Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia "A. Righi"
Corso di Laurea Magistrale in Astrofisica e Cosmologia

Evaluation of galaxy clusters large scale parameters: a weak lensing analysis with Deep Learning based methods

Relatore:
Prof. Lauro Moscardini

Presentata da:
Michele Fogliardi

Correlatori:
Dott. Massimo Meneghetti
Dott.ssa Laura Leuzzi

Anno Accademico 2022/2023

Abstract

Galaxy clusters stand out as the most massive gravitationally-bound structures within the Universe. Their characteristics have always been a focal point of cosmological research, serving as significant evidence to deepen our comprehension of the universe's evolution. Therefore, missions like Euclid, which recent estimates suggest it will be able to survey approximately 13.245 $\deg^2$ of sky over the next six years, are focused on observing them and to determine their characteristics. Given the large volume of data that future missions like this will produce, the development of automated and reliable techniques for the examination of huge datasets is of crucial importance. Convolutional Neural Networks (CNNs) are a Deep Learning technique that has proven particularly effective in the past years as a powerful tool for in this context, because of their speed of execution and capacity of generalization. In particular, we readapted the architectures of three different state of the art CNNs, namely VGG-Net, Inception-v4 and Inception-ResNet-v2, and we implemented them in Pytorch. For each architecture we created various models that differ in size, regularization and optimization techniques. Our models are trained on labeled reduced shear input maps for clusters at $z = 0.25$ created with the MOKA software, which generates semi-analytical mass distributions of galaxy clusters and computes the relevant lensing quantities. The models are then further tested on labeless maps in order to predict the cluster virial mass, the concentration, the number of substructures and the mass fraction in substructures.
The overall performances of the different networks in measuring the cluster parameters is good, and we find the best model to be the one characterized by the simplest architecture, VGG-Net. By adding noise to the maps, we find that the networks ability to predict the correct values gets worse, but still produces a good estimate of the cluster parameters.
Determining these parameters holds significant importance due to their potential applications in various cosmological tests. For instance, they serve as valuable tools for deducing cosmological parameters values from the amplitude and redshift evolution of the cluster mass function or the mass-concentration relation.

Abstract

Gli ammassi di galassie costituiscono le strutture gravitazionalmente legate più massive dell'Universo. Le loro proprietà hanno sempre costituito un punto chiave della ricerca cosmologica, in quanto rappresentano prove significative per l'approfondimento della nostra conoscenza sull'evoluzione dell'Universo. Dunque missioni come Euclid, che potrà osservare fino a 13.245 $\deg^2$ di cielo nei prossimi sei anni, hanno tra i loro obiettivi l'osservazione degli ammssi di galassie e la determinazione delle loro caratteristiche principali. Per via del grande volume di dati che missioni di questo tipo produrranno, lo sviluppo di tecniche automatizzate ed affidabili per l'analisi di grandi dataset è di cruciale importanza. Le Reti Neurali Convoluzionali (CNNs) sono una tecnica di Deep Learning che si è dimostrata particolarmente efficace negli ultimi anni come un potente strumento di indagine in questo contesto, per via della loro velocità di esecuzione e capacità di generalizzazione. In particolare, in questo lavoro di Tesi abbiamo riadattato le architetture di tre differenti reti neurali all'avanguardia, nello specifico VGG-Net, Inception-v4 e Inception-ResNet-v2, implementandole in Pytorch. Per ogni architettura abbiamo creato vari modelli che differiscono per dimensioni, tecniche di regolarizzazione e ottimizzazione. I nostri modelli vengono allenati ricevendo come input mappe di shear ridotto di ammassi localizzati a redshift $z = 0.25$, create con il software MOKA, capace di generare semi-analiticamente distribuzioni di massa di ammassi di galassie realistiche e di ricavarne le principali quantità legate al lensing. Questi modelli vengono poi testati su mappe che non contengono le informazioni riguardo i parametri reali dell'ammasso, con lo scopo di predirne i parametri di grande scala, quali la massa viriale, la concentrazione dell'alone, il numero di sottostrutture e la frazione di massa contenuta nelle sottostrutture. In generale, le prestazioni delle differenti reti nel misurare i parametri sono ottime, e il nostro miglior modello è quello caratterizzato dall'architettura più semplice, ovvero VGG-Net. Aggiungendo un rumore che simula quello delle future osservazioni di Euclid alle mappe di shear, troviamo che la capacità della rete nel predire i valori corretti peggiora, ottenendo comunque una buona stima dei parametri dei cluster.
Determinare i parametri di grande scala degli ammassi è di fondamentale importanza per via delle potenziali implicazioni che questi hanno in vari test cosmologici. Per esempio, costituiscono strumenti molto utili per caratterizzare la relazione massa-concentrazione degli ammassi, o per caratterizzare l'evoluzione con il redshift della funzione di massa degli ammassi.

Contents

Introduction

Gravitational lensing effects are a powerful tool to investigate the distribution of matter in galaxy clusters, since the observations of this phenomenon can be used either to study the mass distribution of cosmic objects dominated by dark matter, and to test models of cosmic structure formation (Blandford & Narayan, 1992).
In evidence, the weak gravitational lensing effect is responsible for a shape distortion (commonly referred to as shear) and a magnification of the images of background sources, due to the gravitational field of some interposed massive objects or large scale structures (Bartelmann & Schneider, 2001). Weak shear lensing by galaxy clusters gives rise to levels of up to a few 10 percent of elliptical distortions in images of background sources. Thus, the weak shear lensing signal, as measured from small but coherent image distortions in galaxy shapes, can provide a direct measure of the projected mass distribution of galaxy clusters (Kaiser & Squires, 1993). Moreover, lensing magnification can influence the observed surface number density of background galaxies seen behind clusters and enhance their apparent fluxes, expanding the area of the observed corresponding sky (Broadhurst et al., 1995).
The resulting mass models from weak and strong lensing can be used in many different applications (Kneib & Natarajan, 2011; Meneghetti et al., 2013; Treu & Marshall, 2016; Umetsu, 2020), aiming at understanding the processes that shape the growth and the evolution of the cosmic structures.
Upcoming wide-field imaging surveys from space, such as the one actually carried out by the Euclid mission (Laureijs et al., 2011), will potentially increase the number of observed clusters by many thousands. In this scenario, Deep Learning (DL) techniques emerge as an efficient alternative to more classical time consuming approaches, automating the process of feature extraction and ensuring a rapid analysis of vast volumes of images. Indeed, they are expected to assume a pivotal role in the evolution of astronomical data analysis methodologies. Specifically, Convolutional Neural Networks (CNNs) are able to autonomously learn the most effective features for the classification of images directly from the training dataset.
In this work, we test the ability of different CNNs to address the problem of measuring the clusters large scale parameters, training our models on reduced shear maps produced with the MOKA software (Giocoli et al., 2012a), which generates semi-analytical

mass distributions of galaxy clusters and computes the relevant lensing quantities. In particular, we implement different models starting from the VGG-Net architecture (Simonyan & Zisserman, 2015), the Inception-v4 architecture (Szegedy et al., 2014, 2015, 2016) and Inception-ResNet-v2 (He et al., 2015; Szegedy et al., 2016). In recent years, these models have been effectively utilized in image classification tasks, and have become a benchmark for the scientific community.
For each architecture, we build several models differing in size and regularization or optimization techniques, then we train each model on 75.000 labeled reduced shear maps in order to teach to our algorithms how to correctly predict the true cluster large scale parameters values. The parameters here considered are the virial mass, the concentration of the halo (accounting also for its smooth component), the number of substructures within the halo and the fraction of mass contained in the substructures with respect to the total mass. To simulate more realistic observations, we also train our best model using reduced shear maps which include shape noise for a given number density of lensed galaxies.Then, we compare the performances achieved by our models, and describe their different properties.
Our work is organized as it follows:

1. In Chapt. 1, we introduce the fundamentals of Cosmology, focusing on the physical laws that describe the evolution of the Universe.

2. In Chapt. 2, we introduce the Gravitational Lensing theory. In detail, in Sect. 2.1 we present the fundamental equations and notions used to characterize the Gravitational Lensing effects. In Sect. 2.2 we describe the specific phenomenology of Gravitational Lensing in galaxy clusters.

3. In Chapt.3 we introduce the basics of Machine Learning theory. We start by describing the fundamental principles of Deep Learning in Sect. 3.1 and then we focus on the description of the structure (Sect. 3.2) and the training process (Sect. 3.3) of Neural Networks. Then, we provide a detailed description of the applications and functioning of Convolutional Neural Networks (Sect. 3.4), describing in depth the architectures behind the models we implemented, presenting them in Sect. 3.4.1 and Sect. 3.4.2.

4. In Chapt. 4, we describe the dataset employed for our networks and the detailed structure of all the models we implemented. In particular, in Sect. 4.1 we present the assumptions and the properties of the software from which we obtain the lensing maps, MOKA, along with the general properties of our training, validation and test sets. In Sect. 4.2 we provide all the details regarding our models structures and purposes. Finally, in Sect. 4.3 we describe the hyperparameters used for the training of our networks, along with the optimizers employed and our definition of accuracy.

5. In Chapt. 5 we describe the all the results obtained with our models. In particular, we first characterize the metrics employed for the evaluation of the performances of our models in Sect. 5.1, then we present our VGG-Net architecture test results in Sect. 5.2. Finally, we describe the remaining tests on the Inception models in Sect. 5.3.

6. In Chapt. 6, we summarize the most important results and discuss the future perspectives and extensions of our work.

Chapter 1

Cosmology

1.1 Fundamentals of cosmology

Cosmology is the branch of Astrophysics that studies the formation and evolution of the Universe on large scales. Its purpose is developing a model capable of describing the features we observe today. Our actual model is the Standard Model of Cosmology, which is based on two fundamental assumptions:

- *the cosmological principle*, which asserts that on sufficiently large scales (today of the order of hundreds of Mpc), the Universe can be considered both homogeneous and isotropic. Homogeneity is the property of being identical everywhere in space, while isotropy is the property of looking the same in every direction. The main observational evidences of this principle, which cannot be proven, come from the Cosmic Microvawe Background (CMB) anisotropies measurements by the Planck collaboration (Planck Collaboration et al., 2016), and the distribution of galaxies on scales of hundreds of Mpc.

- *gravity*, which is the most relevant interaction involving large scale structures *is described by the General Relativity Theory.* We assume then the ubiquitous validity of Einstein's General Relativity Theory, which states that the Universe's geometrical properties are determined by its content of matter-energy.

In order to develop a cosmological model it is first necessary to introduce a metric. The metric represents the interval between two events in the space-time, and has to incorporate the cosmological principle. The general expression for this quantity writes as:

$$ds^2 = g_{ij} dx^i dx^j, \tag{1.1}$$

where i, j vary from 0 to 3 (where index 0 refers to the time coordinate, while the remaining indexes indicate the spatial coordinates) and g_{ij} is the metric tensor describing

space-time geometry. By assuming the Cosmological Principle, Eq. 1.1 rewrites as the Robertson-Walker metric:

$$ds^2 = (cdt)^2 - a(t)^2 \left[\frac{dr^2}{1 - Kr^2} + r^2(d\theta^2 + \sin\theta^2 d\phi^2) \right], \tag{1.2}$$

where we adopted the comoving (time dependent) spherical polar coordinates r, θ, ϕ. Here c is the speed of light; t is the proper time; $a(t)$ is the *scale factor*, a function to be determined which has the dimensions of a length and has time dependence; K is the *curvature parameter*, a constant parameter whose values can only be $1, 0, -1$, related to the Universe's assumed geometry. Specifically:

- if $K = 0$, the Universe is flat, and described by Euclidean geometry
- if $K = +1$, the Universe is closed, and described by hyperspherical geometry
- if $K = -1$, the Universe is open, and described by hyperbolic geometry

1.1.1 Cosmological distances and Universe expansion

Starting from the Robertson-Walker metric, the *proper distance* between a point P and another point P_0, which we can assume to be the origin of a polar coordinate system r, θ, ϕ, can be defined. For simplicity, we can assume that the observer is on the origin of the coordinate system P_0, oriented in a way that $d\theta = d\phi = 0$. The proper distance between P and P_0 measured by an observer at the cosmic time t is then

$$d_p = \int_0^r \frac{a dr'}{(1 - Kr'^2)^{1/2}} = a(t) f(r), \tag{1.3}$$

where $f(r)$ depends on the curvature parameter and is given by:

$$f(r) = \begin{cases} \sin r^{-1} & \text{for} \quad K = +1 \\ r & \text{for} \quad K = 0 \\ \sinh r^{-1} & \text{for} \quad K = -1 \end{cases} . \tag{1.4}$$

The proper distance calculated at the present time t_0 is called *comoving distance*, and is defined as

$$d_C = d_P(t_0) = a(t_0) f(r) = \frac{a(t_0)}{a(t)} d_P(t). \tag{1.5}$$

Hubble-Lemaitre law The proper distance of a source in P changes with time because of the time-dependence of the scale factor a. Then, P is described by a radial velocity with respect to the origin P_0, given by the *Hubble-Lemaitre law* (Hubble, 1929):

$$v_r = \dot{a} f(r) = \frac{\dot{a}}{a} d_P. \tag{1.6}$$

The Hubble-Lemaitre law implies that two points depart from each other with a relative velocity proportional to their proper distance. Then, distant objects recede from us more rapidly than closer objects. In cosmology, we define

$$H(t) = \frac{\dot{a}(t)}{a(t)}, \tag{1.7}$$

as the *Hubble parameter*, a fundamental quantity having the dimension of inverse time. The Hubble parameter characterizes the Universe expansion and its inverse value gives an estimate of the age of the Universe.

Redshift Eq. 1.3 implies that the spectra of distant and luminous objects are subjected to a shift in the radiation wavelength which we refer to as *redshift*. In particular, photons emitted at wavelength λ_e at time t will be observed with wavelength λ_0 at time t_0. This difference is described by

$$z = \frac{\lambda_0 - \lambda_e}{\lambda_e}. \tag{1.8}$$

It can be demonstrated that

$$1 + z = \frac{a_0}{a}, \tag{1.9}$$

where $a_0 = a(t_0)$. Since the observed values of z are positive, we can assume an expanding Universe, with $\dot{a} > 0$.

The definitions of proper and comoving distances given in Eq. 1.3 and Eq. 1.5 are not directly measurable. Nonetheless, we can define other directly measurable (at least in principle) distances.
The *luminosity distance* is defined as

$$d_L = \left(\frac{L}{4\pi F} \right)^{1/2}, \tag{1.10}$$

where L is the luminosity of the source at distance r, emitting light at time t, while F is the flux measured by the observer at time t_0. However, to estimate the actual value of this quantity we need to take into consideration several factors: the expansion of the Universe redshifts the original radiation by a factor a/a_0, then the time-dilation effect

(from General Relativity) introduces a shift of the same factor. Finally, the area of a spherical surface centred on the source and passing through the observer at time t_0 is $4\pi a_0^2 r^2$. Hence, we calculate the resulting flux F as

$$F = \frac{L}{4\pi a_0^2 r^2}\left(\frac{a}{a_0}\right)^2 = \frac{L}{4\pi a_0^2 r^2}(1+z)^{-2}, \tag{1.11}$$

from which

$$d_L = a_0 r(1+z). \tag{1.12}$$

Another measurable distance, particularly important for weak lensing measurements, is the *angular diameter* distance. This quantity is defined as

$$d_A = \frac{D_p}{\Delta\theta} = a(t)r = \frac{d_L}{(1+z)^2}, \tag{1.13}$$

where D_p is the the proper diameter of a source placed at distance r at time t, and $\Delta\theta$ is the angle subtended by D_p. This distance aims to preserve a geometrical property of the Euclidean space, namely the variation of the angular size of an object with its distance from the observer.

1.1.2 Friedmann equations

The foundation of every modern cosmology model begins with the analysis of two fundamental equations, derived from the General Relativity Theory (Einstein, 1916). As Einstein demonstrated, the geometry of space-time is related to its energy-matter content by the *Einstein equation*

$$R_{ij} - \frac{1}{2}g_{ij}R = \frac{8\pi G}{c^4}T_{ij}, \tag{1.14}$$

where R_{ij} is the Ricci tensor and R is the Ricci scalar, g_{ij} is related to the metric and T_{ij} is the energy-momentum tensor. By assuming the Robertson-Walker metric (which incarnates the Cosmological Principle) and the energy-momentum tensor of a perfect fluid with pressure p, density ρ and U_k four-velocity

$$T_{ij} = -pg_{ij} + (p + \rho c^2)U_i U_j, \tag{1.15}$$

we obtain the *Friedmann equations* (Friedman, 1922):

$$\ddot{a} = -\frac{4\pi G}{3}\left(\rho + \frac{3p}{c^2}\right)a, \tag{1.16}$$

$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho - \frac{Kc^2}{a^2}. \tag{1.17}$$

These two equations are not independent: the second one can be recovered from the first one, taking into account the adiabatic expansion of the Universe, expressed by the condition of adiabaticity:

$$d(\rho c^2 a^3) = -p da^3. \tag{1.18}$$

Einstein, convinced of the static nature and eternity of the Universe, reformulated Eq. 1.14 introducing the *cosmological constant* Λ, allowing a static solution for his equation:

$$R_{ij} - \frac{1}{2} g_{ij} R - \Lambda g_{ij} = \frac{8\pi G}{c^4} T_{ij}. \tag{1.19}$$

With the discovery of the expansion of the Universe, Λ was removed from the equations, but was later reintroduced following the observation of the present Universe's accelerated expansion.
If we define an *effective* energy-momentum tensor $\tilde{T}_{ij}$ as

$$\tilde{T}_{ij} = T_{ij} + \frac{\Lambda c^4}{8\pi G} g_{ij} = -\tilde{p} g_{ij} + (\tilde{p} + \tilde{\rho} c^2) U_i U_j, \tag{1.20}$$

where

$$\tilde{p} = p + p_\Lambda = p - \frac{\Lambda c^4}{8\pi G}, \quad \tilde{\rho} = \rho + \rho_\Lambda = \rho + \frac{\Lambda c^2}{8\pi G}, \tag{1.21}$$

we can now rewrite Eq. 1.16 and Eq. 1.17 as

$$\ddot{a} = -\frac{4\pi G}{3} \left(\tilde{\rho} + \frac{3\tilde{p}}{c^2} \right) a, \tag{1.22}$$

$$\left(\frac{\dot{a}}{a} \right)^2 = \frac{8\pi G}{3} \tilde{\rho} - \frac{K c^2}{a^2}. \tag{1.23}$$

Eq. 1.23 can be rewritten in terms of the Hubble parameter. At $t = t_0$ we obtain:

$$H_0^2 \left(1 - \frac{\Lambda c^2}{3 H_0^2} - \frac{8\pi G \rho_0}{3 H_0^2} \right) = -\frac{K c^2}{a_0^2}, \tag{1.24}$$

where $H_0 = H(t_0)$. Eq. 1.24 can be expressed in terms of a *critical density* calculated today $\rho_{c,0}$ and $\rho_{\Lambda,0}$ as

$$H_0^2 \left(1 - \frac{\rho_{\Lambda,0}}{\rho_{c,0}} - \frac{\rho_0}{\rho_{c,0}} \right) = -\frac{K c^2}{a_0^2}, \tag{1.25}$$

where $\rho_{c,0}$ writes

$$\rho_{c,0} = \frac{3 H_0^2}{8\pi G}. \tag{1.26}$$

Introducing now the *density parameter* $\Omega_{i,0} = \frac{\rho_{i,0}}{\rho_{c,0}}$ for the i-th component of the Universe, Eq. 1.25 becomes

$$H_0^2 (1 - \Omega_{\Lambda,0} - \Omega_0) = -\frac{K c^2}{a_0^2}. \tag{1.27}$$

We have then three different possibilities when relating the curvature of the Universe to its energy content, based on the value assigned to K:

- if $K = 0 \rightarrow \Omega_{tot} = \Omega_{\Lambda,0} + \Omega_0 = 1$ (flat Universe)
- if $K = +1 \rightarrow \Omega_{tot} = \Omega_{\Lambda,0} + \Omega_0 > 1$ (closed Universe)
- if $K = -1 \rightarrow \Omega_{tot} = \Omega_{\Lambda,0} + \Omega_0 < 1$ (open Universe)

The main components of the Universe can be divided into three groups: ultrarelativistic matter and radiation; non-relativistic matter (baryonic and Dark Matter); dark energy, of which the cosmological constant is one of the possible forms. Without the introduction of Λ, by observing Eq. 1.16 we can conclude that $\ddot{a} < 0$, since p and ρ are positive quantities. Then, the Universe should be characterized by a decelerating expansion. This deceleration is defined by a *deceleration parameter* expressed as q:

$$q = -\frac{\ddot{a}a}{\dot{a}^2}. \tag{1.28}$$

However, the measurements of the deceleration parameter clearly indicate that today $\ddot{a} > 0$ (Riess et al., 1998).

1.1.3 Equation of state

We now introduce an important equation that defines the relation between pressure and density of the cosmic fluid. This is necessary to find solutions for $P(t)$, $\rho(t)$ and $a(t)$. This equation can be written as

$$P = w\rho c^2, \tag{1.29}$$

where w is called *equation of state parameter* and can take different values depending on the cosmic component under consideration:

- For non-relativistic matter or "dust", $w \approx 0$.
- For radiation and ultra relativistic matter, $w = \frac{1}{3}$.
- For the cosmological constant Λ , $w = -1$.

Inserting Eq. 1.18 in Eq. 1.29, we can calculate an expression for ρ in terms of a or z

$$\rho_w \propto a^{-3(1+w)} \propto (1+z)^{3(1+w)}. \tag{1.30}$$

This means we have three possible trends characterizing the evolution of the density Universe, based on the dominating component:

- For matter dominated Universe: $w = 0 \rightarrow \rho_M = \rho_{M,0} \left(\frac{a_0}{a}\right)^{-3} = \rho_{M,0}(1+z)^3$.
- For radiation dominated Universe: $w = \frac{1}{3} \rightarrow \rho_R = \rho_{R,0} \left(\frac{a_0}{a}\right)^{-4} = \rho_{R,0}(1+z)^4$.
- For Λ dominated Universe: $w = -1 \rightarrow \rho_\Lambda = \rho_{\Lambda,0} \left(\frac{a_0}{a}\right)^{0} = \rho_{\Lambda,0}(1+z)^0 = \rho_{\Lambda,0}$.

As the evolution of energy density changes with cosmic time, it is possible to divide the history of the Universe into three main epochs, according to the dominant component: the radiation-dominated era, the matter-dominated era and the dark energy-dominated era. Fig. 1.1 shows the evolution of densities with cosmic time. If we compare the

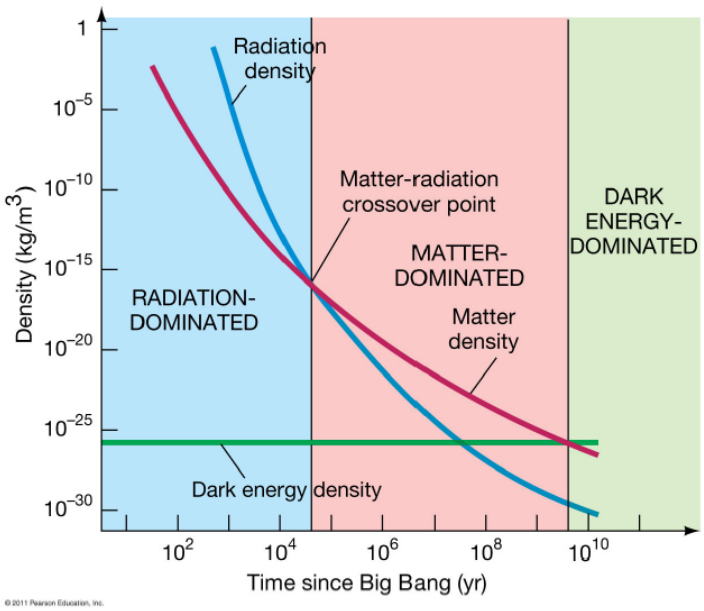


Figure 1.1: Density evolution of cosmic components with cosmic time. The blue, red and green solid lines represent the evolution of radiation, matter and dark energy densities respectively. Before the radiation-matter equivalence, radiation dominated the Universe. After that threshold, matter becomes the dominant component. Nowadays, the matter density has decreased enough to be dominated by the dark energy component. Copyright: Pearson Education, Inc. (2011).

blue, red and green solid lines we can identify two dominant component transition, described as

- radiation-matter equivalence; where $\rho_R = \rho_M$, at $z \sim 3 \cdot 10^4$
- matter-dark energy equivalence; where $\rho_M = \rho_\Lambda$, at $z = 0.67$

As described in Sec. 1.1.2, during the radiation-dominated and the matter-dominated epochs the expansion of the Universe was decelerated, having $\ddot{a} < 0$. Assembling this information with the observational evidence that today $\dot{a} > 0$, we can easily conclude that before the dark energy-dominated era, the scale factor $a(t)$ is identified by a negative concavity, leading to a value of $a(t) = 0$ in the past. The time corresponding to a null scale parameter is called *Big Bang*, and at this instant the density and the Hubble parameter diverge. Moreover, since $a(t)$ is a concave function, the time between the singularity and the epoch t must always be less than the characteristic expansion time of the Universe $\tau_H = \frac{1}{H} = \frac{a}{\dot{a}}$. Then, the current age of the Universe t_0 has to be less than *Hubble time*, defined as $\frac{1}{H_0}$. This concept is shown in Fig. 1.2.

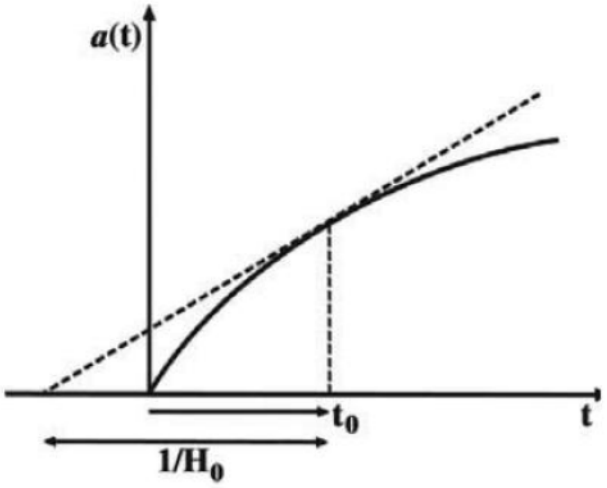


Figure 1.2: Evolution with time of the scale factor $a(t)$. The curve has negative concavity, leading to the Big Bang at $t = 0$. The evidence of $\dot{a} > 0$ also implies that the age of the Universe t_0 is less than the Hubble time $\frac{1}{H_0}$ (Coles & Lucchin, 2003).

1.1.4 Single component models

The Friedmann equations can be solved by assuming a single component Universe, both for flat or curved (open and closed) models. In particular, it is possible to modify Eq. 1.16 to obtain

$$\left(\frac{\dot{a}^2}{a_0^2}\right) = H_0^2 \left[1 - \Omega_0 + \Omega_0 \left(\frac{a_0}{a}\right)^{3(1+w)}\right]. \tag{1.31}$$

For the flat model, usually referred to as *Einstein-De Sitter model*, $\Omega_0 = 1$ ($K = 0$). Thus, by substituting this value in Eq. 1.31, we can obtain explicit solutions for

$a(t), H(t), q(t), \rho(t)$:

$$a(t) = a_0 \left(\frac{t}{t_0}\right)^{\frac{2}{3(1+w)}}, \tag{1.32}$$

$$H(t) = \frac{\dot{a}}{a} = \frac{2}{3(1+w)t}, \tag{1.33}$$

$$q(t) = -\frac{\ddot{a}a}{\ddot{a}} = \frac{1+3w}{2}, \tag{1.34}$$

$$\rho(t) = \frac{1}{6\pi G t^2}, \tag{1.35}$$

$$t_0 = \frac{2}{3H_0(1+w)}. \tag{1.36}$$

As Eq. 1.32 points out, a flat Universe undergoes an indefinite expansion independently from the value assumed by w (i.e., its main component).
For curved models, solutions of Eq. 1.31 are not as straightforward as in the flat scenario. However, it can be demonstrated that in its early phase a curved Universe behaviours like a flat one, regardless of the value assumed by K. As the scale factor $a(t)$ increases, the difference between these models becomes not negligible and we identify two possible scenarios.

- If $\Omega_0 < 1$ (open models), we find from Eq. 1.31 that $\dot{a} > 0$ at any time, therefore leading to a monotonically growing $a(t)$. In this model, the Universe undergoes an infinite expansion with $a \propto t$, a constant Hubble parameter ($H \propto \frac{1}{t}$) and $q = 0$.
- If $\Omega_0 > 1$ (closed models), the solution of Eq. 1.31 depends on the competition between the two terms in the squared parenthesis. Since the last term is inversely proportional to the scale parameter, we can identify an instant of the Universe history in which $(1 - \Omega_0) < 0$. This implies that at some point $\dot{a} = 0$, and the curve of growth of $a(t)$ will stop at a maximum value called a_{max}. After reaching this point, the Universe will undergo a process of re-collapse, leading to another singularity, this time called *Big Crunch*.

Fig. 1.3 shows the evolution of $a(t)$ for all the different one-component models.

