



HAL
open science

Segmentation et détection de lésions cérébrales combinant apprentissage profond et factorisation à partir d'IRM multi-paramétriques

Pooya Ashtari

► **To cite this version:**

Pooya Ashtari. Segmentation et détection de lésions cérébrales combinant apprentissage profond et factorisation à partir d'IRM multi-paramétriques. Imagerie médicale. Université Claude Bernard - Lyon I; KU Leuven (1970-..), 2023. Français. NNT : 2023LYO10225 . tel-04733178

HAL Id: tel-04733178

<https://theses.hal.science/tel-04733178v1>

Submitted on 11 Oct 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.



**THESE de DOCTORAT DE
L'UNIVERSITE CLAUDE BERNARD LYON 1
en cotutelle avec
KU LEUVEN - Belgique**

Ecole Doctorale N° 205
Interdisciplinaire Sciences Santé (EDISS)

Discipline :
Santé publique, Recherche clinique, Innovation thérapeutique et
diagnostique

Soutenue publiquement le 21/11/2023, par :
Pooya ASHTARI

**Brain Lesion Segmentation and
Detection on Multi-parametric MRI Data**

Marrying Deep Learning with Low-rank Factorization

Devant le jury composé de :

Laura-Adela Harsan	Maître de Conférences – Praticien Hospitalier, Université de Strasbourg	Rapporteur
Josien P. W. Pluim	Professeur, Eindhoven University of Technology	Rapporteur
Lieven De Lathauwer	Professeur, KU Leuven	Président
Thomas Grenier	Maître de Conférences des Universités, INSA Lyon	Examineur
Delphine Maucort-Boulch	Professeur des Universités, Université Claude Bernard Lyon 1	Examineur
Sabine Van Huffel	Professeure émérite, KU Leuven	Directeur
Dominique Sappey-Marinier	Maître de Conférences – Praticien Hospitalier, Université Claude Bernard - Lyon 1	Directeur
Frederik Maes	Professeur, KU Leuven	Co-directeur



Brain Lesion Segmentation and Detection on Multi-parametric MRI Data

Marrying Deep Learning with Low-rank Factorization

Pooya Ashtari

Supervisors:

Prof. dr. ir. Sabine Van Huffel
(KU Leuven)

Prof. dr. Dominique Sappey-Marinier
(Université Claude Bernard Lyon 1)

Prof. dr.ir. Frederik Maes
(KU Leuven)

Dissertation presented in partial
fulfillment of the requirements
for the degree of Doctor of
Engineering Science (PhD):
Electrical Engineering

November 2023

© 2023 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Pooya Ashtari, Kasteelpark Arenberg 10, box 2446, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Abstract

Brain lesion segmentation plays a crucial role in clinical neuroimaging, aiding in diagnosis, treatment planning, disease monitoring, and research. Accurate identification of lesion location and extent empowers clinicians to make informed decisions about patient care and provides researchers insights into the mechanisms of these conditions. Magnetic resonance imaging (MRI) is the primary modality for diagnosing and assessing brain lesions. A combination of MRI sequences, known as multiparametric MRI (mpMRI), is often employed to provide a comprehensive assessment of the lesion tissue under study. This technique capitalizes on the unique tissue characteristics highlighted by each sequence, offering a richer representation than any single sequence alone could provide.

However, manual segmentation of brain lesions in mpMRI data is time-consuming, costly, and subjective, with potential variability between observers. Automated methods ensure consistent, reproducible results and eliminate both inter- and intra-observer variability, which is essential for longitudinal studies and multi-center trials. While deep learning models, particularly convolutional neural networks (CNNs), have revolutionized image segmentation, several challenges remain in their application to brain lesion segmentation in clinical practice. These include data scarcity, variability, class imbalance, high computational requirements, and the “black box” nature of deep learning models.

This thesis aims to develop efficient data-driven models for automated segmentation and detection of brain lesions, specifically brain tumors, Multiple Sclerosis (MS) lesions, and stroke lesions, using mpMRI. By fusing advanced deep learning with low-rank factorization techniques, we introduce a diagnostic tool that segments and detects brain lesions accurately and interpretably without added computational complexity.

In the first part of this thesis, we focus on the application of CNNs for the automated segmentation of new MS lesions in 3D FLAIR images. Our goal is to

identify new lesions between two longitudinal MRI scans of an MS patient, which is a key indicator of disease progression. We introduce Pre-U-Net, a 3D encoder-decoder architecture consisting of pre-activation residual blocks. To mitigate limited training data and class imbalance, we employ data augmentation and deep supervision for optimal model training. Comparative analysis reveals that Pre-U-Net outperforms both U-Net and Res-U-Net on the MSSEG-2 dataset.

The centerpiece of this thesis presents a novel method that integrates low-rank factorization with deep learning for enhanced medical image segmentation. Recognizing the limitations of CNNs in exploiting global context and the quadratic complexity of attention in transformers, we introduce a family of models, called *Factorizer*, which leverage the power of low-rank matrix factorization to construct a scalable and interpretable segmentation model. More specifically, we formulate nonnegative matrix factorization (NMF) as a differentiable layer and incorporate it into a U-shaped architecture. Moreover, we use the shifted window technique in combination with NMF to effectively aggregate local information. Our results indicate that Factorizers outshine CNNs and transformers in terms of accuracy, complexity, and interpretability, setting new benchmarks on the BraTS and ISLES'22 datasets. Notably, our experiments show that NMF components are highly meaningful, with each component highlighting specific regions, offering Factorizers a unique edge in interpretability over CNNs and Transformers. Moreover, our ablation studies reveal a distinctive feature of Factorizers that allows a significant speed-up in inference for a trained Factorizer model, without requiring additional steps or significant accuracy trade-offs.

In the final part, we propose a method utilizing low-rank tensor networks to enhance CNNs for brain tumor segmentation. Given that many effective 3D CNNs are prone to overfitting due to their complexity and limited training data, we introduce a 3D U-Net-like architecture integrated with residual blocks. By imposing low-rank constraints on convolutional layer weights, we aim to avoid overfitting. This approach allows the creation of networks with considerably fewer parameters. We assess our method performance in the BraTS 2020 challenge data.

Beknopte samenvatting

Hersenletsel segmentatie speelt een cruciale rol in klinische neurobeeldvorming, als ondersteuning bij diagnose, behandeling, ziektemonitoring en onderzoek. Nauwkeurige identificatie van de locatie en omvang van het letsel stelt klinici in staat geïnformeerde beslissingen te nemen over patiëntenzorg en biedt onderzoekers inzicht in de mechanismen van deze aandoeningen. Magnetische resonantie beeldvorming (MRI) is de primaire modaliteit voor het diagnosticeren en beoordelen van hersenletsels. Een combinatie van MRI-sequenties, bekend als multiparametrische MRI (mpMRI), wordt vaak gebruikt voor een uitgebreide beoordeling van het weefsel van het letsel. Deze techniek maakt gebruik van de unieke weefseleigenschappen die door elke sequentie worden benadrukt, wat een rijker beeld biedt dan een enkele sequentie alleen zou kunnen bieden.

Echter, handmatige segmentatie van hersenletsels in mpMRI-gegevens is tijdrovend, kostbaar en subjectief, met mogelijke variabiliteit tussen waarnemers. Geautomatiseerde methoden zorgen voor consistente, reproduceerbare resultaten en elimineren zowel inter- als intra-waarnemer variabiliteit, wat essentieel is voor longitudinale studies en multicenterproeven. Hoewel modellen voor diep leren, met name convolutionele neurale netwerken (CNN's), beeldsegmentatie hebben gerevolutioneerd, blijven er uitdagingen bestaan bij hun toepassing op hersenletsel segmentatie in de klinische praktijk. Deze omvatten gegevensschaarste, variabiliteit, klasse-onbalans, hoge computationele vereisten en zwarte dooskarakteristieken van modellen voor diep leren.

Dit proefschrift beoogt efficiënte datagedreven modellen te ontwikkelen voor geautomatiseerde segmentatie en detectie van hersenletsels, met name hersentumoren, Multiple Sclerose (MS) letsels en beroerteletsels, met behulp van mpMRI. Door geavanceerd diep leren te combineren met lage-rang factorisatietechnieken introduceren we een diagnostisch hulpmiddel dat hersenletsels nauwkeurig en interpreteerbaar segmenteert en detecteert zonder toegevoegde computationele complexiteit.

In het eerste deel van dit proefschrift richten we ons op de toepassing van CNN's voor de geautomatiseerde segmentatie van nieuwe MS-letsels in 3D FLAIR-beelden. Ons doel is het identificeren van nieuwe letsels tussen twee longitudinale MRI-scans van een MS-patiënt, wat een belangrijke indicator is voor ziekteprogressie. We introduceren Pre-U-Net, een 3D encoder-decoder architectuur bestaande uit pre-activatie restblokken. Om beperkte trainingsgegevens en klasse-onbalans te verminderen, gebruiken we data-augmentatie en diep toezicht voor optimale modeltraining. Vergelijkende analyse toont aan dat Pre-U-Net zowel U-Net als Res-U-Net overtreft op de MSSEG-2 dataset.

Het centrale deel van dit proefschrift presenteert een nieuwe methode die lage-rang factorisatie integreert met diep leren voor verbeterde medische beeldsegmentatie. Gezien de beperkingen van CNN's in het benutten van globale context en de kwadratische complexiteit van aandacht in transformatoren, introduceren we een familie van modellen, genaamd *Factorizer*, die de kracht van lage-rang matrix factorisatie benutten om een schaalbaar en interpreteerbaar segmentatiemodel te construeren. Meer specifiek formuleren we niet-negatieve matrix factorisatie (NMF) als een differentieerbare laag en incorporeren deze in een U-vormige architectuur. Bovendien gebruiken we de verschoven venstertechniek in combinatie met NMF om effectief lokale informatie te aggregeren. Onze resultaten geven aan dat Factorizers beter presteren dan CNN's en transformatoren qua nauwkeurigheid, complexiteit en interpretatie, en zetten nieuwe benchmarks voor de BraTS en ISLES'22 datasets. Opmerkelijk is dat onze experimenten aantonen dat NMF-componenten zeer betekenisvol zijn, waarbij elk component specifieke regio's belicht, waardoor Factorizers een uniek voordeel hebben in interpretatie ten opzichte van CNN's en Transformatoren. Bovendien onthullen onze ablatiestudies een onderscheidend kenmerk van Factorizers dat een aanzienlijke versnelling in inferentie voor een getraind Factorizer model mogelijk maakt, zonder dat extra stappen of significante nauwkeurighedscompromissen nodig zijn.

In het laatste deel stellen we een methode voor die gebruik maakt van lage-rang tensor netwerken om CNN's te verbeteren voor hersentumor segmentatie. Gezien het feit dat veel effectieve 3D CNN's gevoelig zijn voor overfitting vanwege hun complexiteit en beperkte trainingsgegevens, introduceren we een 3D U-Net-achtige architectuur geïntegreerd met restblokken. Door lage-rang beperkingen op te leggen aan de gewichten van de convolutionele laag, willen we overfitting vermijden. Deze aanpak maakt de creatie van netwerken met aanzienlijk minder parameters mogelijk. We beoordelen de prestaties van onze methode op de BraTS 2020 challenge data.

Résumé

La segmentation des lésions cérébrales joue un rôle crucial en neurologie et en neurochirurgie, aidant au diagnostic, à la planification du traitement, à la surveillance des maladies et à la recherche. Une identification précise de la localisation et de l'étendue des lésions permet aux cliniciens de prendre des décisions éclairées concernant les soins aux patients et offre aux chercheurs des perspectives sur les mécanismes de ces conditions. L'imagerie par résonance magnétique (IRM) est la modalité principale pour le diagnostic et l'évaluation des lésions cérébrales. Une combinaison de séquences IRM, connue sous le nom d'IRM multiparamétrique (mpMRI), est souvent utilisée pour fournir une évaluation complète du tissu lésionnel étudié. Cette technique tire parti des caractéristiques uniques des tissus mises en évidence par chaque séquence, offrant une représentation plus riche que ce qu'une seule séquence pourrait fournir.

Cependant, la segmentation manuelle des lésions cérébrales dans les données mpMRI est chronophage, coûteuse et subjective, avec une variabilité potentielle entre les observateurs. Les méthodes automatisées garantissent des résultats cohérents et reproductibles et éliminent à la fois la variabilité inter- et intra-observateur, ce qui est essentiel pour les études longitudinales et les essais multicentriques. Bien que les modèles d'apprentissage profond, en particulier les réseaux neuronaux convolutionnels (CNN), aient révolutionné la segmentation d'images, plusieurs défis subsistent dans leur application à la segmentation des lésions cérébrales en pratique clinique. Cela inclut la rareté des données, la variabilité, le déséquilibre des classes, les exigences computationnelles élevées et la nature "boîte noire" des modèles d'apprentissage profond.

Cette thèse vise à développer des modèles efficaces basés sur les données pour la segmentation et la détection automatisées des lésions cérébrales, en particulier des tumeurs cérébrales, des lésions de Sclérose en Plaques (SEP) et des lésions d'accident vasculaire cérébral (AVC), en utilisant mpMRI. En fusionnant l'apprentissage profond avancé avec des techniques de factorisation de rang faible, nous introduisons un outil diagnostique qui segmente et détecte les lésions

cérébrales de manière précise et interprétable sans complexité computationnelle ajoutée.

Dans la première partie de cette thèse, nous nous concentrons sur l'application des CNN pour la segmentation automatisée des nouvelles lésions de SEP dans les images 3D FLAIR. Notre objectif est d'identifier de nouvelles lésions entre deux scans IRM longitudinaux d'un patient atteint de SEP, ce qui est un indicateur clé de la progression de la maladie. Nous introduisons Pre-U-Net, une architecture encodeur-décodeur 3D composée de blocs résiduels pré-activation. Pour pallier le manque de données d'entraînement et le déséquilibre des classes, nous utilisons l'augmentation des données et la supervision profonde pour une formation optimale du modèle. Une analyse comparative révèle que Pre-U-Net surpasse à la fois U-Net et Res-U-Net sur le jeu de données MSSEG-2.

Le cœur de cette thèse présente une méthode novatrice qui intègre la factorisation de rang faible à l'apprentissage profond pour améliorer la segmentation des images médicales. Reconnaisant les limites des CNN à exploiter le contexte global et la complexité quadratique de l'attention dans les transformateurs, nous introduisons une famille de modèles, appelée "Factorizer", qui exploitent la puissance de la factorisation matricielle de rang faible pour construire un modèle de segmentation évolutif et interprétable. Plus précisément, nous formulons la factorisation matricielle non négative (NMF) comme une couche différentiable et l'intégrons dans une architecture en forme de U. De plus, nous utilisons la technique de fenêtre décalée en combinaison avec la NMF pour agréger efficacement les informations locales. Nos résultats indiquent que les Factorizers surpassent les CNN et les transformateurs en termes de précision, de complexité et d'interprétabilité, établissant de nouvelles références sur les jeux de données BraTS et ISLES'22. Notamment, nos expériences montrent que les composants NMF sont hautement significatifs, chaque composant mettant en évidence des régions spécifiques, offrant aux Factorizers un avantage unique en matière d'interprétabilité par rapport aux CNN et aux Transformateurs. De plus, nos études d'ablation révèlent une caractéristique distinctive des Factorizers qui permet une accélération significative de l'inférence pour un modèle Factorizer formé, sans nécessiter d'étapes supplémentaires ou de compromis significatifs en matière de précision.

Dans la dernière partie, nous proposons une méthode utilisant des réseaux tensoriels de rang faible pour améliorer les CNN pour la segmentation des tumeurs cérébrales. Étant donné que de nombreux CNN 3D efficaces sont sujets au surajustement en raison de leur complexité et du manque de données d'entraînement, nous introduisons une architecture de type U-Net 3D intégrée avec des blocs résiduels. En imposant des contraintes de rang faible sur les poids des couches convolutionnelles, nous visons à éviter le surajustement. Cette approche permet de créer des réseaux avec nettement moins de paramètres. Nous évaluons la performance de notre méthode sur les données du défi BraTS.

List of Abbreviations

- AdaGrad** adaptive gradient. 50
- Adam** adaptive moment estimation. 51
- ADC** apparent diffusion coefficient. 19–22, 31
- BCD** block coordinate descent. 51, 52, 101, 103, 104
- CNN** convolutional neural network. xiv, xviii, 2, 3, 6–8, 69–71, 73, 74, 77, 80–83, 85, 93, 95–97, 186, 189
- CNS** central nervous system. 22, 26, 29
- CP** canonical polyadic. 187
- CSF** cerebrospinal fluid. 17–19
- DFT** discrete Fourier transform. 17
- DNN** deep neural network. 120, 121
- DWI** diffusion-weighted imaging. 6, 18, 20–22, 31
- ERM** empirical risk minimization. 37–40
- FAVOR+** fast attention via positive orthogonal random features. 91
- FID** free induction decay. 15
- FLAIR** fluid-attenuated inversion recovery. 6, 7, 17, 18, 20, 22, 31, 185
- GBM** glioblastoma multiforme. 25

- HALS** hierarchical alternating least squares. xxi, 7, 104–108
- HD** Hausdorff distance. 186
- iid** independent and identically distributed. 35–39, 58
- KDE** kernel density estimation. 58, 60–62
- MBGD** mini-batch gradient descent. 49
- MHA** multi-Head attention. 86, 87
- MLE** maximum likelihood estimation. 38, 39, 54, 56, 58
- MLP** multilayer perception. 66–70, 73, 80, 84, 87, 88, 90
- MM** majorization-minimization. 53
- mpMRI** multiparametric MRI. 1, 3, 31, 185, 187
- MRI** magnetic resonance imaging. 1, 6, 7, 11–13, 15–19, 26, 28, 29, 31
- MS** multiple sclerosis. 1, 3, 7, 21, 26–31, 185
- MSE** mean squared error. 40
- MU** multiplicative updates. xxi, 104–108
- NMF** nonnegative matrix factorization. xviii, 3, 5, 7, 8, 100–110, 123, 124, 186–189
- OLS** ordinary least squares. 55
- RF** radio frequency. 14
- RMSProp** root mean square propagation. 50, 51
- RSS** residual sum of squares. 55
- SGD** stochastic gradient descent. 48–50
- TE** echo time. 14, 17
- TN** tensor network. 111–113, 115, 116, 119–123
- TR** repetition time. 14, 17
- TT** tensor train. 119, 120, 122, 123
- ViT** vision transformer. xviii, 84, 85, 87–90, 95, 96, 186

List of Notations

Numbers and Arrays

- a A scalar (integer or real)
- \mathbf{a} A vector
- \mathbf{A} A matrix
- \mathcal{A} A tensor

Indexing

- a_i Element i of vector \mathbf{a} , with indexing starting at 1
- a_{-i} All elements of vector \mathbf{a} except for element i
- $A_{i,j}$ Element (i, j) of matrix \mathbf{A}
- $A_{i,:}$ Row i of matrix \mathbf{A}
- $A_{:,i}$ Column i of matrix \mathbf{A}
- $\mathcal{A}_{i,j,k}$ Element (i, j, k) of a 3rd-order tensor \mathcal{A}
- $\mathcal{A}_{::,i}$ A slice of a 3rd-order tensor \mathcal{A}

(Multi)linear Algebra Operations

- \mathbf{A}^\top Transpose of matrix \mathbf{A}
- \mathbf{A}^\dagger Moore-Penrose pseudoinverse of \mathbf{A}
- $\langle \mathbf{A}, \mathbf{B} \rangle$ Inner product of \mathbf{A} and \mathbf{B}

$\mathbf{A} \odot \mathbf{B}$	Element-wise (Hadamard) product of \mathbf{A} and \mathbf{B}
$\mathbf{a} \otimes \mathbf{b}$	Outer (tensor) product of \mathbf{a} and \mathbf{b}
$\mathbf{A} \otimes_{\mathbf{k}} \mathbf{B}$	Kronecker product of \mathbf{A} and \mathbf{B}
$\det(\mathbf{A})$	Determinant of \mathbf{A}

Sets

\mathbb{A}	A set
\mathbb{R}	The set of real numbers
$\mathbb{R}_{\geq 0}$	The set of nonnegative real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, N\}$	The set of all integers between 0 and N
$\{\mathbf{x}_n\}_{n=1}^N$	The set containing $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$ \mathbb{A} $	The cardinality of set \mathbb{A} , i.e., the number of elements

Sequences

(\mathbf{x}, \mathbf{y})	An ordered pair
$(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$	An N -tuple, i.e. a finite sequence
$(\mathbf{x}_n)_{n=1}^N$	An N -tuple, i.e. a finite sequence
$(\mathbf{x}_n)_{n=0}^{\infty}$	An infinite sequence

Functions

$f : \mathbb{X} \rightarrow \mathbb{Y}$	The function f with domain \mathbb{X} and range \mathbb{Y}
$f(\mathbf{x}; \boldsymbol{\theta})$	A function of \mathbf{x} parametrized by $\boldsymbol{\theta}$
$\ln(x)$	Natural logarithm of x
$\sigma(x)$	Sigmoid function, $\frac{1}{1 + \exp(-x)}$

$\ \mathbf{x}\ _p$	L^p norm of vector \mathbf{x}
$\ \mathbf{x}\ $	Euclidean (L^2) norm of vector \mathbf{x}
$\ \mathbf{X}\ _F$	Frobenius norm of matrix \mathbf{X} , i.e., $\sqrt{\sum_{i,j} X_{ij}^2}$
x_+	Positive part of x , i.e., $\max(0, x)$
$\mathbb{1}(\text{condition})$	Indicator function, i.e., 1 if the condition is true, 0 otherwise

Calculus

$\frac{df}{dx}$	Derivative of f with respect to x
$\frac{\partial f}{\partial x}$	Partial derivative of f with respect to x
$\nabla_{\mathbf{x}} f$	Gradient of f with respect to \mathbf{x}
$\int_{\mathbb{X}} f(\mathbf{x}) d\mathbf{x}$	Definite integral over the entire domain \mathbb{X}

Probability

$\mathbf{y} \mathbf{x}$	\mathbf{y} conditional on \mathbf{x}
$P(\mathbf{x})$	A probability distribution function
$p(\mathbf{x})$	A probability density or mass function
$p(\mathbf{y} \mathbf{x})$	Conditional probability distribution of \mathbf{y} given \mathbf{x}
$\mathbf{x} \sim P$	Random vector \mathbf{x} has distribution P
$\mathbf{x}_n \stackrel{\text{iid}}{\sim} P$	Data points $\{\mathbf{x}_n\}_{n=1}^N$ are iid from distribution P
$\mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})]$	Expectation of $f(\mathbf{x})$ with respect to $p(\mathbf{x})$
$D_{\text{KL}}(P, Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\text{Ber}(\pi)$	Bernoulli distribution with parameter $\pi \in [0, 1]$
$\text{Cat}(\boldsymbol{\pi})$	Categorical distribution with parameter $\boldsymbol{\pi}$

Contents

Abstract	i
Beknopte samenvatting	iii
Résumé	v
List of Abbreviations	viii
List of Notations	ix
Contents	xiii
List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 Problem Statement and Scope	1
1.2 Research Objectives	3
1.2.1 General Objectives	3
1.2.2 Specific Objectives	4
1.3 Thesis Structure	5
1.4 Collaboration and Funding	8
2 A Primer on MRI and Its Applications to Brain Lesion Diagnosis	11
2.1 Magnetic Resonance Imaging	12
2.1.1 MRI Principles	12
2.1.2 MRI sequences	17
2.1.3 Multiparametric MRI for More Accurate Diagnosis . . .	19
2.2 Brain Lesions	22
2.2.1 Brain Tumors	22

2.2.2	Multiple Sclerosis Lesions	26
2.2.3	Stroke Lesions	30
2.3	Conclusion	31
3	Machine Learning Essentials	33
3.1	When Models Meet Data	34
3.1.1	Learning Scenarios	34
3.1.2	Data and Models	35
3.1.3	Model Fitting	36
3.1.4	Generalization and Overfitting	39
3.1.5	Model Selection and Validation	42
3.2	Optimization	45
3.2.1	Gradient Descent	45
3.2.2	Stochastic Gradient Descent	48
3.2.3	Block Coordinate Descent	51
3.2.4	Majorization-Minimization Algorithm	53
3.3	Learning Algorithms	54
3.3.1	Linear Regression	54
3.3.2	Logistic Regression	55
3.3.3	Kernel Density Estimation	58
3.3.4	Kernel Regression	61
3.4	Conclusion	63
4	Deep Learning for Medical Image Segmentation	65
4.1	From Handcrafted Features to Hierarchical Representations	66
4.2	Multilayer Perceptron	66
4.3	Convolutional Neural Networks	69
4.3.1	Convolutions for Images	71
4.3.2	Pooling Layers	75
4.3.3	Normalization Layers	76
4.3.4	Transposed Convolutional Layers	79
4.3.5	Modern CNN Architectures	80
4.4	Transformers and Attention Mechanisms	84
4.4.1	Attention Mechanism	85
4.4.2	Multi-Head Attention	86
4.4.3	Vision Transformer	87
4.4.4	Swin Transformer	88
4.4.5	Performer: Linear Complexity Transformer	90
4.5	Medical Image Segmentation Models	92
4.5.1	CNNs	93
4.5.2	Hybrid Architectures	95
4.5.3	Pure Transformers	95
4.6	Conclusion	96

5	Low-Rank Factorization	99
5.1	Nonnegative Matrix Factorization	100
5.1.1	NMF Problem	101
5.1.2	NMF Algorithms	103
5.1.3	Choice of Rank	108
5.2	Tensor Networks	111
5.2.1	Tensors, Tensor Networks, and Tensor Diagrams	111
5.2.2	Basic Tensor Operations	112
5.2.3	Contraction Order	115
5.2.4	Tensor Decomposition Formats	117
5.3	Tensor Networks Meet Machine Learning	120
5.3.1	Network Compression and Acceleration	120
5.3.2	Lightweight Parametrization	121
5.3.3	Quantum-Inspired Learning	122
5.4	Conclusion	123
6	New Multiple Sclerosis Lesion Segmentation	125
6.1	Introduction	126
6.2	Related work	128
6.3	Method	128
6.3.1	Overall Architecture	128
6.3.2	Baseline Models	129
6.4	Experiments	130
6.4.1	Setup.	131
6.4.2	Results and Discussion	134
6.5	Conclusion	137
7	Factorizer: A Scalable Interpretable Approach for Medical Image Segmentation	139
7.1	Introduction	141
7.2	Related Work	143
7.3	Method	145
7.3.1	Matrix Factorization for Context Modeling	145
7.3.2	Overall Architecture	147
7.3.3	Factorizer Block	148
7.3.4	Wrapped NMF Module	148
7.4	Experiments	154
7.4.1	Datasets	154
7.4.2	Setup	155
7.4.3	Results and Discussion	158
7.4.4	Ablation Studies: Training	164
7.4.5	Ablation Studies: Inference	166
7.5	Conclusion and Future Work	168

7.6	Appendix	169
7.6.1	Data Augmentation Pipeline	169
7.6.2	Matricize Implementation Details	170
8	Low-Rank Convolutional Neural Networks for Brain Tumor Segmentation	173
8.1	Introduction	174
8.2	Related work	175
8.3	Method	176
8.3.1	Data Preprocessing and Augmentation	177
8.3.2	Network Architecture	178
8.3.3	Low-rank Convolution	178
8.3.4	Loss Function	180
8.3.5	Optimization	180
8.3.6	Postprocessing	181
8.4	Experiments and Results	181
8.5	Conclusion	183
9	Conclusion	185
9.1	Contributions	185
9.2	Future Perspectives	188
	Bibliography	191
	Curriculum Vitae	207
	List of publications	211

List of Figures

1.1	Overview of the chapters in this PhD manuscript.	6
2.1	When placed in a magnetic field B_0 , protons will fall into one of two energy states.	14
2.2	Net magnetization during excitation and relaxation.	14
2.3	Net magnetization plotted against time during relaxation.	16
2.4	MRI sequences used in this thesis for glioma segmentation.	21
2.5	MRI sequences for MS lesion segmentation.	21
2.6	MRI sequences used in this thesis for ischemic stroke lesion segmentation.	22
2.7	Incidence rate ratios by gender (males:females) for primary brain tumors.	24
2.8	Incidence rates of brain tumors by age [1].	25
2.9	Damaged myelin in multiple sclerosis.	27
2.10	The multiple sclerosis prevalence by region [2].	28
2.11	Different multiple sclerosis types based on disease progression.	29
3.1	Comparison of polynomial regression models of different degrees.	41
3.2	Illustration of underfitting, optimal fitting, and overfitting in model training.	42
3.3	An illustration of the holdout method.	43
3.4	Schematic of 5-fold cross validation.	44
3.5	Comparison of different learning rate schedules over iterations.	47
3.6	Illustration of momentum. Here, \mathbf{g} denotes the gradient ∇J	48
3.7	Illustration of the MM algorithm.	53
3.8	A comparison of some popular kernels.	59
3.9	An example of kernel density estimate from 6 data points.	61
3.10	Nadaraya-Watson kernel regression with a Gaussian kernel.	62
4.1	An MLP with 2 hidden layers, each has 4 units.	68

4.2	Illustration of 2D cross-correlation.	71
4.3	Illustration of padding and strides in 2D convolution.	72
4.4	Illustration of 2D convolution applied to an input with 2 channels.	74
4.5	Illustration of pointwise convolution.	75
4.6	Illustration of max pooling with a 2×2 filter and a stride of $(2, 2)$	76
4.7	Illustration of different normalization methods for neural networks	77
4.8	Transposed convolution with 2×2 kernel.	80
4.9	Modern CNNs for image recognition.	81
4.10	Blocks used in the modern CNNs.	83
4.11	Illustration of the attention mechanism.	86
4.12	Overview of ViT.	88
4.13	Overview of Swin Transformer.	89
4.14	Illustration of the shifted window approach for attention computation in Swin Transformer.	90
4.15	Approximation of the attention \mathbf{SV} via (random) feature maps.	91
4.16	Generic forms of U-shaped segmentation model architectures.	93
4.17	The U-Net architecture [3].	94
5.1	An illustration of NMF.	102
5.2	Illustration of NMF components for the Olivetti faces dataset.	108
5.3	Comparison of NMF algorithms.	108
5.4	Choosing rank of NMF using the Kneedle algorithm for elbow detection.	110
5.5	Basic symbols for tensor diagrams.	112
5.6	Tensor diagram for graphical representation of complex tensor contractions.	113
5.7	Tensor diagram of some basic matrix and tensor operations.	114
5.8	Sequential contraction of a tensor network.	116
5.9	Tensor diagrams of some popular tensor decompositions.	118
5.10	Illustration of regression with tensor networks.	122
6.1	Qualitative results on new MS lesion segmentation.	127
6.2	The proposed encoder-decoder architecture for new MS lesion segmentation.	129
6.3	The proposed blocks.	130
6.4	Comparison of different models on the MSSEG-2 test set.	135
7.1	The overall architecture of Factorizer.	147
7.2	An overview of the Factorizer block and its components.	148
7.3	The Dice score and 95% Hausdorff distance of different models on the BraTS dataset.	160

7.4	Performance of Factorizer models in terms of average Dice on BraTS, versus the number of parameters (left) and versus their speed in terms of the number of FLOPs (right).	161
7.5	Qualitative results on brain tumor segmentation.	161
7.6	NMF Components of Factorizer models on BraTS.	162
7.7	Qualitative results on stroke lesion segmentation.	165
7.8	Inference-phase ablation experiments with Swin Factorizer pre-trained on BraTS.	167
8.1	The multimodal MRI images along with the corresponding ground truth and prediction for a HGG (top row) and a LGG (bottom row) representative cases.	175
8.2	The U-Net architecture (a) and the ResNet block (b) used for brain tumor segmentation.	177
8.3	Canonical Polyadic tensor network.	180
8.4	Validation metric plotted against epochs for two different values of F (the initial number of feature maps). The low-rank convolutions with $\alpha = 0.5$ are used. For visualization purposes, the curves are smooth via LOESS [166].	182

List of Tables

2.1	General intensities of different tissues in various MRI sequences.	18
3.1	Various loss functions used in supervised learning.	37
3.2	List of some popular normalized kernels in 1D.	59
4.1	Some popular activation functions and their derivatives. Here, the smoothness indicates the order of continuity measured by the number of continuous derivatives, e.g., C^1 is the class of functions with zeroth and first continuous derivatives. $\text{erf}(\cdot)$ denotes the error function, and $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution, respectively.	98
5.1	MU computational complexity for the update of \mathbf{F} .	107
5.2	HALS computational complexity for the update of \mathbf{F} .	107
6.1	An overview of the MSSEG-2 dataset.	131
6.2	Results obtained by 5-fold cross-validation on the MSSEG-2 training set.	135
6.3	Results on the MSSEG-2 test set.	136
7.1	Comparison of different models on the BraTS dataset.	159
7.2	Comparison of different models on the ISLES22 dataset.	164
7.3	The impact of each subblock on the performance of Factorizers on BraTS.	165
7.4	The impact of positional embedding on the performance of Factorizers on BraTS.	166
8.1	The average scores on the BraTS 2020 validation set.	183
8.2	Summary statistics of the scores on the BraTS 2020 test set (166 cases). They are computed by the online evaluation platform (WT: whole tumor, TC: tumor core, and ET: enhancing tumor).	183

Chapter 1

Introduction

1.1 Problem Statement and Scope

Brain lesions refer to abnormal changes or damages in brain tissue, which can arise from various causes such as injuries, infections, tumors, strokes, or neurological disorders like multiple sclerosis (MS). The prevalence of brain lesions is alarming. According to the National Brain Tumor Society, nearly 95,000 new cases of brain tumors are expected to be diagnosed in the United States in 2023, with over 20% deaths resulting from malignant brain tumors in the same year. MS is estimated to affect nearly 2.8 million people worldwide. MS is the most common disabling neurological disorder among young adults in Europe and North America, with the highest incidence rates in northern Europe [2]. Stroke remains a major cause of death and a the first cause of disability, with an estimated 12.2 million new cases emerging each year [4].

Magnetic resonance imaging (MRI) is widely used to diagnose and assess brain lesions in clinical practice. A combination of various types of MRI sequences is often used for a more comprehensive and accurate assessment of the lesion tissue under study. This technique, known as *multiparametric MRI (mpMRI)*, leverages the unique tissue characteristics highlighted by each sequence, offering a more detailed picture of the lesion than a single sequence alone could provide. mpMRI proves especially valuable when examining intricate structures like the brain. It provides essential information about the presence, location, size, and characteristics of various brain lesions. For instance, in the context of brain tumors, T2-weighted images can highlight areas of *edema*, crucial for tumor grading, while post-gadolinium T1-weighted images can expose enhancing

tumors, often indicating higher grades.

The process of identifying and delineating areas of brain lesions is called *brain lesion segmentation*, which is a pivotal task in clinical neuroimaging, with applications spanning diagnosis, treatment planning, disease monitoring, and clinical research. **By accurately identifying the location and extent of the lesions, brain lesion segmentation particularly allows clinicians to make informed decisions about patient care and helps researchers better understand the underlying mechanisms of these conditions.** However, manual segmentation of brain lesions is tedious, time-consuming, and costly, especially because experts must deal with 3D images across possibly multiple modalities. Even with sufficient resources for manual segmentation, certain applications, such as image-guided surgery and radiotherapy, demand near real-time segmentation, making manual methods impractical. As a result, there is a need for accurate computer-assisted techniques that can automatically execute such tasks in a timely and cost-effective manner. Moreover, manual segmentation is inherently subjective and may vary between observers. **Automated methods, on the other hand, ensure reproducibility by providing consistent outcomes and eliminating both inter- and intra-observer variability, essential for longitudinal studies and multi-center trials.**

Despite significant advances in medical imaging and computer-aided diagnosis, the automated segmentation of brain lesions remains a formidable challenge. Deep learning models, especially convolutional neural networks (CNNs) and transformers, have shown remarkable performance in image segmentation and have become the cornerstone in this field. However, several challenges persist when applying deep learning models to brain lesion segmentation:

- **Data Scarcity:** Training deep learning models effectively requires large amounts of data to prevent overfitting. Unlike natural images, acquiring labeled brain lesion images is difficult due to the need for expert annotation, the rarity of certain lesions, the high costs of data acquisition, and patient privacy concerns.
- **Variability:** Medical images can vary significantly due to differences in acquisition protocols, scanners, inherent patient-to-patient variations, annotation inconsistencies, and the presence of artifacts. In particular, brain lesions can vary dramatically in shape, structure, and location across patients and even within the same patient over time. This makes it challenging for a model to generalize across different scenarios.
- **Class Imbalance:** In many brain lesion segmentation tasks, the region of interest (i.e., lesion) may only occupy a minor portion of the

image, leading to class imbalance. Standard training techniques may undesirably prioritize the majority class (i.e., background), leading to poor segmentation of the region of interest.

- **Computational Requirements:** Deep learning models, especially those processing 3D volumetric data, demand substantial computational resources, which may not be readily available in all clinical setting.
- **Interpretability:** Deep learning models are often criticized for being “black boxes.” In medical contexts, understanding why a model made a particular decision can be important for clinician trust and safety.

In the context of machine learning, low-rank models, including nonnegative matrix factorization (NMF) and tensor decomposition, initially proved highly beneficial for *data compression and mining*. More recently, these low-rank techniques have been adeptly utilized to compress and accelerate deep neural networks for both vision [5], [6] and language [7], [8] tasks. In this thesis, beyond the development of CNN- and transformer-based models, we introduce innovative, seamless integrations of low-rank factorization techniques into deep learning. This integration aims to address the aforementioned challenges, thereby enhancing generalization, efficiency, and interpretability in the segmentation of brain lesions. **To the best of our knowledge, the research presented in this thesis is among the pioneering efforts to combine low-rank techniques with deep learning in the domain of medical imaging.**

1.2 Research Objectives

1.2.1 General Objectives

The general goal of this thesis is to design computer-aided diagnostic tools for the automated segmentation and detection of various brain lesions using mpMRI data. More specifically, **this work aims to develop deep learning models capable of accurately, efficiently, and interpretably segmenting and detecting brain tumors, MS lesions, and stroke lesions in mpMRI images.** To address the limitations of existing models, discussed in the previous section, we integrate advanced deep learning with low-rank factorization techniques. The core contributions of this thesis include the development of novel deep learning methods for medical image segmentation. The objectives of this thesis bifurcate into methodological objectives, which make a substantial contribution to the field of deep learning, and medical application objectives,

which potentially lead to improved diagnostic procedures for patients with brain lesions.

1.2.2 Specific Objectives

Our general objective is translated into specific objectives as follows:

1. **Objective 1: Improve the generalization capability of deep learning models for brain lesion segmentation**

In the pursuit of enhancing the efficacy of deep learning models for brain lesion segmentation, our first objective is to improve their generalization capability. Several challenges contribute to the difficulty in achieving this goal. Primarily, data scarcity often limits the robustness of the model, making it susceptible to overfitting. Additionally, class imbalance can skew the model's predictions, favoring the majority class and neglecting the minority. Furthermore, the variability observed both between different patients (inter-patient) and within the same patient over time or under different conditions (intra-patient) complicates the training process. The inconsistency in expert opinions, both among experts (inter-expert) and from the same expert under varying circumstances (intra-expert), adds another layer of complexity. To mitigate the effects of overfitting caused by these challenges, this research proposes the use of data augmentation techniques to artificially increase the dataset's size and diversity. Concurrently, the implementation of low-rank regularization using tensor decomposition formats serves to constrain the model space, ensuring it captures only the most salient features, thereby enhancing its generalization capabilities.

2. **Objective 2: Enhance scalability and efficiency of deep learning models in context modeling for brain lesion segmentation**

Existing deep neural networks for 3D medical images are characterized by their vast size, often having millions of learnable parameters. As a result, they can be too complex, requiring a significant amount of floating-point operations (FLOPs) and computational resources. Specifically, CNNs often exhibit slow inference times, while transformers do not scale linearly, both demanding substantial memory. To mitigate these challenges, two primary approaches are proposed in this thesis. The first involves the development of lightweight or accelerated networks by reparametrizing their weights using low-rank tensor networks, which effectively reduces the

number of parameters. The second solution seeks to integrate low-rank matrix factorization techniques into end-to-end deep learning models, specifically tailored for context modeling in brain lesion segmentation.

3. **Objective 3: Enhance interpretability of deep learning models for brain lesion segmentation**

The primary motivation for this objective stems from the inherent “black box” nature of deep neural networks. Often, it remains ambiguous as to why these models arrive at a particular decision, which can be a significant concern, especially in critical applications like medical imaging. To address this, we propose a solution that formulates NMF as a differentiable layer that can be seamlessly integrated into the deep learning framework. This approach results in highly interpretable NMF components, with each component discriminating a distinct region of interest, thereby providing clearer insights into the model’s decision-making process.

4. **Objective 4: Achieve accurate segmentation of new MS lesions in longitudinal images**

New MS lesions present significant variability in terms of shape, size, and location. Additionally, there is a notable class imbalance due to the rarity of these new MS lesions, compounded by the limited availability of extensive longitudinal datasets. These factors collectively contribute to issues such as low segmentation accuracy, undersegmentation, and the potential for model overfitting. To address these challenges, this research proposes a multifaceted solution. Firstly, effective data augmentation techniques are employed to enhance the robustness of the model. Secondly, deep supervision is incorporated to guide the learning process more effectively. Lastly, rather than training a cross-sectional segmentation model and then subtracting the predictions at two time points, the model is directly trained on longitudinal segmentation, ensuring a more tailored and accurate representation of new lesions.

1.3 Thesis Structure

Figure 1.1 provides a schematic overview of the structural organization of the chapters in this thesis. Chapters 2–5 provide an extensive review of the concepts and methods essential for understanding our methodological contributions. Chapters 6–8 present our scientific contributions to the development and implementation of brain lesion segmentation models.

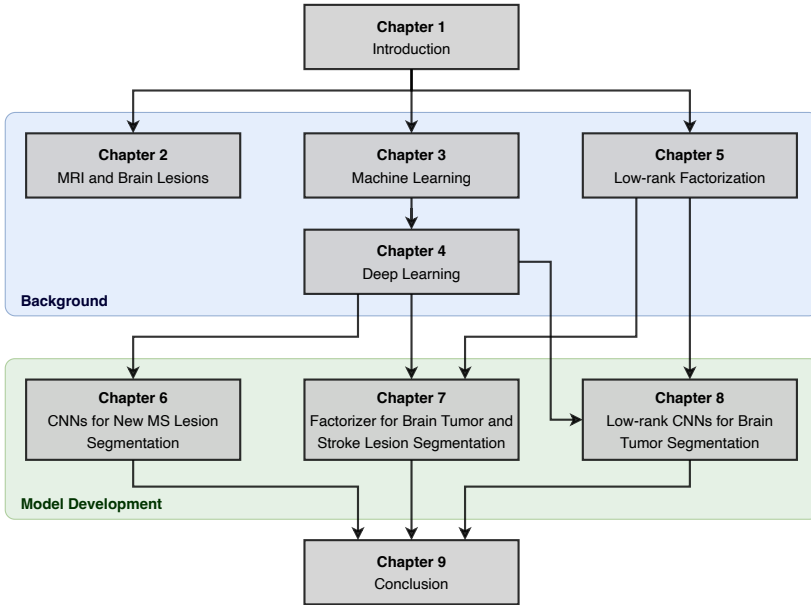


Figure 1.1 Overview of the chapters in this PhD manuscript (MS: multiple sclerosis, CNN: convolutional neural network)

Chapter 2 provides an overview of magnetic resonance imaging (MRI) fundamentals, accompanied by an introductory explanation of MRI sequences relevant to for the diagnosis of brain lesions, specifically: T1, T2, fluid-attenuated inversion recovery (FLAIR), and diffusion-weighted imaging (DWI). This chapter also touches upon foundational concepts related to the biology of brain lesions, with an emphasis on glioma, multiple sclerosis (MS), and stroke. We further discuss the imaging techniques most relevant for monitoring and diagnosing each of these brain diseases.

Chapter 3 delves into the foundational concepts of machine learning, with a primary emphasis on supervised learning and model fitting methods. These form the basis of our approach to brain lesion segmentation using deep learning. We also explore first-order optimization techniques, notably stochastic gradient descent and Adam, which are essential for training deep models on a large scale. The chapter concludes by examining linear models and kernel regression, shedding light on their evolution into more complex, non-linear models, specifically deep neural networks.

Chapter 4 expands on the previous chapter, presenting the foundations of deep learning vital for medical image segmentation. We first explore convolutional

neural networks (CNNs) and their key components. We then delve into models that incorporate transformers and attention mechanisms, which have recently emerged as effective methods in addressing computer vision challenges. The chapter concludes with an overview of convolutional, attention-based, and hybrid models with U-shaped architectures tailored for medical image segmentation.

Chapter 5 provides an overview of low-rank factorization techniques, which are pivotal in the brain lesion segmentation models presented in Chapters 7 and 8. Initially, we delve into non-negative matrix factorization (NMF), emphasizing the popular multiplicative update and hierarchical alternating least squares (HALS) algorithms. This factorization can act as an efficient method for global context modeling, presenting a viable alternative to the attention mechanism. This foundation supports the methodology introduced in Chapter 7. Furthermore, we touch upon tensor networks, highlighting their importance both as computational tools and in forming low-rank representations. Our discussion is augmented by an examination of common tensor operators and tensor decomposition formats. The chapter concludes with an exploration of the different applications of tensor networks in machine learning, such as model compression and acceleration, which form the basis of the approach presented in Chapter 8.

Chapter 6 presents the application CNNs in automated segmentation of new MS lesions in 3D FLAIR images. The main objective is to identify new lesions between two consecutive MRI scans of an MS patient, serving as a crucial marker monitoring and quantifying disease progression. In this chapter, we propose Pre-U-Net, which is a 3D encoder-decoder architecture enhanced with pre-activation residual blocks, for the segmentation and detection of new MS lesions. Given the constraints of a limited training set and the challenge of class imbalance, we employ extensive data augmentation and deep supervision to optimize our model training. We will see that while retaining the same U-shaped architecture but incorporating distinct blocks, Pre-U-Net outperforms both U-Net and Res-U-Net on the MSSEG-2 dataset.

Chapter 7 introduces the first method that integrates low-rank factorization with deep learning for medical image segmentation. The inherent locality of convolution causes CNNs to inadequately exploit the global context, which is crucial for accurately recognizing structures like brain lesions. Recently, transformers have demonstrated impressive performance in vision tasks, including semantic segmentation, primarily due to their ability to model long-range dependencies. However, the quadratic complexity of attention in transformers necessitates the use of self-attention layers only after reducing the image resolution. This compromises their capacity to capture global contexts present at higher resolutions. To address this, this chapter presents a family of models, termed *Factorizer*, that harness the potential of low-

rank matrix factorization to construct an end-to-end segmentation model. Specifically, we introduce a linearly scalable approach to context modeling by formulating nonnegative matrix factorization (NMF) as a differentiable layer within a U-shaped architecture. We also employ the shifted window technique in tandem with NMF to aggregate local information effectively. We will see that Factorizers outperform CNNs and transformers in terms of accuracy, scalability, and interpretability, achieving state-of-the-art results on the BraTS dataset for brain tumor segmentation and the ISLES'22 dataset for stroke lesion segmentation. Highly meaningful NMF components give an additional interpretability advantage to Factorizers over CNNs and transformers. Furthermore, our ablation studies highlight a unique feature of Factorizers that allows for a substantial acceleration in inference once trained, without requiring additional steps or significant accuracy compromises.

Chapter 8 presents another method that leverages low-rank tensor networks to improve CNNs for brain tumor segmentation. Most effective CNNs for 3D image segmentation have millions of learnable parameters. Such complex models are susceptible to overfitting, particularly when there is a limited amount of training data—a common scenario in medical imaging. In this work, we introduce a 3D U-Net-inspired architecture integrated with residual blocks. By imposing low-rank constraints on the weights of the convolutional layers, we aim to mitigate overfitting. Within the same architecture, this approach allows for the design of networks with significantly fewer parameters. We evaluate the performance of our proposed method on the BraTS 2020 challenge data.

Chapter 9 summarizes the contributions and findings of this thesis and offers insights into potential future developments.

1.4 Collaboration and Funding

The research for my PhD thesis was conducted collaboratively at KU Leuven, Department of Electrical Engineering (ESAT), STADIUS Center, and Université Claude Bernard Lyon 1, CREATIS (CNRS UMR 5220 & INSERM U1294). I was privileged to be supervised by Prof. Sabine Van Huffel and Prof. Dominique Sappey-Marinier. Additionally, I benefited from the co-supervision of Prof. Frederik Maes and the invaluable insights and collaboration of Dr. Diana M. Sima and Prof. Lieven De Lathauwer. I also received informal advice, guidance, and support from many colleagues and friends within the BIOMED research group and the INSPiRE-MED network. This research was generously supported by the European Union's Horizon 2020 MSCA-ITN program, funded by the European Commission under grant agreement No. 813120 (INSPiRE-

MED). Further funding was provided by the Flemish Government's AI Research Program. Pooya Ashtari, Sabine Van Huffel, and Frederik Maes are affiliated with Leuven.AI—KU Leuven Institute for AI.

Chapter 2

A Primer on MRI and Its Applications to Brain Lesion Diagnosis

This chapter first provides an overview of the basics of magnetic resonance imaging (MRI), along with a preliminary explanation of MRI sequences that are relevant for diagnosing brain lesions explored in this thesis: T1, T2, fluid-attenuated inversion recovery (FLAIR), and diffusion-weighted imaging (DWI). We then proceed with some elementary concepts related to the biology of brain lesions, such as the origins and symptoms. We primarily focus on glioma, multiple sclerosis (MS), and stroke, discussing which imaging techniques can play a significant role in monitoring and diagnosing each of these brain diseases.

2.1 Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) is a powerful diagnostic tool that utilizes the principles of nuclear magnetic resonance, which involves the interaction of atomic nuclei with an external magnetic field, to produce detailed images of the internal structure of the body. The underlying principles of MRI were first proposed by Bloch [9] and Purcell *et al.* [10] in the 1940s. Since then, MRI has become an integral part of modern medical diagnostics, providing valuable information about various diseases and conditions. MRI has a number of advantages over other imaging modalities, including the ability to produce images in any plane, excellent soft tissue contrast, and the lack of ionizing radiation. MRI can be used to image a wide variety of tissues and organs, including the brain, spine, liver, and heart. In this section, we briefly discuss the basic principles and mechanism of MRI, along with common sequences and their diagnostic applications.

2.1.1 MRI Principles

Nuclear spin. The concept of nuclear spin, which is intrinsic to all particles, is essential to understanding MRI. Atomic nuclei with an odd number of protons and/or neutrons, such as hydrogen (^1H) and carbon (^{13}C), have a quantum mechanical property known as spin, which can be visualized as if the nucleus is spinning about an axis. The overall spin of the nucleus is determined by the spin quantum number, often denoted as I , which may take positive half-integer values (e.g., $I = 0, 1/2, 1, 3/2, 2, \dots$) [11]. In MRI, we are typically imaging hydrogen nuclei ^1H (also referred to as protons) with a spin of $\frac{1}{2}$, which are abundant in water and lipids in the human body [12].

Magnetic moment. A spinning nucleus creates a magnetic field, and therefore, acts as a tiny magnet, results in a magnetic moment ($\boldsymbol{\mu}$), which is a vector quantity that describes the magnetic properties of the nucleus and is proportional to its nuclear spin [12]

$$\boldsymbol{\mu} = \gamma \mathbf{I}, \quad (2.1)$$

where γ is the *gyromagnetic ratio*, a constant specific to the type of nucleus. For hydrogen, $\gamma \approx 2.675 \times 10^8$ ($\text{Rads}^{-1}\text{T}^{-1}$). MRI involves in four main steps briefly explained in the following.

1. Polarization. In the absence of an external magnetic field, the nuclear spins are randomly oriented. However, when placed in an external magnetic field (B_0), at equilibrium the magnetic moments of the nuclei align either parallel (low energy state) or anti-parallel (high energy state) to the field (see Figure 2.1).

The energy of a magnetic moment in a state is given by [12]

$$E = -\boldsymbol{\mu}B_0 = -\gamma\mathbf{I}B_0, \quad (2.2)$$

and hence, the energy difference between the two states is $\Delta E = 2\boldsymbol{\mu}B_0$. By the Planck's relation, $\Delta E = \hbar\omega$, where \hbar is the reduced Planck constant, and ω is the *Larmor frequency* (or *precessional frequency*), we can deduce the *Larmor equation* [12]

$$\omega = \gamma B_0, \quad (2.3)$$

which means the spins precess about the axis of B_0 (commonly taken as the z -axis) at a frequency proportional to the magnitude of the external magnetic field. The ratio between the number of nuclei in the two states (N_+ and N_-) is determined by the Boltzmann distribution [13]

$$\frac{N_+}{N_-} = \exp\left(\frac{\Delta E}{kT}\right) = \exp\left(\frac{\gamma\hbar B_0}{kT}\right), \quad (2.4)$$

where $k \approx 1.38 \times 10^{-23}$ is the Boltzmann constant and T is the temperature in Kelvin. For an external magnetic field of 3 Tesla, at a body temperature of 37 °C (or 310.15 Kelvin), the ratio will be

$$\frac{N_+}{N_-} = 1.0000098826155204. \quad (2.5)$$

This means spins in the low-energy state, N_+ , only slightly outnumbers spins in the high-energy state. If we increase the magnetic field strength, we increase the energy difference and hence also the population difference between the states. Note that at equilibrium after the nuclei are placed in the magnetic field, the number of nuclei in the low-energy state is greater than the number in the high-energy state, resulting in the longitudinal (z -axis) magnetization M_0 .

An MR signal results from the difference between the energy absorbed by the spins which make a transition from the lower-energy state to the higher-energy state, and the energy emitted by the spins which simultaneously make a transition from the higher energy state to the lower energy state. Therefore, the signal is proportional to the population difference between the states. MRI is rather sensitive since it is capable of detecting these very small population differences.

2. Excitation. Once protons are polarized by an external magnetic field B_0 , during excitation an RF pulse is applied perpendicular to the B_0 , which boosts some of the protons from the lower-energy state to the higher-energy state (see Figure 2.1). This causes the net magnetization vector to tip away from the longitudinal position (along B_0 , i.e. z -axis) into the transverse (perpendicular

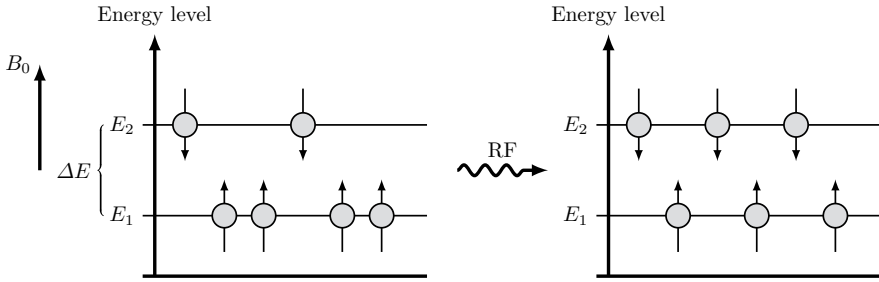


Figure 2.1 When placed in a magnetic field B_0 , protons will fall into one of two energy states: in the lower energy state, protons are lined parallel to B_0 , whereas in the higher energy state they are anti-parallel to it. A radio frequency (RF) pulse can boost some of the protons, flipping the spins from the lower energy state to the higher energy state.

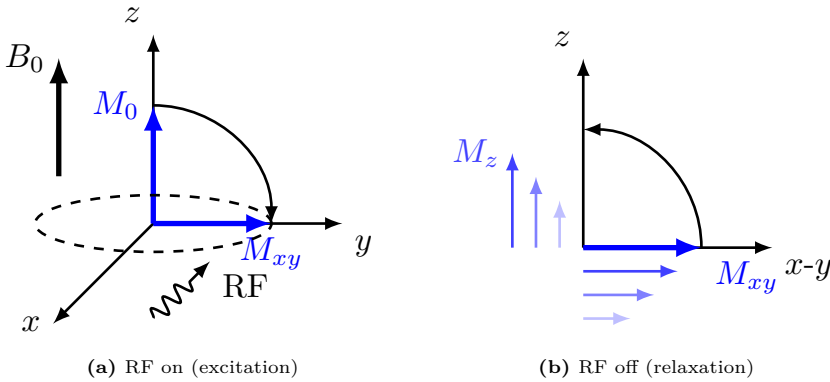


Figure 2.2 Net magnetization during excitation and relaxation. (a) After the RF pulse, the longitudinal magnetization vector is flipped into the x - y plane. (b) Once the RF is turned off, the transverse magnetization vector begins to decay while the longitudinal component begins to recover.

to B_0 , i.e. x - y) plane [12], as illustrated in Figure 2.2a. This only occurs when the frequency of the RF pulse matches the Larmor frequency, a condition known as resonance.

Two key parameters define an RF pulse: the repetition time (TR), which is the time between two consecutive RF pulses, and the echo time (TE), which is the time from the RF pulse to the peak of the echo signal induced in the receiver coil.

3. Relaxation. After excitation, the RF pulse is turned off, making the system return to equilibrium, a process known as relaxation. Precisely, the net magnetization vector returns from the transverse plane to the longitudinal axis [12] (See Figure 2.2b). This behavior is described by the Bloch equations [9], [14]:

$$\begin{aligned}\frac{dM_z(t)}{dt} &= \frac{1}{T_1}(M_0 - M_z(t)), \\ \frac{dM_{xy}(t)}{dt} &= -\frac{1}{T_2}M_{xy}(t),\end{aligned}\tag{2.6}$$

where M_0 is the equilibrium magnetization, M_z is the longitudinal magnetization, and M_{xy} is the transverse magnetization. T_1 and T_2 are the longitudinal and transverse relaxation times, respectively. The relaxation times are properties of the tissue being imaged and influence the contrast in MRI images. Assuming $M_z(0) = 0$ and $M_{xy}(0) = M_0$ at the start of relaxation, the solution to (2.6) is:

$$\begin{aligned}M_z(t) &= M_0\left(1 - \exp\left(-\frac{t}{T_1}\right)\right), \\ M_{xy}(t) &= M_0 \exp\left(-\frac{t}{T_2}\right).\end{aligned}\tag{2.7}$$

Therefore, T_1 is the time it takes for the longitudinal magnetization to recover to about 63% of its initial value, and T_2 is the time it takes for the transverse magnetization to decay to about 37% of its initial value. In fact, both longitudinal and transverse relaxation processes occur simultaneously with the only restriction being that T_2 is less than or equal to T_1 . In Figure 2.3, magnetization components over the relaxation phase are plotted against time.

In an ideal situation, if we had a perfectly uniform magnetic field, the received transverse signal would be $M_{xy}(t) = M_0 \exp\left(-\frac{t}{T_2}\right)$. However, What really happens in practice is that because of spin *dephasing* (namely, spin-spin interactions and external magnetic field inhomogeneities), an RF receiver coil receives a complex-valued function of time, called *free induction decay* (FID) signal [12]:

$$s(t) = M_0 \exp\left(-\frac{t}{T_2^*}\right) \exp(-j\omega t)\tag{2.8}$$

where T_2^* is the apparent transverse relaxation time that characterizes the decay of the FID signal, and j is the imaginary unit. The T_2^* relaxation time is shorter than the pure T_2 relaxation time, as it takes into account additional decay caused by inhomogeneities in the magnetic field [12]; that is

$$\frac{1}{T_2^*} = \frac{1}{T_2} + \gamma\Delta B,\tag{2.9}$$

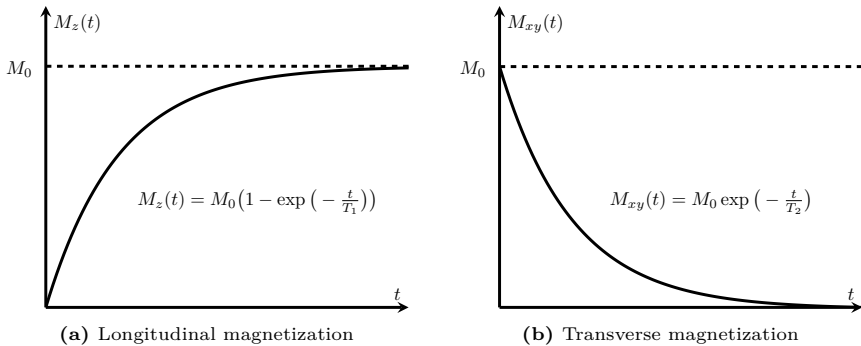


Figure 2.3 Net magnetization plotted against time during relaxation. (a) The plot of recovery of longitudinal magnetization with a time constant of T_1 . (b) The plot of transverse magnetization with a time constant of T_2 .

where $\Delta\mathbf{B}$ represents the inhomogeneity in the magnetic field. These inhomogeneities can arise from factors such as susceptibility variations, field gradients, or local magnetic field distortions. If we have a perfect magnet that does not introduce any inhomogeneity, then $\Delta\mathbf{B} = 0$ and hence $T_2^* = T_2$. The recent systems have less magnetic field inhomogeneity, thus making T_2^* effects less strong; however, complete homogeneity is not possible.

