Emulation and prediction of cosmic web simulations through deep learning

Marion Ullmo

université PARIS-SACLAY

Emulation and prediction of cosmic web simulations through deep learning

Émulation et prédiction de simulations de la toile cosmique par apprentissage profond

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 127, Astronomie et Astrophysique d'Île-de-France (AAIF)
Spécialité de doctorat: Astronomie et Astrophysique
Graduate School: Physique. Référent: Faculté des Sciences d'Orsay

Thèse préparée dans l'unité de recherche **Institut d'Astrophysique Spatiale** (Université Paris-Saclay, CNRS), sous la direction de **Nabila AGHANIM**, Directrice de recherche, et le co-encadrement d'**Aurélien DECELLE**, Maître de conférences

Thèse soutenue à Paris-Saclay, le 1er Février 2022, par

Marion ULLMO

Composition du jury:

Laurent VERSTRAETE Professeur, IAS, Université Paris-Saclay	Président
Miguel ARAGON-CALVO Professeur, Astronomy Institute, Ensenada, Universidad Nacional Autonoma de Mexico	Rapporteur & Examinateur
Dominique AUBERT Professeur, Observatoire astronomique de Strasbourg, Université de Strasbourg	Rapporteur & Examinateur
Alexandre BEELEN Astronome adjoint, Institut Pythéas, Aix Marseille Université	Examinateur
Nabila AGHANIM Directrice de recherche, IAS, Université Paris-Saclay	Directrice de thèse

Thèse de doctorat

NNT: 2022UPASP012

université PARIS-SACLAY

ÉCOLE DOCTORALE
Astronomie
et Astrophysique
d'Île-de-France (AAIF)

Titre: Emulation et Prédiction de Simulations de la toile cosmique par apprentissage profond

Mots clés: Cosmologie, Simulations, Apprentissage Automatique, Méthodes Statistiques, GANs, Autoencodeurs

Résumé: Le modèle cosmologique standard fournit une description de l'Univers dans son ensemble : son contenu, son évolution et sa dynamique. Une façon classique de déterminer l'évolution de la matière dans l'univers repose sur l'utilisation de simulations numériques qui sont très coûteuses en termes de temps d'exécution, de stockage et de puissance de calcul. Nous explorons l'utilisation de réseaux de neurones profonds (DNN) comme une alternative à ces simulations coûteuses. Dans une première partie, nous avons construit et entraîné un réseau antagoniste génératif (GAN) pour extraire la distribution sous-jacente d'un ensemble de données construit à partir d'un champ de densité de matière noire simulé, et ainsi générer rapidement de nouvelles données de type simulation avec des statistiques identiques. Nous avons déterminé, en détail, les forces et les limites de l'utilisation des GAN à cette fin, et constaté que le GAN génère avec succès de nouvelles images et de nouveaux cubes de données qui sont statistiquement cohérents avec les données sur lesquelles il a été entraîné. Dans une deuxième partie, nous avons montré comment utiliser le GAN entraîné pour construire un autoencodeur (AE) réplicatif simple qui peut conserver les propriétés statistiques des données, et nous avons développé un AE prédictif pour inférer des données à $z = 0$ (présent) à partir d'époques précédentes ($z = 1, 2, 3$). Nous avons constaté que l'AE réplicatif peut extraire efficacement des informations des données de simulation pour les encoder dans un nombre réduit de paramètres. Par ce biais, l'AE peut récupérer les images et les cubes de manière satisfaisante, en conservant notamment leurs propriétés statistiques en termes de distribution de densité, de spectre de puissance et de nombre de pics. Enfin, nous montrons que l'AE prédictif, bien que montrant une faible capacité prédictive dans sa forme la plus simple, réussit très bien à inférer l'évolution des données dès lors que nous lui fournissons suffisamment d'informations en entrée, notamment en utilisant le champ de vitesse associé. Avec ces preuves de concept, nous concluons que les DNNs sont des outils prometteurs pour générer rapidement de grands ensembles de données réalistes. De plus, lorsqu'ils sont entraînés et qu'ils reçoivent les bonnes informations (par exemple, la dynamique du champ de densité), les DNN contiennent les informations nécessaires pour décrire l'évolution de la structure.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Astronomie
et Astrophysique
d'Île-de-France (AAIF)

Title: Emulation and Prediction of Cosmic Web Simulations through Deep Learning

Keywords: Cosmology, Simulations, Deep Learning, Statistical Methods, GANs, Autoencoders

Abstract: The standard cosmological model provides a description of the Universe as a whole: its content, its evolution and its dynamics. A standard way of determining the evolution of matter in the Universe rests on the use of numerical simulations that are very expensive in terms of running time, storage and computing power. We explore the use of deep neural networks (DNN) as an alternative to these costly simulations. In a first part, we built and trained a Generative Adversarial Network (GAN) to extract the underlying distribution of a dataset, built from a simulated dark matter density field, and to quickly generate new simulation-like data with identical statistics. We have determined, in details, the strengths and limitations of use of GANs for this purpose, and found that the GAN successfully generates new images and data cubes that are statistically consistent with the data on which it was trained. In a second part, we have shown how to make use of the trained GAN to construct a simple replicative autoencoder (AE) that can conserve the statistical properties of the data, and further developed a predictive AE to infer data at $z = 0$ (today) from earlier epochs ($z = 1, 2, 3$). We found that the replicative AE can efficiently extract information from simulation data to encode it into a reduced number of parameters. By this means, the AE can recover the images and cubes satisfactorily, notably conserving their statistical properties in terms of density distribution, power spectrum and peak counts. Finally, we show that the predictive AE, while showing poor predictive capacity in its simplest form, succeeds very well to infer data evolution once we supply it with sufficient information in input, notably when using the associated velocity field. With these proofs of concept, we conclude that DNNs are promising tools to quickly generate realistic large datasets. Moreover, when trained and supplied with the right information (e.g. dynamics of the density field) DNNs contain the necessary information to describe the structure evolution.

Synthèse en français

Le modèle cosmologique standard fournit une description de l'Univers dans son ensemble : son contenu, son évolution et sa dynamique. Une façon classique de déterminer l'évolution de la matière dans l'univers repose sur l'utilisation de simulations numériques qui sont très coûteuses en termes de temps d'exécution, de stockage et de puissance de calcul. Nous explorons l'utilisation de réseaux de neurones profonds (DNN) comme une alternative à ces simulations coûteuses.

Dans une première partie, nous développons un réseau antagoniste génératif (GAN) pour extraire la distribution sous-jacente d'un ensemble de données construit à partir d'un champ de densité de matière noire simulé, et ainsi générer rapidement de nouvelles données de type simulation avec des statistiques identiques. Nous détaillons le concept de GAN, puis expliquons la construction, à partir de simulations N-corps, des données que nous utilisons pour entraîner nos réseaux neuronaux ; trois types de données, représentant des champs de densité, sont créées: des cubes 3D et des images 2D à partir de simulations 3D, ainsi que des images 2D à partir de simulations 2D. Ensuite, nous montrons le processus par lequel nous construisons et entraînons nos GANs, détaillant l'architecture de ces derniers et les méthodes d'optimisation de l'entraînement. Puis nous présentons les estimateurs statistiques avec lesquels déterminons la qualité des données générées par nos réseaux. Enfin, nous présentons les résultats des GAN pour les trois types de données susmentionnées. Nous avons déterminé, en détail, les forces et les limites de l'utilisation des GAN, et constaté que le GAN génère avec succès de nouvelles données qui sont statistiquement cohérents avec les données sur lesquelles il a été entraîné.

Dans une seconde partie, nous montrons comment utiliser le GAN entraîné pour construire un autoencodeur (AE) réplicatif simple qui peut conserver les propriétés statistiques des données, et nous avons développons un AE prédictif pour inférer des données à $z = 0$ (présent) à partir d'époques précédentes ($z = 1, 2, 3$). Une fois de plus nous détaillons le concept d'AE, présentons l'architecture et l'entraînement de nos AE, ainsi qu'un nouvel estimateur statistique, le coefficient de Dice, quantifiant la précision avec laquelle l'AE inferre une image individuelle. Nous développons plusieurs types d'AE reproductifs et prédictifs dans un souci de

tester différentes approches pour optimiser nos résultats. Pour l'AE réplicatif, nous partons d'un modèle de base (baseline) composé d'un encodeur traditionnel et d'un décodeur construit à partir d'un générateur de GAN, et entraîné avec une fonction de perte utilisant le discriminateur d'un GAN. Nous testons également un AE traditionnel (traditional), ainsi que deux variations sur l'AE de base, l'un (ℓ_2-loss) avec une fonction de perte ℓ_2 entre donnée inférée et donnée réelle, et l'autre (latent layer) avec une fonction de perte ℓ_2 entre code latent inféré et code latent réel d'une donnée. Nous avons constaté que l'AE réplicatif peut extraire efficacement des informations des données de simulation pour les encoder dans un nombre réduit de paramètres. Par ce biais, l'AE peut récupérer les images et les cubes de manière satisfaisante, en conservant notamment leurs propriétés statistiques en termes de distribution de densité, de spectre de puissance et de nombre de pics. Pour l'AE prédictif (timewarper), nous testons à nouveau différents types d'AE, partant d'un modèle de base (baseline) que nous entraînons en ne lui fournissant qu'une donnée à une époque $z = 1, 2$ ou 3 et comparant sa prédiction à la même donnée à époque $z = 0$. Ensuite nous testons un AE (curriculum learning) entraîné progressivement à prédire une donnée à $z = 0$ depuis $z = 0$ pendant cinquante époques en incrémentant le redshift d'entrée de 1 toutes les 50 époques. Nous testons aussi un AE (multiple redshift input) entraîné à prédire une donnée à $z = 0$ en lui fournissant la même donnée à deux redshifts différents (ex: 1 et 2). Enfin, nous testons un AE (velocities) entraîné à prédire une donnée (champ de densité) à $z = 0$ en lui fournissant la donnée, mais aussi le champ de vitesse associé, à $z = 1, 2$ ou 3. Nous montrons que l'AE prédictif, bien que montrant une faible capacité prédictive dans sa forme de base, réussit très bien à inférer l'évolution des données dès lors que nous lui fournissons suffisamment d'informations en entrée, notamment en utilisant le champ de vitesse associé. Avec ces preuves de concept, nous concluons que les DNNs sont des outils prometteurs pour générer rapidement de grands ensembles de données réalistes. De plus, lorsqu'ils sont entraînés et qu'ils reçoivent les bonnes informations (par exemple, la dynamique du champ de densité), les DNN contiennent les informations nécessaires pour décrire l'évolution de la structure.

Remerciements

C'est emplie d'une gratitude sans borne et d'une certaine dose de mauvaise conscience pour ce que j'ai fait subir à tous ceux que je m'apprête à remercier, que je repense à ces trois (et demie..?) dernières années. Ces dernières ont été tellement denses et riches en rencontres, en expériences, et en émotion qu'il est difficile de pleinement leur faire justice dans les lignes qui vont suivre.

Je tiens tout d'abord à remercier mes merveilleux maîtres de thèse, Nabila et Aurélien, victimes principales de mon stress et de ma mauvaise organisation, mais qui ont su malgré tout et avec beaucoup de patience me ~~fouetter~~ pousser jusqu'à la finish line. Un parfait tandem, avec Aurélien en force calme avec lequel développer des nouvelles voies d'exploration, et Nabila en moteur mille watt, fournissant force motivation et recadrage pour les tâches auxquelles je rechignais (rédaction, présentations... socialisation...). Je vous en ai fait voir de toutes les couleurs, mais j'ai une reconnaissance sans limite pour le soutien bienveillant que vous avez su fournir tout au long de ma thèse, et j'espère pouvoir transmettre un jour l'impact positif que vous avez eu sur moi.

Ensuite mes remerciements vont, concentriquement, aux personnes qui constituent et ont constitué l'IAS, laboratoire chaleureux et composé semble-t-il uniquement de gens aimables.

Un merci tout d'abord à la direction ainsi qu'aux équipes administrative et informatique, et tout particulièrement à Stéphane et Clément du service info qui ont passé un temps non-négligeable à gérer mes problèmes récurrents d'ordinateur, et Patricia qui était là dès mon inscription retardataire en stage et toujours dispo pour aider avec n'importe quel problème administratif.

Merci à toute l'équipe de Cosmo, constituée de gens à la fois brillants, humbles et généreux, véritable deuxième famille pendant ces dernières années, et qui a forgé la personne que je suis aujourd'hui. C'est forte de leur conseils, de leur motivation et de leur bienveillance que j'ai pu présenter avec fierté les résultats de mon travail lors de ma soutenance.

A commencer par les permanents, Marian, fournisseur principal de listes de lecture, de gadgets informatiques et de sucreries, Mathieu, prof de Swedish Fit à ses heures perdues et mine d'or de connaissances et d'avis francs, Julien, grand

sorcier des équations à la moustache parfaitement entretenue, et plus fraîchement adoubée, Laura, reine du style et du café. Merci énormément à Laurent et Emilie pour leur écoute et leur soutien dans ma phase la plus misérable de la rédaction.

Merci aux post-docs, frères et soeurs ainés de coeur, sources primaires de savoirs et de conseils avisés lors des réunions et des repas (pré-covid...snif). Nicola, compagnon de boxe et conversationniste hors pair, Hideki, qui a toujours le mot pour rire ou conforter, Joseph qui, malgré sa fâcheuse tendance à poser des questions pertinentes quand on cherche à finir discrètement sa présentation, arrive toujours à se faire pardonner grâce à sa jovialité contagieuse, Alex qui m'a intimé de croire en ma recherche indépendemment des coups à l'égo, Fabien parti trop tôt et Giulio et Jenny qui ont su me donner des supers conseils dans le court laps de temps où je les ai vus.

Merci aux thésards, compagnons d'infortune et de rigolade dans cette grande aventure qu'est la thèse, pour les innombrables souvenirs et délires partagés pendant les ByoPiC weeks, les sorties, et les pauses midi/café. Merci Louis dit "le chaud", Edouard le bon gars, Victor d'Hossegor, et aussi Nadège, Adélie, Danilo, Thomas, Hubert, et Valentin. Merci tout particulièrement à la cuvée 2019 qui a évolué à mes côtés lors des trois années, Dany, son énergie débordante et son sourire de supernova, toujours à motiver les troupes pour sortir s'amuser après le travail, Thibaut partenaire n°2 de boxe et camarade de discussion privilégié pour mes obsessions weebesques, mais aussi pour parler de tout et n'importe quoi, mais toujours intensément.

Enfin il faut mentionner la cour des miracles qu'est le bureau 216 dit bureau des princesses, PMU local et lieu privilégié d'échange de potin, de commisération, de trafic de tisanes et de plantes grasses (RIP Augustine I et II), de plaintes chantées sur des airs de Claude François, et éventuellement (surtout!) de travail. Merci à ses principaux habitants, plus récemment Stefano à la voix douce et Raphaël "the broken", qui semble marcher au karma inversé tant ses multiples actes généreux dont organisation de super vacances et pots/cadeaux de thèse se sont suivis d'autant de blessures rugbystiques. Vous avez adouci ces derniers mois difficiles et je regrette seulement que notre temps ensemble ait été si court. Merci enfin à mes deux fabuleux co-bureau du premier jour, Tony et Céline, pour avoir su créer le meilleur environnement de travail possible, avec votre humour déjanté, votre gentillesse à tout rompre et cette once d'asocialité qui m'a tout de suite mise à l'aise parmis vous. Merci Tony d'avoir été mon canard de débuggage et pour ton beatboxing de qualité. Merci Céline d'avoir été ma senpai, m'apprenant les arcanes de la recherche, de la "relaxation" et de la cassosserie. Il y a mille autres trucs que je pourrais citer, mais ce serait plus long que la thèse qui suit...

Nabila, encore merci d'avoir constitué une équipe aussi chouette et pour l'effort que tu as mis et mets encore à créer du lien, et sortir les introvertis les plus

récalcitrants (je ne vois pas de qui vous parlez) de leur coquille.

Merci aux amis de l'école d'été Euclide, particulièrement le groupe des Euclidiens Heureux pour des chouettes moments de balades, cartes et pétanque à Banyuls et Hyères.

Merci à ma famille et mes amis pour votre soutien pendant cette thèse. Merci Carole de m'avoir sortie pour les balades, la piscine et les matchs de tennis, m'évitant ainsi de fusionner avec mon canapé et me permettant de sortir toute ma frustration de façon saine. Merci Flore, Stéphane, Simon, Olivia et Sylvain pour les soirées et vacances reposantes, et pour votre amitié par-dessus tout. Merci Maman de m'avoir ramassée à la petite cuillère et de t'être occupée de moi pendant les périodes les plus sombres. Merci Pops d'avoir aidé pendant mes pics de stress sans me transmettre le tiens. Merci Maud et Nathou de votre patience et de vos paroles d'affirmation pendant mes moments les plus vulnérables. Merci à mes grands-parents, oncles, tantes, cousins et cousines pour leur soutien confiant.

Merci Flavien de m'avoir accompagnée et soutenue pendant cette aventure, et de nous avoir trouvé un nid rien qu'à nous. Je t'aime.

Enfin, merci à mon jury de thèse pour l'intérêt qu'ils ont porté à mon travail et les précieux retours et conseils qu'ils m'ont apportés lors de la lecture et de la soutenance.

Contents

Chapter 1

Introduction

1.1 Context

The standard model of cosmology, also referred to as the ΛCDM or concordance model, provides a description of the Universe as a whole: its content, its evolution and its dynamics.

Combining information gathered through the observation of multiple sources, such as the large-scale structure in the distribution of galaxies, the abundance of various gases in the Universe, and the temperature map of the cosmic microwave background, it proposes a parametrization that comprehensively accounts for each of these observations.

This model is based on several assumptions; namely it posits the validity of Einstein's theory of General Relativity, the fact that the universe is isotropic and homogeneous, and the existence of dark energy (associated with the cosmological constant Λ) responsible for the accelerating expansion of the universe, and dark matter, whose only interaction with any type of matter including itself is gravitational. As their "dark" names suggest, they cannot be directly observed but rather deducted from their effect on observations.

In this model the early universe took form starting from an originating event, the Big Bang, following which the universe went through a first exponential expansion from an initial high-density and high-temperature state, increasing in size by a factor 10^{30} in the span of 10^{-32} seconds. During this rapid inflation quantum fluctuations were stretched to macroscopic proportions and thus etched into the ensuing matter density field in the form of gaussian-distributed density fluctuations (ie primordial fluctuations) within an otherwise homogeneous field, that can be observed today via the CMB (Fixsen et al., 1996; Aghanim et al., 2018; Tegmark et al., 2003; Bond et al., 1996; Coles and Chiang, 2000; Forero-Romero et al., 2009).

Furthermore according to this model, the structures observed today (galaxies and clusters of galaxies) have evolved from these small fluctuations through the progressive gravitational collapse of matter towards overdense regions to form what is today a complex network of structures known as the cosmic web (Bond et al., 1996).

The hierarchical assembly of matter is governed by the combined effects of gravity and expansion, which lead to highly nonlinear equations of motion when studying the evolution of the matter density field in the universe; thus when applying our model to predict structure evolution from an initial density field (that is set so as to be coherent with the CMB), we cannot provide an analytical equation that comprehensively describes an evolution of the field according to time.

Consequently we rely on numerical simulations that can compute the evolution of a density field by iteratively computing the equations of motion over time lapses short enough that a linear equation of trajectory sufficiently approximates the true trajectory of massive matter particles.

Such simulations allow us to confront the model to observations, with increasingly large numerical simulations such as Millennium (Springel et al., 2005) and Illustris (Vogelsberger et al., 2014) providing a finer prediction of cosmological structure formation which have subsequently been confirmed with actual observations of large-scale matter distribution in galaxy surveys such as the Sloan Digital Sky Survey (SDSS) (York et al., 2000).

However, the large and detailed simulations that included detailed baryonic physics, such as Horizon-AGN(Dubois et al., 2016), BAHAMAS (McCarthy et al., 2016), or IllustrisTNG (Pillepich et al., 2018), which are needed to compare theory with observations, are computationally expensive. Faster fully analytical approaches (Shandarin and Zeldovich, 1989; Kitaura and Heß, 2013) and semianalytical simulations that combine traditional simulation methods and analytical approximations (Monaco et al., 2002; Tassev et al., 2013) both relying on first- or second-order perturbation theory, exist, but they cannot address the highly nonlinear stages of the structure formation.

The recent advances in computer technology and in machine learning (ML) (LeCun et al., 2015; Goodfellow et al., 2016) have led to an unprecedented boom in the development and use of ML methods, notably in the form of neural networks, used for supervised and unsupervised learning. This has prompted an increasing interest from the astronomical community in proposing ML as an interesting alternative for the fast generation of mock simulations and mock data. The ever larger quantities and quality of astronomical data call for systematic approaches to properly interpret and extract the information that can be based on machine-learning techniques such as in Villaescusa-Navarro et al. (2020), Schawinski et al. (2018), or Bonjean (2020). Machine learning can also be used to produce density

maps from large N-body simulations of dark matter (DM) (Rodríguez et al., 2018; Feder et al., 2020) in a computationally cheaper manner, to predict the effects of DM annihilation feedback on gas densities (List et al., 2019), or to infer a mapping between the N-body and the hydrodynamical simulations without resorting to full simulations (Tröster et al., 2019; Zamudio-Fernandez et al., 2019).

In this context, certain types of neural networks, called convolutional neural networks (CNN) (LeCun et al., 1990), excel in the general field of image processing through their automatic pattern detection property (for a comprehensive review, see Ntampaka et al. (2019)). Generative models among CNN, such as generative adversarial networks (or GANs) (Goodfellow et al., 2014), have shown promising results in computer science and physics (Casert et al., 2020; de Oliveira et al., 2017; Ahdida et al., 2019). These networks aim to learn a probability distribution as close as possible to that of a considered dataset in order to later generate new instances that follow the same statistics. GANs have proven to be very promising tools in terms of media generation (Donahue et al., 2018; Clark et al., 2019), and notably in the generation of high-resolution images (Karras et al., 2020). In astronomy, they have recently been used in several cases, and more specifically, by Rodríguez et al. (2018) and Feder et al. (2020), to provide a fast and easy alternative to simulations and images. Wasserstein GANs (or WGANs) have also been used to detect anomalies in astronomical images (Margalef-Bentabol et al., 2020; Storey-Fisher et al., 2020).

In chapter 2 of this thesis, we first explore the use of GANs in generating simulation-like data in an attempt to test the limits of the model and see how well it can extract and replicate statistical properties from the data and generalize from these to generate new data. We will first present the general concept of the GAN, and then we will describe how from input simulations we build the data on which we train the GAN. Thirdly, we will describe the process by which we build our GAN, and fourthly we present the statistical estimators we choose to test the quality of our generated images. Finally, we show our trained GAN's results and conclude.

Additionally to GANs, a certain type of neural network known as Autoencoder (AE) (Hinton and Salakhutdinov, 2006) has garnered our interest, given its versatility. This network, made up of an encoder and a decoder, takes in data of a certain dimension (typically an image), learns a representation of reduced size of the data (typically a vector) in a latent space, and provides the means, through the encoder and decoder, to translate data from one space into the other. Developing a meaningful latent encoding space for data can have several applications, such as semisupervised classification, disentangling style and content of images, unsupervised clustering, and dimensionality reduction (Hinton and Salakhutdinov, 2006), as can be seen, for example, in the case of variational autoencoders (Makhzani

et al., 2015) and adversarial autoencoders (Kingma and Welling, 2013). In the case of astrophysics or cosmology, AEs could be used to help remove instrumental or astrophysical signal contamination (e.g., point sources, beam, and instrumental noise) (Vojtekova et al., 2021) or for inpainting masked areas while preserving the statistics of the data (Sadr and Farsian, 2020; Puglisi and Bai, 2020).

In chapter 3, we present how we can build on the trained GAN constructed in chapter 2 to construct an AE that is able to preserve the statistics of a dataset. In a first step, we present the general concept of the AE, our own process to build and train an AE, and results for a simple *replicative AE*, that is simply tasked with recovering its input data after having encoded in a vector of small dimension. In a second step, we experiment to see if we can make use of this network structure to predict structure evolution in time within simulations with an aptly-named *predictive AE*.

Finally, we will conclude and present some perspectives in chapter 4.

But first, let us start by an overview of Deep Learning in the following section.

1.2 Deep Neural Networks: an overview

1.2.1 DNN Context

In the past years the field of machine learning (ML) has made increasingly fast progress thanks to the rapidly developing technology to process large amounts of information. Such advancements, made widely accessible due to their now relatively low price in terms both of energy and components, have made possible the widespread emergence of deep learning (DL), which rests on the training of neural networks with the help of large (>10GB) datasets.

Thus the uses and experiments based on neural networks have exploded in the past few years, with a wide array of applications anywhere from self-driving cars (Ettinger et al., 2021; Zhao et al., 2020; Rhinehart et al., 2019) to singing portraits (Zakharov et al., 2019; Vougioukas et al., 2019), in fields as various as art (Yalçın et al., 2020; Mordvintsev et al., 2015; Bethge et al., 2016; Foster, 2019), media and entertainment (Skinner and Walmsley, 2019; Covington et al., 2016; Amato et al., 2019), and of course science at large (Baldi et al., 2015; Wang et al., 2019; Carleo et al., 2019; Salman et al., 2015).Neural networks can be applied to a large variety of data types, most popularly to still images, but also to sounds (Deng et al., 2013; Li et al., 2017), videos (Zhang et al., 2016; Lotter et al., 2016; Mathieu et al., 2015), text (Iqbal and Qureshi, 2020; Yousefi-Azar and Hamey, 2017; Kowsari et al., 2017), and even symbolic equations (Cranmer et al., 2020; Lample and Charton, 2019)

They excel in the fast performance of automated tasks, which can take many

forms, such as prediction (Lv et al., 2014; Poplin et al., 2018; Chong et al., 2017; Qiu et al., 2018), detection (Zhao et al., 2019; Chalapathy and Chawla, 2019; Liu et al., 2020; Badjatiya et al., 2017), clustering and data visualisation (Aljalbout et al., 2018; Min et al., 2018; Tian et al., 2014), reconstruction (Rivenson et al., 2018; Hyun et al., 2018), generation (Goodfellow et al., 2014; Briot and Pachet, 2017; He and Deng, 2017), translation (Singh et al., 2017; Popel et al., 2020; Varela-Salinas et al., 2018) and much more.

The field of astronomy is a good candidate for the application of deep learning. For one, much of current research in this field relies heavily on observation that is conducted with the help of telescopes generating very large amounts of data through surveys that map the sky in increasingly finer detail both in terms of angular and spectral resolution. Examples include the Sloan Digital Sky Survey (SDSS)(York et al. (2000), $>$ 100 TB available), the James Webb Space Telescope(JWST) (Beichman et al. (2014), $>$50 GB per day), the Vera Rubin Observatory Legacy Survey of Space and Time (LSST)(Ivezić et al. (2019), 15 TB per night), Euclid (Laureijs et al. (2010), $>$ 1 PB a year, and the Square Kilometer Array (SKA)(Dewdney et al. (2009), 600 PB a year).

This rich supply of available data is suited to the training of deep neural networks, as they necessitate large datasets for optimal training. Moreover, when trying to make sense of these observations, many challenges, that we will detail below, appear. While there exist standard analytical methods to counter them, they can often only be approximate, or slow to develop and apply. Deep learning can be considered as a valuable alternative to quickly obtain similar or even better results through the use of more unorthodox means, occasionally bringing to light the unsuspected relevance of certain parameters in a datum to obtain some given information.

Here we will list a few of common challenges faced in observations and cite examples of uses of ML/DL to tackle them. For example, when studying the data recovered by a given telescope or detector, one has to account for instrumental limitations (noise, point spread function, limited amount of detectors, etc), or signal contamination caused by the local environment such as atmospheric effects, bending, absorbing or blurring light signal). While several analytical approaches, usually centered on deconvolution, have been used to tackle these issues (e.g. Starck et al., 2002), ML has provided fast alternatives to efficiently reduce these various unwanted effects (Paschalis et al., 2013; Baso et al., 2019; Long et al., 2021; Jia et al., 2020b).

Even after correction for all of these issues, many more arise from physical limitations, first and foremost the fact that observations are a 2D projection of a 4D (3D space + time) universe. Problems include contamination of the line of sight when trying to observe distant objects, such as the superposition of emission of

foreground and background objects (ie milky way for CMB, etc), and light contamination by bright sources; tackling these issues entails reconstructing incomplete observations (Flamary, 2017) and disentangling different light sources (also called component separation) (Nuzillard and Bijaoui, 2000; Picquenot et al., 2019; Vos et al., 2019; Bonjean, 2020; Tanimura et al., 2021).

Moreover, some components such as dark matter cannot be directly observed as it does not emit light, but rather modifies background emissions in such a way that they can be detected, with strong and weak lensing for example (Davies et al., 2019).

Finally, another challenge when dealing with large scale observations comes with selecting objects of interest within sizeable datasets. Once more neural networks prove tailor-made for such tasks of detection and selection, surpassing analytical methods that often rely on too few parameters to determine the nature of an image; they prove efficient both in finding known types of sources (Lukic et al., 2020; Hassan et al., 2019), and novel types that have not been studied before (Shamir and Wallin, 2014; Margapuri et al., 2021). Within a selection of sources of interest (galaxies, clusters, supernovae, etc), there is often a need for classification, either according to predefined categories (such as elliptical or spherical galaxies for example), or conversely as a way to construct new categories (Ball et al., 2006; Jia et al., 2020a).

1.2.2 From Machine Learning to Deep Neural Networks

To better understand the CNNs which we use in our work, it is worth presenting them in their global context. Indeed, CNNs are a type of application of *deep learning*, itself a sub-category of *machine learning* which in turn falls into the broader category of *artificial intelligence*. We will progressively define these categories while specifying the characteristics of each subcategory, all the while providing examples both inside and outside of the astrophysics domain.

While several examples within the categories that we will cover may contain purely mechanical aspects, such as the interaction of a machine with physical objects or a physical response of the machine, we will focus on the strictly algorithmic aspect of each example, which we will refer to as *model*.

In the same vein, we clearly define the terms of input and output to avoid any confusion at their mention. Any input, be it sound, video, coins, etc, can be represented as an array of values that the model will interpret, process and respond to with an output, which itself will be produced as an array that can then be transformed into a list of tasks to be executed (e.g. "stop", "return x amount of cash", "turn left", etc) or a datum that can be easily interpreted by a human (e.g. sound, image, label(s), etc).

Artificial intelligence (AI) combines all forms of models built to use a "human-like" reasoning method to perform complex tasks (e.g. chess, automatic vacuum cleaning, etc), and specifically to make decisions when responding to novel, previously unencountered problems, usually within a defined frame. This can be through explicitly defined "if/then" responses for a set of possible scenarios; alternatively, the model can be trained to learn the best response on its own.

This is the case for the subcategory of AI referred to as Machine Learning (ML) which focuses on making systems automatically learn and improve from *experience* without being *explicitly* programmed. This *experience* comes in the form of large quantities of data that the system is made to interact with and learn from (i.e. "train on"), usually referred to as a training set. Rather than an *explicit* chain of commands to complete a specific task, the system is given a relatively simple objective, usually in the form of reducing a loss function, that is contingent on the task in question being optimally performed.

Loss Functions A loss function will map a model's output and target onto a real number representing some "cost" associated with the output/target discrepancy. The loss ideally has to be a continuous and continuously derivable function of the model's parameters, as the model's parameters will be gradually modified according to the loss's gradient over the parameters. The choice of loss will depend on the task at hand, with the loss falling into two main categories: regression losses and classification losses. Regression losses concern cases where the target output contains values that are continuous (e.g. trying to generate a specific image or determine the ellipticity of a galaxy) whereas classification losses are for cases where the output is one of a set of finite categorical values representing labels (e.g. determining if an image is of a cat or dog, or categorizing a written number into one of 0-9 digits). Below we list a few examples of commonly used losses.

- ℓ_2 **loss:** also known as mean square error or quadratic loss, it is simply defined as $L = \|o - t\|^2$, where o is the output and t is the target; this is typically used for regression.
- ℓ_1 **loss:** or mean absolute error, this regression loss is defined as $L = |o - t|$; also used for regression. Although it is not continuously derivable, it is more robust to outliers.
- **Cross-entropy loss:** typically used for classification, this loss is defined as $L = \sum_i o_i \log(t_i) + (1 - o_i) \log(1 - t_i)$ where the i represent the different labels a datum can have, t_i is either 0 or 1 depending on whether it has the given label or not, and o_i is a float between 0 and 1.

Examples of ML include support vector machines, random forests, and most notably artificial neural networks.

A large branch of ML is dedicated to artificial neural networks (ANNs); originally made following a schematic model attempting to mimic biological neurons (see fig.1.1), this type of algorithm quickly proved effective in many domains.

The Neuron In the biological context, a neuron is part of a larger network and serves as a type of messenger; receiving a set of "messages", or impulses of various upstream neurons, it compounds the information thus received and passes it on (or "fires") provided that it is deemed significant enough by the neuron. In the case of an artificial neuron, the "messages" come in the form of numbers x_i representing the significance of each incoming information. The neuron then outputs a new message x' by attributing weights w_i to each incoming input and returning the sum with an additional bias b. The significance of the outcome is determined by a final activation f, which prevents the message from being passed on if it is below a certain threshold. We hence obtain the following equation for the output message:

$$x' = f(\sum_i w_i x_i + b) \tag{1.1}$$

y Where the x_i are yielded by upstream neurons (or input parameters), and w_i, b and f are properties specific to an individual neuron.

If the activation function is a simple step function ($f(x) = 0$ if $x < 0$ and $f(x) = 1$ if $x > 0$), we can think of a neuron as a binary classifier that divides the input parameter space in two with a linear hyperplane and assigns one of two classes (0 or 1) to every input datum depending on which side of the dividing hyperplane they fall on. This is also known as a simple *perceptron*, and the most early use of ANNs. The composition of several neurons can be regarded as finer divisions in parameter space, allowing the network to produce increasingly meaningful partition. Thus, mimicking a biological brain by connecting neurons together to form a network that spans from input to output, we can model complex classifiers or functions depending on the choice of activation functions. Typically, neurons are organized in layers, wherein each neuron receives information from neurons of the previous layer and passes on a signal to neurons of the next layer. In this case, the input is considered as the first layer and the output as the last. Layers that lie in between are referred to as *hidden layers*, as their content and effect are not immediately visible to users of the network. Multi-layered networks are referred to as Deep Neural Networks (DNN) and, given their versatility, they represent the vast majority of ANNs used today.

