.

Dans ce contexte, cette thèse a pour objectif de trouver un modèle d'analyse d'image polyvalent pouvant fonctionner pour une variété de bioimages grâce à l'apprentissage auto-supervisé. En particulier, le manque de jeux de données à grande échelle pour l'apprentissage automatique reste un défi récurrent, et sera un des fils conducteur du reste du manuscrit.

Les contributions comprennent la création d'un méta-ensemble de données en combinant des ensembles de données existants, le développement de la bibliothèque Python

`bioimagerloader` pour faciliter sa gestion, l'évaluation de l'apprentissage supervisé pour l'analyse des bioimages, l'introduction d'un nouveau modèle et d'une perte auto-supervisée innovants ainsi que l'application de ces idées à deux problèmes en pratique.

Dans le chapitre 2, nous mettons en évidence les limites des modèles pré-entraînés en apprentissage supervisé sur des données locales et soulignons la nécessité d'un ensemble de données générique et vaste. Une approche supervisée a été tentée, montrant des améliorations significatives, mais soulignant également la nécessité d'une diversité accrue dans l'ensemble de données.

Le chapitre 3 se concentre sur l'apprentissage auto-supervisé, décrivant la création d'un modèle, NU-NET, basé sur une perte morphologique novatrice. NU-NET s'est révélé être un filtre polyvalent pour les objets en forme de blob, avec un potentiel de généralisation à d'autres morphologies. L'accent a été mis sur la création d'un ensemble de données générique et diversifié.

Dans le chapitre 4, des applications pratiques de l'apprentissage automatique ont été abordées, démontrant l'efficacité de l'apprentissage supervisé dans la segmentation des fibroblastes. Deux projets spécifiques, FILM et `BigAnnotator`, ont été présentés, montrant les opportunités et les défis de l'application de l'apprentissage automatique dans des contextes biologiques spécifiques.

En conclusion générale, la thèse souligne l'importance cruciale des ensembles de données dans l'apprentissage automatique, en particulier dans le domaine des bioimages. L'approche supervisée a montré des résultats prometteurs, mais les limites inhérentes ont conduit à une exploration de l'apprentissage auto-supervisé. Les résultats obtenus et les modèles développés ouvrent la voie à des applications futures de l'apprentissage automatique dans le domaine de l'imagerie neurodéveloppementale. La thèse offre également des perspectives sur les défis persistants tels que la normalisation des formats d'annotation et l'expansion des ensembles de données existants. Enfin, elle souligne l'importance de l'accessibilité et de l'utilité des outils logiciels dans le domaine.

ABBREVIATIONS AND SYMBOLS

ABBREVIATIONS

Abbrv.	
CNN	Convolutional Neural Network
ConvNet	Convolutional Neural Network (deterred in this thesis)
GAN	Generative Adversarial Network
AE	Autoencoder
VAE	Variational Autoencoder
NLP	Natural Language Processing
FOSS	Free and open-source software
FAIR	Findability, Accessibility, Interoperability, Reusability
CPU	Central Processing Unit
GPU	Graphical Processing Unit
TPU	Tensor Processing Unit
API	Application Programming Interface
GUI	Graphical User Interface
OOP	Object-Oriented Programming
SNR	Signal-to-Noise Ratio
ChroMS	Chromatic Multiphoton Serial microscopy
EM	Electron Microscopy
DoG	Difference of Gaussians
LUT	Logarithmic Lookup Table
UINT	Unsigned Integer
INT	Signed Integer
IoU	Intersection over Union
CR	Contrast Ratio

SYMBOLS

From time to time, I am going to give a minimal explanation of a certain concept with mathematical symbols. This table will be your guide for them. Styling and font weight, all means something different. Sometimes, I used a vector and a matrix interchangeably because a matrix can be thought as a set of vectors.

Symbol	Meaning	Example	Read
\mathbb{R}	Real number	$42 \in \mathbb{R}$	42 is a real number
\mathcal{D}	Dimension		
\vec{x}	Vector		
\mathbf{x}	Vector		
\mathbf{X}	Matrix		
\mathcal{Z}	Vector space		
\in	in	$\mathbf{x} \in \mathbb{R}^{\mathcal{D}}$	\mathbf{x} is a real vector in dimension \mathcal{D}
		$\mathbf{z} \in \mathcal{Z}$	\mathbf{z} is a vector in a vector space \mathcal{Z}
\mathcal{L}	Loss function	$\mathcal{L}(x, y)$	\mathcal{L} is a loss function of x and y

CHAPTER 1

INTRODUCTION

QUESTION “Do you think there will ever be a machine that will think like human beings and be more intelligent than human beings?”

R. FEYNMAN “They will think like human beings? I would say no. ... There is no question that the later machines are not going to think like people think.”

- Q&A excerpt from a recording of a lecture
by Richard Feynman in 1985

Contents

1	Microscopy and Bioimage Data	11
1.1	Neurodevelopmental Imaging Techniques	11
1.2	Bioimage Informatics and Computer Vision	17
1.3	Deep Learning Approaches for Bioimages	20
1.4	Limitations and Challenges of Current Solutions	29
2	General Trends of Computer Vision Solutions	34
2.1	Supervised Learning Dominates	34
2.2	Unsupervised Learning is Pursued	35
2.3	Self-supervised Learning Arrives	37
2.4	Self-supervised Learning in Computer vision	38
3	Self-supervised Learning in This Thesis	43
3.1	Generative Model	43
3.2	Representation Learning	47
4	Objectives of The Thesis	53

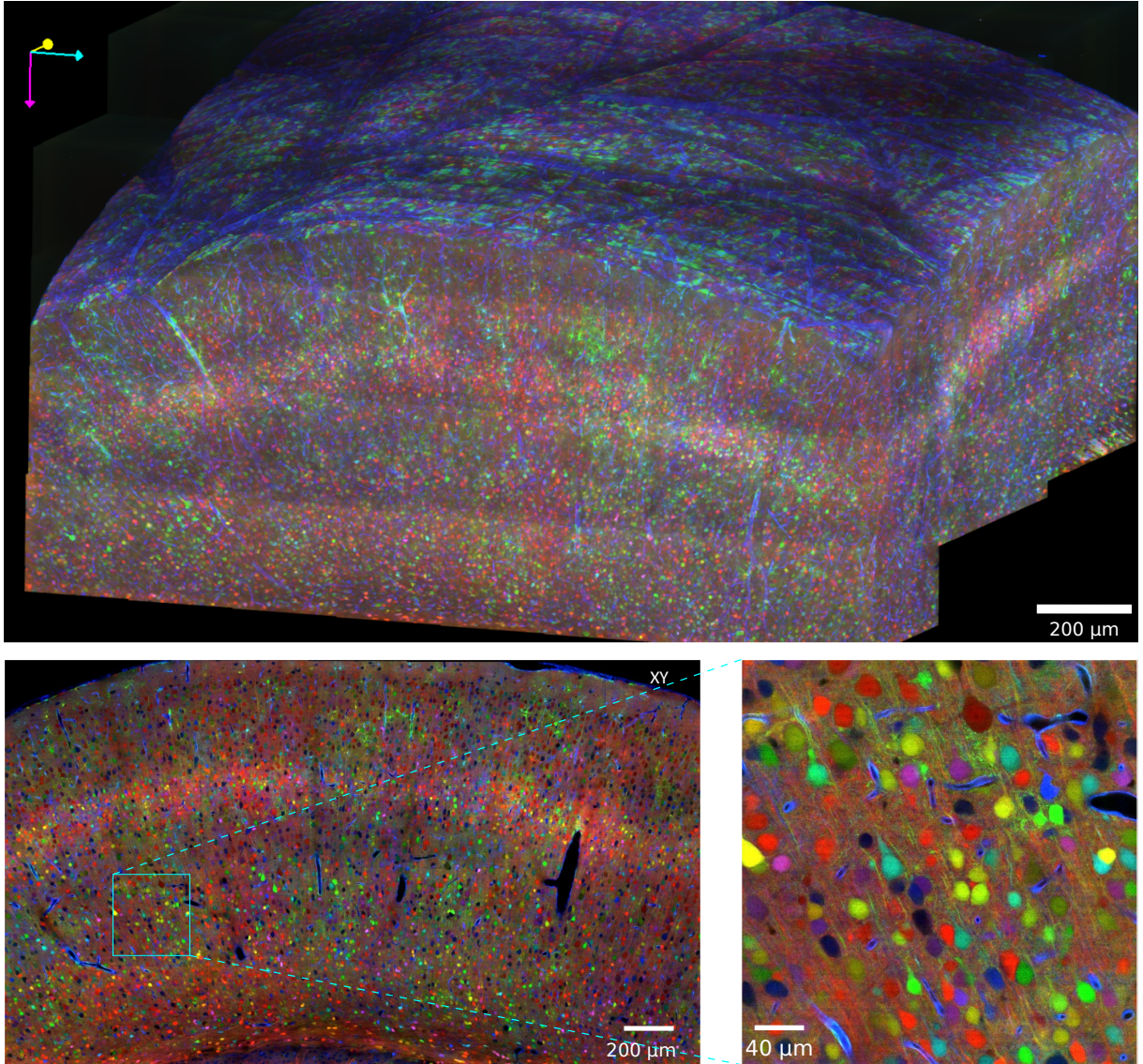


Figure 1.1: A whole mouse brain image with Brainbow[1] staining technique, acquired by ChroMS[2] (Chromatic multiphoton serial microscopy) at LOB (Laboratoire d'Optique et Biosciences). ChroMS is an advanced microscopy platform developed at LOB, which combines color multiphoton excitation, automated tissue slicing, and piece-wise acquisition. The resulting image patches go through sequences of digital processing, and are finally registered and fused. Image was provided by H. Blanc.

In the past two decades, we have witnessed an explosion of data production both in size and rate of acquisition. Data science has become in need across every corner of society to simply deal with the enormity of it, and data-driven approaches became a major tool to understand data and to mine patterns that we had not been able to interpret with simple logical explanation. In the front line machine learning, a data-driven method to model data and its underlying distribution, become the defacto toolbox of choice to adress those problems.

Our data of interest are biological images, one example of which is shown in the above figure 1.1, more specifically those of cellular or nuclear scale, *i.e.* around μm resolution, which I will call simply bioimage. Automating the processing and analysis of bioimages is

essentially a task for computer vision. But bioimages have a number of unique challenges that we cannot find in typical types of image data, so-called natural images. Consequently, computer vision for the bioimage need to adapt developments from other vision problems. Especially, we lack versatile or generic computer vision methods for bioimages, which is critical in practical applications to solve actual biological questions. This is the general question that this manuscript will cover.

Throughout the introduction, firstly, I will briefly present how bioimages are acquired by microscopes and what characteristics bioimages have with respect to data science and machine learning. You will see that there have been a lot of progress made for bioimages, benefiting from unprecedented success in computer vision in general for the last decade or so. However, due to their inherited characteristics, computer vision for bioimages has been showing unique difficulties. Secondly, we will see current trends of general computer vision and how those can inspire ways to drive capabilities of computer vision for bioimages. Lastly, I will state objectives and outline contributions of the thesis in order.

1 MICROSCOPY AND BIOIMAGE DATA

Biology is inherently spatio-temporal and advances of microscopy are crucial to observe biological samples at the cellular and nuclear scale. Images of biological samples acquired by microscopes are called bioimages, which can be extended to include larger family of samples, such as histopathological or medical images. Microscopy technology for bioimages has been moving forward to capture larger field of view in the shortest possible interval for longer span of experiments. It allows us to explore new regimes of life science, like the developmental process which aims to reveal the specifics of how complex tissues like neuronal systems are formed and developed over time [3]. As the neurodevelopmental imaging, that captures snapshots for the process, considers field of views as large as an entire organ or a whole atlas of an organism as well as long time intervals, the amount of acquired data for individual experiment is greatly increasing.

Furthermore, bioimages themselves are unique on their own and have distinct characteristics that typical image data does not share. In the perspective of machine learning, it signifies that bioimages should be handled differently than natural images, and that computer vision solutions should keep these characteristics in mind. Computer vision for bioimages cannot directly benefit from the advances of computer vision in general that have been happening since about a decade or so in an unprecedented speed without understanding its unique challenges. Thus we will start by presenting the tool used to acquire bioimages, that is microscopy.

1.1 NEURODEVELOPMENTAL IMAGING TECHNIQUES

Beyond just hardware, a modern microscope is a set of technologies involving software as well. Specifically, a modern microscope is a system that combines different technologies, namely optical components including laser, lens, image sensor, and even the computer technology. Nowadays, a microscope is likely to come with a powerful workstation not only to store images but also to process and visualize them with dedicated software. To fully understand the specificities of biological images, we need to go beyond the image

Chapter 1. Introduction

sensor (or photodetector) and the software (computer) when a microscope converts analog signals into digital ones; we will see a few major microscopy techniques and what unique looks and intrinsic properties they give to bioimages.

optical
sectioning

One limitation of standard (or “wide-field”) fluorescence microscopes is that they do not provide 3D resolution. For example, in a wide-field microscope, the intensity recorded from a thin horizontal fluorescent plane does not depend on the position of the plane along the Z-axis (or depth-axis). Therefore, the plane cannot be localized in the Z direction. In addition, the image of a thick 3D object will be the superposition of a sharp image from the plane in focus and blurred images from the other planes. In other words, a wide-field microscope does not provide optical sectioning. To address this issue and obtain real 3D images of thick samples, several microscopy techniques have been developed over the past decades. I will briefly review three well-established strategies, plus the fluorescence microscopy in the beginning: **(i)** confocal microscopy, **(ii)** multiphoton microscopy, and **(iii)** light sheet microscopy. Finally, I will briefly outline ChroMS, a large-volume multiphoton imaging technique developed at LOB.

Fluorescence microscopy Typical resolution¹ of microscopy images is sub-micron ($\mu\text{m} = 10^{-6}$ m) to see individual cells, nuclei, and other subcellular structures. The amount of light coming back to the image sensor is tiny, and the light also goes through attenuation processes such as scattering and absorption. In short, it is pitch black inside a tissue, and we could frequently observe Poisson noise² due to scarcity of the detected light. The most common contrast modality to highlight objects of interest is fluorescence.

electro-
magnetic
wave
photon

Fluorescence is a phenomenon by which some molecules emit light back after absorbing light at a different wavelength. Light is a packet of electromagnetic waves and carries energy. Energy of a photon, unit of light, is determined by its frequency, as shown below. E stands for energy, h is the Planck constant, ν is a frequency, c is the speed of light in vacuum, and λ is a wavelength.

$$\begin{aligned} E &= h\nu \\ E &= h\frac{c}{\lambda} \end{aligned} \tag{1.1}$$

energy
state

Every material has energy states corresponding to different electron configurations as well as crystal structures (see figure 1.2). Fluorescent materials happen to have selective energy gaps (or band gaps) that correspond to an absorption and an emission of a certain amount of energy. Once a material absorbs light and emits light back, we call it fluorescence³. Since the absorption gap is predefined by a molecular structure, we do not

fluorescence

¹ Concept of resolution in microscopy is slightly different from a display device. It is the smallest size that a microscope can resolve and related to the numerical aperture (NA).

² It is also called shot noise. Poisson distribution describes discrete probability. For example, you have a coin and flip it 10 times. Ideally, you would get 5 faces and 5 tails. But that is not always the case. This time, let’s say you flip it 3 times, you would not get 1.5 faces and 1.5 tails. And it is more likely to get extreme results like 0 faces or 3 faces. Same thing happens to photons. With abundance of photons, you observe averaged and continuous light. In microscopy environment, you are much more vulnerable to Poisson noise.

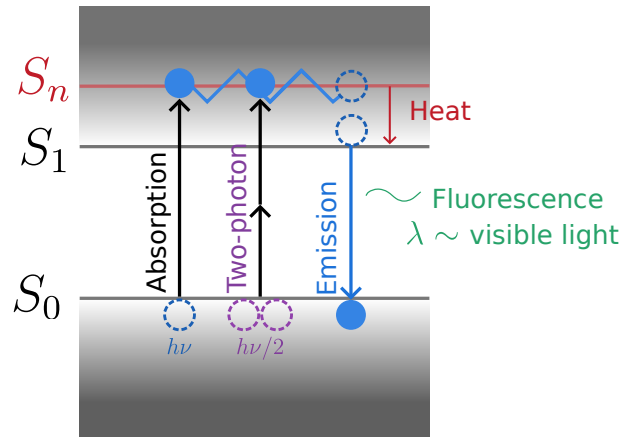


Figure 1.2: Jablonski diagram. Fluorescence can occur when an electron gets excited and goes through a transition back to a stable state (S_0). A material has predefined energy states that allow electrons to stay on. In general, most electrons stay below a stable energy state S_0 . When they gain energy larger than a certain threshold ($S_1 - S_0$), they jump to excited states. In the end, all drops back to a stable state losing energy. Some of them can contribute to fluorescence of a wavelength λ and the rest result in heat. Multiphoton absorption can occur once multiple photons interact within the same molecule and transfer energy. The diagram depicts two-photon absorption where each photon has energy of $h\nu/2$. One electron gets excited beyond the gap and emits nearly half the wavelength λ . Read more in multiphoton microscopy section on page 4.

need to use a light source that has a broad spectrum of light. Fluorescence microscopy usually uses a laser to illuminate a certain wavelength of light to induce fluorescence. Researchers use fluorophores, or fluorescent materials, to stain biological objects of interest. These materials usually meet at least three requirements: **(i)** They need to emit visible light. **(ii)** They need to have high quantum yield, which refers to a conversion rate of fluorescence. **(iii)** They easily attach to certain objects. The advent of the *GFP* (green fluorescent protein) in the 90s (Nobel prize in 2008) was instrumental in the rise of fluorescence microscopy. They are a group of proteins that absorb blue or near ultraviolet light and express green light. Certain GFPs bind well to cell membranes, and thus are used to mark cell membranes. *Hoechst* and *DAPI* are known to stain DNA and emit blue light. There are many other fluorophores and using them in one sample allows to observe different objects by controlling the wavelength of the excitation laser. When images with multiple fluorescent tags are resolved to data, they form multichannel images thanks to their selective responses⁴.

fluorophore

multichannel image

³ Pretty much all materials that have low conductivity absorb electromagnetic waves (a.k.a. light) and radiates waves back with lower energy, in general. Metals usually form continuous bands of energy states and electrons can freely jump over these states with tiny amount of energy. As soon as light hits a metal, free electrons are excited and gather at the surface to greet it, reflecting light and heating up the material.

⁴ Note that the natural color of fluorescence does not match the final look of bioimages once resolved to a digital format. The color is rather a response to a certain wavelength of the laser and plays a role as a marker. Imagine a sample with five different markers. You cannot easily translate them to the common RGB image format that has three channels. Even if it has only three markers. I am going to address this issue again soon.

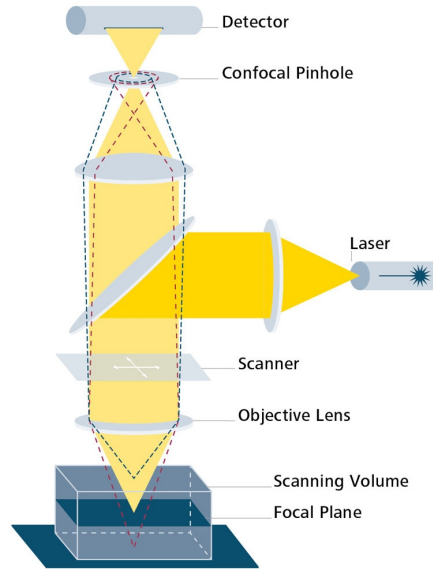


Figure 1.3: Diagram depicting principle of confocal microscopy. Source: [Zeiss webpage](#)

optical
section

Confocal microscopy Confocal microscopy is a point-scanning imaging method. At any time, an illumination laser is focused to a micrometer-size (diffraction-limited) spot within the sample, and the fluorescence originating from this focal point is isolated from the fluorescence coming from other planes using a small aperture (the confocal pinhole) placed in front of the detector. The illumination spot is then scanned in 2D or 3D to produce a 2D optical section or a 3D image. Although confocal microscopy is widely used to study transparent objects such as fluorescently labeled cells, it has three main limitations. (i) Confocal microscopy does not perform well deep inside scattering media such as live biological tissues. The reason is that scattering and aberrations progressively reduce the true signal and increase the out-of-focus background as the excitation beam is focused deeper in scattering samples, thereby degrading the optical sectioning ability. (ii) Confocal microscopy requires to illuminate the entire object in 3D to obtain a 2D section. Therefore, it can induce significant photo-perturbation during long-term imaging of live samples. (iii) Being a point-scanning technique, it is inherently slow due to the time required to record enough fluorescence on each pixel. These disadvantages can be partly addressed using multiphoton or light sheet microscopy.

phototoxicity

Multiphoton microscopy Multiphoton microscopy is a mean to achieve deeper imaging and potentially less phototoxicity than confocal microscopy [4]. Multiphoton microscopy uses a multiphoton process as its contrast mechanism. Multiphoton excitation is a phenomenon where n (generally 2 or 3) lower-energy photons of energy $E = h\nu/n$ interact with a single fluorophore to promote it to an excited state with energy $h\nu$ by combining their energies. One characteristic of multiphoton excitation in a microscope is that it occurs efficiently only near the microscope focus where the intensity is highest, due to the nonlinearity of the signal generation mechanism. Scanning the focal point in 2D or 3D across the sample results in an optically sectioned image.

In practice, multiphoton microscopy is implemented using pulsed infrared excitation light, with a laser source delivering typically 100 fs (femtosecond; 10^{-15}) pulses at a rate of 80

MHz, in the wavelength range of 800-1200 nm. Using pulsed excitation is a means to enhance the probability of multiphoton effects without increasing the average laser power. Multiphoton microscopy performs much better than confocal microscopy deep inside scattering samples, because (i) the excitation remains confined to micron-scale volume due to the nonlinearity of the signal generation, and therefore the signal-to-background ratio is preserved even in the presence of scattering; (ii) the infrared excitation light used for multiphoton excitation experiences less scattering than the visible excitation light used in confocal microscopes. In addition, only the plane of interest is being illuminated through a multiphoton process, thereby limiting the phototoxicity in the case of long-term imaging. To give a practical example, two-photon microscopy enables to image mouse brain cortical tissue *in vivo* at depths of half a millimeter with subcellular resolution during hour-long experiments. However, like confocal microscopy, multiphoton microscopy it is a point-scanning and therefore rather a slow imaging technique.

Light sheet microscopy Confocal microscopy probes a point at a time and, by nature, has a slow acquisition rate. One way to address this issue is to parallelize the imaging process. Light sheet microscopy is a parallelized technique where an entire plane in the sample is illuminated from one side, and an entire image is recorded on a camera in a direction orthogonal to the illumination direction [5]. This orthogonal illumination/detection geometry provides much faster imaging rates by recording multiple pixels simultaneously. In turn, faster imaging rate enables to develop new applications: the speed gain can be used either to image larger volumes within the same acquisition time, or to perform fast acquisition of time-series data. An additional advantage of light sheet microscopy over confocal imaging is that only the imaging plane is being illuminated, resulting in much less phototoxicity, and in turn being more compatible with live imaging of cells or small embryos.

One limitation of light sheet microscopy, however, is that it works well only with transparent samples because it suffers from light scattering and the like. This is why it is often combined with optical clearing methods that make a specimen transparent. An interesting perspective is the ongoing development of multiphoton light sheet microscopy[4], which provides an interesting combination of fast and deep imaging.

ChroMS ChroMS[2] stands for **Chromatic Multiphoton Serial** microscopy. It is a technique for large-scale high-resolution imaging of color-labeled *ex vivo* tissues recently developed and operated at LOB [2]. ChroMS combines colors by two-photon excitation using a wavelength mixing technique with automated serial slicing/recording of 3D blocks of a tissue. It provides a unique combination of high-resolution multicolor imaging of large volumes of uncleared tissues, and has been successfully combined with the Brainbow method [1], which is a multicolor transgenic fluorescence labeling approach developed by our collaborators at *Institut de la vision* (J. Livet's lab). ChroMS datasets contain $10^9 - 10^{11}$ color voxels, and naturally became a main target of my thesis. See sample images in figure 1.1.

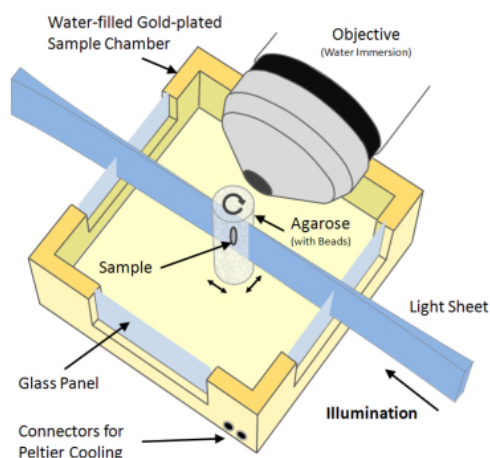


Figure 1.4: Concept diagram of OpenSPIM light sheet microscopy[6]. OpenSPIM stands for Open Access platform for applying and enhancing Selective Plane Illumination Microscopy (SPIM). Note that there are many other configurations of light sheet microscopy. Source: <https://openspim.org/>. For those who are not familiar with all the optical components like me when I started my PhD, I will briefly explain each. The objective is the main lens to collect light and to send it to an image sensor. Agarose is a transparent gel to hold sample fixed. Mounting is an essential skill to use microscopy and an agarose can help a lot. Chamber is water filled because the objective is designed to be used immersed in water. Refractive index matching is the essential step to see things with microscopes. So you would like to have the same media between the objective and the sample or at least one that has a similar refractive index. Gold is known stable and does not interact with other substances. Light induces heat eventually and one session of light sheet microscopy can last over a week or more. Peltier cooler uses electrons to carry heat away and is something you can find in a water purifier or a wine refrigerator. Why beads in the agarose? Sample holder can rotate the sample during acquisition with the objective fixed. Later you would like to register these fragmented images (or multiview images) back to a complete volume. Beads can help registration algorithms to find overlapping area easily.

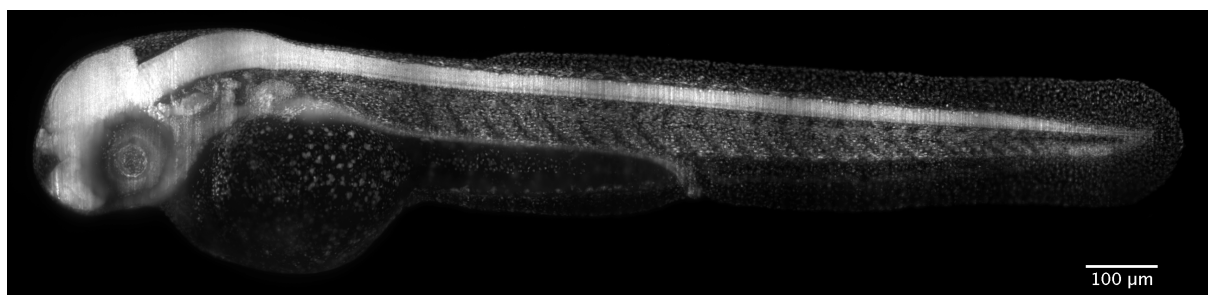


Figure 1.5: Whole zebrafish image acquired by OpenSPIM[6] light sheet microscopy. Source: <https://openspim.org/>

1.2 BIOIMAGE INFORMATICS AND COMPUTER VISION

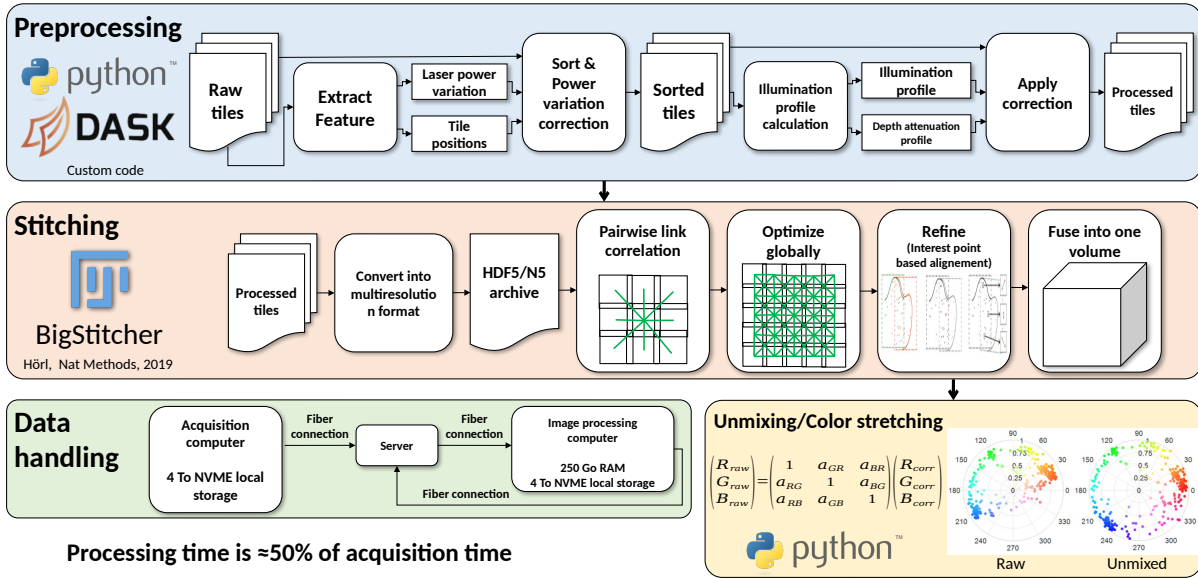


Figure 1.6: Data pipeline of ChroMS. It consists of three parts: preprocessing, stitching, and data handling. As a confocal microscopy, ChroMS point-scans samples and the acquired images are tiles and needs to be stitched. BigStitcher[7] is a free and open source software for registration. Preprocessing and stitching occur in a powerful workstation since they require big system memory and high computing power. Lastly, the final bioimages are stored in a storage server. For a whole mouse brain sample, an average acquisition time is around 24 hours and the whole data pipeline for it takes around 12 hours. Diagram was provided by H. Blanc.

Bioimages, acquired by microscopes introduced in the last section, aim to capture a rather large field of view in great detail. They are meant for researches that need to observe interactions between many cells to the smallest, a whole organ, connections between organs, or even a whole organism to the largest. The size of such bioimage data is considerable, and its growing size poses questions from how to store to how to analyze such large data given limited computing resources. In addition, bioimage data possesses unique challenges distinct from other types of image data. To address these questions and challenges, informatics naturally got involved in the bioimage analysis pipeline. Consequently, bioimage informatics, a branch of bioinformatics, has evolved rapidly to address aforementioned issues [8].

Bioimage informatics concerns every step when data is computationally processed. It starts from how to store and visualize image data, which is not trivial from the beginning. For instance, the diagrams in figure 1.6 demonstrate how ChroMS microscope[2] handles its image data. In an advanced microscopy such as ChroMS, image processing is a part of the system. There are many steps for different purposes, and they form a pipeline. One of the most important image processing steps in advanced microscopy is the registration. Image registration is a processing step to bring all the fragments of acquired images from scanning and to reconstruct a whole view. It is performed spatially. This is a long pipeline that would take about half a day to process about 500GB of image data that would result in 250GB at the end. Its main goal is to gather many tiles of point acquisition and to stitch them back to one complete volume, from which the experiment started. You may notice that it involves several image processing techniques including feature extraction

and color correction. It also regards ways to store and access data through a storage server since the data is too large to be duplicated on client computers every time they demand access to it. This example merely outlines the beginning or a preparation step towards analysis.

After data is ready to be consumed, the actual image analysis awaits biological discovery. However, as for quantitative image analysis, which has become a major way to conduct image analysis [3, 9], there is just one remaining step to it, that is image segmentation. Image segmentation refers to a task to localize regions of interest given an image by drawing boundaries or painting area (consult figure A.2 for more comprehensive definition). These segmentation annotations can be used to deduce response of experimental treatment through its observed intensity [10], to track movement or development [11], or to follow axons that connect neurons [12, 13].

There long-existed ways to automate image segmentation tasks. Background removal is a popular practice that can achieve similar results to those of segmentation or could be used as a preprocessing step in a longer pipeline. For instances, difference of Gaussians or Laplacian of Gaussian are such solutions whose assumption is based on morphological characteristics of the objects of interest and that use a linear filter with a predefined kernel function to leverage them. Following that, for an example, the Otsu's threshold method is used to determine a value to binarize distribution based on histogram analysis. To identify instances and generate a segmentation mask, Watershed algorithm that starts from a position and progressively "sheds water" until it reaches the boundary. This approach to construct a pipeline of linear steps has been a common way to solve segmentation tasks in bioimage analysis tools [14, 15, 16]. However, such pipelines has heterogeneous complexity between steps, which makes them highly reliant on strict conditions of inputs and hard to tune parameters of each step. At bottom, they lack of versatility. Classical techniques as used in bioimage informatics are reviewed in [17, 9, 8]. They involve a great variety of techniques and algorithms, from partial differential equations to markov random feilds or mathematical morphology. We will not cover them here and focus on machine learning techniques.

Because of rather limited usages and inconsistent accuracy of such pipelines, scientists tended to rely on their own eyes and use their own hands to perform a segmentation task, which may take from a few hours to several days but will guarantee much higher quality in contrast to automation. It is much so especially when it comes to bioimages where contrast is low and objects of interest are more subtle to recognize even to human eyes.

As the amount of bioimage data grows immensely and the demands of it grow more precise, however, it became no longer feasible nor realistic to perform the segmentation task by hands or even manually tune classical algorithms. There was a breakthrough around 2012 in computer vision that made a huge impact on how to make computer to see better and perform better on most vision tasks such as a segmentation than ever before [18]. This new type of computer vision mechanism was based on three main technologies: machine learning, deep neural networks, and advance in computational processing units such as CPU (central processing unit) and GPU (graphical processing unit). Using machine learning in particular with deep neural networks turned out to be much effective to computer vision tasks in general. For instance, deep learning could replace a complex

pipelines of linear steps and still provide more accurate and general solutions.

Deep learning, or machine learning combined with neural networks, provides powerful solutions for many computer vision tasks not just constrained to bioimages, yet it is not a definite one for them either. It is relatively a young technology that needs more time to explore and has potential to advance further. Also, there are limitations and challenges to address, two of which are versatility and generalizability. These two ideas are the main topics of this thesis and constitute its overall theme.

Note that I covered machine learning and neural networks not in the main text but in appendices to provide lengthy contexts and summary, given that there have been so much progress made in a short period of time. I compiled a brief introduction to machine learning and recent advancement of neural networks. If you are not familiar with these concepts or would like to remind yourself of their development, I recommend you, before going forward, to start from appendix [A](#) to [4](#) where I put together a small introduction to those methods.

1.3 DEEP LEARNING APPROACHES FOR BIOIMAGES

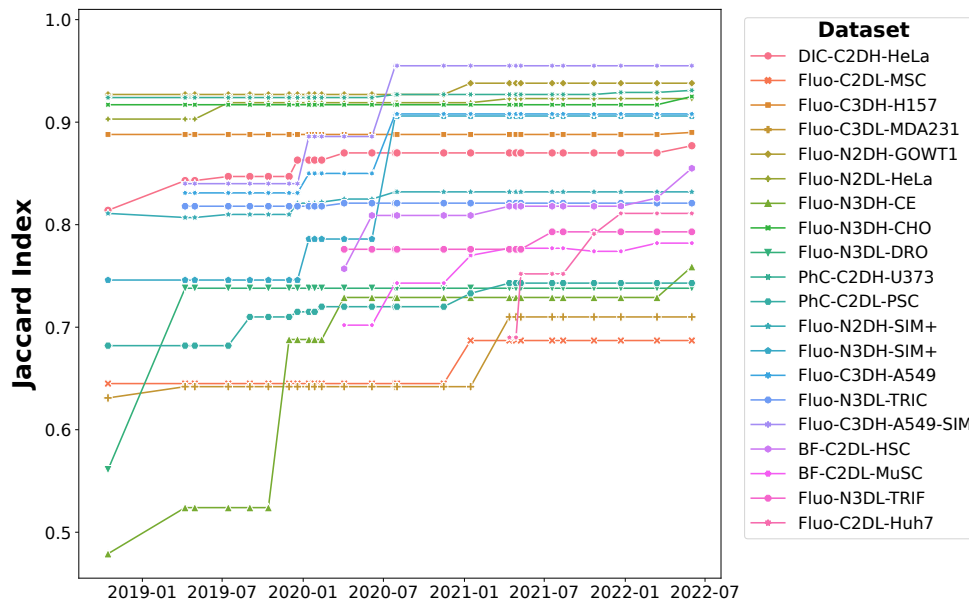


Figure 1.7: Evolution of the top performance of segmentation models for Cell tracking challenge[19]. They used Jaccard similarity index for the metric (find appendix C for its definition). So far as time of writing, Cell tracking challenge hosts 20 datasets, and the number keeps growing. Note that the challenge was initiated in 2017, but the record goes back only to 2019.

Many deep learning solutions have been proposed to solve computer vision tasks for bioimages. I structured an overview in **four subjects**: (i) segmentation algorithms and models, (ii) algorithms and models for other computer vision tasks, (iii) datasets, and finally (iv) software tools. Segmentation is probably the task that has the greatest impact since the rise of deep learning, since as mentioned, segmentation is an essential step to quantitative analysis. There are other computer vision tasks in which deep learning proved effective for bioimages, like image restoration and denoising. All those advances are dependant on datasets, the foundation of machine learning. Lastly, software tools play an important role in bioimage analysis because they provide interactivity and high level abstractions that could alleviate logistical complexity and challenges deep learning comes with, and are thus of key importance in practice.

quantitative
analysis

Segmentation Identifying objects of interest is the beginning of every analysis and further enables quantitative analysis. Though there is endless diversity in biological morphology, I will focus on segmentation of blob-shaped objects that usually correspond to nuclei and cells. Elongated objects, like axons, turned out to be much harder to segment for the moment because it inevitably concerns 3D volume, for which segmentation algorithms are not mature enough due to limitation of resolution and of computing resources⁵. There are two major segmentation tasks that we concern: semantic segmentation and instance segmentation (see figure A.2 for their difference). I am going to review a few representative deep convolutional neural networks (CNNs). While this is a non-exhausted list of

⁵ Deep learning requires quite a lot of computing resources even for 2D images. For instance, current state-of-the-art generative models often need roughly more than 8GBs of GPU memory for images as large as 512×512. Commercial GPUs for normal consumer do not have enough capacity, in general. 3D volume grows in cubic order, and deep learning still struggles to accommodate large volumes.

models in a very fast moving field, they are the most prominent ones and exemplify the state of the field.

U-Net[20] was a key foundational work which was introduced as an entry of *cell segmentation challenge* at ISBI (IEEE International Symposium on Biomedical Imaging) in 2015. At that time, not many researchers and practitioners were exposed to deep learning in biomedical imaging community. U-Net outperformed all the other entries, trained only with 35 images. The key insight of U-Net is the encoding-decoding architecture with skipped connections that enable both the learning of low dimensional features and the use of those features in a multiscale fashion. Trained with densely annotated data, it essentially performs pixel classification. Its main limitations, addressed in part in following works[21, 22], are the lack of explicit geometrical constraints to account for the fact that objects in bioimages tend to be simpler and more stereotyped than in natural images, and its inability to deal with touching objects without further refinement or post-processing.

U-Net is important for a few reasons apart from its outstanding performance. **(i)** It motivated machine learning researches, especially deep CNNs (convolutional neural networks), in biomedical imaging community. Figure 1.7 shows evolution of cell segmentation section of *Cell tracking challenge*[19]. Researchers suggested many variants based on U-Net. Some addressed limitation of U-Net and attempted to improve them [23, 21, 24], others extended the original 2D model to 3D models [25, 26, 27]. Also, it contributes to development of instance segmentation deep CNNs too. StarDist[28, 22] and Cellpose[29] are the most frequently used deep CNNs for the instance segmentation task, which I will cover shortly in more depth. **(ii)** It got adopted in many existing software tools, for example, in Fiji (ImageJ)[30, 14] and Cellprofiler[31]. Implementation in tools was important because U-Net was one of the first deep learning models that users actually found useful. **(iii)** It inspired other architectures even outside biomedical imaging. FPN (feature pyramid network)[32] was one of them and addressed the scale variance issue in object detection CNNs. Furthermore, U-Net recently became more used thanks to diffusion models[33, 34, 35, 36], the current state-of-the-art image-generative model.

StarDist[22] is an instance segmentation deep CNN, specifically designed for star-convex polygons. It uses U-Net as a base model and assumes that objects of interest have star-convex polygon shapes. They devised, what they called, *star-convex polygon distance*. In a nutshell, StarDist picks up all possible candidates that has star-convex polygon shapes and uses non-maximum suppression (NMS) to filter out proposals with low probability. It was tailored for blob-shaped biological objects like nuclei and effectively altered a semantic segmentation model (U-Net) to an instance segmentation model (StarDist). The group soon extended their model to 3D volume[28] and ported their models to free and open source image viewers, namely Fiji[14] and Napari[37]. To this date, StarDist is one of the most accessible and practical deep CNN models for instance segmentation in bioimages.

Cellpose[29] is another popular instance segmentation deep CNN and shares a common interest with StarDist, which is to use shape priors to segment instances. It also used U-Net as a base CNN and simulated diffusion from centers of each cell. Compared to StarDist where it parameterized radial directions to deduce polygons, Cellpose’s diffusion simulation was not parameterized and thus more flexible and adaptable to shapes beyond so-called star-convex polygons. Cellpose assumed two channel images and differentiate

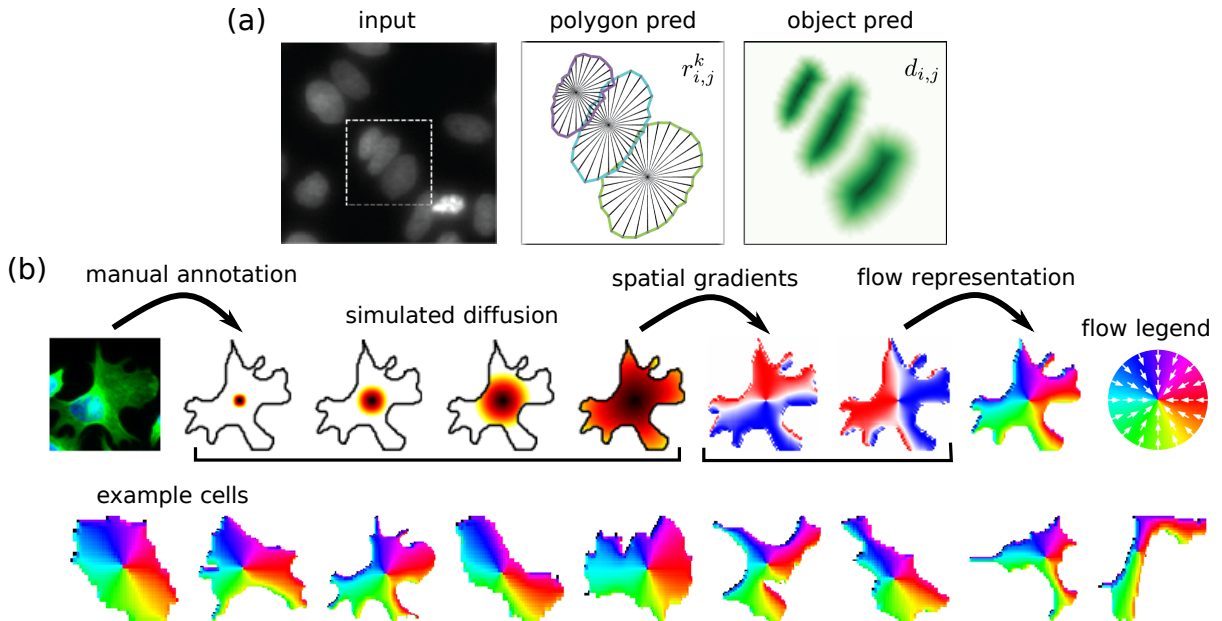


Figure 1.8: (a) StarDist predicts polygon shapes using parameterized radial distance ($r_{i,j}^k$; distance of pixel (i, j) to the boundary on k -th ray) and object probabilities for each pixel ($d(i, j)$). (b) Cellpose simulates “poses” of cells by using diffusion concept and making horizontal and vertical flows. It predicts two flows as well as object probabilities. Source: (a) annotation modified from StarDist[22], (b) Cellpose[29]

nuclei and cells, which can be easily confused by algorithms. In addition, Cellpose attempted to become a generalized model for segmenting blob-shaped objects by including images even outside biology, e.g. those of fruits, rocks, etc. Like StarDist, Cellpose team concerned about accessibility and usability, and released an application with graphical user interface (GUI). They ported Cellpose to Napari image viewer[37] as well. The team has kept improving their models and applications since the publication. Recently, they introduced Omnipose[38] which aimed to segment more skewed objects, and Cellpose v2.0 which shipped more features. Though it is not directly available in Fiji[14], a go-to image viewer in the community, TrackMate[15, 39], a cell tracking plugin in Fiji, started supporting Cellpose for advanced users.

NucleAIzer[40] is a little special in this category because it chose an object-detection-based instance segmentation CNN, namely Mask R-CNN[41], as a base, instead of a semantic segmentation CNN like U-Net. Mask R-CNN was already proven performant and a good choice for objects that fit into bounding boxes, thus used in other bioimage targets too[42, 43]. The NucleAIzer team also aimed to make a generalized model but for nuclear segmentation. Their approach was to simulate synthetic images. Although, simulating a training dataset was not particularly a new idea [44, 45, 46], they used a generative deep CNN to do so. What they used was pix2pix[47], which is a style transfer application using a GAN[48]. First, they gathered diverse nuclear bioimages and clustered them into a limited set of “styles”. Then, they simulated images for all clusters by applying each style using pix2pix. The group made their model available through Cellprofiler[31].

Mesmer is yet another instance segmentation model introduced with TissueNet[43] and the latest one of its kind. Since they had a large scale data available, much more

than previous models, they wanted more capacity in terms of architecture and went for ResNet[49] as a backbone. Instead of relying on shape priors, Mesmer predicts borders along with an inner distance map and chose to use Watershed algorithm to mark instances. Additionally, they employed FPN[32] to take into account various scales.

The authors reported that Mesmer outperformed existing segmentation models such as said pre-trained StarDist, Cellpose, etc. Furthermore, they compared performance of their model to that of humans and showed that they were comparable. Given that a major issue in previous models was a limiting scale of dataset, it might have been apparent that they managed better performance. However, what they did and the most important contribution to the research was the TissueNet dataset. They raised the issue of small scale of dataset when it comes to researching computer vision using machine learning and actually provided a large dataset as a solution. As a matter of fact, the change of architecture barely contributed to its gain. It turned out that StarDist and Cellpose resulted in matching performance to Mesmer when they were trained with TissueNet from scratch.

Image restoration and Denoising Apart from segmentation, image restoration and denoising are fundamental image processing that can be understood as inverse problems, reverting the image acquisition process to remove the noise and resolution loss it introduced. Thus they attempt to cope with innate limitations of microscopy, particularly low contrast and noise. See figure 1.9.

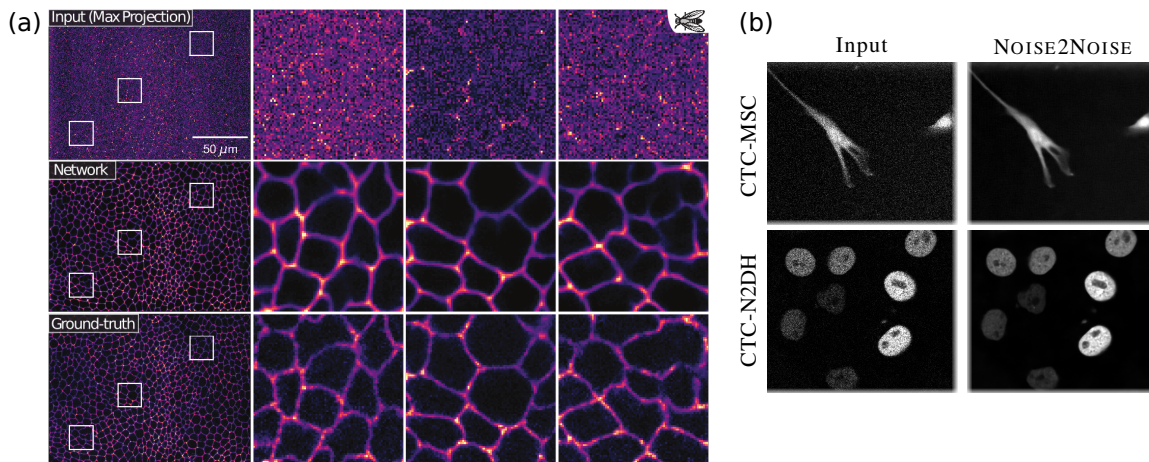


Figure 1.9: Effectiveness of image restoration and denoising techniques using deep learning. Images sourced from (a) CARE[50] and (b) Noise2Void[51]

As for **image restoration**, CARE[50] (content-aware image restoration) network is by far the most favored model to enhance low-SNR (signal-to-noise ratio) images. You would need a pair of low-SNR and high-SNR images of the same field of view, which can be acquired in a few ways, e.g. alternating illumination power or scanning fast and slow. The idea is to benefit from low power illumination or fast scanning to avoid various issues from photo damages, once you have a CARE model.

Denoising is an endless battle in microscopy. Classical denoising algorithms that do not rely on machine learning focused on structured noise and try to track down the source

of noise and remove it [52, 53]. These methods may frequently end up with artifacts when the source of noise is unclear. Machine learning, in particular deep learning methods, apparently excelled in denoising applications because they can deduce noise sources from data. *Noise2Noise*[54] supposed that noise is independent of content of images and tried to learn it by pairs of noisy images. They artificially applied Gaussian, Poisson, and Bernoulli noises on the same image to generate noisy pairs. Its limitations were that (i) it needs pairs of noisy images, (ii) limited distributions in terms of variety of noise, (iii) noise can be dependent to contents like PSF in microscopy. Soon after, *Noise2Void*[51] came out, specifically aiming to remove noise in biomedical images. Their novel *blind-spot network* takes into account neighboring pixels to predict a center pixel and allowed learning noise from a single image, not from a pair. They pointed out Noise2Void takes in relatively small amount of information due to small receptive field, compared to other models including Noise2Noise, and thus has a theoretically upper bound. But the result turned out more than good enough. More importantly, Noise2Void was easy to use because users do not have to engineer training process, e.g. how to prepare noisy pairs as in Noise2Noise.

Dataset Deep learning, which is under the umbrella of machine learning, is essentially a data-driven method. This is particularly true since all the methods presented so far use supervised learning, a machine training scheme where the model is explicitly shown the desired result, and thus need annotated data. Therefore, data is the alpha and the omega that affects every aspect of deep learning from the beginning to the end. A dataset, a set of data curated for certain tasks, could be thought as the ‘unit’ that data for machine learning comes as, that is actually used in machine learning.

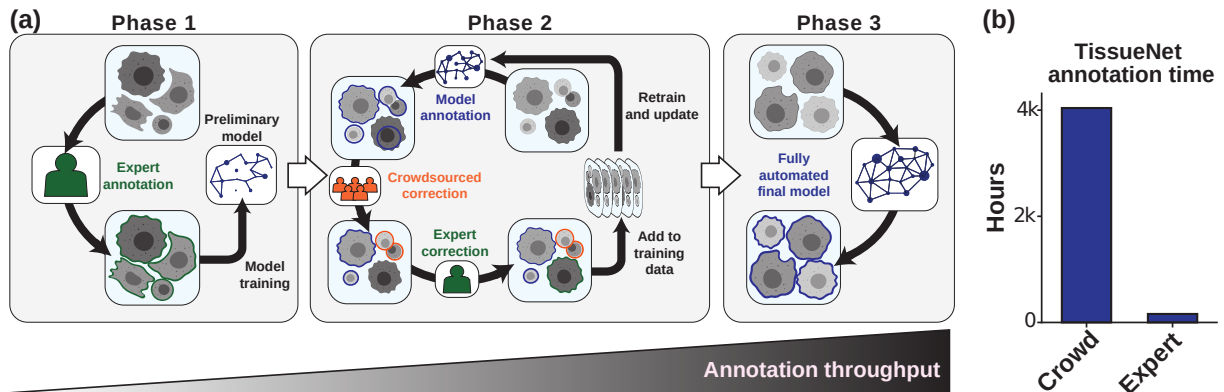


Figure 1.10: (a) The way TissueNet[43], currently the largest bioimage dataset, was curated. They used a human-in-the-loop approach and managed to build a large dataset with 1.3 million whole-cell annotations and 1.2 million nuclear annotations. (b) They also estimated price of constructing such large dataset based on the time spent. They supposed \$6 per hour for crowd-sourced annotators, \$30 per hour for highly trained annotators, and \$150 per hour for expert pathologists. According to their prediction, TissueNet would cost \$25,000, which is not cheap by no means but far less expensive than a classical approach where only the experts annotate the whole data from scratch.

First, there are a number of generic repositories and collections of datasets that could be used for machine learning approaches. The *Broad Bioimage Benchmark Collection* (BBBC)[55] by Broad institute hosts more than 50 individual datasets for various biomedical computer vision tasks. They categorize them into three main applications: identifi-

cation and segmentation, phenotype classification, and image-based profiling. *Cell Image Library* (CIL)[56] was created in 2010 and treats images in whichever platforms and formats as long as they concern cellular images. But *CIL* is not necessarily designed to provide datasets curated for machine learning and rather accepts all sorts of cellular images. IEEE *ISBI* (International Symposium on Biomedical Imaging) has held various biomedical computer vision challenges annually since 2012. As name suggests, there have been challenges not only for bioimages but for medical images. *Cell tracking challenges*[19] was first introduced in ISBI2012 and has since become the most notable datasets for cell tracking vision tasks. The number of supported datasets has been growing since its introduction, and it has 20 datasets as of Oct 2022. *Image Data Resource* (IDR)[57] by OMERO (Open microscopy environment) team hosts massive amount of bioimages. As of IDR at the time of writing (version 0.11.1, released on 2022-10-31), it contains 335 TB of multidimensional images consisting of 13M multidimensional images or 109M individual 2D planes. IDR contains over 1M different experiments. But, just like CIL, IDR does not necessarily aim to host curated datasets for machine learning purposes.

Speaking of challenges or competitions, there was *Data science bowl* cell segmentation challenge[58] in 2018 (DSB2018) on a web platform Kaggle⁶, sponsored by *Booz Allen Hamilton*. *DSB2018* was a particularly important bioimage dataset for a number of reasons. **First**, *DSB2018* was well curated. It managed to gather wide range of cellular images across platforms and samples. It was designed for instance segmentation labels, meaning every cell, no matter how many, was all annotated individually in all images. The evaluation metric was solid and fair. **Second**, the timing was just right. Instance segmentation was a hot topic beyond object detection task⁷ with advent of state-of-the-art deep learning models between 2014 and 2018, namely R-CNN family[59, 60, 61] and Mask R-CNN[41], that demonstrated incredible potential (I believe that the arrival of state-of-the-art object detection and instance segmentation deep learning models heated up self-driving car industries, and it definitely had a nudge on Elon Musk and *Tesla*⁸). At the moment, bioimaging community was adopting advanced machine learning and deep learning techniques and *DSB2018* became an invaluable resource. **Third**, the competition platform attracted people from many other domains. Especially, contributions from computer scientists and data scientists in other domains introduced baseline of this

⁶ Kaggle is a web platform that hosts machine learning competitions. Money is certainly a great motivator. For example, Data science bowl 2018 had \$100,000 award in total and attracted 3,634 teams around the world. Kaggle started as a startup in 2010 and had a great success among data scientists. It was finally acquired by Google in 2017. Kaggle is a great platform not only to share knowledge through competitions but also to learn machine learning skills. Contributors often voluntarily provide baseline codes for each competition and many are willing to share their accomplishment (it usually happens when they fall behind the leaderboard to ask for ideas and help).

⁷ Object detection task is to find positions of objects given an image using bounding boxes. See figure A.2.

⁸ It is a bit tangential, but I would like to introduce Andrej Karpathy. Arguably, *Tesla* is a leading company in the self-driving car technology for the moment. Andrej Karpathy became the director of artificial intelligence in *Tesla* around 2017. He contributed a lot to both natural language processing and computer vision in general, when deep learning was still young. He is also a founding member of *OpenAI*, which is a startup currently crushing Google, Meta, and many other tech giants in machine learning domain. The course *CS231n* that he designed and instructed with his advisor Fei-Fei Li (leading scientist behind *ImageNet*) in Stanford University, his Alma Mater, was one of the earliest deep learning courses in university across the globe. It was legendary and still is one of the most notable courses regarding computer vision. He left *Tesla* on July 2022. I am looking forward to his next move.

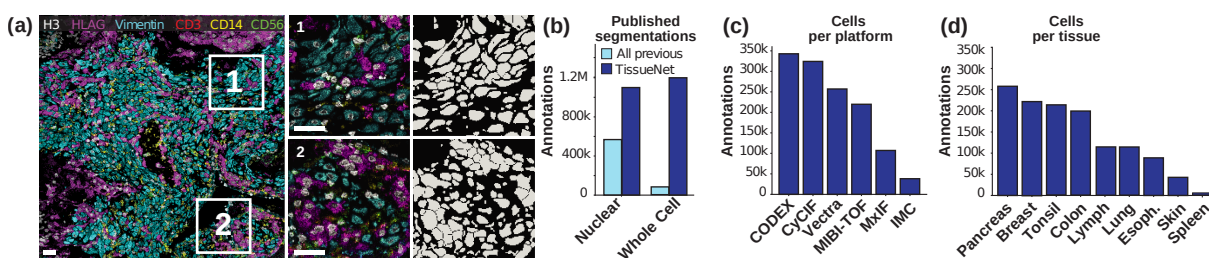


Figure 1.11: (a) Example images of TissueNet[43] and two predictions of Mesmer, an instance segmentation model introduced together with TissueNet. All scalebars indicate $50\mu\text{m}$. (b) Statistics of TissueNet dataset. TissueNet has greater number of annotations even compared to summation of all public nuclear annotated datasets as well as to that of whole-cellular annotated datasets. In addition, it is inclusive, covering diverse platforms and samples. Note that they only considered those with instance segmentation annotations when comparing numbers in (b).

new technology called deep learning and accelerated its adaptation and development in bioimaging community.

Finally, *TissueNet*[43] is the latest manifestation of building a large scale bioimage dataset. It was clearly named after ImageNet[62] that sped up the general development of machine learning and deep learning in computer vision. TissueNet covers diverse platforms and samples, and provides nuclear and whole-cell labels, suitable for instance segmentation computer vision task. What's more important is the scale. They incorporated a human-in-the-loop approach (see figure 1.10) to reduce the cost of expensive annotation process in bioimages and achieved unprecedented numbers of annotations: 1.2 million cells and 1.3 whole cells (see figure 1.11). Furthermore, they attempt to standardize formats of annotations, at least for the two-channel nuclear instance segmentation, by developing *DeepCell Label* which is an application for labeling.

human-in-the-loop

There are other bioimage datasets, most of which targeted narrow conditions and specific tasks. For example, *EVICAN*[63] and *LiveCell*[64] are a little special because they specifically treated simple label-free cellular images, like bright field and phase-contrast imaging. While their images are not as clear as fluorescence microscopy, these microscopes are not invasive to specimens and thus important in pharmaceutical research. Histological images, such as H&E-stained images, are unique and have native RGB channels because they usually use simple light sources, not lasers [21, 65, 66, 67]. *DIADEM challenge*[68] targets tracing long neurons such as dendrites and axons.

Software tools Bioimage informatics sits in between image data and results and is an increasingly indispensable part of biological studies, developed and used by a wide variety of people with diverse skills and background. This means that the practical implementation of the algorithms, impacting the way they are developed, shared and used, is of key importance. In current software development environment, open-source software gained significant momentum. Open-source software opens its source codes to public and allows contributions from individual developers. The arrival of web platforms, such as *GitHub* and *GitLab*, to host codes and provide communication channels to users and contributors further accelerated open-source software development. For the moment, there are two cohorts in bioinformatics in terms of the choice of programming languages: *Java* and *Python*. While Python is a popular programming language of choice these days,

open-source software

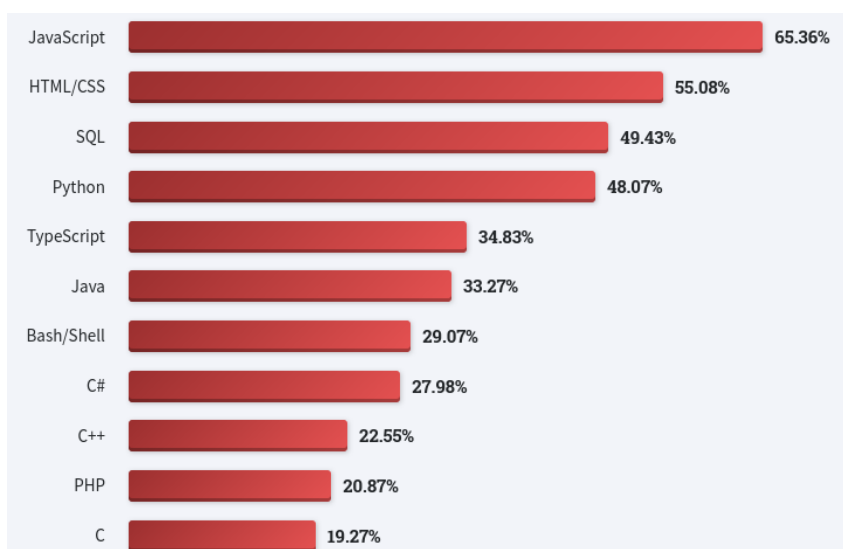


Figure 1.12: 11 most popular programming, scripting, and markup languages: Stack Overflow’s 2022 survey. It was conducted on May 2022 by over 70,000 developers. Python is practically the most popular programming language except those for web development (partially true, because Python is also a popular choice for backend web development). JavaScript and TypeScript are the languages for web technologies. HTML/CSS are markup languages for web pages. SQL is only for database. Java is used everywhere, but mostly for desktop applications and mobile android applications. Source: 2022 Developer Survey by Stack Overflow (<https://survey.stackoverflow.co/2022/>)

as revealed in figure 1.12, Java is still used widely and a programming language of ImageJ, ImageJ which has been the go-to image software tool in the community for over a decade.

Bioinformatics community was aware of issues commercial software might bring about in the long run. They already sorted out a way to free and open-source software (FOSS) long FOSS time ago, and a group of people developed **ImageJ**[14]. Since then, ImageJ has become an essential tool to the community. What is less known is that it went through some big changes towards open-source development environment. **ImageJ2**[30] is a rewrite of the previous version, focusing on extensibility for its plugin system. Plugin system plugin system allows individual developers to deploy their algorithms with minimal efforts into a familiar system environment to users. In parallel, **ImgLib2**[69] was released as a generic library for image processing to replace, now deprecated, **ImgLib1**. ImageJ2 team tightly integrated the library in their software to help developers. **Fiji**[14] (recursive acronym; “Fiji is just an ImageJ”) is a battery included ImageJ, meaning it packages libraries, scripting languages, and plugins that have become useful and essential over the years in ImageJ. One plugin worth mentioning is **Bio-Formats**[70] by OME (Open microscopy environment) team. Bio-Formats reads many biomedical image formats and parses their metadata properly. It sounds simple, but it handles 162 different image formats as of v6.10.1 in 2022. With that number, it already is an essential tool for users. Another one is **CLIJ**[71] that brings interactive image processing interfaces in ImageJ and accelerates certain operations by utilizing OpenCL⁹ and GPU.

⁹ OpenCL provides universal application programming interfaces (API) across diverse hardware. It is known for its optimized performance.

Speaking of **OME** (Open microscopy environment) team, they offer other FOSS tools as well as Bio-Formats. OMERO (OME Remote Objects)[72] is a software to manage and share data using server/client architecture. Server/client architecture is a centralized scheme where a server broadcasts data to clients on demand. **OME-TIFF** and **OME-XML**[73] are a pair. TIFF format is a universal image format but lacks structures and consensus, because it existed long time and there have been too many variants. OME team created OME-TIFF to define a new TIFF format for biomedical images compatible with Bio-Formats. OME-XML attempts to standardize metadata formats. **OME-NGFF**[74] is the latest movement that OME team is leading. It was launched to meet the need of ever-growing large scale biomedical images and cloud storage services.

Python environment in bioinformatics expanded rapidly in a short period time. Its growth is strongly related to the fact that Python is the current language of choice for machine learning and data science in general. It is relatively easy to learn and start writing useful software compared to other languages, e.g. Java or C++. Despite its short history in bioinformatics, there are a number of tools already frequently used and cited in real research. For examples, **ilastik**[75] provides machine learning algorithms in interactive manner with a graphical user interface (GUI). Its approach is easy and effective using random forest algorithm, and adapted to **LABKIT**[76] in Fiji. **Cellprofiler**[31] is a full suite for single-cell analysis. It contains a lot of functions for quantitative analysis and supports interfaces for machine learning and deep learning as well.

Napari[37] is a multidimensional image viewer written in Python. It is one of the fastest growing FOSS tools in the community. Napari aims high extensibility by supporting native integration of well established Python libraries as well as plugin system. It exposes extensible APIs (application programming interfaces) that make developers love Napari. It also utilizes IPython[77] (interactive Python), which enables interactive coding and gives users more freedom. When it comes to deploying machine learning models, Napari comes the first. You can already find many plugins on **Napari-hub**, e.g. StarDist[22, 28], Cellpose[29], ilastik[75], napari-accelerated-pixel-and-object-classification (**APOC**), AnnotatorJ[78], and many others.

BioImage Model ZOO[79] is a website to host and share pre-trained deep learning models for bioimages. It is a community-driven hub built on top of FOSS and FAIR (findability, accessibility, interoperability, and reusability) principles. Developers share their deep learning models via a specification, which is called RDF (resource description file). It describes technical details about the models and can be parsed by a Python library called **bioimageio.core**. It partners with various tools and entities, such as ImJoy[80], ZeroCostDL4Mic[81], DeepImageJ[82], Fiji[14], and ilastik[75], so that users can easily access and use deep learning models.

1.4 LIMITATIONS AND CHALLENGES OF CURRENT SOLUTIONS

Deep learning demonstrated great successes in general computer vision tasks (c.f. appendix 4), as well as in computer vision tasks for bioimages. However, while many general vision tasks can be described as “solved”, the solutions for bioimages demonstrated in the last section still show limitations and face challenges. In particular, they have been struggling in terms of scaling, generalizability, and transferability, i.e. of how versatile they are. While we will see in 2 some quantitative evaluation of that, this section will more broadly present the specificities of biological images with respect to natural images and the problems they pose.

Supervised learning is limiting All the notable segmentation models introduced in the last section, U-Net[20], StarDist[22], Cellpose[29], NucleAIzer[40], and Mesmer[43], are not as versatile as users might wish. The authors of these works, aware of the limitations of their models, propose retraining or fine-tuning them. The fundamental reason is that they are all based on supervised learning scheme and trained with limited source of data. Supervised learning is a learning scheme that requires a full supervision to train a model. It essentially means that the dataset should have annotations, segmentation labels in this case, for every input during training. While supervised learning could result in high general computer vision ability [18, 49], it comes down to the dataset whether the resulting model can be said versatile in respect of more diverse tasks. Aforementioned models were all trained with relatively small sources of data, as small as from a few hundreds [22, 29, 78] to a few thousands [43], which is comparably smaller than 1.3 million images of ImageNet-1K¹⁰.

The most notable deep learning image restoration method, CARE[50], also relies on supervised learning. Consequently, its ability is bound to a fully-curated dataset provided during training too. Originally, this dataset is supposed to be curated with a set of experiments conditioned in specific settings. Denoising solutions based on deep learning actually overcame limitations of supervised learning and evolved using self-supervised learning scheme [51, 83]. Their transition was natural considering that their assumption was that the noise present in a dataset, whose data likely required from either the same or similar conditions of experiments, should be consistent at least within the dataset. In other words, these denoising solutions are not versatile right off the shelf but can become an effective on datasets that has rather consistent and well-behaving distribution of noise.

But overall supervised learning itself is not what limits the ability for deep learning models to be generic. Capability of deep neural networks is unknown for the moment and is still growing [84]. It is rather the lack of a dataset large enough to be called general one. For a simple explanation, VGG[85], a pillar of vision foundation models trained on ImageNet[62], was already able to classify and get general characteristics of 1,000 separate classes in 2015. For its general vision ability, VGG has been used numerous times to extract vision features and used for applications[86, 87, 88, 89, 90]. It shows how much capable an almost decade-old CNN architecture is, once fed with enough data.

¹⁰ ImageNet-1K is a subset of ImageNet database that is used to train so-called foundation models for computer vision, which are considered as general computer vision solutions. Read more about the impact of ImageNet-1K and foundation models trained with it in appendix 4.

Bioimage datasets curated for machine learning are rare Datasets are essentially what make supervised machine learning models. The minimum requirement to make a versatile machine learning model is to have large and general enough dataset to begin with. Therefore, the biggest challenge of making a versatile bioimage model is the lack of large, well curated, and thus representative bioimage dataset. But the size is just one of several factors. In fact, there are large datasets released even before machine learning or deep learning became a major mean to solve computer vision tasks. But the issue is that they are not ready to be consumed for machine learning algorithms, at least under supervised learning setting. Currently, all the big biomedical nuclear or cellular image datasets are all just raw images without or few annotations (BBBC021[55], Cell Image Library[56], and Image Data Resource[57]). Most of useful datasets for segmentation task, meaning having annotations, were curated and released relatively recently. They are too small to cover diverse conditions of image sources [22, 29, 78, 43] and are far from becoming representative datasets. Though the lack of diversity within a dataset is an emerging issue that might cause biases in general machine learning, the situation in bioimage datasets is exacerbated. There are many reasons why large curated datasets have not appeared yet, but they mostly stem from the nature of bioimages themselves.

Biomedical computer vision is unique and diverse Computer vision has been evolved to catch up human vision, and current deep learning models have been showing unprecedented success, for instance, recognizing faces, classifying pomeranians and corgis, *etc.* While researchers are excited about a generalized computer vision and some call it a foundation model[91], transferring models across domains is not easy. In other words, a “generic” vision model, such as VGG16[85] trained on ImageNet[62], simply does not work well or at all on computer vision problems for bioimages, which I will call them biomedical computer vision to differentiate from the “normal computer vision”.

foundation
model

biomedical
computer
vision

Biomedical computer vision has different dimensions, and they are dynamic. When it comes to typical computer vision, images have three dimensions: two spatial dimensions and one for colors, *i.e.* photos. Videos have become a typical vision format and have four dimensions: two spatial dimensions, one for colors, and the other for temporal dimension. These two are the most common “normal computer vision” formats. However, bioimages often have three to five dimensions, and on top of that, have varying channels. Figure 1.13 demonstrates a few examples and the diversity of bioimages. The simplest is a three-dimensional bioimage that has height (Y) and width (X) dimensions and a color channel (C). Four-dimensional bioimages could either have another spatial dimension (Z), depth, or time dimension (T), frame. Lastly, five-dimensional ones have all of them ($TCZYX$).

The number of channels (‘color’) of bioimages is experiment dependent. This fact is related to the principle of fluorescence microscopy and its illumination light source. If you have two fluorophores that get excited from different wavelengths in your experiment, you can essentially have two channel images in the end. It is worth noting that the color response of fluorophores in fluorescence microscopy does not exactly correspond to RGB response of human eyes and should not be treated so. Each channel should be rather treated as independent markers. The cases you get bioimages with RGB channel are ei-

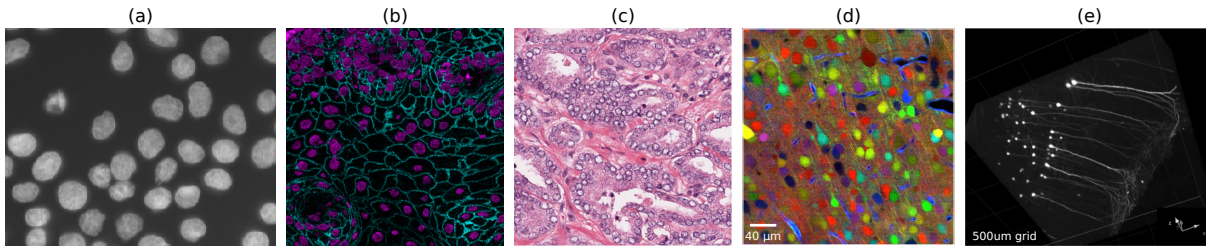


Figure 1.13: Diversity of biomedical image contrasts. (a) Single channel cellular image from [92]. (b) Two channel image with nuclear and whole-cell markers from TISSUENET[43]. (c) H&E-stained pathology tissue image from [93]. (d) Mouse brain image by ChroMS[2] with Brainbow technique[1]. (e) 3D view of a part of a mouse brain with GFP, acquired and processed by *3i* light sheet microscopy from EMBO light sheet microscopy course in 2022.

ther when you cannot separate each channel or when your light source has a full spectrum of light, such as in bright-field microscopy

Lastly, visual features are specific and diverse. The study of shapes and structures of biological objects is a branch of biology called morphology. Morphology takes into account appearance of biological objects, which comprises size, color, pattern, and external structure as well as internal structure. Morphological features are more subtle in biomedical computer vision than normal computer vision and especially so when noise is involved. morphology

All these challenges not only construct special looks of bioimages, but also affect the way to do data science with them. Unfortunately, the discrepancy between the biomedical computer vision and the normal computer vision is one of the main reasons why great successes in computer vision over recent years cannot be directly transferred over to biomedical computer vision tasks. Data science with bioimages has its own issues that I am going to cover in the next section and also throughout the thesis. data science

Curating a bioimage dataset is expensive Upon all the limitations of current machine learning algorithms and all the difficulties of handling bioimages, it is clear that the first answer towards versatile machine learning models should be the dataset. A well-curated large-scale dataset is a starting point for general and versatile models. There is good news and bad news, concerning curating a bioimage dataset. **Good news** is that the biology produces large datasets, suitable for data-driven approaches. Though it takes a lot of time, efforts, money and expertise to produce bioimages, scientists managed to collect a lot of them and are producing even more persistently. Light sheet microscopy, one of the fastest microscopy, can easily generate several TBs (terabytes; 10^{12} bytes) of data within a day. **Bad news** is that machine learning is not only about having big data but a quality data. Putting together or curating a dataset is the actual beginning of machine learning and an underrated job.

