universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Predicting Galactic Orbits with Deep Learning“

verfasst von / submitted by

Léah Skusa BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 861
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Astronomie
Betreut von / Supervisor:	Prashin Jethwa, MSc MA PhD

Acknowledgements and Disclaimers

Acknowledgements I want to thank my supervisor, Prashin, for the patience and constant support I received during my thesis, as well as for many useful comments on my thesis chapters. I also thank him and Sabine Thater for helping me understand DYNAMITE. I thank Philipp Petersen for trying to explain Deep Learning to me, for solving the 'mask'-problem and for providing the structure for the DCNN. Finally, I want to thank my partner Tamo for providing me with breakfast and coffee many times before I worked on my thesis.

Disclaimers In this thesis, I will use the pronoun 'we', but all work was done by me, supported by my supervisors. Sections where I have used work from others are marked with a reference and/or citation. All figures and plots that have no reference/citation are made by me, Léah Skusa.

Abstract

Stellar orbits in galaxies can tell the story of the host galaxy's formation and merger history. Orbit models can unveil details in galaxy kinematics that are not directly visible in observations. Methods to model the distribution of orbits are computationally expensive. Large data sets and many iterations are necessary to generate an orbit library, which is computationally expensive. In this work, we try to lower this computational cost with the use of Deep Learning. We generate orbit libraries with the DYNAMITE code, an implementation of the Schwarzschild Orbit Superposition method. We train a Deconvolutional Neural Net on data of Gauß-Hermite coefficients we extract from the orbit libraries. With the trained neural net, we predict new coefficients and compare them to the coefficients from the original orbit library. We also test the limits of the neural net by training on smaller data sets. When we compare the predicted coefficients to the ones from Schwarzschild orbit modeling, we find that they are very similar and the predictions can be used instead of the expensive orbit models. We also discuss the limitations of our method concerning Deep Learning predictions and the simplifications made in the orbit modeling process.

Kurzfassung

Dies ist eine deutsche Kurzfassung dieser in Englisch verfassten Masterarbeit. Umlaufbahnen von Sternen in Galaxien verraten viel über die Entstehung und Vergangenheit einer Galaxie. Orbitmodelle geben einen Einblick in die kinematische Struktur einer Galaxie, die aus Beobachtungen nicht immer direkt ersichtlich ist. Gängige Methoden, um die Orbitverteilung zu modellieren, haben einen hohen Rechenaufwand. Es werden große Datensätze und vielfache Wiederholungen benötigt, um eine akkurate Orbitbibliothek für eine Galaxie zu erstellen. In dieser Arbeit versuchen wir, den Rechenaufwand zu verringern, indem wir Methoden des maschinellen Lernens nutzen (Deep Learning). Wir nutzen den DYNAMITE-Code, eine Implementierung der Schwarzschild-Methode zur Modellierung von Orbits. Aus den berechneten Orbitbibliotheken extrahieren wir Gauß-Hermite Koeffizienten und erstellen damit einen Datensatz um ein künstliches neuronales Netz zu. trainieren. Mit dem trainierten neuronalen Netzwerk generieren wir einen neuen Satz an Koeffizienten und vergleichen sie mit den Daten aus den Orbitmodellen. Wir untersuchen außerdem die Grenzen des Netzes, indem wir den Trainingsdatensatz verkleinern und die damit erstellten neuen Ergebnisse analysieren. Beim Vergleich der mit Deep Learning generierten und der mit DYNAMITE modellierten Koeffizienten weisen beide eine große Ähnlichkeit auf. Aus den generierten Koeffizienten berechnen wir eine Orbitbibliothek, die genauso verwendet werden kann wir die aus den modellierten Orbits. Wir besprechen auch die Grenzen von maschinellem Lernen für unseren Fall, sowie die Vereinfachungen, die für unsere Herangehensweise notwendig waren.

Contents

1. Introduction and Context

1.1. Motivation

Galaxies are large and fascinating objects in our Universe. Their formation is not fully explored yet, they evolve constantly during their lifetime and they are changed at every encounter with other galaxies. At first glance, galaxies are large accumulations of stars moving together through the universe. When looking closer, they contain gas and dark matter and are hosts for black holes. All components are interesting objects of research, but from simply looking at the stars, we can learn much about the galaxy and its history.
Stars move on orbits through a galaxy. These orbits can have many different shapes. The simplest is a circle around the center of the galaxy, but orbits can be much more complex and interesting, looking like a butterfly or a pretzel. If we know how the orbits are distributed in a galaxy, or in other words, how the stars move, we can deduce its structure and history. To get this distribution, we use Schwarzschild orbit modeling. It is a method where we calculate a library of many different orbits and compare it to data from observations, until the models look as close to the observed galaxy as possible. With every new instrument that provides new observations, the quality of data gets better, but with higher resolutions, the data size increases as well. Schwarzschild orbit modeling needs many iterations to compute orbit libraries and usually runs on computer clusters. This can be very expensive, so we are looking for a method to save computation time. If we could generate a large orbit library by making predictions based on a smaller library, we would need less data and save computation time.
Machine learning gets used more and more in physics and astrophysics. The idea behind it is to build an artificial neural net, 'feed' it with some data and train the net to characterize the data by assigning it labels, classifying it or by learning its properties. Neural networks can make predictions based on the things they learnt. Training a net is computationally expensive, but once it is trained, making predictions is very fast. If we applied machine learning to orbit modeling, could we feed a net with orbit libraries and predict a large sets of orbits, saving precious time that we would need to compute Schwarzschild models?
In this work, we attempt to implement this idea. We want to answer three questions: Can we predict galactic orbits with deep learning? Is it possible to save computing time and resources by doing this? And can we use less data while still getting good results?

1.2. A brief introduction to galaxy kinematics

The movements of stars in galaxies contain precious information about a galaxy's structure and history. They can be described in different ways, and there are multiple observational methods to get there. We will use spectroscopic observations to look at the velocities in a galaxy, and generate a set of stellar orbits to model them.
The motion of stars in a galaxy is described by the kinematics. When observing, we cannot measure them directly, as we only observe the two-dimensional projection of a three-dimensional

object on the sky. The velocity of the stars affects the light that reaches our telescopes: stars moving towards us shift the light into blue colors (to shorter wavelengths), stars moving away from us shift it into redder colors (to longer wavelengths). If we measure this shift in the light, we can extract the velocity distribution in the galaxy.
To observe velocity distributions, we need galaxy spectra. A spectrum shows light from a source split into the different wavelengths. For a galaxy, the spectrum is the sum of the individual stellar spectra. Stars moving through the galaxy change the spectra by shifting and broadening the lines that we observe (Binney and Merrifield, 1999). Integral Field Spectroscopy (IFS) is an observational method that allows us to get multiple spectra for a galaxy. The galaxy image is split into small segments and a spectrum is obtained for each of them, which results in a three-dimensional data cube. This cube with axes (x, y, λ) can be understood as an image of the galaxy in every observed wavelength λ. With integral field spectroscopy, we gain knowledge on the three-dimensional movements in the galaxy. This information can be represented with two-dimensional kinematic maps, where a velocity distribution is associated with each spatial bin of the image.
This distribution is called Line-Of-Sight Velocity Distribution (LOSVD). When we look at the projected galaxy image, the light that reaches us in the line-of-sight carries information about the distribution of the velocity at this point in the galaxy (Bender, 1990). In the simplest case, the LOSVD is usually modelled with a Gaussian function: the distribution follows a bell-curve, as shown in Fig.1.1a. The peak of the curve models the mean velocity v in the line-of-sight, and the width of the curve gives the velocity dispersion σ, which is a measure for the chaotic motion in a galaxy. If σ is small, the velocity distribution is narrow, i.e. the stars have a more regular motion. If σ is large, the distribution is broad, meaning that there is a large range of velocities, and therefore more 'chaos'. The Gaussian formulation of the LOSVD as a function of velocity $\tilde{v}$[1] is shown in Eq.(1.1).

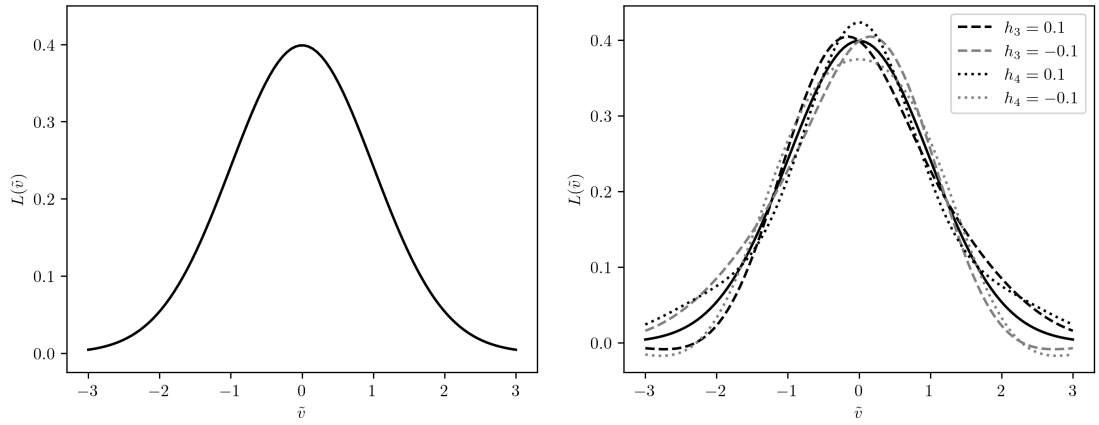
$$\mathcal{L}(\tilde{v}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-0.5\left(\frac{\tilde{v} - v}{\sigma}\right)^2\right) \tag{1.1}$$

However, in realistic cases, the LOSVD is not purely Gaussian and velocity distributions are not really symmetrical. The Gaussian model of the LOSVD can be expanded with a series of Hermite polynomials. For a detailed description and the complete equations, see van der Marel and Franx (1993) and Gerhard (1993). Eq.(1.2) shows a simplified version of the Gauß-Hermite (GH) series. The LOSVD, as a function of the velocity, is modeled by a Gaussian (first part of the equation), multiplied with a series of Hermite polynomials H_j multiplied with coefficients h_j (second part of the equation). The variable y is a function of the velocity: $y(\tilde{v}) = \frac{\tilde{v}-v}{\sigma}$.

$$\mathcal{L}(\tilde{v}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2}\right) \sum_{j=0}^{N} h_j H_j(y) \tag{1.2}$$

The number of polynomials N can be chosen depending on the resolution of the data. In most cases, $N = 4$ is used (e.g. Santucci et al. (2022), Loubser et al. (2022), Pinna et al. (2019)), but higher orders are also possible for very high resolution data (e.g. $N = 6$ used by den Brok et al. (2021)). From van der Marel and Franx (1993), the first three Hermite coefficients are chosen to be $h_0 = 1, h_1 = h_2 = 0$, so Eq.(1.2) simplifies to Eq.(1.3), using only coefficients up to $N = 4$. With this choice, the mean and the width of the LOSVD are very close to the values of v and σ.

[1] We use the symbol v for the mean velocity here and in the following chapters. We only use $\tilde{v}$ here to show that the LOSVD is a function of the velocity.



(a) **(b)**

Figure 1.1.: Left: Gaussian LOSVD with $\sigma = 1$ and $v = 0$ (values for a standard Gaussian). Right: LOSVD using the Gauß-Hermite expansion up to the coefficient h_4. The solid curve shows the standard Gaussian, the dashed and dotted lines show the effects on the curve if h_3 or h_4 are non-zero. h_3, also called the 'coefficient of skewness', tilts the curve; the peak is shifted on the x-axis and the tails are asymmetrical. The 'coefficient of kurtosis' h_4 shifts the peak on the y-axis and changes the tails of the distribution while keeping the symmetry. This figure is adapted from van der Marel and Franx (1993).

$$\mathcal{L} \approx (1 + h_3 H_3 + h_4 H_4) \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2}\right) \tag{1.3}$$

Now, we can quantify galaxy kinematics with four different parameters: the radial velocity v, velocity dispersion σ and the Gauß-Hermite coefficients h_3 and h_4. Each parameter can be represented in a 2D map; we will show an example in Chapter 2, Fig.2.2.

The observed kinematics are the sum of the motion of stars with individual orbits. With 3D kinematic data and complex physical models, we can deduce the distribution of orbits and orbit types for a given galaxy. Together, velocity maps and orbit models are key ingredients to learn about a galaxy's structure and history.

1.3. Extracting galactic orbits with Schwarzschild Orbit Modelling

Kinematics tell the story of the structure and the past of a galaxy, which we can investigate with velocity maps and orbit models. Every galaxy has an intrinsic structure. Spiral galaxies have very distinct components: a disk with spiral arms and a bulge in the center. Elliptical galaxies seem to be more uniform: a blob of stars with a bright center and decreasing brightness towards the outer regions. Every elliptical galaxy can be described as an object with three axes. If all three axes have the same length, the galaxy is spherical. If one axis is different, we call it axisymmetric. In the last case, where all three axes have different lengths, the galaxy is triaxial. Triaxial galaxies are often characterized by the axis ratios $p = c/a$ and $q = b/a$, with a being the long axis and b, c

being the short axes of the ellipsoid. We illustrate the different geometries in Fig.1.2.
However, galaxies are often more complex. Most of their structure is not visible and requires

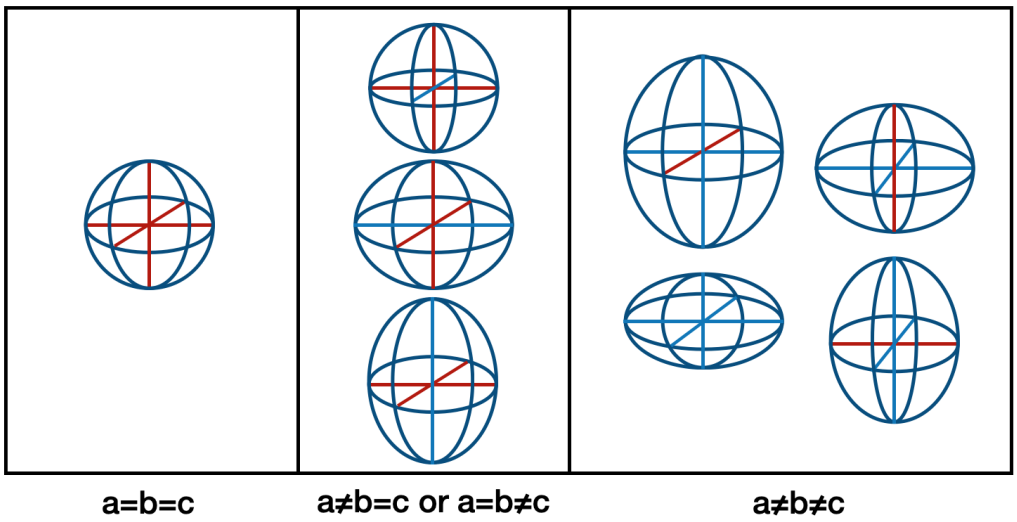


Figure 1.2.: Sketch of different geometries of an ellipsoid. The axes with same length are marked red. Left: spheroidal geometry; all axes have the same length. Middle: axisymmetric geometry; two axes are the same, only one of them is longer or shorter. Right: multiple possibilities for triaxial geometry. The three axes all have different lengths.