We are now equipped with the basic concepts of neuron and DNN; however, this can be assimilated to presenting the general concepts of "brick" and "building". How best to combine these simple building blocks to obtain a solid architecture? Which combination for what purpose? In other words, which neurons do

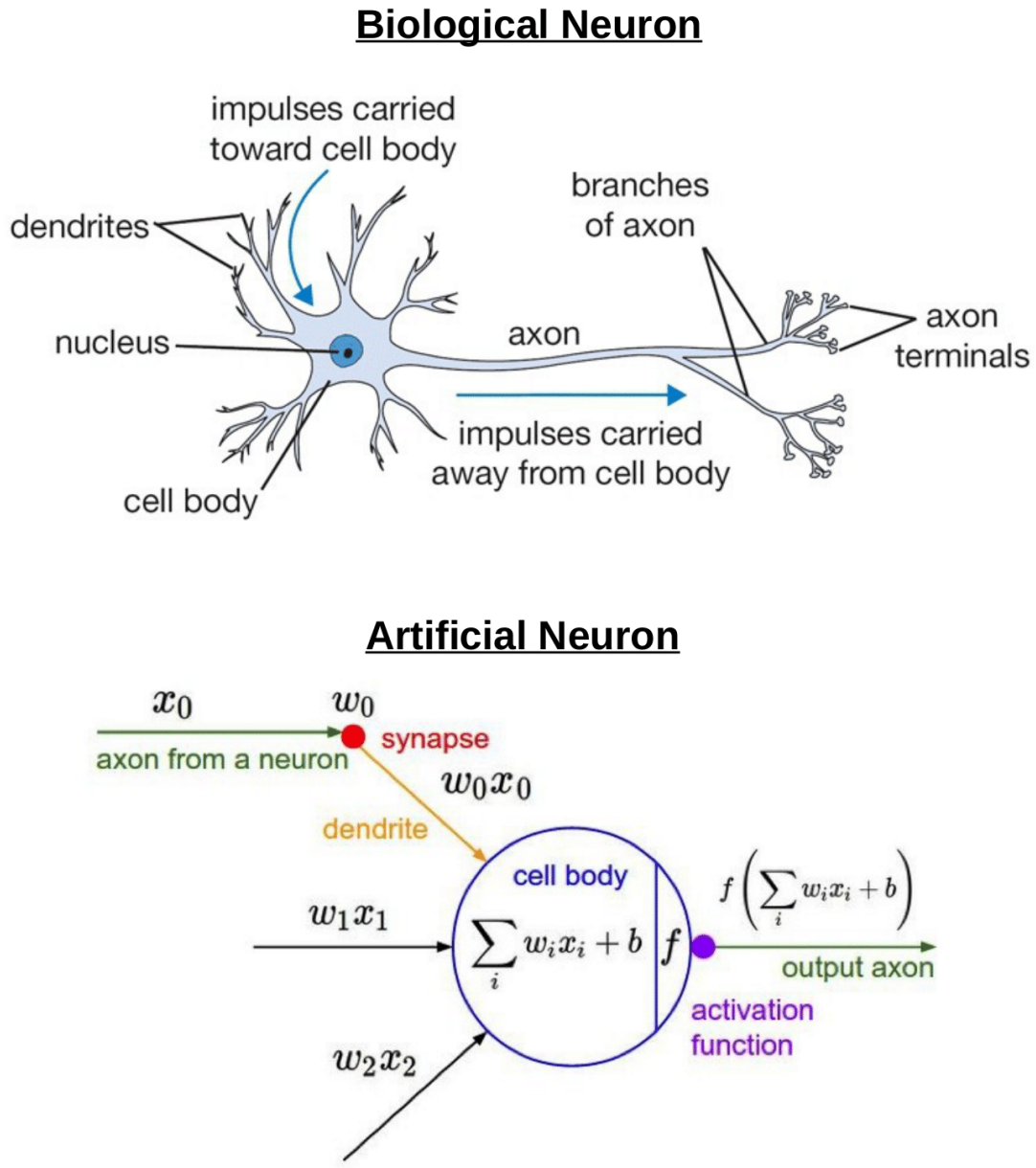

Figure 1.1: Diagram of a biological neuron (*top*) and of an artificial neuron (*bottom*).

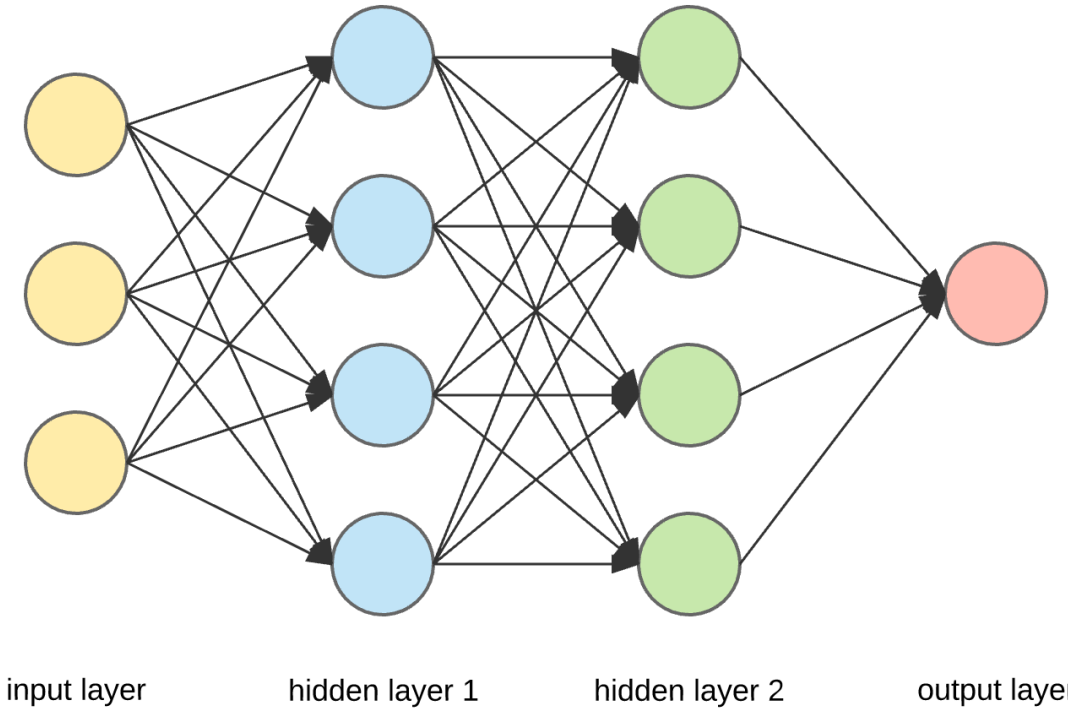

Figure 1.2: Diagram of a simple artificial neural network. Pictured in *yellow* is the input, and in *red* is the output. In *blue* and *green* we can see hidden layers. Each neuron from a layer n receives a signal from neurons of the layer $n-1$ and passes on its signal to neurons of the layer $n+1$. The network is considered deep (DNN) if there are three or more hidden layers.

credit: Gavril Ognjanovski (shorturl.at/iFIT1)

we choose to connect to one another? How many layers? How many neurons for which layer?Can we organize the neurons in such a way that information is more efficiently transferred or extracted? Which activation function(s) to use? While there is no definitive answer to these questions, some techniques propose efficient manners to connect neurons from one layer to the next for optimal transfer of information. We present some of these the following section.

1.2.3 Deep Neural Network components

Having described the basic concept of the neuron and DNNs, we will now present in more detail the different components that typically make up the architecture of a standard DNN. We will refer to these components in chapters 2 and 3.

We begin by looking into neural organization and choices of connection from one neural layer to the next.

Dense Layer The most neutral (in the sense that it does not favor any specific link from neural layer to the next), but most costly manner in which to link neurons from one layer to the next is to use a *dense layer*. Dense layers, or fully connected layers, connect all input parameters, or neurons of a previous layer, to each of the neurons in the layer, as exemplified in Fig. 1.2.

Calling x_i all the input parameters and x_j the outputs of a given layer's neurons, we have:

$$x_j = f(\sum_i w_{ij} x_i + b_j) \tag{1.2}$$

Supposing that we have n input parameters from the previous layer and m neurons in the current layer, this yields $n(w_{ij})+1(b_j)$ parameters to train for every single neuron, or $m(n+1)$ parameters for a given layer.

This manner affords plenty of options, notably for the ANN to "break" a link on its own by setting the weight of an incoming parameter to 0, but it results in more costly training than any other linking choice; hence this makes sense when dealing with a relatively small set of input or output parameters, but can quickly lead to pointlessly time-consuming training when too many parameters are involved (typically when dealing with high-dimension objects like images or videos). Here the neurons of the layer do not have any *a priori* specific organization, as they take in all input parameters indiscriminately and without any form of hierarchy. As a result, a dense layer is a type of linking which is preferable when dealing with data consisting of independent parameters (such as lists of properties, e.g. (*size, luminosity, ellipticity, etc*)).

However, the input data can often have an underlying spatial organization (such as sound, images, videos or frequency spectra), wherein each constituting

parameter is highly correlated to those in its spatial vicinity, but where combining two far-apart parameters makes little sense when looking for informative patterns. To account for this correlation, we can make localized links from one layer to the next making use of convolutions.

Convolutions When dealing with spatially organized data, whether for generating purposes or extracting information, generating or looking for patterns in smaller subspaces within a datum tends to prove quite efficient and much less costly than looking at all the parameters of the datum at once.

This can be done in an ANN with the help of convolutional layers, wherein each neuron of a layer is linked to a small area of the input data (e.g. a set of neighboring pixels in an image). The set of weights of the neuron are organized spatially to detect a given oriented pattern (e.g. vertical lines, corners, circles, etc.); this set of weights, organized in the dimension of the input data, is referred to as *kernel*. It is slid across the input data as pictured in fig 1.3, multiplying its weights with those of the area it covers, passing on a strong signal if the kernel's pattern coincides with a region's signal.

This process, as its name would imply, can be likened to a mathematical convolution, the standard operation described by the following equation (here an example for 2D):

$$(f * g)(x, y) = \iint_{-\infty}^{\infty} f(k, l)g(x - k, y - l)\, dkdl \tag{1.3}$$

With f corresponding to the input data with indices k and l, g representing the convolution kernel and $f * g$ corresponding to the resulting output data with indices x and y.

However in the case of ANNs, given that we are convolving data made up of a discrete and finite number of elements, the convolution can thus be expressed as (for a 2D example):

$$(f * g)(x, y) = \sum_{k=-\infty}^{k=\infty} \sum_{l=-\infty}^{l=\infty} f(k, l)g(x - k, y - l) \tag{1.4}$$

The resulting output is a new array of same dimension as the input but with a variable size, which is determined by *padding* and *stride* (see fig 1.3 for a graphic representation).

- *Padding* allows the kernel to fully reach the input datum's borders during convolution, such that the output is of the same shape as the input. It can be made of zeros, a reflection of the datum's border, a constant, etc.

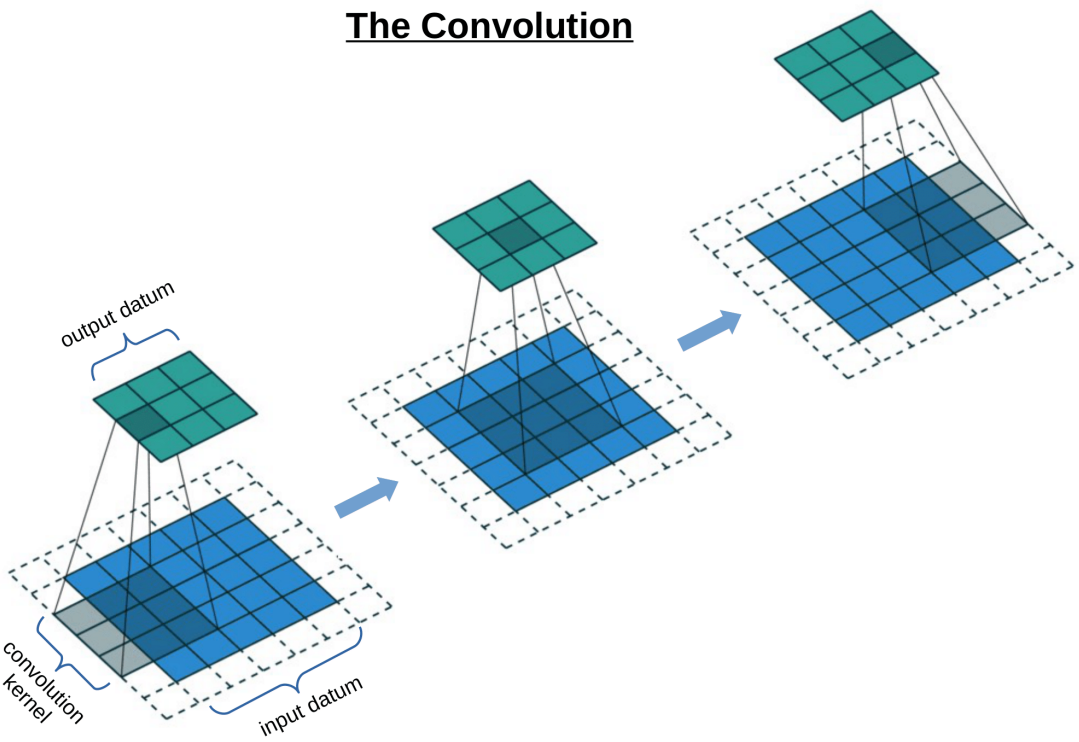

Figure 1.3: A schematic representation of a convolution. The blue array represents the input, whose parameters are multiplied with those of the gray array representing the convolution kernel, the sum of which make up the parameters of the green array, representing the output. In this case the convolution has a padding of one(represented by the white squares surrounding the input), and a stride of two (the kernel is shifted by two squares at every step).

- *Strides* are the method with which we can directly up-sample or down-sample within the convolution; an integer stride of n corresponds to sliding the kernel by n pixels at each step (or increment l and k by n in Eq.1.4). This, if used along with padding, outputs an array n times smaller than the input. Conversely, to up-sample data one can use dilation (sometimes referred to as fractional strides), wherein the input data is "dilated" by a factor n (i.e. input pixel values are placed in a larger null matrix and spaced $(n-1)$ pixels apart. A standard, stride-less convolution is then applied to the ensuing array, resulting in an array n times larger than the input.

Up-/Down-Sampling The up- and down-sampling operations are a means to respectively produce higher-dimensional data from a smaller amount of parameters or conversely extract the substantial elements of high-dimensional data, while keeping a sense of spatial coherence. The most simple examples of such operations are *up-sampling* and *pooling*. Up-sampling consists simply of turning an n^d matrix into a $(c \times n)^d$ matrix by up-sampling each of a datum's pixels into c^d pixels. Pooling consists of turning a $(c \times n)^d$ matrix into a n^d matrix by dividing the original datum into blocks of c^d pixels and returning one pixel for each block whose value can be defined in several ways, such as the average of the block's elements ($p' = \bar{p_i}$ where p' is the value of resulting pooled pixel and the p_i are the values of the pixels making up the pooled block), or the maximum of the block's elements ($p' = \max(p_i)$). All of these operations are simple, however it is generally considered more efficient to up-sample and down-sample directly during convolutions with the help of strides, which we described above.

Activation Name	**Associated equation**
Sigmoid	$f(x) = \frac{1}{1+\exp -x}$
tanh	$f(x) = \tanh x$
ReLU	$f(x) = \max(0, x)$
LeakyReLU	$f(x) = \epsilon x + (1 - \epsilon) \max(0, x)$
SoftReLU	$f(x) = log(1 + \exp x)$

Table 1.1: Examples of activation functions

Activation function As mentioned earlier, activation functions help to filter out signal that is beneath a certain threshold of relevance, and depending on the choice of function, can allow for the neuron to pass on a nonlinear transformation of the incoming parameters. Since network weights are updated through gradient descent, it is important to avoid situations where the activation functions regularly saturates, as this leads to a zero gradient and implies that weights are not updated.

Thus we tend to avoid putting sigmoid functions in intermediate layers as they tend to saturate quickly and ReLU in classification networks, as classification tends to favor the complete disappearance of irrelevant signals, pushing the network to vanishing gradient regions. Finally, final activation functions can help to map final output values to a target domain, such as pixel values within a certain range, or 0 and 1 values for classification. In table 1.1 we list a few examples of activation functions.

Chapter 2

Emulating Cosmological Simulations with GANs

2.1 Introduction

We have seen in the Introduction(1) the importance of cosmological simulations to predict the non-linear formation of large scale structures of matter in our universe according to various cosmological parameters. Given the high cost of simulations in terms of time, computing power and storage, and given the ability of DNNs to extract properties from datasets to perform complex tasks or create new data sharing the same properties, we inquire into the possibility of constructing a DNN that would allow us to bypass the use of these costly simulations when looking to create new simulation data.

Among the DNNs, generative models such as the Generative Adversarial Networks (or GANs) (Goodfellow et al., 2014) have shown promising results both in computer science and physics (Casert et al., 2020; de Oliveira et al., 2017; Ahdida et al., 2019). These networks aim to learn a probability distribution as close as possible to a considered dataset's in order to then generate new instances that follow the same statistics. GANs have proven to be very promising tools in terms of media generation (Donahue et al., 2018; Clark et al., 2019). In Astronomy, they have recently been used in several cases (Rodríguez et al., 2018; Feder et al., 2020; Ullmo et al., 2021) to provide a fast and light alternative to simulations.

In this chapter, we further explore the use of GANs to generate simulation-like data.

Developing a GAN that can effectively emulate data requires work on three fronts, namely building a good model and training method, pre-processing the data such that it is compatible with the model with as little loss of information as possible, and selecting a set of criteria that can determine as objectively as possible

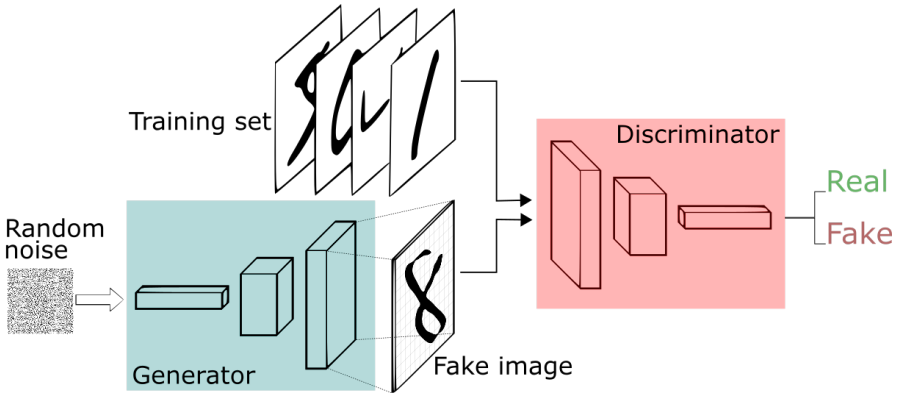

Figure 2.1: Simple diagram of a Generative Adversarial Network.

credit: Thalles Silva (shorturl. at/ zGH12)

the efficiency with which our model is able to emulate the data. This process is far from linear, with problems arising in one domain entailing modifications in another.

In Sec. 2.2, we describe the general concept of the GAN. We will then describe the data on which we train the GAN in Sec. 2.3, from the simulations from which they are constructed, to the criteria needed for them to suit our models and the process with which we construct them from the original simulations. Next we detail the method with which we develop our GAN in Sec.2.4, from finding a functional architecture to the methods to optimize our structure and detect and avoid common GAN pitfalls. In Sec. 2.5 we describe the estimators we use to determine the statistical quality of our generated images. Finally we will present and discuss our results in Sec.2.6 and conclude in Sec.2.7.

2.2 GANs - Generalities

Given a dataset to train on, GANs extract the underlying modes of its distribution and can then generate new data sharing the same distribution and thereby similar properties to the training dataset. Trained correctly, GANs can hence be used to produce an infinite amount of new images given a large but finite number of input images (i.e. training dataset).

The GAN consists of two competing neural networks. The first, a *Generator*, takes a random vector as input from which it produces data (an image or cube in our cases). The second network, a *Discriminator*, tells apart these generated data from *true* ones from the training set. As both networks begin with no information about the data, the tasks of both Generator and Discriminator start out as simple:

the Generator easily "fooling" the Discriminator and the Discriminator having to tell apart very dissimilar images. However as each of the two networks becomes more efficient, one at generating convincing images and the other at differentiating them from the true set, the task is made harder for the other network. Through this competition both networks train each other by gradually increasing the difficulty of the other's task while simultaneously improving themselves.

In practice, the networks work in the following way. The Generator takes a random (Gaussian-distributed in our case) vector (z) as input and from it builds a datum ($G(z)$) through a series of deconvolutions and activations further described in section 2.4.1. The Generator's goal is to intake random variables that are easy to generate (typically following a Gaussian or uniform distribution) to then transform them towards a complex distribution of correlated variables. The probability learnt by the Generator can be expressed in the following way:

$$p_{gen}(\boldsymbol{x}) = \int d\boldsymbol{z}\delta\left(\boldsymbol{x} - G(\boldsymbol{z})\right) p(\boldsymbol{z}) \tag{2.1}$$

where $p(\boldsymbol{z})$ corresponds to the chosen distribution for the input variables.

The Discriminator takes in a datum, either from the training set (x), or from the set produced by the Generator ($G(z)$), and through a series of convolutions and activations further described in section 2.4.1 yields a single number $D(x)$ or $D(G(z)) \in [0; 1]$ which can be interpreted as the probability that the input datum is drawn from the training set.

The training procedure can therefore be described as follows. First, the Generator will generate a batch of images/cubes (in our case 50 for 2D and 100 for 3D) and the same amount of images/cubes will be drawn from the training set (which we will describe in section 2.3). Then, the parameters θ_D of the Discriminator will be adjusted such that the probability given by the Discriminator that the generated data are *true* decreases, while at the same time the same probability computed on the training set increases. Denoting $m = 0, \ldots, N$ the indices of the generated images/cubes $z^{(m)}$ and the training set of images/cubes $x^{(m)}$ in the batch of size N, we want to maximize the following loss:

$$l_G = \prod_m D(x^{(m)}) \prod_m (1 - D(z^{(m)})) \tag{2.2}$$

which is equivalent to minimizing the following log-loss

$$L_D = -\frac{1}{2}\mathbb{E}_x \log D(x) - \frac{1}{2}\mathbb{E}_z \log(1 - D(G(z))) \tag{2.3}$$

where $\mathbb{E}_x$ represents the average over the dataset and $\mathbb{E}_z$ the average over the random vector z. To minimize this expression, the Discriminator should yield a prediction near to one for the data of the training dataset and near to zero

for the data produced by the Generator. Conversely, the Generator should aim at producing images/cubes that look like "true" images/cubes, and so for the Discriminator to yield predictions close to one when assessing its generated data. Therefore the Generator's loss is simply defined as:

$$L_G = -L_D. \tag{2.4}$$

At the end of the training stage, the two networks should converge to an equilibrium wherein the Discriminator is unable to distinguish between the two sets of data and the Generator is outputting data sampled from the training set's true underlying distribution.

In practice, most GANs, including ours, never perfectly reach this equilibrium and instead reach a point where the quality and diversity of the generated images fluctuates with training(Mescheder et al., 2018). Therefore instead of stopping training and collecting the resulting networks at a specific point we elect to save the weights of our networks regularly during training and choose the best set of weights by comparing the quality of images they generated and their statistical properties.

This done, we are equipped with a functioning Generator and Discriminator which we will further use in the construction and training of an Autoencoder.

2.3 Data

We build networks and conduct statistical tests for three types of data: 2D images built from 2D simulations and projected 2D images and 3D cubes, these last two both built from 3D simulations (we will refer to the images from projected 3D simulations as *projected images*). The 2D simulations provide a simplified best-case scenario, and the 3D projected images have potential applications on observational *projected* probes such as lensing (Kaiser et al., 1994) and Sunyaev-Zeldovich (SZ) effects (Birkinshaw, 1999). Finally the 3D simulations are used for direct application in studying simulations of the density field.

2.3.1 Simulations

The 2D images are produced from a publicly available 2D particle-mesh N-body simulation code[1] to simulate 1000 2D snapshots of size $(100Mpc/h)^2$ with 512^2 particles using the standard ΛCDM cosmology.

In detail, the evolution of matter distribution along the cosmic time is described by a Hamiltonian system of equations that are solved using the Leap-frog method.

[1]credit: Johannes Hidding https://zenodo.org/record/4158731#.X5_ITJwo-Ch

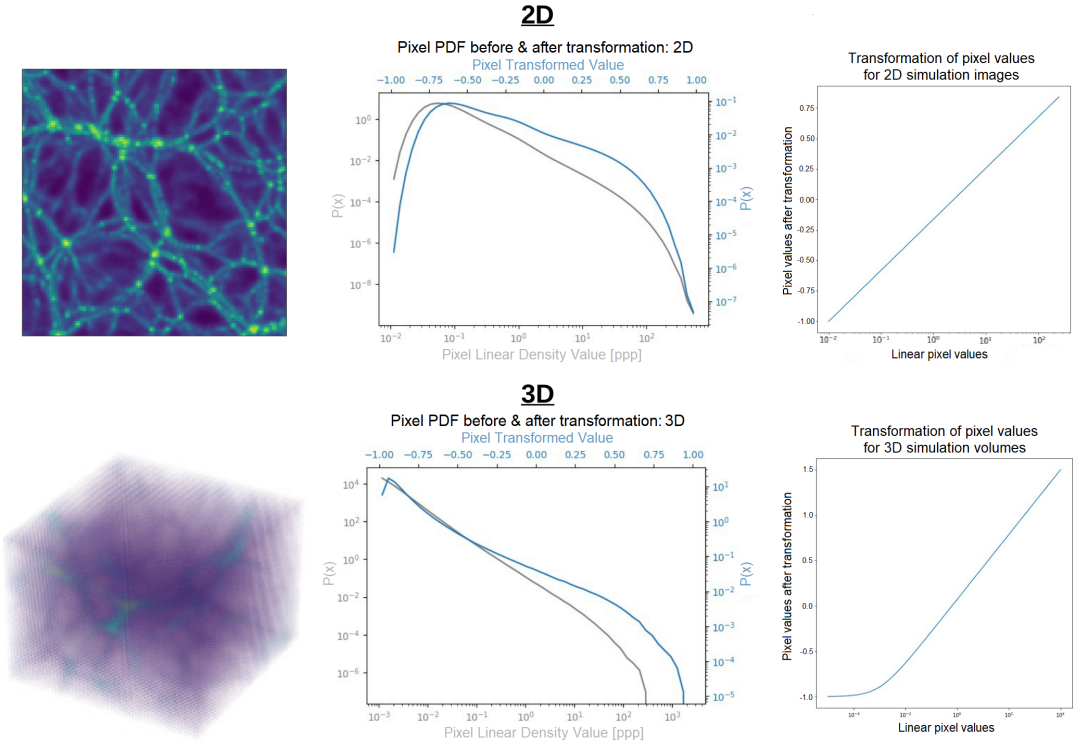

Figure 2.2: Example of a simulation image (*upper left*) or cube (*lower left*), histogram of the pixel values before and after log-like transformation (*middle*), and pixel value transformation function (*right*) for the images (*top*) and cubes (*bottom*). In the middle column, grey represents the pixel values histogram in linear scale and blue the pixel value histogram after log-like transformation (top: eq. 2.5 and bottom: eq.2.6) of the images. For both cases the GAN has been trained using the log-transformed sets of images.

Moreover, the gravitational potential is computed by solving the Poisson equation in Fourier space from the 2D grid density field.

The 3D data used for this analysis are snapshots from numerical simulations of large scale structures produced with the publicly available code GADGET2 (Springel et al., 2001; Springel, 2005). These are dark matter (DM) only simulations, referred to as N-body simulations. GADGET2 follows the evolution of a self-gravitating collisionless particles. This is a good description of DM dynamics in accordance with the cosmological model, since DM only interacts gravitationally with itself as well as with baryons. In practice, the GADGET2 code computes gravitational forces with a hierarchical tree algorithm to reduce computing time and to avoid having to compute the gravitational effect of N particles on each particle (which would mean N^2 computations at each time step). The algorithm divides space using a grid. Then to compute the gravitational forces exerted on an individual particle, GADGET2 groups particles more and more coarsely according to their distance and computes the gravitational pull of the groups rather than that of the individual particles.

The simulation starts at redshift $z = 99$ with a 3D box of 100 Mpc^3 size (chosen to contain representative large-scale structures) with a quasi homogeneous distribution in space of 512^3 DM particles, with Gaussian distributed very low-amplitude inhomogeneities, and an initial velocity associated with each particle. The inhomogeneities stand for the initial density perturbations produced in the early Universe that will eventually evolve into galaxies, clusters and filaments. The system is then evolved with the particles only being subject to gravity. Cosmic expansion is also taken into account and we use the cosmological parameters Ω_m: 0.31, Ω_Λ: 0.69, and H_0: 0.68 from Planck 2018 (Aghanim et al., 2018). The simulation is run up to the present epoch ($z = 0$). At any time step, we can retrieve the individual particles' positions and velocities in the 3D box; we additionally retrieve them at $z = 3, 2.5, 2, 1.5, 1$, and 0.5. These data, describing dynamical state of the system at a particular time, are referred to as a *snapshots*. To build our dataset, we only retain the positions. They will be used as inputs for the network.

2.3.2 Construction of the sample

To be able to apply our networks to our simulations we must turn our snapshots that contain particle positions into discrete arrays representing the particle density fields. This phase in which we pre-process the data is a key step of the overall training process, as ill-constructed data (in the sense that it does not have certain necessary properties that we will detail below) can make the difference between a failing model and a successful one.

We describe below the construction of our samples, and note that the data

construction process is described schematically in Fig.2.4 for 2D images and Fig.2.3 for 3D projected images.

Pre-augmentation data

First for the 2D images, we use a set of 1000 2D simulations. From these, 1000 independent discrete 256×256 density maps are obtained by estimating local densities from 2D snapshots with the help of a Delaunay tessellation field estimator (Aragon-Calvo, 2020). We use this as a basis to construct the images.

For the 3D cubes and projected 3D images, we build a 3D discrete density field from one 3D ($z = 0$) snapshot by computing the histogram of particles over a $768 \times 768 \times 768$ grid. After applying a log-like transformation (see Eq. 2.6), the grid is smoothed with a Gaussian filter of standard deviation the size of three pixels with a stride of three pixels. This choice yields cubes in which structures are smooth while preserving the fine low-density structures and hence results in significantly better results than standard stride-less Gaussian smoothing when used with the different networks in our study. This leaves us with a cube of side 256 pixels and $100Mpc$.

Data augmentation

From the initial 1000 2D images, and the 3D cube (all of side 256 pixels corresponding to $100Mpc$), we extract and augment the final smaller training images and cubes (128 pixels and $50Mpc$ side for 2D/projected and 64 pixels and $25Mpc$ side for 3D cube). This is often done by dividing up the larger arrays into smaller non-overlapping arrays (eg one 256×256 array yields four 128×128 arrays). However, we consider that for the sake of variety and continuity within our training sets, we can instead extract all possible sub-arrays (given the periodic boundary conditions of our larger arrays, this corresponds to one $n \times n$ array yielding $n \times n$ possible sub-arrays, regardless of their size). This quickly leads to a dataset that is too sizeable to load or store (1000×256^2 2D images, 3×256^3 3D images, and 256^3 cubes before rotations); thus we elect to load the larger arrays and randomly extract the smaller arrays on a need-basis, to create training batches or a subset.

We thus generate batches by choosing a set of random positions within the images/cube. Then we extract respectively a set of squares of side 128 pixels or cubes of side 64 pixels for the 2D or 3D case, centered on the random positions. In addition, we rotate/flip these images or cubes by randomly inverting and permuting the axes ($d! \times 2^d$ possibilities for an array of dimension d, i.e. 8 for the images and 48 for the cubes)

These transformations augment the dataset to yield a total of 5.10^8 different possible training images for the 2D images case, 4.10^8 for the projected images

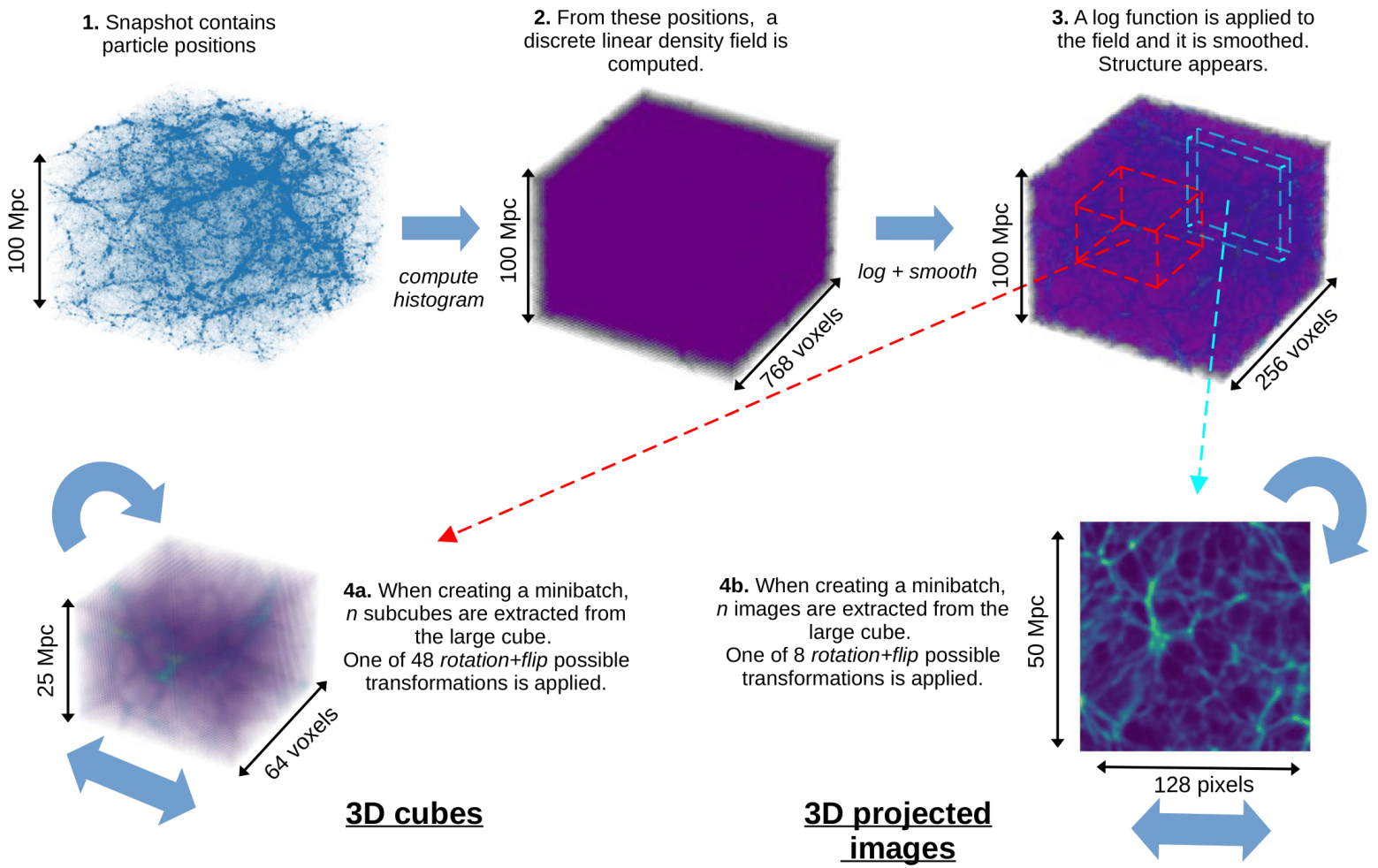

Figure 2.3: A schematic representation of the process of building the 3D cubes and 3D projected images datasets.