Unfortunately, it is much harder to curate a bioimage dataset than normal computer vision datasets for **a number of reasons**. **(i)** It requires specialty and expertise. Recognizing daily objects like cars and trucks is something most people can do it without much effort. Identifying biological objects is not. It relies on their morphology, surrounding objects, experimental context, etc. **(ii)** It lacks platforms and tools. Many datasets were curated by a single group or a few groups in the past. It was easy to organize it

crowdsourcing that way in terms of logistics. As the size of data grows, however, crowdsourcing has become a norm. Crowdsourcing is an approach to outsource individual workers or volunteers through internet. Platforms such as *Amazon Mechanical Turk* and *Crowdsource* by *Google* are representative. These platforms also provide tailored web applications to facilitate curation and to collect it in homogeneous formats. When it comes to bioimages, crowdsourcing is hard to justify. First, it is not reliable to outsource a crowd who does not have much knowledge about the subject. Second, the community is lacking unified tools let alone people to develop them. **(iii)** Image formats and metadata formats are not standardized, yet. It takes time to make consensus over formats, and bioimages were one of those data that were always closed behind the laboratory. It led numerous number of formats and consequently caused, what I would call, file format juggling, which means that you keep converting a file from a format to another to get the job done. I am going to review the format issue again in chapter 2. **(iv)** It is actually expensive. A group of scientists who created TissueNet[43], presently the largest bioimage dataset, estimated that it would have cost \$25,000.

file format
juggling

In summary, curating a large scale dataset is underrated and is especially a predominant issue in bioimages. It is inevitable that machine learning suffers and shows slow progress for biomedical computer vision. I am going to cover data issue more in depth in a separate context in the next chapter.

programming
language

Accessibility and usability matter Suppose that an amazing algorithm has been developed. How can it be made useful in real biology research? In the past, you had no choice but to publish your work on a scientific journal and wait for other scientists to read it. But reading a paper is not the same thing as making and using a program. Thanks to the advance of web technology, it became easy to upload an archive file as a supplementary material, containing code files written in programming languages. Though it can help accessibility, it has little to do with usability. Machine learning models are predominantly developed and deployed in Python programming language, which makes barrier to users, especially to those in biology, to use these models. It is the user interface that can boost usability. In particular, developers may consider distribute or deploy their algorithms or models as an application with graphical user interfaces (GUIs). But making a user interface is demanding and not trivial, and it is an extra work for developers. In the end, many useful algorithms become difficult to access and are left unused. Figure 1.14 describes reciprocal relation of developer's effort and user's effort.

Where to deploy is also an important decision to take as a developer. In bioinformatics, tools are extremely fragmented because sometimes commercial microscopes come with their own software suits. Not only that, their software development kits (SDK) are often not available to individual developers. The same problem recurs in commercial software. Developing a full standalone application is possible but practically ineffective and has low visibility in the presence of so many tools around. Fortunately, free and open-source software (*FOSS*) movement has been growing fast in recent years as well as supporting individual developers with *plugin system*. We will shortly see these options.

Overall, we saw that acquired biological images can be very diverse, acquired using many protocols, methods and biological models to try and answer a great diversity of biological question. While machine learning is a natural candidate to perform automated quantifi-

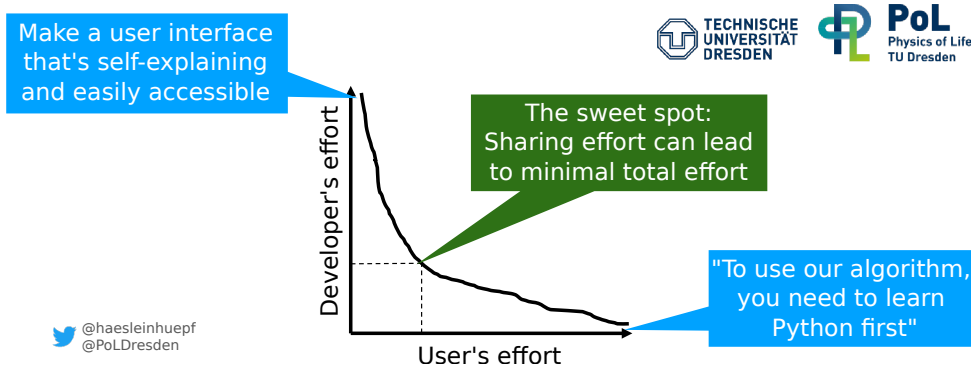


Figure 1.14: Deploying apps or scripts? Scripts are easy to write and light to distribute, but not necessarily easy to use. Applications provide user interfaces for easy usage. But making them is a full-time work. A proper documentation and a simple command line interface, for example, can reduce user's effort by great amount. Source: A presentation slide from EMBO light sheet microscopy course in 2022 by R. Haase

cation, current state-of-the-art methods suffers from many limitations preventing them from being as versatile and general as we would wish. We will see in the next section what methods are proposed in state-of-the-art machine learning to go beyond supervised learning and expensive processes of curating large datasets.

2 GENERAL TRENDS OF COMPUTER VISION SOLUTIONS

Deep learning certainly made big progresses in data science and data mining. But it seemed that deep neural networks can generalize things quite well in computer vision and natural language processing tasks, in particular. The question have always been, however, *how far we can push deep learning*. Once researchers reached deep enough layers in neural networks, they realized that the answer was not about the size of neural networks anymore. The key was more fundamental, that was **how to train them**. The way researchers conducted machine learning was mainly supervised learning, and it worked very well, until they started facing issues. The response was self-supervised learning to make a generalized model, or a “foundation” model[91], and to apply transfer learning or domain adaptation techniques afterwards to transform the base model into practical ones. The argument between supervised learning and self-supervised learning is a question of memorization and generalization.

self-
supervised
learning
transfer
learning
domain
adaptation

2.1 SUPERVISED LEARNING DOMINATES

Supervised learning has been the way to get the best machines. With a decent amount of data, machines seemed to figure out common traits of given data and learn general rules. The figure 1.15 has straight lines, each of which represents a general rule, that can classify two classes¹¹. This notion of finding general rules or patterns is called generalization. Generalization is what we hope for machines to learn and not only to memorize. It turned out that supervised learning could generalize quite well, especially once combined with deep neural networks or deep learning method, as we saw in the previous section as well as in appendix 4. However, as the size of data gets larger, more diverse, and more complicated, it started showing drawbacks and its innate limitations.

generalization

One immediate problem is that you cannot supervise everything. Teachers want some degree of **autonomy** from their students. You really learn something once you do your homework on your own. Supervising everything could also bear a risk of overfitting, means that the student memorizes problems and solutions altogether. Problems you face in the wild is not as clean as those from classrooms. You would encounter many small bumps that your teachers have never mentioned.

Another problem is the **bias**. By construction, a dataset tends to have a purpose or a goal, and it is especially so for supervised learning. As a designer or a curator of a dataset, you would make some decisions subconsciously. They accumulate and form biases. For instance, a gender bias is already a known issue in many datasets[94].

An individual owns limited amount of knowledge. And there is always **gray area**. Some people might not be familiar with breeds of dogs. But there is a dataset to classify 120 different breeds[95] of the dog, and there are supervised models using this data, which can classify dog breeds far better than these people. To them, they are simply dogs, and that is not a wrong answer. But it could be in supervised learning. In other words, there is no uncertainty in supervised learning. You do not have options for “I do not know” or

¹¹ This type of task to classify data is called classification, and it is another pillar that constitutes machine learning together with regression.

“I am not sure”¹².

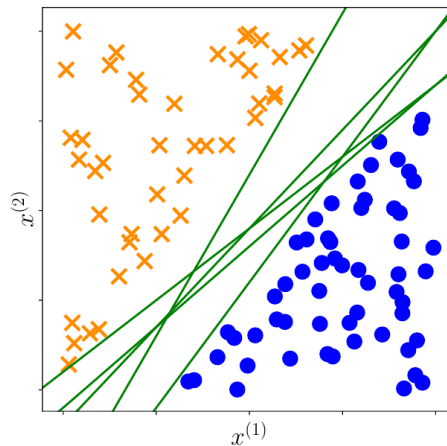


Figure 1.15: Linearly separable two-class data. The general rule that machines learned would be that any points on the left side of the lines belong to orange and x group and those on the right side belong to blue and o group. This image actually represents a supported vector machine (SVM) from the source[98]. Green lines are actually hyperplanes constructed on the supported vector space. SVM used to sit on the alter of machine learning before deep learning gained a momentum.

2.2 UNSUPERVISED LEARNING IS PURSUED

Building a chatbot has always been one of the biggest problem in computer technology (read more details in appendix 1.3.). There are rules, but there are just too many and all kinds of exceptions are accepted. Basically, we cannot consider all the possible cases, not to mention to memorize them all with brute-force approach. We want models to have autonomy, less bias, and embrace gray area. It means that supervised learning would not be an optimal choice, and it was NLP (natural language processing) community where self-supervised learning started thriving around 2017. Self-supervised learning is a type of unsupervised learning, which means it does not require a dataset paired with curated answers. To narrow down the definition of self-supervision, *at least in this thesis*, I am going to present other types of learning because these terminologies are relatively new, sometimes overlapping with others, and still going through changes¹³: weak supervision, semi-supervision, and also clustering. For an easy understanding of their relationship, see figure 1.16.

unsupervised learning

Clustering is what has been representing unsupervised learning and used to be used almost as a synonym for unsupervised learning, because most algorithms that were not supervised were to solve clustering problems in the past. As an example, there is KMeans clustering algorithm that assumes K number of clusters and assigns memberships to each data point by calculating distance among their attributes or features.

features

¹² There is a technique called label smoothing[96], which became popular in many models. Instead of giving a definite answer, label smoothing makes it less confident. It may help generalization but inevitably makes models less confident. It was reported that label smoothing could bring negative impact on knowledge distillation[97], in which a teacher network distills knowledge to a student network.

¹³ The boundaries of these definitions are very thin, therefore they could be defined differently in other literatures.

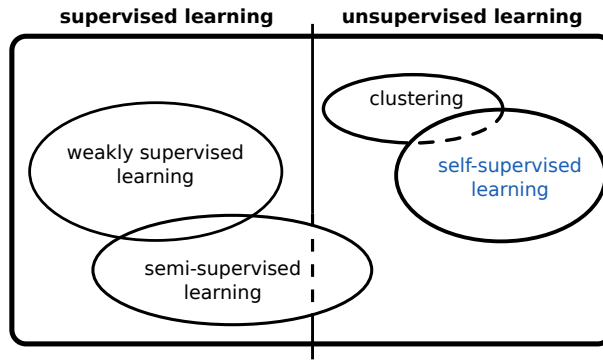


Figure 1.16: Non-exhausted types of learning and their relationship. Unsupervised learning refers to what is not supervised learning. Self-supervised learning is emphasized in blue because it is the main interest. Dotted lines mean loose relationships.

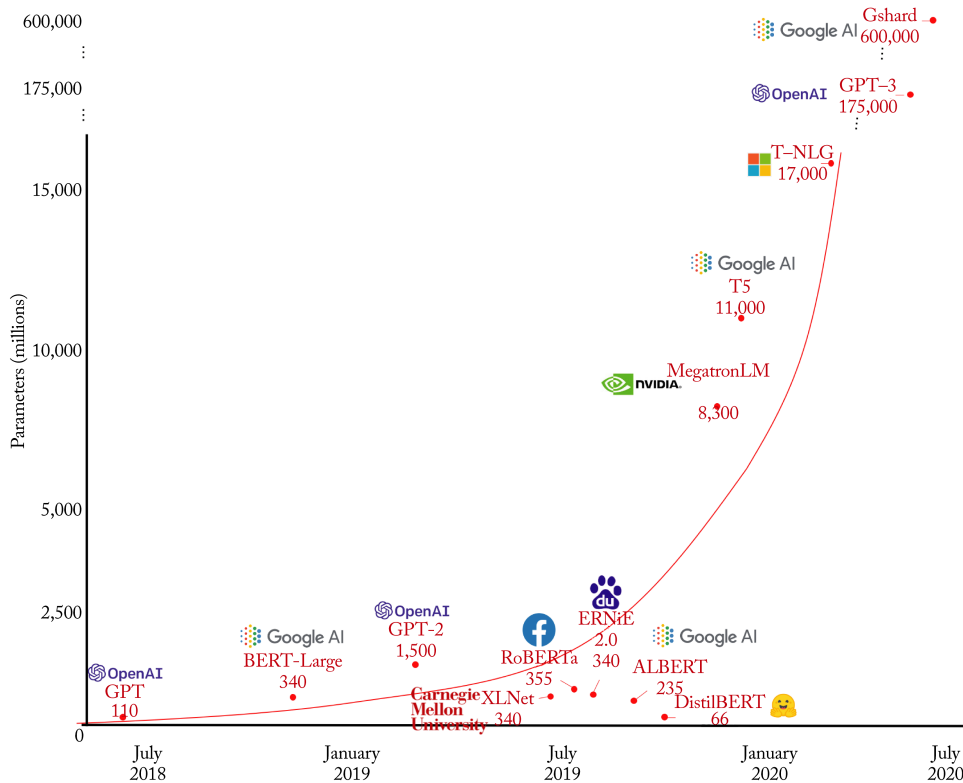


Figure 1.17: Scales of self-attention language models (a.k.a. TRANSFORMER). Models before TRANSFORMER are not even comparable in terms of the number of parameters to those after TRANSFORMER, but the number kept soaring. Around BERT, the size of models started increasing exponentially, and models after GPT-3 have literally out-of-chart sizes. The graph covers language models from July 2018 to July 2020. The graph was sourced from [99], published sometime in 2021 before *Facebook* changed their name to *Meta* in October 2021. *Meta* joined the game later and released their large language model, named OPT[100], which also has 175B parameters to its largest, same as GPT-3.

Semi-supervision is simply a mixture of supervision and unsupervision. Usual cases are to train your model in supervised way with available answers or labels then to train further in unsupervised way because no label is available. The opposite, that is to train in unsupervised way first then to do supervision, is also possible, and it has become popular recently and is not referred as semi-supervised learning anymore. People rather call it

transfer learning that transfers learned knowledge to solve bigger or other problems.

transfer
learning

Weak supervision is a type of supervised learning where you do not provide full labels. For examples, you could label a dog as an animal or non-human, or label 3 people out of 10 people. It has loose connection to semi-supervision, especially for the latter case. In addition, it is usually employed as an initial or an intermediate step to fully supervised learning. It is easy to imagine that machines from this approach would end up suboptimal, but it generalizes better than fully supervised learning and has its place.


Self-supervised learning, last but not least, explores data without labels, and finds (or predicts) relationship within. Clustering can be considered as a self-supervision depending on how you look at it. However, while clustering has its goal at finding clusters and distances of data points to them, self-supervised learning aims to find solely relationships or similarities of data points and do no further.

2.3 SELF-SUPERVISED LEARNING ARRIVES

Y. LeCun's cake Y. LeCun is a renowned computer scientist who received Turing Award together with J. Bengio and G. Hinton for their contribution to deep learning in 2018, which is considered as “Nobel Prize of Computing”. He expressed the importance of self-supervised learning at the NIPS conference (The Conference and Workshop on Neural Information Processing Systems) in 2016. He made an analogy between self-supervised learning and a cake. His cake is in figure 1.18. It roughly summarized three learning methods by the amount of information models get during training: reinforcement learning, supervised learning, and self-supervised learning. He made a point that supervised learning is like licking only the icing of the cake, which is important but represents only the surface of a cake. Compared to supervised learning, self-supervised learning provides far more information during training and is the actual cake we would like to conquer.

How Much Information is the Machine Given during Learning? Y. LeCun

- ▶ **“Pure” Reinforcement Learning (cherry)**
 - ▶ The machine predicts a scalar reward given once in a while.
 - ▶ **A few bits for some samples**
- ▶ **Supervised Learning (icing)**
 - ▶ The machine predicts a category or a few numbers for each input
 - ▶ Predicting human-supplied data
 - ▶ **10→10,000 bits per sample**
- ▶ **Self-Supervised Learning (cake génoise)**
 - ▶ The machine predicts any part of its input for any observed part.
 - ▶ Predicts future frames in videos
 - ▶ **Millions of bits per sample**



© 2019 IEEE International Solid-State Circuits Conference 1.1: Deep Learning Hardware: Past, Present, & Future 59

Figure 1.18: LeCun's cake. This is a slide from IEEE International Solid-State Circuits Conference in 2019, but it is identical to the one from his talk in NIPS2016 (Video link: [101]).

self-attention
Transformer

I believe that the term “self-supervised learning” must have existed for long time, however, it was not until one important paper in 2017 that everyone started differentiating it from other learning methods and mentioning it. That paper by *Google* introduced what’s called self-attention module specifically in deep learning and its resulted model, named TRANSFORMER[102]. And you could see the power of self-supervised learning and TRANSFORMERS in my conversation with GPT-3[84] in the appendix at this section (NLP example at introduction [?]). Researchers, from then on, developed large language models (LLMs) based on TRANSFORMER, such as BERT[103] by *Google*, GPT family[104, 84] by *OpenAI*, and OPT[100] by *Meta* (previously *Facebook*), thanks to its amazing generalizability followed by its massive scalability. Speaking of scale, GPT-3 model has 175 billion parameters for its architecture, used around 300 billion tokens (*token* is a unit almost equivalent to word), and it would have taken a several hundred years¹⁴ to train. Figure 1.17 shows how the scale of the language model has evolved.

query, key, value
dictionary look-up
attention

Dictionary look-up is the core idea of self-attention[102]. Its concept can be easily found in database software. To navigate data and retrieve certain information, you would provide a query that matches a key. With the key, you can access to its value, the actual information. When it is applied to representation learning, dictionary look-up looks like in figure 1.19. First, it encodes query (Q), key (K), and value (V) representations. Then it calculates similarity (A) of a given query with all the keys. The final attention (M) is determined by the value associated to the query and the similarity.

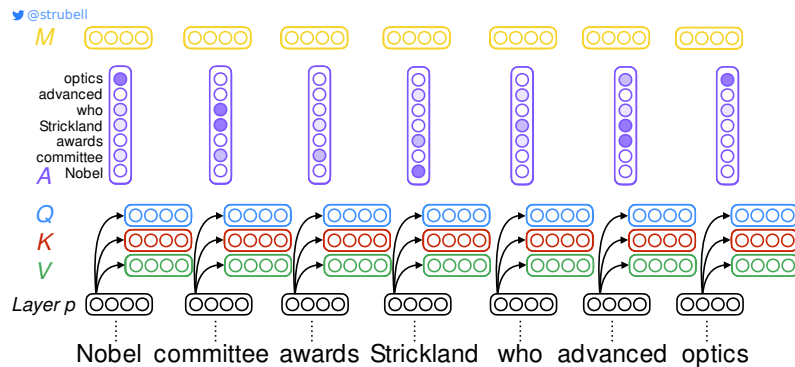


Figure 1.19: Self-attention module[102]. Layer p refers to any hidden layers in a neural network. Dictionary look-up is a key idea of self-attention. It generates three embeddings: query (Q), key (K), and value (V). The attention (M) represents relationship and importance among tokens. Read **dictionary look-up** section in the main text for more explanation. Diagram by E. Strubell.

2.4 SELF-SUPERVISED LEARNING IN COMPUTER VISION

Self-supervised learning quickly became dominant in NLP for years now, but its main mechanism, known as self-attention, was not directly adapted right away to computer

¹⁴ Obviously, it did not take hundreds of years. *OpenAI* parallelized computing across processing units, mostly graphical processing units (GPUs), on a cloud service. The amount of required computation to train GPT-3 was 3.14×10^{23} FLOPS (floating point operations per second; a common performance unit for GPUs) and the GPU product they chose was *Nvidia* V100, which has 14 TFLOPS (tera-FLOPS; 10^{12}). With simple arithmetic, a single V100 GPU gives around 711 years. It means that in order to train a GPT-3 within a month, you need about 8532 GPUs. Some estimated that the cost must have been several millions of dollar (do not quote me on that). Note that this calculation is all theoretical.



Figure 1.20: Evolution of inpainting vision task. (a, b) Before contrastive learning v.s. (c) After contrastive learning. They all predict occluded area. The most notable difference is quality. Note that (c) actually has two stages: first to reconstruct a segmentation mask and finally to generate an image on top of it. Source: (a) [106], (b) [107], (c) [108]

vision due to intrinsic difference in data types¹⁵. But rather, it was its idea that was first applied to so-called contrastive learning. For that, we will have a look at the inpainting contrastive learning application to see how self-supervision was manifested in computer vision. Inpainting inpainting vision task¹⁶ is also called image completion, and its goal is to fill blanks or occluded parts in an image. See figure 1.20.

Inpainting before contrastive learning Early forms of unsupervised learning in computer vision were closer to autoencoder. CONTEXT ENCODER[106] ((a) in figure 1.20) was an unsupervised CNN for the inpainting application that looked like a variant of autoencoder. During training, it took patches out of complete images and was provided pairs of an image with a blank and its missing patch as a target. In essence, CONTEXT ENCODER reads images around the blank and learns how to fill it. It worked in some cases, but the results, in general, looked awkward and not plausible enough because it had a lot of visual artifacts. To mitigate these artifacts, a group integrated a state-of-the-art generative model, namely GAN (generative adversarial network). Their model[107] ((b) in figure 1.20) consisted of three networks: one generator, and two discriminators, each of which takes care of a local context and a global context. GAN and the separation of context encoders improved the quality by a large margin, but it was still a type of autoencoder at its core.

Autoencoder can be considered as a self-supervised learning without a doubt, but there is little room of uncertainty. The target is too specific and there exists only one answer for each corruption during training, at least, for two inpainting applications above. In terms of the amount of information during training, this approach was still licking the icing.

¹⁵ Direct translation of self-attention from NLP to computer vision is ViT (Vision Transformer)[105].

¹⁶ There is outpainting application too. Currently, outpainting refers to two applications: one for completing an image from a rudimentary sketch, the other for expanding the context, or surrounding, of a given image.

Contrastive learning Contemporary state-of-the-art self-supervised vision models are mostly based on contrastive learning. Its goal is to learn generalized visual features, not necessarily for a particular vision task, *e.g.* inpainting. Once you achieve that, utilizing it to multiple vision tasks should be easier through techniques such as transfer learning and domain adaptation. It is built upon **three key ideas**: representation learning, InfoNCE, and dictionary look-up. Though contrastive learning is not the same as self-attention[102], it shares a lot of similarity. In particular, it is apparent that InfoNCE and dictionary look-up idea are inspired by self-attention.

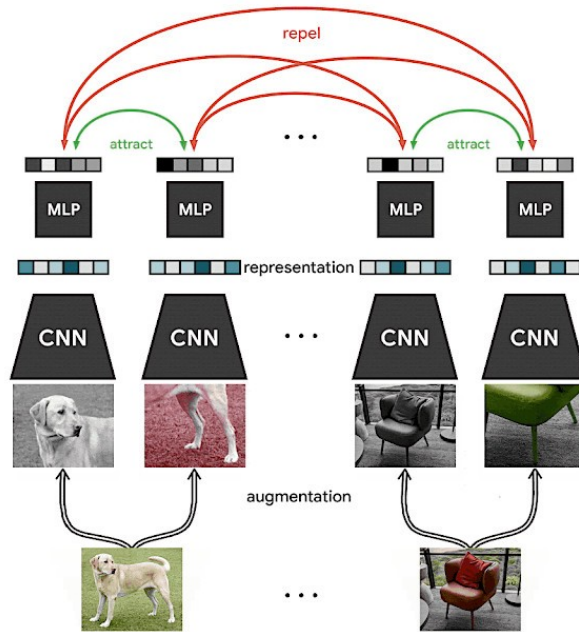


Figure 1.21: InfoNCE employs representation learning. This particular diagram demonstrates SimCLR[109] that adapted InfoNCE and used data augmentation to have $2N$ samples from a batch (size N) for a single forward pass. For each data point, the loss considers one positive case (*attract*) and $2(N - 1)$ negative cases (*repel*). Batch size (N) is not limited, but it usually ranges from hundreds to thousands. Notice that SimCLR does not use dictionary look-up approach. Source: SimCLR[109] and its [blog post](#)

negative sampling

InfoNCE[110] is a loss that gave rise to contrastive learning. The group brought in representation learning and negative sampling together. They called high-level latent representations (features from deep layers; see figure A.18) slow features that contain a global structure of data thus less sensitive to local noise variations. Even though the slow features could provide richer signals during training and help generalization, the key contribution of InfoNCE is the negative sampling. They sampled N data points which comprise one “positive” sample and $N - 1$ “negative” samples. In comparison to previous approaches, negative samples allow roughly $\times N$ signals to the training process. This approach is similar to self-attention in the sense that it compares multiple embeddings¹⁷ and exploits their similarities like mutual information in InfoNCE. Figure 1.21 shows how SimCLR[109] (for simple contrastive learning) works. InfoNCE was already referred as contrastive learning, but in practice it also considers dictionary look-up.

¹⁷ In language models, a latent variable or a hidden representation is often called embedding, not as a map in computer vision.

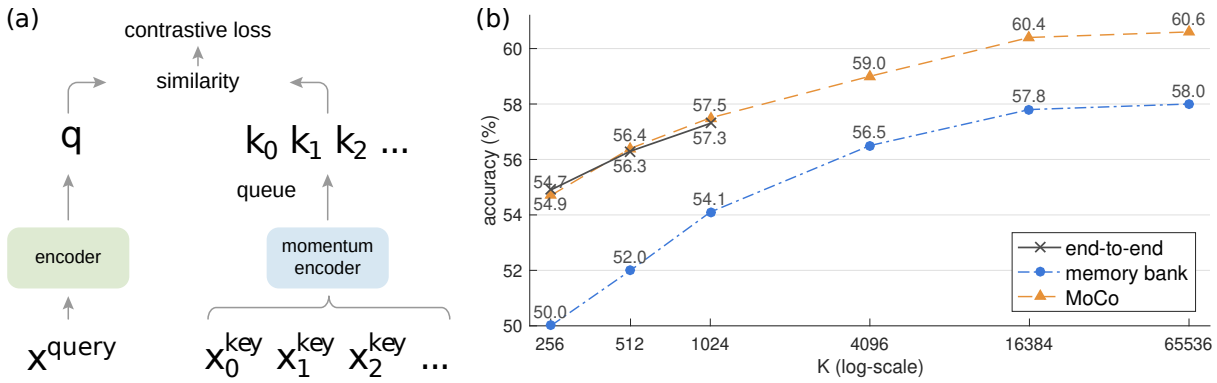


Figure 1.22: Momentum contrast (MoCo)[111] is the current state-of-the-art contrastive learning framework in computer vision. (a) It successfully incorporated dictionary look-up idea from self-attention[102] on top of InfoNCE loss[110] using so-called dynamic dictionary and queues. (b) The accuracy of ImageNet classification task. Consequently, it could achieve unprecedented performance beyond the numbers of samples (K) that had not been possible in a single forward pass. *End-to-end* approach refers to classical contrastive learning (InfoNCE) where K equals to the mini-batch size. *Memory bank* used a static dictionary and no momentum.

In the end, it was **MoCo**[111] (momentum contrast) that finally applied dictionary look-up idea to InfoNCE and became a current state-of-the-art contrastive learning framework in computer vision. Its biggest contribution is the *dynamic dictionary*, which is an adaptation of self-attention. When it comes to self-supervised learning, the scale is more important than to supervised learning. Compared to languages, it was hard to construct a large batch of images due to memory constraints. The dynamic dictionary essentially decouples the dictionary size from the mini-batch size, which in turn makes the dictionary look-up approach more affordable in computer vision. A static dictionary pre-computes keys, which means that it will not be trained. Momentum encoder slowly adjusts the dictionary so that it can be trained as well. See figure 1.22.

Transfer learning and Domain adaptation Self-supervision frameworks such as TRANSFORMER [102] and contrastive learning[110, 111, 109] are not actually directly used in real tasks. Even though TRANSFORMER is known as a zero-shot learner, which means that it does not need to be fine-tuned, it usually goes through additional shots of learning (a.k.a. a few-shot learning). Their primary goal is to build a foundation[91] that holds general rules of a certain type of data, such as the language or the vision, so that people can easily branch it to more sophisticated tasks. Foundation models have huge advantages: it is generalized, versatile, relatively unbiased, significantly reduces research cost¹⁸, and actually more performant. It is usual that they are served as a pretext task and go through a fine-tuning process for a downstream task. Fine-tuning refers to a process of taking an already trained model and training it further. Transfer learning and domain adaptation are two representative techniques frequently used in production.

Transfer learning transfers knowledge from a task, with which a model was trained, to other tasks. In practice, in deep neural networks, the process is following. **First**, you would grab one or more large scale dataset(s) and train a model no matter the task. Once the model was trained, we could assume that it could recognize the similar type of data

¹⁸ Training a large scale model may cost millions of dollars.

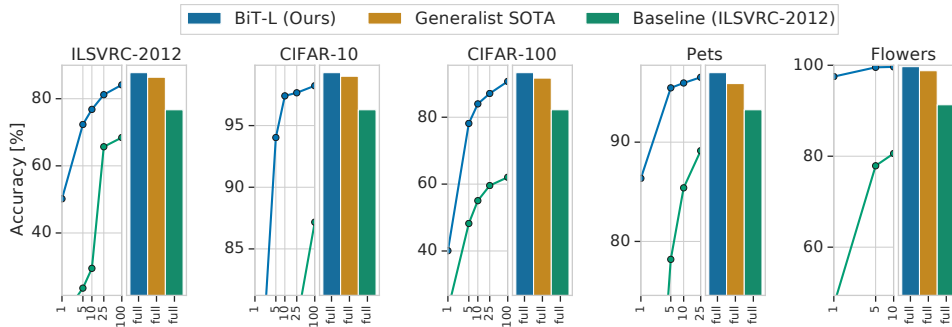


Figure 1.23: Effectiveness of transfer learning. BiT[112] refers to transferred models. The group trained BiT with JFT dataset[113] which contains 300M images and tested an image classification task with 5 different datasets by transfer learning. The x-axis indicates the number of images per class for fine-tuning process. Source: *Big Transfer (BiT): General Visual Representation Learning*[112]

that you trained it with. **Second**, you would divide your network into two pieces: a body and a head. Head is the last or penultimate layer(s) responsible to transform latent representations for the loss you set. **Third**, you would simply discard the head and keep the body which is capable of encoding generalized representations. **Lastly**, you would construct a new head for a downstream task you want, and may want to fine-tune it a bit. Transfer learning is becoming more and more common not just because the cost of training a large-scale model started soaring exponentially, but because it actually helps to push performance, as demonstrated in figure 1.23.

domain adaptation

Domain adaptation considers relatively small changes in comparison to transfer learning. A given pretext task stays the same and only the data changes. The shift that occurs when data distribution changes is called domain shift. See figure 1.24.



Figure 1.24: Examples of domain shift. Here are presented 6 different datasets. Domain adaptation has a goal to adapt a model trained with a *source* dataset and to a *target* dataset. Source: [114]

3 SELF-SUPERVISED LEARNING IN THIS THESIS

As computer vision research is seeking ways out of supervised learning towards self-supervised learning, there are already a few promising frameworks introduced in the previous section. But a common caveat that these frameworks share is that they were explicitly designed as a pretext task. A pretext tasks could help models to get used to data before a downstream task that actually has a practical use but eventually involves supervised learning. But for bioimages, even labels for a downstream task are already hard to come by. The ideal route would be that self-supervised models not only learn input data without any labels but also perform real tasks to solve problems. To develop such algorithms or to construct losses for machine learning approaches, one needs to know fundamental principles of self-supervised learning. Two fundamentals are generative model and representation learning, which will be briefly explained here and will be the main subject in chapter 3.

3.1 GENERATIVE MODEL

Generative model is, in a way, a machine learning itself. If supervised learning forms a model that links answers from given inputs, a generative model, from a perspective of self-supervised learning, generates new data points by learning a distribution of data itself. A brief introduction to the generative model can be found in appendix 2.

GAN Generative adversarial network[48] introduced what's called discriminator to help model $p(\mathbf{x})$ beside the generator. The generator is a network which I denoted \mathbf{G} in the figure 1.25 to simulate new data from a structured noise. Discriminator, referred as \mathbf{D} , is another network to perform binary classification whether a generated data \mathbf{x}' is real or fake. They play a minimax game that the generator tries to maximize its capability to fool the discriminator while the discriminator will be randomly given a sampled data \mathbf{x} or a generated data \mathbf{x}' each time. The game starts easy, because the generator would not do well at first and the discriminator could easily pick up fake images. This game ends when discriminator is no longer able to tell the given data \mathbf{x}' is fake. This adversarial approach is clever, because modeling $\mathbf{x} \in \mathbb{R}^{\mathcal{D}}$ with large \mathcal{D} is not a trivial task, but the discriminator effectively reduces it down to a binary classification task.

The generated images, as shown in figure 1.26, were incredible since the quality was guaranteed by the objective of the discriminator and the generator was capable enough thanks to a powerful deep CNN. However, GANs almost always experienced mode collapsing, mode collapsing which means that the generated data \mathbf{x}' covers only a part of $p(\mathbf{x})$ (imagine that \mathbf{x}' covers only one of three branches of $p_{\theta}(\mathbf{x})$ in figure 1.25). Conceptually, it means that the generator would create the same types of images again and again however the input \mathbf{z} changes and the discriminator does not even see other types of images. In the end, this generator can only generate images of adult faces, for instance, even though we had a full spectrum of faces. **One** of the main causes of mode collapsing is simply that the objective of GAN allows this case. **Another** fundamental cause is the way of sampling \mathbf{z} . I said that it is a structured noise that becomes the input to the generator. Fundamentally, the generator needs an input and researchers simply sample it from a known distribution, such as the beloved normal distribution. As a result, in this case, the generator learns to map¹⁹ the normal distribution to $p(\mathbf{x})$. In fact, it might have been possible to avoid a map

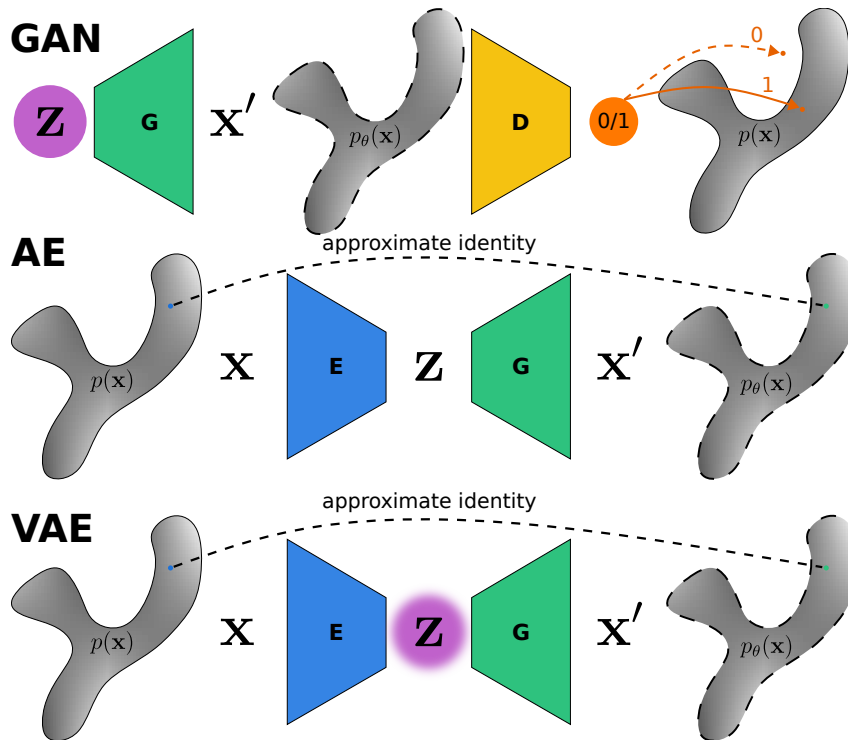


Figure 1.25: Overview of generative models: generative adversarial network (GAN), autoencoder (AE), and variational autoencoder (VAE). An input is \mathbf{x} and an output is \mathbf{x}' . \mathbf{z} is a latent variable. $p(\mathbf{x})$ is a probability density distribution of a dataset and $p_\theta(\mathbf{x})$ is a parameterized approximation of it. In terms of models, \mathbf{G} is a generator (or a decoder), \mathbf{D} is a discriminator, and \mathbf{E} is an encoder. A couple of notes: (i) Probability density distribution $p(\mathbf{x})$ no longer has a nice shape because it has a large dimension \mathcal{D} . It looks 2-dimensional, but it is not. It helps to visualize an arbitrary distribution in a large \mathcal{D} -dimension. (ii) $p(\mathbf{x})$ with a dotted outline indicates approximation, namely $p_\theta(\mathbf{x})$ parameterized by θ . (iii) Latent variable \mathbf{z} can be sampled from a nice prior distribution like the one in a solid circle background, from nice but multiple priors like the one in blurred circle, or from nothing. (iv) \mathbf{z} for GAN is not a latent variable, in fact, and simply an input to the generator. Read each section to learn more. The diagram was inspired by [115].

mode collapsing, if the sampling method had been “correct”. Who does model and sample \mathbf{z} then? It is us, humans. Can we fix it? Maybe, if we optimize sampling methods too. But it should not be us humans, because human intelligence is specialized²⁰. In a word, both the generator and the discriminator are capable enough, but lack of our imagination to sample \mathbf{z} is limiting their potentials.

autoencoder (AE)

encoder
decoder
latent layer
latent variable

VAE Variational autoencoder is a variant of the autoencoder (AE). Autoencoder is a model to learn the way to encode an input and to decode it to reconstruct the input, basically learning identity, therefore “auto-”. As shown in figure 1.25, an autoencoder has a pair of an encoder (E) and a decoder (G), and in the middle, a bottleneck layer, often called latent layer. In principle, the encoder encodes or compresses data \mathbf{x} to small codes, so-called latent variable \mathbf{z} . We can write down how the process of estimation is done,

¹⁹ Mapping is equivalent to making a function, though it usually indicates 1 to 1 transformation.

²⁰ Can you say something completely random languages and somehow cover all the existing languages over the world? I do not think so. It is difficult even to make any sounds that you do not know. It is another evidence that human intelligence is very specialized.



Figure 1.26: Astonishing results from GANs. All images are not real. (left) BigGAN [116]. (right) StyleGAN[117, 118], known as “thispersondoesnotexist”. It is a webpage hosting generated face images from StyleGAN2[118]. The address is <https://thispersondoesnotexist.com/>. Refresh the page to see other faces once you are there.

involving the generation process from \mathbf{z} :

$$p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z} \tag{1.2}$$

We can interpret above equation as follows: (i) we find the distribution of \mathbf{z} , namely $p_{\theta}(\mathbf{z})$, (ii) we scan every \mathbf{z} and generate \mathbf{x} ($p_{\theta}(\mathbf{x}|\mathbf{z})$) to make an estimation $p_{\theta}(\mathbf{x})$ for the data distribution $p(\mathbf{x})$. In practice, we do not get to choose $p_{\theta}(\mathbf{z})$ nor have any constraints on \mathbf{z} . As a result, the latent space ($\mathbf{z} \in \mathbf{Z}$) where the latent variable \mathbf{z} lies in, becomes too large and unpredictable. Instead, we have direct access to both $p_{\theta}(\mathbf{x})$ and $p(\mathbf{x})$. Hence, we can simply compare \mathbf{x}' to \mathbf{x} . Though it is useful to have a concise feature of a data in a low dimension in the latent layer, we end up having a huge latent space and have *little control* over it²¹.

Variational autoencoder is an attempt to model the distribution of the latent variable \mathbf{z} from a known distribution to have control over it. See the diagram of VAE in figure 1.27. **First**, the encoder maps \mathbf{x} to \mathbf{z} , and its process can be written as $p_{\phi}(\mathbf{z}|\mathbf{x})$, same as the autoencoder. But VAE does not do it that way. Instead, VAE estimates $p_{\phi}(\mathbf{z}|\mathbf{x})$ with a prior distribution, and we will call the estimated posterior distribution $q_{\phi}(\mathbf{z}|\mathbf{x})$. Simply put, we would like to condition the encoding method with a known distribution, and the popular choice of $p(\mathbf{z})$, as you may have guessed, is the Gaussian distribution. But if we were to sample \mathbf{z} randomly from a Gaussian distribution, it loses connection to the encoding process, which means that we cannot use the gradient descent to optimize it. It is the reparameterization trick that fixes it, and it lets the encoder samples \mathbf{z} , not us. It is called “reparameterization” because it uses the output of the encoder, \mathbf{z} , and use it to parameterize $q_{\phi}(\mathbf{z}|\mathbf{x})$ with a multivariate Gaussian distribution with a diagonal covariance $\mathcal{N}(0, I)$. At the end of the day, the reparameterized \mathbf{z}' looks like a set of Gaussian

²¹ It is generally true that a variable in a high dimension has a *bigger* space than that in a low dimension. But in the case of autoencoder, the dimension of the latent space is given as a hyperparameter and fixed, and researchers want it small. Yet, it could grow sparse and large in terms of the volume.

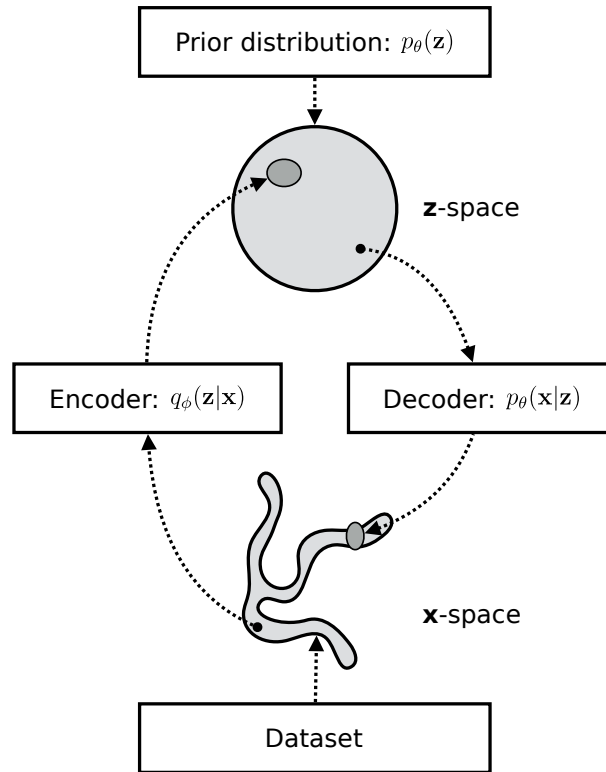


Figure 1.27: Variation autoencoder. Source: [119]

distributions²² in figure 1.28. Notice that what was a vector \mathbf{z} becomes a set of samples from a multivariate Gaussian distribution, \mathbf{z}' .

$$\begin{aligned}
 (\boldsymbol{\mu}, \boldsymbol{\sigma}) &= \mathbf{z} = p_{\phi}(\mathbf{z}|\mathbf{x}) \\
 \boldsymbol{\epsilon} &\sim \mathcal{N}(0, \mathbf{I}^2) \\
 \mathbf{z}' &= \boldsymbol{\mu} + \boldsymbol{\sigma}\boldsymbol{\epsilon} \\
 \mathbf{z}' &\sim q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}^2)
 \end{aligned} \tag{1.3}$$

variational
inference

Second, now we can describe a data distribution $p_{\theta}(\mathbf{x})$ from a known latent variable \mathbf{z}' by replacing \mathbf{z} in equation 1.2. Accordingly, its generation process is referred as the variational inference $p_{\theta}(\mathbf{x}|\mathbf{z}')$. Furthermore, this latent variable is not totally random and has connection to the encoder so that we can calculate gradients to train a VAE.

Third, the loss will have another term to approximate the posterior distribution $q_{\phi}(\mathbf{z}|\mathbf{x})$ to $p_{\phi}(\mathbf{z}|\mathbf{x})$, in addition to maximize the log-likelihood of data $\log p_{\theta}(\mathbf{x})$.

$$\mathcal{L}_{VAE}(\theta, \phi) = -(\log p_{\theta}(\mathbf{x}) - D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\phi}(\mathbf{z}|\mathbf{x}))) \tag{1.4}$$

Advantages of VAE are clear. **(i)** VAE is mathematically sound. The objective is also known as the variational lower bound from variational Bayesian methods and has long-lived. **(ii)** It is more explainable. One issue that people keep raising questions against

²² We can consider this multivariate Gaussian distribution as a set of Gaussian distributions, because they are not correlated.

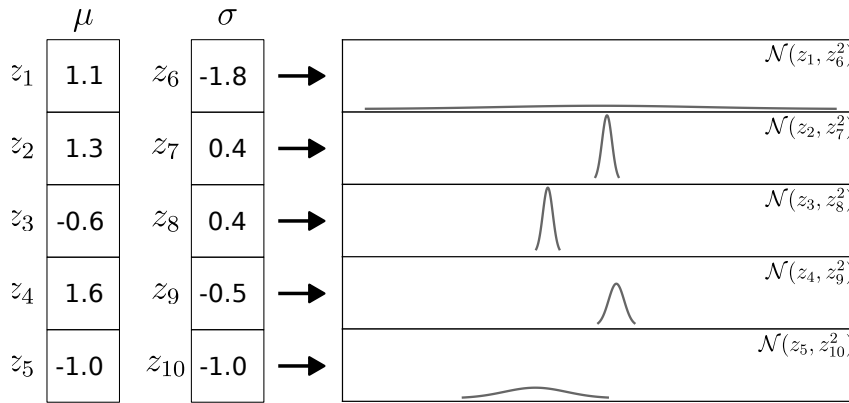


Figure 1.28: An example of reparameterization trick with numbers in equation 1.3. $\mathbf{z} \in \mathbb{R}^{10}$ is a vector that has 10 dimension. Through the process, the reparameterized latent variable \mathbf{z}' is sampled from a multivariate Gaussian distribution constructed by \mathbf{z} . Usually, $\mathbf{z}' \in \mathbb{R}^5$ has half dimension (not necessarily). The reparameterization trick is why VAE is sometimes called a “stochastic mapping”.

deep neural networks is that they are like *black boxes* that give us little explanation how they work. VAE has separate parts that serve separate roles: an encoder, a decoder, and a latent variable. (iii) We can actually manipulate the latent variable to generate *somewhat* intended results to a degree. Figure 1.29 demonstrates β -VAE[120] which was designed to enforce disentanglement of the latent variable.

There are disadvantages, too. Variational inference is nice to gain control over the latent space, but it is compromising the quality at the same time because of the stochastic nature of the reparameterized latent variable (\mathbf{z}'). Besides, the prior distribution yet has a simple form just like GANs, while the real data distribution (or a resulting posterior distribution) is presumably much more complex than that. As you may have noticed in figure 1.29, the results become blurry and not as realistic as those from a GAN.

3.2 REPRESENTATION LEARNING

Representation learning is an important component for incorporating uncertainty to training process. Again, learning is not solely about a memorization. When we recognize a chair, we construct an abstraction of it, *e.g.* legs, low height, sometimes a backrest, then you can recognize other chairs. Autoencoder usually has targets in the data space which is a direct observation. As we saw throughout section ??, deep CNNs are powerful abstraction machines. So instead of constructing a loss in data space, representation learning does it in latent space or feature space to learn the abstraction.

Deep convolutional neural networks (CNNs) excels at encoding data and its features. Resulted feature maps from each hidden layer are representations of a given data. Style transfer vision task became viral because of its expressiveness and visually plausible results. It did so, utilizing representation learning at its full potential. I will cover basic ideas of neural style transfer (style transfer using deep neural network learning), but it will be discussed further in chapter 3.

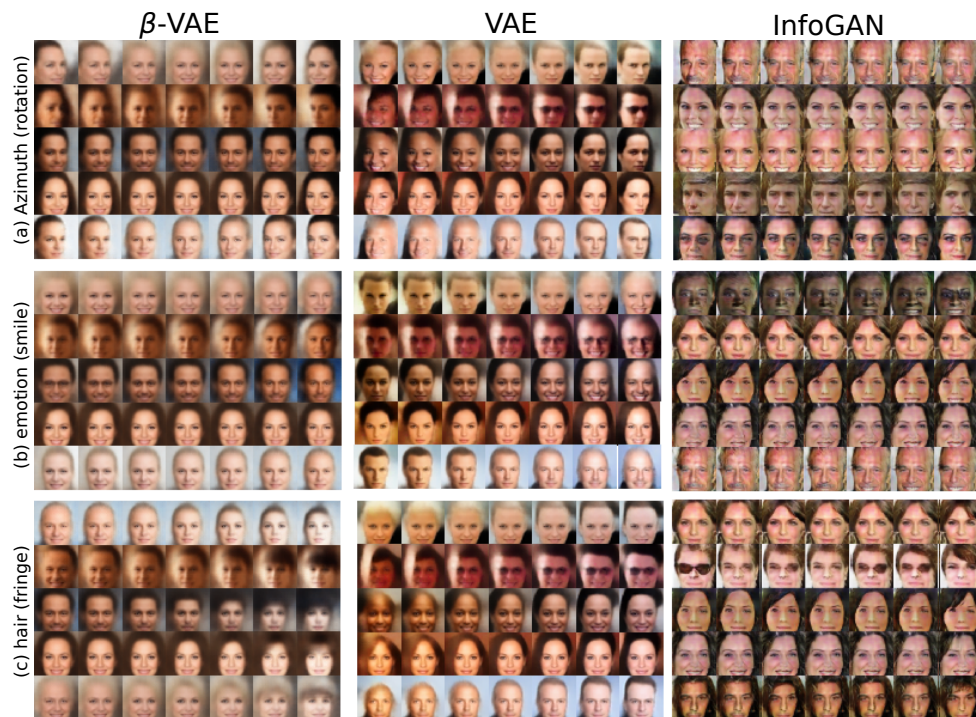


Figure 1.29: β -VAE[120] investigated disentanglement of the latent variable \mathbf{z} in VAE. *Each row represents a different seed image used to infer the latent values in the VAE-based models, or a random sample of the noise variables in InfoGAN[121].* In general, changing a dimension of the latent variable does not mean changing a single attribute that we perceive. For instance, the rotation and the emotion change simultaneously in VAE. In addition, the rotation attribute is hardly possible to manipulate except in β -VAE. Note: italicized sentence was kept original from the source[120].



Figure 1.30: What is a texture or a visual feature? These are all images of corks, randomly sampled from KTH-TIPS database[124]. They look different, but there is something that we perceive them all as corks. Julesz ensemble supposes that there will exist a set of filters whose responses are all identical across these images.

Julesz ensemble and neural style transfer Style transfer is categorized as texture modeling and representation learning.

Julesz ensemble[122, 123] is a concept from texture modeling and a fascinating one. It says that images are identical if their visual features are the same (it may sound obvious), or an image is an averaged response of many visual features (this one may sound obvious too). Let's consider a filter F_l that can extract a certain feature from an image \mathbf{x} , a discrete pixel space $\Omega = \{1, \dots, H\} \times \{1, \dots, W\}$, and a location $u \in \Omega$. Thus, we can express a feature from a filter at a location as $F_l(x, u)$. The overall observation μ_l through a filter F_l is a spatial average of a feature over all locations $\mu_l(\mathbf{x}) = \sum_{u \in \Omega} F_l(\mathbf{x}, u) / |\Omega|$.

Now, let's imagine two images \mathbf{x}_A , \mathbf{x}_B , and a filter F_l . If you wear this filter and their visual observations look the same $(\mu_l(\mathbf{x}_A) - \mu_l(\mathbf{x}_B))^2 < \epsilon$ (ϵ is a number small enough to ignore), can you say that they are identical? No, a single filter might not be enough. But what if you have a bank of filters $\{F_1, F_2, \dots, F_L\}$ and say:

$$\begin{aligned} \mu_l(\mathbf{x}) &= \sum_{u \in \Omega} F_l(\mathbf{x}, u) / |\Omega| \\ \mathcal{L}(\mathbf{x}_A, \mathbf{x}_B) &= \sum_{l=1}^L (\mu_l(\mathbf{x}_A) - \mu_l(\mathbf{x}_B))^2 < \epsilon \end{aligned} \tag{1.5}$$

Then, yes, it is likely that they are identical. Julesz ensemble is a set of images that satisfies the following condition:

$$\mathcal{J}_\epsilon(\mathbf{x}') = \{\mathbf{x} \in \mathcal{X} : \mathcal{L}(\mathbf{x}, \mathbf{x}') < \epsilon\} \tag{1.6}$$

Say \mathbf{x}' is a cork image (see figure 1.30), then $\mathcal{J}(\mathbf{x}')$ represents all cork images. What's interesting is that, in reverse, we can think that an image \mathbf{x} is an averaged response of a bank of filters, like following:

$$\mathbf{x} = \sum_{l=1}^L \frac{\mu_l(\mathbf{x})}{L} \tag{1.7}$$

If you think of it, this is exactly what convolutional neural networks (CNNs) are doing. We could consider each layer as a filter F_l and its feature map as μ_l . Example from a CNN in figure A.21 is also a Julesz ensemble.

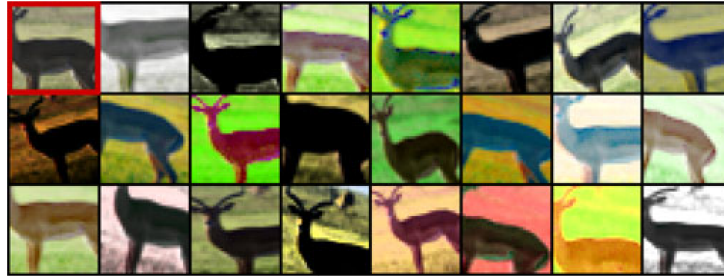


Figure 1.31: They are image patches augmented from a single image. The one in a red box in the top left corner is the only patch not augmented. Used augmentation are translation, scaling, contrast and color changes. Source: [125]

data aug-
mentation

Representation learning is a learning method to learn representations instead of data itself in data space. Data ($p(x)$) is an observation of a thing. Depending on many conditions, its observation can change small or dramatic. The idea is to learn an abstraction, an essence ($p^*(z)$), or an invariant feature. Figure 1.31 shows a set of images augmented from a single image. Data augmentation technique simulates different aspects of data while we want that it will not change the essence of data. Just as we saw in Julesz ensemble, we could assume a feature map $F_l(\mathbf{x})$ from a layer l in CNNs and consider it as a representation to learn.

neural
style transfer

Neural style transfer Neural style transfer is a highlight of a representation learning as well as of chapter 3 of this thesis. Once researchers found that a deep CNN trained with a large-scale of dataset can express rich representations, an application[87] appeared which became the genesis of the neural style transfer²³. Neural style transfer approach made a huge impact in texture modeling[127]. The neural style transfer model could model very sophisticated styles of artistic drawings beyond simple textures, and the result was good enough to yield many variants (see figure 1.33). The model took two input images and mixed them in a way that one keeps content (objects), and the style (colors, touches, textures, *etc.*) of the other is to be transferred.

content
style

style
representation

What made the style transfer model special was how it manipulated feature maps from a pre-trained deep CNN to represent a *style* (a.k.a. style representation). In particular, they employed Gram matrix to achieve that. Gram matrix being $G^l \in R^{N_l \times N_l}$, the vectorized feature map $F_l \in R^{N_l \times M_l}$ has N_l number of filters and has the size of M_l , where l indicates l -th block²⁴ inside a CNN:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \quad (1.8)$$

The idea of **Gram matrix** is to collapse spatial dimension of the feature map so that it can sort of correlate and represent *overall style* of a given image. Simply optimizing a mean square error of Gram matrices made it possible to apply one style to the other image along with content loss, which plays a role to keep the contents. Diagram in figure 1.32 explains the whole process.

²³ The primary author Leon Gatys had already worked on texture modeling and synthesis[126] and used CNNs to model textures even before the neural style transfer.

²⁴ Beware that it does not necessarily indicate l -th layer.

It is worth mentioning that *style* represented by Gram matrix still remains an ambiguous concept in literatures, because *style* itself is a complicated notion. When it comes to style transfer application, it seems that *style* concerns spatial characteristic, luminance profile, color distribution, brush stroke (pattern or texture), etc [128, 129].

Concerning what optimizing Gram matrix means, one particular study [130] claimed that the mean square loss of Gram matrices is equivalent to finding maximum mean discrepancy with a second order polynomial kernel in Hilbert space. The study showed that transferring style worked well even with a linear kernel or a Gaussian kernel.

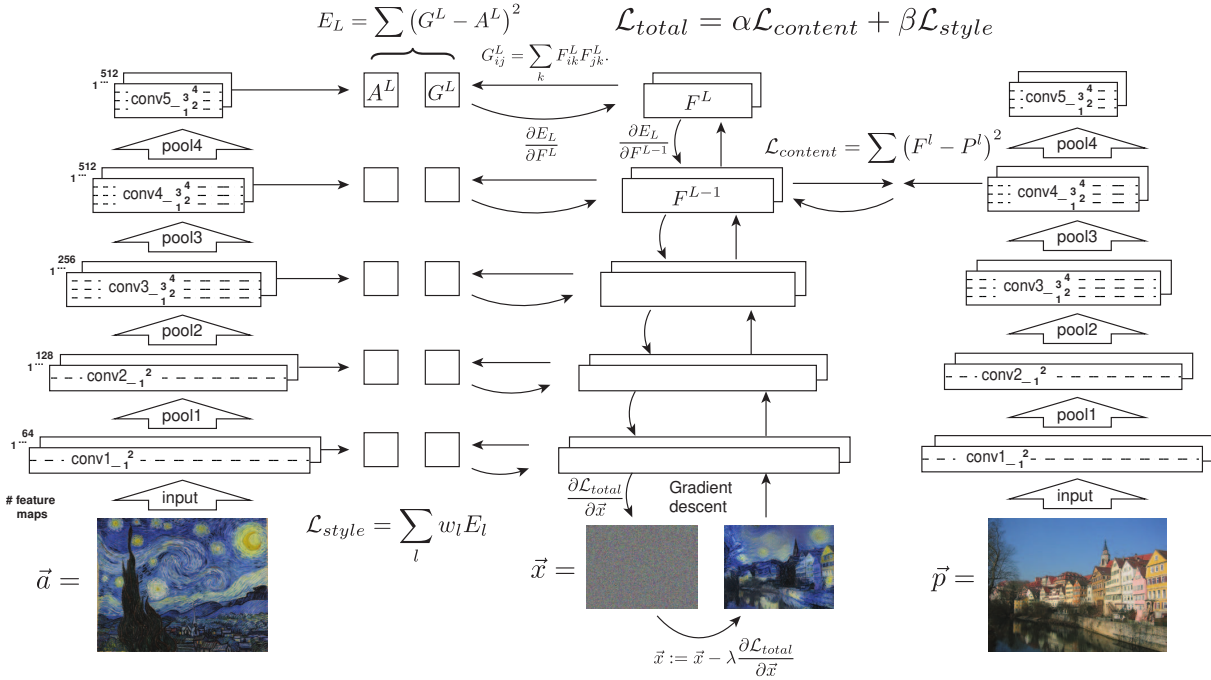


Figure 1.32: Style transfer algorithm. First content and style features are extracted and stored. The style image \vec{a} is passed through the network and its style representation A^l on all layers included are computed and stored (left). The content image \vec{p} is passed through the network and the content representation P^l in one layer is stored (right). Then a random white noise image \vec{x} is passed through the network and its style features G^l and content features F^l are computed. On each layer included in the style representation, the element-wise mean squared difference between G^l and A^l is computed to give the style loss \mathcal{L}_{style} (left). Also, the mean squared difference between F^l and P^l is computed to give the content loss $\mathcal{L}_{content}$ (right). The total loss \mathcal{L}_{total} is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle). This gradient is used to iteratively update the image \vec{x} until it simultaneously matches the style features of the style image \vec{a} and the content features of the content image \vec{p} (middle, bottom). Caption is kept original from the source[87].

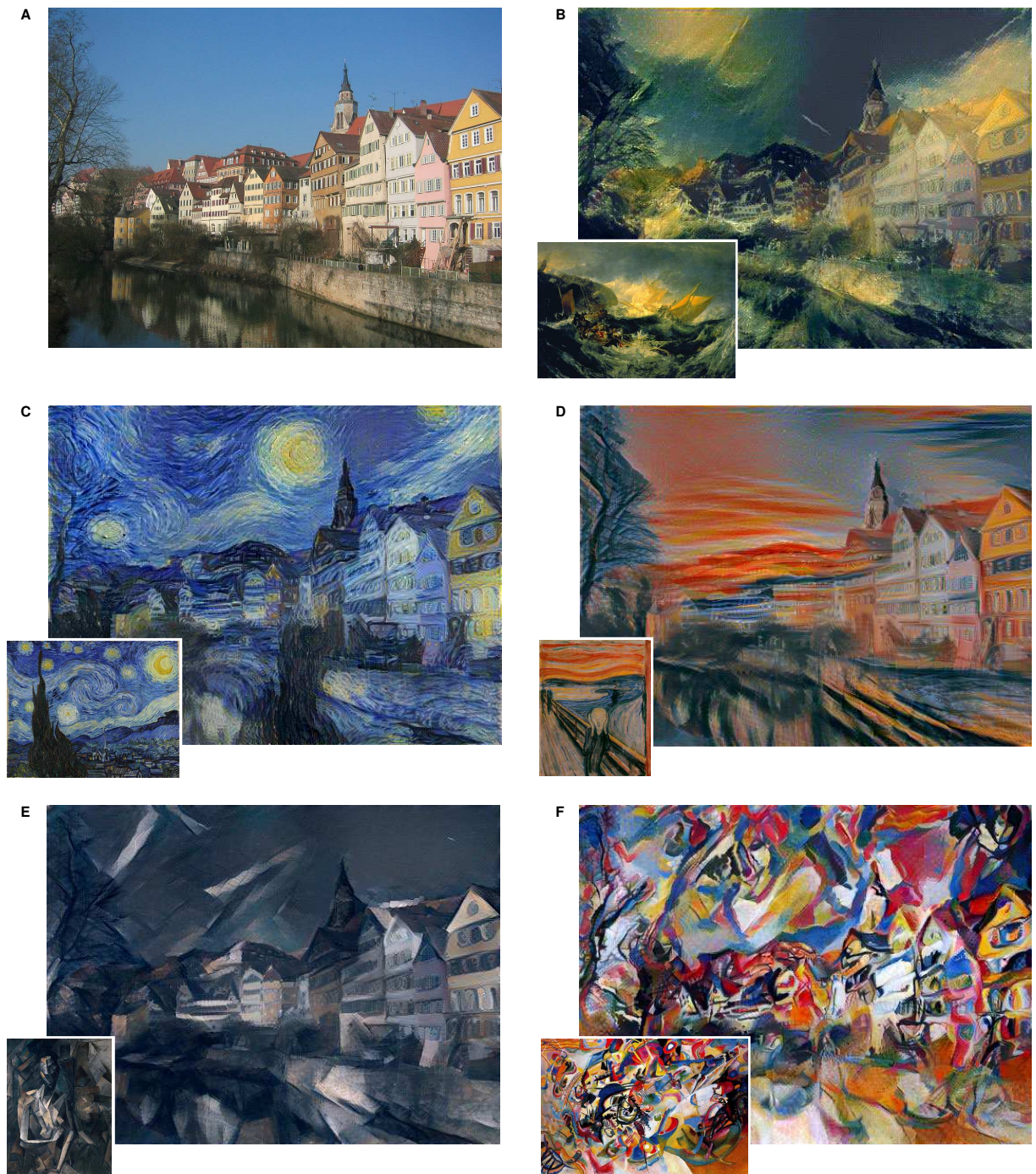


Figure 1.33: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork. The original photograph depicting the Neckarfront in Tübingen, Germany, is shown in **A** (Photo: Andreas Praefcke). The painting that provided the style for the respective generated image is shown in the bottom left corner of each panel. **B** The Shipwreck of the Minotaur by J.M.W. Turner, 1805. **C** The Starry Night by Vincent van Gogh, 1889. **D** Der Schrei by Edvard Munch, 1893. **E** Femme nue assise by Pablo Picasso, 1910. **F** Composition VII by Wassily Kandinsky, 1913. Caption is kept original from the source[87].

4 OBJECTIVES OF THE THESIS

This thesis concerns developing and applying versatile machine learning model for neurodevelopmental imaging. Throughout the introduction, I presented:

- What kind of data is the interest of this work and techniques to acquire it. This thesis deals with bioimages that are large in data size and have resolutions as small as cells nuclei. Microscopy techniques able to acquire such data are introduced.
- What tasks and problems are there to solve in bioimage analysis. Bioimage informatics handle topics such as how to store data, how to transfer data, how to visualize data, and eventually how to prepare or process data to conduct the analysis. Image segmentation task is the essential computer vision task required for qualitative analysis and the main focus of this thesis.
- What benefits deep learning brings in bioimage analysis compared to existing algorithms and methods that are linear or based on strong assumptions. Deep learning is a particular machine learning technique combined with deep neural networks. It demonstrated unprecedented generalizability in computer vision tasks. Many models and datasets were curated and released to segment, restore, or denoise bioimages far better than before. Software tools gained attention again to better distribute these models to users.
- What the current state of deep learning for bioimages is, and what limitations and challenges of current deep learning solutions are facing. They mainly stemmed from supervised learning which is a machine learning scheme that requires a fully-curated dataset. And it is not trivial to curate a bioimage dataset due to its innate properties.
- What general trends are in general computer vision tasks to overcome these limitations and challenges and to further build foundations. General computer vision tasks had the same issue with supervised learning. People started questioning its versatility, biases, and generalizability.
- What other learning schemes there are other than supervised learning. Unsupervised learning is the opposite scheme that does not involve any supervision. Whereas, there are intermediate schemes in-between, such as semi-supervised, weakly supervised, and self-supervised learning. Among them, self-supervised learning is the one currently standing.
- What self-supervised learning is and how it changed the prospect of deep learning, especially in natural language processing. Self-attention, the key module enabling the current framework of self-supervision, essentially builds relationships among data points. It generated a plethora of language models and opened the door to large language models (LLMs).
- What progresses are made in terms of self-supervision in computer vision. Although self-attention could not be easily applied to image data, other methods were devised based on the similar dictionary look-up idea. For examples, contrastive learning and InfoNCE loss turned out to be effective as pretext tasks, they were never meant to perform real tasks.

- What fundamental ideas of self-supervised learning are to develop and seek for a new framework that fits to bioimages. It all comes down to generative modeling and representation learning. There have been already powerful computer vision solutions for generating images, such as GAN, VAE, and neural style transfer.

Eventually, the goal of this thesis is to find versatile image analysis model that could work for a variety of bioimages through self-supervised learning. Community still lacks of large-scale datasets curated for machine learning and, as a result, lacks of versatile and generic models. Therefore, tackling the data issue will be a recurrent topic throughout the rest of the manuscript. In the following chapters, first, I tried supervised learning and assessed its versatility and its limits. Second, I explored self-supervised learning to go beyond those limits. Lastly, I applied these techniques to biological applications via tools integrated in open source software, as I believe that usability and accessibility are what matter at the end of the day.

Contributions of this work are:

- Building a meta-dataset by collecting and combining existing datasets
- Building `bioimagerloader`, a python library to easily manage it
- Using it to assess supervised learning as a path to versatile models in bioimage analysis, using in particular a new kind of ablation study
- Proposing the use of deep filter as a smart drop-in replacement for old linear filters
- Introducing a novel self-supervised loss and its resulted model NU-NET, a blob enhancement filter
- Building software tools to integrate machine learning in image processing pipelines
- Developing applications in collaboration with microscopists and biologists

WHAT TO EXPECT AHEAD

CHAPTER 2: SUPERVISED LEARNING

We have a number of challenges involving data science even before getting into machine learning. I mentioned overall data issues already, but I will cover them in more technical details. In particular, I propose **bioimageloader**, a Python library that I wrote to answer data issues in bioimages. I gathered a large and diverse ensemble of datasets and thanks to **bioimageloader**, I used it to benchmark the versatility of well known methods and push supervised learning to its maximum.

CHAPTER 3: SELF-SUPERVISED LEARNING

Deep neural networks trained with large scale datasets are the prevailing answers for generalized models. Versatility can be achieved by transfer learning and domain adaptation afterwards. The main focus here is **NU-Net**, a self-supervised deep CNN which is a generic filter for cellular-like or nuclear-like objects.

CHAPTER 4: PRACTICALITY

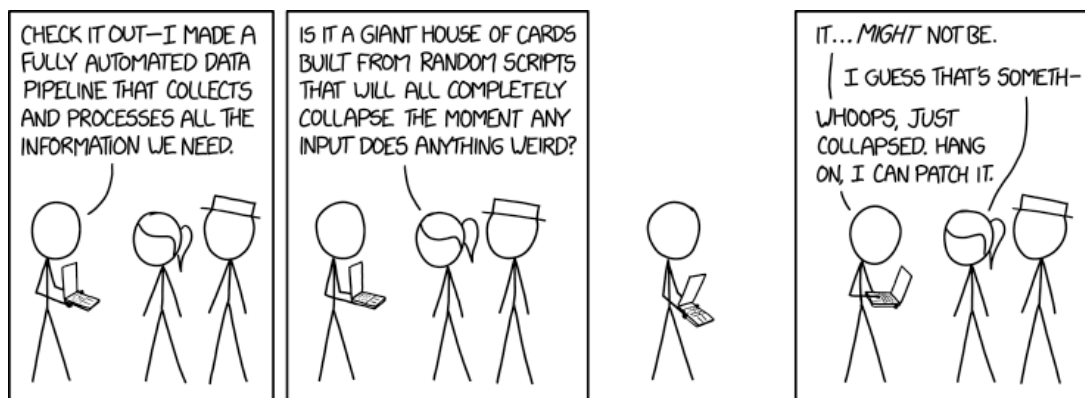
Developing an algorithm is cool but making it useful is even cooler and can be more fulfilling. I will detail the every project I was involved in during my thesis and highlight some additional contributions to the bioimage informatics software ecosystem.

CHAPTER 5: CONCLUSIONS AND BEYOND

I will conclude my works and discuss things beyond them. Some advances in machine learning are not directly related to my work but have high impacts. Vision transformer, diffusion model, and DALL·E that bridged text and vision are definitely shaping new movements. The more capable deep learning gets, the more people are concerned about its bias, interpretability, and privacy vulnerability.

CHAPTER 2

SUPERVISED LEARNING AND VERSATILITY



- Data Pipeline -
<https://xkcd.com/2054/>

Contents

1	How powerful and versatile SOTA supervised models?	58
1.1	Existing solutions	59
1.2	Why they failed and What to do next	60
1.3	Supervised learning in practice	60
2	Bioimagerloader: facilitate machine learning for bioimages . .	64
2.1	Why bioimagerloader?	65
2.2	Overview	65
2.3	Technical details	71
3	Versatile supervised learning	75
3.1	Clustering	75
3.2	Versatile supervised instance segmentation models	80
4	Conclusions	87
5	Discussions and Perspectives	87

This chapter is dedicated to maximizing versatility of supervised machine learning models for segmentation tasks for bioimages. Machine learning solutions based on supervised learning have been proven powerful in a short period of time. However, because they are relatively new, there are still a lot of rough edges when it comes to either developing or to using them. It is often the case that developers do not know where to start research with machine learning models and that users find it difficult to use models, considering that they were given a set of raw programming codes. Therefore, before going into its main topic, this chapter will cover practicality first. To be precise, it will try to answer *how typical supervised learning looks like in practice* and will *attempt to perform instance segmentation task on bioimages*, which is a typical and real case problem. Afterwards, it will pose the problem of versatility in bioimages analysis: *how do existing models perform given a new and unseen dataset by the models? How can we improve versatility on that?* **The work presented here gave rise to one preprint [131], presenting BioImageLoader and included as appendix E, currently submitted to Bioinformatics.**

1 HOW POWERFUL AND VERSATILE SOTA SUPERVISED MODELS?

Let's suppose that we just collected a new set of images to analyze cells within. This happens to be LOB-THG dataset acquired from THG (third-harmonic generation) microscopy which utilizes nonlinear optical process to make imaging contrast in interfaces and optical heterogeneities. See a few samples in figure 2.1. To analyze this cerebellum folia through THG microscopy, we would like to use machine learning methods to segment cells as a first step. The state-of-the-art solutions are namely StarDist[22] and Cellpose[29], and we would like to try them out.

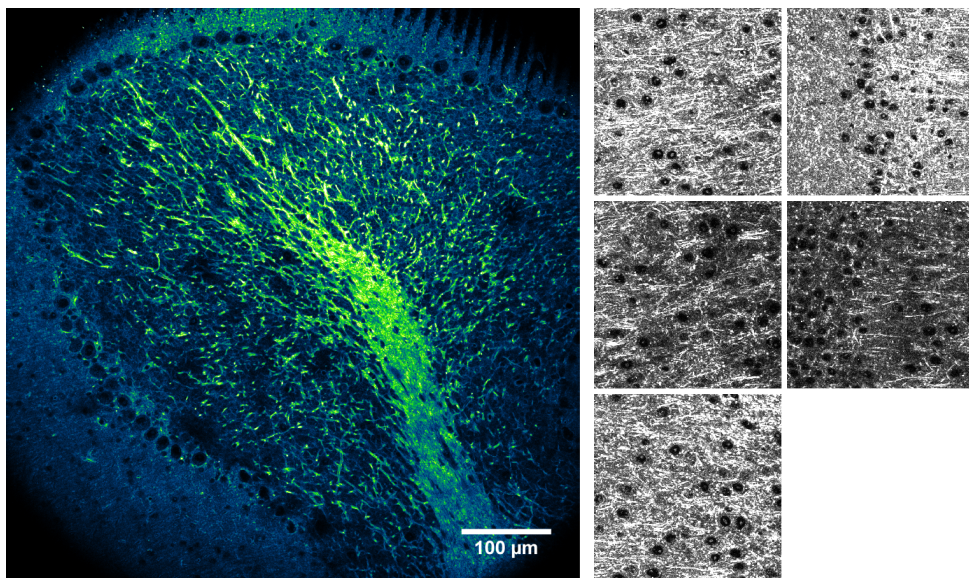


Figure 2.1: (left) Cerebellum folia image acquired from THG (third-harmonic generation) microscopy at LOB by J. Morizet. (right) A few crops from a mouse cerebral cortex image from the same microscope. Black dark spots are cells to segment. These crops will be our bioimages to try out. This dataset will be referred to as LOB-THG.

1.1 EXISTING SOLUTIONS

Accompany results in figure 2.2. Jumping to conclusions, they all failed to segment cells in our THG images. It was because THG images are nothing like typical fluorescence microscopy images that these models were trained on.

StarDist StarDist[22] offers two pre-trained models suitable for our images¹: `paper-dsb2018-diam` and `versatile-fluo-diam`. Both are trained with a subset of DSB2018 dataset[58]. They are supposedly versatile models for segmenting nuclei and cells in fluorescent microscopy images.

Cellpose Cellpose[29] offers more diverse pre-trained models: `nuclei`, `cyto`, `cyto2`, `tissuenet`, `livecell`. For `nuclei`, `cyto`, and `cyto2` models, the group used their own dataset, while they used TissueNet[43] for `tissuenet` model and LIVECell[64] for `livecell` model. It was found that the `cyto` model showed the best results.