specific observational methods to be seen: some elliptical galaxies have stellar streams that are only visible in long exposures, radio observations can show the centers of galaxies and 3D spectroscopy can unveil their kinematics. Additionally, merger events and close encounters lead to disturbances in a galaxy and change its interior structure, i.e. the movements of its stars. To model this inner structure, we can construct a library of stellar orbits: different regions in the galaxy hold different types of stellar orbits, and if we find out how the orbits are distributed, the structure becomes clear. Generally, there are two families of orbits: box orbits and tube orbits (de Zeeuw, 1985). Box orbits have a boxy shape and can cross the center of a galaxy, while tube orbits avoid the center and are tube or ring-shaped. Fig.1.3 illustrates the different orbit types (Statler, 1987). The tubes can be divided into sub-families, depending on their axis of rotation. Box orbits can have very different shapes, which are reminiscent of brezels, pasta or shrimps (Parul et al., 2020). To model a set of orbits for a given galaxy, we need to take into account the different orbit types and their properties.
We use the Schwarzschild orbit superposition method to generate and fit a library of orbits to a galaxy. It is based on Martin Schwarzschild's original method (Schwarzschild, 1979) but has been modernized and extended since (e.g. van den Bosch et al. (2008); Thomas et al. (2004)). We start with an LOSVD and the brightness distribution of a galaxy. On this basis, a gravitational potential is calculated, and, from there, a set of orbits. It is then fitted to the data until the difference between the model and the observations is small enough. While the original method was made for a simple model of a triaxial galaxy, the method has since been expanded to be more efficient and to reproduce more realistic luminosity distributions (van den Bosch et al., 2008). The luminosity is calculated by de-projecting the surface brightness. Such a complex surface brightness (SB) profile takes into account multiple components of a galaxy by using a Multi-Gaussian Expansion (MGE); see Cappellari (2002) and Emsellem et al. (1994):

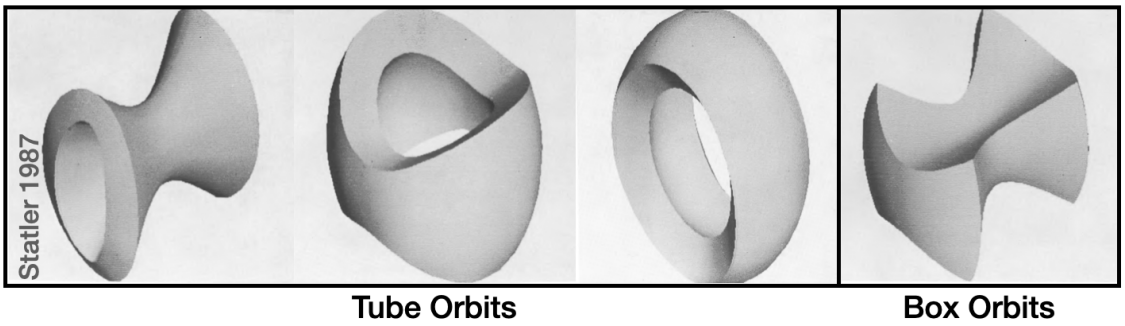


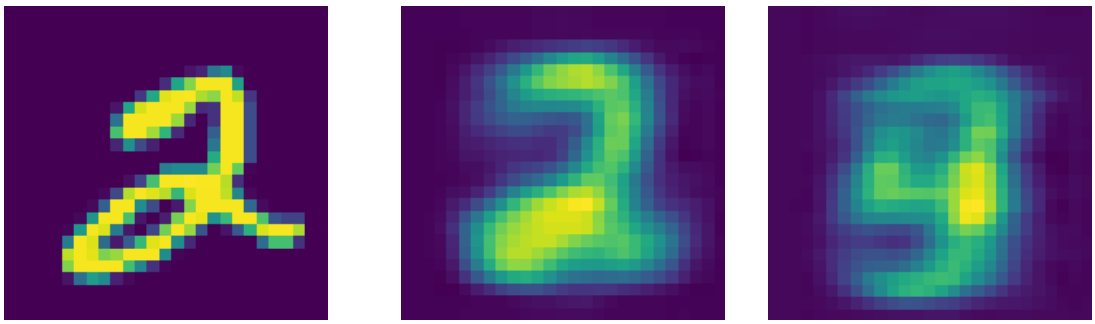
Figure 1.3.: Orbit families; images are taken from Statler (1987). From left to right: Inner long-axis tube, short-axis tube, outer long-axis tube, box type orbits. Tube-type orbits avoid the center and exist around the long and the short axis of the galaxy. Box-type orbits can cross the center and have many different shapes.

$$\mathrm{SB} = \sum_j \frac{L_j}{2\pi\sigma_L^2 q_j} \exp\left(-\frac{1}{2\sigma_L}\left(x_j^2 + \frac{y^2}{q_j^2}\right)\right) \tag{1.4}$$

Eq.(1.4) shows the MGE of the projected two-dimensional surface brightness. It is a function of the projected coordinates x, y, the axis ratios p, q, the luminosity L and the luminosity dispersion σ_L. Once the surface brightness profile is calculated, the gravitational potential can be derived. This takes into account multiple components of the galaxy: the central black hole, the dark matter distribution, the stars, among others (more on that in Chapter 2). Based on this potential, a first set of box and tube orbits is calculated. For this set, we calculate weights and check how well the orbits fit the observational data. If the orbit library does not fit the data well enough, a new set is generated, based on an improved potential. This process is repeated until we get the best fit. We show and explain the methods used here in Chapters 2 and 4, where we use an implementation of the Schwarzschild orbit superposition method: DYNAMITE.
In the end, we obtain a library of orbits that reproduce the galaxy kinematics, and with this, we get a much deeper understanding of the galaxy. There are many steps involved in the process of orbit superposition and the iterations of finding the potential, orbits and weights are computationally costly. But maybe we can speed up the process? Can we use few iterations to get a satisfying result? Can we train a network to predict physical data based on a set of parameters?

1.4. Making predictions with Deep Learning

There are many different types of neural networks. Machine Learning (ML) has a number of different applications in the field of physics and astronomy (see Carleo et al. (2019) for an overview), and when trying to solve a problem with machine learning, we have to decide what type of network works best in the specific case. Depending on the input data, ML can be used for classification, decision making, predictions, and more. Some examples for different applications are the estimation of photometric redshifts using decision trees (Carrasco Kind and Brunner, 2013), identifying exoplanets in stellar light curves using classification with a convolutional net (Shallue and Vanderburg, 2018) and predicting the origin of globular clusters using a feed-forward net (Trujillo-Gomez et al., 2023).



(a) MNIST input image, label: 2 **(b)** MNIST prediction, label: 2 **(c)** MNIST prediction, label: 3.6

Figure 1.4.: Example predictions with machine learning. The image to the left (1.4a) shows the input image: a handwritten '2'. The images in the middle and on the right show predictions after the network was trained. (1.4b) is a prediction of the number '2'; (1.4c) shows that the network is also able to interpolate between labels: it predicts an image for '3.6', which looks like intermediate stage between '3' and '4'. The network used here is a deconvolutional net.

Machine Learning uses a trained neural network to process an input into a desired output. The network learns a connection between the two, which can be applied to new input data once the network is trained.
The MNIST data set[2] (Lecun et al., 1998) is often used as an illustrative example. It is a set of images with handwritten digits. We show an example in Fig.1.4. A network can be trained to recognize the digits, i.e. to make a connection between a number (label) and a given image (input data). Once trained, it can generate (i.e. predict) an image given a certain number. With a set of test data, which has to be different from the training data, the network can associate numbers to new images with unknown labels.
For the large number of applications, there are many different types of networks, distinguished by their structure. Each network is composed of multiple layers: the input layer, hidden layers (also called deep layers) and the output layer. If deep layers are used, the term Deep Learning (DL) applies. Every layer contains 'neurons' that are connected to the following layer. A network is defined by the type of layers used and their connections. The simplest network only has an input and an output layer and is fully connected, which means that every input neuron has a connection to every output neuron. Most neural nets have a more complex structure. This will be explored further in Chapter 3.
We will now give some mathematical background. The ML approach to a problem can be simplified as a minimization problem. If we consider a network Φ and a set of training data (x, y), x being the data and y the associated labels, then the network is trained when the difference L between the output $\Phi(x)$ and the labels y is minimal (Goodfellow et al., 2016):

$$L = \sum_i \|\Phi(x_i) - y_i\| \tag{1.5}$$

Training the network is equivalent to finding a set of matrices $A_{1,2,...M}$ that are applied to the input data x:

$$\Phi(x_i) = A_M \cdot s\left(A_{M-1} ... \cdot s(A_1 x)\right) \tag{1.6}$$

[2] publicly available here.

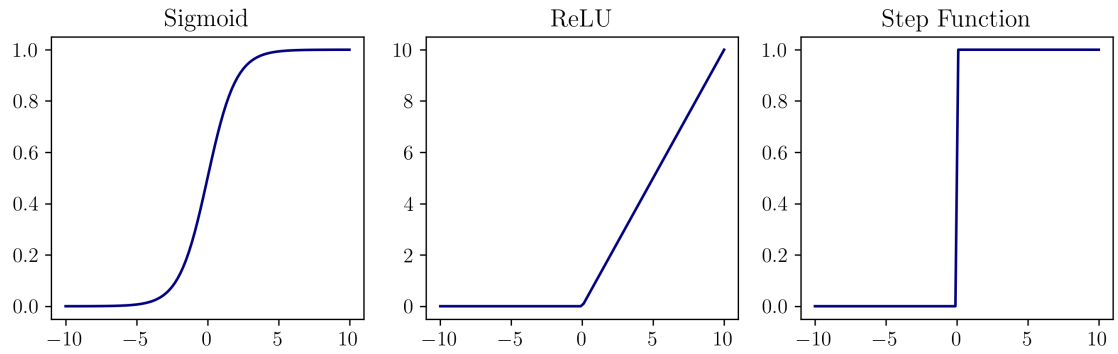


Figure 1.5.: Three possible activation functions. Sigmoid functions (left) are a group of functions with similar properties. The logistic function (depicted here) is one of them. They allow for a smoother threshold than Step functions like the Heaviside function (right). The graph in the middle shows the ReLU function. It is 0 for negative and linear for positive values. All functions have the same purpose but slightly different effects on the output.

The function L is called *loss function* (1.5). There are many different losses, in fact they can be defined individually for each problem. This will further be discussed in Chapter 3.
The parameter s in Eq.(1.6) represents the activation function[3]. It serves the purpose of a threshold and decides if a neuron is 'activated' or not. Similar to the loss, one can choose from different types of activation functions depending on the problem (Nielsen, 2015). Fig.1.5 shows three different activation functions: Sigmoid, ReLU and the Step function. Each layer in the net needs an activation function and the choice of the function influences the training and the results.
With these basic ingredients, a neural net can be built. There are different types of nets. Next to the simplest case, the fully connected net, there is the Convolutional Neural Net (CNN) that uses sparse connectivity or even more complex types like the Generative Adversarial Network (GAN), which is built with two separate networks. While we will briefly discuss the GAN in Chapter 5, we will focus on the convolutional net here and in the rest of this work.
The main advantage of a CNN over the fully-connected net is the above mentioned *sparse connectivity*: not all neurons are connected to all other neurons, however the information still passes from one layer to another. This is achieved by using kernels. A kernel can be understood as a matrix that is applied to an input vector. In this process, the information in the input, i.e. the features, gets grouped together and prioritized. To give an example, this is useful when training with image data: the pixels in the center contain most of the information, e.g. a face or an animal, while the pixels at the edge only show unimportant background. With the use of kernels, the convolutional net is able to prioritize this information in the data. Additionally, networks with sparse connectivity can be faster and save computation time and memory. In Chapter 3, we will build a special type of CNN: the Deconvolutional Neural Net (DCNN). While its structure is similar to the convolutional net, it performs the inverse process: it performs convolutions of the learnt features to produce input-like data (Zeiler et al., 2010). In the example of handwritten digits in MNIST: instead of finding labels for input images, it learns how to construct an image based on a input number.
In the following chapters, we will generate image data from kinematics, train a DCNN with it

[3]It is usually denoted with σ, but this symbol will be reserved for the velocity dispersion.

and look at possible predictions. Deep learning has some limits that we have to keep in mind. Instead of finding the global minimum in the process of minimizing the error, it can happen to 'get stuck' in a local minimum. This problem can be solved by using larger sets of data. The input data can also be the source of other issues. The predictions, however accurate they are, can only be as good a the training data. If it is biased or unbalanced, predictions will reproduce this. To give an example, if we want to predict galaxy images and the training set contains significantly more ellipticals than spirals, then the network will be much better at predicting images for elliptical galaxies. With this knowledge, we will will take a critical look at the predictions in Chapters 6 and 7.

2. Generating orbit libraries

Using Schwarzschild's orbit superposition method, we want to generate a library of orbits for the galaxy NGC 6278. The code DYNAMITE (Jethwa et al., 2020) is an implementation of the method developed by van den Bosch et al. (2008). We use this code to model an orbit library based on specific parameters and to calculate Gauß-Hermite-coefficients describing the LOSVD for the generated orbits. Then, we make some simplifications and generate sets of data to train a deep learning algorithm.

2.1. Schwarzschild orbit superposition with DYNAMITE

DYNAMITE is a tool for orbit modeling based on the code from van den Bosch et al. (2008). It was first released in 2020, we will use version 3.0.0. from May 2022.[1] While the basis of the code is written in FORTRAN, it has a Python wrapper, and all functions can be called within Jupyter Notebooks.[2] Using kinematic data and an MGE for the surface brightness, DYNAMITE can generate dynamical models of a galaxy.

2.1.1. Input

The code can compute orbit models for a wide range of galaxies, as long as the right input is provided: an MGE of the galaxy and kinematic data, e.g. from integral field spectroscopy. DYNAMITE has implemented routines to generate its own input from there, like the configuration file we will be using. We use IFS data from the CALIFA survey (Sánchez et al., 2016) for the galaxy NGC 6278. This data has already been used with DYNAMITE for test and demonstration purposes, e.g. in the tutorials. This comes with the advantage that the galaxy has been tested with the code, so no problems concerning the input format should arise and the results can be checked and compared with existing orbit libraries.

2.1.2. The galaxy: NGC 6278

NGC 6278 is a lenticular galaxy (Buta, 2019). The galaxy has a redshift of $z \approx 0.009$ and a Hubble distance of 41.8 Mpc (Cappellari et al., 2011), which makes it a relatively close-by object. CALIFA integral field spectroscopy provides a data set with three-dimensional data (x, y, λ), from which kinematic maps can be made. Figure 2.1 shows the mean velocity v and the velocity dispersion σ of this galaxy. These maps are the foundation for all further calculations: we use them for the orbit models and their observational errors for the training of the neural net.

[1] `https://github.com/dynamics-of-stellar-systems/dynamite`
[2] See the DYNAMITE documentation: `https://dynamics.univie.ac.at/dynamite_docs/index.html`

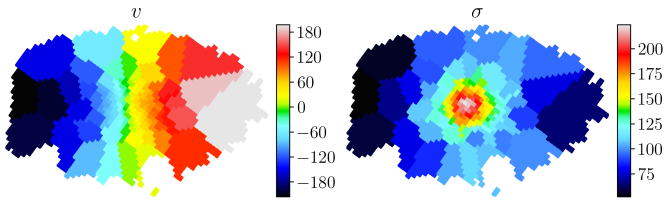


Figure 2.1.: Kinematic maps from spectroscopic observations of NGC 6278. The mean velocity shown on the left indicates that the galaxy rotates. The blue side moves towards us and the red side away from us. The axis of rotation is vertical in the center of the map. The right map shows the velocity dispersion. It is high in the center, indicating more chaotic motion, and low in the outer parts of the galaxy, where regular rotation dominates.

2.1.3. Output

With this data, we can generate dynamical models, given in the format of a table. Each row corresponds to a different model with a library of orbits, which is used to calculate weights and Gauß-Hermite coefficients. The output can be represented in different ways. DYNAMITE has an implemented plotting tool which generates plots of the orbits, kinematic maps and more. In Chapters 4 and 7, we will have a look at the plots when we visualize our results.

Now that the input is ready, we can finally generate orbit libraries. Then, we try to find a representation of the results to generate a training set for the deep learning algorithm. But first, we have to familiarize with the functions and options DYNAMITE provides.