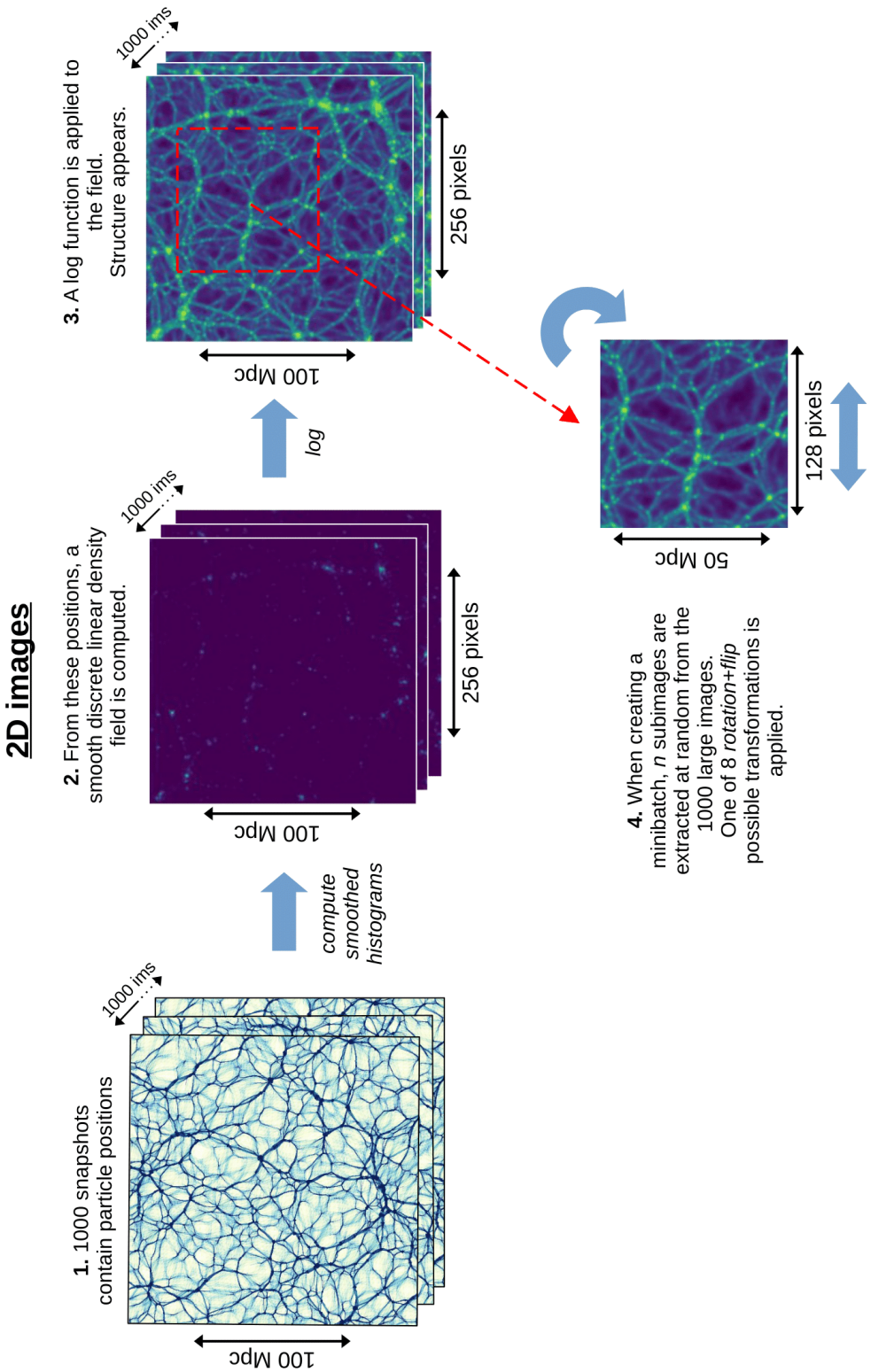

Figure 2.4: A schematic representation of the process of building the 2D images datasets.

case, and 8.10^8 training cubes for the 3D cubes case. However, we expect the networks to capture the original datasets' key features long before having encountered all of the possible images/cubes. Hence, we will not define an epoch as the network having encountered all aforementioned possibilities, but will instead arbitrarily define an epoch as the network having encountered $40,000$ images/cubes. This corresponds approximately to the size of the 2D dataset if we used non-overlapping sub-arrays (32000), half that of the projected set, and ten times that of the 3D dataset (≈ 3000); thus we can expect the network to have on average encountered all possible structures at every angle by the time it has encountered this amount of data.

Data transformation

The GANs operate by using a set of filters to recognize and learn patterns at different size scales in an image. Therefore, we need to work with images with clearly apparent patterns such that the GAN can easily detect the set of salient features. However, linear density maps of the cosmic web show a poor array of shapes, with images appearing mostly uniformly dark with occasional bright pixels corresponding to dense halo centers; on the other hand a log representation of the same density maps make the cosmic web's filaments apparent, providing shapes and texture that the GAN can more readily detect and reproduce. Finally, our GAN is built to intake and output images with pixel values $\in [-1, 1]$; we thus need to map the original pixel's distribution into this interval.

We hence apply a log-like transformation to the images' pixel values (2.2). For the 2D images, the pixel value v' in the "transformed" images writes:

$$v' = \frac{2\log(v) - (b + a)}{b - a} \tag{2.5}$$

v is the original pixel value, a and b are chosen such that $a \lesssim \min(\log(v))$ and $b \gtrsim \max(\log(v))$ so as to have $v' \in]-1, 1[$ compatible with the network outputs. We use these strict inequalities to give the network freedom to exceed its training set's boundaries when generating images with pixel values $\in [-1, 1]$. We thus set $a = \log(0.01)$ and $b = \log(600)$.

The cubes and projected images from the 3D simulation have a significantly larger range of values than the 2D ones with values up to 2000 particles per pixel as well as zero values. For these data, we adapt the above-described transformation to better suit their pixel range and 0 values. We recall that this transformation is applied *before* the smoothing with a Gaussian filter. The obtained pixel value v' writes:

$$v' = \frac{2\log(v + c) - (b_2 + a_2)}{b_2 - a_2} \tag{2.6}$$

where b_2 is chosen such that $b \gtrsim \max(\log(v))$, c is chosen such that $c > 0$ but $c << \bar{v}$ to increase the contrast, while allowing for a log transform, and $a_2 = \log(c)$. We set $b_2 = \log(2632)$, and $c = 0.001$ and $a_2 = \log(0.001)$.

It worth noting that while adding the constant c allows for a log-like transformation, it makes the linear values smaller than c difficult to distinguish from one another after transformation. This can be observed in the lower right panel of Fig. 2.2 which represents the transformation function given by Eq. 2.6. We clearly see a saturation effect for values below $c = 10^{-3}$. We therefore do not expect our networks to recover the pixel pdf correctly for values below c.

Lessons learned when experimenting with data construction

While our 2D images were obtained fully constructed with the help of a collaborator, it bears mentioning that our data created from the 3D simulations went through multiple modifications before converging to the two forms presented here. Indeed, working originally on the 3D projected images, we initially built them with a different transformation function and smoothing based on the method found in Rodríguez et al. (2018). The original transformation function was the following:

$$x' = (x + a)/(x - a) \tag{2.7}$$

with a being first set at 2. This transformation has the advantage of allowing 0 values and making the cosmic web structure stand out, but causes a saturation effect at high density values, which means that dense regions are not well represented by the GAN. We attempted to limit this effect by raising the value of a but lost the filamentary structure. Additionally this less simple transformation (compared to log) led to difficulties in the second stage of our work when trying to use our models for further predictive work. The original smoothing also proved too coarse; rather than smooth from a higher resolution, we used a gaussian kernel directly on 256×256 images, which led to more blurry structures that were harder for the models to detect and emulate. Incidentally it did not combine well with a *log* transformation, as the filamentary structure melded into the low-density background. We also attempted to use linear density images to see if the GAN was able to reproduce them directly but this proved too difficult a task for it, with the generator never managing to converge. Also we used a more standard augmentation, wherein we did not use random positions but instead divided the original 3000 256×256 images into four 128×128 images before applying rotations and flips. This meant a much smaller training set with much less continuity within it, leading to more likely overfitting of the models. Experiencing how much more effectively the different models worked with first the 2D images and eventually the 3D projected images put into light the strong importance of a well-selected pre-processing method for the images.

2.4 Constructing the GAN

Developing a GAN rests on two major aspects: finding a good network architecture and developing an efficient training process. In a first section we detail the components of a GAN's architecture and briefly present a few of the available options for each. We show our experimentation and explain our final choice of architecture. In a second section, we will present our training procedure, the various pitfalls that one can encounter during training and how to detect and avoid them.

2.4.1 Architecture

A GAN is made up of two competing convolutional networks, the generator and discriminator, that are somewhat symmetric in their makeup and tasks, with the generator building up a datum from a small amount of random parameters through a set of upscaling convolutional layers, and the discriminator breaking down a datum into a small amount of meaningful numbers, from which it can identify the datum's nature, through a set of downscaling convolutional layers.

Using the description of the typical CNN components as seen in section 1.2.3, we can detail the architecture of our models.

The generator is typically made up of a first (dense + activation) layer. Its following layers are a set of (upscaling + convolution + normalization + activation) convolutional layers, with the final layer ending with a tanh activation.

The discriminator on the other hand is made up of a set of (downscaling + convolution + normalization + activation) convolutional layers, and a final dense layer ending with a *sigmoid* activation, to allow it to classify data into the "real" or "fake" category.

In a first part of our work we spent a significant time attempting to construct both models by developing them from a very simple architecture (see table 2.5) optimized to emulate an MNIST dataset. We were aiming to make use of our model to emulate the 3D projected data described in section 2.3.2.

MNIST (Deng, 2012) is a large set of 28×28 pixel greyscale hand-drawn pictures of numbers (see fig.2.6, upper left). These small, simple and contrasted data differ greatly from our own target data, which is of size 128×128 pixels and presents more complex structures with smooth pixel value gradients. Thus, to make the task easier while we gradually modified the models, we constructed "intermediate" datasets that were more similar to MNIST. A first "smooth" dataset (see fig. 2.6 upper left) was built by building our data with less resolution so as to have smaller 64×64 pixel data that the GAN could more readily process. Additionally, from these we also built a "contrasted" dataset by setting all pixel values above a certain threshold to 1 and those below to -1, so as to emulate MNIST's contrasted nature. With these datasets we aimed to set intermediate goals for the modified GAN to

Filter sizes	$\{5, 5\}$
$n_{filter}(G)$	$\{64, 1\}$
$n_{filter}(D)$	$\{64, 128\}$
Pooling Size:	$\{2, 2\}$
Layer Act.	Tanh (G), Tanh (D)
Final Act.	Tanh (G), Sigmoid (D)
Latent dimension	100

Figure 2.5: Architecture specifications for each layer of our original GAN's generator network (G) and discriminator network (D). This base structure follows a publicly available GAN structure[2]. The networks are trained using *Stochastic Gradient Descent* (SGD) as an optimizer and minimize a binary crossentropy loss (see eq. 2.3).

reach. We planned to transition from the contrasted images to the smooth images, eventually reaching the high-definition target images. In parallel we experimented with every possible component (n° of layers, activations, optimizers, convolution specifics etc) as can be seen in fig 2.6, further building upon models that showed progress. While progress could occasionally happen, many attempts led to dead ends, and even the best results were clearly distinguishable from true data, often because of lack of diversity in the generated data. Additionally the models proved quite unstable, with two distinct modifications oftentimes making the models dysfunctional while independently giving better results. Too much change inevitably led to the models' complete dysfunction.

Although this first phase proved useful to gain a good understanding of the different components used and the way they interacted to form our models, we came to the conclusion that our original model structure was too simple and far-removed from current models built to emulate natural images and could not be brought to emulate our data through gradual change. All in all, attempting to build a GAN from the ground up proved to be a Sisyphean task that was best avoided altogether.

Consequently, in a second phase we chose to develop our GANs based on a pre-existing one[3] built to emulate natural images and requiring minimal modification to make it compatible with our data. Although our data remains quite different to that which this second GAN is intended for (RGB bird pictures), we can expect it to translate well as our data is overall simpler (greyscale, self-similar, isotropic and homogeneous). Luckily, transition to 3D does not cause any issues.

We thus build our GANs following the architectures detailed in 2.7. This done,

[2]credit: Jacob Gildenblat, https://github.com/jacobgil/keras-dcgan
[3]Tiago Freitas, https://github.com/tensorfreitas/DCGAN-for-Bird-Generation

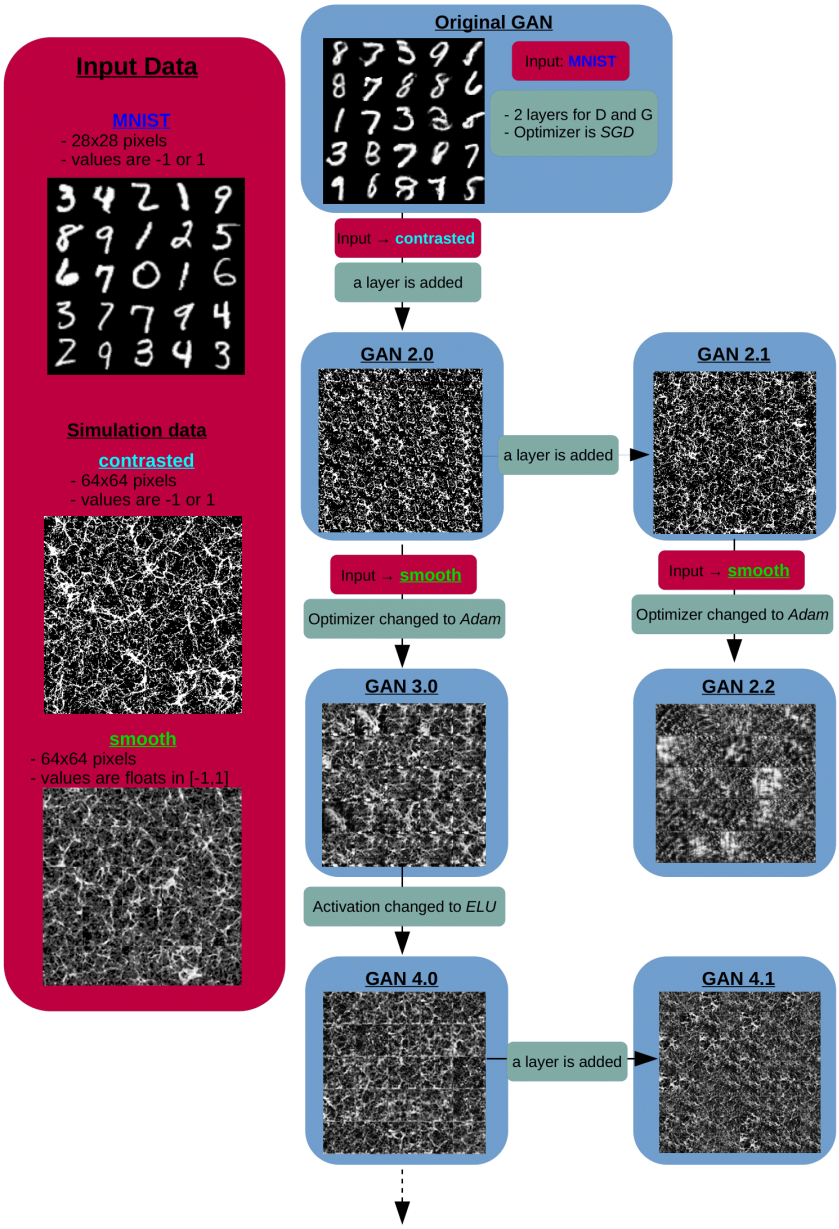

Figure 2.6: A non-comprehensive diagram representing the attempt to develop a GAN from a simple architecture made to emulate 28×28 pixel MNIST data.In the upper left box the "true" data the models are tasked with emulating are pictured. The central tree structure represents the best batches generated by a given architecture, and the modifications leading from one architecture to the other.

we can start training our model.

	2D GAN	**3D GAN**
Filter sizes	$\{5, 5, 5, 5, 5\}$	$\{4, 4, 4, 4\}$
$n_{filter}(G)$	$\{256, 128, 64, 32, 1\}$	$\{128, 64, 32, 1\}$
$n_{filter}(D)$	$\{32, 64, 128, 256, 512\}$	$\{32, 64, 128, 256\}$
Strides:	$\{2, 2, 2, 2, 2\}$	$\{2, 2, 2, 2\}$
Layer Act.	ReLU (G), Leaky ReLU (D)	ReLU (G), Leaky ReLU (D)
Final Act.	Tanh (G), Sigmoid (D)	Tanh (G), Sigmoid (D)
Latent dimension	100	200

Figure 2.7: Architecture specifications for each layer of the 2D (*left*) and 3D (*right*) GANs' generator networks (G) and discriminator networks (D). They are all based on a publicly available GAN structure[4]and are trained using the *Adam* optimizer with parameters ($lr = 0.0002, \beta_1 = 0.5$) and minimize the loss given in eq. 2.3.

2.4.2 Training process and Optimization

We recall that CNNs including our GAN are trained in the following way: at every time step of training they perform their given task on a set number of data called a batch, from which can then be computed a loss. At each time step the weights are updated to reduce the loss by computing its gradient and updating the weights to reduce it. Training is led and supervised in the following way: the model is run over several epochs, and we regularly save data generated by the GAN, GAN losses, and GAN weights. In our case we find that a batch size of 100 gives good results. We set the number of epochs high enough for the models to have well converged, at 100 epochs.

As a first check, observing the losses and the generated data allows us to ensure that training is proceeding properly. When so, we can expect the data created by the generator to progressively transition from completely random pixel distributions to simulation-like data, with structures within the images/cubes gradually gaining more detail with training (see Fig.2.8). Additionally we expect the data to be diverse, with no two data in a generated batch looking alike. The losses of the generator and discriminator are expected to be very noisy as an indication of the two models competing for their respective goals, and successively "taking the upper hand" in the competition (see Fig.2.9, *above*).

Training can fail in several ways. A first scenario is when neither Generator nor Discriminator manage to progress in their respective tasks from initial random

[4]Tiago Freitas, https://github.com/tensorfreitas/DCGAN-for-Bird-Generation
[5]https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

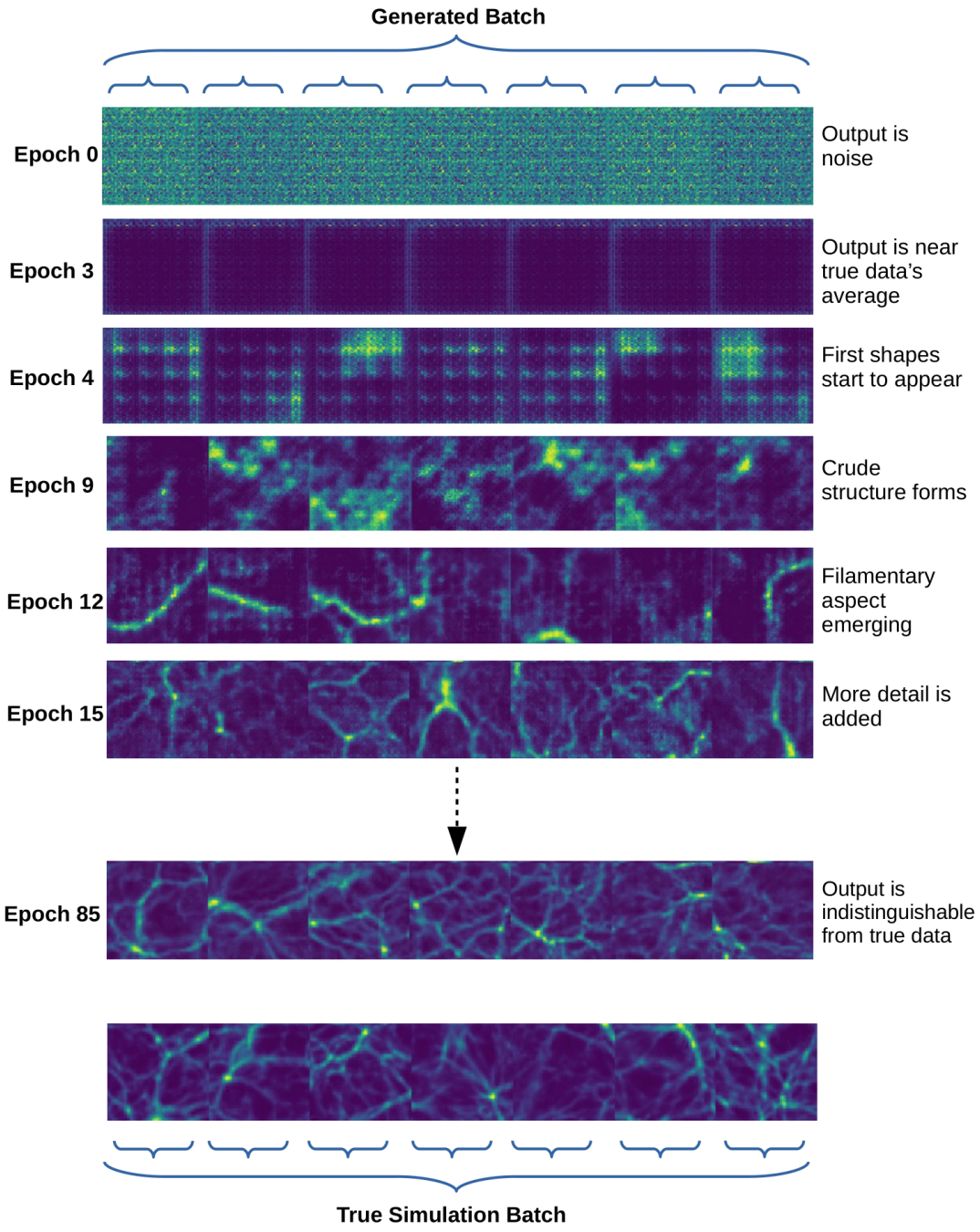

Figure 2.8: An example of a "healthy" GAN training progression. Pictured above are batches of data generated at different epochs by the GAN. Below is an example of true simulation-issued data.

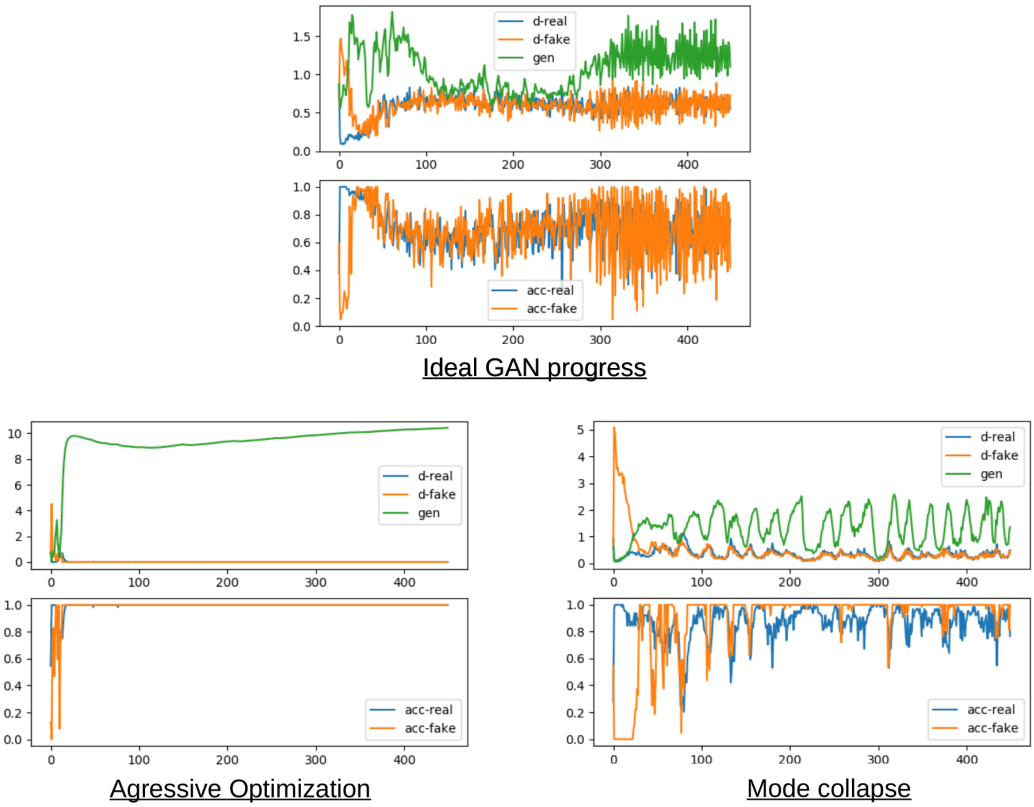

Figure 2.9: Examples showing three scenarios for the GAN training process. For all scenarios, the upper plot shows the generator's loss in green, the discriminator's loss on real data in blue and the discriminator's loss on fake data in orange. The lower plot shows the discriminator's accuracy in determining real data in blue and fake data in orange. The first scenario (*above*) shows an ideal training progress. All losses vary noisily over training time, along with the discriminator's accuracy for both data types. The second scenario (*lower left*) shows a case of aggressive optimization, where one network (here the discriminator) surpasses the other completely, preventing it from progressing altogether. The losses are thus constant (maximal for the generator and minimal for the discriminator), and the discriminator has 100% accuracy. The third scenario (*lower right*) shows a case of mode collapse, where the generator fools the discriminator with fake data that is similar to the true data, but has little diversity (ie it only displays one mode of the true data distribution). Once the discriminator has learned to spot them, the generator switches to another mode and the process is repeated indefinitely. This is exhibited in the losses' sinusoidal evolution, where downward peaks in the generator's loss correspond to a change of loss and the following upward slopes correspond to the discriminator's progress in discovering the new emulated mode.

credit: Jason Brownlee[5]

conditions. This is easy to diagnose when looking at the generated data as it never visually approaches the true data, often remaining as a random distribution of pixels or displaying simple patterns.This is generally due to bad model architecture or bad data (in the sense that it is incompatible with the model) and is solved by reverting to previous working conditions (earlier model structures or data type). A second common scenario happens when one model, usually the discriminator because of the relative ease of its task, takes too much lead in the competition. This can lead to the other model not being able to progress, because its loss's local gradient is zero (small weight modifications are insufficient to impact the loss). Looking at the loss evolution makes this very clear as the losses remain constant or near-constant with very little noise as soon as this happens (see Fig.2.9, *lower left*). This can be solved by improving the under-performing model or updating its weights more frequently, but we find that simply adding noise to the labels (1 for true data and 0 for fake data become 1-n and n respectively, n being randomly generated noise), suffices to completely avoid this problem, at the cost of slowing down the training. A final common problem is mode collapse, wherein the generator creates data that can successfully fool the discriminator but that have very little diversity among them. This can be detected quite easily when looking at the generated images, with specific patterns appearing repeatedly in a generated batch, and with the losses as well (see Fig.2.9, *lower right*). Although this was an occasional problem during our different tests, it was random and uncommon enough that we did not find the need to resolve it so much as cast out the models in the instances where mode collapse had occurred and resumed training from earlier saved models.

2.5 Statistical estimators

In cases where all such pitfalls are avoided, and the generated data seem visually diverse and indistinguishable from the true data, we must use finer statistical estimators to ensure that our generated data is consistent with our true simulation data.

2.5.1 Pixel PDF and Distribution of the mean density

A first basic test is to compare the distributions of pixel values, which correspond to a density measure in particles per pixel (ppp) in both sets of data. We also compute the mean particle density, μ, of each image/cube and compare their PDF over the simulated (truth) sets and generated and inferred sets from the GAN and AE.

Whereas the pixel PDF is informative of the density distribution of a datum

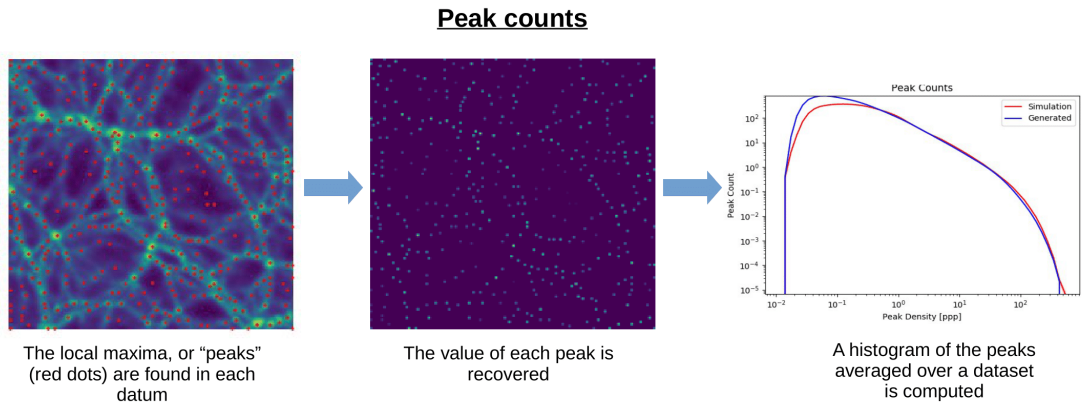

Figure 2.10: An example of the peaks recovered in a simulation image.

on average, and therefore ensures that two sets of images/cubes are similar on average, the mean density serves as a simple one-dimensional visualisation of the distribution of images/cubes over a set. This type of information is important to ensure that we recover both datasets' underlying distribution, and recover different cosmic regions and different halo densities in the right proportions.

Furthermore GANs can often suffer from "mode collapse" (Thanh-Tung and Tran, 2018), a situation where the images/cubes generated are indistinguishable from the original set but show little to no diversity. Although visual inspection of the images can help to exhibit mode collapse, a visualisation of the overall distribution through the mean density provides additional information to confirm its absence.

2.5.2 Peak counts

We compute the average peak counts over the generated or simulated dataset. A peak is a local maximum ($\rho_{\max}$) defined as a pixel whose contiguous neighbors (8 for images and 26 for cubes) are of smaller values. For each image/cube, we compute the number of peaks for a given value of $\rho_{\max}$, and average this number over the whole dataset (see Fig.2.10.

In the simulated data, the higher peaks, being dense local maxima, are expected to correspond to halo centers, whereas smaller near-zero peaks are more likely to be the result of noise from the simulation or the image-making process. Therefore, we are more interested in the former, which give us an indication as to our recovery of halo distribution.

2.5.3 Power Spectrum

We compute the 2D/3D power spectrum of each image/cube from the different sets (input, generated and inferred). For a frequency ν it is given by:

$$P(\nu) = \langle \|A_{kl}\|^2 \rangle_{(k,l)|k^2+l^2=\nu^2} \tag{2.8}$$

where $A_{k,l}$ are the image's discrete Fourier transform elements.

2.6 Results

2.6.1 2D images

We first present the results of the GAN trained on images from 2D simulations. The network consistently outputs sets of verisimilar images as early as 30 epochs but in our study the GAN is trained for 85 epochs for best results. It is worth noting that we can expect the relative simplicity of the images of our training set to result in a faster convergence than a GAN trained on natural images. We also note that our chosen number of epochs, consisting of 40,000 images each, corresponds to longer training than that of (Rodríguez et al., 2018) (20 epochs for a dataset of 15,000 projected cubes).In addition, training the GAN for too long (e.g. > 100 epochs) eventually results in mode collapse whereas the quality of the generated data stops improving long before that point is reached.

Two sets of 50 images taken at random from the simulations and from the GAN's generated images show, in Fig. 2.11, the GAN's ability to generate images of convincing similarity. Visually, we observe that the large scale structure is well recovered; this is most perceptibly the case for the filaments, reproduced in all their diversity of length, thickness and frequency. It is also the case for high-density regions, or halos, in terms of their occurrence, brightness (or density) and positions within the structures.

This observation is further corroborated by the statistical estimators as seen in Fig. 2.12. The pixel PDFs (lower left panel of Fig. 2.11) show a near-perfect overlap for the majority of the pixel density values, with a very slight under-representation in the generated images of the densest values. This agreement shows that the density distribution of the images is very well recovered by the GAN.

The mean particle density distribution of the generated set (displayed in the upper right panel of Fig. 2.12) shows a near-perfect agreement with the simulation set. The overall agreement indicates that the diversity of the original set is globally well represented in the generated set. The median power spectra and their *mad* layer (Fig. 2.12 upper left panel), for both simulated and generated sets, show

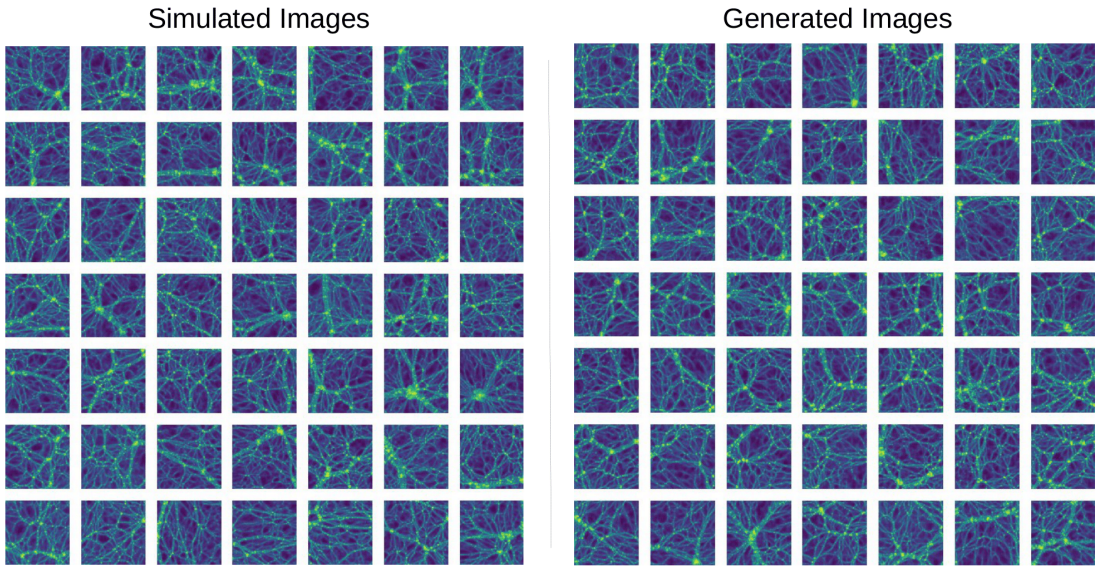

Figure 2.11: Two subsets of 50 images taken at random from a set of 2D simulation images (left) and a set of images generated by the GAN (right). Every image represents a 128×128 log density map of side $50Mpc$. They are virtually indistinguishable by eye.

a satisfactory overlap, indicating a good recovery of the correlations at various distances and thus a good representation of the different scales in the images. Finally, in the lower right panel of Fig. 2.12, we show the peak counts. The very good agreement between the true simulated images and the generated ones confirms that the dense regions are well represented. Indeed, the matching curves show us that both the average number of peaks in a datum and the distribution of values among these peaks are conserved in the generated set, with a slight under-representation of the densest peaks and a more notable over-representation of low-density peaks. However, the peaks at low density are due to simulation noise and not physical, so this is not an issue.