4. Detection and reconstruction. The detection and reconstruction of the MRI signal is a complex process beyond the scope of this thesis. Here, we briefly sketch the big picture. As mentioned, the MRI signal is detected by a receiver coil. This coil detects the changing magnetic field caused by the precessing magnetization, which induces an electric current in the coil according to Faraday's law of electromagnetic induction

$$\mathcal{E} = -\frac{d\Phi}{dt} \quad (2.10)$$

where \mathcal{E} is the induced voltage, Φ is the magnetic flux through the coil. This induced voltage is commonly referred to as the MRI signal. In general, the MRI signal from an individual RF coil can be modeled as [15]

$$s(t) = \int \rho(\mathbf{r}) e^{-j2\pi\mathbf{k}(t)\cdot\mathbf{r}} d\mathbf{r}, \quad (2.11)$$

where $\mathbf{r} = (x, y, z)$ is the position in space, $\rho(\mathbf{r})$ is the spin density (a measure of the concentration of spins in a volume element), $\mathbf{k}(t) = (k_x(t), k_y(t), k_z(t))$ is the k-space trajectory representing spatial frequencies, and \cdot denotes the dot product of two vectors. To reconstruct the MRI image, the MRI signal is

collected in k -space, a space defined by spatial frequencies rather than normal spatial coordinates. Each signal sample fills a specific location in the spatial frequency domain, known as k -space, to form the detected $S(k_x, k_y, k_z)$ in the frequency domain. Therefore, the raw data obtained in MRI is in k -space and must be transformed into the spatial domain to yield an image $I(x, y, z)$. This is accomplished through the inverse Fourier transform of the raw signal [15]

$$I(x, y, z) = \iiint S(k_x, k_y, k_z) e^{j2\pi(k_x x + k_y y + k_z z)} dk_x dk_y dk_z. \quad (2.12)$$

In simple terms, each point in k -space contributes to all points in the image, and the Fourier transform makes it possible to combine these contributions into a coherent image. In practice, the MRI signal is typically sampled on a grid or spiral in k -space, and 2D inverse discrete Fourier transform (DFT) over the slices is used to reconstruct the image [15].

2.1.2 MRI sequences

T1-weighted. T1-weighted images are obtained by adjusting the repetition time (TR) and echo time (TE) parameters of the MRI sequence to emphasize the differences in longitudinal relaxation times (T_1) between tissues. Specifically, a short TR and TE are used.

This means that the RF pulse is applied before the longitudinal magnetization has fully recovered, creating a situation where tissues with shorter T_1 times recover more quickly and produce a stronger signal. Consequently, tissues with shorter T_1 times (e.g., fat) appear brighter (higher signal intensity) on T1-weighted images, while those with longer T_1 times (e.g., water) appear darker (lower signal intensity). Notably, white matter in a T1 image appears slightly brighter than gray matter. Table 2.1 provides a general guideline for the intensities of different tissues in various MRI sequences.

T2-weighted. T2-weighted images are obtained by using a long TR and TE to emphasize the differences in transverse relaxation times (T_2) between tissues. A long TR allows for full recovery of longitudinal magnetization, while a long TE allows for more decay of the transverse magnetization. Therefore, tissues with longer T_2 times (e.g., water) appear brighter on T2-weighted images, while those with shorter T_2 times (e.g., fat) appear darker. Notably, gray matter in a T2 image appears slightly brighter than white matter.

Fluid-attenuated inversion recovery. Fluid-attenuated inversion recovery (FLAIR) [16] is a specific type of T2-weighted imaging that suppresses the signal from fluids, such as cerebrospinal fluid (CSF). This is achieved by applying an inversion recovery pulse to nullify the signal from fluids, followed by a delay and then the usual T2 acquisition.

Table 2.1 General intensities of different tissues in various MRI sequences. Note that the exact signal intensities can vary depending on the MRI scanner specifics, the imaging protocol used, individual patient differences, and other factors ((Very) High: The tissue appears (very) bright in the image. Medium: The tissue appears gray in the image. (Very) Low: The tissue appears very dark in the image. Variable: The signal can vary significantly depending on various factors).

Tissue	T1	T2	FLAIR	DWI
Water	Very Low	Very High	Very Low	Very Low
Fat	High	High	Low	Low
Cerebrospinal Fluid (CSF)	Very Low	Very High	Very Low	Very Low
White Matter	Medium	Low	Medium	Medium
Gray Matter	Low	Medium	High	Medium
Inflammation	Variable	High	Very High	High
Edema	Low	Very High	Very High	High
Necrosis	Variable	High	Variable	High
Gadolinium (contrast agent)	Very High	Low	Low	Low

FLAIR is particularly useful in brain imaging, as it allows the detection of white matter lesions adjacent to the CSF that would otherwise be obscured due to the high signal of the CSF on T2-weighted images [17]. In fact, fluid-filled tissues (e.g., CSF) appear dark in a FLAIR image, and therefore, lesions that appear bright on T2 images and are near fluid-filled areas are more easily visualized.

Diffusion-weighted. Diffusion-weighted imaging (DWI) provides an image contrast that is sensitive to the random (Brownian) motion of water molecules and measures it within a voxel of tissue. The diffusion of water molecules is influenced by factors such as temperature, viscosity, and structures that obstruct the motion of water molecules, like cell membranes and macromolecules. This sensitivity makes DWI particularly useful for detecting acute ischemic stroke [18], abscesses, and other conditions where diffusion is restricted.

In DWI, a pair of strong magnetic field gradients are applied before and after the 180° refocusing pulse in a spin-echo sequence. These gradients are characterized by their strength (G), duration (δ), and the time between them (Δ). If no diffusion is occurring, the gradients are perfectly balanced, and the signal is refocused at the echo time. However, if diffusion is occurring, the movement of water molecules during the time between the gradients causes a loss of signal because the spins are no longer in phase. This signal loss is what gives DWI its contrast and is described by the Stejskal-Tanner equation [19]

$$S = S_0 \exp(-b \cdot \text{ADC}), \quad (2.13)$$

where S is the measured signal intensity, S_0 is the signal intensity without the diffusion gradient, ADC is the apparent diffusion coefficient, and b is the diffusion weighting factor, which depends on the gradient strength, the duration of the gradient pulse (δ), and the time between the beginning of the two gradient pulses (Δ)

$$b = \gamma^2 G^2 \delta^2 \left(\Delta - \frac{\delta}{3} \right). \quad (2.14)$$

Here, γ is the gyromagnetic ratio and G is the gradient strength. By adjusting the b value, the sensitivity of the DWI sequence to diffusion can be modulated. In practice, at least two images with different b values are acquired, often one with $b = 0$ (no diffusion weighting) and another with a higher b value (diffusion-weighted image). The ADC can then be calculated on a pixel-by-pixel basis by rearranging the Stejskal-Tanner equation (2.13)

$$\text{ADC} = -\frac{1}{b} \ln \left(\frac{S}{S_0} \right) \quad (2.15)$$

The resulting ADC map provides quantitative information about the diffusion of water molecules in the tissue. Areas of restricted diffusion (e.g., ischemic stroke and tumors) have lower ADC values, while areas of increased diffusion (e.g., CSF and edema) have higher ADC values.

2.1.3 Multiparametric MRI for More Accurate Diagnosis

Multiparametric Magnetic Resonance Imaging (mpMRI) refers to the technique that combines various types of MRI sequences, providing a more comprehensive and accurate assessment of the specific area or tissue under study. This approach exploits the different tissue characteristics highlighted by each sequence, providing a more detailed picture of the lesion than a single sequence alone would allow.

It is particularly useful for the evaluation of complex structures, such as the brain, where it can provide critical information about the presence, location, size, and characteristics of various lesions, particularly brain tumors, multiple sclerosis, and ischemic stroke. In the following, we discuss which MRI sequences are more relevant for diagnosing each of these lesions.

Brain tumors and gliomas. The diagnosis and characterization of brain tumors and gliomas often involve several MRI sequences, among, each of which highlight different aspects of the tumor. The most common sequences for brain tumors are:

1. **T1-weighted** provides information about the anatomy of the brain and also reveal the general size and location of the tumor, as illustrated in Figure 2.4a.
2. **Gadolinium-enhanced T1-weighted (T1Gd)** has been widely used and become a standard technique in clinical practice for identifying enhancing tumors as it is able to distinguish enhancing tumors from necrotic and non-enhancing tumors more clearly than other imaging techniques [20]. In fact, after the administration of gadolinium contrast, enhancing tumors (which often indicates a high-grade glioma) often appear brighter on T1 images [21], while necrotic and non-enhancing parts of the tumor appear darker on T1Gd, as exemplified in Figure 2.4a and 2.4b.
3. **T2-weighted/FLAIR** are beneficial for identifying areas of edema (i.e., swelling surrounding the tumor and infiltrative tumor growth), which is important for tumor grading. These regions typically appear hyper (bright) on T2/FLAIR images [22], as exemplified in Figure 2.4c and 2.4d. FLAIR is particularly useful when it comes to lesions near the ventricles, making it easier to see them due to suppression of the signal from CSF.
4. **DWI/ADC** can help identify areas of high cellularity within tumors, which often exhibit restricted diffusion, appearing hyperintense (bright) in DWI and hypointense (dark) in ADC images. This particularly aids in differentiating between high- and low-grade gliomas [23]. DWI and ADC are not used in this thesis for brain tumor segmentation.
5. **Perfusion-weighted imaging (PWI)** is a technique that measures the blood flow within the tumor, which can be helpful for grading tumor aggressiveness and evaluating response to treatment [24]. PWI is not used in this thesis.

Multiple Sclerosis. In the case of multiple sclerosis (MS), the most relevant sequences include:

1. **T2-weighted/FLAIR** are commonly used to visualize and identify most types of MS lesions, including white matter, gray matter, and juxtacortical lesions, all of which appearing hyperintense, as exemplified in Figure 2.5c and 2.5d. FLAIR is particularly sensitive to both inflammation and demyelination, which allows it to effectively detect the overall lesion load. However, they are unable to differentiate between active (recently formed or actively inflamed) and inactive (older or chronic) lesions. In this thesis, we use only FLAIR to segment and detect new white matter lesions.

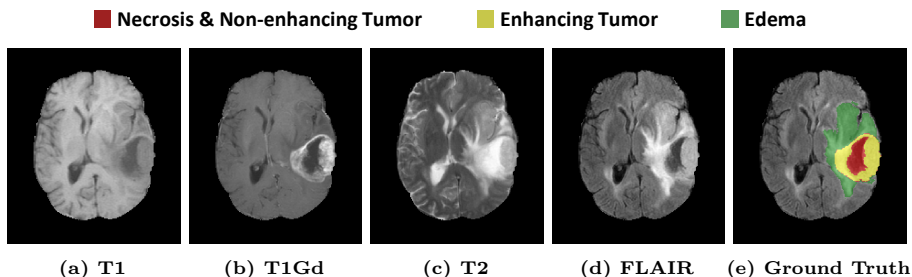


Figure 2.4 MRI sequences used in this thesis for glioma segmentation. The images belong to a glioblastoma patient from the BraTS dataset [25]. (e) The ground truth represents the tumor subregions segmentation map manually created by multiple experts. The areas of necrosis and non-enhancing tumor are shown in red, enhancing tumor in yellow, and edema in green.

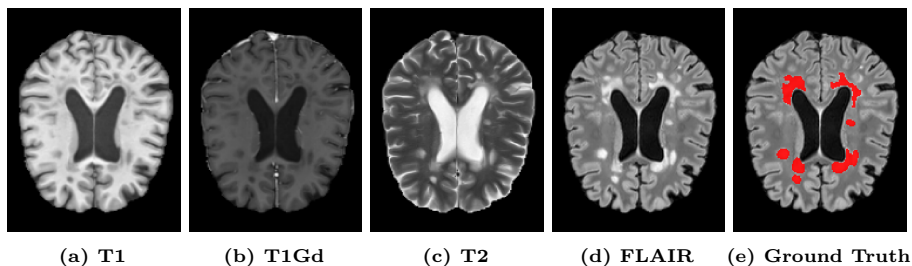


Figure 2.5 MRI sequences for MS lesion segmentation. The images belong to a MS patient from the MSSEG 2016 dataset [26]. (e) The ground truth represents the areas of MS lesions (shown in red) manually created by multiple experts.

2. **T1-weighted** can highlight T1-hypointense lesions, known as "black holes," which are chronic MS lesions that appear hypointense and represent areas of severe tissue damage generally associated with disability in MS.
3. **Gadolinium-enhanced T1-weighted (T1Gd)** is the most useful technique for identifying active lesions, which are new or enlarging lesions indicative of ongoing inflammatory activity. These lesions appear hyperintense on T1 images after the administration of gadolinium.

Ischemic Stroke. For ischemic stroke, the following sequences are typically used:

1. **DWI/ADC** is the most sensitive and specific sequence for the early detection of acute ischemic stroke since areas of restricted diffusion appear

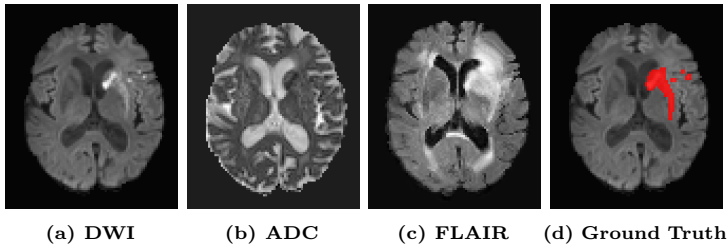


Figure 2.6 MRI sequences used in this thesis for ischemic stroke lesion segmentation. The images belong to a ischemic stroke patient from the ISLES'22 dataset [18]. (e) The ground truth represents the areas of stroke lesions (shown in red) manually created by multiple experts.

hyperintense in DWI images and hypointense in ADC images, as illustrated in Figure 2.6a and 2.6b.

2. **T2-weighted/FLAIR** can detect the presence of blood products, aiding in the identification of hemorrhagic transformation, which can complicate the treatment of ischemic stroke. Therefore, they can be useful in distinguishing ischemic stroke from hemorrhagic stroke. However, it is important to note that in the acute stages (referring to the early stages of stroke diagnosis and treatment; often the first few hours or up to a few days after a stroke occurs), it is generally less sensitive than DWI/ADC in detecting stroke-related changes. Figure 2.6c shows the FLAIR image of an ischemic stroke patient in the acute stages.

2.2 Brain Lesions

Brain lesions are abnormal areas of the brain resulting from an injury or disease, including brain tumors, multiple sclerosis (MS), and ischemic strokes. These prevalent conditions represent a significant health burden. Sections 2.2.1, 2.2.2, and 2.2.3 delve into each of these, respectively.

2.2.1 Brain Tumors

Brain tumors are a complex and diverse group of neoplasms that originate from various cells within the central nervous system (CNS). They are characterized by uncontrolled growth and proliferation of abnormal cells within the brain, leading to numerous clinical problems and complications. In the

following, we seek to provide a general understanding of brain tumors, their pathophysiology, prevalence, classification, specific types like gliomas and glioblastomas, characteristics, symptoms, causes, diagnostic methods, prognosis, and treatments.

Pathophysiology. Brain tumors are the result of genetic mutations in normal cells, which lead to uncontrolled cell division. This genetic mutation disrupts the normal cell cycle, making cells proliferate abnormally and form tumors. This means that instead of dying and being replaced as healthy cells do, these mutated cells accumulate and form a mass, known as tumor. As they grow, brain tumors can compress, displace, and infiltrate the surrounding brain tissues, causing a variety of symptoms and complications, depending on their location and size. Importantly, the brain and spinal cord have a unique protective barrier—the blood-brain barrier—which can prevent many systemic therapies from reaching the tumor region.

Prevalence. Brain tumors are not as common as other malignancies but still hold significant importance due to their serious outcomes. Brain tumors account for approximately 1.6% of all new cancer cases and 2.5% of all cancer-related deaths worldwide [27]. Malignant tumors account for nearly one-third of all primary brain tumors, with glioblastoma being the most common primary malignant brain tumour occurring in adults, accounting for nearly half of all primary malignant tumours [28]. In the United States, approximately 700,000 people are living with primary brain tumors [29]. According to the National Brain Tumor Society, around 94,390 new cases of primary brain tumors are expected to be diagnosed in the United States in 2023, with an estimated 18,990 deaths resulting from malignant brain tumors in the same year.

Based on data from 2015 to 2019, the annual incidence rate of primary malignant brain tumors in the United States is 7.02 per 100,000 population, with the rates slightly being higher in males than females [1]. Glioblastoma has the highest incidence rate (3.26 per 100,000 population) among all malignant brain tumors. Incidence rates for specific brain tumor types vary by gender (See Figure 2.7 for detailed statistics). For example, meningioma is more common in females while glioblastoma is more common in males.

Brain tumors can occur at any age, but certain types are more common in specific age groups (See Figure 2.8 for detailed statistics). For example, glioblastomas are more frequent in adults, while medulloblastomas are most often seen in children. Notably, pediatric brain tumors are the most common malignancy and the leading cause of cancer-related death among children and adolescents ages (0-19 years) [1].

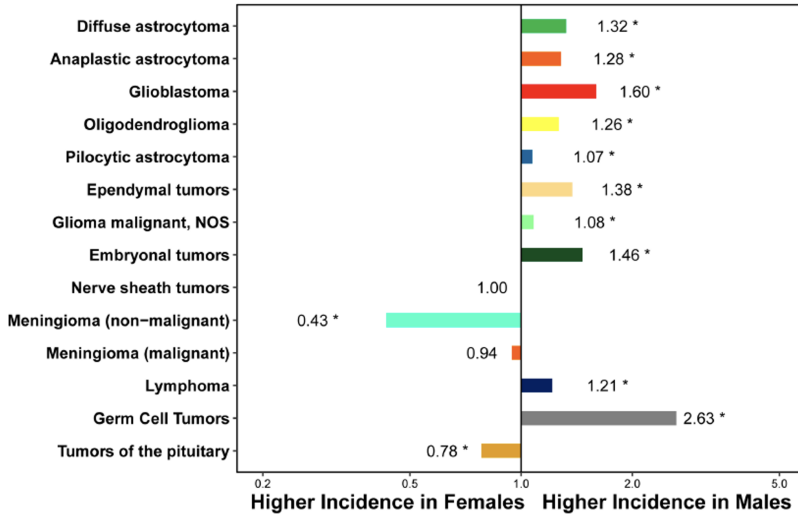


Figure 2.7 Incidence rate ratios by gender (males:females) for primary brain tumors [1]. * indicates a significant difference between the groups with p-value < 0.05.

Prognosis. Prognosis varies greatly depending on the type and grade of the tumor, the patient’s age and overall health, and the treatment’s success. For example, patients with low-grade gliomas have a median survival of several years, while those with glioblastoma have a median survival of 12-15 months with optimal treatment [30]. The five-year relative survival rate for all patients with primary brain tumors is 76% while this number for glioblastoma patients is only 6.9% [1].

Classification. Brain tumors are classified based on several factors, including their grade, whether they are primary or metastatic, the type of cell they originate from, and their tissue of origin.

- *Primary vs. Metastatic:* Primary brain tumors originate in the brain itself or in nearby tissues, such as brain-covering meninges or pituitary gland. Metastatic brain tumors, on the other hand, start somewhere else in the body (e.g., lung and breast) and spread to the brain.
- *By Grade:* The World Health Organization (WHO) [31] classifies brain tumors from grade I (least malignant) to grade IV (most malignant). Low-grade tumors (I and II) grow slowly but may become high-grade tumors (III and IV) over time. Lower-grade tumors are less severe and commonly associated with longer-term survival.

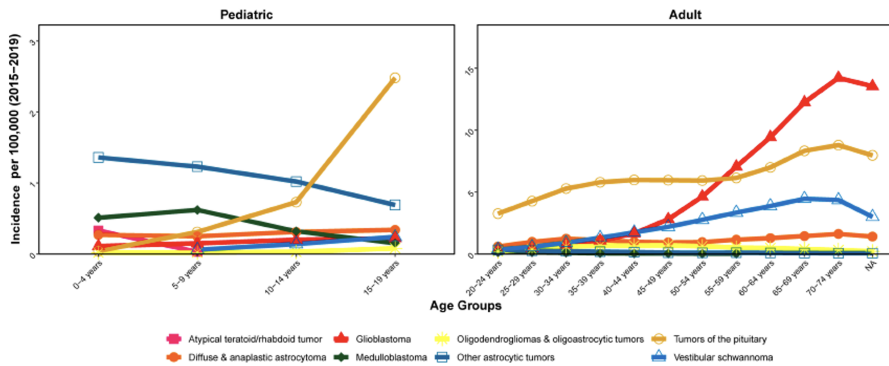


Figure 2.8 Incidence rates of brain tumors by age [1].

- *By Type of Cell:* Tumors are also classified according to the type of cells from which they originate. For instance, gliomas arise from glial cells, meningiomas from the meninges, and pituitary adenomas from the pituitary gland.

Gliomas and Glioblastoma. Gliomas are a category of brain tumors that arise from glial cells, which provide support and protection for neurons. Gliomas are further divided into astrocytomas, oligodendrogliomas, and ependymomas, named after the specific type of glial cell from which they originate. Gliomas account for approximately 30% of all brain tumors and 80% of malignant brain tumors [28].

Glioblastoma, previously known as glioblastoma multiforme (GBM), represents the Grade IV astrocytoma and is the most aggressive type of brain tumors. Glioblastomas are characterized by rapid growth, extensive vascularization, and a high degree of invasiveness. Despite modern treatments, the prognosis for glioblastoma remains poor. Glioblastomas can either occur spontaneously, known as primary glioblastomas, or evolve from lower-grade gliomas, known as secondary glioblastomas.

Tumor Subregions. Brain tumors consist of several subregions with distinct characteristics:

- *Necrosis (NRC):* This is an area of dead tumor cells resulting from a lack of oxygen or insufficient blood supply. Necrosis is common in high-grade gliomas, particularly glioblastomas.

- *Edema (ED)*: This is the swelling around the tumor, which is caused by accumulation of excess fluid.
- *Non-Enhancing Tumor (NET)*: This refers to parts of the tumor that do not "light up" after the injection of a contrast agent during MRI, often representing infiltrative tumor cells or lower-grade regions less aggressive than enhancing regions.
- *Enhancing Tumor (ET)*: This refers to the parts of the tumor that surround NRC/NET and take up contrast during MRI, often reflecting a blood-brain barrier disruption and aggressive, higher-grade tumor regions.

We evaluate our segmentation models based on three nested subregions: i) enhancing tumor (ET), ii) tumor core (TC), which is the union of ED and NCR/NET, and iii) whole tumor (WT), which entail all the tumor subregions. Figure 2.4e shows a sample image with the annotation of tumor subregions.

Causes. The causes of brain tumors are largely unknown. While some risk factors have been identified, most brain tumors occur sporadically, with no clear triggering factor. Potential risk factors include exposure to ionizing radiation, family history of brain tumors, and certain genetic disorders such as neurofibromatosis or Li-Fraumeni syndrome.

Diagnosis. The diagnosis of brain tumors heavily relies on medical imaging techniques, among which (mutiparametric) MRI the most sensitive for identifying the extent and location of brain tumors, as explained in Section 2.1.3. The definitive diagnosis requires a biopsy or surgical resection of the tumor to examine the tissue under a microscope and determine the tumor's histological grade and type.

2.2.2 Multiple Sclerosis Lesions

Pathophysiology. Multiple sclerosis (MS) is a chronic neurological disorder of the central nervous system (CNS), characterized by inflammation, demyelination, and subsequent axonal damage [32]. The pathophysiology of MS is centered around an abnormal immune response that leads to inflammation and the destruction of myelin, i.e., the insulating layer around nerve fibers in the CNS. As illustrated in Figure 2.9, when myelin is damaged or destroyed (known as demyelination), the ability of the nerves to transmit electrical impulses to and from the brain is disrupted, leading to a wide range of symptoms.

Prevalence. MS affects around 2.8 million people worldwide and is the most common neurological disorder and cause of disability in young adults. The prevalence is 36 per 100,000 people although this varies widely by region (see

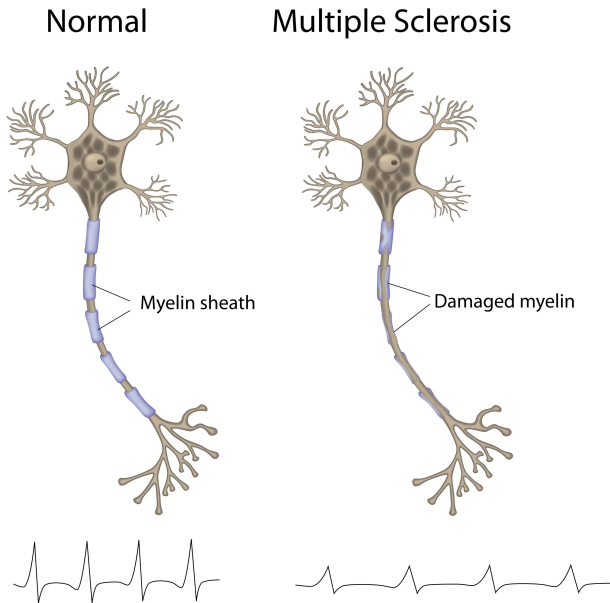


Figure 2.9 Damaged myelin in multiple sclerosis. The image is taken from <https://acls123.com/what-is-multiple-sclerosis/>.

Figure 2.10), with higher prevalence rates in North America and Europe and lower rates in Asia and Africa. MS is most commonly diagnosed between the ages of 20 and 40, and it is known to be more prevalent in colder climates further from the equator. There are at least twice as many females with MS as males [2].

Classification. MS is categorized into four types based on the disease progression pattern [33]: Relapsing-Remitting MS (RRMS), Primary Progressive MS (PPMS), Secondary Progressive MS (SPMS), and Progressive-Relapsing MS (PRMS). RRMS, the most common type, characterized by clearly defined relapses of worsening neurological function followed by periods of partial or complete recovery. SPMS begins as RRMS before eventually transitioning into a steadily progressive disease. PPMS involves a steady progression of disease from onset, without relapses. PRMS, the least common type, involves a steady progression of disease from onset with superimposed relapses. See Figure 2.11.

Multiple Sclerosis Lesions. MS lesions refer to the areas of damage caused by demyelination and inflammation. The presence, number, and location of these

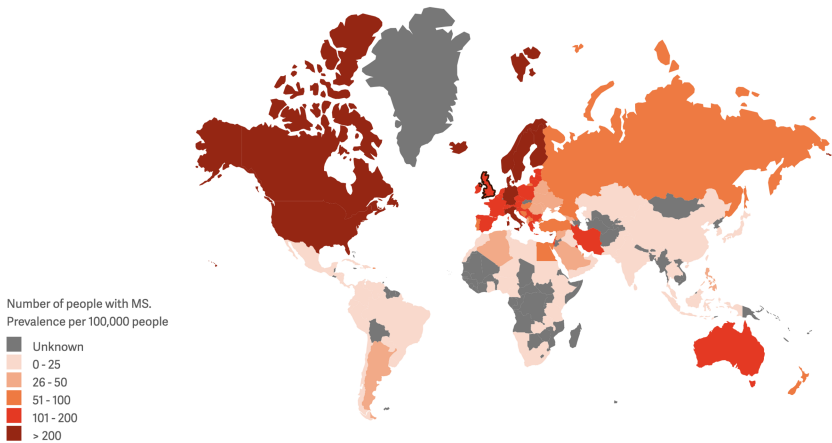


Figure 2.10 The multiple sclerosis prevalence by region [2].

lesions can be indicative of the type and severity of MS a patient may have. MS lesions can be classified based on several factors:

By Activity: MS lesions can be active or inactive (aka chronic). Active lesions are areas of ongoing inflammation where demyelination is currently occurring, usually associated with relapses. Inactive lesions are, on the other hand, areas of old damage where there is no ongoing inflammation.

By MRI Image: MS lesions are sometimes classified into three types according to the MRI image on which they appear:

- *T1 Lesions:* These are inactive lesions, known as “black holes,” often appearing persistently hypointense on T1 images even after the administration of gadolinium.
- *Gadolinium-Enhancing Lesions:* These are active inflammatory lesions appearing brighter on T1 images after the administration of gadolinium.
- *T2 Lesions:* T2-weighted images on MRI reveal areas of high signal intensity, indicating more recent disease activity. These lesions appear hyperintense on T2 images, representing areas of demyelination and inflammation. They can include both active (or new) and inactive (or old) lesions.

By Location: MS lesions can be also classified based on their locations:

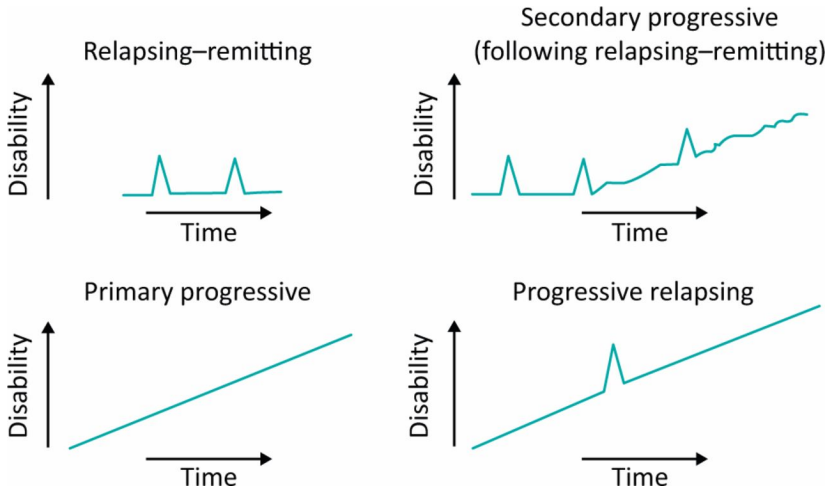


Figure 2.11 Different multiple sclerosis types based on disease progression.

- *Periventricular Lesions*: These are lesions located around the ventricles of the brain, which are fluid-filled spaces.
- *Juxtacortical/Cortical Lesions*: Juxtacortical lesions are located near the cortex, while cortical lesions are located within the cortex itself. Cortical lesions are more difficult to see on conventional MRI scans.
- *Infratentorial Lesions*: These are lesions found in the infratentorial region of the brain, which includes the brainstem and cerebellum.
- *Deep White Matter Lesions*: These refer to lesions located deep within the white matter, often in the deeper parts of the cerebral hemispheres.

Causes. MS is thought to be autoimmune in nature, meaning that the body's immune system mistakenly attacks the myelin sheath. However, the exact cause of MS remains unknown although it is believed to be a combination of genetic and environmental factors, including viral infections (like Epstein-Barr virus), low vitamin D levels, and smoking [34].

Diagnosis. The diagnosis of MS involves the demonstration of lesion dissemination in both time and space. This means lesions must be shown to occur at different times and in different locations within the CNS. MRI plays a crucial role in the diagnosis, as it provides clear visualization of demyelination and inflammation in the brain and spinal cord. Moreover, MRI can distinguish among various types of lesions and significantly assist in monitoring disease progression.

Prognosis. MS is a chronic condition with a variable prognosis. Most people with MS have a normal or near-normal life expectancy, but MS can significantly affect their quality of lives. People with MS live, on average, about 7-14 years less than the general population [35]. Factors that may influence prognosis include the type of MS, age at diagnosis, and individual response to treatment.

2.2.3 Stroke Lesions

Stroke is an acute neurological condition affecting millions worldwide [4]. Strokes occur when blood flow to the brain is obstructed, causing brain cells to die from lack of oxygen and essential nutrients. The ensuing cell death results in a region of damage, known as a stroke lesion, which directly impacts the brain's function.

Pathophysiology and Classification. A stroke is classified into two primary types [36]: ischemic and hemorrhagic. About 87% of strokes are ischemic [4], which result from a blockage in the blood vessels supplying the brain. This blockage often stems from a thrombus (blood clot) or an embolus (a clot that travels from elsewhere in the body). Ischemic stroke lesions result from the immediate and progressive death of brain cells deprived of oxygen and glucose. The severity and location of these lesions dictate the symptoms experienced by the patient.

Hemorrhagic stroke, a less common type, occurs when a blood vessel in the brain ruptures, causing bleeding in the surrounding brain tissue and increasing pressure on the brain [37]. The damage caused is twofold: from the direct effects of bleeding and from the compression of brain tissue due to swelling.

Prevalence. According to the World Health Organization (WHO), strokes are the second leading cause of death worldwide. Globally, one in four people over age 25 will have a stroke in their lifetime. There are over 12.2 million new strokes each year, with more than half of the incidents resulting in death while another half resulting in permanent disability [4].

Causes. Several factors contribute to the risk of stroke, including modifiable and non-modifiable factors. The primary non-modifiable risk factors include age, sex, and family history. The modifiable risk factors include hypertension, diabetes, smoking, obesity, physical inactivity, and poor diet. Other conditions like atrial fibrillation, which can cause clots to form in the heart and subsequently lodge in the cerebral arteries, can also lead to ischemic stroke.

Diagnosis. Diagnosis of a stroke is a medical emergency. Medical imaging is a critical tool in diagnosing a stroke and determining its type, location, and

extent. Computed tomography (CT) scan is usually the first imaging test to rule out a hemorrhage, but magnetic resonance imaging (MRI) is more sensitive in detecting ischemic stroke lesions, particularly in the early stage. As mentioned in Section 2.1.3, diffusion-weighted imaging (DWI) can reveal ischemic lesions within minutes of symptom onset, offering a significant advantage in initiating early treatment.

Prognosis. Prognosis after stroke varies widely, depending on the extent of brain injury and the patient's general health status. While some patients recover fully, others may have long-term or permanent disabilities. Stroke is the leading cause of adult disability, with many survivors requiring extensive rehabilitation to regain lost skills. Moreover, the risk of another stroke or a related medical condition is elevated after a stroke.

2.3 Conclusion

In this chapter, we delved into the fundamental physics behind MRI and discussed various MRI sequences, highlighting the potential of mpMRI as a powerful imaging technique for diagnosing glioma, MS, and stroke. We examined T1, T1Gd, T2, FLAIR, DWI, and ADC sequences, detailing the types of lesions each can reveal. In Chapter 6, the FLAIR sequence will be employed to detect MS white matter lesions. Chapters 7 and 8 will use T1, T1Gd, T2, and FLAIR sequences for glioma segmentation. Chapter 7 will also incorporate DWI and ADC sequences for segmenting stroke lesions.

Chapter 3

Machine Learning Essentials

In this chapter, we begin by reviewing the fundamental concepts of machine learning. Our primary focus is on supervised learning and model fitting methods, which form the basis of our approach to brain lesion segmentation using deep learning. We also discuss first-order optimization techniques, including stochastic gradient descent and Adam, which are essential for training deep models on a large scale. Lastly, we delve into linear models and kernel regression, shedding light on their evolution into more complex, non-linear models, specifically deep neural networks.

3.1 When Models Meet Data

Machine learning is a subfield of artificial intelligence that involves the development of algorithms and models that enable machines to learn from data and make predictions or decisions without being explicitly programmed [38]. It seeks to extrapolate patterns and understand complex behaviors from input data through a process that begins with feeding data into an algorithm, and ends with the algorithm predicting unknown output or future behavior.

3.1.1 Learning Scenarios

Machine learning tasks are typically classified into four categories [39]: *supervised learning*, *unsupervised learning*, *semi-supervised learning*, and *reinforcement learning*. The first two types are the most commonly used and are discussed in the following.

Supervised Learning. This is the most common paradigm of machine learning. Suppose, we have a training set of N input-output pairs $\mathbb{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{X}$ represents the input (also known as *feature vector*) of n th example and $\mathbf{y}_n \in \mathbb{Y}$ is its corresponding output, known as *label*. Here, \mathbb{X} is the input space, and \mathbb{Y} is the label space. The primary goal is to infer a function $f : \mathbb{X} \rightarrow \mathbb{Y}$ that maps inputs to their corresponding labels. This function can then be used for mapping any valid unseen input to accurately predict its label. Depending on the label space, supervised learning problems are classified into two categories:

- **Classification:** In classification problems, the label space is a discrete set of K unordered and mutually exclusive *classes*, i.e., $\mathbb{Y} = \{1, 2, \dots, K\}$. If there are just two classes, $\mathbb{Y} = \{0, 1\}$ or $\{-1, +1\}$, it is called binary classification. For example, a tissue can be classified as “normal” or “abnormal”. The task is to predict the class of an input.
- **Regression:** In regression problems, the label space is continuous, i.e., $\mathbb{Y} \subseteq \mathbb{R}$. For example, a tumor can be labeled with its volume. The task is to predict a real-valued output given an input.

Unsupervised Learning. In this paradigm, observed training inputs are not paired with a corresponding output, which means that we just have an unlabeled training set $\mathbb{D} = \{\mathbf{x}_n\}_{n=1}^N$. In contrast to supervised learning, the task is to try to “make sense of” data often by discovering interesting structure in the data, such as clusters, densities, or low-dimensional structures [39].

3.1.2 Data and Models

Data as Tensors. In machine learning, data is typically represented as tensors, which are a generalization of vectors and matrices to an arbitrary number of dimensions (or indices). Tensors can be represented as *multi-dimensional arrays* of numbers. For example, a 1st-order tensor is a vector, a 2nd-order tensor is a matrix, and a 3rd-order tensor can be visualized as a cube of numbers.

In general, an M th-order tensor can be represented as an M -dimensional array $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_M}$. The elements of this tensor are denoted by $\mathcal{X}_{i_1, \dots, i_M}$, with i_m index running from 1 to I_m . The *order* of a tensor refers to the dimension of its corresponding array, which equals the total number of indices (M) required to identify each element uniquely. Thus, an M th-order tensor may be also referred to as M -dimensional (or MD) tensor.

Tensors allow us to represent complex data structures in a uniform way. For instance, a simple tabular dataset with N D -dimensional feature vectors, $\{\mathbf{x}_n\}_{n=1}^N \subseteq \mathbb{R}^D$, is commonly stored in an $N \times D$ *data matrix* $[\mathbf{x}_1 \mid \dots \mid \mathbf{x}_N]^\top$, in which each row represents an example, and each column represents a feature. A grayscale image of size (H, W) can be represented as a matrix $\mathbf{I} \in \mathbb{R}^{H \times W}$, where each element corresponds to the pixel intensity at a particular position, while a color image can be represented as a 3D tensor $\mathbf{I} \in \mathbb{R}^{3 \times H \times W}$, with the first dimension corresponding to the color channels (red, green, and blue).

Models as Functions. In supervised learning, a model can be a prediction function that takes input data and transform it into output predictions. For instance, in a problem of semantic segmentation of natural images, a model could be a function $f : [0, 1]^{3 \times H \times W} \rightarrow \{1, \dots, K\}^{H \times W}$ that maps an image to its mask, where K is the number of segmentation classes.

A prediction function f is typically parameterized by parameters θ , which are learned from the data. We write $f(\mathbf{x}; \theta)$ to denote a model with parameters θ . In the simplest case, for example, a model can be a linear function:

$$f(\mathbf{x}; \theta) = \mathbf{w}^\top \mathbf{x} + b \quad (3.1)$$

where $\mathbf{x} \in \mathbb{R}^D$ is the input, $\theta = (\mathbf{w}, b)$ are all parameters, $\mathbf{w} \in \mathbb{R}^D$ are the weights, and $b \in \mathbb{R}$ is the bias or intercept. More complex models include nonlinear functions, such as those used in neural networks, where multiple linear transformations and nonlinear activations are applied.

Models as Probability Distributions. In machine learning, particularly in statistical modeling and Bayesian inference, the assumption is that a probability distribution exists over the example space, and that training examples are drawn independently and identically distributed (iid) from this distribution. As such, it is logical to consider models as probability distributions.

In supervised learning, we consider training examples $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_N, \mathbf{y}_N)$ drawn iid from an unknown joint probability distribution $p(\mathbf{x}, \mathbf{y})$ over \mathbb{X} and \mathbb{Y} . The statistical models seek to estimate the conditional distribution $p(\mathbf{y} | \mathbf{x})$, which represent the probability of output \mathbf{y} given input \mathbf{x} . In the case of a parametric model, this distribution falls within a parametric family $\{p(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}) | \boldsymbol{\theta} \in \Theta\}$. Here, Θ denotes the *parameter space*, and the parameters $\boldsymbol{\theta}$ are estimated based on the given data. Note that the assumption of a joint probability distribution allows us to model uncertainty in predictions (e.g., from noise in data) because \mathbf{y} is not a deterministic function of \mathbf{x} , but rather a random variable with conditional distribution $p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$ for a fixed \mathbf{x} and estimated $\boldsymbol{\theta}$. We can also obtain a prediction function by computing the conditional expectation

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathbb{E}[\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}] = \int \mathbf{y} p(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}) d\mathbf{y} \quad (3.2)$$

This probabilistic perspective holds particular benefits in situations where we want to quantify the uncertainty of our predictions, incorporate prior knowledge, or manage noise in the data. Instead of viewing a predictor as a deterministic function, it can be more beneficial to consider predictors as statistical models. For instance, in classification problems, the model might output a probability distribution over class labels, which can be achieved using transformations like the softmax function. This distribution indicates the probability of each class label given an input, helping to express the level of confidence about a prediction for a specific test data point.

3.1.3 Model Fitting

Once we have defined our data and models, the next step is to fit our models to the data. Model fitting involves adjusting the model parameters to best match the training data.

Empirical Risk Minimization

Consider a general supervised learning setting, where a joint distribution $p(\mathbf{x}, \mathbf{y})$ generates the data. Our aim is to learn a parameterized function $f : \mathbb{X} \times \Theta \rightarrow \hat{\mathbb{Y}}$, where \mathbb{X} is the input space, Θ is the parameter space, and $\hat{\mathbb{Y}}$ is the prediction space. Note that while the prediction space often equals the label space \mathbb{Y} , they may differ in some cases.

Suppose we are provided with a loss function $\ell : \mathbb{Y} \times \hat{\mathbb{Y}} \rightarrow \mathbb{R}_{\geq 0}$, which measures the discrepancy between the actual output and the model output. For instance,

Table 3.1 Various loss functions used in supervised learning.

Loss Function ($\ell : \mathbb{Y} \times \hat{\mathbb{Y}} \rightarrow \mathbb{R}_{\geq 0}$)	Label Space (\mathbb{Y})	Prediction Space ($\hat{\mathbb{Y}}$)	Task
Binary Cross-Entropy (aka Binary Log Loss): $-y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y})$	$\{0, 1\}$	$[0, 1]$	Binary Classification
Multiclass Cross-Entropy (aka Multiclass Log Loss): $-\sum_{k=1}^K \mathbf{1}(y = k) \ln(\hat{y}_k)$	$\{1, 2, \dots, K\}$ K : # classes	$[0, 1]$	Multiclass Classification
Hinge Loss: $\max(0, 1 - y\hat{y})$	$\{-1, 1\}$	\mathbb{R}	Binary Classification
Squared Error: $(y - \hat{y})^2$	\mathbb{R}	\mathbb{R}	Regression
Dice Loss: $1 - \frac{2 \sum_{m=1}^M y_m \hat{y}_m}{\sum_{m=1}^M (y_m + \hat{y}_m)}$	$\{0, 1\}^M$ M : # pixels/voxels 0: background, 1: foreground	$[0, 1]^M$ M : # pixels/voxels	Semantic Segmentation
Soft Dice Loss: $1 - \frac{2 \sum_{m=1}^M y_m \hat{y}_m}{\sum_{m=1}^M (y_m^2 + \hat{y}_m^2)}$	$\{0, 1\}^M$ M : # pixels/voxels 0: background, 1: foreground	$[0, 1]^M$ M : # pixels/voxels	Semantic Segmentation

in regression tasks, a frequently used loss function is the *squared error* loss, defined as:

$$\ell(y, \hat{y})_{se} = (y - \hat{y})^2, \quad y, \hat{y} \in \mathbb{R}$$

Table 3.1 provides some common loss functions. The *risk* (also known as the *expected loss*) of a function $f(\mathbf{x}; \boldsymbol{\theta})$ is defined as the expectation of the loss over the entire example space according to the joint distribution $p(\mathbf{x}, \mathbf{y})$ [39]

$$R(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathbf{x}, \mathbf{y}}[\ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))] \quad (3.3)$$

A lower risk means that the prediction function performs better on average across all possible examples from the distribution. Therefore, the ultimate goal of a learning algorithm is to find a parameter $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that minimizes the risk. However, since in practice the true distribution $p(\mathbf{x}, \mathbf{y})$ is unknown and inaccessible, we cannot directly compute the risk. Instead, we have a dataset $\mathbb{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ sampled iid from the distribution $p(\mathbf{x}, \mathbf{y})$ to estimate the risk. By simply computing the average loss over this dataset \mathbb{D} , we get an unbiased estimation of the risk, referred to as the *empirical risk*, defined as:

$$R(\boldsymbol{\theta}; \mathbb{D}) \triangleq \frac{1}{N} \sum_{n=1}^N \ell(\mathbf{y}_n, f(\mathbf{x}_n; \boldsymbol{\theta})) \quad (3.4)$$

The law of large numbers guarantees that the empirical risk converges to the risk as the number of examples N approaches infinity.

The empirical risk minimization (ERM) is a principle in statistical learning that suggests choosing the parameters that minimize the empirical risk on the

training set. Mathematically, ERM can be written as:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \Theta} R(\boldsymbol{\theta}; \mathbb{D}) \quad (3.5)$$

The ERM strategy is most effective when we have a large number of training examples so that the empirical risk is a good approximation of the true risk. While ERM provides a theoretically grounded approach to learning, in practice, it can lead to overfitting if the model space is too complex. This aspect will be discussed further in Section 3.1.4.

Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is a method of estimating the parameters of a statistical model. The goal of MLE is to find the parameter values that maximize the “likelihood” of generating the observed data, under the given model.

Let’s consider a dataset $\mathbb{D} = \{\mathbf{x}_n\}_{n=1}^N$ in unsupervised learning or $\mathbb{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ in supervised learning. Assume we have a statistical model with a set of parameters $\boldsymbol{\theta}$, which generates examples.

The likelihood function $\mathcal{L}(\boldsymbol{\theta})$ is then defined as the probability of observing the data given the parameters, that is

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq p(\mathbb{D} \mid \boldsymbol{\theta}). \quad (3.6)$$

In unsupervised learning with the assumption of iid examples, the likelihood function can be expressed as a product of probabilities of individual examples, that is

$$p(\mathbb{D} \mid \boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n \mid \boldsymbol{\theta}). \quad (3.7)$$

The MLE approach aims to find the parameters $\boldsymbol{\theta}$ that maximize this likelihood function

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} \triangleq \arg \max_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}), \quad (3.8)$$

It is often more practical to work with the natural logarithm of the likelihood function, called the *log-likelihood*, defined as

$$\ell(\boldsymbol{\theta}) \triangleq \ln p(\mathbb{D} \mid \boldsymbol{\theta}). \quad (3.9)$$

Taking the logarithm does not change the parameter values that maximize the likelihood due to the monotonicity of the log function, but it simplifies the

expression and the calculation of the differentiation by transforming the product into a sum when the iid assumption holds

$$\ell(\boldsymbol{\theta}) \triangleq \ln p(\mathbb{D} \mid \boldsymbol{\theta}) = \sum_{n=1}^N \ln p(\mathbf{x}_n \mid \boldsymbol{\theta}). \quad (3.10)$$

In supervised learning, where each input \mathbf{x}_n has a corresponding label \mathbf{y}_n , we are interested in the conditional likelihood of the outputs given the inputs. The *conditional likelihood function* is expressed as

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{y}_n \mid \mathbf{x}_n, \boldsymbol{\theta}), \quad (3.11)$$

and the *conditional log-likelihood* as

$$\ell(\boldsymbol{\theta}) = \sum_{n=1}^N \ln p(\mathbf{y}_n \mid \mathbf{x}_n, \boldsymbol{\theta}). \quad (3.12)$$

Again, the MLE finds the parameter values $\boldsymbol{\theta}$ that maximize these likelihoods, in other words, the parameter values that make the observed outputs most probable given the inputs.

The process of MLE often involves differentiating the log-likelihood and setting the derivatives to zero, and then solving for the parameters, given the constraint that the likelihood is maximized. In some cases, this can be done analytically, but in many situations, especially in machine learning with complex models, numerical methods must be used.

3.1.4 Generalization and Overfitting

Generalization refers to the ability of the model to perform well on new, previously unseen data, drawn from the same distribution as the one used to train the model [39]. In other words, a good learning algorithm should not only perform well on the training data, but it should also generalize well to new examples. Formally, generalization is related to the ability of a prediction function $f : \mathbb{X} \times \boldsymbol{\Theta} \rightarrow \mathbb{Y}$ with estimated parameters $\boldsymbol{\theta}$ to have a small true risk $R(\boldsymbol{\theta})$, which is often referred to as “generalization error” in this context. We say a model generalizes well if the true risk $R(\boldsymbol{\theta})$ is small.

As explained earlier in Section 3.1.3, we can use the ERM principle to estimate the parameters. However, the estimated parameters $\boldsymbol{\theta}$ are based on a finite dataset \mathbb{D} , and thus the empirical risk $R(\boldsymbol{\theta}; \mathbb{D})$ might not reflect the true

risk $R(\boldsymbol{\theta})$. The difference between the true risk and the empirical risk, i.e., $R(\boldsymbol{\theta}) - R(\boldsymbol{\theta}; \mathbb{D})$, is known as the *generalization gap*. In the ideal scenario, where the dataset is sufficiently large, the generalization gap should be small for all $\boldsymbol{\theta}$, implying that the empirical risk accurately estimates the true risk. However, this is not always the case in reality, and we often experience a phenomenon called *overfitting*.

In practice, we cannot compute the true risk $R(\boldsymbol{\theta})$ since we do not know the true distribution that generates the data. However, we can partition the dataset we have into two subsets, known as the *training set* ($\mathbb{D}_{\text{train}}$) and the *test set* (\mathbb{D}_{test}). Following the ERM principle, we first estimate the parameters by minimizing the training empirical risk $R(\boldsymbol{\theta}; \mathbb{D}_{\text{train}})$. Then we can approximate the true risk using the test empirical risk [39]:

$$R(\boldsymbol{\theta}; \mathbb{D}_{\text{test}}) = \frac{1}{|\mathbb{D}_{\text{test}}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathbb{D}_{\text{test}}} \ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta})) \quad (3.13)$$

Overfitting refers to the situation where a model performs very well on the training data but performs poorly on unseen test data [39]. In terms of risk, overfitting corresponds to a small empirical risk $R(\boldsymbol{\theta}; \mathbb{D}_{\text{train}})$ but a large true risk $R(\boldsymbol{\theta})$, which results in a large generalization gap. This typically happens when the model is too complex and captures the noise in the training data. In fact, with a suitably flexible model, we can drive the empirical risk to zero by simply memorizing the correct output for each input.

As an example, let's consider a polynomial regression problem where the training examples are noisy observations of a quadratic function. In Figure 3.1, we fit polynomials of different degrees and compare their performances. The plot in Figure 3.1a shows that a linear function does not capture the quadratic nature of the data. It is too simple to represent the data accurately, resulting in poor performance. In contrast, Figure 3.1c illustrates that a polynomial of degree 20, which passes through every training point, might seem ideal. However, this model is too complex and ends up learning the noise and outliers in the data. Consequently, it performs poorly on new, unseen data, as it is tailored too specifically to the training data. Finally, Figure 3.1b shows that a quadratic function accurately matches the true function and the data. This model strikes the right balance of complexity: it is complex enough to capture the underlying pattern but not so complex that it absorbs the noise in the data.

In Figure 3.1d, we plot the mean squared error (MSE) of polynomial regression as a function of degree for both the training and the test sets. We observe that the training error diminishes to zero as the degree increases, and the model becomes more complex. However, the test error displays a characteristic U-shaped curve: on the left, where the degree is 1, the model is underfitting; on

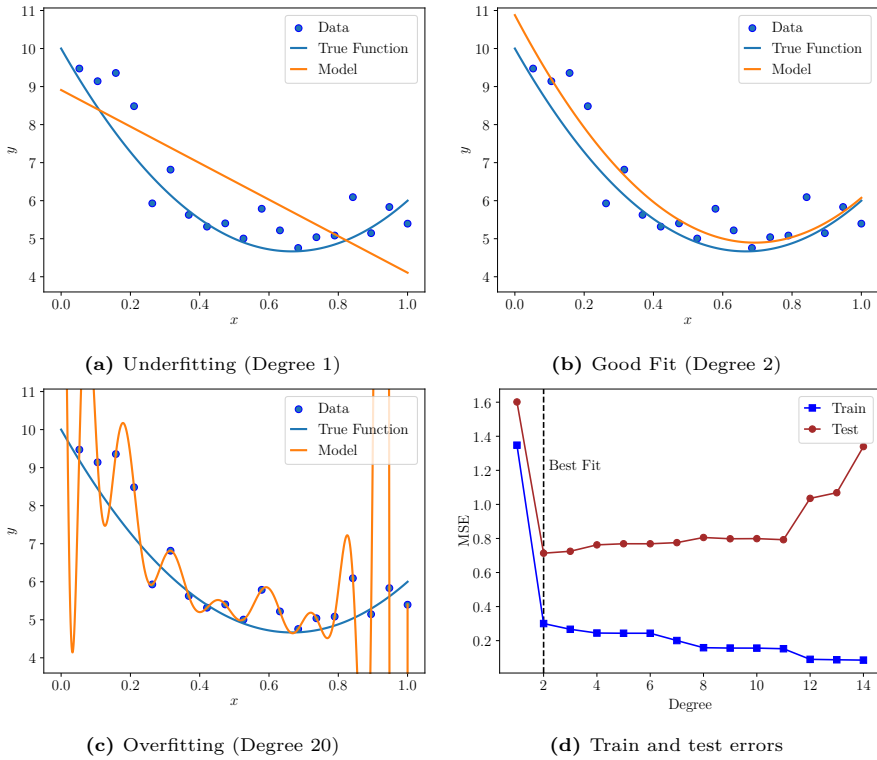


Figure 3.1 Comparison of polynomial regression models of different degrees. The true function is a quadratic polynomial $f(x) = 12x^2 - 16x + 10$. The training examples are derived from noisy observations of this function, specifically $y_n = f(x_n) + r_n$, where $r_n \stackrel{\text{iid}}{\sim} \mathcal{N}(0, 0.5)$.

the right, where the degree is 20, the model is overfitting; and when the degree is 2, the model complexity is "just right". Figure 3.2 illustrates the common scenario observed in risk curves as the model complexity increases.

In practice, some form of regularization may be used to control the complexity of the learned model and avoid overfitting. For instance, a regularization term is sometimes added to the empirical risk to limit the complexity of the model, resulting in

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \Theta} R(\boldsymbol{\theta}; \mathbb{D}) + \lambda C(\boldsymbol{\theta}) \quad (3.14)$$

where $C(\boldsymbol{\theta})$ measures the complexity of the prediction function $f(\mathbf{x}_n; \boldsymbol{\theta})$, and λ is the regularization hyperparameter that controls the strength of the complexity penalty. Common examples of the complexity measure $C(\boldsymbol{\theta})$ include the L^2

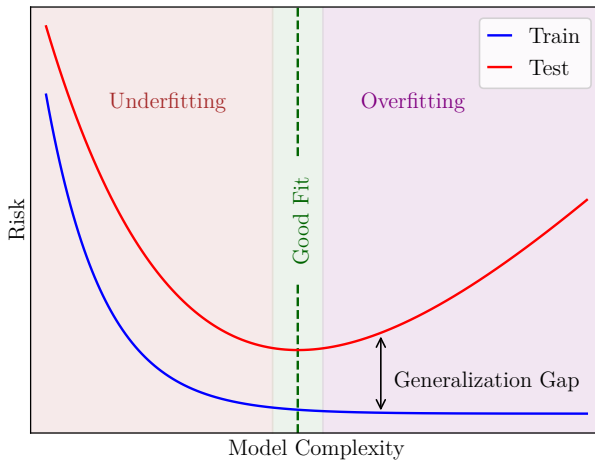


Figure 3.2 Illustration of underfitting, optimal fitting, and overfitting in model training. The figure shows how the training error decreases with model complexity, while the test error initially decreases and then increases, indicating overfitting. The area between the two curves represents the generalization gap. The shaded regions represent underfitting (brown), optimal fitting (green), and overfitting (purple) areas.

norm (i.e., $C(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$), which penalizes large parameter values, as seen in ridge regression [40], and the L^1 norm (i.e., $C(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$), which encourages sparsity in the parameters, as utilized in LASSO [41]. The choice of λ can be crucial in balancing the trade-off between fitting the training data well (minimizing $R(\boldsymbol{\theta}; \mathbb{D})$) and keeping the model simple (minimizing $C(\boldsymbol{\theta})$). When λ is too large, the model may become too simple and underfit the data, which means the model does not capture the underlying structure of the data, resulting in a large empirical risk $R(\boldsymbol{\theta}; \mathbb{D})$ and a large true risk $R(\boldsymbol{\theta})$. Conversely, when λ is too small, the model may become too complex and overfit the data, resulting in a small $R(\boldsymbol{\theta}; \mathbb{D})$ but a large $R(\boldsymbol{\theta})$.

3.1.5 Model Selection and Validation

After fitting various models to the data, we need to select the best model and validate its performance. Model selection involves choosing the best model from a set of candidate models based on their performance according to some criterion. These models can be from different families or from the same family but with different hyperparameters. Model validation, on the other hand, is the process of estimating the generalization performance of the chosen model

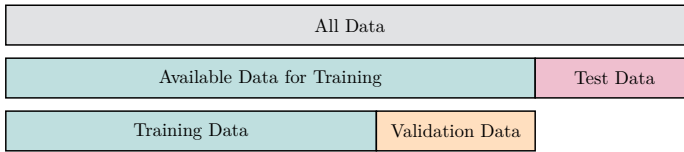


Figure 3.3 An illustration of the holdout method.

on unseen data. These two steps are crucial in preventing overfitting and in developing a successful prediction model for any machine learning problem.

Note that we always assume there is an “available dataset” \mathbb{D} which we can use throughout training, selecting, and validating models, whereas the test set \mathbb{D}_{test} is kept in a “vault” in all the stages of model development and brought out only at the end of our data analysis to evaluate the performance of the final selected model.

Cross-Validation

Cross-validation is the most common technique used for model selection and validation. It provides a robust method to estimate the performance of a model on unseen data. Cross-validation divides the dataset into subsets, iteratively trains a candidate model on some of these subsets, and validates the model on the remaining subsets.

Holdout Method. The simplest form of cross-validation is the holdout method, where the available dataset \mathbb{D} is, as illustrated in Figure 3.3, randomly split into two disjoint subsets: a training set $\mathbb{D}_{\text{train}}$ and a validation set (or holdout set) \mathbb{D}_{val} . The model is trained on the training set, and its performance is then evaluated on the validation set. If we are in a data-rich situation, this is the best approach for both model selection and model validation.

The ratio between the training and test sets can vary depending on the size of the dataset and the desired evaluation accuracy. A common practice is to use 70-80% of the data for training and the remaining 20-30% for validation. The model’s performance is generally measured in terms of some metrics, such as accuracy, F1 score, and Dice, providing an estimate of how well the model will perform on unseen data.

The main advantage of the holdout method is its simplicity and speed, as training is done once and validation is also done once. However, the major drawback of this method is its high variance, meaning the evaluation may depend heavily on which data points end up in the training set and which end

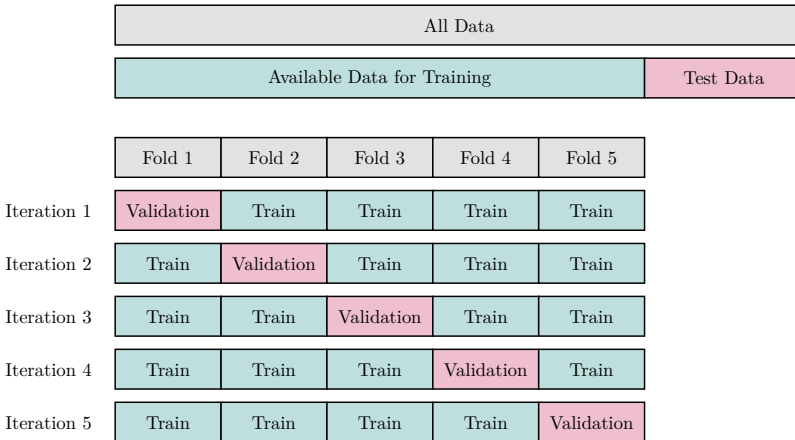


Figure 3.4 Schematic of 5-fold cross validation.

up in the validation set. Furthermore, the holdout method does not utilize the entire available dataset for both training and validation. This is where K-Fold cross-validation comes into play.