1.1.5 Measurements of the cosmological parameters

We now give a brief review of the most accurate estimates of the main cosmological parameters, which will be adopted in this work. The most important and reliable research in this field is the one carried out by the *Planck* collaboration (Planck Collaboration

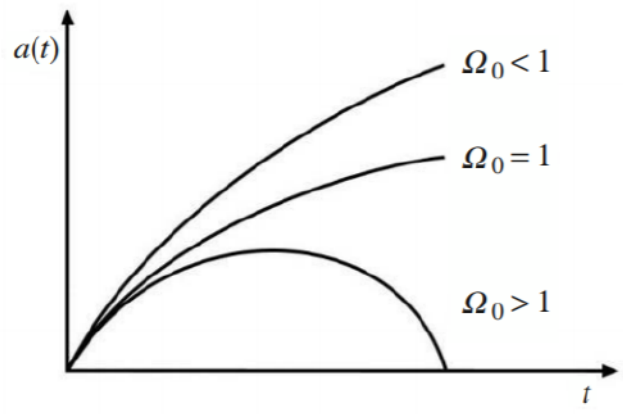


Figure 1.3: Evolution with time of the scale factor $a(t)$ for an open model ($\Omega_0 < 1$), a flat or Einstein-de Sitter model ($\Omega_0 = 1$) and a closed model ($\Omega_0 > 1$), (Coles & Lucchin, 2003).

$H_0[km/s\,Mpc]$	$\Omega_{M,0}$	$\Omega_{b,0}$	$\Omega_{\Lambda,0}$
67.4 ± 0.5	0.315 ± 0.007	0.0224 ± 0.0001	0.6847 ± 0.0073

Table 1.1: Results for the main cosmological parameters published in Planck Collaboration et al. (2020b).

et al., 2020a,b), based on the analysis of the anisotropies in the CMB spectrum. Tab. 1.1 summarize the estimated values found by Planck mission for the Hubble constant H_0, the matter density parameter $\Omega_{M,0}$, the barion density $\Omega_{b,0}$ and the dark energy density parameter $\Omega_{\Lambda,0}$. The Hubble constant is though subjected to a so-called "tension", as another local guess of H_0 obtained by measuring the distances to relatively nearby galaxies and using their recession velocities to determine the Hubble constant yields a different value (Riess et al., 2018). In particular, the Hubble constant value measured by Riess et al. (2018) is $74 \pm 0.5\,km/s\,Mpc$.
It is then conventional to indicate $H_0 = 100\,h \cdot km/s\,Mpc$, where h is the *dimensionless Hubble constant* that takes into consideration uncertainties related to the value of H_0. Then we can estimate the present-day value for the critical density as

$$\rho_{c,0} = \frac{3H_0^2}{8\pi G} \sim 1.9 \cdot 10^{-29}\,h^2 g\,cm^{-3}. \tag{1.37}$$

Our currently believed cosmological model is the one with the lowest number of free parameters in agreement with the theory of General Relativity: this is the Λ-CDM model, according to which the Universe is flat and is composed of Dark Matter and Λ.

Chapter 2

Gravitational Lensing

Gravitational lensing is a consequence of Einstein's theory of General Relativity and has revolutionized our understanding of the universe's structure and dynamics. This phenomenon occurs when the gravitational field of a massive object, such as a galaxy or a galaxy cluster, bends and distorts the paths of photons passing nearby. As a result, distant objects appear magnified, distorted, or even multiply imaged, offering a unique window into the distribution of matter and the geometry of spacetime on cosmic scales. The massive entity responsible for this deflection is termed the "lens," which can take the form of a point-like object like a star, or an extended one such as a galaxy or galaxy cluster. Conversely, the luminous entity whose light undergoes deflection is referred to as the "source," which may include background galaxies or distant quasars.

2.1 Introduction to gravitational lensing theory

2.1.1 Deflection angle

As outlined in Sect. 1, the Einstein equation (Eq. 1.14) implies that the geometric characteristics of spacetime depends on its energy-matter content: gravitational forces caused by masses positioned between the light source and the observer bend the paths of light. Moreover, the Universe is assumed to be homogeneous and isotropic on large scales, and well described by the Robertson-Walker metric (Eq. 1.2). Conversely, on smaller scales, the presence of structures like galaxies or clusters introduces inhomogeneities that can perturb the space-time metric and alter the trajectories of light rays, thereby giving rise to the lensing phenomena. In order to study the light deflection in proximity of the lens, we first adopt the *"weak field approximation"*, assuming that the light deflection occurs in a region small enough that the expansion of the Universe is negligible. Then, we define the local Newtonian gravitational potential of the lens as

$$\Phi = -\frac{Gm}{r}.$$

To compute the *deflection angle* of the light path, we operate under the assumption that the lens is weak, a condition that holds true in nearly all astrophysical scenarios, corresponding to the request that $\frac{\Phi}{c^2} \ll 1$.
Under this approximation the flat Minkowski metric, that describes unperturbed space-time, is modified only by a small perturbation, and the line element writes:

$$ds^2 = \left(1 + \frac{2\Phi}{dt^2}\right) c^2 dt^2 - \left(1 - \frac{2\Phi}{dt^2}\right) (d\vec{x})^2. \tag{2.1}$$

Since photons travel on null geodetics, corresponding to $ds = 0$, Eq. 2.1 gives us the light speed in the gravitational field as

$$c' = \frac{d\vec{x}}{dt} \sim c\left(1 + \frac{2\Phi}{c^2}\right). \tag{2.2}$$

As $\Phi \leq 0$, we have $c' \leq c$. Hence, we can describe the space-time as a medium with an effective refraction index n given from Eq. 2.2 as

$$n = \frac{c}{c'} \sim 1 - \frac{2\Phi}{c^2} \geq 1. \tag{2.3}$$

Assuming the Fermat's principle, if the spatial scales considered are smaller than the distances between source, lens and observer, it can be shown (Schneider et al., 1992) that the deflection angle is equivalent to

$$\hat{\vec{\alpha}}(b) = \frac{2}{c^2} \int_{-\infty}^{+\infty} \nabla_{\perp} \Phi dz, \tag{2.4}$$

where the light ray follows the initial direction given by $\vec{e}_z$ and passes close to the lens when $z = 0$ with an impact parameter b. In case the lens is a point mass with mass M, Eq. 2.4 becomes

$$|\hat{\vec{\alpha}}|(b) = \frac{4GM}{c^2 b}. \tag{2.5}$$

Since the deflection angle exhibits a linear dependence on the mass M, the deflection angle of a group of lenses can be derived by summing the contributions from each individual lens, following the superposition principle. Supposing we have a group of N point masses of mass M_i randomly distributed on a plane, the superposed deflection angle is described by

$$\hat{\vec{\alpha}}(\vec{\xi}) = \sum_i \hat{\vec{\alpha}}_i \left(\vec{\xi} - \vec{\xi}_i\right) = \frac{4G}{c^2} \sum_i M_i \frac{\vec{\xi} - \vec{\xi}_i}{|\vec{\xi} - \vec{\xi}_i|^2}, \tag{2.6}$$

where $\vec{\xi}_i$ are the positions of the lenses, while $\vec{\xi}$ corresponds to the position in which the light ray crosses the lens plane.

In all general astrophysics scenarios, we are justified to apply the *thin screen approximation*, since the typical distances of observer, lens and source are much larger than the physical size of the lens. By applying this approximation we assume that the lens matter distribution is described by its surface mass density

$$\Sigma(\vec{\xi}) = \frac{4G}{c^2} \int_{-\infty}^{+\infty} \rho(\vec{\xi}, z) dz, \tag{2.7}$$

where ρ is the three-dimensional density of the lens, $\vec{\xi}$ defines the position on the lens plane and z is the direction of the line of sight. In this approximation, following the superposition principle, the deflection angle for an extended lens can be calculated as

$$\hat{\vec{\alpha}}(\vec{\xi}) = \frac{4G}{c^2} \int \frac{(\vec{\xi} - \vec{\xi'})\Sigma(\vec{\xi'})}{|\vec{\xi} - \vec{\xi'}|^2} d^2\xi'. \tag{2.8}$$

Fig. 2.1 illustrates the geometrical configuration of a typical gravitational lensing system. The thin screen approximation makes the extension of the lens along the line of sight neglectable, hence we can consider the light deflection to occur on the *lens plane*, at distance D_L. At the same time, we can assume that photons coming from the source originate from the same distance D_S, meaning that the source lies on the *source plane*.

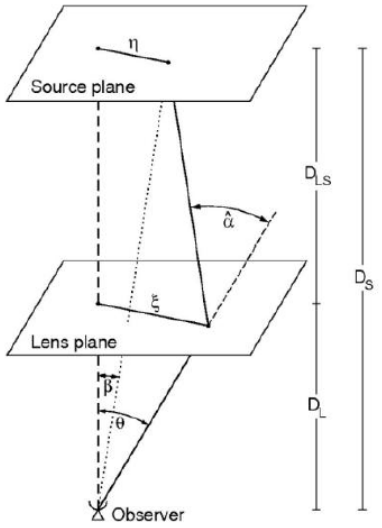


Figure 2.1: Example of a typical gravitational lensing system (Bartelmann & Schneider, 2001).

2.1.2 Lens equation

Gravitational lensing effects depend on the relative positions and distances between the observer, the lens and the source. As shown in Fig. 2.1, an object positioned in $\vec{\beta}$ on the source plane will be observed by the observer in $\vec{\theta}$, with a corresponding deflection angle $\hat{\vec{\alpha}}$. Assuming $\vec{\beta}$, $\vec{\theta}$ and $\hat{\vec{\alpha}}$ are small, the true and apparent positions of the source are related by the *lens equation*

$$\vec{\theta} D_S = \vec{\beta} D_S + \hat{\vec{\alpha}} D_{LS}, \tag{2.9}$$

where D_{LS} refers to the distance between the lens and the source.
By introducing the *reduced deflection angle* as

$$\vec{\alpha}(\vec{\theta}) = \frac{D_{LS}}{D_S} \hat{\vec{\alpha}}, \tag{2.10}$$

we can rewrite Eq. 2.9 as

$$\vec{\beta} = \vec{\theta} - \vec{\alpha}(\vec{\theta}). \tag{2.11}$$

Eq. 2.11 is usually defined in a dimensionless form, considering a length scale ξ_0 on the lens plane and the corresponding length scale $\eta_0 = \xi_0 \frac{D_S}{D_L}$ on the source plane. We can then define two vectors

$$\vec{x} \equiv \frac{\vec{\xi}}{\vec{\xi_0}}, \quad \vec{y} \equiv \frac{\vec{\eta}}{\vec{\eta_0}},$$

as well as a scaled deflection angle

$$\vec{\alpha}(\vec{x}) = \frac{D_L D_{LS}}{\xi_0 D_S} \hat{\vec{\alpha}}(\xi_0 \vec{x}), \tag{2.12}$$

and Eq. 2.11 can be rewritten as

$$\vec{y} = \vec{x} - \vec{\alpha}(\vec{x}). \tag{2.13}$$

2.1.3 Lensing potential

If we project the three-dimensional Newtonian potential Φ on the lens plane we obtain the *effective lensing potential*, which characterizes extended matter distributions through a scaling factor:

$$\hat{\Psi} = \frac{2}{c^2} \frac{D_{LS}}{D_S D_L} \int \Phi(D_L \vec{\theta}, z) dz. \tag{2.14}$$

The effective lensing potential can be written in a dimensionless form as well:

$$\Psi = \frac{D_L^2}{\xi_0^2} \hat{\Psi}.$$

It can be demonstrated that this quantity satisfies two fundamental properties:

- The gradient of the lensing potential is equal to the reduced deflection angle:

$$\nabla_x \Psi(\vec{x}) = \vec{\alpha}(\vec{x});$$

- The Laplacian of the lensing potential is twice the *convergence*:

$$\Delta_x \Psi(\vec{x}) = 2\kappa(\vec{x}).$$

.

The convergence is defined as a dimensionless surface mass density:

$$\kappa(\vec{x}) = \frac{\Sigma(\vec{x})}{\Sigma_{crit}}, \tag{2.15}$$

where

$$\Sigma_{crit} = \frac{c^2}{4\pi G}\frac{D_S}{D_L D_{LS}}. \tag{2.16}$$

is the *critical surface density*, that characterizes each lensing system and depends on the distances of lens and source (i.e., the redshift).

2.1.4 Magnification and distorsion

Among the distinctive phenomena attributed to gravitational lensing, the most important is the distortion of background source shapes. For example, galaxies frequently manifest as elongated arcs within galaxy clusters. Determining the real shape of these sources traduces into solving the lens equation for all source points. Notably, when the source is smaller than the angular dimension at which the lens physical attributes change, these positions can be locally linearized, and the image distortion is delineated by a *lensing Jacobian matrix*

$$A \equiv \frac{\partial \vec{y}}{\partial \vec{x}} = \left(\delta_{ij} - \frac{\partial \alpha_i(\vec{x})}{\partial x_j}\right) = \left(\delta_{ij} - \frac{\partial^2 \Psi(\vec{x})}{\partial x_i \partial x_j}\right) = \delta_{ij} - \Psi_{ij}, \tag{2.17}$$

where x_i is the i-component of x and δ_{ij} corresponds to the Kronecker delta.
The lensing Jacobian can be considered as consisting of two parts:

$$A \equiv \left(A - \frac{1}{2}\mathrm{Tr}A \cdot I\right) + \left(\frac{1}{2}\mathrm{Tr}A \cdot I\right). \tag{2.18}$$

The first part is responsible for an anisotropic distorsion, and is represented by a matrix called *shear matrix*:

$$\left(A - \frac{1}{2}\mathrm{Tr}A \cdot I\right) = \begin{pmatrix} -\frac{1}{2}(\Psi_{11} - \Psi_{22}) & -\Psi_{12} \\ -\Psi_{12} & \frac{1}{2}(\Psi_{11} - \Psi_{22}) \end{pmatrix} \tag{2.19}$$

This shear matrix is often indicated as

$$\Gamma = \begin{pmatrix} \gamma_1 & \gamma_2 \\ \gamma_2 & -\gamma_1 \end{pmatrix}, \tag{2.20}$$

where γ_1 and γ_2 are called *shear components* and correspond to

$$\begin{aligned} \gamma_1(\vec{x}) &= \frac{1}{2}(\Psi_{11} - \Psi_{22}), \\ \gamma_2(\vec{x}) &= \Psi_{12}. \end{aligned} \tag{2.21}$$

These two components are usually written in the form of a pseudo-vector on the lens plane called *shear*, defined as $\vec{\gamma} = (\gamma_1, \gamma_2)$. The shear matrix eigenvalues are $\pm\sqrt{\gamma_1^2 + \gamma_2^2} = \pm\gamma$. Hence, it is possible to define a coordinate rotation by an angle ϕ such that

$$\begin{pmatrix} \gamma_1 & \gamma_2 \\ \gamma_2 & -\gamma_1 \end{pmatrix} = \gamma \begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix}. \tag{2.22}$$

The second part is instead responsible for an isotropic distorsion and can be simplified as

$$\left(\frac{1}{2}\mathrm{Tr}A \cdot I\right) = (1-\kappa)\delta_{ij}. \tag{2.23}$$

By using both Eq. 2.22 and Eq. 2.23, we can rewrite the Jacobian matrix as

$$A \equiv (1-\kappa)\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \gamma \begin{pmatrix} \cos 2\phi & \sin 2\phi \\ \sin 2\phi & -\cos 2\phi \end{pmatrix}. \tag{2.24}$$

Eq. 2.24 clarifies the role of convergence and shear in distorting images: convergence introduces an isotropic distortion that causes the images to be rescaled by a factor $\frac{1}{1-\kappa}$, uniformly rescaling images in all directions; whereas shear elongates the intrinsic shape of the source along specific directions given by the eigenvectors of A. An example of the distorsion effects caused by shear and convergence on a circular source is shown in Fig. 2.2.
In this example a circular source of radius r is mapped onto an ellipse with semi-axes

$$a = \frac{r}{\lambda_t}, \quad b = \frac{r}{\lambda_r}. \tag{2.25}$$

Here, λ_t and λ_r are respectively the tangential and radial eigenvalues of the Jacobian matrix, defined as

$$\begin{aligned} \lambda_t &= 1 - \kappa - \gamma, \\ \lambda_r &= 1 - \kappa + \gamma. \end{aligned} \tag{2.26}$$

Another important quantity linked to this type of distorsions, particularly useful in weak lensing studies, is a parameter called *ellipticity*, defined as

$$\epsilon = \frac{a-b}{a+b} = \frac{\gamma}{1-\kappa} = g. \tag{2.27}$$

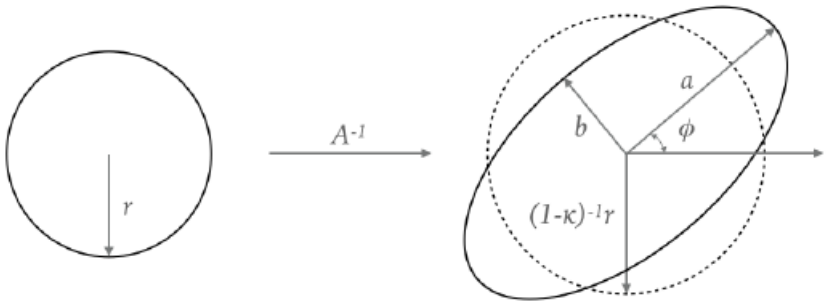


Figure 2.2: Example of the distortion caused by convergence and shear on a circular source (Meneghetti, 2021).

Here, g is named *reduced shear*.
After the distorsion effects, another important consequence of gravitational lensing is the *magnification* of the source's images. As gravitational lensing does not alter neither the quantity and energy (frequency) of photons emitted by the source, the Liouville theorem guarantees the conservation of surface brightness. However, the presence of the lens alters the solid angle under which the source is observed, leading to a potential magnification or demagnification of the source. By using the definition of the Jacobian matrix presented in Eq. 2.17, one can compute the magnification μ by taking the inverse of the determinant of matrix A, corresponding to the *magnification matrix* M:

$$\mu \equiv \det M = \frac{1}{\det A} = \frac{1}{(1-\kappa)^2 - \gamma^2}. \tag{2.28}$$

From the magnification matrix we can obtain the eigenvalues that measure the magnification along the tangential and radial direction with respect to the lens iso-surface density, respectively:

$$\begin{aligned} \mu_t &= \frac{1}{\lambda_t} = \frac{1}{1-\kappa-\gamma}, \\ \mu_r &= \frac{1}{\lambda_r} = \frac{1}{1-\kappa+\gamma}. \end{aligned} \tag{2.29}$$

The lens plane is now characterized by two curves corresponding to $\lambda_t = 0$ and $\lambda_r = 0$, called respectively *tangential* and *radial critical lines*. For instance, this means that if an image forms near the tangential critical line, it will be distorted in that direction, and vice-versa. These curves can be mapped by Eq. 2.11 onto the source plane, becoming the *tangential* and *radial caustic lines*. When sources transit across the caustics, their flux is magnified and the images are tangentially or radially stretched. Furthermore, sources falling inside of a caustic line will be multiply imaged on the lens plane.

2.1.5 Time delay surfaces

In addition to light deflection and image distortion or magnification, another notable aspect of gravitational lensing is the phenomenon of *time delay*. This delay can be separated into two distinct contributions:

- *Geometrical time delay*: this effect is associated to the different path followed by the deflected light rays with respect to the unperturbed ones, and can be defined as:
$$t_{geom} = \frac{1}{2c}\frac{D_L D_S}{D_{LS}}(\vec{x} - \vec{y})^2. \tag{2.30}$$
- *Gravitational time delay*: this delay arises from the different time it takes for light to travel across two regions with distinct refractive indices. Specifically, photons moving through the gravitational field of the lens decelerate compared to unperturbed photons, resulting in an increase of the time delay corresponding to
$$t_{grav} = -\frac{D_L D_S}{D_{LS}}\frac{1}{c^2}\Psi(\vec{x}). \tag{2.31}$$

Therefore, taking into consideration also the expansion of the universe, the total time delay can be obtained by summing the two contributions, with a final expression given by

$$t(\vec{x}) = \frac{(1+z_L)}{c}\frac{D_S \xi_0^2}{D_L D_{LS}}\left[\frac{1}{2}(\vec{x} - \vec{y})^2 - \Psi(\vec{x})\right]. \tag{2.32}$$

As the gradient of the effective lensing potential and the deflection angle are related, the previous expression can be rewritten as

$$(\vec{x} - \vec{y}) - \nabla\Psi(\vec{x}) = \nabla\left[\frac{1}{2}(\vec{x} - \vec{y})^2 - \Psi(\vec{x})\right] = 0. \tag{2.33}$$

Eq. 2.33 implies that solving the lens equation is equivalent to searching the stationary points of the time delay surface described in Eq. 2.32. In particular, images should satisfy the condition $\nabla t(\vec{x}) = 0$.
The time delay surface is characterized by the Hessian matrix, which corresponds to the Jacobian matrix:

$$T = \frac{\partial^2 t(\vec{x})}{\partial x_i \partial x_j} \propto (\delta_{ij} - \Psi_{ij}) = A. \tag{2.34}$$

The Hessian matrix characterizes the curvature of the time delay surface and is inversely proportional to the magnification. As a result, magnification increases in the direction where curvature is smaller. By studying the shape of the time delay surface near the stationary points one can obtain information on the shape of the images. Specifically, it is possible to distinguish between three types of images:

- *Type I images*; when $\det A > 0$, $\mathrm{Tr} A > 0$; the eigenvalues of T are both positive: images form at the minima of $t(\vec{x})$ and have a positive magnification
- *Type II images*; when $\det A < 0$; the eigenvalues of T have opposite signs: images form at the saddle point of $t(\vec{x})$ and have a negative magnification
- *Type III images*; when $\det A > 0$, $\mathrm{Tr} A < 0$; the eigenvalues of T are both negative: images form at the maxima of $t(\vec{x})$ and have a positive magnification

However, it should be noted that having a negative magnification does not traduce into a demagnification (a condition instead related to $|\mu| < 1$), but into a parity change of the image. In particular, the image on the lens plane is flipped compared to the original source image.

2.2 Lensing regimes and galaxy clusters

Indeed, lensing events in galaxy clusters are crucial for studying both the characteristics of the clusters and the distribution of dark matter within them: by observing how light from background galaxies is distorted and bent by the gravitational fields of galaxy clusters, along with the positioning of multiple images of a single background source, researchers can infer valuable information about the mass distribution and gravitational properties of the clusters.
Lensing events fall into two categories, according to the importance of the alignment between the source, the lens and the observer:

- *Strong lensing events* can be observed when the source and the lens are well aligned with the observer (along the line of sight). As we are considering extended sources, strong lensing traduces into having a small angular distance separation between the center of mass of the lens and the source;
- *Weak lensing events* instead are characterized by a larger angular separation between the source and the lens.

The different properties of these two regimes will be briefly described in the following subsections.

2.2.1 Strong lensing

Strong lensing events are characterized by a high magnification, giving us the possibility to observe very faint objects otherwise impossible to detect. Their importance is crucial in describing cluster potential and mass distribution, as the observation of gravitational lensing events with resolved sources allows the reconstruction of the gravitational lenses responsible for the observed distorsions. Since these events are more

frequent in dense environments as the central regions of galaxy clusters, they represent an important tool to study the characteristics of these regions.
Strong lensing effects manifest in close proximity to the lens critical lines, which separate the images of strongly lensed sources. As a result, *gravitational arcs* form when multiple images come toghether across the critical lines, originated by the overlapping of the caustics of an extended source. Utilizing the size of the critical lines is a logical approach to characterize the scale of a strong lensing event. A first simple approach can be given, for an axially symmetric lens, if we assess the magnitude of the tangential critical line using the *Einstein radius*, defined as

$$\theta_E = \sqrt{\frac{4GM(\theta_E)}{c^2}\frac{D_{LS}}{D_S D_L}}, \tag{2.35}$$

where $M(\theta_E)$ is the mass enclosed within the critical line, corresponding to a circular ring, called *Einstein ring*. However, both galaxies and galaxy clusters are not axially symmetric, but well described by elliptical mass distribution. Thus, their critical lines show irregularities and distorsions from the circular shape. An equivalent to the Einstein radius can still be used to quantify the size of a strong lens in irregular scenarios, named *equivalent Einstein radius* and defined as the radius of the circle with the same area enclosed by the lens critical line A_c:

$$\theta_{E,eq.} = \sqrt{\frac{A_c}{\pi}}. \tag{2.36}$$

This quantity represents a very approximate estimate of the cluster mass within the critical line, obtained by assuming spherical simmetry and that the mean surface density within the critical line is the critical surface density. For galaxies having $\sim 10^{11}$-$10^{12} M_\odot$, the Einstein radius is of the order of ~ 1", while for clusters of bigger mass ($\sim 10^{14}$-$10^{15} M_\odot$), the Einstein radius is of the order of $\sim$ 5-50".
To reconstruct the lens mass distribution we can follow two approaches: the *forward* and the *inverse* methods.
The former involves constructing a lens system model that closely mimics the observed images. The initial stage of this process consists in establishing a model for the lens system, followed by comparing the images generated by this model with the observed ones. Consequently, adjustments are made to the model to minimize the difference between the simulated and actual images. An example of forward method application is presented in Newton et al. (2011).
The second method is based on using three different constraints to reconstruct the the lens mass distribution:

- the positions of the multiple images of lensed source, probing the lens's deflection field which is related to the first derivatives of the lensing potential;

- the fluxes (magnification) and the shapes of the multiple images and gravitational arcs, probing the second derivatives of the lensing potential;
- the relative time delays between multiple images, probing the lensing potential.

These observed constraints are used to obtain the lens matter distribution by the application of a *lens inversion algorithm*. In particular, there are two classes of inversion algorithms, namely the *parametric* and *free-form* (*non-parametric*) algorithms.
Assuming light traces the mass, parametric algorithms consists in combining one or more clumps of matter, each characterized by its density profile and shape, in order to explore the model parameter space until the best combination reproducing the observed positions, shapes, magnitudes, and relative time delays of the multiple images and arcs is found.
Alternatively, in the free-form modeling approach, the lens is subdivided into a structured or an unstructured mesh onto which the lensing observables are mapped. The mesh is then transformed into a pixelized mass distribution using the relations between the observables and the lens surface density. Different interesting examples employing this approach are Birrer et al. (2015); Suyu et al. (2006); Blandford et al. (2000).
A more detailed description of the these parametric and free-form algorithms can be found in the work of Meneghetti (2021).
An example of strong lensing effects in a galaxy cluster is shown in Fig. 2.3, where different lensing distorsions are highlighted. Each of them can be characterized with a different approach: the reconstruction study of the mass model of this galaxy cluster is described in Caminha et al. (2017); Bonamigo et al. (2018); Bergamini, P. et al. (2019).

