Universidade de São Paulo
Instituto de Astronomia, Geofísica e Ciências Atmosféricas
Departamento de Astronomia

Roberta Duarte

Black Hole Weather Forecasting with Deep Learning

São Paulo
2020

Roberta Duarte

Black Hole Weather Forecasting with Deep Learning

Dissertation presented to the Astronomy department of the Institute of Astronomy, Geophysics and Atmospheric Sciences of the University of São Paulo as partial requisite to obtain the Master in Sciences title.

Concentration Area: Astronomy
Advisor: Prof. Dr. Rodrigo Nemmen
Versão Corrigida. O original encontra-se disponvel na Unidade.

São Paulo
2020

To my parents, Roberto e Maria

Acknowledgements

I had to start a new life to start this project since I moved from São Carlos to São Paulo. Since this was a significant impact in my life, I had support from some people that helped me and stayed by my side. This project is the result of hard work, but it also has a behind the scenes history. Here, I want to acknowledge and thank the important people.

First, I want to thank the Black Hole Group. My advisor Rodrigo Nemmen and all the members, Fabio, Gustavo, Ivan, Artur, Raniere, and Roderik. They trusted me, supported me, and gave me fantastic advice during my research. The collaborator of this project, João Paulo, was also a big part of this, he supported this project not only with hardware but also with advice. So thank you.

I want to thank my parents, Roberto and Maria, who supported me in every way possible, and my dogs, Mila and Kika, that are part of my family. I had moral, mental, and financial support from them.

I also want to give a huge thanks to Fabio Cafardo since he became my best friend in the last two years. He was my first friend in São Paulo. Because of him, I felt at home again. This work was possible also thanks to Erik, Mirian, Rita, Julia, André F., Geisa, Humberto, and Laerte since they helped me when my mental health wasn't in the right place and all of them became my dearest friends. I can't thank them enough because they were my most significant support. This fantastic journey was also due to Amanda, Geraldo, Catarina, Danielle, Ana Júlia, André V., Fernanda, Stephane, Rafael, Loreany, Thayse, so thank you all of you.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of Quadro GP100 and Quadro P6000 GPUs used for this research.

This work was supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de

Nível Superior) through PROEX (Programa de Excelência Acadêmica) program. A huge thanks to CAPES because without its support this work wouldn't be possible.

"*I'm falling, so I'm taking my time on my ride*"

Tyler Joseph

"*It seems like the pressure that could have crushed us made us into diamonds instead. And what didn't kill us actually did make us stronger.*"

Taylor Swift accepting Woman of the Decade Award

Resumo

Métodos tradicionais de estudar o comportamento de um disco de acreção ao redor de um buraco negro são compostos principalmente de simulações numéricas computacionalmente caras. Esse custo faz com que as simulações numéricas sejam restringidas por dimensionalidade ou limitações nas equações e, geralmente, leva muito tempo para simular. A física de buracos negros precisa urgentemente de uma nova ferramenta capaz de obter resultados mais rápidos. Queremos propor o uso do aprendizado profundo como uma possível nova ferramenta. O objetivo é desenvolver um método de aprendizado profundo capaz de fazer previsões de estados ao redor de buracos negros. Propomos um modelo que pode reproduzir os resultados de uma simulação com um erro abaixo de $< 3\%$ e ao mesmo tempo acelerar o processo de obtenção dos resultados por um fator de $10^{4.5}$.

Abstract

Traditional methods of studying accretion flows onto black holes mainly consist of computationally expensive numerical simulations. This often imposes severe limitations to the dimensionality, simulation times, and resolution. Computational astrophysics is in urgent need of new tools to accelerate the calculations, thereby leading to faster results. We propose a deep learning method to make black hole "weather forecasting": a data-driven approach for solving the chaotic dynamics of BH accretion flows. Our model can reproduce the results of a hydrodynamic simulation with an error $< 3\%$ and at the same time speeding-up the calculations by a factor of $10^{4.5}$, thus reducing the simulation time.

List of Figures

1.1 The scheme shows the machine learning fields. 28
1.2 A deep neural network architecture: each node is called a neuron. They are composed of the input layer, the hidden layers, and output layer. 29
1.3 Convolutional neural network architectures are composed of the input block, the hidden blocks, and the output block. Filters make the convolutional operations. 30
1.4 Figure extracted from Pathak et al. (2018) 32

2.1 A flowchart of how the data flow from the numerical simulations until the training part. 35
2.2 The plots from Almeida and Nemmen (2020) show the different angular momentum profiles. 37
2.3 The grid of the simulations. 38
2.4 Examples of the density plots from the simulations PL0SS3, PNSS3, and PNST1. 40
2.5 Plots of $\log(\rho)$ from PNSS3 before the crop. 41
2.6 The block is a 3D array with spatial and temporal information. 42
2.7 The input is a block with size $(bs, 256, 192, t_n : t_{n+4})$ and the output is the block with size with $(1, 256, 192, t_{n+5} : t_{n+10})$. 43

3.1 We show a shallow NN and a DNN. 45
3.2 We show a scheme with three neurons. Two neurons in the layer l and one neuron in the layer $l + 1$. 47
3.3 The function $\mathcal{C}(w)$ in relation to w. The green curve is the curve of $\mathcal{C}(w)$. . 49

3.4 A CNN architecture composed of convolutions. 52
3.5 The picture is a $(H \times W)$ matrix and we can separate the 3 channels RGB in 3 matrices. 52
3.6 Example of how a MaxPooling layer acts. 53
3.7 Example of how an UpSampling layer acts. 54
3.8 Illustration from Podareanu et al. (2019). 54
3.9 The shape of the input and the output of a convolution. 55
3.10 The scheme of the architecture U-Net based on Ronneberger et al. (2015). 57
3.11 The inference scheme with an iteration loop. 60
3.12 The inference scheme directly from simulation. 61
3.13 The pipeline of the cGAN model. 62

4.1 The regions of the loss function. 66
4.2 Two kinds of forecasting: direct and iterative. 66
4.3 The prediction of the gravitational time – $t = 611529\ GM/c^3$ – compared with the respective target. 67
4.4 The MAPE metric between the prediction respective to $t = 611529\ GM/c^3$ and other frames from the dataset of the same simulation. 68
4.5 The mean density as a function of Θ and the function of R. 68
4.6 The left panel shows the mean absolute percentage error (MAPE) between the prediction and the ground truth. The right panel shows the difference of the logarithm of the ground truth and prediction ($\Delta log(\rho)$). 69
4.7 The MAPE between the prediction and its respective ground truth. 69
4.8 The comparison between the mass in the target and the mass in the prediction. 70
4.9 The density state for three distinct times. 71
4.10 The density state for three distinct times. 72
4.11 The mass in two regions: $R < 50R_s$ and $R > 50R_s$. 72
4.12 The MAE between the predictions and the respective ground truth. 73
4.13 This is the 10th step after the training stop for all simulations. 75
4.14 The continuation of 4.13 plot. 76
4.15 The differences for 10, 25 and 50 frames in the future for PL0SS3 and PNST1. 77
4.16 The analysis of PL0SS3 and PNST1. 77

4.17 The predictions of the PL0SS3 using the iterative scheme. 78
4.18 The mass of the target vs the mass of the predictions. 79
4.19 The mean of density in θ (left panel) and R (right panel) in the function of the time. 79
4.20 The results are from simulations PL0ST1, PL2SS3, PNSS3, PNST01, and PNST1 from GAN model. 80
4.21 The $log(M)$ for PNSS3 and PNST1 predictions from the GAN. 81
4.22 The PNSS3 density mean in θ and R in the function of the time for the cGAN. 81
4.23 The PNST1 density mean in θ and R in the function of the time for the cGAN. 82
4.24 Duration of regular simulation and DL model. 83
4.25 Processing time comparison the simulations in simulation and in DL model. 84
4.26 Duration of all simulations and DL model. 84
4.27 Processing time the simulations in simulation and in DL model for all simulations. 85

List of Tables

2.1 The table shows all simulations performed by Almeida and Nemmen (2020). 38

4.1 The simulations with the parameters: angular momentum profile, viscosity profile, and the α-parameter. 73

4.2 The resources we use in our works are listed. 82

A.1 The split of the dataset. 105

A.2 Hyperparameters found by the grid search method. 105

A.3 Hyperparameters found by the grid search method. 106

Contents

1. Introduction 23

1.1 Accretion 25

1.2 Numerical Simulations 26

1.3 Deep Learning 27

1.4 Related Work 31

1.5 This Work 32

2. Astrophysical Dataset 34

2.1 Numerical Simulations 35

2.2 Data preparation 39

2.3 Training, Validation and Testing sets 42

3. Methods 44

3.1 Neural Networks 44

3.2 Convolutional Neural Networks 51

3.3 Architecture 57

3.4 Hyperparameters 58

3.5 Inference 60

3.6 GANs 61

3.7 Analysis 62

3.8 Procedure 64

4. Results 65

4.1 One simulation 66

4.2 All simulations 72
4.3 GANs 77
4.4 Speed-Up 81

5. Discussion 86
5.1 One simulation 86
5.2 All simulations 87
5.3 cGANs 88

6. Conclusions 90
6.1 Future Perspectives 92

Bibliography 94

Appendix 103

A. Details 105

List Of Abbreviations

- AGNs - Active Galaxy Nuclei
- BHs - Black Holes
- CNNs - Convolutional Neural Networks
- DL - Deep Learning
- GANs - Generative Adversarial Networks
- GRMHD - General Relativistic Magneto-Hydrodynamical
- HD - Hydrodynamical
- LLAGNs - Low-luminosity Active Galaxy Nuclei
- LSTM - long short-term memory
- MAE - mean absolute error
- MHD - Magneto-Hydrodynamical
- ML - Machine Learning
- MLP - Multi-Layer Perceptron
- RIAFs - Radiatively Inefficient Accretion Flows
- RNN - recurrent neural network
- SMAPE - symmetric mean absolute percentage error
- SMBHs - Supermassive Black Holes

Chapter 1

Introduction

The objects in the Universe with a large gravitational field, a singularity in its center, and surrounded by an event horizon are called black holes (BHs). They first were mathematical entities discovered by Karl Schwarzschild while solving Einstein equations for uncharged spherically-symmetric non-rotating configurations. Only in the early 60s, the first evidence of the existence of a BH was found by Maarten Schmidt when he first observed quasars (Schmidt, 1963). The term "black hole" to name those objects was introduced by Wheeler in 1964 (Misner, Thorne e Wheeler, 1973). In parallel, the theory behind BHs was turning into a hot topic in Physics: Kerr derived the solution of Einstein's equations for a rotating BH in 1963 (Kerr, 1963), Penrose proposed the Penrose-process as a way to extract energy from a rotating BH (Penrose, 1969) and later it was used as one of the processes to explain jets from rotating BHs (Blandford and Znajek, 1977). Besides, more evidence of BHs was found, e.g, Bolton (1972); Wagner et al. (1990); Schodel et al. (2002). Recently, gravitational waves generated by the merge of two BHs were detected (Abbott et al., 2016) as well as the first picture of a BH captured by the EHT (Event Horizon Telescope Collaboration, 2019). Observations and theoretical discoveries about BHs make it possible to understand better gravity itself and how those objects are related to their environment.