K-fold Cross-Validation. To address the limitations of the holdout method, K-fold cross-validation is often used. In this method, the dataset \mathbb{D} is split into K disjoint subsets of approximately equal size, referred to as “folds”. The model is trained K times, with each fold serving as the validation set once while the remaining $K - 1$ folds as the training set, as sketched in Figure 3.4. The model’s performance is then estimated by averaging the scores obtained from the K iterations. Figure 3.4 illustrates this method.

The value of K can vary depending on the size of the dataset and computational resources. Common choices for K include 5 and 10. The advantage of this method over the holdout method is that it matters less how the data gets divided. Every data point appears in a validation set exactly once and appear in a training set $K - 1$ times. By averaging the performance over multiple folds, it also provides a more reliable estimate of the model’s performance compared to the holdout method.

Nested Cross-Validation. To enhance the assessment of the model’s generalizability, *nested* cross-validation can be implemented. In this approach, two cross-validation processes are nested within one another. The outer cross-validation splits all the data into training and testing sets, while the inner cross-validation, executed within each training set of the outer loop, is responsible for model selection and hyperparameter tuning. This nested structure ensures

that the evaluation of the model's performance is not biased by the selection of hyperparameters, resulting in a more rigorous evaluation.

Conversely, *non-nested* cross-validation is a simpler approach where a single set of folds is used for both model assessment and tuning. Although less computationally intensive, it may result in a biased evaluation and overly optimistic performance estimates due to the leakage of information. However, for large datasets where computational efficiency is a concern, *non-nested* cross-validation may still provide a reasonable estimate of the model's ability to generalize.

3.2 Optimization

Machine learning is inherently an optimization problem, as it seeks the optimal parameter values that minimize a cost function, such as empirical risk or negative log-likelihood. Generally, the optimization problem in machine learning can be formulated as follows:

$$\theta^* \in \arg \min_{\theta \in \Theta} J(\theta), \quad (3.15)$$

where $J : \Theta \rightarrow \mathbb{R}$ is the cost function that measures the error of our model, and $\Theta \subseteq \mathbb{R}^D$ is the parameter space, comprising all possible values of the model's parameters. The parameters θ^* that minimize the cost function are the solution to our optimization problem. In this section, we provide a brief overview of the common optimization methods used in machine learning to find local minima.

Definition 3.1 (Local Minima). θ^* is a local minimum if there exists $\epsilon > 0$ such that $J(\theta^*) \leq J(\theta)$ for all θ satisfying $\|\theta - \theta^*\| \leq \epsilon$.

Definition 3.2 (Global Minima). θ^* is a global minimum if $J(\theta^*) \leq J(\theta)$ for all $\theta \in \Theta$.

3.2.1 Gradient Descent

Gradient descent is an iterative optimization algorithm used to find a local minimum of a differentiable function [42]. The idea is to start from an initial guess and iteratively update the parameters in the direction opposite to the gradient at the current point.

Definition 3.3 (Gradient). *The gradient of a function $J : \mathbb{R}^D \rightarrow \mathbb{R}$ at a point $\boldsymbol{\theta} \in \mathbb{R}^D$ is defined as the vector of its first-order partial derivatives:*

$$\nabla J(\boldsymbol{\theta}) \triangleq \left[\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2}, \dots, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_D} \right]^\top.$$

Definition 3.4 (Directional Derivative). *The directional derivative of a function $J : \mathbb{R}^D \rightarrow \mathbb{R}$ at a point $\boldsymbol{\theta} \in \mathbb{R}^D$ in the direction of a unit vector \mathbf{u} is given by*

$$\nabla_{\mathbf{u}} J(\boldsymbol{\theta}) = \lim_{\eta \rightarrow 0} \frac{J(\boldsymbol{\theta} + \eta \mathbf{u}) - J(\boldsymbol{\theta})}{\eta} \quad (3.16)$$

The motivation behind gradient descent is that the gradient of a function at a point provides the direction of steepest ascent at that point. Therefore, moving in the direction opposite to the gradient should locally decrease the function value at the fastest rate. To understand this, let's consider the first-order Taylor expansion of the function $J(\boldsymbol{\theta} + \eta \mathbf{u})$ around the point $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta} + \eta \mathbf{u}) = J(\boldsymbol{\theta}) + \eta \mathbf{u}^\top \nabla J(\boldsymbol{\theta}) + o(\eta), \quad (3.17)$$

where \mathbf{u} is a unit vector indicating the direction of the step and $o(\eta)$ is a term that diminishes faster than the other terms as $\eta \rightarrow 0$, that is $\lim_{\eta \rightarrow 0} \frac{o(\eta)}{\eta} = 0$. By plugging equation (3.17) into the definition of the directional derivative in equation (3.16), we obtain $\nabla_{\mathbf{u}} J(\boldsymbol{\theta}) = \mathbf{u}^\top \nabla J(\boldsymbol{\theta})$. In other words, the local change in the function value is represented by the dot product $\eta \mathbf{u}^\top \nabla J(\boldsymbol{\theta})$. According to the Cauchy–Schwarz inequality, this dot product is minimized if and only if

$$\mathbf{u} = -\frac{\nabla J(\boldsymbol{\theta})}{\|\nabla J(\boldsymbol{\theta})\|} \quad (3.18)$$

Therefore, the gradient gives the direction of steepest descent. Starting from an initial guess $\boldsymbol{\theta}_0$, gradient descent updates the parameters at each iteration t using the following rule [42]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla J(\boldsymbol{\theta}_t), \quad t \geq 0, \quad (3.19)$$

where $\eta_t \in \mathbb{R}_+$ is the step size or learning rate. For a sufficiently small η_t , we have $J(\boldsymbol{\theta}_{t+1}) \leq J(\boldsymbol{\theta}_t)$. In machine learning, the sequence of step sizes η_t is called the learning rate schedule, which plays a crucial role in the convergence and performance of the optimization process. There are several widely used methods for picking this, some of which we discuss below.

Fixed Learning Rate. For gradient descent with a fixed learning rate $\eta_t = \eta$, convergence to a local minimum is guaranteed under mild conditions on the

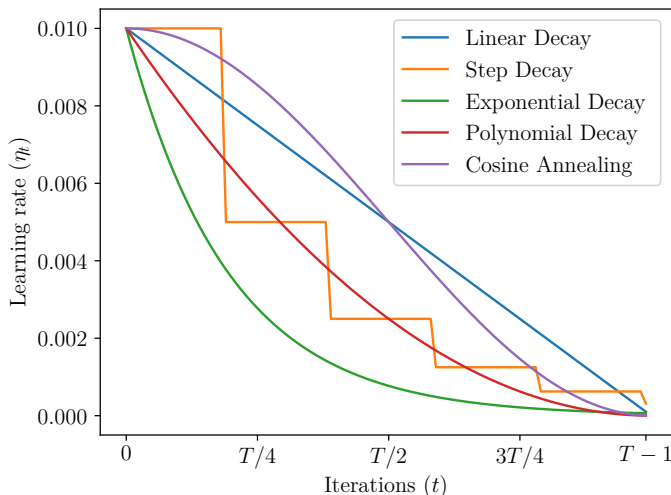


Figure 3.5 Comparison of different learning rate schedules over iterations. (i) Linear Decay: $\eta_t = \eta_0(1 - t/T)$, where T is total number of iterations. (ii) Step Decay: The learning rate decreases by a constant factor after a fixed number of iterations. (iii) Exponential decay: $\eta_t = \eta_0\gamma^t$, where γ is the decay rate. (iv) Polynomial Decay: $\eta_t = \eta_0(1 - t/T)^p$, where t is iteration number, T is total number of iterations and p is the power parameter. (v) Cosine Annealing: $\eta_t = \frac{\eta_0}{2}(\cos(\pi t/T) + 1)$, where t is iteration number and T is total number of iterations.

function (specifically, Lipschitz continuous gradient) if η is small enough (See [42] for the further details). However, choosing an appropriate fixed learning rate can be challenging. If the learning rate is too large, it may lead to oscillations or overshooting, failing to converge. On the other hand, if the learning rate is too small, convergence to the optimal solution may be slow, requiring a large number of iterations.

Decaying Learning Rate. To overcome the limitations of a fixed learning rate, a prevalent approach in deep learning is to start with a high learning rate and progressively decay it over time or after a certain number of iterations. This allows the algorithm to take larger steps at the beginning and smaller steps as it gets closer to the minimum [38]. Some common learning rate schedules are plotted in Figure 3.5.

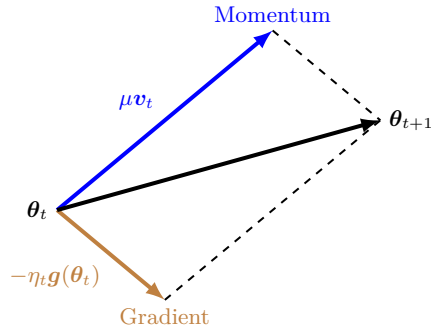


Figure 3.6 Illustration of momentum. Here, \mathbf{g} denotes the gradient ∇J .

Momentum

Gradient descent may converge slowly, especially in landscapes with high curvature, noisy gradients, or saddle points. It may oscillate excessively in steep, narrow ravines of the cost function—regions where the gradient significantly changes direction, thereby slowing progress towards the minimum. The momentum method [43] helps mitigate these issues by adding a fraction of the update vector of the past time step to the current update vector. The update rule for the momentum method is as follows:

$$\begin{aligned}\mathbf{v}_{t+1} &= \mu\mathbf{v}_t - \eta_t\nabla J(\boldsymbol{\theta}_t), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \mathbf{v}_{t+1}.\end{aligned}\tag{3.20}$$

Here, \mathbf{v}_{t+1} is the velocity at iteration $t+1$, which is a combination of the current gradient $\nabla J(\boldsymbol{\theta}_t)$ and the previous velocity \mathbf{v}_t , as illustrated in Figure 3.6. This tends to increase for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change direction, thereby dampening oscillations and accelerate convergence. $\mu \in [0, 1]$ is the momentum coefficient which decides how much of the past gradients are to be stored. When it is set to a high value, the updates will become more dependent on the past gradients.

3.2.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a simple yet efficient optimization algorithm for learning the parameters of a machine learning model. The primary advantage of SGD is its efficiency, which is particularly beneficial when dealing with large datasets [44].

Many model fitting problems in machine learning, such as risk minimization, involve stochastic optimization. The objective here is to minimize the expected value of the cost over the distribution of examples [39]:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}}[J(\boldsymbol{\theta}, \mathbf{z})], \quad (3.21)$$

where \mathbf{z} represents a random example from the data with the distribution $p(\mathbf{z})$, and $J(\boldsymbol{\theta}, \mathbf{z})$ is the cost given the model parameters $\boldsymbol{\theta}$ and example \mathbf{z} . In the case of risk as defined in equation (3.3), we have $\mathbf{z} = (\mathbf{x}, \mathbf{y})$ and $J(\boldsymbol{\theta}, \mathbf{z}) = \ell(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))$. At each iteration t in SGD, we assume we observe $J(\boldsymbol{\theta}, \mathbf{z}_t)$, where $\mathbf{z}_t \sim p$ is a randomly drawn example. We also assume a way to compute an unbiased estimate of the gradient of J . If the distribution $p(\mathbf{z})$ is independent of the parameters we are optimizing, we can use $\mathbf{g}_t = \nabla J(\boldsymbol{\theta}_t, \mathbf{z}_t)$. With this, the SGD update rule can be written as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \mathbf{g}_t. \quad (3.22)$$

A related variant of gradient descent, called mini-batch gradient descent (MBGD), makes use of a set of randomly chosen examples, known as *mini-batch*, at each iteration instead of just one [38]. The gradient in MBGD is approximated as the average gradient over a mini-batch $\mathcal{B}_t \subseteq \{1, \dots, N\}$:

$$\mathbf{g}_t \approx \frac{1}{|\mathcal{B}_t|} \sum_{m \in \mathcal{B}_t} \nabla J(\boldsymbol{\theta}_t, \mathbf{z}_m), \quad (3.23)$$

where $|\mathcal{B}_t|$ denotes the mini-batch size, typically much smaller than the total number of examples N . Mini-batches help strike a balance between the computational efficiency of SGD and the robustness of gradient descent.

When using SGD and MBGD, it is important to carefully select the learning rate to ensure convergence. One heuristic for choosing an effective learning rate, proposed by Smith [45], is to start with a small learning rate and gradually increase it while evaluating performance using a small number of mini-batches. As discussed earlier, instead of choosing a single constant learning rate, a learning rate schedule can be used to adjust the step size over time. Theoretically, SGD and MBGD are guaranteed to converge if the learning rate schedule satisfies the Robbins-Monro conditions [39]:

$$\sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty \quad (3.24)$$

This ensures that the learning rates are large enough to maintain progress, but also small enough to avoid divergence.

AdaGrad

AdaGrad (adaptive gradient), proposed by Duchi *et al.* [46], is a variant of SGD with per-parameter learning rate. It adapts the learning rate to the parameters, performing larger updates for sparser parameters and smaller updates for denser ones. The update rule in AdaGrad looks as follows:

$$\begin{aligned}\mathbf{s}_{t+1} &= \mathbf{s}_t + \mathbf{g}_t^2, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\eta_t}{\sqrt{\mathbf{s}_{t+1}} + \epsilon} \odot \mathbf{g}_t,\end{aligned}\tag{3.25}$$

where \mathbf{s}_{t+1} is a variable that accumulates the past squared gradients, i.e., $\mathbf{s}_{t+1} = \sum_{\tau=0}^t \mathbf{g}_\tau^2$, and $\epsilon > 0$ is a small smoothing term to avoid division by zero. Here, all operations are applied element-wise; that is, $\frac{1}{\sqrt{\mathbf{u}}}$ has entries $\frac{1}{\sqrt{u_i}}$, \mathbf{u}^2 has entries u_i^2 , and $\mathbf{u} \odot \mathbf{v}$ has entries $u_i v_j$. While AdaGrad is effective in settings where data is sparse and sparse parameters are more informative, its learning rate is monotonically decreasing which might lead to premature and excessive decrease of the learning rate, thereby making the algorithm converge too early.

RMSProp

RMSProp (root mean square propagation) is an adaptive learning rate method proposed by Tieleman, Hinton, *et al.* [47]. It was developed to address the diminishing learning rates of AdaGrad for long-running tasks.

RMSProp is similar to AdaGrad, but changes the gradient accumulation into an exponentially weighted moving average. RMSProp uses a decay factor to discard history from the extreme past, which allows rapid converge after finding a convex bowl, behaving as if it were an instance of the AdaGrad algorithm initialized within that bowl. Here is the update rule:

$$\begin{aligned}\mathbf{s}_{t+1} &= \beta \mathbf{s}_t + (1 - \beta) \mathbf{g}_t^2, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\eta_t}{\sqrt{\mathbf{s}_{t+1}} + \epsilon} \odot \mathbf{g}_t,\end{aligned}\tag{3.26}$$

where \mathbf{s}_{t+1} is the running weighted average of the squared gradient, decayed by a factor $\beta \in [0, 1)$ often set to 0.99. Expanding the definition of \mathbf{s}_{t+1} yields:

$$\mathbf{s}_{t+1} = (1 - \beta) \sum_{\tau=0}^t \beta^\tau \mathbf{g}_{t-\tau}^2.\tag{3.27}$$

The weights $\{(1 - \beta)\beta^\tau\}_{\tau=0}^t$ make up a geometric sequence, and hence, their sum is computed as $(1 - \beta) \sum_{\tau=0}^t \beta^\tau = 1 - \beta^{t+1}$, which approaches 1 as t approaches infinity. As such, after enough iterations, the sum of the weights becomes normalized to 1.

Adam

Adam (adaptive moment estimation), proposed by Kingma and Ba [48], combines the ideas of RMSProp and momentum. Adam computes adaptive learning rates for different parameters. In addition to storing an exponentially decaying average of past squared gradients like RMSProp, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. Adam's update rule looks like this:

$$\begin{aligned} \mathbf{m}_{t+1} &= \beta_1 \mathbf{m}_t + (1 - \beta_1) \mathbf{g}_t, \\ \mathbf{s}_{t+1} &= \beta_2 \mathbf{s}_t + (1 - \beta_2) \mathbf{g}_t^2, \\ \hat{\mathbf{m}}_{t+1} &= \frac{\mathbf{m}_{t+1}}{1 - \beta_1^t}, \\ \hat{\mathbf{s}}_{t+1} &= \frac{\mathbf{s}_{t+1}}{1 - \beta_2^t}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \frac{\eta_t}{\sqrt{\hat{\mathbf{s}}_{t+1} + \epsilon}} \odot \hat{\mathbf{m}}_t. \end{aligned} \tag{3.28}$$

Here, \mathbf{m}_t and \mathbf{s}_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, and $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{s}}_t$ are bias-corrected versions of these estimates. $\beta_1 \in [0, 1)$ (e.g., 0.9) and $\beta_2 \in [0, 1)$ (e.g., 0.999) are the forgetting factors for the first moment and second moments of gradients, respectively.

3.2.3 Block Coordinate Descent

Block Coordinate Descent (BCD) [49] is a powerful optimization technique that is especially useful when dealing with high-dimensional problems. Instead of simultaneously updating all parameters, as is the case in methods like gradient descent, BCD operates by dividing the parameters into different blocks and then optimizing over one block at a time while keeping the others fixed. This is done in a cyclic or randomized manner until a convergence criterion is met. In many cases, BCD can lead to simpler subproblems and can thus be more efficient than optimizing over all parameters at once.

Algorithm 1: Block coordinate descent algorithm.

Input: Cost function $J(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_B)$

Output: Parameters $(\boldsymbol{\theta}_1^*, \dots, \boldsymbol{\theta}_B^*)$ that minimizes J

```

1 Initialize parameters  $(\boldsymbol{\theta}_1^0, \dots, \boldsymbol{\theta}_B^0)$ 
2  $t = 0$ 
3 while not converged do
4   for  $b = 1, \dots, B$  do
5      $\boldsymbol{\theta}_b^{t+1} \leftarrow \arg \min_{\boldsymbol{\theta}_b} J(\boldsymbol{\theta}_1^{t+1}, \dots, \boldsymbol{\theta}_{b-1}^{t+1}, \boldsymbol{\theta}_b, \boldsymbol{\theta}_{b+1}^t, \dots, \boldsymbol{\theta}_B^t)$ 
6   end
7    $t \leftarrow t + 1$ 
8 end

```

Consider a cost function $J(\boldsymbol{\theta})$ to be minimized, where the parameters $\boldsymbol{\theta}$ are partitioned into B blocks of coordinates, i.e., $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_B)$. BCD cycles through these blocks one by one and minimizes J over one block of parameters, say $\boldsymbol{\theta}_b$, while fixing the other blocks. That is, starting with initial parameters values $(\boldsymbol{\theta}_1^0, \dots, \boldsymbol{\theta}_B^0)$, at each iteration t , the parameters are updated by iteratively solving the single-block optimization subproblems

$$\boldsymbol{\theta}_b^{t+1} = \arg \min_{\boldsymbol{\theta}_b} J(\boldsymbol{\theta}_1^{t+1}, \dots, \boldsymbol{\theta}_{b-1}^{t+1}, \boldsymbol{\theta}_b, \boldsymbol{\theta}_{b+1}^t, \dots, \boldsymbol{\theta}_B^t) \quad (3.29)$$

for each parameter block $\boldsymbol{\theta}_b$ of $\boldsymbol{\theta}$, for b from 1 to B . The pseudocode of BCD is outlined in Algorithm 1.

BCD is most effective in certain problem settings where optimizing over one block of parameters at a time is much easier than jointly optimizing over all the parameters [49]. This often occurs if the cost function is separable over the different coordinate blocks, or if there is a structure in the problem that allows for efficient optimization over individual blocks. Such scenarios typically arise in machine learning problems, including LASSO, centroid-based clustering, and matrix factorization, to name a few.

BCD guarantees a decrease in the cost function at each update, thereby constantly improving the parameters. There may be cases where finding a global minimum for a subproblem may not be possible through a closed-form solution or a straightforward method. Nevertheless, if we can find a “good minimum” or merely improve upon the current parameter block, we can still ensure that the cost function decreases monotonically. In practice, this approach also often converges to a satisfactory solution for the original problem.

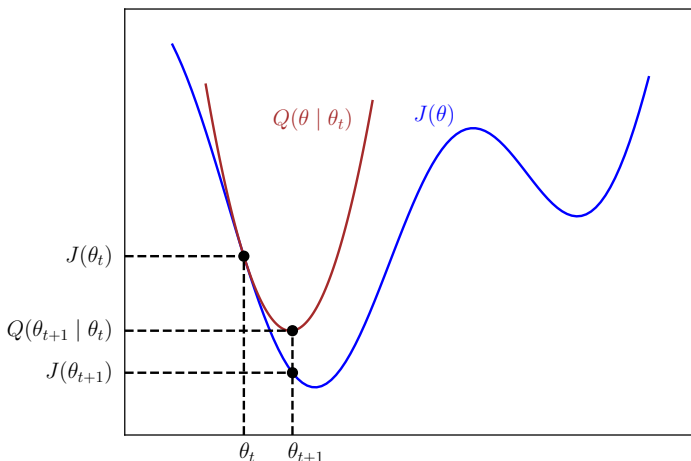


Figure 3.7 Illustration of the MM algorithm.

3.2.4 Majorization-Minimization Algorithm

Majorization-minimization (MM) algorithms [50] are a general class of optimization algorithms that operate by iteratively solving simpler subproblems. The main idea of the MM algorithm is to find optimal parameters of a non-convex, difficult-to-minimize cost function by constructing and minimizing a sequence of simpler *surrogate functions* that *majorize* (i.e., provide an upper bound) the original one.

Given a cost function $J(\boldsymbol{\theta})$, the MM algorithm iterates the following two steps until convergence [50]:

1. **Majorization Step.** Given the current parameters estimate $\boldsymbol{\theta}_t$ at iteration t , find a surrogate function $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$ that satisfies two properties:

- *Domination condition:*

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) \geq J(\boldsymbol{\theta}), \text{ for all } \boldsymbol{\theta} \quad (3.30)$$

- *Tangency condition:*

$$Q(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t) = J(\boldsymbol{\theta}_t) \quad (3.31)$$

Together, these two properties ensure that the surrogate function $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$ forms an upper bound to the original cost function $J(\boldsymbol{\theta})$, but touches it at the current parameters estimate $\boldsymbol{\theta}_t$, as illustrated in Figure 3.7.

2. **Minimization Step.** Update the parameters by minimizing the surrogate function. That is,

$$\boldsymbol{\theta}_{t+1} = \arg \min_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t). \quad (3.32)$$

Theorem 3.1. *The sequence $\{J(\boldsymbol{\theta}_t)\}$ generated by the MM algorithm is non-increasing. That is, $J(\boldsymbol{\theta}_{t+1}) \leq J(\boldsymbol{\theta}_t)$ for all t .*

Proof From the domination property of the surrogate function, we have $J(\boldsymbol{\theta}_{t+1}) \leq Q(\boldsymbol{\theta}_{t+1} \mid \boldsymbol{\theta}_t)$. Since $\boldsymbol{\theta}_{t+1}$ minimizes $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$, we have $Q(\boldsymbol{\theta}_{t+1} \mid \boldsymbol{\theta}_t) \leq Q(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t)$, and from the tangency property, we have $Q(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t) = J(\boldsymbol{\theta}_t)$. Therefore, by transitivity, $J(\boldsymbol{\theta}_{t+1}) \leq J(\boldsymbol{\theta}_t)$, which proves the theorem. ■

3.3 Learning Algorithms

3.3.1 Linear Regression

Linear regression is a simple yet widely-used model in statistics and machine learning. The key property of this model is that the expected value of the output is a linear function of the input, making it easy to interpret and fit data.

Let's denote our training set as $\mathbb{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where each input \mathbf{x}_n is a D -dimensional real vector, and $y_n \in \mathbb{R}$ is the corresponding real-valued output. Linear regression assumes the model of the following form:

$$y_n \mid \mathbf{x}_n, \mathbf{w} \stackrel{\text{iid}}{\sim} \mathcal{N}(b + \mathbf{w}^\top \mathbf{x}_n, \sigma^2), \quad (3.33)$$

where $\boldsymbol{\theta} = (b, \mathbf{w}, \sigma^2)$ are all parameters of the model, the vector of parameters \mathbf{w} is known as *weights* or *regression coefficients*, and b is the *bias* or *intercept* term. In simpler terms, the output is a linear function of the inputs plus some Gaussian noise. By convention, we extend a feature vector \mathbf{x} to include a constant 1, i.e., $(1, x_1, \dots, x_D)$, to absorb the bias term w_0 into the weight vector \mathbf{w} , simplifying the expression $b + \mathbf{w}^\top \mathbf{x}_n$ to $\mathbf{w}^\top \mathbf{x}_n$.

We can use the MLE approach to fit a linear regression model to data, maximizing the log-likelihood on the training set. The objective function is given by

$$\ell(\mathbf{w}, \sigma^2) = -\frac{1}{2\sigma^2} \underbrace{\sum_{n=1}^N N(y_n - \mathbf{w}^\top \mathbf{x}_n)^2}_{\text{residual sum of squares}} - \frac{N}{2} \ln(2\pi\sigma^2). \quad (3.34)$$

We see that this is equivalent to minimizing the residual sum of squares (RSS), which can be expressed in matrix notation as

$$\text{RSS}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2, \quad (3.35)$$

where $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$ is the data matrix with each row an input vector, and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all outputs. Setting the gradient of $\text{RSS}(\mathbf{w})$ to zero, we get the system $\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0$, which are known as the *normal equations* since $(\mathbf{y} - \mathbf{X}\mathbf{w})$ is normal (or orthogonal) to the range of \mathbf{x} at the optimal solution. If the Gram matrix $\mathbf{X}^\top\mathbf{X}$ is invertible (or equivalently \mathbf{X} has full column rank), then the unique solution to this system is given by ordinary least squares (OLS) estimator:

$$\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}, \quad (3.36)$$

where $\mathbf{X}^\dagger = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ is the Moore–Penrose pseudoinverse of the (non-square) matrix \mathbf{X} .

However, the OLS solution can suffer from overfitting if the features are highly correlated or the data is high-dimensional ($D \gg N$). To address this, we can use *ridge regression*, which adds a regularization term $\lambda\|\mathbf{w}\|^2$ to the cost function to shrink the coefficients:

$$J_{\text{ridge}}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2, \quad (3.37)$$

where $\lambda > 0$ is the regularization hyperparameter (in practice, we do not penalize the bias term w_0 , since that only affects the global mean of the output and does not contribute to overfitting).

3.3.2 Logistic Regression

Logistic regression is a widely used discriminative classification model $p(y | \mathbf{x}, \boldsymbol{\theta})$, where $\mathbf{x} \in \mathbb{R}^D$ is a fixed-dimensional input vector, $y \in \{1, \dots, C\}$ is the class label, and $\boldsymbol{\theta}$ are the parameters. If $C = 2$, this is known as *binary logistic regression*, and if $C > 2$, it is known as *multiclass logistic regression*. We overview these models in the following.

Binary Logistic Regression

Binary logistic regression is a statistical model used for binary classification tasks. It predicts the probability that a given input belongs to a specific class (denoted as 1), as opposed to the alternative class (denoted as 0).

Given a training set $\mathbb{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^D$ represents an input, and $y_n \in \{0, 1\}$ is its corresponding output, binary logistic regression assumes that the class label y_n given an input vector \mathbf{x}_n follows a Bernoulli distribution

$$y_n \mid \mathbf{x}_n, \mathbf{w} \stackrel{\text{iid}}{\sim} \text{Ber}(\sigma(\mathbf{w}^\top \mathbf{x}_n)), \quad (3.38)$$

where the weights $\mathbf{w} \in \mathbb{R}^D$ are learnable parameters, and $\sigma(\cdot)$ is the sigmoid function, defined as

$$\sigma(z) \triangleq \frac{1}{1 + \exp(-z)}. \quad (3.39)$$

This sigmoid function transforms the weighted sum of the inputs $\mathbf{a} = \mathbf{w}^\top \mathbf{x}$, known as the *logit*, into the $(0, 1)$ interval, which can be interpreted as a probability.

By utilizing MLE, we can estimate the weight vector \mathbf{w} by maximizing the log-likelihood of the data, given by

$$\begin{aligned} \ell(\mathbf{w}) &= \sum_{n=1}^N \ln(\pi_n^{y_n} (1 - \pi_n)^{(1-y_n)}) \\ &= \sum_{n=1}^N [y_n \ln(\pi_n) + (1 - y_n) \ln(1 - \pi_n)] \\ &= - \sum_{n=1}^N H_{\text{bce}}(y_n, \pi_n) \end{aligned} \quad (3.40)$$

where $\pi_n = \sigma(\mathbf{w}^\top \mathbf{x}_n)$, and $H_{\text{bce}}(y_n, \pi_n)$ is the *binary cross entropy* loss. This shows that maximizing this log-likelihood is equivalent to minimizing the average of the binary cross-entropy loss over the training set.

Multiclass Logistic Regression

Multiclass logistic regression is an extension of the binary logistic regression model to multiclass classification problems.

Definition 3.5 (Categorical distribution). Let $\boldsymbol{\pi} = (\pi_1, \dots, \pi_C)$ be the probabilities associated with C classes, i.e., $\sum_{c=1}^C \pi_c = 1$ and $\pi_c \geq 0$ for all $c = 1, \dots, C$. Then, the Categorical distribution is a discrete probability distribution over a finite set of classes, $\{1, \dots, C\}$ with the probability mass function defined as

$$\text{Cat}(y \mid \boldsymbol{\pi}) \triangleq \prod_{c=1}^C \pi_c^{\mathbf{1}(y=c)}, \quad y \in \{1, 2, \dots, C\}. \quad (3.41)$$

The Categorical distribution generalizes Bernoulli to more than two possible outcomes ($C > 2$).

The multiclass logistic regression model assumes that the conditional probability of a class label $y_n \in \{1, \dots, C\}$ given an input vector $\mathbf{x}_n \in \mathbb{R}^D$ follows a categorical distribution

$$y_n \mid \mathbf{x}_n, \mathbf{w} \stackrel{\text{iid}}{\sim} \text{Cat}(\text{softmax}(\mathbf{W}^\top \mathbf{x}_n)), \quad (3.42)$$

where the weight matrix $\mathbf{W} \in \mathbb{R}^{D \times C}$ is the parameter to be learned. The softmax function is an extension of the sigmoid function that normalizes a C -dimensional vector of *logits* into a probability distribution over C classes. Formally, it is defined as

$$\text{softmax}(\mathbf{z})_c = \frac{\exp(z_c)}{\sum_{c=1}^C \exp(z_c)}, \quad \text{for all } c = 1, \dots, C. \quad (3.43)$$

The weight matrix \mathbf{W} in multiclass logistic regression is an extension of the weight vector in binary logistic regression. Instead of learning a single weight vector, we learn a different weight vector for each class. We can estimate the weight matrix \mathbf{W} by maximizing the log-likelihood function, given by

$$\begin{aligned} \ell(\mathbf{W}) &= \sum_{n=1}^N \ln \left(\prod_{c=1}^C \pi_{nc}^{\mathbf{1}(y_n=c)} \right) \\ &= \sum_{n=1}^N \sum_{c=1}^C \mathbf{1}(y_n = c) \ln(\pi_{nc}) \\ &= - \sum_{n=1}^N H_{\text{mce}}(y_n, \boldsymbol{\pi}_n) \end{aligned} \quad (3.44)$$

where $\pi_{nc} = \text{softmax}(\mathbf{W}^\top \mathbf{x}_n)_c$, and $H_{\text{mce}}(y_n, \boldsymbol{\pi}_n)$ is the *multi-class cross entropy* loss. This shows that maximizing this log-likelihood function is

equivalent to minimizing the average of the multiclass cross-entropy loss over the training set. In contrast to linear regression, there is no closed-form solution for the MLE of binary or multiclass logistic regression. However, it can be shown that the cost function (which is negative log-likelihood) is strictly convex, and therefore, the global minimum can be found using optimization techniques such as (stochastic) gradient descent.

3.3.3 Kernel Density Estimation

Kernel density estimation (KDE) [51], [52], aka Parzen–Rosenblatt window, is a non-parametric method for estimating the probability density function of a given dataset. It basically creates a smooth surface over the histogram representation of the data by placing a kernel (essentially a smoothing function) on each data point.

Given a training set $\mathbb{D} = \{\mathbf{x}_n\}_{n=1}^N$, where each data point $\mathbf{x}_n \in \mathbb{R}^D$ is drawn iid from an unknown true distribution $p(\cdot)$, the kernel density estimator $p_{\text{kde}}(\mathbf{x})$ is defined as

$$p_{\text{kde}}(\mathbf{x}) \triangleq \frac{1}{N} \sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n), \quad (3.45)$$

where $K_h(\mathbf{x}) \triangleq \frac{1}{h^D} K(\frac{\mathbf{x}}{h})$ is the scaled version of the *kernel* K , and $h > 0$ is a hyperparameter called the *bandwidth*, which controls the degree of smoothness of our density estimator. The kernel $K(\cdot)$ can be any non-negative function that integrates to one, i.e., $K(\mathbf{x}) \geq 0$ and $\int K(\mathbf{x})d\mathbf{x} = 1$. An example of such a kernel is the Gaussian kernel, defined as

$$\phi(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right). \quad (3.46)$$

Although Gaussian kernels are widely used, they have unbounded support. Some alternative univariate kernels which have compact support (which can be computationally faster), are listed in Table 3.2. See Figure 3.8 for a plot of these kernel functions. In a general form of KDE, a scaled multivariate kernel can be constructed from a univariate kernel K by a product of the same univariate kernels scaled (possibly) with a different bandwidth for each dimension, that is

$$K_{\mathbf{h}}(\mathbf{x}) = \frac{1}{h_1 \cdots h_D} \prod_{d=1}^D K\left(\frac{x_d}{h_d}\right), \quad (3.47)$$

where h_d represents the bandwidth for the d th dimension.

Table 3.2 List of some popular normalized kernels in 1D. Compact=1 means the function is non-zero for a finite range of inputs. Smooth=1 means the function is differentiable over the range of its support. Boundaries=1 means the function is also differentiable at the boundaries of its support.

Kernel	Definition	Compact	Smooth	Boundaries
Gaussian	$(2\pi)^{-\frac{1}{2}} e^{-\frac{x^2}{2}}$	0	1	1
Boxcar	$\frac{1}{2} \mathbf{1}(x \leq 1)$	1	0	0
Epanechnikov	$\frac{3}{4} (1 - x^2) \mathbf{1}(x \leq 1)$	1	1	0
Tri-cube	$\frac{70}{81} (1 - x ^3)^3 \mathbf{1}(x \leq 1)$	1	1	1

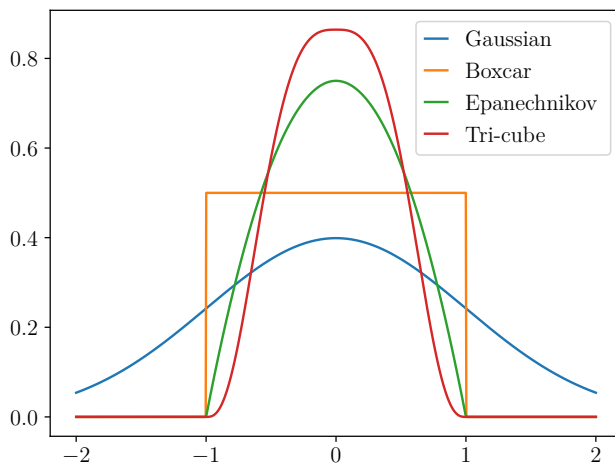


Figure 3.8 A comparison of some popular kernels.

Definition 3.6 (Empirical distribution). Consider a set of N data points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, drawn iid from a distribution $P(\cdot)$ over the domain \mathbb{X} . The empirical distribution of this dataset is defined with the probability measure

$$P_{emp}(\mathbb{A}) \triangleq \frac{1}{N} \sum_{n=1}^N \mathbf{1}(\mathbf{x}_n \in \mathbb{A}), \quad (3.48)$$

where \mathbb{A} represents any subset of \mathbb{X} , and $\mathbf{1}(\cdot)$ denotes the indicator function. The probability density function corresponding to the empirical distribution can be expressed as

$$p_{emp}(\mathbf{x}) \triangleq \frac{1}{N} \sum_{n=1}^N \delta(\mathbf{x} - \mathbf{x}_n), \quad (3.49)$$

where $\delta(\mathbf{x})$ denotes the Dirac delta function. In essence, the empirical distribution assigns an equal probability mass to each observed data point.

The KDE can also be viewed as the convolution of the scaled kernel and the empirical probability density function, defined by equation (3.49):

$$p_{\text{kde}}(\mathbf{x}) = K_h(\mathbf{x}) * p_{\text{emp}}(\mathbf{x}). \quad (3.50)$$

This shows that KDE is, in fact, the filtered and smoothed version of the empirical distribution, with the bandwidth h controlling the bandwidth of this (low-pass) filter defined in the frequency domain.

Bandwidth Selection. As illustrated in Figure 3.9, the bandwidth h regulates the smoothness of the KDE and has a large impact on the quality of KDE. A large h will result in a low-variance but high-bias estimate (underfitting). In this case, the KDE will be too smooth and may fail to capture important features of the data. A small h will lead to a high-variance but low-bias estimate (overfitting). The KDE will follow the data very closely, which may capture the noise in the data. There are two common approaches for bandwidth selection:

1. *Plug-in Selectors:* These methods compute the bandwidth directly from the data. A widely used plug-in method is Silverman’s rule of thumb [53], which provides a good initial guess for the bandwidth. For a univariate, normally distributed dataset, this gives $h = (4\sigma^5/3N)^{1/5} \approx 1.06\sigma N^{-1/5}$, where σ is the standard deviation of the samples. This is sometimes empirically modified by replacing the standard deviation σ with the parameter $A = \min(\sigma, \frac{IQR}{1.34})$, where IQR is the interquartile range, and by reducing the factor from 1.06 to 0.9. Then, the final formula would be

$$h = 0.9 \min\left(\sigma, \frac{IQR}{1.34}\right) N^{-\frac{1}{5}}.$$

Note that, this rule assumes that the true underlying distribution of the data is Gaussian, and it might underperform for distributions that are significantly non-Gaussian.

2. *Cross-validation Selectors:* These selectors determine the bandwidth by optimizing the predictive performance of the KDE estimated by some sort of cross-validation. For instance, one common method is leave-one-out cross-validation, where we remove one data point, fit the KDE, and then evaluate the likelihood of the removed point [54], [55]. This process is repeated for each data point and the bandwidth that maximizes the average likelihood is selected.

Drawbacks. KDE suffers from the “curse of dimensionality,” performing poorly when the dimensionality of the data is high (typically greater than 5). As the dimension increases, the data becomes sparse and the amount of data KDE

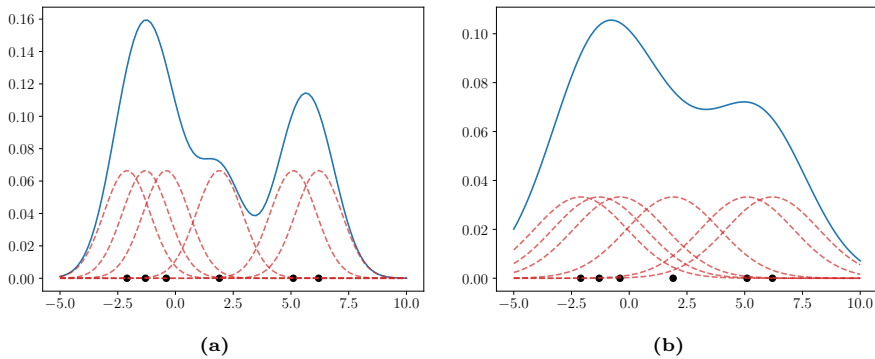


Figure 3.9 An example of kernel density estimate from 6 1D data points. We used a Gaussian kernel with bandwidth of $h = 1$ (a) and $h = 2$ (b).

requires for a reliable density estimate more data grows exponentially with the dimensionality of the data. Moreover, KDE can be computationally intensive for large datasets because it requires computing the distance from each point to every other point.

3.3.4 Kernel Regression

Kernel regression [56], [57] is a non-parametric technique to estimate the conditional expectation of the output $y \in \mathbb{R}$ given the input $\mathbf{x} \in \mathbb{R}^D$. The objective is to find a relationship between the output y and the input \mathbf{x} by estimating the regression function

$$f(\mathbf{x}) = \mathbb{E}[y | \mathbf{x}] = \int yp(y | \mathbf{x})dy = \frac{\int yp(\mathbf{x}, y)dy}{\int p(\mathbf{x}, y)dy} \quad (3.51)$$

It is especially useful in cases where this relationship is not assumed to be linear, and when there is no specific model for this relationship.

We can use KDE to estimate the joint density $p(\mathbf{x}, y)$ as follows:

$$\hat{p}(\mathbf{x}, y) = \frac{1}{N} \sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n)K_h(y - y_n) \quad (3.52)$$

If we replace the joint densities in equation (3.51) with this estimate and move the sums outside the integrals, we get

$$\hat{f}(\mathbf{x}) = \frac{\frac{1}{N} \sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n) \int yK_h(y - y_n)dy}{\frac{1}{N} \sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n) \int K_h(y - y_n)dy} \quad (3.53)$$

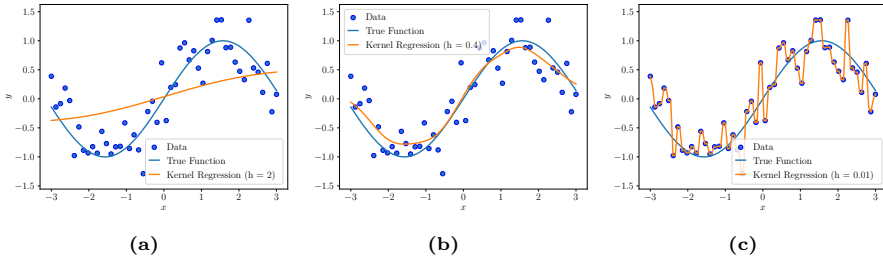


Figure 3.10 Nadaraya-Watson kernel regression. The data are noisy samples of the true function $y = \sin(x)$. From left to right: (a) Kernel regression with a large bandwidth of 1, which leads to underfitting. (b) Kernel regression with a proper bandwidth of 0.4, which gives a good fit. (c) Kernel regression with a small bandwidth of 0.01, which leads to overfitting.

We can simplify the numerator using the fact that $\int yK_h(y) = 0dy$, which in turn implies $\int yK_h(y - y_n)dy = 1$. We can simplify the denominator using the fact that kernels integrate to one, i.e., $\int K_h(y - y_n)dy = 1$. Hence, we have

$$\hat{f}(\mathbf{x}) = \frac{\sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n)y_n}{\sum_{n=1}^N K_h(\mathbf{x} - \mathbf{x}_n)} \quad (3.54)$$

or equivalently

$$\hat{f}(\mathbf{x}) = \sum_{n=1}^N w(\mathbf{x}, \mathbf{x}_n)y_n, \quad w(\mathbf{x}, \mathbf{x}_n) \triangleq \frac{K_h(\mathbf{x} - \mathbf{x}_n)}{\sum_{n'=1}^N K_h(\mathbf{x} - \mathbf{x}_{n'})}, \quad (3.55)$$

which is known as kernel regression or *Nadaraya-Watson estimator* [56], [57]. This can be understood as a weighted average of y_n s, where the weights depend on the distance between \mathbf{x} and the \mathbf{x}_n 's. When \mathbf{x} is close to a given \mathbf{x}_n , the corresponding weight will be larger. In a general form of kernel regression, $K_h(\mathbf{x} - \mathbf{x}')$ can be replaced by any similarity measure $S : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$.

Just as with KDE, the choice of bandwidth is crucial in kernel regression as it controls the trade-off between bias and variance. A small bandwidth will result in low bias but high variance (overfitting), where the regression function learns noise, while a large bandwidth will result in low variance but high bias (underfitting), where the regression function is oversimplified and too smooth to learn all the important patterns. The same bandwidth selection techniques as that of KDE can be also used for kernel regression.

Figure 3.10 illustrates the impact of bandwidth on Kernel Regression with a Gaussian kernel in learning from samples of the sinusoidal function. With a

large bandwidth of 1 (Figure 3.10a), the model underfits, failing to capture the data's structure. A small bandwidth of 0.01 (Figure 3.10b) leads to overfitting, capturing noise instead of true patterns. An optimal bandwidth of 0.4 (Figure 3.10c) strikes the best balance, effectively capturing the underlying data structure.

3.4 Conclusion

In this chapter, we presented fundamental concepts of machine learning, with a particular focus on supervised learning. We delved into model fitting using empirical risk minimization and highlighted the importance of optimization methods, e.g., stochastic gradient descent and Adam, in this process. From a probabilistic perspective, we explored foundational models such as logistic regression and kernel regression, which serve as the basis for more complex models, particularly neural networks, presented in the next chapters. Notably, logistic regression format is used as the final layer in our segmentation networks, presented in Chapters 6, 7, and 8. Kernel regression offers insights into the *attention mechanism*, which will be elaborated upon in Section 7.3.1.

Chapter 4

Deep Learning for Medical Image Segmentation

Deep learning models have dominated the field of medical image segmentation, underpinning our approach in this thesis. This chapter provides an insight into the core principles of deep learning essential for medical image segmentation. Specifically, we delve into convolutional neural networks and their key components and also explore models that leverage transformers and attention mechanisms, which have recently gained prominence in tackling computer vision challenges. We conclude with a review of convolutional, attention-based, and hybrid models that have U-shaped architectures designed specifically for medical image segmentation.

4.1 From Handcrafted Features to Hierarchical Representations

Traditional machine learning involves manually designing and extracting features from raw data. This often requires domain-specific knowledge to determine what features are likely to be relevant for a specific task. These “handcrafted” features are then used by machine learning algorithms, such as logistic regression and random forests, to learn predictive models. The success of the model largely depends on the quality and relevance of the “handcrafted” features.

Representation learning, on the other hand, is an approach in machine learning that aims to automatically learn the features or representations needed for a task directly from raw data [38]. Rather than requiring human experts to design and extract features, representation learning algorithms identify and extract high-level, abstract features themselves. The idea is to let the model discover not only the mapping from representation to output but also the mapping from input to the representation itself in an end-to-end fashion. It is often used in tasks where handcrafting features would be challenging or impractical, such as image recognition or natural language processing.

Deep learning is a subfield of machine learning characterized by the use of artificial neural networks with numerous layers for representation learning [38]. These layers enable the model to automatically learn hierarchical representations of data, progressively extracting higher-level features from the raw input. For instance, in image recognition, shallower layers may identify edges, while deeper layers may identify more complex structures or patterns. These models are particularly effective at processing large volumes of complex, high-dimensional data such as images, speech, and text, and they excel in tasks like image and speech recognition, where manual feature engineering proves difficult or impractical. For such tasks, they often outperform machine learning techniques that rely on handcrafted features. In this section, we provide an overview of the fundamental concepts and important methods in deep learning that are relevant to our work on brain lesion segmentation.

4.2 Multilayer Perceptron

The most basic types of neural networks are known as multilayer perceptions (MLPs) [58], which consist of multiple layers of neurons, each fully connected with neurons in the adjacent layers. Specifically, each neuron receives input

from the neurons in the previous layer and, in turn, influences the neurons in the subsequent layer.

Before delving into a formal presentation of MLPs, let's revisit logistic regression. This model directly maps inputs to outputs by applying a single linear (affine) transformation

$$\mathbf{f}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}^T \mathbf{x} + \mathbf{b}, \quad (4.1)$$

subsequently followed by a softmax operation. If the relationship between our labels and the input data were as simple as a basic affine transformation, then this approach would be adequate. Nevertheless, the assumption of linearity in affine transformations can be overly simplistic and restrictive. Linearity implies, for example, the weaker assumption of *monotonicity*, i.e., that any increase in a feature must either always cause an increase in the model's output (if the corresponding weight is positive), or always cause a decrease in our model's output (if the corresponding weight is negative). This may seem to be natural property in some real scenarios, but it does not make sense in many cases, for example, classifying images of cats and dogs. Should increasing the intensity of the pixel at position (13, 17) always increase (or always decrease) the likelihood that the image depicts a dog? Reliance on a linear model corresponds to the implicit assumption that the only requirement for differentiating cats vs. dogs is to assess the brightness of individual pixels. This approach is doomed to fail in a world where inverting an image preserves the category.

We can overcome the limitations of linear models by incorporating one or more *hidden layers*. The easiest way to do this is to stack many fully connected layers on top of each other. Each layer feeds into its next layer until we generate outputs. We can think of the first $L - 1$ layers as our representation and the final layer as our linear predictor. Formally, an MLP with L layers to approximate a function $\mathbf{f} : \mathbb{R}^D \rightarrow \mathbb{R}^C$ can be expressed as:

$$\begin{aligned} \mathbf{a}^{(0)} &= \mathbf{x}, \\ \mathbf{a}^{(\ell)} &= \mathbf{g}^{(\ell)} \left(\mathbf{W}^{(\ell)T} \mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)} \right), \quad \ell = 1, \dots, L \\ \mathbf{o} &= \mathbf{a}^{(L)}, \end{aligned} \quad (4.2)$$

where $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{o} \in \mathbb{R}^C$ represent inputs and outputs, respectively. $\mathbf{W}^{(\ell)} \in \mathbb{R}^{D_{\ell-1} \times D_{\ell}}$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{D_{\ell}}$ are the weight and bias parameters of the ℓ th layer, respectively. D_{ℓ} is the output dimension of the ℓ th layer, with $D_0 = D$ and $D_L = C$. Here, the input layer corresponds to $\ell = 0$, the *hidden layers* to $\ell = 1, \dots, L - 1$, and the output layer to $\ell = L$. $\mathbf{g}^{(\ell)}$ is called an *activation function*, which is the key ingredient to make MLP a nonlinear model, and the outputs of the activation functions, $\mathbf{a}^{(\ell)} \in \mathbb{R}^{D_{\ell}}$, are referred to as *feature maps* or

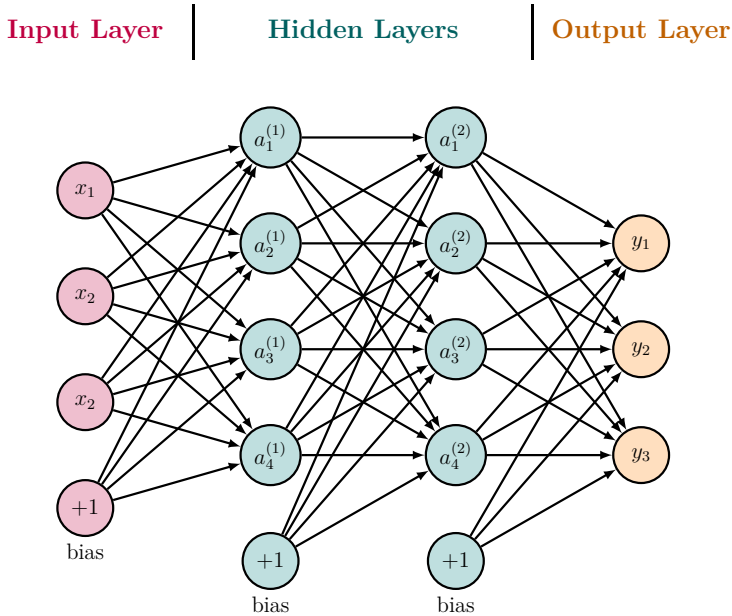


Figure 4.1 An MLP with 2 hidden layers, each has 4 units.

activation maps. The activation functions of the hidden layers were traditionally a sigmoid function or hyperbolic tangent on each node (coordinate), but today the ReLU (rectified linear unit) activation function [59], $\sigma(x) = \max(0, x)$, is a more popular choice. Table 4.1 present a list of some popular activation functions. Note that these functions operate on their arguments element-wise. The activation function of the last layer, $\mathbf{g}^{(L)}$, is often the sigmoid function for binary classification, softmax for multi-class classification, and the identity function for regression.

As an example, Figure 4.1 illustrates an MLP with 3 inputs, 2 hidden layers, and 3 outputs. Each hidden layer contains 4 neurons. Since the input layer does not involve any calculations, producing outputs with this network requires implementing the computations for both the hidden and output layers; thus, the total number of layers in this MLP is 3. Note that both layers are fully connected. Every input influences every neuron in the hidden layer, and each of these in turn influences every neuron in the output layer.

Given a dataset and a loss function, we can construct the empirical risk and minimize it using (stochastic) gradient descent. Training a neural network is not much different from training any other parametric machine learning

model with gradient descent. The largest difference between the linear models we have seen so far and neural networks is that the nonlinearity of a neural network causes most interesting loss functions to become non-convex. This means that neural networks are usually trained by using iterative, gradient-based optimizers that merely drive the cost function to a very low value, rather than the linear equation solvers used to train linear regression models or the convex optimization algorithms with global convergence guarantees used to train logistic regression [38]. Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters. For MLPs, it is important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values.

To be able to apply any gradient-based optimization algorithm for training a neural network, we first need the gradient of the loss function with respect to the parameters. This can be done by the famous *backpropagation* algorithm [58], which is a special case of automatic differentiation. Modern deep learning frameworks such as PyTorch are equipped with automatic differentiation and are able to efficiently calculate the gradient of almost any loss function. This has profoundly simplified the implementation of deep learning algorithms as we only need to implement the forward pass and leave the backward pass to be taken care of by the framework. In the past, before automatic differentiation, even small changes to complicated models required recalculating complicated derivatives by hand.

4.3 Convolutional Neural Networks

The development of modern convolutional neural networks (CNNs) is primarily attributed to the seminal work of Yann LeCun and his collaborators in the late 1980s and early 1990s [64], [65]. Their work was originally designed for handwriting and character recognition tasks, resulting in the creation of LeNet [65], one of the earliest CNN architectures. LeNet, while relatively simple by today's standards, laid the groundwork for future advancements in the field, introducing key elements of CNNs, including convolutional layers and subsampling (now more commonly referred to as pooling).

A significant leap in the advancement of CNNs came in 2012 with the introduction of AlexNet [66], designed to classify the 1.2 million high-resolution images in the ImageNet database into 1000 different classes. AlexNet significantly outperformed other methodologies in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that year. In fact, for much of the

intervening time between the early 1990s and the watershed results of 2012 [66], neural networks were often surpassed by other machine learning methods, such as kernel methods [67] and random forests [68]. AlexNet's deeper and wider architecture, compared to LeNet, leveraged new techniques like rectified linear units (ReLU) [59] and *dropout* [69], and also utilized graphics processing units (GPUs) for efficient training. This marked the advent of 'deep' learning and sparked a revolution in the application of CNNs for computer vision tasks.

MLPs, discussed in Section 4.2, are appropriate options when we are dealing with tabular data, they have several shortcomings in image processing tasks. Firstly, MLPs are incapable of exploiting the 2D (or 3D) spatial structure of images since they flatten the images, irrespective of the spatial relation between pixels. This lack of spatial awareness makes MLPs inefficient in learning localized features, which are crucial in most vision-related tasks. Secondly, MLPs encounter scalability issues due to their fully connected nature. This means that the number of parameters grows significantly with the increase in resolution, leading to computational inefficiencies and potential overfitting.

CNNs, on the other hand, are designed to overcome these limitations, making them particularly suitable for computer vision tasks due to the following main reasons:

- *Local Receptive Fields*: CNNs use filters or *kernels* that slide across the input image, allowing them to recognize local patterns within different regions. This mimics the way the human visual system operates and enables CNNs to capture intricate details that MLPs often miss.
- *Shared Weights*: CNNs use shared weights, where the same filter is applied across the entire image. This makes CNNs significantly reduce the number of parameters and become more computationally efficient while less prone to overfitting compared to their MLP counterparts.
- *Translational Invariance*: Shared weights together with pooling layers also make CNNs translational invariance, enabling them to detect the same pattern regardless of its position in the image, providing robustness against changes in the position of objects.
- *GPU-friendly Parallelization*: The operations within CNNs are embarrassingly parallel, enabling them to leverage the computational capabilities of GPUs effectively. This leads to a significant reduction in training and inference times.

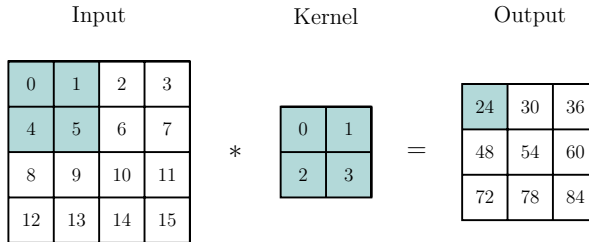


Figure 4.2 Illustration of 2D cross-correlation.

4.3.1 Convolutions for Images

Cross-Correlation. Cross-correlation is a fundamental operation in CNNs. It is worth noting that the term “cross-correlation” in the context of CNNs is often used interchangeably with “convolution,” as done in this thesis. However, strictly speaking, the operation performed in CNNs is actually cross-correlation, not convolution, as the latter involves flipping the kernel, which is not done in CNNs.

Let’s denote a 2D input image as $\mathbf{X} \in \mathbb{R}^{H \times W}$ and a *kernel* (or *filter*) as $\mathbf{V} \in \mathbb{R}^{M \times N}$, where H and W represent the height and width of the input respectively, and M and N denote the height and width of the kernel. We always assume that the tensors representing images and kernels are zero-based, where the indices for all the dimensions start at 0. The cross-correlation operation works by sliding the kernel over the input, from left to right, top to bottom, and at each position, the dot product between the kernel and the portion of the input under the kernel is computed. See Figure 4.2 for an illustration of this process. Formally, the cross-correlation output is a matrix $\mathbf{Y} = \mathbf{X} * \mathbf{V} \in \mathbb{R}^{(H-M+1) \times (W-N+1)}$, defined as

$$\begin{aligned}
 Y_{h,w} &= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{h+m,w+n} V_{m,n} \\
 &= \langle X_{h:h+M-1,w:w+N-1}, \mathbf{V} \rangle,
 \end{aligned} \tag{4.3}$$

where $Y_{h,w}$ represents the intensity value at position (h,w) in the output, and $X_{h:h+M-1,w:w+N-1}$ denotes the slice of the input covered by the window corresponding to the position (h,w) . The key characteristics of the cross-correlation operation include:

- *Linearity*: It behaves linearly with respect to both the input and the kernel, allowing for the superposition of effects.

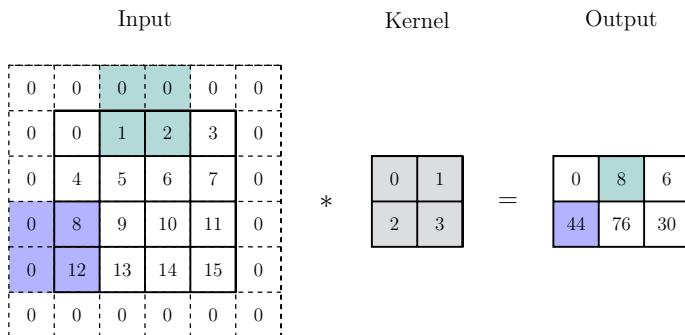


Figure 4.3 Illustration of padding and strides in 2D convolution. The input is zero-padded by 1 pixel on all sides and then convolved with a 2×2 kernel using strides of (3, 2).

- *Shift Invariance*: If the input is shifted, the output will also be shifted by the same amount, preserving the spatial relationship. This gives a model the ability to recognize patterns irrespective of their spatial position in the input.
- *Locality*: The kernel operates locally on the input, focusing on a specific region at a time, enabling the extraction of local features, such as edges, corners, or textures.
- *Parameter Sharing*: The same kernel is used across the entire image, greatly reducing the number of parameters, promoting generalization, and enabling the detection of the same feature regardless of its position.

Padding. In Figure 4.2, we see that convolving a 4×4 image with a 2×2 filter results in a 3×3 output. In general, convolving an $M \times N$ filter over an image of size (H, W) produces an output of size $(H - M + 1, W - N + 1)$, which reflects how the kernel’s size influences the output’s size, with larger kernels resulting in smaller spatial dimensions for the output. This is sometimes called *valid convolution* since we only apply the filter to “valid” parts of the input, i.e., we do not let it “slide off the ends”. If we want the output to have the same size as the input, we can use zero-padding, which means we add a border of 0s to the image, as illustrated in Figure 4.3. This is sometimes called *same convolution*. If we pad P_H rows on both sides of the height and P_W columns on both sides of the width, then the convolution output will be of size $(H + 2P_H - M + 1, W + 2P_W - N + 1)$.

Stride. Since each output pixel is generated by a weighted combination of inputs in its receptive field (based on the size of the filter), neighboring outputs

will be very similar in value, since their inputs are overlapping. We can reduce this redundancy (and speedup computation) by skipping every S th input. This is called *strided convolution*, which is illustrated in Figure 4.3, where we convolve a 4×4 image with a 2×2 filter with a zero-padding of 1 pixel on all sides and with strides of $(3, 2)$ to obtain a 2×3 output. In general, if we pad P_H rows on both sides of the height and P_W columns on both sides of the weight and use strides of size (S_H, S_W) , then the convolution output will have the size $(H_{\text{out}}, W_{\text{out}})$, computed as

$$\begin{aligned} H_{\text{out}} &= \lfloor \frac{H + 2P_H - M + S_H}{S_H} \rfloor \\ W_{\text{out}} &= \lfloor \frac{W + 2P_W - N + S_W}{S_H} \rfloor. \end{aligned} \quad (4.4)$$

Strided convolutions are commonly used in modern CNNs like ResNet [70] to downsample the feature maps at the beginning of each stage.