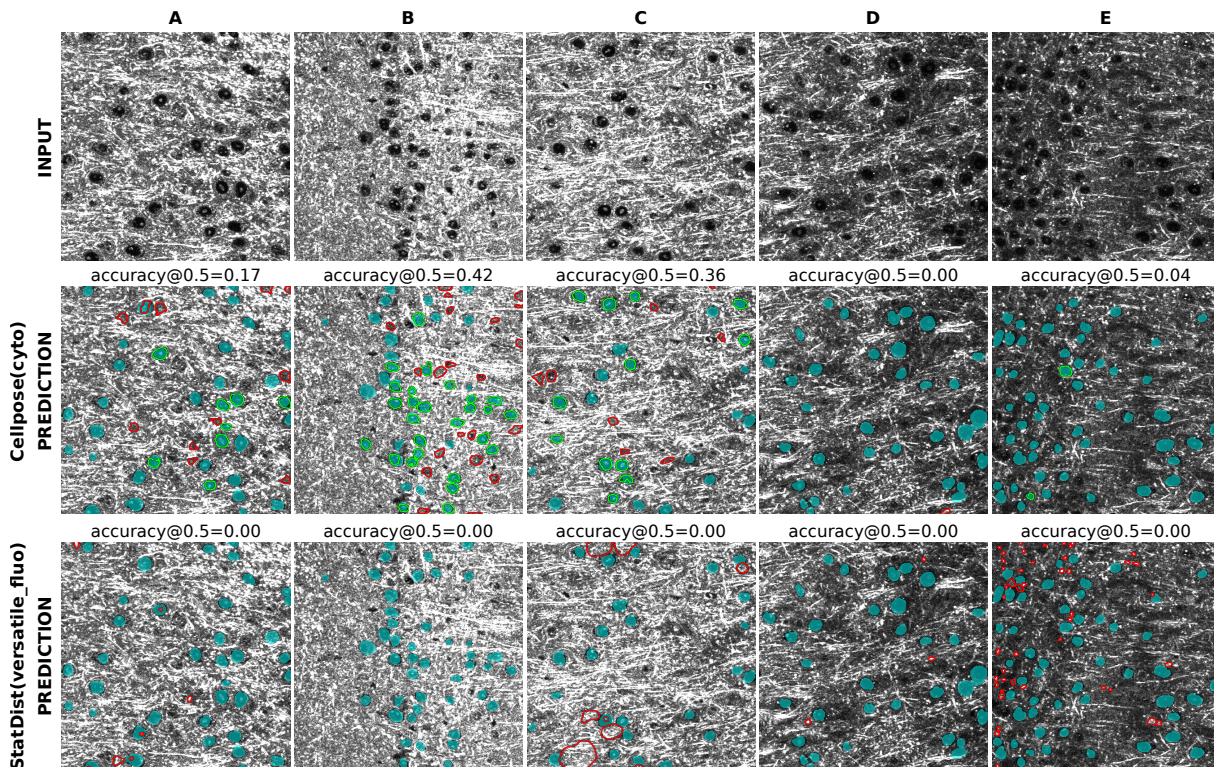


Figure 2.2: Instance segmentation results of pre-trained StarDist (`versatile-fluo`) and Cellpose (`cyto`) models on LOB-THG images. They are current state-of-the-art models for nuclear/cellular segmentation but failed to segment cells in our images. Annotation: cyan blobs = ground truth, red contour = false positive, green contour = true positive. False negatives are masks in cyan without a contour. “Accuracy” metric is what both StarDist and Cellpose used in their papers. Find its definition in appendix C.

¹ There is another model `versatile-he`, specialized for H&E stained cell images from brightfield microscopy. But it is not suitable for our images.

1.2 WHY THEY FAILED AND WHAT TO DO NEXT

There are multiple factors why supposedly versatile and generic pre-trained models failed. Superficially speaking, the models probably have never seen modalities of images like THG. In other words, THG images were out-of-distribution of the tried models. As a matter of fact, there is a trick to make these pre-trained models to work better on the LOB-THG images. We could invert the images, and they would work better because typical cells and nuclei in fluorescence microscopy images look brighter than the background. Consequently, data distribution of LOB-THG will become closer to that of typical cellular bioimages. But *that is not the fundamental solution*. The problem is that these pre-trained models do not know how to deal with THG images even though they are cell images which these models were designed to target.

That being said, one apparent solution is to train a model using your own images. However, then you need to manually label targets for supervised machine learning if you would like to use the state-of-the-art models such as StarDist[22], Cellpose[29], NucleAIzer[40], Mesmer[43], Mask R-CNN[41], etc., because they are all based on supervised learning for the moment. It may not be the worst idea since you might need only a few tenths of pairs. But there are better ways. Provided pre-trained models are already capable by learning to solve similar problems, what we need is domain shift so that they can adapt themselves to a new domain. In other words, we can do domain adaptation by fine-tuning pre-trained models. It would require fewer data than training a new model from scratch. Now *the question is* where to find more generic and versatile models or how to build them, because the more generic they are the easier it gets to achieve domain adaptation or transfer learning.

It is not like we do not know how to build generic models. There are already a lot of generic vision models pre-trained with large scale datasets for transfer learning, and people have been using them for various applications. The fundamental reasons why such models, in biomedical images, are hard to find, or rather do not exist at the moment, are in the nature of biomedical images in general, thus the data itself. Building a versatile and generic models primarily comes down to curating a large scale generic dataset. Challenges of biomedical computer vision were already discussed in section 1.4 and how they are different from “normal” computer vision. In summary, it is challenging to build a dataset that covers inherently diverse bioimages across samples, platforms, and whatnot, let alone to make it large scale. In the next section, the mentioned data issues in bioimages will be addressed and a so-called meta-dataset which is a diverse and large in scale will be introduced by using `bioimageloader`, a programming library. Then, with this meta-dataset, this chapter will demonstrate how to achieve versatility in machine learning models based on supervised learning.

1.3 SUPERVISED LEARNING IN PRACTICE

Since available pre-trained machine learning models failed, the next step is to fine-tune them or to train a new model from scratch. Either ways, it will be supervised learning to be used. Supervised learning is the simplest, yet the most powerful and practical machine learning strategy to this date. Suppose that you would have pairs of data and its labels, train a model, and use it to solve your problem. It all sounds *simple* in theory,

1. How powerful and versatile SOTA supervised models?

but machine learning and deep learning are still relatively young and transforming fast. There are already a number of tools available for developers, and it is the first challenge for a developer to choose which to use. Assume that you, as a machine learning developer, chose one of the most popular tools out there. What would be the next step? How does machine learning research look like? The answer is the **data from the beginning to the end**. Data will determine how you would curate and structure your dataset, what types of models would be suitable, and what metrics to use to evaluate your model.

DATA PIPELINE

Chances are, you already have some data and want to use supervised learning on it to use the results for your analysis. However, in most cases, you would like to collect as much data as you can before you start or even after. After all, machine learning is a data-driven and an iterative process. You can automate the whole data pipeline, but it will not be easy (or almost impossible, see xkcd comics in the beginning of this chapter).

1. Collect You would collect data of your interest. A good strategy is just to grab anything you can find. At this stage, the quantity is more important than the quality. You would like to have a rough idea what data you want and what you do not. Automated process can be helpful. However, at the same time, you do not want to overdo it, because of biases it might bring.

2. Curate You would curate the collected data and create a dataset. A dataset is more than a collection of data. It carries a purpose, and it is especially so in supervised learning, because you need target labels for each data point. You would like to be more cautious about biases in this stage. Labeling data takes a lot of time, energy, and money, thus you want to do it properly. Crowdsourcing is an option, but in essence, labeling is a manual labor. Additionally, it is time to think about organization, structure, and formats of collected data. It is likely that you sourced the data from different places, and that they come in all different forms. You want to standardize and have them in canonical formats for easy handling.

3. Load It may seem obvious, but you do not drag and drop your data into a machine learning model. From this stage on, you need a programming environment to load data and make it ready to be consumed by models, which are usually available through APIs (application programming interfaces). Loading process is fundamentally related to computer resources such as file I/O, RAM (random-access memory), CPU (central processing unit) if your data is compressed and needs to be decompressed, or even network if data is stored in remote servers. They all have limited resources or bandwidths, thus it is important to balance them since we deal with large amount of data.

4. Clean and Normalize You do not want to use data without inspection. You may want to check each data point one by one, but it is not practical nor correct. This step is often referred as data mining. Data mining is a statistical analysis to find patterns in data. What's important is to build a dataset that is representative for the given task



Figure 2.3: Source:

and “clean”. By clean, it means something either not corrupted or not entirely unrelated. In particular, sorting out unrelated data is usually not apparent. Aggressive filtering might lead biases in your models and could further degrade the performance. Normalization helps especially when your data is multidimensional and has high discrepancy across them.

5. Augment Every machine learning algorithm is data-hungry. Data augmentation is a technique not only to increase the amount of data but also to reach to underlying data distribution. Not every augmentation strategy is necessary, but some are indispensable. See figure 2.4.

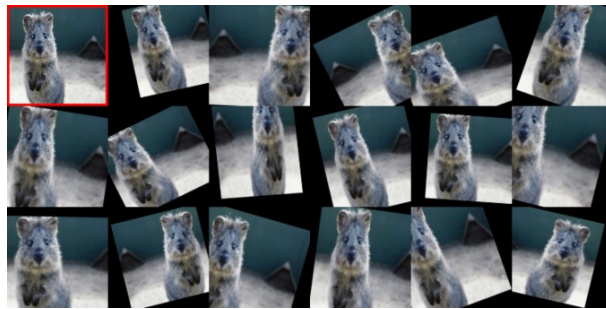


Figure 2.4: Some data augmentations are not an option anymore for the sake of generalization. In vision, transformations such as cropping, affine transformation (scaling, translation, rotation, shearing), or flipping are considered essential basics and important for models to learn an object itself rather than its look when it was imaged. For instance, an image of a cute quokka in the top left corner was augmented not just to have more cute quokkas. By learning how a quokka may look like in different perspectives and scales, the model will be able to find one in the wild more easily with fewer errors. Quokka is a mascot of `imgaug`[132], which used by a popular image augmentation Python library. It inspired `albumentations`[133], a current popular library and the one used to create this figure.

MACHINE LEARNING FRAMEWORKS

Once you went through the data pipeline and built a dataset, next step is to train a model. There are a lot of frameworks for machine learning, which are free and open-source software (FOSS). A framework already implemented a lot of functions ready to be used and expose them through a programming interface so that developers can build an application with ease. Compared to the data pipeline, the machine learning part is relatively easy with help of these frameworks, introduced in appendix 3.3.

EVALUATION

When you have a model after preparing a dataset and using one of above machine learning frameworks, the last step is to evaluate it. Generally, there are two ways to evaluate a machine learning model. **One way** is to monitor loss values during training. The magnitude of training loss cannot be an absolute measure because it depends on many

1. How powerful and versatile SOTA supervised models?

hyperparameters and variables, but validation loss could indicate overfitting. **The other way** is to have proper metrics for the task. A metric is an absolute measure and designed to quantify a specific quality. Yet, you may not want to rely completely on a single metric for evaluation because metrics do have biases[134]. It is always a good idea to monitor losses, coupled with multiple metrics.

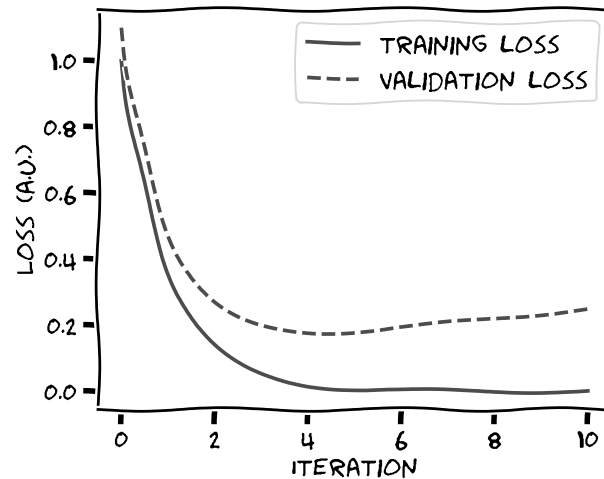
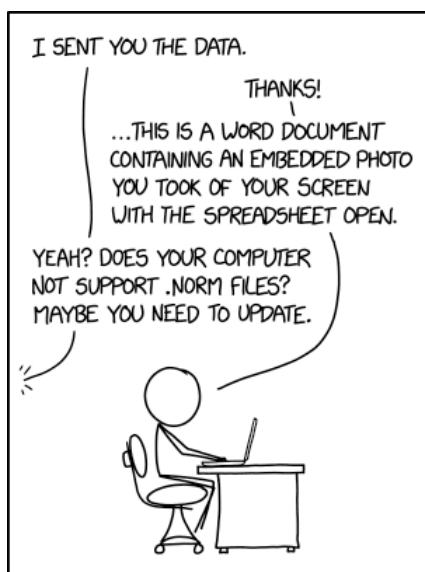


Figure 2.5: Typical loss curves. Validation loss is often slightly higher than training loss. In general, the model is being overfitted once the validation loss started increasing. This point to stop training would be around iteration 4 or 5 in this example.

2 BIOIMAGELOADER: FACILITATE MACHINE LEARNING FOR BIOIMAGES



SINCE EVERYONE SENDS STUFF THIS WAY ANYWAY, WE SHOULD JUST FORMALIZE IT AS A STANDARD.

“Bio-Formats currently supports **162** formats”

- Bio-Formats[70] v6.10.1 by OME team
as of 2022 -

- .NORM Normal File Format -
<https://xkcd.com/2116/>

`bioimageloader` is a Python library to make it easy to load bioimage datasets for machine learning and deep learning development. Bioimages come in numerous and inhomogeneous structures and formats. `bioimageloader` attempts to wrap them in unified application programming interfaces (APIs), so that you can easily concatenate, perform image augmentation, and batch-load them to develop machine learning models.

`bioimageloader` provides

- collections of interfaces of popular and public bioimage datasets for machine learning applications
- image augmentation using `alumentations`, which is popular and powerful image augmentation library (for 2D images)
- compatibility with `pytorch`
- and with others such as `scikit-learn` and `tensorflow`

`bioimageloader` is FOSS and available on The Python Package Index (PyPI) and GitHub.

PyPI : <https://pypi.org/project/bioimageloader/>

GitHub : <https://github.com/LaboratoryOpticsBiosciences/bioimageloader>

I supervised Xingjian Zhang, an M1 student who contributed to `bioimageloader` by implementing new collections and improving documentation. The preprint of `bioimageloader` is available at [131] on arXiv and can be found in appendix E.

2.1 WHY BIOIMAGELoader?

Power of machine learning comes from the data. Making data approachable is the first step and important one to build a great machine learning model. I needed a lot of diverse bioimages to make a versatile and generic vision model for biomedical images. While I managed to find many great datasets, each dataset often represented a single platform or microscope, or a single tissue or experiment. Specificity of each dataset was not desired for my goal. My idea was to combine them to make one big dataset while keeping their identities. Also, they all came with different folder structures and formats. Because of that, I encountered many issues to load and process them, which were sometimes technical or just rooted from the nature of bioimages.

For instances of technical issues, some datasets were missing one or two pairs of image and annotation, had broken files, had very specific file formats that cannot be easily read in python, or provided mask annotation not in image format but in XML format. Some filenames have typos, so sometimes I failed to iterate them.

For an example of intrinsic issues of bioimages, selecting a certain channel was an important functionality that I needed, and it was not easy for bioimage datasets. When a dataset provided separate files for each channel image, it was easy to select one. But in many cases, they just put all channels together in one image file. And even worse for 2 channel images (which are quite common), if they chose to use RGB(A) image formats such as JPEG or PNG other than TIFF, I needed to figure out manually which channel refers to what and which channel is the empty one.

There were other issues not mentioned above of course, but we made sure to deal with all these edge cases one by one. Overall, this work is valuable to package and share it with community so that others avoid this low level data curation and enjoy developing machine learning models with bioimages.

2.2 OVERVIEW

Goals of `bioimagerloader` are following

- easy to load a bioimage dataset via APIs
- easy to combine multiple datasets via unified APIs
- supporting as many datasets as possible
- easy to do machine learning
- facilitating reproducible experiments
- compatible with existing libraries and frameworks
- extensible
- easy to contribute

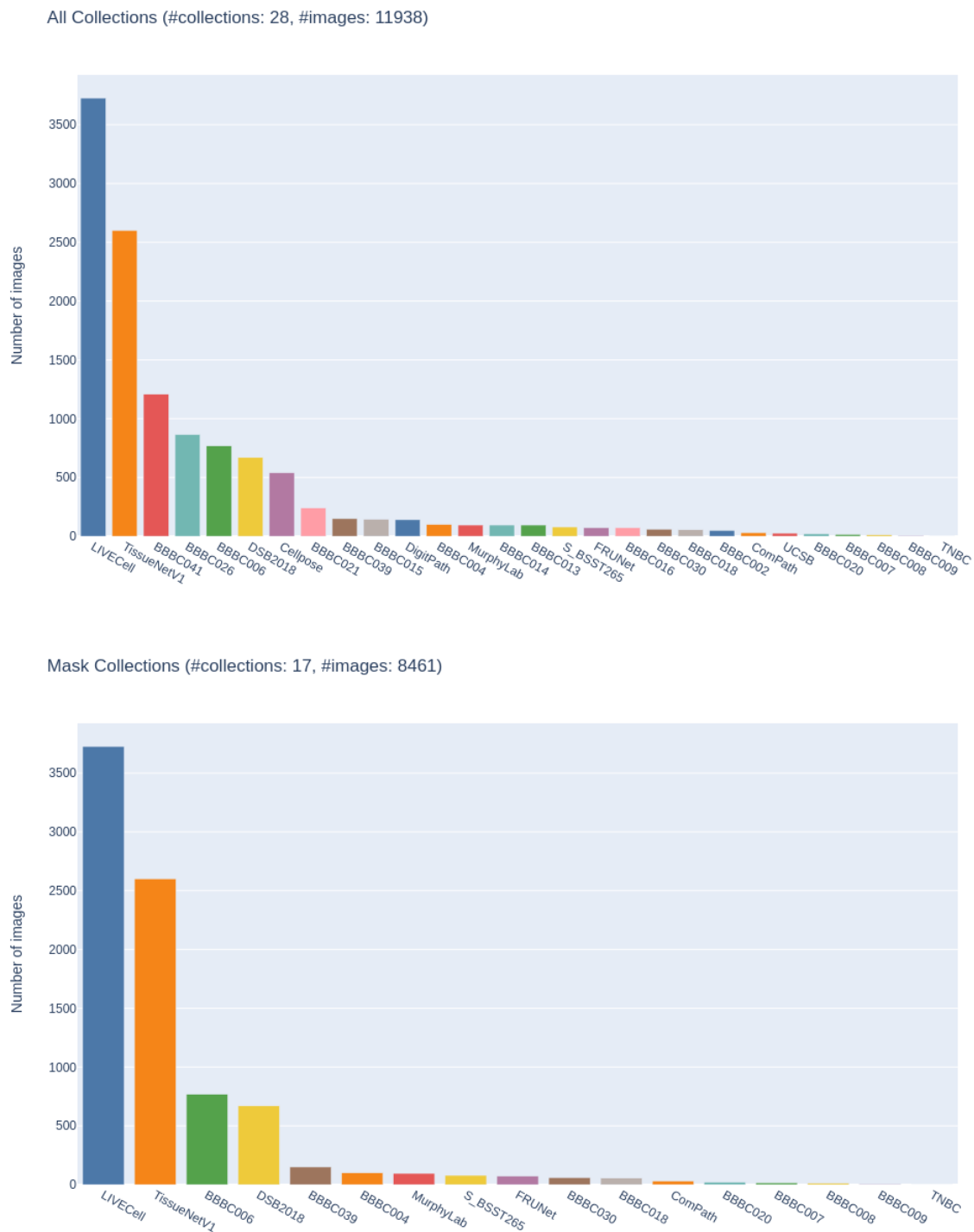


Figure 2.6: Numbers of images of available collections in two categories: total, mask collections. For now, they are all datasets of nuclei, cells or both. Beware of the difference between the number of images and the number of annotations (in comparison to figure 1.11). LIVECell[64] and TissueNetV1[43] certainly stand out in terms of numbers. Find sample images in the next figure 2.7.

2. Bioimagerloader: facilitate machine learning for bioimages

Collections One of the biggest merits `bioimagerloader` is the collections. My answer to a generic dataset is simply to gather a lot of data from different sources. For the moment, I have focused on nuclear and cellular bioimages which preferably have the mask annotation, which includes semantic segmentation masks, instance segmentation masks, and outline masks. Having annotations is helpful not only for supervised learning but in general for evaluation. For the time being, `bioimagerloader` offers 28 collections in total, 17 of which have mask annotations. In addition, the number of images amount to around 12,000 in total. Find numbers of images for each dataset in figure 2.6.

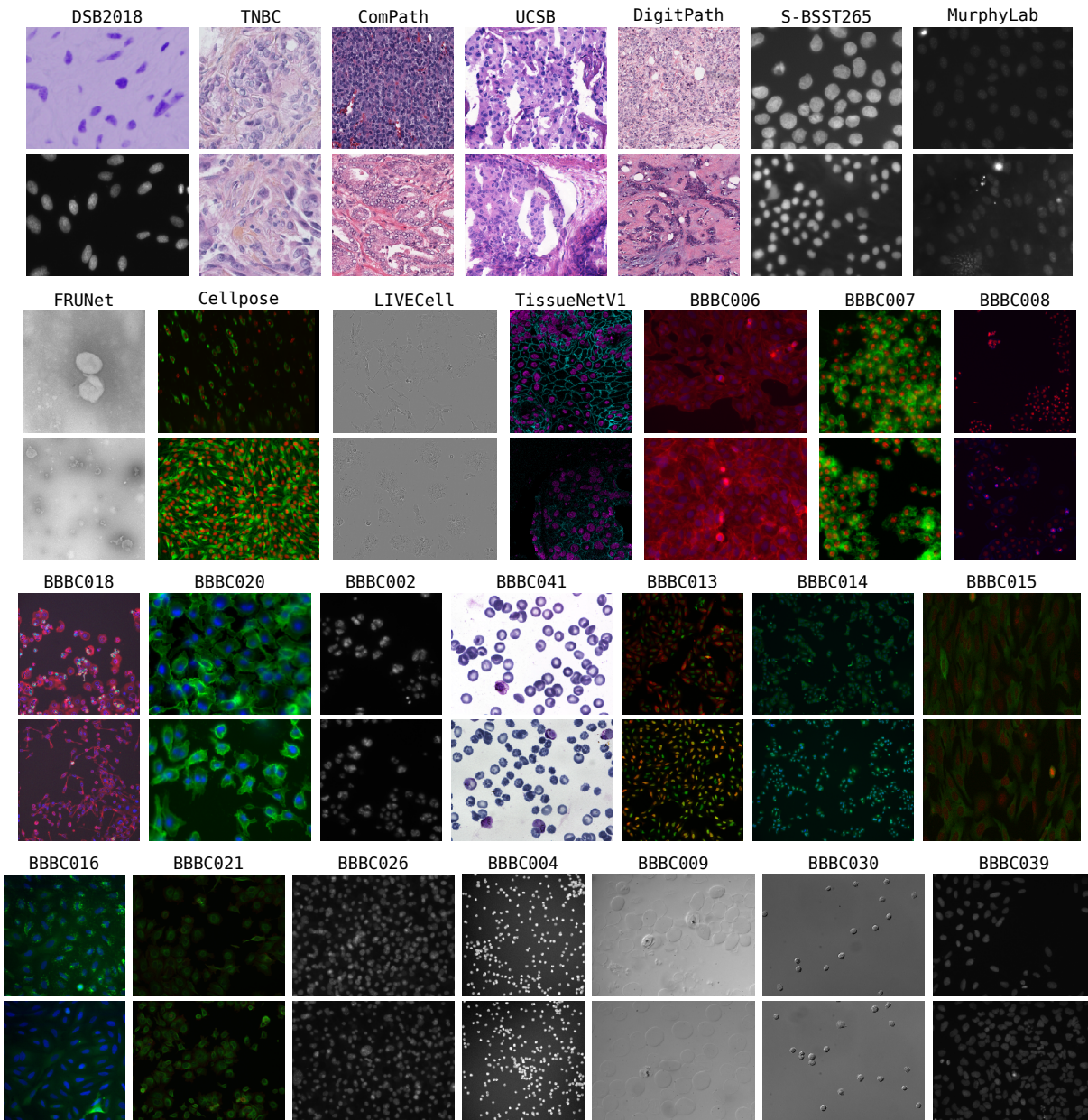


Figure 2.7: Two sample images from 28 datasets that `bioimagerloader` currently supports. Find sources in the table at appendix B.

How to get started Another focus of `bioimagerloader` is the unified API and its object-oriented interfaces². It means that every collection is built on top of a `base object` that holds common properties and unified interfaces. Still, collections can remain individual objects that can hold private properties and functions to keep their originalities. Below code snippet is a “Hello World”³ example of how to load a single dataset, for instance DSB2018[58].

From this moment, I will show some Python codes. I hope that they are intuitive enough to understand even if you do not have prior knowledge of Python nor any programming languages. Sharp (#) and *slated texts* indicate comments.

```
1 # import DSB2018 collection from bioimagerloader package
2 from bioimagerloader.collections import DSB2018
3
4 dataset = DSB2018('path_to_data_directory')
5 for data in dataset:      # iterate data one by one
6     image = data['image'] # get image
7     mask = data['mask']   # get mask label
```

Listing 2.1: Hello World

Loading other datasets has the same interface, even though they come with different internal structures and formats. We can join or concatenate them all as a one variable while keeping the iteration process the same.

```
1 # import DSB2018, TissueNetV1, LIVECell
2 from bioimagerloader.collections import DSB2018, TissueNetV1, LIVECell
3 from bioimagerloader import ConcatDataset
4
5 dsb2018 = DSB2018('path_to_DSB018_directory')
6 tissuenet = TissueNetV1('path_to_TissueNetV1_directory')
7 livecell = LIVECell('path_to_LIVECell_directory')
8 # concatenate and assign them to a single variable
9 cat = ConcatDataset([dsb2018, tissuenet, livecell])
10 for meow in cat:      # iterate data one by one
11     image = meow['image'] # get image
12     mask = meow['mask']   # get mask label
```

Listing 2.2: Multiple datasets

Data augmentation relies on a popular image augmentation library `alumentations` [133]. Instead of reinventing a wheel, `bioimagerloader` officially supports `alumentations` and provides a few custom augmenters specific to bioimages as well.

```
1 # import alumentations and simply name it 'A'
2 import alumentations as A
3 from bioimagerloader.collections import DSB2018
4
5 # define augmentations
6 transforms = A.Compose([
7     A.SmallestMaxSize(max_size=256),      # make sure cropping works
```

² Object-oriented programming is often compared to functional programming. Some programmers argue that one is better than the other in general. In my opinion, though sometimes it is just a preference, they have their own places to be preferred.

³ Every programming language tutorial starts with printing out “Hello World”.

```
8     A.RandomCrop(width=256, height=256),
9     A.HorizontalFlip(p=0.5),           # with 50% chance
10    A.RandomBrightnessContrast(p=0.2), # with 20% chance
11 ])
12 # pass it to 'transforms' argument
13 dataset = DSB2018('path_to_data_directory', transforms=transform)
14 for data in dataset:                 # iterate data one by one
15     image = data['image']           # get image
16     mask = data['mask']            # get mask label
17     # they will all have width=256, height=256
18     # assert is a builtin function that asserts a following condition
19     assert image.shape == (256, 256, 3)
20     assert mask.shape == (256, 256, 3)
```

Listing 2.3: Data augmentation with alumentations

Batch loading is important in terms of the computing resource management as well as the dynamic data augmentation. As the size of data grows, developers need more system memory. Batch loading allows to load just as much data each iteration needs, thus reduces the memory usage. Data augmentation is inherently a random process. Combined with it, batch loading permits every new batch to be dynamically augmented. It could potentially help generalization of models. In addition, `bioimagerloader` combines multiprocessing protocol to parallelize heavy load of the dynamic augmentation. The responsible object is `BatchDataLoader`, and its design was inspired by `DataLoader` from *PyTorch*[135], a popular deep learning framework. In fact, you can use `DataLoader` in place of `BatchDataLoader` without any issues.

```
1 from bioimagerloader import BatchDataLoader
2
3 # we will invite a 'cat' from a previous example
4 call_cat = BatchDataLoader(
5     cat,                               # dataset
6     batch_size=16,                     # 16 data points become a batch
7     drop_last=True,                   # if the last batch doesn't have 16
8     num_workers=4,                    # 4 workers will work in parallel
9 )
10 for meow in call_cat:
11     batch_image = meow['image']
12     batch_mask = meow['mask']
13     # len is a builtin function to get a length of an array
14     assert len(batch_image) == 16
15     assert len(batch_mask) == 16
```

Listing 2.4: Batch loading

Config Configuration file helps experiments. When developing a machine learning model, it is likely that you would change your codes occasionally and have multiple scripts using the same configuration of data. Coding environment is deemed to change and not consistent. Furthermore, each collection in `bioimagerloader` is unique and has several individual arguments, which pollutes coding environment. `bioimagerloader` supports config file (configuration file), so that you can write a static file that can help consistent and reproducible experiments. It follows YAML format, whose focus is human readability.

```
1 # (You can comment in YAML)
2 # This is a bioimagerloader configuration file
```

```
3 # Check arguments for each dataset in the documentation
4 # [DocsLink](https://laboratoryopticsbiosciences.github.io/
   bioimagerloader-docs/)
5 # file      : config.yaml
6 # author   : Seongbin
7 # date     : 2022-xx-xx
8 # exp      : test bioimagerloader
9 # used by  : model_test1, model_test2, model_test3
10 TissueNetV1:
11   root_dir : ./data/tissuenet_1.0/
12   image_ch : [nuclei] # TissueNetV1 has two channels
13   anno_ch  : [nuclei] # can load either nuclei, cells, or both
14 LIVECell:
15   root_dir : ./data/LIVECell/
16   mask_tif : true      # LIVECell's annotation needs conversion
17 BBBC020:
18   root_dir : ./data/BBBC020_v1/
19   image_ch : [cells, nuclei]
20   drop_missing_pairs : true # some images do not have annotations
```

Listing 2.5: Config file example (config.yaml)

```
1 from bioimagerloader import Config, ConcatDataset
2
3 # load 'config.yaml' file above
4 config = Config('config.yaml')
5 # load defined datasets
6 datasets = config.load_datasets()
7 print(datasets) # check by printing
8 # print output : [TissueNetV1, LIVECell, BBBC020]
9 # concatenate and assign them to a single variable
10 cat = ConcatDataset(datasets)
```

Listing 2.6: Load datasets using config.yaml

Compatibility bioimagerloader simply helps to load bioimage datasets for machine learning applications. Though it is designed to be most compatible with *PyTorch*, its APIs are generic to be used with other frameworks, such as *TensorFlow* and *Scikit Learn*. To demonstrate, I will use them all three frameworks with bioimagerloader later in the next section 3.

Extensibility What if you have a local dataset and want to benefit all the functionalities that other collections are capable of, or further want to combine it with them? You need to follow the basic structure of bioimagerloader, which may sound a little difficult and cumbersome. However, it is relatively easy in bioimagerloader thanks to the object-oriented programming (OOP) interface. The hierarchical architecture allows users to implement only the particular parts, and the rest is inherited automatically. It is also helpful when a user wants to extend or modify existing collections for different behaviors. The template for implementing new datasets and the guide to modify existing collection are included as examples in the documentation.

Aside from the basic features introduced so far, bioimagerloader provides a lot of utilities, such as custom image augmentations specialized to bioimages, training/validation/testing subset splitting function, filtering data with their indices, etc.

2.3 TECHNICAL DETAILS

This section covers the design choices of `bioimagerloader` and what it actually does. The library was designed in a way that it could be easily managed, extended, and get contribution. Following contents are rather technical. Please feel free to skip it and jump to the next section 3.

Interface and Typing Programming languages evolve and Python has been evolving faster than ever since it attracted a large crowd of users from all different domains. `bioimagerloader` makes use of advanced new features and requires a minimum version 3.8⁴.

Abstract base classes (ABCs) was introduced since Python 3.4, thus particularly not a new feature. It allows writing interfaces, a concept brought from object-oriented languages. Interface is not an actual class, therefore abstract. Rather, it defines how a class and its subclasses should look like. It might seem redundant, but it helps to organize and to structure classes at the programming level, so that it becomes easy for contributors. Once you missed something or attempted to do something illegal to the interface, you will get errors both from a static type checker⁵ or the Python interpreter.

Object-oriented programming (OOP) builds hierarchical relationships of objects. Accompany figure 2.8. Based off of `DatasetInterface`, `Dataset` defines a concrete class which can hold properties and methods. All existing classes in `bioimagerloader` are subclasses of `Dataset` class. In this way, they could all share common properties and methods without reimplementing, and more importantly get combined with other classes. In addition, OOP with ABCs allows users to extend these base classes and make their own objects by subclassing them. It ensures compatibility with existing objects and all the other functionality of `bioimagerloader`.

Typing is a tool for developers to find bugs based on types of variables while they code. Type annotation was thought to be only useful in compiled languages which were choices of large projects. However, as software got more complex, developers started building large and complex software using scripting languages as well⁶. Python started embracing the type system around version 3.5. `bioimagerloader` tries to use types as much as possible, so that developers can avoid errors much easily and quickly. Typing has another benefit in autocompletion. It helps both developers and users by indicating types of expected variables.

Caching could give a huge performance gain. `bioimagerloader` uses cache in particular to search for image files and their labels. As some datasets can have over several thousands

⁴ For the time being, the stable versions of Python are 3.9 and 3.10 in practice. Python 3.11.0 (the first stable version of 3.11) was released 24 October 2022.

⁵ Static type checker is a program to iterate through all the type annotations and report if there are errors related to types. It is static, because you do not have to run and break the actual software.

⁶ One apparent example is TypeScript. JavaScript is the language of web browsers and the most popular language because of internet. JavaScript is also a scripting language like Python. Yet, developers built so many things and complicated software using it that they need other programming language features from compiled languages. One of them was the type annotation and *Microsoft* created TypeScript, which is a superset of JavaScript with many advanced language features.

of files, searching process can become a bottleneck. Since version 3.8, Python offers a native way to cache class properties.

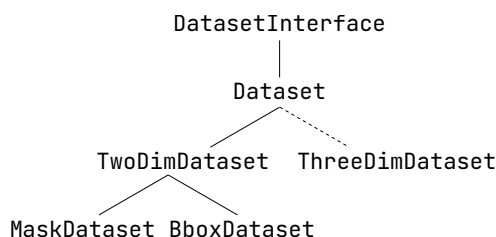


Figure 2.8: Class structure of `bioimagerloader`. Hierarchical architecture helps organize different types of data. It is `DatasetInterface` that defines the very base interface of all the other classes. Dotted line indicates a work in progress.

Cleaning Curating a dataset is a combination of machine automation and human labor. Both entities can make mistakes. Sometimes, annotators forgot some files, which result in missing label pairs. They can make typos. They may have forgotten to save labels and left annotations empty. When transferring or compressing data, files can be broken (corrupted files). These are all I experienced while building `bioimagerloader`. Once a dataset were distributed, these errors get copied too. They look small, but it takes time and energy to correct them all. `bioimagerloader` corrected them as many as possible and documented each of them in each collection.

Standardization Making a standard interface allows combining and processing different datasets. Fragmented formats and structures of biomedical datasets are rooted from the fragmented tools and lack of consensus, already mentioned in section ??.

Image file formats are fine as long as they are well adopted and available in any tools. Dataset curators usually make right choices and provide TIFF, PNG, or BMP extension files which are universal these days. These images formats are easy to read and manipulate, but they may depend on different programming libraries. One thing to note about PNG format is that it actually has two internal types: one with RGB and the other with RGBA channels. “A” refers to the alpha (transparency) channel and is used for composition with other images. It is useless in bioimages and convey no information, thus dropped. In `bioimagerloader`, all these different image formats are loaded according to their extensions and all stored as NumPy[136] array in memory ready to be consumed.

Data types could be more important than you anticipated. Depending on data types, some data may not be recognized properly as intended. Typical images have UINT8 and FLOAT32 data type⁷, where values range from 0.0 to 1.0 in FLOAT32. However, biomedical images have no standard data types, and they come with all different types: UINT8, INT16, UIN16, INT32, UIN32, and rarely FLOAT32. Difference of data types causes a fundamental problem that computer cannot do arithmetic nor stack data with different data types. Another issue with the data type is compatibility with other image-related libraries. Data augmentation library `albumentations`, which `bioimagerloader`

⁷ Unsigned 8-bit integer (UINT8) ranges from 0 to 255. Single-precision floating-point (FLOAT32) encodes numbers in the exponent format and can express decimal digits. I can encode both positive and negative numbers as small as 2^{-126} and as large as 2^{127} .

uses, supports a certain set of data types, namely `UINT8` and `FLOAT32`. Moreover, some augmenters work only with specific data types. `bioimagerloader`'s choice is `FLOAT32` since it can embrace lower-bit data types and keep high enough precision for computation. Additionally, `FLOAT32` is the default data type in GPUs, and thus so in all the deep learning frameworks.

Channels are a big issue in bioimages. Often one channel correspond to a fluorescent response. The issue is that there is no limitation of the number of channels nor there is a standard format such as RGB channel. There are two cases when it comes to multiple channels in bioimages. Either they come in a single image file with multiple channels or in multiple image files each of which refers to a channel. In either cases, `bioimagerloader` gives users an option to select a single channel or multiple channels. When returned they all have three channels because `albumentations` does not recognize two channels or more than three channels. So far, there was no case that a dataset comes with more than three channels, but it is possible. When it happens, it is to be discussed.

Parsing annotation is a chore. In terms of mask datasets (`MaskDataset` in figure 2.8), which consists a majority of the current collections, annotation could be an image format, a set of polygon coordinates, or other formats such as ImageJ's RoI (region of interest) format. For examples, some come with an encoded format such as RLE (run-length encoding) or with a specific format, for instance, one following specification of MS COCO[137]. Image format is the easiest to handle and the default format `bioimagerloader` has chosen for the `MaskDataset`, because it can be directly consumed for segmentation tasks. Parsing different formats to image formats is sometimes computationally heavy. For that reason, `bioimagerloader` recommends and warns users to pre-parse and store them as image formats when it is the case.

Normalization is optional only when it is necessary, but it happens quite often in bioimages. Not to lose precision when converting data types, the common strategy is to use the maximum value for a given data type, e.g. `UINT8` ($255; 2^8 - 1$), `UINT16` ($65536; 2^{16} - 1$). However, as an example, some image sensors have 12-bit resolution, and they store images in 16-bit or 32-bit simply because 12-bit data type is not native to most storage devices and processing units, nor common. As a result, these images ended up having empty bits above 12-bit. In some cases, it is fine to drop the empty bits and normalize images with 12-bit ($4095; 2^{12} - 1$). However, it is not always the best strategy, because the capability of image sensors does not necessarily correspond to the right contrast. In many cases, bioimages look very dim right after acquisition, far below sensors' maximum values. Images become visible once the contrast is adjusted or normalized. For this reason, `bioimagerloader` gives the normalization option to certain collections.

Optional arguments are what allow individual behaviors in each collection. For instance, some datasets have subsets such as a training set and a testing set, or different types of annotations, etc. `bioimagerloader` respects individual specifications of each collection as much as possible and gives optional arguments to opt them in or out.

Documentation `bioimagerloader` provides APIs to both users and developers, therefore it is imperative to have a proper documentation. Documentation contains extensive guides

from basics to application examples, and its source is managed in a separate Git repository⁸. The whole code base is documented in NumPy[136] style docstring⁹. Docstring behave like a markup language and can get automatically rendered in HTML format, so that users can read it on their browser easily.

⁸ <https://github.com/LaboratoryOpticsBiosciences/bioimageloader-docs>

⁹ Docstring is a documentation directly attached to objects. When it exists, users can easily access to documentation in a coding environment. In Python, people usually follow either Google style or NumPy style.

3 VERSATILE SUPERVISED LEARNING

In the previous section 1, as presented, pre-trained StarDist[22] and Cellpose[29] failed to segment cells in THG images. It led options either to *train* a model specialized to the new data from scratch or to *fine-tune* them. Either way may result in good enough performance, but they are hard to do properly. A model specialized to a dataset can become useful when there is large enough data available. As we discussed, curating a bioimage dataset is not an easy job. Fine-tuning pre-trained models is relatively easy, but we will see that the pre-trained models are not generic in the first place, as we will see later through a series of analyses.

As you may have noticed, datasets will be written in upper camel case in a monospaced font, sometimes with a hyphen, such as DBS2018, Cellpose, ComPath, BBBC006, LOB-THG, and so on. Find the details in appendix B.

I supervised Tanguy Rolland, an ENSTA 2A student who contributed to analyzing collections of `bioimageloader`.

Instead of aforementioned solutions that might end up suboptimal, this chapter will explore an alternative solution with the help of `bioimageloader`. We will compose a generic meta-dataset and further build generic supervised models for instance segmentation, in particular using StarDist and Cellpose models.

3.1 CLUSTERING

We have in our hands 28 nuclear/cellular datasets supported by `bioimageloader`, plus 4 internal datasets from LOB, which will be called LOB-THG, LOB-MNTB, LOB-TCYT5, LOB-P14. Their samples are shown in figure 2.9. Except LOB-THG, three datasets are acquired from a confocal microscope and ChromS[2] microscope. Clustering was done through 4 steps in large: pre-processing, computing features, reducing feature dimension for visualization, and finally clustering.

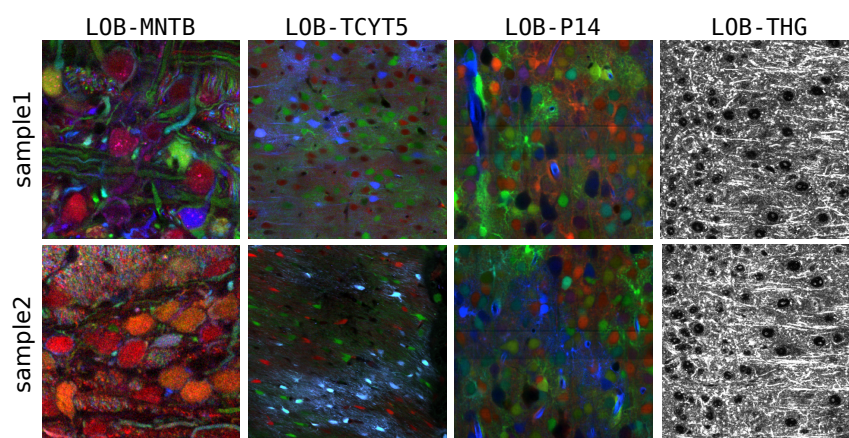


Figure 2.9: Four local datasets at LOB, which will be used in this chapter. Two images were randomly sampled from each dataset. Note that the contrast was adjusted for better visualization. Find their details in appendix B.

Pre-processing Before extracting features, it is important to normalize different datasets. The biggest divergence among them is channel. Some datasets have multiple channels while others have grayscale. It makes no sense to augment grayscaled images to have multichannels. Instead, we can convert all multichannel images to **grayscale** with consideration of the number of channels and how they are constructed. Since they are not RGB images in general, except for some, the typical recipes¹⁰ for grayscale conversion was not used. The most reasonable way is to average multichannels equally. In addition, the size of datasets is all different. To balance it, we could sample 50 images, for example, from each one, by resizing or by cropping images for those whose size was less than 50.

Deep style features There are a lot of image features you can get using traditional methods such as histogram of oriented gradients (HOG) [138] or scale-invariant features (SIFT) [139], etc. However, recently, it has become increasingly popular to use features from pre-trained deep CNNs [128, 40, 18, 140, 141, 29, 142, 143] in the form of representation learning. I used a pre-trained VGG16[85] and computed features from the 10th weight layer (out of 16) and computed Gram matrices, assuming that they represent general “styles”[126, 29, 88, 142, 143, 130], as it was explained in section 3.2 from introduction. Deep style features are suited for clustering datasets since we would like to know their overall relationship.

UMAP visualization Visualizing high-dimensional data is challenging. Dimensionality reduction is a technique to crunch dimensions to lower ones while preserving relevant structures. It is especially useful for visualization once the dimension is reduced to 1-3 dimension with which we are familiar. Our deep style features have dimension of 262,144 (512^2 ; from 10-th layer of VGG16), which is a large number. UMAP (uniform manifold approximation and projection for dimension reduction) [144] is a dimensionality reduction algorithm, designed to work with high dimensional data, like popular t-SNE (stochastic neighbor embedding; t stands for Student-t distribution)[145]. The algorithm cares less about interpretability of reduced axes and focuses on finding relationship among data points. I used UMAP to visualize data in two-dimension as shown in figure 2.10.

KMeans clustering As for clustering the deep features, I chose KMeans, which is an unsupervised clustering algorithm. Given the number of clusters K as a parameter, the algorithm attempts to find centroids of clusters and to minimize distances from data points to these centroids, while keeping distance among the centroids. I set 10 clusters ($K = 10$) after inspecting inertia values sweeping K from 1 to 15. The result is visualized on top of UMAP at (b) in figure 2.10.

Interpretation Overall, UMAP tells that there is *no single dataset that covers the whole space reduced by UMAP*. It matches our expectation that bioimages are highly diverse and existing bioimage datasets are fragmented and unique on their own in most cases. This is the primary reason why both pre-trained StarDist and Cellpose failed on LOB-THG.

¹⁰ Grayscale conversion linearly considers human perceptions for RGB and light source devices (monitors or TVs). The most common one is $Y = 0.299R + 0.587G + 0.114B$. Another variant is $Y = 0.2125R + 0.7154G + 0.0721B$.

StarDist was trained on a subset of DSB2018 and Cellpose was trained on a superset¹¹ of Cellpose, and both have little to do with LOB-THG in terms of visual features. **Second** thing we can notice is an outlier, specifically BBBC004. As a matter fact, BBBC004 is a synthetic cell dataset[146, 147] and became an outlier apart from all the other datasets. **Third**, we can observe large overlaps of datasets, and that they are forming groups in UMAP (a) figure 2.10. These groups are in good accordance with results of KMeans in (b) 2.10.

Based on the result of KMeans, correlation coefficients can be derived based on the numbers of images in each cluster ($K = 10$), and they are shown in figure 2.11. Two big groups from the correlation figure correspond to **groupA** and **groupB** from (b) in figure 2.10. **GroupA** and **groupC** make the third-largest group, which contains all ChroMS datasets (LOB-MNTB, LOB-TCYT5, LOB-P14). **GroupC** represents relatively a few datasets. **GroupD** comprises TNBC and LOB-THG and has little to do with the others. Lastly, again, BBBC004 is a big outlier as a synthesized dataset.

¹¹ Cellpose team supposedly received a bit of anonymous contribution and published new lines of pre-trained models. But they have not updated the public Cellpose dataset as well as nuclei and cyto models. Dataset Cellpose used in this manuscript is the one that they released when they published the preprint and that I could download at the time of writing.

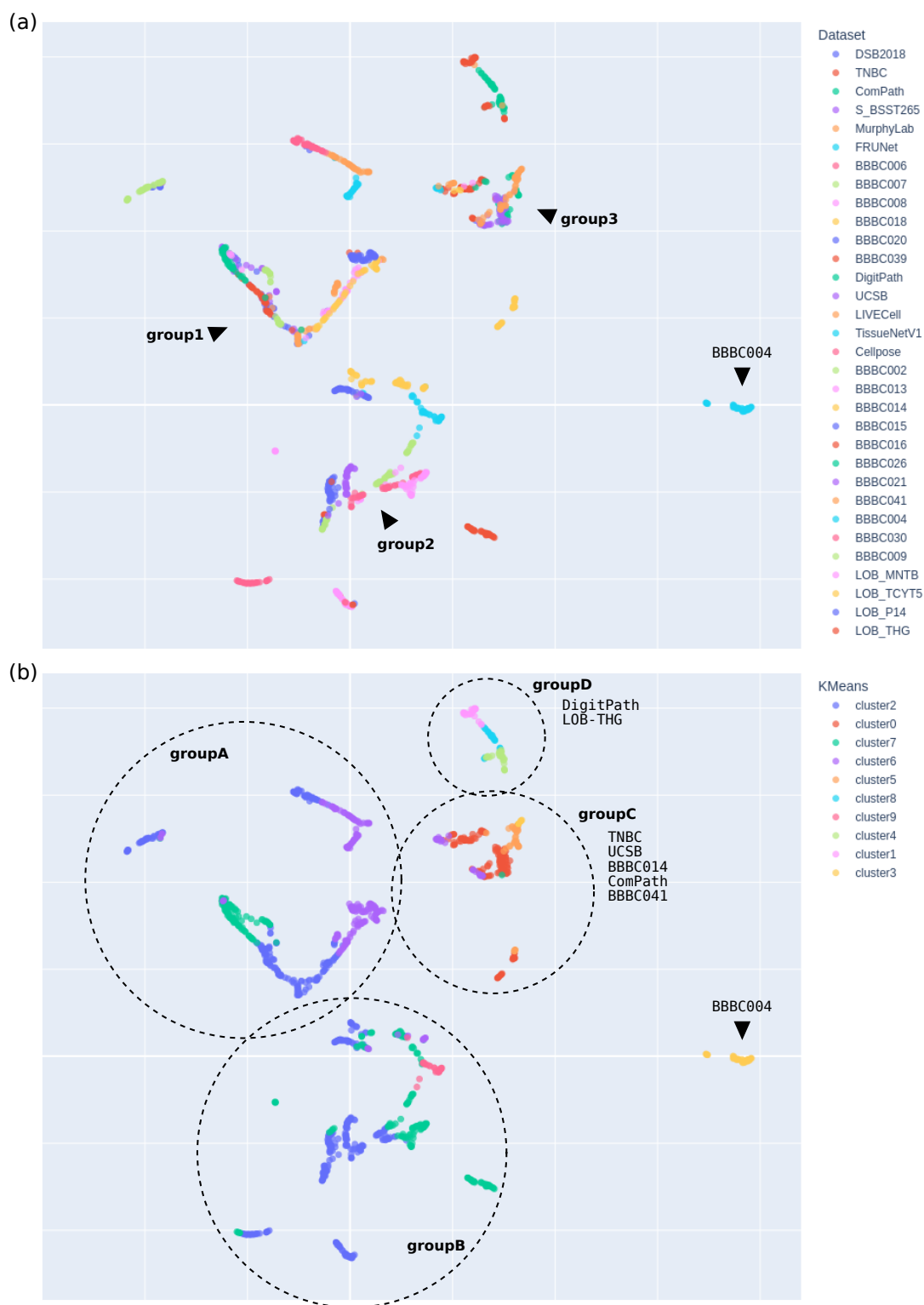


Figure 2.10: Results of (a) UMAP dimensionality reduction and (b) KMeans clustering overlaid on UMAP. Best viewed in color. 50 images were sampled from all 32 dataset, which makes 1600 data point in total. As for features, “deep style features” were used from pre-trained VGG16[85]. BBBC004 clearly does not go along with all the other datasets, because it was synthesized. Added annotations are in **black**: (a) groups (1, 2, 3) and BBBC004, (b) dotted circles, groups (A, B, C, D), and BBBC004. Note that the grouping did not have a rigorous rationale, though indeed both UMAP and KMeans results were considered.

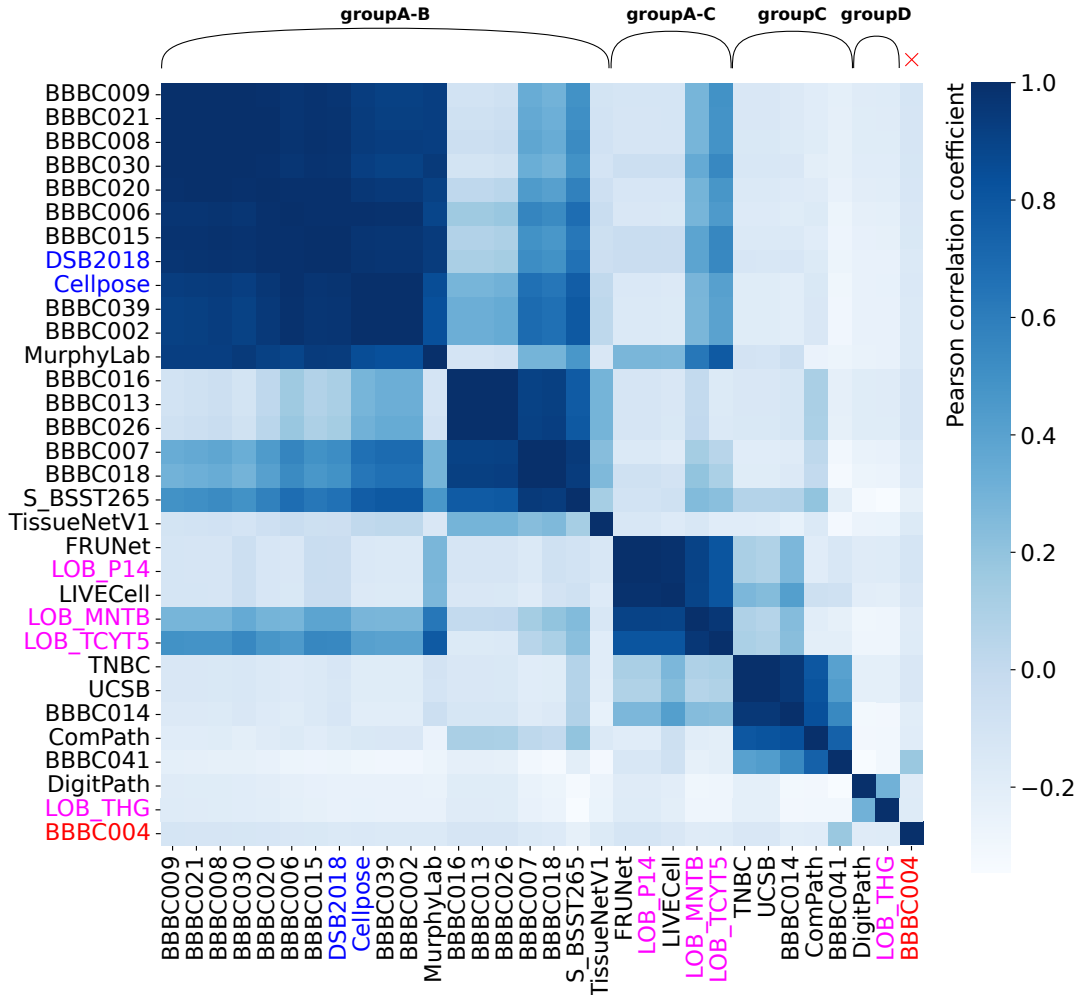


Figure 2.11: Correlation of datasets based on KMeans clustering ($K = 10$) using deep style features from VGG16. Annotation of groups at the top of figure roughly corresponds to that at (b) in figure 2.10. It is clear that BBBC004 in red is an outlier. GroupC includes all the ChromS datasets in magenta. DSB2018 and Cellpose in blue were the ones used to train StarDist and Cellpose, respectively (find details in the main text). The table was first sorted by the size of clusters, then sorted again within each cluster by the portion of datasets.

3.2 VERSATILE SUPERVISED INSTANCE SEGMENTATION MODELS

To reiterate, the goal of this section is to build a versatile model. To do so, we are going to curate a large meta dataset using `bioimagerloader`. Though `bioimagerloader` supports 28 collections in total, not all of them have annotation nor instance mask labels. So, 10 collections that have instance mask annotations were piceked since retraining StarDist[22] and Cellpose[29], designed for the instance segmentation task, requires them. In addition, we could add 4 private datasets from LOB, which were introduced in figure 2.9. Find the list of datasets to be used in table 2.1. Before training and building versatile StarDist and Cellpose, the strategies to combine all 14 unique datasets need to be discussed.

Dataset	# of images	# of training images	# of samples after augmentation	# of samples balanced
DSB2018	735	670	670	300
ComPath	30	25	334	300
S-BSST265	79	72	216	300
FRUNet	72	65	1,007	300
BBBC006	768	692	1,492	300
BBBC020	20	15	215	300
BBBC039	200	150	446	300
Cellpose	100	89	267	300
LIVECell	5,239	1,512	7,884	300
TissueNetV1	6,990	2,601	14,332	300
LOB-MNTB	41	36	144	300
LOB-TCYT5	14	9	226	300
LOB-P14	40	35	367	300
LOB-THG	14	9	146	300
Total				4,200

Table 2.1: List of datasets used to train new StarDist and Cellpose models and their numbers of images and of samples. For the training subset, 15% of or at minimum 5 images were taken from each dataset. Data augmentation was applied to match the target sizes, which were 17 and 20. At last, the same number was sampled for all the datasets, and the total number reached 4,200 samples.

Composing a meta dataset

1. Training/Testing split : The collections and datasets listed are all well curated for machine learning applications. Some collections come with training and testing subsets, or even validation subsets, though the validation subsets were not common¹². We could simply take pre-defined training subsets if they existed. For the rest, we will randomly split each into training and testing subsets by taking 10%. However, 10% for some datasets was already a significant number. To avoid having too small number of images for testing subsets, we may set the minimum number of testing subset to 5. For the validation subset, we could sample 15% from the training subset. It is worth mentioning LIVECell and TissueNetV1 because they both pre-defined huge portions for validation and testing subsets. But the numbers of training subsets are already much bigger than the others, so we could ignore them for the moment.

¹² The reason is reproducibility.

2. Balancing the number of samples : For a reminder, the goal is to build generic models, followed by a generic dataset. As we saw results of UMAP and KMeans, bioimage datasets are highly specific and fragmented. Also, we saw that the numbers of images each dataset provides are all different and have sometimes huge gaps. If we simply combine them as they are, the resulted dataset will have huge biases towards those with large numbers. Hence, we could, for example, sample the same number for all datasets, even if some, such as LIVECell and TissueNetV1, come with the huge numbers of images.

3. Data augmentation considering the target diameter : However, we could not sample more images than what a dataset offers because it is duplication and the models will become more prone to overfitting. Instead, consider a data augmentation strategy with the average size of targets in mind. Intrinsically, object detection or segmentation CNNs, are sensitive to the size of objects since they have a fixed set of receptive field sizes from convolutional layers. Both StarDist and Cellpose give users an option to suggest an expected size of nuclei or cells. Accordingly, they both have expected input sizes too. StarDist expects inputs whose resolution is 256×256 and objects to have diameter of around 20 pixels on average. Cellpose is similar but expects a resolution of 224×224 and the average diameter of 17 pixels (30 for cells, which are larger than nuclei). As you might have guessed, all datasets have different diameters as well as resolutions. There are all combinations: a small diameter with a high resolution, a small diameter with a big resolution, a large diameter with a high resolution, and a large diameter with a low resolution. We could come up with the following rules for data augmentation:

- small diameter, high resolution
 \implies resize (zoom in) and crop
- large diameter, high resolution
 \implies resize (zoom out), pad if needed, and crop
- small diameter, low resolution
 \implies resize (zoom in), pad if needed, and crop
- large diameter, low resolution
 \implies resize (zoom out), pad if needed, and crop

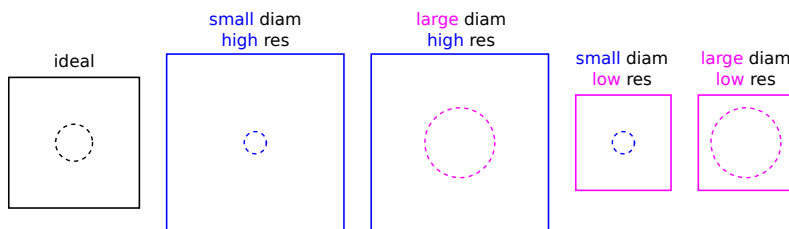


Figure 2.12: Cases of relative sizes of an object to resolutions. The average image resolution and the averaged diameter were precalculated for each dataset and were applied to image augmentation accordingly. The ideal cases were a diameter of 20 pixels in a 256×256 image array for StarDist and a diameter of 17 pixels in a 224×224 image array for Cellpose.

The chief strategy was to pre-calculate average diameters for each dataset and to match the size of objects to the desired one, which was 20 for StarDist and 17 for Cellpose, by

resizing the whole image. Then, they were cropped to the size of 256×256 for StarDist and 224×224 for Cellpose, after padded if needed. The augmentation allowed having enough samples to balance the number among dataset, which was especially important for those with small numbers of images. Consequently, those datasets went through heavier augmentation in order to sample 300 from each dataset. The augmentation included horizontal and vertical flipping and random brightness contrast changes.

4. Single-channel input : To be clear, StarDist was designed to work with single-channel images, while Cellpose with two channels images where both cell bodies (cytoplasms) and nuclei are tagged. But Cellpose is capable even with single-channel images, and so it was used to compare it to StarDist in this experiment. Hence, for each dataset, a nuclei channel was picked when applicable, otherwise the images were converted to grayscale following the same pre-processing protocol for clustering.

BUILDING VERSATILE MODELS

The experiment respected all the suggested default arguments and hyperparameters, except the number of epochs in terms of retraining StarDist and Cellpose with the meta dataset that we just created. Since the size of the dataset got much larger, the training iteration had to be longer. It was found that 400 epochs and 500 epochs yielded the best results for StarDist and Cellpose, respectively. The qualitative results on LOB-THG images that we failed to segment with the pre-trained StarDist and Cellpose can be found in figure 2.13. Also, find the F-1 score metric (find its definition in appendix C) on testing subsets in figure 2.15.

The retrained Cellpose will be called *CP* and the retrained StarDist *SD* from here on with an italic font. Pre-trained models will be denoted in lower-case letters in a monospaced font, like `cyto`, `2D-versatile-fluo`, and the like. Datasets are in upper camel case in a monospaced font, same as before: `Cellpose`, `TissueNetV1`, `LOB-THG`, and so on.

In general, the retrained models outperformed almost all the pre-trained models. **In case of Cellpose**, the result on the `Cellpose` dataset between the pre-trained `cyto` model and the retrained *CP* is particularly interesting, because `cyto` model was specialized to `Cellpose`. Yet, it was outperformed by *CP* model. We can hypothesize that the other datasets were helping to learn segment `Cellpose` dataset better. Results on `TissueNetV1` also tells the same thing. `LIVECell` was the only exception, and back in the results of UMAP and KMeans, `LIVECell` had rather big difference from the other datasets. We could suppose that the capacity of the architecture was limiting or the other datasets outbalanced. But it is still impressive that the F-1 score of *CP* on `LIVECell` dataset was behind `livecell` model only by 0.06, considering *CP* was trained with as little as 20% of the intended size (300/1512; see table 2.1).

Case of StarDist is similar to that of Cellpose. The retrained Stardist *SD* mostly outperformed pre-trained models except for one dataset which was `S-BSST265`. `S-BSST265` is a typical fluorescence microscopy images and has relatively broad connections to many datasets (again, refer to table 2.1). Assuming that `2D-versatile-fluo` model was trained

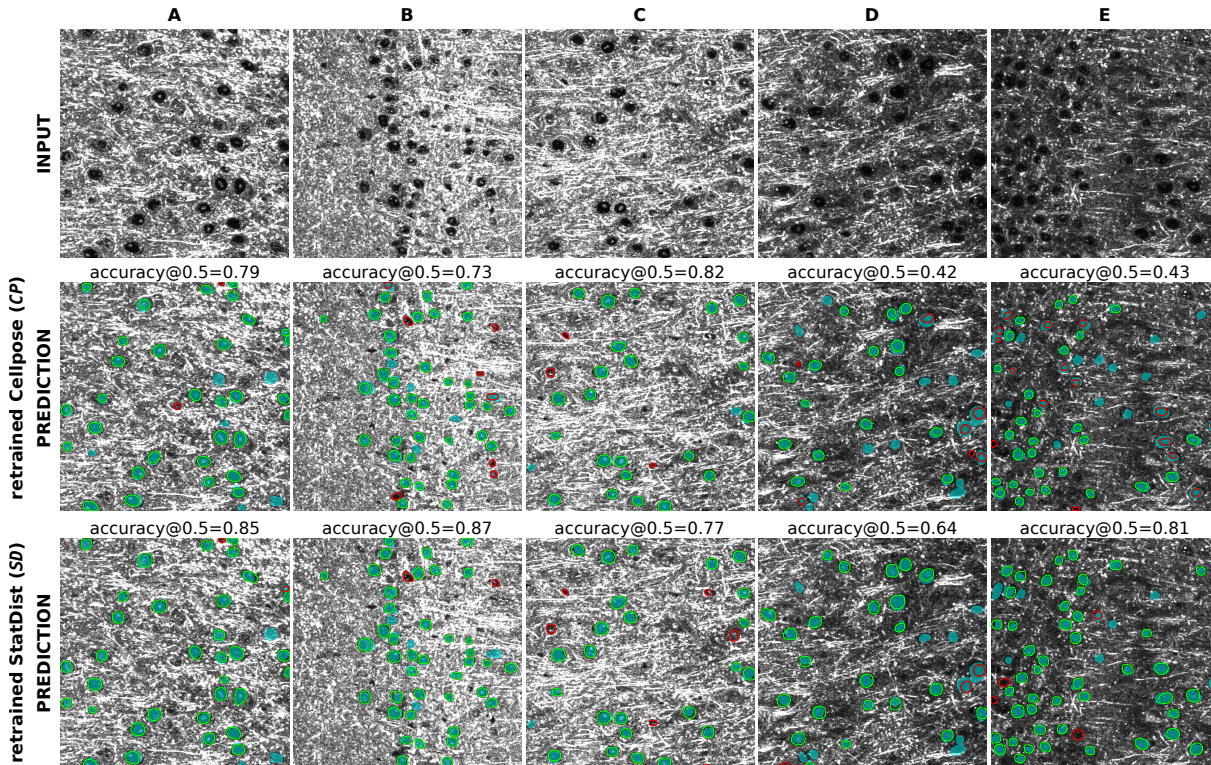


Figure 2.13: Instance segmentation results of retrained StarDist (*SD*) and Cellpose (*CP*) models on LOB-THG images. Compare the results to those of the pre-trained ones in figure 2.2. Annotation: cyan blobs = ground truth, red contour = false positive, green contour = true positive. False negatives are masks in cyan without a contour. “Accuracy” metric is what both StarDist and Cellpose used in their papers. Find its definition in appendix C.

specifically for fluorescence microscopy images, the retrained *SD* may have sacrificed capability for bioimages that are not from fluorescence microscopy. There were notable score differences between *CP* and *SD* on *Cellpose* and *LIVECell1*. Both datasets actually have whole cell annotations unlike most datasets that have nucleus annotations. Whole cells have more arbitrary shapes than nuclei, which was one of the very reasons, which Cellpose[29] was designed for. See figure 2.14.

ABLATION STUDY: LEAVE-ONE-OUT

An immediate follow-up question would be *how much impact does one dataset have?* And conversely *how good is a model on a dataset it has not seen?* To study that, an ablation study was performed, which will be referred to “leave-one-out”, where a single dataset one was taken out one by one out of 14 ones shown in table 2.1. For each ablation, a new model was trained, and dataset-agnostic F-1 scores were calculated. The result of the leave-one-out experiment is at figure 2.15, and that of score difference is at figure 2.16.

Scores across the diagonal showed direct impact from the ablation. Big impacts ($\Delta F-1 \leq -0.20$) were observed for *TissuNetV1*, *LOB-P14*, *BBBC020*, *LOB-THG*, *ComPath*, *FRUNet*, *LIVECell1*, *Cellpose* datasets in both *CP* and *SD*. It means that these datasets have loose connections to other datasets do not resemble other datasets. Especially, *LIVECell1* and *Cellpose* showed huge drops in F-1 score once each was left out. The composed dataset may still lack diversity because the score dropped rapidly when a certain dataset was left

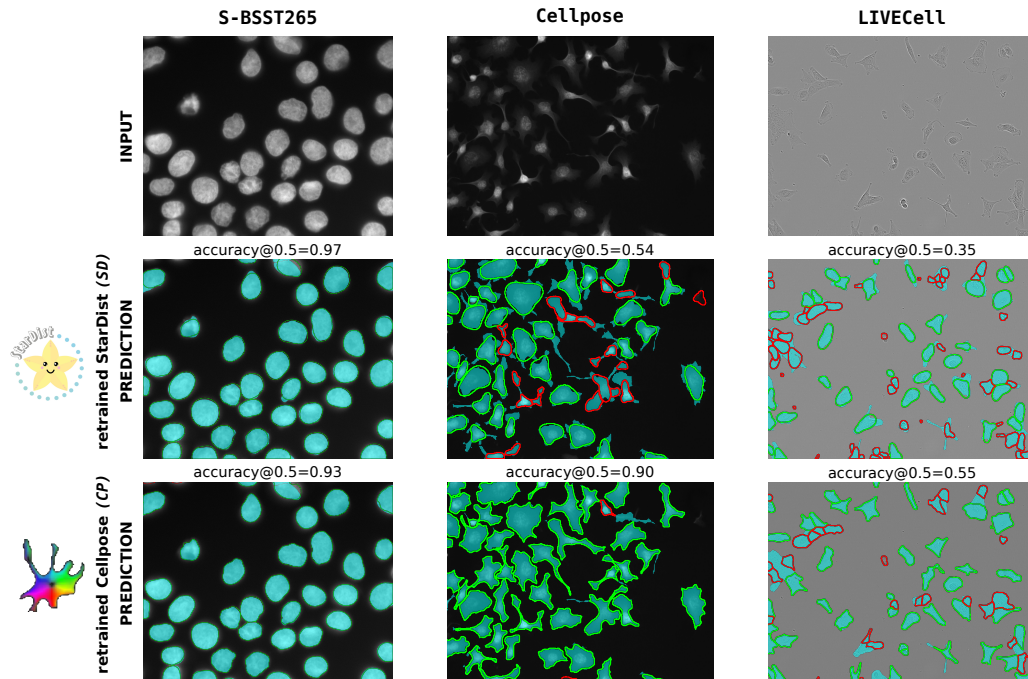


Figure 2.14: What is Cellpose good at? For the instance segmentation task, it is required for a model to recognize morphology of an object class. In comparison to nuclei, cell bodies are more diverse, thus have more diverse shapes. StarDist[22] is good at segmenting circular shapes but not at irregular blobs due to its uniformly parameterized polygon prior. Cellpose[29] is better at segmenting irregular blobs thanks to its diffusion based prior. I think that their logos represent well what their targets are. I used the retrained StarDist (*SD*) and Cellpose (*CP*) for predictions. S-BSST265 is a typical dataset with DAPI, which stains nuclei, whereas Cellpose has whole cell annotations though it is also a fluorescence microscopy dataset. LIVECell is a bright field microscopy dataset with whole cell annotations. Annotation: cyan blobs = ground truth, red contour = false positive, green contour = true positive. False negatives are masks in cyan without a contour.

out, which implies the specificity of that dataset.

At the same time, it seemed okay to remove certain datasets. This is a type of versatility we could achieve from the strategy of combining datasets. For examples, BBBC039, BBBC006, S-BSST265 had little impact whether they were excluded or not, meaning their data domains likely overlap and can be easily transferred from one to another. Once we acquired new images that fall into this domain, we could expect that these models would be versatile enough not to be retrained. This is the main reason why we would like to gather as many datasets as possible from various sources. Also, leaving one dataset out either decreased or increased scores on the other datasets, though their changes were small because the number and the diversity of the generic dataset do not change much with a single dataset left out. It demonstrates that the generic dataset is large enough not to have a huge impact when a single dataset is missing out.

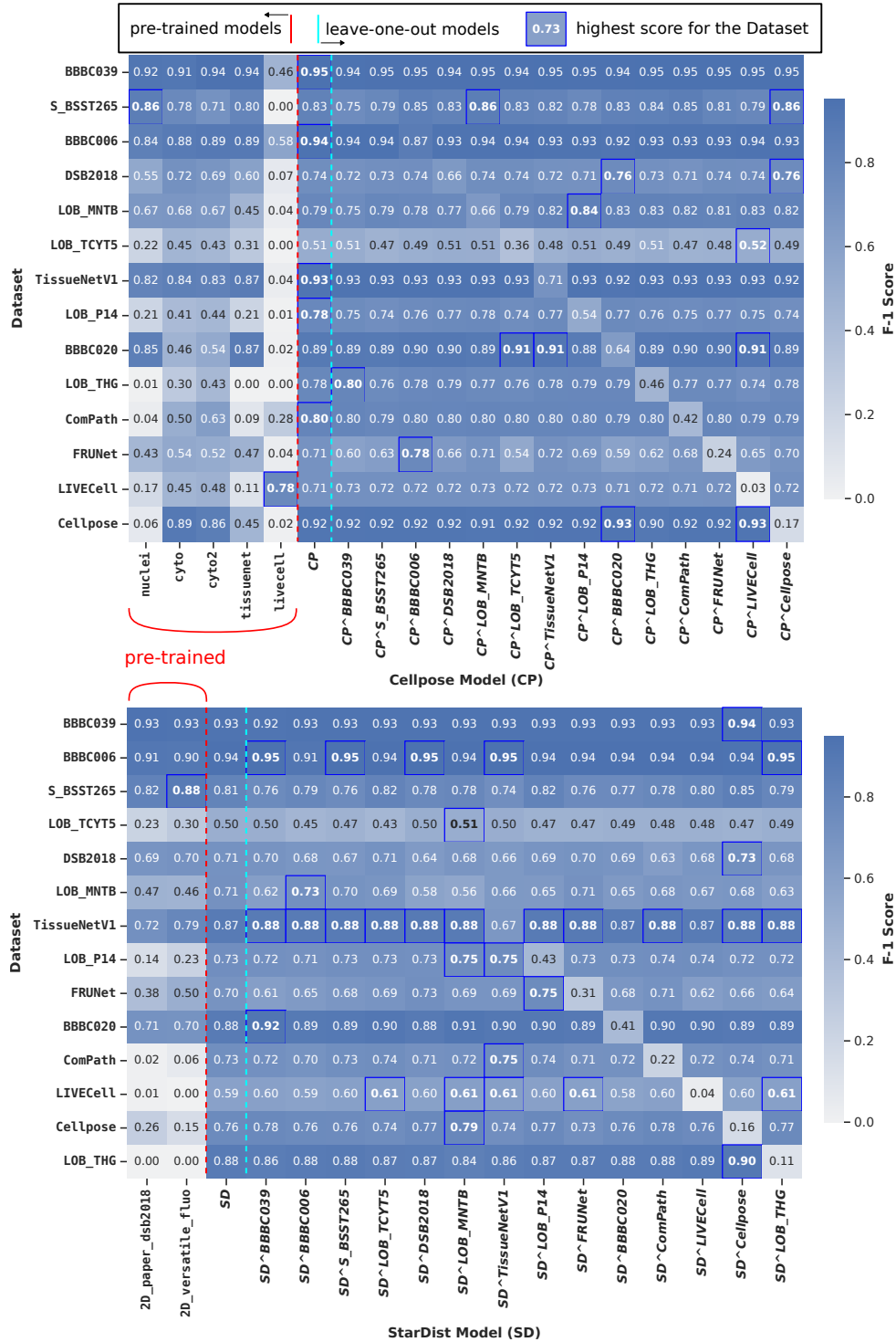


Figure 2.15: F-1 scores on datasets over models, including pre-trained models, models trained with all appeared datasets, and models from the “leave-one-out” experiment. On the model axes (x-axes), the name of a dataset after a caret (^) indicates a model trained with that dataset left out. **Bold** number with **blue** box indicates the best score for the dataset on the same row. Note that a retrained model with the highest score was annotated only once. But, when there are multiple models with the same highest score, they were all annotated. Also, the scores were rounded to two decimal places. The table was sorted by the score drop of diagonal entries of the leave-one-out experiment from the pre-trained CP or SD. For the delta values, see figure 2.16.