2.6.2 3D projected images

Next we look at the results obtained by the GAN trained on the 3D projected images. Once more the GAN progresses in a stable manner, consistently producing verisimilar images after around 20 epochs of training. For best results we train it for 70 epochs.

First we focus on two subsets taken at random from both the original set of simulated images and the set of generated images (Fig. 2.13). Once again our visual inspection shows that the diversity of the simulated images is well recovered

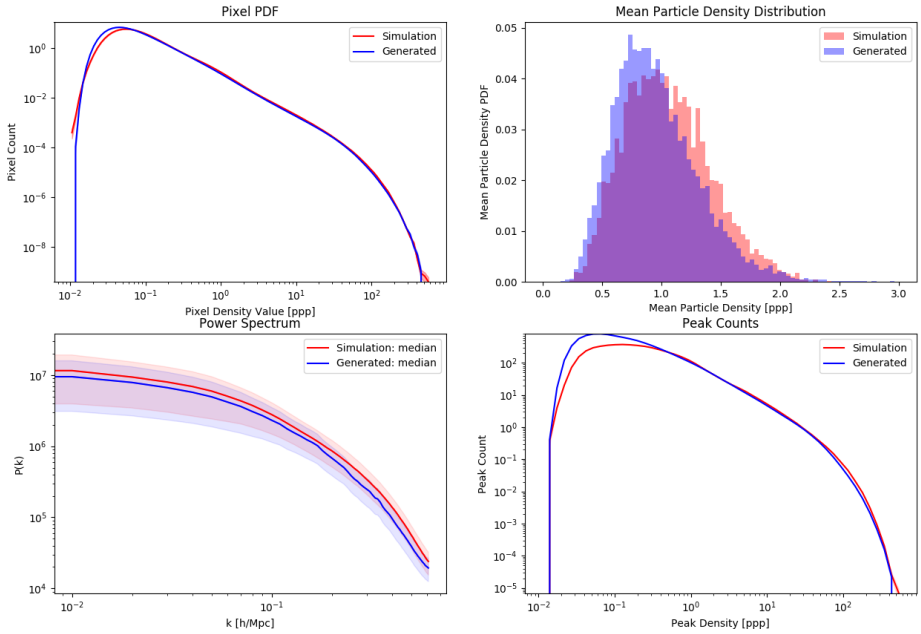

Figure 2.12: Statistics of the 2D simulation images compared to their GAN-generated counterparts. *Upper left* shows the pixel PDF, *upper right* shows mean density distribution, *lower left* shows median power spectrum as well as the median absolute deviation (mad) layer, and *lower right* shows average peak count per image. The curves overlap near-perfectly.

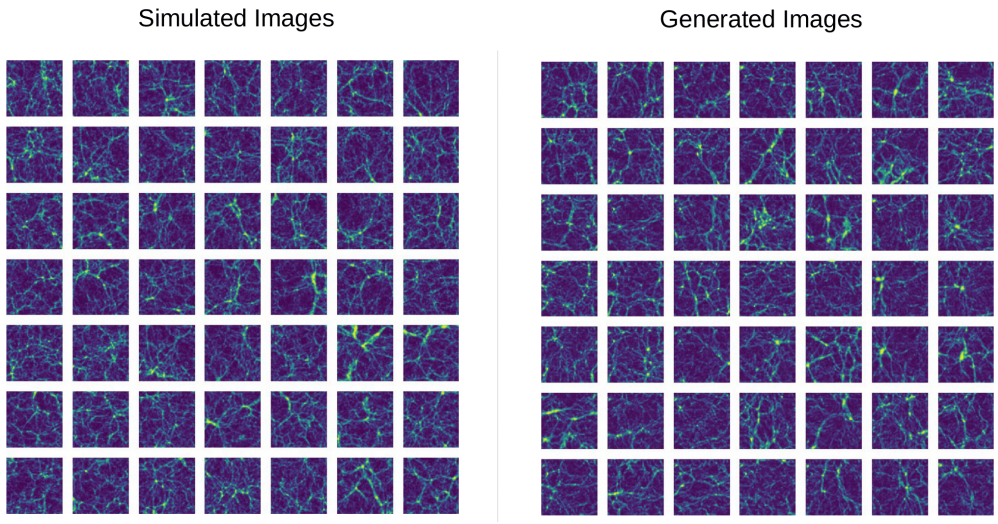

Figure 2.13: Two subsets of 50 images taken at random from a set of 3D projected images (left) and a set of images generated by the GAN (right). Every image represents a 128×128 log density map of side $50Mpc$.

by the GAN in terms of distribution in size and frequency of filaments and number and brightness of high-density regions. A closer look at the statistical properties of the images as seen in Fig. 2.14 further confirms this.

Notably, the pixel PDFs (Fig. 2.16 upper left panel) show a near-perfect overlap, confirming the good recovery of the density distribution on the average images. However, the lower tail of the distribution is poorly represented for pixel values $< 10^{-3} ppp$. The generated images show a deficit, while the simulations show a plateau. This can be explained by the saturation effect related to the constant c in Eq. 2.6; indeed, adding the constant c before log-transforming the linear densities renders linear values $<< c$ difficult to distinguish from one another.

Meanwhile, the mean density PDF (Fig. 2.16 upper right panel) seems to be very well recovered, confirming the good recovery of the image diversity. The median power spectra and their *mad* regions (Fig. 2.16 lower left panel) yield a near-perfect overlap, with a slight under-representation of higher frequencies in the generated images. In the lower right panel of Fig. 2.16, we plot the peak counts. True simulated images and generated ones show once more near-perfect agreement, confirming that high-density region centers are well represented in terms of their numbers as well as their distribution. However, we observe a misrepresentation of the distribution lower tail similarly to the pixel PDF, for similar reasons.

For all of the estimators presented here and that are common with Rodríguez et al. (2018) and Feder et al. (2020), our findings agree with theirs. The mean

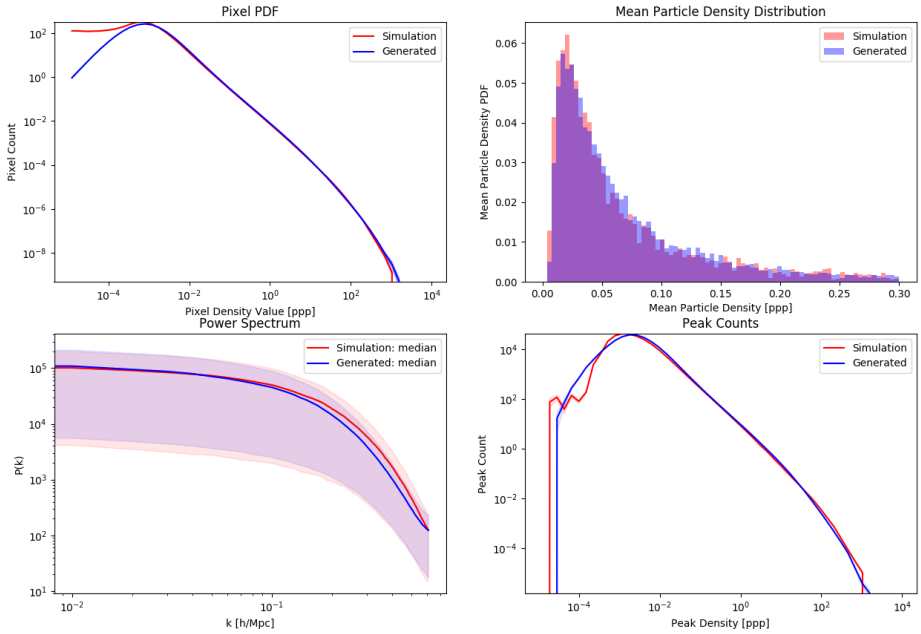

Figure 2.14: Statistics of the 3D projected images (red) compared to their GAN-generated counterparts (blue).Upper left shows the pixel PDF, upper right shows mean density distribution, lower left shows median power spectrum as well as mad (median absolute deviation) layer, and lower right shows average peak count per image.

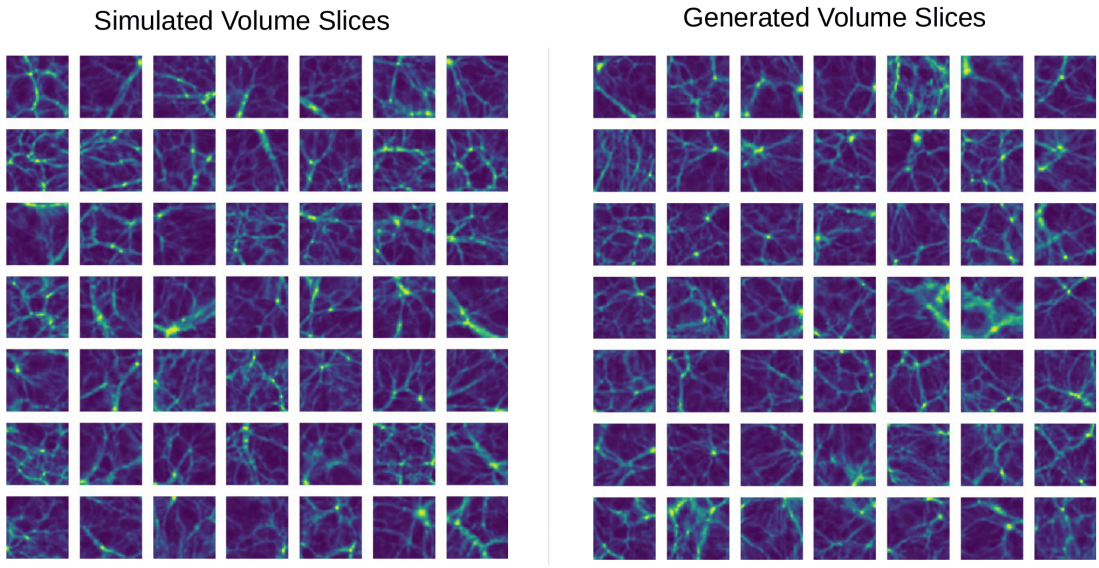

Figure 2.15: Two subsets of cube slices (of thickness $\Delta z \approx 0.4 Mpc$) taken at random from a set of 3D simulation cubes (left) and a set of cubes generated by the GAN (right). Every cube represents a $64 \times 64 \times 64$ log density map of side $25 Mpc$.They are virtually indistinguishable by eye.

density distributions of both simulated and generated sets are somewhat distinguishable but show a very good overlap, and the power spectra show a very satisfactory overlap between both their medians and their *mad* regions. We note that our results seem coherent with those of Feder et al. (2020) who encountered similar saturation issues to ours at low densities for similar reasons. We also note that Rodríguez et al. (2018) do not refer to a poor representation of tails in the pixel PDF and peak counts for their GAN-generated images, despite using a pixel transformation with a saturation effect.

2.6.3 3D cubes

We now turn to the results obtained by the GAN trained on the 3D cubes. Once more, the GAN progresses in a stable manner, consistently producing verisimilar cubes after around 30 epochs of training. For best results we train it for 50 epochs; we find this coherent with the training time of Feder et al. (2020) (150 epochs for a dataset of 16,000 cubes). As for the 2D GAN, training for too long results in mode collapse, but not before the quality of the generated cubes stabilizes.

First, we focus on two subsets taken at random from both the original set of simulated cubes and the set of generated cubes (Fig. 2.15). Once again our visual

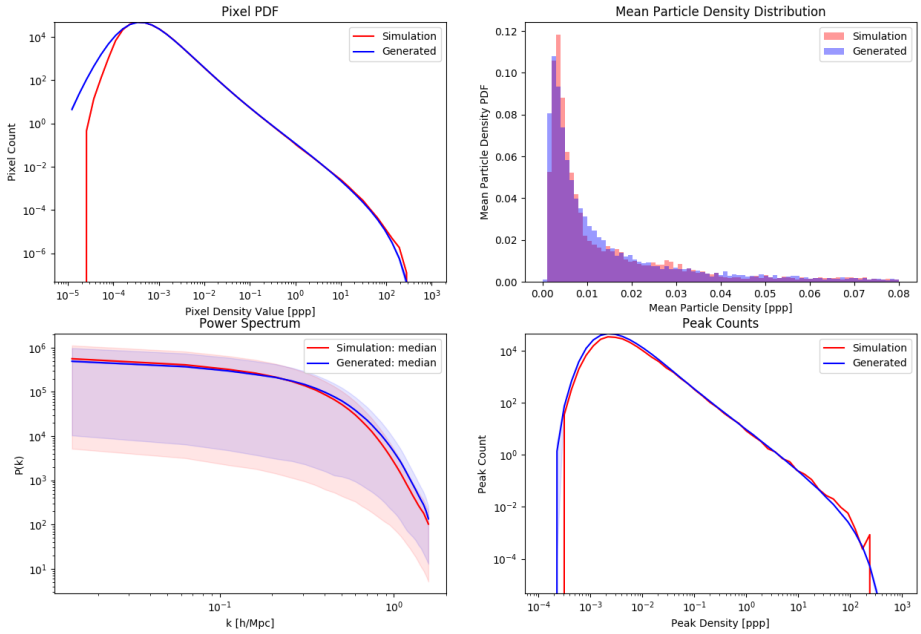

Figure 2.16: Statistics of the 3D simulation cubes (red) compared to their GAN-generated counterparts (blue).Upper left shows the voxel PDF, upper right shows mean density distribution, lower left shows median 3D power spectrum as well as mad (median absolute deviation) layer, and lower right shows average peak count per cube.The curves overlap near-perfectly.

inspection shows that the diversity of the simulated cubes is well recovered by the GAN in terms of distribution in size and frequency of filaments and number and brightness of high-density regions. A closer look at the statistical properties of the images as seen in Fig. 2.16 further confirms this.

Notably, the voxel PDFs (Fig. 2.16 upper left panel) show a near-perfect overlap, confirming the good recovery of the density distribution on the average cubes. However, the lower tail of the distribution is poorly represented for voxel values $< 10^{-4} ppp$. The generated cubes show a deficit compared to the simulations. This can be explained by the saturation effect related to the constant c in Eq. 2.6.

Meanwhile, the mean density PDF (Fig. 2.16 upper right panel) seems to be well recovered, confirming the good recovery of the cube diversity. The median 3D power spectra and their *mad* regions (Fig. 2.16 lower left panel) yield a near-perfect overlap, with a slight over-representation of higher frequencies in the generated cubes. In the lower right panel of Fig. 2.16, we plot the peak counts.

True simulated cubes and generated ones show once more near-perfect agreement, confirming that high-density region centers are well represented in terms of their numbers as well as their distribution. However, we observe a slight misrepresentation of the distribution lower tail similarly to the voxel PDF, for similar reasons.

For all of the estimators presented here and that are common with Feder et al. (2020) namely the mean density distributions and the power spectrum, our findings agree with theirs. We find that the mean density distributions of both simulated and generated sets are somewhat distinguishable but show a very good overlap, and that the power spectra show a very satisfactory overlap between both their medians and their *mad* regions. We note that our results seem coherent with those of Feder et al. (2020) who encountered similar saturation issues to ours at low densities for similar reasons. Given the fact that Rodríguez et al. (2018) dealt with projected 3D simulations whereas we present results for actual 3D cubes it is not possible to properly compare with them.

2.7 Conclusion

We explained the process with which we developed and trained GANs on several types of data built from simulations to produce new data that is statistically consistent with the old as a way to bypass costly simulations and as a first step towards further work that we will detail in the next chapter.

In a first part we explained the general concept behind a GAN.

In a second part we detailed the data we used and the different choices we made when constructing it. The data was constructed from both 2D and 3D DM-only N-body simulations and was pre-processed to be compatible with the GANs. Concretely, this amounted to turning a set of particle positions into 2D and 3D discrete density fields and applying a log-like transformation to them. We concluded the following: given that GANs are CNNs and are therefore made up of filters that detect or create reoccurring salient (i.e. smooth and contrasted) patterns, we must make sure that our input data contains such patterns. The log-like transformations allow for the cosmic web structure to emerge clearly, in significant contrast with the low density background. Smoothing the density field ensures that the structures themselves are smooth and continuous. Finally to ensure that GAN-generated densities were consistent when linear, we had to make sure that our log-like transformation gave sufficient contrast in the density ranges we considered. In the end we came up with three types of data: 2D density maps built from the 2D simulations at z=0, 2D projected density maps built from the 3D simulations at z=0, and finally 3D density field cubes built from the 3D simulations at various z (3 to 0).

In a third part we explained our process when developing and training our

GAN, from architecture choices and presentation of different components to training choices and checks to detect and avoid common issues during training. Experimentation with various architectures led us to conclude that while certain general rules could be followed with regards to component choice for a better chance of stability, there were no universal truths that completely ensured it, and too much modification of a functioning model inevitably led to dysfunctional models. Thus we find that the best course of action is to find a working model that requires relatively minimal modification to be compatible with one's data rather than to build one from the ground up.

In a fourth part we put forward the importance of using statistical estimators to ensure that the data generated by the GAN is consistent with the original data. We came up with a set of estimators that comprehensively verified that the data simultaneously conserved the correct mass distribution (average pixel PDF), represented all the scales within the cosmic web (power spectrum), kept the diversity of data (mean density PDF) and kept the same distribution of local maxima (peak counts).

Thus equipped we were able to train a GAN, and using the previously mentioned estimators, we confirmed the ability of the GAN to extract the underlying statistical distribution of data built from the simulations and generate new data hailing from this distribution. We showed that this was the case for all data types. They were indeed emulated with striking similarity to the true original data, as we showed visually and with the almost perfect overlap of the different statistical estimators. Additionally, the training proved stable. The networks consistently generated images of increasing quality with training up to a stable point after which the generated images were visually indistinguishable from the true ones.

We note that despite the success of GANs to reproduce a desired input with high fidelity, it is important to be careful when these black-box models are used.

However we find our results quite encouraging, and with this first step successfully taken we can look into different paths for optimizing results, from simple modifications to training, such as using larger batches, longer training, and upscaling our models with more intermediate layers or more convolutional filters to add variety to the generated patterns, to complete change to stronger available architectures.

In terms of quality assessment of the generated data, inquiring into more GAN-focused estimators such as Inception Score (Salimans et al., 2016) and Fréchet Inception Distance (Heusel et al., 2017) could help us to best gauge the progress of our GAN during training and make modifications accordingly.

A next logical step would also be to work with higher resolution data. In a similar vein, it could be worth experimenting with more complex hydrodynamical simulations, seeing if we can generate multiple-input maps that contain not

only DM density but also velocity distribution, gas density, galaxy population or temperature maps.

Additionally we can expect more complex applications of GANs to apply well to our data. Namely, cGANs(Mirza and Osindero, 2014) are types of GANs that can be conditioned to generate data that is conditioned on class labels. In our case, this could be interesting in terms of generating simulation data either at various redshifts, or with certain conditions on the type of cosmic region that is generated (dense, cluster-filled regions vs empty regions for example). Many examples of modified GANs have shown stunning results in terms of filling in incomplete information, or *imputation* (Lee et al., 2019; Yoon et al., 2018; Bora et al., 2018), which would have multiple applications on observational data, as a way to fill masked areas, complete flawed observations or reconstruct partially detected structures. Others yet have proved effective in detection of objects within images (Isola et al., 2017), which could be useful when attempting to classify/find structures in observed data, especially in upcoming large surveys.

Additionally, rather than constructing more complex GAN structures, one can exploit a trained GAN's properties to build models with more pointed tasks. We will show in the following chapter how we can make use of both a trained generator and discriminator to first build an autoencoder that is able to first reconstruct data from a small number of parameters and in a second phase predict structure evolution in time.

Chapter 3

Predicting Structure formation in Simulations with GAN-based Autoencoders

3.1 Introduction

Having described in Chapter 2 our work to develop the GAN we now take advantage of its components' properties for a more concrete application; indeed both the generator and discriminator can prove quite useful in independent manners, with both having semantically meaningful spaces to which they can translate our cosmological data. First the generator, which is built to map a space of n independent Gaussian-distributed parameters to a space of statistically consistent cosmic web data, presents the key characteristics one would expect of a decoder, as we will describe below. Second the discriminator, which is optimized to pick up subtle features to determine whether a given datum is issued from true LSS simulations or generated by a model, can be envisioned to hold within its network nodes a meaningful representation of data which we could use to our advantage.

This being posited, we describe how we make use of our GAN to create a series of autoencoders, first simply tasked with reconstructing input data after having encoded it in a low-dimensional vector, then, more interestingly, with the task of predicting structure formation from past snapshots of the same Lagrangian space. Indeed, our single simulation yields snapshots of the same density field at different stages of evolution in time (in our case we focus on redshifts $z = 0, 1, 2$ and 3). We task our autoencoder with recovering density fields at $z = 0$ while providing it with the equivalent field at $z = 1, 2$ and 3.

In a first part, we will introduce the general concept of the autoencoder and describe the method with which we build our own from the trained GAN structure;

hence there will be no section detailing the architecture of the AEs, as they follow the GAN's with little to no modification. From there, we will go over the process of developing and training the autoencoder, and the additional estimators we used to quantify the precision with which each datum's structures are recovered.

Next we will show the results of the simple autoencoder for all three data types (2D simulation images, 3D simulation projected images and 3D simulation cubes). Following this we will present several variations on our baseline AE that we developed to experiment on different aspects that we will detail. These experiments were carried out at different stages of development and thus do not have results for all three data types, and with the training sometimes happening for smaller datasets (standard cutting). We will show summary results for each of these variations and compare them to those of the baseline AE. We will give a brief mention of inconclusive experiments as well as potential future experiments we feel would be worth looking into. We will end this section with a preliminary conclusion.

In a second part, we will detail our work on the prediction of structure formation using the same model. We will first lay out the general concept of such a model and our method to train and acquire information during training. With this baseline set, we will detail the numerous methods and model modifications we came up with to simplify the autoencoder's task and obtain better results. We will then give a comparative presentation of our resulting predictions, keeping in mind that our first simple autoencoder' results will serve as a "gold standard" of sorts, or benchmark of best available results, as it represents the best predicted outcome one can expect when feeding the model all the information we expect it to output.

In a third part, we will discuss the difficulties encountered along the way and the solutions found.

Finally, we will conclude.

3.2 Autoencoders - Generalities and Specifics

An autoencoder (AE) is a neural network that learns in an unsupervised manner a representation (encoding) for a set of data. It is built and trained in the following way. A first network e, called *Encoder* takes as input a datum x and outputs a vector of reduced size $z = e(x)$. A second network d, the *Decoder*, takes as input z and outputs a recovered datum $\tilde{x} = d(e(x))$. The AE is trained by imposing that the resulting $\tilde{x}$ is as close as possible to the initial x. This is typically (but not systematically) done by using a cross-entropy loss in the case of discrete binary variable, or using the ℓ_2 loss $\|x - d(e(x))\|^2$ that can be used if the inputs are continuous.

Incidentally, we can also consider and use our previously built GANs' Gen-

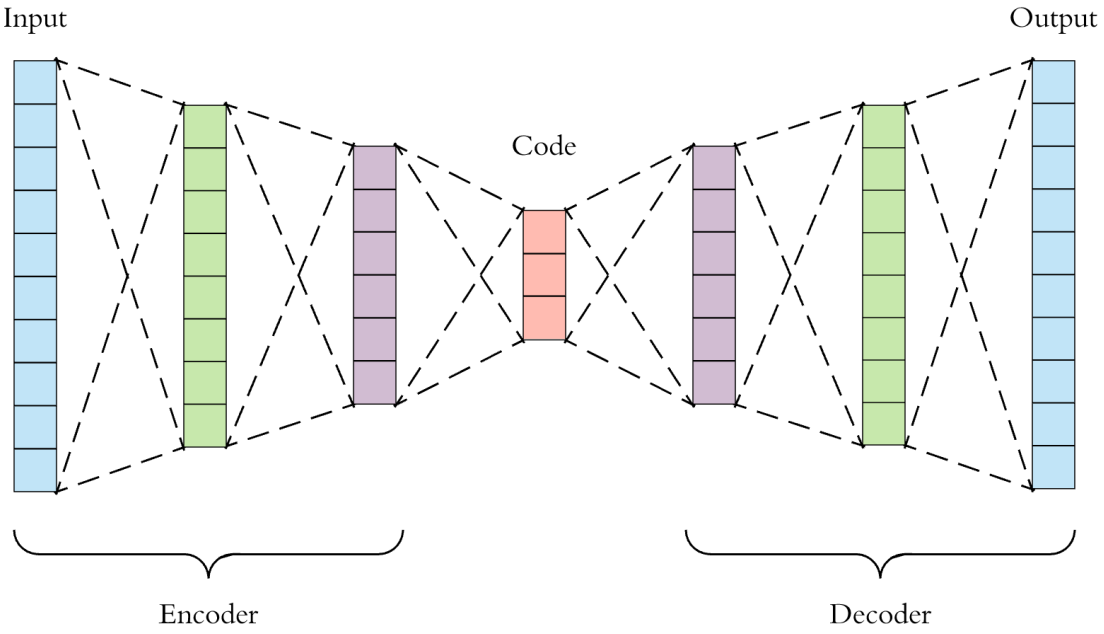

Figure 3.1: A basic autoencoder structure. The data is encrypted in a space of smaller dimension by the encoder before being decrypted by the decoder; in this state it is referred to as *latent code*.

erators as readily trained decoders from a reduced space (R^{100} for 2D and R^{200} for 3D) to our target space (typically $z = 0$ simulation images/cubes). Indeed, the Generator has learned a representation of the simulated data. Taking in an input z of reduced size, the Generator is to output any image $\tilde{x} = g(x)$ from the simulations. Furthermore, given the Generator's ability to generate images/cubes that are statistically consistent with those of the simulations, using it in the AE would constrain the outputs to share the same statistical properties. Thus, we would avoid the AEs' common pitfall to output blurry images (Dosovitskiy and Brox, 2016).

To create a functioning AE, we therefore need only build and train a functioning Encoder that will work as an inverse function of the Generator, such that $\tilde{x} = g(e(x)) = g(g^{-1}(x)) = x$.

It is worth mentioning that in a classical autoencoder, the encoder and decoder are trained alongside each other. In our approach the decoder is trained first separately in an effort to constrain it to output data that are statistically sound (i.e. that hail from the simulations' underlying distribution). These constraints might imply a loss of accuracy in our recovery of structures in individual images. It is therefore important to look at both global statistics (in terms of mean density, pixel pdf, power spectrum and peak counts, as we did for the GAN) and pair-

wise statistics, where we compare individual output/target pairs, to see how the autoencoder fares in both regards, as is done in Sec. 3.5.1.

The different layers of the Encoder are based on the Discriminator's architecture, since the latter is especially developed to extract essential information from simulated data. However since the goal of the network differs from the Discriminator's, we will only retain the architecture and not the weights. We have not tested other architectures barring slight modifications of the one mentioned; we leave this for further studies. Further details on the architectures of the networks can be found in table 3.2.

	2D AE	**3D AE**
Filter sizes	$\{5, 5, 5, 5, 5\}$	$\{4, 4, 4, 4\}$
$n_{filter}(G)$	$\{256, 128, 64, 32, 1\}$	$\{128, 64, 32, 1\}$
$n_{filter}(D)$	$\{32, 64, 128, 256, 512\}$	$\{32, 64, 128, 256\}$
Strides:	$\{2, 2, 2, 2, 2\}$	$\{2, 2, 2, 2\}$
Layer Act.	Leaky ReLU (E), ReLU (De)	Leaky ReLU (E), ReLU (De)
Final Act.	None (E), Tanh (De)	None (E), Tanh (De)
Latent dimension	100	200

Figure 3.2: Architecture specifications for each layer of the 2D (*left*) and 3D (*right*) AEs' encoder (E) and decoder (De) networks. They are all based on our previously built GANs. They are trained using the *Adam* optimizer with parameters ($lr = 0.0002, \beta_1 = 0.5$) and minimize the loss given in eq. 3.2

We can now build the AE by putting the two networks (Encoder and Decoder) end to end. This can be described as the following function: $a(x) = d(e(x))$. We fix the weights of the Decoder and update the weights of the Encoder to decrease the ℓ_2 loss function previously described: $\|x - a(x)\|^2$. However instead of using the ℓ_2 loss in the image/cube's space, comparing pixels at the same location on both "true" and inferred images/cubes, which accounts poorly for well-recovered but slightly shifted structures, we instead make use of the Discriminator[1]. It is expected that the Discriminator manages to learn a latent representation of our datasets during the GAN's training. This representation is given by the penultimate layer, where its elements are used to estimate the probability to be or not a *true* image/cube. This representation in the Discriminator's latent space is semantically meaningful (Bang et al., 2020), accounting for the presence of specific structures or shapes, and tends to put visually similar images/cubes at a small distance in this space, where they would otherwise be more distant in the images/cubes' space. Additionally,

[1]As an independent test, we build a separate "traditional" autoencoder with the same structure but in which the decoder's weights are initially randomized and are updated during training with an ℓ_2-norm loss function. As expected, we obtain the blurry results.

given the discriminator's ability to detect "fake" data, we can expect it to ensure statistical similarity between two data as well.

Therefore we define the autoencoder's loss as:

$$L_{AE} = \Delta(x, \tilde{x}) \tag{3.1}$$

where Δ is the ℓ_2 difference in the discriminator's latent space. Or, calling TD the truncated discriminator with its final layer removed:

$$L_{AE} = \|TD(x) - TD(\tilde{x}))\|^2 \tag{3.2}$$

We can then train the AE by updating its weights to minimize this loss. Training is stopped when the loss measured on a separate *validation* set reaches a minimum.

From now, on we will refer to the images/cubes $\tilde{x}$ reconstructed by the autoencoder as *inferred* images/cubes. We will assess their quality in Sec. 3.5.1.

3.3 Training process

Using the structure described above, we can now exhibit our method to train and further develop our autoencoder.

The autoencoder is trained similarly to the GAN; it is run over several epochs and we regularly save data inferred by the AE such as shown in 3.3, both from training set data and from a separate set the model has not encountered during training called *validation set*. Also saved are model weights, and model losses such as computed on three separate sets: the current training batch, a subset of the training set, and the validation set. We use batch sizes of 200 for both 2D and 3D, though for further experiments, one might try to use gradually increasing batch sizes as this has proved effective in other work (Smith et al., 2017).

During training we observe the losses and inferred data to assess the model's progress. In a successful scenario, we can expect the inferred data to become progressively more similar to data from the training set, and by extension to the validation set.

Meanwhile we expect all losses to gradually decrease (as a tendency; local noise is not uncommon or symptomatic of an issue), with the batch loss inferior to the training set loss, itself inferior the validation set loss. At some point the model might start to overfit the training set, which would lead to the validation set loss starting to rise. This is not too important an issue, as one need only retrieve model weights saved before overfitting: a method called early stopping. One should note that given the way our dataset is built, the model can take a very long time before encountering a given datum twice (i.e 20000 epochs, amounting to $>$ 100 days to

encounter every datum in the dataset once), which greatly postpones the moment at which overfitting starts to occur, if it does at all. As it happens we do not encounter it within our training time, which never exceeds a week, and usually lasts two days at most. By this time however the losses usually seem to approach their limit, requiring exponentially increasing time to decrease by a small fraction.

Once more there are several ways in which the training can go awry. If the task is too complex (e.g. the loss marginalized over the training set is a highly nonlinear function of the network parameters), the model can end up "stuck", never converging to a global minimum of the loss, but instead shifting from one local minimum to another. This is easy to detect, as visual inspection will show that both the data inferred from the training set and validation set never approach the likeness of the true data. Furthermore, the losses on all sets will look erratic rather than presenting the characteristic downward slope one expects.

Another common issue when using smaller training sets occurs when the model, while performing correctly on the training set, is not able to generalize to new data, essentially overfitting too soon for the model to be of any use. This is easy to detect when looking both at the inferred data and losses, as the data inferred from the training set will look progressively similar to the true data, while the data inferred from the validation set will look randomly generated. Looking at the losses, this would translate to a downward slope on the training set and batch losses and erratic behavior on the validation loss. This issue can only be noticed if the training set is encountered entirely multiple times by the model. Given the way we have built our training set, the above situation cannot be observed in our training time scales, and the losses on both training and validation set prove to be quite similar, with the training loss slightly inferior to the validation loss. Since no true overfitting can occur, both losses tend to slightly decrease up to an early point, after which they randomly vary around a constant.

The harder the task the more one can expect one of the above behaviors to occur, and our work focuses on diagnosing such situations and finding methods to render the task more feasible for our model.

3.4 Sørensen–Dice coefficient

Here we detail these different new estimators.

In addition to the statistical measures previously introduced in our work on GANs 2, we need a test to quantify how well the autoencoder infers individual images/cubes. Therefore, we need a pairwise comparison between input images/cubes from the simulations and their inferred counterparts from the autoencoder.

Taking a simulation/inferred pair, we test how well the structures overlap by thresholding the images/cubes at different values and counting the fraction of

pixels/voxels, above the threshold, that overlap.

For a pair of images/cubes a and b, the overlap is expressed in the following way:

$$O_{ab}(t) = \frac{N_{ab}(t)}{N_a(t) + N_b(t)} \tag{3.3}$$

Where $N_{a/b}(t)$ is the number of pixels/voxels whose value is above the threshold t in a or b and $N_{ab}(t)$ is the number of pixels/voxels whose value is above t for both a and b in a given position in an image/cube.

To get a sense of the overall quality of the encoded images/cubes, we plot the dice coefficient averaged over a set of simulated-inferred pairs: $\bar{O}(t) = \langle O_{ab}(t) \rangle_{(a,b)}$. where the notation <> indicates an average. For clarity the overlap $O_{ab}(t)$ is plotted against the top percentage, associated to a given threshold, rather than the threshold itself (Fig. 3.9 and Fig. 3.16). The thresholds are defined for the simulated and the inferred sets independently.

It must be noted that a random pair of "true" and inferred images/cubes will on average provide a non-zero overlap. Indeed, two images/cubes with $n\%$ thresholded pixels/voxels are expected to have an average overlap of $n\%$. To get a better sense of the entire inferred set's performance/recovery, we thus compute a *random* overlap, as defined by an overlap measured over a random set of simulation pairs, and plot it along with our overlap averaged on simulation/inferred pairs.

From this random overlap measure, we can proceed to build a normalized estimator of feature recovery by subtracting it from the overlap measured for simulated/inferred pairs. By itself, this difference is not informative, therefore we must look at it relative to relevant values.

First we observe it relative to its maximum possible score $1-r(t)$, $r(t)$ being the average random overlap for a given threshold t; this provides a completion score between 0 and 1, with 1 corresponding to a perfect overlap and 0 corresponding to a completely random overlap:

$$\bar{O}(t)_1 = \frac{\bar{O}(t) - r(t)}{1 - r(t)} \tag{3.4}$$

We will refer to it as the normalized sd coefficient.

Next we observe it relative to the standard deviation of the random overlap, to ensure that the inferred images/cubes, if imperfect, are well without the random range. This corresponds to a signal to noise ratio.

$$\bar{O}(t)_2 = \frac{\bar{O}(t) - r(t)}{\sigma(r(t))} \tag{3.5}$$

We will refer to it as the sd coefficient significance.