2.2. Generating orbit libraries

DYNAMITE uses a set of tools for the dynamical models. All important information for the models is stored and accessed in the configuration file. Once the models are run, we can evaluate them with the χ^2-test and plot the results, or change the configuration parameters and re-run the models. All important options are set in the configuration file and the respective configuration object (used in the code). It contains information about the input galaxy, restrictions for the models and other settings, e.g. the type of weight solver. In this file, we can determine how models are generated in the iterative process by setting the grid options and the conditions for the parameters. The option `FullGrid` sets up a grid that spans a range over all variable parameters. The model will be iterated over all different parameter values. We choose this option to be able to control the number of generated models, otherwise it will stop once the stopping condition is reached. In the configuration file, we decide on the parameters varied in each iteration, define when the process stops, and which tool of evaluation we want to use. We choose the kinematic χ^2-method, which will be discussed in Chapters 4 and 6.

2.2.1. The model parameters

In Schwarzschild orbit superposition, the gravitational potential is generated based on multiple galaxy components. In each model, the components are slightly modified, i.e. is assigned a different value. The components include:

- **ml**: mass-to-light ratio (M/L). This parameter quantifies the ratio of the galaxy mass to the light of the stellar component.
- **p-stars**, **q-stars**, **u-stars**: p and q are axis ratios, u is a function of p and q.
- **f-dh**, **c-dh**: dark matter halo components
- **m-bh**, **a-bh**: black hole parameters

We can set these parameters to be fixed or variable during the model iterations. The more fixed parameters we use, the simpler the model generation.

2.2.2. Extracting kinematics

Once all models are generated, we calculate the corresponding orbit libraries and transform them into observable quantities, i.e. Gauß-Hermite coefficients h_1, h_2, h_3, h_4. Later, we will calculate v and σ from the resulting LOSVD, but for now, we only look at the coefficients h_i. From every model, DYNAMITE computes a library of 360 orbits over 152 spatial bins. They can be visualized in two-dimensional maps: The x-axis counts the bins and the y-axis the orbit number, the value of h_i is given by the pixel intensity. In the resulting images (Figure 2.2), we notice a certain structure. There is a horizontal separation at 2/3 of the orbits (no. 240). It originates in the way DYNAMITE calculates the orbit libraries: the top 2/3 are tube-type orbits, the bottom 1/3 are box orbits. The color ranges from blue ($h_i = -0.01$) to red ($h_i = 0.01$). White corresponds to 0. Each of these images can be read as follows: Every row corresponds to an orbit. Where this orbit is red, the GH-coefficient is positive, and negative where it is blue. In the spatial bins where the coefficient is 0, the orbit does not exist. Some types of orbit are only present near the center of a galaxy, other types can be found almost anywhere. This is a property we have to keep in mind for the predictions.

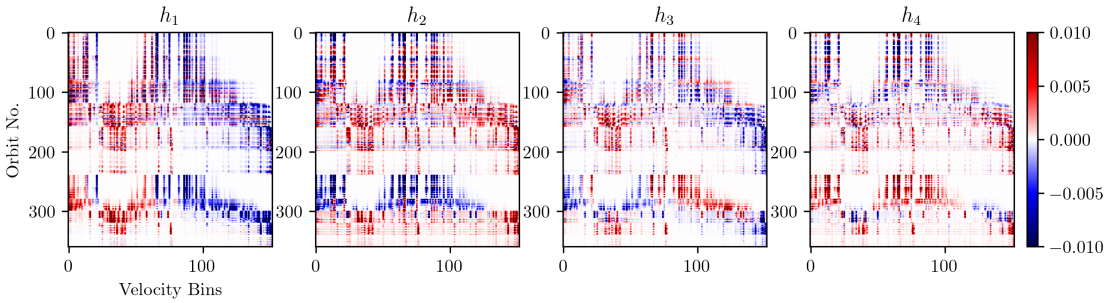


Figure 2.2.: Example images of the Gauß-Hermite coefficients h_1 to h_4. The colorbar shows the the coefficient values. There are 360 orbits for 152 spatial bins. The upper 2/3 of the orbits are tube-type orbits, then lower 1/3 are the box-type orbits. In all images, we notice a separation into blue and red regions, which is most visible in the box orbits.

2.2.3. Choice of parameter: M/L

Each parameter has an influence on the resulting model, and the more parameters are variable, the more complex the model will be. The simplest case is to keep all parameters fixed, except for

one. The resulting models will only differ in one value. To find out what parameter to choose, we generate the first orbit libraries with a very limited number of models. Only f-dh, c-dh, p-stars, q-stars and M/L are examined. From a look at the velocity maps and some test plots, we find that the variation of M/L has the largest influence on the model. The mass-to-light ratio also has a large range in values (from 1 to 9). Other parameters are in logarithmic scale (f-dh) or depend on each other (q-stars, p-stars). M/L is an independent parameter with a large range of values, and it has a high influence on the models, therefore it is ideal for our purpose. In Table 2.1, we list the first 5 models to show how the parameters evolve. All parameters except for M/L are fixed, and their values are set in the configuration file.

orblib	m-bh	a-bh	c-dh	f-dh	q-stars	p-stars	u-stars	M/L
`orblib_000_000/ml5.00`	100000.0	0.001	8.0	10.0	0.54	0.99	0.9999	5.00
`orblib_000_000/ml4.95`	100000.0	0.001	8.0	10.0	0.54	0.99	0.9999	4.95
`orblib_000_000/ml4.90`	100000.0	0.001	8.0	10.0	0.54	0.99	0.9999	4.90
`orblib_000_000/ml4.85`	100000.0	0.001	8.0	10.0	0.54	0.99	0.9999	4.85
`orblib_000_000/ml4.80`	100000.0	0.001	8.0	10.0	0.54	0.99	0.9999	4.80

Table 2.1.: The first 5 models from our calculations. All parameters are fixed except for the mass-to-light ratio.

2.2.4. Orbit distribution

To obtain the distribution of orbits for a model, we apply weights to the orbit library. The resulting distribution can be visualized in a distribution plot, see Figure 2.3. The orbits types, categorized by their circularity, are mapped as a function of radius. Near the centre of the galaxy, the distribution of orbits is relatively even. Tube orbits ($\lambda \rightarrow 1$) and box orbits ($\lambda \sim 0$) are both present. Towards larger radii, the orbit distribution shifts. Box orbits are more rare, while highly circular tube orbits appear. However, we only used limited and simple models for the orbit library. Zhu et al. (2018) have studied this galaxy and produced more detailed results.

2.3. Generating deep learning input data

We decide to hold all parameters fixed, except for the mass-to-light ratio. The Gauß-Hermite coefficients from the generated models can be visualized in 2D maps (Figure 2.2). This means that each of these maps has an associated M/L value. What if we could feed this data to a neural net and train it to generate GH-maps for a given parameter value? The generation of models in DYNAMITE needs a lot of computing time, but once a network is trained, the generation of an image is very fast. Let us explore if we can train a neural net to produce Gauß-Hermite maps that are comparable to DYNAMITE results.

To train a deep learning network, we need a large training set, so we have to maximize the amount of data generated by DYNAMITE. We have the following restrictions: Only one model parameter is variable, but we still want to generate as many models as possible. Additionally, as we have limited computational resources (storage, computing power and time), compromises are made. With the option 'full grid', we generate a grid of models with different M/L-values. To get as many models as possible, we set the boundaries to a minimum of $M/L_{min} = 1.0$ and a

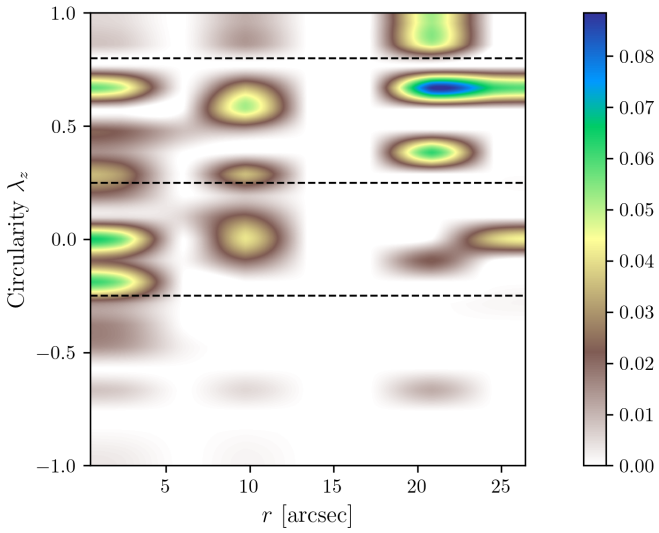


Figure 2.3.: Orbit distribution in NGC 6278, made with a simple orbit model. We show the orbit density as a function of circularity and radius of the galaxy. Tube orbits have circularity 1 and -1 (counter-rotation), box orbits have circularity 0. The distribution of box orbits is denser in the center, while tube orbits are distributed more evenly in the galaxy.

maximum $M/L_{max} = 9.0$. DYNAMITE sets up a grid from 1 to 9, with a step of 0.05 (also set in the configuration file). The goal is to generate a minimum of 100 models. With this step size, we achieve $8/0.05 + 1 = 160 + 1 = 161$ models[3]. Generating this amount of models takes 10 minutes for 161 models, and the size of the resulting data set is 282 Mb. For more models, we would need to vary other parameters too, which would take even longer. With the compromises made above, we already reach the limits of our available computing power. With larger and more complex sets of data, a more powerful computer would be necessary.

We generate 161 models with DYNAMITE, from where we compute the Gauß-Hermite coefficients for each corresponding orbit library. The generated data set has the shape `(161, 360, 152, 4)`. To train the neural net, we split it into 4 different sets, one for each h_i. The new training sets will contain 161 images with size 360×152. With deep learning, we will try to predict the images, then analyze and evaluate them.

[3] +1 because of the 0th model

3. Building a deep neural network

3.1. Choose and build a neural net

Now that the input data is ready, we can focus on the neural net. Before running predictions, we have to choose what type of network fits our problem. Based on our choice, we design a model structure and build the network before training it. This process involves some trial and error in order to find the best model structure and to adjust all of the parameters.
We use python libraries for the implementation of the network. The library *Keras*, embedded in the framework *TensorFlow* (Abadi et al., 2015), provides all the functions necessary for the project.[1] With Keras, the basic approach to set up a neural network involves multiple steps, which are all implemented with built-in functions. First, we **build** a model by from multiple different layers. Then, we **compile** and **fit** the model with the input data and labels. In the end, we are able to **predict** new images using the trained network.

3.1.1. Input Data and Labels

The type of network depends a lot on the input and the goal of the training, so we have to be aware of the properties of the input first. In Chapter 2, we generated maps with values of Gauß-Hermite coefficients for each orbit and bin. We treat them as images with size 360×152 pixels where the pixel value corresponds to the GH-coefficient. To simplify the training, we build four networks to train the coefficients h_1 to h_4 separately. The images show a specific structure that we want to reproduce, because it contains physical information on the orbits. There is a separation between box and tube orbits (see Chapter 2) and large areas are white, where the pixel value is 0. In Chapter 7, we will see how well complex structures can be reproduced with our limited data. For the training, the input data has to be labelled. Our data set was generated with only one variable parameter: the Mass-to-Light ratio. We can therefore associate one M/L-value with each image and use this value as a label. With this set of labels and images, we are ready to choose a type of neural net.

3.1.2. The choice of a DCNN

Convolutional Neural Nets have a wide range of applications, and image classification is one of them. They are able to extract key information from input data by dividing the input into overlapping subsets. The neurons in the convolutional layer are only connected to these subsets, which allows to group information. The net detects features in the input by assigning weights to the neurons. It is called 'convolutional net' because the weights are applied to an input in a convolution operation (Nielsen, 2015). In a simplified way, convolutional nets apply layers of convolutions to an input to extract the features. We want to predict GH-maps with corresponding M/L-values, so we choose to build a Deconvolutional Neural Net. It is similar to a CNN and uses

[1] Keras Documentation: `https://www.tensorflow.org/api_docs/python/tf/keras` or `https://keras.io/api/`

similar techniques, but it performs the inverse process. The DCNN aims to generate input by learning the features (Zeiler et al., 2010). In general, image predictions can also be done with other types of network. In Chapter 5, we will have a look at a Generative Adversarial Network (GAN) and compare the performance. But first, we focus on the DCNN.

3.1.3. Building the DCNN: Layers

A Deconvolutional Neural Net, like any other deep learning network, is built from multiple layers. Each layer has its own function and the order of the layers influences the result. Let's have a look at the types of layer we are going to use.

- **Dense**: This is the simplest layer. It is fully connected, meaning that all neurons are connected with each other. Dense layers are often used as starting point in a model.
- **Conv2DTranspose** (Deconvolution): The transposed convolution layer, also called deconvolution, is the most important layer here. It performs convolution operations on feature maps and generates and output which corresponds to the features. We set the shape of the output in the parameters of this layer. In our model, we use the two-dimensional version for image input.
- **Reshape**: This layer reshapes the input tensor. For our model, we need to reshape the input multiple times, e.g. from a flattened 1D vector to a 2D matrix.
- **Upsampling**: We use this layer for up-sampling the input. It can enlarge the input shape by repeating rows and columns in the input tensor.

With these layers, we can build a basic model for the neural net. However, the data we use for training requires a more complex network, as large areas in the images have a pixel value of 0. In physical terms, this signifies that an orbit does not exist there. In numerical calculations, 'exactly zero' is a difficult value because computations have finite precision. During the first test training using a simpler model, the network does not achieve to reproduce the blank areas correctly. We show an example in Figure 3.1, where the network produces vertical lines instead of the complex structure we expect.

To solve this problem, we implement a 'mask' into the model. It is a part of the net which is trained separately on the input. We use it to learn where the pixel value in an image is 0. The mask is built from two Dense layers. It does not undergo the deconvolution process, and the resulting tensor is multiplied with the result from the other layers at the end. The structure of the model can be illustrated with `tensorboard`[2], as shown in Figure 3.2. For a more detailed but less graphic depiction, the structure with the different layers is listed in Table B.1 in the Appendix. This final structure of the net was provided by Philipp Petersen.

3.2. Hyperparameters

The training of a neural net depends on more than just the model layers. Other important parts are the activation and loss function and parameters like the batch size and the number of epochs. The activation function is used in the model architecture. Multiple types of function can be

[2] `https://www.tensorflow.org/tensorboard/get_started`

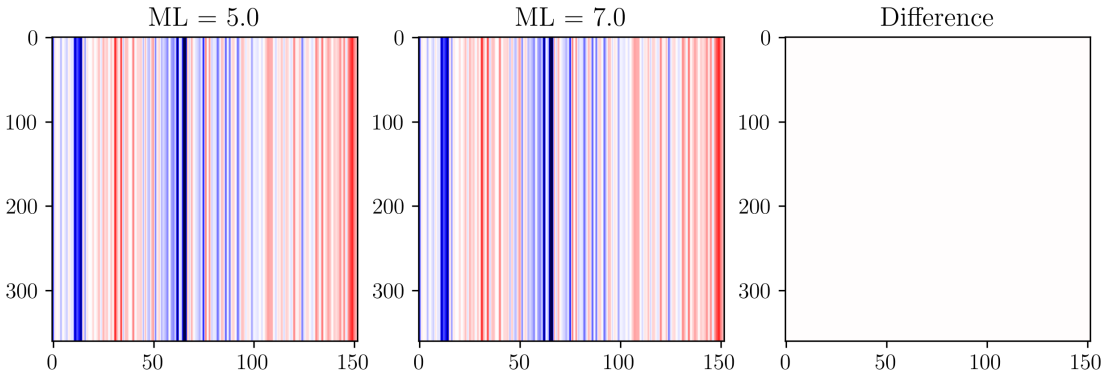


Figure 3.1.: First predictions of h_2 with a deconvolutional net. The images only show vertical stripes. The network has problems with reproducing the structure of the image, as it predicts constant values for each x-value. The blank areas are not reproduced at all. The images on the left and in the middle are predictions for different M/L values. The right image is the difference between the two, which shows that they are identical.

implemented in one model as they are part of the layers. In our model, we use the functions `ReLU` and `sigmoid`. Losses are available as built-in functions, but can also be implemented separately. We implement and test several different functions here; more on that in Chapter 4.