2.2.2 Weak lensing

Weak lensing phenomena manifest when there is a significant angular separation between the source and the lens. However, unlike multiple or heavily distorted images, this effect is observable through subtle distortions in the background sources, primarily faint and distant galaxies.
Assuming that the orientation of distant, faint and irregularly-shaped sources is random, the average shape of a large number of them should be circular. However, as it was explained in Sec. 2.1.4, because of weak lensing, the circular source appears to be elliptical, with axes described by Eq. 2.25 and ellipticity described as in Eq.2.27. In the weak lensing regime, since $\kappa, \gamma \ll 1$, $\epsilon \sim \gamma$.
The ellipticity may also be defined by means of a second-order tensor that describes

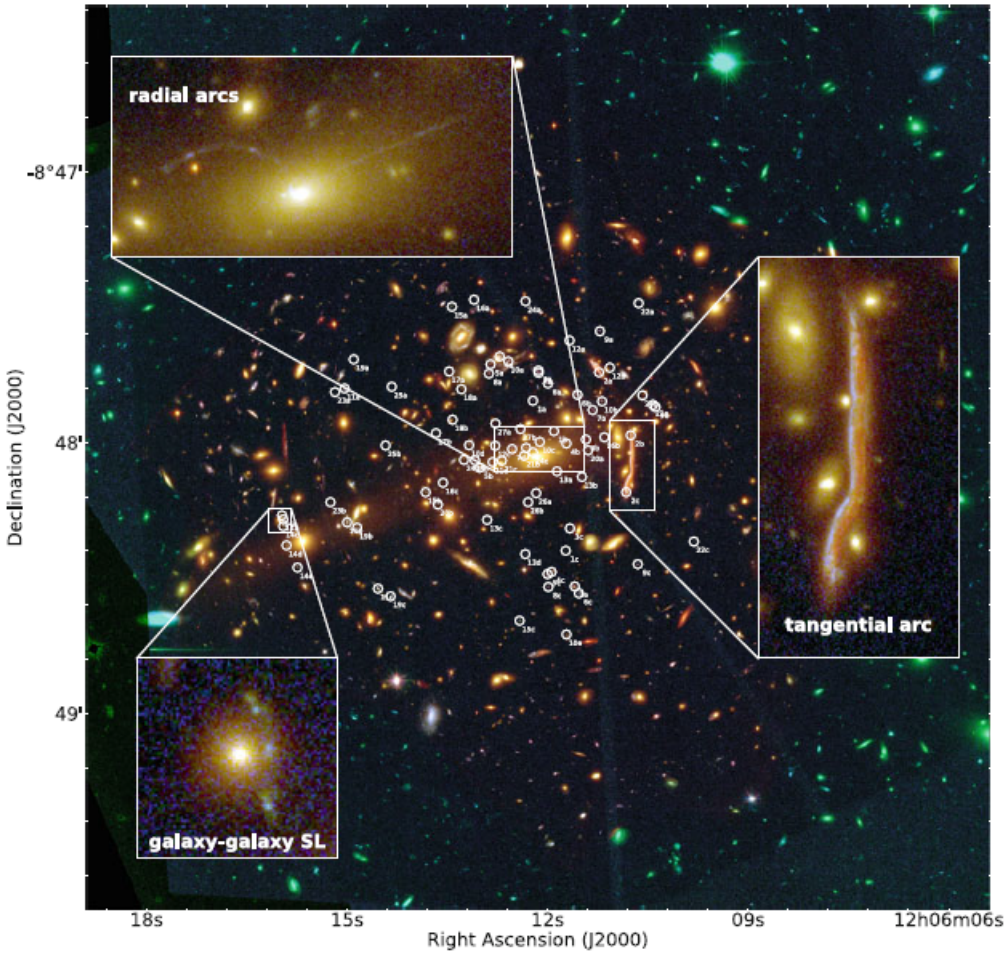


Figure 2.3: Color-enhanced image of the galaxy cluster MACS J1206.2-0847, observed with the Hubble Space Telescope. A variety of strong lensing features can be found in this cluster, namely giant tangential and radial arcs, several families of multiple images of distant sources (marked with circles), and even galaxy–galaxy strong lensing events (as shown in the bottom-left panel). Image from Meneghetti (2021).

the brightness moments on the source and on the lens planes:

$$\begin{aligned} Q_{ij} &= \frac{\int d^2\theta I(\theta) q_I[I(\theta)](\theta_i - \bar{\theta}_i)(\theta_j - \bar{\theta}_j)}{\int d^2\theta I(\theta) q_I[I(\theta)]}, \\ Q^{(s)}_{ij} &= \frac{\int d^2\beta I^{(s)}(\beta) q_I[I^{(s)}(\beta)](\beta - \bar{\beta})(\beta - \bar{\beta})}{\int d^2\beta I^{(s)}(\beta) q_I[I^{(s)}(\beta)]}, \end{aligned} \tag{2.37}$$

where q_I is a weight function that selects the scale covered by the galaxy and limits the integral, $i, j \in (1, 2)$, $I(\theta)$ and $I^{(s)}(\beta)$ are the brightness functions on the lens and on the source planes respectively, and $\bar{\theta}, \bar{\beta}$ are the image centroids:

$$\bar{\theta} = \frac{\int d^2\theta I(\theta) q_I[I(\theta)]\theta}{\int d^2\theta I(\theta) q_I[I(\theta)]}, \quad \bar{\beta} = \frac{\int d^2\beta I^{(s)}(\beta) q_I[I^{(s)}(\beta)]\beta}{\int d^2\beta I^{(s)}(\beta) q_I[I^{(s)}(\beta)]}. \tag{2.38}$$

The trace of Q_{ij} describes the angular size of the image while the traceless part describes its shape and orientation. From Q_{ij} we can define the complex ellipticity as

$$\epsilon = \frac{Q_{11} - Q_{22} + 2iQ_{12}}{Q_{11} + Q_{22}}. \tag{2.39}$$

In the same way, the intrinsic ellipticity for the unlensed source is described by $Q^{(s)}_{ij}$. The observed ellipticity on the lens plane and the intrinsic ellipticity on the source plane are related through the lens equation, i.e. in the first-order approximation $\beta = A\theta$, then it can be demonstrated that

$$Q^s = AQA^T = AQA. \tag{2.40}$$

Using the definition of the complex ellipticity, one finds the transformation (Schramm & Kayser, 1994; Seitz & Schneider, 1996):

$$\epsilon^{(s)} = \begin{cases} \dfrac{\epsilon - g}{1 - g^*\epsilon} & \text{if} \quad |g| \leq 1 \\ \dfrac{1 - g\epsilon^*}{\epsilon^* - g^*} & \text{if} \quad |g| > 1 \end{cases}, \tag{2.41}$$

where $*$ denotes the complex conjugate and g is the reduced shear defined in Eq. 2.27. By averaging over a large sample of galaxies, assuming that the orientations of galaxies is random, it is expected that the intrinsic ellipticities should mediate to zero. Then, Eq. 2.41 becomes

$$\epsilon = \begin{cases} g & \text{if} \quad |g| \leq 1 \\ \dfrac{1}{g^*} & \text{if} \quad |g| > 1 \end{cases}. \tag{2.42}$$

Therefore, the observed image ellipticity provides an unbiased estimate of the reduced shear.
Unfortunately, weak lensing distorsion is contaminated by other effects, increasing the difficulty in the measure of ellipticity. Firstly, the atmosphere and the Point Spread Function (PSF) of the observing instrument introduce anisotropies that could be easily mistaken for weak lensing distorsions. In addition, the shear depends on the redshift, meaning we need a correct characterization of galaxies's distance distribution.
The most common method to isolate the signal generated by the weak lensing effect from the other contributions is the Kaiser Squires Broadhurst (KSB) method (Kaiser & Squires, 1993).

Chapter 3

Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that focuses on developing iterative algorithms that allow computers to learn and identify patterns from data. The aim of the algorithms is to make predictions based on their experience. In this context, although the architecture of the majority of ML algorithms resembles the structure of neurons and synapses, our concept of "learning" should not be confused with the meaning related to human experience, but we should instead exploit the definition used in Mitchell (1997): "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." It is therefore immediate to observe that the purpose of ML programs is to complete tasks correctly. Moreover, it is important to note from the previous formal definition of "task" that the process of learning itself is not the task, but our means of attaining the ability to perform the task.

ML tasks could be described as a set of operations and processes performed over a particular dataset instance, called *example*. The example is a collection of *features*, typically represented as an entry $\mathbf{x}_i$ to the example vector $\mathbf{x} \in \mathbb{R}^n$ (where bold indicates vectors and n indicates the total size of the dataset), measured from the dataset that we want the ML system to process. For example, the features of an image are the pixel values of the image itself. Along with these features, we associate each example with a corresponding *label*, i.e., the output our model has to predict. Although ML can address a large variety of tasks, in this work, we will deal with only one specific type, namely, a *regression* task.

A regression problem can be described as: given some input, i.e., our examples and labels, the ML algorithm is asked to predict a set of numerical values. To solve this task, the program has to output a function $f : \mathbb{R}^{q \times q} \to \mathbb{R}^m$.

In this work, $q \times q$ is the dimension of the dataset images.

To evaluate the performance of our ML algorithms, we need to design a specific quantitative measure that gives the proportion of examples for which the model's output is

correct. This can be quantified using several metrics, for example the so-called *accuracy* (see Sec.4.3). To obtain a significant measure of this quantity, the ML algorithm has to process and analyze all the labeled data during an initial stage called *training phase*, in which the program updates itself at each step to improve its capability to predict the output. This updating process is described in Sec.3.3.1. Once the training has been done, our model goes through an additional stage called *test phase*. In this phase, the model performance is then evaluated on unlabeled data, different from the data used for training, respectively, a *training set* and a *test set*, and the output predictions are compared to the known corresponding labels to calculate the accuracy. Anyway, all the details regarding the implementation of the models and their training and test phases will be discussed in the following sections.

3.1 Deep Learning

Deep Learning (DL) is a subset of machine learning that focuses on the development and training of deep neural networks. These networks are composed of multiple layers of interconnected nodes, known as neurons, that mimic the structure of the human brain. The name deep comes from the stratification of the layers, as the input has to be processed by many layers built on top of each other.
The main advantage offered by neural networks is their ability to discover not only the mapping from the representation of the data to the output but the representation itself (Goodfellow et al., 2016), since automatically learning representation of the data is part of the learning process. This means that instead of relying solely on human intuition or dataset knowledge to handcraft the right features, neural networks can learn to extract relevant features directly from the raw data. This ability is known as *representation learning*, and it allows AI programs to adapt to new tasks with few modifications on the network, and to generalize problems without human intervention. In particular, within the framework of representation learning algorithms, DL techniques are characterized by the introduction of representations that are expressed in terms of other, less complicated, representations: the function f that maps the input to the output is decomposed into simpler functions, each providing a new representation of the data.

3.2 Neural Networks

Neural networks (NNs, Bishop, 2006) are computational models inspired by the structure and function of biological neural networks, mimicking the complex interconnections of the human brain to process and learn from data. They are constituted by simple processing units, also called neurons, linked through connections and organized

in layers. Each neuron receives one or more input signals, either directly from the raw data or or from the neurons in the previous layer, and every connection between neurons has an associated weight that determines the strength of the connection. These weights are adjusted during training to minimize the difference between the predicted output and the actual output. Each neuron then applies an activation function to the weighted sum of its inputs, which calculates the output value for each node: this function introduces non-linearity and enables the network to learn complex patterns and relationships within the data.
The most commonly used activation function in NNs is the Rectified Linear Unit (ReLU), shown in the left panel of Fig.3.1. It sets all negative values in the input to zero and leaves positive values unchanged. Mathematically, it can be defined as follows: $f(y) = max(0, y)$.

An alternative to the ReLU function is given by the Leaky ReLU, a variant of the

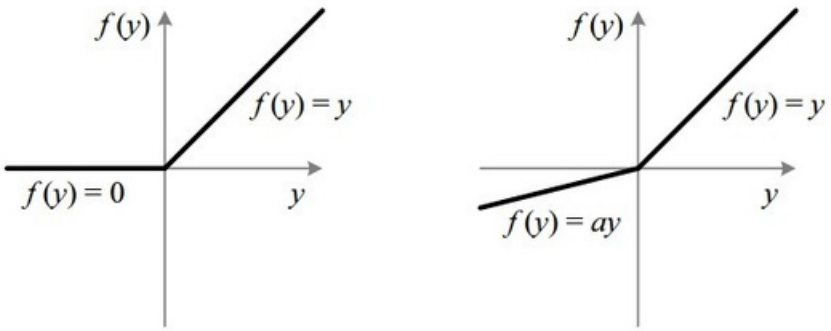


Figure 3.1: ReLU and Leaky ReLU functions (left and right panels, respectively).

ReLU activation function that addresses the "dying ReLU" problem (Pedamonti, 2018), which arises when neurons become inactive and stop learning during the training process. Leaky ReLU introduces a small slope a for negative values, allowing a small, non-zero gradient for negative inputs. It can be defined as:

$$f(y) = \begin{cases} ay, & \text{if } y < 0 \\ y, & \text{if } y \geq 0 \end{cases} \tag{3.1}$$

and it is plotted in the right panel of Fig.3.1.
In general, NNs have at least three layers: the input layer, the output layer, and one hidden layer. However, networks generally have several hidden layers capable of extracting high-level features from the data. In particular:

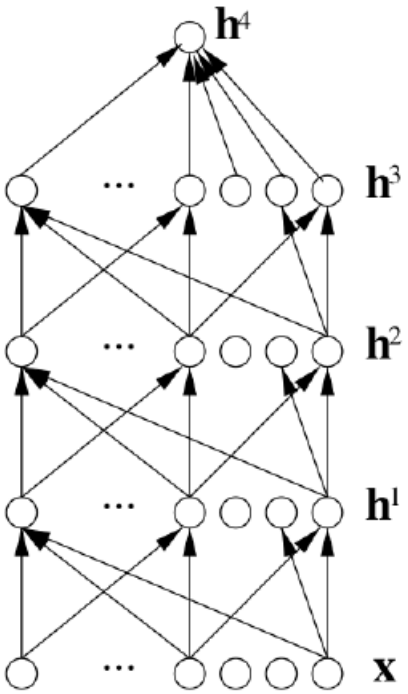


Figure 3.2: Architecture of a generic network with three hidden layers. Here, $\mathbf{x}$ is the input layer, while $\mathbf{h}^i$ with $i = 1, 2, 3$ are the hidden layers. The last layer, $\mathbf{h}^4$, is the output layer. The picure also illustrates the interconnections between the units of each layer, connecting both lower-levels and higher-level layers. Image from (Bengio et al., 2009).

- The input layer, often referred to as the visible layer, is where data is transmitted to the network;
- The hidden layers are designed to capture abstract features from the data. The predetermined number of hidden layers and nodes within each layer is established a priori, but their values are adjusted during the training phase following a random initialization;
- The last layer of the network is the output layer, which provides a further modification to the features, ultimately accomplishing the designated task. The labels linked to the training data will determine how the weights in the output layer are adjusted. The learning algorithm then fixes the hidden layers to approximate the desired output most effectively.

Each layer is composed of several nodes connected to the ones of the previous layer through weighted connections that describe how the input is propagated through the

network. The output of the k-th layer $\mathbf{h}^k$ can be calculated using the output of the previous layer $\mathbf{h}^{k-1}$:

$$\mathbf{h}^k = f(\mathbf{b}^k + w^k \mathbf{h}^{k-1}), \tag{3.2}$$

where f is the activation function, $\mathbf{b}$ and $\mathbf{w}^k$ are the vector of offsets (biases) and the weight matrix associated with the layer respectively: their dimension is defined by the number of units in the layer. The first layer is given by the input: $\mathbf{x} = \mathbf{h}^0$, while the last layer $\mathbf{h}^L$ is used to make a prediction.

3.3 Training Process

After establishing and implementing the model's architecture, the next crucial step is training the network before applying it to any data. The purpose of the training procedure is to pinpoint the most effective values for the weights and offsets: while the network's structure defines only the number of parameters per layer during implementation, their actual values are fine-tuned in the training phase based on how well the model predicts accurate outputs.
The training is an iterative process, and each iteration, known as *epoch*, involves that the network processes the entire training set. Achieving convergence to the best parameter configuration often requires many epochs. After each epoch, the weights and biases are updated based on the network's performance: the effectiveness of the network is evaluated using a *loss function*, which measures the disparity between the network's predictions and the correct outputs.
The most relevant loss function in the context of this work is the Mean Squared Error (MSE), defined as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \ , \tag{3.3}$$

where n is the total number of training examples, $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ are the predicted output value and the real output value respectively.
The purpose of the training process is then to search for the best combination of weights and biases that minimize this loss function. Usually, every epoch has its associated value of loss function, and its trend with time (the epochs) has to be checked to evaluate the effectiveness of the learning process, because the loss function value is expected to decrease as the number of epochs increases. For some examples of loss function trends, see Sect. **??**.

3.3.1 Backpropagation Algorithm

The task of adjusting weights and biases during the training process is assigned to a training algorithm. The most common training algorithm is the *backpropagation algorithm* (Rojas, 2009).
At the heart of the learning process is the idea of iteratively refining the parameters of a neural network by updating the network's weights in tandem with an *optimization method*. In particular, the backpropagation algorithm operates as a *supervised learning technique*, where the network is provided with labeled training data and the algorithm has to compute the gradient of a chosen loss function concerning the weights and biases of the neural network. The gradient serves as a compass, since it gives an indication of how these parameters should be modified to minimize the difference between the predictions of the network and the known output. The optimization method then utilizes the gradient information to update the weights, allowing the neural network to improve its performance iteratively.
We can break down this complex process into two distinct yet interdependent steps:

- Forward propagation, as explained by Nielsen (2015), constitutes the initial step in the backpropagation. During this phase, the input data is propagated through the network's layers, with each hidden layer contributing to the computation of the final output as described in Eq. (3.2). The weights and biases in each layer play a crucial role in determining the output, as they are adjusted iteratively to align the network's predictions with the ground truth. At the end of this process, the prediction of the network, i.e. the output of the final layer, is used in combination with the ground truth to evaluate the loss function, to assess the network's performance, and to initiate the subsequent backpropagation step.

- Backpropagation: following the computation of the forward pass, the backpropagation phase takes center stage. This step involves the systematic calculation of the gradient of the loss function with respect to the current weights and biases of the network. Then, the network is run backwards until the input layer is reached. Nielsen (2015) outlines how the algorithm attributes the error to each parameter by recursively applying the chain rule of differentiation of composite functions backward through the layers, allowing for the precise adjustment of weights and biases. The backpropagation algorithm's reliance on the chain rule highlights its foundation in calculus and mathematical optimization, and its systematic approach to gradient computation enables the algorithm to discern the contribution of each parameter to the overall error, facilitating targeted adjustments. As a result, the network refines its predictions with each iteration, progressively converging toward a state where the loss function is minimized.
  In particular, understanding how changing the weights and biases in a network

changes the loss function means computing the partial derivatives

$$\frac{\partial L}{\partial w_{jk}^{l}} \quad \text{and} \quad \frac{\partial L}{\partial b_{j}^{l}}\,, \tag{3.4}$$

where w_{jk}^{l} denotes the weight for the connection between the k^{th} neuron in the $(l-1)^{th}$ layer and the j^{th} neuron in the l^{th} layer, as shown in Fig.3.3. Similarly, b_{j}^{l} denotes the bias of the j^{th} neuron in the l^{th} layer. Those derivatives represent a quantitative measure indicating the extent to which a weight or a bias deviates from its optimal value, i.e. the one required to minimize the loss function.

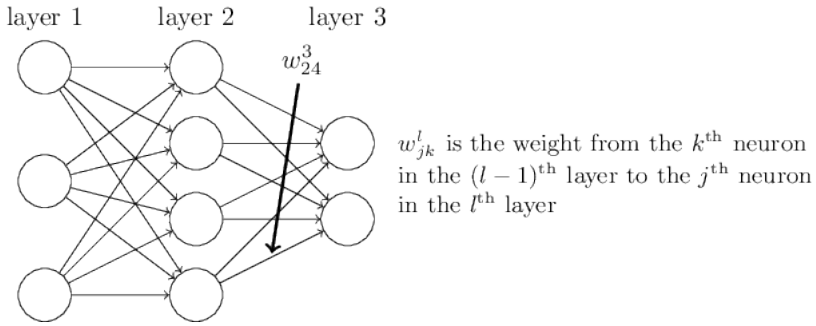


Figure 3.3: Simple scheme illustrating the weights notation. Here, w_{24}^{3} $[w_{jk}^{l}]$ is the weight from the 4^{th} $[k^{th}]$ neuron of the 2^{nd} $[(l-1)^{th}]$ layer, connected to the 2^{nd} $[j^{th}]$ neuron of the 3^{rd} $[l^{th}]$ layer.

To compute the derivatives, we first introduce an intermediate quantity δ_{j}^{l} as the "error" (thought as the discrepancy of the current weight and bias value from the value that minimizes the loss) of the j^{th} neuron in the l^{th} layer, as:

$$\delta_{j}^{l} = \frac{\partial L}{\partial z_{j}^{l}}, \tag{3.5}$$

where we have adopted the notation $\mathbf{z}^{l} = \mathbf{b}^{l} + w^{l}\mathbf{h}^{l-1}$ from the Eq. 3.2. The main task of the backpropagation algorithm is then to obtain δ^{l} for every layer (where δ^{l} denotes the vector of errors associated with layer l) and to relate those errors to the quantities described in Eq. 3.4. This task is achieved with the following steps.
Denoting the activation function as f, the backpropagation algorithm starts by calculating the error δ^{O} on the output layer O with components

$$\delta_{j}^{O} = \frac{\partial L}{\partial h_{j}^{O}} f'(z_{j}^{O}), \tag{3.6}$$

where the first term on the right measures how fast the loss changes as a function of the j^{th} output activation, and the second one measures how fast the activation function f changes at z_j^O. Normally, if the loss function does not depend heavily on a particular output neuron j, then δ_j^O will be small. Eq. 3.6 should be rewritten in a matrix-based form since the algorithm exploits this formalism. We can then define:

$$\delta^O = \nabla_h L \odot f'(z^O), \tag{3.7}$$

where the vector $\nabla_h L$ has the partial derivatives $\partial L / \partial h_j^O$ as components, and the operator $\odot$ stands for the element-wise product of the two vectors.
The algorithm is now capable of computing the error δ^l at each layer in terms of the error in the next layer δ^{l+1}, since the gradient in Eq.3.7 is then converted into a gradient with respect to the weights at each node and is passed to the next lower level hidden layer as:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l), \tag{3.8}$$

where $(w^{l+1})^T$ is the trasposed of the weight matrix w^{l+1} for the $(l+1)^{th}$ layer. If we suppose to know the value of δ^{l+1} in the layer $l+1$, the application of the trasposed weight matrix $(w^{l+1})^T$ can be thought intuitively as moving the error backward through the network, giving us a measure of the error at the output of the l^{th} layer. Then, by computing the elementwise product $\odot f'(z^l)$, we move the error backward through the activation function in layer l, obtaining the value of δ^l in the weighted input to layer l. By combining Eq. 3.7 with Eq. 3.8 we can compute the error δ^l for any layer in the network, until the input layer is reached. The backpropagation algorithm then calculates the derivative of the loss function for any bias in the network as

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l \tag{3.9}$$

since this quantity has already been calculated by Eq. 3.8, meaning that the error δ_j^l is exactly the derivative of the loss with respect to the j^{th} bias in layer l. In the last step, the backpropagation algorithm computes the derivative of the loss function with respect to any weight w as

$$\frac{\partial L}{\partial w_{jk}^l} = h_k^{l-1} \delta_j^l, \tag{3.10}$$

since the algorithm already computed the quantities h_k^{l-1} and δ_j^l.
Eq. 3.10 can be rewritten in a less index-heavy notation as

$$\frac{\partial L}{\partial w} = h_{in} \delta_{out}, \tag{3.11}$$

where h_{in} is the activation of the neuron input to the weight w calculated in the forward step, and δ_{out} is the error of the neuron output from the weight w.
It is important to note that the training dataset is not processed as a whole but it is split into several small *batches*, namely, a given number of examples. The gradients of the loss function passed to the optimizer for the next step are an average estimation of the gradients calculated for the training examples at the end of each epoch.

3.3.2 Optimization

While the backpropagation algorithm computes the gradients essential for parameter adjustment, the optimization method defines how the weights and biases are updated. It uses the previously computed gradients to adjust the parameters iteratively. To do so, we have to define an important hyper-parameter, namely, a parameter that is chosen manually and is not further adjusted during the training procedure, the *learning rate*. This rate defines the step length of each update in the negative gradient direction. If it is too high, the model rapidly converges towards the minimum of the loss function, but it might not reach it exactly, while if it is too small, the model might get stuck into a local minimum or require too many epochs to converge.
Optimization is usually left to a set of pre-built optimizers since differences in the weight-adjustment methods could affect the network's accuracy. The most relevant optimizer in the context of this work is the ADAptive Moment estimation (Kingma & Ba, 2017; Reddi et al., 2019). Anyways, in some cases involving comparisons between the performances of our networks with different optimizers, also Root Mean Square propagation (RMSprop, Hinton et al. 2012) was implemented and tested.

3.3.3 Validation

To assess the model's generalization capability, it is crucial to evaluate its predictions within the training phase on the training set and an independent dataset, known as the *validation set*. Typically, the validation set comprises a small fraction (approximately 10-15%) of the training set and must accurately represent the dataset's characteristics. While the network's performance on the training set guides the updates of the weights and biases, following the procedure outlined in the previous section, the results on the validation set play a key role in determining adjustments such as decreasing the learning rate or stopping the training process. However, these results do not factor into calculating the loss function and thus do not impact the parameter updates.
When training an ML algorithm, the objective is to minimize the loss function on the training set and to narrow the gap between training and validation errors. A highly complex and deep model will likely perform well on the training set, with a steady decrease in training error. However, there is a risk of memorizing non-representative

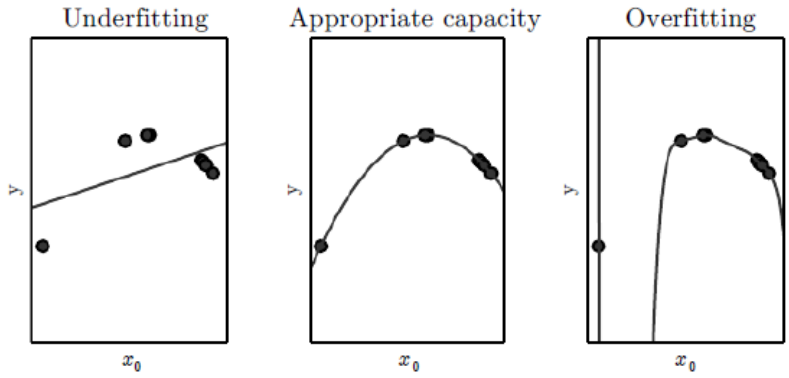


Figure 3.4: Comparison between a linear (*left*), quadratic (*center*) and a polynomial of degree-9 (*right*) predictor attempting to fit a problem where the true underlying function is quadratic. The linear function is unable to capture the curvature in the true underlying problem, so it underfits. The degree-9 predictor is capable of representing the correct function, but it is also capable of representing infinitely many other functions that pass exactly through the training points because we have more parameters than training examples, so it overfits. In this example, the quadratic model is perfectly matched to the true structure of the task, so it generalizes well to new data. Image from (Goodfellow et al., 2016).

properties specific to certain examples, hindering the improvement in predicting the validation set output. This phenomenon is known as *overfitting* and is denoted by a low loss value on the training set and a high one on the validation set. Conversely, if the model is overly simplistic, it may struggle to capture complex properties from the training set and will perform poorly on both training and validation sets, indicating *underfitting*.
So, underfitting occurs when the model cannot obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training and test errors is too large. We can control whether a model is more likely to overfit or underfit by altering its *capacity*, namely, its ability to fit a wide variety of functions (Goodfellow et al., 2016). Capacity is strictly linked to the number of weights/layers of the network: models with low capacity may struggle to fit the training set, while models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set. ML algorithms will generally perform best when their capacity is appropriate, given the true complexity of the task they need to perform and the amount of training data they are provided with. Models with insufficient capacity are

unable to solve complex tasks. Models with high capacity can solve complex tasks, but they may overfit when their capacity is higher than needed to solve the present task. Fig.3.4 shows this principle in action.

3.4 Convolutional Neural Networks

Convolutional networks (LeCun et al., 1995), also known as convolutional neural networks or CNNs, are a specialized kind of neural networks for processing data that have a known, grid-like topology (Goodfellow et al., 2016). In the context of this work CNNs are particularly appropriate since we have to deal with image data, which can be thought of as 2D grids of pixels. The name "convolutional neural network" indicates that the network employs a mathematical operation called *convolution*, which is a specialized kind of linear operation that involves an input image and a kernel: the output of this operation is often referred to as *feature map*.
In machine learning applications, the input is usually a multidimensional array of data (for example, a RGB-2D image), and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as tensors.
In detail, the one-dimensional convolution between two functions is defined by the following integral

$$s(t) = \int x(a) \cdot w(t-a)\, da, \tag{3.12}$$

where $s(t)$ is the feature map depending on the variable t, x is the input and w is the kernel.
However, the convolution between multi-dimensional arrays is better described as a discrete multiplication between matrices. Typically, the kernel is smaller than the image, and the resulting output is a very sparse matrix with many null elements. The output S of the convolution between an image I and a kernel K of dimension $(M \times N)$ can be computed as:

$$S(i,j) = (K * I)(i,j) \sum_{m=1}^{M} \sum_{n=1}^{N} I(i-m, j-m) K(m,n), \tag{3.13}$$

where (m,n) represents a generic point on the kernel grid, while (i,j) defines a point on the image grid. More in detail, the units of a layer that are directly connected to the units of the following layer are also known as its *receptive field*. In a convolutional layer, several convolutions are performed in parallel and then passed to an activation function like the ReLU, which adds non-linearity and produces an output tensor. In this context, *stride* and *padding* are crucial parameters in implementing convolutional operations.

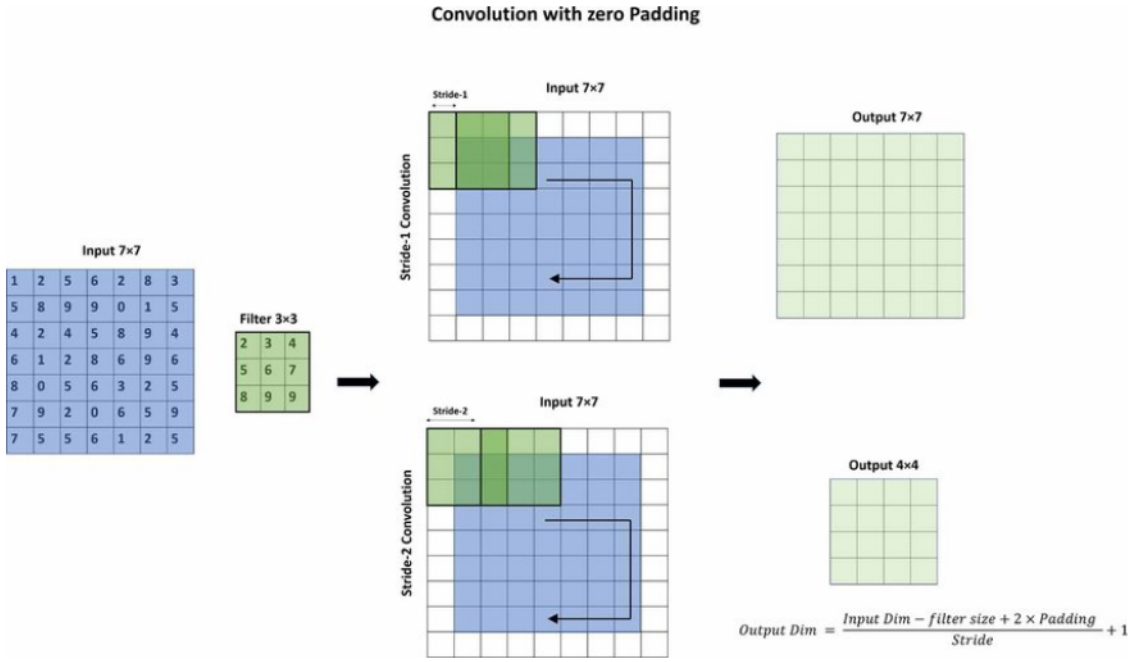


Figure 3.5: Schematic of 2D CNN for stride-1 and stride-2 with zero padding (adding additional layers of zero values) to prevent shrinking. For stride-1, the output channel dimension is the same as the input and for stride-2, the output channel dimensions are reduced to half of the input dimension. Image from ResearchGate website.