Convolutional Layers. The convolutional layer forms the cornerstone of CNNs, serving as the core component for feature extraction and pattern recognition in spatial data. A convolutional layer consists of a set of learnable filters, which are used to perform a cross-correlation operation on the input (recall that strictly speaking, “convolutional layers” are a misnomer since the operations they express, as mentioned earlier, are more accurately described as cross-correlations). Unlike the layers of MLPs, where each neuron connects to all neurons in the previous layer, neurons in a convolutional layer are connected only to a small region of the input volume.

In image processing, input data typically have multiple channels (e.g., RGB image has three channels: red, green, and blue), which necessitates the adaptation of the convolutional layer to handle multiple channels. Let’s denote a multi-channel 2D input as $\mathcal{X} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$, where C_{in} is the number of input channels, and H and W are the height and width of the input, respectively. The kernel must also incorporate these multiple input channels and produce multiple output channels. Therefore, we express a kernel as $\mathcal{V} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times M \times N}$ and a bias as $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$, where C_{out} is the number of output channels (or the number of filters), and M and N are the height and width of the kernel. Here, $\mathcal{V}_{c_{\text{out}}, c_{\text{in}}, :, :}$ is the filter associated with the c_{out} th output channel and c_{in} th input channel, $b_{c_{\text{out}}}$

The cross-correlation operation in this general scenario is performed on the 2D data of each input channel separately, and the results are summed up to yield the final output for each output channel, as illustrated in Figure 4.4. Each output channel has its own bias term, which is added to the output of the cross-correlation before the activation function is applied. Formally, a convolutional

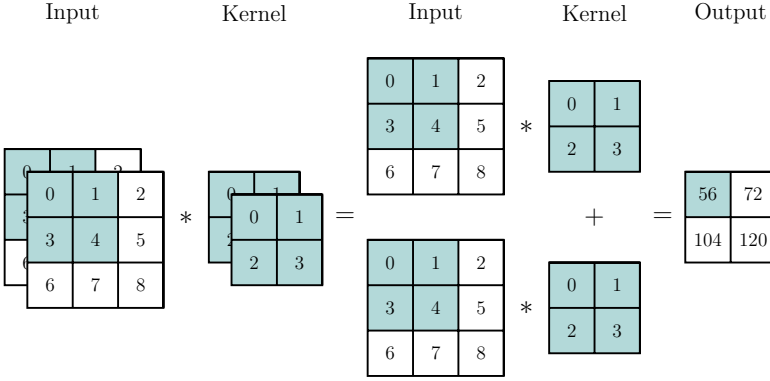


Figure 4.4 Illustration of 2D convolution applied to an input with 2 channels.

layer can be described as follows:

$$\mathcal{Y}_{c_{\text{out}},h,w} = b_{c_{\text{out}}} + \sum_{c_{\text{in}}=0}^{C_{\text{in}}-1} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathcal{X}_{c_{\text{in}},h+m,w+n} \mathcal{V}_{c_{\text{out}},c_{\text{in}},m,n} \quad (4.5)$$

where $\mathcal{Y}_{c_{\text{out}},h,w}$ represents the intensity value at position (h,w) in the c_{out} th output channel. The learnable parameters of the convolutional layer are the kernel \mathcal{V} and the bias \mathbf{b} .

Computational Complexity. We can calculate the number of floating-point operations (FLOPs) involved in a convolutional layer to understand its computational complexity. As defined by equation (4.5), for every output channel, we perform MN multiplications, and an equal number of additions, at each position (h,w) in the output data. This process is repeated for each input channel and output channel. Therefore, the total number of FLOPs for the entire convolutional layer is $\mathcal{O}(2MNH_{\text{out}}W_{\text{out}}C_{\text{in}}C_{\text{out}})$, where H_{out} and W_{out} can be computed using equation (4.4). Note that this expression represents the raw computation count and does not take into account any possible optimizations, such as those achievable through fast Fourier transform (FFT). However, it serves as a theoretical baseline for comparing the computational efficiency of different convolutional layer configurations.

Pointwise Convolution. Sometimes we just want to take a weighted combination of the features at a given position, rather than across positions. This can be done using 1×1 convolution, known as *pointwise convolution*. This can be thought of as a fully connected layer applied to each pixel in parallel. While it might seem trivial at first glance, this operation serves some important functions within a CNN. Indeed, pointwise convolutions mix the channels without

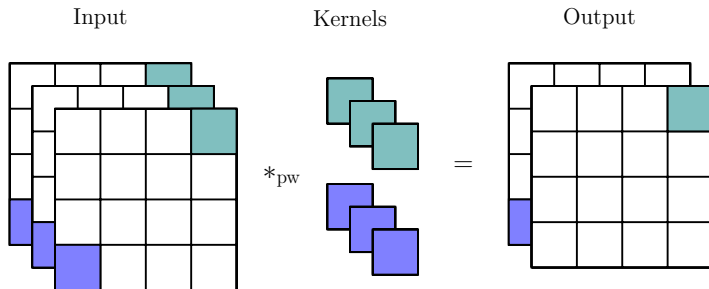


Figure 4.5 Illustration of pointwise convolution. 1×1 convolution operates on a 3-channel input and yields a 2-channel output. The input and output have the same height and width.

considering the spatial dimensions, allowing the modeling of the interactions between different channels. Pointwise convolutions can be also used to reduce the number of channels in the data, controlling the network’s complexity and computational cost. See Figure 4.5 for an illustration of the pointwise convolution on a 2D toy image.

4.3.2 Pooling Layers

Our ultimate task often involves answering a global question about an image, such as whether it contains a cat. Therefore, the units of our final layer should be sensitive to the entire input. This goal can be accomplished by gradually reducing the spatial resolution using *pooling* layers in several stages to create increasingly coarser maps, ultimately learning a global representation. This way, the deeper we go in the network, the more the receptive field for each hidden node grows relative to the input, as the convolution kernels cover a more substantial effective area. Moreover, by downsampling the feature maps, the pooling layers decrease the number of parameters and computations required in subsequent layers.

Like convolutional layers, *pooling* operators consist of a fixed-shape window that is slid over all regions in the input according to its stride, computing a single output for each position traversed by the fixed-shape window. However, unlike the convolutional layer, the pooling layer contains no parameters. Instead, *pooling* operators are deterministic, typically calculating either the maximum or the average value of the elements in the pooling window. These operations are called *max pooling* (see Figure 4.6 for an illustration) and average pooling, respectively. Max pooling, commonly used in intermediate stages, preserves the

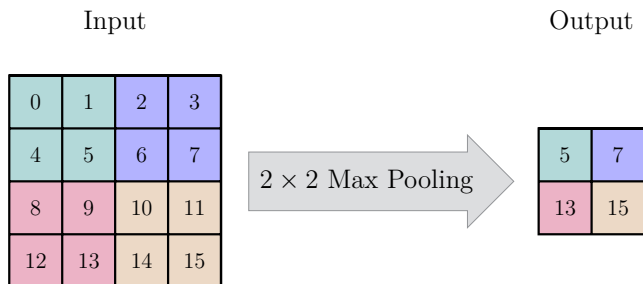


Figure 4.6 Illustration of max pooling with a 2×2 filter and a stride of $(2, 2)$.

strongest features, providing robustness to small translations.

If we average over all the locations in a feature map, the method is called *global average pooling*. This converts a $C \times H \times W$ feature map into a $C \times 1 \times 1$ feature map, which can be reshaped to a C -dimensional vector before entering a fully connected layer, where it is then passed to softmax. The use of global average pooling enables the classifier to handle images of any size, as the final feature map will always be converted to a fixed-dimensional vector before being mapped to a distribution over classes.

As an alternative to explicit pooling, strided convolutions can also be used to downsample the feature maps. By employing a stride of 2, for instance, the convolution layer will skip over one pixel at each step, effectively halving the spatial dimensions of the output. Strided convolutions also incorporate learnable parameters, which allows the network to learn the optimal downsampling method.

4.3.3 Normalization Layers

Normalization layers have emerged as a fundamental building block in deep learning, providing significant benefits in model training. These benefits are motivated by the challenges in training deep neural networks, particularly concerning convergence speed, stability, and model generalization.

The optimization landscape in deep learning can be fraught with challenges. The inputs to each layer's activation functions may vary dramatically in scale throughout the training process, leading to problems known as *internal covariate shift* [71]. This shifting can cause the gradients to vanish or explode, impeding convergence and necessitating careful initialization and smaller learning rates.

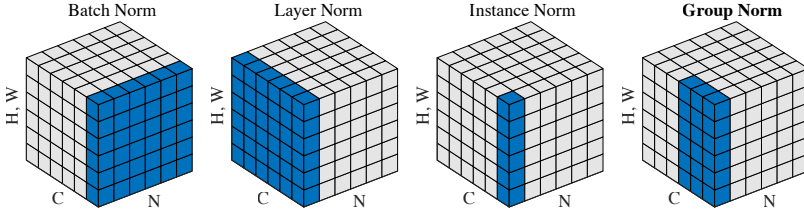


Figure 4.7 Illustration of different normalization methods for a CNN. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. Left to right: batch norm, layer norm, instance norm, and group norm (with 2 groups of 3 channels). From Figure 2 of [74].

Normalization layers are meant to address these challenges by standardizing the feature maps within a network, maintaining a mean close to zero and a standard deviation close to one. This makes the optimization landscape smoother and allows for larger learning rates, leading to faster convergence.

Batch Normalization. One of the most widely-used normalization techniques is called Batch Normalization [71]. This ensures the distribution of a feature map within a layer has zero mean and unit variance, when averaged across the samples in a minibatch. More precisely, if $\{z_n\}_{n \in \mathcal{B}}$ are the values of a feature map over the minibatch \mathcal{B} , we replace the feature map z_n with \tilde{z}_n , computed as follows:

$$\begin{aligned} \mu_{\mathcal{B}} &= \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} z_n, & \sigma_{\mathcal{B}}^2 &= \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} (z_n - \mu_{\mathcal{B}})^2, \\ \hat{z}_n &= \frac{z_n - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, & \tilde{z}_n &= \gamma \hat{z}_n + \beta, \end{aligned} \quad (4.6)$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance of the batch, \hat{z}_n is the standardized feature map, γ and β are learnable parameters to scale and shift it, and $\epsilon > 0$ is a small constant to avoid division by zero.

BN can dramatically speed up and stabilize the training, especially for deep CNNs. The exact reasons for this are still unclear, but BN seems to make the optimization landscape significantly smoother [72]. BN also acts like a regularizer that improves generalization [73].

Layer Normalization. Batch normalization often fails when the batch size is small since the estimated mean and variance parameters can be unreliable. One

solution is to compute the mean and variance by pooling statistics across other dimensions of the tensor, but not across examples in the batch. In this regard, layer normalization (LN) [75] was proposed as an alternative that normalizes the feature maps across the feature dimensions for each individual example. More precisely, if $\mathbf{z}_n \in \mathbb{R}^D$ represent a vector of flattened feature maps in some layer, then LN is defined as:

$$\begin{aligned} \mu_n &= \frac{1}{D} \sum_{d=1}^D z_{nd}, & \sigma_n^2 &= \frac{1}{D} \sum_{d=1}^D (z_{nd} - \mu_n)^2, \\ \hat{z}_{nd} &= \frac{z_{nd} - \mu_n}{\sqrt{\sigma_n^2 + \epsilon}}, & \tilde{z}_{nd} &= \gamma \hat{z}_{nd} + \beta, \end{aligned} \quad (4.7)$$

where γ and β are learnable parameters, and $\epsilon > 0$ is a small constant to avoid division by zero.

Instance Normalization. Alternatively, we can normalize across spatial dimensions for each channel in each example independently. This is known as instance normalization (IN) [76]. Let $\mathbf{Z}_n \in \mathbb{R}^{C \times S}$ represent spatially-flattened feature maps in some layer, where C is the number of channels and S is the number of pixels. IN can then be expressed as:

$$\begin{aligned} \mu_{nc} &= \frac{1}{S} \sum_{s=1}^S z_{ncs} & \sigma_{nc}^2 &= \frac{1}{S} \sum_{s=1}^S (z_{ncs} - \mu_{nc})^2, \\ \hat{z}_{ncs} &= \frac{z_{ncs} - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}}, & \tilde{z}_{ncs} &= \gamma \hat{z}_{ncs} + \beta. \end{aligned} \quad (4.8)$$

The strength of IN lies in its ability to remove example-specific contrast information, which often leads to more stylized features in image generation tasks.

Group Normalization. Group normalization (GN) [74] divides channels into smaller groups and normalizes within each group independently. This is illustrated in Figure 4.7. LN is a special case in which there is a single group, containing all the channels. IN is a special case in which there are C groups, one per channel. GN provides a good balance between LN and IN, offering consistent performance in scenarios where BN performs poorly.

4.3.4 Transposed Convolutional Layers

In (valid) convolution, we transform a large input \mathbf{X} into a small output \mathbf{Y} by taking a weighted combination of the input pixels with the convolutional kernel \mathbf{V} . This concept is most easily explained in code:

```
def conv(X, V):
    H, W = X.shape
    M, N = V.shape
    Y = torch.zeros((H - M + 1, W - N + 1))
    for h in range(H):
        for w in range(W):
            Y[h, w] = torch.sum(X[h : h + M, w : w + N] * V)

    return Y
```

In *transposed convolution*, we perform the reverse operation in order to produce a larger output from a smaller input:

```
def trans_conv(X, V):
    H, W = X.shape
    M, N = V.shape
    Y = torch.zeros((H + M - 1, W + N - 1))
    for h in range(H):
        for w in range(W):
            Y[h : h + M, w : w + N] += X[h, w] * V

    return Y
```

Here, the input image is first padded with $(M - 1, N - 1)$ zeros on the bottom right, where (M, N) represents the kernel size. Subsequently, a weighted copy of the kernel is placed at each input position, with the weight being the corresponding pixel value, and then summed. This process is illustrated in Figure 4.8. We can think of the kernel as a “stencil” that is used to generate the output, modulated by the weights in the input.

Transposed convolutional layers with strides of 2, which effectively double the spatial dimensions, are commonly used in the decoder of segmentation models such as U-Net [3]. These layers progressively upsample the feature maps to eventually match the output size.

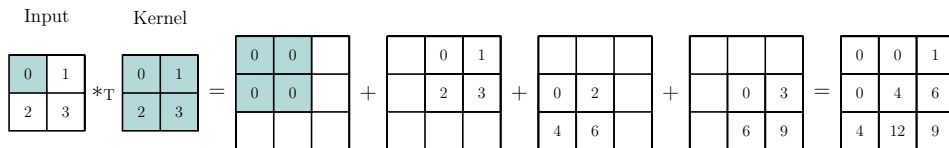


Figure 4.8 Transposed convolution with 2×2 kernel.

4.3.5 Modern CNN Architectures

It is common to use CNNs to perform image classification, which is the task of estimating the function $f : \mathbb{R}^{C \times H \times W} \rightarrow \{0, 1\}^K$, where C is the number of input channels (e.g., $C = 3$ for RGB images), and K is the number of class labels. In this section, we briefly review various CNNs that have been developed over the years to solve image classification tasks.

LeNet. LeNet [64], [65] is often considered the pioneer of CNNs. It was designed for the task of digit recognition and was successfully applied to read handwritten zip code digits on documents.

As illustrated in Figure 4.9a, LeNet’s architecture consists of two sets of convolutional and average pooling layers, followed by an MLP classifier with three fully connected layers (two of which are hidden). The input to LeNet is a 32×32 grayscale image. The first convolutional layer uses six 5×5 filters with stride 1, followed by an average pooling layer. The second convolutional layer uses sixteen 5×5 filters, followed by another average pooling layer. The first and second fully connected layers are followed by sigmoid/tanh and have 120 and 84 neurons, respectively. The last fully connected layer is followed by softmax and has 10 output neurons.

LeNet introduced the concept of local receptive fields, shared weights, and pooling, which are foundational to all modern CNNs. However, it had limitations, such as the use of average pooling which often resulted in loss of useful information, and the use of tanh activation functions which led to the vanishing gradient problem.

LeNet’s significance lies in its pioneering role in demonstrating the potential of CNNs for image recognition tasks. It paved the way for subsequent architectures by establishing the basic structure of CNNs, which includes alternating convolutional and pooling layers followed by fully connected layers.

AlexNet. Having won the ImageNet challenge in 2012 by a significant margin, AlexNet [66] was a breakthrough in the field of deep learning as it showed, for

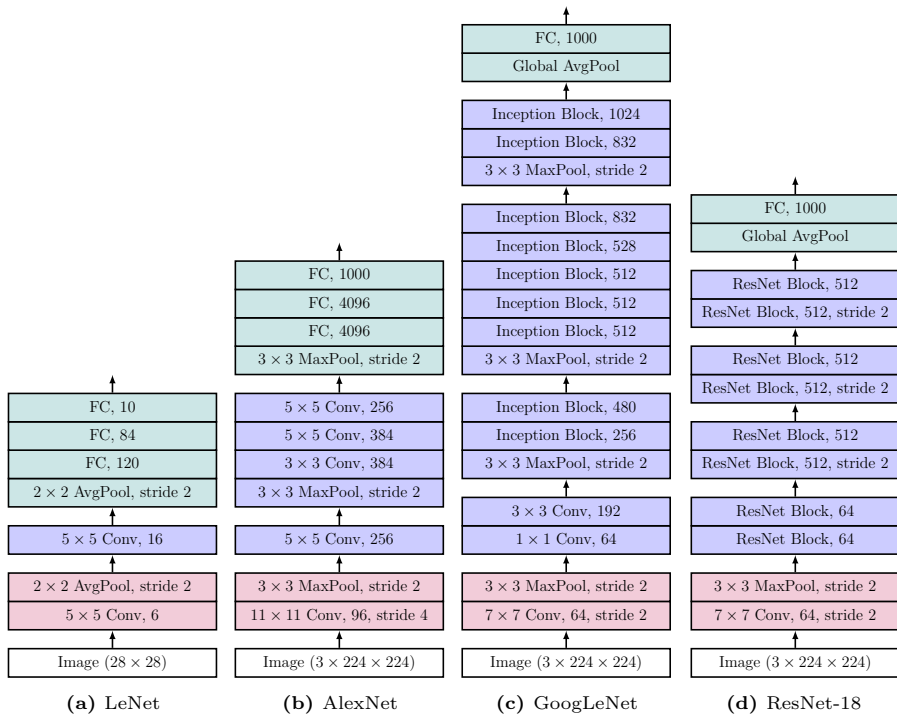


Figure 4.9 Modern CNNs for Image Recognition: The stems are depicted in red, stages in blue, and heads in green. Blocks operating at the same resolution are grouped together for ease of distinction between consecutive stages.

the first time, that the features extracted by learning can transcend hand-crafted features, breaking the previous paradigm in computer vision.

AlexNet built upon the foundational concepts of LeNet but introduced several key modifications. As illustrated in Figure 4.9b, AlexNet was significantly deeper, with five convolutional layers, some of which follow max pooling layers, and three fully connected layers. The network used the ReLU activation function, which helped mitigate the vanishing gradient problem associated with sigmoid/tanh activation functions used in LeNet.

AlexNet also introduced the use of GPU acceleration for training deep neural networks. The original AlexNet was trained on two Nvidia GTX 580 GPUs for about six days. This was a significant development, as it enabled the training of much deeper networks on larger datasets. Another key innovation of AlexNet was the use of *dropout* [69] after hidden fully connected layers. *Dropout* is

a regularization technique to prevent overfitting. Dropout randomly sets a fraction of input units to 0 at each update during training time, which helps prevent units from co-adapting too much.

AlexNet's contributions were transformative, as it demonstrated the feasibility and effectiveness of training deep CNNs, leading to a resurgence of interest in deep learning.

GoogLeNet. The ImageNet challenge in 2014 was won by GoogLeNet [77], a CNN that innovatively combined pointwise convolutions, repeated blocks, and a cocktail of convolution kernels. This approach effectively addressed the challenge of increasing the network depth and width while maintaining a consistent computational budget.

This network was arguably the first to make a clear distinction among the *stem* (data ingest), *body* (data processing), and *head* (prediction) of a CNN. This pattern has been used in the design of deep networks ever since. The *stem* consists of the initial convolution, which has a larger kernel size and extracts lower-level features from the images. This is followed by a *body* of convolutional blocks, which extract higher-level features. Finally, the *head* maps these features to solve the classification, segmentation, detection, or tracking problem at hand.

The core convolutional block in GoogLeNet, known as an *Inception block*, was designed to address the issue of convolution kernel selection in a novel way. While other methods sought to determine the ideal kernel size ranging from 1×1 to 11×11 , the *Inception block* simply concatenated multi-branch convolutions. As illustrated in Figure 4.10a, *Inception block* consists of four parallel branches, each using different types of operations. These include 1×1 , 3×3 , and 5×5 convolutions, as well as a 3×3 max pooling. The outputs of all the branches are then concatenated and sent to the next layer. The use of 1×1 convolutions (pointwise convolutions) is a key aspect of the *Inception block*, as it allows for dimensionality reduction, which helps control the computational cost.

As illustrated in Figure 4.9c, the *stem* (highlighted in red) consists of a 64-channel 7×7 convolutional layer, followed by a 3×3 max pooling layer with strides of 2. The main contribution of GoogLeNet was the design of the network *body* (highlighted in blue), which is composed of four stages. In the first stage of the *body*, a 64-channel 1×1 convolutional layer is followed by a 3×3 convolutional layer that triples the number of channels. The second, third, and fourth stages of the *body* each begin with a 3×3 max pooling layer with strides of 2, halving the resolution. Following this, there are stacks of 2, 5, and 2 *Inception blocks* in the respective stages. The *head* (highlighted in green) of the network consists of a global average pooling and a fully connected layer to generate the prediction.

GoogLeNet was much deeper than AlexNet (22 layers vs. 8 layers), but thanks to

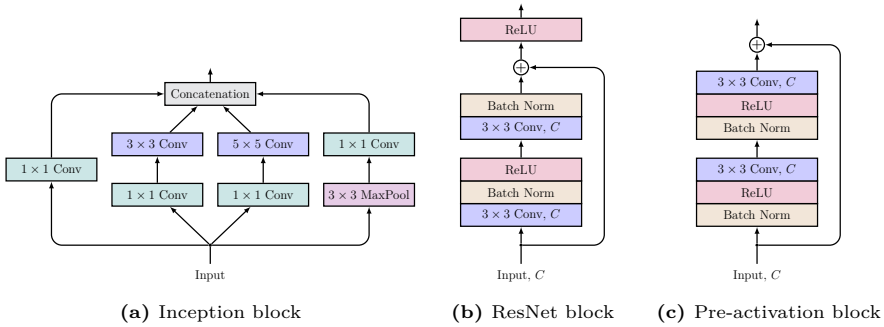


Figure 4.10 Blocks used in the modern CNNs for image recognition.

the Inception modules, it was still computationally efficient. It also introduced the concept of auxiliary classifiers to combat the vanishing gradient problem. These classifiers were attached to intermediate layers of the network, and their loss was added to the total loss of the network with some weight, which provided a regularization effect.

ResNet. The residual network (ResNet) [70] won the ImageNet challenge in 2015. This was a major breakthrough in the field of deep learning, as it enabled the training of extremely deep networks with hundreds of layers without running into the issue of vanishing and exploding gradients. This starkly contrasted previous architectures, which struggled to train networks beyond a depth of 20 layers or so.

The core idea behind ResNet is the introduction of *residual learning principle*. Instead of expecting each few stacked layers directly learn a desired underlying mapping, say $\mathbf{f}(\mathbf{x})$, a ResNet explicitly lets these layers learn the *residual mapping* $\mathbf{g}(\mathbf{x}) \triangleq \mathbf{f}(\mathbf{x}) - \mathbf{x}$, which is the difference between the desired output and the input. The original mapping is, therefore, recast into $\mathbf{g}(\mathbf{x}) + \mathbf{x}$, which can be realized by neural networks with *shortcut connections*. As illustrated in 4.10b, a ResNet block has two 3×3 convolutional layers, each followed by a batch normalization layer and a ReLU activation function. Importantly, there is a *shortcut connection* that adds the input directly before the final ReLU activation function.

He *et al.* [70] argued that optimizing the residual mapping is simpler than optimizing the original, unreferenced mapping. For instance, if the desired underlying mapping is the identity mapping $\mathbf{f}(\mathbf{x}) = \mathbf{x}$, the residual mapping is

just $g(\mathbf{x}) = 0$, which is easier to learn. We only need to adjust the weights and biases of the convolutional layers to zero. Generally, this simple yet powerful idea ensures that even if $g(\mathbf{x})$ becomes very small, which can lead to the vanishing gradients problem, the network can still carry forward a strong gradient via the shortcut connection.

The architecture of ResNet-18 (a ResNet with 18 convolutional and fully connected layers) is illustrated in Figure 4.9d. The *stem* and *head* of ResNet are identical to those of GoogLeNet. The difference is the batch normalization layer added after each convolutional layer in ResNet. Like GoogLeNet, ResNet also has four stages, with the resolution becoming halved in the second, third, and fourth stages. However, ResNet-18 uses two *residual blocks* at each stage. We can create different ResNet models, such as the deeper ResNet-152. Although the main architecture of ResNet is similar to that of GoogLeNet, ResNet's structure is simpler and easier to modify. All these factors have resulted in the rapid and widespread use of ResNet.

Pre-activation ResNet. A later variant of ResNet introduced the concept of *pre-activation* [78], where the order of the layers in the residual block was modified. As depicted in Figure 4.10b, in the original ResNet block, the order was convolution, batch normalization, and ReLU activation. Conversely, in the pre-activation variant, the order is batch normalization, ReLU activation, and convolution, as depicted in Figure 4.10c. The pre-activation design was found to perform better, especially for extremely deep networks.

4.4 Transformers and Attention Mechanisms

Transformers were originally introduced by Vaswani *et al.* [79] as a model based on the *attention mechanism* for machine translation. The *attention*, or *attention* in short, is a mechanism to mix information from input *tokens* in a text to learn its representation (in this context, a *token* refers to a numeric vector that encodes a single unit of text, e.g., word). Since their inception, *transformers* have demonstrated state-of-the-art performance on several natural language processing (NLP) tasks.

Inspired by this, Dosovitskiy *et al.* [80] proposed *vision transformers* (ViTs), which adapts the vanilla *transformer* to computer vision by breaking down input images as a sequence of patches, turning them into vectors, and treating them like tokens in a standard *transformer*. ViT is convolutional-free and solely based on the attention mechanism and MLP. The attention mechanism, as the core component of ViT, is able to capture long-range dependencies without imposing any local locality inductive bias as the convolution operation does.

This weak inductive bias enables ViT to outperform their state-of-the-art CNN counterparts in image recognition when trained on larger datasets (for instance, JFT-300M, which is 300 times larger than ImageNet) [80], [81]. In addition, ViT also scales up more efficiently and has been proven more robust to corruption [82]. Indeed, ViT continues the long-lasting trend of removing hand-crafted visual features and inductive biases from models in an effort to leverage the availability of larger datasets coupled with increased computational capacity.

Existing transformer-based models have demonstrated promising results in medical image segmentation, prompting a surge of interest in further development of such models. This section starts with an overview of ViT and its notable variant, Swin Transformer. It then delves into transformer-based models developed for medical image segmentation, highlighting their key properties, advantages, and shortcomings.

4.4.1 Attention Mechanism

The success of transformers is often credited to the attention mechanism [83], which is designed to capture the long-range internal correlations within a sequence. Let $(\mathbf{x}_1, \dots, \mathbf{x}_N)$ be an input sequence of C -dimensional tokens, represented by matrix $\mathbf{X} = [\mathbf{x}_1 \mid \dots \mid \mathbf{x}_N]^\top$. In an attention layer, each input token \mathbf{x}_n is first transformed into *query* $\mathbf{q}_n = \mathbf{W}^Q \mathbf{x}_n$, *key* $\mathbf{k}_n = \mathbf{W}^K \mathbf{x}_n$, and *value* $\mathbf{v}_n = \mathbf{W}^V \mathbf{x}_n$ vectors, where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{C \times E}$ are learnable weight matrices with E being the *embedding dimension*. The *queries*, *keys*, and *value* can be packed and represented by matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times E}$, respectively. The *attention weights* matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ represents the pairwise similarity between *queries* and *keys*:

$$\mathbf{A} \triangleq \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{E}} \right), \quad (4.9)$$

where softmax is applied row-wise to ensure that the attention weights for a query are nonnegative and sum to 1, i.e., $\sum_{n=1}^N A_{mn} = 1$ and $A_{mn} \geq 0$. The output of *scaled dot-product attention*, or simply *attention*, is defined as [79]:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \triangleq \mathbf{A}\mathbf{V}. \quad (4.10)$$

See Figure 4.11a for an illustration. Note that attention operates in $\mathcal{O}(N^2E)$ time and requires $\mathcal{O}(N^2)$ memory to store the attention weights. This quadratic scaling with sequence length makes attention prohibitively expensive for large inputs, such as high-resolution or 3D images.

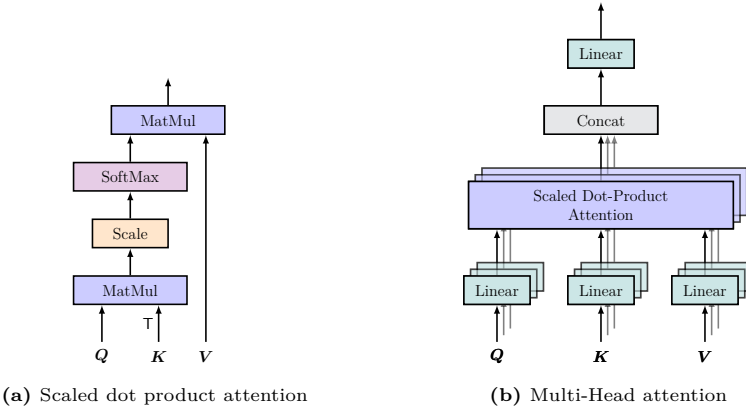


Figure 4.11 (a) Scaled dot product attention. (b) Multi-head attention, where multiple heads are concatenated then linearly transformed.

4.4.2 Multi-Head Attention

Multi-head attention (MHA) is an extension of the attention mechanism that allows the model to attend to different parts of the input sequence simultaneously in multiple representational subspaces [79]. In MHA, the attention operation is executed M times, referred to as “heads,” in parallel. Each head i has its own set of learnable weight matrices $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{E \times E'}$, consequently generating a distinct attention output. The outputs from all these heads are then concatenated and linearly transformed to yield the final multi-head attention output. This process can be formally expressed as:

$$\begin{aligned} \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &\triangleq [\text{Head}_1 \mid \cdots \mid \text{Head}_M] \mathbf{W}^O, \\ \text{Head}_m &= \text{Attention}(\mathbf{Q} \mathbf{W}_m^Q, \mathbf{K} \mathbf{W}_m^K, \mathbf{V} \mathbf{W}_m^V), \end{aligned} \quad (4.11)$$

where $\mathbf{W}^O \in \mathbb{R}^{ME' \times E}$ is a learnable output weight matrix that aggregates information from different heads. To maintain a consistent computational load and parameter count when varying M , the embedding dimension of each head is often set to $E' = E/M$. By allowing the model to attend to multiple aspects of the input simultaneously, MHA leads to a richer representation of the input and enhances the expressive power of the attention mechanism. See Figure 4.11b for an illustration of MHA.

4.4.3 Vision Transformer

Overview. A standard vision transformer (ViT) [80] architecture combines a transformer encoder with a task-specific head, as illustrated in Figure 4.12. When processing 2D images, the image $\mathcal{X} \in \mathbb{R}^{C \times H \times W}$ is first partitioned into a sequence of N non-overlapping patches $(\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_N)$, where each patch $\mathcal{X}_n \in \mathbb{R}^{C \times P \times P}$. Here, C denotes the number of channels, (H, W) the image size, and (P, P) the patch size. Each patch \mathcal{X}_n is then flattened into vector $\mathbf{x}_n \in \mathbb{R}^{CP^2}$ and subsequently linearly transformed into tokens:

$$\mathbf{X}' = [\mathbf{E}\mathbf{x}_1 \mid \mathbf{E}\mathbf{x}_2 \mid \dots \mid \mathbf{E}\mathbf{x}_N]^\top, \quad \mathbf{E} \in \mathbb{R}^{CP^2 \times E}, \quad (4.12)$$

where E is the *embedding dimension*. To preserve the positional information of the patches, a *positional embedding*, \mathbf{E}_{pos} , is added:

$$\mathbf{X} = \mathbf{X}' + \mathbf{E}_{pos}, \quad \mathbf{E}_{pos} \in \mathbb{R}^{N \times D}. \quad (4.13)$$

As illustrated in Figure 4.12, these tokens are then fed into a transformer encoder that consists of L sequentially stacked transformer base blocks. Each block includes a multi-head attention (MHA) module and a multi-layer perceptron (MLP) module. Each module is preceded by a layer normalization (LN) and followed by a shortcut connection. The encoder can be formally described as:

$$\begin{aligned} \mathbf{Z}_0 &= \mathbf{X}, \mathbf{Z}'_\ell = \text{MHA}(\text{LN}(\mathbf{Z}_{\ell-1})) + \mathbf{Z}_{\ell-1}, \ell = 1, \dots, L \\ \mathbf{Z}_\ell &= \text{MLP}(\text{LN}(\mathbf{Z}'_\ell)) + \mathbf{Z}'_\ell, \quad \ell = 1, \dots, L \end{aligned} \quad (4.14)$$

The MHA modules are responsible for the mixing of tokens, while the MLP modules are responsible for the mixing of channels.

Patch Embedding. ViT incorporates a standard transformer for vision tasks with minimal modifications. Consequently, patches are generated in a non-overlapping manner. While non-overlapping patches can disrupt the internal structure of an image, MHA blocks help integrate information from different patches to mitigate this effect. Additionally, using non-overlapping patches avoids computational redundancy in the transformer. Common patch sizes (P) are $\{8, 16, 32\}$, with $P = 16$ typically being the default choice.

Positional Encoding. In standard transformers, each patch is tokenized and processed independently, potentially losing the positional context within the entire image. This is undesirable as the position of each patch is vital for understanding the overall image context. To address this, positional embeddings have been introduced to encode positional information into each patch, ensuring its preservation throughout the network. These embeddings also act as a manually introduced inductive bias in transformers. ViT employs a *learnable positional embedding* \mathbf{E}_{pos} , allowing the network to self-learn positional context.

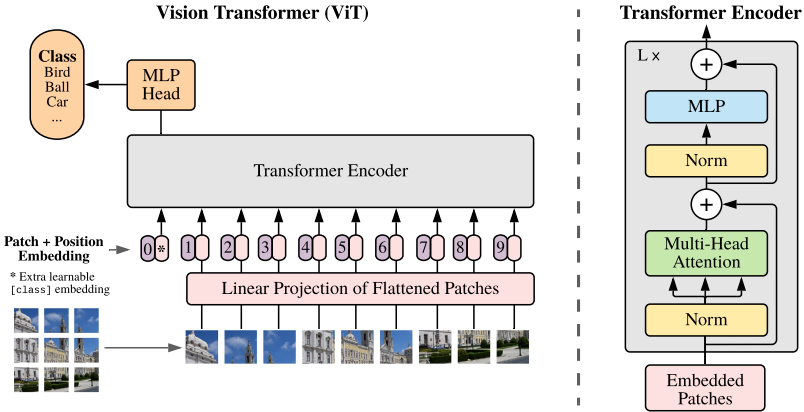


Figure 4.12 Overview of ViT. An image is partitioned into fixed-size patches, each of which is linearly transformed. Position embeddings are then added to these embedded patches, and the resulting sequence of vectors is fed into a standard transformer encoder. For classification, an additional learnable “classification token” is appended to the sequence. From Figure 1 of [80].

MLP. Both in the standard transformer [79] and ViT [80], an MLP follows each attention module. The MLP is pivotal as it introduces inductive bias into the transformer, an element missing in the attention mechanism. This distinction arises because MLP operation is local and translation-equivariant, in contrast to the global nature of the attention mechanism. The MLP comprises two linear (fully connected) layers with GELU activation in between:

$$\text{MLP}(\mathbf{X}) = \text{GELU}(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \quad (4.15)$$

where \mathbf{X} is the input, and \mathbf{W}_i and \mathbf{b}_i denote the weight matrix and bias of the corresponding linear layer, respectively. The weight matrices, \mathbf{W}_1 and \mathbf{W}_2 , typically have dimensions $(E, 4E)$ and $(4E, E)$ [79], [80]. Given the input is a matrix of flattened and tokenized patches as defined in equation (4.13), applying \mathbf{W}_i to \mathbf{X} is analogous to using a convolutional layer with a kernel size of $(1, 1)$. As a result, MLPs in transformers are both highly localized and translation-equivariant.

4.4.4 Swin Transformer

Overview. Swin Transformer [81] is a popular variant of ViT that introduces a hierarchical structure with shifted windows to better capture local and global information and to address the scalability issues associated with ViT, especially for high-resolution images. The architecture of Swin Transformer is illustrated

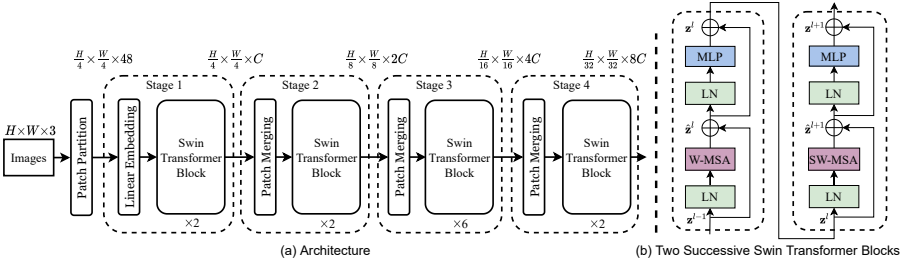


Figure 4.13 Overview of the Swin Transformer. (a) Its Architecture. (b) Two consecutive Swin Transformer blocks. W-MSA and SW-MSA represent local multi-head self-attention modules with regular and shifted window configurations, respectively. From Figure 3 of [81].

in Figure 4.13. Similar to ViT, Swin Transformer partitions an image into non-overlapping patches. However, instead of processing all patches simultaneously at a single scale, Swin Transformer processes them in local windows and hierarchically merges these windows in subsequent layers. To ensure that information is exchanged between adjacent windows, the windows are shifted in subsequent layers, as illustrated in Figure 4.14. Swin Transformer has demonstrated a superior capability to ViT for processing images, particularly in scenarios with limited data and when dealing with diverse scales and contexts.

Hierarchical Structure. Swin Transformer employs a hierarchical structure with multiple stages. In each stage, the resolution is halved, and the number of channels is increased by a factor of 4. This is achieved by merging a 2×2 patch of tokens from the previous stage. The hierarchical nature allows the model to focus on local features in the initial stages and gradually shift its attention to more global features in the later stages.

Shifted Window Approach. At each stage, Swin Transformer divides the image into fixed-size $M \times M$ windows and processes them independently. To capture the information between adjacent windows, the windows are shifted by half of their size in the next layer (Figure 4.14). This approach ensures that each patch in the new window has patches from different original windows, allowing for the integration of local and global contexts. The shifted window mechanism is a key innovation in Swin Transformer, enabling it to capture both fine-grained and high-level features without increasing computational complexity.

Relative Positional Encoding. While ViT employs a fixed positional embedding for each patch, Swin Transformer introduces a relative position bias $B \in \mathbb{R}^{M^2 \times M^2}$ to each attention head when computing the similarities in

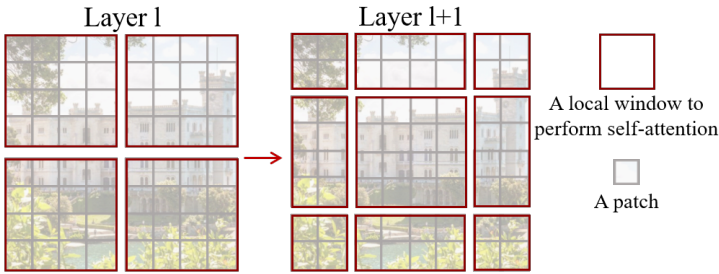


Figure 4.14 Illustration of the shifted window approach for attention computation in Swin Transformer. In layer ℓ (left), a standard window partitioning is used with attention computed within each window. In layer $\ell + 1$ (right), the windows are shifted, forming new partitions. This shift allows attention computation in layer $\ell + 1$ to cross the boundaries set in layer ℓ , enabling connections between them. From Figure 2 of [81].

its local attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{E}} + \mathbf{B} \right) \mathbf{V}. \quad (4.16)$$

Given that the relative position along each dimension spans the range $[-M + 1, M - 1]$, a more compact bias matrix, $\hat{\mathbf{B}} \in \mathbb{R}^{(2M-1) \times (2M-1)}$, is parameterized, and the values in \mathbf{B} are derived from $\hat{\mathbf{B}}$. This design choice is also motivated by the shifted window approach, where the absolute position of a patch can change across stages. The relative positional bias ensures that the model can still capture the spatial relationship between patches despite their changing positions.

MLP. Similar to ViT, Swin Transformer also incorporates an MLP after each attention module. The structure and role of MLP remain consistent with that described in the ViT section. However, due to the hierarchical structure of Swin Transformer, MLP operates at different scales, adapting to the varying resolutions and token counts at each stage.

4.4.5 Performer: Linear Complexity Transformer

The quadratic complexity of the standard attention mechanism poses a significant challenge for processing long sequences, for example resulting from high-resolution images. To address this limitation, Choromanski *et al.* [84] introduced the *Performer*, a transformer variant that approximates the attention using only linear time and space complexity without sacrificing much of the

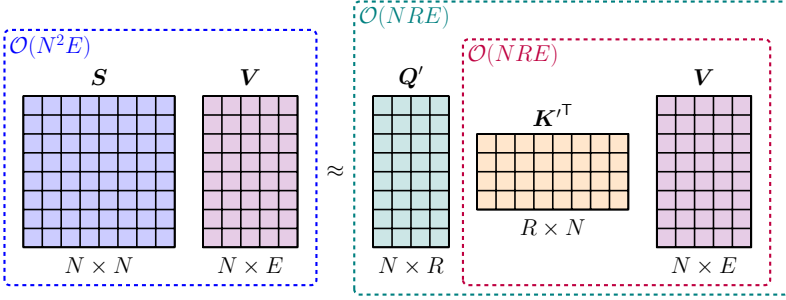


Figure 4.15 Approximation of the attention \mathbf{SV} via (random) feature maps. Dashed blocks indicate the order of computation with corresponding time complexities attached.

expressive power of the original attention mechanism. To approximate the attention, Performers use a *fast attention via positive orthogonal random features (FAVOR+)* approach, which is briefly described in the following.

FAVOR+ leverages the fact that the attention weights matrix \mathbf{A} can be represented as a *kernel matrix*, i.e., $A_{mn} \propto S_{mn} = S(\mathbf{q}_m, \mathbf{k}_n)$, where $S : \mathbb{R}^E \times \mathbb{R}^E \rightarrow \mathbb{R}_+$ is a positive kernel function that measures the similarity between a query and a key. For the regular softmax attention, this kernel is $S(\mathbf{q}, \mathbf{k}) = \exp(\mathbf{q}^\top \mathbf{k} / \sqrt{E})$. Choromanski *et al.* [84] showed that such a kernel function can be approximated as an inner product in the space of a specific *random feature map* $\varphi : \mathbb{R}^E \rightarrow \mathbb{R}_+^R$ ($R \ll N$):

$$S(\mathbf{q}, \mathbf{k}) \approx \langle \varphi(\mathbf{q}), \varphi(\mathbf{k}) \rangle. \tag{4.17}$$

For regular softmax attention, one possible feature map is:

$$\varphi(\mathbf{x}) = \frac{1}{\sqrt{E}} \exp\left(-\frac{\|\mathbf{x}\|^2}{2}\right) [\exp(\mathbf{w}_1^\top \mathbf{x}), \dots, \exp(\mathbf{w}_E^\top \mathbf{x})], \tag{4.18}$$

where $\mathbf{w}_e \stackrel{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_E)$. Refer to Lemma 1 of [84] for further details.

Instead of directly computing the attention weights matrix \mathbf{A} , FAVOR+ substitutes its approximation from equation (4.17) into the attention formula:

$$\mathbf{AV} \propto \mathbf{SV} \approx \mathbf{Q}'(\mathbf{K}'^\top \mathbf{V}), \tag{4.19}$$

where $\mathbf{S} = S(\mathbf{q}_m, \mathbf{k}_n)$ the kernel matrix representing the unnormalized attention weights, and $\mathbf{Q}' = [\varphi(\mathbf{q}_1) \mid \dots \mid \varphi(\mathbf{q}_N)]^\top$ and $\mathbf{K}' = [\varphi(\mathbf{k}_1) \mid \dots \mid \varphi(\mathbf{k}_N)]^\top$ denote the feature maps of queries and keys, respectively. This results in the factorization of the attention weights matrix into two smaller matrices. The

parentheses indicate the order of computations, crucial for achieving linear-time and linear-space complexity with respect to the sequence length. Such a mechanism has a time complexity of $\mathcal{O}(NRE)$ and space complexity of $\mathcal{O}(NR + RN + NE)$, contrasting with the $\mathcal{O}(N^2E)$ time and $\mathcal{O}(N^2 + NE)$ space complexities of regular attention. This is further illustrated in Figure 4.15.

To further diminish the estimator’s variance, enabling the use of even fewer random features R , FLAVOR+ ensures that different random samples $\mathbf{w}_1, \dots, \mathbf{w}_E$ are orthogonal. This is achieved while preserving unbiasedness through the standard Gram-Schmidt orthogonalization procedure.

Remarkably, despite its approximation nature, the Performer has demonstrated performance on par with, or even superior to, standard transformers across various tasks, from language modeling to image classification, all while being significantly more computationally efficient [84].

4.5 Medical Image Segmentation Models

Image segmentation, which involves partitioning an image into distinct regions, is fundamental to numerous applications. A prime example is medical image segmentation, which is a crucial step in computer-aided diagnosis. Automated segmentation not only expedites data processing but also aids clinicians by offering task-specific visualizations and measurements.

Image segmentation tasks are broadly categorized into two types: *semantic segmentation* and *instance segmentation*. *Semantic segmentation* involves pixel-level classification where each pixel in an image is assigned a specific category. On the other hand, instance segmentation involves not only pixel-level classification but also the differentiation of individual objects within the same category based on semantic segmentation. This thesis focuses only on *semantic segmentation* tasks, which hold greater relevance in medical imaging. Notably, there are limited studies on instance segmentation in this field, given the distinct nature of each organ or tissue.

Achieving accurate segmentation models that can identify organ or lesion pixels is often realized through supervised learning. This requires task-specific image data carefully annotated by experts. The prevalent medical imaging modalities include X-ray, computed tomography (CT), magnetic resonance imaging (MRI), and ultrasound. In this thesis, we use MRI data.

Early traditional approaches to medical image segmentation centered on techniques such as edge detection, template matching techniques, region growing, snakes, and machine learning. However, since the advent of deep learning in 2012,

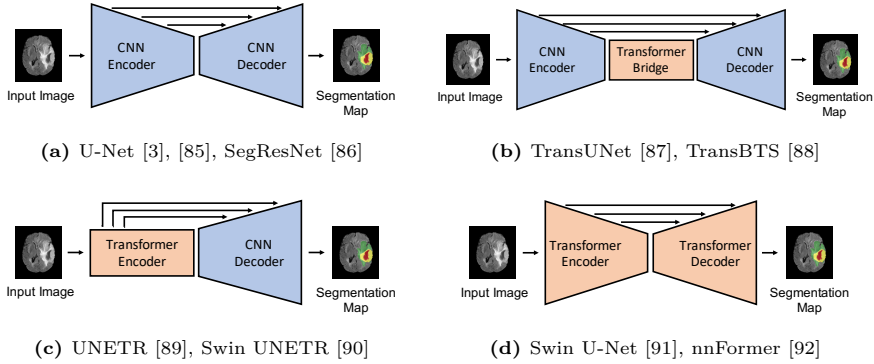


Figure 4.16 Generic forms of segmentation model architectures.

it has become the mainstay in image segmentation. The majority of deep models for semantic segmentation adopt some form of an encoder-decoder architecture. Here, the encoder extracts image features while the decoder restores these features to the original image size and produces the final segmentation map. Particularly, U-Net, introduced by [3] in 2015, is one of the pioneering CNNs with this design, and it emerged as the most popular model for medical image segmentation. Its versatility, modular design, and success across all medical imaging modalities have made it the go-to model for medical image segmentation. The U-shaped architecture with skip connections, as seen in U-Net, has proven effective in leveraging hierarchical features for medical image segmentation.

We can classify existing architectures based on their building blocks into three main categories: (i) CNNs (Figure 4.16a), (ii) hybrid architectures (Figure 4.16b and Figure 4.16c), and (iii) pure transformers (Figure 4.16d). Hybrid architectures, depending on the placement of a transformer, can be further divided into: (a) those with a transformer in their bridge (Figure 4.16b), and (b) those with a transformer in their encoder (Figure 4.16c). In the following, we will delve into each type and highlight representative models.

4.5.1 CNNs

2D U-Net, introduced by Ronneberger *et al.* [3] in 2015, is a CNN designed primarily for the segmentation of 2D medical images. As illustrated in Figure 4.17, the architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. It employs

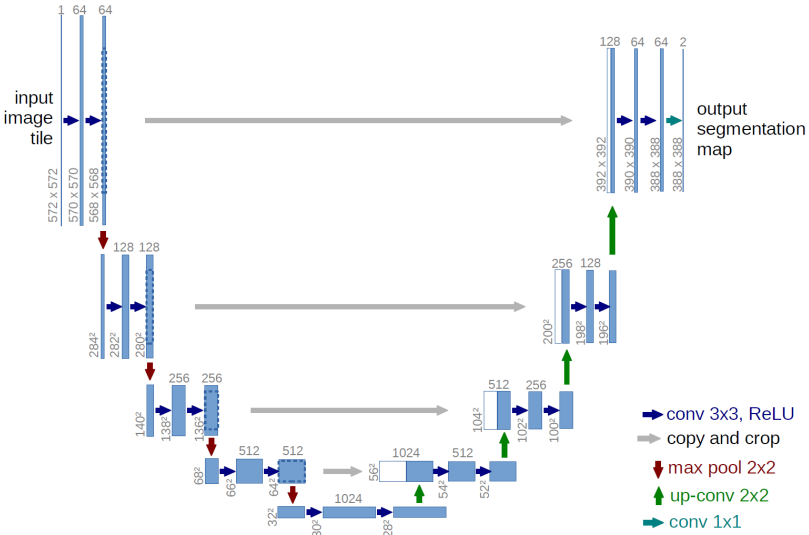


Figure 4.17 The U-Net architecture [3].

skip connections between mirrored layers in the encoder and decoder, allowing for the fusion of high-level and low-level features. Initially applied to cell segmentation tasks, 2D U-Net has since become a foundational model for various medical imaging tasks.

Building on the success of 2D U-Net, Çiçek *et al.* [85] introduced 3D U-Net to process volumetric data directly. By extending the 2D operations to 3D, this model can capture spatial dependencies in three dimensions, making it particularly suitable for tasks like brain lesion segmentation in MRI data.

Built on top of the 3D U-Net architecture with only minor modifications, nnU-Net [93], [94] (no new net) is a self-adapting framework that automatically configures itself, including preprocessing, network width and depth, training, and post-processing for any new task. Without manual intervention, nnU-Net demonstrated state-of-the-art performance on various medical segmentation benchmarks. Notably, nnU-Net won the BraTS 2020 challenges on brain tumor segmentation in mpMRI data [95].

SegResNet [86] integrates the 3D U-Net architecture with a pre-activation residual block. This design is akin to what is depicted in Figure 4.10c, with the exception that it employs group normalization in place of batch normalization. As discussed in Section 4.3.5, residual blocks allow for better gradient flow during training and can lead to improved segmentation accuracy.

4.5.2 Hybrid Architectures

Initial attempts at leveraging the power of transformers for medical image segmentation were to combine ViT and convolutional blocks into a U-Net-like architecture.

Transformer as Bridge. TransUNet [87] combines the strengths of transformers and U-Net for medical image segmentation. It feeds a transformer by the tokenized image patches from a CNN encoder to capture global contexts. A CNN decoder then upsamples the resulting feature maps to output the final segmentation map. TransUNet proved effective for abdominal organ segmentation in CT scans, where capturing both local and global contexts is crucial. Nevertheless, TransUNet is a 2D network that processes the volumetric 3D medical image slice-by-slice and relies on ViT models pre-trained on large-scale image datasets. TransBTS [88] addresses these restrictions by proposing a similar 3D architecture that models local and global contexts in both spatial and slice/depth dimensions.

Transformer as Encoder. There are also hybrid models in which an input image is first encoded into features by a transformer before passing through a CNN decoder. For instance, UNETR Hatamizadeh *et al.* [89] is a 3D network that consists of a pure ViT as the encoder to learn sequence representations of the input image. The encoder is connected to a CNN-based decoder via skip connections to produce the final segmentation map. UNETR has shown impressive performance in different medical image segmentation tasks. Swin UNETR [90] is an adaptation of UNETR that employs Swin Transformer as its hierarchical encoder. UNETR and Swin UNETR demonstrated competitive performance in brain tumor segmentation.

4.5.3 Pure Transformers

Despite the significant success of CNNs in image segmentation, they often face challenges in capturing long-range semantic information within images. As a result, networks exclusively built on transformer blocks have been introduced and demonstrated to be effective in global context modeling for medical image segmentation.

For example, Cao *et al.* [91] introduced Swin-Unet, a purely transformer-based U-shaped architecture tailored for 2D medical image segmentation. This model employs a symmetric hierarchical encoder-decoder structure with skip connections. Notably, Swin-Unet incorporates Swin Transformer blocks at every stage of both the encoder and decoder. A distinctive feature of its

architecture is the utilization of patch-merging layers for downsampling and patch-expanding layers for upsampling. Experiments with abdominal organ and cardiac segmentation tasks showcased the superior performance of Swin-UNet to its CNN or hybrid counterparts. It is important to note that a limitation of relying solely on transformers for segmentation is the quadratic complexity of attention with respect to the number of tokens. This can limit the efficacy of the standard ViT in segmenting high-resolution or large 3D medical images. However, the Swin Transformer blocks within Swin-Unet compute attention only within a localized window, ensuring linear complexity.

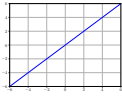
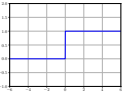
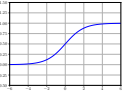
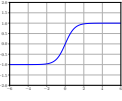
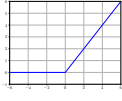
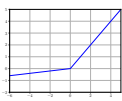
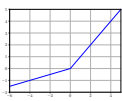
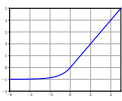
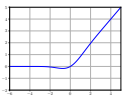
In a related work, Zhou *et al.* [92] proposed nnFormer, which incorporates the local-global concept to refine the multi-head attention mechanism within the transformer structure for 3D medical images. nnFormer is built on a hierarchical U-shaped architecture, employing strided convolution for downsampling and strided transposed convolution for upsampling. The model integrates transformer blocks with local attention in both the encoder and decoder, as well as transformer blocks with global (standard) attention in the bridge, effectively representing 3D images at both local and global scales. The local attention module extends the shifted window attention found in the Swin Transformer. In place of the traditional skip connection, skip attention is used to bridge the gap between the encoder and decoder. nnFormer is initially pre-trained on the ImageNet dataset and leverages symmetrical initialization to repurpose the pre-trained encoder weights in the decoder. Additionally, deep supervision within the decoder layers is applied to improve performance. Experimental results on various tasks, such as brain tumor segmentation, revealed that nnFormer outperforms nnU-Net and earlier transformer-based models like TransUnet, Swin-Unet, and UNETR.

4.6 Conclusion

In this chapter, building upon Chapter 3, we presented the fundamentals of deep learning that underpin modern image segmentation models. We explained how handcrafted features evolved into hierarchical representation learning based on neural networks. We provided an overview of the core building blocks of CNNs, discussing their limitations in capturing global context. We then discussed how transformers can mitigate these challenges with their attention mechanisms. We also highlighted that achieving accurate medical image segmentation models is often accomplished through supervised learning. Based on their building blocks, we classified existing architectures into three major categories: (i) CNNs, (ii) hybrid architectures, and (iii) pure transformers. We emphasized the dominance of the U-Net-like architectural design and reviewed the well-known CNNs,

hybrid models, and pure transformers proposed for medical segmentation. All the models we proposed, as detailed in Chapters 6, 7, and 8, follow a general U-Net-like architecture. The models in Chapters 6 and 8 are based on CNN. Meanwhile, our model and certain baselines in Chapter 7 incorporate elements inspired by the transformer.

Table 4.1 Some popular activation functions and their derivatives. Here, the smoothness indicates the order of continuity measured by the number of continuous derivatives, e.g., C^1 is the class of functions with zeroth and first continuous derivatives. $\text{erf}(\cdot)$ denotes the error function, and $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution, respectively.

Name	Plot	Function, $g(x)$	Derivative, $g'(x)$	Range	Smoothness
Identity		x	1	$(-\infty, \infty)$	C^∞
Heaviside		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\delta(x)$	$\{0, 1\}$	C^{-1}
Sigmoid		$\frac{1}{1 + e^{-x}}$	$g(x)(1 - g(x))$	$(0, 1)$	C^∞
tanh		$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
ReLU [59]		$(x)^+ \triangleq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0, x)$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0, \infty)$	C^0
Leaky ReLU [60]		$\max(0.01x, x)$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
PReLU [61]		$\max(\alpha x, x)$, where α is a learnable parameter	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
ELU [62]		$\begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
GELU [63]		$\frac{1}{2}x \left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.5x, \infty)$	C^∞

Chapter 5

Low-Rank Factorization

Low-rank factorization techniques excel in data compression and mining, and they play a central role in the deep neural networks introduced in this thesis for brain lesion segmentation. This chapter begins with an overview of non-negative matrix factorization (NMF), with a focus on the widely-used multiplicative update (MU) and hierarchical alternating least squares (HALS) algorithms. Such factorization can serve as an approach to global context modeling, offering an efficient alternative to the attention mechanism. This underpins the approach we introduce in Chapter 7. Additionally, we discuss tensor networks, highlighting their significance as a computational tool and in forming low-rank representations. This discussion is enriched by a review of common tensor operators and tensor decomposition formats. This chapter concludes with an exploration of the varied applications of tensor networks in machine learning, including model compression and acceleration.

5.1 Nonnegative Matrix Factorization

Nonnegative matrix factorization (NMF) [96], [97] is a powerful technique for decomposing a nonnegative matrix into the product of two nonnegative factor matrices. The motivation behind NMF lies in its ability to interpret inherently nonnegative data using a reduced number of components while preserving its nonnegativity. Unlike methods such as singular value decomposition (SVD) and principal component Analysis (PCA), which create factors with both positive and negative elements, NMF guarantees that all elements in the factor matrices are nonnegative, making it ideal for analyzing data where negativity does not have a clear interpretation, such as pixel intensities in images or word counts in document corpora.

Given a nonnegative matrix $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$, NMF seeks an approximation

$$\mathbf{X} \approx \mathbf{F}\mathbf{G}^T = \sum_{r=1}^R \mathbf{F}_{:r} \mathbf{G}_{:r}^T, \quad (5.1)$$

where $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$ are nonnegative *factor matrices*, and $R \leq \min(M, N)$ is known as the *rank*. In practical applications, R is often much less than $\min(M, N)$. This implies that the factor matrices \mathbf{F} and \mathbf{G} provide a compressed representation of the original matrix, encapsulating the most significant patterns in the data. Such a low-rank approximation can also reveal hidden data structures, which often yield easily interpretable factors. Another characteristic of NMF is its tendency to encourage sparsity, which is mainly due to the nonnegativity constraint that can make many elements in the factor matrices zero.

Definition 5.1 (Frobenius Norm). *The Frobenius norm of a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ is defined as the square root of the sum of the squares of its elements:*

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^M \sum_{j=1}^N A_{ij}^2} \quad (5.2)$$

This can also be expressed as the square root of the trace of the product of \mathbf{A} and its transpose:

$$\|\mathbf{A}\|_F = \sqrt{\text{Tr}(\mathbf{A}\mathbf{A}^T)}. \quad (5.3)$$

5.1.1 NMF Problem

NMF can be framed as an optimization problem, aiming to minimize the reconstruction error between the original matrix and the product of the two factor matrices, under the constraint that these factor matrices are nonnegative. Formally, the standard NMF problem can be expressed as:

$$\begin{aligned} \min_{\mathbf{F}, \mathbf{G}} \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|_{\mathbb{F}}^2 \\ \text{s.t. } \mathbf{F} \geq 0, \mathbf{G} \geq 0 \end{aligned} \quad (5.4)$$

where $\|\cdot\|_{\mathbb{F}}$ denotes the Frobenius norm. This optimization problem is non-trivial to solve. The cost function is not convex with respect to both parameters \mathbf{F} and \mathbf{G} simultaneously, but it is convex with respect to each parameter individually when the other is fixed. This property motivates the application of BCD, introduced in Section 3.2.3, to solve the NMF problem.