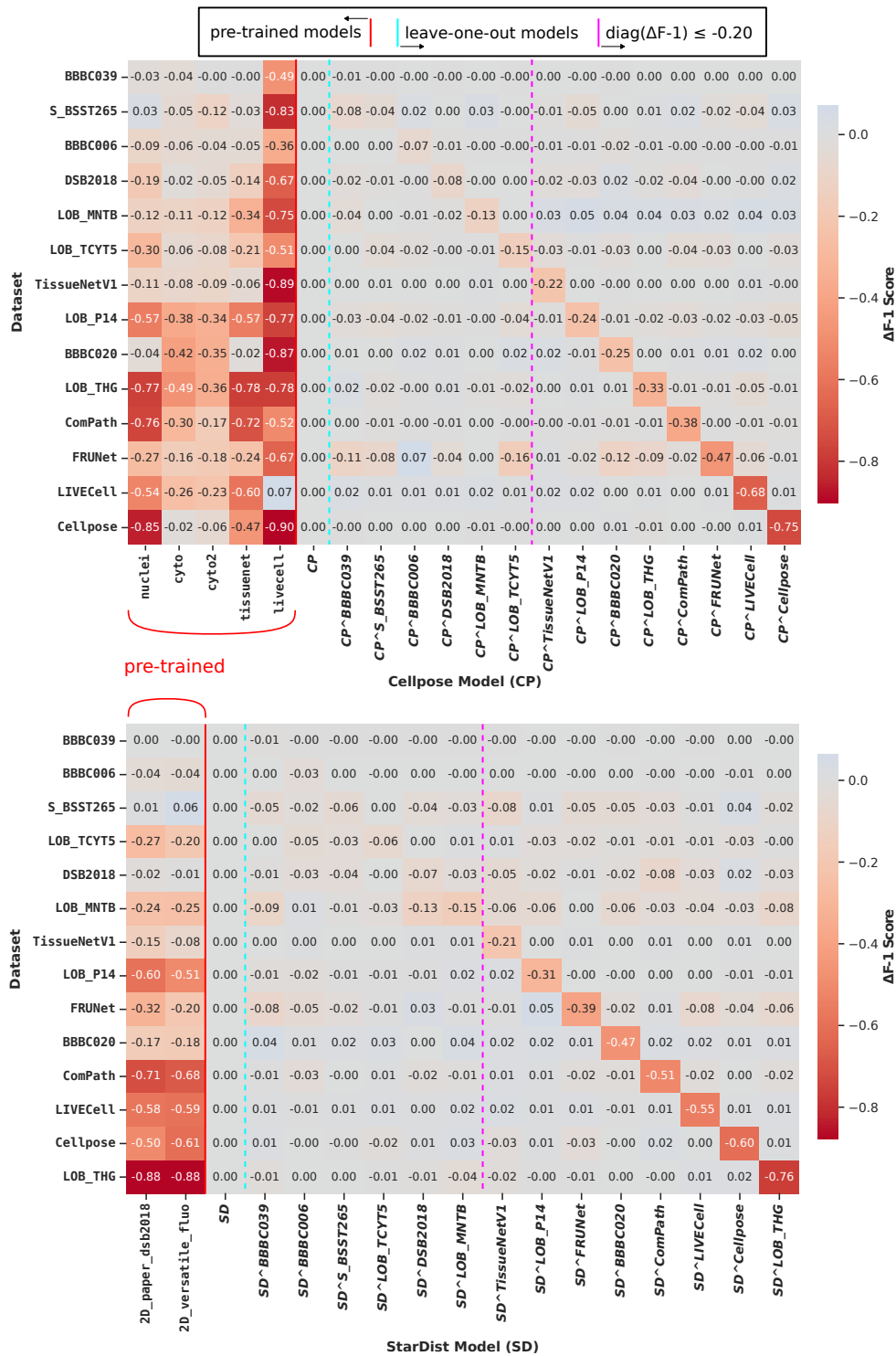


Figure 2.16: $\Delta F-1$ scores. The scores show differences relative to those of the pre-trained models *CP* and *SD*. The delta representation accentuates the diagonal for the leave-one-out study. The table was sorted in the same as in figure 2.15. Leaving one dataset out has a direct impact on the score of that dataset. But the degree differs from dataset to dataset. In the meantime, it does not affect much on the other datasets in general because the generic dataset is large enough to ignore one dataset missing out.

4 CONCLUSIONS

Powerful machine learning algorithms are becoming widely available and easily accessible. We supposed a real case where we have nuclei to segment in local bioimages by using state-of-the-art instance segmentation deep CNNs, pre-trained in supervised ways, namely StarDist[22] and Cellpose[29]. We found that pre-trained models are likely to fail on images out of the distribution with which the models were trained. The problem of making a versatile model essentially boils down to curating a generic dataset, which is not trivial due to highly diverse nature of bioimages as well as fragmented bioimage datasets. I attempted to make a generic bioimage datasets by combining as many existing datasets as possible. To address the inhomogeneous structures and formats of different datasets, I created `bioimageloader`. `bioimageloader` is a Python library to provide application programming interfaces (APIs) of public bioimage datasets, called collections. Users can easily load and combine collections thanks to a unified interface, and perform data augmentation, which is essential for generalization, utilizing a powerful and dedicated image augmentation library `albumentations`. Before retraining instance segmentation models, I demonstrated UMAP, a dimensionality reduction algorithm and KMeans, an unsupervised clustering algorithm based on deep style features to inspect 32 different datasets. The results of UMAP and KMeans provided insights and revealed relationships among datasets. Afterwards, I composed a generic dataset for the instance segmentation task combining 14 different datasets. Then, I retrained Cellpose and StarDist with balancing the size of individual datasets and keeping the target diameter of nuclei in mind through a set of data augmentation. The retrained models outperformed the pre-trained models across most individual datasets, thus achieving versatility. From the ablation study, where I left one dataset out for training, we could observe huge F-1 score drops for certain datasets. It tells that the composed dataset still lacks diversity. At the same time, though, the retrained models were highly robust against other datasets, thus achieving some degree of generalization.

5 DISCUSSIONS AND PERSPECTIVES

The size of the biomedical dataset, I think, is still small and limiting and have a lot of room to grow. Compared to normal computer vision, the size of curated biomedical datasets is far smaller for the moment. Though it is hard to compare it to different domains, COCO[137] has around 164K images whereas the total number of images that I used was a bit shy of 10K even after combining 10 collections. Open Images v4[148] has 1.9M images with 15M bounding box annotations, which is roughly 15 times of what TissueNet[43], the latest and the largest bioimage dataset, offers.

Annotation formats need to be standardized as well as the image formats. While mask annotation formats have been converging and are relatively consistent and shown rather rapid progress, tasks like cell tracking[19] and neuron tracing[68] are showing slow progress. I think that it has to do with the lack of consensus how to annotate them and evaluate them. Unifying tools could be the first step to have uniform formats[43]. The issue of image formats is more about organizing dimension rather than file extensions, though the issue of biomedical file extensions is also important and has been addressed by OME team and community[74, 73].

The diversity is also lacking based on the results I presented. Specifically, the leave-one-out study revealed that certain datasets have no similar alternatives. Balancing collections that I went through is essentially a similar process to ensure diversity when curating a dataset. Though, Cellpose[29] and TissueNet[43] have already tried to gather diverse images from different sources, to classify images within a big dataset, and to measure class-agnostic metrics, I found that their data has rather small overlap with other data.

Speaking of the balancing strategy, I only applied the simplest data augmentation not to overcomplicate the experiment. More elaborate augmentation strategy could further improve overall performance, like NucleAIzer[40] that used deep style transfer.

Study of architecture might be a key to see an immediate improvement over the increased size of the dataset. Two architectures that I tried, StarDist and Cellpose, used roughly a few hundreds of images, which is a small number compared to what I tried (about 8K, though I did not use all due to the balancing procedure), and both were based on small U-Net architectures[20]. Mesmer[43] already attempted a bigger one, namely ResNet[49] to accommodate the larger TissueNet dataset, though Mesmer is not a native instance segmentation model.

In terms of `bioimagerloader` itself, I found that it turned out to be more than I anticipated. It enabled me not only to manage many inhomogeneous datasets, but also to conduct experiments in a reliable and reproducible way. I am glad that I poured a lot of time to make it easy to use and get contribution, so that others can try what I could not think of and further develop machine learning algorithms using `bioimagerloader` as well as the library itself. `Bioimagerloader` has a lot of potential. Firstly, it needs to support more existing datasets. It could support other types of `Dataset` interfaces, such as time series and 3D volume, as well as other annotation types, such as bounding box, key point, panoptic mask, and so on.

Lastly, while I tackled relatively poor versatility of existing supervised machine learning solutions in bioimages from the perspective of the data curation, realizing versatility and generalization can be achieved in different ways. Recently, self-supervised learning has been proven effective to build large and generalized models and been gaining a lot of attention[84, 102, 111, 105, 91]. The next chapter will be about self-supervised learning.

CHAPTER 3

NU-NET: SELF-SUPERVISED VERSATILE CNN FILTER

Contents

1	Introduction	90
1.1	Background	90
1.2	Related works	92
2	Methods	94
2.1	Perceptual loss	94
2.2	Morphological loss	95
2.3	Data	96
2.4	Training	98
2.5	Early NU-NETS	100
3	Results	103
3.1	Perceiving styles	103
3.2	Loss curves	104
3.3	Contrast enhancement	104
3.4	Controlling filtering magnitude	106
3.5	Side effect: Long objects	107
3.6	Application: Napari plugin	108
3.7	Artifacts and Limitations	110
3.8	More figures	111
4	Discussions and Perspectives	113
5	Conclusions	118

ABSTRACT

Supervised deep neural networks have become the dominant method for segmentation tasks in computer vision. Yet, supervised learning suffers from the cost of curating labels, in particularly in life sciences where each experimental dataset is different and highly specific. I propose NU-NET, a self-supervised convolutional neural network architecture. Self-supervision allowed training NU-NET with large amount of images without paired target labels thanks to a novel loss named morphological loss, inspired by deep style transfer applications. By incorporating around 12K images from 25 different datasets, NU-NET is versatile and robust to new images. In practice, NU-NET is an enhancement filter for blobs, such as cells and nuclei. I will demonstrate its ability to improve contrast across a wide range of datasets and be used as a filter prior to segmentation. The pre-trained model is available on online and potentially useful to related vision tasks via transfer learning. NU-NET is also accessible through a plugin for Napari viewer.

The work presented in this chapter form the basis of a preprint, included in appendix D. It has been submitted to the ICCV (International Conference on Computer Vision) 2023 workshop on bioimage computing. The preprint includes a couple of additional numerical experiments and validation not included in the chapter.

1 INTRODUCTION

NU-NET is a recursive acronym, meaning “NU-NET is not a U-NET[20]”^a. U-NET was born in biomedical imaging community and used so many times that it was hard to find any other architectures when it comes to biomedical vision task. Their outputs may look alike, but NU-NET is positioned between a generative model and a segmentation model, whereas U-NET predicted probability maps as a segmentation model, at least when it was first introduced^b. Surprisingly, NU-NET is not at all based on U-NET architecture, though it could have been.

^a Its code name was “nude net” in the sense that it strips out small details and keeps big structures.

^b Diffusion model is the state-of-the-art generative model and has been recognized since 2020[33]. It picked up U-NET architecture. The gap between a generative model and a segmentation model is narrow.

1.1 BACKGROUND

Supervised learning has shown excellent performance in overall computer vision tasks with advent of convolutional neural networks (CNN) as well as of large datasets such as ImageNet (ILSVRC)[149] with 14M images, MS-COCO[137] with 300K, Open Image Dataset V4[148] with 9M, JFT-300M[113] with 300M images. When it comes to bioimages where images are acquired through microscopes, however, it is difficult and expensive to curate such large datasets for supervised learning for a number of reasons. **First**, bioimages are highly diverse. Acquiring bioimages requires a pipeline of multiple steps which take time and add complexity. It means that the look of acquired images could be easily shifted depending on any changes from the pipeline, such as types of

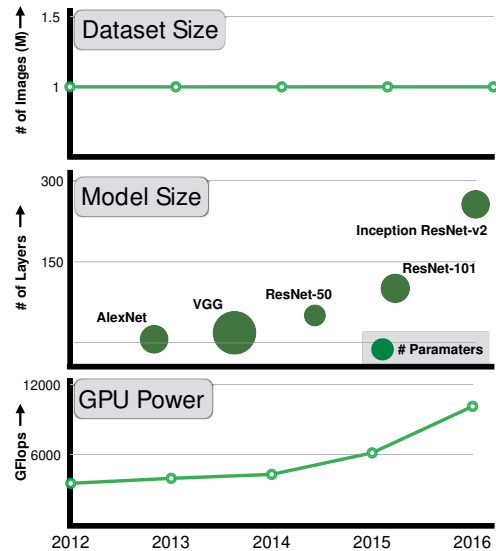


Figure 3.1: JFT-300M team questioned why the size of dataset remains relatively unchanged and studied it by constructing JFT-300M dataset with 300M images. Biomedical imaging community also has been facing the same issue, however, the size of both dataset and model is much smaller compared to general computer vision. One way to detour the issue is self-supervised learning where we do not need to worry about the costly annotation process. Source: JFT-300M dataset[113]

samples, sample preparation steps, lasers, microscopes, and even environments. **Second**, labeling targets demands scientists and experts from preprocessing images to identifying targets. **Third**, making a dataset public, which has become *de facto* standard way to do machine learning, is complicated when it comes to bioimages due to licenses and funds bound to researches as well as logistic difficulty to share. As a consequence of lack of large curated datasets for machine learning, the size of state-of-the-art neural networks in the bioimage community still remains small as well as their capability. For instance, popular networks like StarDist[22] and Cellpose[29] were both based on almost unchanged U-Net architecture[20] and trained just with a few hundreds of images. In the meantime, the computer vision community, in general, moved on to bigger and deeper vision models such as VGG[85], Inception[96], ResNet[49, 150], and ConvNeXt[151], utilizing large datasets. However, recently, the expansion of both networks and datasets seemed slowing down, partially because they started facing the intrinsic limitation of supervised learning: lack of curated, unbiased, and diverse data [113].

Naturally, other learning schemes have gained popularity over the supervised one. Such schemes are unsupervised learning, weak supervised learning, semi-supervised learning, and self-supervised learning. In the nutshell, these schemes attempt to get rid of paired target labels at the cost of compromising performance of fully supervised learning. Self-supervised learning, among these, has become promising, starting from self-attention (transformer)[102] and contrastive learning[152, 110] to vision transformer (ViT)[105], in order to address such limitations of supervised learning in computer vision tasks. Apart from its biggest advantage, that is to get rid of need of target labels, self-supervised learning allows inclusion of more diverse and unbiased data and subsequently building larger and more generic models, which in turn benefits transfer learning[153, 154, 155] and foundation models[91]. But the current self-supervision frameworks have their own limitations.

One of the biggest limitations is that self-supervision alone is often not enough to perform actual downstream tasks, and thus it is used as pretext tasks (or surrogate tasks). The pretext task is in essence a process for neural networks to get accustomed to given data ahead of downstream tasks which are typically accompanied by supervised learning.

I devised a novel self-supervised loss named morphological loss, and built NU-NET, a deep CNN to filter nuclei and cells in bioimages. NU-NET not only takes advantages of self-supervised learning, but it can also be directly used as a content-enhancing filter for better detecting nuclei and cells. In addition, we managed to utilize 28 different datasets together, including 4 local and novel datasets, to train a single generic model. Combining more than 20 datasets has not been reported yet to our knowledge. This strategy to combine datasets, which have the same targets but have been acquired from various sources, allowed us to build a versatile and generic model. Potentially being a generic model, NU-NET can be applied to a broad range of tasks like detecting nuclei and cells, and could play a role as a foundation model[91, 153, 154, 155] and transferred to solve larger and more sophisticated tasks.

Contributions of this work are

- a novel loss for self-supervised learning
- combining multiple datasets from different sources for generalization
- a functioning machine learning model only with self-supervision (without downstream training)
- a novel content-enhancing filter based on a CNN
- sharing pre-trained NU-NET for transfer learning and open science
- an application for easy use of NU-NET (a Napari plugin)

Pre-trained NU-NET is available on its project page [not public yet]^a and is uploaded on Bioimage Model Zoo [WIP, too]^b[79]. In addition, the application of NU-NET is available in a form of plugin for Napari viewer[37] (available but not released yet).

^a Project Git repository on GitHub: <https://github.com/sbinnee/nunet/>

^b BioImage Model Zoo: <https://bioimage.io/>

1.2 RELATED WORKS

NU-NET is a variant of neural style transfer applications[87, 86, 156], which are a type of representation learning. In particular, it shares a lot of ideas from the paper “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”[88]. For examples, the neural network architectures was adapted as well as the training procedure. What is distinct in NU-NET is that it attempts to generalize a single style via, so-called morphological loss, compared to other style transfer applications which predominantly sought ways to improve quality of resulted outputs[157, 158, 159, 160], or ways to incorporate as many styles as possible[161, 143, 162] in a single model.

NU-NET employs the self-supervised learning approach but differs from currently popular self-supervised learning methods such as self-attention[102, 103, 84], contrastive learning[110, 109, 163, 111], and discriminative learning[125]. Self-attention is an architectural module in neural networks, which encodes relationships of given data using their representations or embeddings. Contrastive learning constructs a loss which makes contrast of positive samples and negative samples. Discriminative learning is similar to the contrastive learning in the fact that it classifies targets into two groups, but mainly focuses on data augmentation to realize discrimination. Generally, these self-supervised learning methods act as a pre-text task and use the transfer learning technique for a downstream task [155, 164, 112, 91].

Cellpose[29] is the state-of-the-art instance segmentation neural networks for nuclear and cellular images. Though it is not a directly related work, NU-NET shares one of its goals, that is to build a generic model. Cellpose team gathered bioimages from multiple sources as well as natural images such as those of fruits, rocks, etc., in order to generalize their model. I took a different route and collected existing bioimage datasets instead of collecting images. Also, Cellpose is a supervised model whereas NU-NET is self-supervised. With the recent success of machine learning in bioimages, many public datasets have become curated and available. I gathered 23 public datasets specifically targeting nucleus/cell images, anticipating that diverse sources of the same target would help to generalize NU-NET. The total number of images amounted to 7.6K which is about 10 times bigger than that Cellpose was trained with.

NucleAIzer[40] is another state-of-the-art instance segmentation neural network that aims to be versatile and generic, and shares some ideas with NU-NET. In contrast to StarDist[22] and Cellpose[29] which used U-Net as a base architecture, nucleAIzer is based on Mask R-CNN[41] as well as “style transfer”¹ using pix2pix[47]. The main idea is to simulate training image/label pairs from synthesized nucleus masks for supervised learning. While nucleAIzer utilized “style transfer”, it was used to simulate a supervised training set. NU-NET extends the loss of style transfer, and it is a style transfer network itself, in a sense.

CEM500K[165] is a dataset of 500K cellular images from electron microscopy (EM). Their team collected images without labels and applied a self-supervised learning method, specifically the momentum contrastive learning[163]. They focused on data curation process and effectively reduced 5.3M raw images to a compact set of 500K images by getting rid of uninformative images via image hash and Hamming distance. They demonstrated effectiveness of transfer learning with self-supervised learning through a set of downstream tasks. NU-NET focuses more on fluorescence nuclei which look vastly different from cellular EM. I actually decided to exclude CEM500K from the experiment for that reason. Furthermore, as a contrastive learning model, their model had to be transferred to function, whereas NU-NET can be used an enhancement filter for nuclei and blob-shaped cells.

SaliencyNet[166] was published recently, in fact, while drafting this manuscript. Their work is based on CycleGAN[167], which is an image-to-image translation[47] model. Cy-

¹ “Style transfer” by pix2pix is methodologically different from aforementioned style transfer[87, 88] which NU-NET utilizes. The former uses generative adversarial network (GAN)[48] to generate fake images, whereas the latter uses a pre-trained neural network to replicate input images using their representation.

cleGAN is an unsupervised generative model since it is a GAN[48], and it jointly trains two GANs and was already tried in bioimages by SECGAN[168] (segmentation-enhanced CycleGAN) for EM images. SaliencyNet employed CycleGAN too but targeted light-sheet microscopy images with low SNR (signal-to-noise ratio) to achieve high SNR as a super-resolution application. Overall, the idea of SaliencyNet is somewhat similar to that of NU-NET. It employed an unsupervised learning method and functions as an enhancement filter for nuclei. However, NU-NET’s target is broader and not specific to a dataset. Additionally, NU-NET can be considered as a super-resolution network too since the original architecture[88], which NU-NET is based on, had super-resolution usage. But its behavior is closer to that of semantic segmentation models.

Speaking of behavior of NU-NET in comparison to other applications, the output may look similar to that of image restoration[50, 166] or that of noise removal models[54, 51]. But they all have fundamentally different target losses and different purposes. NU-NET is a content enhancing filter, not an image restoration nor a denoising model.

2 METHODS



Figure 3.2: Neural style transfer applications focused on improving the quality and the capacity to adopt different styles as many as possible. This particular network in the figure[143] was trained on roughly 80K paintings and was even able to interpolate unseen styles. NU-NET was built on the idea that it should be fairly easy to represent a single style once a model has many images with the same style.

Neural style transfer application can be a powerful tool, and the idea came from what if it is extended to the semantic segmentation task. Binary semantic masks have such a distinct look that background is black and foreground is white. Then, it should be easy for deep CNNs to represent the look of these masks and generalize it as a style through representation learning. This is the main idea behind NU-NET, and its manifestation is the morphological loss. In large, this method section has three parts: loss, data, and training scheme.

2.1 PERCEPTUAL LOSS

Perceptual loss is a type of representation learning. It was used to understand what CNNs[141] perceive. It was also picked for texture modeling[126, 169]. DeePSiM (deep

perceptual similarity metrics)[170] used it to generate images together with GANs. But, it was mostly used in neural style transfer applications. The gist of the perceptual loss is to borrow and take advantage of perceptions (representations or features in other words) of a pre-trained deep CNN. Deep CNNs often have logical blocks of layers, and these blocks can individually perceive certain features[142]. Then, notions of so-called content loss and style loss were introduced[87]. The content loss is a generative loss that attempts to reproduce input data through a single latent feature vector from a pre-trained deep neural network. The definition of the content loss \mathcal{L}_c is shown in equation 3.1. The content loss \mathcal{L}_c optimizes $\hat{\mathbf{y}}$ to reproduce content \mathbf{c} based on features at layer F^l of a pre-trained deep CNN. The loss is defined incorporating the mini-batching technique where N denotes a mini-batch size:

$$\mathcal{L}_c(\mathbf{c}, \hat{\mathbf{y}}, l) = \frac{1}{N} \sum_{n=1}^N \sum_{i,j} (F_{ij}^l(\mathbf{c}) - F_{ij}^l(\hat{\mathbf{y}}))^2 \quad (3.1)$$

The choice of layer l has great importance in particular for the content loss, because what is considered as content is determined largely by the size of their receptive fields at layer l . In general, the concept of content is recognized as a high-level feature, thus the required receptive field should not be too small.

2.2 MORPHOLOGICAL LOSS

Morphological loss is what makes NU-NET original. It is an extension of the style loss described in neural style transfer applications[87, 86, 88]. In the context of the neural style transfer, the morphological loss generalizes a certain style, which has binary values with a set of targets having a consistent morphology. In fact, there has been few attempts to model or generalize one particular style, because supposedly a style usually meant a single painting instead of a group of the same style of paintings, probably due to their high complexity. Generalizing a simple style from multiple images with the same style should be easier. In principle, the morphological loss could generalize any morphologies as long as the given morphologies are consistent. We focused on round shapes, which can represent nuclei or cells. This morphological prior will be called, in another name, blob-mask style. In this sense, StarDist[22] has a similar goal to provide star-convex prior to pick nuclei. However, StarDist’s approach is explicit, and the prior is deterministic and parameterized, whereas NU-NET’s approach is implicit, and the prior is learned through the loss.

Morphological loss is calculated in mini-batch fashion. The idea is similar to that of the memory bank[171], to sample mini-batches from a bank of images, which consists of a fixed number of predefined classes. But our bank has only one style, to be specific blob-mask style, defined by a set of targets. Morphological loss (\mathcal{L}_m) is defined in the equation below with N_s referring to a mini-batch size of targets, $\hat{\mathbf{y}}$ to an image to be optimized, and \mathbf{s}_n to n -th target within a mini-batch \mathbf{s} :

$$\mathcal{L}_m(\mathbf{s}, \hat{\mathbf{y}}) = \frac{1}{N_s N_l} \sum_{n=1}^{N_s} \sum_l (G^l(\mathbf{s}_n) - G^l(\hat{\mathbf{y}}))^2 \quad (3.2)$$

Combining equations 3.1 and 3.2, the total loss (L_t) is defined as below, where we linearly weight each loss with coefficients w_c and w_m , respectively.

$$\begin{aligned}
\mathcal{L}_t &= w_c \mathcal{L}_c(\mathbf{x}, \hat{\mathbf{y}}, l) + w_m \mathcal{L}_m(\mathbf{s}, \hat{\mathbf{y}}) \\
\mathcal{L}_t &= \mathcal{L}_c + \frac{w_m}{w_c} \mathcal{L}_m = \mathcal{L}_c + w \mathcal{L}_m \\
w &= \frac{w_m}{w_c}
\end{aligned} \tag{3.3}$$

2.3 DATA

Whether it is supervised learning or self-supervised learning, it is imperative to use various sources of data in order to make a machine learning model generic and versatile. Massive and well curated datasets such as ImageNet[149] and MS-COCO[137] are great examples that paved the road towards generic models for computer vision tasks. But currently such large nucleus/cell datasets do not exist. Even TissueNet[43], which is supposedly the largest dataset for nuclei and provides over 2M annotations, turned out not generic enough in other experiments (such as one in chapter 2). And the total number of images is less than 7K, which is impressive but still lacking.

We gathered 25 datasets having nuclei, cells or both, of which one dataset was from local THG (Third-Harmonic Generation) microscopy. The total number of images amounted to 12K. They are listed in table 3.1 and some sample images can be found in figure 3.3. Not all of them provide annotation, nor they provide the same types of annotation. However, they are all useful for training NU-NET as long as they contain the round nucleus/cell targets in images.

Due to the nature of bioimages, in particular to staining techniques, most datasets offer a single channel which tagged either nuclei or cells. We selected those channels if we could and converted multichannel images to grayscale ones otherwise. Another treatment we applied was related to the average sizes of nuclei and cells within each dataset. We believed that the relative size of target objects to the input image size plays a significant role in fully connected CNNs[172], and that it was important to keep it as consistent as possible. Most datasets had consistent looks within each, including sizes of target objects, however, they all had different resolutions of images and sizes of nuclei and cells across datasets. Therefore, we built data augmentation protocols, which consist of random cropping and resizing for each dataset individually. Then we sampled images to match the relative sizes of nuclei and cells from one dataset to another. Last but not least, balancing number of samples across datasets was the most tricky issue, since all datasets had different numbers of images, inducing biases. We chose the minimum number of samples among the datasets after considering data augmentation, and simply set it as a constant number of samples throughout all datasets mainly to avoid big bias towards those having larger number of samples.

As for actual morphological targets in practice, we used foreground mask annotation of DSB2018 and BBBC006. Although, it could have been all the available annotation, for the proof-of-concept, we selected a few with primary criteria being diversity and availability. DSB2018 is a collection of images from various sources, known for its annotation quality, and widely used in many studies. BBBC006 was selected to prove that morphological targets need not be precise and manual, because its mask annotations were automatically

Id	Acronym	Number	Annotation	Id	Acronym	Number	Annotation
1	TissueNetV1	6990	○	14	S-BSST265	79	○
2	BBBC041	1328	△ (U)	15	FRUNet	72	○
3	BBBC026	864	X (B,C)	16	BBBC016	72	X (B)
4	BBBC006	768	○	17	BBBC018	56	○
5	DSB2018	735	○	18	TNBC	50	○
6	BBBC021	240	X (B)	19	BBBC002	50	X (C)
7	BBBC039	200	X	20	ComPath	30	○
8	BBBC015	144	X (B)	21	UCSB	58	△
9	DigitPath	141	△	22	BBBC020	25	○
10	MurphyLab	100	○	23	BBBC007	16	○
11	BBBC013	96	X (B)	24	BBBC008	12	○
12	BBBC014	96	X (B)	25	LOB-THG	14	○
13	Cellpose	100	○	-	Total	12,336	

Table 3.1: List of nucleus/cell datasets used to train NU-Net. Annotation availability and type are usually the most important factors when it comes to selecting datasets for machine learning, but not for NU-NET thanks to the self-supervised learning characteristic. In total, 25 datasets were used, including an in-house dataset (LOB-THG), and the total number of images amounted to 12,336. Symbols for annotation (○: provide complete segmentation mask targets, △: partially annotated masks, X: do not provide mask targets; C: counts, B: biological labels, U: bounding boxes). Find the full table in appendix B. Note that only a portion of BBBC021 was used.

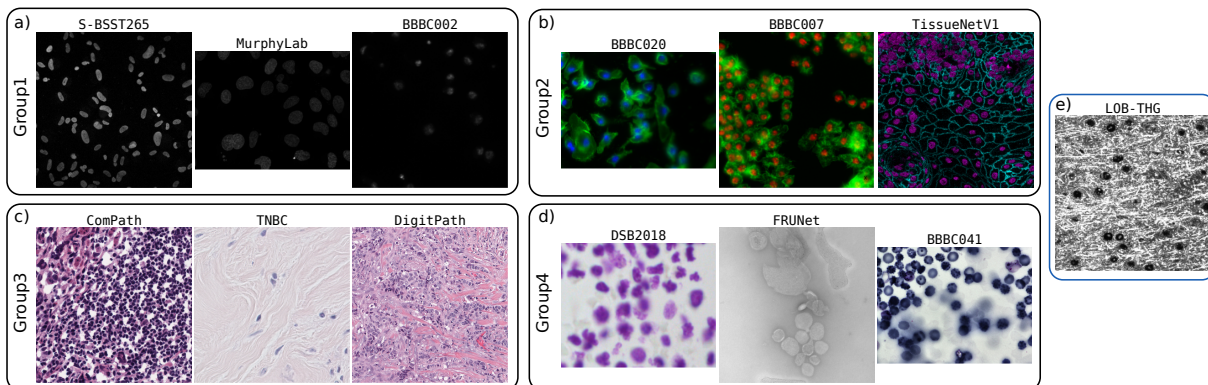


Figure 3.3: Sample images grouped by image types. Note that not all datasets are presented. a) Group1: grayscale images. b) Group2: two channel images, mostly tagging nuclei and cytoplasm. c) Group3: histopathology images. d) Group4: others. DSB2018 is a collection of nucleus and cell images from various sources; FRUNet is an EM (electron microscopy) dataset; BBBC041 presents blood cells. e) LOB-THG: local images.

feature block	VGG19 layer	content loss	morphological loss
block1	relu2_2	X	○
block2	relu3_4	○	○
block3	relu4_4	X	○
block4	relu5_4	X	○

Table 3.2: Feature blocks and their corresponding layers in pre-trained VGG19. And used blocks to calculate each content and morphological loss (a.k.a. perceptual loss).

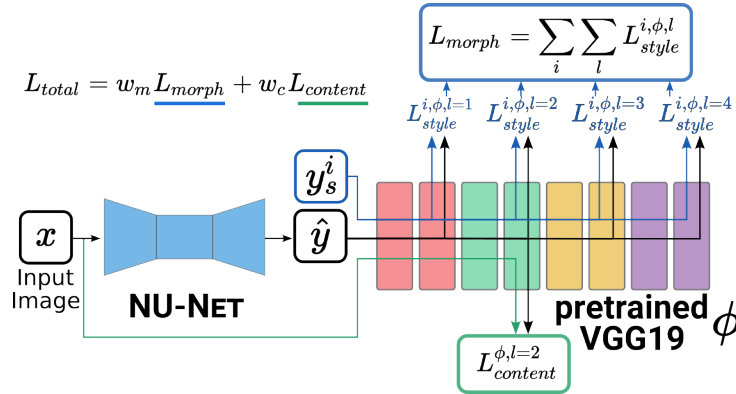


Figure 3.4: Schematic for training NU-Net. x denotes input image, \hat{y} output image. y_s^i is a sample target from a mini-batch used for morphological loss. N_s ($i \in N_s$) refers to a batch size for the morphological loss, same as in equation 3.2. $l \in \{1, 2, 3, 4\}$ indicates the feature block in VGG19. This schematic was inspired by [88].

acquired by using CellProfiler[173]².

2.4 TRAINING

NU-NET overall follows the same training procedure of neural style transfer application using perceptual losses[88]. The main difference is the morphological loss, which needs its own mini-batch from its own bank of targets. Also, we used AdamW[174] as the optimizer. We kept the same architecture as [88] for NU-NET, while replacing a pre-trained VGG16 with VGG19[85] for perceptual losses. We used predefined feature blocks (in table 3.2) to calculate content and morphological losses. Schematic for training procedure is presented in figure 3.4.

Architecture NU-NET is a fully convolutional network (FCN)[172]. The architecture is almost identical to the image transform network in [88], which got inspired by DCGAN[175] and ResNet[49]. In particular, the residual block of ResNet is an important component because it encourages learning the identity function, which aligns well with the generative function of NU-NET. The only change that we made was the normalization layer after convolutional layers. All the batch normalization layers[176] were replaced with the instance normalization layers[157]. The purpose of the batch normalization is to get rid of instance variance and to make training fast. However, it may simplify the input too much that it hurts generalization, especially in the presence of naturally diverse image

² Find out more details at <https://bbbc.broadinstitute.org/BBBC006>.

source like bioimages. Instance normalization does a channel-wise normalization for each instance. The benefit is that each channel is independent thus encodes more diversity. Find the diagram in figure 3.5

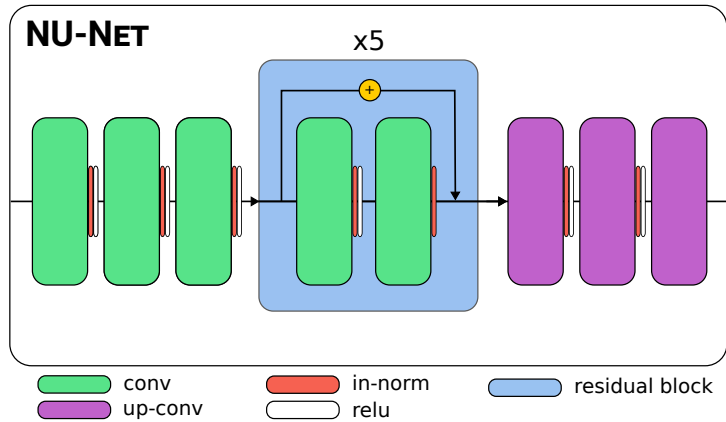


Figure 3.5: Architecture of NU-NET. It is almost identical to one from [88]. The residual block repeats 5 times in the middle, encouraging NU-NET to learn the identity function. Instance normalization[157] allows more diversity compared to the batch normalization[176]. It has 16 convolutional layers in total, which is not particularly deep and makes NU-NET fast and light.

The architecture is intended to be small and fast because the behavior of NU-NET is mainly to reconstruct the input in a coarse way and to apply small changes. In addition, the size of data still remained relatively small, so there was no reason to have a big architecture.

Data augmentation A set of contrast augmentation was performed as well as a set of basic transformations during training. Simply put, one aspect of given task is, in a way, reduced down to detecting local contrast between foreground and background. That being the case, contrast augmentation would make NU-NET robust whichever value ranges input images lie in, and would cover as many contrast ratios from low to high as possible.

In detail, three contrast augmentation algorithms in random order and set certain ranges of each parameter so that the augmented images do not saturate above or cut off below the data range of unsigned 8-bit integer (UINT8). The three augmentations were gamma (v_γ), logarithm (v_{log}) and sigmoid (v_σ) contrast adjustment method, defined such in equation 3.4, where v denotes a pixel value as in UINT8, and γ , A and b are constant scalar parameters.

$$\begin{aligned}
 v_\gamma(v, \gamma) &= 255 \times ((v/255)^\gamma) \\
 v_{log}(v, A) &= 255 \times A \log_2(1 + v/255) \\
 v_\sigma(v, A, b) &= 255 \times (1 + \exp(A(b - v/255)))^{-1}
 \end{aligned}
 \tag{3.4}$$

Contrast inversion Foreground does not always mean having higher pixel values than those of background, which will be denoted WB^3 . It was the case for some datasets that foreground objects appeared darker than background, which will be referred to BW ,

³ WB : **W**hite foreground on **B**lack background. BW : **B**lack foreground on **W**hite background.

when rendered. It turned out that NU-NET got confused in case of *BW*, because the morphological loss targets *WB* images. In the end, all *BW* cases were inverted to *WB* during training for NU-NET to have a consistent behavior.

2.5 EARLY NU-NETS

This small section is retrospective and describes the evolution of designing NU-NET.

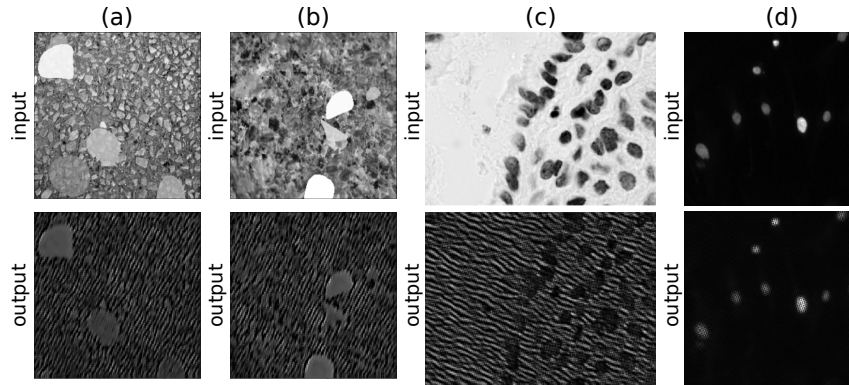


Figure 3.6: Early NU-NETS based on VAE with the attractive and repulsive losses. Three different models were used: (a, b) together from one, (c) from another, and (d) from the other. (a, b) Outputs have angled stripes. (c) An output with a different angle. (d) An output with a highly structured pattern. All the outputs were not expected nor desired, but gave me hints to move forward. Input sources: (a, b) Simulated blobs on textures from KTH-TIPS database[124]; (c, d) DSB2018[58]

VAE and disentanglement The first direction that we took after the main idea of translating the segmentation task into the neural style transfer task was VAE (variational autoencoder)[119]. The assumption was to disentangle the style from the content features in a latent space. Even though, all the neural style transfer applications do a bit of disentanglement of the two, we wanted more control and a proof-of-concept for the purpose of study. VAE was a good choice, at least for the first trial, because its latent space was much smaller than normal feature maps, meaning that latent vectors were easy to control, manipulate, and study. Also, there were some research papers regarding disentanglement of attributes in generative modeling [120, 121, 177, 178, 179].

The initial form of losses was a cosine similarity and had three parts as opposed to two in the final form. The content loss was the same, but there were two other losses for modeling the style part, which we called attractive loss and repulsive loss. The idea was essentially similar to contrastive loss[110] and SimCLR[109] in the sense that there were one force to encourage and the other to discourage a target. The goal of two losses was to search a hyperplane in a latent space, which represents a style of given targets, assuming that all the given targets have a single consistent style. When the dimension of a latent space ($\mathcal{Z} \in \mathbb{R}^{\mathcal{D}}$) is big, finding a hyperplane becomes much harder and needs far more data points than when the dimension is small. Say the hyperplane we aimed to find had a dimension of $\mathcal{D} - 1$, then we need at least $\mathcal{D} - 1$ vectors to define that plane. But we need far more data points in practice, since the solution is only an approximation. For VGG[85], this would mean $\mathcal{D} \sim 10\text{K}$ at the minimum, which we could not afford.

So the latent space must be small, however, at the same time, if \mathcal{D} were too small, the hyperplane may not be capable of representing complicated features. For this reason, we needed to manipulate the size of latent space, and it was easy in VAE.

We trained VAEs and confirmed that Gram matrices of latent vectors of \mathbf{x} and \mathbf{y}_s could be separable just like in figure 3.9 using a couple of clustering algorithms. As for losses, the attractive loss was a simple cosine similarity loss between $\mathbf{z}(\hat{\mathbf{y}})$ and $\mathbf{z}(\mathbf{y}_s)$, and the repulsive loss acted as a regularizer to make $\mathbf{z}(\hat{\mathbf{y}})$ away from $\mathbf{z}(\mathbf{x})$. The repulsive loss could be defined either as L1 loss or L2 loss as below where N is a batch size:

$$\begin{aligned}\mathcal{L}_{\text{repulsive}} &= \frac{1}{N} \sum_{i=1}^N \|G(\hat{\mathbf{y}}_i) \cdot G(\mathbf{x}_i)\|_1 \\ \text{or} &= \frac{1}{N} \sum_{i=1}^N \|G(\hat{\mathbf{y}}_i) \cdot G(\mathbf{x}_i)\|_2^2\end{aligned}\tag{3.5}$$

Fast-forwarding, we ended up getting images that stylized with foreign textures which did not seem to come neither from both content image (\mathbf{x}) nor style image (\mathbf{y}_s). The outputs seemed to have too much influence of \mathbf{y}_s , as shown in figure 3.6. We believed that the assumption had a hole. That is, the target vector $H(\hat{\mathbf{y}})$ being orthogonal to the content vector $H(\mathbf{x})$ does not guarantee regularizing effect. To ensure it, we needed a subspace where $H(\mathbf{s})$ and $H(\hat{\mathbf{y}})$ were possibly orthogonal. If this subspace were not guaranteed, the repulsive loss that supposedly should have regularized only the content loss, could also regularize the attractive loss too. At the same time, the capability of VAE was lacking because we could not get clear outputs no matter how hard we push.

It turned out that VAE was known to have quality issues because it directly optimizes log-likelihood of data, which is difficult [180, 181] VAE alone without methods like autoregressive[182, 183, 184] or normalizing flow[119, 185, 186]. Both methods showed some progress but not quite as much as we liked. In the end, we moved away from VAE and from cosine similarity as well because I wanted quality by having bigger latent spaces over interpretability. We believe that the disentanglement property is a hot topic to study [187, 188, 189], especially in self-supervised learning.

Feature or perception networks Needless to say, pre-trained feature networks that provide perceptions were the main component to realize NU-NET, and we wanted to better understand the effects of the receptive field. It was already reported that the neural style transfer model keeps different sizes of details depending on which layers to use in a pre-trained VGG [87]. However, there were more options available for pre-trained networks since it had been reported. We chose MobileNetV2[190] since it was lighter than VGG but still had deeper architecture. It had 7 feature blocks compared to 4 in VGG16 or VGG19.

However, as it turned out, having more blocks was not better and the quality was compromised. Practically, there was no difference after the 4th feature block and their magnitudes turned out to be smaller in general than those from the 1st to 3rd blocks. Furthermore, we noticed more visual artifacts than VGG. See figure 3.7.

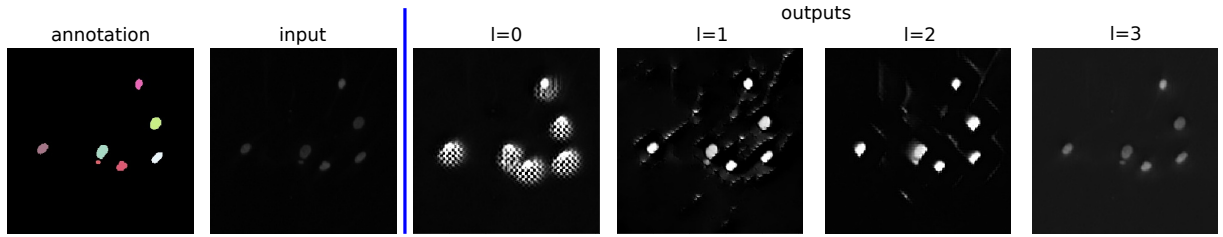


Figure 3.7: Study of feature blocks of pre-trained MobileNetV2[190] for the morphological loss. MobileNetV2 has 7 feature blocks in total. The first three blocks ($l = \{0, 1, 2\}$) were significant, while blocks after the 3rd one ($l = \{3, \dots\}$) had little impact and showed no visual difference from the one $l = 3$. It seemed that the first block did not provide enough information and the output had a checker pattern, which was frequently observed when the weight of morphological loss was pushed too hard. The second ($l = 1$) and the third ($l = 2$) ones enhanced structures on background as well.

This was also when we realized that the magnitude of morphological losses from each block was higher, in general, when an image was *BW* (black foreground on white background), though it was obvious because targets of the morphological loss were all *WB*.

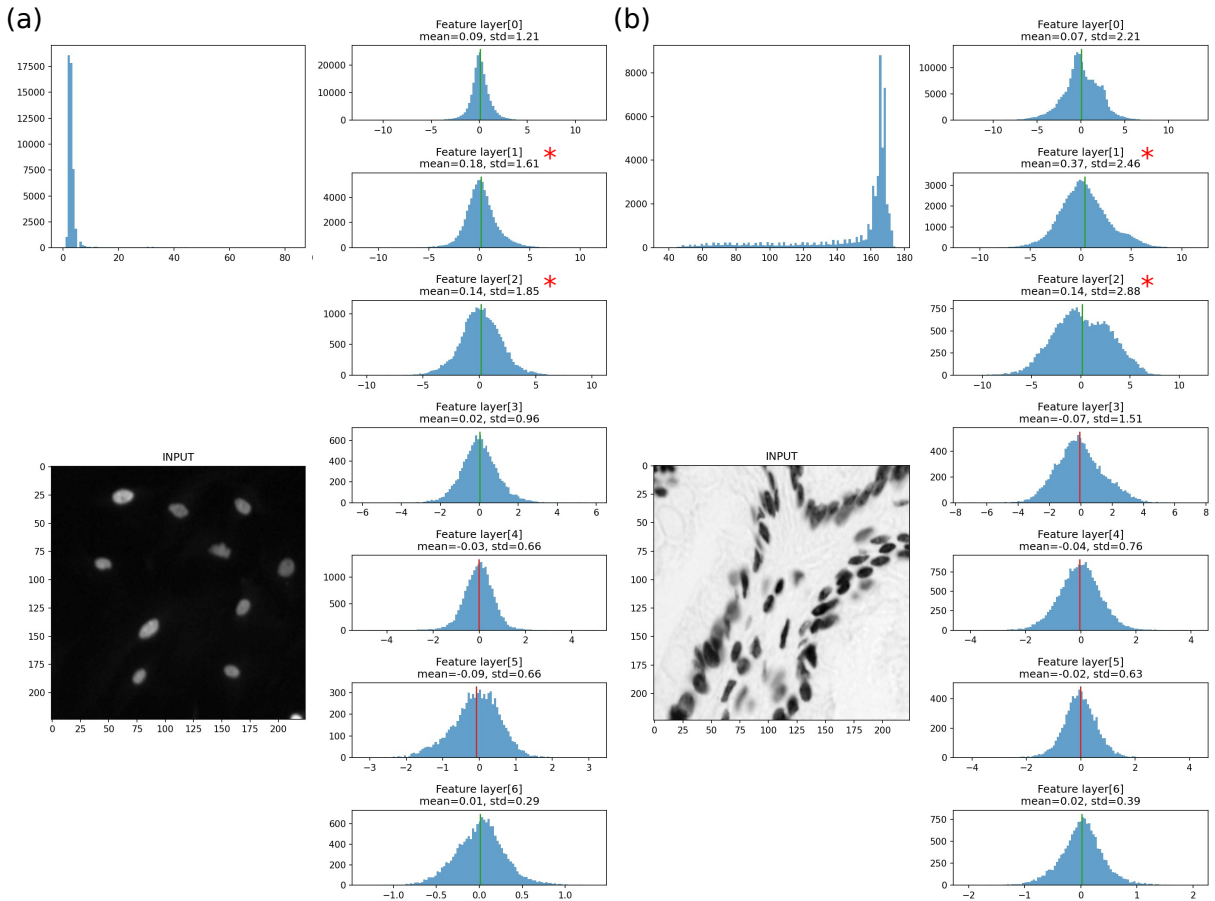


Figure 3.8: Statistics of feature maps from each feature block in MobileNetV2[190], given *WB* and *BW* inputs. In each block, the top left figure represents a histogram of the input image, and the right figure has 7 rows of value distributions, corresponding to 7 feature blocks in MobileNetV2. Depending on whether an input was *WB* or *BW*, statistics differed a lot, especially at blocks tagged with red asterisks (*). (a) Input was *WB*. (b) Input was *BW*.

3 RESULTS

Overall, evaluation of NU-net is complicated by the fact that our aim here is different from segmentation or restoration models, and no clear benchmark and metrics exists that we could use ‘off the shelf’. On top of the results presented here, some additional numerical experiments and evaluation, on real life workflow aiming at showing the practical usefulness of NU-net, are available in the preprint, which is included in appendix D.

Evaluation As a metric to evaluate NU-Net, the contrast ratio (CR) was defined to quantify contrast change and to compare NU-NET to multiple filters, quantitatively. CR is a fraction of a mean value of foreground pixels over that of background pixels, as in equation 3.6.

$$CR = \frac{\mu_{fg}}{\mu_{bg}} \quad (3.6)$$

Post-processing For all the results presented from here on, we applied percentile-based contrast stretching as a post-processing step. There were two reasons. **First**, we compared results of NU-NET to those of methods that enhance contrast, namely median filter, Gaussian filter, logarithmic lookup table (LUT) mapping, global histogram equalization, local histogram equalization, and difference of Gaussians (DoG) filter. In fact, many of these do not stretch the contrast by themselves. Thus, they need a contrast-stretching strategy afterwards. We decided to apply the same contrast stretching strategy to all the method including NU-NET, which was a percentile-based contrast stretching. **Second**, the direct outputs of NU-NET tend to have low values and look dark. Even though the morphological loss enforces binary values and NU-NET to change value ranges, the values are inclined towards 0. This is because targets for the loss have mostly low density, meaning they have far more 0 pixels than 255 pixels. However, thanks to continuous property and floating point computation of neural networks, the output keeps the precision even within a seemingly small value range. For these reasons, we used 99th-percentile (99%) for the upper bound and 2nd-percentile (2%) for the lower bound to stretch value ranges of all outputs except those from NU-NET. For NU-NET, we only applied 99% upper bound since its outputs were already inclined towards 0 and many times meaningless.

3.1 PERCEIVING STYLES

Gram matrix from a deep CNN was already proven effective to represent complicated styles of paintings as well as photographic images [87, 86, 88, 130]. However, what mattered to NU-NET was to know whether perceptions of a pre-trained CNN, trained with mainly photographic images from ImageNet[62], can distinguish blob-mask targets from raw microscopy images. The targets are basically binary mask images where foreground pixels have 255 values and background pixels have 0 values, with one important constraint that they should have consistent morphological features.

We chose a pre-trained VGG19 and DSB2018, which is representative dataset for its diversity and high quality segmentation annotations. We inferred images and masks to get feature vectors from predefined feature blocks of VGG19. Then, their Gram matrices were computed as features for UMAP[144] and for plotting the results in figure 3.9. The figure

shows that all the feature blocks from a pre-trained VGG19 were able to differentiate masks from images using style features, which proved potential of the morphological loss.

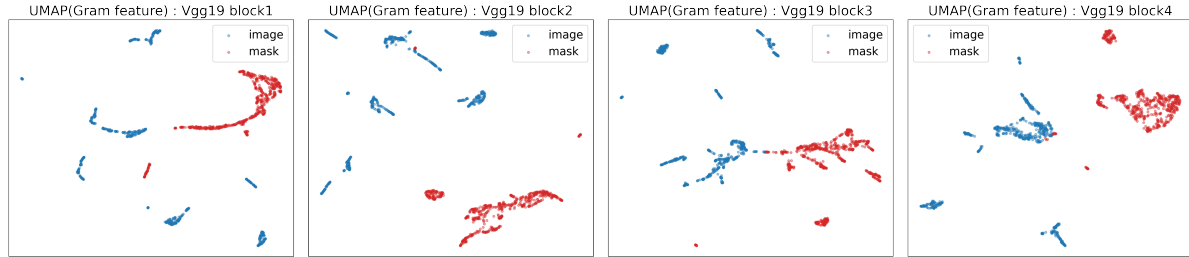


Figure 3.9: 2D UMAP results of Gram matrices from a pre-trained VGG19. 670 data points for each image and mask class are presented, from the DSB2018 training set. Well separated clusters tell that VGG19 perceives and can distinguish them in two groups. In turn, it implies that the morphological loss could generalize the blob-mask style. Actual layers of VGG19 that correspond to feature blocks are shown in table 3.2.

3.2 LOSS CURVES

Two main losses, the content loss and the morphological loss, are supposed to compete, because the content loss, being a generative loss, hinders outputs to become blob-mask style *as per* the morphological loss, and *vice versa*. Thus, balancing two losses was crucial, meaning their magnitudes should match one to the other. Hence, there is a balancing process before the actual training, where NU-NET goes through a few iterations of dry running to estimate magnitudes of the losses and match them. During the training, a desired loss behavior is to have a constantly decreasing morphological loss and a content loss increasing in the middle of training. The rationale is that, in the beginning, we want *NU-Net* to learn images themselves through the content loss, and to learn how to enhance blobs according to the morphological loss afterwards. As a result, the ideal loss curves looks as in figure 3.10. In general, weight coefficient w (in equation 3.3) in range of 5 to 10 yielded good results.

3.3 CONTRAST ENHANCEMENT

We compared results of NU-NET to existing filtering and contrast enhancement algorithms in terms of the contrast gain. Those were median filter (med), Gaussian filter (gaus), logarithmic lookup table (LUT) mapping, global histogram equalization (eq-glob), local histogram equalization (eq-loc), and difference of Gaussians (DoG) filter. Logarithmic lookup table mapping (LUT) is the simplest method among them, which makes use of *log* curve to remap pixel values and to stretch value intervals. Median and Gaussian filters are both denoising filters using parameterized kernels. While denoising does not have a direct impact on contrast, it could enhance contrast removing noisy background signals, combined with range stretching strategy. Histogram equalization methods are not intended to maximize fg/bg contrast, but to rather to see all the details equally.

The result can be found in figure 3.11. Contrast gain was significant once the algorithm had a notion of foreground and background. For examples, simple filtering methods like the median filter (med) and the Gaussian filter (gaus) made the distinction by blurring

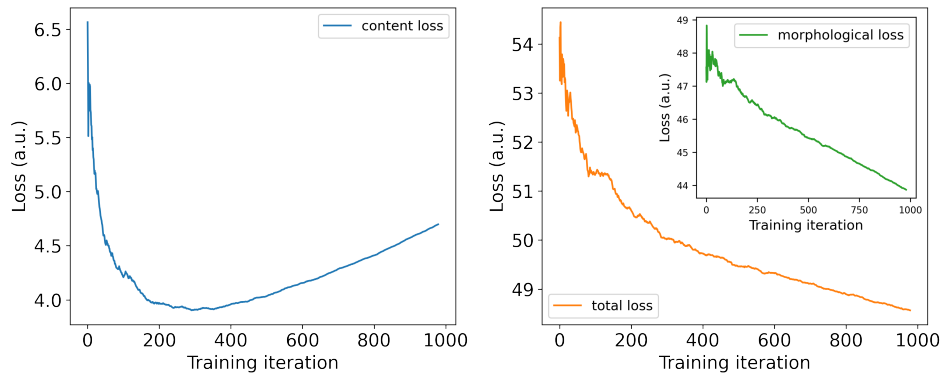


Figure 3.10: Ideal training curves. Total loss is a sum of the content loss and the morphological loss. The desired trajectory would be to have a constantly decreasing morphological loss and a content loss to increase after a decent drop. Note that the curves were from a real case but were smoothed to demonstrate the ideal forms.

off the noise and the background as well as the foreground. They accentuated cells, but the contrast was barely improved because they did not make clear distinction between foreground pixels and background pixels. Two histogram equalization methods also had the same issue in principle. Although they may be helpful to accentuated cells, the gain remained insignificant for the same reason.

Among the filters, the DoG filter showed the most comparable results to NU-NET because they are both aware of contents. Figure 3.12 shows three sample images of comparing NU-NET to DoG filter. DoG filter supposes that the objects of interest have a consistent look (values, sizes, *etc.*) and could enhance visibility of objects and remove noise at the same time, by subtracting two Gaussian filters with different kernel sizes. DoG is, in a way, aware of contents and could detect foreground from background. It is a popular object detection or segmentation tool for bioimage analysis and widely adopted for its easy implementation to many applications and software, as in ilastik[75], TrackMate[15], BigStitcher[7], *etc.*

While both DoG filter and NU-NET picked up objects, DoG suffered from blurring artifacts from its algorithmic nature, just as other rule-based algorithms suffer from their own pitfalls. Blurring merged objects together making it hard to recognize individual objects. Moreover, DoG accentuated other structures too in presence of noise. NU-NET, however, did not suffer from blurring, though it may show others visual artifacts, which we will discuss later. Additionally, NU-NET not just detected contents but also directly enhances contrast by pushing foreground pixels up towards 255 (maximum value in UINT8) and background pixels down towards 0 though the morphological loss, as shown in the histogram. Overall, NU-NET suppressed background texture as in (a, b, c), can handle a dense image (b), and can handle a noisy environment (c) as well.

Contrast gain Changes of contrast ratio after applying these filters are shown in table 3.3. LUT, median filter (med), and Gaussian filter (gaus) showed modest increases. These methods have none to little knowledge about target objects. In contrast, both DoG and NU-NET have significant knowledge about them, thus their contrast gains were huge. Note that a contrast gain largely depended on the type of images and its initial contrast.

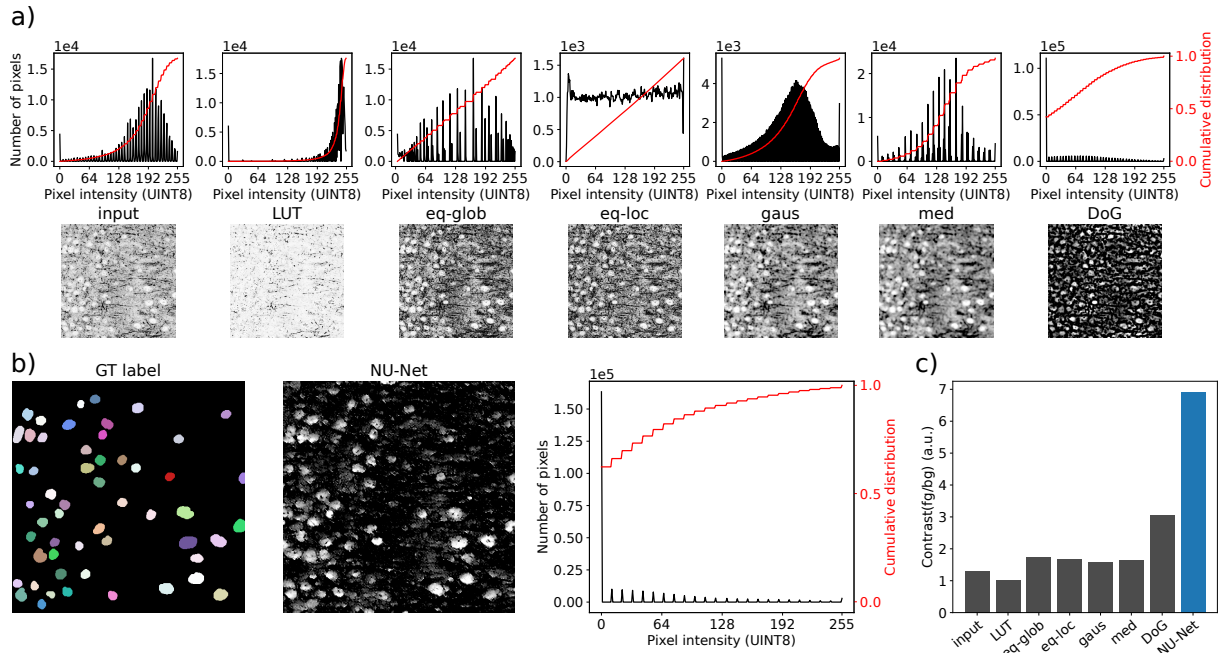


Figure 3.11: Comparing NU-NET to various contrast enhancement filters. They were median filter (med), Gaussian filter (gaus), logarithmic lookup table (LUT) mapping, global histogram equalization (eq-glob), local histogram equalization (eq-loc), and difference of Gaussians (DoG) filter. **a)** It shows the input image and the filtered images as well as their histograms. The red line shows the cumulative distribution of pixels values and gives a rough idea of image contrast. **b)** It shows the ground truth (GT) mask and NU-NET’s result. Though, its cumulative distribution looks similar to that of DoG, the qualitative result is much cleaner than DoG. **c)** Quantified contrast ratio (CR) of foreground pixels over background pixels. Find the definition of CR in equation 3.6.

For instance, DSB2018 dataset shows a huge standard deviation since it comprises various sources of images, which means the initial contrasts have a high variance.

In general, CR of NU-NET was not always higher than DoG. It was usually the case that CR was lower when an input image had rather clear contrast, and NU-NET tended to under-segment foreground objects⁴. However, the qualitative outputs were visually better, as shown in figure 3.18.

3.4 CONTROLLING FILTERING MAGNITUDE

Compared to most segmentation methods, one big advantage of NU-NET as a filter is the ability to control the filtering magnitude. It can be achieved by adjusting the ratio between the morphological loss and the content loss. In practice, it is as simple as to manipulate w in equation 3.3. Essentially, the larger the ratio w is, the more binarized the outputs become. An example of varying w is shown in figure 3.14.

As w decreases in figure 3.10, the content loss curves do not follow ideal curves anymore, which means that the resulted model would become more like an autoencoder. Yet, due

⁴ Under-segmentation means segmenting more than needed with less confidence, covering more area. In contrast, over-segmentation means segmenting too much with high confidence, not fully covering the region of interest.

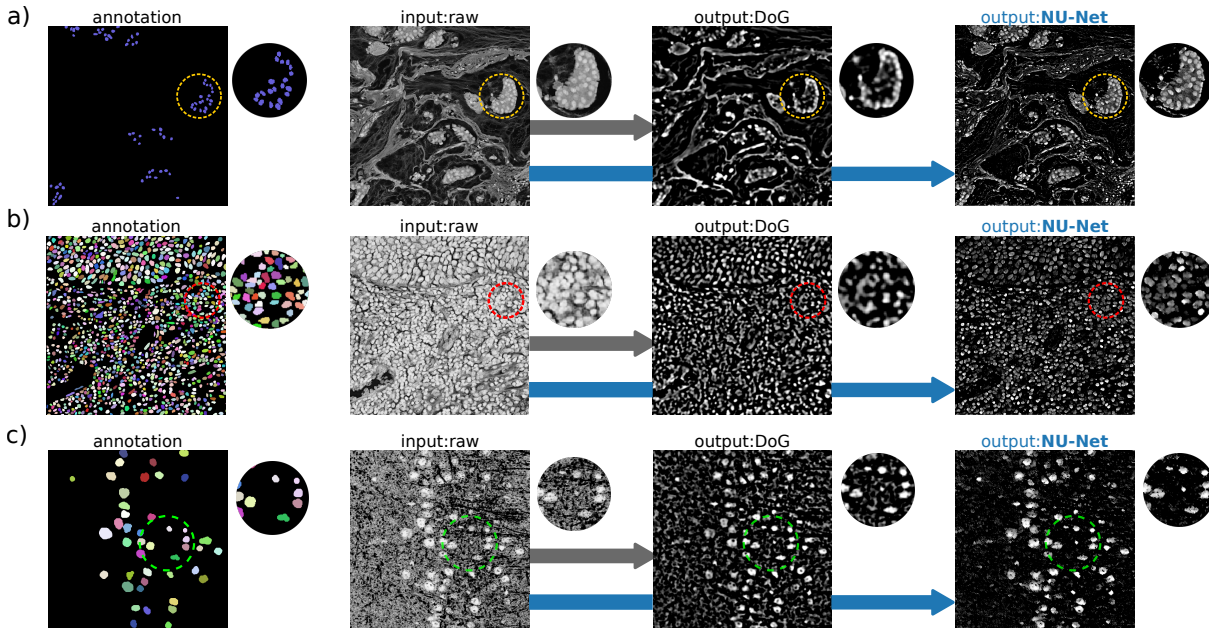


Figure 3.12: Three examples to compare NU-NET to DoG filter. **a)** Clustered cell image with highly textured background. Note that it has no full mask annotation available. **b)** Dense cell image. **c)** Relatively low contrast image with a lot of noise. Overall, DoG blurred images and failed to keep details in comparison to NU-NET. Sources: a) DigitPath [66], b) ComPath [67], c) LOB-THG, in-house dataset. Find details of each dataset in appendix B.

to presence of the morphological loss, resulted NU-NETS could still filter images. In the perspective of histogram of images, filtered output roughly keeps the original distribution and has large portion of 0 pixels which correspond to background.

3.5 SIDE EFFECT: LONG OBJECTS

NU-NET can also enhance long objects, as long as they have a comparable size and a consistent thickness. It could be useful, although it was not expected nor desired behavior since they are not in blob-mask style. But, in fact, nothing prevents NU-NET to do so because long objects can be considered as continuous blobs. This behavior is more pronounced when NU-NET was used in 3D volume via a trick. The trick is to process

filter	Dataset									
	BBBC008	DSB2018	BBBC006	S-BSST265	BBBC039	BBBC020	TNBC	LOB-THG	ComPath	FRUNet
raw	6.0±1.7	7.1±4.8	4.6±1.0	4.3±2.4	3.9±0.6	15.4±6.1	2.1±0.8	1.7±0.3	1.4±0.1	1.0±0.0
eq-glob	1.6±0.1	1.9±0.3	1.8±0.2	2.1±0.3	1.6±0.2	1.7±0.1	1.9±0.1	1.8±0.0	1.8±0.2	0.9±0.2
eq-loc	1.4±0.0	1.7±0.3	1.7±0.2	1.9±0.3	1.5±0.2	1.6±0.1	1.9±0.2	1.7±0.0	1.8±0.2	1.1±0.2
LUT	25.5±1.7	16.0±14.6	18.9±7.4	10.4±2.9	16.8±4.7	15.4±6.2	3.8±3.3	1.7±0.3	1.5±0.1	0.9±0.2
gaus	23.7±6.5	18.6±16.7	16.3±25.9	8.7±2.8	10.4±3.7	14.7±5.8	4.0±3.4	1.9±0.3	1.6±0.1	0.8±0.2
med	27.6±6.6	20.9±20.9	24.2±29.6	10.2±3.0	16.3±5.0	15.2±6.2	3.6±3.3	1.8±0.2	1.6±0.2	0.8±0.3
DoG	232.6±137.9	156.7±148.3	109.0±103.0	94.0±77.2	77.2±68.6	77.5±24.1	11.5±4.3	3.4±0.7	4.4±1.2	5.2±4.8
NU-NET	402.6±202.0	221.7±361.6	78.4±101.5	80.7±44.1	59.1±77.6	63.2±19.7	18.3±7.1	8.0±1.7	4.5±0.9	3.2±1.5

Table 3.3: Averaged contrast ratio (CR), defined in equation 3.6, over test images from each dataset. Plus-minus sign (\pm) indicates a standard deviation. Values after \pm refer to their standard deviations. We have eq-glob (global histogram equalization), eq-loc (local histogram equalization), LUT (logarithmic lookup table mapping), gaus (Gaussian filter), med (median filter), DoG (difference of Gaussians), and finally NU-NET. Check the actual outputs too in figure 3.18

each z-stack individually and stack them later since NU-NET does not support multidimensional images. Since the intersections of a long object could be perceived as blobs in 2D, they form a long object naturally when viewed in 3D. This trick yields its own issues but can produce relatively fine results. See figure 3.13

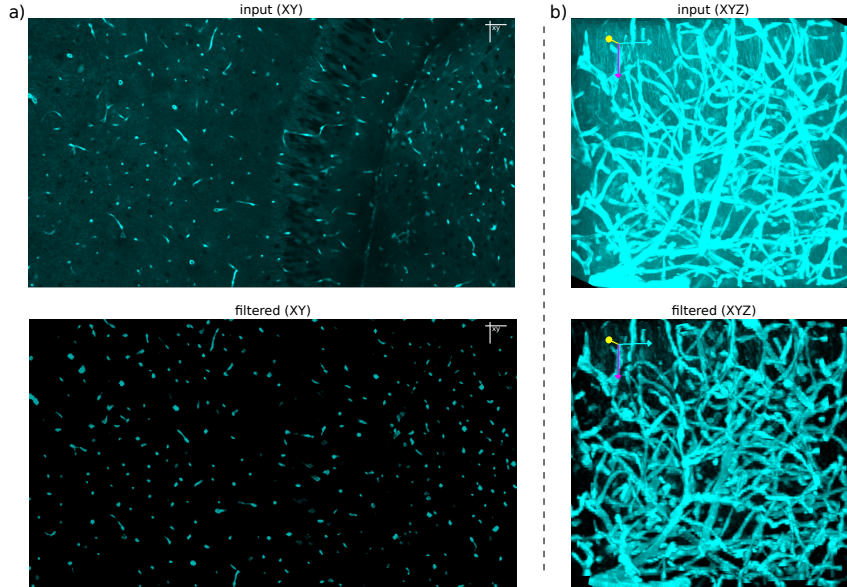


Figure 3.13: NU-NET filters long objects too because they can be considered as continuous blobs. This behavior is more pronounced in 3D volume though the mechanism is different, because intersections of long objects are blobs in 2D. The image is an area of dense blood vessels in a mouse hippocampus, acquired by ChromS.

3.6 APPLICATION: NAPARI PLUGIN

NU-Net is easily accessible and easy to use via Napari viewer[37], a fast-growing image viewer. In fact, NU-Net requires a lot of system memory to train mainly due to computing perceptual losses, and its architecture could potentially become bigger and deeper. However, the current architecture is small to deploy and fast to run because of relatively small amount of training data, for the moment. NU-Net is made available as a plugin of Napari. The plugin comes with a simple GUI and 5 pre-trained NU-Net with varying w value from 5.0 to 10.0, whose size is about 33 MB in total (~ 6.5 MB per model). Users may adjust filtering magnitude via w value and run the process either on CPU or GPU. An example is demonstrated in figure 3.15.

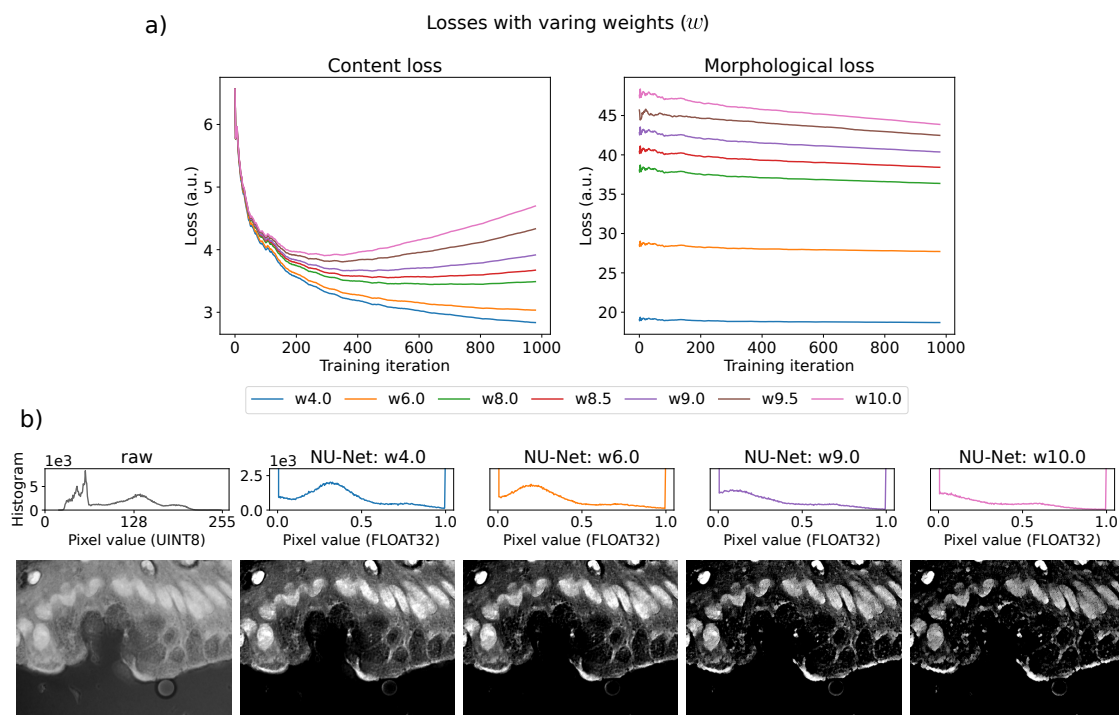


Figure 3.14: Controlling filtering magnitude by manipulating the loss coefficient w . **a)** Loss curves with varying w values. Each morphological loss looks barely changing, but that is because w value is a multiplier of the morphological loss. They kept decreasing individually. **b)** An example of applying NU-NETS trained with different magnitudes of weight (w). Note that w is a ratio of the morphological weight to the content weight in equation 3.3. Note that, in histograms of (b), y-axes share the same range for all NU-NET outputs. Image source: DSB2018 dataset.

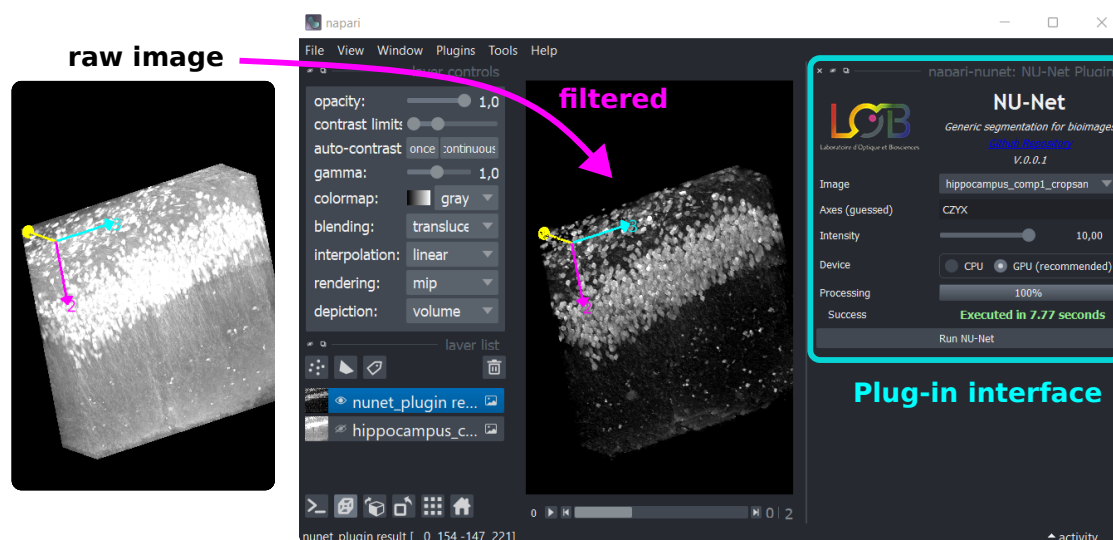


Figure 3.15: Snapshot of NU-NET plugin for Napari viewer[37]. Users may select one of 5 pre-trained models with different filtering intensities. It automatically detects the input dimension and process it either on CPU or on GPU. This volume was cropped from a whole mouse hippocampus image acquired by ChromS[2]. The volume had a size of $130 \times 400 \times 400$ (depth, height, width), consumed 1.3 GB of GPU memory, and took around 8 seconds on a laptop GPU. Note that NU-NET does not support 3D volume inputs natively, meaning that it processes each z-plane one by one. Also, the processing time depends on hardware. On a workstation, it took only 3 second (~ 23 ms/plane), while it took around 20 seconds on CPU (~ 150 ms/plane).