The stride refers to the step size, or the number of pixels that the convolutional filter moves across the input data during the convolution operation. A larger stride value reduces the spatial dimensions of the output volume, effectively downsampling the data. This can help to reduce computational complexity and, in some cases, control overfitting. A smaller stride preserves more spatial information but it may increase the computational requirements. The choice of the stride depends on the specific task and the desired balance between spatial resolution and computational efficiency.
Padding involves adding extra pixels (usually zeros, referred to as zero-padding) around the input data before applying the convolution operation. This technique ensures that the spatial dimensions of the input and output volumes are compatible. It also helps to retain information at the input's borders, preventing a size reduction as the convolutional operation is applied. An example of convolutions with different stride values and zero-padding is given in Fig.3.5.
CNNs consist of multiple convolutional layers, usually interspersed with *pooling layers*. Pooling is used to reduce the spatial dimensions of a grid, executing a form of downsampling that effectively lowers computational expenses while retaining the key

features of the input map. Similar to convolution, pooling operations employ kernels, typically (2×2) or (3×3), with stride (2×2) that halves the size of the input. Two variations of pooling exist. Max pooling yields the highest value within the kernel-covered portion of the grid, whereas average pooling produces the average value.
A typical CNN then is made of several convolutional layers alternating with pooling layers and generally ends with *fully connected layers* (FC) as an output, as shown in Fig.3.6. FC layers are employed for the final classification and are characterized by the full connection between each layer's neurons and the next one's neurons. Being fully connected, these kinds of layers are usually more parameters-heavy than the convolutional ones; thus, their use is limited to the final stages of the network.

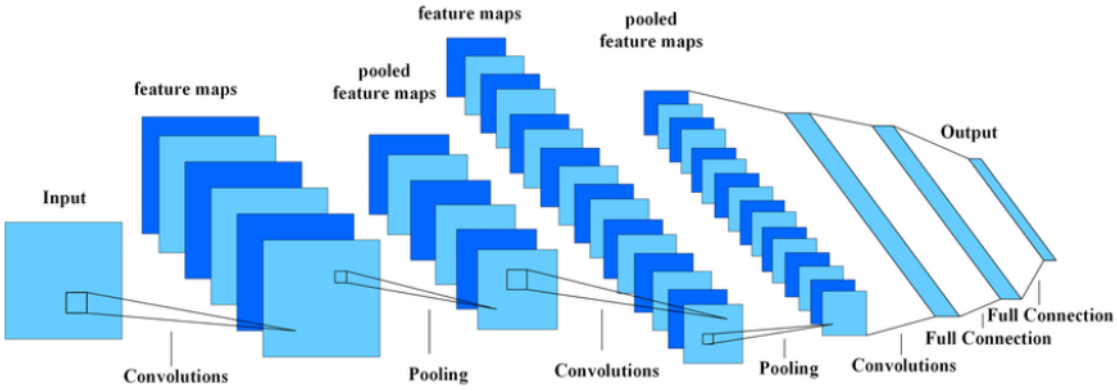


Figure 3.6: The classical structure of convolutional neural networks (CNNs). Image from ResearchGate website.

In the following sections, we describe the architecture and properties of three more complex CNNs: VGGNet (Simonyan & Zisserman, 2015), Inception-v4 and Inception-ResNet (Szegedy et al., 2016) that are the models implemented in this Thesis work.

3.4.1 VGG-Net

VGGNet, or the Visual Geometry Group Network, is a deep convolutional neural network architecture designed for image classification. It was introduced by the Visual Geometry Group (Simonyan & Zisserman, 2015) at the University of Oxford, and was presented at the International Conference on Learning Representations (ICLR) in 2015. The main innovation of VGGNet was its emphasis on using a simple and uniform architecture with very small 3×3 filters (which is the smallest size to capture the notion of left/right, up/down, center), stacking multiple convolutional layers with small receptive fields to deepen the network without overburdening it with many parameters. In fact, previous works typically employed filters with a receptive field of 11×11,

(Krizhevsky et al., 2012) with stride 4, or 7×7, (Zeiler & Fergus, 2013; Sermanet et al., 2014) with stride 2.
The Visual Geometry Group demonstrated that big-size filters could be replaced with a succession of 3×3 ones, as shown in Fig3.7 for a 5×5 convolution. Using small fil-

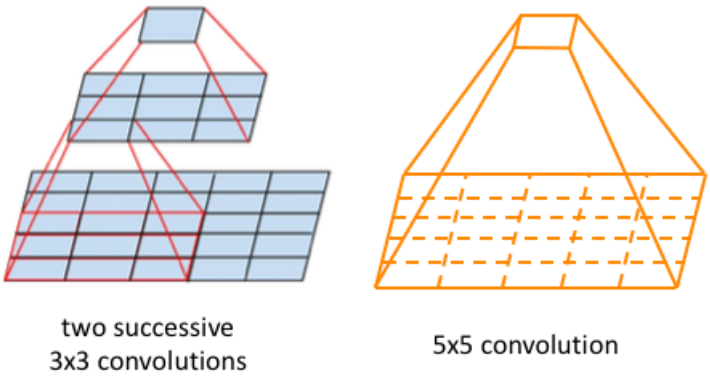


Figure 3.7: Equivalence of receptive fields using two successive 3×3 convolutions and one 5×5 filter (Szegedy et al., 2016).

ters instead of larger ones has the advantage of allowing the implementation of deeper architectures by limiting the number of parameters that the network requires. For example, a 5×5 filter involves 25 parameters ($5 \cdot 5 = 25$), while stacking two 3×3 filters uses only 18 parameters ($3 \cdot 3 + 3 \cdot 3 = 18$).
This is crucial since the authors not only developed an architecture that is easy to understand and implement but also demonstrated that increasing the depth of a neural network can lead to a better performance.
In fact, the authors built different VGG configurations, starting from 11 weight layers up to 19, and compared them. Despite having a different number of weight layers, the general structure of the VGGNet is the same for every configuration.
The input is a fixed-size 224×224 RGB image, passed through a stack of convolutional layers involving 3×3 filters. The convolution stride, as is the padding, is fixed to 1 pixel, meaning that the spatial resolution is preserved after convolution. Then, pooling is carried out by five max-pooling layers over a 2×2 pixel window with stride 2, which follows only some of the convolutional layers. Finally, the last stack of convolutional layers is followed by three FC layers. The final layer is the *soft-max* layer typically used for classification, which contains the softmax function. This function transforms the real output values of the last FC layer into values between 0 and 1 so that they

can be associated to a probability for each one of the classes. The configuration of the fully connected layers is the same in all networks. A visual example of VGGNet architecture is shown in Fig. 3.8. Tab. 3.2 summarizes all the configurations proposed

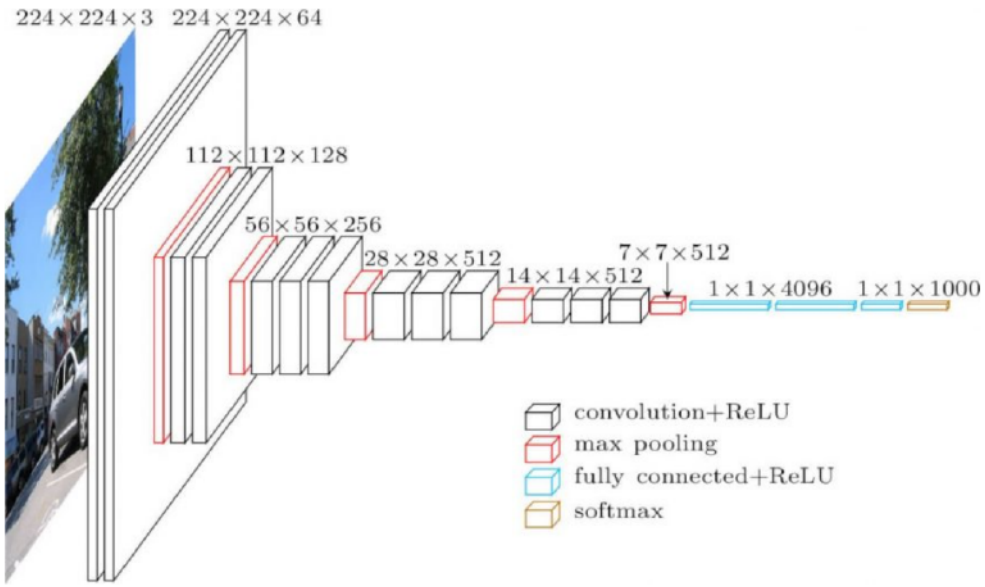


Figure 3.8: Structure of the VGG-16 (Configuration D), (Simonyan & Zisserman, 2015).

for the VGGNet. In particular, configuration C shows, in addition to the standard 3×3 convolutional layers, three 1×1 convolutions that can be seen as a linear transformation of the input channels (followed by non-linearity). This is a way to increase the decision function's non-linearity without affecting the convolutional layer's receptive fields.
Tab. 3.1 reports the number of parameters for each VGGNet configuration. Despite a large depth, the number of parameters of this architecture is not greater than that of a more shallow net with larger convolutional layer widths and receptive fields (for example, 144M parameters in Sermanet et al. 2014). The authors observe that the

Table 3.1: Number of parameters for each configuration of the VGGNet (Simonyan & Zisserman, 2015).

Network	A	B	C	D	E
Number of parameters (10^6)	133	133	134	138	144

classification error decreases with the increasing VGGNet depth from A to E. Moreover, configuration C (which contains three 1×1 conv. layers) performs worse than

configuration D, which uses 3×3 convolutional layers. This states that the additional non-linearity improves the network performance (since C is better than B), but it is more important to use filters that capture the spatial context.

Table 3.2: VGGNet original configurations (shown in columns). The depth of the configurations increases from the left (A) to the right (E) as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity (Simonyan & Zisserman, 2015).

A	**B**	**C**	**D**	**E**
11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)				
conv3-64	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool				
conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool				
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool				
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool				
FC-4096				
FC-4096				
FC-1000				
soft-max				

3.4.2 Inception Networks

We previously emphasized how the architectural simplicity of VGGNet can be considered an advantage about the ease of use of this model. This benefit, though, comes at a high cost: evaluating the network requires a lot of computation since the VGGNet performance relies on its depth, and by enlarging the depth of a network, we are also increasing the model's size and then the computational requirements. To tackle this problem, the first version of Inception Networks (also called GoogLeNet) was presented in 2014 by a group of researchers from Google (Szegedy et al., 2014), as "Inception-v1". The main innovation in the Inception networks lies in their *inception modules*, which are a set of parallel convolutional filters with different receptive field sizes. In detail, the inception module employs (1×1), (3×3), and (5×5) convolutions, as well as a max-pooling layer, all performed in parallel: basically, the author's idea is to apply filters with different size on the same input in a multi-scale approach, making the model learn both local and global features efficiently. An example of the simplest Inception module from Inception-v1 is shown in Fig. 3.9. Since even using a modest number of big filters (like the (5×5) convolution) can be prohibitively expensive, the authors decided to add a convolution with a 1×1 filter before applying the 3×3 and 5×5 operations: this trick reduces the dimension of the input and the overall computational cost since the output map has the same height and width of the input but the number of channels will be the same as the number of 1×1 filters applied. Then, the output of the different filters and pooling layers are stacked together and passed as an input for the following module or layer.

In general, an Inception network consists of modules of the above type stacked upon

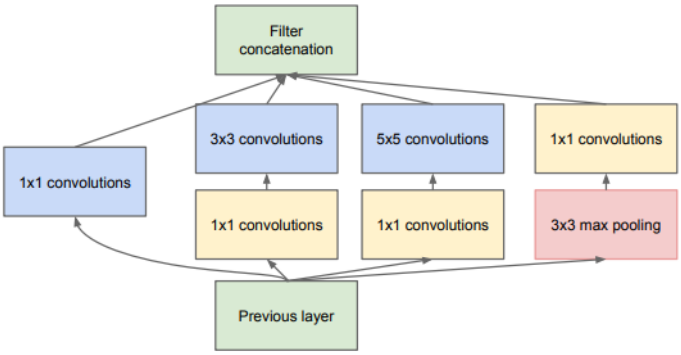


Figure 3.9: Inception module with dimension reductions. (Szegedy et al., 2014).

each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid. The overall structure of an Inception network consists of three parts: the

stem, the *body*, and a *final classifier*. The stem part includes all the basic operations to be done before the introduction of the Inception modules. This was introduced since the authors found using Inception modules only at higher layers while keeping the lower layers in a "traditional convolutional fashion" beneficial. The body is the part that contains the Inception modules and changes between the various Inception network versions. This part also contains in the original work two auxiliary classifiers connected to two of the Inception modules, with the purpose to help to compute the final loss function by computing two intermediate losses. This is because the back-propagation algorithm in such a complex architecture could struggle to compute this function. The total loss then will be the weighted sum of the intermediate losses (each with a weight of 0.3) and the final loss. The authors then found that the auxiliary classifiers were not as effective as expected, giving no improvements in the network training performance. These classifiers were changed in Inception-v2 (Ioffe & Szegedy, 2015) and used as regularizers to reduce overfitting. At last, the final classifier passes the module's final output to the fully connected layers to perform the classification.

Inception-v4

In this work, we used the latest version of these networks, called Inception-v4 (Szegedy et al., 2016). The Inception-v4 contains key innovations that improved the network performance significantly compared to the previous network versions.
The main one comes from the second version of Inception networks (Ioffe & Szegedy, 2015), and consists in the substitution of the 5×5 convolutions with two 3×3 filters, as we have previously described for VGGNet in 3.4.1. Moreover, the authors found that 3×3 convolutions could be substituted by a 1×3 filter followed by a 3×1 kernel, and that also works for any $n \times n$ filter (as a succession of a $n \times 1$ filter with a $1 \times n$ filter). A general example is shown in Fig. 3.10, but can be seen in detail for 7×7 convolutions in the Inception-B block (Fig.3.13). These substitutions help reduce the number of parameters and then the computational requirements of the network.
Another adjustment involves the addition of reduction modules to execute a more efficient grid-size reduction. Traditionally, convolutional networks use pooling operations to decrease the grid size of the feature maps. Usually, starting from a $d \times d$ grid with n filters, pooling is preceded by a stride-1 convolution to obtain $2n$ filters, and then pooling is applied, obtaining a $\frac{d}{2} \times \frac{d}{2}$ final grid. This, though, has a high computational cost on bigger grid sizes since the number of operations grows as $2n^2d^2$. Then we could possibly switch to pooling followed by convolution having $2n^2(\frac{d}{2})^2$ and reducing by a quarter the number of operations, but this would be a "representational bottleneck", meaning that the dimensionality of the feature maps is significantly reduced and then the network's ability to represent information is constrained or bottlenecked at that particular layer. The authors then suggested performing pooling and convolution in parallel, as shown in Fig. 3.11.

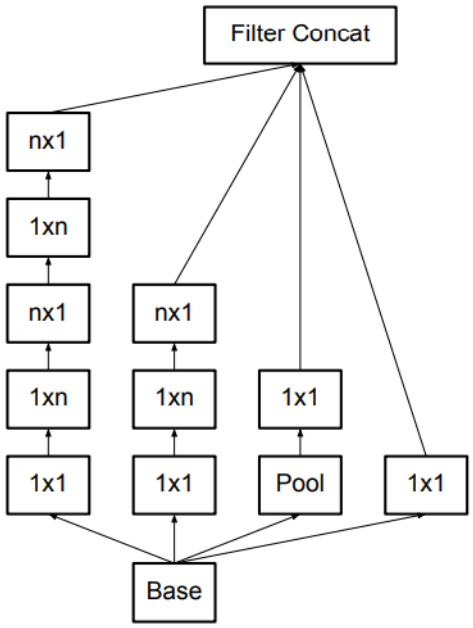


Figure 3.10: Inception modules after the factorization of the $n \times n$ convolutions. (Ioffe & Szegedy, 2015).

Lastly, Inception-v4 inherited from its second (Ioffe & Szegedy, 2015) and third versions (Szegedy et al., 2015) two auxiliary techniques: *batch normalization* and *label smoothing*. Batch normalization is a regularization technique applied to the activations of neurons before each non-linearity function (both for convolutions and fully connected layers) that consists of normalizing the output of the activation layer by subtracting the mean of the batch (seen in 3.3.1) and dividing by its standard deviation, to have zero mean and unit variance. This technique mitigates the change in the distribution of layer inputs during training, which can slow down the training process and then make the learning faster, more stable, and less sensitive to the choice of the initial weights. Label smoothing is another regularization technique that encourages the model to be less confident, preventing it from overfitting. It consists of replacing the label distribution of a training example with a mixture of the original ground-truth distribution and a fixed distribution that is independent of the training examples. This mixture depends on a smoothing parameter ϵ that has to be calibrated to reproduce the accuracy of the network. This technique improves the network generalization capacity and makes the model less sensitive to noise in the training data.
In Fig. 3.12 and Fig. 3.9, we present the detailed structure of the blocks and the reduction modules of Inception-v4 utilized in this work. In Fig. 3.14, we present the stem of the network and the overall schema of Inception-v4. Convolutions marked with

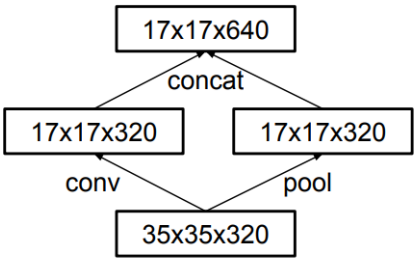
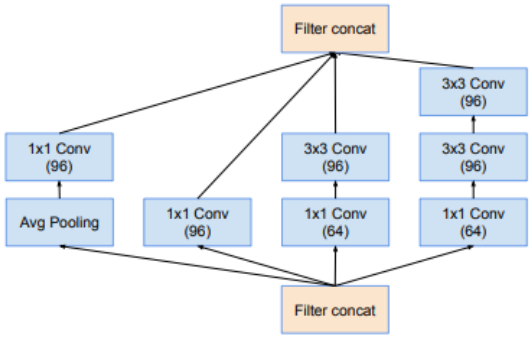


Figure 3.11: Inception module that is both cheap and avoids the representational bottleneck, seen from the perspective of grid sizes rather than the operations. Six hundred forty feature maps are obtained as a sum of two parallel operations containing 320 feature maps each. (Szegedy et al., 2015).

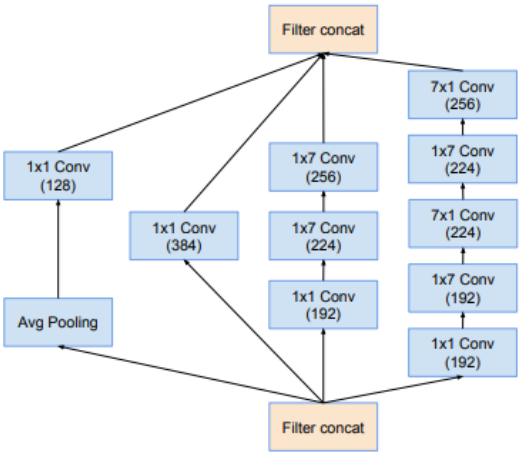
"V" are valid padded, meaning that the input patch of each unit is fully contained in the previous layer, and the grid size of the output is reduced accordingly. Convolutions without the V mark are instead zero-padded (see Section 3.4), meaning that the output and input dimensions are kept the same.
The authors used 4 Inception-A blocks, 7 Inception-B blocks, and 3 Inception-C blocks in the original work. A reduction module follows the A-type and B-type blocks to reduce the dimension of the output. After the C-type blocks, the authors perform an average pooling before applying the dropout technique in the fully connected layers. The softmax layer then performs classification.

Inception-Resnet-v2

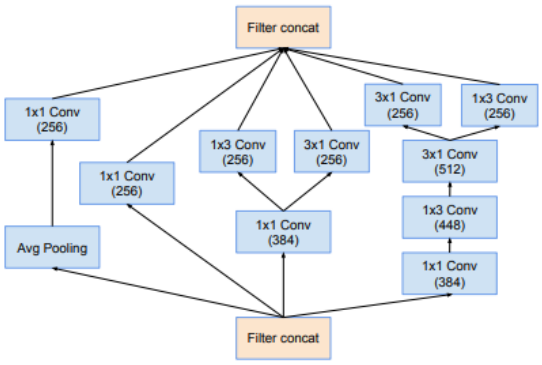
In the same paper, the authors present a second type of network based on the Inception architecture called Inception-Resnet-v2. This network differs from the previously presented Inception architecture in the type of blocks involved for the convolutions and is based on *residual learning*.
Residual learning was introduced by He et al. (2015) as a way to make the training process of deep networks easier and more efficient. This need arises when it comes to very deep architectures since the authors proved theoretically and empirically that having a deeper architecture does not simply translate into a better performance, but could lead instead to a higher degradation in the accuracy, meaning that both training and test error are higher compared to a shallower network. Residual Networks (ResNets) are built using special blocks, shown in Fig.3.15 to train deeper networks without falling into degradation. The idea is to simultaneously propagate the input of the block x both through the layers within the block and store the input without being changed.



(a) Inception-A block

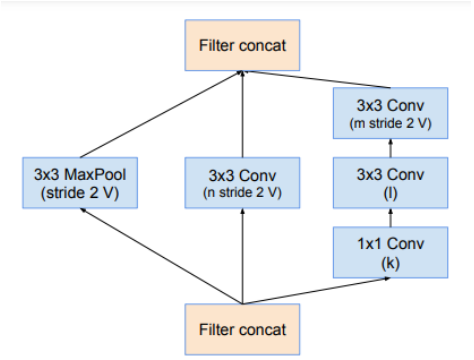


(b) Inception-B block

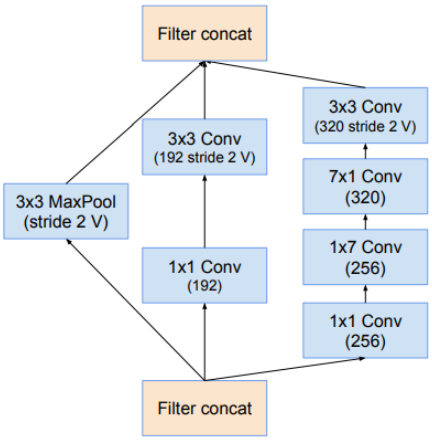


(c) Inception-C block

Figure 3.12: Inception-v4 blocks (Szegedy et al., 2016).

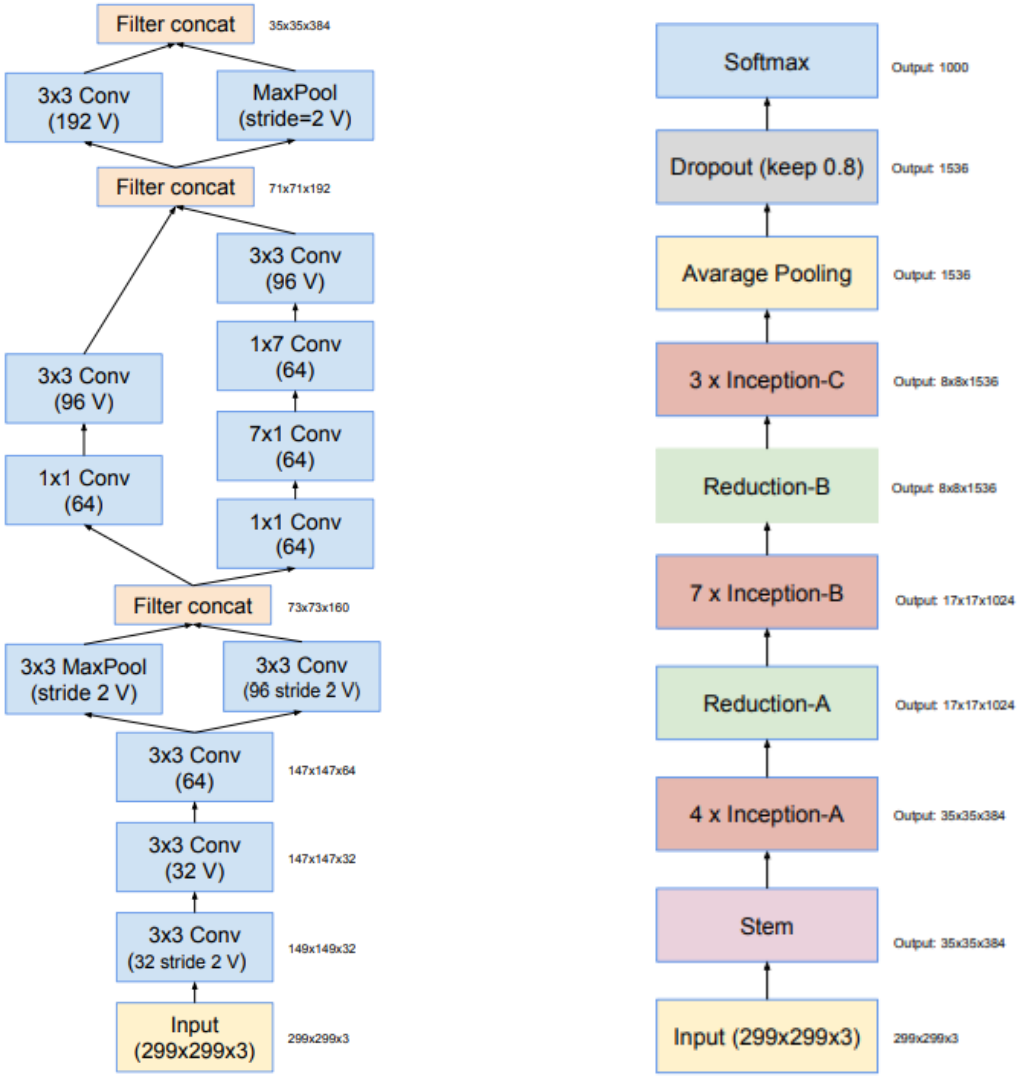


(a) 35×35 to 17×17 reduction module, (Reduction-A). The k, l, m, n numbers represent filter bank sizes.



(b) 17×17 to 8×8 reduction module, (Reduction-B)

Figure 3.13: Inception-v4 reduction modules (Szegedy et al., 2016).



(a) (b)

Figure 3.14: (a) The schema for the stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. (b) The overall schema of the Inception-v4 network. (Szegedy et al., 2016).

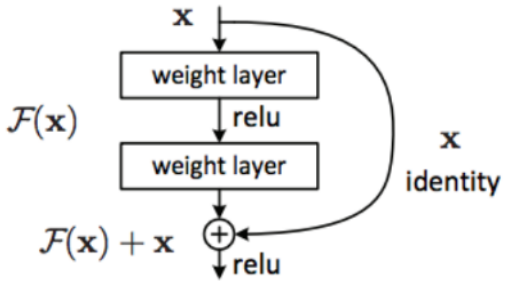


Figure 3.15: A simple residual block. (He et al., 2015).

By doing so, the function that the block has to learn can be described as:

$$F(\mathbf{x}) := H(\mathbf{x}) - \mathbf{x},$$

where $H(\mathbf{x})$ is the original function, and $F(\mathbf{x})$ is the residual function. Then, after passing the input of the first layer of the block to the output of its last layer, the network should be able to predict:

$$F(\mathbf{x}) + \mathbf{x} = H(\mathbf{x}),$$

This kind of architecture can speed up the learning process in some cases, for example, if the convolution layer has to perform an identity mapping (as shown in Fig.3.15): in this particular instance, the corresponding residual block would have to learn all zeros, without adding extra parameters to the network (because the dimension of input and output are the same).
Conversely, if $F(\mathbf{x})$ and $\mathbf{x}$ have different dimensions, the shortcut can still perform identity mapping but using the zero padding technique to increase dimensions. Otherwise, it can be used as a projection shortcut to match dimensions using

$$\mathbf{y} = F(\mathbf{x}) + W_s\mathbf{x},$$

where $\mathbf{y}$ and $\mathbf{x}$ are the output and the input of the layer, and W_s is the matrix responsible of the dimension reduction.
In Szegedy et al. (2016) residual learning is implemented within the Inception blocks, resulting in an evenly deeper network with respect to Inception-v4 that combines either parallel convolutional operations and residual connections. The authors proved that the performance of this hybrid type of architecture is really similar (i.e. just slightly better) to Inception-v4, but with a significant decrease in computational cost, as is shown in Fig.3.16.

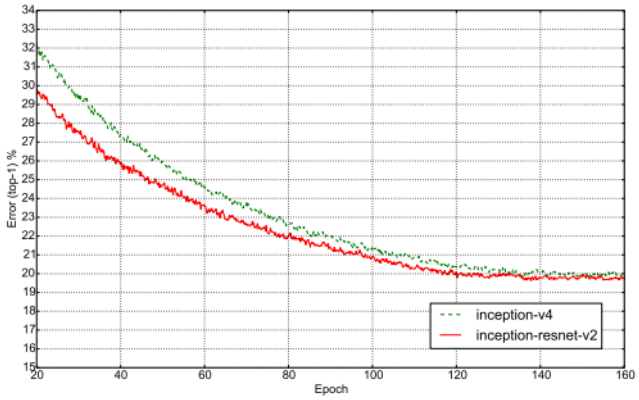
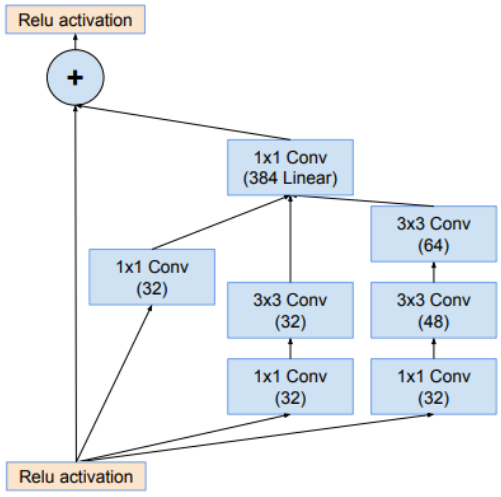
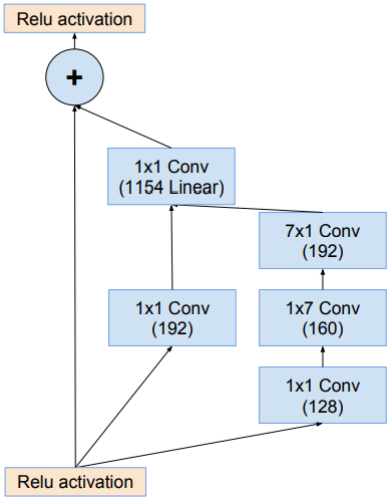


Figure 3.16: Top-1 error evolution during training of Inception-v4 vs an Inception-Resnet-v2 of similar computational cost. The residual version was training much faster and reached slightly better final accuracy than the traditional Inception-v4 (Szegedy et al., 2016).