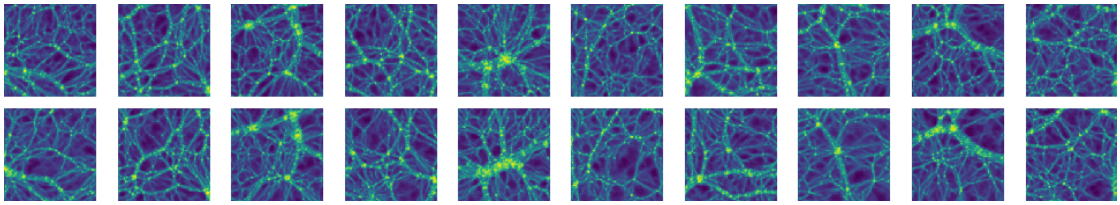

Figure 3.3: 10 images taken at random from the 2D simulations (*top row*) and their autoencoder-inferred counterparts (*bottom row*). Every image represents a 128×128 log density map of side $50Mpc$. The larger dense structures are better recovered than the finer diffuse ones.

These measures will allow us to determine whether the structures are well recovered, and at which scale they are best recovered.

3.5 Replicative Autoencoder

Here, we show and discuss the results of the replicative autoencoder, wherein we only try to replicate the input data after having reduced it to a vector of size 100 for 2D and 200 for 3D. We show the results for data generated for z=0 simulations.

3.5.1 Baseline AE Results

2D images We now focus on the outcome of the *baseline AE* for the set of 2D simulation images. For this dataset, the autoencoder is trained over 195 epochs. To determine the best point at which to stop training, we look at the evolution of our model's loss function when tested on a validation set (Fig.3.4, red); we expect it to decrease up to a point at which our model should start overfitting, that is to say becomes too fine-tuned to its training set and starts to perform poorly on new sets, after which the validation loss should start increasing. In practice, during our training of the AE on the images, we never observed an overfitting, the loss on the validation set instead closely following the loss computed on a subset of the training set, and near-monotonously converging towards a constant.

We are interested in seeing how the AE fares with images it has never encountered during its training, as our goal is to be able to apply it on new datasets. Therefore all the images shown and used to measure the different statistical properties in the results are part of, or inferred from, a separate set than the ones used for training, called a *test set*. This will be the case for both 2D images, 3D projected images and 3D cubes.

We first illustrate the results in terms of the AE's performance and recovery of features with a set of ten simulated images taken at random from the test set

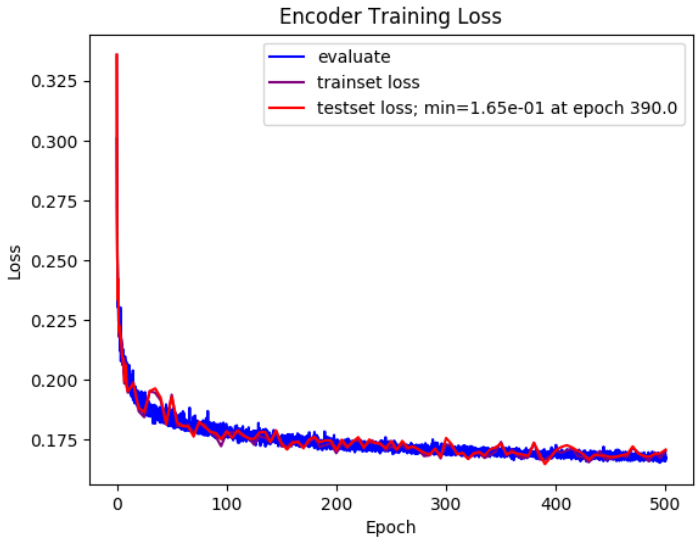

Figure 3.4: Loss evolution for the baseline 2D autoencoder. We can see that while all three losses are somewhat noisy, they all globally decrease throughout training. The test set loss is nearly identical to the training set loss but slightly greater.

(Fig. 3.3, first column from left to right) and their inferred counterparts (Fig. 3.3, second column from left to right). We note that the inferred images visually look similar to the simulated images but the larger and denser structures tend to be recovered better than the smaller diffuse structures.

We recall that while the decoder, having the exact same structure and weights as the GAN's generator, is expected to infer images that are statistically similar to the GAN's, it nevertheless infers images from a different prior. Indeed, while the GAN's inputs are selected randomly from a Gaussian distribution, the decoder's inputs are all constructed in a deliberate fashion by the AE's encoder and are not expected to follow quite the same distribution. Thus a change in statistics is not unexpected (see Fig. 3.7).

A closer inspection of the statistical properties of the images (Fig. 3.5) shows a very satisfactory agreement between the sets of inferred and simulated images. The pixel PDFs (Fig. 3.5 upper left panel) show satisfactory overlap for the two sets, presenting a very slight under-representation of high-density pixels in the inferred images. The mean particle density distributions (Fig. 3.5 upper right panel) show satisfactory agreement, with the inferred images exhibiting a slight skewness towards higher mean densities. On the lower left panel of Fig. 3.5, the power spectra overlap satisfactorily. Finally, the peak counts (Fig. 3.5 lower right panel) show a near-perfect overlap, with a slight over-representation of lower peak values

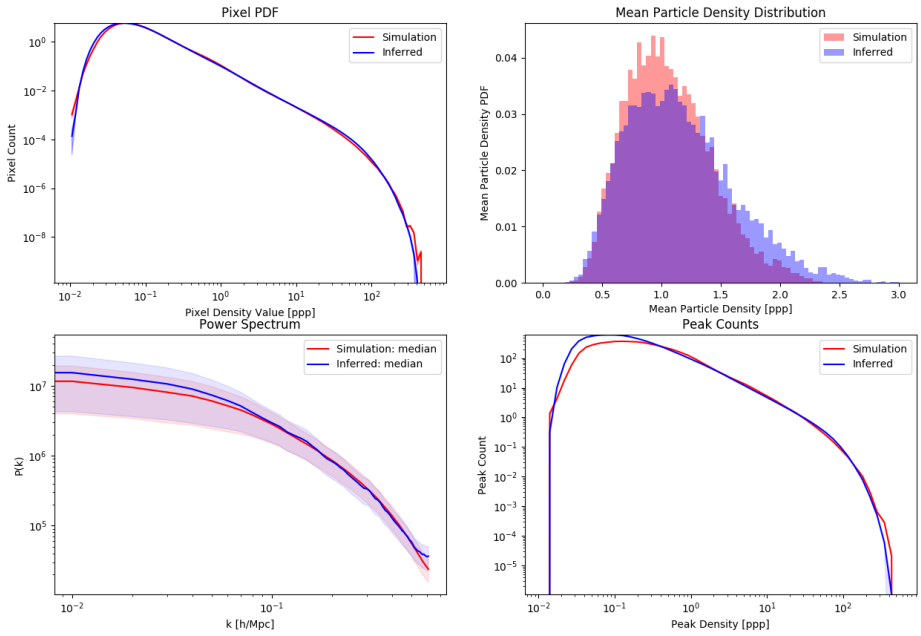

Figure 3.5: Statistics of the 2D simulation images compared to their encoded counterparts. Upper left shows the pixel PDF, upper right shows mean density distribution, lower left shows median power spectrum as well as mad (median absolute deviation) layer, and lower right shows average peak count per image. As for the GAN the curves overlap quite satisfactorily, with only the MPDD showing a slight flattening.

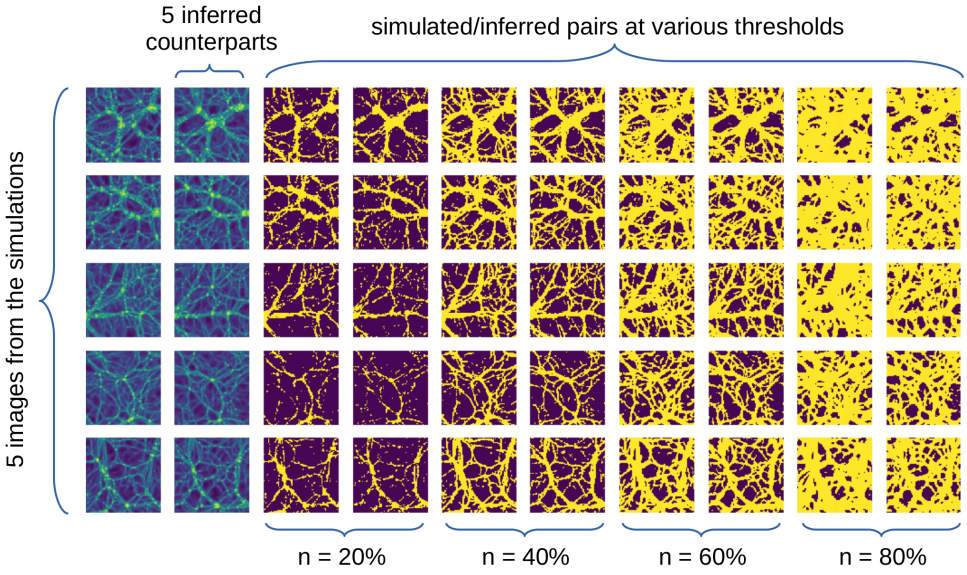

Figure 3.6: Examples of 2D simulation/inferred image pairs (columns 1-2) and their thresholded equivalents as used in the computation of the overlap function. The first column in each pair represents images from the simulations and the second their inferred counterparts. Here they are thresholded for top 20% (c. 3-4), 40% (c. 5-6), 60% (c. 7-8), and 80% (c. 9-10).

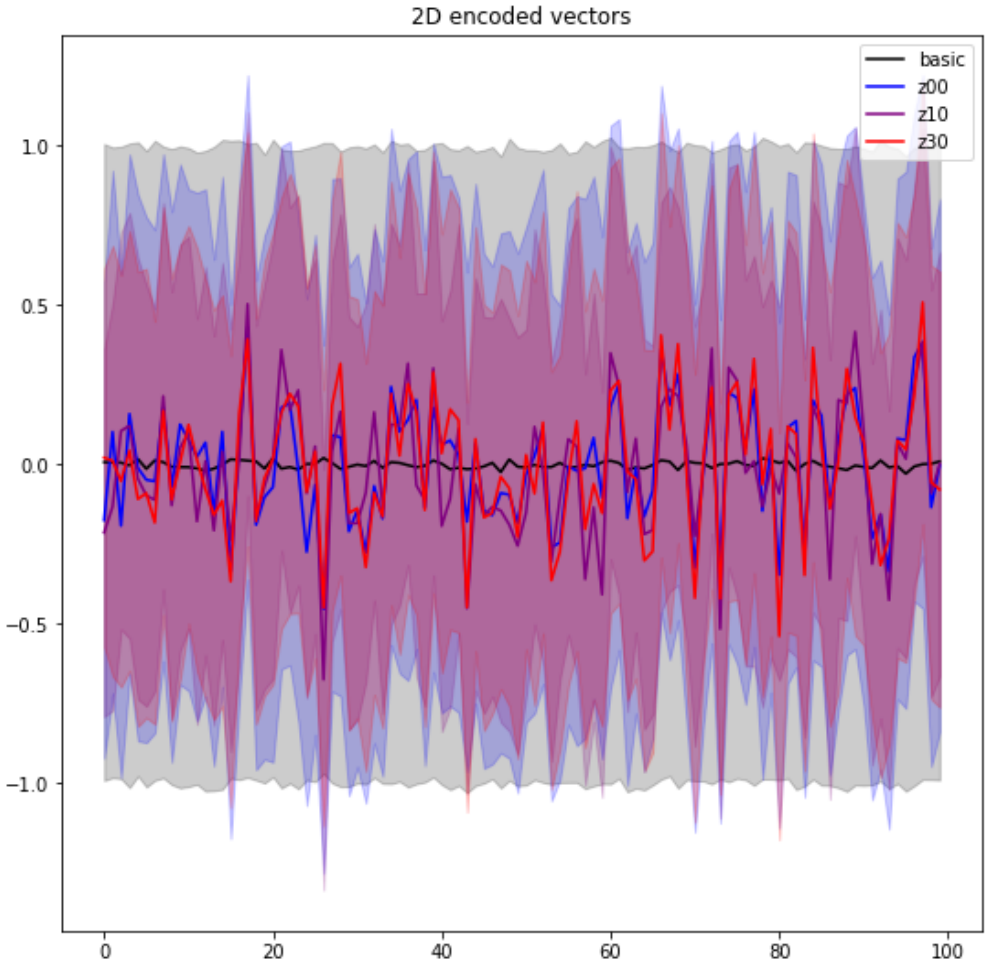

Figure 3.7: Example of latent code value distribution. Each point on the x-axis represents one of the latent code's parameters. *black* corresponds to a gaussian distribution, whereas *blue*, *purple* and *red* represent parameter distribution when training an AE to recover $z = 0$ from inputs $z = 0, 1$ and 3 respectively. The solid lines represent the mean of each parameter, and the transparent layers show standard deviation. We can see that data encoded by an AE does not follow a Gaussian distribution, explaining a difference in statistics to the GAN-generated data (which receives a Gaussian-distributed input) after being decoded.

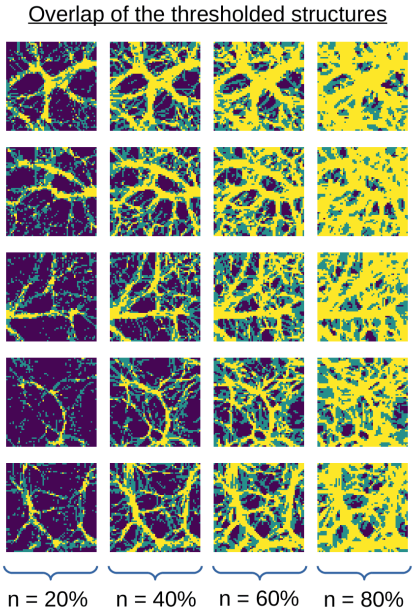

Figure 3.8: Examples of the overlap of thresholded structures for 2D simulation/inferred image pairs. Yellow pixels indicate where the structures overlap and green where they do not. The dark background represents pixels below the threshold. The dice coefficient is simply measured as $\frac{n_{\text{yellow}}}{n_{\text{yellow}}+n_{\text{green}}}$ Here they are thresholded for top 20% (c. 1), 40% (c. 2), 60% (c. 3), and 80% (c. 4).

in the inferred images. Overall, while slightly less so than the GAN-generated images, the images are recovered with almost perfect statistical quality.

We now focus on the Sørensen-Dice coefficient which computes the overlap fraction of two thresholded images. Visual inspection (Fig. 3.6 and Fig. 3.8) of the thresholded simulated/inferred image pairs and how they overlap suggests that dense structures are strikingly well recovered, with the autoencoder favoring the retrieval of thick, contrasted features rather than finer diffuse ones. Once again, this is expected given the CNN's predisposition to detect and construct well-defined shapes.

An inspection of the dice coefficient (Fig. 3.9, left) corroborates this finding. Indeed, despite some slight shifts of structures and aforementioned loss of finer structures, we note that the high density regions overlap satisfactorily and seemingly well without the random region for up to the top 60% pixels. The normalized dice coefficient (Fig. 3.9, right, blue) lets us assess the density threshold at which the autoencoder best captures structures. Here, it peaks for the top 20% pixels.

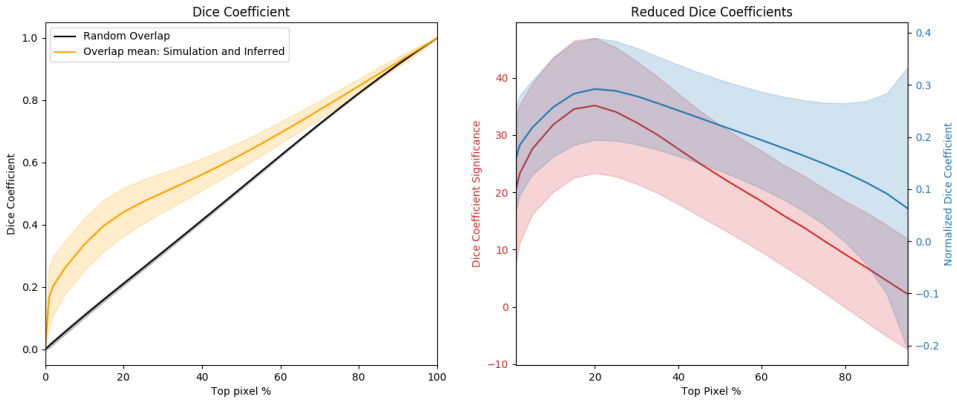

Figure 3.9: Left: dice coefficient (as defined by Eq. 3.3) of top $n\%$ pixels between 2D simulation images and their inferred counterparts, by increments of 5% (yellow). The inner dark yellow layer represents measure uncertainty and the outer yellow layer represents standard deviation over the set. Random overlap is represented in black. Right, red: dice coefficient significance (see Eq. 3.5), blue: normalized dice coefficient (see Eq. 3.4); both are represented with their standard deviation layers.

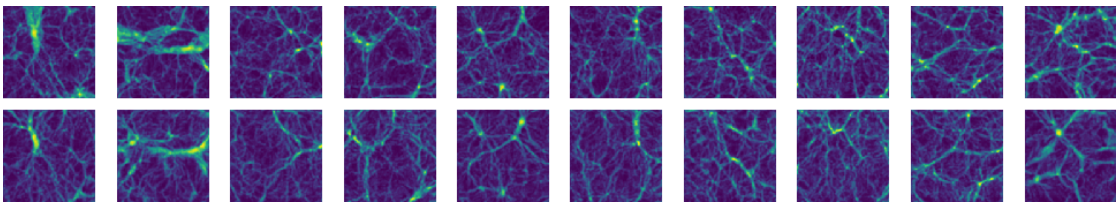

Figure 3.10: 10 images from the 3D simulations (top row) and their 10 autoencoder-inferred counterparts (bottom row). Every image represents a 128×128 log density map of side $50 Mpc$.

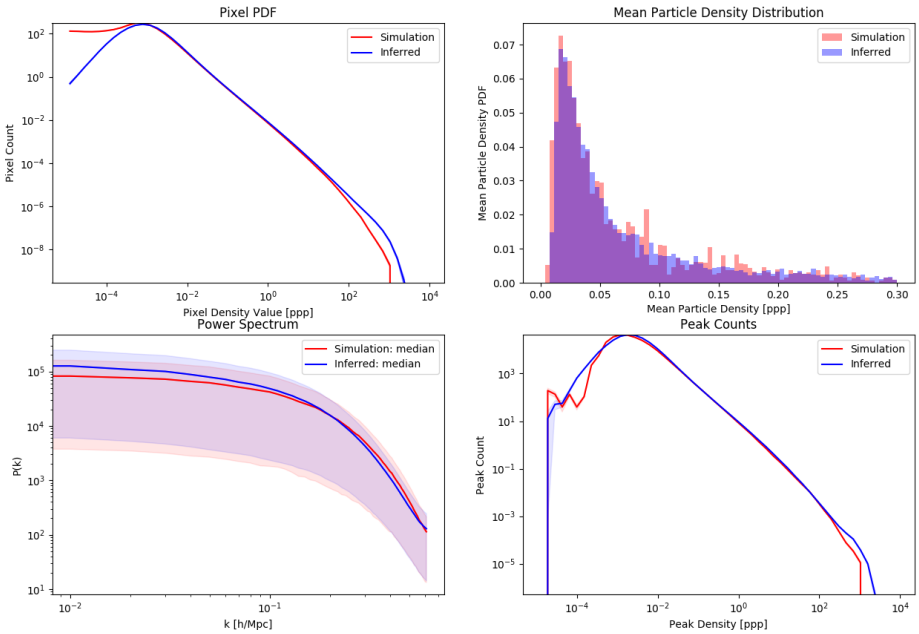

Figure 3.11: Statistics for the 3D simulation images (red) and their AE-inferred counterparts (blue). Upper left shows the pixel PDF, upper right shows mean density distribution, lower left shows median power spectrum as well as mad (median absolute deviation) layer, and lower right shows average peak count per image.

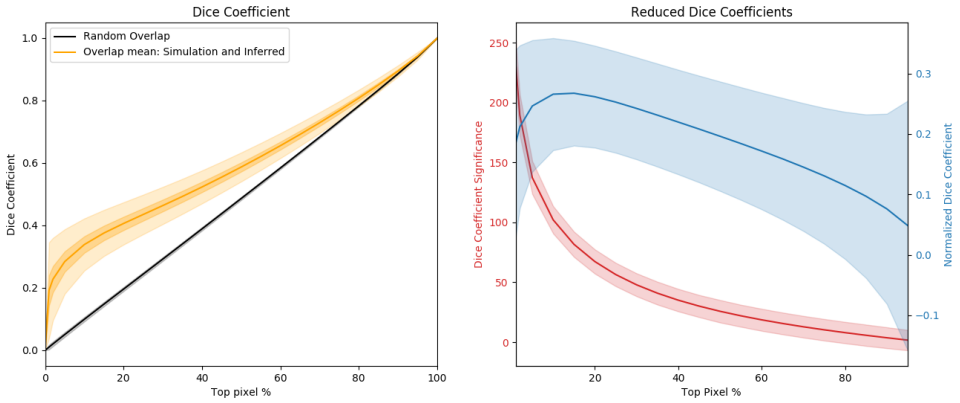

Figure 3.12: Left: dice coefficient (as defined by Eq. 3.3) of top $n\%$ pixels between 2D simulation images and their inferred counterparts, by increments of 5% (yellow). The inner dark yellow layer represents measure uncertainty and the outer yellow layer represents standard deviation over the set. Random overlap is represented in black. Right, red: dice coefficient significance (see Eq. 3.5), blue: normalized dice coefficient (see Eq. 3.4); both are represented with their standard deviation layers.

3D projected images We now turn to the AE's performance when trained on images from the 3D simulations. For these images, the autoencoder is trained over 50 epochs but the validation loss shows that overfitting starts slowly at epoch 25. Therefore, we keep the network's weights saved at that time and analyse the results for this set of weights.

We first select at random ten images from the 3D simulations test-set and their inferred counterparts (two left-most columns of Fig. 3.10). Once again, we can observe that the densest structures seem to be better recovered than the more diffuse ones. We concentrate on the images' statistical properties (Fig. 3.11) to better assess the AE's performance.

First, we see that the pixel PDFs (Fig. 3.15 upper left panel), as for the GAN case, overlap well up to the lower tail of the distribution, with an under-representation of lower densities. However in this case, the higher densities are slightly over-represented in the inferred images. The mean density distribution, as can be seen in the upper right panel of Fig. 3.11, is well recovered. The over-representation in higher densities causes a slight upward shift of the inferred images' power spectra (Fig. 3.11 lower left). As for the peak counts (Fig. 3.11 lower right panel), they show similarly to the pixel PDF a slight over-representation in the high-density regime.

An inspection of the dice coefficient (Fig. 3.12, left) shows an overall satis-

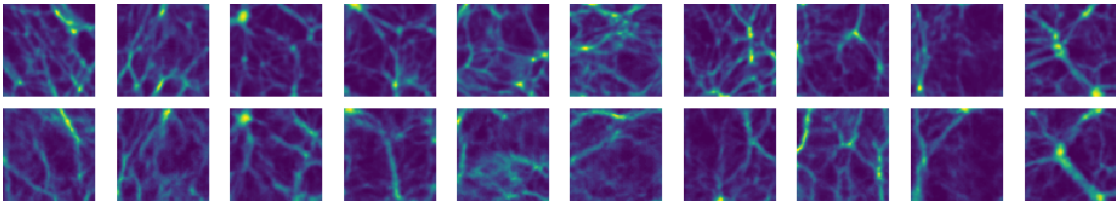

Figure 3.13: 10 cube slices from the 3D simulations (top row) and their 10 autoencoder-inferred counterparts (bottom row). Every image represents a slice from a $64 \times 64 \times 64$ log density map of side $25Mpc$.As for the 2D case the larger dense structures are better recovered than the finer diffuse ones.

factory recovery of the images which significantly differ from random images. We further note that the retrieval of high-density structures is good for the top 60% pixels in a majority of images as exhibited by the dice coefficient significance (Fig. 3.12, right, red). The normalized dice coefficient (Fig. 3.12, right, blue) suggests that structures associated with the top 10% pixels are the ones that are best captured by the autoencoder.

3D cubes We now turn to the AE's performance when trained on cubes from the 3D simulation. For these cubes, the autoencoder is trained for 75 epochs, during which all losses (Fig. 3.14) follow a noisy, but globally decreasing trend. Comparing these noisy losses to the relatively smooth loss evolution of the 2D AE, we can conclude that the task is comparatively more difficult for the model, and that adjustments of the weights for a given batch do not systematically generalize well to the entire dataset. Our validation loss (red) suggests that our model performs best at epoch 55, so we recover the weights from this epoch for the assessment described below.

To illustrate our results, we first show a random set of ten cubes from the 3D simulations test-set and their inferred counterparts (Fig. 3.13). Once again, we can observe that the densest structures seem to be better recovered than the more diffuse ones. We concentrate on the cubes' statistical properties (Fig. 3.15) to better assess the AE's performance.

First, we see that the voxel PDFs (Fig. 3.15 upper left panel), as for the GAN case, overlap well up to the lower tail of the distribution, with an over-representation of lower densities in the inferred cubes. The mean density distribution, as can be seen in the upper right panel of Fig. 3.15, is well recovered, as is the 3D power spectrum (Fig. 3.15 lower left), though once more showing a slight under-representation of higher frequencies in the inferred cubes. As for the peak counts (Fig. 3.15 lower right panel), they show similarly to the voxel PDF a slight over-representation in the high-density regime.

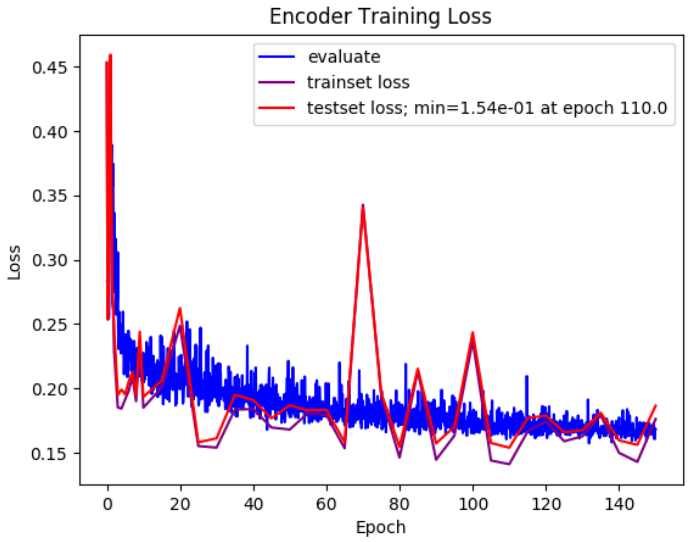

Figure 3.14: Loss evolution for the baseline 3D autoencoder. We can see that while all three losses are somewhat noisy, they all globally decrease throughout training. The test set loss is nearly identical to the training set loss but slightly greater.

An inspection of the dice coefficient (Fig. 3.16, left) shows an overall satisfactory recovery of the cubes which significantly differ from random cubes. We further note that the retrieval of high-density structures is good for the top 60% voxels in a majority of cubes as exhibited by the dice coefficient significance (Fig. 3.16, right, red). The normalized dice coefficient (Fig. 3.16, right, blue) suggests that structures associated with the top 10% voxels are the ones that are best captured by the autoencoder.

3.5.2 Variations on the baseline AE

As a way to ascertain the judiciousness of our choices in building and training our baseline AE, and also as a means to look for potential areas of improvement, we explored other training approaches that vary in terms of losses, initial weights and fixed weights. We summarize each method here and present results when relevant. Since model modifications lead to similar changes in the results regardless of the data (2D, projected 3D or 3D), we choose to only show results for the 2D data for clarity.

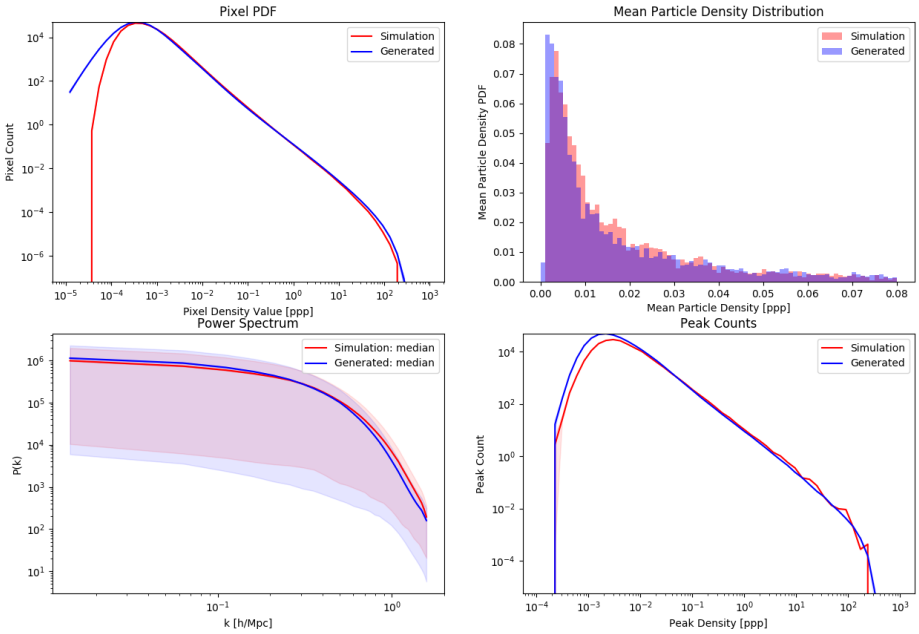

Figure 3.15: Statistics for the 3D simulation cubes (red) and their AE-inferred counterparts (blue). Upper left shows the voxel PDF, upper right shows mean density distribution, lower left shows median 3D power spectrum as well as mad (median absolute deviation) layer, and lower right shows average peak count per cube. As for the GAN, the curves overlap quite satisfactorily.

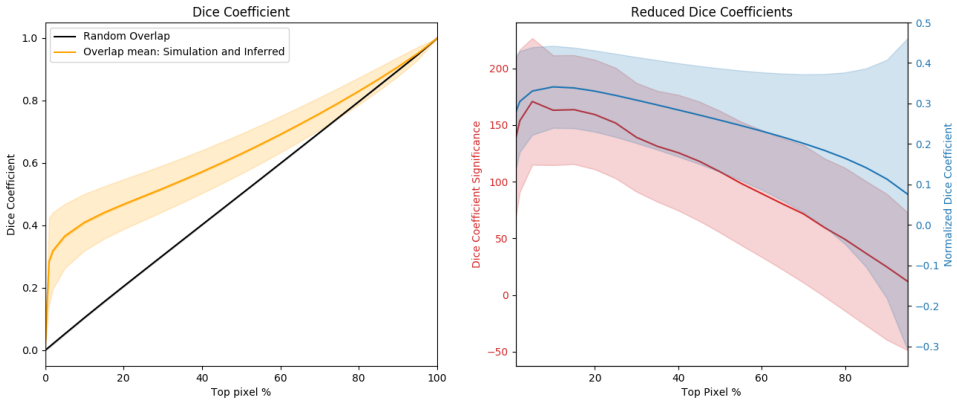

Figure 3.16: Left: dice coefficient (as defined by Eq. 3.3) of top $n\%$ voxels between 3D simulation cubes and their inferred counterparts, by increments of 5% (yellow). The inner dark yellow layer represents measure uncertainty and the outer yellow layer represents standard deviation over the set. Random overlap is represented in black. Right, red: dice coefficient significance (see Eq. 3.5), blue: normalized dice coefficient (see Eq. 3.4); both are represented with their standard deviation layers.

Traditional Autoencoder

Our baseline AE model rests upon the use of a previously trained GAN. Given that developing and training a GAN requires consequent work, it is worth exploring how a traditional AE training process fares in comparison. A traditional AE does not require any prior training, trains both encoder and decoder simultaneously, and generally uses an ℓ_2 loss to update its weights. To build our own *traditional AE* we maintain the same architecture as for the baseline AE's, but randomize all initial weights for both encoder and decoder, and leave them all free for the model to update during training. The loss used here is ℓ_2 in data space, as we do not rely on any part of the GAN, truncated discriminator included.

We find that in terms of dice coefficient (Fig.3.19), the *traditional AE* actually obtains a better score than the *baseline AE*, outputting data wherein the thickest structures are well conserved. This is not entirely surprising as it has more leeway to accomplish this task compared to our baseline structure which is also optimized to output statistically consistent data. As expected, here the output data as seen in Fig.3.17 is blurry, with finer detail completely smoothed out. It is nothing like the original data in terms of our four considered statistics (Fig.3.18), with an expected drop in higher frequencies in the power spectrum because of the lack of fine structure, and a poor overall reconstruction of density statistics.

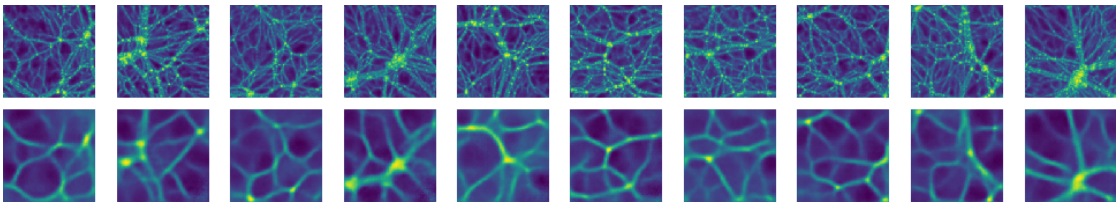

Figure 3.17: Ten images from the 2D simulations (top row) and their ten counterparts as inferred by the *traditional AE* (bottom row). The largest structures seem to be very well recovered but finer details are completely smoothed out.

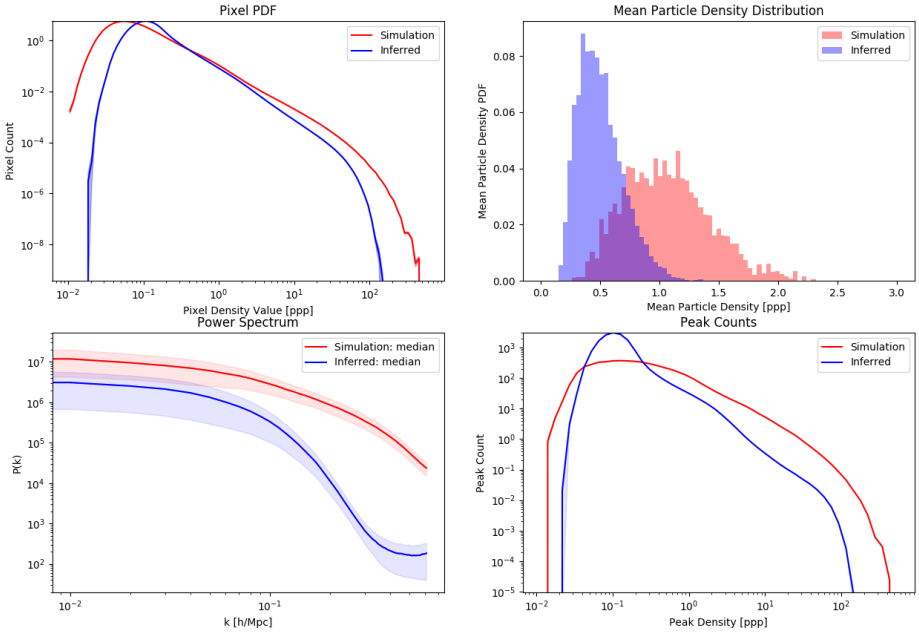

Figure 3.18: Statistics for the 2D simulation data (red) and their counterparts as inferred by the *traditional AE* (blue). As the images would suggest, the inferred data is not statistically consistent with the true data at all.