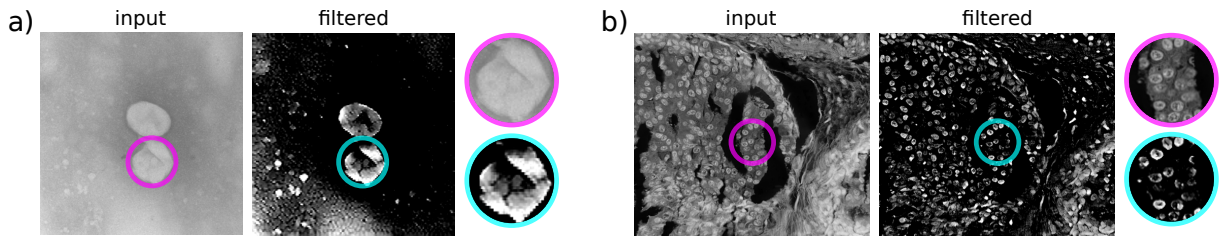


Figure 3.16: Visual artifacts and limitations of NU-NET. **a)** It makes fragments out of a big object when multiple sizes exist in an input and sharpens their edges. **b)** When holes are present, objects may not be detected and become false negatives. Sources: a) FRUNet, b) USCB

3.7 ARTIFACTS AND LIMITATIONS

NU-NET may show some unexpected behaviors and visual artifacts. Some need more study, but some turned out to be intrinsic. Accompany figure 3.16.

Sharp edges and Fragments When there were multiple blob-shaped objects in an input image with big difference in their sizes, NU-NET tended to filter bigger objects into small fragments. Example (a) had two big cells in the center, and NU-NET did not consider them as single objects. Regardless which one is correct, it is true that NU-NET struggles in the presence of objects with different sizes. In addition, in the process, each fragment had sharpened edges, even though the edges were not pronounced in the input image. This behavior is not entirely alien because NU-NET attempts to filter out a certain size of objects individually according to the selected layer for the content loss. Since only a feature map was selected, information during training was limited and some features were supposed to be lost.

Holes When objects of interest have holes, NU-NET tends to fail to fill hollow holes or to even pick them up. NU-NET should have learned that the output should look like the blob-mask style, which does not have any objects with holes, and a style like the one in (b) should not have happened. But it happens rather frequently, and it even contributes to many false negatives.

3.8 MORE FIGURES

Supplementary figures.

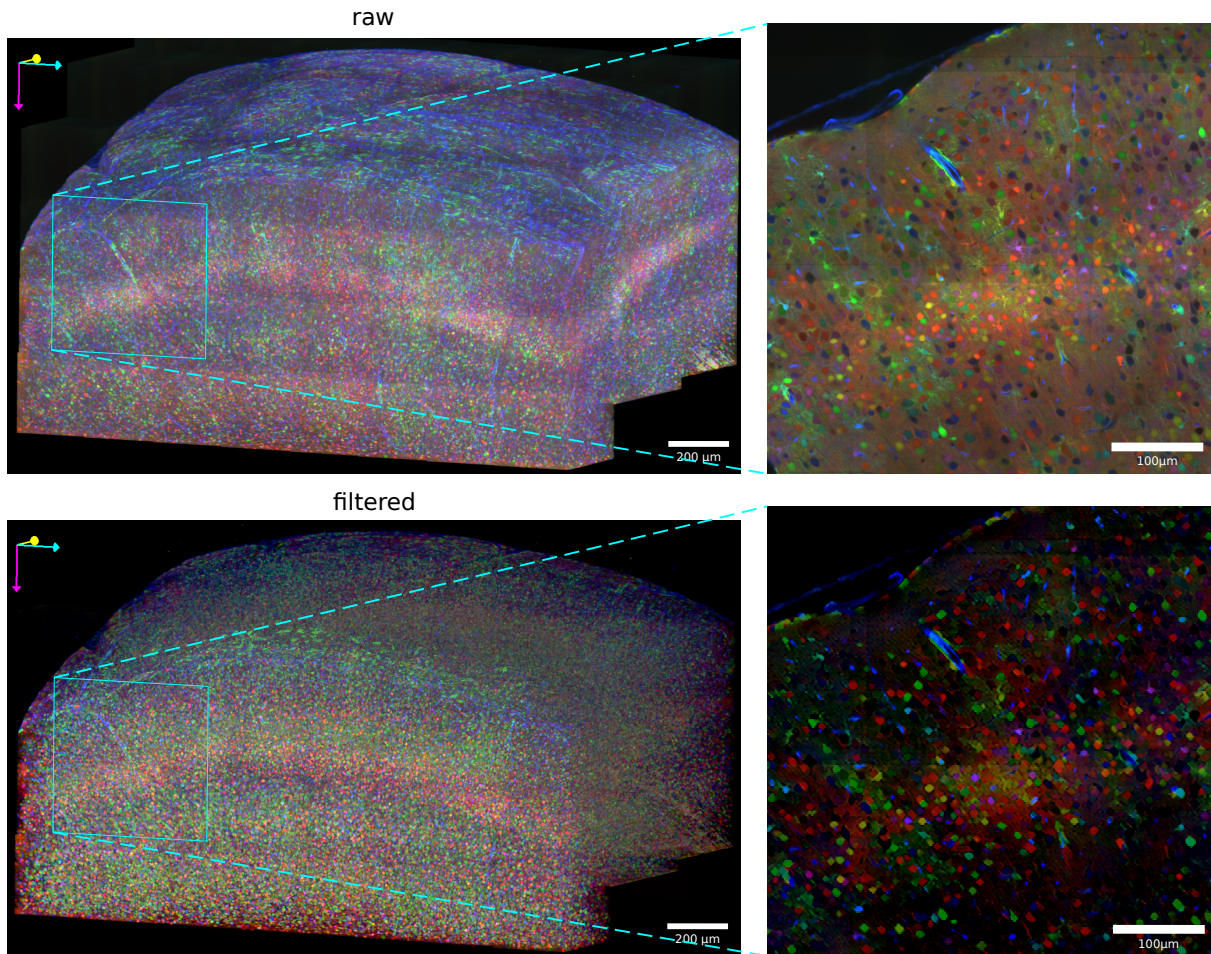


Figure 3.17: Application of NU-NET to a whole volume image with multichannels. Although NU-NET is a 2D model and only works on single-channel images, it is possible to deal with a 3D volume with colors. The trick is to process it across z-stacks and channels, though the result has some issues because of the trick. Additionally, this example was tiled into small tiles to be filtered since each plane were too large to process it at once. This image is the same as 1.1, which is a whole mouse brain image with Brainbow[1] staining technique, acquired by ChromS[2] at LOB.

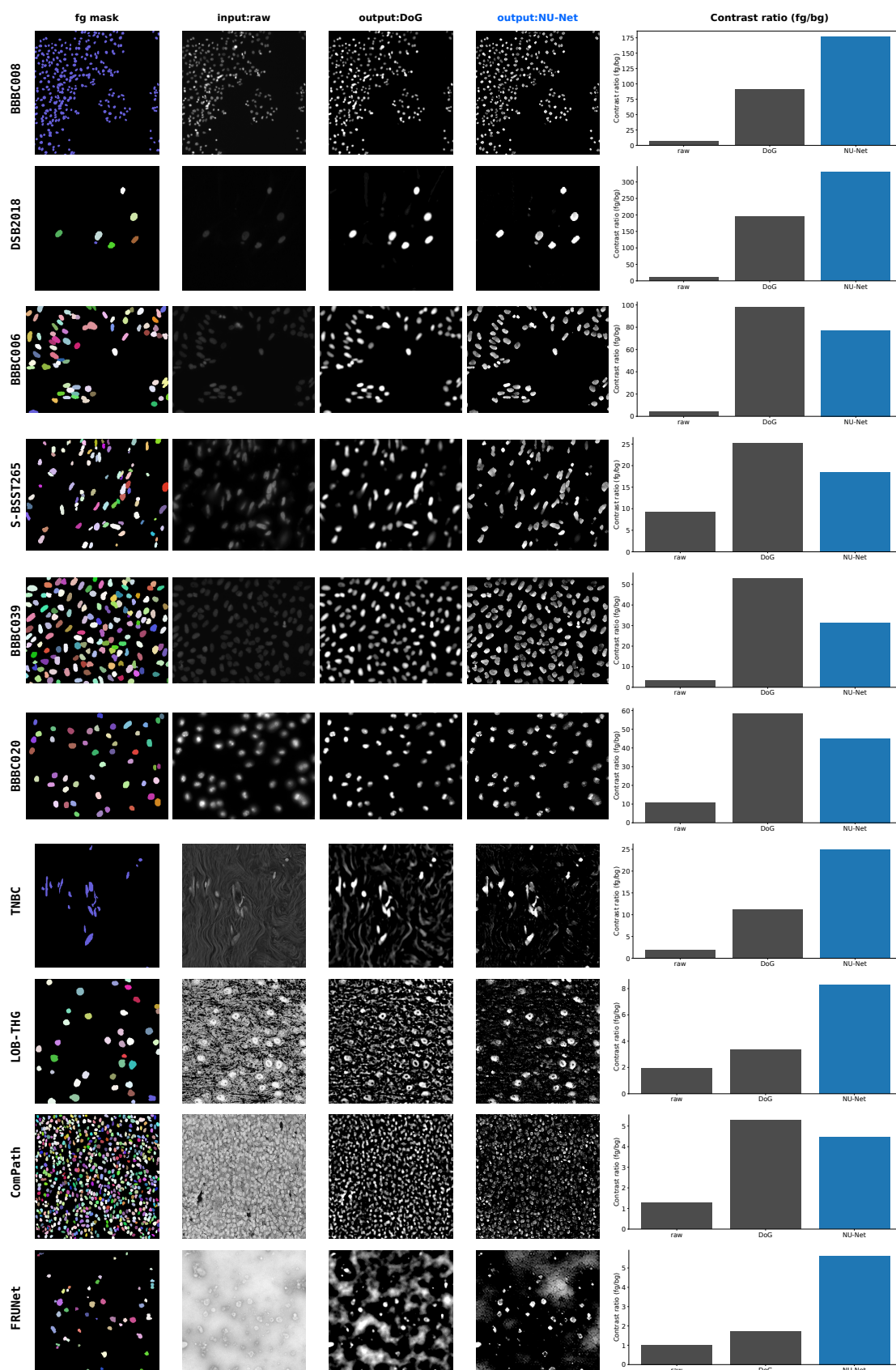


Figure 3.18: Comparing output images and contrast ratios of NU-NET to those of DoG filter. The presented datasets were chosen because they had segmentation masks, so that the contrast ratio (CR) could be calculated. For some datasets, a contrast gain of NU-NET was smaller than that of DoG filter. However, NU-NET’s outputs were visually crisp and easy to identify objects in foreground, in general. This figure is coupled with table 3.3 that showed averaged contrast gains for each dataset.

4 DISCUSSIONS AND PERSPECTIVES

Segmentation after filtering It may seem helpful for a segmentation algorithm to use filtered images instead of raw ones. While it could be true for some combinations of filters and segmentation algorithms, this was a counter-intuitive thought. Filtering may be helpful by filtering out noise or insignificant details. But NU-NET is not a denoising filter, and its role is closer to that of a segmentation algorithm. In consequence, it inevitably generates some false negatives. It is like applying a segmentation algorithm over and over and having nothing left in the end because the algorithm will not filter in what was already filtered out. This is why, for example, using STARDIST after filtering with NU-NET, does not always yield better results. Find samples in figure 3.19 and table 3.4.

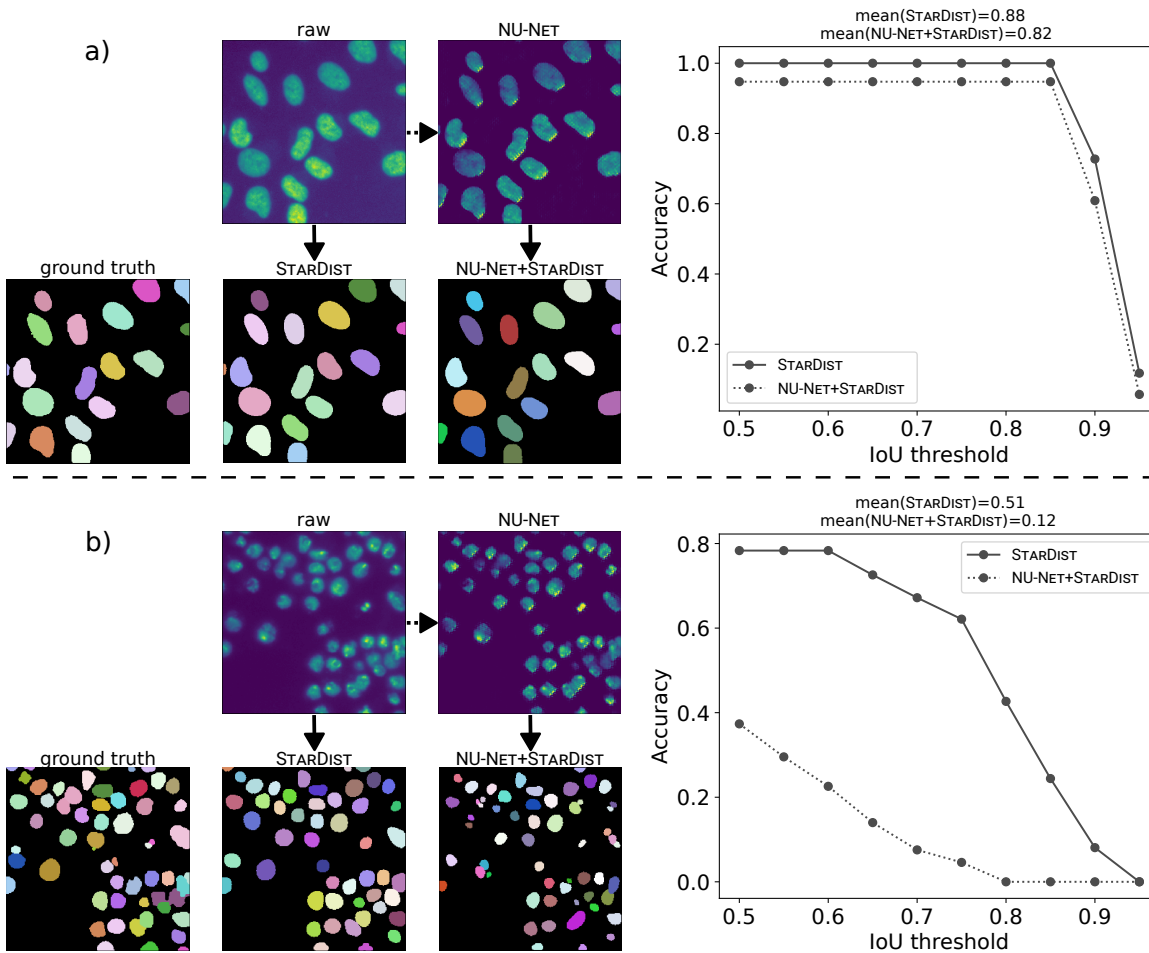


Figure 3.19: Applying a pre-trained StarDist segmentation model on images filtered by NU-NET. **a)** An easy example whose contrast is strong and clear. The output of NU-NET is not bad. But it missed one small object at the left edge, it resulted in lower accuracy. **b)** A hard example where foreground objects have more texture. NU-NET failed to pick up some and divided some into fragments because of heavy texture. Also, overall segmented area got smaller because objects were technically segmented twice. Find the definition of the accuracy metric and IoU (intersection over union) in appendix C. Note that NU-NET that used here was developed a while ago and slightly inferior to the current version. Images were sourced from the testing subset of StarDist dataset, which is a subset of DSB2018, containing only grayscale images.

	STAR _{DIST}	NU-NET+STAR _{DIST}
Accuracy@IoU=0.5	0.85	0.76

Table 3.4: Averaged accuracy at IoU=0.5 over images segmented by a pre-trained StarDist model (STAR_{DIST} column) and those that filtered by NU-NET and segmented by the same StarDist model right after (NU-NET+STAR_{DIST} column). Find the definition of the accuracy metric and IoU (intersection over union) in appendix C. The testing subset was **StarDist** dataset, which is a subset of **DSB2018**, containing only grayscale images.

In details, NU-NET preferred blobs since it was trained so. Some elongated objects were sometimes divided into small blob fragments. Also, it is certain that NU-NET changed the value distribution and might be possible that it resulted in a slight domain shift in a way that the image was no longer familiar to a pre-trained model.

It might be worth trying segmentation after filtering on images that are not optimized for the pre-trained model as a future work.

Perception network Another perspective of the feature network is its data domain. To this date, there is no generic vision model for the biomedical domain. Although available pre-trained vision models are supposed to be generic, they are generic mostly in natural images. Given that, one apparent question is how much domain overlap exists between natural images and biomedical images in deep neural networks. It is something worth investigating and comparing once a generic biomedical vision model appears. In the meantime, what seems promising is to update existing models gradually with momentum[111] so that they can become slowly adapted to a new domain.

Transferability Transfer learning is the main purpose of building generic models. A typical way to measure transferability, at least in vision models, is to freeze the generic model and add a supervised linear classifier to test it with ImageNet classification task[62] (the generic model should not be trained with ImageNet). The criteria are how efficiently it adapts itself and whether it could even outperform supervised models.

We followed the aforementioned common protocol for which we pre-trained NU-NET, froze the weights, and trained it a few shots of supervised learning for a semantic segmentation task. The difference from the common protocol was that I did not have to replace the head because both pretext and downstream tasks were FCN (fully convolutional network)[172]. What we found was that it certainly outperformed one trained only with supervision, given the same number of training iterations. See F-1 scores in table 3.5 and loss curves in figure 3.20.

However, the gain was too small for the effort, and the starting loss did move much away from the one with supervision. It meant that pre-training was not helpful enough to be useful in general. It is known that for self-supervised learning to be useful as a pretext task, it needs a lot of data [91, 103, 84], probably orders of magnitude more compared to what we had for NU-Net. The issue always comes back to data. As you may have noticed, only 3 datasets were used in this experiment. It would be worth investigating again the transferability of NU-NET since a fairly large **TissueNet** dataset[43] had become available recently.

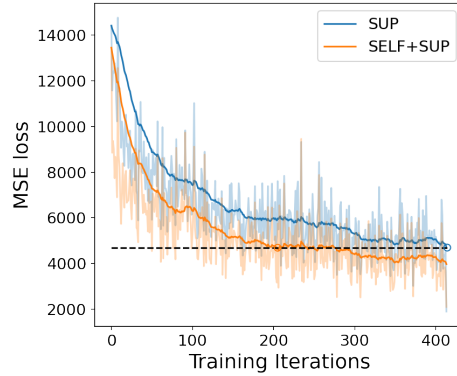


Figure 3.20: Loss curves of a supervised model (SUP) and a combination of self-supervision and supervision (SELF+SUP). More precisely, SELF+SUP model was trained on top of a pre-trained NU-NET. Note that curves were smoothed with an exponential moving average algorithm.

Dataset	Precision				Recall				F-1 score			
	Otsu's	SELF	SUP	SELF+SUP	Otsu's	SELF	SUP	SELF+SUP	Otsu's	SELF	SUP	SELF+SUP
DSB2018	0.90	0.91	0.88	0.88	0.70	0.54	0.80	0.83	0.76	0.65	0.82	0.84
TNBC	0.56	0.52	0.53	0.62	0.84	0.57	0.84	0.83	0.51	0.48	0.59	0.66
ComPath	0.28	0.70	0.74	0.78	1.00	0.50	0.79	0.79	0.43	0.58	0.76	0.78

Table 3.5: F-1 score comparison of Otsu's thresholding method and three NU-NETS. SELF is one trained only with self-supervision, SUP is one trained only with supervision, and SELF+SUP is one trained with a combination of self-supervision and supervision. **Bold** numbers indicate the highest score among four models in each metric for each dataset. Note that metrics are all pixel metrics.

Versatility and model capacity NU-NET was trained with 25 datasets all from different sources and expected to be versatile in that sense. But the issue is that not all datasets have annotations to test the performance against. Also, versatility is something not easy to define across different datasets for many factors. We left designing a proper metric to quantify the degree of versatility for a future work.

For capacity as well as versatility, it is important to consider bigger and deeper neural network architectures once data grows bigger. So far, we have intentionally stuck to relatively small architecture, which is tiny compared to popular deep models. For instance, U-NET[20] architecture is slightly bigger than the current one by default, and there have been a few advances in the design choice[33, 191]. Also, U-NET will be easy to compare and transfer since it has become a *de-facto* standard when it comes to segmentation tasks in bioimages. For a comparison, while the current architecture comprised of around 7 millions parameters, the base U-NET has roughly 130 millions ones. As the amount of data grows, the model would require bigger and deeper architectures to embrace it all.

Synthetic data Simulating and utilizing synthetic data is a popular way to augment data. The way we tried was to randomly generate blobs on various textures from KTH-TIPS database[124]. The idea was soon dropped because its data domain was vastly different from biomedical images. More elaborate data generation approach such as mentioned NucleAIzer[40] or SaliencyNet[166] may be beneficial to multiply numbers of samples and mitigate imbalance among datasets.

Training stability The loss of NU-NET is not typical because it is a multitask learning[192] and two parts of the loss compete each other. Essentially, it attributes to instability in training process. It is rather ambiguous what the best ratio would be to balance them or when to stop training. It took me a fair amount of time to figure out good ratios and ways to normalize components that contribute to the loss during training, so that they do not change depending on hyperparameters, such as a batch size, an input size, etc. For the moment, an empirical approach seems to be most viable.

Scheduling the learning rate has been already studied extensively[193, 194, 195]. It can be useful NU-NET but for NU-NET, scheduling coefficients of two losses would be more dire and effective for stabilizing the training process. The rough idea is to weigh the content loss more in the initial stage and to gradually increase contribution of the morphological loss as the training goes on. This way, NU-NET will be able to learn data itself first and gradually learn how to adapt outputs according to the morphological loss, just as much the ideal loss curve suggest in figure 3.10.

Colors The main target image of NU-NET was in-house ChroMS and Brainbow[1] data that we saw in figure 3.17. Its originality comes from the multicolor labeling technique based on a limited number of colors and their combinations, which is not common in bioimages at all. It is possible to train NU-NET keeping colors, but it would become complicated as soon as it needs more data that do not have colors or different combinations of colors. We have not found a reasonable way to embrace all cases including grayscale images.

So far, the best way is to convert color images into grayscale ones, which is more reasonable. Since colors in microscopy images do not follow conventional conversion equation for natural images, converting colors requires understanding of each color and a great deal of attention. This approach would also be closely linked to the particular biological aims of the experiments and thus would be done in close discussion with the experimentalists. In alternative, NU-NET may benefit from a certain degree of color augmentation techniques.

Morphology prior We tested NU-NET with a blob-shaped prior and made it a blob filter. In theory, the morphological loss should be able to learn and generalize other morphologies. We tried to make a cellpose prior that Cellpose uses [29], which encodes directional information or poses of cells both vertically and horizontally. This cellpose prior has a visually distinct style as shown in figure 1.8, which should make learning process easy. If it had worked, NU-NET could have been an instance segmentation model like Cellpose. That being said, it did not work out, and we have been investigating why since.

Slider The last perspective is about practicality and usability. A particular paper[142] suggested a way to control the degree of style transfer in inference time. Its approach is similar to that of conditional inference[161]. In both training and inference time, the model accepts an additional variable in a limited range. This variable could encode the loss ratio w in NU-NET and provide continuous filtering magnitudes, not like the current discrete numbers. In effect, users may use a slider to control this variable in inference time, and it will also eliminate the need of shipping multiple NU-NET models with different ratios. But this approach has a caveat that the additional variable eventually divides

capacity of a model. Consequently, the model must have big architectures than what NU-NET currently has. As a matter of fact, the training process turned out not optimal. Instead, we resorted to ship multiple models for the moment.

5 CONCLUSIONS

Supervised learning has intrinsic issues bound to curated datasets. In bioimages especially, the lack of large-scale datasets has been one of the biggest bottleneck when it comes to development of versatile and generic machine learning models. Self-supervised learning could address the high cost of annotating bioimages as well as the high specificity and fragmented datasets. In this chapter, we gathered 25 nucleus/cell datasets from various sources to compose a generic dataset. Despite that not every dataset has fully curated annotation, it was possible to build a versatile filter NU-NET based on a self-supervised learning scheme.

NU-NET is built on top of the perceptual loss and the neural style transfer framework. Its novelty is the morphological loss, which generalizes a morphological characteristic via a bank of consistent binary target images. The loss is dissimilar to any other popular self-supervised frameworks such as the contrastive learning or the self-attention. Although, for the sake of comparison, morphological loss used in NU-NET is closer to the contrastive learning than to the prevalent self-attention in the sense that the loss deliberately manipulates training losses. Yet, it differs from the contrastive learning because the morphological loss of NU-NET is a type of perceptual loss and does not suppose counter examples. More importantly, it is not only for a pretext task. NU-NET can be directly used as a filter. We defined a blob-mask style and made NU-NET able to enhance or filter nuclei and cells.

As a content-aware filter, NU-NET is comparable to the difference of Gaussians (DoG) filter. Though their contrast gains were alike, NU-NET does not suffer from the blurring artifact and delivers clear and overall superior outputs. We also demonstrated that it was easy to control a filtering magnitude and packaged it as a plugin for Napari viewer, so that users can easily use it. The plugin is light and fast and easily works on a laptop with no heavy system requirement.

CHAPTER 4

APPLICATIONS

Contents

1	Segmenting fibroblasts on human skins	119
2	Workflow integration	123
2.1	Napari plugin: Proofreading ChromS Brainbow	123
2.2	Napari plugin: BigAnnotator	125
3	Segmentation in ChromS’s pipeline	128

I joined other projects during my study and made technical contributions. Some of them involved machine learning methods, but others did not and were purely technical.

1 SEGMENTING FIBROBLASTS ON HUMAN SKINS

Ung, T. P. L., **Lim, S.**, Solinas, X., Mahou, P., Chessel, A., Marionnet, C., Bornschlöggl, T., Beaurepaire, E., Bernerd, F., Pena, A.-M., & Stringari, C. (2021). Simultaneous NAD(P)H and FAD fluorescence lifetime microscopy of long UVA-induced metabolic stress in reconstructed human skin. *Scientific Reports*, 11(1), 22171. <https://doi.org/10.1038/s41598-021-00126-8> [10].

The goal of this project was to show the value of the multicolor two-photon fluorescence lifetime microscopy (FLIM), by assessing damage to UV (ultraviolet) light (specifically UVA1; 340-400 nm, long UVA wave) on human skin and demonstrating effectiveness of a sunscreen. We used multicolor two-photon FLIM and reconstructed human skins[196] *in vitro*. UV exposure induces damage and metabolic changes in dermal fibroblasts. To monitor this process, label-free and non-invasive microscopy technique is important. We used nicotinamide adenine dinucleotide (NAD(P)H) and flavin adenine dinucleotide (FAD) as biomarkers for multicolor two-photon FLIM. For the single-cell analysis, it was required to implement an automated segmentation process, and we chose to use deep learning.

The direct output from the multicolor two-photon microscopy[197], accordingly, has two channels: one from NAD(P)H intensity the other from FAD intensity. In addition to

them, FLIM generates lifetime maps for each pixel and each response, based on the phasor analysis[198]. That gave us 2 images and 2 maps that correlated each other. All the four modes conveyed information to localize fibroblasts.

There was a need to automate the segmentation process and there were already some annotated labels. It looked a perfect candidate to apply the machine learning technique. A convolutional neural network (CNN), specifically U-NET, was a perfect fit for reasons. (i) It was well studied and performant. (ii) Objects of interest, fibroblasts, have rather free-forms unlike circular nuclei. It meant that models relying on shape priors were of no use, and U-NET did not require such things. (iii) CNN was natively capable of handling multi-modal data. It was imperative to use all the four modes jointly, and the CNN met the need.

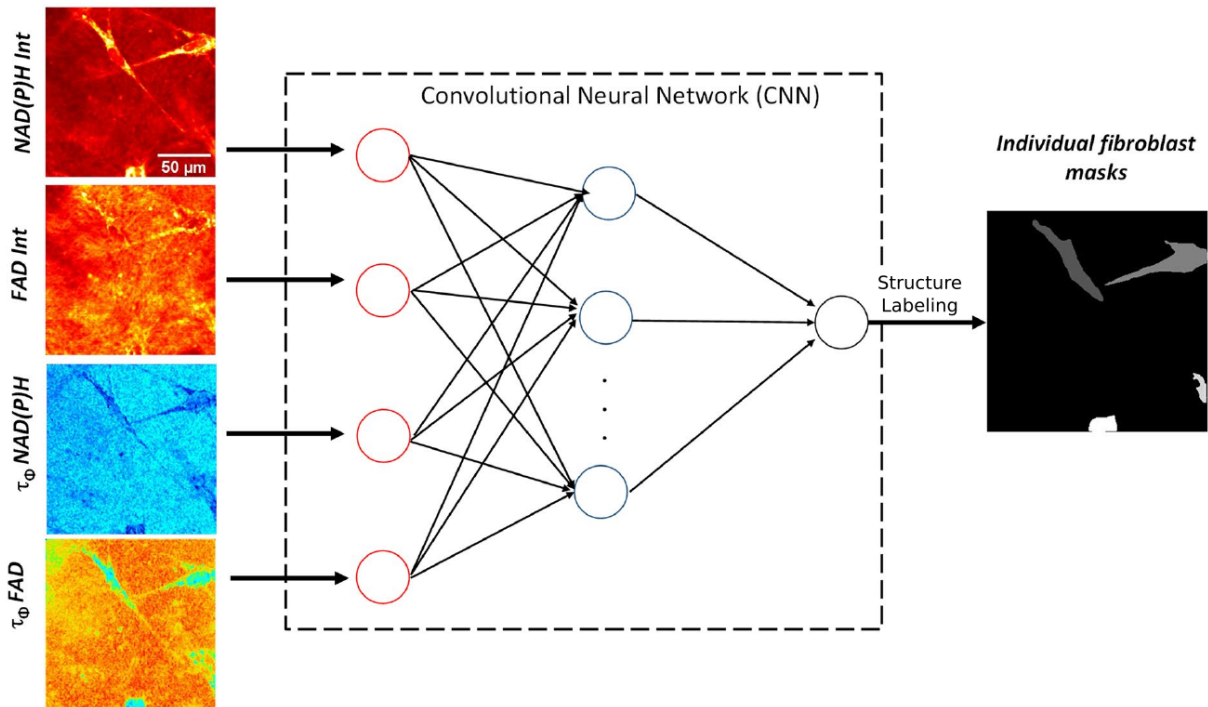


Figure 4.1: Workflow of single-fibroblast segmentation by deep learning with multi-modal images. Four inputs are given to the deep learning network: intensities and lifetimes of NAD(P)H and FAD. The output of the network provides a probability map. The map was thresholded above 0.5, then individual fibroblasts were localized by a simple structure-based labeling algorithm.

METHODS

U-NET had to be modified so that it could take in image arrays with four channels. We also added the batch normalization layer[176] after each convolutional layer, which lacked in the original paper. During training, a set of simple transformations was applied to augment images. The output of the model was a probability map, which was thresholded at 0.5 to get a binary segmentation map. Then, removed too small objects based on the smallest size of fibroblasts from annotations. Lastly, we processed it with a simple structure-based labeling technique to separate individual fibroblasts, which found a connected label via its neighboring 8 pixels. It was possible because fibroblasts were sparse

enough and rarely overlapped.

CHALLENGES

Training a deep learning model was not difficult, but challenges appeared in other parts. First one was the data organization. As mentioned, it throughout chapter 2, managing and cleaning data is the very first step of any computational analysis and often underrated. The project spanned for more than 2 years and the image acquisition occurred several times in different times and conditions, depending on the availability of the reconstructed human skins and the FLIM microscope. The quality of image was constant and was not an issue. The challenge was to organize and arrange files. The specimens were treated in different conditions: control group with or without a sunscreen and UVA exposure group with or without a sunscreen. Also, there were subgroups within the UVA exposure group depending on the exposure duration and power, not to mention four image modes. All these variables could be identified via folder names and file names, but they were not consistent across acquisition dates. So, sorting data was the first challenge.

Another challenge was more critical. We found that manual annotations were not precise. The edges of annotation were presumptuous and rather simple polygons, not aligned well with the actual boundary. See examples in figure 4.2. It was the case that localizing and annotating fibroblasts was not easy even for experts. We assessed value changes around boundaries of the manual annotation and the prediction, assuming that gradients around boundaries should be larger when the segmentation was more precise. As it turned out, indeed, that the prediction showed larger gradients in value around boundaries, confirming our assumption. To mitigate errors in manual segmentation, we closely monitored this statistics as well as validation losses not to overfit the model. Additionally, we post-processed the output with erosion by one or two pixels.

CONCLUSION AND PERSPECTIVES

This project already had a clear goal and enough data to analysis. It just needed a bit of help of an automated segmentation process. The task was a perfect fit to apply a supervised machine learning method thanks to existing manual annotations. U-NET, as a deep CNN, served well because it could handle multi-modal data easily and effectively. The challenges were in the data organization due to many experimental variables and to the quality of manual annotations. I was able to address both challenges and contributed to the project by providing an automated segmentation process for the analysis.

What I learned about this project was that so many projects like this would be out there, which needed a simple automated process to get a momentum. Also, executing machine learning methods was a relatively easy process thanks to advances in computational tools and software. It was rather the data science part to manage data that was challenging.

Besides, I felt that deep learning was versatile and flexible to deal with highly specific types of data. The multicolor two-photon FLIM was not a typical microscope, and handling multi-modal image data would have been difficult if it were not for deep CNNs. Deep CNN showed performance exceeding the poor quality of manual annotations.

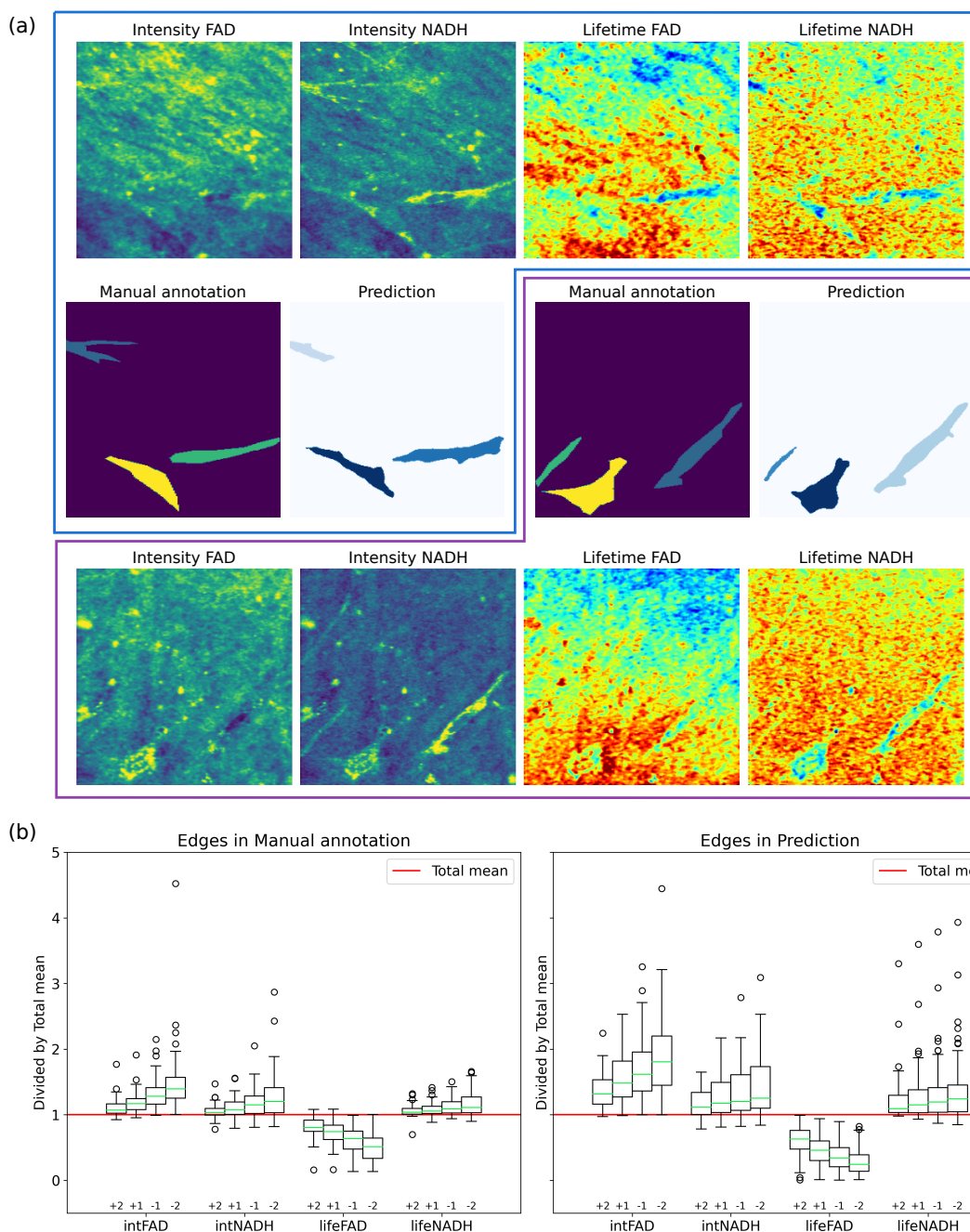


Figure 4.2: Manual annotation of fibroblasts had low quality. (a) Comparison of manual annotations and predictions. Manual annotations had simple polygon shapes likely due to lack of capability of an annotation tool, thus failed to localize fibroblasts precisely. Prediction was often more precise even though the model was trained with incomplete annotations. (b) This simple statistics gave an idea how precise the boundaries of manual annotations or predictions were. The total mean was the averaged pixel values belonging to fibroblasts. Individual data points were ratios of an averaged pixel value on boundary compared to the total mean. Note that the mean values differed in modes and annotation/prediction. +2, +1 means pixels whose distance were 2 and 1 pixels inwards from the boundary. -2, -1 means those outwards from the boundary. In general, signals were strongest at the center of each fibroblast and dampened down towards the boundary. Therefore, small ratio changes meant that the boundary of either a manual annotation or a prediction was far inside or far outside a fibroblast. Note that the FAD lifetime got shorter whereas the NAD(P)H lifetime and both intensities increased. Find details in the paper[10].

2 WORKFLOW INTEGRATION

Other than machine learning, there are a lot of places where computer technology can be helpful to improve workflow of experiments. I developed two plugins for Napari viewer[37]: one to improve the proofreading process of an automated prediction, the other to facilitate the manual annotation process for big data.

2.1 NAPARI PLUGIN: PROOFREADING CHROMS BRAINBOW

This plugin is called `napari-proofread-brainbow` and available through Napari Hub^a, PyPI^b, and GitHub^c.

^a<https://www.napari-hub.org/plugins/napari-proofread-brainbow>

^b<https://pypi.org/project/napari-proofread-brainbow/>

^c<https://github.com/sbinnee/napari-proofread-brainbow>

ChroMS[2] and Brainbow[1] project that appeared multiple times in this manuscript involves many collaborators, one of whom is I. Arganda-Carreras from Computer Science and Artificial Intelligence Department at the University of the Basque Country (UPV/EHU). They provide us a deep learning model which detects centers of Brainbow cells in the multicolor 3D-volume image. We approached it with the human-in-loop strategy where we constantly improve training annotations while training the model, since the number of cells were massive and sometimes too dense to localize them. For this strategy, it was important to set up an efficient proofreading workflow.

We could have used an existing tool, FIJI[14] for instance, to proofread. However, I decided to use Napari instead, for a few reasons. **(i)** We needed many specific pre- and post-processing steps. While FIJI also provided scripting functions and macros, Napari’s native Python scripting environment was simply more intuitive, easier, and flexible. **(ii)** Annotation tool was more modern and intuitive to use. Not every collaborator was familiar with FIJI and struggled with its outdated interface. But they could use Napari with ease because of its modern interface. **(iii)** There were more 3D-viewing functions to the viewer in Napari. Since the image was 3D, some functions were invaluable and missing in FIJI. One example was “out of slice” function on the prediction layer. Prediction layer contains positions of centers and probability information. The “out of slice” function let users see centers on z-planes (depth-planes) other than the current z-plane, which was helpful to localize centers.

DETAILS

The plugin has two modules in large: a main proofreading module and an experimental module. The main module also consists of several submodules. **(i) Convert to RGB** submodule detects the input dimension and converts it to RGB, so that Napari viewer can recognize the input data and display it properly. It is a purely technical operation because of the difference of dimension convention in software. **(ii) Normalize** submodule auto-detect value ranges of the input and adjust contrast with a one button. Users could adjust contrast natively in Napari viewer, but the function is basic. This module clips values at 99 percentile to filter out big outliers as well. **(iii) Contrast max** submodule adjust contrast

of every loaded layers with a single slider. It is especially useful when the viewer was set to the 3D rendering mode. Without this module, users need to visit each layer and adjust it one by one. (iv) **ZYX scale** submodule is only effective in the 3D rendering mode. It adjusts the scaling factor of three axes. (v) **Points size** submodule manipulates the size of the all predicted points at once. Again, without this module, users needed to select points manually and change the size. (vi) **Grid** submodule generates grids. Grids turned out to be extremely useful to plan and focus on the proofreading process. It also helped to communicate with other proofreaders by pointing out corresponding coordinates. (vii) **Threshold probability** is an experimental module that filter prediction points with a probability greater than a certain threshold value.

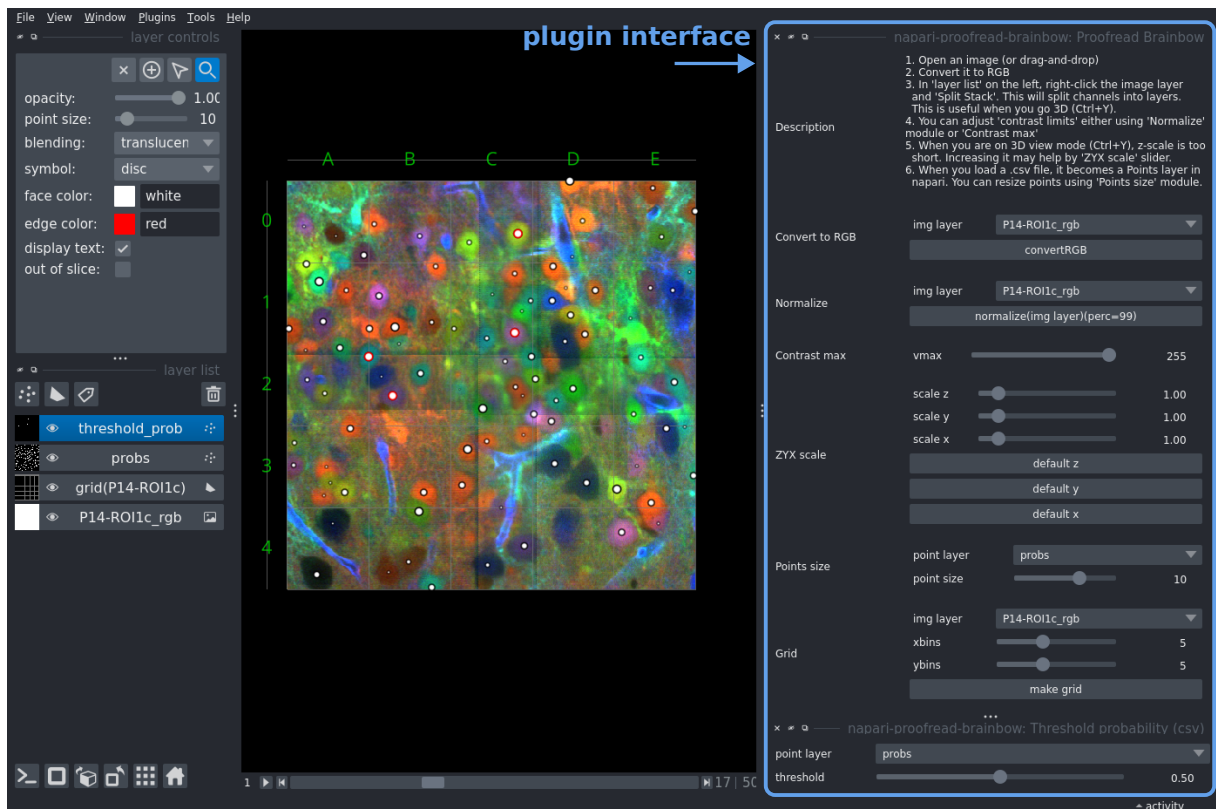


Figure 4.3: Snapshot of the Napari viewer window and the plugin interface. There are two groups of interfaces: a main module and an experimental module. Read the main text for the details.

2.2 NAPARI PLUGIN: BIGANNOTATOR

This plugin remains incomplete and unpublished. But its ideas are still valid. I supervised Slimane Baamara who was involved in developing ideas of `bigannotator`.

BigAnnotator is a plugin for Napari viewer[37] with multiple goals for interactive machine learning in large images. Its main idea was to integrate machine learning, especially deep learning, segmentation methods in Napari and to provide interactive interfaces. It helps users to annotate cells and nuclei in biomedical images, taking advantage of Napari's capability to view large n-dimensional images interactively and Python's availability to use powerful machine learning algorithms from open-science libraries, such as `sklearn`, `opencv`, `scikit-image`, as well as deep learning libraries such as `TensorFlow` and `PyTorch`. Also, it is expected to handle big images taking advantage of the octree rendering¹, which is still a work-in-progress feature of Napari viewer.

Its main goal is the interactive learning. Machine learning and deep learning showed remarkable accuracy for segmentation task in papers. However, these models would not work very well on your own data, simply because they were not optimized for your data. As a result, their predictions are not going to be perfect and almost always need human efforts in the end. BigAnnotator will provide a set of interfaces to facilitate machine learning processes which include image/annotation loading and correcting predicted annotations. Then, BigAnnotator will use the corrected annotation to fine-tune the model to be optimized for your own data.

Its features are

- Iterating images from the root directory of a dataset
- Merging or splitting mask annotations as well as painting
- (WIP) Contour assistance that guides the contour of each object
- (WIP) Interface to train a machine learning model

DETAILS

The plugin is supposed to have four modules in total. Currently, only two of them are available. (i) **DataLoader** module iterates through a dataset directory and exposes buttons for users to load the next image and annotation more easily. It supports simple glob pattern² to match image files and annotation files. (ii) **Annotator** module let users modify, add, or delete annotations. It has three modes: paint, merge, and split. The paint mode allows free painting and deletion. The merge mode provides an intuitive way to merge two or more than two labels. The split mode is used to split a label into two separate labels. These modes come with shortcuts or keybindings to help the operations. There are two utility functions in this module. One is to show label numbers, which is

¹ Octree rendering constructs an octree by dissecting a big volume into small subvolumes and loads only necessary parts in the view window. It also supports the mipmap rendering (a.k.a. pyramid rendering) which supports multiple resolutions of the same view depending on a zoom level in the view window.

² Glob pattern is a simple way to match multiple strings with wild card characters, which are usually a question mark character `?` and an asterisk `*`.

Chapter 4. Applications

trivial but useful to localize labels. The other is to relabel the modified labels in continuous ID numbers. After modifying labels, the label IDs get corrupted. This utility corrects that. Lastly, users can save the modified label.

Contour assistance module is supposed to suggest tight contour annotation. This module is inspired by AnnotatorJ plugin[78] for ImageJ[30]. AnnotatorJ uses a small CNN to segment a single object in an interactive way, integrated in ImageJ. Users may draw a bounding box around an object of interest. Then, the region is cropped and fed into the CNN to segment the object. The output is the boundary of that object. This approach can result in precise contour because free drawing is often shaky. The goal of the contour assistance module of BigAnnotator was to improve the model and port it to Napari. However, it has been already ported to Napari, recently³.

Interfaces to train a deep learning model are going to be complex, because there are too many hyperparameters to consider, and they will likely differ from an architecture to another. The plan is to integrate BioImage Model ZOO[79] and to use its resource description file specification (RDF) to dynamically generate GUIs for each variable.

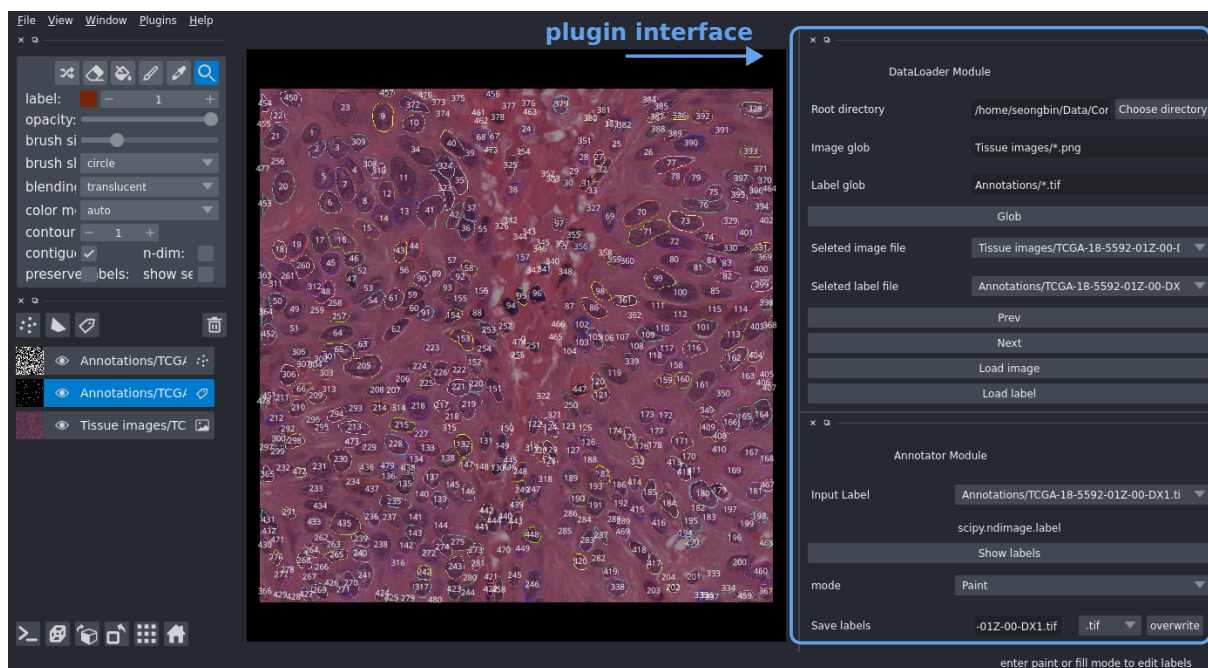


Figure 4.4: Snapshot of BigAnnotator plugin for Napari viewer. An image is overlaid with its mask annotation. Read the main text for the details. Sample image is from ComPath dataset[67].

DISCUSSIONS AND PERSPECTIVES

This plugin started off with a small idea but a grand goal. Many features ended up incomplete either because some APIs were not available from Napari side or because the goal was too big. OME (Open microscopy environment) team was pushing OME-NGFF[74] format for multidimensional large-scale image data and implemented the OME-NGFF specification in Napari. I thought that OME-NGFF would evolve fast and Napari would

³ Napari-AnnotatorJ plugin is available on Napari Hub (<https://www.napari-hub.org/plugins/napari-annotatorj>).

be able to handle multidimensional large-scale image data soon. Apparently, OME-NGFF itself was not a simple project and required a lot of discussions. Also, the octree rendering, the most important feature to handle multidimensional large-scale data, was not stable and still is not at the time of writing.

I believe that the idea of BigAnnotator will circulate and appear constantly and that there will be similar attempts to accomplish the same goal either a part of it or the whole until one definite solution will emerge. For examples, for the annotation part, there are already multiple solutions, such as Annotator plugin[42] for ImageJ and napari-segment-blobs-and-things-with-membranes plugin[199] for Napari. I also think that I put too many high goals together in a plugin, while the purpose of the plugin is to make a small utility per plugin.

3 SEGMENTATION IN CHROMS'S PIPELINE

The ChromS[2] with Brainbow[1] project has two main goals: to develop and improve ChromS at *LOB* and to analyze a mouse brain with Brainbow at *Institut de la vision*. I built a segmentation pipeline to help the analysis. The pipeline segmented cells in the image and quantized colors to identify Brainbow cells. The colors from Brainbow technique come from three main stains, and they get combined with others throughout the development cycle. This mechanism is similar to how we perceive colors in RGB, and therefore displaying Brainbow images in RGB is natural. However, fundamentally, the colors are just markers, and the ratio of channels is what is important and meaningful. Eventually, the goal was to build a pipeline to identify and localize Brainbow cells using ratios of three channels.

Part of the work described in this section will be published as part of a larger article in preparation on the ChromS microscopy platform later this year

METHOD

I simply used a pre-trained Cellpose[29] for segmentation mainly because there was no annotation to train a new model. Although the Cellpose model was not supposed to work well with Brainbow images, since they were not common and very specific, the result turned out to be good enough to build a pipeline even if I replace it later with a model, more tailored to Brainbow images.

I parallelized processes by making chunks of data because the whole data was too big to fit into memory. To do so, I used new Zarr file format[200], which supports chunked N-dimensional data, and is also a default file format of OME-NGFF[74]. The raw image was converted to Zarr file format with multiple resolutions, then I picked the second largest one because it provided fine resolution for Cellpose to identify cells. During inference of Cellpose, the channels were separated first. After, a batch of patches was loaded using Zarr APIs with a pre-defined size appropriate for Cellpose. The batch size was adjusted empirically to maximize the use of GPU resources, thus to maximize the efficiency. Finally, the patches were stitched together. Figure 4.5 shows a result.

After the segmentation, I tried to quantize colors by using KMeans clustering algorithm with the number of clusters set to 6, hoping the algorithm to extract 6 main clusters. The result is presented in figure 4.6 and turned out a failure. The color palette did not contain colors close to yellow (red + green) nor cyan (green + blue). Overall, the palette was too dark too. The main issue was that each cell had more diverse color pixels than it seemed.

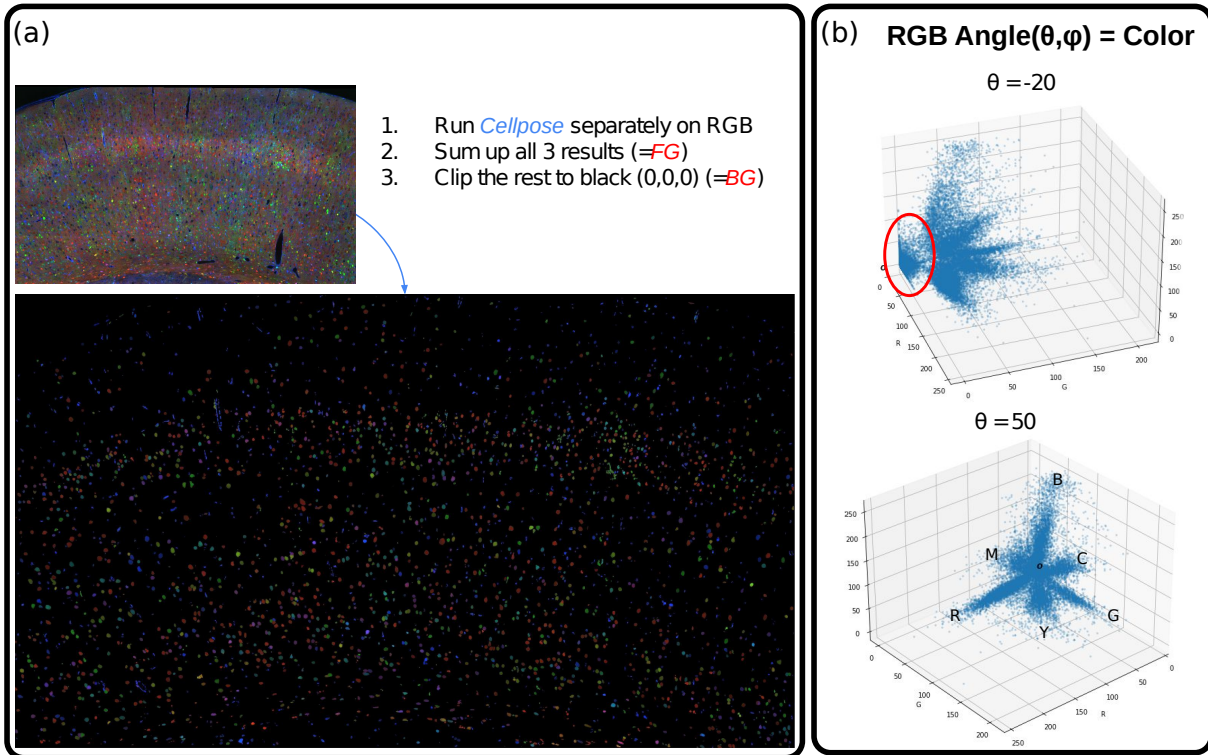


Figure 4.5: Segmenting Brainbow cells[1] for each channel using a pre-trained Cellpose model[29]. (a) After stitched, combining three channels was simply a summation of them. Using this mask, I set all the other pixels to black. The ratio of foreground area to the total area was about 15% (4,071,265/26,369,840). (b) I sampled 1% of masked pixels, which were from about 40K pixels, and plot in 3D space. A huge cluster at the origin (O), which has low values in all three axes, was identified. In addition, it was clear that Brainbow cells have the largest populations along main axes and second largest along three combinations of these main axes. This result well matches to the mechanism of Brainbow staining technique. For convenience, identified clusters were named red (R), green (G), blue (B), cyan (C), yellow (Y), magenta (M).

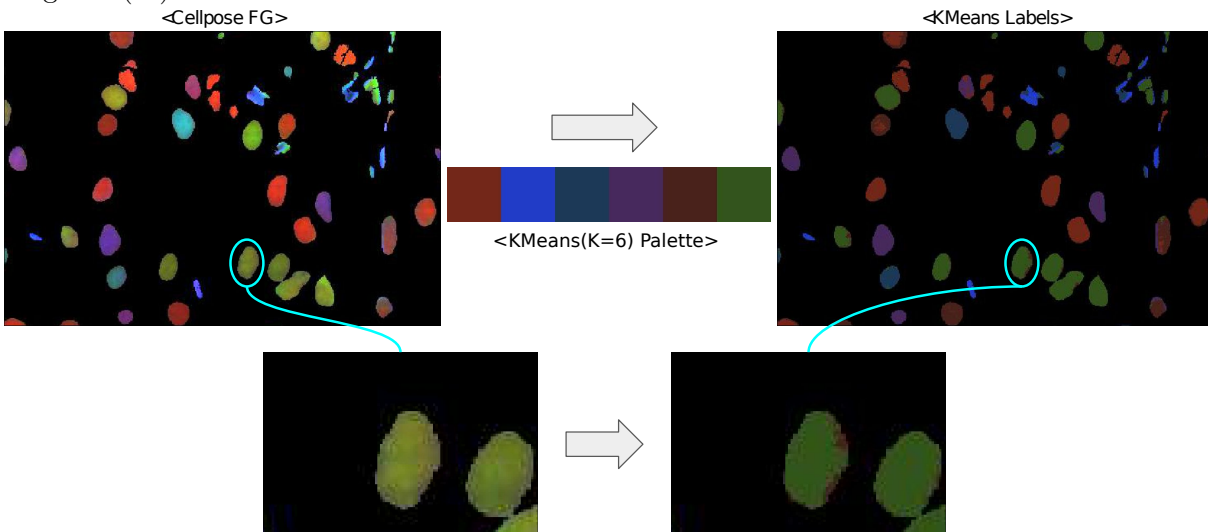


Figure 4.6: Kmeans clustering with $K = 6$ on masked pixels from figure 4.5. The clustering turned out to a failure by looking at the obtained color palette. The issue comes from the fact that the image is dense, and each object is essentially superposed with many layers of other colors, which makes a huge color deviation within each detected cell.

CONCLUSIONS AND PERSPECTIVES

I identified two main issues in the pipeline. One issue was that using a segmentation model which was not designed for this task would always be a suboptimal solution. We needed a 3D model specific to Brainbow images. The other was that the color quantization process was a struggle because of the nature of the image. The image is dense and superimposed with layers of cells, which made unclear boundaries of cells. The solution we came up with was to detect only the center of cells and to extract the color from it. I could have extracted centers from the segmentation result that I got, but we also had to resolve the first issue. In the end, we found a collaborator, who was, as you may have noticed, I. Arganda-Carreras from Computer Science and Artificial Intelligence Department at the University of the Basque Country (UPV/EHU).

I tried other things too, like averaging colors in each cell and cluster them, or clustering colors with less or greater than 6 clusters in Kmeans algorithm. However, in the end, I have been holding this pipeline until a new model is ready from the collaborators because the model will change the pipeline in many ways.

CHAPTER 5

CONCLUSIONS

1 GENERAL CONCLUSIONS AND PERSPECTIVES

Machine learning has become more and more prevalent as we entered the era of digital data in which everything is stored as data, in a digital form. Among many other methods, deep learning has established a solid ground and demonstrated unprecedented advances in machine learning technology. Deep learning methods led paradigm shifts in every domain including our daily life. So far, deep learning stood out mainly in two domains: natural languages and vision tasks, which had been the hardest machine learning problems. In my work in interest, which is to solve computer vision tasks and to develop a versatile model for neurodevelopmental images, to be precise, it was natural to use deep learning.

As a data-driven method by nature, machine learning requires a well curated dataset. It means that the dataset should cover a large variety of data from different sources and big enough to represent the real data distribution behind the observations. Also, to perform the supervised learning, which is the most common and easiest way to get the best performance, every data point should be paired with a target annotation that guides and teaches a machine learning model. However, unlike natural images, biomedical images have multiple issues to expect such large-scale curated datasets, which are not easy to resolve.

My approach to address the data issue was to gather as many existing datasets as possible. I collected more than 20 datasets from multiple sources to cover diverse images, and I managed to curate a generic training dataset by combining them. In the process, I created a programming library called `bioimagerloader` to easily manage individual datasets. I confirmed that collected datasets have high variability in general and some share similar characteristics via a clustering algorithm based on machine learning. Afterwards, I trained StarDist and Cellpose, both of which are the state-of-the-art instance segmentation models, after I found that their pre-trained models performed poorly on local data at *LOB*. The resulted models outperformed their pre-trained baselines by large margins for some dataset and were robust to missing a dataset. However, it also turned out that an absence of a certain dataset had a huge impact, which means that the combined dataset was not generic and diverse enough. Until there will appear a large-scale generic dataset in biomedical imaging community, I believe that this approach will be the most effective way to obtain a generic and versatile model through the supervised learning.

While the supervised learning sets a direct target task and gives us good results to that, it has intrinsic limitations. It relies too much on annotations, is prone to biases, and too rigid to be transferred to other tasks. Consequently, the self-supervised learning, a type of the unsupervised learning, emerged as a promising solution to a generic and versatile machine learning model. I devised a new self-supervised loss, called morphological loss, based on the representation learning, to be precise, the perceptual loss and the neural style transfer framework. The idea was to make a model to generalize a blob-mask style from a bank of examples. Combined with a generative loss, or the content loss, the resulted model, so-called NU-NET could filter blob-shaped objects and enhance their contrast. Again, I combined multiple datasets to secure versatility and generalizability. The morphological loss has high potential to represent other morphologies. NU-NET itself could be used as a foundation model to be transferred to other tasks or domains.

Machine learning is essentially a tool or a part of a pipeline to automate a certain process. I contributed to other projects either using machine learning or even without. Supervised learning to segment fibroblast turned out to be fruitful. A simple plugin could greatly improve the proofreading process. Some projects seemed to need more time and branching.

All around, this has been a good journey to data science and machine learning. In biomedical imaging, machine learning will thrive more and more, and deep learning will dominate and solve many problems. I explored machine learning in between the transition from the supervised learning to the self-supervised learning. I hope that my contribution will help bridge this transition smoothly and result in more and more versatile and generic machine learning models.

2 WIDER PERSPECTIVES

In this section, I would like to discuss topics that are important in machine learning research in general and see how they could affect bioimage analysis.

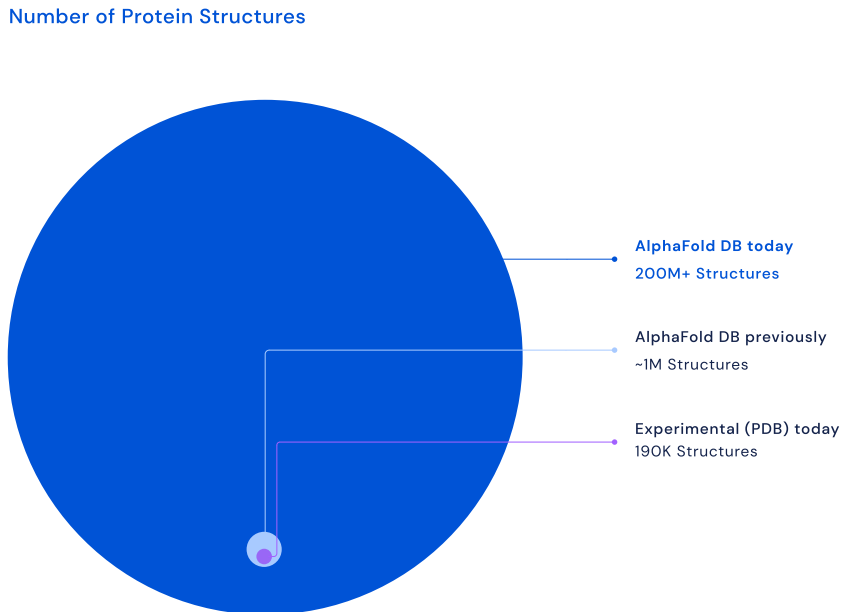


Figure 5.1: Growth of AlphaFold database over time. Since its publication in 2020, the size of the database has grown exponentially. The image was sourced from one of the official blog posts about AlphaFold, published on July 2022[201].

AlphaFold AlphaFold[202] by Google DeepMind team was a big milestone without a doubt. Protein folding is known as one of the hardest problem because of the high unpredictability of the folding process. The number of proteins is endless and the same composition of molecules may end up with completely different functions depending on how they are folded. What AlphaFold achieved was not only meaningful solely in protein research but also meaningful in general because it demonstrated what deep learning could accomplish.

Largely, I believe that AlphaFold was possible because data was available from PDB (protein data bank) and got recognized in a competition called CASF which provided a benchmark. Although there are a few continuous competitions to push computational methods in biomedical imaging, they are rather fragmented and specific to certain tasks, because of the nature of the biomedical image. What I think we need in biomedical imaging community is to run a large-scale competition with a generic task and a solid benchmark. Once we have that, all the innovations will follow suit.

DALL · E Both generic computer vision models and large-scale natural language models were two big milestones in machine learning, but bridging them together was also a huge one. DALL · E[203] was revealed by OpenAI in 2021, which was a deep learning model to generate an image from a text prompt. DALL · E could be born thanks to

the advent of TRANSFORMER, a self-supervised large-scale natural language model based on self-attention[102], such as GPT-3[84], and contrastive learning[180], specifically CLIP model[204]. CLIP stands for Contrastive Language–Image Pre-training and is a model that connected texts and images. The capability of DALL · E simply amazed people and made a lot of noise in social media.

DALL · E did not stop there, and soon OpenAI released DALL · E 2 which greatly improved overall quality and added more functionalities to generate more creative images. DALL · E 2 was internally referred as unCLIP, which means inverting CLIP mode. Additionally, DALL · E 2 utilized VISION TRANSFORMER[105] which is an implementation of self-attention in vision, and GLIDE[205], which was based on diffusion model[33, 191]. The diffusion model is the current state-of-the-art generative model. Combining all the state-of-the-art advances in deep learning, the quality and capability of DALL · E 2 was extraordinary. On top of that, OpenAI team added many other functionalities other than just generating an image from a text.

In summary, DALL · E 2 adopted self-supervised learning approach wherever possible, benefiting large-scale datasets. The only reason, that I can think of, why there have not been many self-supervised models in biomedical imaging community is again lack of large-scale datasets and benchmarks. Also, diffusion model is a super-resolution model, by definition, and has high potential in microscopy pipelines.

Model bias and interpretability I felt that the model bias has been silently ignored behind the amazing advance of machine learning technology. There was a big red flag that Google fired AI researchers who were going to publish papers that concerned biases in deep learning models [208, 209, 210]. One of the fired moved to another company and published the paper anyway, though some contributors asked to take out their names from the paper. The paper was named “Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification” and talked about gender biases as well as other biases [94] in deep learning models. The concern was already circulating a long time ago [211] because it was obvious that the model will be biased by a dataset, and the unbiased dataset does not exist or is very hard to come by.

I believe that model interpretability should be an important criterion to consider when developing and releasing a machine learning model, especially when it concerns social problems. But I must admit that there are not many methods to investigate internals of deep learning model. Feature attribution is one of those, which can be used to evaluate how much a feature attributes to the output [212]. There is also a programming library called Captum[213, 214] that tightly works with Pytorch[135], a popular deep learning framework.


It is needless to say that both model biases and interpretability of models are highly important in biological applications. However, biomedical imaging is not in a leading position of developing deep learning and is busy catching up new methods from other domains. Or maybe it should be biomedical applications that start innovating model biases and interpretability since commercial sectors do not seem to care them as much.

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

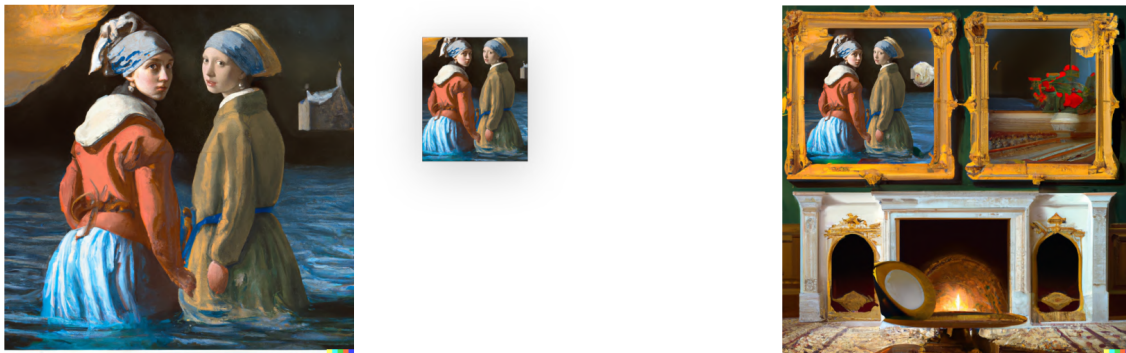


Figure 5.2: Example of DALL · E’s image generation from a given prompt. The image was sourced from its official blog by OpenAI, published on January 2021[206].

 dall·ery gall·ery

Edits: uncropping, context jumping

We can also put our source image in a different context: in a frame on a wall, as a reflection in someone’s eye, in a ‘WANTED’ poster, or something else that suddenly reveals a ‘new layer of reality.’



Here we: position the source image non-centrally, crop it slightly, remove the DALL-E watermark, make it quite small, then prompt for a background: ***‘Framed oil paintings above a fireplace, ornate gold frames, in a Victorian study, detailed digital illustration’***

Figure 5.3: Outpainting feature of DALL · E 2. The starting image was a famous painting *Girl with a Pearl Earring* by Johannes Vermeer. Repeating outpainting processes, DALL · E 2 generated environment outside of paintings based on the given input and the text prompt. DALL · E 2 comes with many other functions other than just generating an image from a text. And the quality is far superior to the previous release. The slide was a page from *The DALL · E 2 Prompt Book*, published on July 2022[207].

Privacy and Open science In early explosion of deep learning technology, everything was open-sourced and shared with anyone. Most resources are still so and many researchers willingly open-source their works. However, more and more major works have been closed by walled gardens. It is apparent for tech-giant companies that make profits using the technology, which include Google, Meta, Microsoft, Netflix, Amazon, etc. But the case of OpenAI was a bit of shock and disappointing to the community.

As its name suggests, OpenAI started off as a non-profit research group whose goal was to do open science to develop AIs. Around 2019, right after they proved their values by publishing GPT family, they gave up the non-profit principle while keeping the name OpenAI. Since then, OpenAI has become basically a for-profit company, getting invest-



Figure 5.4: Machine learning models could be all “stochastic parrots”?

ment mainly from Microsoft. They stopped open-sourcing their resources as well as codes. Their products can be accessed by APIs and need to be paid.

It was true that OpenAI could accomplish so much thanks to a massive investment and develop GPT-3, DALL·E, and CODEX[215], all of which are milestones to the science and technology. Getting investment is not a bad thing. What’s bad is not sharing resources, especially when deep learning method is still young and needs a thorough assessment before deployed and abused. For an example, GPT-3 is extremely capable, but it also showed weird behaviors and vulnerable to certain attacks. One of them is the prompt injection [216] that can be used to exploit the security. For another example, CODEX is a model behind the infamous GitHub Copilot, which auto-completes programming codes (by the way, GitHub is owned by Microsoft). There have been multiple reports that Copilot violated license of some open-source projects. Recently, a group of people filed a lawsuit against the product [217].

Open science in machine learning is becoming more and more closed. In biomedical applications, the aspect of privacy and open science is more dire and sensitive. Biomedical data is more closed than other data by nature. Sharing data needs more attention for its consequences. If we do not address and ignore these issues, open science will have hard time. I feel like it came to a point that it requires a huge amount of resources, which an individual cannot afford, to develop a machine learning model rather than a brilliant idea. I think that deep learning is going too fast, and we need to look back and be conscious to our decision more than ever to secure privacy vulnerability.

Brains and Deep neural networks The artificial neural network (ANN) is a foundation of deep learning and a breakthrough in machine learning technology. The interesting thing is that it shows a lot of similarity how the actual brain works. My knowledge is narrow on neuroscience and cognitive science, but I found a few articles [218, 219, 220] that talked about a mutual reinforcement between neuroscience and machine learning based on ANN. Basically, scientists learn more about brains from ANN and *vice versa*. For instance, a group of researchers recorded brain signals showing images using fMRI (functional Magnetic Resonance Imaging) and tried to reconstruct the images using deep

neural networks [221]. It corresponds to mapping the visual function of a brain to an ANN. Analyzing this ANN may give some insights to how visual signals work in a real brain backwards.

The book *The Brain: The Story of You* [222] contained a lot of examples of how brains work, and I found a lot of similarities with ANN. For examples, learning process is mostly about reinforcing or removing links between neurons. It could partially explain why the dropout layer in ANN, which drops neurons randomly, works. The fact that pruning process can improve the performance is similar. It is said in neuroscience that the brain predicts observations at various levels of abstraction, which explains the way deep layers in ANN work. Procedural memory is a way to link a memory with related memories¹. It is also the case in ANN that features are linked through layers of abstraction [140].