Note that the norm used above is the Frobenius norm, which is a standard choice in many NMF variants. However, depending on the specific application, other types of norms or dissimilarity measures can also be used to define the cost function, such as the Kullback-Leibler divergence [98] or the beta-divergence [99]. Furthermore, additional regularization terms can be added to the cost function to enforce certain desirable properties, such as sparsity or smoothness, on the factor matrices. In this thesis, we only use the standard NMF, presented by problem (5.4).

The solution to the NMF problem provides two factor matrices, which can be interpreted as the bases and the coefficients, respectively. As illustrated in Figure 5.1, each column of the basis matrix \mathbf{F} can be viewed as a basis vector (or component), and each column of the coefficient matrix \mathbf{G}^T represents the weights of the bases in approximating the corresponding column in the original matrix. Thus, the original data matrix is represented as a linear combination of the bases weighted by the coefficients, offering a parts-based representation of the data. This ability to reveal the hidden, often interpretable, structure in the data is one of the main attractive properties of NMF.

First-Order Optimality Conditions. Let's denote the cost function by $J(\mathbf{F}, \mathbf{G}) = \frac{1}{2}\|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|_{\mathbb{F}}^2$. We introduce the Lagrangian function

$$L(\mathbf{F}, \mathbf{G}, \mathbf{A}_{\mathbf{F}}, \mathbf{A}_{\mathbf{G}}) = J(\mathbf{F}, \mathbf{G}) - \text{tr}(\mathbf{A}_{\mathbf{F}}\mathbf{F}^T) - \text{tr}(\mathbf{A}_{\mathbf{G}}\mathbf{G}^T)$$

where the Lagrangian multipliers $\mathbf{A}_{\mathbf{F}} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{A}_{\mathbf{G}} \in \mathbb{R}_{\geq 0}^{N \times R}$ enforce nonnegative constraints $\mathbf{F} \geq 0$ and $\mathbf{G} \geq 0$, respectively. The zero gradient condition gives $\nabla_{\mathbf{F}}L = \nabla_{\mathbf{F}}J - \mathbf{A}_{\mathbf{F}} = 0$ and $\nabla_{\mathbf{G}}L = \nabla_{\mathbf{G}}J - \mathbf{A}_{\mathbf{G}} = 0$. From the

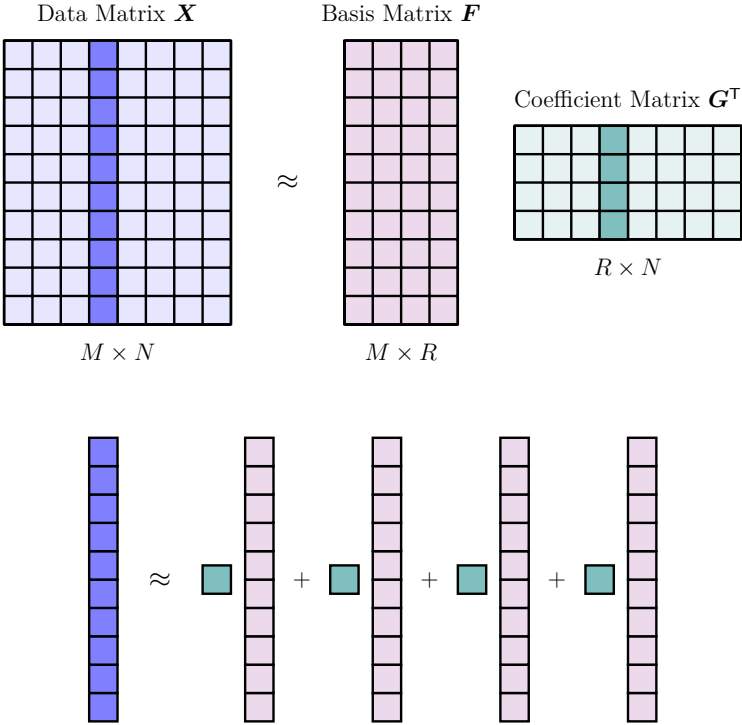


Figure 5.1 An illustration of NMF. Each column of the data matrix \mathbf{X} (a feature) is represented as a linear combination of the basis vectors, i.e., the columns of \mathbf{F} , with their corresponding weights in \mathbf{G} .

complementary slackness condition, we obtain $\mathbf{F} \odot \mathbf{A}_{\mathbf{F}} = \mathbf{F} \odot \nabla_{\mathbf{F}} J = 0$ and $\mathbf{G} \odot \mathbf{A}_{\mathbf{G}} = \mathbf{G} \odot \nabla_{\mathbf{G}} J = 0$, where \odot denotes the element-wise product. Therefore, the first-order optimality conditions for problem (5.4) can be summarized as:

$$\begin{aligned} \mathbf{F} &\geq 0, & \nabla_{\mathbf{F}} J &= \mathbf{F} \mathbf{G}^T \mathbf{G} - \mathbf{X} \mathbf{G} \geq 0, & \mathbf{F} \odot \nabla_{\mathbf{F}} J &= 0, \\ \mathbf{G} &\geq 0, & \nabla_{\mathbf{G}} J &= \mathbf{G} \mathbf{F}^T \mathbf{F} - \mathbf{X}^T \mathbf{F} \geq 0, & \mathbf{G} \odot \nabla_{\mathbf{G}} J &= 0. \end{aligned} \quad (5.5)$$

Any (\mathbf{F}, \mathbf{G}) satisfying these conditions is a stationary point of (5.4). It is interesting to observe that these conditions give a more formal explanation of why NMF naturally generates sparse solutions [51]. In fact, any stationary point of (5.4) is expected to have zero entries because of the conditions $\mathbf{F} \odot \nabla_{\mathbf{F}} J = 0$ and $\mathbf{G} \odot \nabla_{\mathbf{G}} J = 0$ that is, the conditions that for all m, n either f_{mn} is equal to zero or the partial derivative of J with respect to f_{mn} is, and similarly for \mathbf{G} .

Algorithm 2: General two-block coordinate descent scheme of most NMF algorithms.

Input: Nonnegative matrix $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$, factorization rank R

Output: Factor matrices $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$; $\mathbf{X} \approx \mathbf{F}\mathbf{G}^\top$

1 Initialize $\mathbf{F}^0 \in \mathbb{R}_{\geq 0}^{M \times R}$, $\mathbf{G}^0 \in \mathbb{R}_{\geq 0}^{N \times R}$

2 $t = 0$

3 **while** *stopping criterion not satisfied* **do**

4 $\mathbf{F}^{t+1} = \text{update}(\mathbf{X}, \mathbf{F}^t, \mathbf{G}^t)$, typically such that

$$\|\mathbf{X} - \mathbf{F}^{t+1}\mathbf{G}^{t\top}\|_{\text{F}} \leq \|\mathbf{X} - \mathbf{F}^t\mathbf{G}^{t\top}\|_{\text{F}}$$

5 $\mathbf{G}^{t+1} = \text{update}(\mathbf{X}^\top, \mathbf{G}^t, \mathbf{F}^{t+1})$, typically such that

$$\|\mathbf{X} - \mathbf{F}^{t+1}\mathbf{G}^{t+1\top}\|_{\text{F}} \leq \|\mathbf{X} - \mathbf{F}^{t+1}\mathbf{G}^{t\top}\|_{\text{F}}$$

6 $t \leftarrow t + 1$

7 **end**

8 **return** $(\mathbf{F}^t, \mathbf{G}^t)$

5.1.2 NMF Algorithms

A variety of algorithms have been developed to solve the NMF problem. These algorithms often leverage the two-block coordinate descent (BCD) scheme due to the separable nonnegative constraints and the convex nature of the cost function in the NMF problem with respect to \mathbf{F} and \mathbf{G} individually when the other is fixed.

The BCD scheme for NMF algorithms involves iterative updates of the factor matrices \mathbf{F} and \mathbf{G} . More specifically, in each iteration, we fix one of the matrices (e.g., \mathbf{G}) and update the other matrix (e.g., \mathbf{F}) by either exactly or approximately minimizing the cost function with respect to that matrix alone. The roles of the factor matrices are then swapped in the subsequent step. This process is repeated until a stopping criterion is met, such as the change in the cost function value is below a predefined threshold or the maximum number of iterations is reached. This approach is formally presented in Algorithm 2.

The factor matrices \mathbf{F} and \mathbf{G} can be initialized in various ways, such as random initialization or nonnegative double singular value decomposition. The update rule for \mathbf{F} and \mathbf{G} in lines 4 and 5 in Algorithm 2, respectively, are

placeholders and can be replaced with specific update rules depending on the particular NMF algorithm being implemented. Note that if we have an update rule for \mathbf{F} , we can also use that rule to update \mathbf{G} due to the equality $\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\|_F = \|\mathbf{X}^\top - \mathbf{G}\mathbf{F}^\top\|_F$, by simply transposing the original matrix and switching the roles of the factor matrices. Most NMF algorithms are monotone, where the update rule monotonically decreases the cost function at every step, i.e., $\mathbf{F}^{t+1} = \text{update}(\mathbf{X}, \mathbf{F}^t, \mathbf{G}^t)$ such that

$$\|\mathbf{X} - \mathbf{F}^{t+1}\mathbf{G}^{t\top}\|_F \leq \|\mathbf{X} - \mathbf{F}^t\mathbf{G}^{t\top}\|_F \quad (5.6)$$

In the following, we will detail multiplicative updates (MU) [97] and hierarchical alternating least squares (HALS) [100], [101], two representative monotonic NMF solvers based on the BCD scheme.

Multiplicative Updates

The multiplicative updates (MU), proposed by Lee and Seung [97], is one of the earliest and most well-known algorithms for Nonnegative Matrix Factorization (NMF). The algorithm is based on an iterative scheme in which each factor matrix is updated by element-wise multiplication with a ratio of two nonnegative matrices. The multiplicative update rules can be formally written as:

$$\begin{aligned} \mathbf{F} &\leftarrow \mathbf{F} \odot \frac{\mathbf{X}\mathbf{G}}{\mathbf{F}\mathbf{G}^\top\mathbf{G}}, \\ \mathbf{G} &\leftarrow \mathbf{G} \odot \frac{\mathbf{X}^\top\mathbf{F}}{\mathbf{G}\mathbf{F}^\top\mathbf{F}}, \end{aligned} \quad (5.7)$$

where \odot and \div denote element-wise product and division, respectively. The MU can be also interpreted as an adaptive gradient descent:

$$\begin{aligned} \mathbf{F} &\leftarrow \mathbf{F} \odot \frac{\mathbf{X}\mathbf{G}}{\mathbf{F}\mathbf{G}^\top\mathbf{G}} = \mathbf{F} - \frac{\mathbf{F}}{\mathbf{F}\mathbf{G}^\top\mathbf{G}} \odot \nabla_{\mathbf{F}} J, \\ \mathbf{G} &\leftarrow \mathbf{G} \odot \frac{\mathbf{X}^\top\mathbf{F}}{\mathbf{G}\mathbf{F}^\top\mathbf{F}} = \mathbf{G} - \frac{\mathbf{G}}{\mathbf{G}\mathbf{F}^\top\mathbf{F}} \odot \nabla_{\mathbf{G}} J, \end{aligned} \quad (5.8)$$

where $J(\mathbf{F}, \mathbf{G}) = \frac{1}{2}\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\|_F^2$ is the cost function of NMF.

Theorem 5.1. *The standard NMF cost function, $\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\|_F^2$, is monotonically decreasing (nonincreasing) under each of the multiplicative update rules.*

Proof The proof is based on the majorization-minimization technique, discussed in Section 3.2.4. Given the current solution \mathbf{F}^t , we fix \mathbf{G} and consider the NMF cost function

$$J(\mathbf{F}) = \|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\|_{\mathbb{F}}^2 = \sum_{m=1}^M \sum_{n=1}^N \left(X_{mn} - \sum_{r=1}^R F_{mr} G_{nr} \right)^2, \quad (5.9)$$

and define

$$\mathcal{A}_{mnr}^t \triangleq \frac{F_{mr}^t G_{nr}}{\sum_{r'=1}^R F_{mr'}^t G_{nr'}}, \quad (5.10)$$

which satisfies $\sum_{r=1}^R \mathcal{A}_{mnr}^t = 1$ and $\mathcal{A}_{mnr}^t \geq 0$. Now, noting the fact that the quadratic function $\ell(\theta) = (X_{mn} - \theta)^2$ is convex, we can derive a function $Q(\mathbf{F} | \mathbf{F}^t)$ that majorizes $J(\mathbf{F})$ using Jensen's inequality

$$\begin{aligned} J(\mathbf{F}) &= \sum_{m=1}^M \sum_{n=1}^N \left(X_{mn} - \sum_{r=1}^R \frac{\mathcal{A}_{mnr}^t}{\mathcal{A}_{mnr}^t} F_{mr} G_{nr} \right)^2 \\ &\leq \sum_{m=1}^M \sum_{n=1}^N \sum_{r=1}^R \mathcal{A}_{mnr}^t \left(X_{mn} - \frac{F_{mr} G_{nr}}{\mathcal{A}_{mnr}^t} \right)^2 \\ &= Q(\mathbf{F} | \mathbf{F}^t). \end{aligned} \quad (5.11)$$

We can easily show that $J(\mathbf{F}^t) = Q(\mathbf{F}^t | \mathbf{F}^t)$, and hence, $Q(\mathbf{F} | \mathbf{F}^t)$ serves as a surrogate function to be minimized. To find the minimum of $Q(\mathbf{F} | \mathbf{F}^t)$, we solve $\frac{\partial Q(\mathbf{F} | \mathbf{F}^t)}{\partial F_{mr}} = 0$, which yields the multiplicative update rule for \mathbf{F} . ■

Although MU guarantees the monotonic decrease of the cost function, it often suffers from slow convergence. This is due to the nature of the multiplicative updates, which do not allow a direct step towards the minimum of the cost function, but rather a slower, proportional adjustment of the factor elements.

It is worth mentioning that the MU algorithm is a first-order method, which means that each update involves the computation of the gradient. This can be easily revealed by comparing $\nabla_{\mathbf{F}} J$ in (5.5) and the multiplicative update rule for \mathbf{F} in (5.7) and noticing that the multiplicative update rules require taking the ratio of the two terms in the corresponding gradient.

Hierarchical Alternating Least Squares

Hierarchical Alternating Least Squares (HALS), proposed by Cichocki *et al.* [100] and Cichocki and Phan [101], is another effective, popular algorithm for solving

Algorithm 3: Hierarchical alternating least squares (HALS) for NMF.

Input: Nonnegative matrix $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$, factorization rank R

Output: Factor matrices $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$; $\mathbf{X} \approx \mathbf{F}\mathbf{G}^\top$

```

1 Initialize  $\mathbf{F}^0 \in \mathbb{R}_{\geq 0}^{M \times R}$ ,  $\mathbf{G}^0 \in \mathbb{R}_{\geq 0}^{N \times R}$ 
2 while stopping criterion not satisfied do
3    $\mathbf{A} \leftarrow \mathbf{X}\mathbf{G}$ ,  $\mathbf{B} \leftarrow \mathbf{G}^\top\mathbf{G}$ 
4   for  $r = 1, \dots, R$  do
5      $F_{:r} \leftarrow \max\left(0, \frac{A_{:r} - \sum_{s \neq r} B_{sr}F_{:s}}{\|\mathbf{G}_{:r}\|^2}\right)$ 
6   end
7    $\mathbf{A} \leftarrow \mathbf{X}^\top\mathbf{F}$ ;  $\mathbf{B} \leftarrow \mathbf{F}^\top\mathbf{F}$ 
8   for  $r = 1, \dots, R$  do
9      $G_{:r} \leftarrow \max\left(0, \frac{A_{:r} - \sum_{s \neq r} B_{sr}G_{:s}}{\|\mathbf{G}_{:r}\|^2}\right)$ 
10  end
11 end
12 return  $(\mathbf{F}, \mathbf{G})$ 

```

the NMF problem. The HALS algorithm sequentially solves subproblems, each focusing on updating a single column of \mathbf{F} or \mathbf{G} at a time. The subproblem for updating the r th column of \mathbf{F} can be expressed as:

$$\min_{F_{:r} \geq 0} \|\mathbf{X} - F_{:r}\mathbf{G}_{:r}^\top - \sum_{s \neq r} F_{:s}\mathbf{G}_{:s}^\top\|_{\mathbb{F}}^2. \quad (5.12)$$

This boils down to a single-variable quadratic optimization problem, where its optimal solution can be easily found in a closed form. This simplifies the update process and yields the following update rule:

$$F_{:r} \leftarrow \max\left(0, \frac{A_{:r} - \sum_{s \neq r} b_{sr}F_{:s}}{\|\mathbf{G}_{:r}\|^2}\right) \quad (5.13)$$

where $\mathbf{A} = \mathbf{X}\mathbf{G}$, and $\mathbf{B} = \mathbf{G}^\top\mathbf{G}$. A similar update rule can be derived for columns of \mathbf{G} . A pseudocode of HALS is presented in Algorithm 3.

Computational Complexity

We can estimate the number of floating point operations (FLOPs) for MU and HALS. In our analyses, we assume the NMF gives a compressed representation. This means that storing both \mathbf{F} and \mathbf{G} combined must require less space than

Table 5.1 MU computational complexity for the update of \mathbf{F} .

Operation	# FLOPs
$\mathbf{A} \leftarrow \mathbf{X}\mathbf{G}$	$2MNR$
$\mathbf{B} \leftarrow \mathbf{G}^\top \mathbf{G}$	$2NR^2$
$\mathbf{C} \leftarrow \mathbf{F}\mathbf{B}$	$2MR^2$
$\mathbf{D} \leftarrow \frac{\mathbf{A}}{\mathbf{C}}$	MR
$\mathbf{F} \leftarrow \mathbf{F} \odot \mathbf{D}$	MR
Total	$R(2MN + 2NR + 2MR + 2M)$

Table 5.2 HALS computational complexity for the update of \mathbf{F} .

Operation	# FLOPs
$\mathbf{A} \leftarrow \mathbf{X}\mathbf{G}$	$2MNR$
$\mathbf{B} \leftarrow \mathbf{G}^\top \mathbf{G}$	$2NR^2$
$c_{:r} \leftarrow \sum_{s \neq r} b_{sr} \mathbf{f}_{:s}$	$2MR(R-1)$
$\mathbf{d}_{:r} \leftarrow \mathbf{a}_{:r} - \mathbf{c}_{:r}$	MR
$e_{:r} \leftarrow \frac{\mathbf{d}_{:r}}{\mathbf{b}_{:r}}$	MR
$\mathbf{f}_{:r} \leftarrow \max(0, e_{:r})$	MR
Total	$R(2MN + 2NR + 2MR + M)$

\mathbf{X} , i.e., $R(M+N) \leq MN$. Table 5.1 and Table 5.2 list all the operations used in a single update of \mathbf{F} together with their corresponding number of FLOPs in MU and HALS, respectively

We observe that both MU and HALS have the same computational complexity of $\mathcal{O}(MNR)$ per update, which is linear in the size of the data matrix and rank (HALS requires only MR fewer FLOPs, which is negligible). For both algorithms, the first step, where $\mathbf{X}\mathbf{G}$ is computed, is the most expensive one under the assumption $R(M+N) \leq MN$.

Comparison

We compare the performance of the MU and HALS algorithms on the Olivetti faces dataset [102], taken at AT&T Laboratories Cambridge (<https://cam.ac.uk/research/research-centres/at-t-laboratories>). This dataset consists of 400 grayscale images of size 64×64 . Figure 5.2a shows sample images from the dataset. Each image was flattened into a row vector, and these vectors were then stacked row-wise to create a 400×4096 data matrix. In our experiment, we set $R = 20$. The reconstruction performance was evaluated using the *relative error*, defined as

$$\text{RE} = \frac{\|\mathbf{X} - \mathbf{F}\mathbf{G}^\top\|_{\text{F}}}{\|\mathbf{X}\|_{\text{F}}} \quad (5.14)$$

Figure 5.3 illustrates the relative error plotted as a function of the number of FLOPs. Under the same number of FLOPs, it is clear that the HALS algorithm consistently achieves a lower relative error compared to the MU algorithm. Figure 5.2b illustrates the NMF components, i.e., columns of the matrix \mathbf{G} , with HALS being used as the solver.



Figure 5.2 Illustration of NMF components for the Olivetti faces dataset. (a) Sample images from the Olivetti faces dataset. (b) NMF components obtained by HALS.

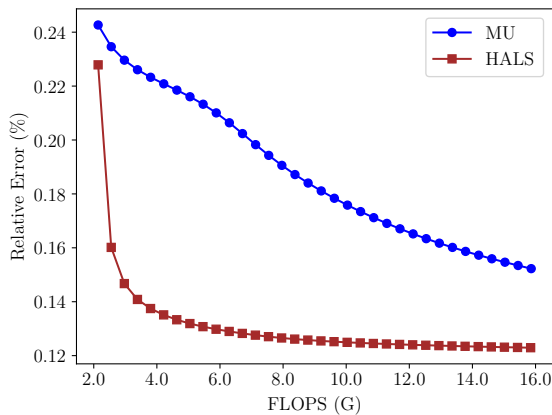


Figure 5.3 Comparison of NMF algorithms. The experiments were conducted using $R = 20$ on the Olivetti faces dataset, represented by a 400×4096 data matrix. Each point on the curves corresponds to a certain number of iterations (ranging from 1 to 100). Within the same computational cost, HALS shows an improved relative error compared to MU.

5.1.3 Choice of Rank

The rank R in NMF determines the number of components in the resulting factorization, and thus, plays a crucial role in the interpretability of the factorization. It is akin to selecting the number of clusters in clustering algorithms or the number of latent factors in other factorization methods. The appropriate choice of rank is dependent on the specific data and the

purpose of the analysis, and it may significantly affect the outcomes of the factorization. Unfortunately, there is no universal rule to select the best rank due to the diversity of data and tasks, but several heuristic methods have been proposed. Here, we describe one of such methods based on the *elbow method* [103], which is also applicable in the context of NMF.

Elbow Method with the Kneedle Algorithm

The elbow method is a heuristic method widely used in statistics to determine the number of clusters in a dataset and can be adapted to select the rank in NMF. The underlying idea of the elbow method is to find a point in a curve, often a cost or error curve, where the *elbow* or *knee* occurs. This *elbow* is the point of inflection where the curve starts to flatten or the rate of decrease significantly slows down, indicating diminishing returns. In the context of NMF, the curve can represent the relative error as a function of the rank, where the error decreases as the rank increases.

However, the elbow method is often criticized for its subjectivity since the selection of the “elbow” is usually done visually and may vary depending on the observer. To overcome this drawback, Satopaa *et al.* [103] proposed “Kneedle,” which offers a simple algorithmic way to detect the elbow in a curve. Figure 5.4 illustrates how Kneedle can be used to find the rank of NMF in the example of the Olivetti faces dataset. Here is a high-level overview of the Kneedle algorithm applied to rank selection in NMF:

1. Compute the NMF for a range of ranks and record the corresponding relative errors, as illustrated in Figure 5.4a.
2. Normalize and transform the rank and error values to $[0, 1]$ to get a curve starting from $(0, 0)$ and ending with $(1, 1)$, as illustrated in Figure 5.4c.
3. Form the line that connects the first and the last points of the normalized error curve, as illustrated in Figure 5.4c.
4. Compute the difference curve, which is the difference between the normalized error curve and the line formed in the previous step. See Figure 5.4c and Figure 5.4d.
5. Calculate the “best” local maximum point on the difference curve. This point corresponds to the elbow of the original curve. See Figure 5.4d.

Equivalently, Kneedle identifies the elbow point on the normalized error curve as the point with the maximum perpendicular distance from the line represented by $y = x$, as depicted in Figure 5.4b.

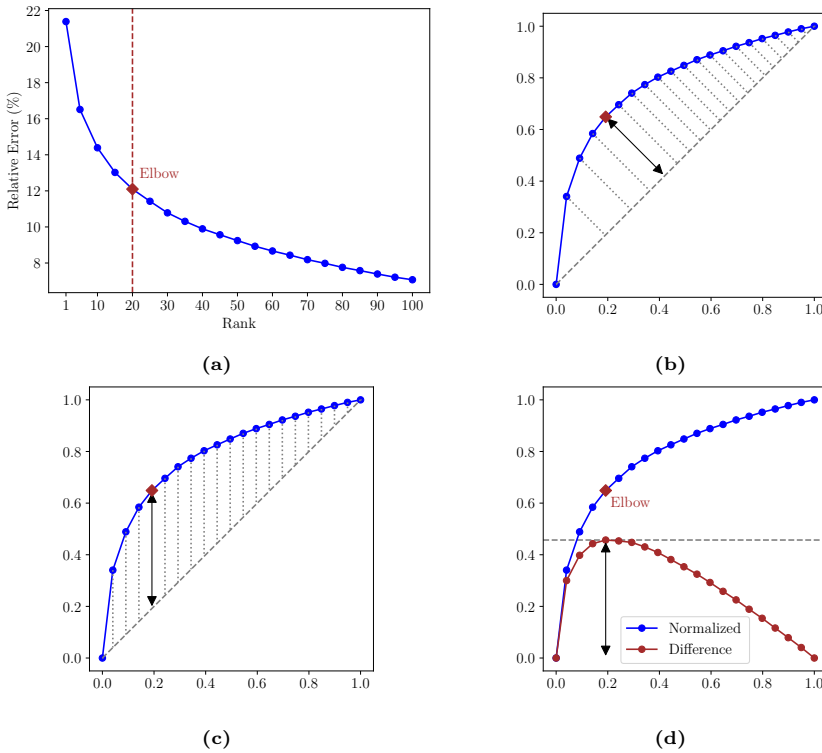


Figure 5.4 Choosing rank of NMF using the Kneedle algorithm [103] for elbow detection. Here, we experimented with the Olivetti faces dataset, represented by a 400×4096 data matrix (a) shows the original plot of relative error as a function of rank, where the Kneedle algorithm yields the optimal rank of 20. (b) depicts the normalized data used by Kneedle, with dashed bars indicating the perpendicular distance from the line $y = x$ with the maximum distance indicated. (c) shows the same data, but this time the dashed bars are rotated 45 degrees. The magnitude of these bars correspond to the difference values used in Kneedle. (d) shows the plot of these difference values, with the elbow occurring at $x = 0.2$.

The rank corresponding to the elbow point found by the Kneedle algorithm is chosen as the optimal rank for the NMF. This rank is expected to balance between the complexity of the model (i.e., the number of basis vectors) and the quality of the approximation (i.e., the reconstruction error). The elbow method with the Kneedle algorithm provides an objective, automated, and principled way to select the rank in NMF. It can be particularly useful when the optimal rank is not known a priori and cannot be inferred from the context of the application. However, it is worth noting that this method, like other

heuristic methods, does not always guarantee the optimal solution and might need to be combined with domain knowledge or other validation techniques for robust rank selection.

5.2 Tensor Networks

5.2.1 Tensors, Tensor Networks, and Tensor Diagrams

As previously noted in Section 3.1.2, multi-dimensional data is ubiquitous in machine learning and can be effectively represented by tensors. For our purposes, a tensor can be seen as a multidimensional array of numbers. An M th-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ is a multi-dimensional array where the m th mode or dimension has a size I_m . Thus, a 0th-order tensor is scalar $a \in \mathbb{R}$, a 1st-order tensor is a vector $\mathbf{a} \in \mathbb{R}^D$, and a 2nd-order tensor is a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$, and so on.

An *index contraction* refers to the summation over all possible values of an index for a set of tensors. A simple illustration of this is the matrix product

$$C_{ij} = \sum_{k=1}^K A_{ik} B_{kj}, \tag{5.15}$$

which represents the contraction of index k , summing over its K possible values. One can also have more complex contractions, such as this one:

$$\mathcal{E}_{ijk} = \sum_{\ell, m, n, p, q=1}^R \mathcal{A}_{i\ell m} \mathcal{B}_{j\ell n p} \mathcal{C}_{k p q} \mathcal{D}_{m n q}, \tag{5.16}$$

where we assume for simplicity that contracted indices ℓ, m, n, p , and q can each take R different values. These examples illustrate how the contraction of indices generates new tensors, akin to how the product of two matrices yields a new matrix. Indices that are not contracted (i, j , and k in this example) are termed *open indices*.

A *tensor network* (TN) is a set of tensors where some or all of the indices are contracted according to some pattern. The above two equations serve as instances of TNs. In equation (5.15), the TN is equivalent to a matrix product and yields a new matrix with two open indices. In equation (5.16), the TN involves contracting indices ℓ, m, n, p , and q in tensors $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and \mathcal{D} to produce a new 3rd-order tensor \mathcal{E} with open indices i, j , and k .

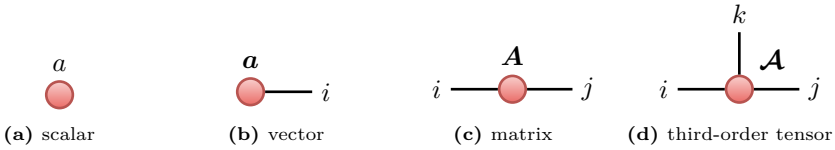


Figure 5.5 Basic symbols for tensor diagrams. (a) scalar, (b) vector, (c) matrix, and (d) third-order tensor.

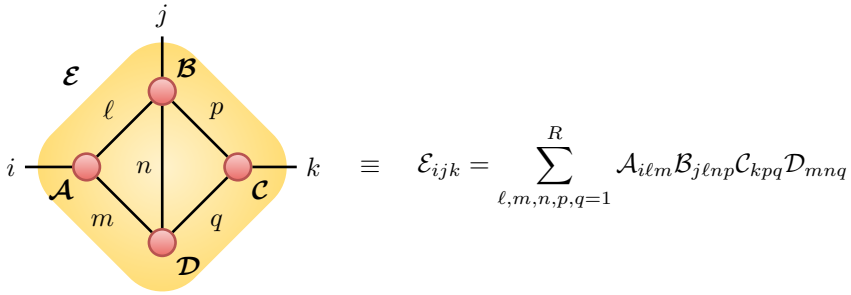
Upon reaching this point, it is beneficial to introduce a diagrammatic notation for tensors and TNs, referred to as *tensor diagrams*. As illustrated in Figure 5.5, these diagrams symbolize tensors as nodes and their indices as edges extending from the nodes. For instance, a node with one edge represents a vector, a node with two edges denotes a matrix, and a node with three edges denotes a tensor. A TN is consequently represented by a network of nodes interconnected by edges. The edges connecting certain tensors correspond to contracted indices, while *dangling edges*, which do not link one tensor to another, represent open indices within the TN.

Tensor diagrams simplify calculations involving TNs. As an illustration, the contractions in equation (5.16) can be represented by the tensor diagram in Figure 5.6a. As another example of complex calculations, the trace of the product of six matrices can be represented by a simple ring-like diagram as seen in Figure 5.6b. From this tensor diagram, the cyclic property of the trace becomes clear. This is a simple example of why tensor diagrams are advantageous. Unlike conventional equations, tensor diagrams visually handle complex expressions, making properties like the cyclic property of the trace of a matrix product more apparent. Indeed, understanding through tensor diagrams is often more intuitive and visually engaging than processing lengthy equations. As such, we will frequently employ tensor diagrams to represent tensor operations and TNs.

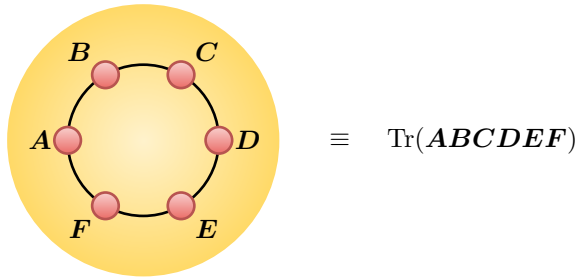
Many matrix and tensor operations perform some form of TN contraction, and hence, can be represented graphically using tensor diagrams. Figure 5.7 illustrates the tensor diagrams of some basic operations.

5.2.2 Basic Tensor Operations

Definition 5.2 (Fiber). A mode- m fiber of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ is a vector obtained by fixing every index except the m th, i.e., $\mathcal{A}_{i_1 \dots i_{m-1} : i_{m+1} \dots i_M}$.



(a)



(b)

Figure 5.6 Tensor diagram for graphical representation of complex tensor contractions. (a) an example of the complex TN. (b) trace of the product of six matrices.

Definition 5.3 (Matricization). *The mode- m matricization (or unfolding) of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ is defined as the matrix $\mathbf{A}_{(m)} \in \mathbb{R}^{I_m \times (I_1 \dots I_{m-1} : I_{m+1} \dots I_M)}$ obtained by stacking all mode- m fibers of \mathcal{A} along the columns, arranged in lexicographical order.*

Definition 5.4 (Element-wise Product). *The element-wise product (or Hadamard product) of two equally-sized tensors $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ and $\mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ is the tensor $\mathcal{C} = \mathcal{A} \odot \mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ with elements*

$$c_{i_1 i_2 \dots i_M} = \mathcal{A}_{i_1 i_2 \dots i_M} \mathcal{B}_{i_1 i_2 \dots i_M} \tag{5.17}$$

for all indices.

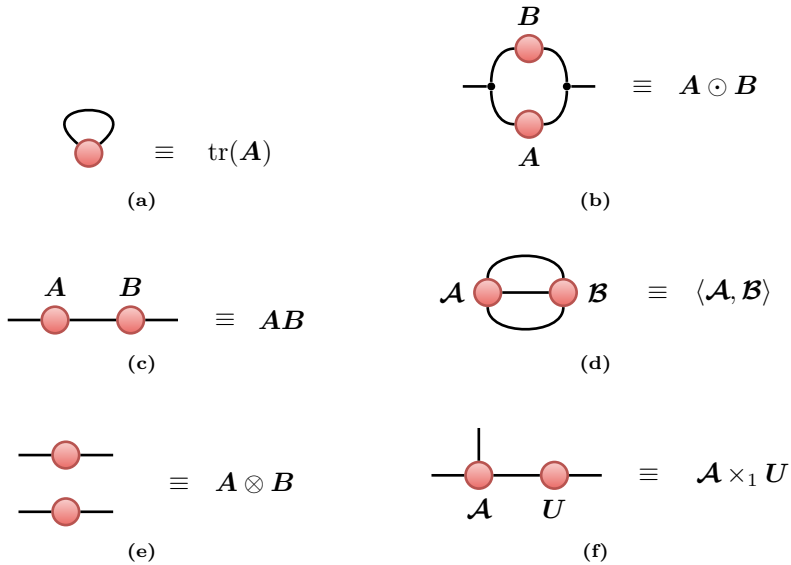


Figure 5.7 Tensor diagram of some basic tensor operations. (a) trace of a matrix, (b) element-wise (Hadamard) of matrices, (c) matrix product, (d) dot product of two 3rd-order tensors, (e) outer product of two matrices, and (f) tensor-matrix product.

Figure 5.7b illustrates the tensor diagram for the element-wise product of two matrices.

Definition 5.5 (Inner Product). *The inner product of two equally-sized tensors $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ and $\mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ is the scalar*

$$\langle \mathcal{A}, \mathcal{B} \rangle \triangleq \sum_{i_1, i_2, \dots, i_M} \mathcal{A}_{i_1 i_2 \dots i_M} \mathcal{B}_{i_1 i_2 \dots i_M}. \quad (5.18)$$

Figure 5.7d illustrates the tensor diagram for the inner product of two third-order tensors.

Definition 5.6 (Outer Product). *The outer product (or tensor product) of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ and a tensor $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times \dots \times J_N}$ is the tensor $\mathcal{C} = \mathcal{A} \otimes \mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M \times J_1 \times J_2 \times \dots \times J_N}$ with elements*

$$C_{i_1 i_2 \dots i_M j_1 j_2 \dots j_N} = A_{i_1 i_2 \dots i_M} B_{j_1 j_2 \dots j_N} \tag{5.19}$$

for all indices.

Figure 5.7e illustrates the tensor diagram for the outer product of two matrices. Given two nonzero vectors $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$, the outer product $\mathbf{a} \otimes \mathbf{b}$ is an $I \times J$ matrix that always has rank 1. Indeed, the columns of the outer product are all proportional to the first column. Therefore, they are all linearly dependent on that one column, hence the matrix is rank-1. In general, the outer product of M nonzero vectors $\mathbf{a}^{(1)} \in \mathbb{R}^{I_m}$, $m = 1, \dots, M$, gives a rank-1 tensor:

$$\mathcal{A} = \mathbf{a}^{(1)} \otimes \mathbf{a}^{(2)} \otimes \dots \otimes \mathbf{a}^{(M)}. \tag{5.20}$$

Definition 5.7 (Tensor-Matrix Product). *The mode- m product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ and a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_m}$ is the tensor*

$$\mathcal{C} = \mathcal{A} \times_m \mathbf{U} \in \mathbb{R}^{I_1 \times \dots \times I_{m-1} \times J \times I_{m+1} \times \dots \times I_M} \tag{5.21}$$

with elements $C_{i_1 i_2 \dots i_{m-1} j i_{m+1} i_M} = \sum_{i_m=1}^{I_m} A_{i_1 i_2 \dots i_M} U_{j i_m}$.

Figure 5.7f illustrates the tensor diagram representation for the mode-1 product of a third-order tensor and a matrix. An equivalent matrix representation of the mode- m product is $\mathcal{C}_{(m)} = \mathbf{U} \mathbf{A}_{(m)}$, which allows us to employ established fast matrix-by-vector and matrix-by-matrix multiplications when dealing with very large-scale tensors.

5.2.3 Contraction Order

The total number of operations required to compute the final result of a TN contraction is heavily affected by the order in which the indices within the TN are contracted, as exemplified in Figure 5.8. In Figure 5.8a, the matrix \mathbf{A} and the tensor \mathcal{C} are contracted first (mode- m product), then the resulting tensor is contracted with the matrix \mathbf{B} . This process incurs $\mathcal{O}(R^4)$ operations overall. On the other hand, In Figure 5.8b, matrices \mathbf{A} and \mathbf{B} are contracted (multiplied) first, yielding a matrix that is then contracted with tensor \mathcal{C} . This process incurs a total of $\mathcal{O}(R^3)$ operations.

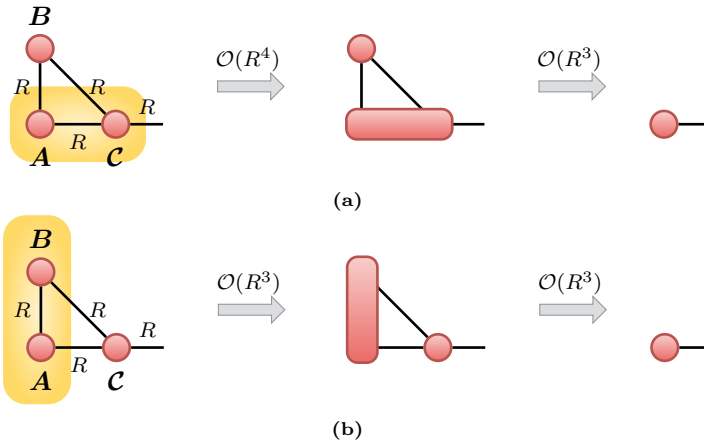


Figure 5.8 Sequential contraction of a tensor network. All dimensions are of size R . In (a), the matrix A and the tensor C are first contracted (mode- m product), then the resulting tensor is contracted with the matrix B , requiring $\mathcal{O}(R^4)$ operations in total. In (b), the matrices A and B are first contracted (multiplied), then the resulting matrix is contracted with the tensor C , requiring $\mathcal{O}(R^3)$ operations in total.

This aspect is significant, particularly in TN methods that involve numerous contractions. The goal is to perform the full contraction as efficiently as possible, which necessitates identifying the optimal contraction order within a TN. This step is particularly critical when it comes to implementing these methods for large-scale applications. Minimizing the computational cost of a TN contraction involves optimizing over the different possible orderings of pairwise contractions to find the optimal case. While this task poses substantial mathematical challenges, practical instances often allow for solutions via simple inspection.

5.2.4 Tensor Decomposition Formats

Definition 5.8 (Canonical Polyadic Decomposition). Canonical polyadic (CP) decomposition [104], [105] factorizes a higher-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ into a finite sum of rank-1 tensors:

$$\mathcal{X} \approx \sum_{r=1}^R \mathbf{u}_r^{(1)} \otimes \mathbf{u}_r^{(2)} \otimes \dots \otimes \mathbf{u}_r^{(M)}, \quad (5.22)$$

where $\mathbf{u}_r^{(m)} \in \mathbb{R}^{I_m}$, and R , referred to as the CP rank, is the minimal number of rank-1 tensors. The matrix $\mathbf{U}^{(m)} = [\mathbf{u}_1^{(m)} \mid \mathbf{u}_2^{(m)} \mid \dots \mid \mathbf{u}_R^{(m)}] \in \mathbb{R}^{I_m \times R}$ is the factor matrix corresponding to the m th mode.

From the definition of the outer product, the CP format can be expressed element-wise as

$$\mathcal{X}_{i_1 i_2 \dots i_M} \approx \sum_{r=1}^R \prod_{m=1}^M U_{i_m r}^{(m)}, \quad (5.23)$$

which aids in obtaining a tensor diagram for the CP decomposition, illustrated in Figure 5.9a.

Definition 5.9 (Tucker Decomposition). Tucker decomposition [106] factorizes a higher-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ into a core tensor multiplied by factor matrices along all modes, that is

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \dots \times_M \mathbf{U}^{(M)}, \quad (5.24)$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_M}$ is the core tensor, $\mathbf{U}^{(m)} \in \mathbb{R}^{I_m \times R_m}$ is the factor matrix corresponding to the m th mode, and (R_1, R_2, \dots, R_M) are referred to as Tucker ranks.

The Tucker decomposition can be represented element-wise as

$$\mathcal{X}_{i_1 i_2 \dots i_M} \approx \sum_{r_1=1}^{R_1} \dots \sum_{r_M=1}^{R_M} \mathcal{G}_{r_1 r_2 \dots r_M} \prod_{m=1}^M U_{i_m r_m}^{(m)}, \quad (5.25)$$

by which we can obtain a tensor diagram for the Tucker decomposition, illustrated in Figure 5.9b. The Tucker decomposition can be also represented as a sum of rank-1 tensors:

$$\mathcal{X} \approx \sum_{r_1=1}^{R_1} \dots \sum_{r_M=1}^{R_M} \mathcal{G}_{r_1 r_2 \dots r_M} \mathbf{u}_{r_1}^{(1)} \otimes \mathbf{u}_{r_2}^{(2)} \otimes \dots \otimes \mathbf{u}_{r_M}^{(M)}, \quad (5.26)$$

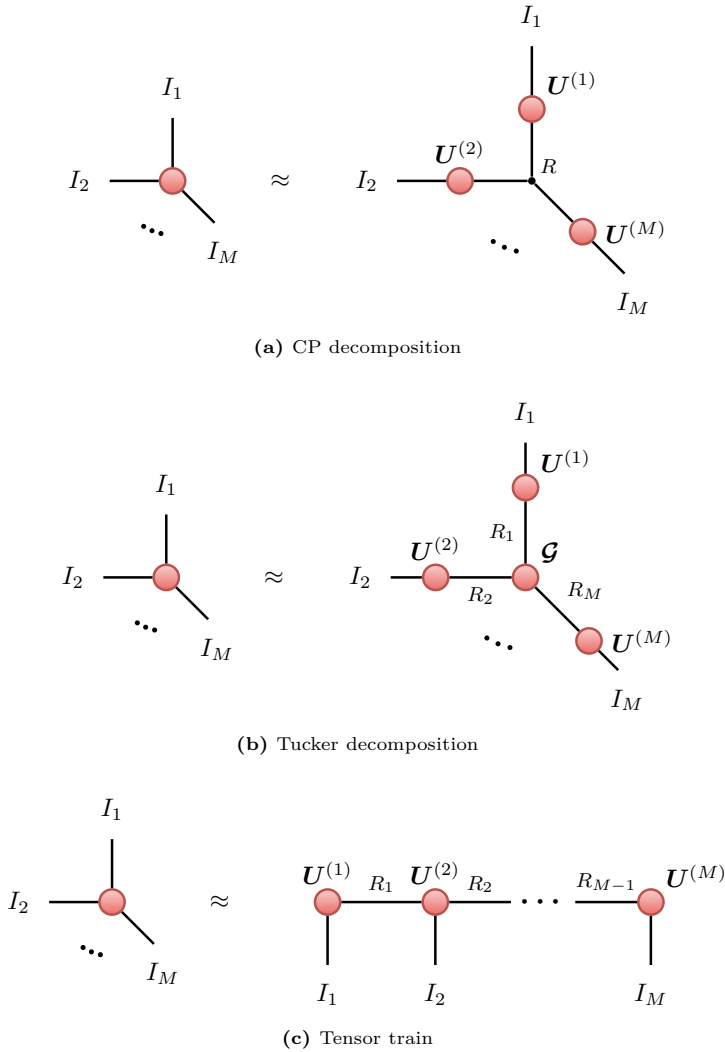


Figure 5.9 Tensor diagrams of some popular tensor decompositions. (a) canonical polyadic decomposition, (b) Tucker decomposition, and (c) tensor train decomposition.

where $\mathbf{u}_{r_m}^{(m)}$ is the r_m th column of the factor matrix $\mathbf{U}^{(m)}$. The above equation reveals that the core tensor \mathcal{G} models the interactions between the different components of the original tensor.

Definition 5.10 (Tensor Train Decomposition). Tensor train (TT) decomposition [107], also known as the matrix product state in quantum mechanics, factorizes a higher-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$ into a sequence of third-order core tensors. Formally, it is expressed as

$$\mathcal{X}_{i_1 i_2 \dots i_M} \approx \mathbf{G}_{i_1}^{(1)} \mathbf{G}_{i_2}^{(2)} \dots \mathbf{G}_{i_M}^{(M)}, \tag{5.27}$$

$$= \sum_{r_1=1}^{R_1} \dots \sum_{r_{M-1}=1}^{R_{M-1}} \prod_{m=1}^M \mathcal{G}_{r_{m-1}, i_m, r_m}^{(m)} \tag{5.28}$$

where each $\mathbf{G}_{i_m}^{(m)} = \mathcal{G}(:, i_m, :)$ is a slice matrix of the corresponding third-order core tensor $\mathcal{G}^{(m)} \in \mathbb{R}^{R_{m-1} \times I_m \times R_m}$, $(R_1, R_2, \dots, R_{M-1})$ are referred to as TT ranks, and $R_0 = R_M = 1$, which means $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(M)}$ are actually two matrices.

Figure 5.9c illustrates a tensor diagram for the TT decomposition.

TNs and tensor decompositions provide a theoretical and computational framework for the analysis of large volumes of data that are computationally prohibitive by segregating the data into “relevant” and “irrelevant” information. As such, TN representations often enable the super-compression of huge data to affordable sizes [108]. Our particular focus is on low-rank TN representations, which allow for huge higher-order tensors to be approximated (compressed) by interconnected lower-order core tensors. We elaborate on this point in the following.

Manipulating high-order tensors quickly becomes impractical as the order increases. For an M th-order tensor of size $I \times I \times \dots \times I$, the number of elements, I^M , grows exponentially with the tensor order, M . Even for $I = 2$, and $M = 200$, we end up with $2^{200} \approx 10^{60}$ elements, which exceeds the total number of atoms in the earth. Thus, storing such huge tensors is unfeasible, let alone processing them with standard numerical algorithms. Nevertheless, if a high-order tensor is represented or approximated as a *sparsely connected* TN, i.e., low-rank representation, the TN can be stored and even used to implicitly compute many tensor operations, without having to recover the entire original tensor—an otherwise computationally intractable task.

As an example, consider the computation of the inner product of two M th-order tensors \mathcal{A} and \mathcal{B} of size $I \times I \times \dots \times I$. Without making any assumption about \mathcal{A} and \mathcal{B} , the explicit calculation of the inner product $\langle \mathcal{A}, \mathcal{B} \rangle$ using Equation (5.18) typically involves $\mathcal{O}(I^M)$ operations, which is impossible for a large M . However, when \mathcal{A} and \mathcal{B} are rank-1 tensors, i.e., $\mathcal{A} = \mathbf{a}^{(1)} \otimes \mathbf{a}^{(2)} \otimes \dots \otimes \mathbf{a}^{(M)}$

and $\mathcal{B} = \mathbf{b}^{(1)} \otimes \mathbf{b}^{(2)} \otimes \dots \otimes \mathbf{b}^{(M)}$, the inner product can be expressed as

$$\begin{aligned}
 \langle \mathcal{A}, \mathcal{B} \rangle &= \sum_{i_1, i_2, \dots, i_M=1}^I \mathcal{A}_{i_1 i_2 \dots i_M} \mathcal{B}_{i_1 i_2 \dots i_M} \\
 &= \sum_{i_1, i_2, \dots, i_M=1}^I \left[\prod_{m=1}^M a_{i_m}^{(m)} \right] \left[\prod_{m=1}^M b_{i_m}^{(m)} \right] \\
 &= \prod_{m=1}^M \left[\sum_{i_m=1}^I a_{i_m}^{(m)} b_{i_m}^{(m)} \right] \\
 &= \prod_{m=1}^M \langle \mathbf{a}^{(m)}, \mathbf{b}^{(m)} \rangle,
 \end{aligned}$$

which only requires $\mathcal{O}(MI)$ operations if we first calculate the inner products $\langle \mathbf{a}^{(m)}, \mathbf{b}^{(m)} \rangle$ and then multiply the results. There are numerous practical applications where the computational complexity can be significantly reduced by representing high-order tensors with low-rank TNs, such as CP, Tucker, and TT, and by executing efficient contractions in optimal orders. Examples of such applications of TNs in machine learning will be discussed in the subsequent section.

5.3 Tensor Networks Meet Machine Learning

The TNs methodology is a promising paradigm for analyzing extreme-scale multidimensional data. TNs “super” compression abilities and the distributed way in which they process data enable fruitful integration with machine learning for tackling a wide range of large-scale problems. TNs are used in machine learning in three common ways discussed in the following.

5.3.1 Network Compression and Acceleration

Deep neural networks (DNNs) are characterized by exceedingly high spatial and temporal complexities, resulting from their deeply stacked layers which contain large-scale matrix multiplications. Consequently, training DNNs can require several days and considerable memory, and inference processes are time-consuming. Furthermore, it has been proven that substantial weight redundancy exists within DNNs [109], suggesting the feasibility of compressing DNNs

without compromising their performance. This observation has motivated the development of numerous techniques designed to compress and accelerate large pre-trained DNNs. Among these, low-rank TN factorization stands out, as TNs efficiently approximate original weights using far fewer parameters [5]. Numerous studies in this vein have been conducted, notably regarding the reconstruction of convolutional and linear layers using various tensor decomposition formats [6], [110], [111]. The resulting “tensorial neural networks,” with their compact architectures, can yield improved performance and less redundancy.

For instance, tensor decompositions can be utilized to reduce the number of parameters in Transformers. To understand this, recall that a typical Transformer comprises primarily multi-head self-attention (MHSA) and MLP layers. The parameters of MHSA include \mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V , and \mathbf{W}^O weight matrices, and the main parameters of MLP consist of \mathbf{W}^{in} and \mathbf{W}^{out} weight matrices. Therefore, the number of parameters in a Transformer is mainly determined by its linear transformation matrices, i.e., \mathbf{W}^Q , \mathbf{W}^K , \mathbf{W}^V , \mathbf{W}^O , \mathbf{W}^{in} , and \mathbf{W}^{out} , leading to most compression studies focusing on reducing the parameters of these six matrices. Liu *et al.* [7] proposed decomposing each matrix in a pre-trained Transformer, generating central tensors (which contain the core information) and small auxiliary tensors. They further suggested a tuning strategy to continue training the auxiliary tensors for performance improvement while freezing the central tensor’s weight to preserve the original matrix’s primary information. Tucker-BERT [8] implements Tucker decomposition to the third-order tensor created by stacking all the weight matrices in the pre-trained BERT model. This strategy significantly reduces the number of parameters, allowing for extreme compression while maintaining comparable results.

5.3.2 Lightweight Parametrization

Low-rank TNs can be also used to re-parameterize the weights of a DNN to achieve lightweight models with fewer parameters and lower computational complexity. Unlike the previous approach, this method does not apply TN decompositions to approximate the weights of a pre-trained model. Instead, it leverages low-rank TNs, such as CP and Tucker format, to represent the weights from the beginning before the training. This technique can substantially reduce both training and inference times [112]. Additionally, imposing low-rank constraints on parameters can regularize the model and limit overfitting. Some reports even suggest significant improvements using CNNs with low-rank regularization [113]. In Chapter 8, we present a CNN based on this concept for brain tumor segmentation.

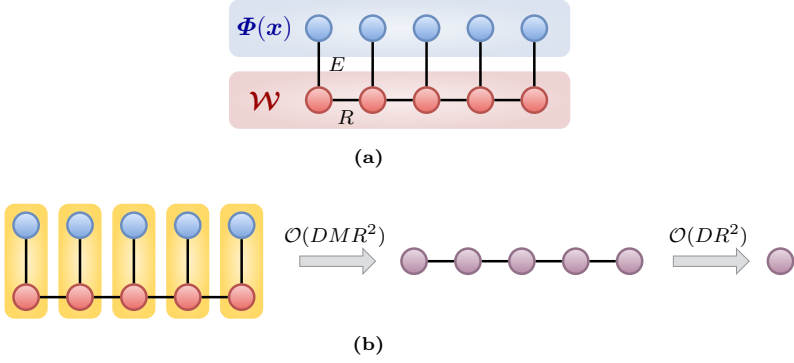


Figure 5.10 Illustration of regression with tensor networks. (a) represents a tensor diagram for the inner product of the input mapped high-dimensional space, Φ , and a weight tensor \mathcal{W} represented by a TT whose ranks are R . (b) represents the order in which the indices are contracted. First, each mapped feature $\phi(x_d)$ is contracted with its adjacent core tensor in the TT, which requires $\mathcal{O}(DER^2)$ operations, then the resulting TT that has no dangling edges is contracted, which requires $\mathcal{O}(DR^2)$ operations. Therefore the overall computational complexity is linear with the dimension D . In contrast, if we first compute $\Phi(\mathbf{x})$ and \mathcal{W} and then the inner product $\langle \mathcal{W}, \Phi(\mathbf{x}) \rangle$, it requires $\mathcal{O}(M^D)$ operations, which scales exponentially with the dimension D and thus becomes computationally intractable as D increases.

5.3.3 Quantum-Inspired Learning

A series of machine learning models based on TNs exists [114]–[116], which are inspired by the successful employment of TNs in quantum mechanics for approximating exponentially large tensors while taking into account the entanglement structure of a quantum system. We sketch this approach in the following.

Consider a regression problem on the training set $\mathbb{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^D$ represents an input vector and $y_n \in \mathbb{R}$ its corresponding output. Let $\mathbf{x} = (x_1, \dots, x_D)$ be a feature vector and $\phi : \mathbb{R} \rightarrow \mathbb{R}^E$ be a multi-valued function. We can use the outer product to form a *rank-1 feature map* to a high-dimensional space as follows [114]:

$$\Phi(\mathbf{x}) = \phi(x_1) \otimes \phi(x_2) \otimes \cdots \otimes \phi(x_D). \quad (5.29)$$

For instance, if $\phi(x) = [1 \ x]^\top$, we obtain the feature map

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ x_2 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} 1 \\ x_D \end{pmatrix},$$

which contains all the possible interactions of features. Another common choice is $\phi(x) = [\cos(\pi x/2), \sin(\pi x/2)]^\top$. We then apply a linear map to the feature $\Phi(\mathbf{x})$ to get a nonlinear prediction function:

$$\mathbf{f}(\mathbf{x}; \mathcal{W}) = \langle \mathcal{W}, \Phi(\mathbf{x}) \rangle \quad (5.30)$$

where \mathcal{W} is a M th-order weight tensor of size $E \times E \times \cdots \times E$. Since \mathcal{W} is a high-order tensor, it becomes astronomically large as M increases. To mitigate this, one strategy is to represent (or re-parametrize) the weight \mathcal{W} as a low-rank TN, which enables us to regularize and optimize this tensor efficiently. For example, [114] proposed to use a TT-like structure for the weight. Figure 5.10a shows a tensor diagram for the inner product $\langle \mathcal{W}, \Phi(\mathbf{x}) \rangle$, assuming a TT structure for \mathcal{W} . As illustrated in Figure 5.10b, we can efficiently compute this inner product $\langle \mathcal{W}, \Phi(\mathbf{x}) \rangle$ with linear complexity in the dimension D by first contracting each mapped feature $\phi(x_d)$ with its adjacent core tensor in the TT and then contracting the TT rank indices. The cost function we would like to minimize could be

$$J(\mathcal{W}^{(1)}, \dots, \mathcal{W}^{(K)}) = \sum_{n=1}^N \left(y_n - \mathbf{f}(\mathbf{x}_n; \mathcal{W}^{(1)}, \dots, \mathcal{W}^{(K)}) \right)^2, \quad (5.31)$$

where $(\mathcal{W}^{(1)}, \dots, \mathcal{W}^{(K)})$ are the core tensors of the TT that we seek to find. Gradient descent-based algorithms [116] or alternating least squares [117] can be used to solve the above problem.

5.4 Conclusion

In this chapter, we examined two prominent low-rank factorization techniques: NMF and tensor networks, exploring their potential applications and integrations in machine learning. NMF is the core of our model introduced in Chapter 7, while tensor networks forms the basis of our approach presented in Chapter 8.

In the first section, we introduced the NMF problem and detailed its key characteristics. We emphasized the ability of NMF factors to provide a compressed representation of a data matrix, encapsulating the most significant patterns within the data. It was observed that NMF factors are often highly

interpretable and capable of revealing hidden data structures in real-world scenarios. We detailed MU and HALS, two popular algorithms for addressing the NMF problem. This part concluded with a discussion on the importance of rank and an illustration of how rank can be effectively selected in practice using the elbow method with the Kneedle algorithm.

In the second section, our focus shifted to tensor networks, highlighting their dual significance both as a computational tool and in forming low-rank representations. The relationship between tensor networks and tensor contractions was explored, illustrating that various tensor operations can be conceptualized as tensor networks. We introduced the diagrammatic notation for tensor networks, emphasizing its importance in simplifying computations involving such networks. We provided an overview of basic tensor operations and tensor decomposition formats from both equational and tensor diagram perspectives. The importance of selecting an optimal contraction order to minimize computational overhead in large-scale applications was underscored. The chapter concluded with an exploration of different ways that low-rank tensor networks can be usefully applied in machine learning, including model compression and acceleration.

Chapter 6

New Multiple Sclerosis Lesion Segmentation and Detection Using Pre-activation U-Net

ABSTRACT | Automated segmentation of new multiple sclerosis (MS) lesions in 3D MRI data is an essential prerequisite for monitoring and quantifying MS progression. Manual delineation of such lesions is time-consuming and expensive, especially because raters need to deal with 3D images and several modalities. In this chapter, we propose Pre-U-Net, a 3D encoder-decoder architecture with pre-activation residual blocks, for the segmentation and detection of new MS lesions. Due to the limited training set and the class imbalance problem, we apply intensive data augmentation and use deep supervision to train our models effectively. Following the same U-shaped architecture but different blocks, Pre-U-Net outperforms U-Net and Res-U-Net on the MSSEG-2 dataset, achieving a Dice score of 40.3% on new lesion segmentation and an F_1 score of 48.1% on new lesion detection. The codes and trained models are publicly available at <https://github.com/pashtari/xunet>.

This chapter is an adapted version of [118] P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marini er, “New Multiple Sclerosis Lesion Segmentation and Detection Using Pre-Activation U-Net,” *Frontiers in Neuroscience*, vol. 16, Oct. 2022, doi: 10.3389/fnins.2022.975862

6.1 Introduction

Multiple sclerosis (MS) is a common chronic, autoimmune demyelinating disease of the central nervous system (CNS), which causes inflammatory lesions in the brain, particularly in white matter (WM). Multiparametric MRI is widely used to diagnose and assess MS lesions in clinical practice. Particularly, FLuid Attenuated Inversion Recovery (FLAIR) images provide high contrast for white matter lesions appearing as high-intensity regions. It is highly relevant to monitor lesion activities, especially the appearance of new lesions and the enlargement of existing lesions, for several purposes, including prognosis and follow-up. More specifically, lesional changes between two longitudinal MRI scans from an MS patient are the most important markers for tracking disease progression and inflammatory changes. To this end, the accurate segmentation of new lesions is an essential prerequisite to quantifying lesional changes and measuring features, such as new lesion volumes and locations. However, manual delineation of such lesions is tedious, time-consuming, and expensive, especially because experts need to deal with 3D images and several modalities; therefore, accurate computer-assisted methods are needed to automatically perform this task.