The parameters that describe a BH are mass, spin, and charge according to the no-hair theorem (Misner et al., 1973). Since the charge is not relevant in astrophysical BHs, they are classified depending on their spin, Schwarzschild BHs (non-spinning) and Kerr BHs (spinning). Also, they can be classified in terms of their mass as follows: BHs with a mass $M > 10^5 M_\odot$ are called supermassive black holes (SMBHs) - those are found in the center of galaxies (Ferrarese and Ford, 2005) and BHs with a mass $10 M_\odot < M < 10^3 M_\odot$

are called stellar BHs - found in X-ray binaries. Their mass relates to their size by the Schwarzschild radius, $R_s = 2GM/c^2$, where G is the gravitational constant, c is the light speed, and M is the mass. The Schwarzschild radius gives us the size of the event horizon.

When gas comes close to a BH, it creates a disk-like structure around the BH. Due to the conservation of angular momentum, the matter would rotate indefinitely around the event horizon unless it loses angular momentum. The dynamical friction due to magnetic fields (Balbus, 2003) transports momentum outwards in the disc. The angular momentum loss causes the matter to spiral inwards, adding mass to the BH thereby leading to accretion. The ensuing dynamical friction will convert part of the mechanical energy into light. This eletromagnetic emission is observed in X-ray binaries, gamma-ray bursts, and active galaxy nuclei (AGNs) (Meier, 2012; Gilfanov, 2010).

The accretion flow behavior will depend on how this energy is dissipated (Abramowicz and Fragile, 2013), which is in turn related to the accretion rate $\dot{M}$. For low values of $\dot{M}$, a small fraction of the energy is radiated, and the accretion disk starts to heat, becoming hot, geometrically thick, and optically thin, meaning a medium with low density leading to a radiatively inefficient accretion flow (RIAF) (Yuan and Narayan, 2014). Low luminosity AGNs (LLAGNs) are the result of SMBHs accreting in RIAF mode. SMBHs accreting at low $\dot{M}$ are dominant objects in the local Universe, including Sagittarius A* (Sgr A*) - the $4 \times 10^6 M_\odot$ SMBH in the center of the Milky Way (Yuan et al., 2003). It is indispensable to study those SMBHs to understand how they affect the environment around it and to understand our Local Group, including our Galaxy.

To understand accretion physics and to model the dynamics of the system, it is necessary to use numerical simulations. Numerical simulations of BH accretion started to make progress in the late 90s. The first simulations were hydrodynamical and Newtonian (Stone et al., 1999). Proga and Begelman (2003) introduced the pseudo-Newtonian potential but still without viscosity. Numerical simulations adding viscosity, cooling effects, and magnetic fields were published (Stone and Pringle, 2001; Turner et al., 2003; Proga and Begelman, 2003) increasing the complexity of those systems. Nowadays, general relativistic magneto-hydrodynamical (GRMHD) simulations are the primary numerical tools to understand accretion physics near BHs (McKinney and Narayan, 2007; McKinney and Gammie, 2004; McKinney et al., 2012) and to explain jet production from Kerr BHs (Tchekhovskoy and McKinney, 2012).

While numerical simulations have been the primary tool to investigate the accretion physics of BHs, they are computationally expensive and may take a long wall time to run. For example, a 2D hydrodynamical simulation with 400×200 takes seven days running in 400 CPU-cores. For more realistic descriptions, the models become even more demanding, i.e., they require more processors and longer times. This demand is significant when we go from 2D simulations to 3D simulations since we are adding another dimension - the curse of dimensionality (Bellman, 1966). New methods are desirable to accelerate the computations involved while maintaining a good trade-off between speed and precision.

Deep learning (DL) shows up as a promising field in Astronomy in recent years (Huertas-Company et al., 2019; Fabbro et al., 2017; Shallue and Vanderburg, 2018). The success of DL inside Astronomy is due to the massive availability of data (Siemiginowska et al., 2019). DL is already being used to analyze data (Nieto et al., 2019) and to extract parameters from observations (Ribli et al., 2018). Also, DL is speeding-up analysis in fields such as Cosmology when simulating those complex systems is computationally expensive. At the same time, DL methods are finding applications in other fields such as fluid mechanics (King et al., 2018; Mohan et al., 2019; Mohan et al., 2019) and weather forecasting (Gensler et al., 2016; Grover et al., 2015). The progress of DL in such fields is showing the capacity of deep neural networks to deal with complex and non-linear systems.

1.1 Accretion

The angular momentum transport responsible for making the inner parts of the accretion flow move towards the BH is due to magnetic stresses. When a magnetic line connects two elements of the fluids with two different rotational velocities, the line's stress leads to momentum transport. Since the inner element loses momentum, it releases energy, so these stresses are responsible for converting some of the gravitational potential energy to thermal energy. This phenomenon is called magneto-rotational instability (MRI) presented by Balbus (2003). A fraction of the thermal energy is radiated away. What remains heats up the gas.

The behavior of the accretion flow is dictated by the fraction ϵ of the rest-mass energy associated with accreted mass, $\dot{M}c^2$, which is converted to radiation. The ϵ in Equation 1.1 represents the radiative efficiency, where L is the luminosity, and $\dot{M}$ is the accretion

rate:

$$\epsilon = \frac{L}{\dot{M}c^2}. \tag{1.1}$$

$\dot{M}$ is usually expressed in terms of Eddington accretion rate $\dot{M}_{EDD}$ ($\dot{M}_{EDD}$ is the accretion rate if the luminosity is the Eddington luminosity). In this sense, $\dot{M}$ will be a fraction of $\dot{M}_{EDD}$, $\dot{M} = \lambda \dot{M}_{EDD}$. When $0.01 < \lambda < 1$, the disk is geometrically thin and optically thick, radiating like a black body, and attaining temperatures in the range $10^4 - 10^7 K$. If $\dot{M} \gtrsim \dot{M}_{EDD}$ ($\lambda \gtrsim 1$), the disk is optically thick, but in this case the radiation gets trapped and it is advected onto the BH. This model is called a slim disk. For cases where $\lambda < 0.01$, the disk becomes geometrically thick and optically thin, the energy is not radiated, so it heats the fluid to higher temperatures $T \sim 10^{12} K$ (Frank et al., 2002).

The interest in studying different values of $\dot{M}$ for SMBHs connects with the different types of AGNs. Thin disks are relevant for Seyfert 1s and quasars. RIAFs can model LLAGNs and most SMBHs in nearby galaxies. The closest example of a SMBH accreting in RIAF mode is Sagittarius A* (Sgr A*) (Beckman, 1993). Also, SMBHs accreting at low $\dot{M}$ may have an important role when it comes to feedback in the form of winds and jets. That feedback may explain the quenching star formation in the center of quiescent galaxies (Roy et al., 2018).

1.2 Numerical Simulations

Analytical solutions to the fluid equations that describe the accretion flow are rarely possible due to the complexity of those systems. The complexity of those systems can be due to turbulence driven by MRI. Numerical simulations have been the best tool to solve fluid dynamics and thereby advance our understanding of BH accretion flows.

The hydrodynamical (HD) regime is well established since a considerable number of papers has been published on this topic, e.g., Stone et al. (1999); Igumenshchev et al. (2000); Fragile and Anninos (2005); De Villiers and Hawley (2002); Almeida and Nemmen (2020). In HD simulations some viscosity acts as the cause of stress and angular momentum transport.

The next step is naturally the addition of magnetic fields in the numerical simulations. Magnetic fields have an essential role in removing angular momentum via the MRI as

explained before (Balbus and Hawley, 1991; Balbus, 2003). The impact of different magnetic fields configurations on the accretion flow and outflows is an active area of numerical astrophysics.

BHs are general relativistic objects, so to explain how those objects affect its immediate environment, it is essential to use general relativity. GRMHD simulations are especially relevant for investigating the near horizon physics (Porth et al., 2019) and the production of relativistic jets (Moscibrodzka, M. et al., 2016).

To properly study the region close to a BH from a realistic point of view, GRMHD is required. When it comes to studying winds from an accretion flow far away from the BH, it is also useful to use of HD simulations with pseudo-Newtonian potential. In this work, we use simulations from Almeida and Nemmen (2020) who performed HD simulations of large tori in order to investigate winds and feedbacks in LLAGNs.

Numerical simulations have some shortcomings. One of them is the large amount of time needed to numerically solve the equations. The longest hydrodynamical RIAF simulation run by Almeida and Nemmen (2020) required $\sim$ 7 days running in 200 CPU cores. This motivates the investigation of alternatives to explicit solvers of the underlying conservation equations. Here, we address the use of deep learning techniques, which could dramatically accelerate (M)HD numerical simulations while keeping a reasonable accuracy.

1.3 Deep Learning

Machine learning (ML) is a field that uses data to create a mathematical model able to fit new data. The ML models are dynamical and can adapt when they are fed with data. There are essentially two types of ML models: *supervised* and *unsupervised* learning. Supervised learning is when the model has access to information that we already know about the data. More precisely, we give the model an input (e.g., a picture), and we also give the outputs (e.g., the classification of the picture as a cat). The goal in supervised learning is that the model learns from data and then applies this knowledge in new unseen data. On the other hand, unsupervised learning is when we give just the inputs or the features to the model, and the model has to figure on its own the dependencies and underlying structures in the data. Reinforcement learning has also been included under the umbrella of ML. Reinforcement learning uses input and outputs to learn as a supervised

learning method but also uses new tactics to perform the goal (Salian, 2018). In Figure 1.1, we show the scheme representing ML and its fields.