All of these similarities may have been intentional because ANN was an implementation of a biological neural network anyway from the start. But for me, who do not have much knowledge in neuroscience, finding the common mechanism is extremely fascinating and makes me wonder if I can have better understanding of ANN by learning neuroscience, or the other way around. I am sure that I can, but I do not have intention to do another PhD degree at least for a while.

¹ Like a mind palace that Sherlock, played by Benedict Cumberbatch, uses.

ABOUT ME

First of all, thank you for picking my thesis up. And I appreciate that you read this seemingly long and daunting story about three years of my PhD. I would like to make a brief introduction to myself and this book, because when you pick up some films or books, you expect something from their authors, directors, or genres (at least, I do). I hope that my background makes you interested enough to read it through.

Gaming to display engineering As a child born in early 90s, I was always fascinated by computer technology, read “I spent a lot of time gaming in front of computer”. From simple games like pacman and pinball to tactic games like StarCraft2 and Warcraft3 as well as console games on super nintendo and PlayStation2, *computer games* painted a big part of my childhood. Then I had to make a big decision, a life-time decision when I just became an adult and went to college. That was what I would like to do and study for the rest of my life (now I know better that the choice may not be a life-time choice and I do not have to keep it for life). So I started studying *display engineering*. The study was basically to learn how displays work and how to make them. I liked it because the display is always a part of every computer, desktop, TV, and smartphone (iPhone4 was around that time. My first smartphone was iPhone5, and it was such a beauty) and it could display amazing images and movies. I watched the movie Avatar in 2009 by James Cameron three times. And I remembered that everyone was replacing old monitors and TVs by FHD LCDs (Full High Definition Liquid Crystal Display) and the display technology itself was still very much evolving for advanced stuffs like OLEDs (Organic Light Emitting Diodes).

To France How did I end up doing Master’s degree, and what did I study? The time I was about to graduate from Bachelor’s degree, companies like Samsung, LG, and other giants were selling a ton of LCDs. Had I graduated right then, I would have ended up in a factory doing some boring quality check for them. Frankly speaking, I did not want that and wanted to do real science and research for future technologies. Coming to France was actually easy, since my department had a *dual Master’s degree program* with École Polytechnique in France, and not many wanted to study more at the cost of getting an easy but well paying job in Samsung or LG. I applied and got in. The program was to do half of study in Korea and the rest in France². I liked the opportunity because I wanted to learn other things, and I liked traveling (I know that the decision was not purely academic. But come on, it is France, a big European country. And surely it took a big part in my decision.). For disclaimer, I made transition to computer science as soon

² It later turned out that it was not exactly a year for each half. Due to the difference in semester system, that Korean semester starts on March whereas French one starts on September, I took me a year in Korea and a year and a half in France.

as I came to France. You ask what I studied in Korea then? It was about doing research to develop future display devices using graphene (it is a super-duper material, which is the thinnest, strongest and has even amazing conductivity, *theoretically speaking*) and its photosensitivity, which I would say had little to do with displays in near future, but maybe do in far future.

Computer science So I made transition to computer science as soon as I arrived at École Polytechnique, main reason being that everyone, including me, was into it. I found that the scenery in display technology, over the years, had been shifted from hardware to software. Hardware was already good enough. More and more films relied on VFX (Visual Effects) like Marvel movies, and animations from Pixar and Disney showed totally different level of visual details. In fact, there were paradigm shifts everywhere due to computer technology. There was *AlphaGo* in 2016, a computer program beat one of the best go players³. And computer started playing complex tactic games, such as AlphaStar playing StarCraft II, and beating professional human players. Tech giants, such as FAANG⁴, became influential to all industries and to our daily life. The most fascinating technology to me at the moment was a powerful image processing on smartphones and how good the pictures were even from a tiny computing processing unit (CPU) and from an even tinier image sensor. Then I noticed that machine learning was the driving force, and I liked the idea of solving computer vision problems using this new technology (new to me back then).

Finally, PhD Chances are you already know where this story is going, either because you already knew me or because you picked up this book by the title and the abstract. This PhD gave me chances to work on real computer vision problems and bought me time to dip my feet completely into machine learning and of course to stay more in France.

³ Who happened to be Sedol Lee, a Korean. All the Korean media talked about AlphaGo and him all day. They still mention him when they talk about AI (artificial intelligence) or *vice versa*.

⁴ The term refers to Facebook, Amazon, Apple, Netflix, and Google. FAANG already does not represent all the tech giants anymore. It changes now and then. Facebook changed their name to Meta. Some include Microsoft, Nvidia, or Tesla.

APPENDIX A

SHORT INTRODUCTION TO MACHINE LEARNING

I explain basics of machine learning and deep learning in general. This appendix is included for those who are not familiar with the topics and may have difficulty to follow the main text.

1 MACHINE LEARNING WITH EXAMPLES

You may think that machine learning is simply to make a machine that learns or to make a machine to learn. You are not wrong, but when scientists talk about a certain subject, we first make sure whether you and I are actually talking about the same thing. So in order to talk about machine learning in depth, I will give you some context and explain it in the current climate, because scientific terminologies are more prone to change over time and convey more meanings than you think.

Firstly, you may think a machine as a robot¹ that slides down to the kitchen on wheels and picks up an apple for you with their metal hands. I believe that this kind of robot is not far away from current technology, but I do not expect to see it at least in a decade. When computer scientists say a “machine”, it is rather an abstract thing. It can be a system, a logic, a mathematical equation, a statistical model, or more concretely an actual program from some programming codes. The process of making a machine is often called “modeling” and the resulted machine as model. I prefer to use model to refer to a machine, and I will use it a lot from now on. **Secondly**, a machine does not “learn” in a way that we humans learn. To this date, the most well known definition of “machine learning” is written below. It is an excerpt from the book *Machine Learning* by Tom Mitchell and McGraw Hill in 1997.

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”

- [book]*Machine Learning*, Tom Mitchell and McGraw Hill, 1997. -

¹ For me, robots always remind me of either those in the movie *A.I.* by Steven Spielberg in 2001 or *Bicentennial Man* played by Robin Williams in 1999. Okay, I will slide in R2-D2 and C-3PO from *Star Wars*. Speaking of AI (artificial intelligence), we will define it later.

Imagine a baby learning how to walk or how to pick up an apple. Can you tell what each E, T, and P means in this case? Maybe yes, but you will find it hard to specify or even to quantify them. Essentially, the definition needs rigorous mathematical measures, and this mathematical foundation, til now, has been statistics and linear algebra.

statistics
linear
algebra

In early days of machine learning, these so-called tasks (T) have been relatively “simple”. They were not even about picking an apple nor sliding down the kitchen, because they are not actually a single task, but a set of tasks for machines and, I tell you, hard ones. They were things like, **(i) predicting human population**, **(ii) playing checkers game²**, **(iii) having a chat with a computer**, or relatively recently, **(iv) recognizing handwritten digits**. These problems were hard, and some still are. You may have already guessed why they can be hard, but let’s see these four tasks one by one together and let me tell you the current state of machine learning with above examples. For a disclaimer, in this introduction to machine learning, I am going to cover not only computer vision, which is the main topic of thesis, but also other topics, one example being language processing to show you larger pictures of the machine learning landscape. As a matter of fact, computer vision and language processing have been two leading forces of a modern form of machine learning, known as deep learning, and they have grown together sharing ideas from one to the other.

computer
vision
language
processing
deep
learning

1.1 POPULATION: REGRESSION

First problem is to predict population, say, til the end of 21st century. Supposing that you are given population data from 1950, you may want to draw a graph: time on x-axis and population on y-axis and put all data points in. Then you feel like that you want to connect these points with a smooth line, because you know that the line is supposed to be smooth and continuous, as per our intuition of big numbers (we are talking about billions). Once you have a line, you would notice a certain tendency of the curve and think that you found an answer. This method is called regression³. Nobody can tell you that your answer is wrong, because who knows the future, but one can tell that you lack of reasoning. This is why mathematics and statistics came up with machines that could model data. Now, see UN’s prediction of world population in 2022 in figure A.1. It is actually a band of lines with probability indicated. Although I do not know the details of actual data and how they built their regression model, predicting population must have required them not only population data in the past but also other data, at least mortality rate and birth rate. As soon as new types of data are added, the problem gains layers of complexity, and you need more proper tools than eyeballing on a graph.

regression

Main takeaways from this example are

- Big data is smooth and continuous
- Prediction comes with probability
- More layers of data would yield more accurate prediction but require more sophisticated models

² Search Arthur Samuel for more; in 1956

³ In practice, depending on the context, this is often called fitting, because we are literally fitting a line to data. Interpolation or extrapolation are also used.

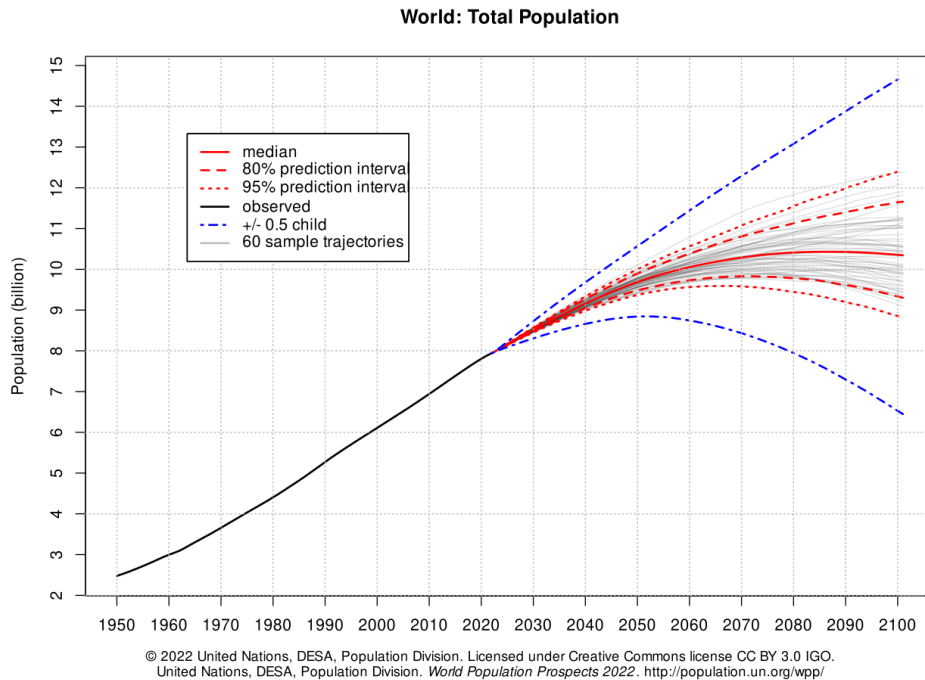


Figure A.1: UN’s prediction of world population in 2022

1.2 CHECKERS: MEMORIZATION VS. LEARNING

Second problem is to make a program that plays checkers. This one might be daunting, because you may not know how to code a computer program. But I am not talking about an actual computer program, we can design a program on a piece of paper with rules and logics. Checkers have rules: it has a board, two players, players take turn, pieces jump, and so on. Given these rules, we can imagine opponent’s moves and make winning sequences. You know that it is already too many cases and seems almost infinite. But it is not infinite and computer can solve things much faster than us, given a problem is finite. We can make a program that plays checkers over and over and records all possible cases. This method is called brute-force approach, and the checkers was solved in 2007 [223]. Best you can do against this machine is a draw. Even so, the searching space amounts to 5×10^{20} and you need more than your laptop only to navigate this space let alone to compute it⁴. brute-force

I call this machine stupid not that it is not efficient enough to run it on my laptop. The resulted machine did not actually learn how to play checkers but rather memorized all the sequences in checkers. It has the book of winning and the only thing it does is to look for sequences in the book. For example, if I were to play variations, I simply cannot with this machine. Memorization does not hurt, but we want an efficient and intelligent machine which can learn the rules and play with logics, and it would be cool if it is just good enough to beat the world champion on your laptop. You know what would be cooler? What if it can adapt itself to variations of checkers with some rules changed? In a word, we want efficient and intelligent machines that learn and have potential to improve. This concept can be summarized by a one word, that is generalization. I will bring up the term generalization

⁴ Chess is known to have complexity between 10^{40} to 10^{50} and go has around 10^{172} .

versatility

generalization throughout this book, and it is the key towards versatility as you might have guessed. ⁵

Takeaways from this example are

- Brute-force approach memorizes, which is not bad. But we want machines to learn logics.
- Efficiency matters.
- We want versatility, meaning a room to improve and even an ability to adapt.
- Generalization is what we want to achieve.

1.3 CHATBOT: NATURAL LANGUAGE PROCESSING AND SELF-SUPERVISED LEARNING

Third one is to make a chatbot. *You may* have already seen some on certain websites, like I did. I bought a product, and it turned out having some flaws, so I wanted to get a refund. I opened their website looking for a phone number to their customer service, but to no avail. Then I saw this chat window at the bottom of the webpage. I started explaining my issues and realized that the conversation was a bit awkward because maybe they were too nice with perfect punctuations and capital letters. At some point, I noticed that I was not actually talking to a person but to a program, which basically looks for certain keywords, such as problem, issue, refund, return, delay, or delivery, and parrots prerecorded sentences. At last, I got a phone number because my problem was not simple enough for a chatbot to deal with (why would they make a refund process easy?), only to discover that there was also prerecorded audio on the line, and I had to wait for 10 minutes. Alas, how annoying the world has become!

Natural language processing I wanted to know how you would feel about my story. Instead of asking you, which is physically impossible over this book, I asked a machine called GPT-3[84], developed by a company called *OpenAI*. Below are the prompts I put in and the answers from the machine.

Prompt 1:

(copy the whole paragraph above from “*You may ...*”) + What do you think?

Answers 1:

I think it is quite annoying. I also think that it is quite sad that companies

⁵ If you are a gamer, you may know about the ray tracing. Ray tracing is another good example why people want machine learning approach rather than brute-force one. Ray tracing calculates all the light paths and interactions given a 3-D environment. Since the complexity is so enormous, it has mainly used what’s called Monte-Carlo method, which is a half-way brute-force approach. Ray tracing using Monte-Carlo method randomly samples a portion of light at a time just until you get plausible results and call it done. Although there are many ways of doing Monte-Carlo, such as Markov-Chain Monte-Carlo (MCMC), ray tracing is still too demanding to catch up 60Hz refresh rate (in another term, 60 frames per second; 60FPS is a minimum specification many feel that the scene is real-time). However, already out there, is a machine learning solution from the company Nvidia, called Deep Learning Super Sampling (DLSS). Check it out, because the result is amazing.

are resorting to this type of customer service instead of hiring actual people to help their customers.

Prompt 2:

(copy the whole paragraph above from “*You may ...*”) + Why do you think that chatbots are so dumb?

Answers 2:

There are a few reasons. First, it is still unclear what is the best way to design a chatbot. It is said that people do not like talking to chatbots, and often find them frustrating. Second, chatbots are often not very good at understanding natural language. They may be able to understand some simple questions, but often fail to understand more complex questions. Finally, chatbots are often not very good at dealing with unexpected situations. For example, if a customer asks a chatbot a question that it is not prepared for, the chatbot may not be able to give a sensible answer.

The first answer is some sympathetic answer which people and I used to not expect to get from a machine for a long time. The second one actually seemed dare to ignore all my story and to take only the question in the end. Although I did not actually hope to get analytical answer from the question, the answer seems to be a correct self-critique that makes sense. Overall, the answers feel natural⁶.

Let’s go back to how we should make a less annoying chatbot. Language is far more complicated than checkers. Furthermore, natural languages have rules, but there are too many and all sorts of exceptions are accepted. Naturally, you could imagine that *brute-force* approach would not work, let alone coming up with all different cases. In fact, natural language processing (NLP) is one of two big topics that current machine learning has been addressing and seen big progress over the years, together with computer vision. Though the language modeling has been one of the oldest machine learning problems⁷, with really hard one such as NLP, for language to be felt natural like examples above, we need to talk about a particular way of teaching, or more generally, training machines, which is self-supervised learning. Self-supervised learning turned out to be a good way to make a generalized model, given a large amount of data. However, I am going to save answers what self-supervised learning is and how to use it to make NLP models such as GPT-3, for now. We will deep dive into it shortly after, because self-supervised learning is the main focus of my thesis and deserves its own section. If you cannot wait, you could jump to section ?? and have a peek. But I recommend you to stay and continue, because the next topic is about computer vision which is another main focus of the thesis.

natural
language
processing
(NLP)

training
self-
supervised
learning
generalized
model

⁶ Do you feel unproductive? You should ask GPT-3 some tips! <https://adolos.substack.com/p/feeling-unproductive-maybe-you-should>. This is a blog post generated by GPT-3, not long after it was released in 2020. It gained a lot of attention because it managed to trick some people.

⁷ Turing test is one of the earliest attempts to define artificial intelligence. Basic idea is following. You talk to a computer and a human only with texts, not knowing which one is which. If you cannot tell a computer from a human, this computer is considered passing Turing test, and thus intelligent. Turing test was proposed in 1950 by Alan Turing. Search Turing test or imitation game, if you are interested.

Appendix A. Short Introduction to Machine Learning

We have an answer but not a full one, yet. Currently, to make the best chatbot, we had better take self-supervised learning approach. Takeaways from chatbots or natural language processing are

- ~~Chatbots are annoying.~~
- Current state-of-the-art NLP models, *e.g.* GPT-3, feel natural.
- For tasks where there is no rule or are too many rules, such as natural languages, we cannot consider a brute-force approach.
- We need to think about other ways to train a generalized model.
- Self-supervised learning is the answer.

1.4 HANDWRITTEN DIGITS: COMPUTER VISION AND CONVOLUTIONAL LAYER

computer
vision

Fourth and the last task is to make a machine that recognizes handwritten digits. Computer vision is a term to describe a group of tasks for computer to recognize images. Main computer vision tasks are **image classification**, **semantic segmentation**, **object detection**, **instance segmentation**, **keypoint detection**, and **panoptic segmentation**. Their examples are shown in figure A.2. Image classification itself went through some changes. At first, images were small, and a machine was given an image with a single object centered, like in (a) PascalVOC[224] and (b) ImageNet[62] datasets. Then, people wanted to classify normal images with different objects within (c) and to know to which class each pixel belongs (d), and they realized that it was not the image classification task they had known, so they called it semantic segmentation or pixel classification. Naturally, they wanted to know where each object is located (e) and to separate it from others by putting bounding boxes and called this task object detection. (f) Instance segmentation goes further to overcome constraints of rectangular bounding boxes and classifies pixels instead. (g) Panoptic segmentation assigns classes and instances for every pixel. (h) Keypoint detection focuses on each object and predicts their structures and poses. Note that these are not all the computer vision tasks and that most tasks introduced here are from a dataset called Microsoft COCO (Common Object in Context) [137].

instance
segmentation

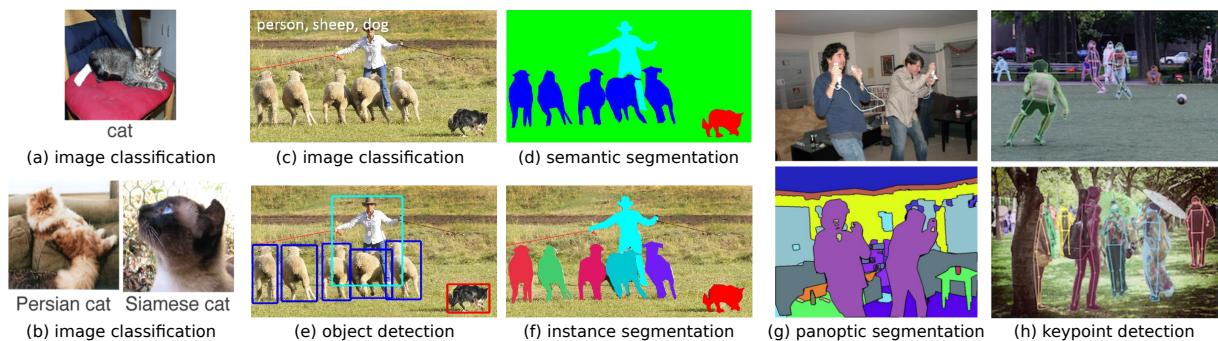


Figure A.2: Common computer vision tasks: (a, b, c) image classification, (d) semantic segmentation, (e) object detection, (f) instance segmentation, (g) panoptic segmentation, (h) keypoint detection. Sources: (a) PascalVOC[224], (b) ImageNet[62], (c, d, e, f, g, h) MSCOCO[137].

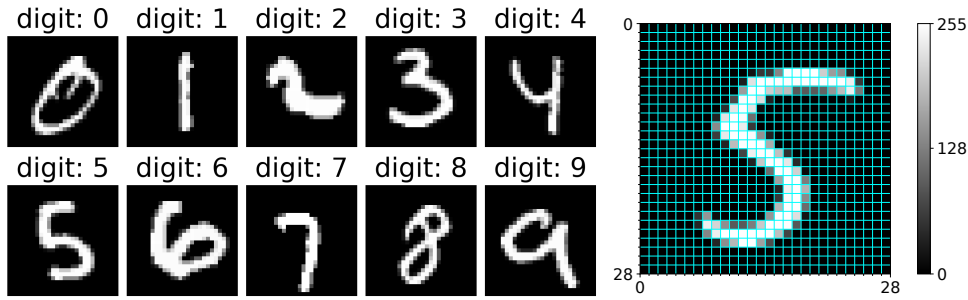


Figure A.3: Samples from MNIST database, which is a classic image classification task. Right one shows pixels with grid lines and their values coded as brightness in 8-bit integer (as known as UINT8 that ranges in 0-255).

Recognizing handwritten digits is a classic image classification task. One of the most famous datasets for the task is MNIST database (Modified National Institute of Standards and Technology database). It was introduced in 1998 together with a convolutional neural network (CNN) called *LeNet*[225]. The database contains 70,000 digital images of handwritten digits in 28x28 pixels, whose samples are shown in figure A.3. And *LeNet* already achieved an error rate less than 1% on 10,000 test images. To this date, the convolutional layer which is a core building block of *LeNet*, is used for the state-of-the-art models for this task. So we will take a shortcut and look into the convolutional layer right away.

convolutional neural network (CNN)

Convolutional layer To process image data, which is basically a sequence of numbers, we need to make it an array or a matrix ready to be computed. Naïve way is flattening pixels to make an array with 784 ($= 28 \times 28$) elements and to consider each element as a feature. As soon as it is flattened, however, it loses relationship of rows and columns depending on the direction of flattening. We could already imagine that this approach is not going to work well. The most important characteristics of image data is the localized information with neighboring pixels, and we want to preserve that.

Convolutional layer is the core building block of CNNs. It is named after the convolution operation in mathematics and was mainly designed for 2-dimensional (2D) data, such as images⁸. In fact, there are a lot of image filters based on the same convolution operation in 2D, such as a Sobel filter, Gaussian filter, etc⁹. The difference is, for many cases, in kernels. You can think a kernel as a window or a pair of glasses to look through something, in our case images. They are not just transparent though. Sobel filter has fixed kernels, Gaussian filter has those parameterized by standard deviation values (σ). Convolutional layer is one of those parameterized, but it is specifically designed for machine learning. Its parameters are randomly initialized and learned during training process. Figure A.4 shows these three kernels, figure A.5 demonstrates how to apply them, and figure A.6

convolutional layer
kernels
parameterized
initialized

⁸ Convolutional layer works well in any data that has localized properties. It turned out that it works well on audio data too. Another application is to develop a fast PDE (partial differential equation) solver. It makes sense to use convolutional layer because derivatives look for tiny changes or neighboring values.

⁹ If you use microscopes, you may know PSF (point spread function). Fundamentally, PSF is a scattering pattern of light, and it looks a lot like Gaussian filter or elongated kernels, once reproduced spatially. Scattering is one of the main factors that hinder resolving crisp images in microscopy. Compensating PSF is called deconvolution and is a common post-processing after acquiring microscopy images. This type of problem that finds the origin of cause is called inverse problem, in general.

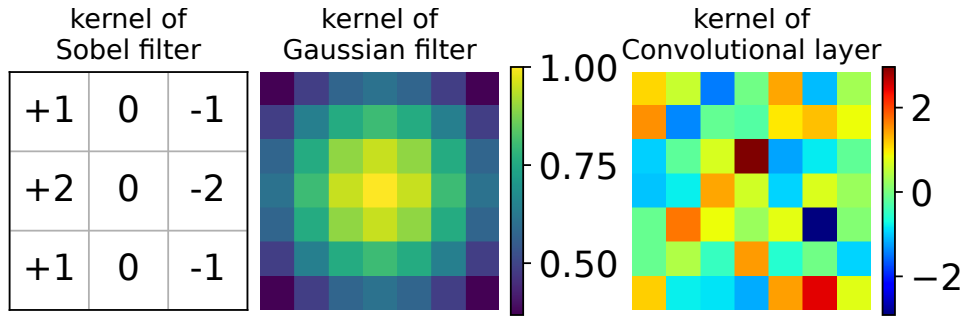


Figure A.4: Kernels of the Sobel filter, the Gaussian filter, and a randomly initialized 7×7 convolutional layer, from left to right. Note that the kernel of the convolutional layer was initialized with standard normal distribution, and did not go through any training.

displays results convolved with these kernels.

optimized
channels
feature maps

Sobel filter is an edge detection filter as you can see in figure A.6. Gaussian filter blurs images, which does not seem to do much in this example, but it becomes useful when you have a lot of noise or small objects that you do not want. Lastly, we have a result with a kernel of convolutional layer, and it looks useless and in fact it is. It is mainly because it was not learned or optimized for a certain task through a training process. In addition, a convolutional layer, in fact, has more than one kernel, and they are called channels. By stacking all channels with different initial values, convolutional layer makes feature maps of a given input, just like a certain composition of a kernel can be used for detecting edges or for blurring effect. In a way, a convolutional layer can be considered as a group of filters, each of which captures different characteristics of a given input. We just do not know how these filters will end up after a training process. But what we know for certain is that they would become optimized for a task that we will set.

weight

We can formulate the actual calculation as in equation A.1 below. \mathbf{X} refers to an input, \mathbf{W} is called a weight and represents kernels of a convolutional layer, and \mathbf{Y} is an output. \vec{b} is called bias because it makes easy to make biases across channels. Remember that I initialed convolutional kernels by randomly sampling from the standard normal distribution, which has mean value at 0 and has standard deviation of 1, $\mathcal{N}(0, 1^2)$ in formal notation. Bias also has the same number of channels, but each channel has a scalar, meaning a single value. We can use these values to give biases to each kernel across channels by adding them, essentially making each channel to have its own mean value $\mathcal{N}(b_c, 1^2)$. Since it is a 2D operation, we can simply consider them as 2D matrices or tensors.¹⁰ Let N indicate the number of channels of a convolutional layer. We can assign actual numbers to make it easy to understand the below equation A.1 and its operation in figure A.7. $\mathbf{X} \in \mathbb{R}^{28 \times 28}$; $\mathbf{W} \in \mathbb{R}^{N \times 7 \times 7}$; $\vec{b} \in \mathbb{R}^N$; the resulted output $\mathbf{Y} \in \mathbb{R}^{N \times 28 \times 28}$.

tensors

¹¹ As you might have noticed, the operator $*$ has its own parameters, namely the size of kernel, the number of stride, the number of padding, and the mode of padding. We can tweak these to have different behaviors of a convolutional layer.

¹⁰ N-dimensional array, where N is greater than 2, is often called tensor. Google developed an open source library called TensorFlow and a new type of processing unit called TPU (Tensor Processing Unit), specialized for tensor calculation. You can see that both are named after tensors.

¹¹ Bold variables, such as \mathbf{X} , \mathbf{Y} , \mathbf{W} , mean matrices or tensors. Variables with an arrow on top, like \vec{b} , mean 1D arrays or vectors.

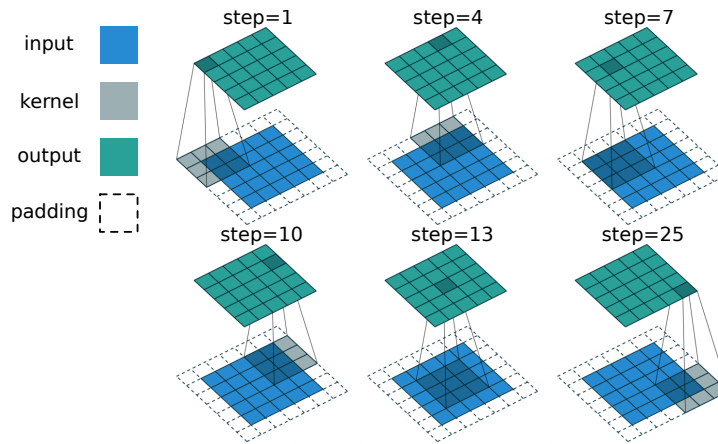


Figure A.5: 2D convolution operation. It performs pixel-wise product and summation between the kernel and input to yield an output. It starts from the top left corner and move on to the next to the right one step at a time. Once it reaches to the end of right corner, it goes down by one and starts from the left corner again. Given that the input has 25 elements and the stride (step size) is 1, the total number of steps is 25. Notice that dotted area is padded in input to ensure the same size of output. By changing the size of kernel, stride, or number of padding, we can make a lot of variations. As a side note, in practice, these computations are efficiently parallelized on GPU. In this case, theoretically, if you have more than 25 cores, GPU can do all the jobs at the same time. And current GPUs usually have more than thousands of cores. Illustration was taken from [226] and annotated.

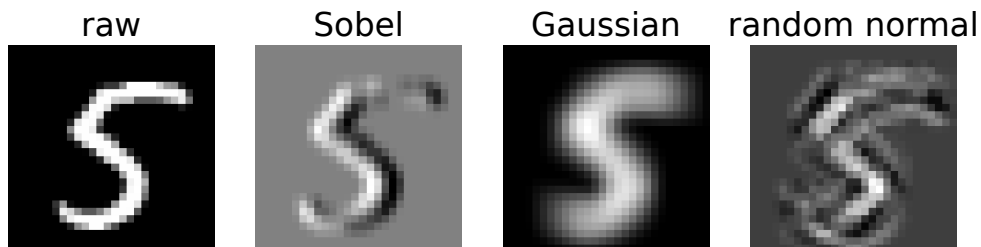


Figure A.6: Resulted images convolved with kernels of Sobel filter, Gaussian filter, and convolutional layer from figure A.4. Since the kernel of convolutional layer was not trained, the resulted image (at the right most) conveys little meaning or information.

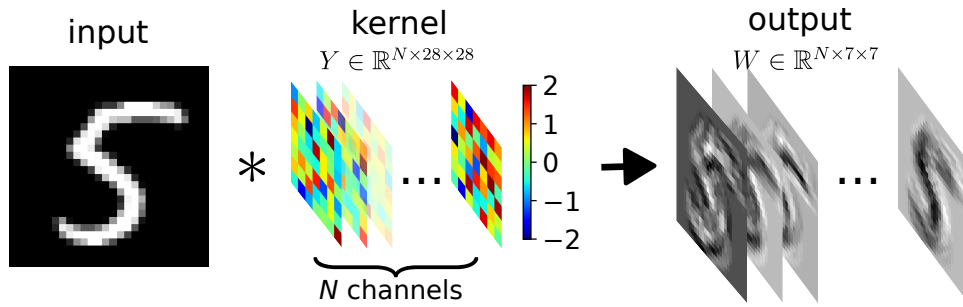


Figure A.7: Convolutional layer has multiple channels and all of them are initialized separately to produce rich feature maps. Symbol $*$ represents convolution, and here I used 1 for the step size (stride) and 3 for the zero-padding (for each edge). Beware that input, kernel and output are different in size and the spatial ratio is not respected.

$$\mathbf{Y}_c = \mathbf{W}_c * \mathbf{X} + b_c, \quad c \in \{1, 2, 3, \dots, N\} \quad (\text{A.1})$$

We have a half answer to build a state-of-the-art machine through machine learning to recognize handwritten digits, that we would like to use the convolutional layer. Lastly, believe or not, MINST database together with *LeNet* is one of the milestones that ignited development of neural networks and deep learning, which is the remaining half answer and the most crucial part enabling the current success of machine learning.

neural
networks
deep
learning

Takeaways from recognizing handwritten digits are

- Digital images consists of pixels, each of which has a numerical value.
- Common computer vision tasks are classification, semantic segmentation, object detection, instance segmentation, *etc.*
- For images, we want to preserve localized information taking into account neighboring pixels.
- Convolutional layer is one way to achieve that. It is a core building block of convolutional neural networks (CNNs) which is the base for many current state-of-the-art machine learning models for computer vision tasks.
- Convolutional layer is parameterized and is optimized for a given task through a training process.
- Convolutional layer has multiple channels to make a handful of feature maps.
- Convolutional layer is one of two keys to make a vision model, and the other one is neural networks and deep learning that we are going to talk about shortly.

2 GENERATIVE MODEL

Generative model is at the heart of machine learning and a general interest of all deep CNNs. Generative model is a machine to generate data, which sounds simple but not so much to do properly. Combined with machine learning, the goal of the generative model is to make or simulate new data. Essentially, to make new data, the generative model needs to learn how to model the given data, which is close to the definition of machine learning itself. Generative modeling in computer vision became viral when the generative adversarial network (GAN) came out[48]. Therefore, starting with GAN, I am going to give brief introduction to two generative models in vision: **generative adversarial network (GAN)**, and **variational autoencoder (VAE)**. For the heads-up, VAE will be covered more in depth in chapter 3. Accompany figure 1.25 to follow along.

2.1 PROBABILITY DENSITY DISTRIBUTION

Before talking about generative models, I am going to present the probability density distribution. It is an abstract concept and might be hard to grasp the idea at first, but it will be helpful to move forward. We will begin with a set of one-dimensional data and its probability density distribution and generalize it to \mathcal{D} -dimensional data. This one-dimensional dataset looks as in figure A.8. I generated 1,000 numbers, and it is hard to see patterns when they are written as digits. Once you put them on a line, you start seeing some. Though I applied transparency to see overlapped data points, it is still not clear. A better way is to make a density function, so that data sums up to 1. I randomly sampled 20 numbers ($n = 20$) and draw a dotted line, where a solid line is the actual density function ($p^*(x)$) from which I generated, or sampled, 1,000 numbers.

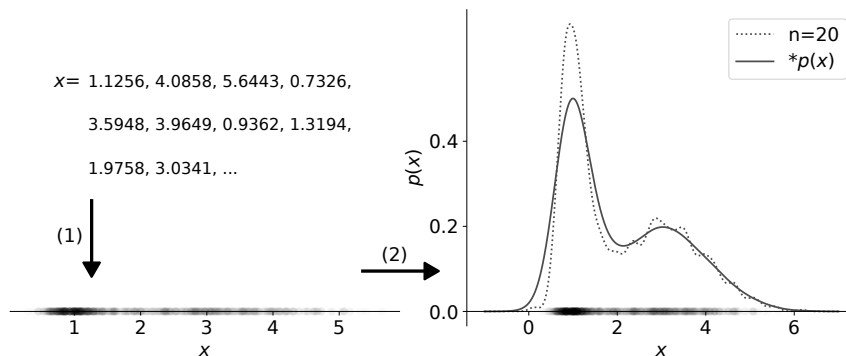


Figure A.8: One dimensional data, its approximated probability density distribution, and its true density function. Displaying numbers is not useful to spot patterns. Probability density distribution could reveal the underlying true distribution, or an oracle distribution $p^*(x)$. The oracle distribution $p^*(x)$ was a sum of two Gaussian distributions. For the dotted line of 20 samples, I used the kernel density estimation method with Gaussian kernels.

Real data is usually multidimensional and image data is especially so. We can generalize this notion of probability density distribution to represent our data $\mathbf{x} \in \mathbb{R}^{\mathcal{D}}$ that has arbitrary dimension of \mathcal{D} . We assume that it has an underlying true distribution, or an oracle distribution, $p^*(\mathbf{x})$. To find it out, we make observations, or sample \mathbf{x} and model a probability density distribution $p(\mathbf{x})$. In theory, generative modeling is a process of making $p(\mathbf{x})$ to approximate $p^*(\mathbf{x})$. However, in practice, we rarely use $p^*(\mathbf{x})$ because it is

the ideal goal that is not supposed to be achievable¹². More precisely, the approximated distribution is written as $p_\theta(\mathbf{x})$, where θ implies a set of parameters of a model, suggesting that $p_\theta(\mathbf{x})$ is a parameterized approximation.

2.2 GENERATION PROCESS

Conditional probability is an essential tool since we set our data distribution as the probability density function. A conditional probability describes a probability of an event after another. It is written as $P(A|B)$ and read “probability of A given B”. Naturally, $P(B)$ is a prior probability and $P(A)$ becomes a posterior probability. Once applied to a density function, $p(\mathbf{x}|\mathbf{z})$ represents the generative process of \mathbf{G} , which is read “a probability density distribution of \mathbf{x} given \mathbf{z} , or conditioned on \mathbf{z} ”.

2.3 DISTANCE BETWEEN DISTRIBUTIONS

Kullback-Leibler divergence is a simple and popular choice to calculate a distance of two probability distributions. Kullback-Leibler divergence (D_{KL}) is defined in the below equation A.2 for two discrete functions p and q . As you may have noticed, $\log(p/q)$ is just a difference of their log-likelihoods, and $p(x)$ term weights it (therefore, D_{KL} is not commutative nor a symmetric measure, meaning $D_{KL}(p|q) \neq D_{KL}(q|p)$). What’s important to remember is that D_{KL} computes a distance of two probability density functions and becomes 0 when they are identical. With this in mind, let’s dive into generative models.

$$\begin{aligned} D_{KL}(p|q) &= \sum p(x) \log \frac{p(x)}{q(x)} \\ D_{KL}(p|q) &= \sum p(x) (\log p(x) - \log q(x)) \end{aligned} \tag{A.2}$$

¹² It is hardly possible to find the oracles or the universal rules in real life, just like physics. Newtonian physics can explain and successfully approximate many things, but it cannot explain quantum phenomena such as nuclear fusion and fission, or predict deviation from quantum effects such as the speed of light changing under the general relativity. That being said, versatile and generalized modeling can be considered as a process of finding the oracle $p^*(\mathbf{x})$ from a limited set of data $p(\mathbf{x})$.

3 ARTIFICIAL NEURAL NETWORK

For the time being, the word artificial intelligence (AI) is almost equivalent to (artificial) neural network or deep learning. It only requires basic knowledge of linear algebra, probability theory, and calculus to understand the building blocks of the artificial neural network. That being said, its strength does not come from some advanced mathematical breakthroughs that few people understand, but rather from its simplicity combined with its ability to scale big.

3.1 AI OR AGI? CALL IT SIMPLY DEEP LEARNING

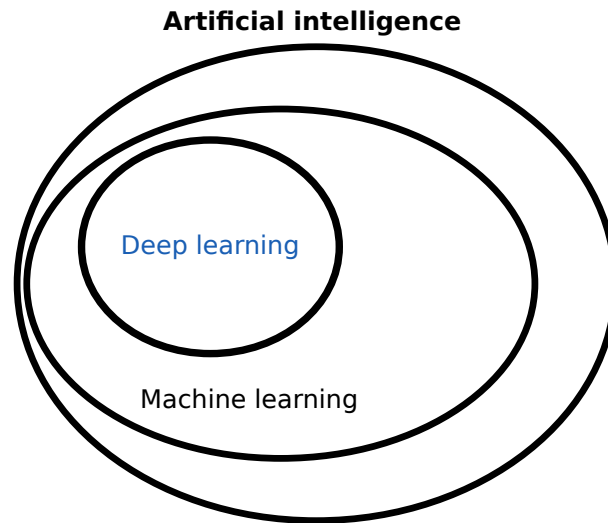


Figure A.9: Relationship among AI, machine learning, and deep learning. Reproduced from book *Deep learning*[227].

I must admit that the word artificial intelligence (AI) has become a buzzword in media over the years. As a matter of fact, AI, at least how it used to be in computer science, was almost equivalent to computer itself that automates any little things. It was partially because computer never had enough intelligence to content human expectations, and the definition of computer intelligence was elusive, let alone any intelligence¹³. As a result, it was never a main stream topic and the term itself became mediocre (some call it literally “weak AI”). If you think of human-like androids for AI, that you saw in science fiction films, the correct terminology, you are looking for at the moment, is artificial general intelligence (AGI)¹⁴. AGI is also often called “strong AI” to differentiate from the weak one. Its definition is still a hot-topic and very much in discussion. And some apparently do not like “general” in AGI either, because it means more or less human-like intelligence, but they believe that human intelligence is a highly specialized one (and I agree to that). For that reason, computer scientists rarely say either AI or AGI referring to

¹³ Richard Feynman, one of the most important and influential scientists in modern history commented on AI in 1985. You can easily find the recording on internet, and I highly recommend it (Video link: [228]). He said that artificial intelligence will never be the same as human intelligence as long as computer does not have the same neurons that we have. Also, the quote at the very first page of introduction is his. But you will see shortly the artificial neural network and may find its similarity to biological neural networks.

¹⁴ Even AGI may not be enough to describe such intelligent robots at the current time.

artificial
neural
network
deep
learning

current machine learning technologies. They rather directly use (artificial) neural network and deep learning, which are two foundations that have been leading current success of machine learning and what we are going to cover in the following sections.

3.2 PERCEPTRON

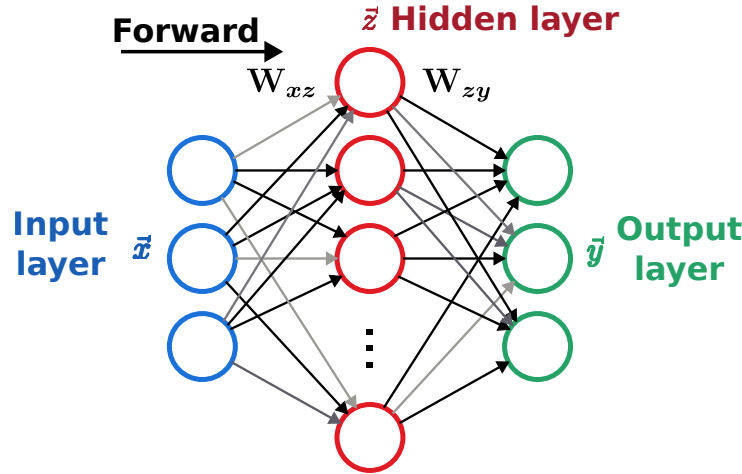


Figure A.10: An artificial neural network with two layers of perceptrons. Every circle except blue ones, a.k.a. the input, is a perceptron or a neuron, if you like. \vec{x} is an input, \vec{y} is an output, and \vec{z} is a hidden layer. \mathbf{W}_{xz} is a group of weights between \vec{x} and \vec{z} , and \mathbf{W}_{zy} is another between \vec{z} and \vec{y} . Notice that it has a direction, and for that, it is sometimes called feed-forward neural network. Also, lines have differences in brightness, referring to weights of bonds.

perceptrons

Artificial neural networks (ANNs) was born around 60s, and it is said that it got inspired from a biological neural network. We have knowledge of individual biological neurons and a concept of synapse that transmits electrical signal from one neuron to others. Each neuron is simple, but a network built upon many of them can do complex things and have certain functions¹⁵. Artificial neural network consists of a lot of simple elements too, so-called perceptrons, which are equivalent to neurons (researchers just call them neurons in the context of artificial neural networks). Single perceptron has a single parameter (I omitted a bias because it is not as important and some even choose not to optimize) that will be optimized and performs a simple linear operation, literally a multiplication with input and an addition.

weight

layer

The first line of below equation A.3 has w as a weight, x for an input, y for an output, and b for a bias. This equation is almost identical to the equation A.1 for the convolutional layer. Once you scale it and make an array of N perceptrons, then you have a layer with a size of N ¹⁶. Let's assume that we have an input array $\vec{x} \in \mathbb{R}^M$, and we want an output array $\vec{y} \in \mathbb{R}^N$. If all elements in \vec{x} are connected to \vec{y} , we have $M \times N$ connections in between, and these connections become a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times M}$. It also has a bias term \vec{b} just like the convolutional layer.¹⁷

¹⁵ Collective intelligence is common and natural. I remember that I was amazed by collective intelligence of ants while reading *Les Fourmis* by Bernard Werber in my childhood. Single entity looks tiny and trivial *per se*, but once they are gathered they can do complex tasks.

¹⁶ This layer is called linear layer, dense layer, or fully connected layer.

¹⁷ Again, a variable with an arrow on top refers to an array or a vector, and bold one is a matrix or a tensor.

$$\begin{aligned} \text{Single perceptron} &: y = wx + b \\ \text{Layer of perceptrons} &: \vec{y} = \mathbf{W} \cdot \vec{x} + \vec{b} \end{aligned} \tag{A.3}$$

One missing ingredient is the activation function or transfer function that comes right after this calculation. Activation function also has similarity to biological neurons. As we know, a biological neuron has a threshold, a certain electric potential, that allows for a signal to be transferred. Popular choices of the activation function are simple non-linear functions just to add a bit of non-linearity in the process, because the whole network, without an activation function, is a big linear function, which is not an interesting thing. The most popular and the simplest one is the rectified linear unit[229] (ReLU) in figure A.11. ReLU lets a signal through if it has a value greater than 0 and blocks it otherwise, which can be written as $ReLU(x) = \max(0, x)$. Combined with a bias \vec{b} , ReLU makes gates with different threshold values to a layer of perceptrons $ReLU(x) = \max(0, x + b)$.¹⁸

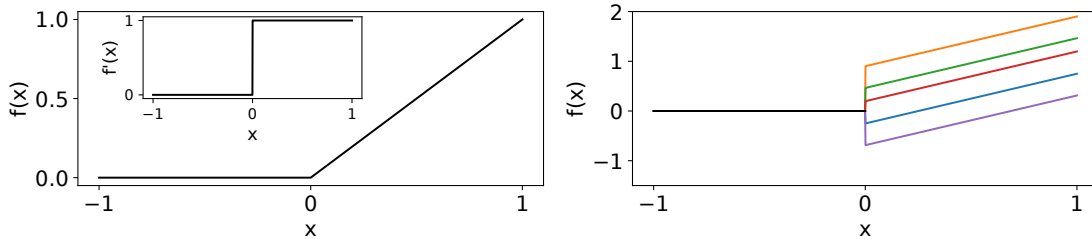


Figure A.11: (left) ReLU activation function $f(x) = \max(0, x)$ and its derivative $f'(x)$ in inset, (right) ReLUs with 5 different biases.

Once we stack layers of perceptrons, we have a multi-layer perceptron (MLP) or an artificial neural network as in figure A.10. This example has two layers of perceptrons, one of which being an output layer. All the perceptron layers that have learnable or trainable parameters are called hidden layer and are commonly represented as \vec{z} (notice that this is not anymore \vec{y} as in equation above. \vec{y} denotes the output layer in common.). Though we have only two layers, the total number of parameters to optimize has already become relatively huge.

To give an example for MNIST, let's set the number of perceptron $N = 1000$ and make the output a vector whose size is 10 for each element to represent a digit from 0 to 9, which is known as one-hot vector¹⁹. The image input vector is $\vec{x} \in \mathbb{R}^{784}$ (once it is flattened), thus the weight of the hidden layer becomes $\mathbf{W}_{xz} \in \mathbb{R}^{1000 \times 784}$. The output layer also has a weight that would be $\mathbf{W}_{zy} \in \mathbb{R}^{3 \times 1000}$. As a result, this seemingly simple neural network with one hidden layer ended up having **787,000 trainable parameters** (not counting biases). When you have two hidden layers with the same size, the number grows to **1,787,000**. You can see that neural networks can grow really fast in terms of complexity, and training a neural network was not a simple task back in the past when

¹⁸ ReLU is generally not used in the last layer as known as the output layer, because we want as much information as possible.

¹⁹ The idea is to have a vector whose size is the same as the number of classes, or categories, we want and to treat values as pseudo-probability. For example, if the first element has the greatest value, we would know that it is likely to be digit 0, the second is to be digit 1, and so on.

dimensions

the computing power was limited. *LeNet-5*^[225] that had 5 hidden layers and **60,000** trainable parameters took **2-3 days** to train on a CPU (Computing central Processing Unit) in 1998. High complexity and slow training time were main reasons why neural networks were not a popular choice back in the past, because computer scientists thought that networks' huge complexity is an overkill and that they could find more efficient algorithms to solve the problem. And they did find more efficient algorithms for MNIST that performed on par with *LeNet*. However, as data has higher dimensions and more features, and as tasks become more complicated, it turned out that neural networks are more scalable, more adaptable, and result in by far the best performing models. Now the question is how to train your neural networks?

3.3 OPTIMIZATION AND TRAINING

objective

function,

loss function

entropy loss

mean square

error

probabilities

The goal of pretty much all optimization algorithms is to find a minimum, just like how physics wants to reach the minimal energy, and optimizing a neural network is no exception. What we need is a performance measure P (remember the definition of machine learning by Tom Mitchell and McGraw Hill?), that we are going to call objective function, loss function, or simply loss ²⁰.

Entropy loss and Mean squared error Two most common losses are the entropy loss and the mean squared error (MSE). The choice depends on what we want for our output to be. When we want our outputs to be probabilities, which by definition have values in range between 0 and 1, the entropy loss is likely what we want. Otherwise, we can use the mean squared error, in general.

log-

likelihood

Shannon

information

Entropy is a concept brought from thermodynamics as you may have noticed and has a lot to do with probability theory. Natural phenomena can be mathematically best explained in the form of exponential and logarithm functions²¹, and probability theory uses them as well. A probability is a value in range between 0 and 1, and we follow special rules when we do arithmetic with it. For example, we multiply probabilities ($p_1 \cdot p_2$) when we talk about a sequence of events with their probabilities²². And its logarithmic form, log-likelihood ($\log p$), not only makes computation easy but also quantifies information. Multiplication becomes an addition $\log(p_1 \cdot p_2) = \log p_1 + \log p_2$. Probability is intuitive to us, but the concept of it is not something linear where we can just add them as we saw right before. Shannon information defines the content of information as $-\log p$. Figure A.12 visualizes logarithm functions in value range between 0 and 1.

Entropy loss is essentially a loss function of probabilities and employed for classification tasks. When defined for a binary classification task where we define a target to be

²⁰ What if you want higher performance, higher P ? Then you can simply negate it and make a loss function $-P$ to minimize. Optimization algorithms are designed to minimize values, in general.

²¹ When defined with the base being Euler's number (e ; in my mother tongue, Korean, it is called approximately *natural constant*), they become the natural exponential ($y = e^x$) and the natural logarithm function ($y = \ln(x)$). One is the inverse function of the other, which means that you get the other one by exchanging y and x . Note that, often, \ln refers to a logarithm with e being the base and \log with 10 as the base. I am going to use \log not specifying a base, because computationally it is not important.

²² Multiplication of probabilities is something we take naturally. We can simply multiply two probabilities if one does not impact the other. But when two events are entangled, you need a theorem to estimate its probability. Bayer's theorem defines generalized rules of probability arithmetic.

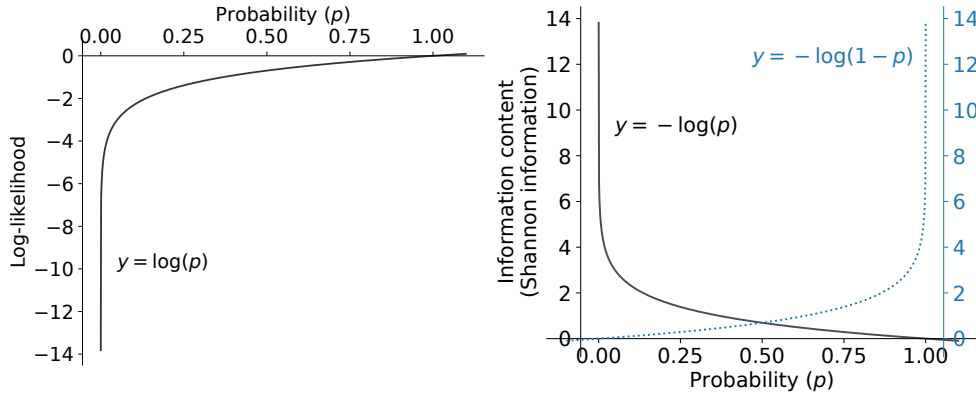


Figure A.12: When computing probabilities, they are often converted to logarithms. Log-likelihood (left) and Shannon information (right) have close relationship to entropy and quantify information. Shannon information intuitively tells that when an event has probability of 1, it is deterministic and carries no information and when it has probability of 0, we cannot predict anything, thus it has infinite information. Blue dotted line (\cdots) indicates binary cross entropy when the target is 0 (read the entropy loss and see equation A.4).

either 0 or 1 ($y \in \{0, 1\}$), the binary entropy loss function $\mathcal{L}(p, y)$, given a probability p can be written as followed, and you can see that it is measuring Shannon information in figure A.12:

$$\begin{aligned} \mathcal{L}(p, y) &= -(y \log p + (1 - y) \log(1 - p)) \\ \mathcal{L}(p, y) &= \begin{cases} -\log p & \text{if } y \text{ is } 1 \\ -\log(1 - p) & \text{if } y \text{ is } 0 \end{cases} \end{aligned} \tag{A.4}$$

However, how can we ensure the output layer to have probabilities from a neural network? The output from a linear layer without an activation function technically can have any values and its values are not bound between 0 and 1. We can either truncate each prediction value x by using a function such as a sigmoid or normalize all predictions ($\vec{x} = (x_1, x_2, \dots, x_i, \dots)$; where $i \in \{0, 1\}$ for binary case) to be summed up to 1 by using functions such as Softmax. Below equation A.5 shows the logistic function which is a Softmax popular choice of sigmoid functions, and the Softmax function $\mu(\vec{x})$ and its binary case. logistic function Figure A.12 visualizes the logistic function and Softmax, more generally, for 10 cases. function

$$\begin{aligned} \text{logistic: } p &= f(x) = \frac{e^x}{1 + e^x} \\ \text{Softmax: } p_i &= \mu(\vec{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}, \text{ so that } \sum_i p_i = 1 \\ \text{Binary Softmax: } \vec{p} &= (p_0, p_1) = (\mu(\vec{x})_0, \mu(\vec{x})_1) = \left(\frac{e^{x_0}}{e^{x_0} + e^{x_1}}, \frac{e^{x_1}}{e^{x_0} + e^{x_1}} \right) \end{aligned} \tag{A.5}$$

Mean square error (MSE) is a loss function more appropriate for regression, allowing for predictions to have any values. MSE is fundamentally an averaged value of squared differences and is defined as below in equation A.6. What is nice about MSE is that it is only a second-order polynomial and that its order is an even number. It is cheap enough to compute, and its derivative becomes a linear function that allows negative value as well.

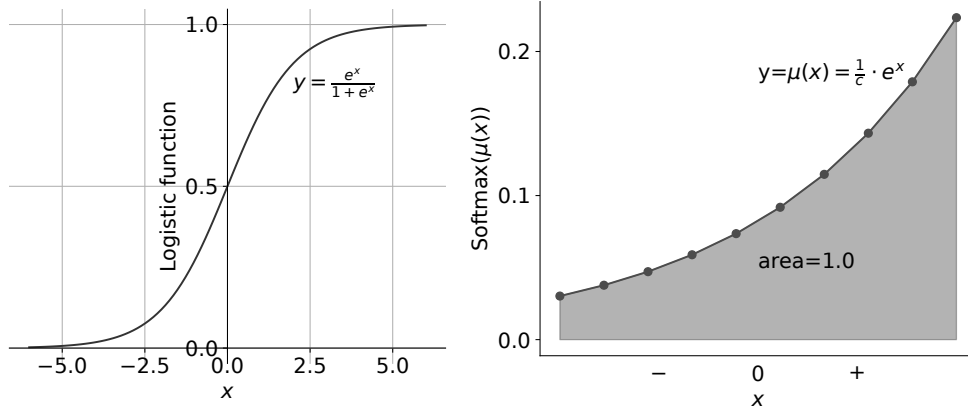


Figure A.13: Logistic function (left) and Softmax function for 10 cases (right). Output values from neural networks are not guaranteed to be probabilities. To treat them as such, they need to be converted to behave like probabilities. Logistic function and Softmax are two popular choices. They also have relatively simple derivatives, which are helpful for gradient based optimizations.

An alternative would be the L1 loss function²³ ($\mathcal{L}(x, y) = |x - y|$), which has simpler, maybe too simple, derivatives: -1 if $x - y < 0$ and 1 if $x - y > 0$. The reason I keep talking about the derivative is that it is our main tool to optimize neural networks.

derivative

$$\begin{aligned} \mathcal{L}(x_i, y_i) &= (x_i - y_i)^2 \\ \mathcal{L}(\vec{x}, \vec{y}) &= \frac{1}{N} \sum_{i=1}^N \mathcal{L}(x_i, y_i) \\ \frac{\partial \mathcal{L}(x, y)}{\partial x} &= 2(x - y) \left(1 - \frac{\partial y}{\partial x}\right) \end{aligned} \tag{A.6}$$

Blindfolded hiker When it comes to optimizing neural networks, the stochastic gradient descent (SGD) is the foundation of all the other algorithms that are widely used. To understand SGD, we need basic calculus. But I am going to first explain it without calculus through the *blindfolded hiker analogy*. Imagine that you are in a mountain with blindfolded and would like to descend. To help you, I have a 3D visualization of an object function in figure A.14. You would like to be very careful with every step you take, so you make a following set of rules. **First**, figure out your surrounding, especially slopes of each direction. You could use a foot, for example, to do so. Mathematically, this action corresponds to finding derivatives and gradients²⁴. **Second**, you would go forward a certain distance to a direction whose slope is the steepest downward, because it is likely to lead you further down. **Third**, you repeat until you get to the bottom. The process is essentially iterative.

stochastic
gradient
descent
(SGD)
calculus

gradients

iterative

You may have a lot of questions for these rules, because your life may depend on them. Most of all, why are you blindfolded? How many directions you would like to set? How to determine a step size, one foot or two feet? How far should you go for a step? And how do you know that you reached the bottom? Optimization process is highly complicated and slow process and even so for neural networks because they have a huge number of

²³ MSE is sometimes called L2 loss because it is a second-order compared to L1 loss.

²⁴ Derivative is a function and a gradient is a value of it given a point.

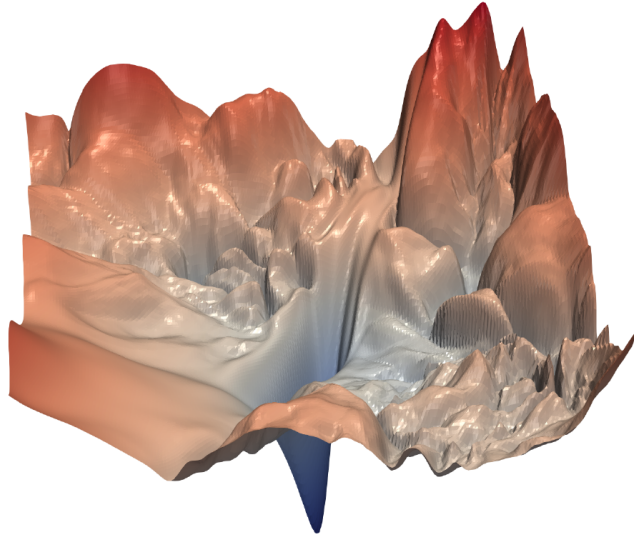


Figure A.14: Visualization of a loss surface in 3D of a popular neural network, called ResNet-56[85]. Note that loss function has very high dimension which is not easy to visualize. This surface was generated by using a very specific technique from [230]. The goal is to get to the lowest point of this mountain though it looks like a free fall rather than home.

neurons, and the optimizer should figure out how each individual neuron should behave. One major breakthrough to speed up the process was a stochastic exploration (or a divide-and-conquer approach, if you like).

Stochastic Gradient Descent (SGD) Do you think that it is easier and quicker to find the bottom of a mountain if you were not blindfolded and thus could see far ahead? Normally yes, if you were hiking on an easy terrain. But it turns out that the *loss surface* of neural networks has the number of dimensions as large as that of trainable parameters, meaning it is very complex and processing all the information at the same time is highly complicated and time and energy consuming. It is sometimes more confusing when you have more information available around you and a step further does not seem to change much of landscapes you see. So blindfolding limits information and actually makes you more decisive and thus makes you move quickly.

Stochastic gradient descent (SGD) is an optimization algorithm to randomly sample a small portion of data, called mini-batch, from the entire dataset and to update trainable parameters of the network. In this case, you may want to take a small step, because you know that your information is limited. So it is common to have a small learning rate with a small mini-batch size. Note that parameters other than those of neural networks are called hyperparameter to be differentiated, and the size of mini-batch and the learning rate are considered as the most important hyperparameters. Updating a parameter is a simple process once you know the gradient. Let t indicate the current step and $t - 1$ a previous step. θ is a parameter (or your position) to update. α is a constant learning rate (or a step size). Lastly, a gradient (a slope) is g . Hence, updating process can be described as follows:

stochastic
gradient
descent
mini-batch
update
learning rate
hyper-
parameter

updating
process

$$\theta_t \leftarrow \theta_{t-1} - \alpha g_{t-1} \quad (\text{A.7})$$

momentum

You could add more rules and strategies, one of which being a momentum. Basically, once you find that you have been walking in the same direction for a while, you may want to take a further step or may not want to change the direction. You could add a constant momentum hyperparameter to SGD to make it faster, or there are other choices, for instance *Adam*[231] that dynamically calculates a momentum for each step²⁵.

Validation and Testing subsets How do you know that you reached the bottom or when to stop? The answer is *you never know*. In fact, you do not even know there will be “the bottom”, and you will likely find a lot of “bottoms”, which are not the one you were looking for. Gradient-based optimization assumes convexity of the objective function so that it has minima. The smallest minimum is called global minimum among local minima. At all the minima, the slope becomes flat and has slopes towards it, as known as convex slopes. So when you, as a blindfolded hiker, feel that the terrain started becoming flat downwards, you could assume that you likely reached a bottom. You will find flat surfaces a lot, and still you never know if the one you just found is the global minimum or even a local minimum. To find that out, you just need to walk more and inspect its surrounding. Solely relying on flat surfaces to find out minima is lacking, as you can imagine.

Validation subset is what researchers use and can help find minima more quickly. The hiker analogy becomes a bit of *science fiction* at this point and breaks some physics²⁶. For a hiker, it is like making a replica of the mountain by taking out some parts of it, say 20%, in a virtual universe. There is more. The parts taken out are physically moved to the virtual universe, so you are no longer able to see or walk on them in the real world. You can open a portal to this virtual world every several steps you take in the real world and test your hiking skill virtually and check whether you could reach further down. Once it is done, you can come back to where you were left off and continue hiking in the real world. Isn't it like learning to hike in a parallel universe as well? Unfortunately, you cannot bring what you learned in the parallel universe, because knowledge belongs to their own universes, if you did not know (or you might actually have gone to future and forgot what you learned, because it is actually what you will learn, thus you cannot learn what you will learn, just yet). No, actually you do not want to learn on the virtual mountain. You want to isolate the validation process from the learning one to get an objective measure of your skills. Otherwise, you get confused between what you learn and what you can do.

Your validation subset, or a replicated mountain in a virtual universe, serves two purposes. **First**, it gives you a *relatively objective* measure how you are performing. Though the subset still comes from the same dataset, isolating a subset from a training one gives you a counter metric to observe your progress. **Second**, it tells you whether you are cheating and memorizing data. Let's say that you hiked a mountain so many times. You would start memorizing passages without noticing. The danger is that you may not perform well and not challenge on other types of terrains once you get too familiar with one terrain. This slump or a stagnating state, not learning much of new knowledge and

²⁵ Two most popular optimizers are probably SGD and Adam. There are a lot of variants of SGD and Adam where Adam itself is a variant of SGD. The key factor is how efficient they can be at the cost of accuracy. The name, Adam, is derived from adaptive momentum estimation. Adam estimates a momentum based on first-order derivative of gradients, which is cheap to compute and provides enough accuracy.

²⁶ Do not forget to get a towel before hitchhiking to the galaxy!

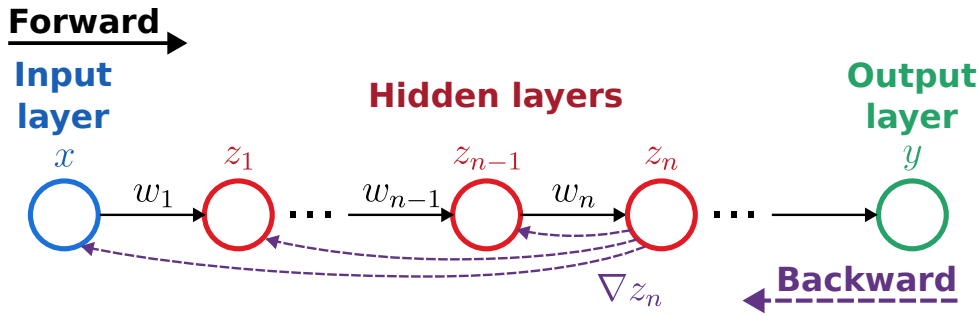


Figure A.15: A simple neural network where all layers have only one neuron, including the input and output layer. Neurons are linked forward. To get a gradient, it needs to propagate all the linked neurons backwards til the input. Note that all variables are scalars.

memorizing, is called overfitting in machine learning. The validation set tells when you start becoming overfitted, and that you need to stop and face a new challenge.

Testing subset tells whether you could hike other mountains, so to speak. Remember that our goal is to learn how to hike and to improve your hiking skills in general. The validation set is not objective enough because you already used it during training to measure your performance. The testing set should be a subset which you never saw before, especially during training, to give you an objective measure (this is why I said that the validation set is *relatively objective*) to tell how versatile and how general your acquired skills are. It would be best if you could try out other mountains to test your ability. However, often in practice, we sample and set aside a portion of data from the given dataset and save it for testing later once you finished your training. In short, before you doing anything, you would like to split your dataset into three subsets: training, validation, and testing sets.

Vanishing gradients and ReLU Gradients are very useful tools for optimization. What we need to think about is how to actually compute them. A neural network consists of numerous neurons linked together as in figure A.10. It means that a gradient of a neuron is linked to other gradients. Thankfully, calculating gradients has an analytic solution²⁷ and is easy thanks to the chain rule. We will start from a simple network that has N hidden layers ($z_n, n \in \{1, 2, \dots, N\}$) between an input x and an output y . Every layer has only one neuron, thus all become scalar values. In addition, there is no activation function after each layer. We can visualize it in figure A.15 and write down how a hidden neuron z_n is calculated:

$$\begin{aligned}
 z_1 &= w_1x + b_1 \\
 z_{n-1} &= w_{n-1}z_{n-2} + b_{n-1} \\
 z_n &= w_nz_{n-1} + b_n
 \end{aligned}
 \tag{A.8}$$

Getting a gradient of z_n becomes very easy thanks to the aforementioned chain rule.

²⁷ Analytic solution is a closed form of solution, which we are familiar with from a math classroom. Polynomial equations have nice analytic solutions for their derivatives, for example, and you can avoid using a numerical solver. Numerical solver is what a computer uses to find a solution relying on values, when it does not know the underlying formula. It is fundamentally slow and requires a heavy computation.

$$\begin{aligned}
 \frac{d}{dx} z_1 &= w_1 \\
 \frac{d}{dx} z_n &= w_n \frac{d}{dx} z_{n-1} \\
 &= w_n w_{n-1} \frac{d}{dx} z_{n-2} \\
 &= w_n w_{n-1} w_{n-2} \dots w_2 \frac{d}{dx} z_1 \\
 &= \prod_{i=1}^n w_i
 \end{aligned} \tag{A.9}$$

Once we applied ReLU activation functions (f) to all the layers, the gradient of z_n (∇z_n) becomes:

$$\nabla z_n = \frac{d}{dx} z_n = \begin{cases} 0 & \text{at least one of } f(z_i) \text{ is } 0 \\ \prod_{i=1}^n w_i & \text{otherwise} \end{cases} \tag{A.10}$$

This process of calculating gradients going backward is called backpropagation. Both the forward pass and the backpropagation are computationally demanding processes when updating a neural network. However, it is easy to solve the backpropagation at least in neural networks, and neural networks were designed to be so. Since we know their analytic derivatives for every neuron, and we can simply chain their gradients.

You may think that the result is a bit weird because ReLU activation outputs 0 with half probability (when the input is less than 0) and the gradient ∇z_n is most likely to become 0, meaning not being updated. You are wrong and correct. We talked about a single neuron in each hidden layer, but a functioning neural network would have from several hundreds to a few thousands in a layer, if not more. So, it is okay to have a lot of zero gradients. Also, it is not like that those neuron are not going to be updated at all, because we considered a single data point and a single iteration. In truth, ReLU activation function is a well-thought-out choice and actually helps to train the network fast[229, 18]. Researchers used to use a sigmoid for an activation function, because, you know, sigmoid functions have nice curves and are natural. But if you look back at the logistic function in figure A.13, you would notice long tails at the both ends of the curve. These long tails, in derivative, caused what's called vanishing gradients, since their derivatives easily become less than 1 and many times close to 0. Once you chain them all, gradients preferably become 0, and they end up barely meaningful though many neurons would get updated.

GPU, TPU, and CUDA We saw that getting a gradient was an easy task in neural networks. However, when we consider the actual network that has a countless number of neurons, it becomes a heavy burden. Graphical processing unit (GPU) was designed to handle computation of graphical elements for the display or the monitor. GPUs have typically thousands of cores to parallelize computing visuals of each pixel²⁸, compared to several or a few tenths at the most for CPUs (central processing units). As industries

²⁸ Basic visual traits are color, transparency, and texture.

moved towards FHD, 4K, and 8K resolution²⁹ display devices, chip companies started developing powerful GPUs to meet the demands. In parallel, the machine learning community discovered that GPUs can be used for neural networks and be a better fit than CPUs, because of the sheer numbers of neurons and computations.

CUDA is an API (application programming interface)³⁰ for GPUs produced by company *Nvidia*. CUDA provides a set of instructions which could be useful for computational tasks other than the displaying functionality. *Nvidia* pioneered GPU computation earlier than its competitors and has been dominating in the market. It is not an overstatement that majority of recent neural network researches used GPUs from *Nvidia* and CUDA API. *Google* invested computing cloud platforms early and developed the TPU (tensor processing unit) which focuses on parallel computation, the omitting displaying functionality. Their API is called XLA, and it has been well received by researchers thanks to their mature frameworks such as TensorFlow and JAX. There are other APIs such as ROCm by *AMD* and oneAPI by *Intel*, which are supposed to be more open and to support wider ranges of processing units and operating systems³¹ (OSs) but lagging far behind CUDA and XLA.

In brief, it was powerful GPUs that accelerated development of neural networks thanks to their capability of parallel computation. As they get more and more powerful, people could afford more demanding computation, meaning that neural networks got larger and deeper. And we have arrived in the deep learning era.

Depending on the task, you may need certain algorithms, or you may simply want to try things out. Machine learning frameworks can be grouped into two: machine learning frameworks that are *not deep learning* and *deep learning*.

Deep learning Frameworks To utilize GPU, TPU, and CUDA, people use frameworks. There are two main runners in deep learning frameworks: *Google* and *Meta*. Thankfully, their frameworks are FOSS³², and their APIs are consistent and easy to use. GPU support is the most important feature to accelerate and parallelize tensor (or matrix) computation. These frameworks, in common, can calculate gradients automatically, which is the key technology to optimize neural networks. Three frameworks have stood out: TensorFlow, PyTorch, and JAX.

TensorFlow^[232] is maintained by *Google*. It was one of the first deep learning frameworks available in Python. For every instruction, TensorFlow draws a computation graph which is translated to XLA (accelerated linear algebra) API and CUDA API (by *Nvidia*)

²⁹ FHD (Full high-definition) is denoted as 1080p in many cases. It refers to a resolution that has 1920 × 1080 pixels, which has 16:9 ratio. The letter “p” means the progressive scan as opposed to the interlaced scan (“i”), which alternates half of pixels at a frame to mimic 60 Hz refresh rate with two 30 Hz frames and to reduce power by half. Interlaced scan died out rather quickly, as consumers prefer high refresh rate displays like 120 Hz. 4K and 8K correspond to rough numbers of vertical pixels.

³⁰ API, in a plain language, is a set of programming instructions to make useful applications. Driver is also an interface that takes these instructions and directly talks to hardware through operating systems (OSs). API exists on top of drivers.

³¹ Computer operating systems such as Windows, Linux, and MacOS.

³² I believe that they think that technical development at the moment is more important than monetizing it. And it is not like they are not making money out of machine learning technologies. Another trend, that I see, is to keep software free and open, but make and sell hardware that is optimized for it.

that perform actual computation later. Graph mode is fast for execution, but it turned out inconvenient because it behaves like a compiled language, meaning that actually computation happens once a graph is complete. Therefore, it felt unnatural in Python, which is a scripting language. TensorFlow started supporting the eager mode, which executes computation right away, since version 2.0. TensorFlow's native APIs are known unfriendly. **Keras** is a library on top of TensorFlow that provides intuitive high-level APIs. TensorFlow later decided to ship Keras in the package. TensorFlow and Keras are good choice if you care more about performance than development, or you have TPU (tensor processing unit) which is *Google* hardware. TensorFlow also has a JavaScript library, which can be helpful if you are aiming to deploy your deep learning model through a web application.

PyTorch[135] is probably the most popular framework for developing neural networks. It is maintained by *Meta*. It pioneered eager execution thanks to the *autograd* module, which automatically keeps track of gradients for the later optimization step. Thus, PyTorch is more development-friendly because you get the results right away even without losing gradients. Additionally, PyTorch's APIs are Pythonic, which makes it more approachable and easier to integrate other Python libraries compared to other frameworks (for me, TensorFlow feels like a wrapper for XLA, which is closer to pure linear algebra than deep learning).

JAX[233] is yet another framework designed for deep learning, developed by *Google*. JAX also uses XLA for its backend, but it is more Pythonic than TensorFlow. It focuses on eager execution as well as JIT (just-in-time) compilation. JIT compiles a piece of codes on demand, so it has advantages of both compiled languages and scripting languages. It is useful to compensate performance loss in scripting languages, which is the biggest drawback in Python.

4 DEEP LEARNING IN COMPUTER VISION

Small neural networks are good enough to be useful, but larger networks can excel. The real power of neural networks comes from their scales, once they get bigger to accommodate millions and billions of learnable parameters. Deep learning initially gained its fame through computer vision tasks, and luckily our main interest in this thesis is the computer vision. I am going to mention notable deep neural networks and attempt to answer how and what they see.