3.2.1. Loss function

The loss function is very important in the training of a neural net. After each training step, the loss is calculated. It is used as an indicator for the accuracy of the net, by computing the error of the output generated by the network compared to the data. A very common loss is the mean squared error (3.1). It corresponds to the difference between $\Phi(x)$ and y using the *Euclidean norm*.

$$L_2 = \sum_i \left(|\Phi(x_i) - y_i|^2 \right) \tag{3.1}$$

Loss functions have to be adapted to the data we are using, as every function has a different effect on the final output. The mean-squared loss can lead to smoothed images, which can work for handwritten digits, as in the MNIST example, but is not desirable in our case. Alternatively, we can use the *absolute-value norm*. In the context of machine learning, the Euclidean and absolute-value norm are often called L2- and L1-Norm.

$$L_1 = \sum_i \left(|\Phi(x_i) - y_i| \right) \tag{3.2}$$

For now, the loss is an abstract number measuring how good the net performs. To include the data quality into the process, we divide the loss by the observational error ϵ. We apply this normalization to Eq.(3.1) and (3.2):

$$L_{2/\mathrm{err}} = \sum_i \left(\frac{|\Phi(x_i) - y_i|^2}{\epsilon_i} \right) \tag{3.3}$$

$$L_{1/\mathrm{err}} = \sum_i \left(\frac{|\Phi(x_i) - y_i|}{\epsilon_i} \right) \tag{3.4}$$

One could implement other loss functions, e.g. using different norms or introducing more parameters, as long as they quantify the difference between $\Phi(x)$ and y. The loss functions (3.1)-(3.4) are implemented and tested in Chapter 4, then the best one is chosen for the final predictions.

3.2.2. Other parameters

The performance of a network can also be improved by tweaking the hyperparameters. While some of them are found with trial and error, others depend largely on the input data. The number of epochs corresponds to the number of iterations the network goes through before the training is done. The optimizer uses gradient descent to make sure that training converges, e.g. the loss function reaches a minimum. We choose standard values for the epochs and the optimizer. For the other parameters, we have to take into account the small size of our input data. The validation split separates the data into a training and a validation set, so a sensible value here would be around $20 - 50\%$. Because our set is so small, we choose 10% to keep the actual training set large enough. We set the batch size to 2, which is a small value. This parameter groups the data into batches that are processed at once, which speeds up the training especially for very large data sets. If the batches are small, the training is longer, which can increase the accuracy. In Table 3.1, we list the main parameters and the values we choose. While we use default values for some, we have to adapt others to the small size of our data set.

Parameter	**Value**
Learning rate	0.01
Epochs	20
Validation Split	0.1
Batch Size	2
Optimizer	Adam

Table 3.1.

3.3. Train and Predict

There are many different ways to set up a neural net. We have to adapt it to certain conditions, in our case a very small set of input data and limited computational resources. Once the network is built, we start the training. We use the same network architecture to train 4 different nets, one for each GH-coefficient. When the loss is small enough or the maximum number of epochs is reached, the training is done. For the predictions, we generate a new set of labels over the range of possible M/L-values and let the network predict the corresponding images. In Chapter 4, we have a look at the results.

To predict Gauß-Hermite maps, we choose to build and train a deconvolutional net. The net has to be adapted to our input by setting the hyperparameters and carefully choosing the layers.

Libraries like Keras make this process easier and more accessible. After the network is built and tested, it is trained with the input data and predictions can be made. In the next chapter, we will have a look at our first results. What can be improved, how do the different loss functions perform and how do we compare the results?

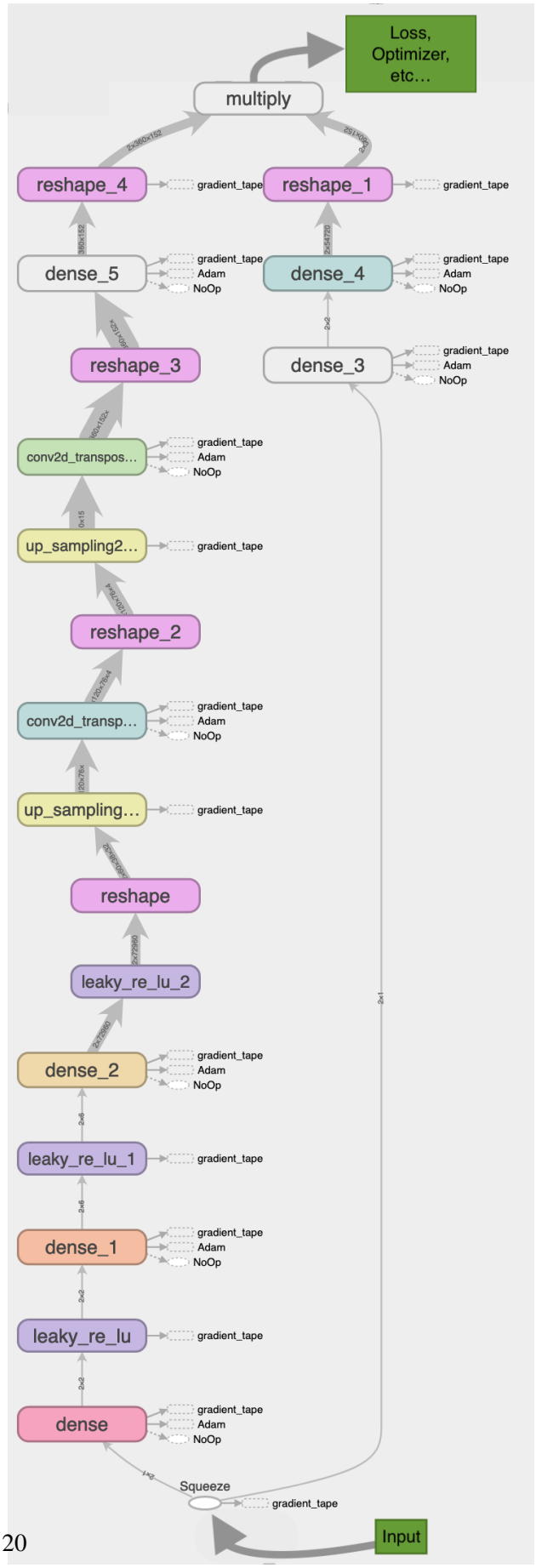


Figure 3.2: Schematic representation of the network model, generated with `tensorboard`. This model has two branches: the left branch generates the image through dense and deconvolution layers, the right branch generates the mask that is then multiplied with the image in a final step. After the multiplication, the loss function and the optimizer are applied to the model.

4. First Results

For the galaxy NGC 6278, we modeled an orbit library and prepared a set of input data for a neural net. The input data contains 161 different models with maps of 4 Gauß-Hermite coefficients, for 360 orbits and 152 spatial bins. This results in the following data format: $(161, 4, 360, 152)$. We built a Deconvolutional Neural Net for two-dimensional input data, so we separated the input into 4 sets and trained 4 different networks, one for each coefficient h_i. In this chapter, we will look at the first predicted images and compare the results for different loss functions.

4.1. First Predictions

The first predictions with the DCNN look quite similar to the DYNAMITE models. We train the network that we set up in Chapter 3, then we make predictions. With Keras, predicting with a neural net simplifies to running a single command. The network is given an input label, in our case a M/L value, then it produces an image based on this value. This only works if the input is within the same range as the labels used for training, so values smaller than 1 and larger than 9 will not work here. Most known galaxies fall into this range (Faber and Gallagher, 1979), so this won't be an issue here. To compare the predictions to the input, we use M/L values similar to the ones used in the DYNAMITE grid. We show the first predictions for Model 0, which corresponds to M/L = 5.0 in Figure 4.1. There, we calculate the difference between the input maps and the predicted model and normalize it by the observational error. This model (Model 0) is not necessarily the best fitting-model, as we will investigate later, instead we use it only for comparison. It is the first model generated by DYNAMITE.

First impression When comparing the first predictions to the DYNAMITE maps, the images look very similar. Differences only show if we look very closely. This first impression is confirmed when we calculate the residuals (see Figure 4.1). The difference is smaller than the observational error of the original data, and there is no apparent systematic difference. The neural net produces results that are very close to the input data, which is a first indicator that we can use our predicted data for further DYNAMITE calculations.

For the first predictions, we trained a neural net and used the 'mean-squared-error' loss function. While they are quite good already, we want to know how the losses discussed in Chapter 3 perform, and if they are possibly better suited for our data. To compare the different losses, we have to implement the loss functions into our neural net, train the net, make predictions and find a way to compare the results.

4.2. How good is a model? The chi-squared test.

For any further models, predictions and comparisons, we need a tool to quantify how good (or bad) the results are. DYNAMITE uses the chi-squared (χ^2) test to constrain the parameters of the

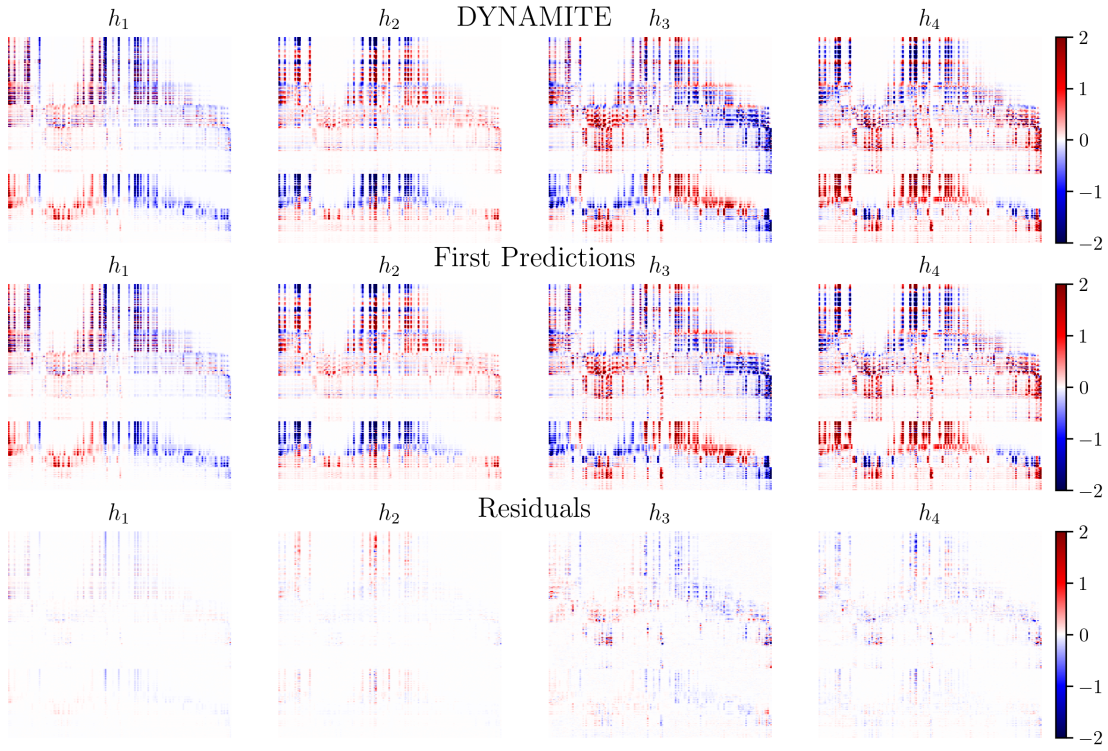


Figure 4.1.: First predictions and their differences to the input maps for M/L = 5.0. All maps are scaled to the observational error, i.e. $\epsilon = 1$ on the colorbar. The upper row shows the coefficients from the DYNAMITE model and the middle row depicts the first predictions with the new network structure. On the bottom row, we show the difference DYNAMITE - Predictions. We used the mean squared loss for this set of predictions. The maps show the same structure as the DYNAMITE model. There is a horizontal split between the tube orbits and the box orbits, and the blank spaces (where $h_i = 0$) are well reproduced, although not completely white. The maps look almost identical, so we show the differences in residual maps. The residuals are within the observational error of the data (between -1 and 1 on the colorscale), even if there are differences between the coefficients. In this example, it seems that the error for h_3 is the largest, but when looking at other models, we notice no systematic error.

model. χ^2 is a statistical tool to measure how good a fit is, or in our case, a model. It is obtained by fitting the models and computing a set of weights from the Gauß-Hermite coefficients. The model with the lowest χ^2-value is chosen by DYNAMITE as the best one. To compare our results to the data, we apply the method to the predictions. This results in a χ^2-value for each M/L, which we can depict in a plot.
We also want to compare the shape of the χ^2-curves, as narrower curves give better constraints for the parameters. For this purpose, we introduce $\Delta\chi^2$, the difference between χ^2 and $\min(\chi^2)$. When we calculate $\Delta\chi^2$ for a curve, the minimum gets shifted to 0. This allows us to compare the shapes and the exact positions of the minima for multiple curves. In Figure 4.2, we will use this to compare the use of different loss functions in the predictions. But first, we have to calculate the chi-squared values with DYNAMITE.

4.2.1. Hacking DYNAMITE

Usually, DYNAMITE uses an orbit library to directly get the Gauß-Hermite coefficients and to calculate the weights. However, we want to predict the coefficients, not the orbit libraries. So, we have to find a way to input them into the weight-solving process. To achieve this, we have to modify the source code. In theory, this could be done in the DYNAMITE distribution we usually work with, but it is better to take precautions and generate a new copy of the code, that we call `dynamite_hacked`. There, we can modify the file where the weight solver is implemented (`weight_solvers.py`) to use our predictions as input file instead of calculating coefficients from the orbit library. For the analysis, we also have to calculate the weights and chi-squared values for the original data. In the next section, we use the modified code to compare this data to predictions made with different loss functions.

4.3. Comparison of the loss functions

With the same network architecture as above, we train the networks using four different loss functions. A total of 16 networks are trained, as we have to train each Gauß-Hermite coefficient separately (four losses × four coefficients). In the previous training, we used the mean squared error loss from Eq. (3.1). Now, we implement Eq. (3.2), (3.4) and (3.3) to train the nets and make new predictions. We can then use the chi-squared tool to evaluate our results.

4.3.1. Computing time

A direct method of comparison is the computing time. The generation of input data takes approx. 10 min, so the duration of the training adds on top of this, and a shorter training can make a large difference. The computing time depends a lot on the computer performance, and the time measurements are specific to the device that is used. The fastest training was achieved with the mean-squared loss. This is not unexpected, as it is part of the Keras implementation and therefore optimized for speed and efficiency. We also notice that the computation time increases with the coefficients. The shortest training time was approximately 4 min for h_1, increasing to 10 min for h_4. When comparing the other losses, L_1 is a little slower, while $L_{1/\mathrm{err}}$ and $L_{2/\mathrm{err}}$ take the longest to train, independent from the coefficient, with a maximum of 11 min. Overall, the losses (3.2)-(3.3) are slower than the mean squared loss L_2, but the differences are tolerable. In terms of computing time, L_2 would be the best, but we will now do the chi-squared test for a more qualitative comparison.