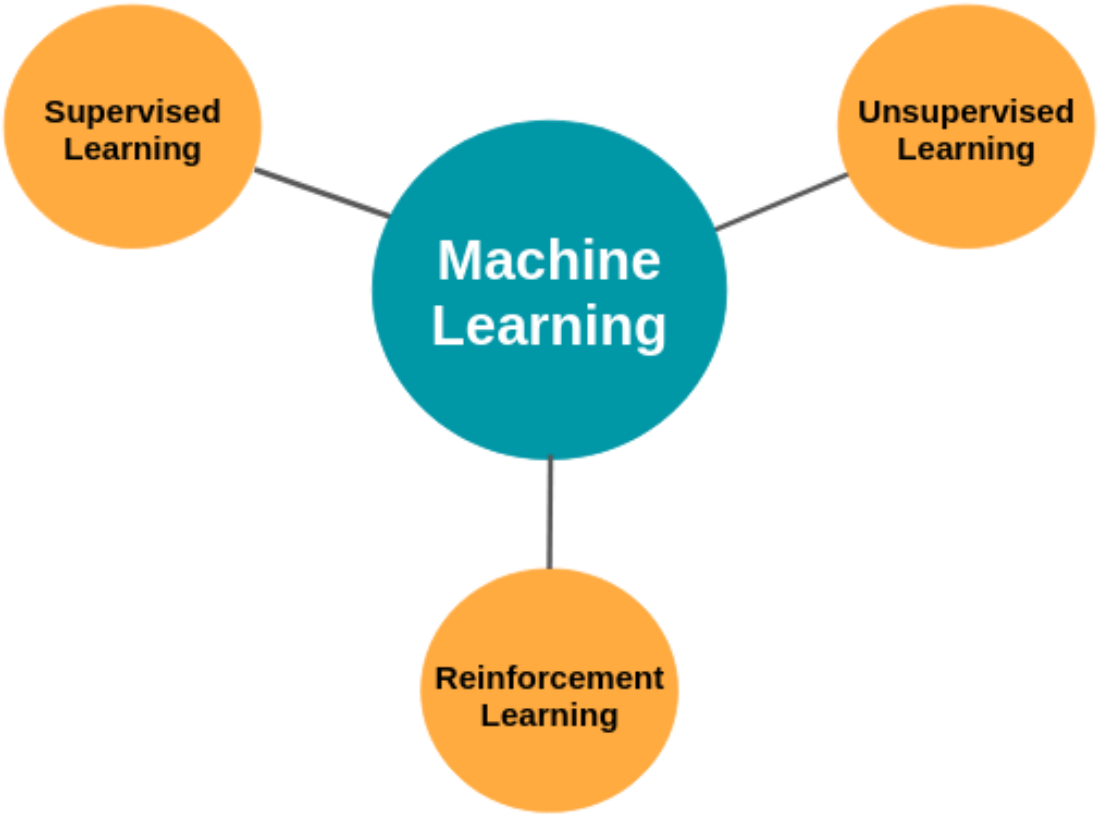

Figure 1.1: The scheme shows the machine learning fields.

Unsupervised learning is a useful tool to find morphology, clustering, and identifications (Goldsmith, 2006; Jain et al., 2008). Some methods, such as K-means (Lloyd, 1982), dimensionality reduction (Kirby, 2000), and clustering, are examples of unsupervised learning methods. Reinforcement learning is a method that is useful to find solutions for a problem by itself. The most famous reinforcement learning use was DeepMind's AlphaGo (Silver et al., 2016). (Silver et al., 2016) used a combination of reinforcement learning with supervised learning, feeding the model with solutions learned by trial and error.

Usually, supervised learning may be associated with DL. DL is a field of ML that uses deep neural networks. The first neural network was a single neuron called the perceptron (Rosenblatt, 1960). Then neural networks started to evolve into multi-layer perceptron (MLP) (Murtagh, 1991). A MLP is composed of neurons and layers. Usually, a MLP with one-layer is called a shallow neural network, and with more than one-layer is called deep

neural networks (Figure 1.2). The output layer contains the target value, and then an error function calculates the difference between the model prediction and the target. This error function is called the loss function. The loss function is a function that measures the errors for the inputs and weights. We train the model to minimize the loss function, i.e., minimize the error between the prediction and the target. The weights are updated, and the goal is to find the minimum error.

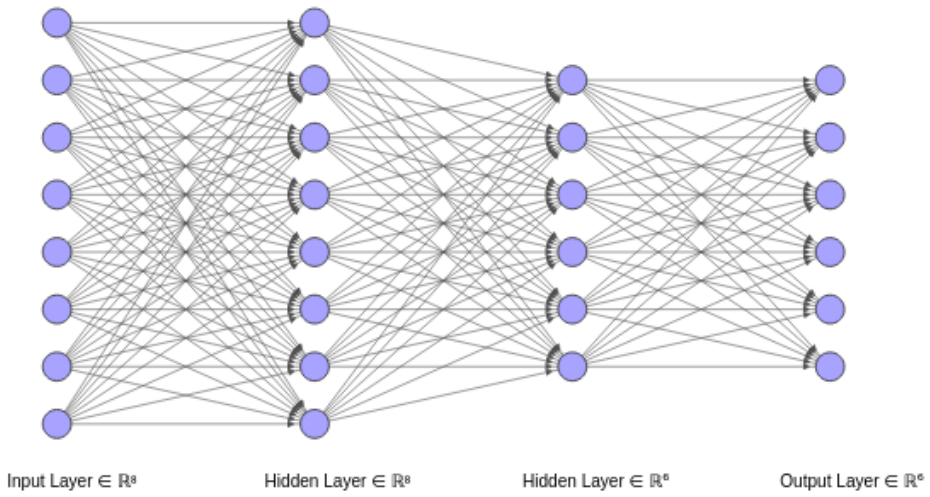

Figure 1.2: A deep neural network architecture: each node is called neuron. The first layer (the input layer) is the layer that receives the data. The hidden layers are composed of neurons that compute non-linear functions based on what they received. Hidden layers are useful for extracting information and they learn features. The output layer is the layer that will compare the prediction with the target and compute the loss function. Architecture built in NN-SVG.

Convolutional neural networks (CNNs or ConvNets) were introduced to capture the spatial relations in the data (LeCun and Bengio, 1998). Here, filters that are represented by matrices replace the neurons. In the CNNs, the weights are in the filters as the matrix elements. Also, each filter is responsible for a convolution, hence the name for this type of neural network. In Figure 1.3, we show the architecture of a CNN. CNNs are widely used to image classification (Krizhevsky et al., 2012) and to image segmentation (Çiçek et al., 2016) for two-dimensional and three-dimensional data that presents spatial and temporal correlations (e.g., pictures, snapshots from videos, time frames) (Çiçek et al., 2016; Wang et al., 2020).

DL methods have several applications in Astronomy. NNs are being used to analyze a

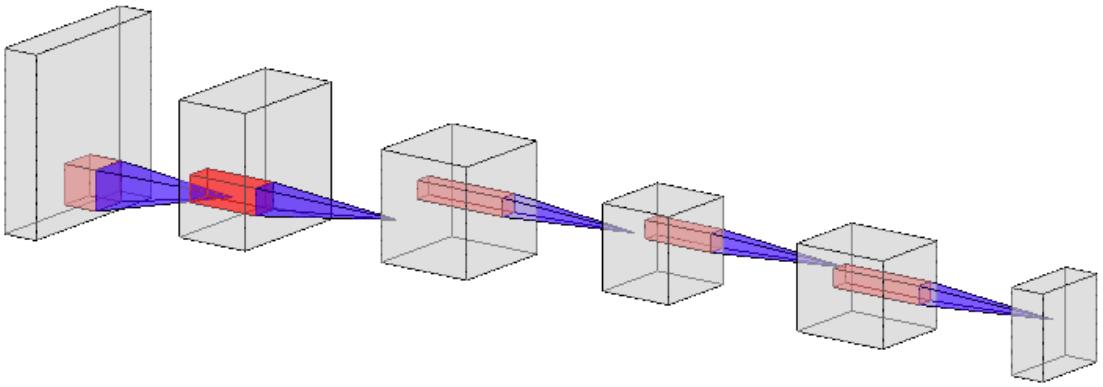

Figure 1.3: Convolutional layers from a CNN-based architecture. Each gray block is output after a convolution. The first gray block is the input block. It receives the data (in the form of 4D arrays) and will start to propagate in the model. The last gray block is the output block, and it will compare the prediction with the target. The hidden blocks act as the hidden layers in the deep neural networks. Red blocks inside the gray blocks are the filters performing the convolutions. In blue, we can see how the convolutions map the regions in the next block. Architecture built in NN-SVG.

large volume of observational data that would be impracticable otherwise (Rezaie et al., 2019; Pearson et al., 2019). It allows to automatically classify objects (Hon et al., 2017) and find their properties (Akeret et al., 2017) in a faster way than traditional methods. Recently, a NN was used to find evidence of exoplanets in the system Kepler-90 from data of its spectrum (Shallue and Vanderburg, 2018). Also, CNNs were used to analyze in a pixel level of astronomical images (Hausen and Robertson, 2019). Hausen and Robertson (2019) used data from the flux of astronomical observations and performed a segmentation using CNNs. Through the segmentation and flux data, the model could learn the morphological classification.

The use of DL in cosmological simulations is a new and promising application (Perraudin et al., 2019; Mathuriya et al., 2018; He et al., 2019). Cosmological simulations are examples of simulations with high computational cost and time. DL can potentially speed-up those simulations and make it possible to investigate a larger fraction of the parameter space (Perraudin et al., 2019).

1.4 Related Work

Several papers showed the promising results of using ML to solve challenging physical problems (Mohan et al., 2019; Pathak et al., 2018; Cranmer et al., 2020; Greydanus et al., 2019). There is progress in the direction of using ML to understand chaotic systems. Some of the works published in this field served as an inspiration for this project (Pathak et al., 2018; Mohan et al., 2019; Breen et al., 2020).

Pathak et al. (2018) explore ML as a tool to predict the future of spatiotemporally chaotic systems from observations of their previous states. The reservoir computing is the ML approach in Pathak et al. (2018), and it is a framework of recurrent neural networks (RNNs). The reservoir computing technique (Lukosevicius and Jaeger, 2009) is effective for data with a temporal sequence. In Pathak et al. (2018), the system is spatiotemporally chaotic modeled by a modified Kuramoto-Sivashinsky equation, which is a fourth-order partial differential equation:

$$\frac{\partial y}{\partial t} = -y\frac{\partial y}{\partial x} - \frac{\partial^2 y}{\partial x^2} - \frac{\partial^4 y}{\partial x^4} + \mu \cos\left(\frac{2\pi x}{\lambda}\right) \tag{1.2}$$

The last term is the modification with μ and λ being constants. The solution of Equation 1.2, i.e., the scalar field $y(x,t)$ is the data used to train the model. The goal is to predict $y(x,t)$ for $t > 0$ from solutions $-T \leqslant t \leqslant 0$ for x varying in the range $0 \geqslant x \geqslant L$ where L is integer of λ.

Figure 1.4 shows the comparison between the numerical solutions (a), the predictions from the ML (b), and the error (c). The reservoir computing can predict based only on previous states of the system from the numerical solutions. They achieved the predictions until half of the chaotic behavior of the system in the test set. After $\Lambda_{max}t = 6$, a single reservoir could not predict the system correctly, and the $\Lambda_{max}t = 10$ is the limit for 64 reservoirs. $\Lambda_{max}t$ is the Lyapunov time.