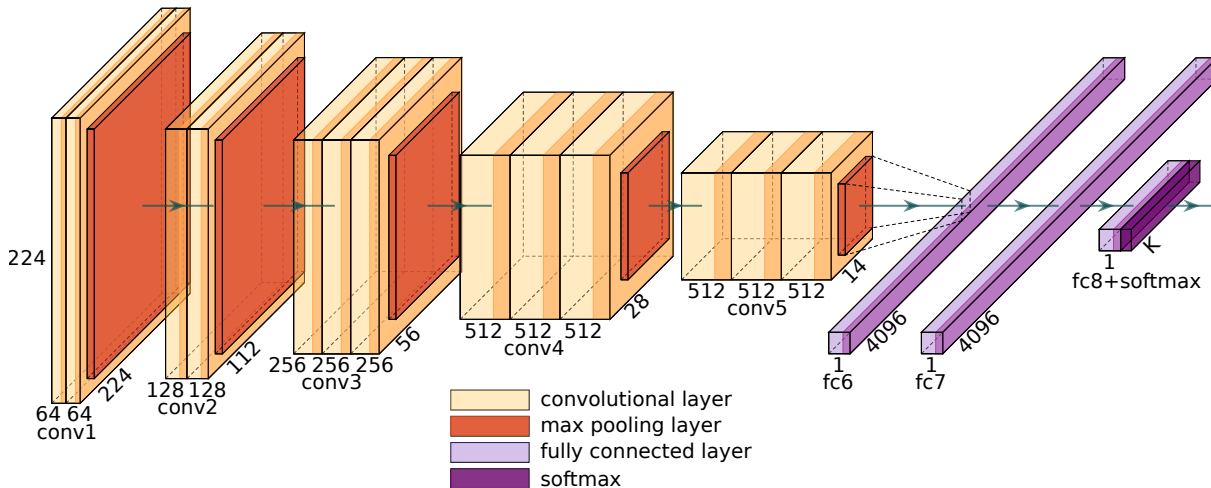


Figure A.16: VGG16[85] architecture. It has 16 layers in total, not counting max pooling layers, ReLUs, and softmax at the end. Dimmed area in convolutional layers and fully connected layers indicates ReLU activation. The input is an image whose size is 224×224 pixels and three channels. The output size remains a variable for K categories of classification task. In my opinion, VGG is the “pure” deep neural network which consists of simple types of layers. Deep neural networks after VGG introduced custom modules to go deeper. The figure is created using codes from PlotNeuralNet[234] and the legend is added.

4.1 DEEP NEURAL NETWORK AND DEEP LEARNING

The term “deep”, used for deep neural network and deep learning, is said to be coined from the DEEP BELIEF NETS[235] by G. Hinton³³. However, it was not until ALEXNET[18]³⁴ in 2012 that deep learning really took off. ALEXNET won the famous ILSVRC (ImageNet large scale visual recognition challenge) in 2012 by a large margin against the second place. His network, clearly named after his name, had 8 weight layers (or trainable layers), which was deep enough to be called “deep” at the time being. A lot of people saw the potential of deep neural networks or deep learning and started developing larger and deeper networks as well as hardware and software to facilitate researches.

Deep vision models could emerge thanks to large-scale datasets. In particular, ImageNet dataset[62] used for ILSVRC 2012 played an important role. ImageNet-1K ImageNet

³³ G. Hinton won Turing Award in 2018 together with J. Bengio and Y. LeCun for their contribution to deep learning, which is considered as “Nobel Prize of Computing”.

³⁴ The primary author Alex Krizhevsky was a PhD candidate supervised by G. Hinton at the moment and also created famous CIFAR-10 and CIFAR-100 datasets in 2009.

dataset provides over a million of annotated images for 1,000 categories³⁵ and resulted in novel neural architectures that are still frequently used and inspired more advanced ones. Followed by ALEXNET which had 8 layers, VGG[85] had 16 layers and up to 19 layers (see figure A.16). Though after VGG, it seemed that simply putting more layers would not do magic (see equation A.10), and some tricks were needed to go deeper. GOOGLNET[236], also known as INCEPTION v1³⁶, devised, what they called, an inception module and achieved 22 layers. RESNET[49] invented what's called residual block and was particularly impressive, because it reached up to 152 layers. WRN (wide residual network)[150] suggested that it is also important to go wider. Concurrently, researchers started investigating efficiency and redundancy of deep neural networks and looking for a way to use them on small devices, for instance, our smartphones. MOBILENET[239, 190, 240] and EFFICIENTNET[238, 241] have a lot of layers, but effectively reduced the number of parameters to fit themselves into small devices. The current state-of-the art CNN as of 2022 is CONVNEXT[151] which did not go deeper than others but compiled advanced modules and layers, and even suggested learning methods too to *modernize* vision models.

But why deeper? Because the deeper the network, the more capacity you can afford. The hype was real and researchers could see visible improvement simply by doing so, telling how big ImageNet dataset was as well as how scalable neural networks were. In addition, their ability to generalize seemed unprecedented. The error rate for ILSVRC already reached below that of humans, supposedly[149]³⁷, and it kept going further down. Fundamentally, it led to the question: how and what do they generalize?

generalize

4.2 DEEP CONVOLUTIONAL NEURAL NETWORKS

All the deep neural networks that I introduced above are, in fact, convolutional neural networks (CNNs or ConvNets). I will focus on its core component, the convolutional layer which I covered for recognizing handwritten digits inside section 1.4 to give my answer how deep CNNs achieve a generalized vision.

Receptive field As we stack multiple convolutional layers together, deeper layers have more and more links with all the previous layers, as depicted in figure A.17. Naturally, a neuron deep in a network receives more signals through all its previous layers than one right after an input layer. Receptive field tells the amount of signals that a neuron receives. Difference in the size of receptive field creates levels of feature maps and of abstraction of

receptive
field

feature
maps

abstraction

³⁵ To this date, there are two most used subsets: ImageNet-1K and ImageNet-22K. ImageNet-1K was what was used for ILSVRC in 2012. ImageNet dataset had been growing ever since it was introduced around 2010, and stopped growing and reached to ImageNet-22K (or ImageNet-21K because the number is slightly less than 22K) with nearly 14 million images.

³⁶ The title of the paper was “Going deeper with convolution” and was named after an internet meme “We need to go deeper” inspired by the movie *Inception* by Christopher Nolan from 2010. I enjoy seeing such titles that *Google* sometimes puts out, because they feel friendly and less daunting. “Attention is all you need”[102] is another and there is also “rethinking” series[96, 237, 238] that improved INCEPTION models and EFFICIENTNET.

³⁷ Human accuracy depends on limited knowledge of individuals. For example, I do not know all the dog breeds and would fail many of them because ImageNet has images of dog breeds that I have never heard of. However, it is not only about an error rate. It is also important to consider how fast you can finish the task. If I were assigned with 100,000 images to test, which is the size of the testing set of ImageNet-1K, I would probably stop after 100 images or so, spending about an hour, and then give random labels for the rest and go home.

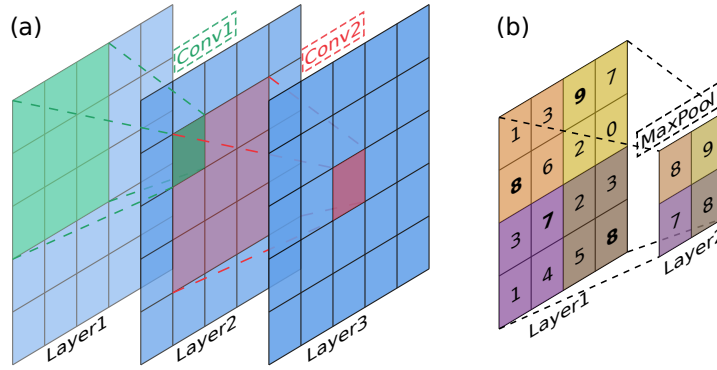


Figure A.17: Going deeper means larger receptive fields. (a) Two layers of convolutional layers that have 3×3 kernels and strides of 1. (b) Max pooling layer is simple and useful to reduce the field of a previous layer as well as the size. It picks a maximum value within its kernel. This max pooling layer has a 2×2 kernel and a stride of 2.

data in a trained CNN. A common strategy of building a CNN is to gradually reduce the spatial size of inputs while increasing the number of channels as go deeper. This strategy is well suited for the object classification task, where we want to reduce a group of pixels, an image, down to a single vector, or a one-hot-encoded answer. The figure of VGG16 [A.16](#) demonstrates it well. Pooling layers are often combined with convolutional layers to pool down the spatial size (width and height) of feature maps, because they are intuitive and cheap to compute. Max pooling layer is a popular choice because it also reduces the amount of computation for backpropagation by picking only the maximum values.

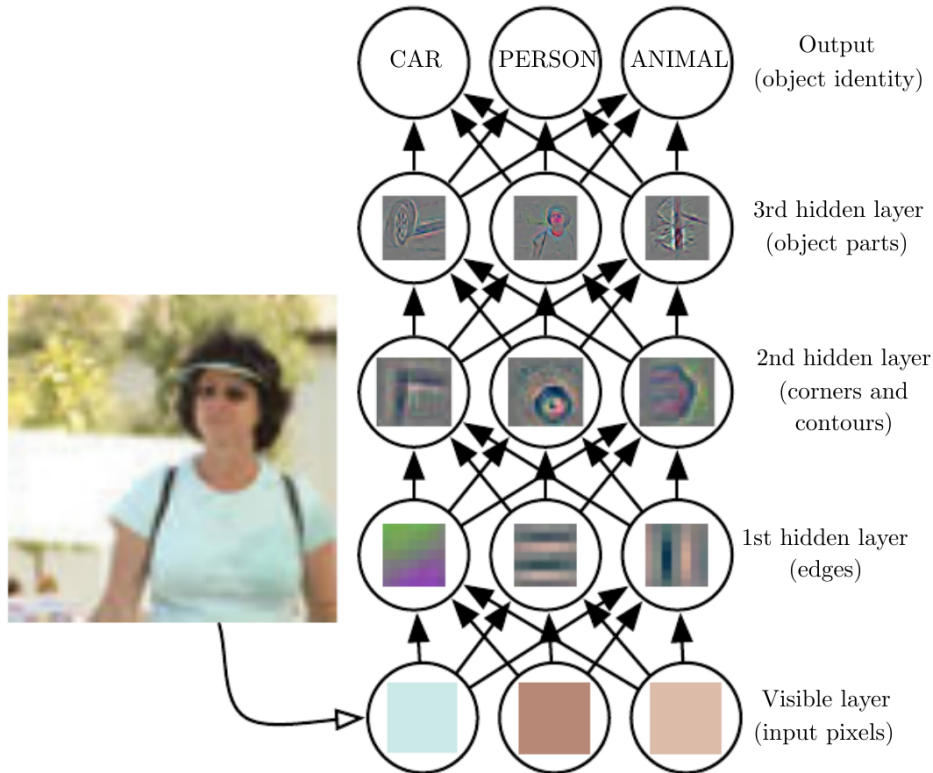


Figure A.18: Illustration of a deep neural network. This figure is taken from *Deep learning*, I. Goodfellow et al. 2016[227], and in turn, it was an adaptation of [140]

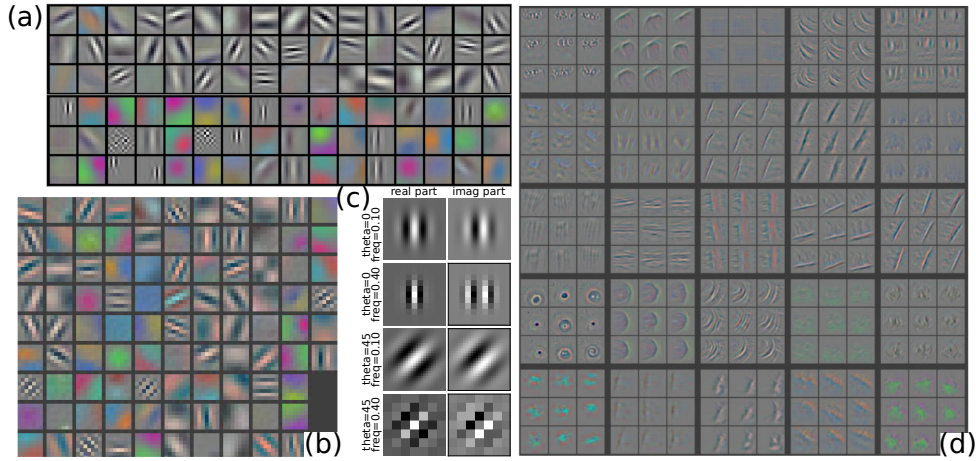


Figure A.19: Visualization of kernels of convolutional layers. (a) Sources: (a)[18], (b, d)[140], (c) was generated using [16].

What and How do CNNs see? Frankly speaking, I do not have a definite answer. What and how CNNs see is hard to tell, and is still and will always be a big research topic. I am going to demonstrate a few approaches and related researches. **First**, we could visualize weights themselves. **Second**, we could inspect direct outputs of each convolutional layer. **Third**, we could indirectly find out what's encoded in each layer.

Visualizing kernels is not the best way because kernels of convolutional layers are usually spatially as small as from 3×3 to 11×11 , and they have a few tenths or hundreds of channels which makes it difficult to visualize them³⁸. But at least, weights of the first convolutional layer have three channels (assuming that the input has three channels), and we could easily visualize them. Figure A.19 (a, b) shows 96 kernels, whose size is $11 \times 11 \times 3$, from the first convolutional layer of a trained neural network[18, 140]. They found that some of them actually looked like existing image filter, for instance, Gabor filters ((c) in the same figure A.19) which are used for analyzing textures. Kernels of the second layer, (d), also shows consistent visuals. Though it is hard to precise what each layer is doing, it is generally accepted that early layers capture low-level features, such as edges and corners, and deeper layers capture high-level features, like bigger visual appearances and also semantic features, as shown in figure A.18.

low-level
features
high-level
features

Inspecting direct outputs could be more intuitive even though it is still not easy to visualize feature maps having many channels. Figure A.20 shows an experiment that it covered a certain area in the input image and observed changes in the activation of a feature map and those in the final prediction[140]. They wanted to know whether networks are truly able to read the context and tell positions of objects. They constructed a CNN with 8 layers, where the 5th one is the last convolutional layer. We could imagine that feature maps from this layer contain coarse and object-level features. In figure A.20 (b), when they occluded a part of the given images, they found a drop of activation signals in certain regions, not necessarily the region they covered. We could consider this region what the network thought the most important feature in the image. For examples,

³⁸ By definition, digital images can have limited numbers of color channels: one channel, three channels, or four channels. One channel image has a gray scale that only tells brightness. Three channel ones usually represent RGB. Fourth channel can optionally indicate amount of transparency.

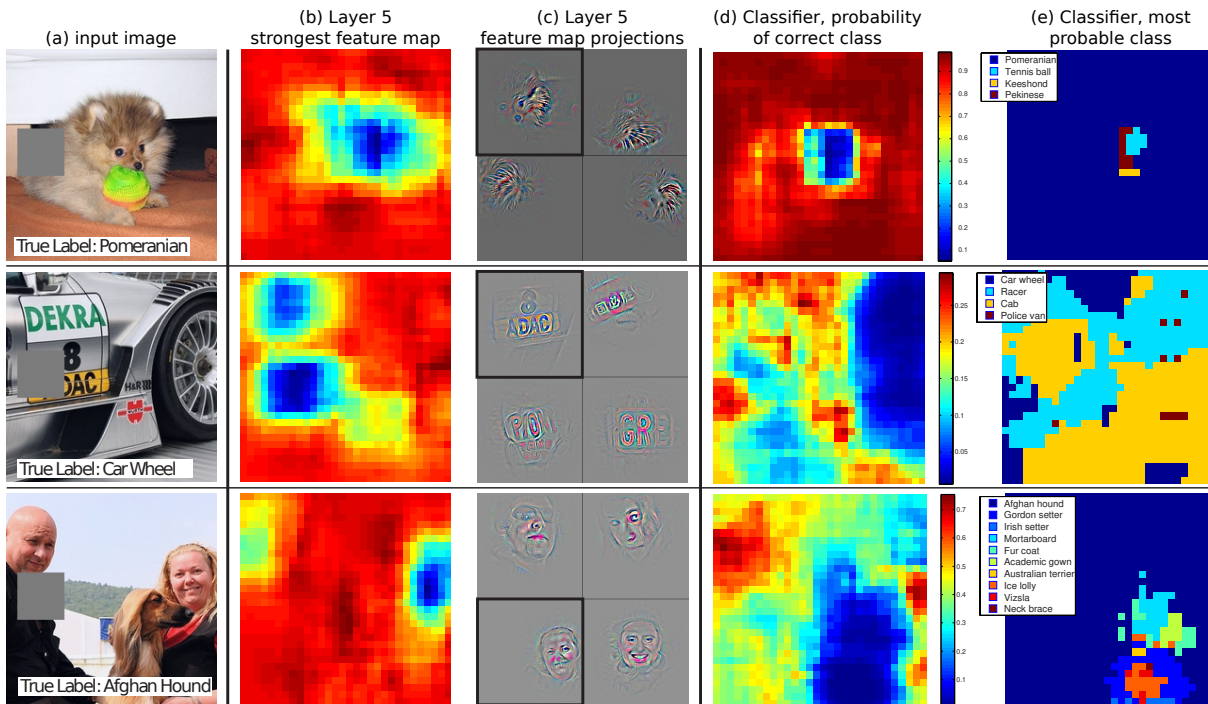


Figure A.20: An experiment to observe responses of feature maps of a CNN, sourced from [140]. (a) Three input images. Gray boxes were used to occlude appeared regions for (b) and (c). (b) Feature maps that showed the strongest response to the input from the 5th layer. (c) Visualization of feature maps of the 5th layer for the input (black squared ones) and three other randomly selected images. (d) Changes in probability for the correct class as a function of occlusion position. (e) The most probable class as a function of occlusion position. Note that this network had 8 layers in total where the 5th layer was the last convolutional layer.

when they covered a part of a pomeranian in the first row (a), the activation of the face went down, which we could assume the most important feature in the image. Indeed, when they covered the face, the probability for the correct label dropped significantly, as appeared in (d). However, sometimes the most important features have negative correlation with the label, as in the third row. When they covered a face, which seemed to be the most important feature, the network was less confused to predict the afghan hound. Overall, this experiment confirmed the hypothesis that deeper layers contain larger features based on receptive field, and more importantly, managed to visualize importance of features. Yet, the visuals of feature maps by themselves, for example (b), were still hard to interpret. Projecting feature maps to the image space seemed intuitive and easy for us, humans, to understand, just like (c) showed rough but interpretable images.

Indirectly decoding layers is proven an effective way to get intuition what and how deep CNNs see the image. The task becomes basically an inverse problem to figure out how an image looks like when we start from a convolutional layer. To answer this question, a group [141] took the pre-trained ALEXNET [18] and trained another CNN to reconstruct or optimize images from the codes encoded through the ALEXNET. See the results in figure A.21. Notice that reconstructed images do not look real, but for the model they trained, all six images are the same, since they optimized it so. This approach to optimize visuals is commonly called feature visualization and soon became a main tool to investigate how individual layer or neuron works.

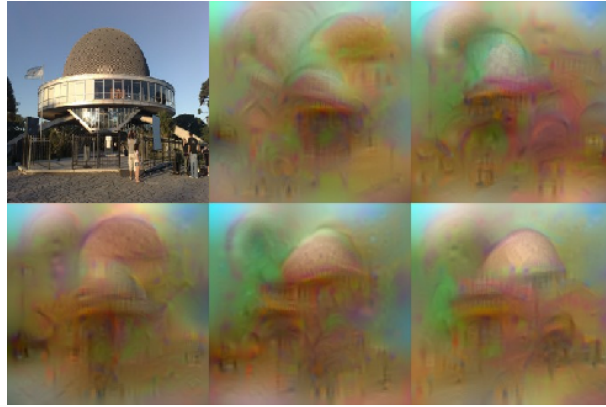


Figure A.21: What’s encoded by a CNN? The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN[18] (before the softmax is applied) trained on the ImageNet data[62]. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen. Caption was kept original from the source[141].

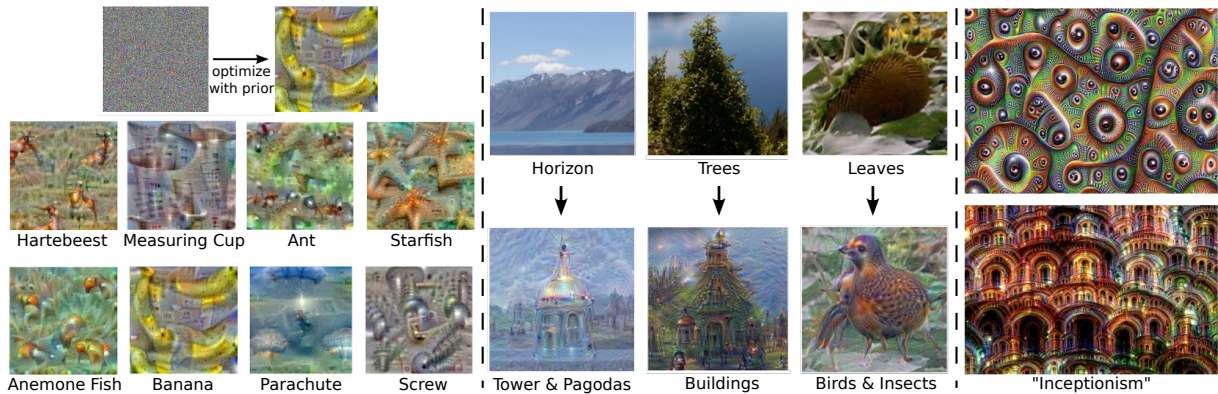


Figure A.22: Examples, a.k.a. dreams, of DeepDream. (left) Starting with a random noise, DeepDream enforces a certain label. (middle) Starting with an image, it overrides it with other labels. (right) Make a fractal-like image by iteratively zooming and applying DeepDream. Source: the original blog post[242] and its “inceptionism” gallery.

DeepDream[242] was one of them and pushed the feature visualization approach further than ever, not just that it was intuitive, but also that the resulted images were intriguing enough to attract a lot of attention. Check them out for yourself in figure A.22. What they did was to pick a layer and keep telling it “I want more of {something}”. For examples, you could start with a noise and tell how a banana looks like. Or you could start with an exiting image and alter it to {something}, so that the network think that it is {something}. Or make an infinite loop enhancing the same target again and again by cropping and zooming within an image to end up with a fractal-like image, which they called “inceptionism”³⁹.

Feature visualization approach did not stop at the level of layers and went down to individual channels and neurons[243]. Figure A.23 displays selected sets of images optimized through channels in convolutional layers. Even though it was still challenging

³⁹ Soon after DeepDream was revealed, it was used for NSFW (not safe for work), in particular nudity, because of its bizarre looking results, and you know, *because of people*. For instance, an inception of penises was a popular thing (do not ask me why).

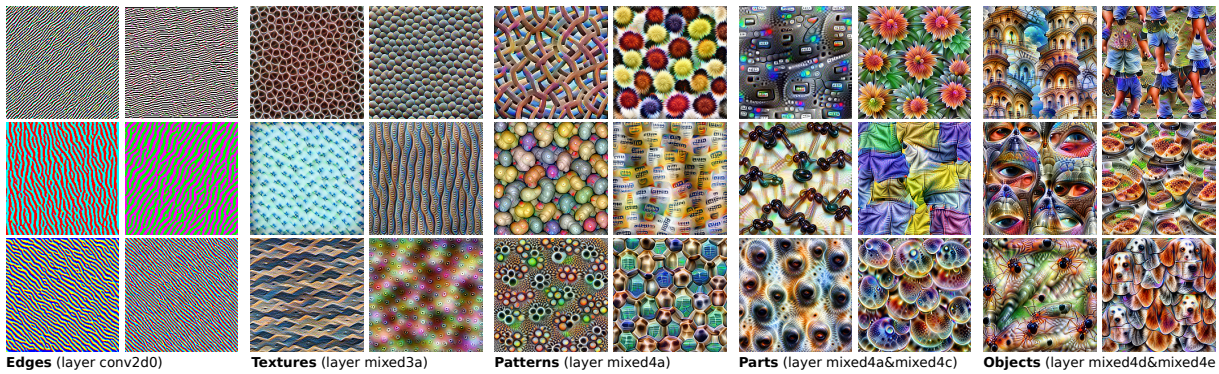


Figure A.23: Feature visualization of some channels in selected layers. Layer names are denoted at the bottom. Note that the CNN they used was GOOGLNET[236]. Visit this website <https://microscope.openai.com/models> to explore more feature visualization from various neural networks. Source: [243].

to interpret, they found that resulted images were visually rich, and it was an intuitive way to have a glimpse of how components of CNNs beyond layers were working. The results clearly showed that each channel was distinctive, as intended when we constructed a convolutional layer with biases. For instance, some units preferred certain directions when enhancing edges. Some units enhanced polka-dot like textures while others enhanced stripes, and so on.

APPENDIX B

LIST OF DATASETS

The following table in the next two pages lists biomedical datasets that were either mentioned or used in experiments.

Note that

- not all the datasets were used
- only subsets were considered for some datasets for individual reasons
- find table annotations ($a,b,1,2,3$) at the end of the table
- links in the source column work only when viewed in the digital format
- APIs of most datasets listed here are implemented in `bioimagerloader`

Table B.1: Collected datasets (page 1/2)

Id	Acronym	Resolution	Channel	Number ¹	Dtype	Format	Annotation ²	Description	Full name	Source
1	DSB2018	(~300,~300)	1,3	735 (670/65)	uint8	PNG	○	Kaggle; Mixture of images here and there	Data Science Bowl 2018	link^a [58]
2	TNBC	(512,512)	3	50	uint8	PNG	○	H&E; Triple Negative Breast Cancer (TNBC)	Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map	link^a [21]
3	ComPath	(1000,1000)	3	30	uint8	PNG	○	H&E; Instance segmented, dense	A Dataset and a Technique for Generalized Nuclear Segmentation for Computational Pathology	link^a [67]
4	UCSB	(768,897)	3	58	uint8	TIF	△	H&E; Partially annotated (benign, malignant)	A biosegmentation benchmark for evaluation of bioimage analysis methods	link^a [65]
5	DigitPath	(2000,2000)	3	141	uint8	TIF	△	H&E; Partially annotated	Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases	link^a [66]
6	S-BSST265	(430,550) -(1024,1360)	1,3	79 (42/37)	uint8	TIF	○	IF images; Designed for ML	An annotated fluorescence image dataset for training nuclear segmentation methods	link^a [92]
7	CRCHisto	(500,500)	3	100	uint8	BMP	△	H&E, Only center coordinates for each cell; Cells are quite small	Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images	link^a [244]
8	MurphyLab	(1024,1344), (1030,1349)	1	100	uint8	PNG	○	Two annotation formats; Photoshop and GIMP, 97 segmented images	Nuclei Segmentation In Microscope Cell Images: A Hand-Segmented Dataset And Comparison Of Algorithms	link^a [245]
9	FRU-Net	(2048,2048)	1	72	uint16	TIF	○	TEM images	FRU-Net: Robust Segmentation of Small Extracellular Vesicles	link^a [246]
10	Cellpose	(383, 512)	2	100 (89/11)	uint8	PNG	○	Only biologically relevant subset (a.k.a. “specialized images”); nuclear and whole-chell channels and annotation	Cellpose: a generalist algorithm for cellular segmentation	link^a [29]
11	LIVECell	(520, 704)	1	5,239 (3,727/1,512)	uint8	TIF	○	Bright field images	LIVECell—A large-scale dataset for label-free live cell segmentation	link^a [64]
12	TissueNet	(512, 512) (256,256)	2	6,990 (2,601/3,140/1,249)	float32	TIF	○	Nuclear/whole cell annotation. 2M in total 1M for each; Version1	Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning	link^a [43]
13	BBBC006	(520,696)	2	768	uint16	TIF	○	z-stack; z=16 is in-focus ones; sites (s1, s2); w1=Hoechst, w2=phalloidin	Human U2OS cells (out of focus)	link^a [55]
14	BBBC007	(400,400) -(512,512)	2	16	uint8	TIF	○	Outline annotation	Drosophila Kc167 cells	link^a [55]
15	BBBC008	(512,512)	2	12	uint8	TIF	○	F/B semantic segmentation	Human HT29 colon-cancer cells	link^a [55]
16	BBBC018	(512,512)	3	56	uint8	DIB	○	Outline anno; One missing annotation (55)	Human HT29 colon-cancer cells (diverse phenotypes)	link^a [55]
17	BBBC020	(1040,1388)	2	25	uint8	TIF	○	Cell & Nuclei annotation; 5 missing annotations	Murine bone-marrow derived macrophages	link^a [55]
18	BBBC004	(950, 950)	1	100	uint8	TIF	○	Synthetic cells; five subsets with increasing degree of clustering	Synthetic cells	link^a [55, 146, 147]
19	BBBC039	(520,696)	1	200	uint16	PNG	○	Annotated samples from BBBC022; May have some overlap with DSB2018	Nuclei of U2OS cells in a chemical screen	link^a [55]
20	BBBC009	(600, 800)	3	5	uint8	TIF	○	DIC images; 3 channels (R, G, B)	Human red blood cells	link^a [55]
21	BBBC030	(1032, 1376)	3	60	uint8	TIF	○	DIC images; 3 channels (R, G, B)	Chinese Hamster Ovary Cells	link^a [55]

Table B.2: Collected datasets (page 2/2)

Id	Acronym	Resolution	Channel	Number ¹	Dtype	Format	Annotation ²	Desc	Full name	Source
22	BBBC041	(1200,1600), (1383,1944)	3	1,364 (1,208/120)	uint8	PNG, JPEG	Δ (U ²)	Bounding box annotation; 36 bbs missing (1364-(1208+120)=36)	P. vivax (malaria) infected human blood smears	link^a [55]
23	BBBC002	(512,512)	1	50	uint8	TIF	X (C ²)	Might overlap with DSB2018	Drosophila Kc167 cells	link^a [55]
24	BBBC042	(708,990)	3	1,117	uint8	TIF	X (U,B ²)	Bounding box annotation; partially annotated	Rat astrocyte cells	link^a [55]
25	BBBC013	(640,640)	2	96	uint8, uint16	BMP, FRM	X (B ²)	Cytoplasm	Human U2OS cells cytoplasm–nucleus translocation	link^a [55]
26	BBBC014	(1024,1360)	2	96	uint8	BMP	X (B ²)	Second channel is usually very clear with a few artifacts	Human U2OS cells cytoplasm–nucleus translocation	link^a [55]
27	BBBC015	(768,1000)	2	144	uint8	JPEG	X (B ²)	2 channels (Green, Crimson); texture in green channel	Human U2OS cells transfluor	link^a [55]
28	BBBC016	(512,512)	2	72	uint8	TIF	X (B ²)	2 channels (G,B)/ cells are Blue	Human U2OS cells transfluor	link^a [55]
29	BBBC021	(1024,1280)	3	720 (132,000) ³	uint16	TIF	X (B ²)	HUGE dataset; 3 channels; DAPI(w1), Tubulin(w2), Actin(w4); Only 720 images were used	Human MCF7 cells – compound-profiling experiment	link^a [55]
30	BBBC026	(1040,1392)	1	864	uint8	PNG	X (B,C ²)	Only centers are annotated for 5 images	Human Hepatocyte and Murine Fibroblast cells – Co-culture experiment	link^a [55]
31	LOB-THG	(512, 512)	2	14	uint16	TIF	○	LOB in-house; THG and GFP channels; mouse cortex; cropped and tagged	Mouse brain cortex THG and GFP images	LOB ^b
32	LOB-P14	(512, 512)	3	40	uint16	TIF	○	LOB in-house; Brainbow + ChroMS; cropped and tagged	Mouse brain cortex using ChroMS and Brainbow	LOB ^b
33	LOB-TCYT5	(512, 512)	3	14	uint16	TIF	○	LOB in-house; Brainbow + ChroMS; cropped and tagged	Mouse brain cortex using ChroMS and Brainbow	LOB ^b
34	LOB-MNTB	(512, 512)	3	41	uint16	TIF	○	LOB in-house; Brainbow + Confocal; cropped and tagged	Mouse brain auditory neurons; medial nucleus of the trapezoid body (MNTB) using confocal and Brainbow	LOB ^b
-	Sum	-	-	19,303	-	-	-			

Table annotation:

¹ numbers in parentheses indicate training/testing split provided by the dataset (number of training, number of testing).

² ○: provide complete segmentation mask targets, Δ : partially annotated masks, X: do not provide mask targets; C: counts, B: biological labels, U: bounding boxes

³ BBBC021 is such a huge dataset, but all the images look more or less the same. Only a portion was taken.

^a link is only available in digital format

^b Laboratoire d’optique et biosciences; in-house data

APPENDIX C

METRIC

A metric measures performance of machine learning models. Following metrics concern computer vision tasks, in particular, the semantic segmentation or the instance segmentation. I recommend this comprehensive review paper[\[134\]](#) to learn more.

JACCARD INDEX OR IOU

Jaccard index is identical to IoU (intersection over union). But when it is used in the context of object detection, it also considers a matching threshold. Say R is a group of pixels to a reference and P is for a prediction. Jaccard index (\mathcal{J}) is:

$$\mathcal{J} = \frac{|R \cap P|}{|R \cup P|} \quad (\text{C.1})$$

In object detection, a matching of an object is counted when the intersection, or the overlap, is greater than a certain ratio of the reference (R), say 0.5. For objects that do not meet this condition, the index value is set to 0.

$$\mathcal{J}_n(R_n, P) = \begin{cases} \mathcal{J}_n(R_n, P) & \text{if } |R_n \cap P| > 0.5|R_n| \\ 0 & \text{else} \end{cases} \quad (\text{C.2})$$

The final value is the mean of Jaccard indices for all reference objects.

$$\mathcal{J}_{object} = \frac{1}{N} \sum_{n=1}^N \mathcal{J}_n(R_n, P) \quad (\text{C.3})$$

ACCURACY (VER. DSB2018)

The famous data science bowl 2018 competition (DSB2018), which was hosted on online platform *Kaggle*, defined their own metric for instance segmentation task. I will simply call this metric “accuracy”, which has a rather generic name. But this is the only metrics with this name in this thesis, so I will use “accuracy” as a proper noun. It is similar to an object detection variant of Jaccard index. However, instead of averaging indices, accuracy counts numbers of true positives (TP), false positive (FP), and false negatives (FN) given a IoU threshold (t).

$$\text{accuracy}_t = \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \quad (\text{C.4})$$

The threshold values ranged from 0.5 to 0.95 with a step size of 0.05: ($t = 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95$) in the actual competition. The **final metric** is an averaged value across the thresholds, where $|T|$ indicates the number of threshold values.

$$\text{accuracy} = \frac{1}{|T|} \sum_t \frac{TP(t)}{TP(t) + FP(t) + FN(t)} \quad (\text{C.5})$$

As a side note, DSB2018 called their metric “mean average precision” (mAP), but it is a misnomer. mAP that researchers are familiar with is different from their “accuracy”. The average precision (AP) metric that researchers know was recognized by Pascal VOC[224], where “average” meant averaging over categories because Pascal VOC was a classification task. But in fact, DSB2018 has nothing to average because it is an instance segmentation task (Technically, it is not false though, because they have only one category. So it is the same as doing nothing). Moreover, precision is defined as $TP/(TP + FP)$, but DSB2018 added FN in the denominator, thus they should not have called it precision in the first place. “mean” was added later to AP and meant a mean over multiple IoU threshold values. Mean average precision (mAP) is used in datasets such as MS COCO[137].

F-1 SCORE

F-1 score is a common metric in both binary semantic segmentation and instance segmentation tasks. It is defined as a harmonic mean of precision and recall. Precision is a ratio of true positives over positives. Recall is a ratio of true positives over trues. When it is used for instance segmentation, it is common to consider IoU with threshold 0.5 for each instance and count the numbers of TP, FP, and FN, same as Jaccard index.

$$\text{precision} = \frac{TP}{TP + FP} \quad (\text{C.6})$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = \frac{2TP}{2TP + FN + FP} \quad (\text{C.7})$$

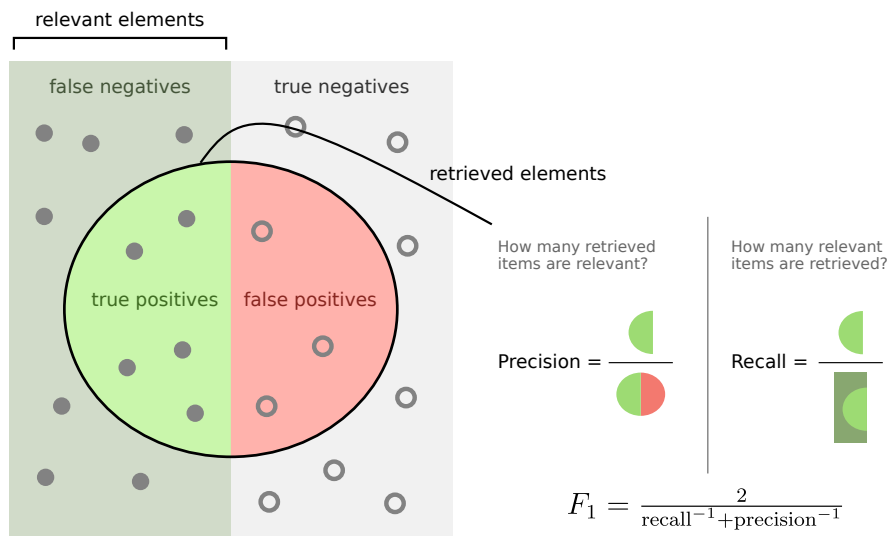


Figure C.1: F-1 score is a harmonic mean of precision and recall. Image source: [247]

APPENDIX D

[PUBLICATION] NU-NET

This is a paper for NU-Net presented in chapter 3, which has been accepted to BIC (Bio-image Computing) workshop at ICCV (International Conference on Computer Vision) 2023. It was streamlined for the workshop paper and included additional experiments compared to what was introduced in this manuscript.

NU-Net: a self-supervised smart filter for enhancing blobs in bioimages

Seongbin Lim

Laboratoire d'Optique et Biosciences, CNRS, INSERM, École Polytechnique, Institut Polytechnique de Paris
91128 PALAISEAU Cedex, France

sungbin246@gmail.com

Emmanuel Beaufreire

emmanuel.beaufreire@polytechnique.edu

Anatole Chessel

anatole.chessel@polytechnique.edu

Abstract

While supervised deep neural networks have become the dominant method for image analysis tasks in bioimages, truly versatile methods are not available yet because of the diversity of modalities and conditions and the cost of re-training. In practice, day-to-day biological image analysis still largely relies on ad hoc workflows often using classical linear filters. We propose NU-Net, a convolutional neural network filter selectively enhancing cells and nuclei, as a drop-in replacement of chains of classical linear filters in bioimage analysis pipelines. Using a style transfer architecture, a novel perceptual loss implicitly learns a soft separation of background and foreground. We used self-supervised training using 25 datasets covering diverse modalities of nuclear and cellular images. We show its ability to selectively improve contrast, remove background and enhance objects across a wide range of datasets and workflow while keeping image content. The pre-trained models are light and practical, and published as free and open-source software for the community. NU-Net is also available as a plugin for Napari.

1. Introduction

Machine learning solutions, especially those based on deep neural networks, have had huge impact on bioimage analysis. Very successful segmentation models, in particular, have been introduced, starting with U-Net [32]. Over the years, semantic segmentation task evolved into instance segmentation with its ability to address touching boundaries of nuclei and cells and identify instances, such as StarDist [34], nucleAIzer [17], Cellpose [37], CDNet [15], and Mesmer [14].

These segmentation models were essentially based on supervised learning, resulting in capable and performant models once sufficient annotated data were provided. How-

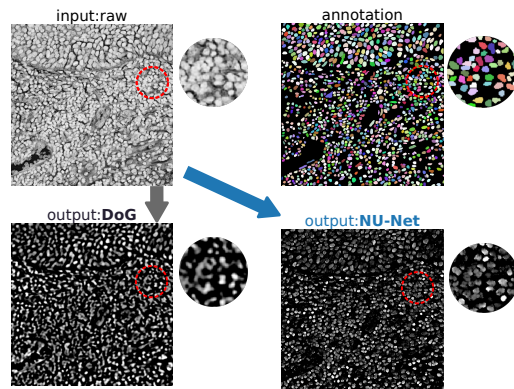


Figure 1. Proposed model, NU-Net, is a nuclear and cellular filter built on convolutional neural network and self-supervised learning scheme via perceptual losses, to be used in place of linear filters. This example compares NU-Net to difference of Gaussians (DoG). Sample image is a histopathology image containing dense cells and is sourced from [26]. (The sample image was converted into gray-scaled).

ever, in biological imaging, each experiment is different, with varied image modality or sample type, condition and labeling, making the design of a truly generic pre-trained model complicated. In practice, it is common in analysis pipelines to annotate a few local data again, which is a time-consuming process needing expertise, and to retrain or fine-tune these models. Alternatively, users resort to less demanding machine learning software, compromising performance, e.g. ilastik [5] and LABKIT [3], to interactively acquire a model, usually built on random forest classifier.

One very common way to circumvent this issue, in particular, since machine learning expertise is not always available in experimental labs, is to use the same set of classical linear filters in increasingly complex pipelines. Use cases include a quick and light way to adjust contrast, a post-acquisition step to denoise, enhance images, or clean their

background or a pre-analysis step to detect/segment/count objects of interest. Linear filters are reliable and well proven over time, but most of them are parametric algorithms with strong assumption, meaning needs of parameter search and low versatility, respectively.

In this paper, we propose a novel filter, we call NU-Net (Not a U-Net and read new net), built on convolutional neural networks (CNNs) and self-supervised learning scheme. NU-Net enhances contrast of images based on morphological traits of nuclei and cells perceived by a large pre-trained vision model. NU-Net's behavior is positioned between a binary semantic segmentation model that classifies foreground and background pixels and an autoencoder that reconstructs input images as accurately as possible. During training, it learns morphological perception as well as a generative loss and does not require paired target labels as in supervised learning. This self-supervision advantage can address highly diverse nature of biological images across many modalities because one can include as many images as suited regardless of their target label availability. Also, it makes NU-Net highly adaptable to local images and practical in real application.

To leverage its training benefit, we gathered 25 nuclear and cellular datasets with or without target labels and trained NU-Net with this meta dataset to promote it as a generalized filter that can cover a large range of different modalities of nuclear and cellular images. We kept the architecture of NU-Net rather small, so it can be used on a local CPU without heavy computing power or reliance of graphical processing units (GPUs). In addition, training NU-Net from scratch can be done within an hour with a proper setup, if needed.

Evaluation of NU-Net is complicated by the fact that it is not directly comparable to other recent work and thus cannot reuse existing metrics and benchmarks. Since the main objective of this work is a practical one, adding a new tool to the bioimage analysis toolbox, we focused on integrating NU-Net into actual object detection workflow, to show its usefulness in practice, as well as try it on a wide range of images to show its versatility.

Contributions of this work can be summarized as followed:

- introducing a novel filter built on machine learning and CNN, which can replace classical linear filters
- using perceptual losses for self-supervised learning to circumvent supervised learning and avoid costly labeling process in biological images
- gathering 25 different nuclear and cellular datasets and training NU-Net to cover large range of modalities

1.1. Related work

Perceptual losses were used effectively for neural style transfer application [22]. They take advantage of a pre-trained large vision model such as VGG [35] and its perception or feature vectors from certain neural layers. The losses reconstruct input images given features or details that each layer keeps. In neural style transfer, there are two losses in general [11, 10, 19]: content loss and style loss. The content loss is a generative loss to preserve content and the style loss is a modified generative loss using Gram's matrices that attempts to reconstruct a certain style.

To seek generalizability in machine learning models, foremost one needs diverse sources of data. Many works focused on segmentation tasks. In biological images, Cellpose [37] is an instance segmentation model that transforms target masks based on diffusion process, and for training they scraped nuclear and cellular images from multiple sources including internet to make their models more generic. NucleAIzer [17] is another instance segmentation model that defined a set of modalities and combined simulation and neural style transfer (pix2pix [20]) to augment training data. TissueNet [14] is a dataset that provides about 2.5M of annotations for tissue images across 9 organs and 6 different modalities to support general-purpose tissue segmentation models.

For machine learning applications beside segmentation tasks, image restoration and denoising models turned out to be practical and effective. CARE [40] is a super-resolution model that learns how to restore poorly resolved microscopy images from pairs of high and low quality images. Noise2Noise [27] demonstrated denoising MRI images as a supervised model. To avoid the necessity of having paired labels, Noise2Void [25] proposed self-learning model to denoise images. Self-learning approach made it easy to apply Noise2Void model to any images since it does not need paired images nor annotations.

As supervised learning approach remains difficult in biological images in practice, classical filters are still frequently used as off-the-shelf methods in many biomedical image analysis pipelines. For contrast enhancement, Gamma and logarithm adjustment methods are the simplest, yet remain versatile and available in most software tools. More elaborated approach is to remove background, that is to identify foreground and background pixels, often with morphological assumption. DoG (difference of Gaussians) and LoG (Laplacian of Gaussian) are such filters that consider blobs as objects of interest or foreground pixels given univariate Gaussian kernels. These linear filters are not dependent on data unlike machine learning approach and thus universally applicable. However, they are rigidly bound by their strong assumption and prone to visual artifacts and errors. Despite being thoroughly characterized mathematically, their use is still very much empirical and problem

specific. DoG and LoG are popular choices for object detection or segmentation task when it comes to bioimage analysis and widely adopted for their easy implementation to many applications and software, as in Fiji [33], ilastik [5], TrackMate [38], BigStitcher [18] and so on.

2. Method

We will describe losses, training scheme and architecture of NU-Net, a self-supervised nuclear and cellular filter we propose. In addition, we will introduce 25 datasets that we used and the concept of meta-dataset with which we trained NU-Net.

2.1. Losses

NU-Net uses perceptual losses and the work of Johnson *et al.* [22] as a starting point. They used a pre-trained VGG [35] for perception and demonstrated neural style transfer and super-resolution applications. Neural style transfer applications usually have two losses: content loss and style loss [11, 10, 19, 12, 4].

2.1.1 Content loss

The content loss is a generative loss that attempts to reproduce input data through a single latent feature vector from a certain layer of pre-trained deep neural network. The definition of content loss is shown as in equation 1. The content loss (L_c) optimizes $\hat{\mathbf{y}}$ to reproduce *content* c (which is identical to \mathbf{x} in our setting) based on features at layer F^l of a pre-trained deep neural network. The loss is defined incorporating the mini-batching technique where N denotes a mini-batch size:

$$L_c(\mathbf{x}, \hat{\mathbf{y}}, l) = \frac{1}{N} \sum_{n=1}^N \sum_{i,j} (F_{ij}^l(\mathbf{x}) - F_{ij}^l(\hat{\mathbf{y}}))^2 \quad (1)$$

The choice of layer l has great importance in particular for content loss, because what is considered as *content* is determined largely by convolutional layers and the size of their receptive fields at layer l . In general, the concept of *content* is recognized as a high-level feature, thus the required receptive field should not be too small or too large.

2.1.2 Morphological loss

The morphological loss we propose is an extension of the style loss from [22]. The plain style loss from literatures [11, 10, 22] uses Gram matrix to represent *styles*. Gram matrix being $G^l \in R^{N_l \times N_l}$, the vectorized feature map $F_l \in R^{N_l \times M_l}$ has N_l number of filters and has the size of M_l , where l indicates l -th block inside a CNN: $G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$.

In the context of style transfer applications [11, 10, 22], the morphological loss can be interpreted to generalize a certain style, which has binary values with a set of targets having a consistent morphology. In fact, there have been few attempts to model or generalize one particular style with multiple images, because supposedly a style usually meant one from a single painting instead of a group of the same style of paintings, possibly due to their high complexity. Our idea started from an assumption that generalizing a simple style from multiple images with the same style, as in our case, should be easier. In principle, the morphological loss could generalize any morphologies as long as the given morphologies are consistent. We focused on round shapes or blobs, which can represent nuclei or cells. We will call this morphological prior “blob-mask style”.

Morphological loss is calculated in mini-batch fashion same as for the content loss in equation 1. The idea is similar to that of the memory bank [42], to sample mini-batches from a bank that contains a fixed number of predefined classes. Our bank represents only one style, to be specific blob-mask style. Morphological loss (L_m) is defined as below with N_s referring to a mini-batch size of morphological targets \mathbf{s} , $\hat{\mathbf{y}}$ to an image to be optimized, and \mathbf{s}_n to n -th target within a mini-batch \mathbf{s} :

$$L_m(\mathbf{s}, \hat{\mathbf{y}}) = \frac{1}{N_s N_l} \sum_{n=1}^{N_s} \sum_l (G^l(\mathbf{s}_n) - G^l(\hat{\mathbf{y}}))^2 \quad (2)$$

2.1.3 Total loss

Combining equation 1 and 2, total loss (L_t) is defined as below, where we put linear weights on each loss with w_c and w_m , respectively.

$$L_t = w_c L_c(\mathbf{x}, \hat{\mathbf{y}}, l) + w_m L_m(\mathbf{s}, \hat{\mathbf{y}}) \quad (3)$$

$$L_t \propto L_c + \frac{w_m}{w_c} L_m = L_c + w L_m \text{ where } w = \frac{w_m}{w_c}$$

The fraction (w) between two coefficients play an important role to balance two losses and can bring in practical functionality later in inference time as we will show in the result section.

2.2. Training scheme

Overall training scheme is displayed in figure 2. There are two separate groups of image sources. One is for the content loss and the other is for the morphological loss. We will call the former content images and the later style images for brevity. The content images are a set of raw images whose contents need to be preserved. The style images are also a set of binary images or mask labels that serves two purposes. The primary purpose of the style images is to define “contents” to keep, against what its name suggests, or

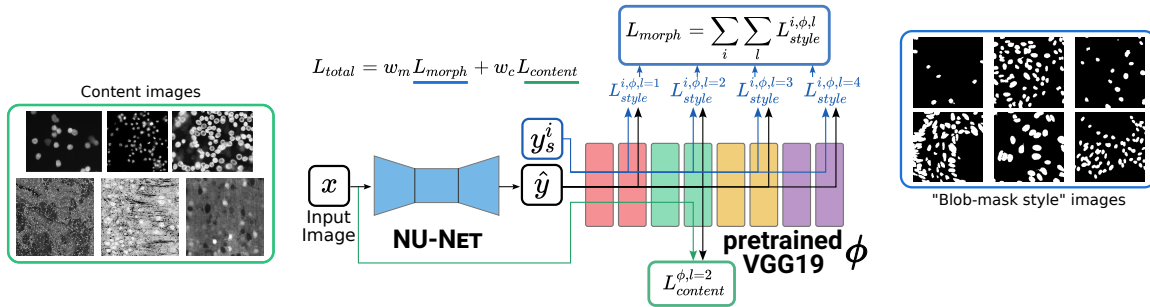


Figure 2. Diagram describing training losses. Total loss consists of two losses: content loss and morphological loss. Both are perceptual losses from a pre-trained large vision model, VGG19 [35] in this case. Note that the content images and the “blob-mask style” images are not paired, neither are the two losses. x denotes input image, \hat{y} output image. y_s^i is a sample target from a mini-batch used for morphological loss. Find details in equation 1, 2 and 3. $l \in \{1, 2, 3, 4\}$ indicates four feature block in VGG19, described in [22].

to apply “blob-mask style” to the content images in the perspective of neural style transfer. The secondary purpose is to maximize contrast between foreground and background. Note that the content images and the style images are entirely separate and not paired unlike a typical supervised learning scheme.

We chose a pre-trained VGG19 [35] as our perception network and employed the same architecture from the work of Johnson *et al.* [22]. The architecture of NU-Net makes use of residual blocks [16] which are important for the purpose because they are effective to preserve input data. We followed the recipe from [22] to calculate perceptual losses. The number of parameter is about 1.7M, which seems small but capable enough to apply complex styles of paintings. NU-Net is light and fast benefiting from small architecture.

2.3. Data

2.3.1 Meta dataset

We wanted to make NU-Net as generic and applicable to many modalities of nuclear and cellular images as possible and promote it as a superior alternative to classical linear filters. Whether it is supervised learning or self-supervised learning, it is imperative to use various sources of data in order to make a generic machine learning model. Massive and well curated datasets such as ImageNet [23] and MS-COCO [29] are great examples that paved the road for computer vision foundation models [6]. It was not until TissueNet [14] that such large dataset appeared for nuclear or cellular images. TissueNet sourced images from 6 platforms and 9 organs, which were unprecedented diversity for a dataset curated for machine learning. However, it cannot cover all types of images due to highly diverse nature of biomedical image modalities, such as our local images of mouse cerebral cortex acquired via THG (third-harmonic generation) microscopy as TissueNet simply does not have the same platform nor the same sample.

To address the issue of lacking diversity of data source in nuclear and cellular datasets, we gathered 24 datasets having nuclei, cells or both, plus one additional local THG dataset. Thus, the total number of gathered datasets was 25 and the total number of images amounted to 12K. They are listed in table 1. Not all of them provide annotation, nor they provide the same types of annotation. However, they are all useful for training NU-Net as long as they contain nuclei or cells in images.

Dealing with the diversity of datasets, each with its format and preprocessing step, across many numerical experiments is a non-trivial MLOps task. To handle it, we used BioImageLoader [28].

2.3.2 Data pre-processing and augmentation

We carried out a set of data pre-processes to have consistent inputs format across different datasets. Due to the nature of microscopy and biological images, in particular of staining techniques, most datasets offer a single channel which tagged nuclei or cells. We selected those channels if we could and converted multichannel images to gray-scaled ones otherwise. Another process was to ensure consistent contrast of foreground and background pixels across datasets. While most datasets have object of interests in higher pixel values than background pixels, a few datasets have the opposite contrast. Therefore, we inverted contrast of these few datasets to match the others. This process turned out to be important since all the target images for the morphological loss had the same contrast.

Balancing volumes of each dataset was an inevitable step. Every dataset is different not only in the number of images, but in resolution of images, size of objects and density of objects. To account for all the datasets rather equally, we designed data augmentation protocols, which consist of random cropping, resizing and flipping for each dataset with appropriate parameters. Then we sampled im-

ID	Acronym	number	Annotation	Ref.	ID	Acronym	number	Annotation	Ref.
1	TissueNet	6,990	○	[14]	14	S-BST265	79	○	[24]
2	BBBC041	1,328	△ (U)	[30]	15	FRUNet	72	○	[13]
3	BBBC026	864	X (B,C)	[30]	16	BBBC016	72	X (B)	[30]
4	BBBC006	768	○	[30]	17	BBBC018	56	○	[30]
5	DSB2018	735	○	[1]	18	TNBC	50	○	[31]
6	BBBC021	240	X (B)	[30]	19	BBBC002	50	X (C)	[30]
7	BBBC039	200	○	[30]	20	ComPath	30	○	[26]
8	BBBC015	144	X (B)	[30]	21	UCSB	58	△	[9]
9	DigitPath	141	△	[21]	22	BBBC020	25	○	[30]
10	MurphyLab	100	○	[8]	23	BBBC007	16	○	[30]
11	Cellpose	100	○	[37]	24	BBBC008	12	○	[30]
12	BBBC013	96	X (B)	[30]	25	LOCAL-THG	14	○	-
13	BBBC014	96	X (B)	[30]	-	Total	12,336		

Table 1. List of nuclear and cellular datasets used to train NU-Net. Annotation availability and type are usually the most important factors when it comes to selecting datasets for machine learning, but not for NU-Net thanks to a self-supervised learning scheme. In total, 25 datasets were used, including a local dataset (LOCAL-THG), and the number of images amounted to 12,336. Symbols for annotation (○: provide complete segmentation mask targets, △: partially annotated masks, X: do not provide mask targets; C: counts, B: biological labels, U: bounding boxes). Note that BBBC021 originally has massive 132,000 images and only a portion was used because data comes from the same platform and would have led huge imbalance.

ages to roughly match the relative sizes of nuclei and cells from one dataset to another. Last but not least, we chose the minimum number of samples among 25 datasets after considering data augmentation, and simply set it as a constant number of samples throughout all the datasets mainly to avoid big bias towards those having larger number of samples. Our rationale was that we did not need huge amount of one modality when our primary goal was to cover many modalities and the architecture was rather small. In the end, we randomly sampled 900 images as inputs for every epoch.

2.3.3 Blob-mask style images

As shown in figure 2, so-called blob-mask style images are merely binary segmentation masks. Therefore, in a sense, NU-Net is a neural style transfer network that mimics behavior of binary segmentation. We selected mask labels of DSB2018 [1], BBBC039 and BBBC006 datasets [30] and used them as targets for the morphological loss. DSB2018 dataset was introduced through a competition on online platform Kaggle and has been used in many computer vision models for biological images [34, 17, 7]. We selected DSB2018 because its wide inclusiveness of data sources. BBBC039 is one of sources of DSB2018 and represents typical cellular microscopy images. We included BBBC006, whose annotation was automated, to show that it is possible to use automated labels to train NU-Net.

3. Numerical experiments

We think of NU-Net as filling a new niche in biological image analysis and as such it does not belong to a clear existing class of deep learning models, like segmentation or image restoration models. Thus, classical metrics and com-

parison are hard to apply. To evaluate it, we will 1) use a simple contrast enhancement metric and compare with classical filters, 2) use a realistic biological image analysis quantitative workflow, to show how it can improve day-to-day bioimage informatics work 3) apply it to a range of openly available images, to qualitatively show its versatility.

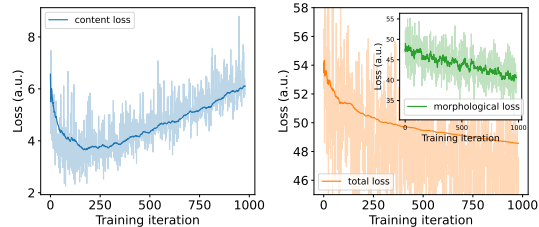


Figure 3. Ideal training loss curves. Total loss is a weighed sum of a content loss and a morphological loss. The desired trajectory would be to have a constantly decreasing morphological loss and a content loss to increase after a decent drop. Note that the curves were from a real case but smoothed out using exponentially weighted moving average (EWMA) to demonstrate the ideal forms. Transparent lines indicate raw data.

3.1. Training losses

Two main losses, the content loss and the morphological loss, are supposed to compete, because the content loss, being a generative loss, hinders outputs to become blob-mask style as per the morphological loss, and vice versa. Thus, balancing two losses was crucial, meaning their magnitudes should match one to the other. Hence, we implemented a dry-run before an actual training, where NU-Net goes through a few iterations of dry-run to estimate magnitudes of the losses and match them. During the training, a desired loss behavior is to have a constantly decreasing morphological loss and a content loss increasing in the middle of training. The rationale is that, in the beginning, we want NU-Net to learn images themselves through the content loss, and to learn how to enhance blobs according to the morphological loss afterwards. As a result, the ideal loss curves looks as in figure 3. In general, weight coefficient w (in equation 3) in range of 5 to 10 yielded good results. Training took about 1-2 hours on a workstation with a computational GPU for 20 epochs with batch sizes being 16 and 128 for the content loss and the morphological loss, respectively.

3.2. Comparison to classical filters

We compared results of NU-Net to existing filtering and contrast enhancement algorithms in terms of a contrast gain. Those were logarithmic lookup table (LUT) mapping, median filter (med), Gaussian filter (gaus), and difference of Gaussians (DoG) filter. Logarithmic lookup table mapping

	LUT	Gauss	Med	DoG	NU-Net
contrast gain avg.	12.9±24.3	4.9±6.7	8±8	90.6±100.4	99.5±121.5
change avg. (times)	x2.7±2.2	x2±1.1	x2.4±1.4	x14.6±11.7	x16±16.7

Table 2. Improvement of contrast ratio (CR) over raw images from several classical algorithms and NU-Net, as the before/after difference and the fold increase. Average over datasets of average over test images from each dataset. NU-Net shows slightly better contrast gains compared with DoG filter across the datasets. See Fig. 1 and 4 for the much more significant but harder to assess difference in image quality.

(LUT) is the simplest method among them, which makes use of logarithm to remap pixel values and to stretch value intervals. Median and Gaussian filters are both denoising filters using parameterized kernels. While denoising does not have a direct impact on contrast, it could enhance contrast by smoothing out noisy background signals, combined with value range stretching strategy based on percentile values (upper bound to 99.8% and lower bound to 2%).

We defined contrast ratio ($CR = \frac{\mu_{fg}}{\mu_{bg}}$) to quantify contrast change and to compare NU-Net to other filters, quantitatively. CR is a fraction of a mean value of foreground pixels over that of background pixels. Note that contrast gain depends a lot on the initial contrast of a given image. Note as well that this metric is blind to resulting image quality.

The result of contrast change can be found in table 2. In addition, a qualitative result on a sample from LOCAL-THG dataset can be found in figure 4. Contrast gain was significant once the algorithm had a notion of foreground and background. For examples, LUT mapping is a merely pixel-wise method, and kernel-based filtering methods like median filter (med) and Gaussian filter (gaus) made the distinction by blurring off the noise and the background as well as the foreground. They accentuated cells, but the contrast was barely improved because they did not make clear distinction between foreground pixels and background pixels.

Among the tested filters, the DoG filter showed the most comparable results to NU-Net. The DoG filter is a band pass filter, and assumes that the objects of interest have a consistent size. It enhances visibility of objects by subtracting two Gaussian filters with different kernel sizes. DoG is, in a way, aware of contents and could detect foreground from background.

Figure 1 compares details of NU-Net to those to DoG filter. While both DoG filter and NU-Net picked up objects, DoG suffered from blurring artifacts from its algorithmic nature, just as other rule-based algorithms suffer from their own pitfalls. Blurring merged objects together making it hard to recognize individual objects. Moreover, DoG accentuated other structures too in presence of noise. NU-Net, however, did not suffer from blurring and was more robust to noise. Additionally, NU-Net not just detected contents but also actively enhances contrast by pushing foreground pixels up towards 255 (maximum value in unsigned integer

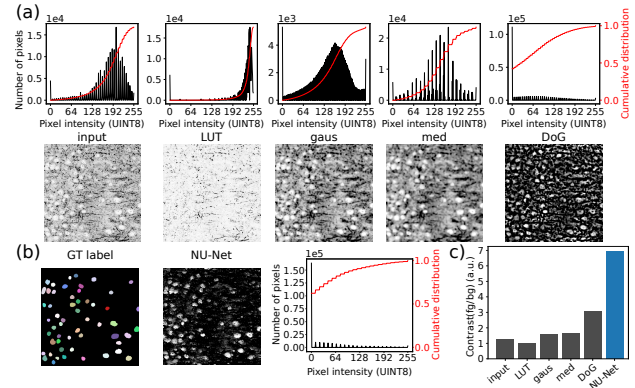


Figure 4. Comparing NU-Net to various filters. They were logarithmic lookup table (LUT) mapping, Gaussian filter (gaus), median filter (med) and difference of Gaussians (DoG) filter. (a) It shows the input image and the filtered images as well as their histograms. The red line shows the cumulative distribution of pixel values and gives a rough idea of image contrast. (b) It shows the ground truth (GT) mask and NU-Net’s result. Though, its cumulative distribution looks similar to that of DoG, the qualitative result is much cleaner than DoG. (c) Quantified contrast ratio (CR) of foreground pixels over background pixels.

8-bit) and background pixels down towards 0 though the morphological loss. As shown in the histogram in figure 4, histogram of image after NU-Net had all over lower background pixel values than that after DoG or than those after any filters presented.

3.3. Evaluation on realistic workflows

Given the clear aim of having an impact on actual, daily use of image analysis across labs and facility, the clearest path to evaluate NU-Net is through studying its use within a classical, end-to-end workflow.

3.3.1 Evaluation on a training set dataset

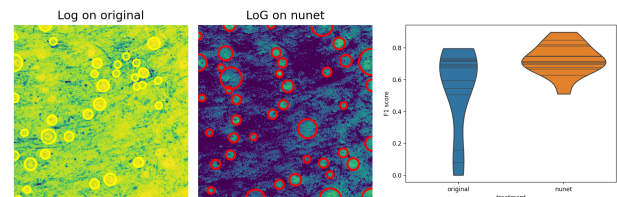


Figure 5. Use of NU-Net to improve a cell detection workflow of neuron in inverted contrast in a THG dataset. Left image: original image with detections from scikit-image blob-lob in red, right image: same with NU-Net filtered image. Right: distribution of F1 score across the dataset

First we study the effect of NU-Net in an object detection workflow on the LOCAL-THG dataset. To voluntarily use

a simple and straightforward workflow, similar to one that would be tried first in practice, we use the scikit-image[39] function `blob_log` as a point detection algorithm. It is actually not so simple as it scan through the Gaussian scale space of a Laplacian of Gaussian filter for stable detection, using a set threshold. The dataset comprises 14 2D images taken from a 3D Third Harmonic Generation (THG) microscopy volume of a slice of mouse brain. Due to the fact that THG signal is mainly from myelinated axons and not from cell bodies, the neuron are seen as noisy black spots in inverted contrast. A manual annotation of the positions of the cells was done, and the `blob_log` function was applied, with and without NU-Net. The results can be seen Fig. 5 and show a clear improvement of the detection, with the average F1 score across the dataset going from 0.54 to 0.72.

3.3.2 Evaluation on an unseen dataset

Obviously this assessment is biased since the LOCAL-THG dataset is part of the training set. As an unbiased assessment, we performed a similar numerical experiment on another dataset, unseen by the algorithm. We used a published dataset from [2], available on the Image Data Resource[41] <https://idr.openmicroscopy.org> under the accession idr0048, image ID 9846151, accessed through an OME-zarr endpoint (see code in section 5). It shows a section of a mouse cortex where the astrocytes are labeled with brainbow, and thus where the neuron are not labeled and again seen in inverted contrast. This problem is more challenging though since the background, consisting of astrocytes of various colors and intensities, is quite diverse. We randomly selected 15 crops out of the large volume and manually annotated them. This time, since the problem is more challenging and to be fair, the detection threshold was optimized independently for both conditions. The results are shown Fig. 6, with the mean F1 score rising from 0.49 to 0.61 from the use of NU-Net.

3.4. Controlling filtering magnitude

One big advantage of NU-Net as a filter is an ability to control the filtering magnitude. It can be achieved by adjusting the ratio between the morphological loss and the content loss. In practice, it is as simple as to manipulate the fraction of loss weights, that is w in equation 3, during training. Essentially, the larger the ratio w is, the more binarized the outputs become. An example of varying w is shown in figure 7.

We noticed that as w decreases, the content loss curves do not follow ideal curves anymore, which means that the resulted model would become closer to an autoencoder. Yet, due to presence of the morphological loss, resulted NU-Nets could still filter blobs. In the perspective of histogram

of images, it also means that filtered output roughly keeps the original distribution as close as possible. In comparison, when w increases, outputs becomes more like those of binary segmentation.

3.5. Inference time and resources

NU-Net is light and fast. The size of a model is about as small as 6.5 MB. To run inference over a volume whose size is 130 x 400 x 400 pixels (depth, height, width), it consumed 1.3 GB of GPU memory, which can fit in laptop hardware. On a workstation, it took only 3 second (23 ms/plane), while it took around 20 seconds on CPU (150 ms/plane).

3.6. Assessing versatility on open images

To test pre-trained NU-Net and to assess its versatility, we explored IDR (Image Data Resource) [41], an online database where researchers share their biological image data, and selected a handful of image crops that contained nuclei or cells; they include various modality in various sample and conditions. A pre-trained NU-Net with $w = 8.0$ was applied, and the results are shown in figure 8. Overall, NU-Net could spot and enhance blobs, acting as denoising, textured or inhomogeneous background subtraction and selective object enhancement, depending on the image.

3.6.1 Graphical user interfaces

NU-Net is easily accessible and available via Napari viewer [36], a multidimensional image viewer. We made NU-Net available as a plugin of Napari. The plugin comes with simple graphical user interfaces (GUIs) and 5 pre-trained NU-Net with varying w value from 5.0 to 10.0. Users may adjust filtering magnitude via w value and run the process either on CPU or GPU.

4. Conclusion

We propose a nuclear and cellular filter, NU-Net, built on perceptual losses as a drop-in alternative to classical filters. Those are very useful in biological image analysis pipelines but limited by their simplicity and strong assumptions. We introduced a novel loss based on perceptual losses, which attempts to generalize a single style given multiple images. Training is self-supervised as it does not require paired labels contrary to supervised learning. Our model result in appreciable contrast gain against even complex and textured background while keeping details, proved useful in practical use cases and is versatile, giving good results across the diversity of biological images. By adjusting loss coefficients during training, NU-Net can exert different degrees of filtering power. We made NU-Net easy to use via a Napari plugin with GUIs, including 5 pre-trained models. Overall,

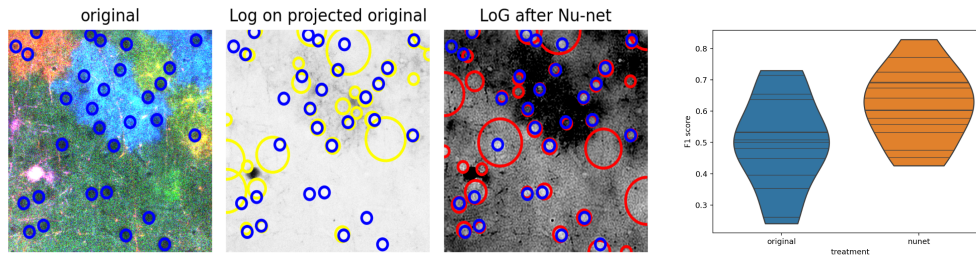


Figure 6. Use of NU-Net to improve a cell detection workflow of neuron in inverted contrast in a public brainbow astrocytes dataset. Left image: original image with annotation in blue, middle image: grayscale projection with detections from scikit-image blob-lob in red, right image: same with NU-Net filtered image. Right: distribution of F1 score across the dataset

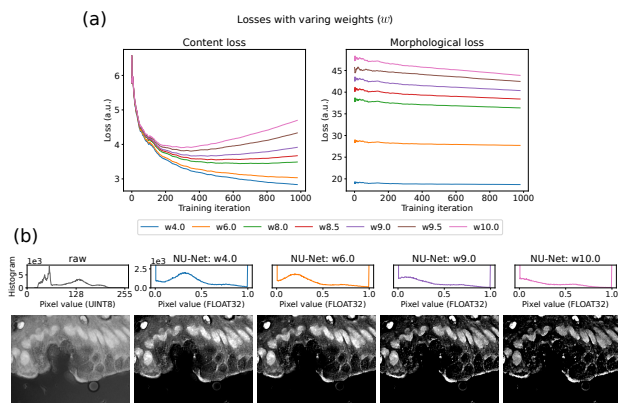


Figure 7. Controlling filtering magnitude by manipulating the loss coefficient w . (a) Loss curves with varying w values. Each morphological loss looks barely changing, but that is because w value is a multiplier of the morphological loss. (b) An example of applying NU-Nets trained with different magnitudes of weight (w). Note that w is a ratio of the morphological weight to the content weight in equation 3. Image source: DSB2018 dataset [1].

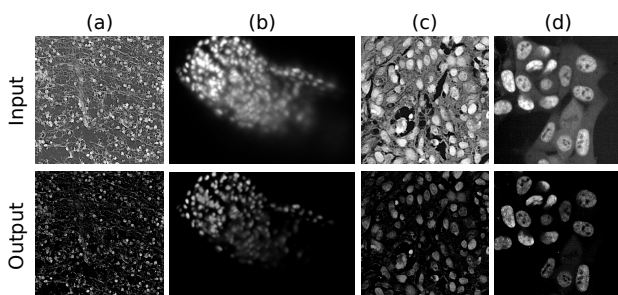


Figure 8. Application of a pre-trained NU-Net ($w = 8.0$) to nuclear and cellular images hosted on IDR (Image Data Resource) [41].

our aim is for NU-Net to be included into existing biological analysis workflow and allow for improved analysis of biological images across the board, to enable biological understanding.

5. Code and data availability

The source code of NU-Net resides at <https://github.com/LaboratoryOpticsBiosciences/nunet> and comes with codes to reproduce a large part of the figure of the paper. Additionally, the module is packaged and available through Python Package Index (PyPI) for easy installation and usage. A Napari plug-in is available at <https://github.com/sbinnee/napari-nunet>. Finally, the data used for training is used through <https://github.com/LaboratoryOpticsBiosciences/bioimagerloader>, where information on how to obtain the various datasets is available.

6. Acknowledgement

We thank Tanguy Rolland for contributing to the Napari plugin. This work was supported by HOPE - European Research Council - Horizon 2020 programme (grant No 951330 HOPE), Morphoscope - Agence Nationale de la Recherche - ANR-EQPX-0029, and France BioImaging - Agence Nationale de la Recherche - ANR-10-INBS-04.