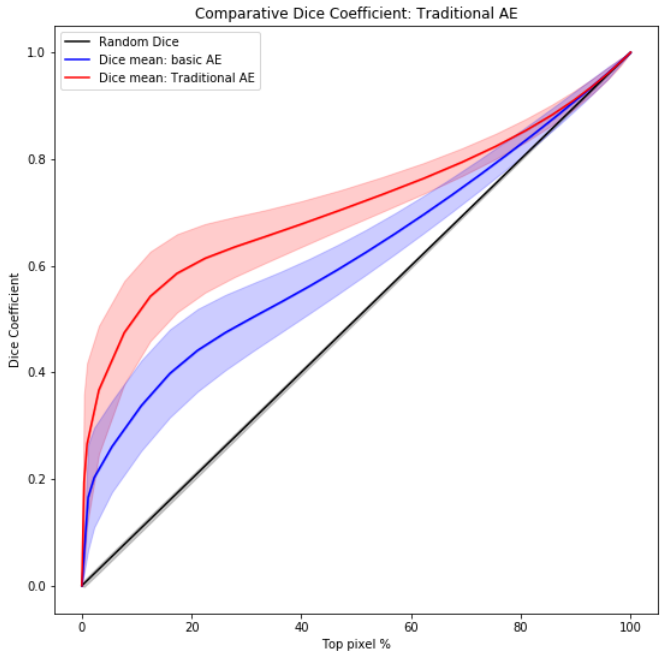

Figure 3.19: In red we represent the *traditional AE*'s dice coefficient and standard deviation. In blue we show the *baseline AE*'s dice coefficient and standard deviation. Random overlap is represented in black. We can note that the *traditional AE* performs significantly better in recovering specific structures at all scales.

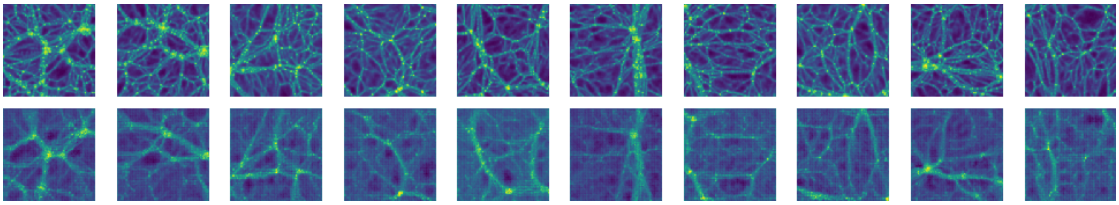

Figure 3.20: Ten images from the 2D simulations (*top row*) and their ten counterparts as inferred by the ℓ_2*-loss AE* (*bottom row*). The largest structures seem to be well recovered but the images look noisy.

ℓ_2-loss AE

In the same spirit as the *traditional AE*, we trained our *baseline AE* structure, using a trained generator as a decoder and keeping its weights fixed, but using a simple ℓ_2 loss instead of the the truncated discriminator; this helps us to determine precisely the impact of our TD loss compared to the simpler ℓ_2 one.

We find that similarly to the *traditional AE*, the ℓ_2*-loss AE*'s inferred data are statistically dissimilar to true data (Fig.3.20 and Fig.3.21), though not in the same manner because of the decoder's constrained weights. Given that on the other hand our *baseline AE* maintains statistical similarity in its inferred data, this shows that the truncated discriminator loss is actually instrumental in ensuring that inferred data are statistically consistent with true data; using only the constrained decoder is not enough. Indeed, the decoder is constrained to output statistically consistent data when intaking a Gaussian-distributed input; we can expect the latent distribution of the data encoded by the ℓ_2*-loss AE* to be far from a Gaussian, as there is no constraint for the model to do so.

In terms of overlap, the ℓ_2*-loss AE* performs better than the baseline AE (because the loss is more geared towards optimizing this statistic). However, it performs worse than the *traditional AE*. Thus we do not retain this option for further work as it is neither optimal in terms of conserving statistics nor in terms of recovering information with minimal loss.

This being said, it might be worth looking into alternating ℓ_2 and TD loss during our training. We did not yet have the occasion to try this in our work.

Latent Layer Autoencoder

Assuming that our GAN's generator g is a function that perfectly maps a 100- or 200-dimensional Gaussian to the space of simulation-like data, we develop our decoder d to make it as close as possible to g^{-1} so that our autoencoder outputs data x' that is as close as possible to their input x. In this manner we can train the encoder independently without needing to append the decoder to it by using

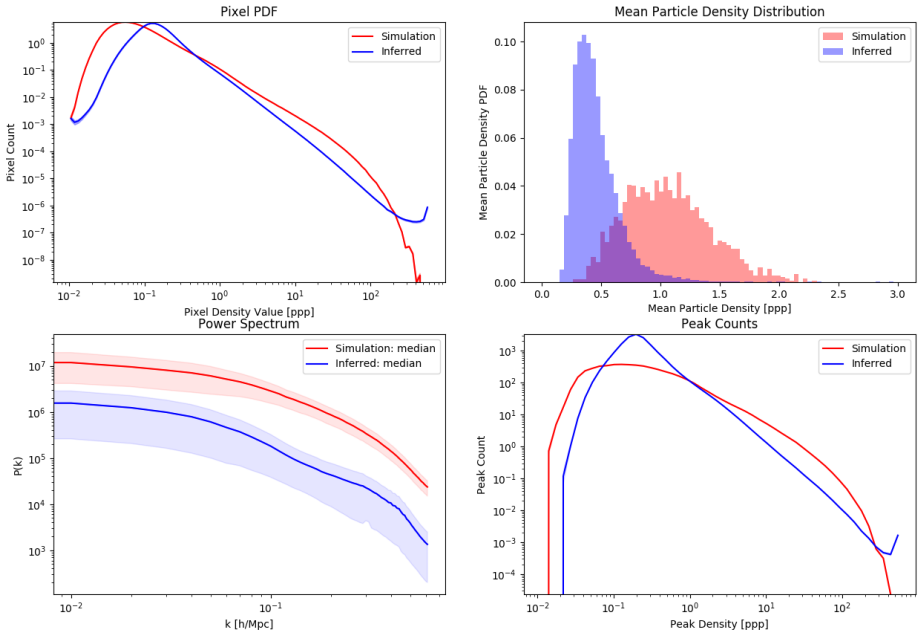

Figure 3.21: Statistics for the 2D simulation data (red) and their counterparts as inferred by the ℓ_2-*loss AE* (blue). As the images would suggest, the inferred data is not statistically consistent with the true data at all.

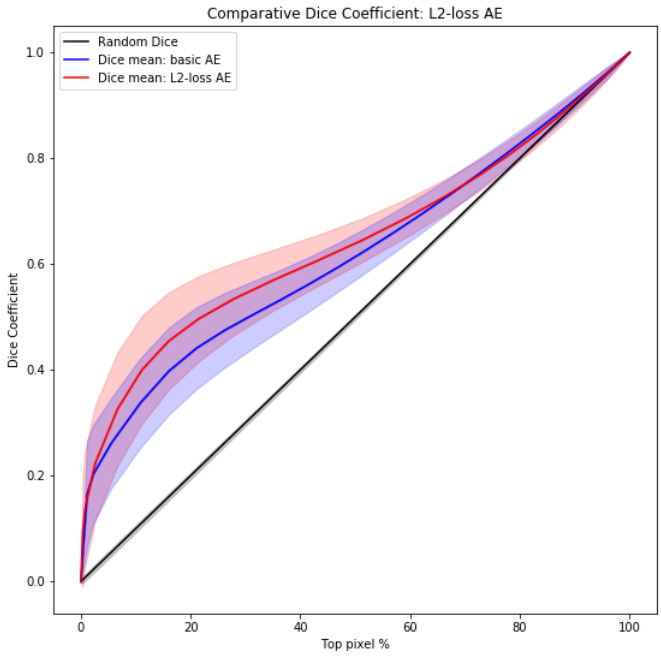

Figure 3.22: In red we represent the ℓ_2-*loss AE*'s dice coefficient and standard dev. In blue we show the baseline AE's dice coeff and standard dev. Random overlap is represented in black. We can note that the ℓ_2-*loss AE* performs better than our baseline model in recovering specific structures at most scales.

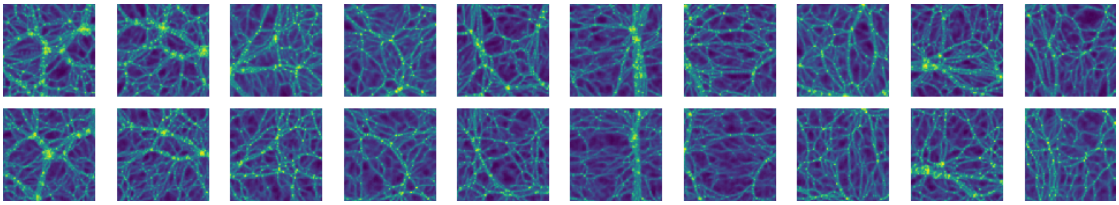

Figure 3.23: Ten images from the 2D simulations (top row) and their ten counterparts as inferred by the *latent layer AE* (bottom row). The largest structures seem to be well recovered and the data statistically consistent.

GAN-generated data. Indeed, using our generator, we can obtain precise x and $g^{-1}(x)$ pairs. We simply choose random GAN inputs y to act as $g^{-1}(x)$ and, from them, generate data that act as the associated x. The loss is then computed as:

$$L_{AE} = \|y - e(g(y))\|^2 \tag{3.6}$$

We train our encoder on GAN-generated data following this method, but we are eventually interested in how it fares on true data. We thus reintegrate it within the autoencoder structure (by appending the decoder at its end) and observe the results on a test set as for the baseline AE.

We find that this method under-performs both in terms of general and one-to-one statistics compared to our baseline model, but not significantly enough to be discarded. We thus find it worth keeping it in our following predictive work.

We additionally test the *latent layer AE* on a GAN-generated set to see how the results compare. We find that the AE trained in this manner performs near-perfectly on GAN-generated data, suggesting that despite the near-perfect similarity between GAN-generated data and true data on our considered stats (pixel PDF, mean density distribution, peak counts and power spectrum), there remains a fundamental difference that allows the AE thus trained to perform relatively poorly on the simulated data and exceptionally on the generated data.

3.5.3 Conclusion on the Replicative Autoencoder

Building on the GAN's properties, we utilised the trained network to devise an AE that used the generator with fixed weights as a decoder and the truncated discriminator as a way to compute the loss; this was referred to as the *baseline AE*. It compressed images or cubes, and related information, to encoded vectors of smaller size that were then decoded with as little loss of information as possible, while satisfactorily conserving the statistical properties of the data.

A visual appraisal and dice coefficient measure of the data inferred by the AE thus constructed, suggested that large dense structures were satisfactorily repro-

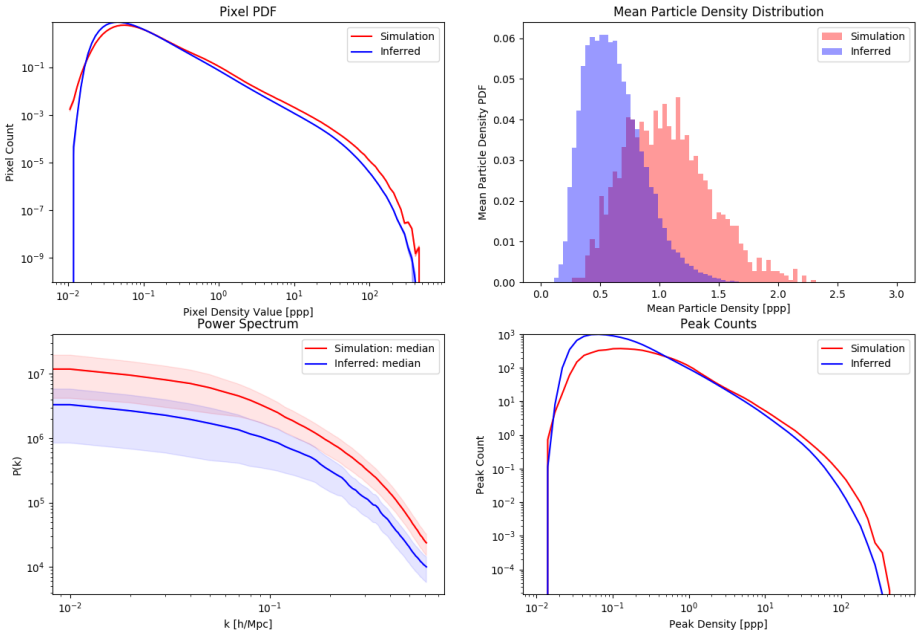

Figure 3.24: Statistics for the 2D simulation data (red) and their counterparts as inferred by the *latent layer AE* (blue). The slight under-representation of higher densities in the pixel PDF seems to lead to a shift in both MPDD and PS.

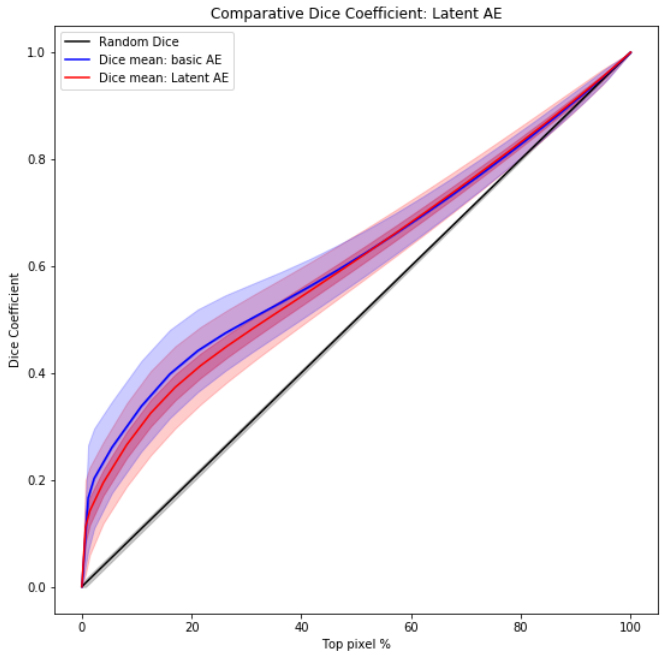

Figure 3.25: In red we represent the *latent layer AE*'s dice coefficient and standard dev. In blue we show the baseline AE's dice coeff and standard dev. Random overlap is represented in black.The *latent layer AE* seems to perform slightly worse, but the difference is not significant enough to conclude; differences could be due to training time.

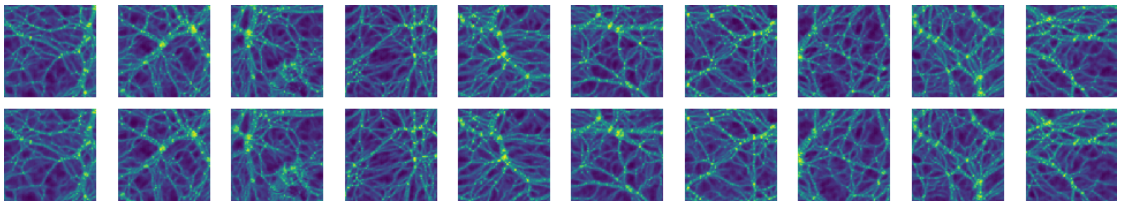

Figure 3.26: Ten images generated by the GAN (top row) and their ten counterparts as inferred by the latent layer AE (bottom row). The structures seem to be very well recovered at nearly all scales.

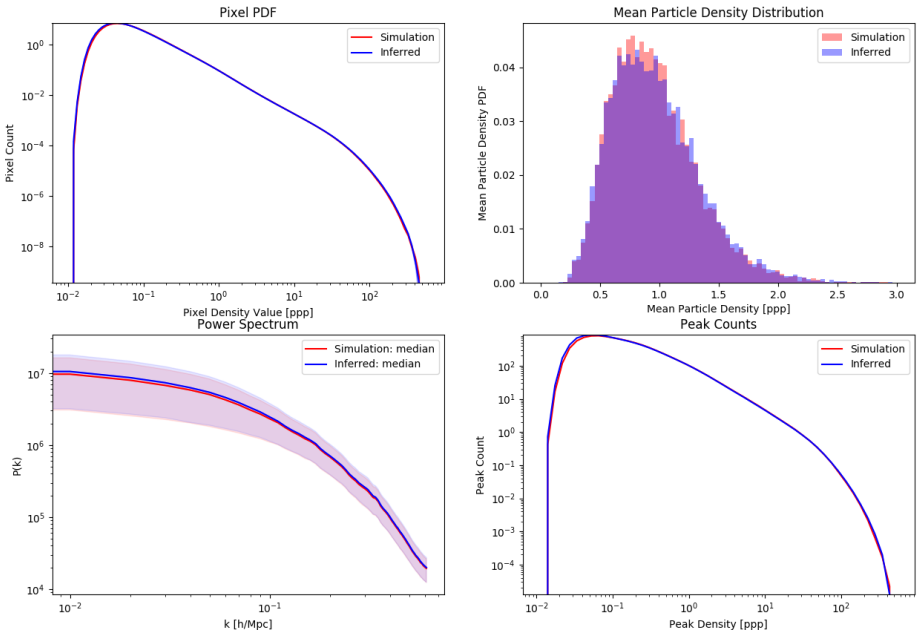

Figure 3.27: Statistics for the 2D generated data (red) and their counterparts as inferred by the *latent layer AE* (blue). All curves overlap perfectly.

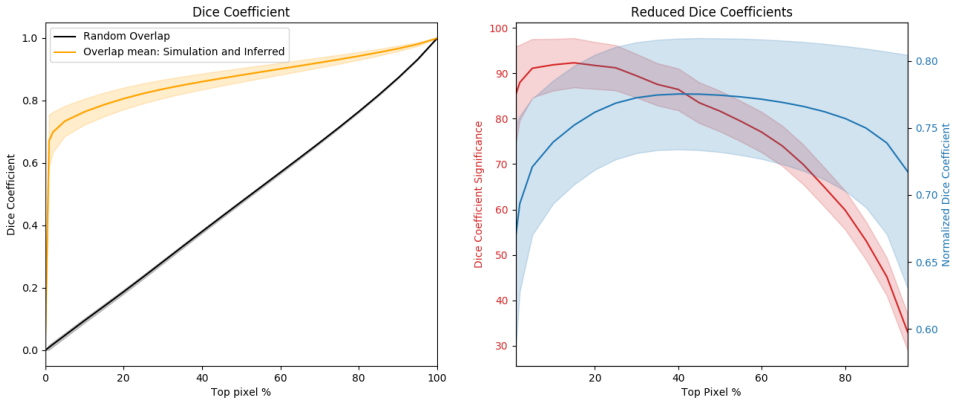

Figure 3.28: In red we represent the *latent layer AE*'s dice coefficient and standard deviation as computed on data generated by a GAN. In blue we show the baseline AE's dice coefficient and standard deviation. Random overlap is represented in black. The *latent layer AE* performs exceptionally well on the generated data.

duced for all data types (images, projected images and cubes), while for the smaller fine structures, the degree of reproduction was not satisfactory at all. Notably, this recovery of structure within individual data was much less efficient than that of a *traditional AE* built with an ℓ_2 loss and free decoder weights; it even fared slightly worse than a baseline AE that was trained with an ℓ_2 loss (ℓ_2*-loss AE*).

However, the baseline AE's inferred data appeared to be visually realistic, unlike the inferred data of the traditional AE and the ℓ_2-loss AE. Statistical estimators showed a quite satisfactory overlap between the baseline AE's inferred data and true data's stats, suggesting only a slight overall decrease in statistical similarity with the original sets when compared with the results from the GAN. On the contrary, the other two showed a considerable drop in statistical resemblance . This was expected for the traditional AE, as it had no incentive to maintain original statistics, besides a general task to infer data individually similar to original data. This came more as a surprise for the ℓ_2-loss AE, as we originally assumed that data output by a GAN's decoder would naturally have the right statistics. This assumption does not account for the prior distribution; indeed, the generator is constrained to output statistically consistent data, but only when receiving a Gaussian-distributed vector as input. The ℓ_2-loss AE puts no such constraint on the encoded data, resulting in the poor statistics we can observe. This means that in the case of our baseline AE, it is in fact the truncated discriminator loss that is imposing statistical quality of the output data, and thus constraining the encoded data to somewhat follow a Gaussian distribution.

This means that our TD loss is quite useful as it constrains both precise individual structure recovery and statistical recovery. However, this statistical constraint is so harsh that it does not allow training from a random position to converge; inferred data must be statistically similar to the target data from the beginning of training.

Circling back to the latent encoding space, we attempted to train an encoder on GAN-generated data, tasking it with yielding a given generated datum's prior input vector. Reattaching it to our baseline decoder, we found that it performed exceptionally well on generated data, recovering precise structure and statistics, but performed significantly more poorly on true simulation data. This suggests that while all our statistics show striking similarity between GAN-generated data and true simulation data, there are still core differences between the two that account for the vast differences in the results of the latent layer AE. However, we can note that at least visually, the baseline AE and by extension the generator's GAN is able to generate a datum that strikingly approaches almost any new datum's likeness, and is not limited to reproducing data that it has already encountered.

For future perspectives, given that AEs trained with ℓ_2 losses yield better results in terms of dice coefficient, we can look forward to possibly better overall

results if we were to try alternating TD and ℓ_2 losses. Additionally, ℓ_1 losses might be worth looking into. Furthermore, we understand that the limitations of the baseline AE are at least partially due to the fact that the decoder's weights are locked and constrained. Hence it might be worth unlocking them after a preliminary training round to see if it can be made even more efficient while retaining the advantages of the baseline AE.

3.6 Predictive Autoencoder

In a second part of our work, we aimed to make use of our AE's ability to extract meaningful information to perform more complex tasks, and specifically see whether we could predict, or conversely "rewind", the evolution of density distributions over time in Lagrangian space using our AE structure. We refer to this predictive AE as *timewarper* (TW). This could eventually prove useful in two main ways, first as an alternative to other simple approximations such as first and second order Lagrangian Perturbation Theory(Buchert, 1995; Zel'Dovich, 1970), or ML-based approach for small simulations (He et al., 2019), to get a rough estimation of density fields at a given time given initial conditions.

Second, rewinding would permit to recover initial conditions from a current density distribution.

While we more recently began attempts to rewind from lower to higher redshifts (such as recovering redshift $z = 3$ from input redshift $z = 0$), most of our current results are for forward evolution to target redshift z=0 from input redshifts 3 to 0.5 by increments of 0.5. We can also expect prediction to be more difficult the farther apart the redshifts are, and will thus compare our results for each z.

Similarly to the previous section, we will first present in detail our results on a basic "baseline" structure, then detail a set of variations on our training method that we put in place in order to obtain better results.

We conclude by reminding the reader that our approach consists of training the neural networks for a duration that usually does not exceed two days, due to time allocation constraints. We regularly save network weights and recover those that yield the best validation loss. Given the limited training time, the best weights often correspond to the latest ones saved, suggesting that convergence has not quite been reached. Thus, we find that our networks could often benefit from longer training, though we only expect slight improvement of results in terms of structure recovery (statistical and one-to-one). Hence, though imperfect, the results obtained on rather short training times give us a good indication of the learning capacity of our various networks. We recap the full training time (in terms of epochs) of our different networks in Table3.29, as well as the epochs at which we recover the best weights for each network.

Name	Training Time
Baseline 2D	50
Baseline 3D	150
Curriculum	50
Multiple Input	50
Velocities	70

Figure 3.29: Training times (in epochs) for each of the TW presented.

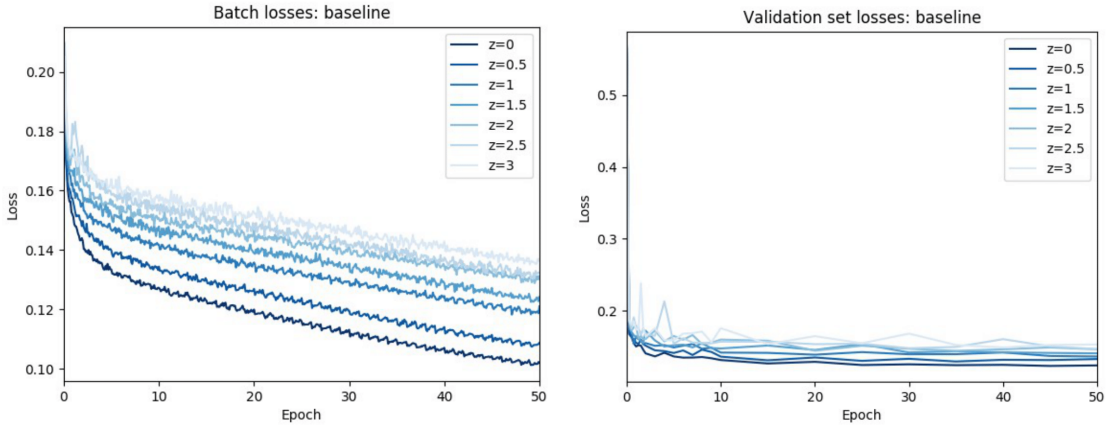

Figure 3.30: Batch and validation set losses for the *baseline TW* trained to predict $z = 0$ from 2D input data at various redshifts.

3.6.1 Baseline Timewarper results

Our first timewarper is simply based on our *baseline AE* (3.2) in terms of structure, loss, and training sequence. We input a datum at a given redshift $z > 0$ and constrain the TW to output the equivalent $z = 0$ datum (with two "equivalent" data referring to the same region within a snapshot at the two redshifts considered for input and target).

2D images

We now focus on the outcome of the TW trained to recover data at redshift $z = 0$ for the set of 2D simulation images, from input redshifts varying from $z = 0$ to $z = 3$ by steps of $\Delta z = 0.5$.

One might note that "predictions" from $z = 0$ to $z = 0$ are simple replicative autoencoding of the kind covered in the section above.

We compare our results on predictions from higher redshifts to this $z = 0$ reference. The task is expected to be harder the higher the input z; indeed, given

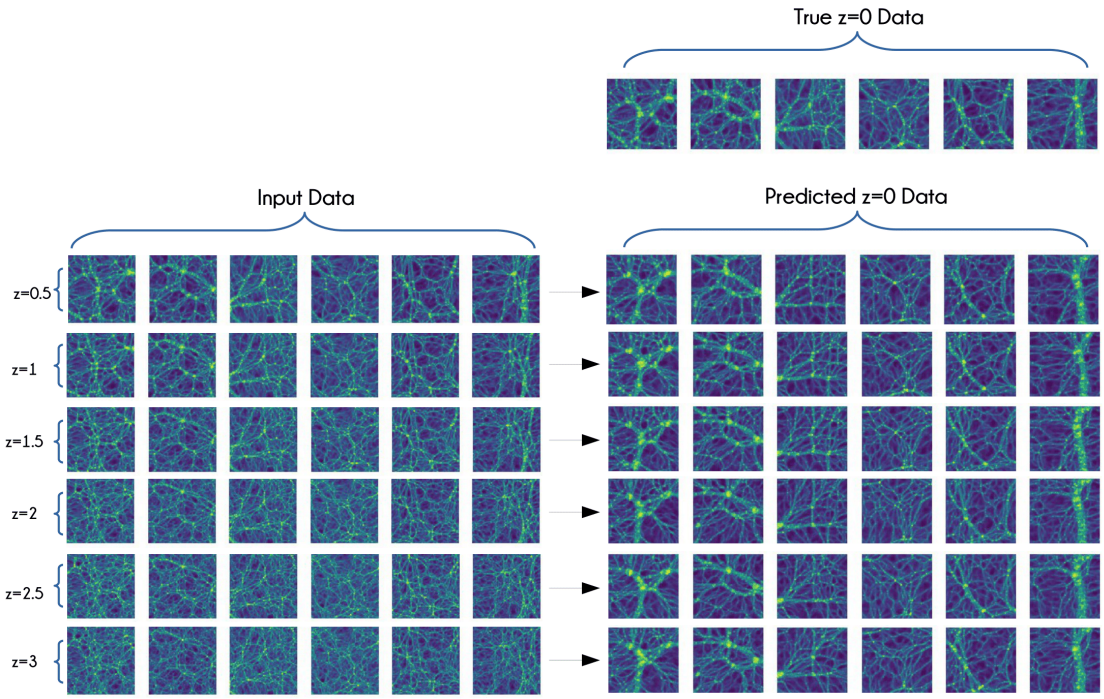

Figure 3.31: Six images from the 2D simulations at various redshifts (*left*), and their equivalent predictions of redshift $z = 0$(*right*) as inferred by the *baseline TW*. The true $z = 0$ simulation images are shown above the predicted images (*upper right*) for comparison.

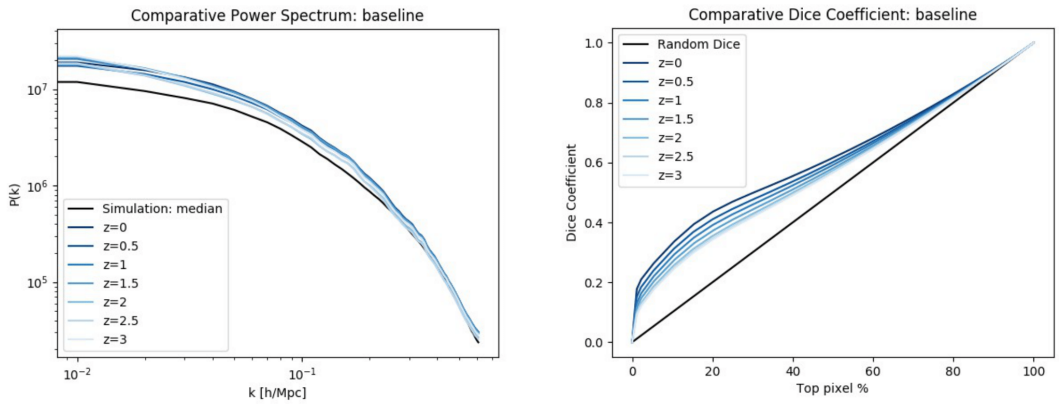

Figure 3.32: *Left:* Median power spectra of the data predicted by the *baseline TW* from various redshifts (*blue*). The median power spectrum of the true $z = 0$ simulation data is shown in *black*. *Right:* Average dice coefficient between target $z = 0$ 2D simulation data and data as predicted by the *baseline TW* from various redshifts.

that the structuration of matter is a highly non-linear process, we can expect that the farther away in time the target is from the output, the farther the density field will steer from a linear evolution.

For all input $z > 0$, the *baseline TW* is run for 75 epochs, and the best weights are recovered for each input z (0.5, 1, 1.5, 2, 2.5, 3), respectively at epochs 50, 75, 75, 50, 75 and 60. For both sets of losses, we can note that the higher the input z, the higher the overall loss is during training. This is coherent with the expected higher difficulty of the prediction task for higher z.

We illustrate the *baseline TW*'s performance in recovering $z = 0$ from different redshifts with a set of six simulation images taken at random from the test set (Fig. 3.31). These data are taken at various redshifts (left block) and used as input for the trained TWs to predict their $z = 0$ equivalent (upper right). Predicted data are shown in the right block. We find that whatever the input z, the TWs are very successful in recovering $z = 0$, with larger/denser structures being globally well recovered and finer detail being more random, as was already the case for the replicative AE. It is difficult to distinguish by eye whether performance changes according to the input z, with structures being well predicted even when inferring from high-z.

Inspecting the power spectrum of the predicted data (fig. 3.32, *left*), we find that the TWs seem to recover similar statistics regardless of the input z; all predicted spectra show a satisfactory shape, but a similar upward shift at lower frequencies compared to the true data. This suggests that denser regions are recovered with slightly excessive density.

Finally, we observe the Dice coefficient(fig. 3.32, *right*); while predictions from all redshifts show similar results to the baseline AE's, we can see quite clearly that for every increment of input z, the obtained dice coefficient is slightly lower than the previous at all pixel thresholds, exhibiting the increased difficulty of the predictive task with an increased input z.

From this first test of the TW on 2D data, we can conclude that our method is sound and that, in this simple case at least, our networks can predict structure evolution quite satisfactorily in the time spans considered. Unsurprisingly, predictions are more precise the closer in time the input is to the target, and we obtain best results for $z = 0$ to $z = 0$ encoding.

3D cubes

We now focus on the outcome of the TW trained to recover data at redshift $z = 0$ for the set of 3D simulation cubes, from input redshifts varying from $z = 0$ to $z = 3$ by steps of $\delta z = 1$.

For predictions from redshift $z = 0$, we once more retain the results of the *baseline AE*, and compare our results on predictions from higher redshifts to this

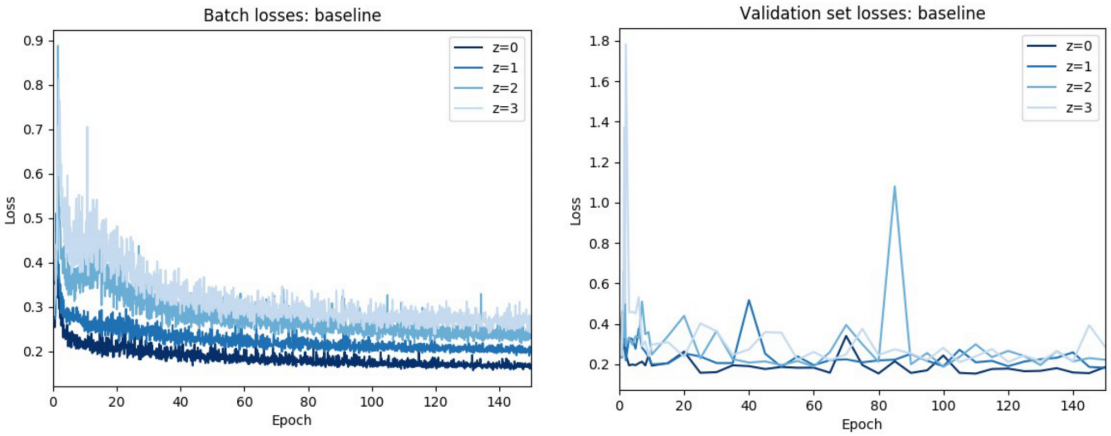

Figure 3.33: Batch and validation set losses for the *baseline TW* trained to predict $z = 0$ from 3D input data at various redshifts.