Breen et al. (2020) proposed using DL to replace numerical solvers for the chaotic three-body problem. Breen et al. (2020) fed a deep NN with simulations of the three-body problem. To have a large dataset, they simulated the three-body problem by using the integrator Brutus with different initial conditions. In their work, they trained the deep NNs with 9900 simulations. The trained NN could provide a solution between 10^5 and 10^8 times faster than a numerical solver. Since the numerical simulations fail when the particles collide, this is a limitation for the ML method as well. The model did not learn

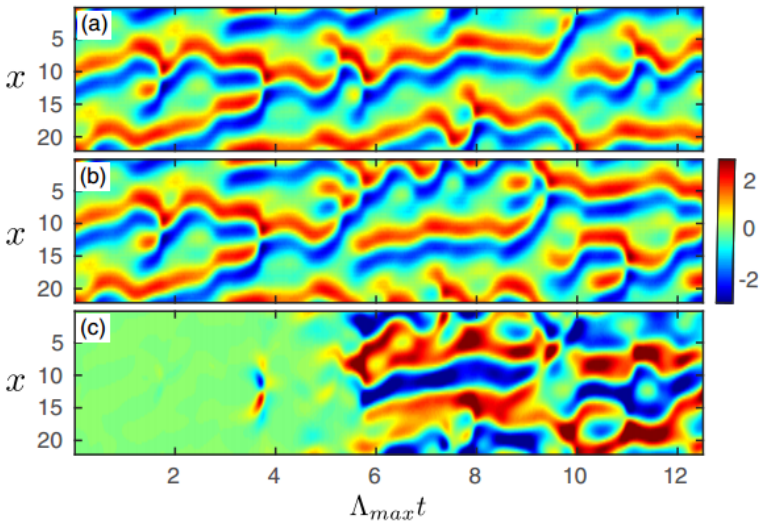

Figure 1.4: The comparison between the numerical solutions to equation 1.2 (a) and the predictions (b) for $\mu = 0$, $L = 22$ and using one reservoir. The last panel is the error - (b) minus (a). Λ_{max} is the maximum Lyapunov exponent of the system. Figure extracted from Pathak et al. (2018).

from observations of the particles colliding or near collision orbits.

Mohan et al. (2019) used a long short-term memory network (LSTM) to model the dynamics of a turbulent flow without directly solving the fluid equations. The work uses a type of RNN as reservoir computing. LSTMs networks have a lower computational cost since they have the presence of a forget gate. The forget gate enables the model only to learn useful information from the data. Mohan et al. (2019) fed the LSTM model with the 3D-velocity fields from forced turbulent flows and magnetized turbulent flows. The results show turbulent fields predicted by LSTMs that could give similar results to the simulated ones. They based the similarity on physics of the system by analyzing energy spectra, Fourier transform of the velocity and the flow morphology. However, the Mohan et al. (2019) did not use datasets with gravity.

1.5 This Work

We propose in this work to use DL methods for creating a model for the complex nonlinear dynamics of an accreting BH - what we call BH weather forecasting. To this date, it is the first time that this application of DL is proposed. We are using the longest HD simulation of RIAFs made by Almeida and Nemmen (2020) to train our model. Since

we are dealing with spatial features, we will use CNNs and CNNs-based networks to build our architecture. Our goal is to create a model that can predict the future states of the accretion flow by itself from observations of the previous states of this system.

The structure of this dissertation is as follows: In Chapter 2, we describe the astrophysical data used to train the models in this work. In Chapter 3, we present the methods such as the learning procedure, the model, and the techniques. We show the results in Chapter 4. In Chapter 5 and Chapter 6, we discuss the results and give the conclusion, respectively.

Chapter 2

Astrophysical Dataset

To train a DL model, it needs to learn from data. We give the observations to the model as the data of the system we want the model to learn from. In this project, we want the DL model to learn black hole physics from the density field of the accretion flow. In Figure 2.1, we show the pipeline of the learning process. The first stage is to obtain data from numerical simulations. The simulated data in this project is from Almeida and Nemmen (2020) who performed HD simulations of a RIAF. We use the matrices corresponding to density field $\rho(\mathbf{r})$ at different times t as the input data. Then, we perform data preparation in every matrix before we feed the DL model with them. The data preparation puts the data in the format appropriate for the learning procedure. We divide the data into three stages: training, validation, and testing. The training is when the model observes and learns from the dataset. The validation is an unbiased evaluation during the training procedure. In the validation stage, the model does not use this dataset to learn directly. Finally, the final step is the test set that is when we obtain the predictions from the trained model, i.e., the BH accretion flow density fields without the need to solve the fluid equations numerically. We will compare the predictions with the ground truth from numerical simulations.

In this chapter, we will first go through the numerical simulations from Almeida and Nemmen (2020) in Section 2.1, explaining the fluid equations that are solved and the data obtained. Then, we will explain the data preparation used in this project in Section 2.2. Finally, in Section 2.3, we will show: training, validation, and testing, as well as the input and output of the DL model.

Figure 2.1: A flowchart of how the data flow from the numerical simulations until the training part. The goal is to obtain a model that outputs $\rho(\mathbf{r}, t)$ without solving the fluid equations numerically.

2.1 Numerical Simulations

Almeida and Nemmen (2020) performed HD simulations of SMBHs accreting at low $\dot{M}$. The main interest of the work was to study winds and feedback from SMBHs accreting in this mode. They considered a Schwarzschild BH, by adopting a pseudo-Newtonian potential to describe the BH's gravity. The viscous stress tensor incorporates the angular momentum transport.

Their work used units such that $GM = 1$, and the Schwarzschild radius is given by $R_S = 2GM/c^2 = 1$. Lengths and times are expressed in terms of units of GM/c^2 and GM/c^3 respectively. The coordinates are as follows: R corresponds to the radius in cylindrical coordinates, r in spherical coordinates. The mathematical method used to solve numerically the Navier-Stokes equations is described in more details by Almeida and Nemmen (2020).

- Equations set

The fluid equations are a set of conservation equations:

$$\frac{d\rho}{dt} + \rho\nabla \cdot \mathbf{v} = 0, \tag{2.1}$$

$$\rho\frac{d\mathbf{v}}{dt} = \nabla P - \rho\nabla\psi + \nabla \cdot \mathbf{T}, \tag{2.2}$$

$$\rho\frac{de/\rho}{dt} = -P\nabla \cdot \mathbf{v} + \frac{\mathbf{T}^2}{\mu}, \tag{2.3}$$

where ρ is the density, $\mathbf{v}$ is the velocity, P is the pressure, and e is the internal energy density. The BH's gravity information is given by the pseudo-Newtonian potential $\psi = GM/(r - R_s)$ (Paczyński and Wiita, 1980). ψ incorporates the features of Schwarzschild' solutions of BH as, for example, the position of the innermost stable circular orbit (ISCO), $r_{isco} = 3R_s$ for a Schwarzschild BH.

The stresses play a fundamental role in the accretion flow. The magnetic stresses are responsible for transporting angular momentum. Thereby leading to accretion. Almeida and Nemmen (2020) incorporate these stresses in an effective way, through the viscous stress tensor $\mathbf{T}$ given by:

$$T_{r\phi} = \mu r \frac{\partial}{\partial r}\left(\frac{v_\phi}{r}\right), \tag{2.4}$$

$$T_{\theta\phi} = \frac{\mu sin\theta}{r}\frac{\partial}{\partial \theta}\left(\frac{v_\phi}{sin\theta}\right), \tag{2.5}$$

where $\mu = \nu\rho$ is the viscosity coefficient and ν is the kinematic viscosity (Landau and Lifshitz, 1959). The only non-zero components of $\mathbf{T}$ are the azimuthal components. We are interested only in the azimuthal values because the dynamical friction, which causes the angular momentum transport, is due to rotation in the azimuthal direction. Almeida and Nemmen (2020) explore two parametrizations of the kinematic viscosity, similarly to Stone et al. (1999):

- $\nu = \alpha r^{1/2}$ is the "K-model" in Stone et al. (1999). This parametrization of ν is referred by Almeida and Nemmen (2020) as ST. Since it depends only on r, it does not depend on the dynamics of the system. With this parametrization, the flow's dynamics do not affect the viscosity.

- $\nu = \alpha c_s^2/\Omega_K$ is the parametrization by Shakura and Sunyaev (1973). Almeida and Nemmen (2020) referred to as SS. The Shakura-Sunyaev's parametrization comes from two considerations: the turbulence's velocity cannot be higher than the medium's sound velocity to avoid shock effects. Another consideration is that the turbulence must be inside the torus to make the angular momentum transport locally.

In the parameterizations of ν, Ω_K is the Keplerian angular velocity, c_s is the sound speed, α is the Shakura-Sunyaev parameter (Shakura and Sunyaev, 1973). In accretion flow simulations, α is the parameter associated with turbulence (Stone et al., 1999; Shakura and Sunyaev, 1973). Almeida and Nemmen (2020) used an α varying with R but for our purposes, we used only the data where α is constant with a value between $0.01 < \alpha < 0.3$.

- Initial Conditions

The initial condition of the simulations is a rotating torus in dynamical equilibrium. The size of the torus varies with the choice of the angular momentum distribution. It varies from $R_{in} = 5 - 20R_s$ to $R_{out} = 500R_s$. The angular momentum configuration is important since it describes the torus' kinetic energy. Almeida and Nemmen (2020) investigated two angular momentum profiles:

- $l(R) \propto R^a$, with $a = 0.0$, $a = 0.2$, and $a = 0.4$. Since it is a power-law, we will refer to it as PL.
- $l(R) = \begin{cases} constant & R < 21R_S \\ 0.71 l_K & \text{otherwise} \end{cases}$, where l_K is the Keplerian specific angular momentum. Since it is based on Penna et al. (2013), so we will refer to it as PN.

The values of a affect the size of the torus. In Figure 2.2, there are four plots from Almeida and Nemmen (2020). It shows how the profiles described above affect the shape and thickness of the torus.

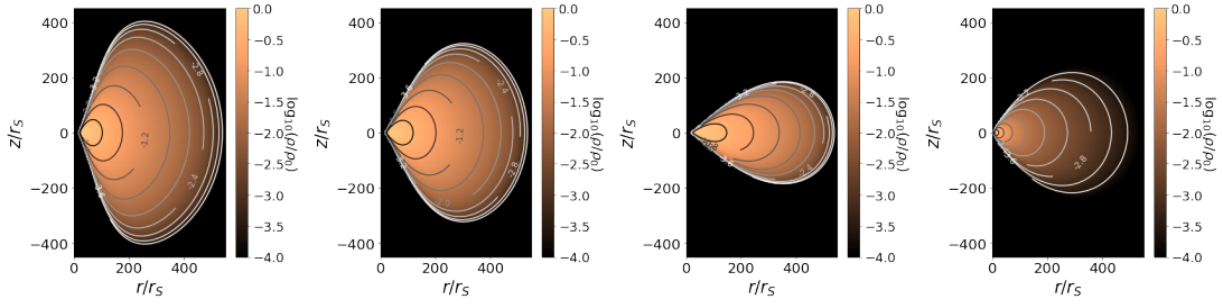

Figure 2.2: Plots showing the different momentum angular profiles, extracted from Almeida and Nemmen (2020). The first one is showing the profile $l(R) = 1$, or the situation where the profile is PL and $a = 0$. The second and the third one are showing $l(R) = R^a$ with $a = 0.2$ and $a = 0.4$, respectively. The last one is showing the torus shape when it uses the Penna's profile. Here we can see how the angular momentum profiles change the shape and thickness of the torus.

Even though the torus has a size $R_{out} = 500R_s$, the computational grid extends to $\sim 10^4 R_s$, to avoid any undesirable effects in the radial direction. Meanwhile in the azimuthal direction it was assumed $2° < \theta < 178°$. The grid, shown in Figure 2.3, was fixed for all simulations and has a size of $(400, 200)$.