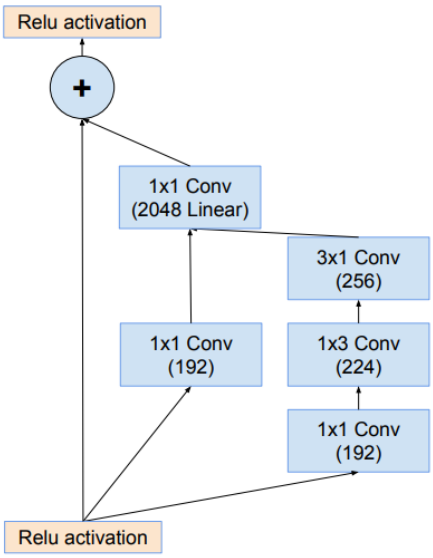
In Fig.3.17 are shown the building blocks of Inception-Resnet-v2. The reduction-A module and the stem of the Resnet version of Inception are the same illustrated respectively in Fig.3.13(a) and Fig3.14(a). The overall schema of the original Inception-Resnet-v2 network is shown in Fig3.18 and includes 5 A-type blocks, 10 B-type blocks, and 5 C-type blocks. Finally, the remaining part of the network architecture follows the same implementation seen for Inception-v4.



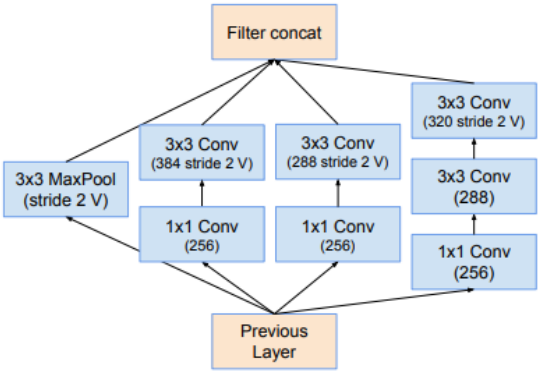
(a) Inception-Resnet-v2 A block



(b) Inception-Resnet-v2 B block



(c) Inception-Resnet-v2 C block



(d) 17×17 to 8×8 Resnet wider reduction module, (Reduction-B)

Figure 3.17: Inception-Resnet-v2 blocks (Szegedy et al., 2016).

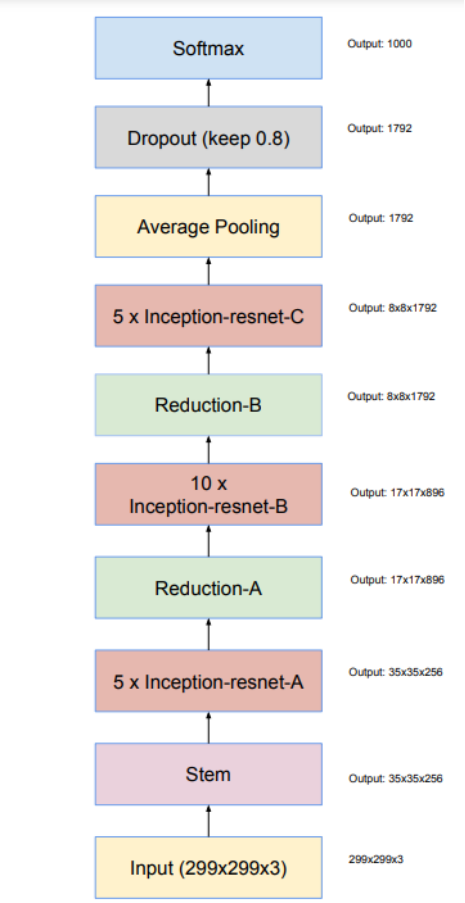


Figure 3.18: The overall schema of the Inception-Resnet-v2 network. (Szegedy et al., 2016).

Chapter 4

Dataset and network implementation

4.1 The dataset

4.1.1 The MOKA software

In this work, we analyze a dataset consisting of weak lensing maps created using the MOKA software (Giocoli et al., 2012a). This algorithm, starting from analytical prescriptions, creates surface mass density distributions of triaxial and substructured haloes. This approach is a valid alternative to full numerical hydrodynamical simulations for several applications, including gravitational lensing studies. Indeed, with a limited computational cost, it achieves very high spatial resolution, which is necessary to resolve the inner structure of galaxies and clusters. This advantage comes at the expense of the degree of realism of the simulated mass distributions, which do not contain features such as asymmetries that are difficult to describe analytically.
The idea of this tool is to construct realistic lenses starting from a set of ingredients taken from state-of-the-art numerical hydrodynamical simulations. Moreover, MOKA takes into account not only the smooth dark halo and stellar components as earlier studies have done (Mandelbaum et al., 2009; van de Ven et al., 2009), but also the presence of substructures that perturb the regular matter distribution.
Each halo has a virial mass defined as

$$M_{vir} = \frac{4\pi}{3} r_{vir}^3 \frac{\Delta_{vir}}{\Omega_M(z)} \Omega_{M,0} \rho_c, \tag{4.1}$$

where ρ_c and Ω_M are the critical density of the Universe and the matter density parameter respectively (see Sect. 1.1.2). The term Δ_{vir} (which is ~ 178 from Cole & Lacey (1996) for the Einstein-de Sitter model) is defined as the virial overdensity enclosed in the virial radius of the halo r_{vir}, which is the radius that separates the region of the

halo in dynamical equilibrium from the surrounding region.
Each halo is also characterized by a dark matter density distribution that follows the NFW profile (Navarro et al., 1996), described by

$$\rho(r) = \frac{\rho_s}{\left(\frac{r}{r_s}\right)\left(1+\frac{r}{r_s}^2\right)}, \tag{4.2}$$

where r_s is a scale radius and ρ_s is defined as the dark matter halo density at the scale radius. Another important parameter describing halos and related to the NFW profile definition is the concentration c_{vir}. This parameter is defined as the ratio between the virial radius and the scale radius:

$$c_{vir} = \frac{r_{vir}}{r_s}.$$

The concentration parameter is intricately linked to halo mass, defining an important relation in cosmology since it is a prediction of the ΛCDM model. This correlation, called the $c - M$ relation, indicates that the concentration is associated to the average density of the Universe during the halo collapse: smaller halos, which form earlier, have greater concentration compared to larger ones presently assembling, reflecting the higher mean density of the early Universe. Consequently, this correlation is also dependant on the redshift and its temporal evolution. Therefore, the halo concentration is a decreasing function of the host halo mass. In the MOKA algorithm, the authors adopt the mass-concentration relation proposed by Zhao et al. (2009), which links the concentration of a given halo with the time at which its main progenitor assembles 4% of its mass. Because of their tidal interaction with the surrounding density field during their collapse, dark matter halos are not spherical but are characterized by triaxiality, which is modeled using the results presented in Jing & Suto (2002) based on dark matter-only simulations.
As previously mentioned, these halos are not smooth but are characterized by many substructures. Their mass distribution follows the mass function from Giocoli et al. (2010), which used data coming from the GIF2 cosmological N-body simulation investigating the substructure abundance as a function of mass and redshift. However, the subhalo mass function is in general described by a power-law with an exponential cut-off:

$$\frac{dN_M}{d\ln m_{sb}} \equiv \frac{m_{sb}}{M_0}\frac{dN}{dm_{sb}} = N_{M_0} m_{sb}^{\alpha} \exp\left(-\beta\xi^3\right), \tag{4.3}$$

where N_{M_0} is the normalization factor, N_M is the number of substructures within a certain mass, m_{sb} is the subhalo mass, M_0 is the host halo mass at $z = 0$, $\xi = \frac{m_{sb}}{M_0}$, $\alpha = -0.9$ and $\beta \sim 12.2715$. Regarding their spatial distribution, Giocoli et al. (2012a) confirmed that the radial distribution of the subhalos resembles that of the smooth dark matter main halo, although with some caveats: subhalos close to the center of the

cluster are more easily destroyed because of their tidal interaction with the main halo. Therefore, the subhalo distribution is less centrally concentrated than the dark matter distribution of the main smooth dark matter halo. The cumulative spatial density distribution of substructures in the host halo follows the study of Gao et al. (2004) and is described by

$$\frac{n(<x)}{N} = \frac{(1+ac)x^{\beta}}{(1+acx^{\alpha})}, \tag{4.4}$$

where x is the distance to the host center in units of r_{200} (corresponding to the radius at which the density is equal to 200 times the critical density of the Universe), $n(<x)$ is the number of subhalos within x, N is the total number of subhalos inside r_{200}, $a = 0.244$, $\alpha = 2$, $\beta = 2.75$, and c is the concentration of the host halo (related to r_{200}). The mass fraction contained in the subhalos with respect to the total mass of the halo is described by

$$f_s = \frac{\sum_i m_{sb,i}}{M_0}. \tag{4.5}$$

Subhalos may host satellite galaxies, which are described through a truncated SIS density profile (Keeton, 2003):

$$\rho_{sub}(r) = \begin{cases} \dfrac{\sigma_v}{2\pi G r^2} & \text{for} \quad r \leq R_{sub} \\ 0 & \text{for} \quad r > R_{sub} \end{cases}, \tag{4.6}$$

where σ_v is the velocity dispersion and $R_{sub} = \frac{Gm_{sub}}{2\sigma_v^2}$.
At last, since the lensing signal is sensitive to the matter distribution in the central region of galaxy clusters ($r \sim 100$ kpc), the brightest central galaxy (BCG) is added to the halo using the halo occupation distribution (HOD) technique, which assumes that the stellar mass of a galaxy is tightly correlated with the depth of the potential well of the halo within which it formed (Wang et al., 2006). For the BCG density profile a Hernquist (Hernquist, 1990) profile is adopted:

$$\rho_{BCG} = \frac{\rho_g}{(r/r_g)(1+r/r_g)^3}. \tag{4.7}$$

Here, ρ_g and r_g are the scale density taken from the Hernquist model and the scale radius related to the half-mass (or effective) radius respectively.
Finally, the virial mass of the halos is given by the sum of smooth plus clumpy components, and if a BCG is present: $M_{vir} = M_{smooth} + \sum_{i=1}^{N_{tot}} m_i + M_{BCG}$, where m_i is the mass of the $i-th$ subhalo and M_{BCG} is the mass of the BCG.
Then, the lensing properties are derived from the three-dimensional matter density of all components characterizing the halos. For each component, MOKA projects the density on a plane perpendicular to the line of sight and obtain analytically the projected

mass density Σ_{NFW}, Σ_{sub}, Σ_{BCG}. These quantities are then scaled with the appropriate Σ_{cr} value depending on the source and lens redshifts, assumed to be $z_s = 2$ and $z_l = 0.25$ respectively, and the total convergence map is obtained as

$$\kappa(x, y) = \kappa_{DM}(x, y) + \sum_{i=1}^{N_{tot}} \kappa_{sub,i}(x - x_c, y - y_c) + \kappa_{BCG}(x, y), \tag{4.8}$$

where x_c and y_c represent the center of mass of the $i - th$ substructure. Starting from the convergence map, the effective lensing potential and the scaled deflection angle can be obtained, and finally by calculating the derivatives of the effective lensing potential, the shear components are also obtained. Fig. 4.1 shows an example for the convergence and shear maps produced using the MOKA algorithm for a galaxy cluster acting as a lens located at $z = 0.25$.

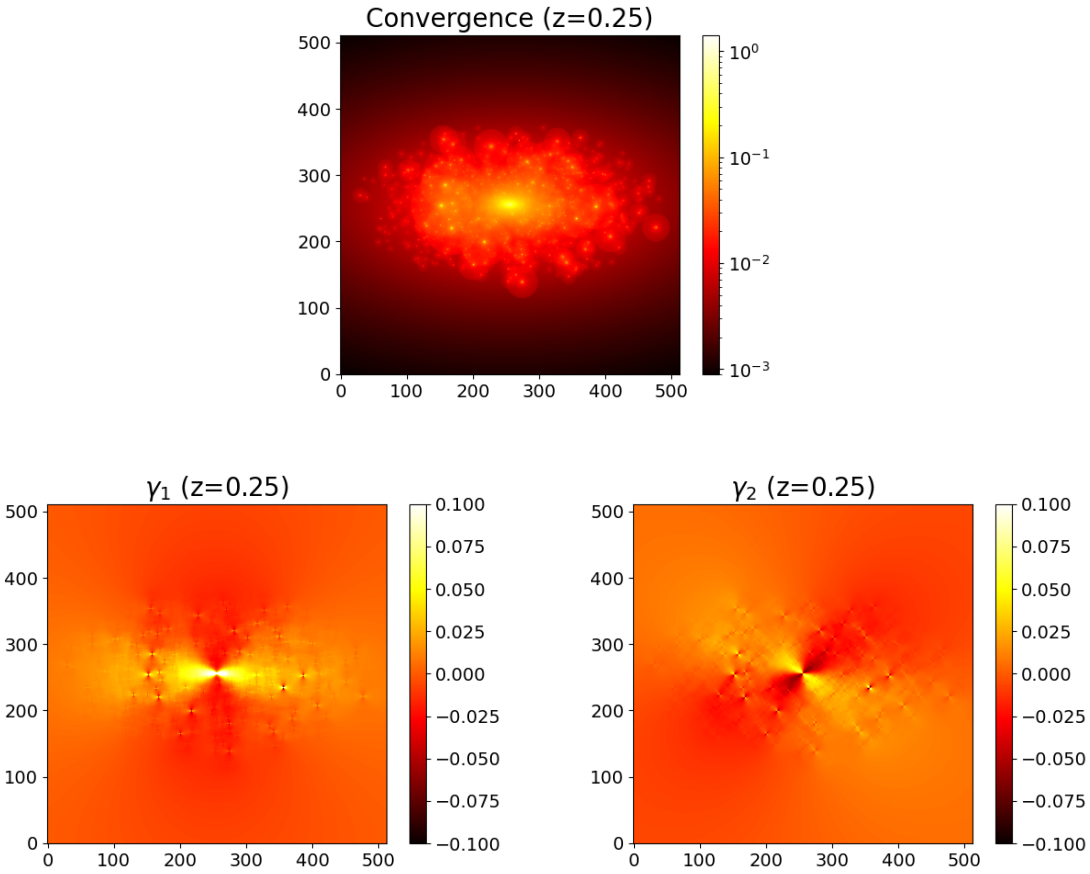


Figure 4.1: Simulated convergence (upper panel) and shear (lower panels) maps obtained through the MOKA algorithm for a cluster located at redshift $z = 0.25$.

Noise simulation Since noise can affect lensing measurements, we also conducted a test using our best model, the VGG19-heavy (see Sect. 4.2.1), to evaluate the CNNs' performance in more realistic conditions. To simulate realistic noise in future observations, we take into consideration the Euclid capabilities, i.e., its ability to measure the shapes of 30-40 resolved galaxies per $arcmin^2$ in one broad visible $R+I+Z$ band (between $550nm$ and $920nm$), covering a total of 13.245 squared degrees of the extragalactic sky in the following 6 years (Laureijs et al., 2011). We therefore assume a number density of background galaxies of $n_{gal} = 30(galaxies \times arcmin^{-2})$, responsible for galaxy shape noise in the reduced shear maps.
Then, we assume an intrinsic ellipticity distribution of the sources characterized by a dispersion of $\sigma_\eta = 0.3$. This intrinsic ellipticity is dominant for the distortion introduced by the lensing effect, meaning we need to dilute the intrinsic term by mediating on a large number of sources, hence averaging zero and making the lensing signal measurement possible.
Our shape noise per pixel is then calculated as described in Giocoli et al. (2010):

$$\sigma_{noise,pixel} = \sqrt{\frac{\sigma_\eta^2}{(2 \cdot n_{gal} \cdot p_{size,arcmin}^2)}}, \tag{4.9}$$

where $p_{size,arcmin}$ is the dimension of our pixel in arcminutes. We then generate noise on the reduced shear maps, assuming a Gaussian distribution, with zero mean and $\sigma = \sigma_{noise,pixel}$. An example of the resulting noisy reduced shear maps is given in Fig. 4.2.

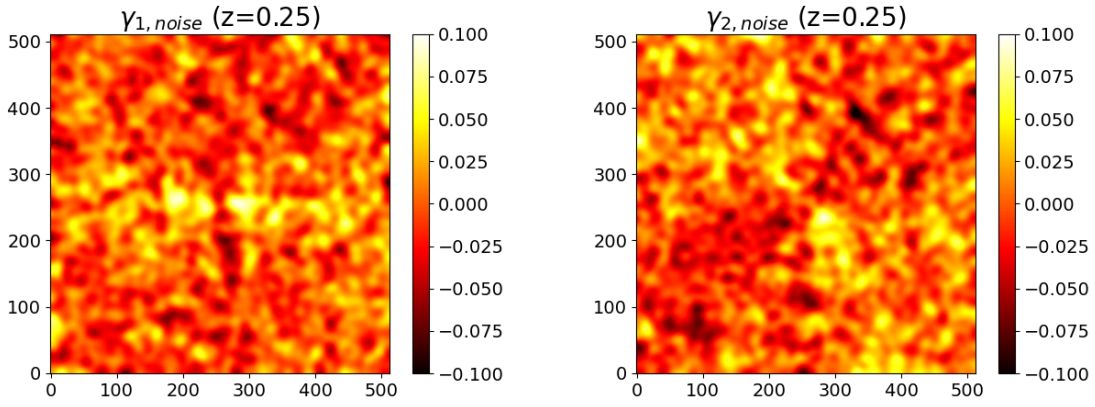


Figure 4.2: Noisy reduced shear maps for a cluster at redshift $z = 0.25$. This is the same cluster shown in Fig. 4.1.

4.1.2 Dataset properties

The previously described maps are stored as FITS (Flexible Image Transport System) files, a digital file format useful for storage, transmission, and processing of data organized in multidimensional arrays (2D images, in this case).
Every cluster file has three extensions: one for the convergence and two for the two shear components. All the files also come with a *header* unit, namely, a text extension that contains either the labels we want to predict with the networks or other parameters associated with the cluster maps (e.g., the cosmological parameters involved in the simulation, the ones related to the halo triaxiality, image dimensions in pixels, and more). To feed our networks with the appropriate training data, we first need to extract the labels from the headers, selecting only the parameters that our networks have to predict and store both the images and the labels into a single tensor. The labels are the halo virial mass M_{vir}, the concentration of the NFW profile of the halo c_{NFW}, the concentration of the NFW profile of the smooth component of the halo $c_{smooth,NFW}$, the number of subhalos N_{sub} and the mass fraction in subhalos compared to the total halo mass, f_{sub}.
The extraction process involves the entire dataset, which includes a catalog of 100.000 simulated clusters. Then, the whole dataset is split into a training set featuring 75.000 maps, a test set with 20.000 maps, and a validation set for the accuracy (defined in Sec. 4.3) calculation with the remaining 5.000 maps.
Fig.4.3 and Fig. 4.4 show the distributions of the training and test set parameters, respectively. It is fairly straightforward to observe that these plots cover the same range of values, with very similar mean and median values (represented as a red and a green dotted line respectively), meaning the test set is a reliable depiction of the dataset.
After the label extraction and storage, the last step in data preprocessing consists of normalizing the distribution of each parameter by subtracting the mean value and dividing it by the standard deviation to obtain values varying in the same range, i.e., between 0 and 1. Normalizing the data helps bring features to a similar scale, preventing some features from dominating others during training. This uniformity aids in the convergence of the optimization algorithm, allowing the model to learn more efficiently and generalize better to unseen data by reducing the influence of biases in the input data. Normalization also makes the optimization process more stable by preventing large gradients that might lead to oscillations or convergence issues, ensuring that the updates of the weights are consistent and predictable. Moreover, by normalizing the data, we reduce the computational complexity by helping the model converge faster. This efficiency is particularly important when dealing with large datasets and complex architectures, like in this case.
Once the labels are normalized, we finally proceed by resizing the images from the original 512×512 pixels to the appropriate input sizes of the VGG-Net and Inception networks, which are respectively 224×224 pixels and 299×299 pixels. Images are also

normalized with a zero mean and a $\sigma = 1$ for the same previously described purposes.

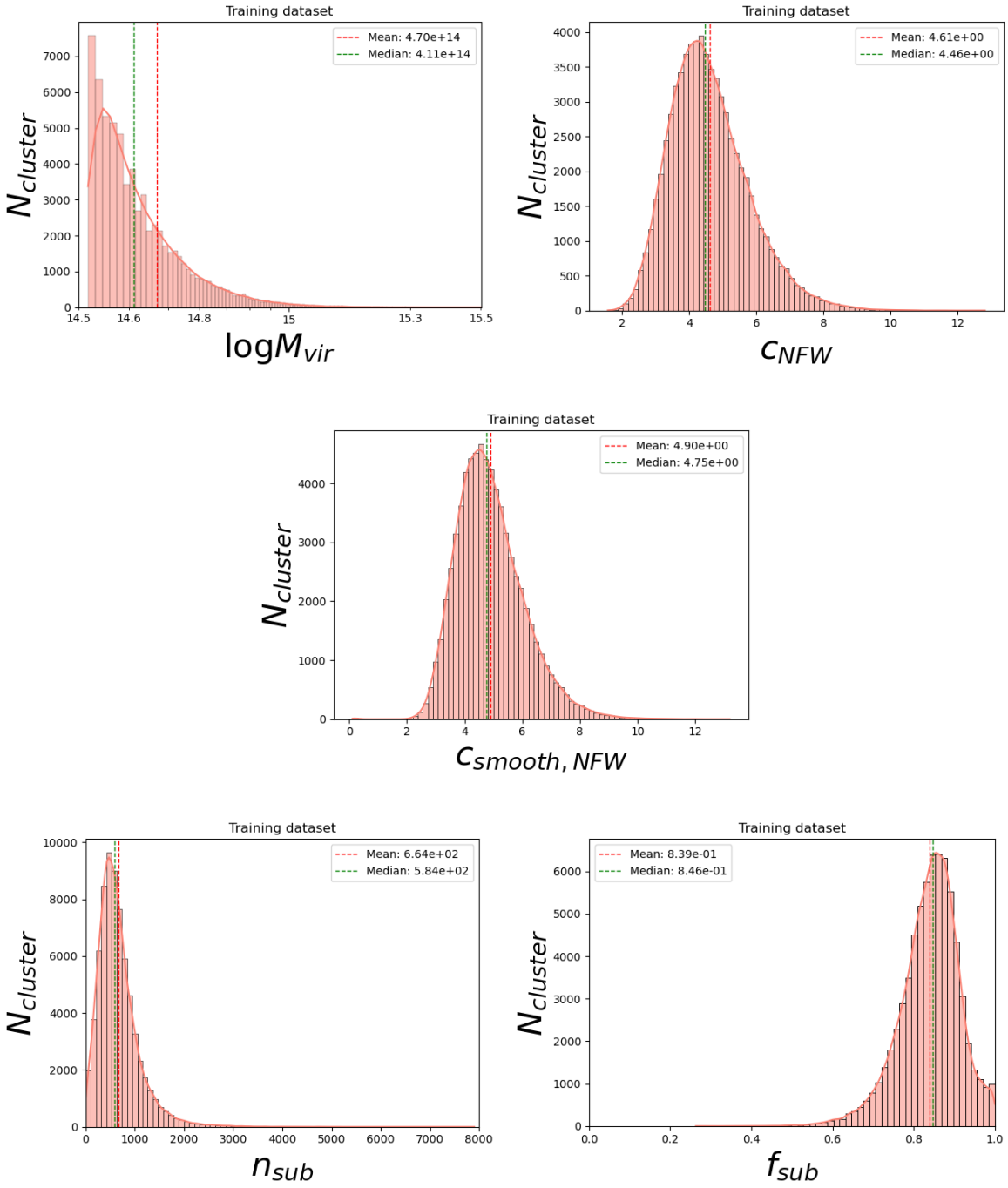


Figure 4.3: Training dataset parameters distribution, where $N_{cluster}$ refers to the number of clusters. All the clusters are located at $z = 0.25$. A vertical dotted line representing the mean (red) and median (green) value is drawn for every parameter. The solid line represents the KDE probability function.

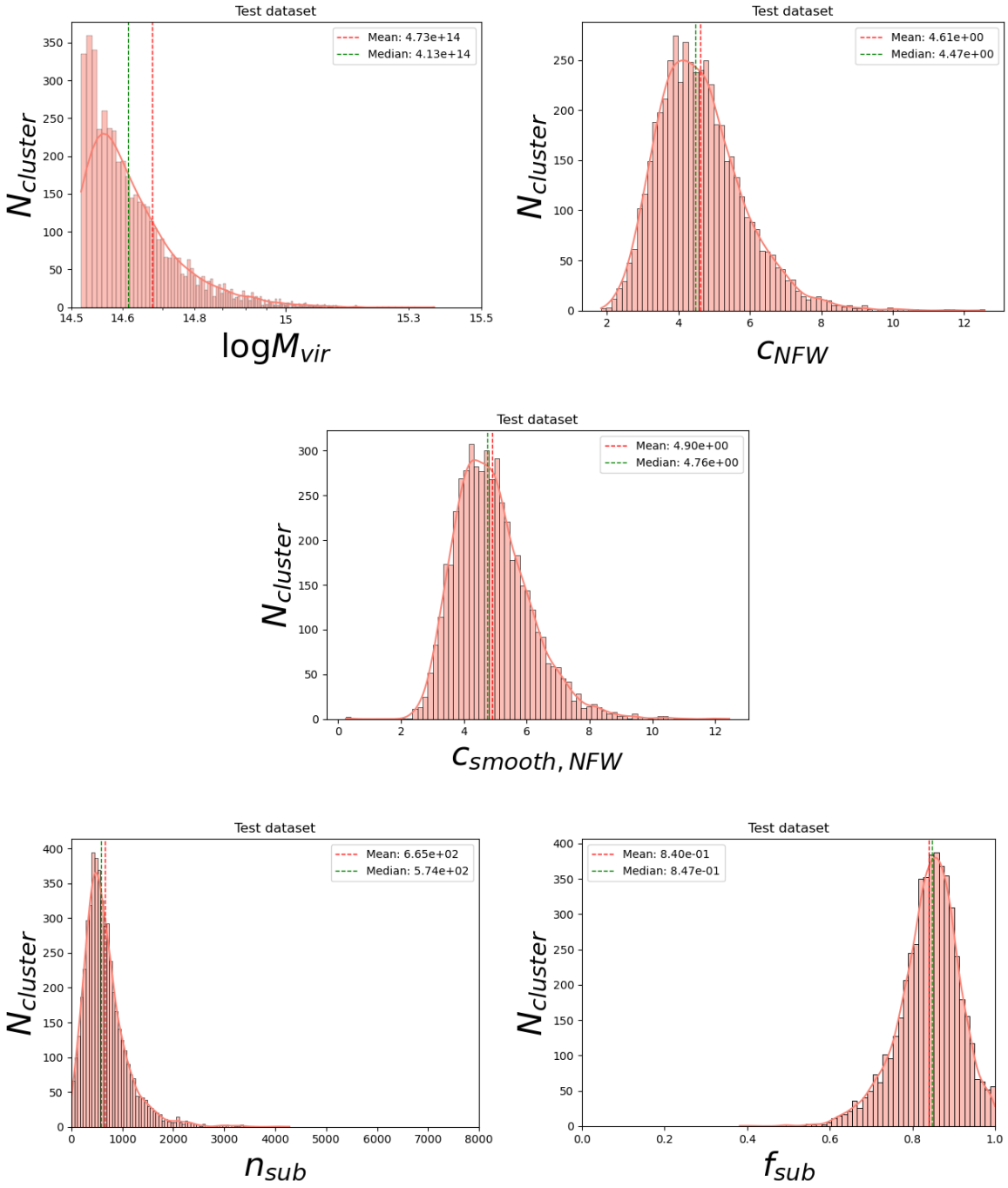


Figure 4.4: As for Fig. 4.3 but for the test dataset.

4.2 Network's implementations

In this section, we introduce the models used in this study. All of our models are developed using PyTorch (Paszke et al., 2019) and have been adapted from their original versions, as described below. Specifically, the models are meant to be fed with the two shear component maps derived from the MOKA simulation as input, hence having dimensions of $224 \times 224 \times 2$ for the VGG-inspired models, and $299 \times 299 \times 2$ for the Inception ones. All the models then produce an output tensor of dimension 1×5 consisting of five elements representing the predicted parameters of the cluster.