4.3.2. Chi-Squared Test and Delta-Chi-Squared

With `dynamite_hacked`, we get weights for the predictions and calculate the chi-squared values for each model parameter, i.e. mass-to-light value. Then, we subtract the minimum to normalize the curves for an easier comparison. Figure 4.2 shows the $\Delta\chi^2$-curves. The shape reflects DYNAMITE's iteration process. For all methods, the minimum is in the small range between 5.5 and 6, which shows that all loss functions could be used for the predictions. But when looking at the shapes, the losses using the L_1-norm perform visibly better than the ones using L_2. Especially $L_{1/\mathrm{err}}$ is very close to the models, and the best around the minimum. From all the predictions, the width of this curve is the smallest. If the $\Delta\chi^2$-curve is wider, the constraint for the minimum is smaller.

From all the loss functions that we implemented, the $L_{1/\mathrm{err}}$-loss performs the best in the chi-squared test. It takes into account the observational error when computing the loss; and L_1 in general is less stable, meaning that it is able to reproduce the smaller details. L_2 on the other hand can lead to more blurry results. When comparing the time needed for training with each loss function, L_2 is the fastest, but the difference to the other three is not very large. Taking the time and the chi-squared values into account, we choose $L_{1/\mathrm{err}}$ for further predictions, as it is comparable to the other losses in computing time and produces visibly better results in the chi-squared test.

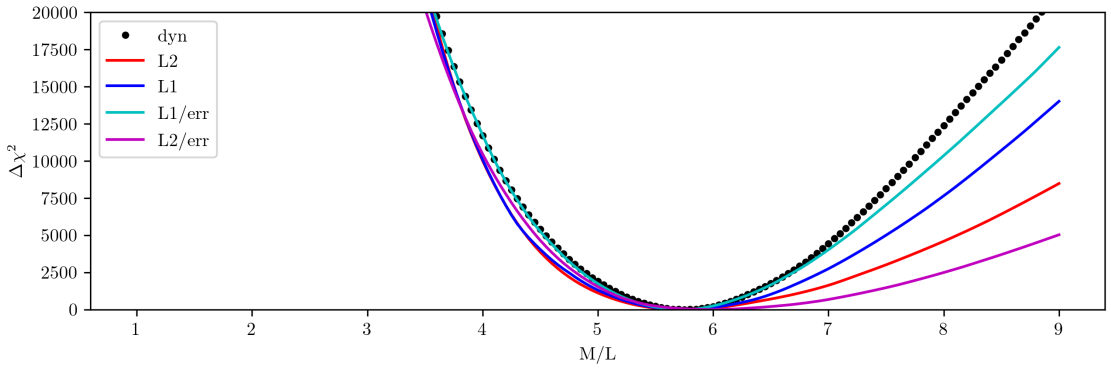


Figure 4.2.: Comparison of the normalized chi-squared values for the DYNAMITE model (dotted black line) and the predictions (solid lines). We test four different loss functions on a M/L-range from 1 to 9. All five curves have been normalized to 0 in order to compare their shapes, and the plot has been limited to $\chi^2 < 20000$ to focus on the minima. The $L_{1/\mathrm{err}}$-curve follows the DYNAMITE model very closely, they only separate around M/L$\sim$ 7. Both L_1 losses perform better than the L_2 losses. While the χ^2-value appears to be lower for the L_2 losses in this plot, we have to keep in mind that all curves are normalized to 0. We evaluate the curves according to their shape compared to the model curve.

4.4. Further changes

The first predictions are already very good, and the DCNN performs well. However, the total computing time is very high, with approx. 20 min for the generation of input data, the training and the predictions. We now want to optimize our method and explore its limits. Using the loss which performed the best ($L_{1/\mathrm{err}}$), we want to find out how much or rather how little training data we need to get satisfying predictions. First, we try a different network architecture, to see if it has a performance comparable to the DCNN. Then, we will explore the limits of our network.

5. GAN: A different network structure

We can test our data set on other types of neural nets. As we want to *generate* images, we have to use a generative network. During a lecture on data science, while different types of networks were discussed, the Generative Adversarial Network (GAN) was introduced. Next to convolutional nets, this type of net is also used for dealing with image data, specifically with generating images. In the context of the lecture, we used it to generate images of galaxies, but this presented the opportunity to try it on the Gauß-Hermite images. After a brief overview of GANs, we will train such a network with our full-sized input data set and evaluate its performance.

5.1. How does a GAN work?

We can only give a very brief description of GANs. They are object of a diverse and growing field of research. We simply want to try a different network type here, so only the basics will be covered.
Neural nets can use generative and/or discriminative models. As the name implies, generative models generate output based on a learned distribution, while discriminative models learn to recognize if an input belongs to the distribution or not (Bengio, 2009). Most networks are specialized in one of the tasks; e.g. our previously used deconvolutional net is a generative model. The generative adversarial network however uses both elements. It has two components: a generator and a discriminator. These components can be seen as adversaries. During the training of the network, the generator learns to produce images similar to the training data: it learns the joint distribution of data x and labels y. The discriminator is trained to recognize the training images and to distinguish them from the 'fake' output from the generator: it learns the conditional distribution of y given x (Mirza and Osindero, 2014).
The adversarial nature of the network comes with a problem: neither of the two nets can be trained perfectly, i.e. the loss can never be minimal for both of them. The training of a network is successful if the loss is small enough, as discussed in Sec.3. In the case of the GAN, the generator loss is low if it 'fools' the discriminator, and the discriminator loss is low if it detects the fakes from the generator. In other words: Either the discriminator detects all fakes, which means that the generator is not good enough, or it does not detect enough fakes, which implies that the discriminator does not perform well. The mathematical formulation and implementation of the loss functions are discussed in Sec.5.2.
Apart from the two-part network structure, the GAN works in a similar way as the convolutional net we used previously. Once the generator, the discriminator and their loss functions are implemented, we can train the GAN on our Gauß-Hermite parameter images and make predictions.

5.2. Implementation

Before we are ready to train, we prepare the input data, then we build our network and set the losses and parameters. First, we adjust the input data. As this section is based on a data science

exercise, we modify the code provided during the lecture.[1] The images are normalized to values between $[-1, 1]$. Then, we can build the generator. We use a slightly adapted version of the DCNN implemented in Chapter 3. The 'mask' is implemented as well. At first, we test if it is necessary, but again, the network produces very noisy output without the use of the mask, so the mask will be used in the generator from now on. The generator takes a random distribution within the value range [-1,1] and produces an image with size 360×152 from there. The discriminator does the opposite: with the image as input, it returns a one-dimensional object.
Now we implement the loss function. The generator and the discriminator have separate losses which reflect their purpose. We use the binary cross-entropy loss which is optimal if we have two cases: true and false, or in our case: real (training images) and fake (predicted images). Binary cross-entropy loss has the following formulation (Goodfellow et al., 2014):

$$L_{BCE} = -\sum_{i}^{N} t_i \log(p_i) \tag{5.1}$$

The coefficients t_i are the true labels (either 0 or 1) and p_i are the event probabilities. The generator event corresponds to having a fake image and the discriminator can have the events 'real image' and 'fake image'. For the number of events N, we have $N = 1$ for the generator and $N = 2$ for the discriminator. We use the Keras implementation[2] of this function to calculate our losses:

- Discriminator loss: $L_D = -\left(t_D \log(p_D) + t_G \log(p_G)\right)$
- Generator loss: $L_D = -t_G \log(p_G)$

In an ideal training situation, we want the loss to converge to a minimum. In the case of the GAN, if one of the losses is as small as possible (i.e. close to 0), the other loss will be large. In other words, if the discriminator detects every fake from the generator, the generator doesn't get the possibility to improve and therefore reduce its loss. To solve this problem, a balance needs to be found. We will discuss this problem later. But first, we are ready for the training of the network.

5.3. Training

To train the GAN, we follow the same procedure as for the DCNN: we set the hyperparameters, then run the training and generate predictions. Because the training of GANs can be computationally expensive, we use a GPU on Google Colab to accelerate the computations. We want to compare our results to the predictions with the DCNN, and while the networks already work very differently, we try to use as many similar settings as we can.

Computing time: The computation time for each of the data sets (h_1 to h_4) is significantly faster than the convolutional net (Chapter 4), however we did not use a GPU for the first predictions. If we try to run the code for the GAN on the laptop used previously, training a single epoch takes approximately 20s, and the training is completed after 7.7 min (20 epochs). The DCNN takes between 4 and 11 min, depending on the coefficient. In terms of computation time, both network structures are comparably slow if we don't use a GPU. The GAN has no advantage on the deconvolutional net here.

[1] Oliver Hahn, Sudeshna Boro Saikia, Florian List: 280482 VU WM-c-Dat Data Science in der Astrophysik (PI) (2022S). Code provided by Florian List.
[2] TensorFlow/Keras documentation here.

Parameter	Value
Learning rate	None
Epochs	20
Validation Split	None
Batch Size	2
Optimizer	Adam

Table 5.1.: Free parameters in the GAN. Learning rate and validation split are not used here.

Loss: We have two loss functions for the GAN which depend on each other. Where the generator loss is high, the discriminator loss is low: the discriminator recognizes the fake images produced by the generator. On the other hand, if the generator loss is low, the generator 'fools' the discriminator, even if the generated images are not very good. So, neither of the two losses can be extremely small if we want good results, but both losses should be as small as possible. Figure 5.1 shows how the two losses are connected. Even though they have a large scatter, most of points are in the region where both losses are small. This indicates that the network works correctly.

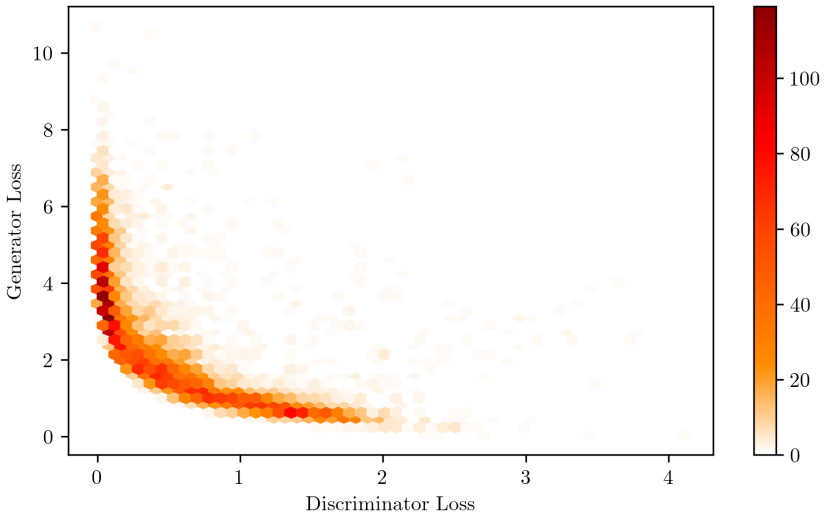
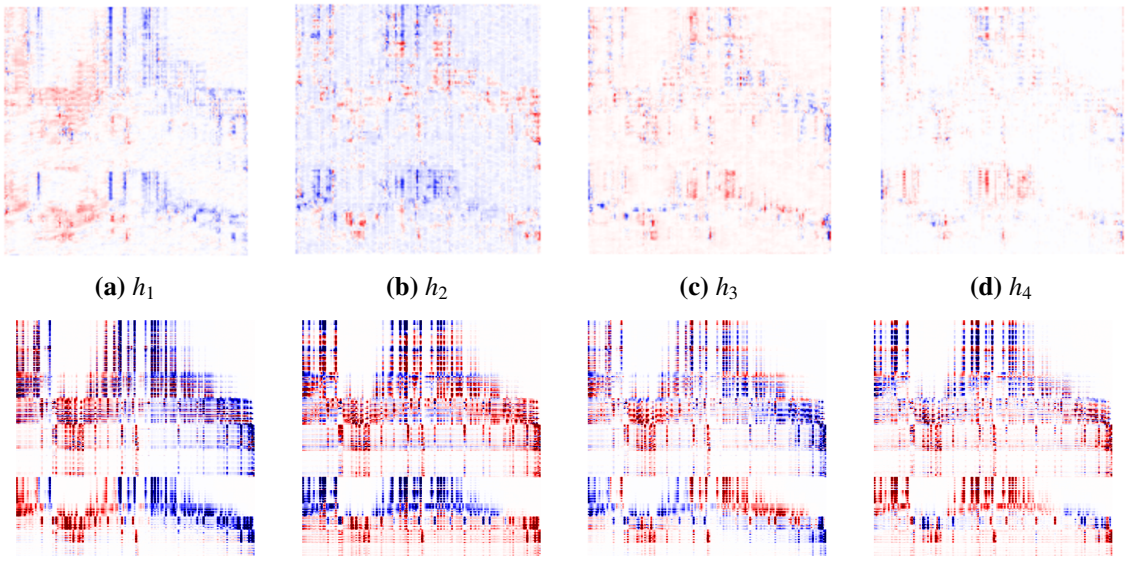


Figure 5.1.: Density distribution of the generator loss versus discriminator loss for all coefficients. The losses from all training steps and epochs are plotted here together. The generator loss is overall higher than the discriminator loss. While there is some scatter at higher losses ($L_D \gtrsim 1$ and $L_G \gtrsim 4$), the density is the highest where both losses are small. The shape of the distribution results from Eq.5.1, which describes a $-\log(...)$-function. In summary, the loss has a large scatter but is small enough to indicate good results.

5.4. Results

The GAN produces results that resemble the Gauß-Hermite coefficient maps used for the training. We show a set of predictions next to the input images in Figure 5.2. The generated images show the same main features as the input, but most of them are very faint. We are also missing the

blank areas, where the pixel value should be equal to 0. These areas are approximately constant in the predictions, but they also show some noise. If we don't use the mask in the generator implementation, the predictions are noisier and barely show any features at all, so we decide to always use the mask for the training. We include a prediction without mask in the Appendix for the sake of completeness (see Figure A.1).
There is a random element involved in the predictions, coming from the random distribution that the generator starts with. In Figure 5.2, we chose to show the best-looking images, but there is a large range in results. While all show the basic structure, some have a bias towards positive or negative pixel values, which results in mostly blue or mostly red images. The images generated with our GAN implementation go into the right direction, but they are less close to the input maps than our previous results from the deconvolutional net.



(a) h_1 **(b)** h_2 **(c)** h_3 **(d)** h_4

(e)

Figure 5.2.: Images generated by the GAN for all coefficients in a training with 20 epochs, using the mask. Figures **a-d** show the GAN predictions, Figure **e** the input images for comparison. They are randomly generated, without a given label. The images reproduce the structure we are looking for: they have blue and red areas and show the horizontal separation between orbit types (as discussed in Chapter 2). However, the blank areas are not reproduced well. While they are approximately constant, they are not white, meaning that their values are not equal to zero. The GAN predicts images that show the basic structure of our input images, but the differences to the input are much larger than for the previous predictions.

5.5. Conclusions on the GAN

Our results from the GAN resemble the input, but they don't show enough detail to be used further with DYNAMITE. We obtain the best predictions if the generator is trained multiple times and the best looking images are chosen. The random distribution used by the generator introduces an element of uncertainty. This generates results that are hard to control and reproduce, which makes

the predictions unreliable and less suited for further calculations.

Another issue with the GAN is that our implementation was based on different data (images of galaxies) and only the generator was adapted to the new input. There is probably more potential for better results in a different implementation, e.g. a conditional GAN, which works better if we want to use labels (see Mirza and Osindero (2014)). Other possibilities to improve our results would be to train longer (by increasing the number of epochs) and to use more input data. Our input data set is very small for any machine learning purpose, but it is limited by the orbit modeling done with DYNAMITE.

A GAN is an interesting type of network, but it is complex in the implementation and has no advantage over the deconvolutional net in terms of computation time in our case. The major disadvantage is that the results are not very reproducible. It is in principle possible to use a GAN for our predictions, but this would be a task for future works. For now, the DCNN works well with our data and is very reliable. In the next chapter, we will explore the limits of the DCNN and try to reduce the size of the training data.