- List of Simulations

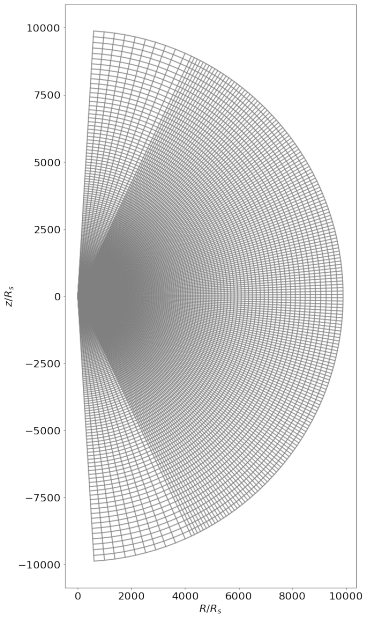

Figure 2.3: The simulation grid. While in the inner part, the resolution is better, the outer parts have less resolution. The same goes for the polar regions.

Almeida and Nemmen (2020) performed eleven simulations varying viscosity profile, α, and the angular momentum profile. Table 2.1 shows the simulations with their respective parameters as well as the number of snapshots generated. To our interests, we will distinguish the angular momentum profile as PN or PL and the viscosity profile ν as ST or SS. The time difference between two frames is $\Delta t = 197.97\, GM/c^3$, and it is the condition between two frames defined by Almeida and Nemmen (2020)'s work.

Table 2.1 - This table shows all simulations performed by Almeida and Nemmen (2020). The name is based on the parameters of the simulations. In the second column, they are distinguished as Penna (PN) and power-law (PL). The third column shows the name of the parametrization of the viscosity ν: Shakura-Sunyaev (SS) and Stone (ST) with the Shakura-Sunyaev's parameter α in the fourth column. The last column is the number of snapshots. This last feature is essential for this present work.

Name	$l(R)$	ν	α	Number of Snapshots
PNST01	PN	ST	0.01	4059
PNST1	PN	ST	0.1	456
PNSS1	PN	SS	0.1	4355
PNSS3	PN	SS	0.3	3678
PNSSV	PN	SS	$\alpha(R)$	3840
PL0ST1	PL	ST	0.1	380

Tabela 2.1 - Continuation

Name	$l(R)$	ν	α	Number of Snapshots
PL0SS3	PL	SS	0.3	1114
PL2SS1	PL	SS	0.1	1068
PL2SS3	PL	SS	0.3	1068
PL2ST1	PL	ST	0.1	190
PL4ST01	PL	ST	0.01	1097

We stress that the α parameter here is not exactly the Shakura-Sunyaev parameter. The exact Shakura-Sunyaev's prescription are the ones in simulations with Shakura-Sunyaev's characterization. In Figure 2.4, we show some examples of the density maps.

2.2 Data preparation

Before beginning the DL analysis, we need to prepare the data. We will discuss the data preparation through the following steps:

- Normalization
- Crop
- Create data cubes
- Organize the input and output

Normalization is required when the ranges of values of the data are too broad. In our data, we have values of the density as low as $\sim 10^{-4}$ and others as large as $\sim 10^{1}$, we have the same min-max values for training set and testing set. If we do not normalize the data, we introduce a bias in the training, where the model will give more weight to larger values, thereby losing information regarding the lower ones. The main goal of normalization is to have unbiased learning, so instead of a range $(10^{-4}, 10^{1})$ it is preferable to use a range $[0, 1]$. Usually, in deep learning methods, the most common normalization is to change to a range $[-1, 1]$. Still, since our data consists of positive values as the density only accepts positive values, then we used a range $[0, 1]$. We have also tested a min-max normalization:

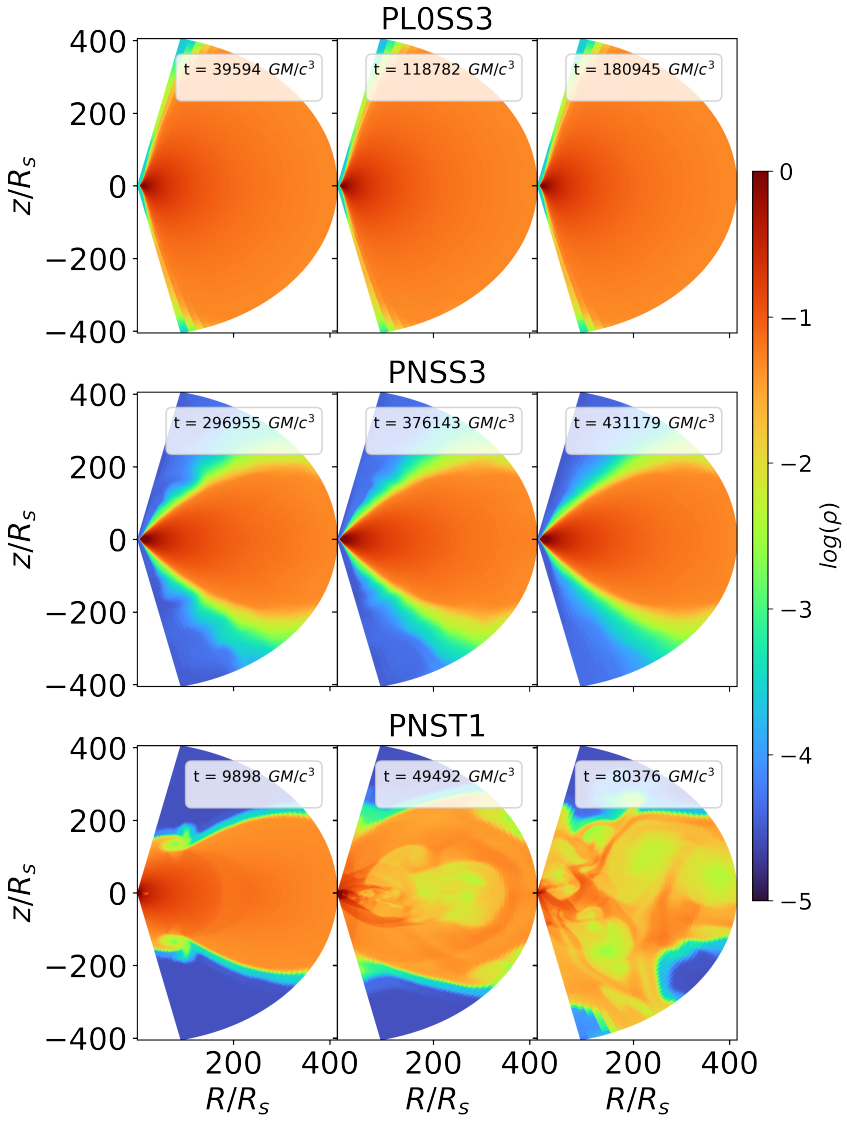

Figure 2.4: Examples of the density plots from the simulations PL0SS3, PNSS3, and PNST1. The plots show the evolution of the fluid in three distinct times. Both axes are in R_s units, and the colorbar is $\log(\rho)$.

$$\rho_{NORM} = \frac{\rho - \min(\rho)}{\max(\rho) - \min(\rho)}, \tag{2.6}$$

where ρ_{NORM} are the normalized values, and ρ are the original values but the model didn't converge in preliminary tests due to a very biased data. Instead, the best approach we found was to use a logarithmic min-max normalization:

$$\rho_{NORM} = \frac{log(\rho) - log(\min(\rho))}{log(\max(\rho)) - log(\min(\rho))}. \tag{2.7}$$

We also need to crop the original data. In Figure 2.5, we show how the transformation

works. The atmosphere presents a problem to the model since it fills a vast area in the data and the model ends up ignoring the torus region over the atmosphere region.

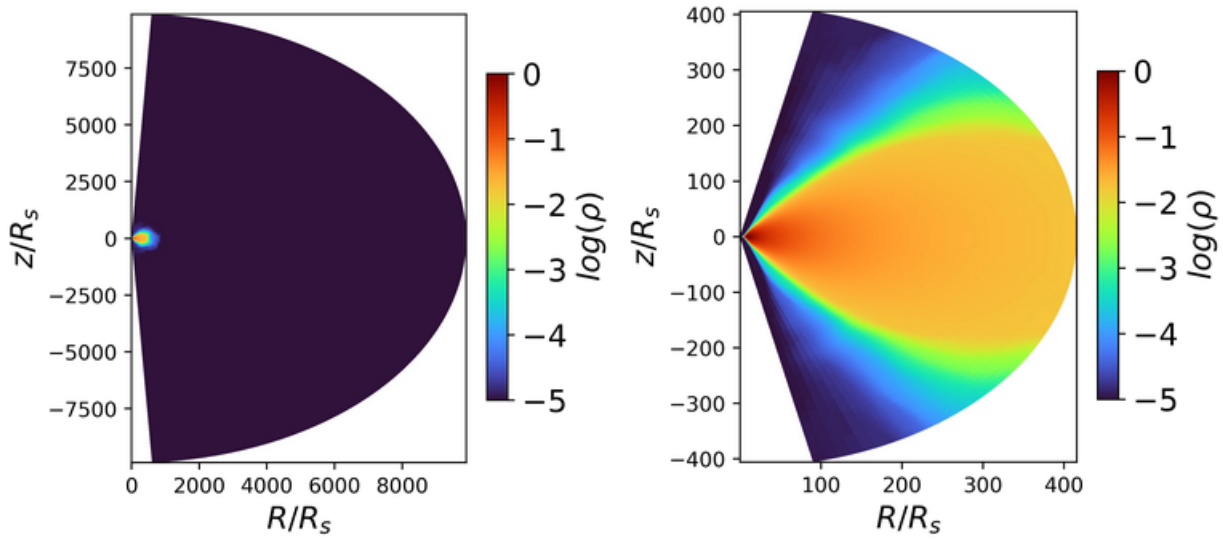

Figure 2.5: Plots of $log(\rho)$ from PNSS3 before the crop in Cartesian coordinates. In the first panel, we see the plot before the crop. The original matrix is in polar coordinates with 200×400 size. The second panel is the same plot after the crop of the atmosphere part. The original matrix is in polar coordinates with 192×256 size, it represents a crop of 8 in the polar direction and 144 in the radial direction.

The crop was performed as follows: in the radial direction, we removed 144 cells that span $9500\,GM/c^2$, encompassing mostly atmosphere with $\rho < 10^{-3}$. Meanwhile, in the polar direction, we remove eight cells.

The last step of the data preparation is to organize how the data will go into the model. Spatial dependence is already intrinsic in the data since each frame has ρ as a function of the positions. Taking into account the temporal structure is more challenging because we need to incorporate this in the model. To incorporate the temporal structure, we create a 3D array containing five consecutive frames. In this work, we call these five consecutive frame, channels. In Figure 2.6, we show how we build a channel.

The final format of the data consists of an array with size $(N, 256, 192, 5)$, where N is the number of temporal frames available. We will feed the array to the architecture as the input and the output will have the same size. If the input is the block where the first snapshot is the one of time t_n, then the array will be $(bs, 256, 192, t_n \text{ to } t_{n+4})$, the output will be $(bs, 256, 192, t_{n+5} \text{ to } t_{n+9})$, where bs is the batch size, shown in Figure 2.7.