4.2.1 VGG-Net

In our VGG-Net implementation, we followed the blueprint of the original D configuration (see Tab.3.1), commonly referred to as "VGG16" due to its composition of 16 convolutional layers. Its detailed structure can be seen in Tab.3.1. While the original VGG16 model is designed for classification problems, our network has to perform a regression task, as our goal is to predict numerical values for five parameters that define cluster halos. To accommodate this requirement, we replaced the original last layer's soft-max activation function, which yields class membership probabilities, with a Leaky ReLU activation function, as we want a function capable of producing continuous numerical values for each parameter instead of class probabilities. Every convolution layer is also followed first by a batch normalization layer and later by a Leaky ReLU activation function, with a negative slope equal to 0.3 (see Eq.3.1). A detailed example of the convolution layer employed for VGG and Inception models is shown in Tab.4.2. Next, we adjusted the dimensions of the FC layers preceding the last Leaky ReLU activation. This adjustment was necessary since the original VGG-Net is designed for 1000 classes, whereas we only need to predict five parameters. To do so, since our research focuses on exploring the influence of different architectures on the prediction accuracy of cluster's parameters, as well as assessing how a single architecture might yield improved performance, we developed and evaluated multiple VGG-Net models, and FC layers underwent distinct adjustments from one model to another. The overall structure of the models is shown in Tab.4.1. Their differences extend beyond mere structural distinctions (i.e., the number of parameters involved) and also lie in the selection of learning rates along with their variations during the training phase and the inclusion of *dropouts*, where this last term refers to a technique applied within the last FC layers of the network.
The dropout technique (Srivastava et al., 2014) consists of randomly dropping units in the network during training. In particular, dropout layers temporarily remove certain units from the network, including their incoming and outgoing connections. The units selected for removal are chosen randomly, each with an independent fixed probability denoted as p. In our case, we have set a dropout rate of $p = 0.3$, indicating that 30% of

the following connections are randomly eliminated. Implementing dropout in a neural network involves creating a "thinned" network comprised solely of the remaining units after dropout. With each presentation of a training batch, a new thinned network is sampled and trained. Consequently, training a model with dropout can be likened to training a collection of thinned networks. At test time, a single model without dropout is used: the weights of this network are scaled-down versions of the trained weights. In particular, if a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at the test time
The following paragraphs delineate the structure and the underlying concepts of the models.

VGG16-basic The first model was designed to closely resemble the original VGG-16 architecture regarding layer composition. It adopts the structure outlined in the first column of Tab.4.1, featuring only 3 FC layers at the end, thereby totaling 16 layers overall. The substantial difference with respect to the original VGG-Net lies in the channel count per layer. Specifically, VGG-basic is distinguished by its reduction to 2.5 million parameters, which was made to minimize computational expenses and evaluate the performance of a very basic network compared to more complex models. This network was tested either with or without dropouts applied after every of the final FC layers (except for the last one, given that it gives the predictions).

VGG19 The following models were instead designed to be comparable with the Inception and Resnet networks regarding the number of parameters. Hence we created two different types of "heavier" VGG networks (each of whom is characterized by 19 layers), called respectively **VGG19-light** and **VGG19-heavy**. Their detailed structure is shown in the second and third columns of Tab.4.1, and their main difference with the original VGG-Net architecture lies in the number of final FC layers, as these are the layers demanding the most number of parameters. The term "light" denotes fewer channels utilized for the convolution layers in this specific model than the "heavy" counterpart, which maintains the same convolution layer structure as the VGG-Net D configuration. Consequently, the VGG19-light model is characterized by 30 million parameters, while 69 million parameters describe the VGG19-heavy one. Our purpose behind the creation of two distinct models was twofold: first, to create a lighter model for comparison against the Inception and ResNet networks (with $\sim 20/30$ millions of parameters), and second, to construct a heavier model to assess how accuracy evolves with increasing network complexity, considering that our dataset consists of relatively simple maps with no particular features. Hence, it is uncertain whether augmenting complexity will necessarily result in performance enhancements.
Since the training phase improved the heavier model's performance, we also evaluated its results by incorporating dropouts. Specifically, we conducted three tests by adding

one, two, and three dropout layers in the final FC layers (after the FC-2048 layer, the FC-512 layer and the FC-128 layer respectively) to assess the effects of this technique on the performances. Results can be seen in Sect. 5.

Table 4.1: Detailed structures of the different VGG-inspired models implemented in this work. The convolution layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The batch normalization layer and the Leaky ReLU activation function are not shown for brevity.

16-base	**19-light**	**19-heavy**
input (224 × 224 2D image)		
conv3-16 conv3-16	conv3-16 conv3-16	conv3-64 conv3-64
maxpool		
conv3-32 conv3-32	conv3-32 conv3-32	conv3-128 conv3-128
maxpool		
conv3-64 conv3-64 conv3-64	conv3-64 conv3-64 conv3-64	conv3-256 conv3-256 conv3-256
maxpool		
conv3-128 conv3-128 conv3-128	conv3-128 conv3-128 conv3-128	conv3-512 conv3-512 conv3-512
maxpool		
conv3-128 conv3-128 conv3-128	conv3-256 conv3-256 conv3-256	conv3-512 conv3-512 conv3-512
maxpool		
FC-256 FC-128	FC-2048 FC-1024 FC-512 FC-256 FC-128	FC-2048 FC-1024 FC-512 FC-256 FC-128
FC-5		

4.2.2 Inception networks

In implementing the Inception networks, we began by developing the stem for both the Inception-v4 and ResNet-v2 architectures, as they share the same initial structure. The stem was implemented exactly as shown in the left panel of Fig.3.14. We then created the following models based on the Inception-v4 and ResNet-v2 architectures. It is important to note that our base convolutional block is composed of three stacked layers imported from the Pytorch library, as shown in Tab. 4.2:

Table 4.2: Scheme of the basic convolutional block employed in the VGG-Net, Inception-v4, and Inception-ResNet-v2 models. Conv2D, BatchNorm2d and LeakyReLU are commands imported in Python from the Pytorch library. In the VGG-Net models, the *kernel_size* is fixed at 3×3, with the stride and padding both set to 1. Conversely, for the Inception architectures, the Conv2D settings are adjusted according to the authors' recommendations outlined in the original papers.

Conv. block
Conv2D(*in_channels*, *out_channels*, *kernel_size*, *stride*, *padding*, *bias*)
BatchNorm2d(*out_channels*, $eps = 10^{-5}$, $momentum = 0.1$)
LeakyReLU($negative\_slope = 0.3$, $inplace = True$)

The Conv2D configurations vary from one layer to the next. As for the batch normalization layer, we set the ϵ value (*eps*) to 10^{-5}, which is added to ensure numerical stability within the batch normalization expression. Additionally, this layer maintains ongoing estimates of its calculated mean and variance, crucial for normalization during evaluation, with a default *momentum* of 0.1. Regarding the LeakyReLU activation layer, we opt for a negative slope value of 0.3, consistent with the VGG implementation.

Inception-v4 For our Inception-v4-based network, we created the three building blocks shown in Fig.3.12, namely, Inception A-block, B-block, and C-block. After their definition, we proceeded by also developing the two reduction modules (called Inception reduction-A and reduction-B modules) shown in Fig.3.9, remembering that the V-marked convolutions are valid padded. In contrast, the convolutions without this specification are zero-padded. When developing the Reduction-A module, we employed filter bank sizes of $k = 192$, $l = 224$, $m = 256$, and $n = 384$, mirroring the dimensions

utilized in the original Inception-v4 research (Szegedy et al., 2016). We then stacked all the building blocks and reduction modules in the correct order, following the scheme shown in the right panel of Fig.3.14, along with substantial differences.
First of all, as previously described for the VGG case in Sec.4.2.1, we substituted a Leaky ReLU activation function to the original final softmax layer of Inception-v4 since we tackle a regression problem. Afterward, we created two distinct models, namely **Inception-v4 "light"** and **Inception-v4 "heavy"**. The first is characterized by only one Inception module of each type, as shown in the first column of Tab.4.3, and is described by 15 million of parameters. The second one is instead described by three Inception A-blocks, followed by five Inception B-blocks and then three Inception C-blocks, and is characterized by 36 million parameters, thus being the "heavy" counterpart. Its structure is shown in the second column of Tab.4.3. The purpose behind creating two separate models was to explore the relationship between the complexity of the networks and their performance.
In the original work, the authors include a dropout layer before the softmax activation function, recommending keeping $p = 0.8$. We also tested our Inception network, taking an intermediate model between our light and heavy networks called **Inception-drop** and implementing two dropout layers. In particular, these dropouts were placed after the FC-768 and FC-384 layers. This last Inception network has the same structure as the light model but with two Inception modules of each type and is described by 23 million parameters. This choice is due to a previous work conducted on the same weak lensing maps (Spinelli, 2021) that pointed out how this architecture was the best compromise between accuracy and time-consumption for the Inception-based models.

Inception-ResNet-v2 In implementing the Inception-ResNet-v2 architecture, we followed the procedures outlined in the previous paragraph. We then started by creating the three building blocks (IncRes A,B and C) exactly as shown in Fig. 3.15, along with the reduction modules, shown in Fig. 3.13(a) and Fig. 3.15(d). As can be seen, the reduction-A module employed for the ResNet architecture is the same developed for the Inception-v4 network, the only difference being the filter bank sizes employed, now corresponding to $k = 224$, $l = 224$, $m = 384$, and $n = 384$. We then stack the blocks and reduction modules following the scheme in Fig.3.18 and employing the stem shown in Fig.3.14(a). At last, as carried out in the previous paragraph, we substituted the final softmax activation function with a LeakyReLU.
For this architecture, two models were created, along the same lines as the Inception models, called **IncResNet-v2 "light"** and **IncResNet-v2 "heavy"**. The first one is characterized by one IncRes module of each type and follows the scheme shown in the third column of Tab.4.3, with 17 million parameters. Its heavier counterpart is instead described by three IncRes A-blocks, five IncRes B-blocks, and three IncRes C-blocks, as shown in the fourth column of Tab.4.3. This model is characterized by a total of 26

million parameters.
As dropout techniques were found to degrade the network's performance in our Inception model, they were not tested for this architecture.

Table 4.3: Detailed structures of the different Inception-inspired and ResNet-inspired models implemented in this work.

Inception-v4 "light"	**Inception-v4 "heavy"**	**IncResNet-v2 "light"**	**IncResNet-v2 "heavy"**
input (299 × 299 2D image)			
STEM			
1×Inc. A-block	3×Inc. A-block	1×IncRes A-block	3×IncRes A-block
Reduction-A			
1×Inc. B-block	5×Inc. B-block	1×IncRes B-block	5×IncRes B-block
Reduction-B			
1×Inc. C-block	3×Inc. C-block	1×IncRes C-block	3×IncRes C-block
Average pooling			
FC-1536			
FC-768			
FC-384			
FC-192			
FC-96			
FC-5			

4.3 Training the networks

Our model training was performed using an NVIDIA Titan Xp GPU, equipped with a memory of 12 GB and 3840 CUDA cores. Employing a GPU substantially reduces the computation time and speeds up the training process of the artificial neural networks. Before training the models, it is essential to define the number of epochs, which refers to how many times the network processes the entire dataset. Specifically, our models begin with a predefined value of 100 epochs to enable the learning algorithm to effectively

minimize the loss. However, this value serves as an upper limit due to incorporating an *early stopping* mechanism in the training code. This mechanism dictates that the training process halts if the loss function fails to improve within a specified number of epochs. Then, the final weights and biases being saved correspond to the ones linked to the minimum loss achieved during the training phase. This condition becomes active starting from the 40^{th} epoch, establishing the minimum threshold epochs to reach.
A comparable consideration applies to the learning rate management in the models. The learning rate is perhaps the most critical hyperparameter during neural network training, as it dictates the magnitude of adjustments made to network weights to minimize loss. At the start of the training phase, we set the learning rate to 10^{-5}. We then implement a condition that halves the current learning rate value if the minimum of the loss function observed over the preceding 20 epochs exceeds its absolute minimum. Employing this technique alongside the early stopping condition helps conserve computational resources and prevent the training algorithm from overfitting. The choice of the initial learning rate value (10^{-5}) derived from a prior study that employed either ML algorithms and MOKA simulated maps, as outlined in (Spinelli, 2021). In our work, we also conducted tests on the three different network archetypes with higher starting learning rate values (10^{-3} and 10^{-4}), finding that their convergence to the minimum was unstable and prone to oscillations. Therefore we opted for the previously proven 10^{-5}. This conclusion is based on the statistical estimator values employed to analyze the network results (i.e., bias, σ, RMS, MAD, and NMAD), defined in Chapter 5.
Moreover, we selected the Adam (Adaptive Moment Estimation) optimizer (Kingma & Ba, 2017; Reddi et al., 2019). This optimizer combines the benefits of both Momentum Optimization and RMSprop. Momentum Optimization helps the gradient descent algorithm converge faster and more reliably by adding a term incorporating the past weights update history, leading to more stable and efficient updates towards the optimal solution. On the other hand, RMSprop (Root Mean Square Propagation) is an unpublished adaptive learning rate optimizer proposed by Geoff Hinton. It adjusts the learning rate adaptively for each parameter based on the magnitudes of recent gradients by dividing the learning rate by the root mean square of past gradients. This operation normalizes the weights update step, making it more consistent across different dimensions and improving convergence.
Adam's main characteristics lie in its adaptive learning rate mechanism, which adjusts the learning rates for each parameter based on the past gradients and squared gradients. This adaptivity allows Adam to converge quickly and efficiently, especially in scenarios with large datasets and high-dimensional parameter spaces like in this work. Additionally, Adam incorporates bias correction to counteract the effects of initial learning rate bias and momentum correction, contributing to more stable and reliable convergence behaviors.

In conclusion, we chose the MSE as the loss function for our models (see Eq.3.3). Our accuracy calculation relies on a function we implemented. It evaluates each output prediction from the test set and determines if its value deviates by less than 20% from the corresponding correct value stored in the labels. If the output prediction value differs less than 20% from the real target, then it is considered as correct.

Chapter 5

Results

5.1 Statistical estimators

In this section, we present the results obtained by our networks. The traditional techniques for error analysis rely on the Bayesian framework, which provides a formalism for updating beliefs or making predictions in light of new evidence.
The standard error analysis with Bayesian methods typically involves estimating the probability of the outcomes based on prior knowledge and updating that probability as new evidence becomes available. Bayesian methods start with an initial belief about the likelihood of different outcomes. This initial belief is represented by a *prior probability distribution*, which accounts for what is known or assumed about the parameters or variables of interest before observing any data. As new data or evidence becomes available, Bayesian methods incorporate this information to update the prior beliefs. This update is performed using the Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{5.1}$$

where A and B are events (i.e., a set of outcomes of an experiment to which a probability P is assigned) and $P(B) \neq 0$. In our case A is the parameter we want to update and B is our dataset. The Bayes theorem then relates the *posterior probability* $P(A|B)$, which is the probability of event A occurring given that B is true, to the prior probability $P(A)$, which in this example is the probability of observing A without any given conditions, and with the likelihood of the data given the parameters $P(B|A)$. It can be demonstrated that $P(B|A)$ is equivalent to $L(A|B)$, defined as the *likelihood function* (the probability of observing B assuming A is the actual parameter). The application of Bayes' theorem then provides the posterior distribution, representing the updated belief about the parameters or variables of interest after considering the observed data. The posterior distribution combines the prior knowledge with the information provided

by the data.
While Bayesian approaches have been applied in machine learning and neural networks (Srivastava et al., 2014), calculating posterior distributions can be computationally intensive, especially for high-dimensional models like CNNs, since the computational cost of performing Bayesian inference scales with the number of parameters in the model, therefore making it challenging to apply directly to deep neural networks with millions of parameters, like VGG-Net or Inception. Furthermore, CNNs typically rely on optimization techniques based on the gradient descent algorithm (that has a stochastic nature) to learn from data, which may not directly align with the probabilistic framework of Bayesian inference, involving methods like Markov-chain Monte Carlo (MCMC) to approximate posterior distributions.
Therefore, we conduct a statistical error analysis for our networks employing various metrics. Let us define y and $\hat{y}$ as the predicted output value array and the real output value array, with their difference given by $\Delta y = y - \hat{y}$. Assuming N is the total number of test examples, we define:

- the bias, as

$$bias(\hat{y}) = E[\hat{y}] - y = \frac{1}{N}\sum_{i=1}^{N} |\Delta y_i^2|, \tag{5.2}$$

  which refers to the difference between the expected value of an estimator ($\mathrm{E}[\hat{y}]$) and the true value of the parameter being estimated. In the context of CNNs, the bias assesses how accurate the predictions of the model are on average, compared to the true values;

- the standard deviation (σ), as

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta y_i - \langle \Delta y \rangle)^2}, \tag{5.3}$$

  which measures the dispersion of a dataset relative to its mean. In the context of CNNs, σ quantifies the spread of errors in the predictions, similar to the MAD (see below) but taking into account the square of the deviations;

- the Mean Absolute Deviation (MAD), as

$$\mathrm{MAD} = \frac{1}{N}\sum_{i=1}^{N} |\Delta y_i - \langle \Delta y \rangle|, \tag{5.4}$$

  which is instead a measure of the average absolute deviation of predictions from the mean, less affected by the outliers of the dataset than σ or the RMS (see below). It provides insights into the variability of the data and helps quantify the spread of errors in the predictions made by the CNNs;

- the Normalized MAD (NMAD), as

$$\text{NMAD} = \frac{\text{MAD}}{\text{MAD}_{\text{FM}}}, \tag{5.5}$$

  which is the MAD normalized by a scale factor, often assumed to be the median absolute deviation from the median (MAD_{FM}, in this case $\text{MAD}_{\text{FM}} = 1,4826$). The NMAD is useful for comparing the spread of errors across different datasets or models and can be used as σ;

- the Root Mean Squared error (RMS), as

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\Delta y_i^2)}, \tag{5.6}$$

  which measures the average magnitude of the errors, is useful for evaluating the overall performance of a predictive model.

5.2 VGG-Net results

In describing the results obtained by our VGG-Net models, we initially provide a broader presentation regarding the structural differences of the various networks, along with their respective performances. In Subsect. 5.2.2 we present the results obtained from the network with the best accuracy achieved in this Thesis work, considering not only computational metrics but also insights from a physical standpoint. As described in Sect. 4.3, our accuracy is calculated on 5000 test set maps with a function we implemented. This function computes the percentage of the predicted parameters that closely match their corresponding true parameter value, allowing for a maximum deviation of 20%.

5.2.1 Computational results

VGG16-basic We start by presenting the performance of our initial model, VGG16-basic. Its training involved a batch size of 75 elements over 44 epochs, lasting approximately 50 hours and reaching an accuracy on the test set of 92.2%. Fig. 5.1 shows the loss function trend as a function of the number of epochs. The minimum value reached by the loss is 0.080
We also tested this model by incorporating two dropout layers while maintaining the same batch size. Throughout this evaluation, the training lasted nearly 80 hours, spanning 43 epochs, representing a 60% increase in time compared to the model without dropout layers. However, the dropout technique did not yield the desired improvement.

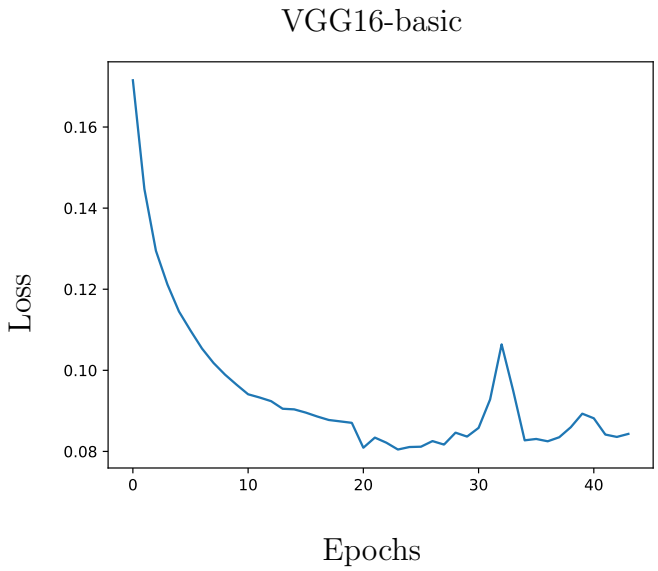


Figure 5.1: Loss trend for the VGG16-basic model.

Specifically, the loss function only reached a minimum value of 0.116, corresponding to an accuracy of merely 90.1%.

VGG19 models Regarding our two VGG19 models, we start by presenting the results of the lightest version. The training of VGG19-light involved a batch size of 50 elements over 73 epochs, lasting approximately 83 hours and reaching an accuracy on the test set of 94.8%. The upper panel of Fig. 5.2 illustrates the trend of the loss function: as shown here, the training presents a visible instability, characterized by convergence to the minimum within just 15 epochs, followed by oscillations in the loss function. Repeating the training with different hyperparameters gave even worse results. The loss function reached a minimum value of 0.078 in its best test. Our observation suggests that the network's performance reaches a plateau beyond this point, leading to instabilities during the learning phase. This conclusion is drawn from the initial descent trend, which exhibits a stable pattern but subsequently loses coherence. On the other hand, the VGG19-heavy model was the best predicting model for this work. Its training involved a batch size of 30 elements over 50 epochs, lasting approximately 47 hours and reaching an accuracy on the test set of 96.7%. For the VGG19-heavy model, the loss function reached a minimum value of 0.067, as shown in the lower panel of Fig. 5.2. The decrease in the loss value shows consistent progress, converging to the minimum in approximately 30 epochs. Our following tests, starting from the endpoint of the previous training, showed no improvements in accuracy. As outlined in Sec. 5.3, the VGG19-heavy model exhibits a superior performance overall

but the longest convergence time. It is worth mentioning that the constraints in this domain arise from our limited computing resources and the difference in the heaviness of the networks (69 million for VGG19-heavy compared to 36/26 million for the Inception architectures). With increased RAM availability, we could have trained on larger batch sizes, significantly reducing the computation time.

The VGG19-heavy model was further tested with one, two, and three dropout layers, respectively (see Sec. 4.2.1), to characterize this technique's impact on the model's performance. The corresponding models are here named VGG19_drop-1, VGG19_drop-2 and VGG19_drop-3, according to the number of dropout layers implemented in the network. Tab. 5.1 summarizes the three dropout models' performance and training characteristics.

Table 5.1: Performances of the dropout versions of the VGG19-heavy model.

Network	Time (hrs)	Epochs	Loss	Accuracy (20%)
VGG19_drop-3	80	92	0.094	92.2%
VGG19_drop-2	91	68	0.075	94.0%
VGG19_drop-1	83	61	0.071	94.5%

As expected from the previous test on the VGG16-basic, incorporating dropouts into our VGG-inspired models results in a decline in network performance, yielding a generally lower accuracy compared to dropout-free models. As the number of dropout layers increases, the accuracy progressively diminishes, indicating that this technique is ineffective for our dataset and should be avoided. Fig. 5.3 illustrates the loss trends of the VGG19_drop models. The analysis of these plots reveals a correlation between the duration of the initial fast descent trend in the loss and the number of dropout layers, with the VGG19_drop-1 model achieving the minimum loss in just seven epochs (after that epoch, the same considerations made for the VGG19-light model apply). With an additional dropout layer, the rapid descent extends to 11 epochs. However, it should be noted that although employing a single dropout layer at the start of the FC section results in a lower accuracy compared to our top-performing model, it significantly accelerates the training process, achieving a 94.5% accuracy within less than 10 hours. Despite similar losses and nearly identical accuracy, the primary distinction between using one and two dropout layers lies in the increased computational time required for the VGG19_drop-2 model. Introducing a third dropout layer after the FC-128 layer, as described in Sec. 4.2.1, significantly slows the convergence to the minimum of the loss and impacts accuracy accordingly.

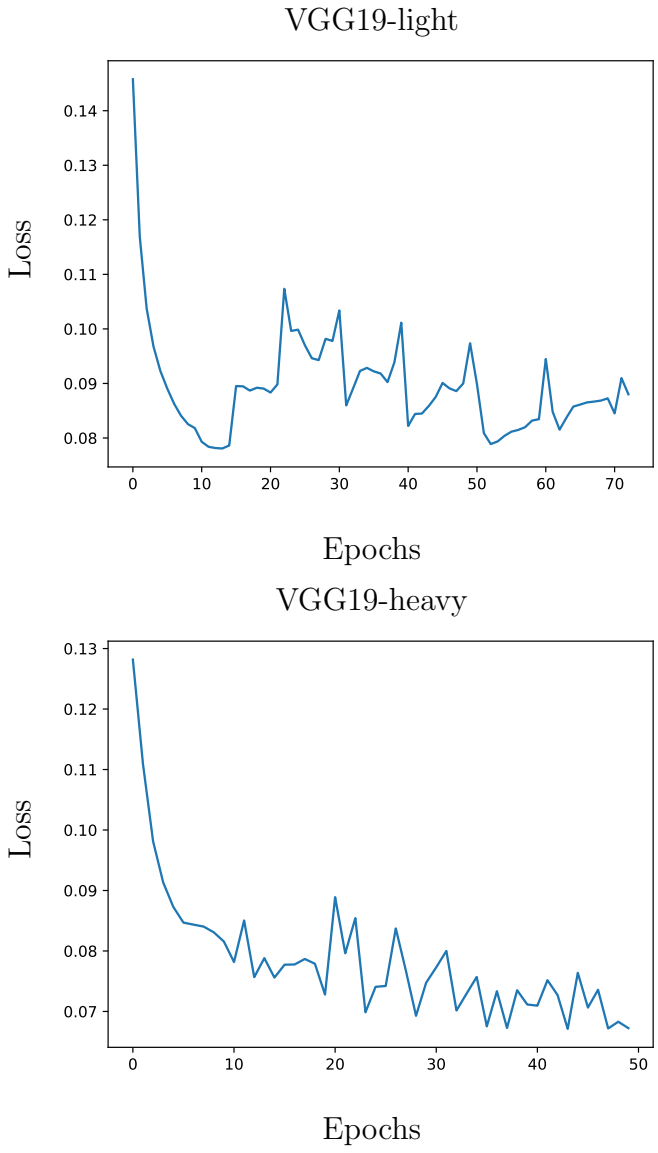


Figure 5.2: Loss trend for the VGG19 models. The upper panel shows the trend for the VGG19-light, the bottom one shows the trend for the VGG19-heavy.

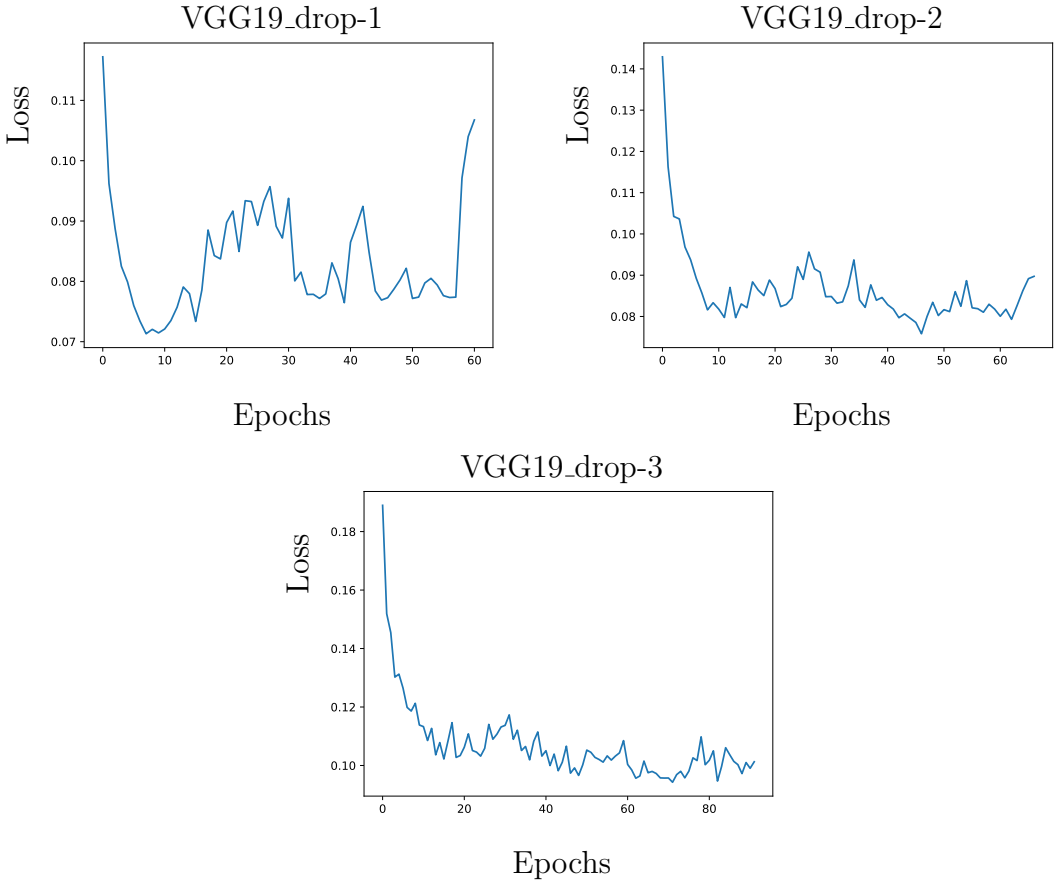


Figure 5.3: Loss trend for the VGG19_drop models. The upper-left panel shows the trend for the VGG19_drop-1. The upper-right panel shows the trend for the VGG19_drop-2. On the bottom one, the trend for the VGG19_drop-3 is plotted.