6. Exploring the limits of prediction

So far, the predictions were successful in reproducing the input to a precision where the residuals are smaller than the observational error. However, the generation of orbit models and Gauß-Hermite coefficients for the training data is computationally expensive. The largest grid of models we achieved had a size of 161, but it is easy to go towards smaller grids. What if we trained the network on a smaller grid of models, then tried to interpolate in between? In this chapter, we generate smaller sets of training data. We test the predictions from the trained networks with the chi-squared method, as seen in Chapter 4. We calculate $\Delta\chi^2$ to constrain the parameters of the model and use it estimate the model uncertainty. In the end, we want to find out how far we can go, i.e. how sparse the input can be.

6.1. A sparse set of input data

6.1.1. Generate the data set

We want to test the limits of our neural net, so we train it on sparse data sets. The first step is to create the input data, going as sparse as possible. To generate orbit models, we previously set up a grid of parameter values, given a minimum, maximum and a step size. This resulted in 161 models. For the sparse grid, we do not have to re-do the complete process of Schwarzschild orbit modeling. Instead, we use our initial 161 models and reduce the size of the full set. We have to keep in mind to cover the full range of parameter values. To achieve this, we create the first set by choosing every second model from the full set, which results in a smaller set with size 80. Similarly, we can take every 4th, 8th, and 16th model, and reduce the size of the new set even further. We repeat this procedure until we obtain six new sets with sizes 80, 40, 30, 10, 5 and 2, corresponding to $1/2, 1/4, 1/16, 1/32$ and $1/64$ of the original set. Except for the smallest sets, which are extreme cases, they still cover the full range of M/L-values.

6.1.2. Train and Predict

Now that the sparse input data is ready, we train the neural net with each of the data sets. For the predictions, we generate an array that spans the complete parameter range for M/L and has length 161, similar to the original predictions. Even if the network only 'learns' from sparse input, it interpolates in between the values and can predict images for a M/L value in between 1 and 9. This results in a set of predictions for every input, which we can then evaluate with the chi-squared method. The predictions are further called *set 161, set 80*, etc., after the size of their training set. In Figure 6.1, we show the predictions with the lowest chi-squared value, what we consider the 'best fit models'. From the first look at the predicted images, we can use sparse sets up to a very small set size with only 10 models and still get reasonable results. But even if the training set has only 2 different models, the predictions still show recognizable structure. With six differently sized sets of training data, we can deduce a lower limit of 10 models per input set. Now, we can

compare the chi-squared values. Please note that from the point of view of machine learning, the ideal is to use as much and diverse training data as possible to get the best results. Reducing the size of the training sets is in contradiction to this, but we are focusing on the physical results instead of an optimal performance of the neural net.

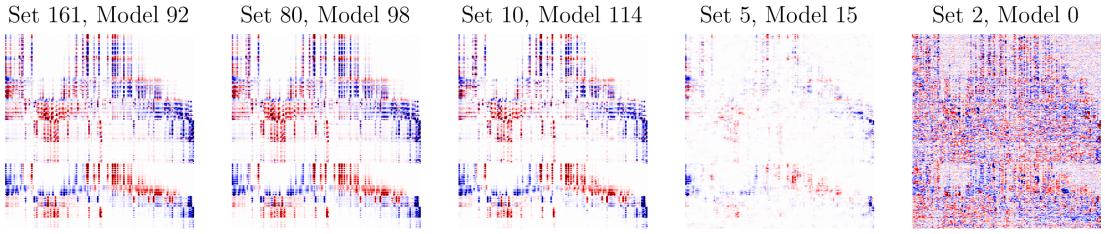


Figure 6.1.: Best predictions of h_3 for decreasing training set size from left to right. The 'best' predictions are found with the χ^2-method. The prediction from the full set (161) is shown next to the half set (80) in comparison to the very small set sizes (10, 5 and 2). Sets 80 to 20 look very similar, which is why only set 80 is included here. We start to see a change in the images at set 10, which becomes dramatic for the very sparse sets 5 and 2. The predictions from the sets in-between don't show very large differences. A detailed image with predictions for all sets is included in the Appendix (Figure A.2).

6.2. First evaluation with Delta-chi-squared

Like we did to compare the loss functions, we apply the χ^2-test to the predictions from the sparse sets. We calculate $\Delta\chi^2$ and compare the resulting curves. We use `dynamite_hacked` to fit the predicted Gauß-Hermite coefficients to the observations and to calculate the weights, which results in chi-squared values for each M/L. Figure 6.2 shows the first results. With decreasing set size, the chi-squared curve gets flatter and resembles less the curve from the DYnamics, Age and Metallicity Indicators Tracing Evolution models. We notice again that sets 5 and 2 differ from the rest. Most of the minima are at M/L= 5...7, whereas the chi squared curves for 2 and 5 are linear in this region. This supports the evidence from Figure 6.1 that we can rule them out. When comparing the shape of the curves, the largest set fits closely to the DYnamics, Age and Metallicity Indicators Tracing Evolution model, then the fit gets worse for the sparse predictions. We only see a large range of chi-squared values, and the differences between the sets are not yet visible. To get a more detailed look, we will estimate the parameter uncertainty.

6.3. Estimating the uncertainty

We want to constrain the predicted parameters by estimating their uncertainty. In the previous section, we excluded sets 5 and 2 from the calculations, and we will not use them from here onward. The confidence limits for the parameters can be derived from $\Delta\chi^2$ (Morganti et al., 2013). For the standard deviation of χ^2, we use Eq. (6.1). We adapt this from Zhu et al. (2018) who have tested this estimate of the confidence level on CALIFA data, i.e. data comparable to ours.

$$\Delta\chi^2 = \chi^2 - \min\chi^2 < \sqrt{2N_{\mathrm{kin}}} \tag{6.1}$$

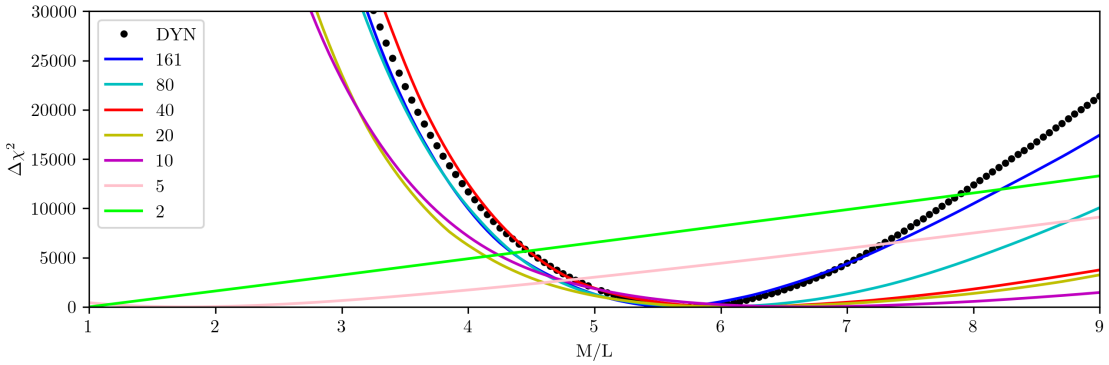


Figure 6.2.: Comparison of the calculated $\Delta\chi^2$-values for the different M/L-ratios for all the training sets. $\Delta\chi^2$ allows to compare the shapes of the different curves. The curves indicate a minimal χ^2 for parameters between M/L=4 and M/L=7. The calculations for training sets 5 and 2 are already visible as outliers here. Their minima are far from the desired region (M/L between 1 and 2) and the shape of the curve does not agree with the DYnamics, Age and Metallicity Indicators Tracing Evolution-curve.

The number of kinematic constraints N_{kin} corresponds to the number of spatial bins multiplied with the number of GH-coefficients: $N_{\text{kin}} = N_{\text{bins}} \cdot N_{\text{coeffs}} = 152 \cdot 4$. In our case, we get the following confidence estimate:

$$\Delta\chi^2 < \sqrt{2N_{\text{kin}}} = \sqrt{2 \cdot 152 \cdot 4} \approx 34.87 \tag{6.2}$$

The 1σ-confidence level for our parameters is 34.87. With $2\sqrt{2N_{\text{kin}}} \approx 69.74$, we similarly obtain the confidence level for 2σ. Now, we apply these values to the predicted data, more precisely to the curves in Figure 6.2. The values for the confidence levels are added as horizontal lines in the plot. Where the lines intersect with the $\Delta\chi^2$-curves, we measure the width between two intersection points and obtain the 1 and 2σ uncertainty for each curve, i.e. for each prediction. Finding the intersections is not trivial, as the data points are sparse on such a small part of the x-axis, there are no points exactly where the lines intersect. With 161 points on a M/L-range of 1 to 9, we are left with 40 points in the region of interest between M/L= 5.25 and 7.25. To solve this problem, we use linear interpolation between the points to create the intersections. The intersections are shown in Figure 6.3. In this plot, we zoom in even further on the axes to get a close look at the minima and the uncertainties. With decreasing set size, the chi-squared curves get more different from the DYnamics, Age and Metallicity Indicators Tracing Evolution model. The distance to the DYnamics, Age and Metallicity Indicators Tracing Evolution minimum gets larger, and the 1 and 2σ uncertainties grow, indicated by the with of the curve between two intersections. Up to set 20, the results are close to each other and to the DYnamics, Age and Metallicity Indicators Tracing Evolution model.
In the next section, we discuss the limits of our predictions. We transform the uncertainties into error bars and evaluate the predictions even further.

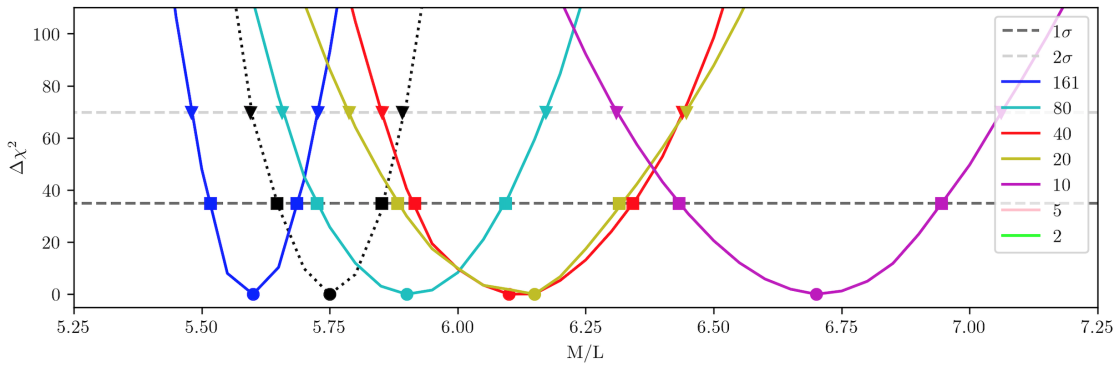


Figure 6.3.: Estimation of the 1σ- and 2σ uncertainties for the predicted parameters. The horizontal dashed lines mark the confidence levels for 1σ (dark grey) and 2σ (light grey). The different colored curves show the $\Delta\chi^2$-values for the different training sets. The intersections between the curves and the horizontal lines are marked with squares (1σ) and triangles (2σ); the width between these points corresponds to an estimate of the uncertainty. The width grows with larger set size, which fits into the picture that the predictions get worse with smaller set size. When looking at the minima (circles), we notice that they are very close to the DYnamics, Age and Metallicity Indicators Tracing Evolution model minimum. While their distance increases with smaller set size, all except set 10 are within ±0.5 M/L. This comparison confirms our first impression: the predictions get worse with smaller training set sizes, but they still produce reasonable results up to a set size of 10 (1/16 of the original set).

6.4. The limits

After we compare the predictions to the model and to each other, we define our limits: we choose a training set that performs well and is small enough to save some computing time. In Figure 6.4, we compare the M/L-values with minimal χ^2 and the 1 and 2σ uncertainties of the DYnamics, Age and Metallicity Indicators Tracing Evolution model and the predictions. Up to set 20, all results are within 2σ of the model. The minimum for set 10 is too high, and sets 5 and 2 have been excluded before. The smaller the training sets get, the more the uncertainties grow. We have an agreement within 1σ only in sets 161 and 80, but sets 40 and 20 agree within 2σ. We choose set 40 (1/4 of the full set) as our limit: it is large enough to cover the range of M/L-values and it performs well in the chi-squared test, but it is still small enough to save computing time and memory. All predictions from the full set to 1/8 of the set perform very well, which surprises a little, as machine learning is usually dependent on very large training sets. We choose to train on 1/4 of the full set and use the resulting predictions for further analysis.

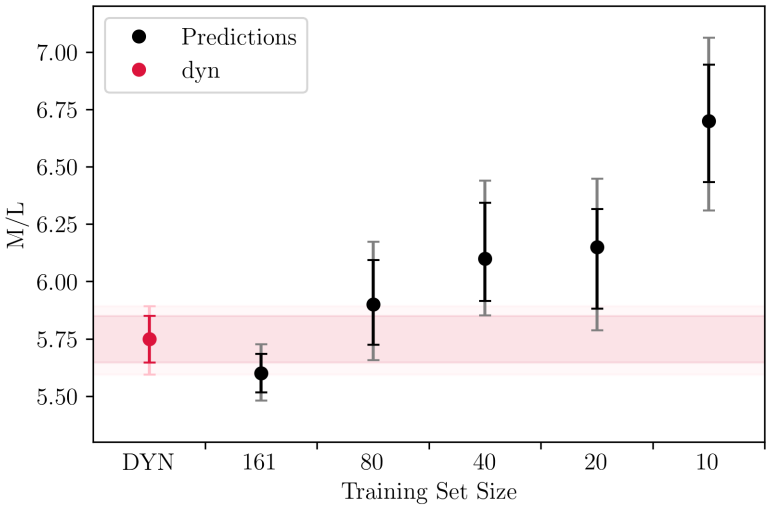


Figure 6.4.: M/L values with minimal χ^2 and their 1σ and 2σ uncertainty of the DYnamics, Age and Metallicity Indicators Tracing Evolution model in comparison to the predictions. The circles represent the M/L-value with minimal χ^2 for each run and the error bars show the 1σ (solid color) and 2σ (light color) uncertainties. The shaded area extends the uncertainties of the DYnamics, Age and Metallicity Indicators Tracing Evolution model for comparison. Set 161 performs the best, its minimum is within 1σ of the DYnamics, Age and Metallicity Indicators Tracing Evolution model. The same is valid for set 80, however here the error bars are larger. For sets 40 and 20, their minima and error bar sizes are very similar and they both are within 2σ of the DYnamics, Age and Metallicity Indicators Tracing Evolution model. As in the previous figures, set 10 gives the worst results: the error bars are the largest and the minimum the highest compared to the other sets. We notice a general upwards trend in the position of the minima. This results from the chi-squared curves getting flatter with smaller set sizes, as seen in Figure 6.2. With a flatter curve, the minimum is shifted to the right, i.e. towards higher M/L values. Up to set 20, all minima are within 2σ of the DYnamics, Age and Metallicity Indicators Tracing Evolution model, which means that the predictions perform well enough to be comparable to the model.

7. Analysis

In the previous chapters, we discussed different runs of predicting orbits while varying multiple parameters. We tried two different types of neural networks, used multiple loss functions and reduced the size of the input data to test the limits of our method. With this, we can define conditions where the network shows the 'best' performance. We define 'best results' by a small computing time and small loss during the training, and physically meaningful predictions, tested with the $\Delta\chi^2$-plot. With these results, we create kinematic maps from the predictions for a final comparison to the observational data.

7.1. Best results

7.1.1. Network architecture

We implemented and trained the DCNN to make predictions of the Gauß-Hermite coefficients. In Chapter 5, we tested the use of a GAN, but the results were not very reproducible, which could perhaps be improved in the future with more tuning. Once we added the mask into the DCNN, the predictions improved significantly. In terms of computing time, training a neural net is always time-consuming, and both network types were similar in training time. The implementation of the deconvolutional net is relatively simple for a neural net, and the results are reliable and reproducible, which is why it has been a good choice for our project.