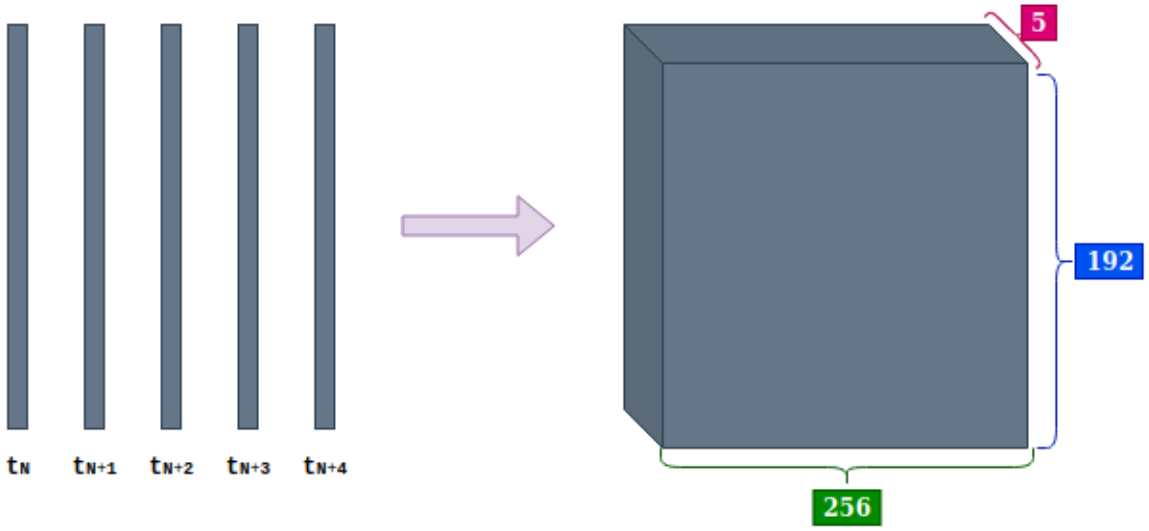

Figure 2.6: Data structuring for the CNN. First, we append five consecutive (256×192) arrays creating a $(256 \times 192 \times 5)$ array. This block will be a 3D array with spatial and temporal information. The vector will then be $(N \times 256 \times 192 \times 5)$ array with N being the number of the snapshots.

2.3 *Training, Validation and Testing sets*

In the model's learning process, there are three critical datasets: the training, validation, and test set. Each one has an important role when it comes to training and evaluating the model:

- **Training set**: this dataset is what the model uses to learn. It observes the data and learns from it. The loss function is calculated from examples of this set. In our work, the training set consists of a sequence of density fields that are solutions of the fluid dynamics equations.

- **Validation set**: this set is used to validate the training. The validation set informs the model's performance with data not seen in the training part. The validation is not used to update the weights directly, it is only used to evaluate the model. This set is used to judge whether the weights are updated correctly and whether the training can stop. The validation set allows an unbiased validation of the model.

- **Testing set**: the testing set is never seen by the model during the training. We feed this set only after the model is trained. The training set is used to evaluate how well

the model reproduces the data.

An important part of the process of building a model is to define how the data it will be the split among the training, validation, and testing sets. In the literature, the most common splits are 70%, 15% and 15% or 80%, 10% and 10%. We employed 80% for the training set and 20% for the testing set. The validation set is given by 10% of the training set. This will give the train, validation and test splits equals to 67.5%, 12.5% and 20%, respectively.

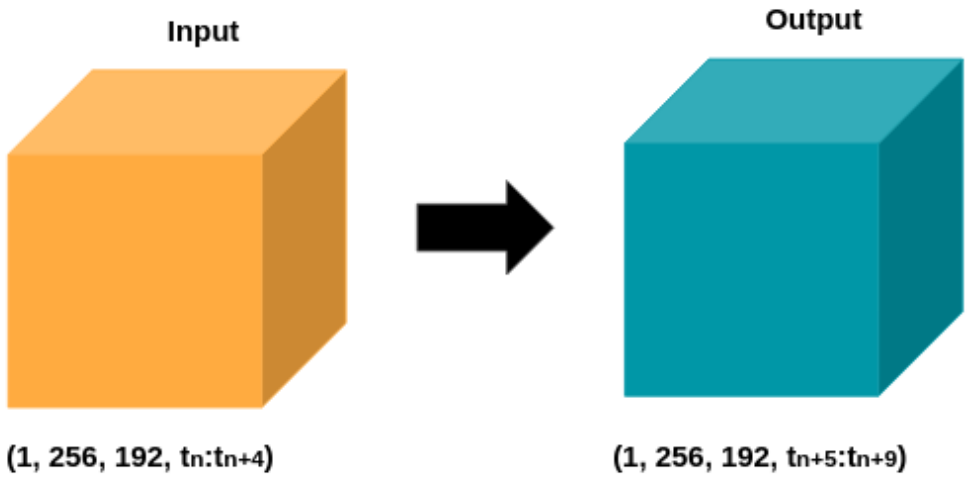

Figure 2.7: The input is a block with size $(bs, 256, 192, t_n : t_{n+4})$ and the output is the block with size with $(1, 256, 192, t_{n+5} : t_{n+9})$. t_n represents snapshot of time t_n. bs is the batch size number. The input and output is $(bs, 256, 192, t_{n+1} : t_{n+5})$ and $(bs, 256, 192, t_{n+6} : t_{n+10})$, respectively.

Chapter 3

Methods

In this work, our primary goal is to develop a DL model that can learn the states of the accretion flow around an SMBH. We want to achieve this by feeding the model with scalar fields from numerical simulations. To be more specific, we will supply a DL architecture with density fields obtained by HD numerical simulations of a RIAF. The expected result is that the model can predict future states by itself. Ideally, it will learn basic features such as density values and the shape of the torus as well as the physics of the system.

The model won't directly solve the Navier-Stokes equations introduced in 2, it will learn by observations of the density states. In this chapter, we will introduce the DL methods and how they are used to learn and make predictions. We want to give an overview of what are neural networks (NNs) and how they learn by observations. Also, we will introduce other types of NNs that we use in this work. It is useful to give a mathematical treatment of how NNs learn.

The main architecture used in this project is a convolutional neural network (CNN) in a U-shape architecture called U-Net (Ronneberger et al., 2015). The U-Net is a network that deals with spatial features since it is CNN-based. The concatenations between the encoder and decoder make the network to consider the input when creating the output. The architecture is fed with the data explained in the previous chapter after the treatment procedure detailed in Section 2.2.

3.1 Neural Networks

Neural networks (NNs) are the primary tool in DL. They were introduced after the work of McCulloch and Pitts (1943) that tries to mimic the learning process of biological

neurons. The first NN - the perceptron - had one neuron and one output. After, the NNs evolved to a multilayer perceptron (MLP). The MLP consists of neurons and layers, as shown in Figure 3.1. The first layer is called the input layer, and the last layer is called the output layer. The layers in the middle are called hidden layers. Usually, we consider a shallow neural network as an MLP with one hidden layer, while an MLP with more layers is called a deep neural network (DNN). Each layer l is composed of nodes that are called neurons, and each arrow that connects the neurons from a layer to the next layer has a weight w associated.

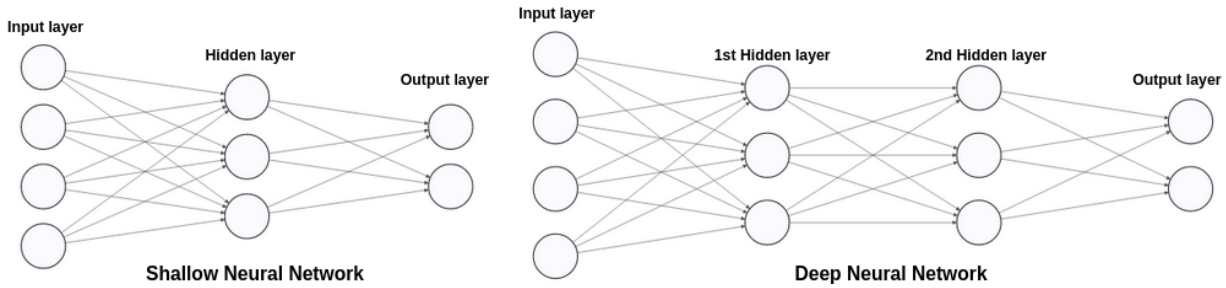

Figure 3.1: On the left, we show a shallow NN. A shallow NN has one hidden layer, which is the layer between the input layer and the output layer. One the right side, we show a DNN. The DNNs have more hidden layers. In the NN showed here, the DNNs have two hidden layers. It is important to note that a DNN can have a large number of hidden layers. More hidden layers, deeper the NN is. Each arrow has a weight w associated with it.

- Learning Process

From here, we will introduce some notation as follows:

- l is the number of the layer. $l = 0$ is the input layer, $l = 4$ is the fourth hidden layer, and $l = L$ is the output layer, where L is the total number of layers.
- i is the number of the neuron in the layer l and j is the number of the neuron in the layer $l + 1$.
- $w_{(i,l),(j,l+1)}$ is the weight in the arrow that connects the neuron i in the layer l to the neuron j in the layer $l + 1$.
- $z_{i,l}$ is a linear combination in the neuron i in layer l.
- $\sigma_{i,l}$ is the output in the neuron i in layer l.

- $b_{i,l}$ is the bias term in the neuron i in layer l.
- $f(\cdot)$ is the non-linear activation function.
- $X = (x_1, x_2, ..., x_M)$ is the input vector of the input layer. This is the data and information we feed the NN. M is the number of the dimensions of the input.
- $Y = (y_1, y_2, ..., y_N)$ is the target vector. This is the data and ground-truth we feed the NN to compare its prediction. N is the number of the dimensions of the output.
- $\hat{Y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_N)$ is the output vector of the output layer. This is the prediction of the NN.

To simplify, we will demonstrate how the information goes into the NN by using the scheme shown in Figure 3.2.