5.2.2 Practical results and application

VGG19-heavy In the VGG19-heavy model analysis, we provide a visual representation of the parameter results through Kernel Density Estimate (KDE) plots shown in Fig. 5.4, facilitating a straightforward comparison between predicted and target values. A KDE plot is a type of data visualization technique used to estimate the probability density function of a continuous random variable. It provides a smooth representation of the underlying distribution of the data and is commonly used in data analysis to understand the shape and characteristics of datasets. As shown in Fig. 5.4, it is evident that the dispersion of virial mass is notably greater compared to the other parameters. This dispersion primarily stems from projection effects; specifically, the target virial mass assigned to each dark matter halo represents the mass within a sphere of radius

R_{vir}, whereas the network estimates it based on the projected mass distribution. It is important to note that the halos exhibit a triaxial shape and are projected along arbitrary lines of sight. A similar logic applies to the NFW concentration parameter (shown in the upper-right panel of Fig. 5.4); however, it appears less influenced by projection effects. This is attributed to its definition as the ratio between the virial and scale radii. Consequently, when projecting a 3D halo onto a 2D plane, both radii are rescaled similarly, resulting in less susceptibility to triaxiality in the ratio between these quantities. In Fig. 5.5, we present the distribution of true versus predicted values obtained by evaluating our VGG19-heavy model on the test set. Upon comparison, we note the distinct effectiveness of our network in predicting c_{NFW}, c_{smooth} and f_{sub}, as they exhibit similar distributions relative to the true values. The n_{sub} distribution shows a little dispersion but the network seems to relatively struggle in predicting this parameter with respect to the others. Moreover, our network underestimates on average its true value by $\sim 15\%$. Concerning the distribution of predicted M_{vir} values, subjected to the previously described projection effects, our observations indicate a tendency of our network to underestimate the mass of the target three-dimensional masses by approximately 5%. This result confirms the results found by Giocoli et al. (2012b), where the cluster virial mass is estimated with a more classical approach that consists of fitting the convergence profile. By choosing a density profile for the lens, typically an NFW profile, fitting the observed radial convergence profile returns the estimated values of cluster virial mass and concentration. In their work, Giocoli et al. (2012b) found with this approach that the 3D cluster mass is underestimated by approximately 20% due to projection effects, since there is a higher probability to observe clusters elongated on the plane of the sky. However, our results are closer to the real values compared to the best-fit ones, showing a predisposition of our DL algorithm in characterizing the 3D mass of the halo. We attribute this result to the fact that the labels associated to the maps contain a three-dimensional information of either virial mass and concentration. By changing the model MOKA assumes for simulating the halos triaxiality (Jing & Suto, 2002), we could obtain different results.
Tab. 5.2 summarizes the statistical estimators for all our VGG-inspired models.

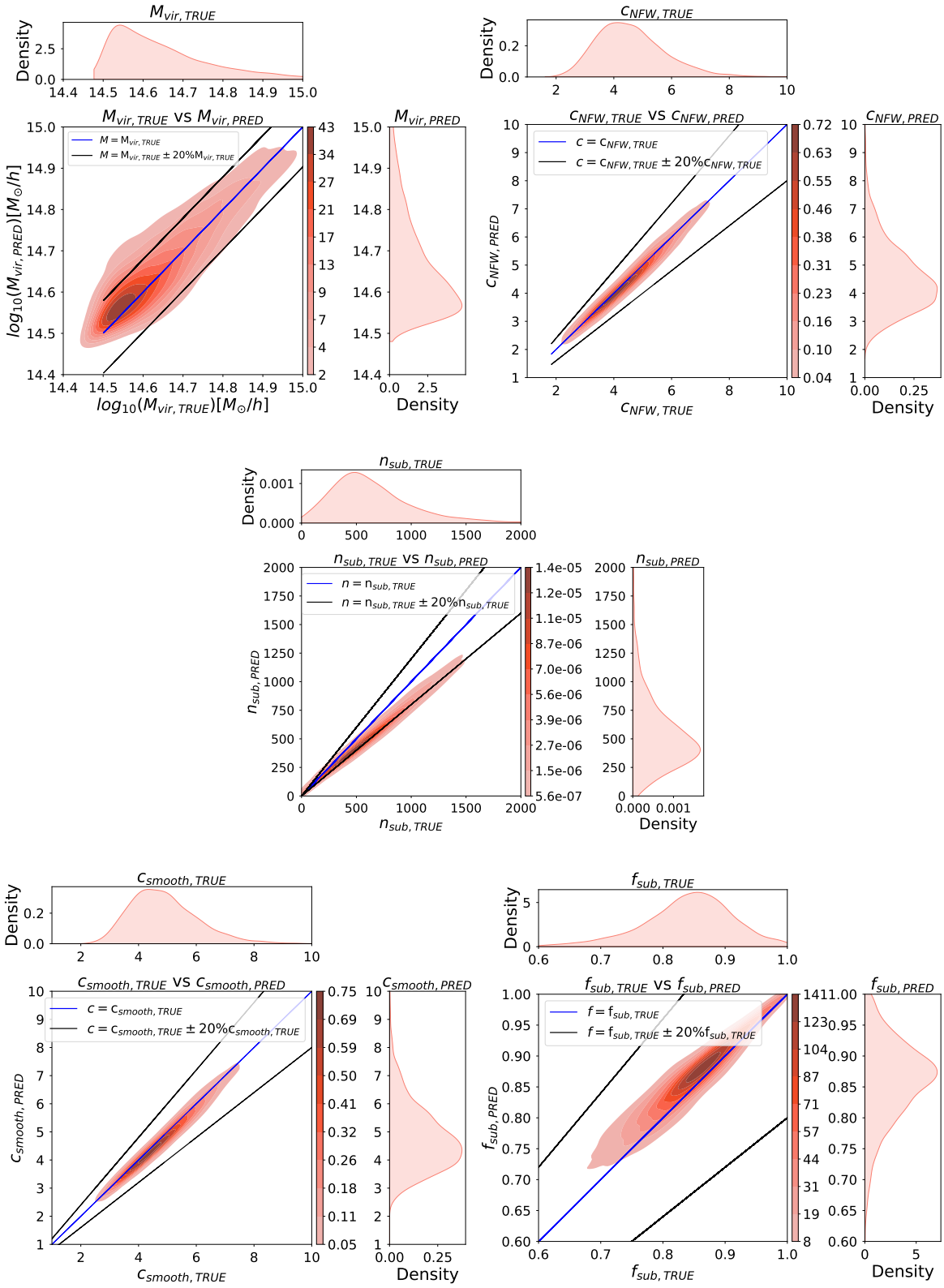


Figure 5.4: KDE plots showing predicted vs target values for M_{vir} (upper-left panel), c_{NFW} (upper-right panel), c_{smooth} (lower-left panel), n_{sub} (middle panel) and f_{sub} (lower-right panel) obtained applying our VGG19-heavy model on noiseless reduced shear maps located at $z = 0.25$.

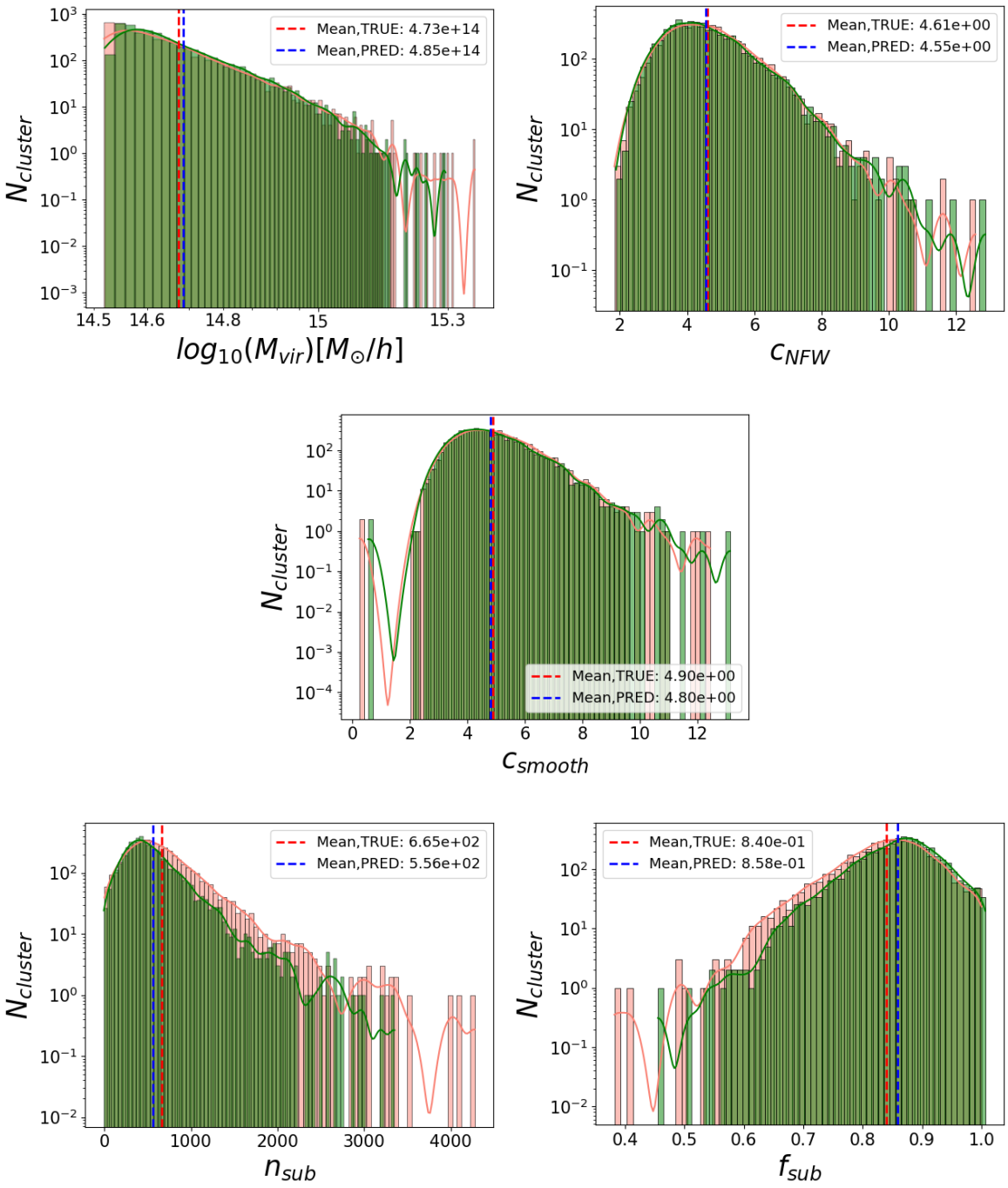


Figure 5.5: Histograms comparing the distribution of the predicted parameters (green) versus the distribution of the true parameters (red), obtained applying our VGG19-heavy model on reduced shear maps. $N_{cluster}$ refers to the number of clusters. All the clusters are located at $z = 0.25$. A vertical dotted line representing the mean value is drawn for every parameter. The solid line represents the KDE probability function.

We also tested the VGG19-heavy model by implementing galaxy shape noise in the reduced shear maps, as described in Sec. 4.1.1. The training process, conducted in two distinct phases of 50 epochs each, utilized a batch size of 40 elements and lasted approximately 90 hours, resulting in a final accuracy of 85.7%. The loss function reached a minimum of 0.22, notably higher than all the VGG models without noise. The loss trend for this test is shown in Fig. 5.6. We also repeated the training starting from the endpoint of the preceding test, but the loss function did not decrease, rising up to ~ 0.4, leading to no improvements in the accuracy.

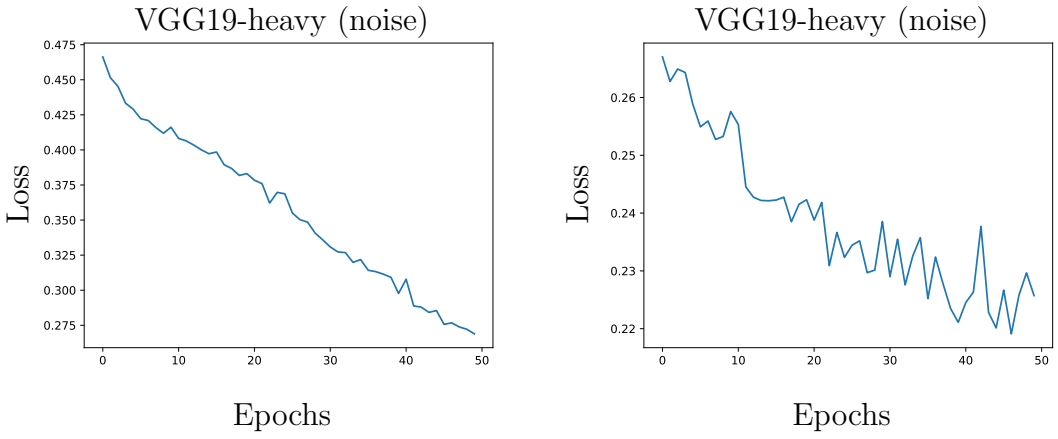


Figure 5.6: The graph illustrates the loss progression during the training of the VGG19-heavy model on noisy reduced shear maps. The left panel shows the training loss over the initial 50 epochs, while the right panel displays the loss trend during the final 50 epochs, starting from the last point of the preceding training phase.

This discrepancy is mainly due to the network's difficulty in accurately predicting the number of substructures within the noisy maps. The predicted value of n_{sub} for approximately 3000 test set maps (out of a total of 5000 maps) is not correctly estimated within a 20% range, exhibiting significant dispersion, as illustrated in Fig. 5.7. Statistical estimators for this test are given in Tab. 5.2 under the "VGGnoise" name. From this test we can conclude that our DL approach could struggle in precisely characterize some of the observed features, that need to be obtained with different methods. Nevertheless, our network is capable of giving a relatively close initial guess for other more accurate but computationally expensive studies, which can be conducted after a first and relatively faster DL technique application. Alternatively, the application of the DL algorithm could follow a more traditional initial approach to supplement the results obtained. However, the other parameters are relatively well predicted.

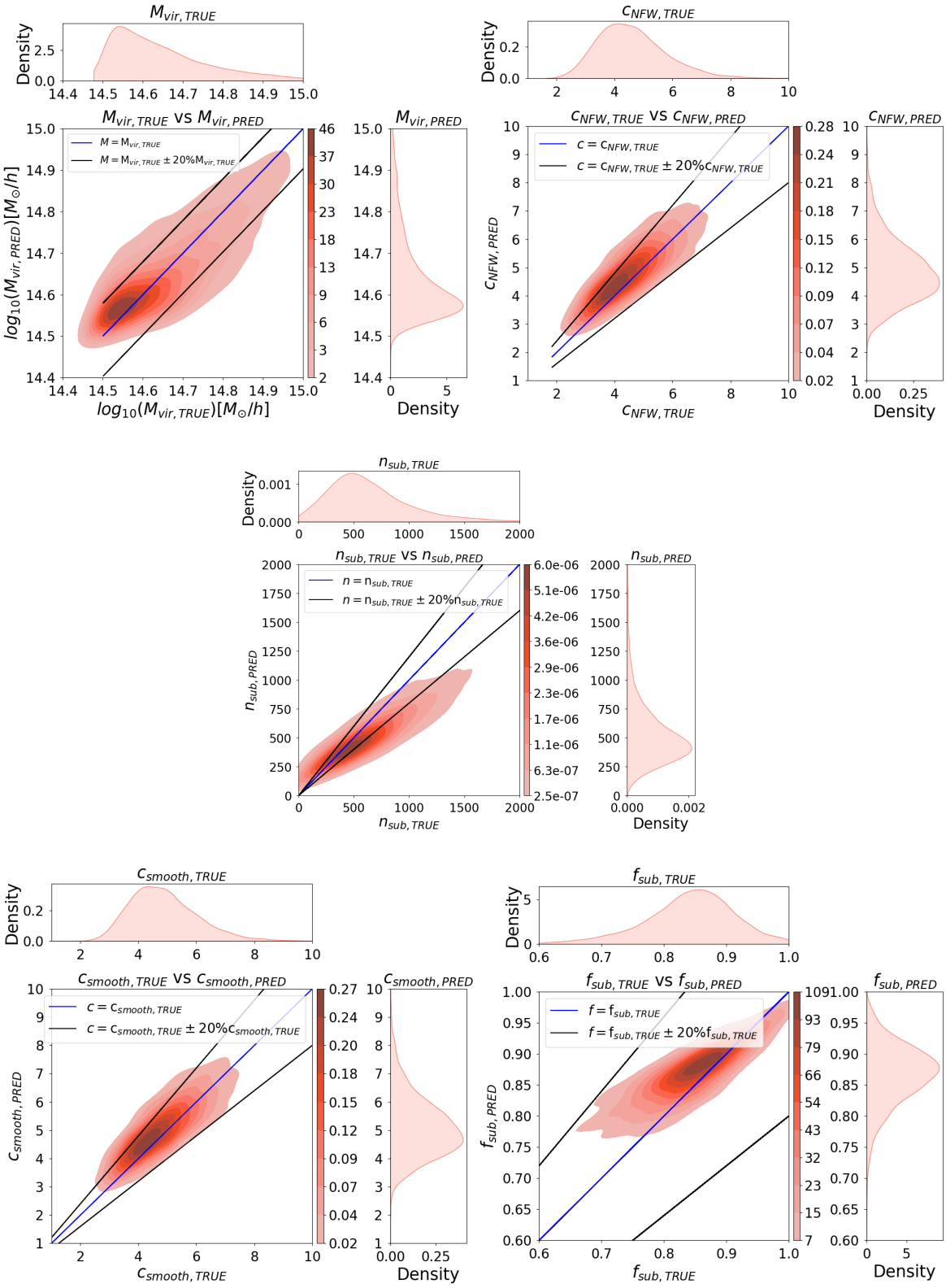


Figure 5.7: Same as Fig. 5.4 but for noisy reduced shear maps located at $z = 0.25$.

Table 5.2: Statistical estimators for M_{vir}, c_{NFW}, c_{smooth}, n_{sub} anf f_{sub} obtained evaluating our VGG-inspired models on reduced shear maps at $z = 0.25$.

M_{vir}					
Network	bias $[10^{27} M^2_{sun}/h^2]$	σ $[10^{13} M_{sun}/h]$	MAD $[10^{13} M_{sun}/h]$	NMAD $[10^{13} M_{sun}/h]$	RMS $[10^{13} M_{sun}/h]$
VGG19-heavy	4.786	6.821	3.223	4.779	6.829
VGG19-light	5.865	7.460	3.388	5.023	7.568
VGG16-basic	6.829	7.990	3.851	5.710	8.164
VGGnoise	8.212	8.823	3.785	5.612	8.846
VGG19_drop-1	5.669	7.138	3.544	5.254	7.394
VGG19_drop-2	5.370	7.218	3.536	5.243	7.239
VGG19_drop-3	6.538	7.700	3.795	5.627	7.933
c_{NFW}					
Network	bias	σ	MAD	NMAD	RMS
VGG19-heavy	0.041	0.193	0.110	0.163	0.200
VGG19-light	0.078	0.258	0.150	0.223	0.276
VGG16-basic	0.117	0.329	0.182	0.270	0.337
VGGnoise	0.381	0.600	0.349	0.518	0.612
VGG19_drop-1	0.062	0.227	0.132	0.196	0.244
VGG19_drop-2	0.091	0.236	0.133	0.197	0.297
VGG19_drop-3	0.144	0.311	0.173	0.256	0.374
c_{smooth}					
Network	bias	σ	MAD	NMAD	RMS
VGG19-heavy	0.035	0.176	0.099	0.147	0.185
VGG19-light	0.072	0.242	0.136	0.202	0.264
VGG16-basic	0.110	0.320	0.166	0.247	0.327
VGGnoise	0.402	0.617	0.361	0.535	0.628
VGG19_drop-1	0.055	0.214	0.125	0.186	0.231
VGG19_drop-2	0.094	0.226	0.124	0.184	0.301
VGG19_drop-3	0.135	0.304	0.171	0.253	0.363
n_{sub}					
Network	bias	σ	MAD	NMAD	RMS
VGG19-heavy	3139.260	53.077	25.439	37.716	54.512
VGG19-light	6524.101	71.217	35.971	53.330	76.798
VGG16-basic	17941.153	98.241	53.641	79.528	129.169
VGGnoise	13794.506	107.743	61.325	90.920	112.615
VGG19_drop-1	5276.539	65.398	35.354	52.416	70.380
VGG19_drop-2	6746.691	67.613	34.097	50.553	79.439
VGG19_drop-3	16086.365	112.176	59.209	87.784	122.685
f_{sub}					
Network	bias	σ	MAD	NMAD	RMS
VGG19-heavy	0.0005	0.0223	0.0115	0.0170	0.0226
VGG19-light	0.0007	0.0255	0.0133	0.0197	0.0259
VGG16-basic	0.0010	0.0260	0.0138	0.0205	0.0307
VGGnoise	0.0013	0.0348	0.0193	0.0286	0.0353
VGG19_drop-1	0.0006	0.0237	0.0125	0.0185	0.0249
VGG19_drop-2	0.0007	0.0244	0.0131	0.0194	0.0254
VGG19_drop-3	0.0009	0.0271	0.0153	0.0227	0.0293

5.3 Inception models results

Concerning our Inception models, they demonstrated inferior performances when compared with our VGG networks. This section presents their results alongside a statistical comparison of their performances.

Inception-v4 Starting from the Inception-v4 architecture, we first tested the Inception-v4 light model. Its training involved a batch size of 75 elements over 60 epochs, lasting approximately 83 hours and reaching an accuracy on the test set of 92.7%. The top panel of Fig. 5.8 illustrates the loss trend observed for this model, which bottoms out at a minimum value of 0.089. The trend highlights this model's fast convergence to the minimum, followed by an increase of the loss values as the number of epochs advances. At the 35^{th} epoch, the learning rate was adjusted to 0.5×10^{-5}, resulting in a local decrease of the loss value. However, this adjustment did not yield any additional improvement. Subsequently, the early stopping condition terminated the training process at the 60^{th} epoch.
We then tested the Inception-v4 heavy model using the same batch size of 75 elements over 62 epochs. The training lasted about 87 hours and reached an accuracy of 92.5%, close to the previous light model. The loss trend for this model is shown in the lower panel of Fig. 5.8. Interestingly, with the increase in network complexity, we notice that the convergence to the minimum takes longer without showing the same tendency to increase with the epochs. The minimum reached by the loss value nearly matches the lighter model's at a value of 0.088.
The last Inception model we tested is the Inception-drop model. By implementing two dropout layers in an intermediate model between our heavy and light networks, we observed again a lower accuracy on the test set. In particular, Inception-drop training lasted about 77 hours over 80 epochs (with no early stopping conditions), and reached a final loss value of 0.118, corresponding to an accuracy of only 89.3%. The trend loss for this model is shown in the middle panel of Fig. 5.8. Despite displaying increased oscillations and a slower decrease time than the dropout-less models, the Inception-drop training process exhibited a more consistent convergence towards the minimum. Fig. 5.9 shows the KDE plots of the distributions of the parameters obtained with Inception-v4 heavy on the noiseless test dataset. Notably, the Inception network demonstrates a stronger inclination towards predicting n_{sub} compared to the VGG19-heavy model, particularly evident in the distribution's outer edges. Tab. 5.3 shows the statistical estimators for the Inception-v4-based models.

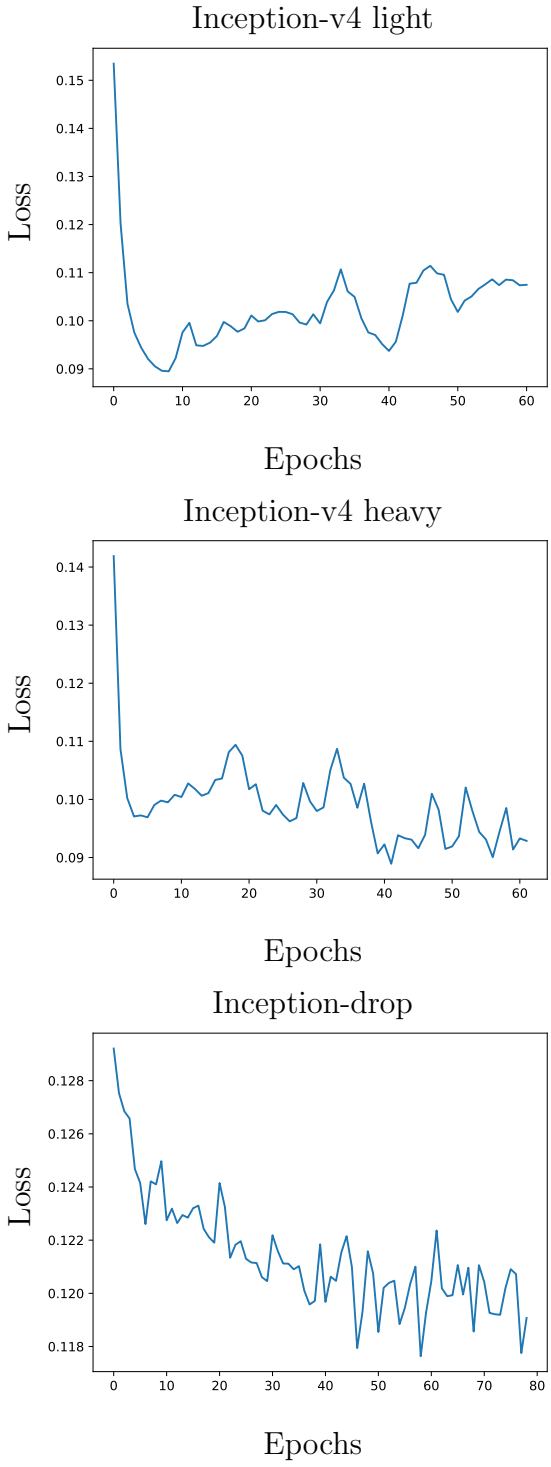


Figure 5.8: Loss trend for the Inception-v4 models. The upper panel shows the Inception-v4 light trend. The middle panel shows the Inception-v4 heavy trend. On the bottom one, the Inception-drop trend is plotted.

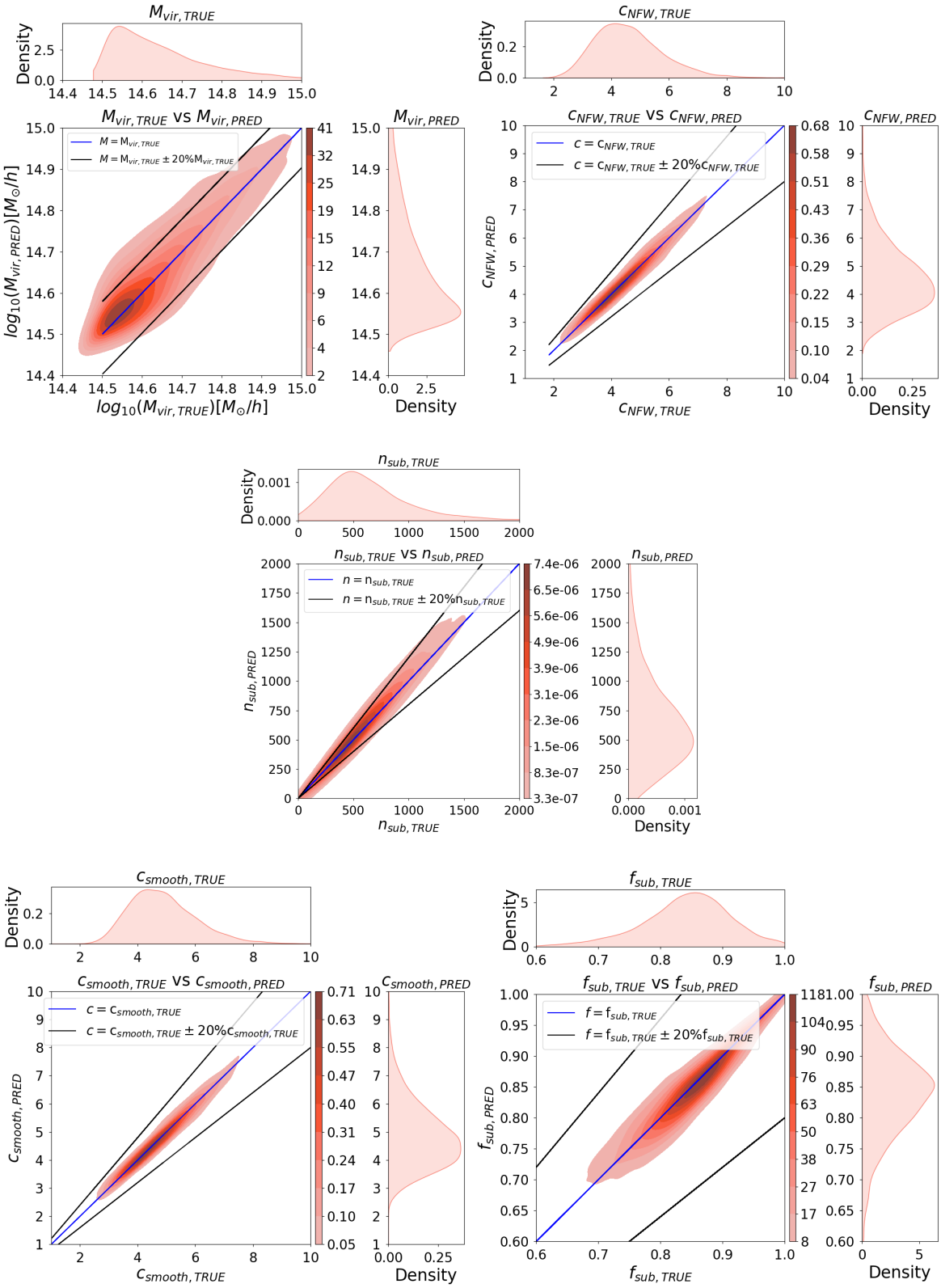


Figure 5.9: Same as Fig. 5.4 but using our Inception-v4 heavy model on noiseless reduced shear maps located at $z = 0.25$.