7.1.2. Loss Function

The choice of a suitable loss function is important, as the type of loss can significantly change the predictions made with a neural net. In Chapter 4, we tested four different losses and found that $L_{1/\mathrm{err}}$ performed best in the comparison of the $\Delta\chi^2$-curves. The L_1-loss is stable and its $\Delta\chi^2$-curve is the closest to the DYNAMITE curve. Introducing the observational error adds physical meaning to the loss, as the predictions only need to be as good as the observational error. We therefore choose the L_1-loss divided by the observational error for all of the predictions.

7.1.3. Input data minimum

The computation time of the steps depends mostly on the size of the input data. To minimize it, we decide on the minimal size that the input data can have while still giving good results. This minimum is limited by the performance in the $\Delta\chi^2$-test but also by the coverage of the M/L range. The data set we created using 1/4 of the full set performs well when we look at the $\Delta\chi^2$-curve. The minimum we found with this method agrees within 2σ with our DYNAMITE models. This condition is also met if we use 20 models, i.e. 1/8 of the data. However, with a set size this small, we only cover the mass-to-light range in steps of 0.4. Around the M/L-minimum, this step size is very large. If we use 40 models instead, this step reduces to 0.2, which gives better coverage of the M/L-range. We therefore choose data set 40 as our input data minimum. It gives good results while being computationally less expensive than the full set.

7.2. Final calculations: Kinematic maps

In a final step, we compare our predictions to the observations. With DYNAMITE, we use the predictions as if they were results from Schwarzschild orbit modeling. DYNAMITE calculates weights for the the predicted Gauß-Hermite coefficients to create a model LOSVD. With this LOSVD, we create maps of the mean velocity v, the velocity dispersion σ and the coefficients h_3 and h_4. We can now compare these kinematic maps to maps from the observations and the DYNAMITE model.

7.2.1. Observations vs. Model

In a first step, we compare the observations to the orbit superposition models from DYNAMITE. The predictions are based on the models, which means that they can only be as good as them. The kinematic maps of the observations, the model and their residuals are shown in Figure 7.1. The implementation of Schwarzschild orbit modeling reproduces kinematic maps very close to the observations. For v and σ, the residuals have absolute values of a few (up to 10) times the observational error. This can be improved by running more models with DYNAMITE, but the observations are still reproduced well enough to be used for predictions. Unfortunately, the observations do not have any data on h_3 and h_4. The orbit modeling method tries to predict these values and creates maps, even though they have large uncertainties. The model shows some differences from the observations, but it still reproduces them well enough. We will now have a look at how the predictions compare to the observations.

7.2.2. Observations vs. Predictions

We now compare kinematics maps from our predictions to the observations. Figure 7.2 shows these maps and their differences in residual maps. The predictions are close to the observations, but they also show some differences. There are additional calculations involved in getting the kinematic maps from the GH-coefficients, so small differences are to be expected. Again, the comparison of h_3 and h_4 is not really possible because we don't have the observational data. Along the vertical axis of the maps, which is the minor axis of the galaxy, the residuals are the smallest. The v-maps indicate a rotation around the minor axis. The mean velocity along this axis is smaller than the velocity along the major axis, where the rotation is high. So, if the velocity is small along the axis, the models and predictions show small values there and the residuals will be proportionally smaller. The residuals are overall very similar to the ones from the models. To get an impression of how good the maps generated from predictions are, we compare them directly to the DYNAMITE maps.

7.2.3. Models vs. Predictions

From the previous comparisons, we learned that the predictions are very similar to the models, although we still find some differences. The residual maps of the two (Model - Prediction) are shown in Figure 7.3. These maps allow us to see how small the differences between the model and our predictions are. The maps for v and σ have very small values. The residual map of v shows a pattern where positive and negative regions alternate. The map in Figure 7.1 also shows an alternating pattern, which the prediction copies. If we take the difference between the models and the predictions, the subtraction of such a pattern creates a more complex alternation. Compared to the residuals from Figure 7.1, this structure is very faint and simply a result from

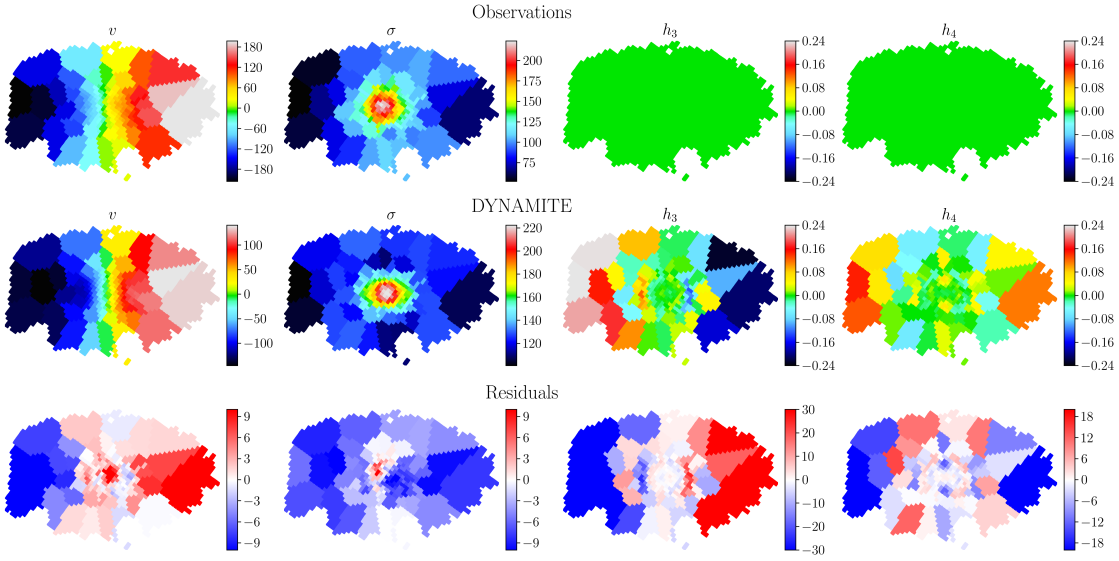


Figure 7.1.: Comparison of the velocity maps. These maps show how close the maps from orbit superposition modeling are to the observations. The residuals of the observations and the DYNAMITE maps are plotted on the bottom, scaled to the observational error (it has a value of 1 on the colorbar). The residual maps for v and σ are both in the range of ± 10 times the observational error. The residual in v is the highest (in absolute values) in the midplane of the map, where the observations show very high values too. The residual for σ has a bias towards negative value, meaning that the modeled σ values are overall higher than the observed ones. The observations don't show any values for h_3 and h_4, so the residuals are the same as the DYNAMITE values, only with opposite sign. The maps show that Schwarzschild orbit superposition produces results that are close to the observations, with residuals in the range of the observational error.

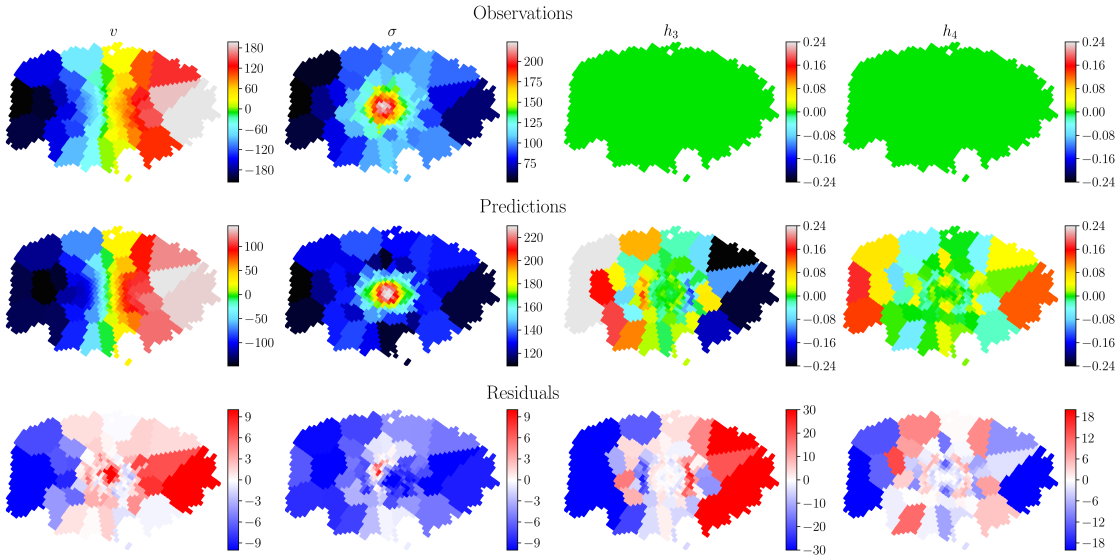


Figure 7.2.: Velocity maps of the DCNN predictions in comparison to the observations. The predictions show similar differences from the observations as the DYNAMITE model does. The residuals for v are overall small, but where v has large values in the observations, the residuals are large too. The residual maps for σ show a tendency to negative values, which indicates that the predicted σ-values are too high. Again, h_3 and h_4 have no observational maps here, so we need to compare the predictions to the DYNAMITE model for a more detailed look.

the inaccuracy of our models. While the residuals are small, we notice that the predictions have a tendency to 'overpredict' the values. We notice this especially in the residual map for σ, where all values are negative. The maps from predictions have overall higher values than the maps from the Schwarzschild model. They could be improved by calculating more models in the first place and by using more data in the predictions. Given that these maps are made with predictions from 40 models, even this limited amount of data produces satisfying results. The differences between the model and predictions are much smaller than the differences from the observations. This supports the idea that it is possible to use the results from our predictions instead of DYNAMITE models.

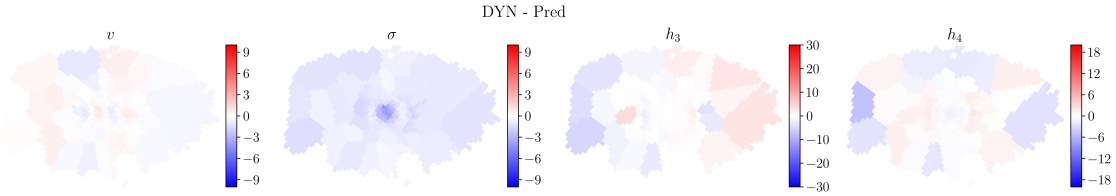


Figure 7.3.: Residual velocity maps of the model and the predictions, scaled to the observational error. Using the same scale as above (Figures 7.1 and 7.2), they all have values around 0. The residuals are close to the observational error, but there are still some nuances. The residual of v has very small values, indicating that our predictions are very close to the DYNAMITE model. The residual map of σ has less features. It has only negative values, indicating that the predictions are larger than the model. In the map center, this difference has its maximum. The residual values in the h_3 map cover the largest range. Negative values can be found more on the left side of the map, positive values are on the right side. This is similar in the residuals from the observations, indicating that, like for σ, the predictions have larger values than the models. The map for h_4 does not show a strong pattern; the positive and negative values are distributed more evenly on the map. Overall, the differences between the predictions and the model are very small compared to the other residual maps. The largest differences can be seen in h_3, which has values slightly larger than the observational error.

8. Conclusions

At the beginning of this project, we wanted to answer three main questions: Can we predict galactic orbits with deep learning? Is it possible to save computing time and resources by doing this? And can we use less data while still getting good results? Now, we can finally answer these questions.

What did we do? In this work, we wanted to know if we could use Deep Learning to predict galactic orbits and if the predictions could replace physical models. With Schwarzschild orbit modeling, we generated orbit libraries for the galaxy NGC 6278. We extracted Gauß-Hermite coefficients from the models and created image data sets. A Deconvolutional Neural Net was trained on this data to make predictions of Gauß-Hermite coefficients based on a single model parameter: the mass-to-light ratio. We then used the χ^2-test to find the best M/L-value for the models and the predictions. The first predictions were very close to the models, but we tested how small we could go with the input data set, and managed to reduce the training set significantly. We showed that the predictions could be used for further calculations, and generated kinematic maps that we compared to the observations and the DYNAMITE models. Our results are very close to the Schwarzschild models and can be used in the same way for further calculations.

Faster Schwarzschild orbit modeling? We managed to reduce the amount of models necessary to train the network, but did we speed up the process? In our setup, we introduce an additional step: training the neural net. But once the net is trained, we can skip the first step of running orbit models. We are even able to use only 25% of the initial training data to achieve similar results, as the network predicts the full parameter range by interpolating between the trained points. Simply put, we use less data but introduce a time-consuming step. This does not make a large difference in our process, but we only use very few models. In a real run of DYNAMITE, several thousands of models would be run. In this case, even the smallest improvement in time would make a difference. The possibility to predict the full parameter range from a relatively small training set is very useful in saving computational resources. But is it possible to implement DL into DYNAMITE? For our predictions, we used the option where orbit models are generated on a grid of parameters: the M/L-value in our case. This grid was necessary for the interpolation between the parameter values, especially for the reduced training sets. In a 'real' run, this option is typically not used. Additionally, the network would have to be trained separately for each galaxy to generate a fitting orbit model. In our simplified case, we achieved good results with a reduced input set. However, our approach cannot be directly applied to a full DYNAMITE run with several variable parameters.

What are the limits of the use of DL here? A tool like Deep Learning can speed up a process once the network is trained, but it also presents some difficulties. Computational resources are a limiting factor. Deep Learning usually needs large training sets and the training itself can take a lot of time. In our case, all steps worked on a normal laptop, but this would not be possible for larger amounts data. Ideally, a GPU should be used for the training, but this is not always

available. For larger data sets, it would be useful to go to a more powerful computing facility. The implementation of a neural net can be quite complex too. We limited the models to one parameter, but usually up to 8 parameters are varied with DYNAMITE. A neural net that depends on all of these would increase in complexity. As we tried to show here, it can be worth it to overcome the difficulties, because the number of necessary Schwarzschild models can be significantly reduced.

Future prospects We tested if we could predict orbits on a very small scale and with some simplifications, which leaves many possibilities to expand on this method. The first simplification was the choice of the mass-to-light ratio as single variable parameter in the models. One possibility to go further would be to test other parameters like the axis ratios of the galaxy, the black hole or the dark matter components. The choice of a single parameter also limited the amount of models that could be made with DYNAMITE. With multiple variable parameters, more models can be made which leads to larger training sets for a neural net. Of course, this would require more computational power and a lot more time. We did our calculations with CALIFA data on the galaxy NGC 6278, for the reason that it is one of the DYNAMITE test galaxies. Another possibility would be to try it on a different galaxy, ideally with higher resolution data. This would increase the size of the data for the models and the training, and the network would have to be adapted to the new data. If, in the end, we would want to speed up the process of Schwarzschild orbit modeling with Deep Learning, all of the steps above would have to be done first. We tested if it is possible to predict kinematics, and this idea can be explored further in many ways.

Can we predict galactic orbits with deep learning? Once we find a suitable network and adapt it to our data, we can use even little input data to predict the kinematics in our galaxy. One of the motivations of this project was to save computation time and resources when running orbit modeling. We introduced additional time-consuming steps into the process, which slows it down. But once the training is done, generating new kinematics with predictions is very fast. To really see how much time can be saved, more studies would need to be made, but we successfully tested the concept. Another goal was to reduce the amount of data necessary for Schwarzschild orbit models. This was quite successful, as the predictions allowed to train on a subset of the data, then to interpolate the full parameter range, and the results were as accurate as the original orbit library. We successfully tested the concept of using Deep Learning to predict galactic orbits. During the project, there were many occasions where we had to simplify and scale down the tasks. Each of the simplifications can be a starting point for more studies. Especially the concept of *predicting kinematics* can be used and expanded in the future. In a simplified way, we sum this work up as follows: A deep neural net can learn the kinematics of galaxies.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bender, R. (1990). Unraveling the kinematics of early-type galaxies. Presentation of a new method and its application to NGC 4621. *A&A*, 229:441–451.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations*, 2:1–55.