In the scheme, the outputs of the layer l are $\sigma_{(1,l)}$ and $\sigma_{(2,l)}$. They will go through the arrow until reaching the neuron $j = 1$ of the layer $l + 1$. The first step is an operation $z_{(1,l+1)}$, given by:

$$z_{(1,l+1)} = w_{(1,l),(1,l+1)} \cdot \sigma_{(1,l)} + w_{(2,l),(1,l+1)} \cdot \sigma_{(2,l)}, \tag{3.1}$$

where $b_{(1,l+1)}$ is the constant bias associated with this neuron. Usually, the operation $z_{(1,l+1)}$ has also another term called bias $b_{(1,l+1)}$. The value $z_{(1,l+1)}$ will then be activated by a non-linear function $f(\cdot)$ called the activation function:

$$f(z_{(1,l+1)}) = f\left(w_{(1,l),(1,l+1)} \cdot \sigma_{(1,l)} + w_{(2,l),(1,l+1)} \cdot \sigma_{(2,l)}\right) \tag{3.2}$$

This function can be any non-linear function. The most common activation functions are ReLU, sigmoid, and tanh given by:

- ReLU: $f(x) = max(0, x)$
- Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh function: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

The activation functions are used to insert non-linearity in the NN. Without non-linearity, the composition of function will be a linear function and it will not capture non-linear features. The output of the neuron is then:

$$\sigma_{(1,l+1)} = f(z_{(1,l+1)}) = f(w_{(1,l),(1,l+1)} \cdot \sigma_{(1,l)} + w_{(2,l),(1,l+1)} \cdot \sigma_{(2,l)}) \tag{3.3}$$

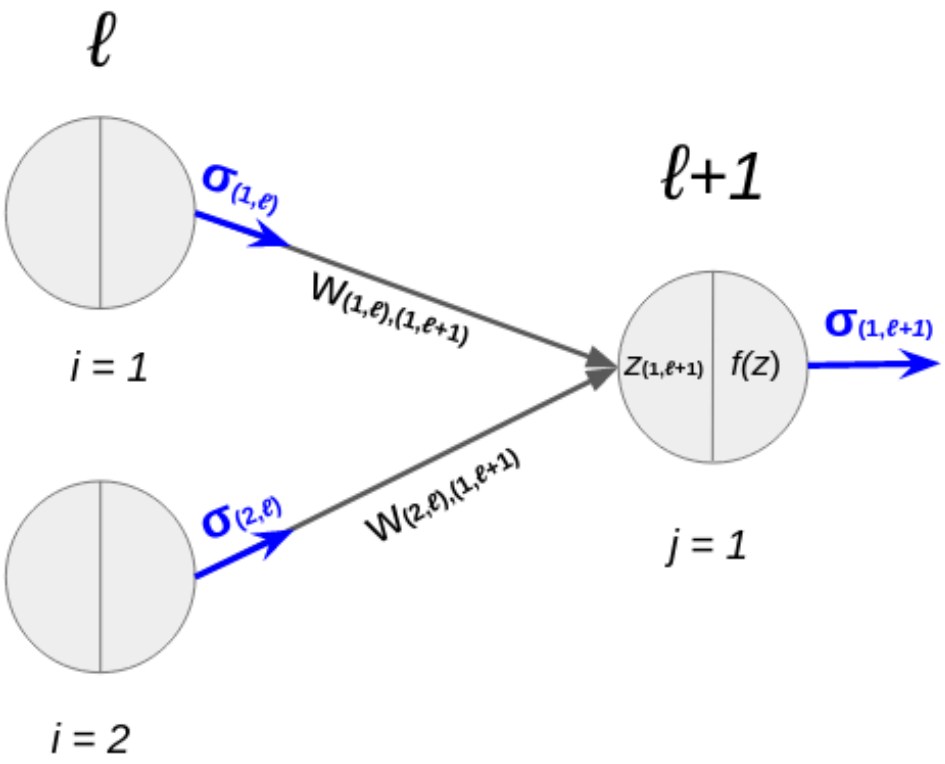

Figure 3.2: We show a scheme with three neurons. Two neurons in the layer l and one neuron in the layer $l+1$. The outputs of the layer l are $\sigma_{(1,l)}$ and $\sigma_{(2,l)}$ and the output of the layer $l+1$ is $\sigma_{(1,l+1)}$. We represent two arrows in this scheme connecting the layer l to layer $l+1$ through its neurons. The weights of each arrow are $w_{(1,l),(1,l+1)}$ and $w_{(2,l),(1,l+1)}$. The first step inside the neuron of the layer $l+1$ is an operation referenced as $z_{(1,l+1)}$ and the second step is the activation function $f(z_{(1,l+1)})$.

This output will then be the input of the next layer. It is important to note that the information of the input layer is the data itself X, and the output of the output layer is the final prediction of the NN, $\hat{Y}$.

The method described above (Goodfellow et al., 2016) will happen in every neuron and layer of the NN in a forward direction, always going from layer l to layer $l+1$ until it reaches the layer $l = L$, the output layer. The output layer will then produce the final prediction $\hat{Y}$. This process is a forward process since it only moves in one direction, and it doesn't go backward.

After the last output $\hat{Y}$ is predicted, it is time to calculate how this prediction differs from the target values Y and how to inform the NN to update its weights $w_{(i,l),(j,l+1)}$. The weights $w_{(i,l),(j,l+1)}$ are responsible for making the NN learn, in other words, the goal here

is to find the combination of $w_{(i,l),(j,l+1)}$ that makes the predictions $\hat{Y}$ close to the target values Y. Mathematically, $\hat{Y}$ is the result of a function F that depends on the information fed to the model and the weights $w_{(i,l),(j,l+1)}$:

$$\hat{Y} = F(X, w_{(i,l),(j,l+1)}) \tag{3.4}$$

Having $\hat{Y}$ available, the NN needs to know how similar $\hat{Y}$ and Y are. A loss function is defined to find the difference between $\hat{Y}$ and Y. The loss function $\mathcal{L}$ can be any error function that is appropriate for the problem, e.g., mean squared error (MSE), mean absolute error (MAE) or mean percentage error (MAPE). In our project, we use a combination of MAE functions as losses:

$$\mathcal{L} = |Y - \hat{Y}| \tag{3.5}$$

The loss function $\mathcal{L}$ is usually called cost function $\mathcal{C}$. However, we can consider the cost function as an average for all data:

$$\mathcal{C} = \frac{\sum_{n=1}^{N} |Y - \hat{Y}|}{N}, \tag{3.6}$$

where N is the number of data points available in training. In our work, we may refer to the cost function as a loss function. We want the minimum value of the difference between Y and $\hat{Y}$, so it is essential to minimize the loss function. In DL, the standard method to minimize a function is called gradient descent (Ruder, 2016). The gradient is a vector that points to the direction where the value of the function increases. The gradient descent is the opposite since it indicates the direction where the function minimizes. Suppose we have a model with only one parameter w and the shape of the function $\mathcal{C}(w)$ as in Figure 3.3. We are at point A, but we want to find the minimum value of $\mathcal{C}(w)$, point B. We take the gradient of the $\mathcal{C}(w)$ (Equation 3.7) and multiply by -1 (Equation 3.8), this will give the direction of the minimum. To control the magnitude of the vector, we can multiply by a constant η (Equation 3.9), this constant is called the learning rate.

$$\nabla\mathcal{C} = \frac{\partial\mathcal{C}}{\partial w} \tag{3.7}$$

$$-1 \cdot \nabla\mathcal{C} = -\frac{\partial\mathcal{C}}{\partial w} \tag{3.8}$$

$$-\eta \cdot \nabla\mathcal{C} = -\eta\frac{\partial\mathcal{C}}{\partial w} \tag{3.9}$$

In Figure 3.3, we show how the learning rate affects the vector that goes in the direction of the minimum. Since we are dealing with infinitesimal values, we can assume that $-\eta\frac{\partial \mathcal{C}}{\partial w}$ is how much the weight w should change to minimize the function $\mathcal{C}$. In this way, the weight w can be updated as:

$$w := w - \eta\frac{\partial \mathcal{C}}{\partial w} \tag{3.10}$$

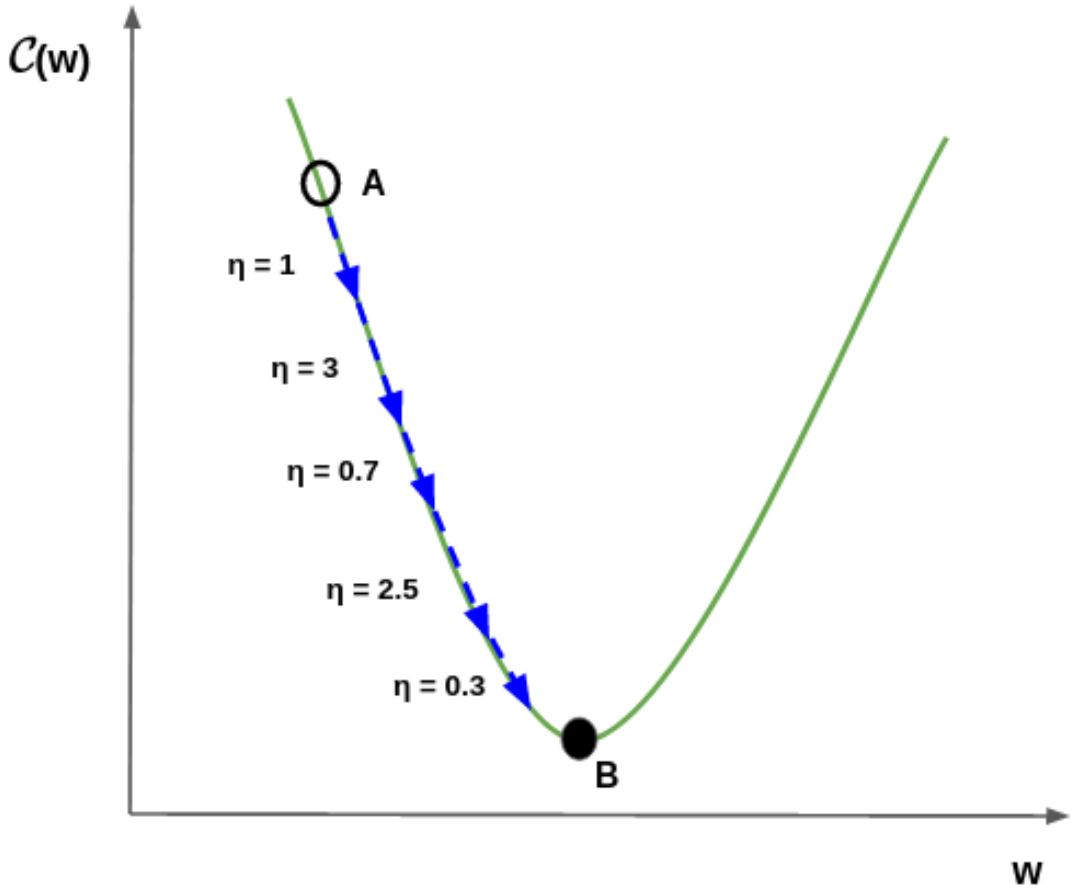

Figure 3.3: The function $\mathcal{C}(w)$ in relation to w. The green curve is the curve of $\mathcal{C}(w)$. The A is the point where we are after one forward step. The ideal is to get to B. In blue arrow, we show how the vectors change with the gradient descent and with the multiplication of η.