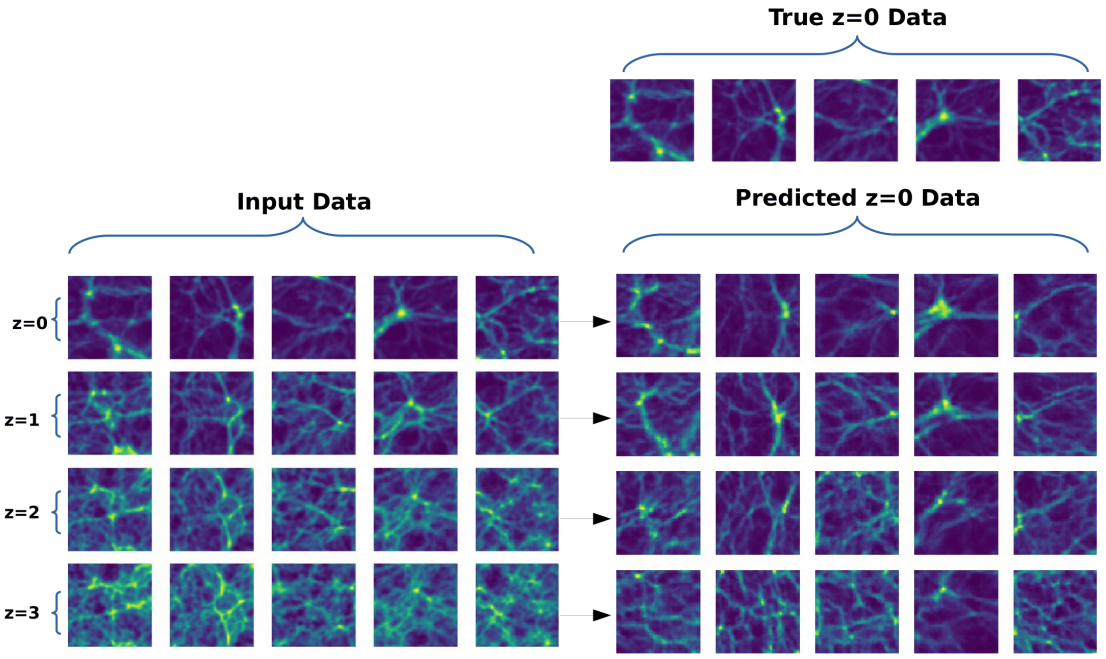

Figure 3.34: Five images from the 3D simulations at various redshifts (*left*), and their equivalent predictions of redshift $z = 0$(*right*) as inferred by the *baseline TW*. The true $z = 0$ simulation images are shown above the predicted images (*upper right*) for comparison.

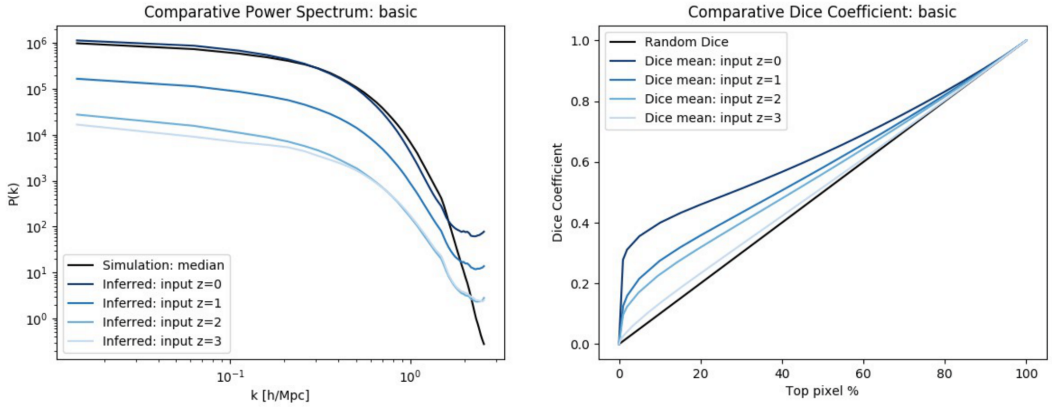

Figure 3.35: *Left:* Median power spectra of the data predicted by the *baseline TW* from various redshifts (*blue*). The median power spectrum of the true $z = 0$ simulation data is shown in *black*. *Right:* Average dice coefficient between target $z = 0$ 3D simulation data and data as predicted by the *baseline TW* from various redshifts.

reference.

For all input $z > 0$, the *baseline TW* is run for 150 epochs, and the best weights are recovered for each input z (1,2,3), respectively at epochs 150, 100 and 150. Here both sets of losses (Fig. 3.33, batch (*left*) and validation (*right*) losses) seem to have two distinct regimes in their evolution; a first, that happens within the first epoch, where the loss decreases dramatically, and a second where it shows a slow and noisy decrease.

Validation losses appear to follow the training losses, albeit with significantly more noise, which seems to increase with the input z and decrease with training time, suggesting that the network is progressively converging towards stable weights.

Here, similarly to the 2D case, lower input z leads to globally lower losses, once more exhibiting the incremental difficulty of predicting $z = 0$ from progressively greater z.

We now observe six simulation cubes taken at random from the test set (Fig. 3.34). These data are taken at various redshifts (*left block*) and used as input for the trained TWs to predict their $z = 0$ equivalent (*upper right*). Predicted data are shown in the right block. Here, we find that contrary to the 2D case, the predicted data becomes notably more random with the increase of the input redshift, with data predicted from $z = 2$ and $z = 3$ showing very little similarity to the target $z = 0$ data. We can note that with minimal input information, the network tends to default to outputting data that shows few to no dense structures, opting for more diffuse structure that can blend in with any target data's background, thus

reducing on average the difference between prediction and any random target.

Thus it is no surprise to find that the power spectra (see Fig.3.35, *left*) of the predicted data become lower for high z inputs, since the lack of dense structures leads to overall lower density and a loss of signal at all frequencies. However in terms of shape, the power spectrum seems to be well recovered.

Finally, we examine the Dice coefficient (Fig.3.35, *rightt*); once more the increased disparity between prediction and target with higher input z is made clear, with overall dice becoming significantly lower for input $z = 1$ compared to $z = 0$ and for inputs $z = 2$ and 3 compared to $z = 1$.

Overall we can observe that the networks perform much more poorly on our 3D data compared to 2D. There could be several reasons for this; a first one being that redshifts in 2D and 3D cannot be considered as strictly equivalent to each other, as large scale structure can be expected to form at a different pace depending on the dimension of the simulation.

Secondly, given that our 3D data and associated models are larger in terms of number of parameters, this naturally leaves more place for failure, with more information that needs to be recovered on one hand and possibly too large models to train on the other. We can add that we may need many more neurons to encode the richer information in 3D fields compared to 2D.

Finally, there is the matter of whether the information provided by the input sufficiently constrains the output. Indeed, while simple $z = 0$ to $z = 0$ encoding provides the network with all the information it needs to recreate its target output, supplying the networks with higher z density-field-only inputs to predict the $z = 0$ density field brings forth an issue: two identical density fields at a given redshift can morph into a variety of density fields at a later redshift, provided that their associated velocity field or neighboring environment changes. Given that this leads to not one but a manifold of possible futures, the best we can expect from the network is to produce an average future density field, marginalized over the unknown data. In this scenario, how "well" we will judge the network to perform (via overlap of structures with the dice coefficient, in our case), will have to do with the variance of the density field over the unknown information. Thus, the more the input constrains the output, the smaller the variance, the higher the agreement between prediction and true target. Comparing the 2D and 3D data, we can determine why the former may be more constraining than the latter, given that in 3D matter can move in an additional direction, and a given cube will interact with environment from six directions instead of four. Additionally, our cubes are twice as small in scale as our 2D images, and thus even more affected by unseen neighboring environment. Therefore it is unsurprising that our networks perform more poorly on 3D data.

Having determined these potential issues that can affect network performance,

we explore several variations on our baseline model that can mitigate their effect.

3.6.2 Variations on the baseline Timewarper

Although the 2D timewarper performed remarkably well in predicting density evolution regardless of the input data's redshift, we could note that on the other hand, results were much more mitigated for the 3D data, with predictions for $z = 3$ data being nearly random. To tackle this problem, we explored a set of possible improvements.

Though we did attempt to apply our improved models on the 2D data, results showed no significant improvement if any, and thus we choose to show our results for the 3D data only, as they are the most compelling.

Because of time limitations, all following models were only trained for 50 epochs; thus results, though generally satisfactory, might not be entirely conclusive, as they would likely benefit from longer training.

All results hereafter (except for predicted cubes visuals) will be shown in conjunction with that of the *baseline TW* for easy comparison.

Curriculum learning To correctly train a network, one must provide it with a task that it can progressively improve upon. However, oftentimes a given task can present too steep a learning curve for the model to efficiently learn, even if it technically has the capacity to complete the aforementioned task.

Knowing this, we can make the progress easier if we can gradually increment the difficulty of the task instead of giving a difficult task to the model from the start.

Curriculum learning (Bengio et al., 2009) is a training approach wherein a network is trained following a *curriculum*: rather than being trained with a large dataset made up of randomly ordered data, the dataset is instead split into subsets following a meaningful order where progressively more complex concepts are presented. How to define this order and determine the time spent training on every subset is a complex issue (Hacohen and Weinshall, 2019; Graves et al., 2017), but when done aptly leads to much more efficient training.

It so happens, as we have seen in the previous results, that we have a set of incrementally difficult tasks in the form of predicting data at $z = 0$ from progressively higher redshifts.

Thus we attempted to train a model progressively, training it first on $z = 0$ input data for 50 epochs, then incrementing the input redshift by 1 every 50 epochs.

Examining both batch (*left*) and validation (*right*) losses in 3.36, we can first see that those of the *curriculum* TW are globally lower than that of the *baseline TW* for all input z. This suggests that the knowledge accumulated by a network

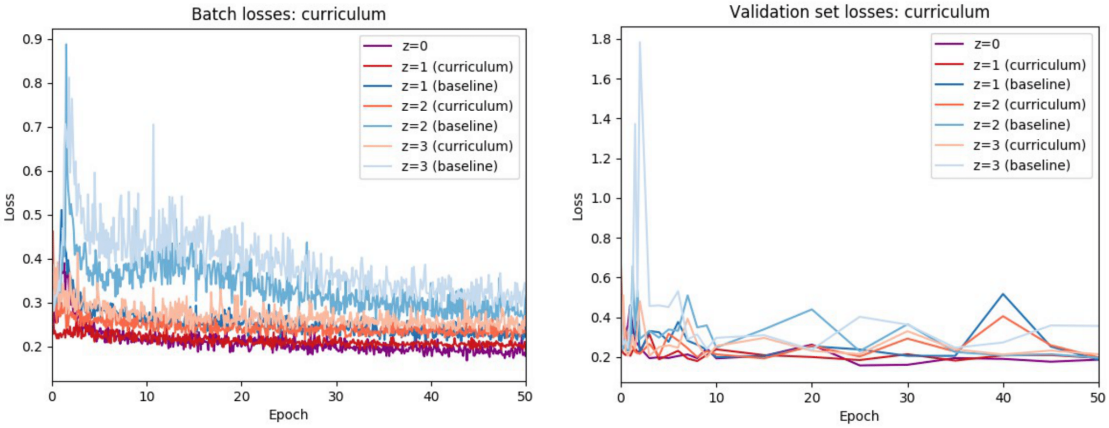

Figure 3.36: Batch and validation set losses for the *curriculum TW* (*red*) shown next to those of the *baseline TW* (*blue*) for training at various input redshifts. As $z = 0$ is common to both it is shown in *purple*.

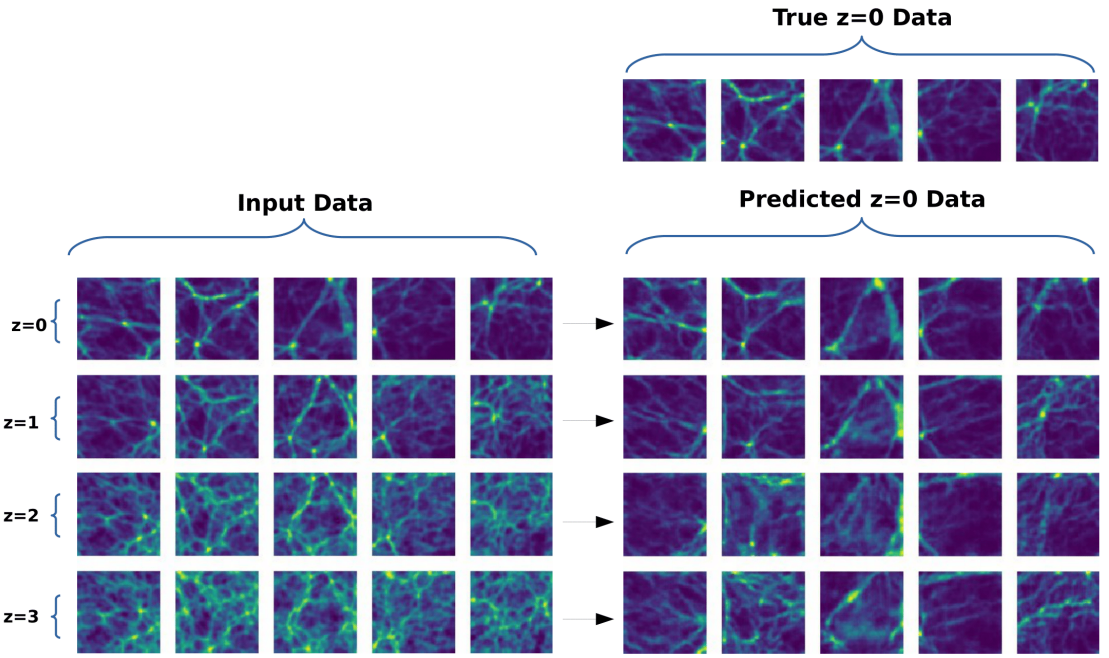

Figure 3.37: Five images from the 3D simulations at various redshifts (*left*), and their equivalent predictions of redshift $z = 0$(*right*) as inferred by the *curriculum TW*. The true $z = 0$ simulation images are shown above the predicted images (*upper right*) for comparison.

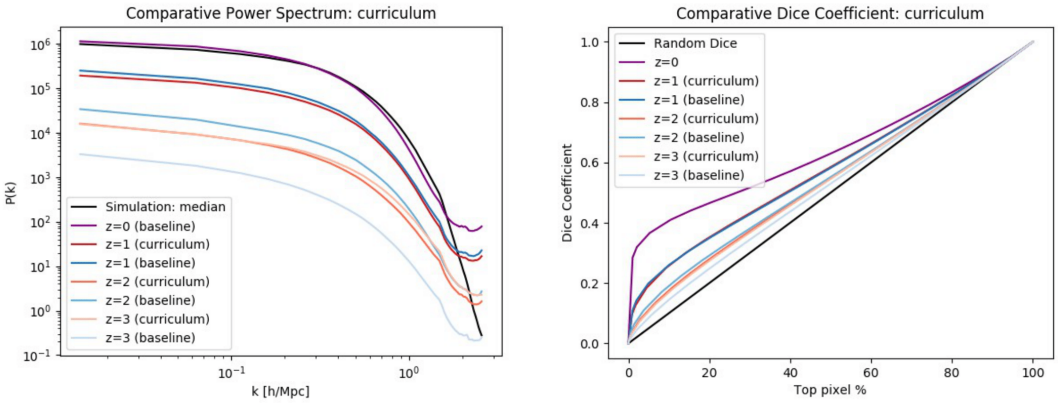

Figure 3.38: *Left:* Median power spectra of the data predicted by both by the *curriculum TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. As $z = 0$ is common to both it is shown in *purple*. The median power spectrum of the true $z = 0$ simulation data is shown in *black*. *Right:* Average dice coefficient between target $z = 0$ 3D simulation data and data as predicted both by the *curriculum TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. $z = 0$ is shown in *purple*.

when teaching it to predict $z = 0$ from a given $z' > 0$ will translate well when giving this network the new task of predicting $z = 0$ from a new $z'' > z'$. This is an interesting result in and of itself as it means that when training multiple networks to predict $z = 0$ from various $z' > 0$ inputs, we can initialize their weights to that of previously trained networks to optimize training time, instead of initializing all of their weights randomly.

A close examination of the predicted images in fig.3.37 suggests somewhat better results than the *baseline TW*'s for $z = 3$, with predicted structures appearing much less random, but no significant difference for other input z. This is confirmed in both the power spectra (fig.3.38,*left*) and dice (fig.3.38,*right*), which show significantly better results for $z = 3$, with the power spectrum gaining nearly an order of magnitude and the dice rising in all regions, suggesting that the network is both displaying denser structures and placing them in the correct regions. However we find that the *curriculum TW* obtains similar or even slightly worse results to the *baseline TW*'s for $z = 1$ and 2.

To conclude, we find that this method shows all its efficiency when training a network for a complex task (in our case predicting $z = 0$ from $z = 3$). We recall that this method is a manner in which one softens a learning learning curve that is too steep for a network to effectively improve in its task. Thus we can establish, in concordance with our earlier hypothesis, that we are indeed faced with this

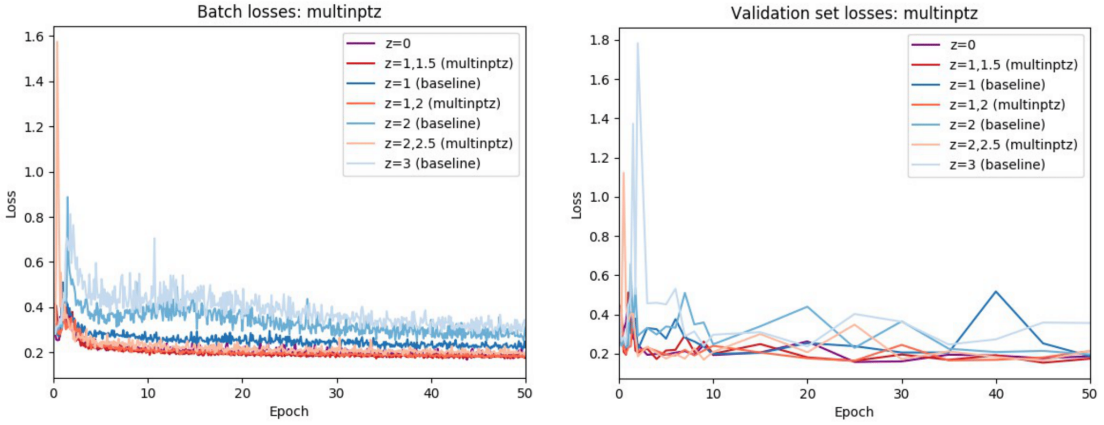

Figure 3.39: Batch and validation set losses for the *Multiple Input TW* (*red*) shown next to those of the *baseline TW* (*blue*) for training at various input redshifts. $z = 0$ is shown in *purple*.

issue of too steep a learning curve when using $z = 3$ as input. Conversely, this does not seem to be the case for inputs $z = 1$ and 2, with the networks yielding comparable (and thus largely inferior to $z = 0$ input) results with this curriculum method, suggesting that here our limiting factor is possibly not a matter of task complexity, but rather one of insufficient information in the input.

Thus it is worth exploring how our networks perform when providing them with more information in the input, to give them more clues as to the dynamics of the matter whose density field evolution they are trying to predict.

Multiple redshift input Another way to make the task of prediction easier is to supply the TW with more information; indeed, in the same way that certain CNN models can predict the following frames of a video given a set of previous frames (Oprea et al., 2020), we attempted supplying the model with input data at multiple redshifts rather than one datum at a single redshift. In our experiment, we used two redshifts at a time as input but one could very well use three or more for further attempts.

Because when running our simulations we saved our snapshots at equal redshift spacings of 0.5, we make use of pairs taken from possible redshifts $0.5, 1, 1.5, 2, 2.5, 3$; but for further work, especially with three or more inputs, we should consider spacings that are equal in time rather than redshift.

From the start, a simple observation of the batch (*left*) and validation (*right*) losses in fig.3.39 suggests that training is proceeding much better than for the *baseline TW*, for any input $z > 0$; all losses are low and close to that of $z = 0$, and validation losses are much less noisy.

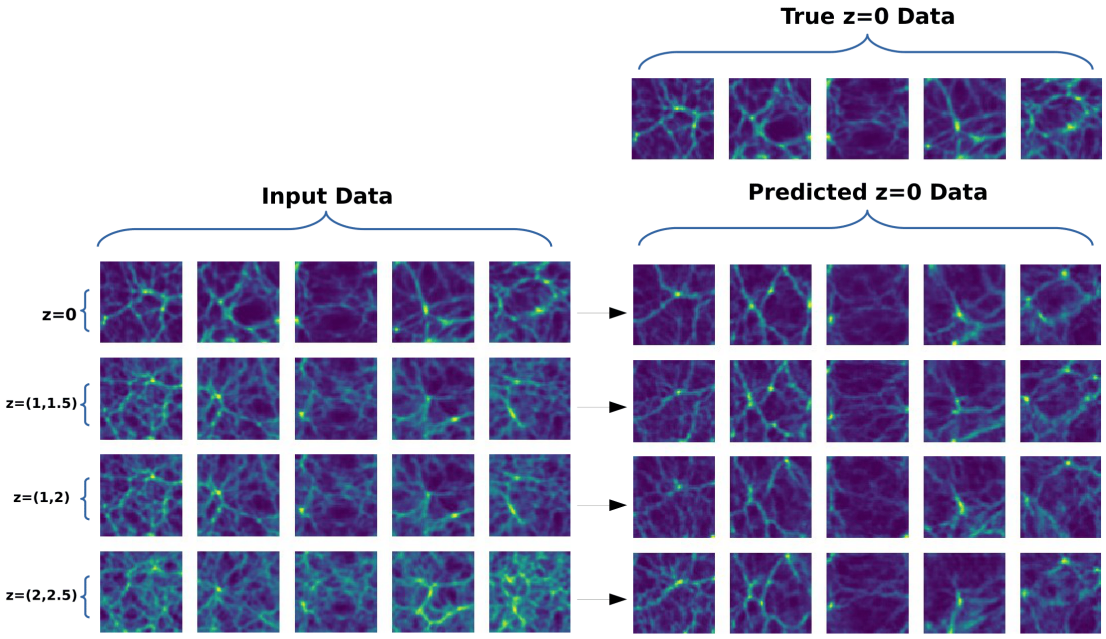

Figure 3.40: Five images from the 3D simulations at various redshifts (*left*), and their equivalent predictions of redshift $z = 0$(*right*) as inferred by the *Multiple Input TW*. The true $z = 0$ simulation images are shown above the predicted images (*upper right*) for comparison. Though we input two sets of data, here, we only show the input set with the lowest z for simplicity.

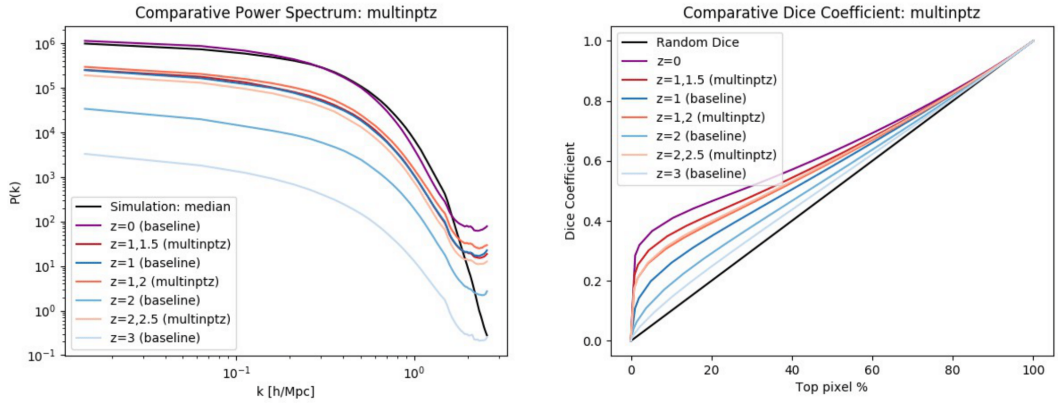

Figure 3.41: *Left:* Median power spectra of the data predicted by both by the *Multiple Input TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. $z = 0$ is shown in *purple*. The median power spectrum of the true $z = 0$ simulation data is shown in *black*. *Right:* Average dice coefficient between target $z = 0$ 3D simulation data and data as predicted both by the *Multiple Input TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. $z = 0$ is shown in *purple*.

An inspection of the predicted data (fig.3.40) further confirms this, with models yielding nearly identical results regardless of the input z. Unsurprisingly, they seem to have the same limitations as the data inferred by the *baseline AE* (see predicted data for $z = 0$), with more diffuse structure being recovered more randomly.

We find that the power spectra of the *multiple input TW*'s predicted data (fig.3.41, *left*) tend to fall closer to that of the baseline TW's data predicted from $z = 1$. This is expected for predictions from inputs containing $z = 1$ as lowest z input (as we expect such results to be equal or better to the baseline with $z = 1$ input, as the input provides additional information), but is a satisfactory result for inputs $z = (2, 2.5)$, which yield a much better power spectrum than baseline with $z = 2$ input.

Finally, examining the dice coefficient (fig.3.41, *right*) of the predicted data, we can observe that the *multiple input TW* significantly outperforms the *baseline TW*, with all input types yielding a higher dice coefficient than the baseline at input $z = 1$, at every pixel threshold. This is not too surprising in the cases where the input contains additional information to the one given to the baseline TW, (ie inputs $z = (1, 1.5)$ and $z = (1, 2)$ performing better than simple input $z = 1$), but more compelling is the fact that this is also true for higher z inputs ($z = (2, 2.5)$ outperforming input $z = 1$). Additionally, although further testing with additional combinations of input z would help to confirm this, it appears that multiple inputs perform better when they are closer in time to each other (inputs $z = (1, 1.5)$ outperform inputs $z = (1, 2)$, and even inputs $z = (2, 2.5)$ seem to

slightly outperform inputs $z = (1, 2)$). If the models perform better with smaller δt between their input data, we might expect that they would perform even better if provided directly with velocity fields.

Velocities Another type of information that we can obtain from the snapshots and that we have not used so far is the velocity distribution of the particles. Indeed, in their raw form and for every saved snapshot, our N-body simulations provide us with every particle's position, but also every particle's velocity, in the form of a 3D vector for each particle. This additional information is bound to constrain more thoroughly the future density fields, as initial velocities notably provide the necessary information for a linear evolution of the density field. However, similar to the particle positions, we must translate the velocities of individual particles into an averaged field such that our networks can use them as inputs. So far we have only attempted this approach for the 3D cubes; we recall that our density cubes are smaller, randomly oriented sub-arrays of a large cube that is built by computing a $768 \times 768 \times 768$ 3D histogram of particle positions, log-transforming (see Eq.2.6) the values of the histogram to make cosmic structures salient, and smoothing the result into a final $256 \times 256 \times 256$ array.

We follow a similar logic to build the velocity field. Dividing the snapshot space into $768 \times 768 \times 768$ voxels, we compute three 3D averaged velocity fields for each direction (x, y, z), by summing the velocities of the particles in each voxel and dividing the sum by the number of particles. Inspecting the resulting cubes, we find that cosmic structures are visually apparent without need for log-transformation. Thus we simply apply a normalization of voxel values:

$$v' = v/N \tag{3.7}$$

Where v' is the new velocity and N is fixed such that $|v'|_{max} \lesssim 1$. This done, we apply the same smoothing as for the 3D density to obtain our final three $256 \times 256 \times 256$ velocity fields for each (x, y and z) direction.

Combining the density field with the velocity fields thus constructed, we are equipped with an array of size $256 \times 256 \times 256 \times 4$ from which we extract smaller arrays of size $64 \times 64 \times 64 \times 4$.

Equipped with these data, we train a TW to recover the density field at $z = 0$ with (ρ, v_x, v_y, v_z) at various $z > 0$ inputs.

From the start, a simple inspection of the batch (*left*) and validation (*right*) losses 3.43 suggests that training is proceeding much better than for the *baseline TW*, for any input $z > 0$; all losses are low and close to that of $z = 0$, and validation losses are much less noisy.

Observing the predicted data (3.44) further confirms this, with models yielding nearly identical results regardless of the input z. Unsurprisingly, they seem to have

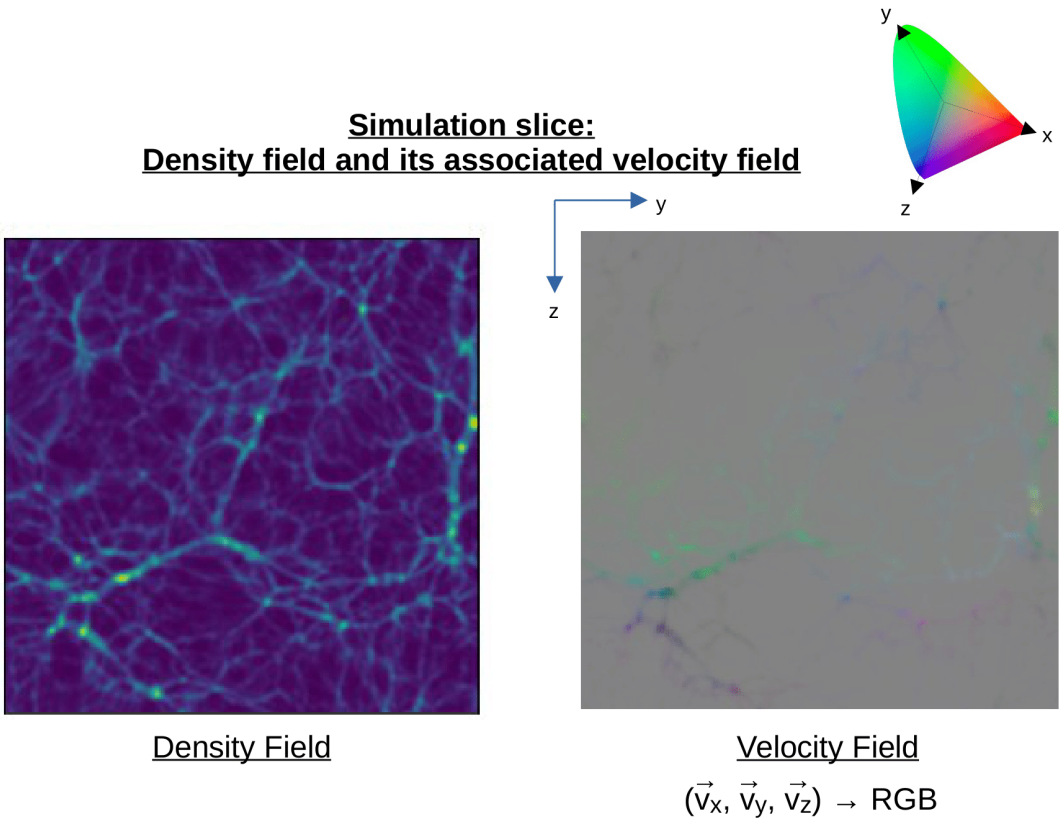

Figure 3.42: Example slice of a 3D simulation, showing the density field (*left*) and its associated velocity field (*right*), represented in ($\vec{v}_x$,$\vec{v}_y$,$\vec{v}_z$) to (R,G,B)

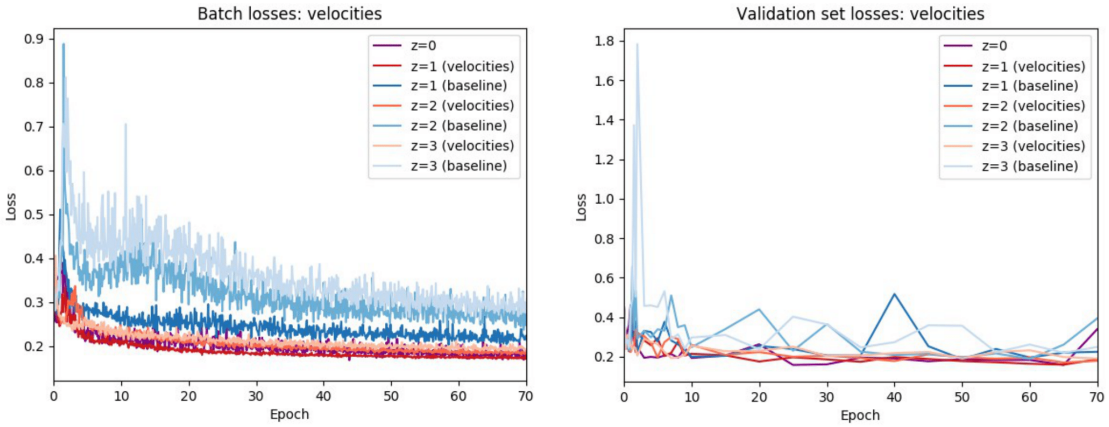

Figure 3.43: Batch and validation set losses for the *velocities TW* (*red*) shown next to those of the *baseline TW* (*blue*) for training at various input redshifts. As $z = 0$ is common to both it is shown in *purple*.

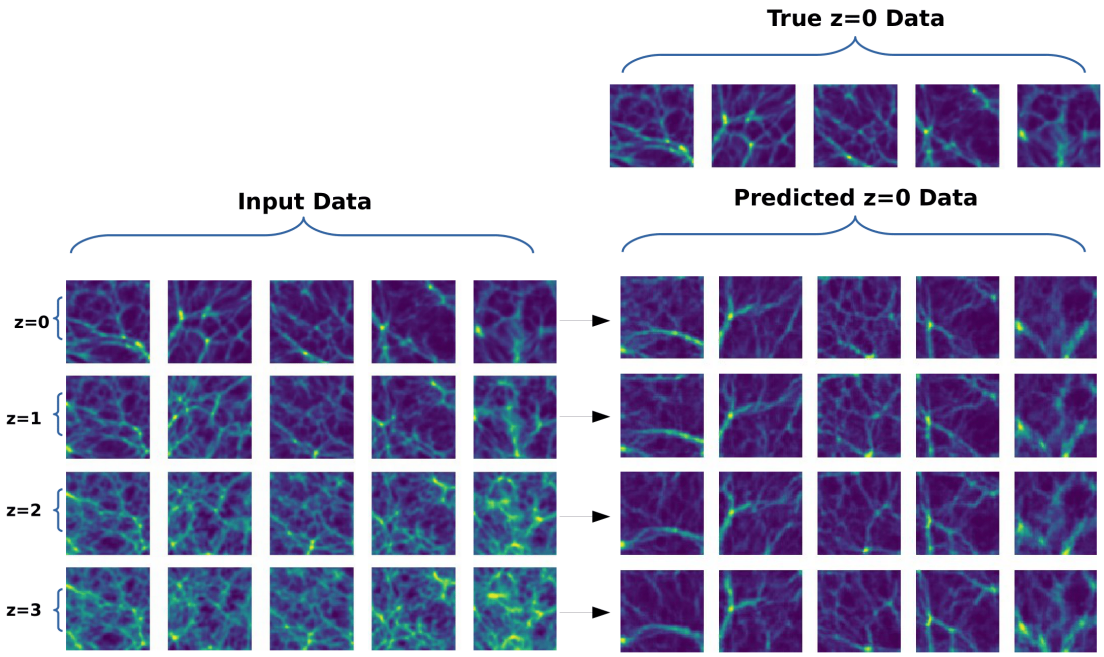

Figure 3.44: Five images from the 2D simulations at various redshifts (*left*), and their equivalent predictions of redshift $z = 0$(*right*) as inferred by the *velocities TW*. The true $z = 0$ simulation images are shown above the predicted images (*upper right*) for comparison.

the same limitations as the data inferred by the *baseline TW* with input $z = 0$ (see fig.3.13), with more diffuse structure being recovered more randomly. Indeed, a TW trained with additional information should not exceed the results of an AE provided with all the needed information as input.