Longitudinal MS lesion segmentation, however, remains very challenging since MS images often change subtly over time within a patient, and new lesions can be very small although they vary dramatically in shape, structure, and location across patients. The MSSEG-2 MICCAI 2021 challenge [119], [120] aims to develop effective data-driven algorithms for the segmentation of new MS lesions by providing a dataset of 40 pairs of 3D FLAIR images acquired at two different time points (with varying intervals) and registered in the intermediate space between the two time points. For each pair, new lesions are manually annotated by multiple raters, and the consensus ground truths are obtained through a voxel-wise majority voting (see Figure 6.1).

Over the past decade, convolution neural networks (CNNs) with an encoder-decoder architecture, known as U-Net [3], have dominated medical image segmentation. In contrast to a hand-crafted approach, U-Net can automatically learn high-level task-specific features for MS lesion segmentation. This work extends our previous effort [121] in the MSSEG-2 and proposes Pre-U-Net, a 3D U-Net architecture with pre-activation residual blocks [70], [78], for segmenting new MS lesions. We use deep supervision [122] and perform intensive data augmentation to effectively train our models. In contrast to the existing methods, our models directly segment new MS lesions on longitudinal 3D FLAIR images in an end-to-end fashion in contrast to the common two-step approach, where cross-sectional segmentation is first performed individually for each

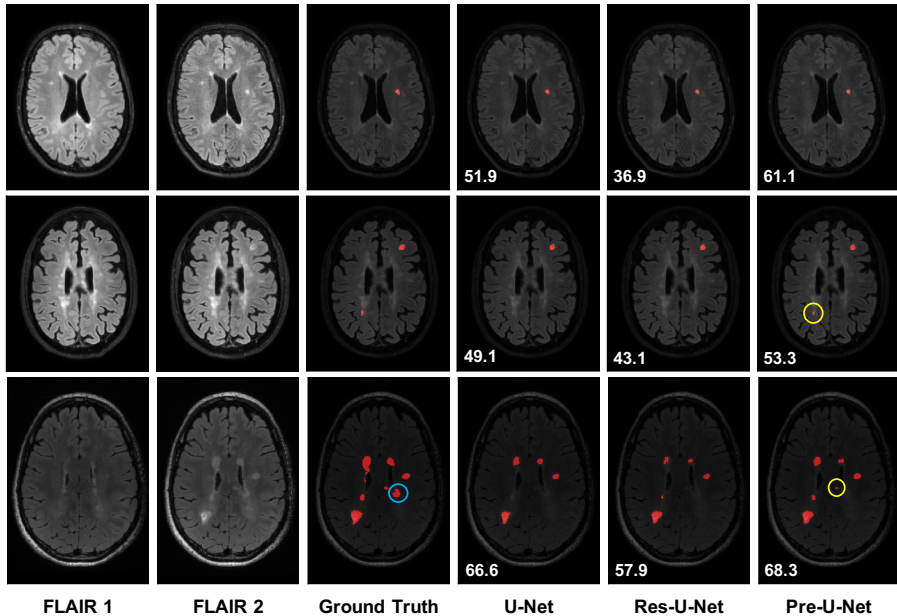


Figure 6.1 Qualitative results on new MS lesion segmentation. The three examples are from three different patients in the test set. The new lesions are shown in red in the segmentation maps. The new lesions circled in yellow (rows 2-3 and column 6) are successfully detected only by Pre-U-Net, while the new lesion circled in blue (row 3 and column 3) is not captured by any of the models, representing a very difficult case. The patient-wise Dice score for each example is displayed on the segmentation map.

time point, and new lesions are then extracted by comparing the longitudinal segmentation maps and applying further post-processing. Depending on the metric used, the MSSEG-2 challenge has four leaderboards. Our Pre-U-Net model achieved competitive scores, and our team, LYLE, was ranked first in two of the leaderboards among 30 participating teams in the challenge.

The rest of this chapter is organized as follows: Section 6.2 briefly reviews relevant semantic segmentation techniques. Section 6.3 presents our approach to longitudinal MS lesion segmentation. Experiments are presented in Section 6.4.2. We conclude this chapter in Section 6.5.

6.2 Related work

Over the past few years, considerable efforts have been made in the development of fully convolutional neural networks for semantic segmentation. Encoder-decoder architectures, in particular U-Net [3] and its variants, are dominant in the segmentation of brain lesions. nnU-Net [93] makes minor modifications to the standard 3D U-Net [85], automatically configuring the key design choices. It has been successfully applied to many medical image segmentation tasks, including longitudinal MS lesion segmentation [94]. McKinley *et al.* [123] proposed an architecture, in which dense blocks [124] of dilated convolutions are embedded in a shallow encoder-decoder network. Myronenko [86] proposed a U-Net-style architecture with a heavier encoder but a lighter decoder for brain tumor segmentation, taking a variational auto-encoder (VAE) approach by adding a branch to the encoder endpoint. Ashtari *et al.* [125] proposed a lightweight CNN for glioma segmentation, with low-rank constraints being imposed on the kernel weights of the convolutional layers in order to reduce overfitting. Aslani *et al.* [126] proposed a deep architecture made up of multiple branches of convolutional encoder-decoder networks that perform slice-based MS lesion segmentation. La Rosa *et al.* [127] proposed a U-Net-like model, to automatically segment cortical and white matter lesions based on 3D FLAIR and MP2RAGE images. These works and most of the MS research in medical imaging have focused on the cross-sectional segmentation of lesions, while only a few efforts have been made to detect and segment new lesions on longitudinal MRI scans. For example, Gessert *et al.* [128] proposed a two-path CNN jointly processing two FLAIR images from two time points to address longitudinal segmentation of new and enlarged lesions. In contrast, this chapter proposes a single-path U-shaped architecture whose input is the 2-channel image constructed simply by concatenating two longitudinal FLAIR images which are co-registered.

6.3 Method

In this section, we describe the proposed encoder-decoder architecture, called Pre-U-Net, and its building blocks.

6.3.1 Overall Architecture

The overall architecture, as shown in Figure 6.2, follows a U-Net-like style made up of encoder and decoder parts. A $3 \times 3 \times 3$ convolution is used as the stem layer. The network takes a 2-channel image of size $128 \times 128 \times 128$ and

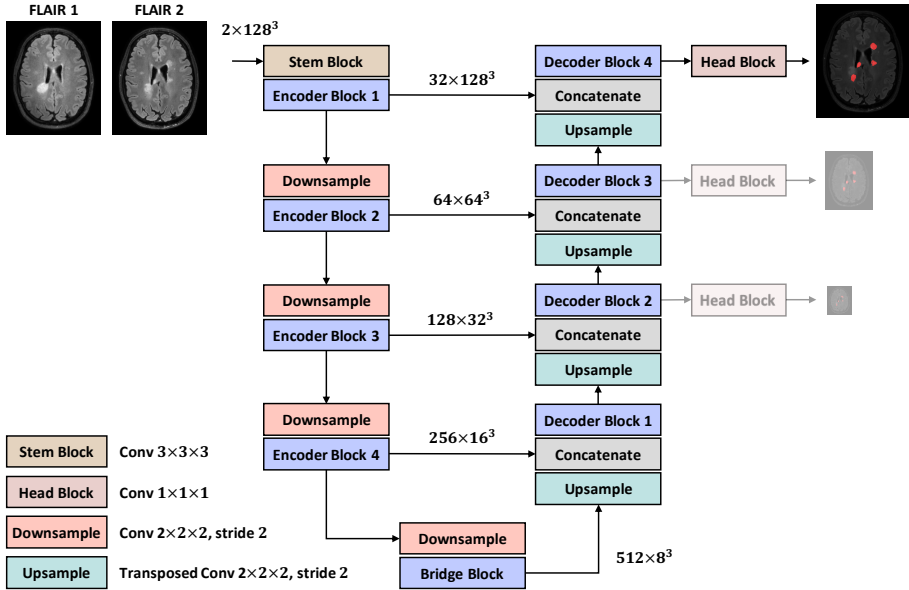


Figure 6.2 The proposed encoder-decoder architecture for new MS lesion segmentation. The two lower-resolution auxiliary maps are only used in the training phase as deep supervisions.

outputs a *probability map* with the same spatial size. The network has 4 levels, at each of which in the encoder (decoder), the input tensor is downsampled (upsampled) by a factor of two while the number of channels is doubled (halved). Downsampling and upsampling are performed via strided convolution and transposed convolution, respectively. The kernel size of all downsamplers and upsamplers is $2 \times 2 \times 2$. We use deep supervision at the three highest resolutions in the decoder, applying pointwise convolutions (head blocks) to get three auxiliary logit tensors.

6.3.2 Baseline Models

Depending on which block is used, we build and compare three baselines: i) U-Net, ii) Res-U-Net, and iii) Pre-U-Net. All these variants follow the same overall architecture as explained in Section 6.3.1 but differ in their encoder/decoder blocks. The block for each model is detailed in the following.

U-Net block. The U-Net block used here is similar to that of nnU-Net [93] except for some minor modifications. As shown in Figure 6.3a, this block is

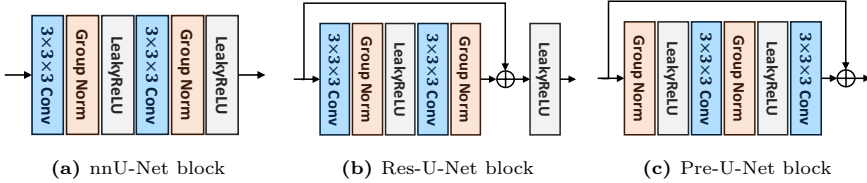


Figure 6.3 The proposed blocks.

composed of two convolutional layers with kernel sizes of $3 \times 3 \times 3$. A Group Normalization [74] layer (with a group size of 8) comes after each convolutional layer and before LeakyReLU activation.

Res-U-Net block. Inspired by the basic ResNet block [70], a Res-U-Net block is, as shown in Figure 6.3b, similar to U-Net block except that a shortcut connection is used between the last Group Normalization layer and the last LeakyReLU activation. A pointwise convolution (i.e. a kernel size of $1 \times 1 \times 1$) may be used in the shortcut connection to match the input dimension with the output dimension of the residual mapping. As investigated by He *et al.* [70], residual connections have been proven effective to avoid vanishing/exploding gradients and speed up the convergence, especially in very deep networks.

Pre-U-Net block. Similar to the pre-activation residual block [78], a Pre-U-Net block consists of two convolutional layers with kernel sizes of $3 \times 3 \times 3$, with LeakyReLU activation coming before each convolutional layer and after Group Normalization (with a group size of 8). Note that the Pre-U-Net block, in contrast to U-Net and Res-U-Net blocks, starts with normalization, applying convolution-activation-normalization in reverse order (see Figure 6.3c). He *et al.* [78] suggest that such a pre-activation design together with identity mappings as the shortcut connections makes information propagate more smoothly than the post-activation design (which is used in the basic ResNet block). Through ablation experiments, they show that the pre-activation design reduces overfitting more significantly, meaning that it leads to slightly higher training loss at convergence but lower test error compared to the post-activation design.

6.4 Experiments

All the models are implemented using PyTorch [129] and PyTorch Lightning [130] frameworks and trained on NVIDIA P100 GPUs. We evaluate the performance of Pre-U-Net for MS lesion segmentation on the MSSEG-2 dataset. We follow

Table 6.1 An overview of the MSSEG-2 dataset. The third column indicates the median value of voxel size for each axis. The fourth column indicates the median number of voxels along each axis.

Data	Modality	Median voxel size (mm)	Median shape	# Total cases	# With-new-lesion cases	# Without-new-lesion cases
Training	FLAIR	(0.53, 0.98, 0.98)	(320, 256, 256)	40	29	11
Test	FLAIR	(0.65, 0.98, 0.98)	(280, 256, 256)	60	32	28
All	FLAIR	(0.60, 0.98, 0.98)	(297, 256, 256)	100	61	39

the same training workflow in all the experiments. In the following, we first provide the details of this workflow, then present the evaluation protocol and the results.

6.4.1 Setup.

Data. A total of 40 and 60 MS patients are represented in the MSSEG-2 training and test set, respectively. For each patient, two longitudinal 3D FLAIR images are acquired at different time intervals (e.g. one year, three years) and registered in the intermediate space between the two time points. New lesions that a patient developed between the two time points were manually delineated by multiple raters, and the consensus ground truths were obtained through a voxel-wise majority voting (see Figure 6.1). The training (test) set includes images that have no new lesions since, in real clinical practice, many patients under treatment do not develop any new lesions during the time interval. Further details on the MSSEG-2 dataset are reported in Table 6.1. Note that both the training and test sets were fixed across our experiments as well as for all the challengers. The instructions to access the MSSEG-2 dataset, used for this study, can be found in the MSSEG-2 challenge site (<https://portal.fli-iam.irisa.fr/msseg-2/>).

Preprocessing. For each case, we first concatenate the two FLAIR images to form a 2-channel 3D image as the input. This is valid since the two FLAIR images are co-registered, and therefore, spatially aligned. The resulting image and its ground truth are then cropped with a minimal box filtering out zero regions. MSSEG-2 data are heterogeneous in the sense that the images may be acquired with different protocols in multiple institutes using different scanners, making intensity values greatly vary across patients and even across time points within the same patient. Therefore, we normalize each image channel-wise using a z-score to have intensities with zero mean and unit variance. Moreover, all the images and their ground truths are then resampled to the same voxel size of 1 mm^3 using trilinear interpolation.

Data Augmentation. To reduce overfitting caused by data insufficiency and heterogeneity, it is crucial to perform an effective data augmentation workflow before feeding the data into the network. During training, the data preprocessing and augmentation are integrated into a single pipeline operating on a batch of 2 samples at each step on the fly. From each sample, we first crop a random $128 \times 128 \times 128$ patch whose center lies within the foreground (i.e., new lesions) with a probability of 66%. Such an oversampling technique ensures that at least 66% of the patches contain some lesion, which in turn alleviates the class imbalance problem caused by the relatively small size of new lesions. The patches then undergo spatial transforms, including random affine and random flip along each spatial dimension, and intensity transforms, including random additive Gaussian noise, random Gaussian smoothing, random intensity scaling and shifting, random bias field, and random contrast adjusting. All the preprocessing operations and augmentation transforms are computed on CPU using the MONAI library [131].

Optimization. All networks are trained for 100000 steps with a batch size of 2 (each patch is processed on one GPU) using AdamW optimizer with an initial learning rate of $1e-5$, weight decay of $1e-2$, and cosine annealing scheduler. Therefore, each network in training is fed by a total of 200000 different patches of size $128 \times 128 \times 128$. It is worth mentioning that since the training set consists of 40 subjects, there are $5000 = 200000/40$ patches per subject, among which around $3300 = 5000 \times 0.66$ patches are expected to contain new lesions.

The loss $\mathcal{L}_{\text{total}}$ is computed by incorporating the three deep supervision outputs and the corresponding downsampled ground truths, according to

$$\mathcal{L}_{\text{total}} = \lambda_0 \mathcal{L}(\mathbf{G}_0, \mathbf{P}_0) + \lambda_1 \mathcal{L}(\mathbf{G}_1, \mathbf{P}_1) + \lambda_2 \mathcal{L}(\mathbf{G}_2, \mathbf{P}_2), \quad (6.1)$$

where $\lambda_0 = 1$, $\lambda_1 = 0.5$, and $\lambda_2 = 0.25$; \mathbf{G}_i and \mathbf{P}_i correspond to the deep supervision at resolution $(128/(2^i))^3$; and the loss function $\mathcal{L}(\cdot, \cdot)$ is the sum of soft Dice [132] and Focal loss [133], that is

$$\mathcal{L}(\mathbf{G}, \mathbf{P}) = \mathcal{L}_{\text{Dice}}(\mathbf{G}, \mathbf{P}) + \mathcal{L}_{\text{Focal}}(\mathbf{G}, \mathbf{P}), \quad (6.2)$$

where

$$\begin{aligned} \mathcal{L}_{\text{Dice}}(\mathbf{G}, \mathbf{P}) &= 1 - \frac{2\langle \mathbf{G}, \mathbf{P} \rangle + \epsilon}{\|\mathbf{G}\|^2 + \|\mathbf{P}\|^2 + \epsilon}, \\ \mathcal{L}_{\text{Focal}}(\mathbf{G}, \mathbf{P}) &= -\frac{1}{N} \langle \mathbf{G}, (1 - \mathbf{P})^\gamma \log(\mathbf{P}) \rangle, \end{aligned} \quad (6.3)$$

where $\mathbf{G} \in \{0, 1\}^{J \times N}$ and $\mathbf{P} \in [0, 1]^{J \times N}$ represent the one-hot encoded ground truth and the predicted probability map for each voxel, respectively, with J

denoting the number of segmentation classes and N denoting the number of voxels in the patch. The small constant $\epsilon = 10^{-5}$ is commonly used to smooth the soft Dice loss and avoid division by zero. The focusing parameter $\gamma = 2$ smoothly controls the rate at which well-classified voxels are suppressed in the Focal loss, and $\mathbf{1}$ denotes a $J \times N$ matrix of ones. The Focal loss has proved effective in tackling the class imbalance problem, which is present in the MSSEG-2 training set since the total volume of new lesions is generally much smaller than that of the background, and nearly one-third of the patients have no new lesions.

Inference. A test image in the inference is first subjected to z-score intensity normalization and resampled to a voxel size of 1 mm^3 . The prediction is then made using a sliding window approach with a 50% overlap and a window size of $128 \times 128 \times 128$ (which is equal to the patch size used in training). For a given voxel from overlapping windows, the mean of the predictions is simply taken as the final value (the `SlidingWindowInferer` module from MONAI was used to perform the sliding window inference). The resulting probability map is resampled back to the original voxel size and finally thresholded by 0.5 to obtain a binary segmentation map.

Evaluation. The Dice score and Hausdorff Distance (HD) are used as metrics to assess the performance of segmentation for the patients that have some new lesions in their ground truths. The Dice score measures the voxel-wise overlap between the ground truth and the prediction, defined as

$$\text{Dice}(\mathbf{g}, \mathbf{y}) = \frac{2 \sum_{n=1}^N g_n y_n}{\sum_{n=1}^N g_n + \sum_{n=1}^N y_n}, \quad (6.4)$$

where $g_n \in \{0, 1\}$ and $y_n \in \{0, 1\}$ represent the ground truth and the binary prediction for a voxel, respectively, and N is the number of voxels. Symmetric Hausdorff Distance (HD) evaluates the distance between the boundaries of ground truth and prediction, computed according to

$$\text{HD}(\mathbb{G}, \mathbb{Y}) = \max\left\{\max_{\mathbf{g} \in \mathbb{G}} \min_{\mathbf{y} \in \mathbb{Y}} \|\mathbf{g} - \mathbf{y}\|, \max_{\mathbf{y} \in \mathbb{Y}} \min_{\mathbf{g} \in \mathbb{G}} \|\mathbf{y} - \mathbf{g}\|\right\}, \quad (6.5)$$

where \mathbb{G} and \mathbb{Y} denote the set of all voxels on the surface of ground truth and prediction, respectively.

Lesion-wise sensitivity (SEN), positive predictive value (PPV), and F_1 score are used as metrics to quantify the detection rate of new lesions. Let \mathbf{G} be the ground truth and \mathbf{Y} be the prediction. To compute these lesion level metrics, we follow [26], according to which the connected components of \mathbf{G} and \mathbf{Y} (with a 18-connectivity kernel) are first extracted, and all new lesions smaller than 3

mm³ in size are removed, yielding new tensors $\tilde{\mathbf{G}}$ and $\tilde{\mathbf{Y}}$. The metrics are then defined as

$$\begin{aligned} \text{SEN} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ \text{PPV} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{F}_1 &= \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}, \end{aligned} \tag{6.6}$$

where TP, FP, and FN are the number of true positives, false positives, and false negatives, respectively, in the detection of new lesions (i.e., connected components). The rules by which a lesion is considered detected are explained in [26].

For cases without any new lesions in their ground truths, we use the following two metrics:

- The **N**umber of new **L**esions **P**redicted (NLP) by the algorithm. This is obtained by counting the number of connected components in the predicted segmentation.
- The **V**olume of new **L**esions **P**redicted (VLP) by the algorithm. This is obtained by simply multiplying the number of voxels in the predicted segmentation by the voxel volume.

All the metrics mentioned above were computed using `animaSegPerfAnalyzer` from the Anima toolbox (available at <https://anima.irisa.fr/>, RRID: SCR_017017 and RRID: SCR_01707).

6.4.2 Results and Discussion

Quantitative Evaluation. We performed 5-fold cross-validation in all the experiments to estimate how capable our models are in generalizing to unseen data. The cross-validation results on the MSSEG-2 training set are reported in Table 6.2. For each network, we used an ensemble of the five models trained during the cross-validation on the training set for predicting the test set labels. The test results are reported in Table 6.3 and illustrated by notched box plots in Figure 6.4, where pairwise Wilcoxon signed-rank tests were used to identify the significant differences in the test scores of baselines.

Pre-U-Net was superior to all the other models in terms of both segmentation and detection performance for the test cases with some new lesions, achieving a Dice

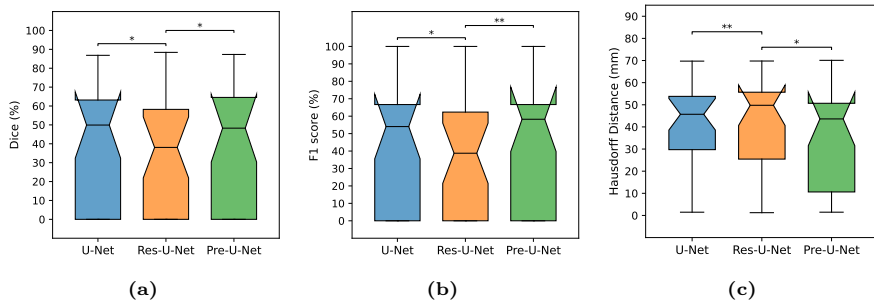


Figure 6.4 Comparison of different models on the MSSEG-2 test set. (a), (b), and (c) show box plots of Dice score (%), F_1 score (%), and Hausdorff Distance (mm), respectively. The asterisks indicate how significantly a model score differs from those of the other baselines when using a pairwise Wilcoxon signed-rank test (*: p-value < 0.05, **: p-value < 0.01).

Table 6.2 Results obtained by 5-fold cross-validation on the MSSEG-2 training set. Symbols \uparrow and \downarrow indicate that a metric is desired to be higher and lower, respectively. The mean and standard deviation (SD) of a score across the folds are reported as “mean (SD)”. The best results are in **boldface**.

Model	#Params	FLOPs	With-new-lesion cases					Without-new-lesion cases	
			Dice (%) \uparrow	HD (mm) \downarrow	SEN (%) \uparrow	PPV (%) \uparrow	F_1 (%) \uparrow	NLP \downarrow	VLP (mm ³) \downarrow
U-Net	28.7M	1264.7G	45.2 (5.5)	39.0 (14.1)	51.0 (12.1)	52.9 (6.1)	48.9 (7.6)	0.1 (0.2)	9.2 (20.6)
Res-U-Net	28.9M	1280.8G	42.4 (11.4)	46.4 (15.9)	49.3 (22.9)	60.6 (6.6)	49.9 (14.8)	0.2 (0.4)	4.0 (9.0)
Pre-U-Net	28.9M	1280.8G	45.6 (9.5)	40.1 (13.2)	54.5 (13.8)	53.8 (6.8)	51.9 (11.3)	0.0 (0.0)	0.0 (0.0)

score of 40.3%, HD of 35.0, SEN of 47.5%, PPV of 53.6%, and F_1 score of 48.1%. While having almost the same number of parameters and the same computational complexity (FLOPs), Pre-U-Net outperformed U-Net, the second-best baseline, and significantly outperformed Res-U-Net, with p-value < 0.05 for the Dice score, p-value < 0.01 for the F_1 score, and p-value < 0.05 for HD. Overall, Pre-U-Net proved more effective than the other models at segmentation and detecting new lesions. Nevertheless, note that Pre-U-Net was only marginally superior to U-Net, and there was no statistically significant difference between the two models in terms of the segmentation or detection metrics.

Res-U-Net, with an NLP of 0.0 and VLP of 0.0, performed slightly better for the test cases that have no new lesions whereas Pre-U-Net is the winner in terms of validation scores. In fact, the differences in NLP and VLP scores are marginal, and all of our models are sufficiently accurate to detect no lesions (i.e., produce a segmentation map in which all elements are zero) for patients without any new lesions. Our team, LYLE, with the Pre-U-Net model [121] was ranked first

Table 6.3 Results on the MSSEG-2 test set. Predictions were made using the 5 models from the cross-validation as an ensemble. Symbols \uparrow and \downarrow indicate that a metric is desired to be higher and lower, respectively. The mean and standard deviation (SD) of a score across patients are reported as “mean (SD)”. The best results are in **boldface**.

Model	#Params	FLOPs	With-new-lesion cases					Without-new-lesion cases	
			Dice (%) \uparrow	HD (mm) \downarrow	SEN (%) \uparrow	PPV (%) \uparrow	F ₁ (%) \uparrow	NLP \downarrow	VLP (mm ³) \downarrow
U-Net	28.7M	1264.7G	38.9 (31.1)	43.1 (27.3)	45.2 (36.8)	51.2 (39.6)	45.3 (35.7)	0.0 (0.2)	0.4 (2.3)
Res-U-Net	28.9M	1280.8G	34.9 (29.5)	44.2 (29.0)	43.6 (38.4)	38.4 (38.5)	33.7 (33.1)	0.0 (0.0)	0.0 (0.0)
Pre-U-Net	28.9M	1280.8G	40.3 (30.5)	35.0 (22.3)	47.5 (37.9)	53.6 (38.3)	48.1 (34.8)	0.0 (0.2)	0.5 (2.5)

in the MSSEG-2 challenge in the two leaderboards based on the NLP and VLP metrics. All the four leaderboards (based on Dice, F₁ score, NLP, and VLP metrics) and the patient-wise scores for each participating team can be found on <https://portal.fli-iam.irisa.fr/msseg-2/challenge-day/>.

Qualitative Evaluation. Figure 6.1 presents qualitative comparisons of baselines. The top row exemplifies a patient with a single lesion that is detected by all the models. However, Pre-U-Net, with a patient-wise Dice score of 61.1%, yields a lesion that overlaps most with the lesion in the ground truth compared to U-Net with a patient-wise Dice score of 51.9% and Res-U-Net with a patient-wise Dice score of 36.9%.

Moreover, Pre-U-Net demonstrates superior performance in detecting new lesions. This capability is evidenced in the middle and bottom rows, where Pre-U-Net detects the two new lesions circled in yellow whereas U-Net and Res-U-Net fail to capture them. Note that as observed in the bottom row, Pre-U-Net, with a patient-wise Dice score of 68.3%, shows only a slight improvement in the segmentation performance over U-Net, with a patient-wise Dice score of 66.6%; however, Pre-U-Net indeed outperforms U-Net significantly when it comes to new lesion detection. Nevertheless, some new lesions are extremely challenging to detect even for experts, and all the models fail to capture them. For example, the lesion circled in blue on the ground truth (row 3 and column 3 in Figure 6.1) is detected by none of the models including Pre-U-Net.

Future work aims at improving new MS lesion detection, especially in the presence of such difficult lesions. This might include, for instance, incorporating the individual delineations of raters into our models. Indeed, in cases where there is more uncertainty due to a weaker consensus among raters (e.g., three raters delineated a set of voxels differently than the other one), our models are also more likely to result in false predictions. Moreover, we will investigate the possibility of transfer learning from a simpler lesion segmentation task with a

bigger dataset for further tackling the data insufficiency and class imbalance problems faced in this work.

6.5 Conclusion

We devised a U-Net-like architecture consisting of pre-activation blocks, called Pre-U-Net, for longitudinal MS lesion segmentation. We successfully trained our models by using data augmentation and deep supervision, alleviating the problem of data insufficiency and class imbalance. The effectiveness of Pre-U-Net was evaluated in segmenting and detecting new white matter lesions in 3D FLAIR images on the MSSEG-2 dataset. Pre-U-Net achieved a Dice score of 40.3% and F_1 score of 48.1%, outperforming the baselines, U-Net and Res-U-Net. In particular, Pre-U-Net is, as reflected by F_1 scores, more effective than the baselines at detecting new lesions, and it is competitive with U-Net in terms of segmentation performance, as evidenced by Dice and HD scores.

Chapter 7

Factorizer: A Scalable Interpretable Approach to Context Modeling for Medical Image Segmentation

ABSTRACT | Convolutional Neural Networks (CNNs) with U-shaped architectures have dominated medical image segmentation, which is crucial for various clinical purposes. However, the inherent locality of convolution makes CNNs fail to fully exploit global context, essential for better recognition of some structures, e.g., brain lesions. Transformers have recently proven promising performance on vision tasks, including semantic segmentation, mainly due to their capability of modeling long-range dependencies. Nevertheless, the quadratic complexity of attention makes existing Transformer-based models use self-attention layers only after somehow reducing the image resolution, which limits the ability to capture global contexts present at higher resolutions. Therefore, this work introduces a family of models, dubbed Factorizer, which leverages the power of low-rank matrix factorization for constructing an end-to-end segmentation model. Specifically, we propose a linearly scalable approach

This chapter is an adapted version of [134] P. Ashtari, D. M. Sima, L. De Lathauwer, D. Sappéy-Marinier, F. Maes, and S. Van Huffel, “Factorizer: A Scalable Interpretable Approach to Context Modeling for Medical Image Segmentation,” *Medical Image Analysis*, vol. 84, p. 102706, Feb. 2023, doi: 10.1016/j.media.2022.102706

to context modeling, formulating Nonnegative Matrix Factorization (NMF) as a differentiable layer integrated into a U-shaped architecture. The shifted window technique is also utilized in combination with NMF to effectively aggregate local information. Factorizers compete favorably with CNNs and Transformers in terms of accuracy, scalability, and interpretability, achieving state-of-the-art results on the BraTS dataset for brain tumor segmentation and ISLES'22 dataset for stroke lesion segmentation. Highly meaningful NMF components give an additional interpretability advantage to Factorizers over CNNs and Transformers. Moreover, our ablation studies reveal a distinctive feature of Factorizers that enables a significant speed-up in inference for a trained Factorizer without any extra steps and without sacrificing much accuracy. The code and models are publicly available at <https://github.com/pashtari/factorizer>.

7.1 Introduction

Medical image segmentation is an essential prerequisite for the analysis of anatomical structures for various clinical purposes, including diagnosis and treatment planning. In recent years, the vast majority of effective segmentation models are based on Convolutional Neural Networks (CNNs), particularly, U-Net [3], consisting of encoder and decoder parts with skip connections in between. In a typical U-Net [3], [85], the encoder learns a low-resolution contextual representation consisting of progressively downsampled feature maps, while the decoder progressively upsamples the low-resolution feature maps to propagate contextual information to the higher-resolution layers. Moreover, skip connections between encoder and decoder layers of equal resolution help to recover spatial information lost during downsampling.

Currently, U-Net models mostly rely on convolution operations with small receptive fields, which are capable of exploiting only local context at each resolution. Hence, they generally fail to effectively model long-range spatial dependencies, often necessary, for example, for better recognition of brain lesions, which can be very infiltrative, extensive, and thus dramatically vary in shape and size. Moreover, capturing even small focal tumors within the receptive field is extremely difficult without any notion about the global context of normal brain anatomy since such tumors can occur anywhere in the brain. Several works [135], [136] have employed dilated convolution for expanding the receptive fields. Nevertheless, the learning capabilities of convolutional layers are still limited due to their inherent locality. As a solution, integrating self-attention modules into CNNs [137], [138] has been proposed to enhance the capability of modeling non-local context.

Transformers [79] have achieved state-of-the-art performance on various natural language processing tasks. The attention mechanism enables Transformers to effectively model the pairwise interactions between the words in a sentence. Recently, Transformer-based models have been applied to vision tasks and demonstrated promising results. Specifically, Vision Transformer (ViT) [80] outperformed state-of-the-art CNNs on image recognition by large-scale pre-training and fine-tuning a pure Transformer. Unlike CNNs, ViT encodes images as a sequence of 1D patch embeddings (known as tokens) and dynamically highlights the important tokens using self-attention layers, which in turn increases the capability of learning long-range dependencies. Due to lack of locality inductive bias, ViT is data-hungry and generally requires a larger dataset to perform as effectively as its CNN counterparts, leading to poor performance when trained on insufficient data, which is usually the case in medical imaging. Furthermore, the quadratic complexity of self-attention

makes Transformers computationally intractable on long sequences of patches. Therefore, existing models use self-attention layers only after somehow reducing the image resolution, thereby failing to fully exploit the global context at the higher resolution.

This work proposes a family of architectures, dubbed as Factorizer, which leverages the power of low-rank matrix approximation (LRMA) to construct an end-to-end medical image segmentation model. Among LRMA methods, Nonnegative Matrix Factorization (NMF) has demonstrated a remarkable ability to compress data and automatically extract easy-to-interpret sparse factors [139], [140]. Hence, we propose a linearly scalable alternative to self-attention by formulating an NMF algorithm as a differentiable layer. Moreover, a series of matricization operations is introduced, which enables NMF to effectively exploit both global and local contexts. The Factorizer block is constructed by replacing the self-attention layer of a ViT block with our NMF-based modules and then integrated into a U-shaped architecture with skip connections.

We evaluated the effectiveness of our approach for the segmentation of brain tumors and stroke lesions in MRI data. Factorizers achieved competitive results on the BraTS [25], [141] and ISLES'22 [18] datasets, having outperformed state-of-the-art methods based on CNN and Transformer. Our experiments showed that NMF components are highly meaningful, which gives a great advantage to Factorizers over CNNs and Transformers in terms of interpretability. Furthermore, our ablation studies revealed a distinctive interesting feature of Factorizers that enables us to easily speed up the inference for a trained Factorizer model with no extra steps and without sacrificing much accuracy.

Contributions. The main contributions of this work are as follows:

- To the best of our knowledge, this work presents the first end-to-end deep model with matrix factorization layers for medical image segmentation.
- A differentiable NMF layer is constructed using a block coordinate descent solver to efficiently model contextual information.
- Shifted Window (SW) Matricize operation is introduced and combined with NMF to fully exploit local contexts.
- Scalable interpretable U-shaped segmentation models based on NMF are proposed.
- The proposed models achieve state-of-the-art results on the BraTS and ISLES'22 datasets.

Notation. We denote vectors by boldface lower-case letters, e.g., \mathbf{x} , matrices by boldface upper-case letters, e.g., \mathbf{X} , and tensors by boldface calligraphic letters, e.g., \mathcal{X} . Elements in a matrix (tensor) are denoted by $X_{i,j}$ ($\mathcal{X}_{i_1,\dots,i_N}$). The i th row and j th column of a matrix is denoted by $X_{i,:}$ and $X_{:,j}$, respectively. A sequence of N vectors (or *tokens*) is denoted by $(\mathbf{x}_n)_{n=1}^N$. We use $[\mathbf{x}_1 \mid \dots \mid \mathbf{x}_N]$ to denote a matrix \mathbf{X} created by stacking \mathbf{x}_n s along the columns. We show the inner product between matrices by $\langle \mathbf{X}, \mathbf{Y} \rangle \triangleq \sum_{i,j} X_{i,j} Y_{i,j}$ and L^2 norm of a matrix by $\|\mathbf{X}\| \triangleq \sqrt{\langle \mathbf{X}, \mathbf{X} \rangle}$.

7.2 Related Work

CNN-based Segmentation Models. Convolutional neural networks (CNNs) have dominated medical image segmentation. Particularly, following an encoder-decoder architecture with skip connections, U-Net [3] has achieved state-of-the-art on various medical image datasets. The simplicity and effectiveness of a U-shaped architecture have led to the emergence of numerous U-Net variants in the field. Çiçek *et al.* [85] extended U-Net by replacing all 2D operations with their 3D counterparts. UNet++ [142] follows a deeply-supervised encoder-decoder network consisting of sub-networks connected through a series of nested, dense skip connections. nnU-Net [94] proved effective in various medical image segmentation tasks by only making minor modifications to the standard 3D U-Net [85] and defining a recipe to automatically configure key design choices. Myronenko [86] proposed a U-Net-like architecture with ResNet blocks, aka ResSegNet, which ranked first in the Brain Tumor Segmentation Challenge (BraTS) 2018. Ashtari *et al.* [125] proposed a lightweight CNN for glioma segmentation, with low-rank constraints being imposed on the kernel weights of the convolutional layers in order to reduce overfitting.

Despite their success, these networks generally fail to effectively model long-range spatial dependencies, often necessary for better recognition of some region semantics such as tumors, since they rely on convolution operations with small kernel sizes, aggregating only local information in an image.

Visual Transformers. Transformers with attention mechanisms [79], introduced originally for language modeling, have recently proven promising on computer vision tasks. Particularly, the pioneering Vision Transformer (ViT) model [80] outperformed state-of-the-art CNNs on image recognition by large-scale pre-training and fine-tuning a pure Transformer applied to sequences of image patches. In contrast to CNNs, ViT lacks any inductive bias such as locality, and therefore, generally shows poorer performance than its CNN counterparts (e.g., ResNets [70]) when trained from scratch on small-size or mid-size datasets, which is usually the case for medical imaging.

Efforts have been made to mitigate this limitation. For example, Tokens-to-Token ViT (T2T-ViT) [143] introduces a hierarchical architecture to ViT by progressively combining neighboring tokens into a single token to reduce the sequence length and aggregate local context. Liu *et al.* [81] proposed a hierarchical Transformer, called Swin Transformer, adopting the shifted windowing scheme, which brings more efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection. As another exemplification of a hierarchical Transformer, Pyramid vision Transformer (PvT) [144] significantly reduces computational and memory overhead by reducing the sequence length at each stage through non-overlapping patch embedding and learning low-resolution key-value pairs via spatial-reduction attention (SRA) in each block. Convolutional vision Transformer (CvT) [145] incorporates depthwise convolutions into self-attention layers and uses strided convolution for simultaneously tokenizing and downsampling the image, exploiting the excellent capability of convolution at capturing low-level local features.

Transformer-based methods were recently proposed to deal with the task of 2D image segmentation. Segmentation Transformer (SETR) [146] uses a ViT encoder and a decoder with progressive upsampling (which alternates Conv layers and upsampling operations) and multi-level feature Aggregation. SegFormer [147] consists of a PvT-based encoder and a lightweight Multilayer Perceptron (MLP) decoder with upsampling operations. Chen *et al.* [87] proposed a model for multi-organ segmentation by incorporating ViT into the bridge of a 2D convolutional U-Net architecture. Zhang *et al.* [148] proposed to combine a shallow CNN with a Transformer in a parallel style. Valanarasu *et al.* [149] proposed a Transformer model with an axial attention mechanism for the segmentation of 2D medical images. Cao *et al.* [91] proposed a U-shaped architecture based purely on Swin Transformer.

For 3D medical image segmentation, Xie *et al.* [150] proposed a model comprising a CNN backbone to extract features, a Transformer to model long-range dependencies, and a CNN decoder to construct the segmentation map. More recently, Hatamizadeh *et al.* [89] proposed UNETR, which utilizes ViT as the main encoder but directly connects it to the convolutional decoder via skip connections, as opposed to using a Transformer only in the bridge. nnFormer [92] uses an initial convolutional tokenizer and interleaves local and global self-attention blocks with convolutional downsamplers. Since self-attention is prohibitively expensive on long sequences, all these models apply Transformer on a low-resolution stage after either patch embedding or a CNN backbone, making them fail to fully exploit the global context at the higher resolutions. In contrast, our proposed approach based on NMF offers a scalable alternative to the attention mechanism, which enables the exploitation of the global context

at the highest-resolution stage of a 3D network.

Matrix Factorization Models. In the context of machine learning, low-rank matrix factorization methods have proven extremely useful for representation learning, dimensionality reduction, and collaborative filtering. NMF has been used for unsupervised and semi-automated segmentation of brain tumors on multiparametric MRI data [151], [152]. However, only a few works have incorporated matrix factorization into an end-to-end deep model to perform a computer vision task. Most notably, Geng *et al.* [153] proposed a framework, called Hamburger, where the global context is modeled as solving a low-rank matrix completion problem by suitable optimization algorithms that guide the design of layers able to capture global information. They demonstrated the effectiveness of a Hamburger model based on NMF with a multiplicative update (MU) solver for semantic segmentation. Our approach to context modeling is based on NMF with a Hierarchical Alternating Least Squares (HALS) solver and introduces a series of matricization operations which enable NMF to effectively exploit both global and local contexts. Moreover, our proposed block is inspired by the overall design of the ViT block and incorporated into a U-shaped architecture.

7.3 Method

7.3.1 Matrix Factorization for Context Modeling

Here we provide the motivation behind incorporating matrix factorization into deep learning by first presenting an alternative view of the attention mechanism and then showing how it relates to the matrix factorization approach to modeling contextual information.

Revisiting Attention Mechanism. The attention mechanism is the key component that enables Transformers to model complex dependencies between the elements of a sequence. Consider an input sequence of C -dimensional tokens $(\mathbf{x}_n)_{n=1}^N$, stacked into the rows of matrix $\mathbf{X} = [\mathbf{x}_1 \mid \cdots \mid \mathbf{x}_N]^T$. In a self-attention layer, the input is first projected onto three learnable weight matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{C \times E}$ to get three different matrices: *query* $\mathbf{Q} = [\mathbf{q}_1 \mid \cdots \mid \mathbf{q}_N]^T = \mathbf{X}\mathbf{W}^Q$, *Key* $\mathbf{K} = [\mathbf{k}_1 \mid \cdots \mid \mathbf{k}_N]^T = \mathbf{X}\mathbf{W}^K$, and *value* $\mathbf{V} = [\mathbf{v}_1 \mid \cdots \mid \mathbf{v}_N]^T = \mathbf{X}\mathbf{W}^V$. The output is then defined by

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{E}}\right)\mathbf{V}, \quad (7.1)$$

where softmax is taken row-wise. Note that attention takes $\mathcal{O}(N^2E)$ time and needs $\mathcal{O}(N^2)$ memory to store the attention weights, scaling quadratically with the sequence length, which is prohibitively expensive for large inputs, such as high-resolution or 3D images.

Taking a closer look at equation (7.1), we notice that attention can be interpreted as a special case of the Nadaraya-Watson kernel regression, discussed in Section 3.3.4. To illuminate this relationship, consider the key-value pairs $\mathbb{D} = \{(\mathbf{k}_n, \mathbf{v}_n)\}_{n=1}^N$ as a training dataset and the queries $\{\mathbf{q}_n\}_{n=1}^N$ as a test dataset. Here, the \mathbf{k}_n s and \mathbf{q}_n s are treated as feature vectors, while the \mathbf{v}_n s serve as labels for the keys. Based on equation (3.54), the predictions for the queries using generalized kernel regression can be expressed as:

$$\hat{f}(\mathbf{q}_n) = \frac{\sum_{m=1}^N S(\mathbf{q}_n, \mathbf{k}_m) \mathbf{v}_m}{\sum_{m=1}^N S(\mathbf{q}_n, \mathbf{k}_m)}, \quad (7.2)$$

where $S(.,.)$ is a similarity function. By setting the similarity function to $S(\mathbf{a}, \mathbf{b}) = \exp(\mathbf{a}^\top \mathbf{b} / \sqrt{E})$ and then rewriting it in matrix notation, we can easily derive the attention formula.

Low-Rank Matrix Factorization. Formulating attention as a kernel regression with softmax kernel suggests that we can probably use other regression methods on queries, keys, and values to model interactions between the sequence elements. Notably, one can take a matrix completion approach to regress on the queries via low-rank approximation of a block matrix of queries, keys, and values with missing entries, that is

$$\begin{pmatrix} \mathbf{K} & \mathbf{V} \\ \mathbf{Q} & ? \end{pmatrix} \approx \begin{pmatrix} \mathbf{B} \\ \mathbf{F} \end{pmatrix} \begin{pmatrix} \mathbf{G} \\ \mathbf{H} \end{pmatrix}^\top, \quad (7.3)$$

where $\mathbf{B}, \mathbf{F} \in \mathbb{R}^{N \times R}$ and $\mathbf{G}, \mathbf{H} \in \mathbb{R}^{E \times R}$; and $R \leq \min(N, E)$ is the rank. The reconstructed matrix $\mathbf{F}\mathbf{H}^\top$ then gives an estimation of the missing block as the output. This can be viewed as a joint matrix factorization of \mathbf{Q} , \mathbf{K} , and \mathbf{V} , i.e., $\mathbf{K} \approx \mathbf{B}\mathbf{G}^\top$, $\mathbf{V} \approx \mathbf{B}\mathbf{H}^\top$, $\mathbf{Q} \approx \mathbf{F}\mathbf{G}^\top$.

In this work, we further simplify the procedure by considering only a single linear map to generate a single matrix $\mathbf{Z} = \mathbf{X}\mathbf{W}$, where $\mathbf{W} \in \mathbb{R}^{C \times E}$ is a learnable weight matrix, and using a regular matrix factorization rather than a joint one, that is $\mathbf{Z} \approx \mathbf{F}\mathbf{G}^\top$, then the output is the reconstructed matrix $\mathbf{F}\mathbf{G}^\top$. Note that this is an unsupervised scheme, unlike the joint factorization, which forms a regression model. Depending on the matrix factorization algorithm, a suitable activation function may be applied before the factorization to constrain the input. Particularly, in the case of NMF, ReLU of the matrix must be first

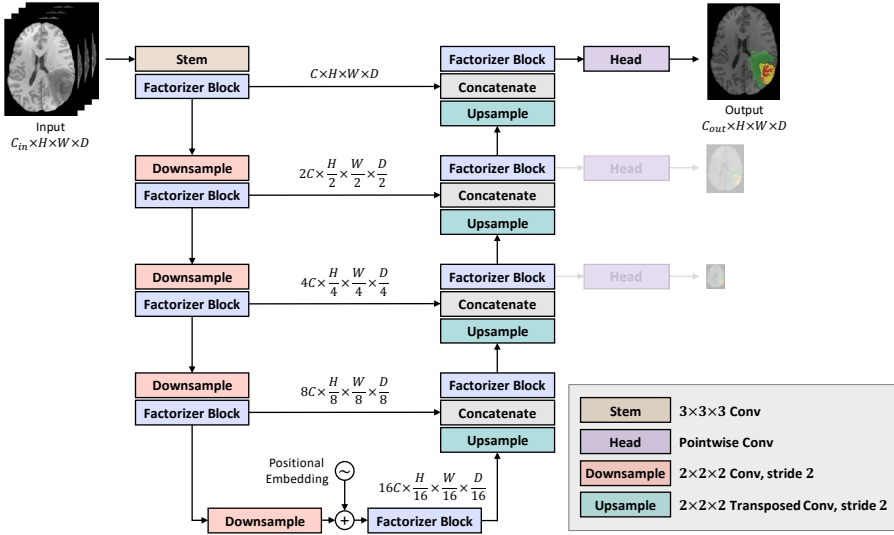


Figure 7.1 The overall architecture of Factorizer.

taken to make all of the entries nonnegative. As we will see in Section 7.4, our results suggest that NMF can potentially be an efficient yet effective alternative to attention-based context modeling for medical image segmentation.

7.3.2 Overall Architecture

As shown in Figure 7.1, a Factorizer model follows a U-Net-style architecture consisting of encoder and decoder parts with skip connections in between at equal resolutions. Given an input image $\mathcal{X} \in \mathbb{R}^{C_{in} \times H \times W \times D}$ with C_{in} channels and resolution (H, W, D) , the network outputs a *logit map* of size (C_{out}, H, W, D) , where C_{out} is the number of foreground classes. A single 3D convolution with a kernel size of $(3, 3, 3)$ is used as the stem to increase the number of channels to $C = 32$. However, note that in contrast to ViT, Factorizer does not flatten the spatial dimensions at the initial stage to generate a sequence of tokens.

The network has four stages, with the resolution decreasing to $1/16$ in the bridge. At each stage of the encoder (decoder), the input tensor is downsampled (upsampled) by a factor of two while the number of channels is doubled (halved). Convolution (transposed convolution) with a kernel size of $(2, 2, 2)$ and stride of 2 is used for downsampling (upsampling). In the bridge, learnable position embeddings are added to the input right after downsampling. We use deep

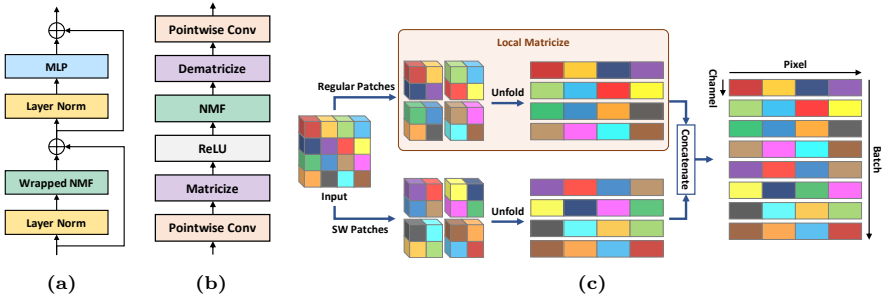


Figure 7.2 An overview of the Factorizer block and its components. (a) overall Factorizer block (b) Wrapped NMF module (c) An illustration of SW Matricize on a 2D toy example.

supervision [122] at the three highest resolutions in the decoder, applying pointwise convolutions (i.e., convolution with a kernel size of (1, 1, 1)) to get the output and two auxiliary low-resolution logit tensors.

7.3.3 Factorizer Block

A Factorizer block is constructed by replacing the multi-head self-attention module in a ViT block [80] with a Wrapped NMF module (described in Section 7.3.4). As shown in Figure 7.2a, a Factorizer block comprises NMF module and MLP, each of which comes after Layer Normalization and before a residual connection, that is,

$$\begin{aligned} \mathcal{Y} &= \text{WrappedNMF}(\text{LayerNorm}(\mathcal{X})) + \mathcal{X}, \\ \mathcal{Z} &= \text{MLP}(\text{LayerNorm}(\mathcal{Y})) + \mathcal{Y}, \end{aligned} \tag{7.4}$$

where MLP has two linear layers with a Gaussian Error Linear Unit (GELU) nonlinearity in between

$$\text{MLP}(\mathcal{X}) = \text{PointwiseConv}(\text{GELU}(\text{PointwiseConv}(\mathcal{X}))). \tag{7.5}$$

The number of input and output channels are the same, but the number of inner channels is double that of input channels.

7.3.4 Wrapped NMF Module

The major component of Factorizer is the Wrapped NMF module, which relies on matricization (i.e., an operation that turns a tensor into a matrix) and NMF.

As shown in Figure 7.2b, a Wrapped NMF subblock first applies a pointwise convolution to linearly project each voxel. The output is then reshaped into a batch of matrices using a *Matricize* operation, which is detailed later on in Section 7.3.4. The resulting matrices are passed through ReLU to clamp all their elements into nonnegative values and then low-rank approximated using NMF. The reconstructed matrices are reshaped back to their original size using the *Dematricize* operation, i.e., the inverse of *Matricize*. Finally, another pointwise convolution is applied, yielding the output. More formally, Wrapped NMF can be computed as follows:

$$\begin{aligned}
 \mathcal{X}^1 &= \text{PointwiseConv}(\mathcal{X}), \\
 \mathcal{X}^2 &= \text{Matricize}(\mathcal{X}^1), \\
 \mathcal{X}^3 &= \text{NMF}(\text{ReLU}(\mathcal{X}^2)), \\
 \mathcal{X}^4 &= \text{Dematricize}(\mathcal{X}^3), \\
 \text{WrappedNMF}(\mathcal{X}) &= \text{PointwiseConv}(\mathcal{X}^4), \tag{7.6}
 \end{aligned}$$

where $\text{NMF}(\cdot)$ is the NMF layer, described in 7.3.4. Intermediate tensors \mathcal{X}^i s and the output have the same size as the input \mathcal{X} .

Matricize

Before applying any matrix factorization method, an input batch of multi-dimensional images, denoted by $\mathcal{X} \in \mathbb{R}^{B \times C \times H \times W \times D}$, must be turned into a batch of matrices, say $\mathcal{Z} \in \mathbb{R}^{B' \times M \times N}$. This operation is called *Matricize*. In this work, we propose three Matricize operations: i) Global Matricize, ii) Local Matricize, and iii) Shifted Window (SW) Matricize. Depending on which operation is used, different variants for Factorizer are obtained: i) Global Factorizer, ii) Local Factorizer, and iii) Shifted **Window** (Swin) Factorizer. The Matricize operations are described in the following.

Global Matricize. This operation simply flattens the spatial dimensions and divides the channels into multiple groups (analogous to heads of Multi-Head Self-Attention). More specifically, Global Matricize reshapes a batch of C -channel 3D images $\mathcal{X} \in \mathbb{R}^{B \times C \times H \times W \times D}$ into a batch of matrices denoted by a 3D tensor $\mathcal{Z} \in \mathbb{R}^{B(C/E) \times E \times HWD}$, where E is called the head dimension, i.e., the number of channels per head (or matrix). This operation is obviously suitable for modeling global context and imposes no locality inductive bias.

Local Matricize. Global Matricize lacks a notion of locality which is typically useful for images, especially in low-data regimes. Local Matricize is proposed to mitigate this shortcoming by splitting an input $\mathcal{X} \in \mathbb{R}^{B \times C \times H \times W \times D}$ into a grid of non-overlapping patches of size (E, P, P, P) . These patches are flattened spatially and then concatenated along the batch dimension, yielding a batch of matrices presented by the tensor $\mathcal{Z} \in \mathbb{R}^{BCHW/(EP^3) \times E \times P^3}$. The entire procedure can be summarized by Einstein notation

$$\mathcal{Z}_{bg_c g_h g_w g_d, e, p_e p_w p_d} = \mathcal{X}_{b, g_c e, g_h p_h, g_w p_w, g_d p_d} \quad (7.7)$$

where (g_c, g_h, g_w, g_d) and (e, p_h, p_w, p_d) indices correspond to grid and patch dimensions, respectively. In PyTorch and TensorFlow, such Einstein operations can be simply implemented using `rearrange` function from `einops` library. Once NMF is applied to the resulting batch of matrices, the inverse operation of Local Matricize, called Local Dematricize, must be applied to transform them back to the initial shape. Local Dematricize can be easily obtained by composing the inverses of sub-operations in reverse order. In practice, Local Dematricize can also be formulated and implemented as an Einstein operation. A PyTorch-style pseudocode of Local (De)matricize module is provided in Algorithm 6.

Note that while Local Matricize seems to reshape an image in a similar way to the input patchifier of ViT, it concatenates patches along batch dimension rather than channel dimension. In fact, Local Matricize can be used when modeling within-patch interactions is desirable, which is different from ViT-based approaches, where interactions between embedded patches are typically modeled.

Shifted Window Matricize. While Local Matricize introduces locality to a model, voxels close to the boundaries of partitioning windows are not represented effectively. Since patches are low-rank approximated independently later on in an NMF layer, two neighboring boundary voxels from two adjacent patches are very likely to end up having excessively different feature maps in the output of the factorizer block, which in turn degrades the prediction performance for such voxels. To mitigate this problem, we utilize a shifted window approach proposed by Liu *et al.* [81] and introduce a Shifted Window (SW) Matricize operation, making the output feature maps smoother around boundaries.

An illustration of SW Matricize is provided in Figure 7.2c. Here, in addition to regular patches similar to those extracted by Local Matricize, shifted window (SW) patches are also included. Let $\mathcal{X} \in \mathbb{R}^{B \times C \times H \times W \times D}$ be the input and (P, P, P) the patch size. To extract SW patches, the input must be first shifted by the offset of $(P/2, P/2, P/2)$ along spatial dimensions such that voxels shifted beyond the boundaries of the images are re-introduced at the first position,

yielding a tensor of the same size (B, C, H, W, D) . We call this operation *Roll* (which can be implemented using `roll` function in PyTorch and TensorFlow). The resulting rolled tensor is then reshaped by Local Matricize to get SW patches. Finally, both the batches of regular and SW patches are concatenated along the batch dimension. Formally, SW Matricize is computed as follows:

$$\begin{aligned}\mathcal{X}^1 &= \text{LocalMatricize}_P(\mathcal{X}), \\ \mathcal{X}^2 &= \text{LocalMatricize}_P(\text{Roll}_{P/2}(\mathcal{X})), \\ \text{SWMatricize}(\mathcal{X}) &= \text{Concatenate}(\mathcal{X}^1, \mathcal{X}^2),\end{aligned}\tag{7.8}$$

where LocalMatricize_P denotes Local Matricize with a patch size of (P, P, P) , and $\text{Roll}_{P/2}$ denotes the roll operator with a shift $(P/2, P/2, P/2)$. To build SW Dematricize, we need to reconstruct the image from both regular and SW patches independently then compute their average to achieve smoother and more accurate feature maps.

Further details are provided in Section 7.6.2, which includes a PyTorch implementation of SW (De)matricize module presented in Algorithm 6.

Nonnegative Matrix Factorization

Once the input is somehow transformed into a batch of matrices, and its negative elements are clipped to zero by ReLU, it is ready to be low-rank approximated by Nonnegative Matrix Factorization (NMF). This is the main component of a Factorizer model that contributes most to modeling local or global context in an image.

NMF [98] seeks to approximate some given nonnegative matrix $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$ by

$$\mathbf{X} \approx \mathbf{F}\mathbf{G}^T,\tag{7.9}$$

where $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$ are factor matrices, and the positive integer $R \leq \min(M, N)$ is the rank. Once the factors \mathbf{F} and \mathbf{G}^T are somehow approximated, the NMF layer in a Factorizer block outputs the reconstructed matrix $\hat{\mathbf{X}} = \mathbf{F}\mathbf{G}^T$. Note that similar to self-attention, the NMF layer can be viewed as an adaptive filter, meaning that the computation of factors involves the input, as opposed to convolution, where kernel weights are fixed and independent from the input and do not change after training. Various loss functions have been used to form the objective function and measure the quality of an approximation. Depending on the loss function, constraints, and regularization, many variants of NMF have been proposed. In this work, we use

Algorithm 4: Multiplicative update for NMF (\odot and \div denote element-wise matrix product and division, respectively).

Input: $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$ (input matrix), R (rank); T (# iterations)

Output: $\hat{\mathbf{X}} \in \mathbb{R}_{\geq 0}^{M \times N}$ (low-rank matrix approximation)

1 Initialize factors $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$

2 **for** $t = 1, \dots, T$ **do**

3 $\mathbf{F} \leftarrow \mathbf{F} \odot \frac{\mathbf{X}\mathbf{G}}{\mathbf{F}\mathbf{G}^T\mathbf{G}}$

4 $\mathbf{G} \leftarrow \mathbf{G} \odot \frac{\mathbf{X}^T\mathbf{F}}{\mathbf{G}\mathbf{F}^T\mathbf{F}}$

5 **return** $\hat{\mathbf{X}} = \mathbf{F}\mathbf{G}^T$

a standard NMF with the squared error, which is the most widely used variant for images, to find factors by solving the following problem

$$\underset{\mathbf{F} \geq 0, \mathbf{G} \geq 0}{\text{minimize}} \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|^2. \quad (7.10)$$

In general, problem (7.10) is nonconvex, and finding global minima is NP-hard. However, numerous iterative algorithms have been proposed to find a “good local minimum”. The majority of existing methods are based on a block coordinate descent (BCD) scheme (aka alternating optimization), where the objective function is iteratively minimized with respect to one factor while the other factor is kept fixed. That is, the convex subproblems

$$\mathbf{F} \leftarrow \arg \min_{\mathbf{F} \geq 0} \|\mathbf{X} - \mathbf{F}\mathbf{G}^T\|^2, \quad \mathbf{G} \leftarrow \arg \min_{\mathbf{G} \geq 0} \|\mathbf{X}^T - \mathbf{G}\mathbf{F}^T\|^2, \quad (7.11)$$

are exactly or approximately solved alternately. This ensures that the objective function value does not increase after each update and guarantees convergence to a stationary point under some mild conditions [154].

Among numerous BCD-based algorithms for NMF, Multiplicative Update (MU) [98] is the best-known due to the advantage of being easy-to-implement and scalable. MU enforces a nonnegativity constraint by updating the previous values of a factor matrix by multiplication with a nonnegative scale factor. The pseudocode of MU is outlined in Algorithm 4. However, slow convergence of MU has been pointed out [140, Chapter 8], and hence more effective algorithms with faster convergence such as Hierarchical Alternating Least Squares (HALS) [101] have been introduced. HALS updates a factor, say $\mathbf{F} = [\mathbf{f}_1 \mid \dots \mid \mathbf{f}_R]$, with inner iterations, in which the columns \mathbf{f}_r s are updated by solving the

Algorithm 5: Hierarchical alternating least squares for NMF.