Binney, J. and Merrifield, M. (1999). *Galactic Astronomy*. Princeton University Press, Princeton.

Buta, R. J. (2019). The systematics of galaxy morphology in the comprehensive de Vaucouleurs revised Hubble-Sandage classification system: application to the EFIGI sample. *MNRAS*, 488(1):590–608.

Cappellari, M. (2002). Efficient multi-Gaussian expansion of galaxies. *MNRAS*, 333(2):400–410.

Cappellari, M., Emsellem, E., Krajnović, D., McDermid, R. M., Scott, N., Verdoes Kleijn, G. A., Young, L. M., Alatalo, K., Bacon, R., Blitz, L., Bois, M., Bournaud, F., Bureau, M., Davies, R. L., Davis, T. A., de Zeeuw, P. T., Duc, P.-A., Khochfar, S., Kuntschner, H., Lablanche, P.-Y., Morganti, R., Naab, T., Oosterloo, T., Sarzi, M., Serra, P., and Weijmans, A.-M. (2011). The ATLAS^{3D} project - I. A volume-limited sample of 260 nearby early-type galaxies: science goals and selection criteria. *MNRAS*, 413(2):813–836.

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

Carrasco Kind, M. and Brunner, R. J. (2013). TPZ: photometric redshift PDFs and ancillary information by using prediction trees and random forests. *MNRAS*, 432(2):1483–1501.

de Zeeuw, T. (1985). Elliptical galaxies with separable potentials. *MNRAS*, 216:273–334.

den Brok, M., Krajnović, D., Emsellem, E., Brinchmann, J., and Maseda, M. (2021). Dynamical modelling of the twisted galaxy PGC 046832. *MNRAS*, 508(4):4786–4805.

Emsellem, E., Monnet, G., and Bacon, R. (1994). The multi-gaussian expansion method: a tool for building realistic photometric and kinematical models of stellar systems I. The formalism. *A&A*, 285:723–738.

Faber, S. M. and Gallagher, J. S. (1979). Masses and mass-to-light ratios of galaxies. *ARA&A*, 17:135–187.

Gerhard, O. E. (1993). Line-of-sight velocity profiles in spherical galaxies: breaking the degeneracy between anisotropy and mass. *MNRAS*, 265:213.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

Jethwa, P., Thater, S., Maindl, T., and Van de Ven, G. (2020). DYNAMITE: DYnamics, Age and Metallicity Indicators Tracing Evolution.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Loubser, S. I., Lagos, P., Babul, A., O'Sullivan, E., Jung, S. L., Olivares, V., and Kolokythas, K. (2022). Merger histories of brightest group galaxies from MUSE stellar kinematics. *MNRAS*, 515(1):1104–1121.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets.

Morganti, L., Gerhard, O., Coccato, L., Martinez-Valpuesta, I., and Arnaboldi, M. (2013). Elliptical galaxies with rapidly decreasing velocity dispersion profiles: NMAGIC models and dark halo parameter estimates for NGC 4494. *MNRAS*, 431(4):3570–3588.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. `http://neuralnetworksanddeeplearning.com/`.

Parul, H. D., Smirnov, A. A., and Sotnikova, N. Y. (2020). Orbital Ingredients for Cooking X-structures in Edge-on Galaxies. *ApJ*, 895(1):12.

Pinna, F., Falcón-Barroso, J., Martig, M., Coccato, L., Corsini, E. M., de Zeeuw, P. T., Gadotti, D. A., Iodice, E., Leaman, R., Lyubenova, M., Martín-Navarro, I., Morelli, L., Sarzi, M., van de Ven, G., Viaene, S., and McDermid, R. M. (2019). The Fornax 3D project: Thick disks in a cluster environment. *A&A*, 625:A95.

Sánchez, S. F., García-Benito, R., Zibetti, S., Walcher, C. J., Husemann, B., Mendoza, M. A., Galbany, L., Falcón-Barroso, J., Mast, D., Aceituno, J., Aguerri, J. A. L., Alves, J., Amorim, A. L., Ascasibar, Y., Barrado-Navascues, D., Barrera-Ballesteros, J., Bekeraitè, S., Bland-Hawthorn, J., Cano Díaz, M., Cid Fernandes, R., Cavichia, O., Cortijo, C., Dannerbauer, H., Demleitner, M., Díaz, A., Dettmar, R. J., de Lorenzo-Cáceres, A., del Olmo, A., Galazzi, A., García-Lorenzo, B., Gil de Paz, A., González Delgado, R., Holmes, L., Iglésias-Páramo, J., Kehrig, C., Kelz, A., Kennicutt, R. C., Kleemann, B., Lacerda, E. A. D., López Fernández, R., López Sánchez, A. R., Lyubenova, M., Marino, R., Márquez, I., Mendez-Abreu, J., Mollá, M., Monreal-Ibero, A., Ortega Minakata, R., Torres-Papaqui, J. P., Pérez, E., Rosales-Ortega, F. F., Roth, M. M., Sánchez-Blázquez, P., Schilling, U., Spekkens, K., Vale Asari, N., van den Bosch, R. C. E., van de Ven, G., Vilchez, J. M., Wild, V., Wisotzki, L., Yıldırım, A., and Ziegler, B. (2016). CALIFA, the Calar Alto Legacy Integral Field Area survey. IV. Third public data release. *A&A*, 594:A36.

Santucci, G., Brough, S., van de Sande, J., McDermid, R. M., van de Ven, G., Zhu, L., D'Eugenio, F., Bland-Hawthorn, J., Barsanti, S., Bryant, J. J., Croom, S. M., Davies, R. L., Green, A. W., Lawrence, J. S., Lorente, N. P. F., Owers, M. S., Poci, A., Richards, S. N., Thater, S., and Yi, S. (2022). The SAMI Galaxy Survey: The Internal Orbital Structure and Mass Distribution of Passive Galaxies from Triaxial Orbit-superposition Schwarzschild Models. *ApJ*, 930(2):153.

Schwarzschild, M. (1979). A numerical model for a triaxial stellar system in dynamical equilibrium. *ApJ*, 232:236–247.

Shallue, C. J. and Vanderburg, A. (2018). Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90. *AJ*, 155(2):94.

Statler, T. S. (1987). Self-consistent Models of Perfect Triaxial Galaxies. *ApJ*, 321:113.

Thomas, J., Saglia, R. P., Bender, R., Thomas, D., Gebhardt, K., Magorrian, J., and Richstone, D. (2004). Mapping stationary axisymmetric phase-space distribution functions by orbit libraries. *MNRAS*, 353(2):391–404.

Trujillo-Gomez, S., Kruijssen, J. M. D., Pfeffer, J., Reina-Campos, M., Crain, R. A., Bastian, N., and Cabrera-Ziri, I. (2023). In-situ or accreted? Using deep learning to infer the origin of extragalactic globular clusters from observables. *arXiv e-prints*, page arXiv:2301.05716.

van den Bosch, R. C. E., van de Ven, G., Verolme, E. K., Cappellari, M., and de Zeeuw, P. T. (2008). Triaxial orbit based galaxy models with an application to the (apparent) decoupled core galaxy NGC 4365. *MNRAS*, 385(2):647–666.

van der Marel, R. P. and Franx, M. (1993). A New Method for the Identification of Non-Gaussian Line Profiles in Elliptical Galaxies. *ApJ*, 407:525.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.

Zhu, L., van den Bosch, R., van de Ven, G., Lyubenova, M., Falcón-Barroso, J., Meidt, S. E., Martig, M., Shen, J., Li, Z.-Y., Yildirim, A., Walcher, C. J., and Sanchez, S. F. (2018). Orbital decomposition of CALIFA spiral galaxies. *MNRAS*, 473(3):3000–3018.

Acronyms

CALIFA Calar Alto Legacy Integral Field Area Survey. 9, 32, 44

CNN Convolutional Neural Net. 7, 15

DCNN Deconvolutional Neural Net. vii, 7, 15, 16, 21, 24, 26, 29, 37, 40, 43

DL Deep Learning. 6, 43, 44

DYNAMITE DYnamics, Age and Metallicity Indicators Tracing Evolution. 5, 9–13, 21–24, 28, 29, 32–35, 37–41, 43, 44

GAN Generative Adversarial Network. 7, 16, 25–29

GH Gauß-Hermite. 2, 3, 10–13, 15, 18, 21–23, 25, 27, 31–33, 37, 38, 43

GPU Graphics Processing Unit. 43

IFS Integral Field Spectroscopy. 2

LOSVD Line-Of-Sight Velocity Distribution. 2–4, 9, 11

MGE Multi-Gaussian Expansion. 4, 5, 9

ML Machine Learning. 5, 6

MNIST Modified National Institute of Standards and Technology database. 6, 7, 17

NGC New General Catalogue of Nebulae and Clusters of Stars. 9, 21

ReLU Rectified Linear Unit. 7

SB surface brightness. 4

A. Additional Figures

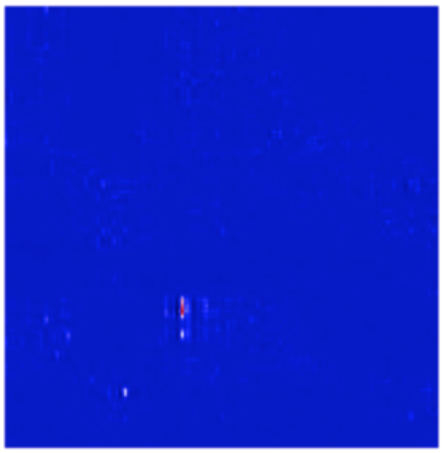


Figure A.1.: Prediction of h_4 from the GAN, trained without the mask. The output is mostly constant. We can distinguish the basic structure of the image, but it is very noisy and faint.

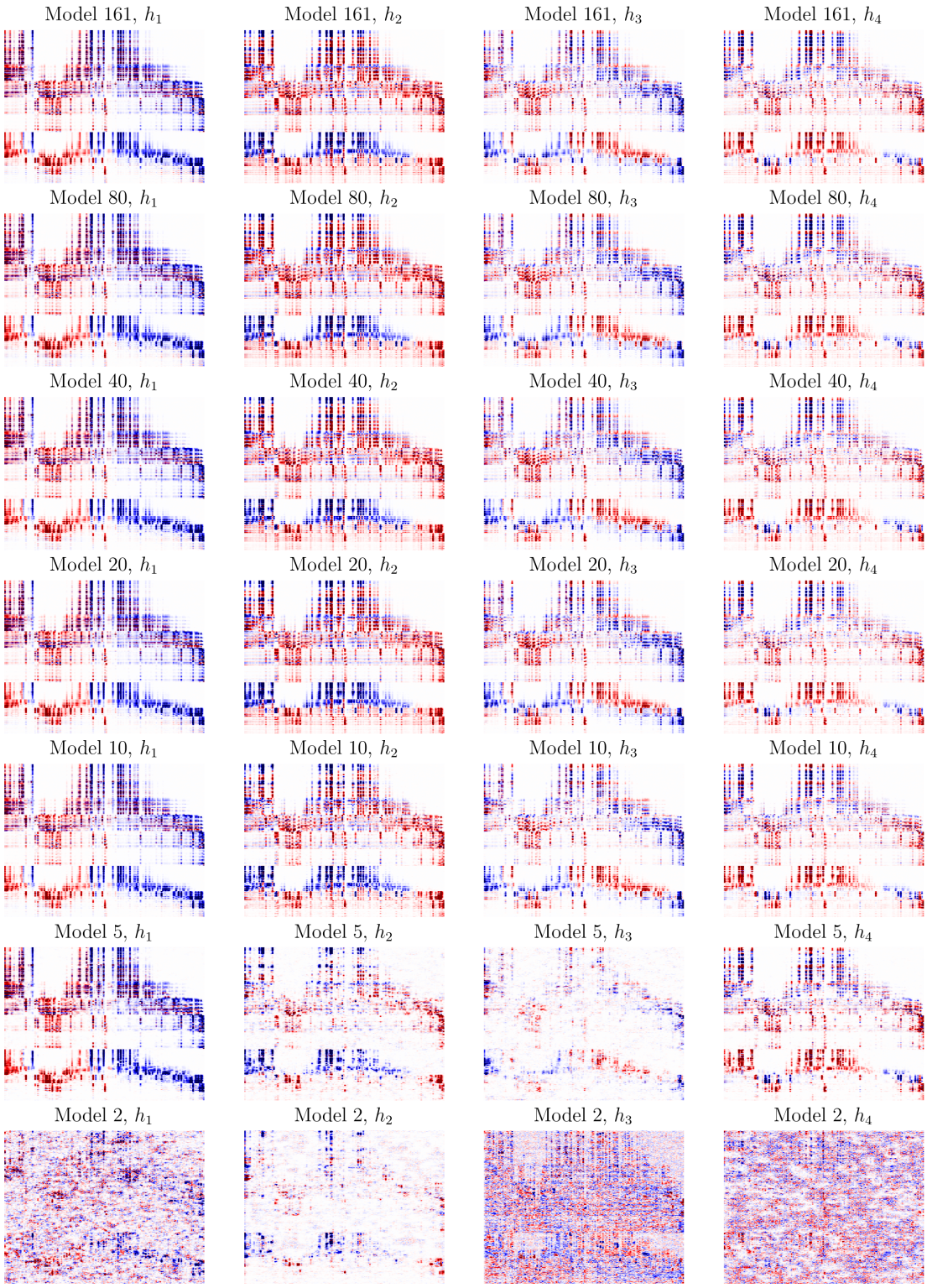


Figure A.2.: Evolution of the predictions with smaller training set sizes, showing all predictions for all sets.

B. Additional Tables

Layer (type)	Output Shape	Number of Params
InputLayer	(None, 1)	0
Dense	(None, 2)	4
LeakyReLU	(None, 2)	0
Dense	(None, 6)	18
LeakyReLU	(None, 6)	0
Dense	(None, 72960)	510720
LeakyReLU	(None, 72960)	0
Reshape	(None, 60, 38, 32)	0
UpSampling2D	(None, 120, 76, 32)	0
Conv2DTranspose	(None, 120, 76, 48)	6192
Reshape	(None, 120, 76, 48)	0
UpSampling2D	(None, 360, 152, 48)	0
Conv2DTranspose	(None, 360, 152, 48)	13872
Reshape	(None, 360, 152, 48)	0
Dense	(None, 2)	4
Dense	(None, 360, 152, 1)	49
Dense	(None, 54720)	164160
Reshape	(None, 360, 152)	0
Reshape	(None, 360, 152)	0
Multiply	(None, 360, 152)	0

Table B.1.: Structure and layers of the deconvolutional neural net.

Quantity	Symbol
Kinematics	
Mean velocity	v
Velocity dispersion	σ
Gauß-Hermite coefficient	h_i
Hermite polynomials	H_i
Surface Brightness	SB
Luminosity dispersion	σ_L
Stellar axis ratios	p, q
Projected coordinates	x, y
Luminosity	L
Observational error	ϵ
Machine Learning	
Bias	b
Weight(s)	w
Activation function	s
Input data	x_i
Input labels	y_i
Loss	L
DYNAMITE	
black hole mass	m-bh
black hole radius (half-mass)	a-bh
dark matter halo concentration	c-dh
dark-matter fraction M_{200}/M_*	f-dh
intrinsic flattening (c/a)	q-stars
intrinsic flattening (b/a)	p-stars
$\sigma_{\mathrm{obs}}/\sigma_{\mathrm{intrinsic}}$	u-stars
mass-to-light ratio	M/L
chi-squared	χ^2
Delta-chi-squared	$\Delta\chi^2$

Table B.2.: Symbols and Parameters