As can be expected given the similarity of the predicted data, the predicted power spectra (Fig. 3.45, *left*) are close to the true simulation power spectrum, especially when compared to those recovered by the *baseline TW*.

Finally, a study of the dice coefficient (Fig. 3.45, *right*) completes the picture by showing that the *velocities TW* outperforms the *baseline TW*, to the point that the recovered dice of the *velocities TW* for input $z = 3$ is better than that of the *baseline TW* with input $z = 1$.

We can conclude that the model makes good use of the velocity field to efficiently predict the $z = 0$ density field for all input z. This confirms the hypothesis according to which the cause of the baseline TW's poor prediction of $z = 0$ from $z = 2$ and 3 is a problem of insufficient input information, leading to an ill-constrained future density field, rather than a weak model. Indeed, knowing the initial distribution and velocity of matter, the network has nearly (neighboring

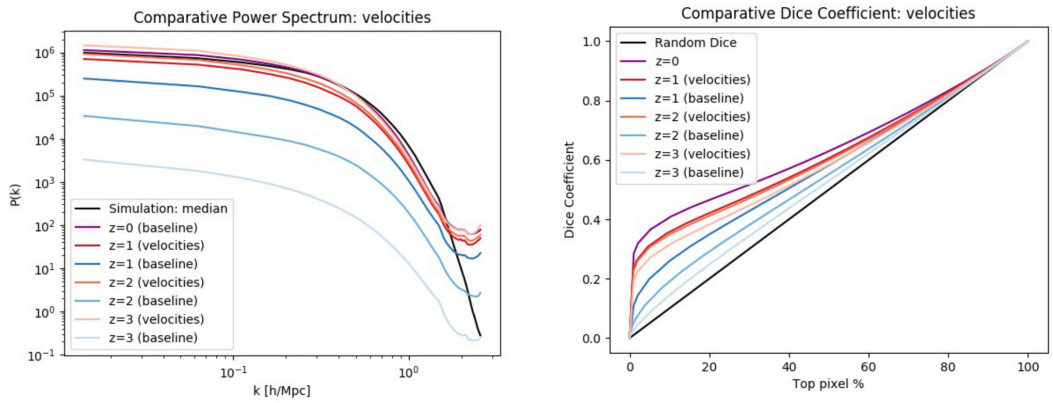

Figure 3.45: *Left:* Median power spectra of the data predicted by both by the *velocities TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. As $z = 0$ is common to both it is shown in *purple*. The median power spectrum of the true $z = 0$ simulation data is shown in *black*. *Right:* Average dice coefficient between target $z = 0$ 3D simulation data and data as predicted both by the *velocities TW* (*red*) and the *baseline TW* (*blue*) from various input redshifts. $z = 0$ is shown in *purple*.

environment aside) all the necessary keys to approximate the matter dynamics, and thus predict its evolution.

3.6.3 Conclusion on the Predictive Autoencoder

Equipped with our Replicative AE, we aimed to go further than basic replication and inquired into this model's ability to predict density field evolution, tasking it with predicting a density field at $z = 0$ from various higher z (1,2,3). We found that in its baseline form it performed quite well in its prediction of 2D density fields, yielding predictions that were nearly equal in precision (as measured by a dice coefficient) to that of the replicative AE, showing a slight decrease in precision the higher the input z. On the other hand, it performed quite poorly on our 3D cubes, with an important decrease both in precision and statistics (pixel pdf, mean density distribution, peak counts and power spectrum) the higher the input z, up to a near-random prediction for input $z = 3$. We supposed that this could be due to two main issues: the first being that the task of prediction from high input z had too steep a learning curve, rendering the network unable to progress effectively in learning its task. The second issue was that the input that we supplied to the network provided insufficient information for a precise prediction; indeed since the same density field at a given $z > 0$ can lead to vastly differing fields at $z = 0$ depending on its velocity field and neighboring environment, the best we

can expect is for the network to yield an end result that is marginalized on the unknown information, and will thus differ from the true field proportionally to field variance over this unknown information.

To test both hypotheses, we came up with variations on our training method that could mitigate both problems.

To ease the learning, we devised a progressive training method wherein we trained the network first to predict $z = 0$ from $z = 0$ and incremented the input z by $\Delta z = 0.5$ every 50 epochs so that the network had time to progress with a simpler task before training it to predict from high input z. We referred to this model as the *Curriculum Timewarper*. Looking at the loss evolution, we found that the network did indeed benefit from prior training with lower input z, with losses starting much lower than that of networks with no prior training for all input z. However in terms of final results this method only showed significant improvement for input $z = 3$, with all results for predicting $z = 0$ from any input $z > 0$ remaining quite poor. Thus we can conclude that the issue of a steep learning curve is only marginal, as it only affects results for high input z.

To counter the second issue, we came up with two methods to supply the network with more information on the density field's dynamics. A first simple method consisted in providing the network with inputs at multiple redshifts (namely $z = (1, 1.5), (1, 2)$ and $(2, 2.5)$) so that the network better infers matter dynamics from several steps of the density field's evolution process. This indeed proved quite effective, with networks trained with input pairs always outperforming their one-input equivalent, and notably input $z = (2, 2.5)$ even outperforming the *baseline TW* with input $z = 1$. For further work it might be worth looking at additional input redshift pairs, or try to input three or more redshifts as well.

Given these satisfactory results we can safely assume that lack of information in the input constituted the main reason for our initial 3D model's poor performance.

To go further in the line of providing the network with field dynamics we also trained a network that we supplied not only with density fields at $z > 0$ but with their associated velocity fields as well. Indeed, were the network able to correctly recover the laws governing the motion of matter, it would need both matter position and velocity at a given time to correctly predict the evolution of its density field. We found that this method proved very promising indeed, with predictions even from input $z = 3$ largely outperforming the *baseline TW's* with input $z = 1$ both in terms of precision and statistics, and with data predicted from all input z looking visually very similar to the true target $z = 0$ fields. An additional step might be to see if the network is able to predict the velocity fields at $z = 0$ as well.

Thus we can positively conclude to the network's ability to predict structure evolution, provided that it is supplied with sufficient input information that can

adequately constrain the ensuing $z = 0$ density field.

This said, we must address the fact that the results for the baseline TW and all the variations that we explored this far have the same limits as that of our baseline AE; indeed, while the combined effects of using a generator with locked weights as a decoder and a truncated discriminator to compute the loss allow us to ensure that predicted data retain good statistics (pixel pdf, mean density distribution, peak counts and power spectrum), this also severely limits us in several ways.

First, as we already observed in the case of the *Traditional AE*, we cannot reproduce a given datum as well (in terms of dice coefficient) as a method wherein the decoder's weights are free. Though we chose to conduct our experiments based on the idea of maintaining satisfactory statistics, it would be worth looking into aiming purely for an optimal dice coefficient.

Second, our setup requires us to train a GAN for every type of data we want to target. For example, training a TW to predict $z = 3$ from $z < 3$ implies training a new GAN that emulates data from $z = 3$ simulations. Alternatively we could train a more complex GAN such as a cGAN (Mirza and Osindero, 2014) to emulate simulation data at various z with z labels, but this remains quite an expensive endeavor compared to a more direct approach.

Thus it might be worth exploring other options, starting with new variations on the TW where the training is more traditional (all weights are free to vary and the loss is more neutral, such as l1 or l2). We attempted this in the baseline setup but results were inconclusive. Still, given the outstanding results of some of our variations, it might be worth looking into combining traditional training with some of these variations, such as adding velocities in the input, for example.

For this purpose, all combinations of the variations we exposed could be worth looking into; for example, earlier work combining multiple input and curriculum training seemed promising.

Alternatively, we could retain our current AE training structure but apply it to tasks where recovering correct statistics is more relevant, such as denoising or reconstructing masked areas. We have attempted denoising but results are as of yet inconclusive.

On another note, it would be worth further testing our results to direct our efforts of optimization. For example, we expect a datum's neighboring environment to affect our network's capacity of prediction; thus it would be worth comparing the dice coefficient computed on pixels in the center of a datum to that of pixels on the borders to see if there is indeed a loss of precision in the prediction of the latter. Additionally it would be interesting to see if our model is better suited to certain types of input, for example by plotting model loss against input parameters such as mean density of the input or input's highest pixel value.

Finally, our future work will focus on evolving the density field backward in

time, starting by predicting $z = 3$ from various $z < 3$. our preliminary attempts have so far been inconclusive, but we expect that combining the different methods exposed above will eventually lead to better results, though we do expect the task to be harder than forward prediction because of rising entropy.

Chapter 4

Conclusion

In this work, we have explored the uses of deep neural networks as a means to extract information from N-body simulation snapshots in order to obtain general properties stemming from these simulations and apply them in a quick and efficient manner through the use of these CNNs.

We first devised a GAN, whose generator is able to closely determine a set of simulation-issued data's underlying distribution, describe it in a latent space of small dimensionality, and yield new data extracted from this discovered distribution, and whose discriminator can efficiently determine whether a given datum is likely to be issued from this distribution. While developing this GAN and the *modus operandi* surrounding its use and evaluation we acquired many crucial takeaways. First in terms of data preprocessing, where we had to convert N-body simulations, that came in the form of a set of particle positions, into numerical arrays representing density fields that our GAN (and following networks) could process. We discovered the importance of building data wherein patterns were salient, both in terms of having smooth continuous structures, but also in the sense of structures standing out starkly against more diffuse background. Given our data, this amounted to building a histogram of the particle positions and finding an appropriate smoothing and log-transformation to apply on the result. Secondly, while the manipulation and progressive modification of a rudimentary GAN in an attempt to render it compatible with our data proved useful in terms of acquainting oneself with the various components that make up a GAN, we found that our models revealed to be quite unstable when modifying their hyperparameters too significantly, leading to extreme difficulty when trying to build a GAN from the ground up. Thus we concluded that the best approach was to rely on pre-existing models that were optimized for data that is as close as possible in nature to the data that we are working with.

In a second phase, we repurposed our trained GAN's generator and discriminator to build an AE, first with the goal to simply replicate data after having

encoded it into a vector of smaller dimension, and in a second stage training one to predict data at redshift $z = 0$ from higher redshifts $z = (1, 2, 3)$. In both cases we found that using a trained generator as a decoder and incorporating a truncated version of a decoder to compute the AE's loss allowed us to create an AE that excels in returning data that maintains its statistical properties (eg density distribution, power spectrum, etc.) after encoding. When tasking the AE with predicting the evolution of density fields from redshifts $z = 1, 2, 3$ to $z = 0$, we found that it performed very well once provided with sufficient input information. While in the case of 2D data, simply supplying the density fields as input sufficed to obtain satisfactory predictions, this was not the case for 3D data. However, providing the additional information of the input density fields' associated velocity fields proved quite efficient in the 3D case, with the predictive AE recovering the large structures with decent precision (as measured by dice coefficient). From this we can optimistically conclude as to our model's ability to approximate the effects of expansion and gravitation on a density field over a set amount of time.

This opens many perspectives, especially given the relative simplicity of our models; indeed, our structure is that of a very simple (encoder+decoder) AE, with half of its weights locked during training, and the data being reduced into a relatively small number of parameters before decoding. While the point of this maneuver is to efficiently extract a datum's key components, one can expect there to be a significant loss of information during encoding. Thus it would be worth exploring several approaches that would allow more information to be extracted and retained during the model's analysis of a datum.

Many simple modifications of our structure could be inquired into, from adding layers and experimenting with kernel sizes to modifying the size of our latent encoding layer, but it would also be worth experimenting with more complex network structures that are more fine-tuned to this type of data transformation. For example U-nets (Ronneberger et al., 2015), while initially developed for image segmentation, have the advantage of linking every stage of feature extraction to an equivalent stage of feature reconstruction, providing much more passing of information from encoder to decoder, and is thus likely to provide good results. Additionally, there exist several variations on GANs that are able to transform a datum in some manner; the most common being the cGAN (Mirza and Osindero, 2014), some of which have been used to simulate face aging (Antipov et al., 2017), which is another sort of time evolution.

Additionally, we can comment upon the experimental method of developing a machine learning algorithm from the ground up for a given scientific task. The process itself is not unlike the training of a neural network, with multiple parameters to account for, from the choice and construction of training data, to the many choices surrounding the network type and architecture for a given task, to the

development of a trusty set of tests and measures to ascertain (or invalidate) the quality of our results. In other words, finding a working network can be seen as a high-dimensional problem with a much lower hyperplane of effective and useful decisions to try. It is quite recursive work, with the updating of one aspect often implying modifications on every other front to account for the changes made. Additionally, as when looking at a loss evolution, it is important to look at the bigger picture and know when to stop pushing forward and change methods, or one risks never converging, always expecting small local improvements to lead to a satisfactory result.

Finally, we can conclude as to the stunning ability of DNNs to extract meaningful information from simulated data to infer underlying mechanisms that are applicable not only to data that they have encountered, but also and especially to new instances. In our case this mechanism is an approximation of the (gravitation + expansion) combination governing the large-scale formation of structure, but we can similarly expect other models to emulate other physical processes or to discover new unexpected relationships between physical variables. Given that most DNNs are black boxes that do not provide a good analytical description of their inner workings, creating even a highly-effective model in any task should only remain a first step into developing a robust scientific method to perform this task. A deeper examination of the network to understand how exactly it is operating should follow, first to determine the limitations of the model, but more importantly because the method is more informative than the result.

Bibliography

Aghanim, N., Akrami, Y., Ashdown, M., Aumont, J., Baccigalupi, C., Ballardini, M., Banday, A., Barreiro, R., Bartolo, N., Basak, S., et al. (2018). Planck 2018 results. vi. cosmological parameters. *arXiv preprint arXiv:1807.06209*.

Ahdida, C., Albanese, R., Alexandrov, A., Anokhina, A., Aoki, S., Arduini, G., Atkin, E., Azorskiy, N., Back, J., Bagulya, A., et al. (2019). Fast simulation of muons produced at the ship experiment using generative adversarial networks. *Journal of Instrumentation*, 14(11):P11028.

Aljalbout, E., Golkov, V., Siddiqui, Y., Strobel, M., and Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*.

Amato, G., Behrmann, M., Bimbot, F., Caramiaux, B., Falchi, F., Garcia, A., Geurts, J., Gibert, J., Gravier, G., Holken, H., et al. (2019). Ai in the media and creative industries. *arXiv preprint arXiv:1905.04175*.

Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Face aging with conditional generative adversarial networks. In *2017 IEEE international conference on image processing (ICIP)*, pages 2089–2093. IEEE.

Aragon-Calvo, M. A. (2020). Smooth stochastic density field reconstruction.

Badjatiya, P., Gupta, S., Gupta, M., and Varma, V. (2017). Deep learning for hate speech detection in tweets. In *Proceedings of the 26th international conference on World Wide Web companion*, pages 759–760.

Baldi, P., Sadowski, P., and Whiteson, D. (2015). Enhanced higgs boson to $\tau+ \tau-$ search with deep learning. *Physical review letters*, 114(11):111801.

Ball, N. M., Brunner, R. J., Myers, A. D., and Tcheng, D. (2006). Robust machine learning applied to astronomical data sets. i. star-galaxy classification of the sloan digital sky survey dr3 using decision trees. *The Astrophysical Journal*, 650(1):497.

Bang, D., Kang, S., and Shim, H. (2020). Discriminator feature-based inference by recycling the discriminator of gans. *International Journal of Computer Vision*, pages 1–23.

Baso, C. D., de la Cruz Rodriguez, J., and Danilovic, S. (2019). Solar image denoising with convolutional neural networks. *Astronomy & Astrophysics*, 629:A99.

Beichman, C., Benneke, B., Knutson, H., Smith, R., Lagage, P.-O., Dressing, C., Latham, D., Lunine, J., Birkmann, S., Ferruit, P., et al. (2014). Observations of transiting exoplanets with the james webb space telescope (jwst). *Publications of the Astronomical Society of the Pacific*, 126(946):1134.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Bethge, M., Ecker, A., and Gatys, L. (2016). Deepart. *URL: https://deepart. io*.

Birkinshaw, M. (1999). The sunyaev–zel'dovich effect. *Physics Reports*, 310(2-3):97–195.

Bond, J. R., Kofman, L., and Pogosyan, D. (1996). How filaments of galaxies are woven into the cosmic web. *Nature*, 380(6575):603–606.

Bonjean, V. (2020). Deep learning for sunyaev–zel'dovich detection in planck. *Astronomy & Astrophysics*, 634:A81.

Bora, A., Price, E., and Dimakis, A. G. (2018). Ambientgan: Generative models from lossy measurements. In *International Conference on Learning Representations*.

Briot, J.-P. and Pachet, F. (2017). Music generation by deep learning-challenges and directions. *arXiv preprint arXiv:1712.04371*.

Buchert, T. (1995). Lagrangian perturbation approach to the formation of large-scale structure. *arXiv preprint astro-ph/9509005*.

Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., and Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002.

Casert, C., Mills, K., Vieijra, T., Ryckebusch, J., and Tamblyn, I. (2020). Optical lattice experiments at unobserved conditions and scales through generative adversarial deep learning. *arXiv preprint arXiv:2002.07055*.

Chalapathy, R. and Chawla, S. (2019). Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*.

Chong, E., Han, C., and Park, F. C. (2017). Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187–205.

Clark, A., Donahue, J., and Simonyan, K. (2019). Efficient video generation on complex datasets. *arXiv preprint arXiv:1907.06571*.

Coles, P. and Chiang, L.-Y. (2000). Characterizing the nonlinear growth of large-scale structure in the universe. *Nature*, 406(6794):376–378.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. (2020). Discovering symbolic models from deep learning with inductive biases. *arXiv preprint arXiv:2006.11287*.

Davies, A., Serjeant, S., and Bromley, J. M. (2019). Using convolutional neural networks to identify gravitational lenses in astronomical images. *Monthly Notices of the Royal Astronomical Society*, 487(4):5263–5271.

de Oliveira, L., Paganini, M., and Nachman, B. (2017). Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science*, 1(1):4.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE.

Dewdney, P. E., Hall, P. J., Schilizzi, R. T., and Lazio, T. J. L. (2009). The square kilometre array. *Proceedings of the IEEE*, 97(8):1482–1496.

Donahue, C., McAuley, J., and Puckette, M. (2018). Adversarial audio synthesis.

Dosovitskiy, A. and Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks.

Dubois, Y., Peirani, S., Pichon, C., Devriendt, J., Gavazzi, R., Welker, C., and Volonteri, M. (2016). The horizon-agn simulation: morphological diversity of galaxies promoted by agn feedback. *Monthly Notices of the Royal Astronomical Society*, 463(4):3948–3964.

Ettinger, S., Cheng, S., Caine, B., Liu, C., Zhao, H., Pradhan, S., Chai, Y., Sapp, B., Qi, C., Zhou, Y., et al. (2021). Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. *arXiv preprint arXiv:2104.10133*.

Feder, R. M., Berger, P., and Stein, G. (2020). Nonlinear 3d cosmic web simulation with heavy-tailed generative adversarial networks. *arXiv preprint arXiv:2005.03050*.

Fixsen, D., Cheng, E., Gales, J., Mather, J. C., Shafer, R., and Wright, E. (1996). The cosmic microwave background spectrum from the full cobe* firas data set. *The Astrophysical Journal*, 473(2):576.

Flamary, R. (2017). Astronomical image reconstruction with convolutional neural networks. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2468–2472. IEEE.

Forero-Romero, J. E., Hoffman, Y., Gottlöber, S., Klypin, A., and Yepes, G. (2009). A dynamical classification of the cosmic web. *Monthly Notices of the Royal Astronomical Society*, 396(3):1815–1824.

Foster, D. (2019). *Generative deep learning: teaching machines to paint, write, compose, and play*. O'Reilly Media.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR.

Hacohen, G. and Weinshall, D. (2019). On the power of curriculum learning in training deep networks. In *International Conference on Machine Learning*, pages 2535–2544. PMLR.

Hassan, S., Liu, A., Kohn, S., and La Plante, P. (2019). Identifying reionization sources from 21 cm maps using convolutional neural networks. *Monthly Notices of the Royal Astronomical Society*, 483(2):2524–2537.

He, S., Li, Y., Feng, Y., Ho, S., Ravanbakhsh, S., Chen, W., and Póczos, B. (2019). Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, 116(28):13825–13832.

He, X. and Deng, L. (2017). Deep learning for image-to-text generation: A technical overview. *IEEE Signal Processing Magazine*, 34(6):109–116.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Hyun, C. M., Kim, H. P., Lee, S. M., Lee, S., and Seo, J. K. (2018). Deep learning for undersampled mri reconstruction. *Physics in Medicine & Biology*, 63(13):135007.

Iqbal, T. and Qureshi, S. (2020). The survey: Text generation models in deep learning. *Journal of King Saud University-Computer and Information Sciences*.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

Ivezić, Ž., Kahn, S. M., Tyson, J. A., Abel, B., Acosta, E., Allsman, R., Alonso, D., AlSayyad, Y., Anderson, S. F., Andrew, J., et al. (2019). Lsst: from science drivers to reference design and anticipated data products. *The Astrophysical Journal*, 873(2):111.

Jia, P., Liu, Q., and Sun, Y. (2020a). Detection and classification of astronomical targets with deep neural networks in wide-field small aperture telescopes. *The Astronomical Journal*, 159(5):212.

Jia, P., Wu, X., Yang, X., Huang, Y., Cai, B., and Cai, D. (2020b). Astronomical image restoration and point spread function estimation with deep neural networks. In *Advances in Optical Astronomical Instrumentation 2019*, volume 11203, page 112030Q. International Society for Optics and Photonics.

Kaiser, N., Squires, G., and Broadhurst, T. (1994). A method for weak lensing observations. *arXiv preprint astro-ph/9411005*.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kitaura, F.-S. and Heß, S. (2013). Cosmological structure formation with augmented lagrangian perturbation theory. *Monthly Notices of the Royal Astronomical Society: Letters*, 435(1):L78–L82.

Kowsari, K., Brown, D. E., Heidarysafa, M., Meimandi, K. J., Gerber, M. S., and Barnes, L. E. (2017). Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE.

Lample, G. and Charton, F. (2019). Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*.

Laureijs, R. J., Duvet, L., Sanz, I. E., Gondoin, P., Lumb, D. H., Oosterbroek, T., and Criado, G. S. (2010). The euclid mission. In *Space Telescopes and Instrumentation 2010: Optical, Infrared, and Millimeter Wave*, volume 7731, page 77311H. International Society for Optics and Photonics.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

Lee, D., Kim, J., Moon, W.-J., and Ye, J. C. (2019). Collagan: Collaborative gan for missing image data imputation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2487–2496.

Li, J., Dai, W., Metze, F., Qu, S., and Das, S. (2017). A comparison of deep learning methods for environmental sound detection. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 126–130. IEEE.

List, F., Bhat, I., and Lewis, G. F. (2019). A black box for dark sector physics: predicting dark matter annihilation feedback with conditional gans. *Monthly Notices of the Royal Astronomical Society*, 490(3):3134–3143.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., and Pietikäinen, M. (2020). Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2):261–318.

Long, M., Soubo, Y., Cong, S., Weiping, N., and Tong, L. (2021). Learning deconvolutions for astronomical images. *Monthly Notices of the Royal Astronomical Society*, 504(1):1077–1083.

Lotter, W., Kreiman, G., and Cox, D. (2016). Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*.

Lukic, V., de Gasperin, F., and Brüggen, M. (2020). Convosource: radio-astronomical source-finding with convolutional neural networks. *Galaxies*, 8(1):3.

Lv, Y., Duan, Y., Kang, W., Li, Z., and Wang, F.-Y. (2014). Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. (2015). Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*.

Margalef-Bentabol, B., Huertas-Company, M., Charnock, T., Margalef-Bentabol, C., Bernardi, M., Dubois, Y., Storey-Fisher, K., and Zanisi, L. (2020). Detecting outliers in astronomical images with deep generative networks. *Monthly Notices of the Royal Astronomical Society*, 496(2):2346–2361.

Margapuri, V., Thapa, B., and Shamir, L. (2021). Automatic detection of novelty galaxies in digital sky survey data. *International journal of computer application*, 28(1).

Mathieu, M., Couprie, C., and LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*.

McCarthy, I. G., Schaye, J., Bird, S., and Le Brun, A. M. C. (2016). The bahamas project: calibrated hydrodynamical simulations for large-scale structure cosmology. *Monthly Notices of the Royal Astronomical Society*, page stw2792.

Mescheder, L., Geiger, A., and Nowozin, S. (2018). Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR.

Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. (2018). A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6:39501–39514.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Monaco, P., Theuns, T., and Taffoni, G. (2002). The pinocchio algorithm: pinpointing orbit-crossing collapsed hierarchical objects in a linear density field. *Monthly Notices of the Royal Astronomical Society*, 331(3):587–608.

Mordvintsev, A., Olah, C., and Tyka, M. (2015). Deepdream-a code example for visualizing neural networks. *Google Research*, 2(5).

Ntampaka, M., Avestruz, C., Boada, S., Caldeira, J., Cisewski-Kehe, J., Stefano, R. D., Dvorkin, C., Evrard, A. E., Farahi, A., Finkbeiner, D., Genel, S., Goodman, A., Goulding, A., Ho, S., Kosowsky, A., Plante, P. L., Lanusse, F., Lochner, M., Mandelbaum, R., Nagai, D., Newman, J. A., Nord, B., Peek, J. E. G., Peel, A., Poczos, B., Rau, M. M., Siemiginowska, A., Sutherland, D. J., Trac, H., and Wandelt, B. (2019). The role of machine learning in the next decade of cosmology.

Nuzillard, D. and Bijaoui, A. (2000). Blind source separation and analysis of multispectral astronomical images. *Astronomy and Astrophysics Supplement Series*, 147(1):129–138.

Oprea, S., Martinez-Gonzalez, P., Garcia-Garcia, A., Castro-Vargas, J. A., Orts-Escolano, S., Garcia-Rodriguez, J., and Argyros, A. (2020). A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Paschalis, P., Sarlanis, C., and Mavromichalaki, H. (2013). Artificial neural network approach of cosmic ray primary data processing. *Solar Physics*, 282(1):303–318.

Picquenot, A., Acero, F., Bobin, J., Maggi, P., Ballet, J., and Pratt, G. W. (2019). Novel method for component separation of extended sources in x-ray astronomy. *Astronomy & Astrophysics*, 627:A139.

Pillepich, A., Springel, V., Nelson, D., Genel, S., Naiman, J., Pakmor, R., Hernquist, L., Torrey, P., Vogelsberger, M., Weinberger, R., et al. (2018). Simulating galaxy formation with the illustristng model. *Monthly Notices of the Royal Astronomical Society*, 473(3):4077–4106.

Popel, M., Tomkova, M., Tomek, J., Kaiser, Ł., Uszkoreit, J., Bojar, O., and Žabokrtskỳ, Z. (2020). Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature communications*, 11(1):1–15.

Poplin, R., Varadarajan, A. V., Blumer, K., Liu, Y., McConnell, M. V., Corrado, G. S., Peng, L., and Webster, D. R. (2018). Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158–164.

Puglisi, G. and Bai, X. (2020). Inpainting galactic foreground intensity and polarization maps using convolutional neural network. *arXiv preprint arXiv:2003.13691*.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., and Tang, J. (2018). Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119.

Rhinehart, N., McAllister, R., Kitani, K., and Levine, S. (2019). Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2821–2830.

Rivenson, Y., Zhang, Y., Günaydın, H., Teng, D., and Ozcan, A. (2018). Phase recovery and holographic image reconstruction using deep learning in neural networks. *Light: Science & Applications*, 7(2):17141–17141.

Rodríguez, A. C., Kacprzak, T., Lucchi, A., Amara, A., Sgier, R., Fluri, J., Hofmann, T., and Réfrégier, A. (2018). Fast cosmic web simulations with generative adversarial networks. *Computational Astrophysics and Cosmology*, 5(1):4.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Sadr, A. V. and Farsian, F. (2020). Inpainting via generative adversarial networks for cmb data analysis. *arXiv preprint arXiv:2004.04177*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *Advances in neural information processing systems*, 29:2234–2242.

Salman, A. G., Kanigoro, B., and Heryadi, Y. (2015). Weather forecasting using deep learning techniques. In *2015 international conference on advanced computer science and information systems (ICACSIS)*, pages 281–285. Ieee.

Schawinski, K., Turp, M. D., and Zhang, C. (2018). Exploring galaxy evolution with generative models. *Astronomy & Astrophysics*, 616:L16.

Shamir, L. and Wallin, J. (2014). Automatic detection and quantitative assessment of peculiar galaxy pairs in sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 443(4):3528–3537.

Shandarin, S. F. and Zeldovich, Y. B. (1989). The large-scale structure of the universe: Turbulence, intermittency, structures in a self-gravitating medium. *Reviews of Modern Physics*, 61(2):185.

Singh, S. P., Kumar, A., Darbari, H., Singh, L., Rastogi, A., and Jain, S. (2017). Machine translation using deep learning: An overview. In *2017 international conference on computer, communications and electronics (comptelix)*, pages 162–167. IEEE.

Skinner, G. and Walmsley, T. (2019). Artificial intelligence and deep learning in video games a brief review. In *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, pages 404–408. IEEE.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Springel, V. (2005). The cosmological simulation code gadget-2. *Monthly notices of the royal astronomical society*, 364(4):1105–1134.

Springel, V., White, S. D., Jenkins, A., Frenk, C. S., Yoshida, N., Gao, L., Navarro, J., Thacker, R., Croton, D., Helly, J., et al. (2005). Simulations of the formation, evolution and clustering of galaxies and quasars. *nature*, 435(7042):629–636.

Springel, V., Yoshida, N., and White, S. D. (2001). Gadget: a code for collisionless and gasdynamical cosmological simulations. *New Astronomy*, 6(2):79–117.

Starck, J.-L., Pantin, E., and Murtagh, F. (2002). Deconvolution in astronomy: A review. *Publications of the Astronomical Society of the Pacific*, 114(800):1051.

Storey-Fisher, K., Huertas-Company, M., Ramachandra, N., Lanusse, F., Leauthaud, A., Luo, Y., and Huang, S. (2020). Anomaly detection in astronomical images with generative adversarial networks. *arXiv preprint arXiv:2012.08082*.

Tanimura, H., Zaroubi, S., and Aghanim, N. (2021). Direct detection of the kinetic sunyaev-zel'dovich effect in galaxy clusters. *Astronomy & Astrophysics*, 645:A112.

Tassev, S., Zaldarriaga, M., and Eisenstein, D. J. (2013). Solving large scale structure in ten easy steps with cola. *Journal of Cosmology and Astroparticle Physics*, 2013(06):036.

Tegmark, M., de Oliveira-Costa, A., and Hamilton, A. J. (2003). High resolution foreground cleaned cmb map from wmap. *Physical Review D*, 68(12):123523.

Thanh-Tung, H. and Tran, T. (2018). On catastrophic forgetting and mode collapse in generative adversarial networks. *arXiv*, pages arXiv–1807.

Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. (2014). Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Tröster, T., Ferguson, C., Harnois-Déraps, J., and McCarthy, I. G. (2019). Painting with baryons: augmenting n-body simulations with gas using deep generative models. *Monthly Notices of the Royal Astronomical Society: Letters*, 487(1):L24–L29.

Ullmo, M., Decelle, A., and Aghanim, N. (2021). Encoding large-scale cosmological structure with generative adversarial networks. *Astronomy & Astrophysics*, 651:A46.

Varela-Salinas, M. J., Burbat, R., et al. (2018). Google translate and deepl: breaking taboos in translator training.

Villaescusa-Navarro, F., Anglés-Alcázar, D., Genel, S., Spergel, D. N., Somerville, R. S., Dave, R., Pillepich, A., Hernquist, L., Nelson, D., Torrey, P., et al. (2020). The camels project: Cosmology and astrophysics with machine learning simulations. *arXiv preprint arXiv:2010.00619*.

Vogelsberger, M., Genel, S., Springel, V., Torrey, P., Sijacki, D., Xu, D., Snyder, G., Nelson, D., and Hernquist, L. (2014). Introducing the illustris project: simulating the coevolution of dark and visible matter in the universe. *Monthly Notices of the Royal Astronomical Society*, 444(2):1518–1547.

Vojtekova, A., Lieu, M., Valtchanov, I., Altieri, B., Old, L., Chen, Q., and Hroch, F. (2021). Learning to denoise astronomical images with u-nets. *Monthly Notices of the Royal Astronomical Society*, 503(3):3204–3215.

Vos, E. E., Luus, P. F., Finlay, C. J., and Bassett, B. A. (2019). A generative machine learning approach to rfi mitigation for radio astronomy. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

Vougioukas, K., Petridis, S., and Pantic, M. (2019). Realistic speech-driven facial animation with gans. *International Journal of Computer Vision*, pages 1–16.

Wang, F., Casalino, L. P., and Khullar, D. (2019). Deep learning in medicine—promise, progress, and challenges. *JAMA internal medicine*, 179(3):293–294.

Yalçın, Ö. N., Abukhodair, N., and DiPaola, S. (2020). Empathic ai painter: A computational creativity system with embodied conversational interaction. In *NeurIPS 2019 Competition and Demonstration Track*, pages 131–141. PMLR.

Yoon, J., Jordon, J., and Schaar, M. (2018). Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pages 5689–5698. PMLR.

York, D. G., Adelman, J., Anderson Jr, J. E., Anderson, S. F., Annis, J., Bahcall, N. A., Bakken, J., Barkhouser, R., Bastian, S., Berman, E., et al. (2000). The sloan digital sky survey: Technical summary. *The Astronomical Journal*, 120(3):1579.

Yousefi-Azar, M. and Hamey, L. (2017). Text summarization using unsupervised deep learning. *Expert Systems with Applications*, 68:93–105.

Zakharov, E., Shysheya, A., Burkov, E., and Lempitsky, V. (2019). Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9459–9468.

Zamudio-Fernandez, J., Okan, A., Villaescusa-Navarro, F., Bilaloglu, S., Cengiz, A. D., He, S., Levasseur, L. P., and Ho, S. (2019). Higan: Cosmic neutral hydrogen with generative adversarial networks. *arXiv preprint arXiv:1904.12846*.

Zel'Dovich, Y. B. (1970). Gravitational instability: An approximate theory for large density perturbations. *Astronomy and astrophysics*, 5:84–89.

Zhang, W., Xu, L., Li, Z., Lu, Q., and Liu, Y. (2016). A deep-intelligence framework for online video processing. *IEEE Software*, 33(2):44–51.

Zhao, H., Gao, J., Lan, T., Sun, C., Sapp, B., Varadarajan, B., Shen, Y., Shen, Y., Chai, Y., Schmid, C., et al. (2020). Tnt: Target-driven trajectory prediction. *arXiv preprint arXiv:2008.08294*.

Zhao, Z.-Q., Zheng, P., Xu, S.-t., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232.