Input: $\mathbf{X} \in \mathbb{R}_{\geq 0}^{M \times N}$ (input matrix), R (rank); T (# iterations)

Output: $\hat{\mathbf{X}} \in \mathbb{R}_{\geq 0}^{M \times N}$ (low-rank matrix approximation)

1 Initialize factors $\mathbf{F} \in \mathbb{R}_{\geq 0}^{M \times R}$ and $\mathbf{G} \in \mathbb{R}_{\geq 0}^{N \times R}$

2 **for** $t = 1, \dots, T$ **do**

3 $\mathbf{A} \leftarrow \mathbf{X}\mathbf{G}, \mathbf{B} \leftarrow \mathbf{G}^T\mathbf{G}$

4 **for** $r = 1, \dots, R$ **do**

5 $F_{:r} \leftarrow \max\left(0, \frac{A_{:r} - \sum_{s \neq r} B_{sr}F_{:s}}{\|\mathbf{G}_{:r}\|^2}\right)$

6 $\mathbf{A} \leftarrow \mathbf{X}^T\mathbf{F}; \mathbf{B} \leftarrow \mathbf{F}^T\mathbf{F}$

7 **for** $r = 1, \dots, R$ **do**

8 $G_{:r} \leftarrow \max\left(0, \frac{A_{:r} - \sum_{s \neq r} B_{sr}G_{:s}}{\|\mathbf{G}_{:r}\|^2}\right)$

9 **return** $\hat{\mathbf{X}} = \mathbf{F}\mathbf{G}^T$

following subproblem

$$\mathbf{f}_r \leftarrow \arg \min_{\mathbf{f}_r \geq 0} \|\mathbf{E}_r - \mathbf{f}_r \mathbf{g}_r^T\|^2, \quad (7.12)$$

where $\mathbf{E}_r = \mathbf{X} - \sum_{\ell \neq r} \mathbf{f}_\ell \mathbf{g}_\ell^T$ is the residual matrix, which is, in fact, approximated by a rank-one matrix. An encouraging aspect of HALS is that each subproblem (7.12) can be easily shown to have a closed-form solution

$$\mathbf{f}_r^* \leftarrow \max\left(0, \frac{\mathbf{E}_r \mathbf{g}_r}{\|\mathbf{g}_r\|^2}\right). \quad (7.13)$$

Similarly, the update formula for columns of \mathbf{G} can be derived. Note that HALS is a $2R$ -block coordinate descent procedure, where at each outermost iteration, first the columns of \mathbf{F} and then the columns of \mathbf{G} are updated. Algorithm 5 provides pseudocode of HALS (further details on MU and HALS can be found in Section 5.1.2).

A special case of NMF is $R = 1$; i.e., $\mathbf{X} \approx \mathbf{f}\mathbf{g}^T$, where $\mathbf{f} \in \mathbb{R}_{\geq 0}^M$ and $\mathbf{g} \in \mathbb{R}_{\geq 0}^N$; for which one can easily derive that both MU and HALS are simplified to the same update rule:

$$\mathbf{f} \leftarrow \frac{\mathbf{X}\mathbf{g}}{\|\mathbf{g}\|^2}, \quad \mathbf{g} \leftarrow \frac{\mathbf{X}\mathbf{f}}{\|\mathbf{f}\|^2}. \quad (7.14)$$

In this chapter, all Factorizer models are trained with $R = 1$, and compression ratios (and indirectly reconstruction errors) are controlled by adjusting the

head dimension (i.e., the number of rows in a matrix, as discussed in 7.3.4) to sufficiently small values. This means that a matricize operation transforms an image into a batch of such fat matrices (i.e., the number of columns is much larger than the number of rows) that rank-one approximation would suffice in practice. This greatly simplifies a factorizer model and improves interpretability while yielding better segmentation performance. However, in one of our ablation studies (see Section 7.4.5), we experimented with both MU and HALS for $R > 1$ to investigate the impact of rank in the inference phase. The computational complexity of both MU and HALS is $\mathcal{O}(MNR)$ per iteration [140, Chapter 8], making the Wrapped NMF layer scale linearly and be much cheaper than self-attention with quadratic complexity (which is computationally intractable on long sequences) and even than Performer [84], as an efficient approximation of attention.

It is worth mentioning that not all NMF algorithms and their settings can be used in the NMF layer of a Factorizer block. The selected algorithm must have some properties so that we can ultimately train the Factorizer model successfully in an end-to-end fashion using a gradient descent-based optimizer on GPU(s) through an existing deep learning framework, such as PyTorch. Firstly, the algorithm should be backpropagation-friendly and amenable to automatic differentiation, that is $\frac{\partial \mathbf{X}}{\partial \mathbf{X}}$ should not only be well-defined and somewhat smooth but also computable by means of an existing deep learning framework, such as PyTorch, so that we can practically train the Factorizer model in an end-to-end fashion using a gradient descent optimizer. Another related aspect is that the gradient $\frac{\partial \mathbf{X}}{\partial \mathbf{X}}$, as explained in [153], starts to vanish during backpropagation after some iterations. Therefore, the number of outer iterations T in an NMF algorithm should be limited in order to have stable gradients. For MU and HALS, $T = 5$ is a reasonable choice in practice. Finally, update rules should be also friendly to GPU parallel processing for exploiting GPUs to train Factorizers in a reasonable amount of time. Taking all these factors into account, MU and HALS are appropriate choices. While HALS has better convergence properties, MU is more favorable for GPU training due to unparallelizable inner iterations of HALS.

7.4 Experiments

7.4.1 Datasets

We evaluate the effectiveness of our models on the Brain Tumour Segmentation (BraTS) dataset [25], [141] from Medical Segmentation Decathlon [155] and

Ischemic Stroke Lesion Segmentation (ISLES) 2022 dataset [18] from a MICCAI 2022 challenge.

BraTS. This dataset consists of 484 multiparametric MRI (mpMRI) scans from patients diagnosed with either low-grade glioma or high-grade glioma (glioblastoma). Each scan comes with four 3D MRI sequences, namely T2 Fluid-Attenuated Inversion Recovery (FLAIR), native T1-weighted (T1), post-Gadolinium contrast T1-weighted (T1Gd), and T2-weighted (T2). Once images are preprocessed (i.e., rigidly co-registered to the same anatomical template, resampled to the same voxel spacing 1mm^3 , and skull-stripped), the ground truths are manually created by experts who label each voxel as enhancing tumor (ET), edema (ED), necrotic and non-enhancing tumor (NCR/NET), or everything else. However, for evaluation, the 3 nested subregions, namely enhancing tumor (ET), tumor core (TC—i.e., the union of ED and NCR/NET), and whole tumor (WT) are used (see the sample ground truths in Figure 7.5).

ISLES'22. This dataset is from the ISLES'22 challenge, which aims to evaluate automated methods of acute and sub-acute stroke lesion segmentation in 3D multiparametric MRI data, namely DWI, Apparent Diffusion Coefficient (ADC), and FLAIR sequences. The DWI and ADC images of a patient are aligned while the FLAIR image in its native space has a different voxel size and must be registered to the DWI space. As DWI and ADC are the most informative modalities for stroke lesions, FLAIR is ignored in this chapter to avoid the complication of FLAIR-DWI registration and simplify the pipelines. The dataset consists of 250 cases, each is skull-stripped and includes an expert-level annotation of the stroke lesions.

7.4.2 Setup

All the models were implemented using PyTorch [129] and MONAI [131] frameworks and trained on NVIDIA P100 GPUs. We followed the same training workflow in all the experiments. In the following, we first provide the details of this workflow and baseline models, then present the evaluation protocol and the results.

Preprocessing. For each scan in a dataset, a multi-channel 3D image as the input was first constructed by concatenating the modalities—i.e., FLAIR, T1, T1Gd, and T2 for BraTS, and DWI and ADC for ISLES'22. The image and its ground truth were then cropped with a minimal box filtering out zero regions of the image. The image was normalized channel-wise using a z-score to have intensities with zero mean and unit variance. Random patches of size (128, 128, 128) for BraTS and (64, 64, 64) for ISLES'22 were extracted during

training. To reduce overfitting, we used data augmentation techniques, including random affine transform, random flip along each spatial dimension, additive Gaussian noise, random Gaussian smoothing, random intensity scaling, random intensity shifting, and random gamma transform. Further details are provided in 7.6.1.

Training. All models were trained for 100000 steps with a batch size of 2 (one sample per GPU) using AdamW optimizer with a base learning rate of 10^{-4} , weight decay of 10^{-2} , warmup of 2000 steps, and cosine annealing scheduler. The loss ℓ_{total} is computed by incorporating the three deep supervision outputs and the corresponding downsampled ground truths according to

$$\ell_{\text{total}} = \lambda_1 \ell(\mathbf{G}_1, \mathbf{P}_1) + \lambda_2 \ell(\mathbf{G}_2, \mathbf{P}_2) + \lambda_3 \ell(\mathbf{G}_3, \mathbf{P}_3), \quad (7.15)$$

where $\lambda_1 = 1$, $\lambda_2 = 0.5$, and $\lambda_3 = 0.25$; \mathbf{G}_i and \mathbf{P}_i correspond to the deep supervision at the i th highest resolution; and the loss function $\ell(\cdot, \cdot)$ is a combination of *soft Dice loss* [132] and cross-entropy loss, defined as

$$\ell(\mathbf{G}, \mathbf{P}) = \ell_{\text{Dice}}(\mathbf{G}, \mathbf{P}) + \ell_{\text{CE}}(\mathbf{G}, \mathbf{P}), \quad (7.16)$$

where

$$\ell_{\text{Dice}}(\mathbf{G}, \mathbf{P}) = 1 - \frac{2\langle \mathbf{G}, \mathbf{P} \rangle + \epsilon}{\|\mathbf{G}\|^2 + \|\mathbf{P}\|^2 + \epsilon}, \quad \ell_{\text{CE}}(\mathbf{G}, \mathbf{P}) = -\frac{1}{N} \langle \mathbf{G}, \log(\mathbf{P}) \rangle, \quad (7.17)$$

where $\mathbf{G} \in \{0, 1\}^{J \times N}$ and $\mathbf{P} \in [0, 1]^{J \times N}$ represent the one-hot encoded ground truth and the predicted probability map for each voxel, respectively, with J denoting the number of foreground classes and N denoting the number of voxels in the patch. The small constant $\epsilon = 10^{-5}$ is commonly used to smooth the soft Dice loss and avoid division by zero.

Inference. A test image in the inference was first subjected to z-score intensity normalization, then the prediction was made using a sliding window approach with a 50% overlap and a window size of $128 \times 128 \times 128$ (same as the patch size used in training). Finally, the resulting probabilities were thresholded by 0.5 to obtain a binary segmentation map.

Evaluation Metrics. The Dice score and Hausdorff Distance 95% (HD95) were used as metrics to assess the performance of models in our experiments. For each segmentation region, the Dice score measures the voxel-wise overlap between the ground truth and the prediction, defined as

$$\text{Dice}(\mathbf{g}, \mathbf{y}) = \frac{2 \sum_{n=1}^N g_n y_n}{\sum_{n=1}^N g_n + \sum_{n=1}^N y_n} \quad (7.18)$$

where $g_n \in \{0, 1\}$ and $y_n \in \{0, 1\}$ represent the ground truth and the binary prediction for a voxel, respectively, and N is the number of voxels. If both the ground truth and the prediction do not have any nonzero values, that is the denominator of equation (7.18) is zero, the Dice score is defined as 1. Hausdorff Distance (HD) evaluates the distance between the boundaries of ground truth and prediction. HD is defined as follows

$$\text{HD}(\mathbb{G}, \mathbb{Y}) = \max\{\max_{\mathbf{g} \in \mathbb{G}} \min_{\mathbf{y} \in \mathbb{Y}} \|\mathbf{g} - \mathbf{y}\|, \max_{\mathbf{y} \in \mathbb{Y}} \min_{\mathbf{g} \in \mathbb{G}} \|\mathbf{y} - \mathbf{g}\|\}, \quad (7.19)$$

where \mathbb{G} and \mathbb{Y} denote the set of all voxels on the surface of ground truth and prediction, respectively. HD95 is a more robust version to outliers, which calculates the 95% quantile rather than the maximum of surface distances.

Models. In all the experiments, for Factorizer models with the overall architecture illustrated in Figure 7.1, the number of output channels of the stem was $C = 32$. For Local and Swin Factorizer, we used a large window size of $(8, 8, 8)$ and $(4, 4, 4)$ on the BraTS and ISLES’22 datasets, respectively, to aggregate local information, which is opposed to typical CNNs comprising convolutions mostly with a small receptive field of size $(3, 3, 3)$. For all Factorizer models, a head dimension of $E = 8$ on BraTS and $E = 4$ on ISLES’22 was used. In NMF modules, the factor matrices were initialized with uniform distribution $\mathcal{U}(0, 1)$. In training, we used a rank-one approximation ($R = 1$) with $T = 5$ outer iterations of HALS (which is equivalent to MU for $R = 1$), as described in Section 7.3.4.

We compare Factorizers against seven baseline models, among which nnU-Net, Res-U-Net, and Performer follow the same overall architecture and setup as those of Factorizers except that each has a different encoder/decoder block, allowing us to better assess the impact of blocks rather than architectures by eliminating the effect of architectural variability. These three baselines are detailed below:

- nnU-Net: This model is based on nnU-Net [93], i.e., a standard 3D U-Net [85] with minor modifications. Each encoder (and decoder) block is composed of two convolutions with a kernel size of $(3, 3, 3)$. Group Normalization (with a group size of 8) is adopted right after each convolution and before LeakyReLU nonlinearity. This model does not have any positional embedding in its bridge since CNNs already have some notion of position.
- Res-U-Net: ResNet block [70] is the cornerstone of Res-U-Net. This block is similar to that of nnU-Net, except that it has a residual connection after the last Group Normalization. This model is similar to SegResNet [86].

- **Performer:** The encoder and decoder blocks of this model are based on ViT (note that the input is first flattened into a sequence of voxels before feeding it to a ViT block). However, attention scales quadratically with the number of voxels, hence is prohibitively expensive in our case, where the input can have up to 128^3 voxels. Therefore, we replace original attention layers of a ViT block with FAVOR+ (Fast Attention Via positive Orthogonal Random features) used in Performer, recently proposed by Choromanski *et al.* [84] as a linearly scalable alternative to Transformer. FAVOR+ gives an unbiased estimate of attention using only linear (as opposed to quadratic) time and space complexity. Note that except for the attention layers, the rest of the components of this baseline are the same as those of Factorizer.

We also use four state-of-the-art Transformer-based baselines, namely TransBTS [88], UNETR [89], Swin UNETR [90], and nnFormer [92], each of which has a different overall architecture than that of Figure 7.1. These models follow U-shaped architectures but apply Transformer blocks only to low-resolution images after somehow downsizing the input using patchifying or convolutional tokenizers, thereby avoiding the computational intractability of self-attention on long sequences. In contrast, Global Factorizer and Performer exploit the global context at all stages of their architectures, from the lowest to highest resolution.

7.4.3 Results and Discussion

Brain Tumor Segmentation (BraTS)

Quantitative Evaluation. For all the experiments, we performed 5-fold cross-validation to estimate how capable our models are in generalizing to unseen data. The results on the BraTS dataset are reported in Table 7.1 and illustrated by box plots in Figure 7.3a and 7.3b, where pairwise Wilcoxon signed-rank tests were used for comparing the performance of our best model, Swin Factorizer, with that of the baselines.

Swin Factorizer is the clear overall winner in the brain tumor segmentation task. With an average Dice score of 84.21% and HD95 of 6.89 mm, Swin Factorizer significantly outperformed Res-U-Net, the best CNN-based and second-best overall baseline, with p-values of < 0.01 for the average Dice score and < 0.05 for the average HD95, while requiring six times fewer computations. Swin Factorizer was the best-performing model on ET and the second best-performing model on TC and WT, particularly yielding the highest Dice score of 79.33% on ET. Despite having over 95% fewer parameters and requiring over 60% fewer FLOPs,

Table 7.1 Comparison of different models on the BraTS dataset. Scores are obtained by 5-fold cross-validation. The best results are in **boldface** and second best ones are underlined.

Model	#Params	FLOPs	Dice (%) \uparrow				HD95 (mm) \downarrow			
			ET	TC	WT	Avg.	ET	TC	WT	Avg.
nnU-Net	28.7M	1152.8G	75.89	81.94	90.09	82.64	8.61	9.13	9.31	9.19
Res-U-Net	28.9M	1168.6G	77.95	82.49	90.39	83.61	6.08	8.18	<u>8.06</u>	7.52
Performer	6.7M	222.7G	78.43	82.34	88.72	83.16	6.72	9.78	12.93	10.21
TransBTS	33.0M	653.0G	73.46	80.00	85.42	79.63	15.48	13.77	20.73	17.12
UNETR	111.5M	1124.9G	76.40	82.06	89.37	82.61	8.09	9.41	10.59	9.56
Swin UNETR	62.2M	1549.8G	76.50	82.65	90.10	83.08	9.69	8.75	11.04	9.91
nnFormer	149.4M	489.9G	77.71	83.41	90.05	<u>83.72</u>	5.42	<u>7.45</u>	7.68	<u>6.93</u>
Global Factorizer	5.9M	170.0G	78.20	82.86	88.65	83.24	6.67	9.53	11.69	9.71
Local Factorizer	5.9M	170.0G	<u>78.67</u>	82.85	89.30	83.61	<u>5.29</u>	7.77	8.91	7.41
Swin Factorizer	5.9M	174.2G	79.33	<u>83.14</u>	<u>90.16</u>	84.21	4.91	7.31	8.23	6.89

Swin Factorizer still outperformed nnFormer, the best-performing baseline, in terms of average Dice and HD95. With an average Dice score of 83.61%, Local Factorizer yielded comparable results to Res-U-Net, and both Global and Local Factorizer demonstrated improved performance compared to Performer, nnU-Net, TransBTS, UNETR, and Swin UNETR. As shown in Figure 7.4, while achieving competitive performance over the baselines, Factorizer models have much fewer parameters and are significantly cheaper.

Performer is the Transformer-based counterpart of Global Factorizer in the sense that they both follow the same overall architecture without imposing any locality inductive bias in their blocks. However, while having lower computational complexity, Global Factorizer, with an average Dice score of 83.24% and an average HD95 of 9.71 mm, marginally outperformed Performer, with an average Dice score of 83.16% and an average HD95 of 10.21 mm. Local Factorizer improved the performance of Global Factorizer by exploiting locality, particularly with the average HD95 significantly dropping from 9.71 mm to 7.41 mm (p-value < 0.0001). As expected, Swin Factorizer improved all the scores of Local Factorizer, which is consistent with the fact that Swin Factorizer modifies Local Factorizer by better representing the boundary voxels in Local Matricize.

Qualitative Comparisons. Qualitative comparisons of glioma segmentation models are presented in Figure 7.5. Swin Factorizer demonstrates superior performance in segmenting TC and ET. This capability is evident in row 1 (row 2), where Swin Factorizer more successfully delineates TC compared to the other models, which do not as accurately distinguish ED (normal tissues) from

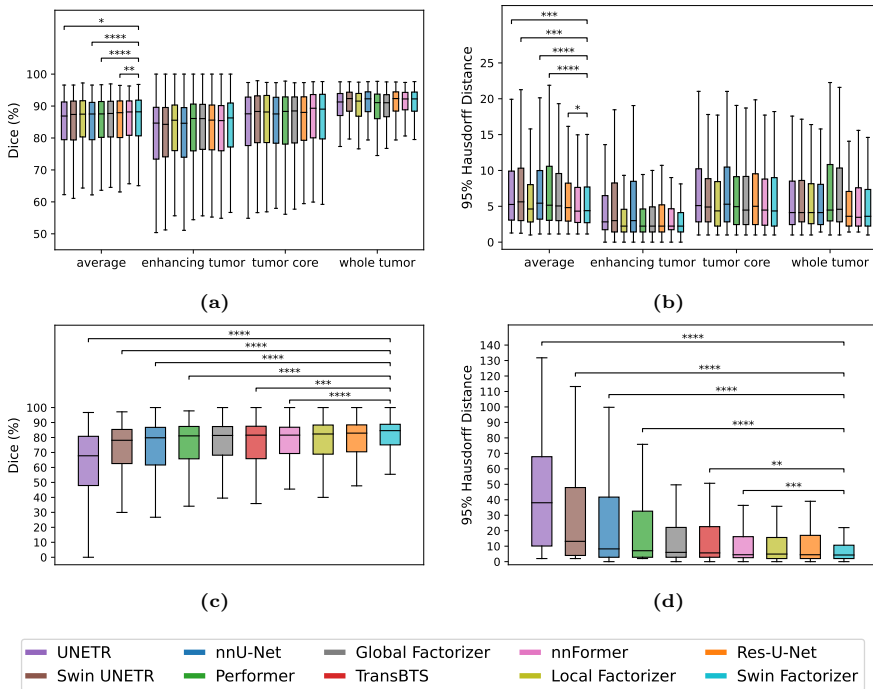


Figure 7.3 The Dice score and 95% Hausdorff distance of different models on the BraTS dataset. The asterisks indicate significance according to pairwise Wilcoxon signed-rank test (*: p-value < 0.05; **: p-value < 0.01; ***: p-value < 0.001; and ****: p-value < 0.0001).

TC. Particularly, nnU-Net misclassifies a significantly larger part of ED (normal tissues) as NCR/NET (ET). Row 3 exemplifies a successful detection of TC by Swin Factorizer, while the other models miss a fairly large NCR/NET region.

NMF Components Interpretability. One additional advantage of Factorizer over Transformer and CNN models is its higher interpretability resulting from meaningful components of NMF in the sense that each component represents specific image semantics in practice. Note that both the first and last Factorizer blocks (i.e., the high-resolution blocks of the encoder and the decoder, which are just after the stem layer and just before the head layer, respectively) have $C = 32$ channels, divided into groups of 8-channel heads during matricization, where the NMF of each head has only a single rank-one term, i.e., $R = 1$. As a result, in total, there are $CR/8 = 4$ components in both the first and last NMF

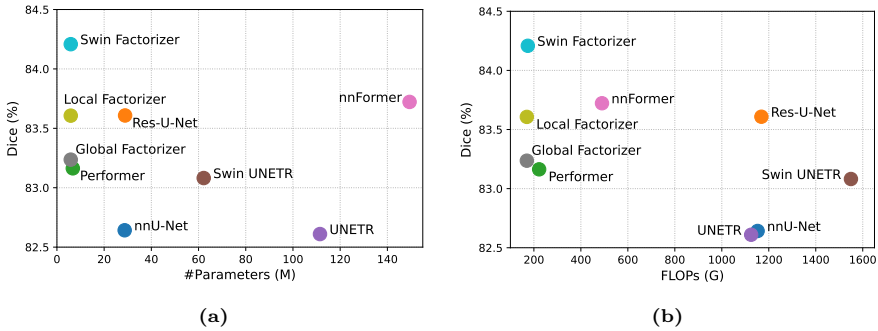


Figure 7.4 Performance of Factorizer models in terms of average Dice on BraTS, versus the number of parameters (left) and versus their speed in terms of the number of FLOPs (right).

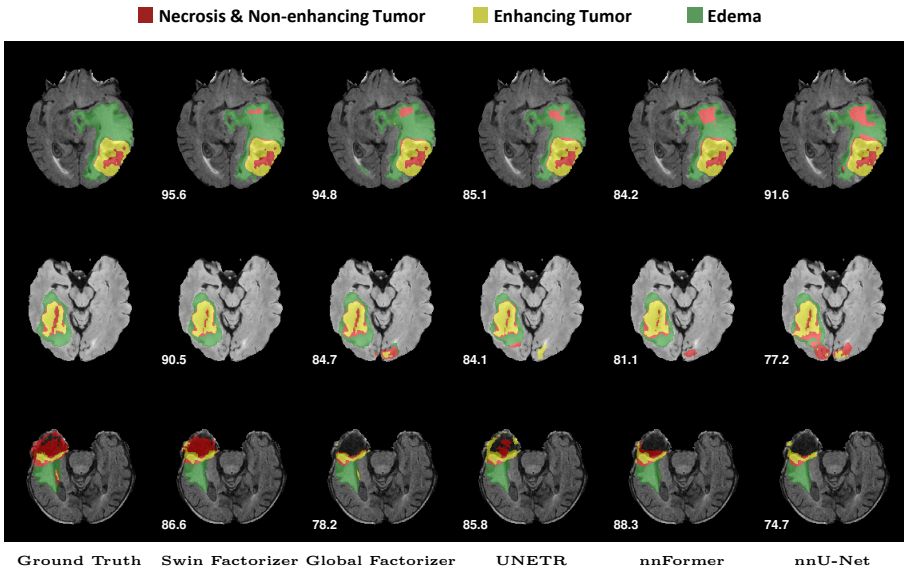


Figure 7.5 Qualitative results on brain tumor segmentation. All the examples are from the validation sets of the 5-fold cross-validation. TC is the union of red (NCR/NET) and yellow (ET) regions, and WT is the union of green (ED), red, and yellow regions. The patient-wise average Dice score is presented for each case.

layers on a high-grade glioma case for Swin Factorizer and Global Factorizer. More precisely, each component illustrates the factor matrix corresponding to spatial dimensions after dematricization.

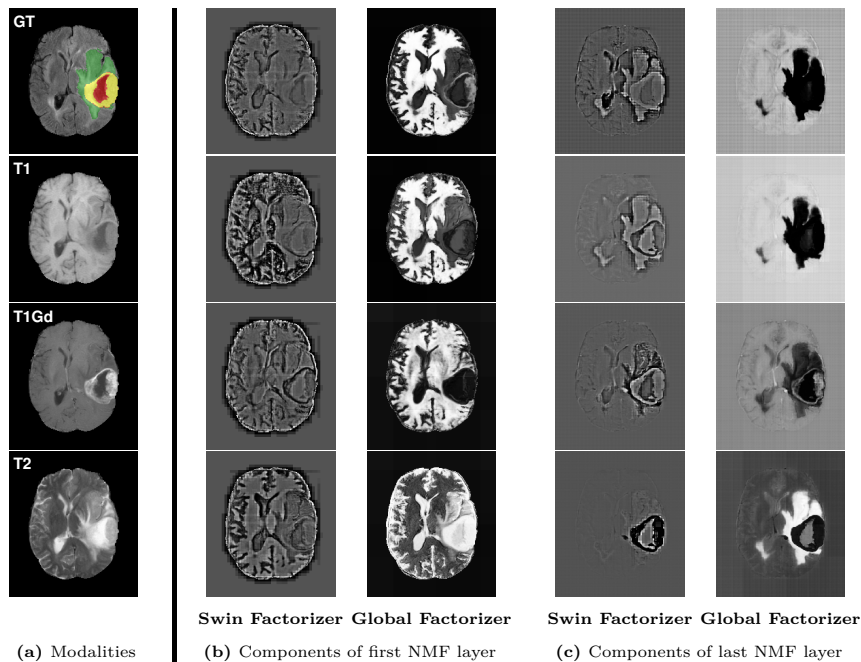


Figure 7.6 NMF Components of Factorizer models on BraTS. The example is from a validation set of the 5-fold cross-validation. Panel (a) shows the modalities and ground truth for a high-grade glioma case. Panel (b) and (c) present the NMF spatial components in the first and last block, respectively.

Interestingly, the components appear highly meaningful and interpretable in the sense that each component gives an interpretation by differentiating one region from another. For instance, as observed in Figure 7.6b, the first (row 1) and second (row 2) components discriminate roughly WT while the third (row 3) and fourth (row 4) components capture TC. For Swin Factorizer, NCR is distinguished more clearly in the third and fourth components whereas for Global Factorizer NCR is more recognizable in the second component. As we get to deeper layers, components become even more meaningful such that in the last layer, each component clearly detects some regions. For example, as seen in Figure 7.6c, the first (row 1) and second (row 2) components very clearly discriminate WT while the third (row 3) and fourth (row 4) components also capture TC and NCR.

Another interesting observation is that Global Factorizer yields more discriminative and higher-level components in the first layer, which can be attributed to the fact Global Factorizer models long-range dependencies through all of its

blocks, including the first one; however, the receptive field of Swin Factorizer in the first stage of the encoder is small but progressively increases as it passes through downsampling layers. Therefore, Swin Factorizer extracts lower-level features in shallower layers and higher-level ones in deeper layers in a similar way to CNNs. Notice that the footprints of shifted windows, which appear as a grid pattern, are also evident in all the components of Swin Factorizer.

Ischemic Stroke Lesion Segmentation (ISLES)

Quantitative Evaluation. Similarly to Section 7.4.3, 5-fold cross-validation was performed, and pairwise Wilcoxon signed-rank tests were used for comparison purposes. The results on the ISLES'22 dataset are reported in 7.2 and illustrated by box plots in Figure 7.3c and 7.3d.

Swin Factorizer, with a Dice score of 76.49% and HD95 of 11.96 mm, demonstrated the best performance, significantly outperforming nnU-Net and all the Transformer-based models with a p-value of < 0.001 for the Dice score and < 0.01 for the HD95 value. Moreover, Swin Factorizer showed improved performance compared to Res-U-Net, which has over three times more parameters and needs over four times more FLOPs.

Despite having over 95% fewer parameters and taking over 50% fewer FLOPs, Local Factorizer yielded a Dice score of 74.28%, which is higher than that of nnFormer, the best-performing Transformer-based model. Local Factorizer also demonstrated a smaller HD95 value with 80% fewer FLOPs than Res-U-Net, the best baseline overall. Finally, Global Factorizer outperformed its Transformer-based model, Performer, by a large margin, whereas it still has the advantage of lower computational cost. As a side note, our Factorizer-based model trained on all the three modalities (i.e., DWI, ADC, and registered FLAIR) and submitted to the ISLES'22 MICCAI challenge ranked among the top three in the final leaderboard, which further verifies the potential of Factorizer as an effective alternative for 3D medical image segmentation (please see <https://isles22.grand-challenge.org/isles22/>).

Qualitative Comparisons. Qualitative comparisons of stroke lesion segmentation models are presented in Figure 7.7. Compared to UNETR and nnU-Net, our Swin Factorizer and Global Factorizer models substantially reduce false positives, as observed in row 1. Both UNETR and nnU-Net produce large regions of incorrect lesions circled in green, but nnU-Net suffers from fewer false positives than UNETR, as evident in row 2.

Although nnFormer seems to yield relatively small false positive regions compared to UNETR and nnU-Net, Swin Factorizer not only produces even slightly fewer false positives but also enjoys more favorable results when it

Table 7.2 Comparison of different models on the ISLES22 dataset. Scores are obtained by 5-fold cross-validation. The best results are in **boldface** and second best ones are underlined.

Model	#Params	FLOPs	Dice (%) \uparrow	HD95 (mm) \downarrow
nnU-Net	28.7M	143.1G	69.71	26.59
Res-U-Net	28.9M	145.1G	<u>75.41</u>	16.51
Performer	8.2M	33.0G	71.30	22.18
TransBTS	31.0M	66.1G	72.44	19.35
UNETR	104.8M	137.8G	61.43	41.62
Swin UNETR	62.2M	194.6G	70.41	31.67
nnFormer	149.2M	60.6G	73.79	<u>12.99</u>
Global Factorizer	7.4M	28.5G	72.51	18.06
Local Factorizer	7.4M	28.5G	74.28	16.14
Swin Factorizer	7.4M	29.1G	76.49	11.96

comes to false negatives. This is exemplified by row 3, where Swin Factorizer successfully captures both the stroke lesions, whereas nnFormer fails to detect the lesion circled in orange. Overall, Swin Factorizer displays very competitive segmentation results, superior to those of UNETR, nnFormer, and nnU-Net in most cases. These results verify the potential of Factorizer as an alternative to state-of-the-art models, such as nnFormer and nnU-Net.

7.4.4 Ablation Studies: Training

We conducted ablation studies on Factorizer to further investigate the effects of Factorizer subblocks and positional embedding. In this section, models with an ablated layer were trained from scratch on the BraTS dataset.

Factorizer Subblock. Ablation results of subblocks on brain tumor segmentation are reported in Table 7.3. When we removed NMF subblocks from a Factorizer model, the performance substantially dropped while Factorizer models without MLP subblocks demonstrated less deterioration in results. Local Factorizer without MLP blocks yielded an average Dice of 82.67%, still outperforming nnU-Net, and Swin Factorizer without MLP blocks yielded an average Dice of 83.57%, which is significantly greater than that of nnU-Net and comparable with that of Res-U-Net. These results indicate the effectiveness of the NMF layer in improving the models.

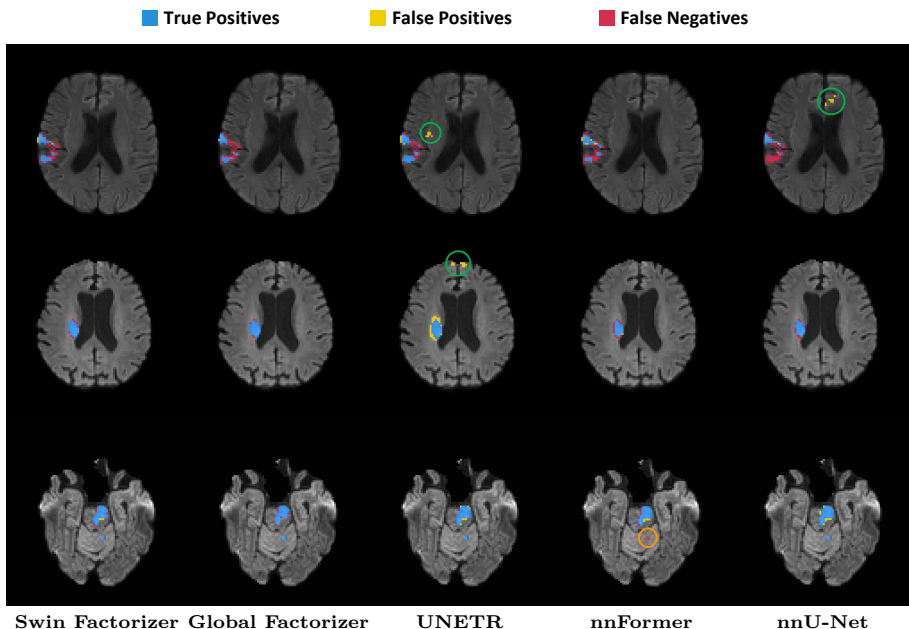


Figure 7.7 Qualitative results on stroke lesion segmentation. All the examples are from the validation sets of the 5-fold cross-validation. UNETR and nnU-Net produce the false positive lesions circled in green (row 1 and 2), and nnFormer fails to detect the lesion circled in orange (row 3).

Table 7.3 The impact of each subblock on the performance of Factorizers on BraTS.

MLP Subblock	NMF Subblock	Matricize	Dice (%) \uparrow				HD95 (mm) \downarrow			
			ET	TC	WT	Avg.	ET	TC	WT	Avg.
✓	✗	✗	77.02	80.89	87.36	81.76	6.55	8.90	11.29	9.02
✗	✓	Global	77.51	80.78	87.85	82.05	8.82	11.24	16.18	12.33
✗	✓	Local	77.79	81.67	88.56	82.67	5.50	8.67	9.44	8.04
✗	✓	SW	78.60	82.61	89.51	83.57	5.00	7.47	7.86	6.87
✓	✓	SW	79.33	83.14	90.16	84.21	4.91	7.31	8.23	6.89

Positional Embedding. Table 7.4 shows the results of the ablation study on positional embedding. In all the cases, adding positional embedding to the bridge of a network improved the performance. Particularly, the average Dice score of Global Factorizer increased from 83.09% to 83.24% significantly with p -value = 0.031, and the average HD95 fell from 10.12 mm to 9.71 mm although this improvement is not statistically significant. Local and Swin Factorizers

Table 7.4 The impact of positional embedding on the performance of Factorizers on BraTS (the asterisks indicate p-value < 0.05 in the pairwise Wilcoxon signed-rank test between a model with and without positional embedding).

Model	Positional Embedding	Dice (%) \uparrow				HD95 (mm) \downarrow			
		ET	TC	WT	Avg.	ET	TC	WT	Avg.
Global Factorizer	\times	77.98	82.54	88.76	83.09	7.57	9.91	11.73	10.12
Global Factorizer	\checkmark	78.20	82.86	88.65	83.24*	6.67	9.53	11.69	9.71
Local Factorizer	\times	78.24	82.59	89.37	83.40	5.87	8.75	9.98	8.31
Local Factorizer	\checkmark	78.67	82.85	89.30	83.61	5.29	7.77	8.91	7.41
Swin Factorizer	\times	78.63	83.18	90.16	83.99	5.50	7.24	8.03	6.97
Swin Factorizer	\checkmark	79.33	83.14	90.16	84.21	4.91	7.31	8.23	6.89

without positional embedding yielded an average Dice of 83.40% and 83.99%, respectively, slightly underperforming compared to their counterparts with positional embedding. Like Transformers, Factorizers lack any notion of voxel position, and therefore typically benefit from a positional embedding mechanism, which is generally consistent with our results.

7.4.5 Ablation Studies: Inference

Since the output of an NMF layer is a low-rank approximation of the input, it makes sense to perform an ablation study in the inference phase, for instance by short-circuiting an NMF layer. For all the experiments in this section, we ablated some NMF layers or changed their settings in the inference phase after training the model on the BraTS dataset.

NMF Layer. We investigated the impact of short-circuiting some NMF layers of the pre-trained Swin Factorizer model in the inference phase. In Figure 7.8a, the average Dice score on BraTS is shown when we kept the first NMF layers and removed (or short-circuited) the rest. As expected, the more layers we kept, the higher the Dice score was achieved. We observed that most of the performance is achieved via the encoder, which includes the first five blocks. Figure 7.8b shows the results of investigating the impact of each individual layer, where we kept all the layers except one at a time. Interestingly, we noticed that the NMF layer of the bridge block (layer 5) makes the greatest contribution to the performance. In fact, if an NMF layer except that of the bridge is ablated, the average Dice still stays above 80%. Particularly, if an NMF layer in the decoder (layers 6 to 9) is ablated, the performance is still better than that of nnU-Net. Note that removing an NMF layer, especially

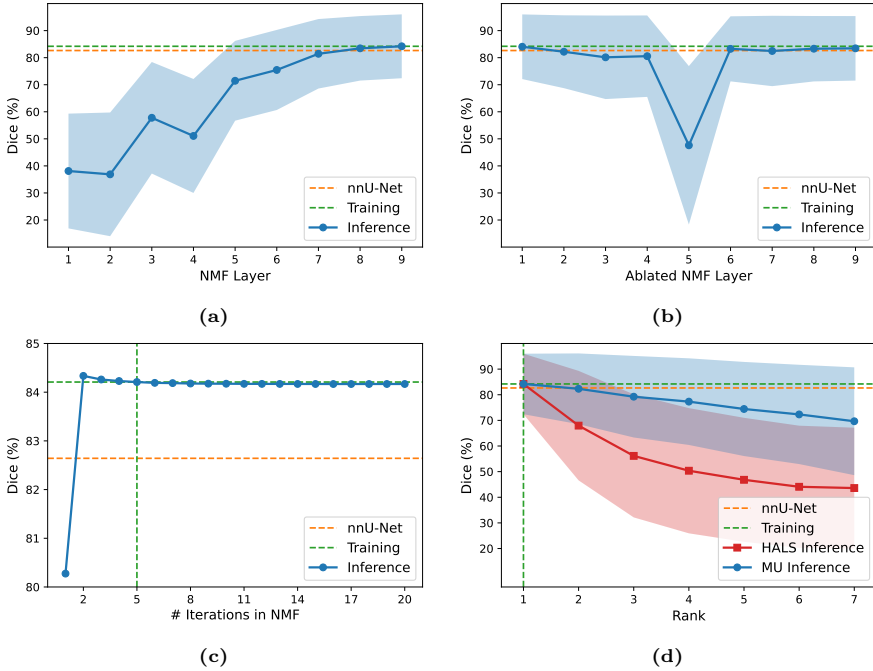


Figure 7.8 Inference-phase ablation experiments with Swin Factorizer pre-trained on BraTS. Panel (a) shows the average Dice score when only the first NMF layers are kept, while Panel (b) shows the Dice score plotted versus the NMF layer removed. Panel (c) shows the plot of Dice against the number of outer iterations in HALS. Panel (d) shows the impact of rank on performance in HALS and MU. The dashed orange lines indicate the nnU-Net Dice score, while the dashed green lines correspond to the pre-trained Swin Factorizer model.

those at higher-resolution stages of the network, can significantly reduce the computational complexity and speed up the inference time.

NMF Solver Iterations. We investigate the effect of changing the number of outer iterations (T) in HALS. Recall that all the Factorizer models were originally trained using HALS with $T = 5$. Figure 7.8c shows the results of experimenting with $T \in \{1, \dots, 20\}$ in the inference phase for the Swin Factorizer model pre-trained with $T = 5$. We noticed that for $T > 1$, the performance is very close to that of the original model ($T = 5$). Interestingly, $T \in \{2, 3, 4\}$ yielded even higher average Dice scores compared to the original model although the improvement was not statistically significant. We attribute this to the possible regularization effect of reducing T . Note that we proportionally decreased the computational cost of NMF layers by reducing T while preserving the accuracy.

In fact, by training a Factorizer, we also obtain lightweight yet accurate versions of that for free without needing to re-train any model from scratch. All we need is to reduce T or ablate some costly NMF layers in the inference phase. When it comes to model speed-up, this brings a great advantage to Factorizers over CNNs and Transformers, which require much more complex mechanisms to achieve their faster versions.

Rank. We investigated the impact of changing the rank of the pre-trained Swin Factorizer model in the inference phase. Recall that Swin Factorizers were trained with $R = 1$, in which both HALS and MU lead to the same update rule. Therefore, for $R > 1$ in inference, we experimented with both HALS and MU in order to make a comparison. In Figure 7.8d, the average Dice score on BraTS is plotted as a function of rank. As observed, the more the rank deviated from $R = 1$ (the one used in training), the more significantly the Dice score dropped. In comparison to HALS, MU demonstrated a less dramatic reduction in performance as we increased the rank. This can be due to the fact that within the same number of outer iterations, HALS typically makes a larger decrease in the NMF objective (given in equation (7.10)), causing the HALS-based model to deviate further from the original model than that of MU.

7.5 Conclusion and Future Work

Vision Transformers, particularly those with hierarchical architectures, have recently achieved results comparable with state-of-the-art CNNs on various computer vision tasks. Nevertheless, the lack of locality inductive bias makes them underperform their CNN counterparts in low-data regimes, which is usually the case in medical image segmentation. Moreover, the quadratic complexity of attention makes existing Transformers apply self-attention layers only after somehow reducing the image resolution, and thus, fail to fully capture long-range contexts present at higher resolutions. Hence, this chapter introduces a family of models, called Factorizer, which leverages the power of low-rank approximation for developing a scalable interpretable approach to context modeling by formulating NMF as a differentiable layer integrated into an end-to-end U-shaped architecture. Built upon NMF and shifted window idea, Swin Factorizer competed favorably with CNN and Transformer baselines in terms of accuracy and scalability. Swin Factorizer yielded state-of-the-art results on BraTS for brain tumor segmentation; with Dice scores of 79.33%, 83.14%, and 90.16% for enhancing tumor, tumor core, and whole tumor, respectively; and on ISLES'22 for stroke lesion segmentation, with a Dice score of 76.49%. Our experiments indicated that NMF components are highly meaningful, which gives a great advantage to Factorizers over CNNs and Transformers in terms of

interpretability. Moreover, our ablation studies revealed a distinctive feature of Factorizers that allows the speed-up of inference for a pre-trained Factorizer with no extra steps and without sacrificing much accuracy.

Matrix factorization models are very flexible and versatile. In this work, we used an ordinary NMF model together with some matricization techniques to model local or global contexts. One possible extension of this work is to customize the NMF objective to simultaneously exploit both local and global contexts. Moreover, it would be useful to explore some ideas for automated selection of the rank hyperparameter, for example, by taking a greedy approach to NMF, aka nonnegative matrix underapproximation, where the components are constructed and added one by one (sequentially) until some criteria are met. This can benefit especially Local Factorizer as different regions happen to have different optimal ranks depending on their contexts. We will explore the effectiveness of other NMF variants, such as Semi-NMF and Convex-NMF, which work on mixed-signed data matrices and relax the need for the ReLU activation function before factorization. Finally, while this chapter focuses on the segmentation of 3D medical images, Factorizer may also potentially serve as an effective approach for efficiently processing high-resolution 2D medical or natural images, which can be further investigated in the future.

7.6 Appendix

7.6.1 Data Augmentation Pipeline

The MONAI library [131] was used for data augmentation. The following random transforms were applied on the fly during training in the following order:

1. **Affine transform.** An affine transform was applied with a probability of 0.15, where the angles of rotation (in degrees) and the scaling factor were drawn from $\mathcal{U}(-30, 30)$ and $\mathcal{U}(0.7, 1.3)$, respectively.
2. **Flip.** All patches were flipped with a probability of 0.5 along each spatial dimension.
3. **Gaussian Noise.** Each voxel of a patch was independently subject to additive white Gaussian noise with a variance drawn from $\mathcal{U}(0, 0.1)$. This augmentation was applied with a probability of 0.15.
4. **Gaussian Smoothing.** Each modality of a patch was filtered independently via a symmetric Gaussian kernel whose width (in voxels) was

drawn from $\mathcal{U}(0.5, 1.5)$. This augmentation was applied with a probability of 0.15.

5. **Scale Intensity.** Voxel intensities were scaled with $s \sim \mathcal{U}(0.7, 1.3)$ with a probability of 0.15.
6. **Shift Intensity.** Voxel intensities were shifted with offsets drawn from $\mathcal{U}(-0.1, 0.1)$ with a probability of 0.15.
7. **Adjust Contrast.** Gamma correction with $\gamma \sim \mathcal{U}(0.7, 1.5)$ was applied voxel-wise with a probability of 0.15.

7.6.2 Matricize Implementation Details

Algorithm 6 presents PyTorch implementations of Local and SW Matricize modules. Here, we use the `einops` library to reshape tensors simply by declaring the corresponding Einstein notations.

Algorithm 6: PyTorch pseudocode of Local and SW Matricize modules.

```

1  import torch
2  from torch import nn
3  from einops.layers.torch import Rearrange
4
5
6  class LocalMatricize(nn.Module):
7      def __init__(self, size, head_dim=8, patch_size=(8, 8, 8)):
8          super().__init__()
9          C, H, W, D = size
10         c2, (h2, w2, d2) = head_dim, patch_size
11         c1, h1, w1, d1 = C // c2, H // h2, W // w2, D // d2
12         dims = dict(c1=c1, h1=h1, w1=w1, d1=d1, c2=c2, h2=h2, w2=w2, d2=d2)
13
14         eq = "b (c1 c2) (h1 h2) (w1 w2) (d1 d2) -> (b c1 h1 w1 d1) c2 (h2 w2 d2)"
15
16         self.unfold = Rearrange(eq, **dims)
17
18         eq = "(b c1 h1 w1 d1) c2 (h2 w2 d2) -> b (c1 c2) (h1 h2) (w1 w2) (d1 d2)"
19         self.fold = Rearrange(eq, **dims)
20
21     def forward(self, x):
22         # x: (B, C, H, W, D)
23         return self.unfold(x)
24
25     def inverse_forward(self, x):
26         # x: (B, C, L)
27         return self.fold(x)
28
29
30  class SWMatricize(nn.Module):
31      def __init__(self, size, head_dim=8, patch_size=(8, 8, 8), shifts=(4, 4, 4)):
32          super().__init__()
33          self.local_matricize = LocalMatricize(size, head_dim, patch_size)
34          self.shifts = shifts
35          self.shifts_inv = tuple(-s for s in patch_size)
36
37     def forward(self, x):
38         # x: (B, C, H, W, D)
39         out1 = self.local_matricize(x)
40         out2 = torch.roll(x, self.shifts, (2, 3, 4))
41         out2 = self.local_matricize(out2)
42         return torch.cat((out1, out2))
43
44     def inverse_forward(self, x):
45         # x: (B, C, L)
46         B = x.shape[0]
47         out1 = self.local_matricize.inverse_forward(x[: (B // 2)])
48         out2 = self.local_matricize.inverse_forward(x[(B // 2) :])
49         out2 = torch.roll(out2, self.shifts_inv, (2, 3, 4))
50         return 0.5 * (out1 + out2)

```

Chapter 8

Low-Rank Convolutional Neural Networks for Brain Tumor Segmentation

ABSTRACT | The automated segmentation of brain tumors is crucial for various clinical purposes from diagnosis to treatment planning to follow-up evaluations. The vast majority of effective models for tumor segmentation are based on convolutional neural networks with millions of parameters being trained. Such complex models can be highly prone to overfitting especially in cases where the amount of training data is insufficient. In this work, we devise a 3D U-Net-style architecture with residual blocks, in which low-rank constraints are imposed on weights of the convolutional layers in order to reduce overfitting. Within the same architecture, this helps to design networks with several times fewer parameters. We investigate the effectiveness of the proposed technique on the BraTS 2020 challenge.

This chapter is an adapted version of [125] P. Ashtari, F. Maes, and S. Van Huffel, “Low-Rank Convolutional Networks for Brain Tumor Segmentation,” in *International MICCAI Brainlesion Workshop: BrainLes 2020. Lecture Notes in Computer Science*, 2021, pp. 470–480. doi: doi.org/10.1007/978-3-030-72084-1_42

8.1 Introduction

Gliomas are brain tumors with the highest mortality rate and prevalence. They can be classified into two grades: low-grade glioma (LGG) and high-grade glioma (HGG), with the former being less aggressive than the latter. Multiparametric MRI is widely used to diagnose and assess gliomas in clinical practice. The accurate segmentation of gliomas is crucial for various clinical purposes, including diagnosis, treatment planning, image-guided surgery, and follow-up evaluations. However, manual delineation of tumors is laborious, time-consuming, and expensive especially because experts need to deal with 3D images and several modalities; therefore, accurate computer-assisted methods are needed to automatically perform this task. Despite considerable advances in medical imaging, glioma segmentation is still a challenging task since tumors can vary dramatically in shape, structure, and location across patients and over time within a specific patient. Moreover, the growing tumor mass may displace and deform the surrounding normal brain tissues, as do resection cavities that are present after surgery.

The brain tumor segmentation challenge (BraTS) [25], [141], [156]–[158] aims to develop effective data-driven algorithms for brain tumor segmentation by providing a large dataset of annotated LGG and HGG 3D MRI scans, each with four MRI modalities (T1 weighted, post-contrast T1-weighted, T2-weighted, and FLAIR) rigidly co-registered, resampled to the voxel size 1mm^3 , and skull-stripped. The BraTS 2020 training dataset consists of 369 cases, each of which is manually annotated by up to 4 raters who label each voxel as enhancing tumor (ET), edema (ED), necrotic and non-enhancing tumor (NCR/NET), or everything else (see Figure 8.1). However, for evaluation, the 3 nested subregions, namely whole tumor (WT), tumor core (TC—i.e., the union of ED and NCR/NET), and enhancing tumor (ET), are used.

Recently, deep learning models, particularly convolutional neural networks (CNN), surpassed traditional computer vision methods for semantic segmentation. In contrast to the conventional approach based on hand-crafted features, CNNs are able to automatically learn high-level features adapted specifically to the task of brain tumor segmentation. Currently, the vast majority of effective CNNs for medical image segmentation are based on a U-Net [3] architecture with millions of trainable parameters. However, such complex models can be highly prone to overfitting especially in cases where the amount of training data is insufficient, which is usually the case for medical imaging. In this work, we introduce a new layer, called low-rank convolution, in which low-rank constraints are imposed to regularize weights and thus reduce overfitting. We make use of a 3D U-Net [85] architecture with residual modules [70] and further improve it by

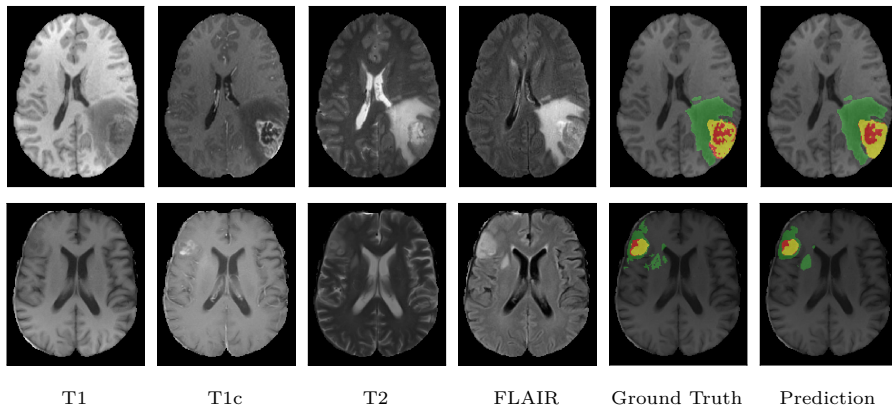


Figure 8.1 The multimodal MRI images along with the corresponding ground truth and prediction for a HGG (top row) and a LGG (bottom row) representative cases (green: edema, yellow: enhancing tumor, red: necrosis and non-enhancing tumor).

replacing ordinary convolution layers with low-rank ones, achieving models with several times fewer parameters than the initial ones. This leads to significantly better performance especially because the amount of training data is limited.

The rest of this chapter is organized as follows: Section 8.2 briefly reviews relevant semantic segmentation techniques. Section 8.3 presents our approach to brain tumor segmentation using low-rank convolutional layer. Experiments are presented in Section 8.4 . We conclude this chapter in Section 8.5 .

8.2 Related work

Over the past few years, considerable research efforts have been directed to the development of fully convolutional neural networks for semantic segmentation. Encoder-decoder architectures and their variants, in particular U-Net [3], are probably the most successful ones in the segmentation of medical images.

In BraTS 2018 and 2019, all top solutions made use of such architectures in their models one way or another. Isensee et al. [93] focused on the training procedure instead of proposing a new network, winning second place in BraTS 2018 by making only minor modifications to the standard 3D U-Net [85], using additional training data, and applying a simple post-processing technique. McKinley et al. [123] proposed an architecture, in which dense blocks [124] of dilated convolutions are embedded in a shallow U-Net-style network. Following

an encoder-decoder CNN architecture, Myronenko [86] won first place in BraTS 2018 by adding a branch to the encoder endpoint and taking a variational auto-encoder (VAE) approach. The winning model [159] in BraTS 2019, was based on a similar architecture but further employed a two-stage cascaded strategy.

CNNs are computationally demanding and memory-intensive. Since convolution operations comprise the vast bulk of computations in a deep CNN during both training and inference, methods have been proposed to speed up and compress convolutional layers. MobileNet [160] exploits depthwise separable filters to represent a standard convolution layer more compactly, leading to a substantial reduction in computational complexity at the cost of a small loss of accuracy. Inception [77] uses bottleneck architectures made of cheap 1x1 convolutions to limit the network size. These methods suggest new architectures by factorizing a convolution into smaller blocks.

An alternative approach is based on low-rank approximations [161], [162] and tensor decompositions [112], [163], [164], where the weights in a convolution layer are constrained to be low-rank. One advantage of this approach is that for a fixed architecture we can easily control the number of parameters and the computational complexity of the model by adjusting the rank. Furthermore, imposing low-rank constraints can regularize the model and reduce overfitting. In addition to speed-up, Tai et al. [113] achieved significant improvements in some cases using CNNs with low-rank regularization. Note that although all the mentioned techniques are only applied to the task of image classification, they can be also deployed effectively in an encoder-decoder architecture for image segmentation. The impact of low-rank regularization on the performance is expected to be greater when less data is available for training.

8.3 Method

In this section, we present our approach to brain tumor segmentation. The baseline architecture used is based on a 3D U-Net (Figure 8.2a) proposed in [85] and customized for BraTS 2018 by [93], except that here convolutional blocks at each level are replaced by ResNet blocks [70]. We also introduce a new layer, called low-rank convolution, as a regularization technique to reduce overfitting. By replacing ordinary convolution layers with low-rank ones, we can achieve significantly better performance especially when the amount of training data is insufficient, where the model is more prone to overfitting. In the following, we describe the training procedure and the building blocks of our networks.

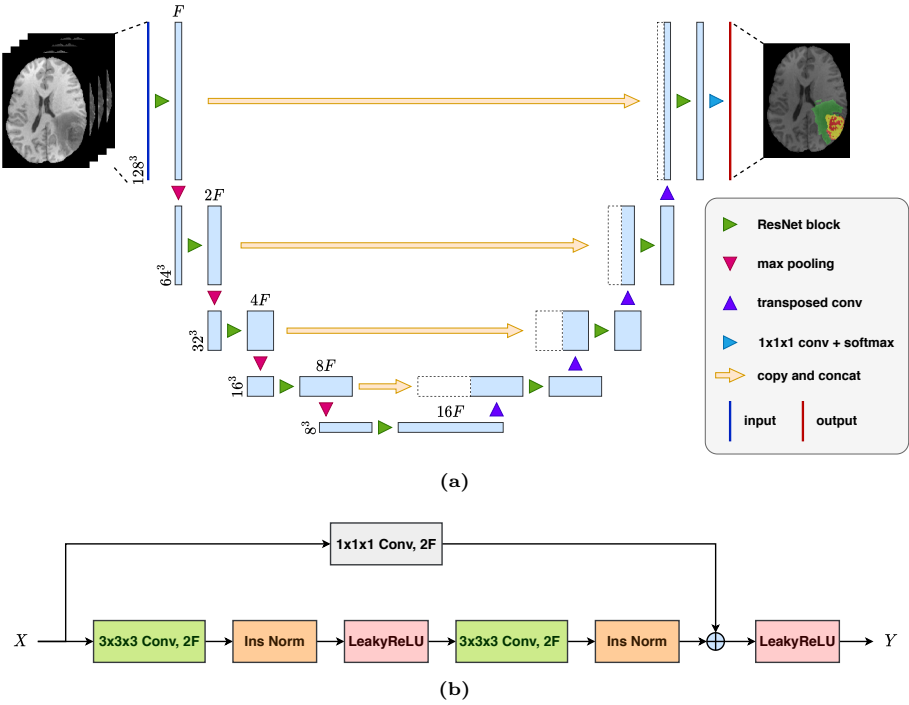


Figure 8.2 The U-Net architecture (a) and the ResNet block (b) used for brain tumor segmentation.

8.3.1 Data Preprocessing and Augmentation

The BraTS data is heterogeneous in the sense that it is multiparametric and acquired with different protocols at multiple institutions using various scanners, making intensity values nonstandardized. There is also high between-subject variability in tumors due to the presence of both low- and high-grade gliomas. To alleviate this heterogeneity and insufficiency of data, it is crucial to perform an effective preprocessing workflow before feeding the data into the network.

For each scan, we first form a 4-channel 3D image as the input, where each channel corresponds to one of the modalities (i.e. T1, post-contrast T1, T2, and FLAIR). We crop each image with a minimal box containing the whole brain region then resize it to the size $128 \times 128 \times 128$. Each channel of each image is then normalized independently using z-score to have intensities with zero mean and unit variance. Three data augmentation techniques are also utilized to reduce overfitting. Firstly, the input image is randomly flipped along the left-right axis.

Secondly, we apply a random affine transform (scale $\sim U(0.9, 1.1)$, rotation $\sim U(-10, 10)$). Finally, a Gaussian noise ($\mu = 0, \sigma \sim U(0, 0.25)$) is added to intensities per-channel.

8.3.2 Network Architecture

As mentioned before, our network, as shown in Figure 8.2a, follows a U-Net-like architecture made up of encoder and decoder parts. The network takes a 4-channel image of size $128 \times 128 \times 128$ and outputs a *probability map* with the same spatial size and with 4 channels that correspond to the 4 segmentation labels. The network has 4 levels, at each of which in the encoder (decoder) part, the input tensor is downscaled (upscaled) by a factor of two while the number of channels is doubled (halved). Downscaling and upscaling are performed via max-pooling and transposed convolution, respectively. In both the encoder and decoder, we use ResNet blocks [70], where each block is composed of convolution, Instance Normalization [76], and LeakyReLU activation layers (Figure 8.2b). Two $3 \times 3 \times 3$ convolutions are used in the residual mapping of each ResNet block, and a $1 \times 1 \times 1$ convolution is used in the shortcut connection in order to match the number of input channels with the number of output channels of the residual mapping. At the decoder endpoint, a $1 \times 1 \times 1$ convolution followed by a *softmax layer* is applied to get the segmentation probability map.

8.3.3 Low-rank Convolution

Convolutions form the backbone of a CNN. A typical U-Net has millions of training parameters, the majority of which are the weights that correspond to convolutional layers (this is true for any CNN with no fully-connected layers). In practice, such a complex model is very likely to overfit in particular when it comes to medical image segmentation applications, where the amount of annotated data is typically limited. Common regularization techniques like *dropout* [69] and *weight decay* can be used to mitigate this problem. However, these methods do not decrease the total number of parameters while modern CNNs are known to be heavily over-parameterized [165], i.e., the number of parameters exceeds the size of training data and what is theoretically sufficient. In this work, by imposing low-rank constraints on weights, we propose a new operation, termed Low-Rank Convolution (LRCONV), enabling the design of deep architectures with much fewer parameters but more robustness to overfitting. It is noteworthy that this idea is unrelated but complementary to other regularization techniques, such as dropout and weight decay.