References

- [1] 2018 Data Science Bowl. <https://kaggle.com/competitions/data-science-bowl-2018>. 5, 8
- [2] Lamiae Abdeladim, Katherine S. Matho, Solène Clavreul, Pierre Mahou, Jean-Marc Sintès, Xavier Solinas, Ignacio Arganda-Carreras, Stephen G. Turney, Jeff W. Lichtman, Anatole Chessel, Alexis-Pierre Bemelmans, Karine Loulier, Willy Supatto, Jean Livet, and Emmanuel Beaurepaire. Multicolor multiscale brain imaging with chromatic multiphoton serial microscopy. *Nature Communications*, 10(1):1–14, Apr. 2019. 7
- [3] Matthias Arzt, Joran Deschamps, Christopher Schmied, Tobias Pietzsch, Deborah Schmidt, Pavel Tomancak, Robert Haase, and Florian Jug. LABKIT: Labeling and Segmentation Toolkit for Big Image Data. *Frontiers in Computer Science*, 4, 2022. 1

- [4] Mohammad Babaeizadeh and Golnaz Ghiasi. Adjustable Real-time Style Transfer. In *International Conference on Learning Representations*, Sept. 2019. 3
- [5] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N. Straehle, Bernhard X. Kausler, Carsten Haubold, Martin Schiegg, Janez Ales, Thorsten Beier, Markus Rudy, Kemal Eren, Jaime I. Cervantes, Buote Xu, Fynn Beuttenmueller, Adrian Wolny, Chong Zhang, Ullrich Koethe, Fred A. Hamprecht, and Anna Kreshuk. Ilastik: Interactive machine learning for (bio)image analysis. *Nature Methods*, 16(12):1226–1232, Dec. 2019. 1, 3
- [6] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Frereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kudithipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avaniika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the Opportunities and Risks of Foundation Models. *arXiv:2108.07258 [cs]*, Aug. 2021. 4
- [7] Tim-Oliver Buchholz, Mangal Prakash, Alexander Krull, and Florian Jug. DenoiSeg: Joint Denoising and Segmentation. *arXiv:2005.02987 [cs]*, June 2020. 5
- [8] L. P. Coelho, A. Shariff, and R. F. Murphy. Nuclear segmentation in microscope cell images: A hand-segmented dataset and comparison of algorithms. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 518–521, June 2009. 5
- [9] Elisa Drelie Gelasca, Boguslaw Obara, Dmitry Fedorov, Kristian Kvilekval, and BS Manjunath. A biosegmentation benchmark for evaluation of bioimage analysis methods. *BMC Bioinformatics*, 10:368, Nov. 2009. 5
- [10] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A Learned Representation For Artistic Style. *arXiv:1610.07629 [cs]*, Feb. 2017. 2, 3
- [11] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016. 2, 3
- [12] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv:1705.06830 [cs]*, Aug. 2017. 3
- [13] Estibaliz Gómez-de-Mariscal, Martin Maška, Anna Kotrbová, Vendula Pospíchalová, Pavel Matula, and Arrate Muñoz-Barrutia. Deep-Learning-Based Segmentation of Small Extracellular Vesicles in Transmission Electron Microscopy Images. *Scientific Reports*, 9(1):13211, Sept. 2019. 5
- [14] Noah F. Greenwald, Geneva Miller, Erick Moen, Alex Kong, Adam Kagel, Thomas Dougherty, Christine Camacho Fullaway, Brianna J. McIntosh, Ke Xuan Leow, Morgan Sarah Schwartz, Cole Pavelchek, Sunny Cui, Isabella Camplisson, Omer Bar-Tal, Jaiveer Singh, Mara Fong, Gautam Chaudhry, Zion Abraham, Jackson Moseley, Shiri Warshawsky, Erin Soon, Shirley Greenbaum, Tyler Risom, Travis Hollmann, Sean C. Bendall, Leeat Keren, William Graf, Michael Angelo, and David Van Valen. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. *Nature Biotechnology*, 40(4):555–565, Apr. 2022. 1, 2, 4, 5
- [15] Hongliang He, Zhongyi Huang, Yao Ding, Guoli Song, Lin Wang, Qian Ren, Pengxu Wei, Zhiqiang Gao, and Jie Chen. CDNet: Centripetal Direction Network for Nuclear Instance Segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4026–4035, 2021. 1
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015. 4
- [17] Reka Hollandi, Abel Szkalitsy, Timea Toth, Ervin Tasnadi, Csaba Molnar, Botond Mathe, Istvan Grexa, Jozsef Molnar, Arpad Balind, Mate Gorbe, Maria Kovacs, Ede Migh, Allen Goodman, Tamas Balassa, Krisztian Kooos, Wenyu Wang, Juan Carlos Caicedo, Norbert Bara, Ferenc Kovacs, Lassi Paavolainen, Tivadar Danka, Andras Kriston, Anne Elizabeth Carpenter, Kevin Smith, and Peter Horvath. nucle-ALzer: A Parameter-free Deep Learning Framework for Nucleus Segmentation Using Image Style Transfer. *Cell Systems*, 10(5):453–458.e6, May 2020. 1, 2, 5
- [18] David Hörl, Fabio Rojas Rusak, Friedrich Preusser, Paul Tillberg, Nadine Randel, Raghav K. Chhetri, Albert Cardona, Philipp J. Keller, Hartmann Harz, Heinrich Leonhardt, Mathias Treier, and Stephan Preibisch. BigStitcher: Reconstructing high-resolution image datasets of cleared and expanded samples. *Nature Methods*, 16(9):870–874, Sept. 2019. 3
- [19] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. *arXiv:1703.06868 [cs]*, July 2017. 2, 3
- [20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-To-Image Translation With Conditional Adversarial Networks. In *Proceedings of the IEEE Conference*

- on *Computer Vision and Pattern Recognition*, pages 1125–1134, 2017. **2**
- [21] Andrew Janowczyk and Anant Madabhushi. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. *Journal of Pathology Informatics*, 7, July 2016. **5**
- [22] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *arXiv:1603.08155 [cs]*, Mar. 2016. **2, 3, 4**
- [23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. **4**
- [24] Florian Kromp, Eva Bozsaky, Fikret Rifatbegovic, Lukas Fischer, Magdalena Ambros, Maria Berneder, Tamara Weiss, Daria Lazic, Wolfgang Dörr, Allan Hanbury, Klaus Beiske, Peter F. Ambros, Inge M. Ambros, and Sabine Taschner-Mandl. An annotated fluorescence image dataset for training nuclear segmentation methods. *Scientific Data*, 7(1):262, Aug. 2020. **5**
- [25] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. Noise2Void - Learning Denoising from Single Noisy Images. *arXiv:1811.10980 [cs]*, Apr. 2019. **2**
- [26] Neeraj Kumar, Ruchika Verma, Sanuj Sharma, Surabhi Bhargava, Abhishek Vahadane, and Amit Sethi. A Dataset and a Technique for Generalized Nuclear Segmentation for Computational Pathology. *IEEE Transactions on Medical Imaging*, 36(7):1550–1560, July 2017. **1, 5**
- [27] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning Image Restoration without Clean Data. *arXiv:1803.04189 [cs, stat]*, Oct. 2018. **2**
- [28] Seongbin Lim, Xingjian Zhang, Emmanuel Beaufrais, and Anatole Chessel. BioImageLoader: Easy Handling of Bioimage Datasets for Machine Learning. <https://arxiv.org/abs/2303.02158v1>, Mar. 2023. **4**
- [29] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*, Feb. 2015. **4**
- [30] Vebjorn Ljosa, Katherine L. Sokolnicki, and Anne E. Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*, 9(7):637–637, July 2012. **5**
- [31] Peter Naylor, Marick Laé, Fabien Reyat, and Thomas Walter. Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map. *IEEE Transactions on Medical Imaging*, 38(2):448–459, Feb. 2019. **5**
- [32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. **1**
- [33] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: An open-source platform for biological image analysis. *Nature Methods*, 9(7):676–682, July 2012. **3**
- [34] Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers. Cell Detection with Star-convex Polygons. *arXiv:1806.03535 [cs]*, 11071:265–273, 2018. **1, 5**
- [35] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, Apr. 2015. **2, 3, 4**
- [36] Nicholas Sofroniew, Talley Lambert, Kira Evans, Juan Nunez-Iglesias, Grzegorz Bokota, Philip Winston, Gonzalo Peña-Castellanos, Kevin Yamauchi, Matthias Bussonnier, Draga Doncila Pop, Ahmet Can Solak, Ziyang Liu, Pam Wadhwa, Alister Burt, Genevieve Buckley, Andrew Sweet, Lukasz Migas, Volker Hilsenstein, Lorenzo Gaifas, Jordão Bragantini, Jaime Rodríguez-Guerra, Hector Muñoz, Jeremy Freeman, Peter Boone, Alan Lowe, Christoph Gohlke, Loic Royer, Andrea PIERRÉ, Hagai Har-Gil, and Abigail McGovern. Napari: A multi-dimensional image viewer for Python. Zenodo, May 2022. **7**
- [37] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: A generalist algorithm for cellular segmentation. *Nature Methods*, 18(1):100–106, Jan. 2021. **1, 2, 5**
- [38] Jean-Yves Tinevez, Nick Perry, Johannes Schindelin, Genevieve M. Hoopes, Gregory D. Reynolds, Emmanuel Laplantine, Sebastian Y. Bednarek, Spencer L. Shorte, and Kevin W. Eliceiri. TrackMate: An open and extensible platform for single-particle tracking. *Methods*, 115:80–90, Feb. 2017. **3**
- [39] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Guillard, and Tony Yu. Scikit-image: Image processing in Python. *PeerJ*, 2:e453, June 2014. **7**
- [40] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, Mauricio Rocha-Martins, Fabián Segovia-Miranda, Caren Norden, Ricardo Henriques, Marino Zerial, Michele Solimena, Jochen Rink, Pavel Tomancak, Loic Royer, Florian Jug, and Eugene W. Myers. Content-aware image restoration: Pushing the limits of fluorescence microscopy. *Nature Methods*, 15(12):1090–1097, Dec. 2018. **2**
- [41] Eleanor Williams, Josh Moore, Simon W. Li, Gabriella Rustici, Aleksandra Tarkowska, Anatole Chessel, Simone Leo, Bálint Antal, Richard K. Ferguson, Ugis Sarkans, Alvis Brazma, Rafael E. Carazo Salas, and Jason R. Swedlow. Image Data Resource: A bioimage data integration and publication platform. *Nature Methods*, 14(8):775–781, Aug. 2017. **7, 8**
- [42] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination. *arXiv:1805.01978 [cs]*, May 2018. **3**

APPENDIX E

[PUBLICATION] BIOIMAGELOADER

This is a preprint for bioimagerloader presented in chapter 2, which is available on arXiv:
<https://arxiv.org/abs/2303.02158>.

BIOIMAGELOADER: EASY HANDLING OF BIOIMAGE DATASETS FOR MACHINE LEARNING

A PREPRINT

Seongbin Lim

Laboratoire d'Optique et Biosciences
CNRS, INSERM, École Polytechnique, Institut Polytechnique de Paris
91120 Palaiseau, France
sungbin246@gmail.com

Xingjian Zhang

Laboratoire d'Optique et Biosciences
CNRS, INSERM, École Polytechnique, Institut Polytechnique de Paris
91120 Palaiseau, France

Emmanuel Beaurepaire

Laboratoire d'Optique et Biosciences
CNRS, INSERM, École Polytechnique, Institut Polytechnique de Paris
91120 Palaiseau, France

Anatole Chessel

Laboratoire d'Optique et Biosciences
CNRS, INSERM, École Polytechnique, Institut Polytechnique de Paris
91120 Palaiseau, France
anatole.chessel@polytechnique.edu

ABSTRACT

BioImageLoader (BIL) is a python library that handles bioimage datasets for machine learning applications, easing simple workflows and enabling complex ones. BIL attempts to wrap the numerous and varied bioimages datasets in unified interfaces, to easily concatenate, perform image augmentation, and batch-load them. By acting at a per experimental dataset level, it enables both a high level of customization and a comparison across experiments. Here we present the library and show some application it enables, including retraining published deep learning architectures and evaluating their versatility in a leave-one-dataset-out fashion.

Keywords Bioimage · Generic model · Python · Machine learning · MLOps

1 Introduction

Machine learning (ML) has taken biology by storm and bioimage informatics (BII) is one of the first discipline impacted. But core methodological advances and impressive application [1, 2, 3, 4] are the tip of the iceberg; the larger part of ML work involves a large number of careful numerical experiments comparing a variety of models, their hyperparameters, datasets or training regimen across losses and metrics. This implies specialized software to handle all those data, metadata and results. This operational side of ML, abbreviated as MLOps, has gathered a lot of attention across machine learning, with for example the libraries MLflow <https://mlflow.org> or Catalyst <https://catalyst-team.com> or start-ups Hugging Face <https://huggingface.co> or Weights & Biases <https://wandb.ai>.

In BII specifically, one can cite ZeroCostDL4Mic [5] which tries and provides a high level interface to train or study deep learning models. It implements an ‘MLOps for all’ approach by using a free web application as well as free but limited resources provided by Google. Other linked efforts includes the Bioimage Model Zoo [6], a free, open, and community-driven hub on web for deep learning models for bioimages that helps developers to share and deploy their models and ImJoy [6], which aims to integrate the whole MLOps through web interfaces. DeepImageJ [7] is an ImageJ plugin aiming at using models, in particular from the model zoo above, within ImageJ.

We propose *BioImageLoader* (BIL), a python library to facilitate the handling of image datasets for ML workflows (Fig 1.A). It introduces the experimental dataset as a unit, corresponding to the practical, daily use in biology of a set of images of similar sample, condition and imaging protocol. By building individual wrapper around each dataset, BIL make it easy to scale numerical experiments across many datasets, tailoring them to the specifics of each experimental datasets. In particular, it allows to assess the versatility of ML models by training them in a leave-one-dataset-out fashion. In the following we briefly present BIL and some applications it directly enables, including the retraining of common segmentation architecture on larger datasets. The code is open-source under BSD-3 license, and full documentation is available at <https://github.com/LaboratoryOpticsBiosciences/bioimageloader>, with the library being available to install through PyPI.

2 Implementation

BIL is a Python programming library. It followed object-oriented programming (OOP) scheme using abstract base classes (ABCs). Each dataset interface is called a collection and is based on the same ABC which allows sharing common properties and methods while keeping originality with concrete classes (cf Fig 1.B). It prioritized compatibility and extensibility with popular ML libraries such as PyTorch and TensorFlow as well as a data augmentation library, in particular, albumentations. Separate configuration file can be used to manage settings and to help keep track of individual experiments with ease. For additional usability, it makes use of cache and batch loading for high performance.

Every collection implemented original structures of each dataset and intended ways to load data as well as its metadata. They expose APIs (application programming interfaces) and allows either shared operations or individual manipulation. Basic capabilities shared across datasets includes unified interface for data and annotation, easy concatenation, batching, train/test splitting, and data augmentation. Custom changes per datasets would include specific I/O tailored to the specific way each dataset files is organized, selecting channels, normalizing value ranges, *etc.*

We have so far built interfaces for 28 open data datasets across many disciplines, sample and modalities, with new ones being easy to add. They include 17 datasets with annotations. We chose to not include the datasets themselves to avoid licensing issues, beyond the scope of this work, but they are all available online. In addition, we are releasing four additional annotated datasets from the Laboratory of Optics for Biosciences (LOB) with more challenging 2D segmentation problems from non-linear THG microscopy and multiphoton multicolor brainbow samples. They will be available on <https://zenodo.org>.

3 Applications

Example applications are developed at <https://github.com/sbinnee/nunet>. In particular, we can more easily look at image clustering across datasets to explore data similarity or retraining/fine-tuning of models on specific datasets. Specifically, the table in Fig. 1.D shows the retraining of StarDist architecture [3], one of the current state-of-the-art instance segmentation neural networks, with Fig 1.C showing an example on one of the new LOB dataset. Suggested model was trained across 14 datasets, either in a leave-one-dataset-out fashion, to assess the versatility of the models, or across all available datasets, to build a new generalist pre-trained StarDist model; see https://laboratoryopticsbiosciences.github.io/bioimageloader-docs/notebooks/train_models.html for details. The pretrained model demonstrated in this example is made available to the community on Bioimage Model Zoo (<https://bioimage.io>) [6].

4 Conclusions

Machine learning will revolutionize the way we do biology but only if running the many numerical experiments to build methods, scale them to larger dataset and/or tailor them to specific problems and workflow is not too cumbersome. We present *BioImageLoader* a python library to help in handling bioimage datasets. A particular exciting perspective is scaling up current deep learning studies toward ever larger datasets, the main limitation keeping us away from proper generic models. Part of a larger MLOps effort, we believe community wide adoption and development of such open source tools will enable rapid growth in the application of ML to biology.

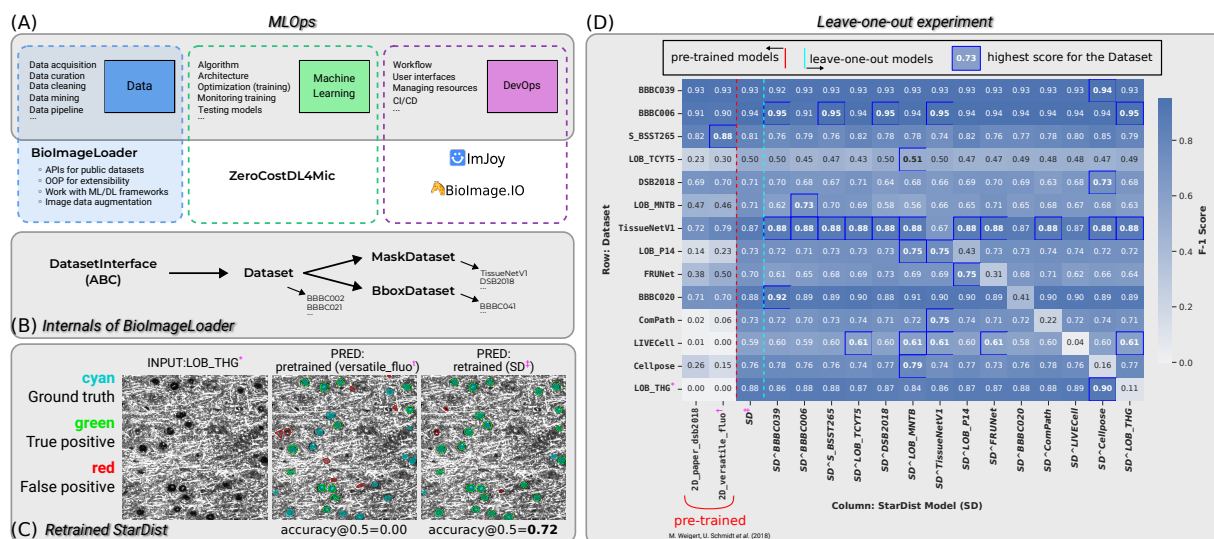


Figure 1: (A) MLOps refers to three main pillars: data, machine learning and DevOps. BioImageLoader (BIL) is a tool to handle data, which was missing. (B) BIL follows OOP (object-oriented programming) approach and makes use of ABC (abstract base class) to provide unified interfaces while keeping originality of each collection or dataset. (C) An example of a retrained StarDist architecture [8] over 14 datasets on one of four datasets we are releasing, called LOB_THG (refer annotations (*, †, ‡) to figure D on its right). (D) An ablation study in leave-one-out fashion that could be easily enabled by BIL. Column represents StarDist models that were either pretrained, retrained, or trained in leave-one-out fashion. Row represents datasets used to train or test models. Performance was measured by F-1 score.

5 Availability

BioImageLoader (BIL) is an open-source project. Source code is available at <https://github.com/LaboratoryOpticsBiosciences/bioimageloader>, and user manual at <https://laboratoryopticsbiosciences.github.io/bioimageloader-docs/>. The four new annotated datasets will be available on <https://zenodo.org>.

References

- [1] Oren Z Kraus, Ben T Grys, Jimmy Ba, Yolanda Chong, Brendan J Frey, Charles Boone, and Brenda J Andrews. Automated analysis of high-content microscopy data with deep learning. *Molecular systems biology*, 13(4):924, 2017.
- [2] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature methods*, 15(12):1090–1097, 2018.
- [3] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: A generalist algorithm for cellular segmentation. *Nature Methods*, 18(1):100–106, January 2021.
- [4] Noah F Greenwald, Geneva Miller, Erick Moen, Alex Kong, Adam Kagel, Thomas Dougherty, Christine Camacho Fullaway, Brianna J McIntosh, Ke Xuan Leow, Morgan Sarah Schwartz, et al. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. *Nature biotechnology*, 40(4):555–565, 2022.
- [5] Lucas von Chamier, Romain F. Laine, Johanna Jukkala, Christoph Spahn, Daniel Krentzel, Elias Nehme, Martina Lerche, Sara Hernández-Pérez, Pieta K. Mattila, Eleni Karinou, Séamus Holden, Ahmet Can Solak, Alexander Krull, Tim-Oliver Buchholz, Martin L. Jones, Loïc A. Royer, Christophe Leterrier, Yoav Shechtman, Florian Jug, Mike Heilemann, Guillaume Jacquemet, and Ricardo Henriques. Democratizing deep learning for microscopy with ZeroCostDL4Mic. *Nature Communications*, 12(1):2276, April 2021.
- [6] Wei Ouyang, Fynn Beuttenmueller, Estibaliz Gómez-de-Mariscal, Constantin Pape, Tom Burke, Carlos García-López-de-Haro, Craig Russell, Lucía Moya-Sans, Cristina de-la-Torre-Gutiérrez, Deborah Schmidt, Dominik Kutra,

- Maksim Novikov, Martin Weigert, Uwe Schmidt, Peter Bankhead, Guillaume Jacquemet, Daniel Sage, Ricardo Henriques, Arrate Muñoz-Barrutia, Emma Lundberg, Florian Jug, and Anna Kreshuk. BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis, June 2022.
- [7] Estibaliz Gómez-de Mariscal, Carlos García-López-de Haro, Wei Ouyang, Laurène Donati, Emma Lundberg, Michael Unser, Arrate Muñoz-Barrutia, and Daniel Sage. Deepimagej: A user-friendly environment to run deep learning models in imagej. *Nature Methods*, 18(10):1192–1195, 2021.
- [8] Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers. Cell Detection with Star-convex Polygons. *arXiv:1806.03535 [cs]*, 11071:265–273, 2018.

BIBLIOGRAPHY

- [1] Jean Livet, Tamily A. Weissman, Hyuno Kang, Ryan W. Draft, Ju Lu, Robyn A. Bennis, Joshua R. Sanes, and Jeff W. Lichtman. Transgenic strategies for combinatorial expression of fluorescent proteins in the nervous system. Nature, 450(7166):56–62, November 2007. [10](#), [15](#), [31](#), [111](#), [116](#), [123](#), [128](#), [129](#)
- [2] Lamiae Abdeladim, Katherine S. Matho, Solène Clavreul, Pierre Mahou, Jean-Marc Sintes, Xavier Solinas, Ignacio Arganda-Carreras, Stephen G. Turney, Jeff W. Lichtman, Anatole Chessel, Alexis-Pierre Bemelmans, Karine Loulier, Willy Supatto, Jean Livet, and Emmanuel Beaurepaire. Multicolor multiscale brain imaging with chromatic multiphoton serial microscopy. Nature Communications, 10(1):1–14, April 2019. [10](#), [15](#), [17](#), [31](#), [75](#), [109](#), [111](#), [123](#), [128](#)
- [3] Periklis Pantazis and Willy Supatto. Advances in whole-embryo imaging: A quantitative transition is underway. Nature Reviews Molecular Cell Biology, 15(5):327–339, May 2014. [11](#), [18](#)
- [4] Willy Supatto, Thai V Truong, Delphine Débarre, and Emmanuel Beaurepaire. Advances in multiphoton microscopy for imaging embryos. Current Opinion in Genetics & Development, 21(5):538–548, October 2011. [14](#), [15](#)
- [5] Yinan Wan, Katie McDole, and Philipp J. Keller. Light-Sheet Microscopy and Its Potential for Understanding Developmental Processes. Annual Review of Cell and Developmental Biology, 35:655–681, October 2019. [15](#)
- [6] Peter G Pitrone, Johannes Schindelin, Luke Stuyvenberg, Stephan Preibisch, Michael Weber, Kevin W Eliceiri, Jan Huisken, and Pavel Tomancak. OpenSPIM: An open-access light-sheet microscopy platform. Nature methods, 10(7):598–599, July 2013. [16](#)
- [7] David Hörl, Fabio Rojas Rusak, Friedrich Preusser, Paul Tillberg, Nadine Randel, Raghav K. Chhetri, Albert Cardona, Philipp J. Keller, Hartmann Harz, Heinrich Leonhardt, Mathias Treier, and Stephan Preibisch. BigStitcher: Reconstructing high-resolution image datasets of cleared and expanded samples. Nature Methods, 16(9):870–874, September 2019. [17](#), [105](#)
- [8] Hanchuan Peng. Bioimage informatics: A new area of engineering biology. Bioinformatics, 24(17):1827–1836, September 2008. [17](#), [18](#)
- [9] Ivo F. Sbalzarini. Seeing Is Believing: Quantifying Is Convincing: Computational Image Analysis in Biology. Advances in Anatomy, Embryology, and Cell Biology, 219:1–39, 2016. [18](#)

Bibliography

- [10] Thi Phuong Lien Ung, Seongbin Lim, Xavier Solinas, Pierre Mahou, Anatole Chesel, Claire Marionnet, Thomas Bornschlöggl, Emmanuel Beaurepaire, Françoise Bernerd, Ana-Maria Pena, and Chiara Stringari. Simultaneous NAD(P)H and FAD fluorescence lifetime microscopy of long UVA-induced metabolic stress in reconstructed human skin. *Scientific Reports*, 11(1):22171, November 2021. [18](#), [119](#), [122](#)
- [11] Ko Sugawara, Cagri Cevrim, and Michalis Averof. Tracking cell lineages in 3D by incremental deep learning, September 2021. [18](#)
- [12] Siqi Liu, Donghao Zhang, Yang Song, Hanchuan Peng, and Weidong Cai. Automated 3-D Neuron Tracing With Precise Branch Erasing and Confidence Controlled Back Tracking. *IEEE Transactions on Medical Imaging*, 37(11):2441–2452, November 2018. [18](#)
- [13] Zohaib Salahuddin, Matthias Lenga, and Hannes Nickisch. Multi-Resolution 3D Convolutional Neural Networks for Automatic Coronary Centerline Extraction in Cardiac CT Angiography Scans. *arXiv:2010.00925 [cs, eess]*, December 2020. [18](#)
- [14] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: An open-source platform for biological-image analysis. *Nature Methods*, 9(7):676–682, July 2012. [18](#), [21](#), [22](#), [27](#), [28](#), [123](#)
- [15] Jean-Yves Tinevez, Nick Perry, Johannes Schindelin, Genevieve M. Hoopes, Gregory D. Reynolds, Emmanuel Laplantine, Sebastian Y. Bednarek, Spencer L. Shorte, and Kevin W. Eliceiri. TrackMate: An open and extensible platform for single-particle tracking. *Methods*, 115:80–90, February 2017. [18](#), [22](#), [105](#)
- [16] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. Scikit-image: Image processing in Python. *PeerJ*, 2:e453, June 2014. [18](#), [168](#)
- [17] Christophe Zimmer. From microbes to numbers: Extracting meaningful quantities from images. *Cellular Microbiology*, 14(12):1828–1835, 2012. [18](#)
- [18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. [18](#), [29](#), [76](#), [162](#), [165](#), [168](#), [169](#), [170](#)
- [19] Vladimír Ulman, Martin Maška, Klas E. G. Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, Ihor Smal, Karl Rohr, Joakim Jaldén, Helen M. Blau, Oleh Dzyubachyk, Boudewijn Lelieveldt, Pengdong Xiao, Yuexiang Li, Siu-Yeung Cho, Alexandre C. Dufour, Jean-Christophe Olivo-Marin, Constantino C. Reyes-Aldasoro, Jose A. Solis-Lemus, Robert Bensch, Thomas Brox, Johannes Stegmaier, Ralf Mikut, Stefan Wolf, Fred A. Hamprecht, Tiago Esteves, Pedro Quelhas, Ömer Demirel, Lars Malmström, Florian Jug, Pavel Tomancak, Erik Meijering, Arrate Muñoz-Barrutia,

- Michal Kozubek, and Carlos Ortiz-de-Solorzano. An objective comparison of cell-tracking algorithms. *Nature Methods*, 14(12):1141–1152, December 2017. [20](#), [21](#), [25](#), [87](#)
- [20] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. [arXiv:1505.04597 \[cs\]](#), May 2015. [21](#), [29](#), [88](#), [90](#), [91](#), [115](#)
- [21] Peter Naylor, Marick Laé, Fabien Reyat, and Thomas Walter. Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map. *IEEE Transactions on Medical Imaging*, 38(2):448–459, February 2019. [21](#), [26](#), [174](#)
- [22] Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers. Cell Detection with Star-Convex Polygons. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, Lecture Notes in Computer Science, pages 265–273, Cham, 2018. Springer International Publishing. [21](#), [22](#), [28](#), [29](#), [30](#), [58](#), [59](#), [60](#), [75](#), [80](#), [84](#), [87](#), [91](#), [93](#), [95](#)
- [23] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation, November 2016. [21](#)
- [24] Hongying Liu, Xiongjie Shen, Fanhua Shang, and Fei Wang. CU-Net: Cascaded U-Net with Loss Weighted Sampling for Brain Tumor Segmentation. [arXiv:1907.07677 \[cs, eess\]](#), July 2019. [21](#)
- [25] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. [arXiv:1606.04797 \[cs\]](#), June 2016. [21](#)
- [26] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. [arXiv:1606.06650 \[cs\]](#), June 2016. [21](#)
- [27] Yongjin Zhou, Weijian Huang, Pei Dong, Yong Xia, and Shanshan Wang. D-UNet: A dimension-fusion U shape network for chronic stroke lesion segmentation. [arXiv:1908.05104 \[cs, eess\]](#), August 2019. [21](#)
- [28] Martin Weigert, Uwe Schmidt, Robert Haase, Ko Sugawara, and Gene Myers. Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3655–3662, Snowmass Village, CO, USA, March 2020. IEEE. [21](#), [28](#)
- [29] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: A generalist algorithm for cellular segmentation. *Nature Methods*, 18(1):100–106, January 2021. [21](#), [22](#), [28](#), [29](#), [30](#), [58](#), [59](#), [60](#), [75](#), [76](#), [80](#), [83](#), [84](#), [87](#), [88](#), [91](#), [93](#), [116](#), [128](#), [129](#), [174](#)
- [30] Curtis T. Rueden, Johannes Schindelin, Mark C. Hiner, Barry E. DeZonia, Alison E. Walter, Ellen T. Arena, and Kevin W. Eliceiri. ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics*, 18(1):1–26, December 2017. [21](#), [27](#), [126](#)

Bibliography

- [31] David R. Stirling, Madison J. Swain-Bowden, Alice M. Lucas, Anne E. Carpenter, Beth A. Cimini, and Allen Goodman. CellProfiler 4: Improvements in speed, utility and usability. *BMC Bioinformatics*, 22(1):1–11, December 2021. [21](#), [22](#), [28](#)
- [32] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. [arXiv:1612.03144 \[cs\]](#), April 2017. [21](#), [23](#)
- [33] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models, December 2020. [21](#), [90](#), [115](#), [134](#)
- [34] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded Diffusion Models for High Fidelity Image Generation, December 2021. [21](#)
- [35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models, April 2022. [21](#)
- [36] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image Super-Resolution via Iterative Refinement, June 2021. [21](#)
- [37] Nicholas Sofroniew, Talley Lambert, Kira Evans, Juan Nunez-Iglesias, Grzegorz Bokota, Philip Winston, Gonzalo Peña-Castellanos, Kevin Yamauchi, Matthias Bussonnier, Draga Doncila Pop, Ahmet Can Solak, Ziyang Liu, Pam Wadhwa, Alister Burt, Genevieve Buckley, Andrew Sweet, Lukasz Migas, Volker Hilsenstein, Lorenzo Gaifas, Jordão Bragantini, Jaime Rodríguez-Guerra, Hector Muñoz, Jeremy Freeman, Peter Boone, Alan Lowe, Christoph Gohlke, Loic Royer, Andrea PIERRÉ, Hagai Har-Gil, and Abigail McGovern. Napari: A multi-dimensional image viewer for Python. Zenodo, May 2022. [21](#), [22](#), [28](#), [92](#), [108](#), [109](#), [123](#), [125](#)
- [38] Kevin J. Cutler, Carsen Stringer, Paul A. Wiggins, and Joseph D. Mougous. Omnipose: A high-precision morphology-independent solution for bacterial cell segmentation, November 2021. [22](#)
- [39] Dmitry Ershov, Minh-Son Phan, Joanna W. Pylvänäinen, Stéphane U. Rigaud, Laure Le Blanc, Arthur Charles-Orszag, James R. W. Conway, Romain F. Laine, Nathan H. Roy, Daria Bonazzi, Guillaume Duménil, Guillaume Jacquemet, and Jean-Yves Tinevez. TrackMate 7: Integrating state-of-the-art segmentation algorithms into tracking pipelines. *Nature Methods*, 19(7):829–832, July 2022. [22](#)
- [40] Reka Hollandi, Abel Szkalitsy, Timea Toth, Ervin Tasnadi, Csaba Molnar, Botond Mathe, Istvan Grexa, Jozsef Molnar, Arpad Balind, Mate Gorbe, Maria Kovacs, Ede Migh, Allen Goodman, Tamas Balassa, Krisztian Koos, Wenyu Wang, Juan Carlos Caicedo, Norbert Bara, Ferenc Kovacs, Lassi Paavolainen, Tivadar Danka, Andras Kriston, Anne Elizabeth Carpenter, Kevin Smith, and Peter Horvath. nucleAIzer: A Parameter-free Deep Learning Framework for Nucleus Segmentation Using Image Style Transfer. *Cell Systems*, 10(5):453–458.e6, May 2020. [22](#), [29](#), [60](#), [76](#), [88](#), [93](#), [115](#)

-
- [41] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. [arXiv:1703.06870 \[cs\]](https://arxiv.org/abs/1703.06870), January 2018. [22](#), [25](#), [60](#), [93](#)
- [42] Thierry Pécot, Maria C. Cuitiño, Roger H. Johnson, Cynthia Timmers, and Gustavo Leone. Deep learning tools and modeling to estimate the temporal expression of cell cycle proteins from 2D still images. *PLOS Computational Biology*, 18(3):e1009949, March 2022. [22](#), [127](#)
- [43] Noah F. Greenwald, Geneva Miller, Erick Moen, Alex Kong, Adam Kagel, Thomas Dougherty, Christine Camacho Fullaway, Brianna J. McIntosh, Ke Xuan Leow, Morgan Sarah Schwartz, Cole Pavelchek, Sunny Cui, Isabella Camplisson, Omer Bartal, Jaiveer Singh, Mara Fong, Gautam Chaudhry, Zion Abraham, Jackson Moseley, Shiri Warshawsky, Erin Soon, Shirley Greenbaum, Tyler Risom, Travis Hollmann, Sean C. Bendall, Leeat Keren, William Graf, Michael Angelo, and David Van Valen. Whole-cell segmentation of tissue images with human-level performance using large-scale data annotation and deep learning. *Nature Biotechnology*, 40(4):555–565, April 2022. [22](#), [24](#), [26](#), [29](#), [30](#), [31](#), [32](#), [59](#), [60](#), [66](#), [87](#), [88](#), [96](#), [114](#), [174](#)
- [44] Sergey I. Nikolenko. Synthetic Data for Deep Learning. [arXiv:1909.11512 \[cs\]](https://arxiv.org/abs/1909.11512), September 2019. [22](#)
- [45] Clément Douarre, Richard Schielein, Carole Frindel, Stefan Gerth, and David Rousseau. Transfer Learning from Synthetic Data Applied to Soil–Root Segmentation in X-Ray Tomography Images. *Journal of Imaging*, 4(5):65, May 2018. [22](#)
- [46] Clément Douarre, Carlos F. Crispim-Junior, Anthony Gelibert, Laure Tougne, and David Rousseau. Novel data augmentation strategies to boost supervised segmentation of plant disease. *Computers and Electronics in Agriculture*, 165:104967, October 2019. [22](#)
- [47] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-To-Image Translation With Conditional Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1125–1134, 2017. [22](#), [93](#)
- [48] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. [arXiv:1406.2661 \[cs, stat\]](https://arxiv.org/abs/1406.2661), June 2014. [22](#), [43](#), [93](#), [94](#), [151](#)
- [49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. [arXiv:1512.03385 \[cs\]](https://arxiv.org/abs/1512.03385), December 2015. [23](#), [29](#), [88](#), [91](#), [98](#), [166](#)
- [50] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, Mauricio Rocha-Martins, Fabián Segovia-Miranda, Caren Norden, Ricardo Henriques, Marino Zerial, Michele Solimena, Jochen Rink, Pavel Tomancak, Loïc Royer, Florian Jug, and Eugene W. Myers. Content-aware image restoration: Pushing the limits of fluorescence microscopy. *Nature Methods*, 15(12):1090–1097, December 2018. [23](#), [29](#), [94](#)

Bibliography

- [51] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. Noise2Void - Learning Denoising from Single Noisy Images. [arXiv:1811.10980 \[cs\]](https://arxiv.org/abs/1811.10980), April 2019. [23](#), [24](#), [29](#), [94](#)
- [52] A. Buades, B. Coll, and J. M. Morel. A Review of Image Denoising Algorithms, with a New One. *Multiscale Modeling & Simulation*, 4(2):490–530, January 2005. [24](#)
- [53] Jérôme Boulanger, Charles Kervrann, Patrick Bouthemy, Peter Elbau, Jean-Baptiste Sibarita, and Jean Salamero. Patch-based nonlocal functional for denoising fluorescence microscopy image sequences. *IEEE transactions on medical imaging*, 29(2):442–454, February 2010. [24](#)
- [54] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2Noise: Learning Image Restoration without Clean Data. [arXiv:1803.04189 \[cs, stat\]](https://arxiv.org/abs/1803.04189), October 2018. [24](#), [94](#)
- [55] Vebjorn Ljosa, Katherine L. Sokolnicki, and Anne E. Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*, 9(7):637–637, July 2012. [24](#), [30](#), [174](#), [175](#)
- [56] David N. Orloff, Janet H. Iwasa, Maryann E. Martone, Mark H. Ellisman, and Caroline M. Kane. The cell: An image library-CCDB: A curated repository of microscopy data. *Nucleic Acids Research*, 41(Database issue):D1241–1250, January 2013. [25](#), [30](#)
- [57] Eleanor Williams, Josh Moore, Simon W. Li, Gabriella Rustici, Aleksandra Tarkowska, Anatole Chessel, Simone Leo, Bálint Antal, Richard K. Ferguson, Ugis Sarkans, Alvis Brazma, Rafael E. Carazo Salas, and Jason R. Swedlow. Image Data Resource: A bioimage data integration and publication platform. *Nature Methods*, 14(8):775–781, August 2017. [25](#), [30](#)
- [58] 2018 Data Science Bowl. <https://kaggle.com/competitions/data-science-bowl-2018>. [25](#), [59](#), [68](#), [100](#), [174](#)
- [59] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, June 2014. [25](#)
- [60] Ross Girshick. Fast R-CNN, September 2015. [25](#)
- [61] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. [arXiv:1506.01497 \[cs\]](https://arxiv.org/abs/1506.01497), January 2016. [25](#)
- [62] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. [26](#), [29](#), [30](#), [103](#), [114](#), [146](#), [165](#), [170](#)

- [63] Mischa Schwendy, Ronald E Unger, and Sapun H Parekh. EVICAN—a balanced dataset for algorithm development in cell and nucleus segmentation. *Bioinformatics*, 36(12):3863–3870, June 2020. [26](#)
- [64] Christoffer Edlund, Timothy R. Jackson, Nabeel Khalid, Nicola Bevan, Timothy Dale, Andreas Dengel, Sheraz Ahmed, Johan Trygg, and Rickard Sjögren. LIVE-Cell—A large-scale dataset for label-free live cell segmentation. *Nature Methods*, 18(9):1038–1045, September 2021. [26](#), [59](#), [66](#), [174](#)
- [65] Elisa Drelie Gelasca, Boguslaw Obara, Dmitry Fedorov, Kristian Kvilekval, and BS Manjunath. A biosegmentation benchmark for evaluation of bioimage analysis methods. *BMC Bioinformatics*, 10:368, November 2009. [26](#), [174](#)
- [66] Andrew Janowczyk and Anant Madabhushi. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. *Journal of Pathology Informatics*, 7, July 2016. [26](#), [107](#), [174](#)
- [67] Neeraj Kumar, Ruchika Verma, Sanuj Sharma, Surabhi Bhargava, Abhishek Vahadane, and Amit Sethi. A Dataset and a Technique for Generalized Nuclear Segmentation for Computational Pathology. *IEEE Transactions on Medical Imaging*, 36(7):1550–1560, July 2017. [26](#), [107](#), [126](#), [174](#)
- [68] Todd A. Gillette, Kerry M. Brown, and Giorgio A. Ascoli. The DIADEM Metric: Comparing Multiple Reconstructions of the Same Neuron. *Neuroinformatics*, 9(0):233–245, September 2011. [26](#), [87](#)
- [69] Tobias Pietzsch, Stephan Preibisch, Pavel Tomančák, and Stephan Saalfeld. ImgLib2—generic image processing in Java. *Bioinformatics*, 28(22):3009–3011, November 2012. [27](#)
- [70] Melissa Linkert, Curtis T. Rueden, Chris Allan, Jean-Marie Burel, Will Moore, Andrew Patterson, Brian Loranger, Josh Moore, Carlos Neves, Donald MacDonald, Aleksandra Tarkowska, Caitlin Sticco, Emma Hill, Mike Rossner, Kevin W. Eliceiri, and Jason R. Swedlow. Metadata matters: Access to image data in the real world. *Journal of Cell Biology*, 189(5):777–782, May 2010. [27](#), [64](#)
- [71] Robert Haase, Loic A. Royer, Peter Steinbach, Deborah Schmidt, Alexandr Dibrov, Uwe Schmidt, Martin Weigert, Nicola Maghelli, Pavel Tomancak, Florian Jug, and Eugene W. Myers. CLIJ: GPU-accelerated image processing for everyone. *Nature Methods*, 17(1):5–6, January 2020. [27](#)
- [72] Chris Allan, Jean-Marie Burel, Josh Moore, Colin Blackburn, Melissa Linkert, Scott Loynton, Donald MacDonald, William J. Moore, Carlos Neves, Andrew Patterson, Michael Porter, Aleksandra Tarkowska, Brian Loranger, Jerome Avondo, Ingvar Lagerstedt, Luca Lianas, Simone Leo, Katherine Hands, Ron T. Hay, Ardan Patwardhan, Christoph Best, Gerard J. Kleywegt, Gianluigi Zanetti, and Jason R. Swedlow. OMERO: Flexible, model-driven data management for experimental biology. *Nature Methods*, 9(3):245–253, March 2012. [28](#)
- [73] Ilya G. Goldberg, Chris Allan, Jean-Marie Burel, Doug Creager, Andrea Falconi, Harry Hochheiser, Josiah Johnston, Jeff Mellen, Peter K. Sorger, and Jason R. Swedlow. The Open Microscopy Environment (OME) Data Model and XML file:

Bibliography

- Open tools for informatics and quantitative analysis in biological imaging. *Genome Biology*, 6(5):1–13, May 2005. [28](#), [87](#)
- [74] Josh Moore, Chris Allan, Sébastien Besson, Jean-Marie Burel, Erin Diel, David Gault, Kevin Kozlowski, Dominik Lindner, Melissa Linkert, Trevor Manz, Will Moore, Constantin Pape, Christian Tischer, and Jason R. Swedlow. OME-NGFF: A next-generation file format for expanding bioimaging data-access strategies. *Nature Methods*, pages 1–3, November 2021. [28](#), [87](#), [126](#), [128](#)
- [75] Stuart Berg, Dominik Kutra, Thorben Kroeger, Christoph N. Straehle, Bernhard X. Kausler, Carsten Haubold, Martin Schiegg, Janez Ales, Thorsten Beier, Markus Rudy, Kemal Eren, Jaime I. Cervantes, Buote Xu, Fynn Beuttenmueller, Adrian Wolny, Chong Zhang, Ullrich Koethe, Fred A. Hamprecht, and Anna Kreshuk. Ilastik: Interactive machine learning for (bio)image analysis. *Nature Methods*, 16(12):1226–1232, December 2019. [28](#), [105](#)
- [76] Matthias Arzt, Joran Deschamps, Christopher Schmied, Tobias Pietzsch, Deborah Schmidt, Pavel Tomancak, Robert Haase, and Florian Jug. LABKIT: Labeling and Segmentation Toolkit for Big Image Data. *Frontiers in Computer Science*, 4, 2022. [28](#)
- [77] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. [28](#)
- [78] Réka Hollandi, Ákos Diódsi, Gábor Hollandi, Nikita Moshkov, and Péter Horváth. AnnotatorJ: An ImageJ plugin to ease hand annotation of cellular compartments. *Molecular Biology of the Cell*, 31(20):2179–2186, July 2020. [28](#), [29](#), [30](#), [126](#)
- [79] Wei Ouyang, Fynn Beuttenmueller, Estibaliz Gómez-de-Mariscal, Constantin Pape, Tom Burke, Carlos García-López-de-Haro, Craig Russell, Lucía Moya-Sans, Cristina de-la-Torre-Gutiérrez, Deborah Schmidt, Dominik Kutra, Maksim Novikov, Martin Weigert, Uwe Schmidt, Peter Bankhead, Guillaume Jacquemet, Daniel Sage, Ricardo Henriques, Arrate Muñoz-Barrutia, Emma Lundberg, Florian Jug, and Anna Kreshuk. BioImage Model Zoo: A Community-Driven Resource for Accessible Deep Learning in BioImage Analysis, June 2022. [28](#), [92](#), [126](#)
- [80] Wei Ouyang, Trang Le, Hao Xu, and Emma Lundberg. Interactive biomedical segmentation tool powered by deep learning and ImJoy. [28](#)
- [81] Lucas von Chamier, Romain F. Laine, Johanna Jukkala, Christoph Spahn, Daniel Krentzel, Elias Nehme, Martina Lerche, Sara Hernández-Pérez, Pieta K. Mattila, Eleni Karinou, Séamus Holden, Ahmet Can Solak, Alexander Krull, Tim-Oliver Buchholz, Martin L. Jones, Loïc A. Royer, Christophe Leterrier, Yoav Shechtman, Florian Jug, Mike Heilemann, Guillaume Jacquemet, and Ricardo Henriques. Democratising deep learning for microscopy with ZeroCostDL4Mic. *Nature Communications*, 12(1):2276, April 2021. [28](#)
- [82] Estibaliz Gómez-de-Mariscal, Carlos García-López-de-Haro, Wei Ouyang, Laurene Donati, Emma Lundberg, Michael Unser, Arrate Muñoz-Barrutia, and Daniel Sage. DeepImageJ: A user-friendly environment to run deep learning models in ImageJ. *Nature Methods*, 18(10):1192–1195, October 2021. [28](#)

-
- [83] Joshua Batson and Loic Royer. Noise2Self: Blind Denoising by Self-Supervision. [arXiv:1901.11365](https://arxiv.org/abs/1901.11365) [cs, stat], June 2019. [29](#)
- [84] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. [29](#), [38](#), [88](#), [93](#), [114](#), [134](#), [144](#)
- [85] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. [arXiv:1409.1556](https://arxiv.org/abs/1409.1556) [cs], April 2015. [29](#), [30](#), [76](#), [78](#), [91](#), [98](#), [100](#), [159](#), [165](#), [166](#)
- [86] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A Learned Representation For Artistic Style. [arXiv:1610.07629](https://arxiv.org/abs/1610.07629) [cs], February 2017. [29](#), [92](#), [95](#), [103](#)
- [87] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In [Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition](#), pages 2414–2423, 2016. [29](#), [50](#), [51](#), [52](#), [92](#), [93](#), [95](#), [101](#), [103](#)
- [88] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. [arXiv:1603.08155](https://arxiv.org/abs/1603.08155) [cs], March 2016. [29](#), [76](#), [92](#), [93](#), [94](#), [95](#), [98](#), [99](#), [103](#)
- [89] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards Total Recall in Industrial Anomaly Detection, May 2022. [29](#)
- [90] Hanqiu Deng and Xingyu Li. Anomaly Detection via Reverse Distillation from One-Class Embedding, March 2022. [29](#)
- [91] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Muniyikwa, Suraj Nair, Avaniika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher

Bibliography

- Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the Opportunities and Risks of Foundation Models. [arXiv:2108.07258 \[cs\]](https://arxiv.org/abs/2108.07258), August 2021. [30](#), [34](#), [41](#), [88](#), [91](#), [92](#), [93](#), [114](#)
- [92] Florian Kropf, Eva Bozsaky, Fikret Rifatbegovic, Lukas Fischer, Magdalena Ambros, Maria Berneder, Tamara Weiss, Daria Lazic, Wolfgang Dörr, Allan Hanbury, Klaus Beiske, Peter F. Ambros, Inge M. Ambros, and Sabine Taschner-Mandl. An annotated fluorescence image dataset for training nuclear segmentation methods. *Scientific Data*, 7(1):262, August 2020. [31](#), [174](#)
- [93] Neeraj Kumar, Ruchika Verma, Deepak Anand, Yanning Zhou, Omer Fahri Onder, Efstratios Tsougenis, Hao Chen, Pheng-Ann Heng, Jiahui Li, Zhiqiang Hu, Yunzhi Wang, Navid Alemi Koohbanani, Mostafa Jahanifar, Neda Zamani Tajeddin, Ali Gooya, Nasir Rajpoot, Xuhua Ren, Sihang Zhou, Qian Wang, Dinggang Shen, Cheng-Kun Yang, Chi-Hung Weng, Wei-Hsiang Yu, Chao-Yuan Yeh, Shuang Yang, Shuoyu Xu, Pak Hei Yeung, Peng Sun, Amirreza Mahbod, Gerald Schaefer, Isabella Ellinger, Rupert Ecker, Orjan Smedby, Chunliang Wang, Benjamin Chidester, That-Vinh Ton, Minh-Triet Tran, Jian Ma, Minh N. Do, Simon Graham, Quoc Dang Vu, Jin Tae Kwak, Akshaykumar Gunda, Raviteja Chunduri, Corey Hu, Xiaoyang Zhou, Dariush Lotfi, Reza Safdari, Antanas Kascenas, Alison O’Neil, Dennis Eschweiler, Johannes Stegmaier, Yanping Cui, Baocai Yin, Kailin Chen, Xinmei Tian, Philipp Gruening, Erhardt Barth, Elad Arbel, Itay Remer, Amir Ben-Dor, Ekaterina Sirazitdinova, Matthias Kohl, Stefan Braunewell, Yuexiang Li, Xinpeng Xie, Linlin Shen, Jun Ma, Krishanu Das Bakshi, Mohammad Azam Khan, Jaegul Choo, Adrián Colomer, Valery Naranjo, Linmin Pei, Khan M. Iftekharuddin, Kaushiki Roy, Debotosh Bhattacharjee, Anibal Pedraza, Maria Gloria Bueno, Sabarinathan Devanathan, Saravanan Radhakrishnan, Praveen Koduganty, Zihan Wu, Guanyu Cai, Xiaojie Liu, Yuqin Wang, and Amit Sethi. A Multi-Organ Nucleus Segmentation Challenge. *IEEE Transactions on Medical Imaging*, 39(5):1380–1391, May 2020. [31](#)
- [94] Joy Buolamwini and Timnit Gebru. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, pages 77–91. PMLR, January 2018. [34](#), [134](#)
- [95] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011. [34](#)
- [96] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. [arXiv:1512.00567 \[cs\]](https://arxiv.org/abs/1512.00567), December 2015. [35](#), [91](#), [166](#)

-
- [97] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When Does Label Smoothing Help?, June 2020. 35
- [98] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. Mathematics for Machine Learning. Cambridge University Press, April 2020. 35
- [99] Andres Rodriguez. Deep Learning Systems: Algorithms, Compilers, and Processors for Large-Scale Production. Springer International Publishing, Cham, 2021. 36
- [100] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models, June 2022. 36, 38
- [101] YouTube. Predictive learning, nips 2016 | yann lecun, facebook research. <https://youtu.be/Ount2Y4qxQo>, Aug 2017. [Online; accessed 26-October-2022]. 37
- [102] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762 [cs], December 2017. 38, 40, 41, 88, 91, 93, 134, 166
- [103] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs], May 2019. 38, 93, 114
- [104] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 38
- [105] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs], October 2020. 39, 88, 91, 134
- [106] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context Encoders: Feature Learning by Inpainting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2536–2544, 2016. 39
- [107] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. ACM Transactions on Graphics, 36(4):107:1–107:14, July 2017. 39
- [108] Xiaoqiang Zhou, Junjie Li, Zilei Wang, Ran He, and Tieniu Tan. Image Inpainting with Contrastive Relation Network. In 2020 25th International Conference on Pattern Recognition (ICPR), pages 4420–4427, January 2021. 39
- [109] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709 [cs, stat], February 2020. 40, 41, 93, 100

Bibliography

- [110] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. [arXiv:1807.03748 \[cs, stat\]](https://arxiv.org/abs/1807.03748), January 2019. [40](#), [41](#), [91](#), [93](#), [100](#)
- [111] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. [arXiv:1911.05722 \[cs\]](https://arxiv.org/abs/1911.05722), March 2020. [41](#), [88](#), [93](#), [114](#)
- [112] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big Transfer (BiT): General Visual Representation Learning. [arXiv:1912.11370 \[cs\]](https://arxiv.org/abs/1912.11370), May 2020. [42](#), [93](#)
- [113] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In [2017 IEEE International Conference on Computer Vision \(ICCV\)](#), pages 843–852, October 2017. [42](#), [90](#), [91](#)
- [114] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-Adversarial Training of Neural Networks, May 2016. [42](#)
- [115] Lilian Weng. What are diffusion models? lilianweng.github.io, Jul 2021. [44](#)
- [116] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis, February 2019. [45](#)
- [117] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. [arXiv:1812.04948 \[cs, stat\]](https://arxiv.org/abs/1812.04948), March 2019. [45](#)
- [118] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN, March 2020. [45](#)
- [119] Diederik P. Kingma and Max Welling. An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. [46](#), [100](#), [101](#)
- [120] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. November 2016. [47](#), [48](#), [100](#)
- [121] Xi Chen, Yan Duan, Rein Houthoofd, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. [arXiv:1606.03657 \[cs, stat\]](https://arxiv.org/abs/1606.03657), June 2016. [48](#), [100](#)
- [122] Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of Julesz Ensembles and FRAME Models. *International Journal of Computer Vision*, 38(3):247–265, July 2000. [49](#)
- [123] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis. [arXiv:1701.02096 \[cs\]](https://arxiv.org/abs/1701.02096), November 2017. [49](#)

-
- [124] Fritz Mario, Hayman Eric, and Caputo Barbara. The kth-tips and kth-tips2 image databases. <https://www.csc.kth.se/cvap/databases/kth-tips/index.html>, 1999. 49, 100, 115
- [125] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative Unsupervised Feature Learning with Exemplar Convolutional Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(9):1734–1747, September 2016. 50, 93
- [126] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture Synthesis Using Convolutional Neural Networks. [arXiv:1505.07376 \[cs, q-bio\]](https://arxiv.org/abs/1505.07376), November 2015. 50, 76, 94
- [127] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chellappa, and Matti Pietikäinen. From BoW to CNN: Two Decades of Texture Representation for Texture Classification. *International Journal of Computer Vision*, 127(1):74–109, January 2019. 50
- [128] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling Perceptual Factors in Neural Style Transfer. [arXiv:1611.07865 \[cs\]](https://arxiv.org/abs/1611.07865), May 2017. 51, 76
- [129] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural Style Transfer: A Review. [arXiv:1705.04058 \[cs, eess, stat\]](https://arxiv.org/abs/1705.04058), October 2018. 51
- [130] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying Neural Style Transfer. [arXiv:1701.01036 \[cs\]](https://arxiv.org/abs/1701.01036), July 2017. 51, 76, 103
- [131] Seongbin Lim, Xingjian Zhang, Emmanuel Beaufrepaire, and Anatole Chessel. BioImageLoader: Easy Handling of Bioimage Datasets for Machine Learning. <https://arxiv.org/abs/2303.02158v1>, March 2023. 58, 64
- [132] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. *imgaug*. <https://github.com/aleju/imgaug>, 2020. Online; accessed 01-Feb-2020. 62
- [133] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albuumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. 62, 68
- [134] Annika Reinke, Minu D. Tizabi, Carole H. Sudre, Matthias Eisenmann, Tim Rädtsch, Michael Baumgartner, Laura Acion, Michela Antonelli, Tal Arbel, Spyridon Bakas, Peter Bankhead, Arriel Benis, M. Jorge Cardoso, Veronika Cheplygina, Beth Cimini, Gary S. Collins, Keyvan Farahani, Ben Glocker, Patrick Godau, Fred Hamprecht, Daniel A. Hashimoto, Doreen Heckmann-Nötzel, Michael M. Hoffmann, Merel Huisman, Fabian Isensee, Pierre Jannin, Charles E. Kahn, Alexandros Karargyris, Alan Karthikesalingam, Bernhard Kainz, Emre Kavur, Hannes Kennigott, Jens Kleesiek, Thijs Kooi, Michal Kozubek, Anna Kreshuk, Tahsin Kurc, Bennett A.

Bibliography

- Landman, Geert Litjens, Amin Madani, Klaus Maier-Hein, Anne L. Martel, Peter Mattson, Erik Meijering, Bjoern Menze, David Moher, Karel G. M. Moons, Henning Müller, Felix Nickel, Jens Petersen, Gorkem Polat, Nasir Rajpoot, Mauricio Reyes, Nicola Rieke, Michael Riegler, Hassan Rivaz, Julio Saez-Rodriguez, Clarisa Sanchez Gutierrez, Julien Schroeter, Anindo Saha, Shravya Shetty, Bram Stieltjes, Ronald M. Summers, Abdel A. Taha, Sotirios A. Tsafaris, Bram van Ginneken, Gaël Varoquaux, Manuel Wiesenfarth, Ziv R. Yaniv, Annette Kopp-Schneider, Paul Jäger, and Lena Maier-Hein. Common Limitations of Image Processing Metrics: A Picture Story. [arXiv:2104.05642 \[cs, eess\]](https://arxiv.org/abs/2104.05642), April 2022. [63](#), [177](#)
- [135] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In [NIPS-W](#), 2017. [69](#), [134](#), [164](#)
- [136] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. [Nature](#), 585(7825):357–362, September 2020. [72](#), [74](#)
- [137] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. [arXiv:1405.0312 \[cs\]](https://arxiv.org/abs/1405.0312), February 2015. [73](#), [87](#), [90](#), [96](#), [146](#), [178](#)
- [138] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In [2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition \(CVPR'05\)](#), volume 1, pages 886–893 vol. 1, June 2005. [76](#)
- [139] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. [International Journal of Computer Vision](#), 60(2):91–110, November 2004. [76](#)
- [140] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks, November 2013. [76](#), [137](#), [167](#), [168](#), [169](#)
- [141] Aravindh Mahendran and Andrea Vedaldi. Understanding Deep Image Representations by Inverting Them. November 2014. [76](#), [94](#), [169](#), [170](#)
- [142] Mohammad Babaeizadeh and Golnaz Ghiasi. Adjustable Real-time Style Transfer. In [International Conference on Learning Representations](#), September 2019. [76](#), [95](#), [116](#)
- [143] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. [arXiv:1705.06830 \[cs\]](https://arxiv.org/abs/1705.06830), August 2017. [76](#), [92](#), [94](#)
- [144] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. [arXiv:1802.03426 \[cs, stat\]](https://arxiv.org/abs/1802.03426), December 2018. [76](#), [103](#)

-
- [145] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. Journal of Machine Learning Research, 9(86):2579–2605, 2008. [76](#)
- [146] Antti Lehmussola, Pekka Ruusuvuori, Jyrki Selinummi, Heikki Huttunen, and Olli Yli-Harja. Computational Framework for Simulating Fluorescence Microscope Images With Cell Populations. IEEE Transactions on Medical Imaging, 26(7):1010–1016, July 2007. [77](#), [174](#)
- [147] Antti Lehmussola, Pekka Ruusuvuori, Jyrki Selinummi, Tiina Rajala, and Olli Yli-Harja. Synthetic Images of High-Throughput Microscopy for Validation of Image Analysis Methods. Proceedings of the IEEE, 96(8):1348–1360, August 2008. [77](#), [174](#)
- [148] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. International Journal of Computer Vision, 128(7):1956–1981, July 2020. [87](#), [90](#)
- [149] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3):211–252, December 2015. [90](#), [96](#), [166](#)
- [150] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. arXiv:1605.07146 [cs], June 2017. [91](#), [166](#)
- [151] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. arXiv:2201.03545 [cs], March 2022. [91](#), [166](#)
- [152] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06), volume 2, pages 1735–1742, June 2006. [91](#)
- [153] Sebastian Thrun. Lifelong learning algorithms. In Learning to Learn, pages 181–209. Kluwer Academic Publishers, USA, May 1998. [91](#), [92](#)
- [154] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, October 2013. [91](#), [92](#)
- [155] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features off-the-shelf: An Astounding Baseline for Recognition, May 2014. [91](#), [92](#), [93](#)
- [156] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. arXiv:1603.03417 [cs], March 2016. [92](#)

Bibliography

- [157] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. [arXiv:1607.08022 \[cs\]](#), November 2017. [92](#), [98](#), [99](#)
- [158] Leon A. Gatys, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Preserving Color in Neural Artistic Style Transfer. [arXiv:1606.05897 \[cs\]](#), June 2016. [92](#)
- [159] Chuan Li and Michael Wand. Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis. [arXiv:1601.04589 \[cs\]](#), January 2016. [92](#)
- [160] Eric Risser, Pierre Wilmot, and Connelly Barnes. Stable and Controllable Neural Texture Synthesis and Style Transfer Using Histogram Losses. [arXiv:1701.08893 \[cs\]](#), February 2017. [92](#)
- [161] Xun Huang and Serge Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. [arXiv:1703.06868 \[cs\]](#), July 2017. [92](#), [116](#)
- [162] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal Style Transfer via Feature Transforms. [arXiv:1705.08086 \[cs\]](#), November 2017. [92](#)
- [163] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved Baselines with Momentum Contrastive Learning. [arXiv:2003.04297 \[cs\]](#), March 2020. [93](#)
- [164] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks?, November 2014. [93](#)
- [165] Ryan Conrad and Kedar Narayan. CEM500K – A large-scale heterogeneous unlabeled cellular electron microscopy image dataset for deep learning. [bioRxiv](#), page 2020.12.11.421792, December 2020. [93](#)
- [166] Emmanuel Bouilhol, Edgar Lefevre, Thierno Barry, Florian Lrevet, Anne Beghin, Virgile Viasnoff, Xareni Galindo, Remi Galland, Jean-Baptiste Sibarita, and Macha Nikolski. SaliencyNet: An unsupervised Image-to-Image translation method for nuclei saliency enhancement in microscopy images, October 2022. [93](#), [94](#), [115](#)
- [167] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. [arXiv:1703.10593 \[cs\]](#), March 2017. [93](#)
- [168] Michal Januszewski and Viren Jain. Segmentation-Enhanced CycleGAN. Preprint, Neuroscience, February 2019. [94](#)
- [169] G. Berger and R. Memisevic. Incorporating long-range consistency in CNN-based texture generation. [arXiv:1606.01286 \[cs\]](#), November 2016. [94](#)
- [170] Alexey Dosovitskiy and Thomas Brox. Generating Images with Perceptual Similarity Metrics based on Deep Networks. [arXiv:1602.02644 \[cs\]](#), February 2016. [95](#)
- [171] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination. [arXiv:1805.01978 \[cs\]](#), May 2018. [95](#)

-
- [172] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3431–3440, June 2015. [96](#), [98](#), [114](#)
- [173] Lee Kamentsky, Thouis R. Jones, Adam Fraser, Mark-Anthony Bray, David J. Logan, Katherine L. Madden, Vebjorn Ljosa, Curtis Rueden, Kevin W. Eliceiri, and Anne E. Carpenter. Improved structure, function and compatibility for CellProfiler: Modular high-throughput image analysis software. Bioinformatics, 27(8):1179–1180, April 2011. [98](#)
- [174] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. arXiv:1711.05101 [cs, math], January 2019. [98](#)
- [175] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs], January 2016. [98](#)
- [176] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs], March 2015. [98](#), [99](#), [120](#)
- [177] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a Definition of Disentangled Representations. arXiv:1812.02230 [cs, stat], December 2018. [100](#)
- [178] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations. arXiv:1711.00848 [cs, stat], December 2018. [100](#)
- [179] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc’Aurelio Ranzato. Fader Networks: Manipulating Images by Sliding Attributes. arXiv:1706.00409 [cs], January 2018. [100](#)
- [180] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating Diverse High-Fidelity Images with VQ-VAE-2. arXiv:1906.00446 [cs, stat], June 2019. [101](#), [134](#)
- [181] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models, April 2016. [101](#)
- [182] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. arXiv:1601.06759 [cs], August 2016. [101](#)
- [183] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders. arXiv:1606.05328 [cs], June 2016. [101](#)
- [184] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation. arXiv:1502.03509 [cs, stat], June 2015. [101](#)
- [185] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. arXiv:1505.05770 [cs, stat], June 2016. [101](#)

Bibliography

- [186] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions. [arXiv:1807.03039 \[cs, stat\]](#), July 2018. 101
- [187] Prashna K. Gyawali, Rudra Saha, Linwei Wang, V. S. R. Veeravasaru, and Maneesh Singh. Wavelets to the Rescue: Improving Sample Quality of Latent Variable Deep Generative Models. [arXiv:1911.05627 \[cs, eess, stat\]](#), October 2019. 101
- [188] Rui Shu, Yining Chen, Abhishek Kumar, Stefano Ermon, and Ben Poole. Weakly Supervised Disentanglement with Guarantees. [arXiv:1910.09772 \[cs, stat\]](#), October 2019. 101
- [189] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. [arXiv:1811.12359 \[cs, stat\]](#), June 2019. 101
- [190] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. [arXiv:1801.04381 \[cs\]](#), March 2019. 101, 102, 166
- [191] Yang Song and Stefano Ermon. Generative Modeling by Estimating Gradients of the Data Distribution, October 2020. 115, 134
- [192] Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. [arXiv:2009.09796 \[cs, stat\]](#), September 2020. 116
- [193] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In [International Conference on Machine Learning](#), pages 1139–1147, February 2013. 116
- [194] Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks. [arXiv:1506.01186 \[cs\]](#), April 2017. 116
- [195] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. [arXiv:1608.03983 \[cs, math\]](#), May 2017. 116
- [196] Corinne Vioux-Chagnoleau, François Lejeune, Juliette Sok, Cécile Pierrard, Claire Marionnet, and Françoise Bernerd. Reconstructed human skin: From photodamage to sunscreen photoprotection and anti-aging molecules. [Journal of Dermatological Science Supplement](#), 2(1):S1–S12, December 2006. 119
- [197] Chiara Stringari, Lamiae Abdeladim, Guy Malkinson, Pierre Mahou, Xavier Solinas, Isabelle Lamarre, Sébastien Brizion, Jean-Baptiste Galey, Willy Supatto, Renaud Legouis, Ana-Maria Pena, and Emmanuel Beaufrepère. Multicolor two-photon imaging of endogenous fluorophores in living tissues by wavelength mixing. [Scientific Reports](#), 7(1):3792, June 2017. 119
- [198] Chiara Stringari, Amanda Cinquin, Olivier Cinquin, Michelle A. Digman, Peter J. Donovan, and Enrico Gratton. Phasor approach to fluorescence lifetime microscopy distinguishes different metabolic states of germ cells in a live tissue. [Proceedings of the National Academy of Sciences](#), 108(33):13582–13587, August 2011. 120

- [199] Robert Haase, Draga Doncila Pop, and Laura Žigutytė. Haesleinhuepf/napari-segment-blobs-and-things-with-membranes: 0.3.2. Zenodo, August 2022. 127
- [200] Alistair Miles, John Kirkham, Martin Durant, James Bourbeau, Tarik Onalan, Joe Hamman, Zain Patel, shikharsg, Matthew Rocklin, raphael dussin, Vincent Schut, Elliott Sales de Andrade, Ryan Abernathy, Charles Noyes, sbalmer, pyup io bot, Tommy Tran, Stephan Saalfeld, Justin Swaney, Josh Moore, Joe Jevnik, Jerome Kelleher, Jan Funke, George Sakkis, Chris Barnes, and Anderson Banihirwe. Zarr-developers/zarr-python: V2.4.0. Zenodo, January 2020. 128
- [201] Demis Hassabis. Alphafold reveals the structure of the protein universe. <https://www.deepmind.com/blog/alphafold-reveals-the-structure-of-the-protein-universe>, July 2022. 133
- [202] Andrew W. Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Židek, Alexander W. R. Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, Pushmeet Kohli, David T. Jones, David Silver, Koray Kavukcuoglu, and Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, January 2020. 133
- [203] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation, February 2021. 133
- [204] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision, February 2021. 134
- [205] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models, March 2022. 134
- [206] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, and Scott Gray. Dall·e: Creating images from text. <https://openai.com/blog/dall-e/>, January 2021. 135
- [207] Guy Parsons. The dall·e 2 prompt book. <https://dallery.gallery/the-dalle-2-prompt-book/>, July 2022. 135
- [208] Tom Simonite. What Really Happened When Google Ousted Timnit Gebru. *Wired*, June 2022. 134
- [209] Tom Simonite. A Second AI Researcher Says She Was Fired by Google. *Wired*, February 2021. 134
- [210] Google fires Margaret Mitchell, another top researcher on its AI ethics team. *The Guardian*, February 2021. 134
- [211] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528, June 2011. 134

Bibliography

- [212] Yilun Zhou, Serena Booth, Marco Tulio Ribeiro, and Julie Shah. Do Feature Attribution Methods Correctly Attribute Features?, December 2021. [134](#)
- [213] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for PyTorch, September 2020. [134](#)
- [214] Narine Kokhlikyan, Vivek Miglani, Bilal Alsallakh, Miguel Martin, and Orion Reblitz-Richardson. Investigating sanity checks for saliency maps with image and text classification, June 2021. [134](#)
- [215] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. [136](#)
- [216] Simon Willison. Prompt injection attacks against gpt-3. <https://simonwillison.net/2022/Sep/12/prompt-injection/>, September 2022. [136](#)
- [217] Matthew Butterick. We’ve filed a lawsuit challenging github copilot, an ai product that relies on unprecedented open-source software piracy. because ai needs to be fair & ethical for everyone. <https://githubcopilotlitigation.com/>, November 2022. [136](#)
- [218] Neil Savage. How AI and neuroscience drive each other forwards. *Nature*, 571(7766):S15–S17, July 2019. [136](#)
- [219] Blake A. Richards, Timothy P. Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J. Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W. Lindsay, Kenneth D. Miller, Richard Naud, Christopher C. Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C. Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P. Kording. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, November 2019. [136](#)
- [220] Charlotte Caucheteux and Jean-Rémi King. Brains and algorithms partially converge in natural language processing. *Communications Biology*, 5(1):1–10, February 2022. [136](#)

-
- [221] Zijiao Chen, Jiaxin Qing, Tiange Xiang, Wan Lin Yue, and Juan Helen Zhou. Seeing Beyond the Brain: Conditional Diffusion Model with Sparse Masked Modeling for Vision Decoding, November 2022. [137](#)
- [222] David Eagleman. The Brain: The Story of You. Canongate Books, November 2015. [137](#)
- [223] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers Is Solved. Science, 317(5844):1518–1522, September 2007. [143](#)
- [224] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, 88(2):303–338, June 2010. [146](#), [178](#)
- [225] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, November 1998. [147](#), [156](#)
- [226] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv:1603.07285 [cs, stat], January 2018. [149](#)
- [227] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, November 2016. [153](#), [167](#)
- [228] YouTube. Richard feynman: Can machines think? <https://youtu.be/ipRvjS7q1DI>, Nov 2019. [Online; accessed 26-October-2022]. [153](#)
- [229] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, pages 807–814, Madison, WI, USA, June 2010. Omnipress. [155](#), [162](#)
- [230] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. arXiv:1712.09913 [cs, stat], November 2018. [159](#)
- [231] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs], January 2017. [160](#)
- [232] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. [163](#)
- [233] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye

Bibliography

- Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. [164](#)
- [234] Haris Iqbal. HarisIqbal88/PlotNeuralNet v1.0.0. Zenodo, December 2018. [165](#)
- [235] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July 2006. [165](#)
- [236] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, September 2014. [166](#), [171](#)
- [237] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587 [cs]*, December 2017. [166](#)
- [238] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, June 2019. [166](#)
- [239] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, April 2017. [166](#)
- [240] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for MobileNetV3, November 2019. [166](#)
- [241] Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller Models and Faster Training, June 2021. [166](#)
- [242] Inceptionism: Going Deeper into Neural Networks. [170](#)
- [243] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2(11):e7, November 2017. [170](#), [171](#)
- [244] Korsuk Sirinukunwattana, Shan E Ahmed Raza, Yee-Wah Tsang, David R. J. Snead, Ian A. Cree, and Nasir M. Rajpoot. Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images. *IEEE Transactions on Medical Imaging*, 35(5):1196–1206, May 2016. [174](#)
- [245] L. P. Coelho, A. Shariff, and R. F. Murphy. Nuclear segmentation in microscope cell images: A hand-segmented dataset and comparison of algorithms. In *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 518–521, June 2009. [174](#)
- [246] Estibaliz Gómez-de-Mariscal, Martin Maška, Anna Kotrbová, Vendula Pospíchalová, Pavel Matula, and Arrate Muñoz-Barrutia. Deep-Learning-Based Segmentation of Small Extracellular Vesicles in Transmission Electron Microscopy Images. *Scientific Reports*, 9(1):13211, September 2019. [174](#)
- [247] Wikipedia contributors. F-score — Wikipedia, the free encyclopedia, 2022. [Online; accessed 26-October-2022]. [179](#)

Titre : Apprentissage statistique versatile pour l'imagerie neurodéveloppementale

Mots clés : Imagerie neurodéveloppementale, Modèle d'apprentissage automatique versatile, Analyse des bioimages, Apprentissage auto-supervisé, Défis liés aux données

Résumé : Cette thèse se concentre sur le développement de modèles polyvalents d'apprentissage automatique pour l'imagerie neurodéveloppementale. Elle couvre divers aspects, notamment les techniques d'imagerie neurodéveloppementale, l'informatique des images biologiques et les approches d'apprentissage profond. L'apprentissage profond émerge comme une solution supérieure pour l'analyse d'images biologiques, surpassant les méthodes traditionnelles. Cette thèse aborde les limites des solutions actuelles, en particulier les difficultés de l'apprentissage supervisé, qui nécessite des ensembles de données entièrement curatés. Elle explore l'évolution de l'apprentissage automatique, y compris les tendances dans les tâches de vision par ordinateur et l'importance de l'apprentissage auto-supervisé, qui, stimulé par des mécanismes d'auto-attention, a transformé le do-

main, notamment dans le traitement du langage naturel. L'objectif central de la thèse est de développer des modèles d'analyse d'images polyvalents pour les images biologiques par le biais de l'apprentissage auto-supervisé, en traitant les limitations persistantes des données. Les contributions incluent la création d'un méta-ensemble de données, le développement de la bibliothèque Python "bioimageloader" pour une gestion efficace des ensembles de données, la proposition d'une nouvelle perte auto-supervisée et du modèle "NU-Net" résultant, ainsi que la création d'outils logiciels pour l'intégration de l'apprentissage automatique dans les pipelines de traitement d'images. En résumé, cette thèse vise à faire progresser l'imagerie neurodéveloppementale grâce à un apprentissage automatique auto-supervisé polyvalent, en traitant les défis liés aux données et en favorisant la convivialité et l'accessibilité.

Title : Versatile machine learning for neurodevelopmental imaging

Keywords : Neurodevelopmental imaging, Versatile machine learning model, Bioimage analysis, Self-supervised learning, Data challenges

Abstract : This thesis focuses on developing versatile machine learning models for neurodevelopmental imaging. It covers various aspects, including neurodevelopmental imaging techniques, bioimage informatics, and deep learning approaches. Neurodevelopmental imaging involves capturing data with varying resolutions, from cell nuclei to complex microscopic structures, using specialized microscopy techniques. Bioimage informatics deals with the challenges of bioimage analysis, with a primary focus on image segmentation as a critical task for qualitative analysis. Deep learning emerges as a superior solution for bioimage analysis, outperforming traditional methods. It has led to the development of various models and datasets, enhancing bioimage segmentation, restoration, and denoising. This has also increased the focus on software tools for model distribution. The thesis addresses the limitations and challenges of current solutions, particularly the difficulties associated with supervised learning, which requires fully curated datasets. The evolution of machine learning is explo-

red, including trends in computer vision tasks and the pursuit of unsupervised learning, with a specific focus on self-supervised learning. Self-supervised learning, driven by self-attention mechanisms, has transformed deep learning, especially in natural language processing. It has shown transformative potential and paved the way for large language models. The core objective of the thesis is to develop versatile image analysis models for bioimages through self-supervised learning, addressing persistent data limitations. Contributions include creating a meta-dataset, developing the "bioimageloader" Python library for efficient dataset management, proposing a novel self-supervised loss and resulting "NU-Net" model, and building software tools for machine learning integration in image processing pipelines. Collaborations with microscopists and biologists have led to the development of applications. In summary, this thesis aims to advance neurodevelopmental imaging through versatile self-supervised machine learning, addressing data-related challenges and promoting usability and accessibility.