Inception-ResNet-v2 At last, we present the results of our Inception-ResNet-v2-inspired models. Both models were tested with a batch size of 75 elements. We started by testing the Inception-ResNet-v2 light model. The training phase of this model lasted approximately 47 hours, reaching 61 epochs and a minimum loss value of 0.095. An accuracy of 93.3% characterizes this model.
Considering our last model, the training of the Inception-ResNet-v2 heavy network lasted about 51 hours, reaching 60 epochs and a minimum loss value of 0.093, leading to an accuracy of 94.3%.
Both ResNet models exhibit slightly higher accuracy compared to their corresponding Inception-v4 counterparts. Once again, increasing the network size leads to improved performances. The loss trend for our ResNet models is shown in Fig. 5.10. Both trends exhibit a rapid decline followed by the evidence of an inability to achieve a lower loss value. As observed by the authors in Szegedy et al. (2016) and reported in Fig. 3.16, we notice a similar improvement in our ResNet performances compared to the Inception-v4 architectures since they train faster and reach higher accuracy. Moreover, unlike the Inception-v4 and VGG-Net models in this work, the ResNet architecture reaches the minimum loss value very rapidly, within 7-8 epochs, in approximately 10 hours of computational time. This characteristic makes the ResNet architecture, particularly the Inception-ResNet-v2 heavy model, the optimal compromise between accuracy and computation cost within the scope of our project, alongside VGG19_drop-1. The KDE plots for the Inception-ResNet-v2 heavy model are shown in Fig. 5.11. In addition, Tab. 5.3 shows the statistical estimators for our ResNet models: although the VGG performance on the characterization of n_{sub} is statistically superior, ResNet-v2 models display better results compared to both the Inception-v4 and the VGG-Net ones. Specifically, considering the accuracy calculation on the test set, Inception-v4 heavy and ResNet-v2 heavy show about the same probability of predicting M_{vir}, c_{NFW}, c_{smooth} and f_{sub} within the 20%. When it comes to n_{sub}, the Inception-v4 heavy model shows twice as bad performance compared to ResNet-v2 heavy, giving a wrong prediction on ~ 1000 test maps, while ResNet only mistakes ~ 500 n_{sub} values. This makes the ResNet architecture our best network for predicting this parameter. Fig. 5.12 shows the histograms comparing the predicted parameters distribution versus the true parameters distribution for our Inception-v4 and Inception-ResNet-v2 models.

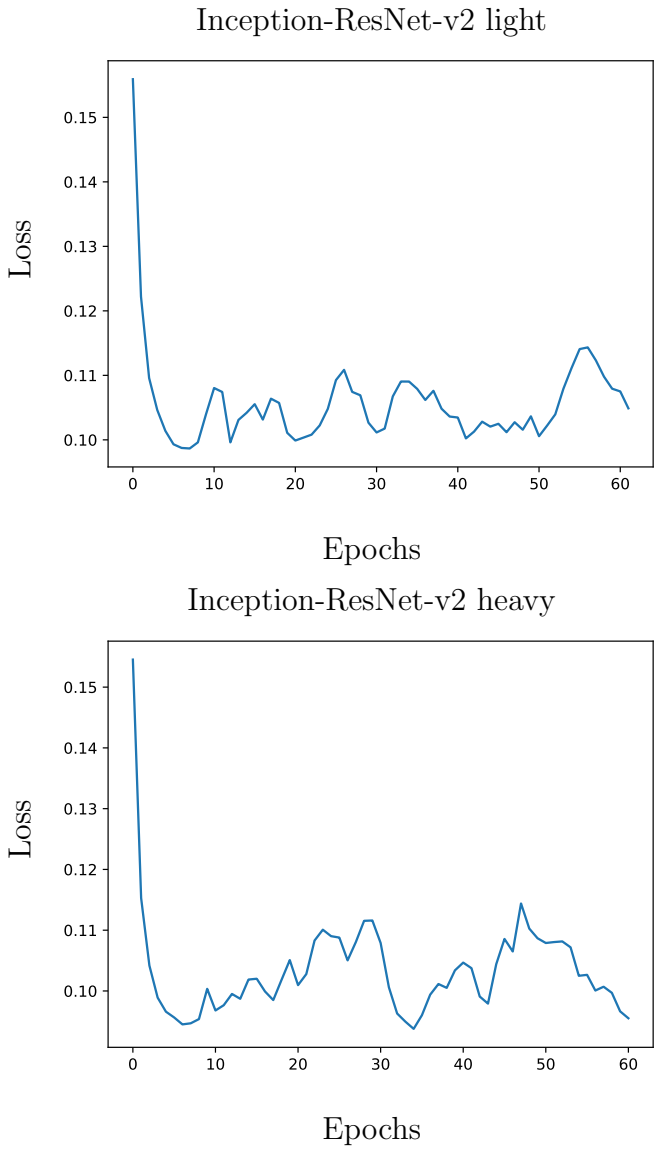


Figure 5.10: Loss trend for the Inception-ResNet-v2 models. The upper panel shows the Inception-ResNet-v2 light trend. On the bottom one the Inception-ResNet-v2 heavy trend is plotted.

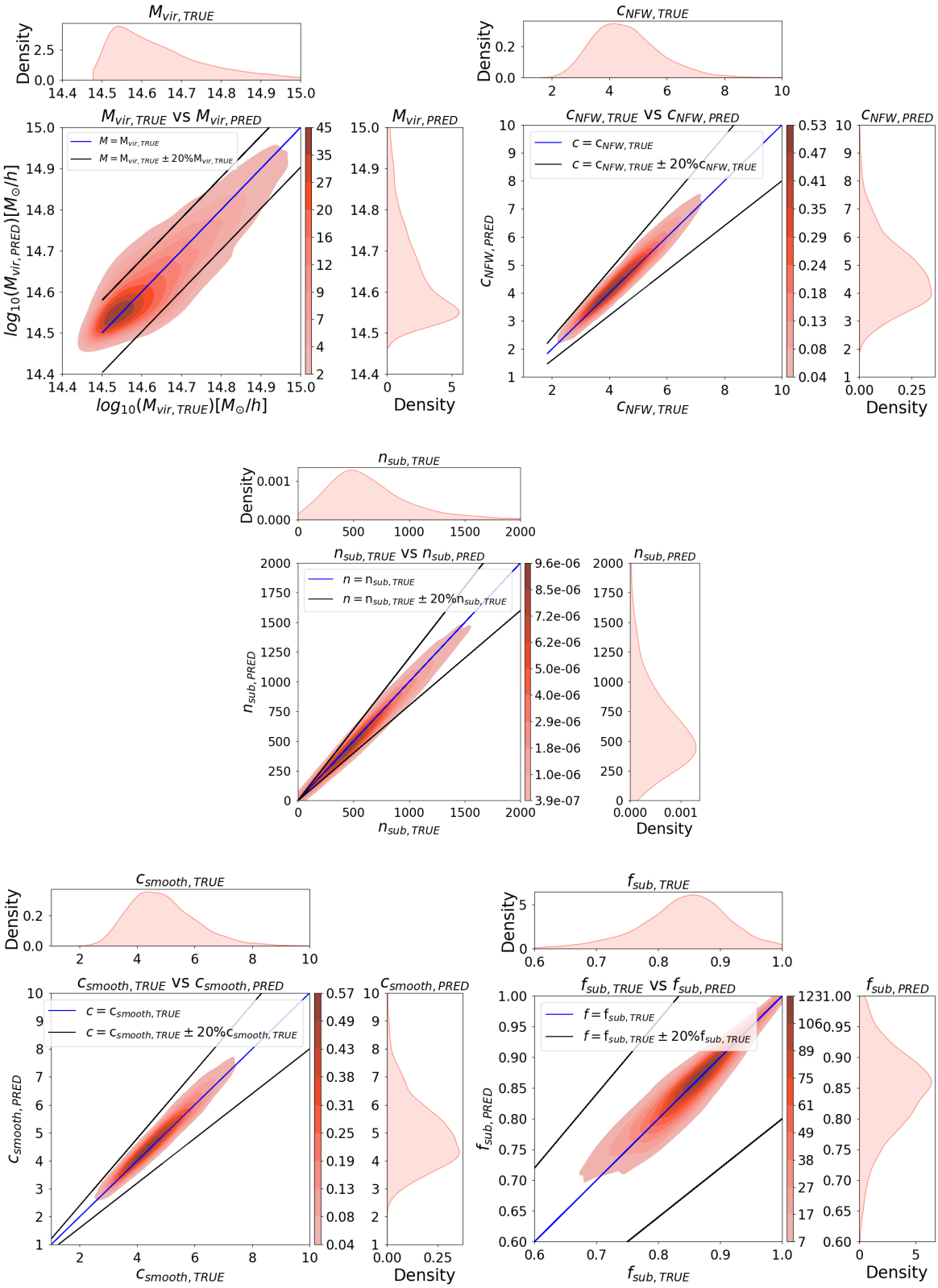


Figure 5.11: Same as Fig. 5.9 but using our Inception-ResNet-v2 heavy model on noiseless reduced shear maps located at $z = 0.25$.

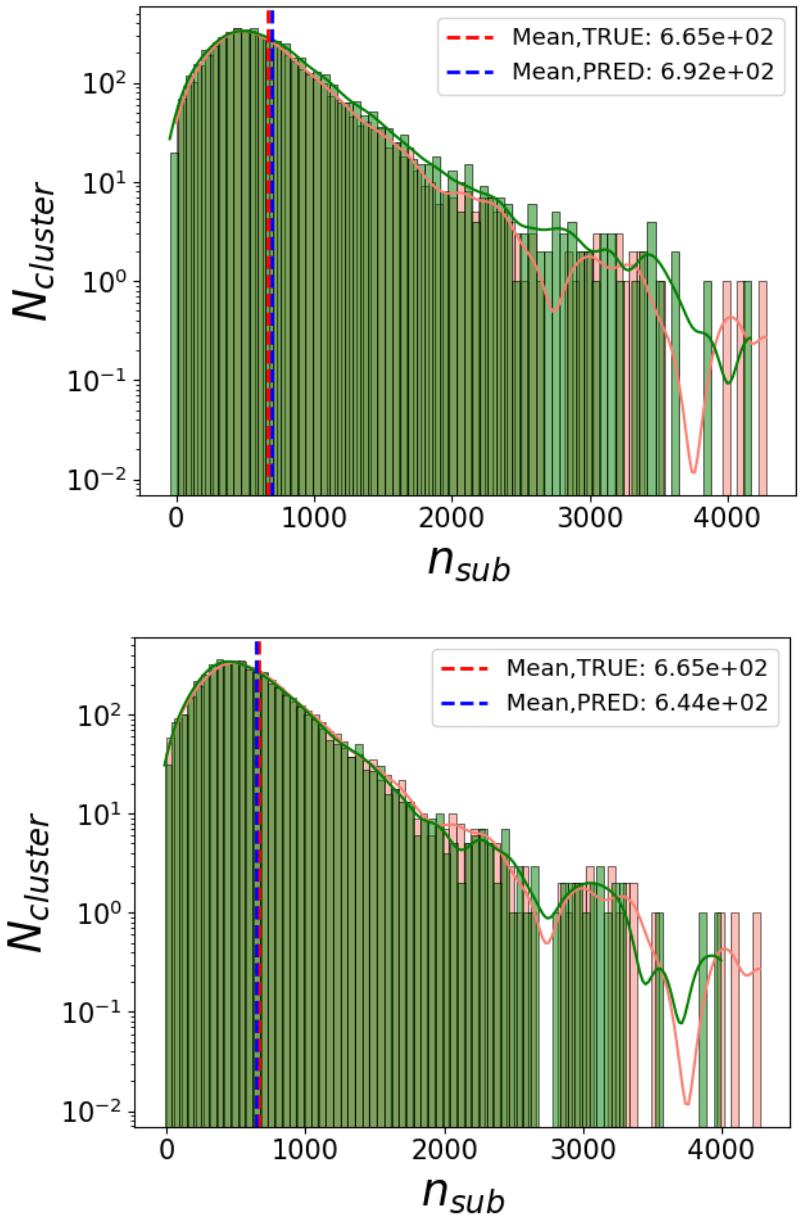


Figure 5.12: As Fig. 5.5, showing the n_{sub} predicted vs true distribution obtained applying our Inception-v4 heavy model (up) and our Inception-ResNet-v2 model (down) on noiseless reduced shear maps located at $z = 0.25$.

Table 5.3: Statistical estimators for M_{vir}, c_{NFW}, c_{smooth}, n_{sub} and f_{sub} obtained evaluating our Inception-inspired models on reduced shear maps at $z = 0.25$. The H and the L indicate the "heavy" and the "light" model respectively.

M_{vir}					
Network	bias $[10^{27} M^2_{sun}/h^2]$	σ $[10^{13} M_{sun}/h]$	MAD $[10^{13} M_{sun}/h]$	NMAD $[10^{13} M_{sun}/h]$	RMS $[10^{13} M_{sun}/h]$
Inception-v4 H	5.848	7.371	3.515	5.211	7.539
Inception-v4 L	7.004	8.052	3.694	5.477	8.270
Inception-drop	6.939	8.063	4.033	5.980	8.182
ResNet-V2 H	6.162	7.707	3.743	5.550	7.790
ResNet-V2 L	6.746	7.946	3.622	5.370	8.062
c_{NFW}					
Network	bias	σ	MAD	NMAD	RMS
Inception-v4 H	0.080	0.248	0.143	0.213	0.277
Inception-v4 L	0.100	0.286	0.166	0.246	0.312
Inception-drop	0.137	0.343	0.192	0.285	0.366
ResNet-V2 H	0.089	0.266	0.154	0.228	0.289
ResNet-V2 L	0.149	0.314	0.172	0.255	0.381
c_{smooth}					
Network	bias	σ	MAD	NMAD	RMS
Inception-v4 H	0.073	0.233	0.134	0.199	0.264
Inception-v4 L	0.095	0.277	0.164	0.243	0.303
Inception-drop	0.135	0.339	0.187	0.278	0.363
ResNet-V2 H	0.086	0.257	0.148	0.220	0.283
ResNet-V2 L	0.141	0.304	0.168	0.250	0.370
n_{sub}					
Network	bias	σ	MAD	NMAD	RMS
Inception-v4 H	11829.452	93.004	47.314	70.148	104.593
Inception-v4 L	8422.521	76.760	38.503	57.084	88.726
Inception-drop	17673.274	126.275	64.265	95.279	130.588
ResNet-V2 H	8329.362	81.828	39.180	58.088	88.577
ResNet-V2 L	13396.827	94.583	48.534	71.957	110.314
f_{sub}					
Network	bias	σ	MAD	NMAD	RMS
Inception-v4 H	0.0007	0.0246	0.0136	0.0202	0.0260
Inception-v4 L	0.0008	0.0262	0.0135	0.0200	0.0274
Inception-drop	0.0009	0.0291	0.0165	0.0244	0.0300
ResNet-V2 H	0.0007	0.0263	0.0137	0.0203	0.0270
ResNet-V2 L	0.0008	0.0263	0.0137	0.0202	0.0279

Chapter 6

Conclusions

In this Thesis, we have evaluated the ability of different Convolutional Neural Networks to predict galaxy clusters' structural parameters, such as the mass, concentration (for both the smooth and total mass distribution), subhalo number and mass fraction. We accomplished this by implementing three different architectures, the VGG-Net from Simonyan & Zisserman (2015), the Inception-v4, and the Inception-ResNet-v2 from Szegedy et al. (2016). In particular, we created distinct models employing different numbers of layers and Deep Learning techniques, such as dropout layers. Starting from these architectures, we trained each model on simulated weak lensing maps of galaxy clusters produced through the MOKA software (Giocoli et al., 2012a). A summary of the results we obtained with our tests is presented in Tab. 6.1.
We first trained VGG-Net-based models. We tested this architecture using only a few million parameters. Then we increased the network size and created "light" and "heavy" variants of the same architecture. By observing the results obtained with our best model dubbed VGG19-heavy, which reached a 96.7% accuracy, we found that the virial mass estimates are only slightly affected by projection effects, showing a tendency of the network to underestimate the virial mass by approximately 5%. This bias is substantially lower than previously found in the literature, based on more traditional analysis methods, such as parametric fitting of the weak lensing signal (Giocoli et al., 2012b). We also tested the VGG19-heavy model using up to three dropout layers, which revealed a decrease in the network accuracy. Finally, we aimed at reproducing more realistic measurements by adding galaxy shape noise to reduced shear maps, resulting in a deterioration of the performances, which in turn translates to a loss of accuracy. However, our measurements on noisy maps remain generally good, since the deterioration in performance significantly affects only some of the parameters, particularly the number of substructures, while for the other values the estimate is relatively accurate.
Subsequently, we trained models based on the Inception architecture, characterized by a higher structural complexity than VGG-Net. For both the Inception-v4 and

Inception-ResNet-v2 architectures, we created a "light" and a "heavy" model differing in size (i.e., the number of Inception or ResNet modules employed), and compared them with the results obtained using the VGG-Net architecture. In general, we found these models to reach a lower accuracy compared to VGG-Net but with great improvements in the computational costs. In particular, our best Inception-based model, the ResNet-v2 heavy, reached 94.3% accuracy in less than 10 hours, while VGG19-heavy takes approximately 47 hours to reach its higher accuracy. As for the VGG-Net models, we also tested dropouts for the Inception-v4 architecture, finding no improvements in the performance.
In conclusion, in this work, we have shown an alternative method to the more classical approaches for measuring cluster structural parameters, which allows the analysis of large datasets in a relatively short time. We believe this method could constitute a viable and valuable technique to study or to obtain initial guesses for the cluster parameters, especially given the huge amount of data that upcoming surveys will provide. Our dataset consisted of simulated halo maps obtained by assuming a relatively simple structure for the halos, as the main halo is modeled as a triaxial halo, and the substructures added to the smooth matter distribution are spherical halos. In the future, we plan on conducting the same kind of study on more complex and realistic cluster mass maps, using hydrodynamical simulations, which also consider the effects of the baryon component..

Table 6.1: Overall performances of our best models for different architectures. The columns indicate in the following order: the name of the model, the number of parameters of the model (in millions), the batch size of the training, the total number of epochs reached during the training, the final Learning Rate, the minimum value reached by the Loss Function, and the accuracy of the model in predicting the test set parameters within the 20% of the real value.

Network	nPar	Batch Size	epochs	LR	Loss	Accuracy 20%
VGG19-heavy	69M	30	50	10^{-5}	0.067	96.7%
VGG19-light	30M	50	73	$0.5 \cdot 10^{-5}$	0.078	94.8%
ResNet-v2 H	36M	75	60	$0.5 \cdot 10^{-5}$	0.093	94.3%
ResNet-v2 L	17M	75	61	$0.5 \cdot 10^{-5}$	0.096	93.3%
Inception-v4 L	15M	75	60	$0.5 \cdot 10^{-5}$	0.088	92.7%
Inception-v4 H	36M	75	62	$0.5 \cdot 10^{-5}$	0.089	92.5%
VGG16-basic	2.5M	75	44	$0.5 \cdot 10^{-5}$	0.080	92.2%
VGGNoise	69M	40	100	10^{-5}	0.22	85.7%

Bibliography

Bartelmann M., Schneider P., 2001, Physics Reports, 340, 291

Bengio Y., et al., 2009, Foundations and trends® in Machine Learning, 2, 1

Bergamini, P. et al., 2019, http://dx.doi.org/10.1051/0004-6361/201935974 A&A, 631, A130

Birrer S., Amara A., Refregier A., 2015, http://dx.doi.org/10.1088/0004-637x/813/2/102 The Astrophysical Journal, 813, 102

Bishop C., 2006, Springer google schola, 2, 5

Blandford R. D., Narayan R., 1992, http://dx.doi.org/10.1146/annurev.astro.30.1.311 , https://ui.adsabs.harvard.edu/abs/1992ARAA..30..311B 30, 311

Blandford R., Surpi G., Kundic T., 2000, Modeling Galaxy Lenses (http://arxiv.org/abs/astro-ph/0001496 arXiv:astro-ph/0001496)

Bonamigo M., et al., 2018, http://dx.doi.org/10.3847/1538-4357/aad4a7 The Astrophysical Journal, 864, 98

Broadhurst T. J., Taylor A. N., Peacock J. A., 1995, http://dx.doi.org/10.1086/175053 The Astrophysical Journal, 438, 49

Caminha G. B., et al., 2017, http://dx.doi.org/10.1051/0004-6361/201731498 Astronomy & Astrophysics, 607, A93

Cole S., Lacey C., 1996, http://dx.doi.org/10.1093/mnras/281.2.716 Monthly Notices of the Royal Astronomical Society, 281, 716

Coles P., Lucchin F., 2003, Cosmology: The origin and evolution of cosmic structure. John Wiley & Sons

Einstein A., 1916, http://dx.doi.org/https://doi.org/10.1002/andp.19163540702 Annalen der Physik, 354, 769

Friedman A., 1922, Zeitschrift für Physik, 10, 377

Gao L., White S. D. M., Jenkins A., Stoehr F., Springel V., 2004, http://dx.doi.org/10.1111/j.1365-2966.2004.08360.x , https://ui.adsabs.harvard.edu/abs/2004MNRAS.355..819G 355, 819

Giocoli C., Tormen G., Sheth R. K., van den Bosch F. C., 2010, http://dx.doi.org/10.1111/j.1365-2966.2010.16311.x Monthly Notices of the Royal Astronomical Society

Giocoli C., Meneghetti M., Bartelmann M., Moscardini L., Boldrin M., 2012a, http://dx.doi.org/10.1111/j.1365-2966.2012.20558.x Monthly Notices of the Royal Astronomical Society, 421, 3343–3355

Giocoli C., Meneghetti M., Ettori S., Moscardini L., 2012b, http://dx.doi.org/10.1111/j.1365-2966.2012.21743.x , https://ui.adsabs.harvard.edu/abs/2012MNRAS.426.1558G 426, 1558

Goodfellow I., Bengio Y., Courville A., Bengio Y., 2016, Deep learning, vol. 1, No. 2

He K., Zhang X., Ren S., Sun J., 2015, Deep Residual Learning for Image Recognition (http://arxiv.org/abs/1512.03385 arXiv:1512.03385)

Hernquist L., 1990, http://dx.doi.org/10.1086/168845 , https://ui.adsabs.harvard.edu/abs/1990ApJ...356..359H 356, 359

Hubble E., 1929, http://dx.doi.org/10.1073/pnas.15.3.168 Proceedings of the National Academy of Sciences, 15, 168

Ioffe S., Szegedy C., 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (http://arxiv.org/abs/1502.03167 arXiv:1502.03167)

Jing Y. P., Suto Y., 2002, http://dx.doi.org/10.1086/341065 , https://ui.adsabs.harvard.edu/abs/2002ApJ...574..538J 574, 538

Kaiser N., Squires G., 1993, http://dx.doi.org/10.1086/172297 , https://ui.adsabs.harvard.edu/abs/1993ApJ...404..441K 404, 441

Keeton C. R., 2003, http://dx.doi.org/10.1086/345717 , https://ui.adsabs.harvard.edu/abs/2003ApJ...584..664K 584, 664

Kingma D. P., Ba J., 2017, Adam: A Method for Stochastic Optimization (http://arxiv.org/abs/1412.6980 arXiv:1412.6980)

Kneib J.-P., Natarajan P., 2011, http://dx.doi.org/10.1007/s00159-011-0047-3 The Astronomy and Astrophysics Review, 19

Krizhevsky A., Sutskever I., Hinton G. E., 2012, in Pereira F., Burges C., Bottou L., Weinberger K., eds, Vol. 25, Advances in Neural Information Processing Systems. Curran Associates, Inc., https://proceedings.neurips.cc/paper$_f iles/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b–Paper.pdf$

Laureijs R., et al., 2011, Euclid Definition Study Report (http://arxiv.org/abs/1110.3193 `arXiv:1110.3193`)

LeCun Y., Bengio Y., et al., 1995, The handbook of brain theory and neural networks, 3361, 1995

Mandelbaum R., van de Ven G., Keeton C. R., 2009, http://dx.doi.org/10.1111/j.1365-2966.2009.15166.x , https://ui.adsabs.harvard.edu/abs/2009MNRAS.398..635M 398, 635

Meneghetti M., 2021, Introduction to gravitational lensing: with Python examples. Vol. 956, Springer Nature

Meneghetti M., Bartelmann M., Dahle H., Limousin M., 2013, http://dx.doi.org/10.1007/s11214-013-9981-x Space Science Reviews, 177, 31–74

Mitchell T. M., 1997, The McGrawHill Companies, inc

Navarro J. F., Frenk C. S., White S. D. M., 1996, http://dx.doi.org/10.1086/177173 The Astrophysical Journal, 462, 563

Newton E. R., Marshall P. J., Treu T., Auger M. W., Gavazzi R., Bolton A. S., Koopmans L. V. E., Moustakas L. A., 2011, http://dx.doi.org/10.1088/0004-637x/734/2/104 The Astrophysical Journal, 734, 104

Nielsen M. A., 2015, Neural networks and deep learning. Vol. 25, Determination press San Francisco, CA, USA

Paszke A., et al., 2019, in , Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedamonti D., 2018, CoRR, abs/1804.02763

Planck Collaboration et al., 2016, http://dx.doi.org/10.1051/0004-6361/201526681 , https://ui.adsabs.harvard.edu/abs/2016AA...594A..16P 594, A16

Planck Collaboration et al., 2020a, http://dx.doi.org/10.1051/0004-6361/201833880 , https://ui.adsabs.harvard.edu/abs/2020AA...641A...1P 641, A1

Planck Collaboration et al., 2020b, http://dx.doi.org/10.1051/0004-6361/201833910 , https://ui.adsabs.harvard.edu/abs/2020AA...641A...6P 641, A6

Reddi S. J., Kale S., Kumar S., 2019, On the Convergence of Adam and Beyond (http://arxiv.org/abs/1904.09237 arXiv:1904.09237)

Riess A. G., et al., 1998, http://dx.doi.org/10.1086/300499 The Astronomical Journal, 116, 1009–1038

Riess A. G., et al., 2018, http://dx.doi.org/10.3847/1538-4357/aac82e The Astrophysical Journal, 861, 126

Rojas R., 2009

Schneider P., Ehlers J., Falco E. E., 1992, Gravitational Lenses. Springer New York, http://dx.doi.org/10.1007/978-1-4612-2756-4 doi:10.1007/978-1-4612-2756-4, http://dx.doi.org/10.1007/978-1-4612-2756-4

Schramm T., Kayser R., 1994, The complex theory of gravitational lensing: Beltrami equation and cluster lensing (http://arxiv.org/abs/astro-ph/9408064 arXiv:astro-ph/9408064)

Seitz C., Schneider P., 1996, Steps towards nonlinear cluster inversion through gravitational distortions: III. Including a redshift distribution of the sources (http://arxiv.org/abs/astro-ph/9601079 arXiv:astro-ph/9601079)

Sermanet P., Eigen D., Zhang X., Mathieu M., Fergus R., LeCun Y., 2014, OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks (http://arxiv.org/abs/1312.6229 arXiv:1312.6229)

Simonyan K., Zisserman A., 2015, Very Deep Convolutional Networks for Large-Scale Image Recognition (http://arxiv.org/abs/1409.1556 arXiv:1409.1556)

Spinelli C., 2021, PhD thesis, http://amslaurea.unibo.it/22406/

Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., 2014, Journal of Machine Learning Research, 15, 1929

Suyu S. H., Marshall P. J., Hobson M. P., Blandford R. D., 2006, http://dx.doi.org/10.1111/j.1365-2966.2006.10733.x Monthly Notices of the Royal Astronomical Society, 371, 983

Szegedy C., et al., 2014, Going Deeper with Convolutions (http://arxiv.org/abs/1409.4842 arXiv:1409.4842)

Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z., 2015, Rethinking the Inception Architecture for Computer Vision (http://arxiv.org/abs/1512.00567 arXiv:1512.00567)

Szegedy C., Ioffe S., Vanhoucke V., Alemi A., 2016, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning (http://arxiv.org/abs/1602.07261 arXiv:1602.07261)

Treu T., Marshall P. J., 2016, http://dx.doi.org/10.1007/s00159-016-0096-8 The Astronomy and Astrophysics Review, 24

Umetsu K., 2020, http://dx.doi.org/10.1007/s00159-020-00129-w The Astronomy and Astrophysics Review, 28

Wang L., Li C., Kauffmann G., De Lucia G., 2006, http://dx.doi.org/10.1111/j.1365-2966.2006.10669.x Monthly Notices of the Royal Astronomical Society, 371, 537–547

Zeiler M. D., Fergus R., 2013, Visualizing and Understanding Convolutional Networks (http://arxiv.org/abs/1311.2901 arXiv:1311.2901)

Zhao D. H., Jing Y. P., Mo H. J., Börner G., 2009, http://dx.doi.org/10.1088/0004-637X/707/1/354 , https://ui.adsabs.harvard.edu/abs/2009ApJ...707..354Z 707, 354

van de Ven G., Mandelbaum R., Keeton C. R., 2009, http://dx.doi.org/10.1111/j.1365-2966.2009.15167.x , https://ui.adsabs.harvard.edu/abs/2009MNRAS.398..607V 398, 607