Let tensor $\mathcal{X} \in \mathbb{R}^{C_{\text{in}} \times H \times W \times D}$ be the input of a 3D convolutional layer (for simplicity, we assume unit stride and dilation, and a zero-padded input) with the kernel tensor $\mathcal{V} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times H' \times W' \times D'}$ and bias $\mathbf{b} \in \mathbb{R}^{C_{\text{out}}}$. The output tensor $\tilde{\mathcal{X}} \in \mathbb{R}^{C_{\text{out}} \times H \times W \times D}$ is obtained as follows

$$\tilde{\mathcal{X}}_{chwd} = b_c + \sum_{c'=1}^{C_{\text{in}}} \sum_{h'=1}^{H'} \sum_{w'=1}^{W'} \sum_{d'=1}^{D'} \mathcal{X}_{c'(h+h')(w'+w)(d+d')} \mathcal{V}_{cc'h'w'd'}, \quad (8.1)$$

where (H, W, D) is the resolution of the input image; C_{in} and C_{out} denote the number of channels in the input and output, respectively; and (H', W', D') is the size of the convolution kernel. To define a low-rank convolution, we can use a Canonical Polyadic [108] form and re-parameterize the weight as a sum of rank-1 tensors

$$\mathcal{V} = \sum_{r=1}^R \mathbf{u}_r^{(1)} \circ \mathbf{u}_r^{(2)} \circ \mathbf{u}_r^{(3)} \circ \mathbf{u}_r^{(4)} \circ \mathbf{u}_r^{(5)}, \quad (8.2)$$

where “ \circ ” denotes the vector *outer product*; $\mathbf{u}_r^{(1)} \in \mathbb{R}^{C_{\text{out}}}$, $\mathbf{u}_r^{(2)} \in \mathbb{R}^{C_{\text{in}}}$, $\mathbf{u}_r^{(3)} \in \mathbb{R}^{H'}$, $\mathbf{u}_r^{(4)} \in \mathbb{R}^{W'}$, and $\mathbf{u}_r^{(5)} \in \mathbb{R}^{D'}$; and R is the rank. Equivalently, the above equation can be re-written elementwise as

$$\mathcal{V}_{cc'h'w'd'} = \sum_{r=1}^R U_{cr}^{(1)} U_{c'r}^{(2)} U_{h'r}^{(3)} U_{w'r}^{(4)} U_{d'r}^{(5)}, \quad (8.3)$$

where $\mathbf{U}^{(j)}$ is a *factor matrix* whose columns are $\{\mathbf{u}_1^{(j)}, \dots, \mathbf{u}_R^{(j)}\}$. Since the equation (8.3) can be treated as an Einstein summation, we can illustrate it using a *tensor network* diagram [108] as shown in Figure 8.3. In this chapter, we only use the Canonical Polyadic form, but other sparsely connected tensor network like Tucker and *tensor train* can be also utilized. In a LRCONV layer, the factor matrices and the bias are the parameters to be learned. The rank R is a hyperparameter by which we can control the number of parameters although, throughout the rest of this chapter, we tune the ratio $\alpha = R/C_{\text{in}}$ rather than R for controlling the layer complexity. Obviously, the smaller α , the fewer parameters the layer has.

It is worth noting that our approach exploits low-rank representations to regularize a CNN before the training in contrast to methods compressing pre-trained CNNs using low-rank approximations [161] and tensor decompositions [112].

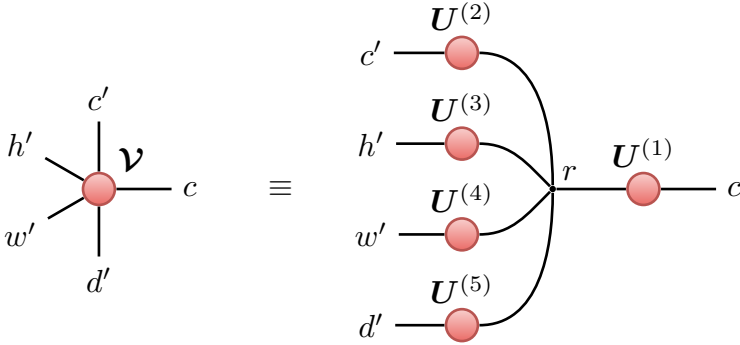


Figure 8.3 Canonical Polyadic tensor network. The 5th-order weight tensor (left panel) in a convolutional layer is represented with a Canonical Polyadic tensor network (right panel). In the graphical notation of an Einstein summation, nodes and edges denote the tensors and their corresponding indices, respectively.

8.3.4 Loss Function

The loss function used to train the network is the *soft Dice loss* [132], defined as

$$\ell_{\text{Dice}}(\mathcal{P}, \mathcal{G}) = 1 - \frac{2\langle \mathcal{P}, \mathcal{G} \rangle + 1}{\|\mathcal{P}\|^2 + \|\mathcal{G}\|^2 + 1} \quad (8.4)$$

where $\langle \cdot, \cdot \rangle$ denotes the *dot product* of tensors; $\|\cdot\|$ denotes the Frobenius norm of a tensor; \mathcal{P} is the predicted probability segmentation map (the out of the softmax layer); and \mathcal{G} is the *one-hot binary mask* encoding the corresponding ground truth. Both \mathcal{P} and \mathcal{G} are $4 \times 128 \times 128 \times 128$ tensors, where the 4 channels correspond to the 4 segmentation labels. We add 1 to both the numerator and denominator (sometimes known as *additive smoothing*) to smooth the loss and avoid division by zero. Although we optimize the Dice loss obtained by the labels, e.g. enhancing tumor, edema, necrosis and non-enhancing, we also monitor the Dice for the three overlapping regions, i.e., whole tumor, tumor core, and enhancing tumor.

8.3.5 Optimization

All networks are trained for 50 epochs, with a batch size of one. We use Adam optimizer with initial learning rate of 10^{-4} and regularize models with ℓ_2 weight decay of 10^{-5} . The learning rate is scheduled to decrease by a factor of 5 if the validation metric sees no improvement within 5 epochs. The training set

is randomly split into 80% (298 cases) used for training and the rest 20% (73 cases) used for validation.

8.3.6 Postprocessing

For a given test image in the inference phase, the probability map from the network output is resized to its original size. The map is then padded to have the same size as it had before cropping in the preprocessing step. We need to process the resulting probability map to obtain the final binary segmentation mask. The most trivial way is to select the label with the highest probability for each voxel. However, this does not exploit the fact that the tumor subregions, i.e., necrosis, tumor core, and whole tumor, are nested within each other. To overcome this shortcoming, we perform a hierarchical scheme, where the whole tumor is first extracted by thresholding the probability map. Having restricted to the voxels of the whole tumor, the edema channel of the probability map is then thresholded to extract the tumor core. The final threshold is applied to separate the necrosis and enhanced tumor within the tumor core. As shown in the next section, this postprocessing improves the performance significantly, with the Hausdorff distance of enhancing tumor decreasing by $\approx 15\%$.

8.4 Experiments and Results

All the models were implemented using PyTorch [129] and PyTorch Lightning [130] frameworks and trained on a NVIDIA P100 SXM2 GPU. We experimented with different values for the initial number of feature maps (denoted by F in Figure 8.2) and found the larger values to perform better although the GPU memory limitation did not allow us to try values greater than 60. We noticed that low-rank convolution can improve the results particularly for larger networks, roughly those with $F > 40$. We obtain the low-rank version of a U-Net by replacing $3 \times 3 \times 3$ ordinary convolutions with low-rank ones in all except the first level (the number of parameters at the first level is already relatively small). As seen in Figure 8.4, the networks with low-rank convolutions converge faster, attributed to the fact that the number of training parameters of the initial networks is dramatically reduced (see Table 8.1), resulting in much less complex models and optimization problems. The impact of low-rank regularization on convergence and performance is more substantial for the case $F = 60$ (with 92 million parameters) compared to the case $F = 50$ (with 64 million parameters), which is somewhat expected since the former has far more parameters.

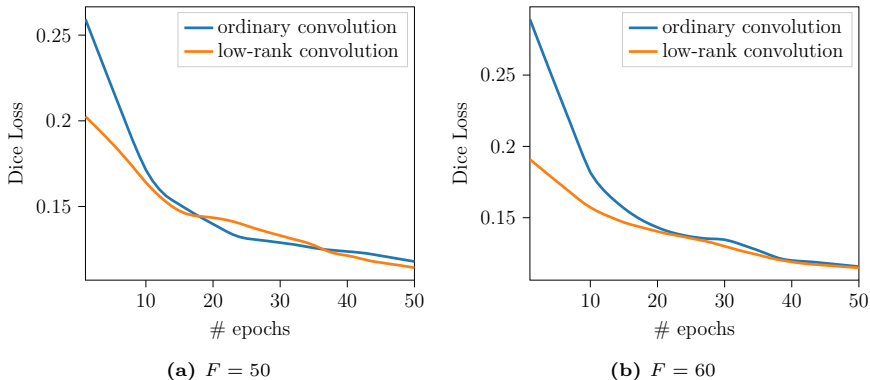


Figure 8.4 Validation metric plotted against epochs for two different values of F (the initial number of feature maps). The low-rank convolutions with $\alpha = 0.5$ are used. For visualization purposes, the curves are smooth via LOESS [166].

The results on the BraTS 2020 validation set (125 cases) obtained by the BraTS online evaluation framework are reported in Table 8.1. Our base model follows the Res-U-Net architecture (with $F = 60$ initial feature maps) described in Section 8.3.2. By postprocessing the probability maps using the hierarchical scheme described in Section 8.3.6, we achieved an improvement, particularly on the validation Dice of enhancing tumor. The results were further improved by training the low-rank version (with $\alpha = 0.5$) of the network, with the Hausdorff distance of enhancing tumor falling from 32.22 to 25.01. The low-rank network with 20.8 million parameters is far more memory efficient than the initial network with 92.2 million parameters. Finally, an ensemble of 8 models, including networks with $F \in \{30, 40, 50, 60\}$ and their low-rank versions (with $\alpha = 0.5$), was used to reduce the variance in predictions. To build the ensemble model, the probability maps were first averaged and then thresholded via the hierarchical scheme. This further improved all the scores, leading to the Dice score of over 90% for the whole tumor. The greatest improvement was observed for the tumor core, with the Dice score increasing from 77.84% to 79.1% and the Hausdorff distance dropping from 16.96 mm to 7.76 mm. Figure 8.1 shows the results of this model on representative HGG and LGG cases.

Table 8.2 presents the performance of the method on the BraTS 2020 test set. The average Dice scores for enhancing tumor core, whole tumor, and tumor core are 77.73%, 87.37%, and 81.24%, respectively. The corresponding values of Hausdorff distance are 16.21 mm, 6.28 mm, and 20.52 mm, respectively. Overall, the test results are consistent with those of the validation set, which reflects our model is neither biased towards the validation set nor has a high variance.

Table 8.1 The average scores on the BraTS 2020 validation set (125 cases). They are computed by the online evaluation platform (WT: whole tumor, TC: tumor core, and ET: enhancing tumor).

Model	#Params	Dice (%) \uparrow			HD95 (mm) \downarrow		
		ET	TC	WT	ET	TC	WT
Res-U-Net	92.2M	71.99	89.68	78.79	38.18	5.74	14.49
Res-U-Net + post	92.2M	73.32	89.84	78.77	32.22	5.75	14.46
Low-rank Res-U-Net + post	20.8M	74.82	89.12	77.84	25.01	7.61	16.94
Ensemble of 8 models	—	75.06	90.37	79.1	25.61	5.23	7.76

Table 8.2 Summary statistics of the scores on the BraTS 2020 test set (166 cases). They are computed by the online evaluation platform (WT: whole tumor, TC: tumor core, and ET: enhancing tumor).

	Dice (%) \uparrow			HD95 (mm) \downarrow		
	ET	TC	WT	ET	TC	WT
Mean	77.73	87.37	81.24	16.21	6.28	20.52
StdDev	21.6	13.93	25.45	69.39	11.76	74.75
Median	83.97	91.58	91.05	2.0	3.25	2.83
25th quantile	75.08	86.60	82.82	1.41	2.0	1.49
75th quantile	89.59	94.54	94.92	3.0	5.72	5.34

8.5 Conclusion

In this chapter, we proposed a regularization technique for CNNs by re-parameterizing convolutional layers as a low-rank structure, particularly canonical polyadic form. We devised a U-Net architecture with ResNet blocks consisting of low-rank convolutions. We examined the impact of this low-rank regularization on performance, verifying its effectiveness for brain tumor segmentation in multimodal MRI scans. The results on the BraTS 2020 data show that despite having much fewer parameters, the low-rank networks can outperform the unregularized versions especially in terms of Dice coefficients and Hausdorff distances on the enhancing tumor.

Chapter 9

Conclusion

9.1 Contributions

The primary goal of this thesis was to develop accurate and efficient models for the segmentation of brain lesions—specifically brain tumors, MS lesions, and stroke lesions—in mpMRI scans. For each task, we introduced end-to-end deep learning models that mostly incorporated low-rank factorization techniques into their core components.

In Chapters 2-5, we extensively reviewed the literature, covering foundational concepts in MRI physics, machine learning, deep learning, and low-rank factorization techniques. This thorough background facilitated the introduction and comprehension of our novel contributions, presented in Chapters 6, 7, and 8.

In Chapter 6, we dealt with the task of segmenting and detecting new white matter lesions between two longitudinal 3D FLAIR images acquired from an MS patient at two different time points. This is a crucial step in following-up and quantifying disease progression in clinical practice. For this problem, we developed a U-Net-inspired architecture with pre-activation blocks, termed Pre-U-Net. To address the challenges of data scarcity and class imbalance, we effectively trained our models using data augmentation and deep supervision techniques. The performance of Pre-U-Net was assessed and evaluated on the MSSEG-2 dataset. Pre-U-Net achieved a Dice score of 40.3% and an F_1 score of 48.1%, surpassing the baseline models, U-Net and Res-U-Net. **Notably, we observed that Pre-U-Net, as indicated by the F_1 scores, proved more effective in detecting new lesions than the baselines, while its**

segmentation performance was on par with U-Net, as suggested by the Dice and HD scores.

In Chapter 7, which represents the core contribution of this thesis, we introduced Factorizer, a novel approach to medical image segmentation that synergizes deep neural networks with NMF. We noted that recent advancements have demonstrated that ViTs with hierarchical structures can achieve results comparable to leading CNNs across various computer vision tasks. However, the inherent lack of locality inductive bias in ViTs often causes them to underperform their CNN counterparts, particularly in low-data scenarios typical of medical image segmentation. We also emphasized the quadratic complexity of attention in these models, which necessitates the application of self-attention layers after some reduction in image resolution. This limits their ability to fully capture long-range contexts available at higher resolutions. To address these challenges, we introduced Factorizer—a family of deep models that leverage the power of low-rank approximation. By formulating NMF as a differentiable GPU-friendly layer within a U-shaped architecture, we presented a scalable and interpretable solution for context modeling. Grounded in NMF and the shifted window concept, Swin Factorizer emerged as the most effective variant of Factorizer, competing favorably with CNN and transformer baselines in terms of accuracy and scalability. Specifically, Swin Factorizer achieved state-of-the-art results on BraTS for brain tumor segmentation, with Dice scores of 79.33%, 83.14%, and 90.16% for enhancing tumor, tumor core, and whole tumor, respectively. It also excelled on ISLES’22 for stroke lesion segmentation, achieving a Dice score of 76.49%. **Our experiments on brain tumor segmentation also revealed that NMF components are highly meaningful, with each component offering a distinct interpretation by differentiating certain tumor subregions from others. This provides a significant interpretability advantage for Factorizers over both CNNs and transformers. Furthermore, our ablation studies unveiled a unique characteristic of Factorizers: by training a Factorizer model, we also obtain lightweight yet accurate versions for free without the need to re-train any model from scratch. This can be accomplished by either reducing the number of iterations in the NMF algorithm or by ablating some expensive NMF layers (e.g., those at high-resolution stages) during the *inference phase*. In terms of model acceleration, this presents an added benefit for Factorizers compared to CNNs and transformers, which require more intricate mechanisms to produce their faster variants.**

In Chapter 8, we introduced a regularization technique for CNNs that re-parameterizes convolutional layers using a low-rank tensor network, specifically the canonical polyadic decomposition format. We developed a U-Net architecture

integrated with ResNet blocks that employ these low-rank convolutions. **Our investigation into the impact of this low-rank regularization revealed its efficacy for brain tumor segmentation in mpMRI scans.** Results from the BraTS 2020 dataset demonstrate that, even with significantly fewer parameters, the low-rank networks can surpass their unregularized counterparts, particularly in terms of Dice coefficients and Hausdorff distances for the enhancing tumor.

In summary, this thesis integrates low-rank factorization into deep learning to enhance medical image segmentation in two distinct ways:

- A *differentiable NMF layer* combined with matricization operations, which transforms an input tensor into matrices. This approach adeptly captures both local and global contexts.
- A *low-rank convolutional layer* that employs the CP format to re-parameterize the convolution kernel. This results in fewer parameters, aiding in model regularization and mitigating overfitting.

Our NMF layer presents an efficient alternative to the attention mechanism. Similarly, our low-rank convolution serves as a lightweight substitute for standard convolution. We seamlessly integrated both modules into a symmetric U-shaped architecture, which consist of a downsampling encoder, an upsampling decoder, and intermediate skip connections. Moreover, we have developed an open-source modular package that facilitates brain lesion segmentation and allows users to construct and tailor our proposed models for diverse applications.

To our knowledge, the research presented in this thesis represents pioneering efforts in combining low-rank techniques, specifically NMF and CP, into deep learning within the field of medical imaging. Both the NMF layer and the low-rank convolution stand as low-complexity building blocks for devising lightweight models adept at processing 3D images. Notably, the NMF layer enhances model interpretability due to the inherently meaningful components of NMF. This advancement edges us closer to models that are less of a “black box,” a crucial development for clinical practice. For instance, clinicians can examine these components layer by layer, gaining insights into the evolution of the final segmentation map across the layers of a Factorizer model.

9.2 Future Perspectives

While this thesis represents a major step forward in integrating low-rank techniques with deep learning for scalable and interpretable medical image segmentation, related ideas still deserve exploration to address the unresolved challenges associated with our work. In this section, we provide an overview of these challenges and outline potential solutions to guide future research.

Novel Matrix Factorization Methods

The NMF layer in Factorizer requires nonnegative input, necessitating the use of ReLU activation beforehand. Exploring other NMF variants, such as Semi-NMF and Convex-NMF, which accommodate mixed-signed data matrices, could eliminate the need for the ReLU activation function prior to factorization.

Our Factorizer approach utilized a consistent rank across all NMF layers and inputs. However, the rank is intrinsically tied to the input data structure and may differ between inputs. Investigating automated rank selection methods, like a greedy NMF approach (aka nonnegative matrix underapproximation), where components are sequentially added until specific criteria are satisfied, could be beneficial. This can especially improve Local/Swin Factorizer, where different windows might require varying optimal ranks based on context.

Another challenge with our NMF layer was the potential for vanishing or exploding gradients when increasing the NMF algorithm iterations. This is due to the lack of mechanisms like normalization or skip connections across iterations. Addressing this might involve modifying the MU and HALS update formulas to stabilize gradients.

Tensor Factorization Methods

While NMF was effective in our context model, its application is limited to matrices. Given that images are 2D or 3D arrays, we employed matricization operations in Factorizer, which unfortunately disrupt the multidimensional data structure. A potential solution is to devise tensorization operations that maintain the data's structure. Implementing tensor factorization methods as a differentiable layer could then fully leverage this structure for context modeling. For instance, nonnegative tensor factorization using an algorithm akin to MU could be a viable option.

Towards Blind Deconvolution

The Local/SW matricize, as employed in the Local/Swin Factorizer, splits an input tensor into non-overlapping patches. We observed that this results in a discontinuous representation of voxels near the boundaries of partitioning windows, often appearing as a grid pattern in NMF components.

A potential solution might involve gathering all possible patches. However, this approach proved computationally intensive, and its substantial memory requirements made it impractical for the 3D segmentation tasks addressed in this thesis. A promising alternative we are exploring is the blind deconvolution approach, which directly represents the input as the convolution of a source with a filter. While certain blind deconvolution algorithms, such as the Richardson–Lucy algorithm, bear similarities to the MU algorithm for NMF, they are not expected to produce the grid effect seen in the Local/Swin Factorizer. This is because they eliminate the need for (de)matrix methods. Moreover, the main operation in the Richardson–Lucy update formula is convolution, which involves sliding a window across the entire input tensor. This operation can be executed efficiently with minimal memory footprint on modern GPUs.

Self-supervised Pre-training

One of the biggest challenges in medical imaging is the scarcity of labeled data. Manual annotation of medical images is time-consuming, requires expert knowledge, and can be expensive. Self-supervised learning can exploit the vast amounts of unlabeled medical images by designing tasks where the model learns by predicting parts of the data from other parts (e.g., predicting a missing patch in an image).

Once a model is pre-trained using a self-supervised task, it can be fine-tuned on a smaller labeled dataset specific to the segmentation task at hand. This process of transfer learning can lead to better performance than training on the small labeled dataset alone. In fact, self-supervised pre-training can act as a form of regularization, reducing the risk of overfitting when the model is subsequently fine-tuned on a small labeled dataset. The improvement is expected to be more significant for transforms and Factorizers, which impose less inductive bias compared to CNNs.

To conclude, we believe that this thesis can pave the way for marrying more advanced low-rank factorization techniques with deep neural networks in order to develop lightweight, interpretable models for the more accurate segmentation of medical images.

Bibliography

- [1] Q. T. Ostrom, M. Price, C. Neff, *et al.*, “CBTRUS statistical report: Primary brain and other central nervous system tumors diagnosed in the United States in 2015–2019”, *Neuro-oncology*, vol. 24, no. Supplement_5, pp. v1–v95, 2022.
- [2] MSIF Consortium, *Atlas of MS*. The Multiple Sclerosis International Federation (MSIF), 2020. [Online]. Available: <https://www.msif.org/wp-content/uploads/2020/10/Atlas-3rd-Edition-Epidemiology-report-EN-updated-30-9-20.pdf>.
- [3] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.
- [4] V. L. Feigin, M. Brainin, B. Norrving, *et al.*, “World Stroke Organization (WSO): Global Stroke Fact Sheet 2022”, *International Journal of Stroke*, vol. 17, no. 1, pp. 18–29, 2022.
- [5] Y. Panagakis, J. Kossaiji, G. G. Chrysos, *et al.*, “Tensor methods in computer vision and deep learning”, *Proceedings of the IEEE*, vol. 109, no. 5, pp. 863–890, 2021.
- [6] K. Hayashi, T. Yamaguchi, Y. Sugawara, and S.-i. Maeda, “Exploring unexplored tensor network decompositions for convolutional neural networks”, *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [7] P. Liu, Z.-F. Gao, W. X. Zhao, Z.-Y. Xie, Z.-Y. Lu, and J.-R. Wen, “Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators”, *arXiv preprint arXiv:2106.02205*, 2021.

- [8] Y. Ren, B. Wang, L. Shang, X. Jiang, and Q. Liu, “Exploring extreme parameter compression for pre-trained language models”, *arXiv preprint arXiv:2205.10036*, 2022.
- [9] F. Bloch, “Nuclear induction”, *Physical review*, vol. 70, no. 7-8, p. 460, 1946.
- [10] E. M. Purcell, H. C. Torrey, and R. V. Pound, “Resonance absorption by nuclear magnetic moments in a solid”, *Physical review*, vol. 69, no. 1-2, p. 37, 1946.
- [11] “The Quantum Mechanical Basis of Precession and Excitation”, in *Magnetic Resonance Imaging*. John Wiley & Sons, Ltd, 2014, ch. 5, pp. 67–83, ISBN: 9781118633953. DOI: <https://doi.org/10.1002/9781118633953.ch5>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118633953.ch5>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118633953.ch5>.
- [12] R. H. Hashemi, W. G. Bradley, and C. J. Lisanti, *MRI: the basics: The Basics*. Lippincott Williams & Wilkins, 2012.
- [13] “The Quantum Mechanical Basis of Thermal Equilibrium and Longitudinal Relaxation”, in *Magnetic Resonance Imaging*. John Wiley & Sons, Ltd, 2014, ch. 6, pp. 85–94, ISBN: 9781118633953. DOI: <https://doi.org/10.1002/9781118633953.ch6>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118633953.ch6>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118633953.ch6>.
- [14] “Magnetization, Relaxation, and the Bloch Equation”, in *Magnetic Resonance Imaging*. John Wiley & Sons, Ltd, 2014, ch. 4, pp. 53–66, ISBN: 9781118633953. DOI: <https://doi.org/10.1002/9781118633953.ch4>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118633953.ch4>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118633953.ch4>.
- [15] “Multi-Dimensional Fourier Imaging and Slice Excitation”, in *Magnetic Resonance Imaging*. John Wiley & Sons, Ltd, 2014, ch. 10, pp. 165–206, ISBN: 9781118633953. DOI: <https://doi.org/10.1002/9781118633953.ch10>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118633953.ch10>. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118633953.ch10>.
- [16] J. V. Hajnal, D. J. Bryant, L. Kasuboski, *et al.*, “Use of fluid attenuated inversion recovery (FLAIR) pulse sequences in MRI of the brain”, *Journal of computer assisted tomography*, vol. 16, no. 6, pp. 841–844, 1992.

- [17] T. Okuda, Y. Korogi, Y. Shigematsu, *et al.*, “Brain Lesions: When Should Fluid-Attenuated Inversion-Recovery Sequences Be Used in MR Evaluation?”, *Radiology*, vol. 212, no. 3, pp. 793–798, 1999.
- [18] M. R. H. Petzsche, E. de la Rosa, U. Hanning, *et al.*, “ISLES 2022: A multi-center magnetic resonance imaging stroke lesion segmentation dataset”, *arXiv preprint arXiv:2206.06694*, 2022.
- [19] E. O. Stejskal and J. E. Tanner, “Spin Diffusion Measurements: Spin Echoes in the Presence of a Time-Dependent Field Gradient”, *The journal of chemical physics*, vol. 42, no. 1, pp. 288–292, 1965.
- [20] J. G. Smirniotopoulos, F. M. Murphy, E. J. Rushing, J. H. Rees, and J. W. Schroeder, “Patterns of Contrast Enhancement in the Brain and Meninges”, *Radiographics*, vol. 27, no. 2, pp. 525–551, 2007.
- [21] M. Smits and M. J. van den Bent, “Imaging Correlates of Adult Glioma Genotypes”, *Radiology*, vol. 284, no. 2, pp. 316–331, 2017.
- [22] M. Essig, N Anzalone, S. Combs, *et al.*, “MR Imaging of Neoplastic Central Nervous System Lesions: Review and Recommendations for Current Practice”, *American journal of neuroradiology*, vol. 33, no. 5, pp. 803–817, 2012.
- [23] K. Kono, Y. Inoue, K. Nakayama, *et al.*, “The Role of Diffusion-Weighted Imaging in Patients with Brain Tumors”, *American journal of neuroradiology*, vol. 22, no. 6, pp. 1081–1088, 2001.
- [24] M. Law, R. J. Young, J. S. Babb, *et al.*, “Gliomas: Predicting Time to Progression or Survival with Cerebral Blood Volume Measurements at Dynamic Susceptibility-weighted Contrast-enhanced Perfusion MR Imaging”, *Radiology*, vol. 247, no. 2, pp. 490–498, 2008.
- [25] B. H. Menze, A. Jakab, S. Bauer, *et al.*, “The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)”, *IEEE Transactions on Medical Imaging*, vol. 34, no. 10, pp. 1993–2024, 2014.
- [26] O. Commowick, A. Istace, M. Kain, *et al.*, “Objective Evaluation of Multiple Sclerosis Lesion Segmentation using a Data Management and Processing Infrastructure”, *Scientific reports*, vol. 8, no. 1, pp. 1–17, 2018.
- [27] R. L. Siegel, K. D. Miller, N. S. Wagle, and A. Jemal, “Cancer statistics, 2023”, *CA: a cancer journal for clinicians*, vol. 73, no. 1, pp. 17–48, 2023.
- [28] K. D. Miller, Q. T. Ostrom, C. Kruchko, *et al.*, “Brain and other central nervous system tumor statistics, 2021”, *CA: a cancer journal for clinicians*, vol. 71, no. 5, pp. 381–406, 2021.

- [29] K. R. Porter, B. J. McCarthy, S. Freels, Y. Kim, and F. G. Davis, "Prevalence estimates for primary brain tumors in the United States by age, gender, behavior, and histology", *Neuro-oncology*, vol. 12, no. 6, pp. 520–527, 2010.
- [30] P. Y. Wen and S. Kesari, "Malignant Gliomas in Adults", *New England Journal of Medicine*, vol. 359, no. 5, pp. 492–507, 2008.
- [31] D. N. Louis, A. Perry, G. Reifenberger, *et al.*, "The 2016 World Health Organization Classification of Tumors of the Central Nervous System: a summary", *Acta neuropathologica*, vol. 131, pp. 803–820, 2016.
- [32] R. Dobson and G. Giovannoni, "Multiple sclerosis—a review", *European journal of neurology*, vol. 26, no. 1, pp. 27–40, 2019.
- [33] C. Baecher-Allan, B. J. Kaskow, and H. L. Weiner, "Multiple sclerosis: mechanisms and immunotherapy", *Neuron*, vol. 97, no. 4, pp. 742–768, 2018.
- [34] A. Ascherio and K. L. Munger, "Environmental Risk Factors for Multiple Sclerosis. Part I: The Role of Infection", *Annals of neurology*, vol. 61, no. 4, pp. 288–299, 2007.
- [35] A. Scalfari, V. Knappertz, G. Cutter, D. S. Goodin, R. Ashton, and G. C. Ebers, "Mortality in patients with multiple sclerosis", *Neurology*, vol. 81, no. 2, pp. 184–192, 2013, ISSN: 0028-3878. DOI: 10.1212/WNL.0b013e31829a3388. eprint: <https://n.neurology.org/content/81/2/184.full.pdf>. [Online]. Available: <https://n.neurology.org/content/81/2/184>.
- [36] G. A. Donnan, M. Fisher, M. Macleod, and S. M. Davis, "Stroke", *The Lancet*, vol. 371, no. 9624, pp. 1612–1623, May 2008. DOI: 10.1016/S0140-6736(08)60694-7. [Online]. Available: [https://doi.org/10.1016/S0140-6736\(08\)60694-7](https://doi.org/10.1016/S0140-6736(08)60694-7).
- [37] A. I. Qureshi, A. D. Mendelow, and D. F. Hanley, "Intracerebral haemorrhage", *The Lancet*, vol. 373, no. 9675, pp. 1632–1644, 2009.
- [38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [39] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: probml.ai.
- [40] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems", *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
- [41] R. Tibshirani, "Regression shrinkage and selection via the lasso", *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

- [42] D. Bertsekas, *Nonlinear programming*, en. Athena Scientific, Sep. 2016.
- [43] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning”, in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, Jun. 2013, pp. 1139–1147. [Online]. Available: <https://proceedings.mlr.press/v28/sutskever13.html>.
- [44] L. Bottou and O. Bousquet, “The Tradeoffs of Large Scale Learning”, in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds., vol. 20, Curran Associates, Inc., 2007. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.
- [45] L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1—learning rate, batch size, momentum, and weight decay”, *arXiv preprint arXiv:1803.09820*, 2018.
- [46] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.”, *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [47] T. Tieleman, G. Hinton, *et al.*, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, 2012.
- [48] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *arXiv preprint arXiv:1412.6980*, 2014.
- [49] S. J. Wright, “Coordinate descent algorithms”, *Mathematical programming*, vol. 151, no. 1, pp. 3–34, 2015.
- [50] K. Lange, *MM optimization algorithms*. SIAM, 2016.
- [51] E. Parzen, “On estimation of a probability density function and mode”, *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.
- [52] M. Rosenblatt, “Remarks on some nonparametric estimates of a density function”, *The annals of mathematical statistics*, pp. 832–837, 1956.
- [53] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC press, 1986, vol. 26.
- [54] M. Rudemo, “Empirical choice of histograms and kernel density estimators”, *Scandinavian Journal of Statistics*, pp. 65–78, 1982.
- [55] A. W. Bowman, “An alternative method of cross-validation for the smoothing of density estimates”, *Biometrika*, vol. 71, no. 2, pp. 353–360, 1984.
- [56] E. A. Nadaraya, “On estimating regression”, *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964.

- [57] G. S. Watson, “Smooth regression analysis”, *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 359–372, 1964.
- [58] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [59] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines”, in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
- [60] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models”, in *Proc. icml*, Atlanta, GA, vol. 30, 2013, p. 3.
- [61] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.
- [62] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus)”, *arXiv preprint arXiv:1511.07289*, 2015.
- [63] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus)”, *arXiv preprint arXiv:1606.08415*, 2016.
- [64] Y. LeCun, B. Boser, J. S. Denker, *et al.*, “Backpropagation Applied to Handwritten Zip Code Recognition”, *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.
- [65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.
- [66] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
- [67] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [68] L. Breiman, “Random forests”, *Machine learning*, vol. 45, pp. 5–32, 2001.
- [69] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

- [70] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.
- [71] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 448–456. [Online]. Available: <https://proceedings.mlr.press/v37/ioffe15.html>.
- [72] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How Does Batch Normalization Help Optimization?”, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.
- [73] P. Luo, X. Wang, W. Shao, and Z. Peng, “Towards understanding regularization in batch normalization”, *arXiv preprint arXiv:1809.00846*, 2018.
- [74] Y. Wu and K. He, “Group Normalization”, in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.
- [75] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization”, *arXiv preprint arXiv:1607.06450*, 2016.
- [76] D. Ulyanov, A. Vedaldi, and V. Lempitsky, “Instance Normalization: The Missing Ingredient for Fast Stylization”, *arXiv preprint arXiv:1607.08022*, 2016.
- [77] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going Deeper with Convolutions”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [78] K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks”, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer, 2016, pp. 630–645, ISBN: 978-3-319-46493-0.
- [79] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention Is All You Need”, in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.
- [80] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *arXiv preprint arXiv:2010.11929*, 2020.
- [81] Z. Liu, Y. Lin, Y. Cao, *et al.*, “Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 10 012–10 022.

- [82] M. M. Naseer, K. Ranasinghe, S. H. Khan, M. Hayat, F. Shahbaz Khan, and M.-H. Yang, “Intriguing Properties of Vision Transformers”, in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 23 296–23 308. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/c404a5adb90e09631678b13b05d9d7a-Paper.pdf.
- [83] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate”, *arXiv preprint arXiv:1409.0473*, 2014.
- [84] K. Choromanski, V. Likhoshesterov, D. Dohan, *et al.*, “Rethinking Attention with Performers”, *arXiv preprint arXiv:2009.14794*, 2020.
- [85] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation”, in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, S. Ourselin, L. Joskowicz, M. R. Sabuncu, G. Unal, and W. Wells, Eds., Cham: Springer, 2016, pp. 424–432, ISBN: 978-3-319-46723-8.
- [86] A. Myronenko, “3D MRI Brain Tumor Segmentation Using Autoencoder Regularization”, in *International MICCAI Brainlesion Workshop: BrainLes 2018. Lecture Notes in Computer Science*, A. Crimi, S. Bakas, *et al.*, Eds., vol. 11384, Cham: Springer, 2019, pp. 311–320, ISBN: 978-3-030-11726-9.
- [87] J. Chen, Y. Lu, Q. Yu, *et al.*, “TransUnet: Transformers Make Strong Encoders for Medical Image Segmentation”, *arXiv preprint arXiv:2102.04306*, 2021.
- [88] W. Wang, C. Chen, M. Ding, H. Yu, S. Zha, and J. Li, “TransBTS: Multimodal Brain Tumor Segmentation Using Transformer”, in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*, M. de Bruijne, P. C. Cattin, S. Cotin, *et al.*, Eds., Cham: Springer, 2021, pp. 109–119, ISBN: 978-3-030-87193-2.
- [89] A. Hatamizadeh, Y. Tang, V. Nath, *et al.*, “UNETR: Transformers for 3D Medical Image Segmentation”, in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2022, pp. 574–584.
- [90] A. Hatamizadeh, V. Nath, Y. Tang, D. Yang, H. R. Roth, and D. Xu, “Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images”, in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi and S. Bakas, Eds., Cham: Springer, 2022, pp. 272–284, ISBN: 978-3-031-08999-2.

- [91] H. Cao, Y. Wang, J. Chen, *et al.*, “Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation”, *arXiv preprint arXiv:2105.05537*, 2021.
- [92] H.-Y. Zhou, J. Guo, Y. Zhang, L. Yu, L. Wang, and Y. Yu, “nnFormer: Interleaved Transformer for Volumetric Segmentation”, *arXiv preprint arXiv:2109.03201*, 2021.
- [93] F. Isensee, P. Kickingereder, W. Wick, M. Bendszus, and K. H. Maier-Hein, “No New-Net”, in *International MICCAI Brainlesion Workshop: BrainLes 2018. Lecture Notes in Computer Science*, A. Crimi, S. Bakas, *et al.*, Eds., Cham: Springer, 2019, pp. 234–244, ISBN: 978-3-030-11726-9.
- [94] F. Isensee, P. F. Jaeger, S. A. A. Kohl, J. Petersen, and K. H. Maier-Hein, “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”, *Nature Methods*, vol. 18, no. 2, pp. 203–211, Feb. 2021, ISSN: 1548-7105. DOI: 10.1038/s41592-020-01008-z. [Online]. Available: <https://doi.org/10.1038/s41592-020-01008-z>.
- [95] F. Isensee, P. F. Jäger, P. M. Full, P. Vollmuth, and K. H. Maier-Hein, “nnU-Net for Brain Tumor Segmentation”, in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi and S. Bakas, Eds., Cham: Springer International Publishing, 2021, pp. 118–132, ISBN: 978-3-030-72087-2.
- [96] P. Paatero and U. Tapper, “Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values”, *Environmetrics*, vol. 5, no. 2, pp. 111–126, 1994.
- [97] D. Lee and H. S. Seung, “Algorithms for Non-negative Matrix Factorization”, in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13, MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf.
- [98] D. D. Lee and H. S. Seung, “Learning the parts of objects by non-negative matrix factorization”, *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.
- [99] “Algorithms for nonnegative matrix factorization with the β -divergence”,
- [100] A. Cichocki, R. Zdunek, and S.-i. Amari, “Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization”, in *International conference on independent component analysis and signal separation*, Springer, 2007, pp. 169–176.
- [101] A. Cichocki and A.-H. Phan, “Fast Local Algorithms for Large Scale Nonnegative Matrix and Tensor Factorizations”, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 92, no. 3, pp. 708–721, 2009.
- [102] A. L. Cambridge. “The Olivetti Faces Dataset”. (1994), [Online]. Available: <https://cam-orl.co.uk/facedatabase.html>.

- [103] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior", in *2011 31st International Conference on Distributed Computing Systems Workshops*, 2011, pp. 166–171. DOI: 10.1109/ICDCSW.2011.20.
- [104] R. A. Harshman *et al.*, "Foundations of the PARAFAC procedure: Models and conditions for an " explanatory" multimodal factor analysis", 1970.
- [105] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition", *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.
- [106] L. R. Tucker, "Some mathematical notes on three-mode factor analysis", *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
- [107] I. V. Oseledets, "Tensor-train decomposition", *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
- [108] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor Networks for Dimensionality Reduction and Large-scale Optimization: Part 1 Low-Rank Tensor Decompositions", *Foundations and Trends® in Machine Learning*, vol. 9, no. 4-5, pp. 249–429, 2016.
- [109] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, Springer, 2014, pp. 818–833.
- [110] A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks", *Advances in neural information processing systems*, vol. 28, 2015.
- [111] Y. Pan, J. Xu, M. Wang, *et al.*, "Compressing recurrent neural networks with tensor ring for action recognition", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4683–4690.
- [112] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition", *arXiv preprint arXiv:1412.6553*, 2014.
- [113] C. Tai, T. Xiao, Y. Zhang, X. Wang, *et al.*, "Convolutional neural networks with low-rank regularization", *arXiv preprint arXiv:1511.06067*, 2015.
- [114] E. Stoudenmire and D. J. Schwab, "Supervised learning with tensor networks", *Advances in neural information processing systems*, vol. 29, 2016.

- [115] I. Glasser, R. Sweke, N. Pancotti, J. Eisert, and I. Cirac, “Expressive power of tensor-network factorizations for probabilistic modeling”, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/b86e8d03fe992d1b0e19656875ee557c-Paper.pdf.
- [116] H. Chen and T. Barthel, “Machine learning with tree tensor networks, CP rank constraints, and tensor dropout”, *arXiv preprint arXiv:2305.19440*, 2023.
- [117] N. Kargas and N. D. Sidiropoulos, “Supervised learning and canonical decomposition of multivariate functions”, *IEEE Transactions on Signal Processing*, vol. 69, pp. 1097–1107, 2021.
- [118] P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marinier, “New multiple sclerosis lesion segmentation and detection using pre-activation U-Net”, *Frontiers in Neuroscience*, vol. 16, p. 975 862, 2022.
- [119] S. Vukusic, R. Casey, F. Rollot, *et al.*, “Observatoire Français de la Sclérose en Plaques (OFSEP): A unique multimodal nationwide MS registry in France”, *Multiple Sclerosis Journal*, vol. 26, no. 1, pp. 118–122, 2020.
- [120] C Confavreux, D. Compston, O. Hommes, W. McDonald, and A. Thompson, “EDMUS, a European database for multiple sclerosis.”, *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 55, no. 8, pp. 671–676, 1992.
- [121] P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marinier, “Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-activation U-Net”, in *MSSEG-2 challenge proceedings: Multiple sclerosis new lesions segmentation challenge using a data management and processing infrastructure*, Strasbourg, France, Sep. 2021, pp. 45–51. [Online]. Available: <https://hal.inria.fr/hal-03358968v1/document/#page=54>.
- [122] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, “Deeply-Supervised Nets”, in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, G. Lebanon and S. V. N. Vishwanathan, Eds., ser. Proceedings of Machine Learning Research, vol. 38, San Diego, California, USA: PMLR, May 2015, pp. 562–570. [Online]. Available: <https://proceedings.mlr.press/v38/lee15a.html>.

- [123] R. McKinley, R. Meier, and R. Wiest, “Ensembles of Densely-Connected CNNs with Label-Uncertainty for Brain Tumor Segmentation”, in *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, A. Crimi, S. Bakas, H. Kuijf, F. Keyvan, M. Reyes, and T. van Walsum, Eds., Cham: Springer International Publishing, 2019, pp. 456–465, ISBN: 978-3-030-11726-9.
- [124] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [125] P. Ashtari, F. Maes, and S. Van Huffel, “Low-Rank Convolutional Networks for Brain Tumor Segmentation”, in *International MICCAI Brainlesion Workshop: BrainLes 2020. Lecture Notes in Computer Science*, A. Crimi and S. Bakas, Eds., Cham: Springer, 2021, pp. 470–480, ISBN: 978-3-030-72084-1.
- [126] S. Aslani, M. Dayan, L. Storelli, *et al.*, “Multi-Branch Convolutional Neural Network for Multiple Sclerosis Lesion Segmentation”, *NeuroImage*, vol. 196, pp. 1–15, 2019.
- [127] F. La Rosa, A. Abdulkadir, M. J. Fartaria, *et al.*, “Multiple sclerosis cortical and WM lesion segmentation at 3T MRI: a deep learning method based on FLAIR and MP2RAGE”, *NeuroImage: Clinical*, vol. 27, p. 102335, 2020.
- [128] N. Gessert, J. Krüger, R. Opfer, *et al.*, “Multiple sclerosis lesion activity segmentation with attention-guided two-path CNNs”, *Computerized Medical Imaging and Graphics*, vol. 84, p. 101772, 2020.
- [129] A. Paszke, S. Gross, F. Massa, *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library”, in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.
- [130] W. Falcon, “PyTorch Lightning”, *GitHub. Note: <https://github.com/PyTorchLightning/lightning> Cited by*, vol. 3, 2019.
- [131] M. J. Cardoso, W. Li, R. Brown, *et al.*, “MONAI: An open-source framework for deep learning in healthcare”, *arXiv preprint arXiv:2211.02701*, 2022.

- [132] F. Milletari, N. Navab, and S.-A. Ahmadi, “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation”, in *2016 Fourth International Conference on 3D Vision (3DV)*, 2016, pp. 565–571. DOI: 10.1109/3DV.2016.79.
- [133] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection”, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.
- [134] P. Ashtari, D. M. Sima, L. De Lathauwer, D. Sappey-Marinier, F. Maes, and S. Van Huffel, “Factorizer: A scalable interpretable approach to context modeling for medical image segmentation”, *Medical image analysis*, vol. 84, p. 102706, 2023.
- [135] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2017.
- [136] Z. Gu, J. Cheng, H. Fu, *et al.*, “CE-Net: Context Encoder Network for 2D Medical Image Segmentation”, *IEEE Transactions on Medical Imaging*, vol. 38, no. 10, pp. 2281–2292, 2019.
- [137] X. Wang, R. Girshick, A. Gupta, and K. He, “Non-local Neural Networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7794–7803.
- [138] J. Fu, J. Liu, H. Tian, *et al.*, “Dual Attention Network for Scene Segmentation”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3146–3154.
- [139] N. Gillis, “The Why and How of Nonnegative Matrix Factorization”, *Regularization, Optimization, Kernels, and port Vector Machines*, vol. 12, no. 257, pp. 257–291, 2014.
- [140] N. Gillis, *Nonnegative Matrix Factorization*. SIAM, 2020.
- [141] S. Bakas, H. Akbari, A. Sotiras, *et al.*, “Advancing the Cancer Genome Atlas Glioma MRI Collections with Expert Segmentation Labels and Radiomic Features”, *Scientific Data*, vol. 4, p. 170117, 2017.
- [142] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, “UNet++: A nested U-Net architecture for medical image segmentation”, in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Springer, 2018, pp. 3–11.
- [143] L. Yuan, Y. Chen, T. Wang, *et al.*, “Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 558–567.

- [144] W. Wang, E. Xie, X. Li, *et al.*, “Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 568–578.
- [145] H. Wu, B. Xiao, N. Codella, *et al.*, “CvT: Introducing Convolutions to Vision Transformers”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 22–31.
- [146] S. Zheng, J. Lu, H. Zhao, *et al.*, “Rethinking Semantic Segmentation From a Sequence-to-Sequence Perspective With Transformers”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 6881–6890.
- [147] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, “SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers”, *arXiv preprint arXiv:2105.15203*, 2021.
- [148] Y. Zhang, H. Liu, and Q. Hu, “TransFuse: Fusing Transformers and CNNs for medical image segmentation”, *arXiv preprint arXiv:2102.08005*, 2021.
- [149] J. M. J. Valanarasu, P. Oza, I. Hacihaliloglu, and V. M. Patel, “Medical Transformer: Gated Axial-Attention for Medical Image Segmentation”, in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*, M. de Bruijne, P. C. Cattin, S. Cotin, *et al.*, Eds., Cham: Springer, 2021, pp. 36–46, ISBN: 978-3-030-87193-2.
- [150] Y. Xie, J. Zhang, C. Shen, and Y. Xia, “CoTr: Efficiently Bridging CNN and Transformer for 3D Medical Image Segmentation”, in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*, M. de Bruijne, P. C. Cattin, S. Cotin, *et al.*, Eds., Cham: Springer, 2021, pp. 171–180, ISBN: 978-3-030-87199-4.
- [151] N. Sauwen, M. Acou, S. Van Cauwer, *et al.*, “Comparison of unsupervised classification methods for brain tumor segmentation using multi-parametric MRI”, *NeuroImage: Clinical*, vol. 12, pp. 753–764, 2016.
- [152] N. Sauwen, M. Acou, D. M. Sima, *et al.*, “Semi-automated brain tumor segmentation on multi-parametric MRI using regularized non-negative matrix factorization”, *BMC Medical Imaging*, vol. 17, no. 1, pp. 1–14, 2017.
- [153] Z. Geng, M.-H. Guo, H. Chen, X. Li, K. Wei, and Z. Lin, “Is Attention Better Than Matrix Decomposition?”, *arXiv preprint arXiv:2109.04553*, 2021.
- [154] L. Grippo and M. Sciandrone, “On the convergence of the block nonlinear Gauss-Seidel method under convex constraints”, *Operations research letters*, vol. 26, no. 3, pp. 127–136, 2000.

- [155] M. Antonelli, A. Reinke, S. Bakas, *et al.*, “The Medical Segmentation Decathlon”, *arXiv preprint arXiv:2106.05735*, 2021.
- [156] S. Bakas, M. Reyes, A. Jakab, *et al.*, “Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge”, *arXiv preprint arXiv:1811.02629*, 2018.
- [157] S. Bakas, H. Akbari, A. Sotiras, *et al.*, “Segmentation Labels and Radiomic Features for the Pre-Operative Scans of the TCGA-GBM Collection. The Cancer Imaging Archive”, *Nat Sci Data*, vol. 4, p. 170 117, 2017.
- [158] S. Bakas, H. Akbari, A. Sotiras, *et al.*, “Segmentation Labels and Radiomic Features for the Pre-Operative Scans of the TCGA-LGG Collection”, *The cancer imaging archive*, vol. 286, 2017.
- [159] Z. Jiang, C. Ding, M. Liu, and D. Tao, “Two-Stage Cascaded U-Net: 1st Place Solution to BraTS Challenge 2019 Segmentation Task”, in *International MICCAI Brainlesion Workshop*, Springer, 2019, pp. 231–241.
- [160] A. G. Howard, M. Zhu, B. Chen, *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, *arXiv preprint arXiv:1704.04861*, 2017.
- [161] M. Jaderberg, A. Vedaldi, and A. Zisserman, “Speeding up Convolutional Neural Networks with Low Rank Expansions”, *arXiv preprint arXiv:1405.3866*, 2014.
- [162] X. Yu, T. Liu, X. Wang, and D. Tao, “On compressing deep models by low rank and sparse decomposition”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7370–7379.
- [163] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, “Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications”, *arXiv preprint arXiv:1511.06530*, 2015.
- [164] M. Boussé, O. Debals, and L. De Lathauwer, “Tensor-based large-scale blind system identification using segmentation”, *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5770–5784, 2017.
- [165] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, “Predicting Parameters in Deep Learning”, in *Advances in neural information processing systems*, 2013, pp. 2148–2156.
- [166] W. S. Cleveland and S. J. Devlin, “Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting”, *Journal of the American statistical association*, vol. 83, no. 403, pp. 596–610, 1988.

Curriculum Vitae

Pooya Ashtari

Curriculum Vitae

ESAT - STADIUS
Kasteelpark Arenberg 10 - box 2446
3001 Leuven, Belgium
☎ +32 497 06 99 47
✉ pooya.ashtari@esat.kuleuven.be
🌐 pashtari
in pooya-ashtari

Education

- 2019–2023 **Double Ph.D. in Biomedical Engineering**, *KU Leuven*, Leuven, Belgium; and *University of Lyon I*, Lyon, France
(Co)supervisors: Sabine Van Huffel, Frederik Maes, and Dominique Sappey-Marinier
Thesis: Brain Lesion Segmentation and Detection on Multi-parametric MRI Data: Marrying Deep Learning with Low-rank Factorization
Comment: This Ph.D. is part of the INSPiRE-MED Marie Curie research network, founded by the European Union's Horizon 2020 research and innovation program.
- 2014–2017 **M.Sc. in Biomedical Engineering**, *Sharif University of Technology*, Tehran, Iran
Supervisor: Bijan Vosoughi Vahdat
Thesis: Bootstrap-based Ensemble Clustering of fMRI Time Series [pdf (in Persian)] [abstract (in English)] [presentation (in English)]
GPA: 18.08/20
- 2007–2012 **B.Sc. in Electrical Engineering (Communication Systems)**, *University of Zanjan*, Zanjan, Iran
Supervisor: Mostafa Charmi
Thesis: Diffusion Tensor Image Registration
GPA: 15.76/20, Major GPA: 16.64/20
- 2003–2006 **Diploma in Mathematics and Physics**, *Shahid Beheshti High School, NODET (National Organization for Development of Exceptional Talents)*, Zanjan, Iran
GPA: 18.11/20

Research Interests

- **Computer Vision and Medical Image Analysis:** (medical) image segmentation and detection, efficient context modeling
- **Machine Learning and Deep learning:** representation learning, self-supervised learning, model compression, low-rank factorization

Research Experience

- 2019–2024 **Research Associate**, *Bimedical Data Processing (BioMed)*, Department of Electrical Engineering (ESAT), KU Leuven, Leuven, Belgium
Supervisor: Sabine Van Huffel
- 2014–2017 **Research Student**, *Biomedical Signal and Image Processing Laboratory (BiSIPL)*, Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran
Supervisor: Bijan Vosoughi Vahdat

Publications

Journal Papers

- [1] **P. Ashtari**, D. M. Sima, L. De Lathauwer, D. Sappey-Marinier, F. Maes, and S. Van Huffel, "Factorizer: A Scalable Interpretable Approach to Context Modeling for Medical Image Segmentation," *Medical Image Analysis*, vol. 84, p. 102706, Feb. 2023, doi: 10.1016/j.media.2022.102706.
- [2] **P. Ashtari**, B. Barile, S. Van Huffel, and D. Sappey-Marinier, "New Multiple Sclerosis Lesion Segmentation and Detection Using Pre-Activation U-Net," *Frontiers in Neuroscience*, vol. 16, Oct. 2022, doi: 10.3389/fnins.2022.975862.

- [3] B. Barile, **P. Ashtari**, C. Stamile, A. Marzullo, F. Maes, F. Durand-Dubief, S. Van Huffel, and D. Sappey-Marinier, "Classification of Multiple Sclerosis Clinical Profiles Using Machine Learning and Grey Matter Connectome," *Frontiers in Robotics and AI*, p. 250, 2022. doi: 10.3389/frobt.2022.926255.
- [4] **P. Ashtari**, F. Nateghi Haredasht, and H. Beigy, "Supervised fuzzy partitioning," *Pattern Recognition*, vol. 97, p. 107013, 2020, doi: 10.1016/j.patcog.2019.107013.

Conference Proceedings Papers

- [1] B. Barile, **P. Ashtari**, F. Durand-Dubief, F. Maes, D. Sappey-Marinier, and S. Van Huffel, "A kernel based multilinear SVD approach for Multiple Sclerosis Profiles Classification," *ESANN 2022 proceedings*, 2022.
- [2] **P. Ashtari**, B. Barile, S. Van Huffel, and D. Sappey-Marinier, "Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-activation U-Net," in *MSSEG-2 challenge proceedings: Multiple sclerosis new lesions segmentation challenge using a data management and processing infrastructure*, Sep. 2021, pp. 45–51.
- [3] **P. Ashtari**, F. Maes, and S. Van Huffel, "Low-Rank Convolutional Networks for Brain Tumor Segmentation," in *International MICCAI Brainlesion Workshop: BrainLes 2020. Lecture Notes in Computer Science*, 2021, pp. 470–480. doi: doi.org/10.1007/978-3-030-72084-1.42.
- [4] S. Noei, **P. Ashtari**, M. Jahed, and B. V. Vahdat, "Classification of EEG signals using the spatio-temporal feature selection via the elastic net," in 2016 23rd Iranian Conference on Biomedical Engineering and 2016 1st International Iranian Conference on Biomedical Engineering (ICBME), 2016, pp. 232–236. doi: 10.1109/ICBME.2016.7890962.

Presentations

- Sept 2022 Presented "Factorizer: Marrying matrix factorization with deep learning for stroke lesion segmentation," at the BrainLes 2022 workshop, MICCAI 2022, Singapore. [Abstract | Slides]
- May 2022 Presented "Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-Activation U-Net," at *ISMRM 2022*, London, England, UK. [Abstract | Poster]
- Sep 2021 Presented "Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-activation U-Net," at the MSSEG-2 challenge day, virtual. [Poster | Slides]

Honors

- June 2019 Awarded a **Marie Curie fellowship** as a researcher in the INSPIRE-MED network, founded by the European Union's Horizon 2020 research and innovation program.
- May 2014 Ranked 51st in the Nationwide M.Sc. Entrance Exam in Electrical Engineering of Iranian Universities (among over 20000 participants).
- May 2003 Accepted in the NODET (National Organization for the Development of Exceptional Talents) high school entrance examination.

Machine Learning Competition Experience

- Summer 2022 **ISLES 2022 MICCAI Challenge**, Ranked 3rd in the final leaderboard in the task of ischemic stroke lesion segmentation
- Summer 2021 **MSSEG-2 2021 MICCAI Challenge**, Ranked 1st in two leaderboards in the task of new multiple sclerosis lesion detection and segmentation
- Summer 2020 **BraTS 2020 MICCAI Challenge**, Task 1: Brain Tumor Segmentation in MRI scans
- Nov 2017 **Kaggle Competition**, Ranked top 18% in the Porto Seguro's Safe Driver Prediction
- Jul 2017 **Kaggle Competition**, Ranked top 40% in the Mercedes-Benz Greener Manufacturing
- Feb 2016 **Kaggle Competition**, Silver medal, Ranked top 4% in the Winton Stock Market Challenge

Teaching Experience

- Spring 2017 **Machine Learning Workshop**, Invited to give lectures on "Machine Learning and its Applications to Biomedical Engineering and Bioinformatics"
Department of Electrical Engineering, University of Zanjan, Zanjan, Iran.
- Fall 2012 **Teaching Assistant**, Course: *Signals and Systems*
Department of Electrical Engineering, University of Zanjan, Zanjan, Iran.
- Spring 2011 **Teaching Assistant**, Course: *Linear Algebra*
Department of Electrical Engineering, University of Zanjan, Zanjan, Iran.

Technical skills

- Programming PYTHON, R, C++, RCPP
- Deep Learning PYTORCH, JAX
- Image Analysis MONAI, SCIKIT-IMAGE, SIMPLEITK
- Data Analysis PANDAS, SCIKIT-LEARN, TIDYVERSE (DPLYR, GGLOT2, ...), SQL
- Other Tools DOCKER, GIT, L^AT_EX, TIKZ

Languages

English: Advanced, **Turkish & Azeri:** Advanced, **Persian:** Native

List of publications

Journal Papers

1. P. Ashtari, D. M. Sima, L. De Lathauwer, D. Sappey-Marinier, F. Maes, and S. Van Huffel, “Factorizer: A Scalable Interpretable Approach to Context Modeling for Medical Image Segmentation,” *Medical Image Analysis*, vol. 84, p. 102706, Feb. 2023, doi: 10.1016/j.media.2022.102706.
2. P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marinier, “New Multiple Sclerosis Lesion Segmentation and Detection Using Pre-Activation U-Net,” *Frontiers in Neuroscience*, vol. 16, Oct. 2022, doi: 10.3389/fnins.2022.975862.
3. B. Barile, P. Ashtari, C. Stamile, A. Marzullo, F. Maes, F. Durand-Dubief, S. Van Huffel, and D. Sappey-Marinier, “Classification of Multiple Sclerosis Clinical Profiles Using Machine Learning and Grey Matter Connectome,” *Frontiers in Robotics and AI*, p. 250, 2022. doi: 10.3389/frobt.2022.926255.

Conference Proceedings: Papers

1. B. Barile, P. Ashtari, F. Durand-Dubief, F. Maes, D. Sappey-Marinier, and S. Van Huffel, “A kernel based multilinear SVD approach for Multiple Sclerosis Profiles Classification,” *ESANN 2022 proceedings*, 2022.
2. P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marinier, “Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-activation U-Net,” in *MSSEG-2 challenge proceedings: Multiple sclerosis new lesions segmentation challenge using a data management and processing infrastructure*, Sep. 2021, pp. 45–51. [Online]

3. P. Ashtari, F. Maes, and S. Van Huffel, "Low-Rank Convolutional Networks for Brain Tumor Segmentation," in *International MICCAI Brainlesion Workshop: BrainLes 2020. Lecture Notes in Computer Science*, 2021, pp. 470–480. doi: doi.org/10.1007/978-3-030-72084-1_42.

Conference Proceedings: Abstracts

1. P. Ashtari, B. Barile, S. Van Huffel, and D. Sappey-Marinier, "Longitudinal Multiple Sclerosis Lesion Segmentation Using Pre-Activation U-Net," in *ISMRM 2022*, abstract 1967. Presented in ISMRM 2022, London, England, UK, 07-12 May 2022.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
STADIUS CENTER FOR DYNAMICAL SYSTEMS, SIGNAL PROCESSING, AND DATA ANALYTICS
Kasteelpark Arenberg 10, box 2446
B-3001 Leuven
pooya.ashtari@esat.kuleuven.be
<https://www.esat.kuleuven.be/stadius/>