The procedure of update backwards is called backpropagation and it is the learning process of the NN. In a realistic approach, the function $\mathcal{C}(w)$ depends on a large number of weights $w_{(i,l),(j,l+1)} = (w_{(1,1),(1,2)}, w_{(2,1),(1,2)}, ..., w_{(P,L-1),(M,L)})$, and the gradient will also be a vector $\nabla\mathcal{C} = (\frac{\partial C}{\partial w_{(1,1),(1,2)}}, \frac{\partial C}{\partial w_{(2,1),(1,2)}}, ..., \frac{\partial C}{\partial w_{(P,L-1),(M,L)}})$. The function acts in an N-dimensional space so we cannot plot the function in a 3D space. The backpropagation algorithm consists of:

- Find the gradient of the function $\mathcal{C}$.
- Find the appropriate value of η.
- Update the weights $w_{(i,l),(j,l+1)}$: $w_{(i,l),(j,l+1)} := w_{(i,l),(j,l+1)} - \eta \frac{\partial \mathcal{C}}{\partial w_{(i,l),(j,l+1)}}$

Some functions $\mathcal{C}$ may have local minima. To avoid being trapped by a local minimum, the learning rate is essential. A higher value of the learning rate will help the model to escape the local minima. However, if the learning rate is too high, it can miss the best minimum, and it can cause numerical problems, e.g., the cost function gets too high, and the model never converge. The gradient descent and the update described above is one of the optimization methods. Other optimizations methods aim at better and faster convergence such as Adam (Kingma and Ba, 2014), Adadelta (Zeiler, 2012), Adagrad (Lydia and Francis, 2019). In our work, we use Adam (adaptive moment) as the optimization method. Adam uses the same method of gradient descent, but the update is written instead as:

$$w_{(i,l),(j,l+1)} := w_{(i,l),(j,l+1)} - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \eta}, \tag{3.11}$$

where:

$$\hat{m} = \frac{m}{(1-\beta_1)} \quad \text{with} \quad m := \beta_1 m + (1-\beta_1) \frac{\partial \mathcal{C}}{\partial w_{(i,l),(j,l+1)}} \tag{3.12}$$

$$\hat{v} = \frac{v}{(1-\beta_2)} \quad \text{with} \quad v := \beta_2 v + (1-\beta_2) \left[\frac{\partial \mathcal{C}}{\partial w_{(i,l),(j,l+1)}} \right]^2. \tag{3.13}$$

In Equation 3.11 and Equation 3.12, β_1 and β_2 are constants that are defined before the training. The first m and v are initialized as β_1 and β_2. We choose Adam as our optimization method because of its momentum parameters, the optimization is faster and avoids vanishing and exploding gradients when it deals with sparse gradients (Kingma and Ba, 2014).

In summary:

1. The data enters the model through the input layer.
2. The input layer will pass the data to the first hidden layer through connections with weights w.
3. While in the hidden layers, Equation 3.1 and Equation 3.2 shows that will occur operations, this will go until the last layer.

4. In the last layer, the prediction will be compared with the target by a cost function (Equation 3.6).

5. The process of updating backwards starts by finding the gradient descent of the cost function (Equation 3.7).

6. The gradient descent will be used to update the weights (Equation 3.12).

7. After all the weights are updated, the forward process starts again.

The learning process will end, ideally, when the cost function converges to its minimum value.

3.2 Convolutional Neural Networks

While NNs are composed of neurons, the units of CNNs correspond to filters (LeCun and Bengio, 1998) which deals better with spatial features since they preserve spatial dependencies. In CNNs, the data can be two-dimensional (2D) or three-dimensional (3D). CNNs are composed of layers that are 2D or 3D arrays – depending on the size of the input – and the main operation is a convolution in a matrix. Figure 3.4 shows a scheme of a ConvNet2D architecture, the input data is a 2D array that may have channels as well as the output of each convolutional layer. For example, a picture has three channels: red, green, and blue channels that are matrices, so we can split the picture into 3 arrays —the channels —, as shown in Figure 3.5. The picture is a block ($H \times W \times C$), where H is the height, W is the width, and C is the number of channels.

In this project, our input is a density field of an accretion flow around an SMBH accreting at low rates. This density field is a matrix with size (256×192) and the channels in our case are the density arrays at different times. We show a scheme (Figure 2.7) of how our data enters the CNN as an array ($256 \times 192 \times 5$), where $C = 5$ is the number of the channels and we want the CNN to output an array with the same dimensions as the input. The entire CNN is composed of layers appropriate for a 2D problem, as shown in Figure 3.4. We will call the array that goes into a layer as the source and the output as the destination. Input is the data that goes in the input layer, and output is the prediction that the model predicts in the output layer.

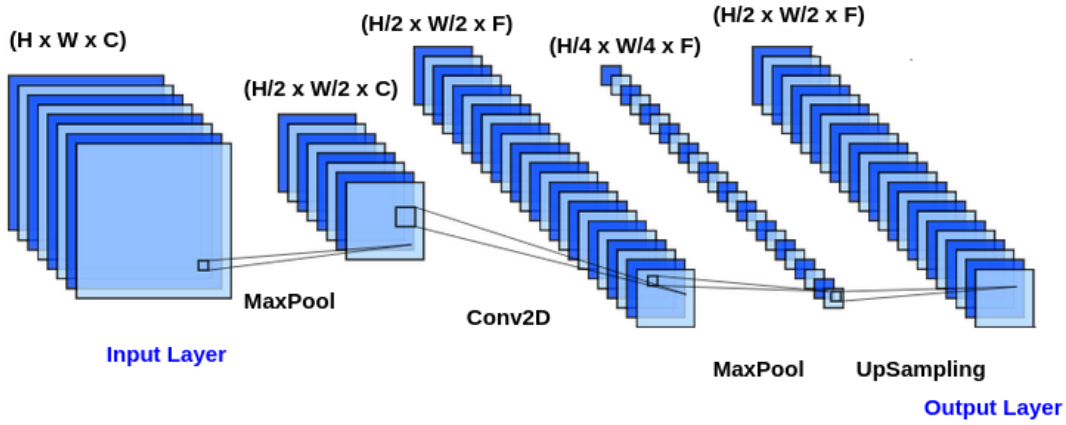

Figure 3.4: An architecture composed of CNNs. The first layer is the input layer - the information we give the model. The layers in the middles are analogous to the hidden layers in NNs. The last layer is the output layer, as in the NNs, the output layer is what gives the final prediction of the model, and it will compare with the target. Conv2D represents the operation made by convolution layers that are the layers that will learn and update their parameters analogous to the neurons. MaxPool layers are layers that realize resizing operation by decreasing the width and height of the matrices. UpSampling is the opposite of MaxPool.

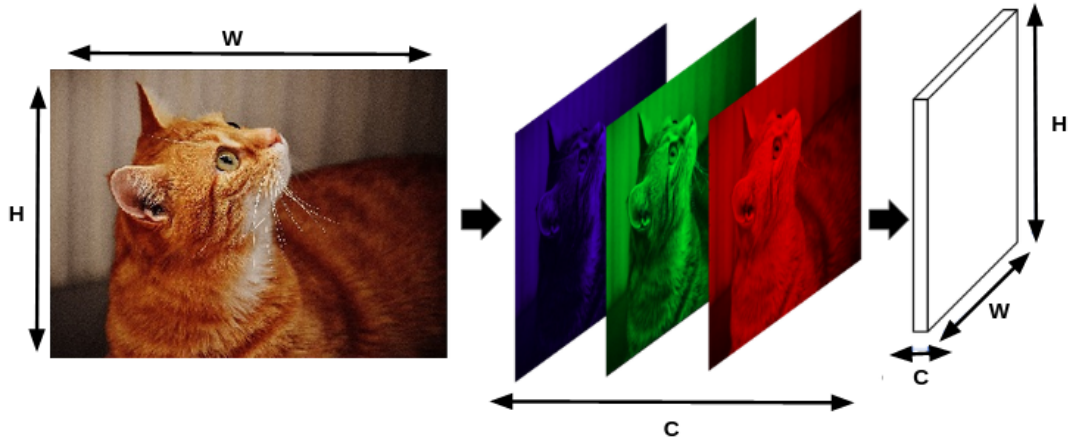

Figure 3.5: Picture extracted and modified from corochann.com. The picture is a $(H \times W)$ matrix. However, we can separate the three channels RGB in 3 matrices. If we append these matrices in a block, we will have an array of $(H \times W \times C \times 3)$.

- Max-Pooling Layer

Figure 3.4 shows an input layer with size $H \times W \times C$. When H and W are large numbers, this can lead to an expensive computational cost. One solution to keep the main information and decrease the size of the matrix is to use a Max-Pooling layer (MaxPool) (Nagi et al., 2011). The MaxPool is a layer —a $M_H \times M_W$ matrix. It will pass through

a matrix and takes the maximum value of the region, i.e., it will take the dominant information. Usually, the MaxPool layer is (2×2), so it decreases the size of the source with dimensions $H \times W$ by half, except the channel dimension that will stay the same. It is important to remind that the MaxPool layer does not have weights to learn, and it can be applied between any layers.

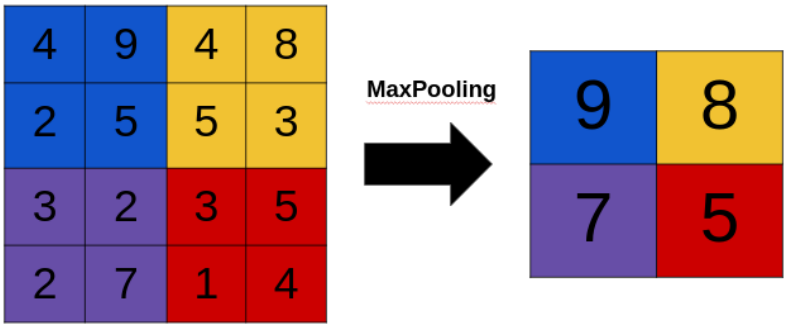

Figure 3.6: This example shows how a 2×2 MaxPool layer acts. It extracts the maximum value of each (2×2) region of the matrix. This operation is important to decrease the size of the arrays inside the model and to extract dominant information of each region. It prevents the model from learning the non-important features of the data.

- UpSampling Layer

The UpSampling layer (Gopinath and Burrus, 1994) acts as the opposite of the MaxPool layer. The UpSampling layer will repeat the most important features of the source. In Figure 3.7, we show an UpSampling layer repeating the information from a (2×2) matrix until it increases to a (4×4). The UpSampling layer is important to get the data back to its original size in our case. Our input has a dimension of $(256 \times 192 \times 5)$ and our output also has the same dimension. After the MaxPooling layers, we get our data the same size as before with UpSampling layers.

Other options for resizing layers are available. The most common is the deconvolution layer. We choose to use the UpSampling layers since it does not have weights to learn, meaning a lower computational cost and thus a better performance for our problem.

- Convolution Layer

The convolutional layers (Conv2D) are layers that perform a convolution operation (LeCun and Bengio, 1998). The convolution is the operation where we take the filters, and

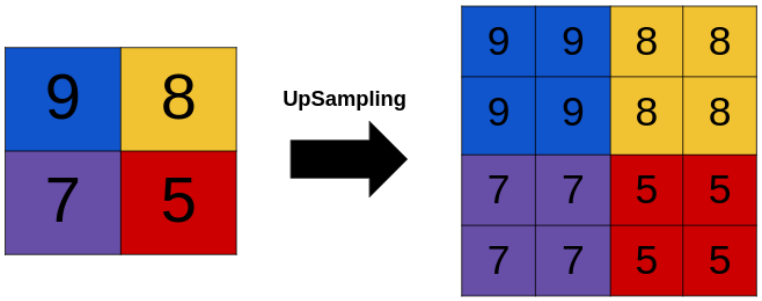

Figure 3.7: This example shows how a (2×2) UpSampling layer would act. It will take the values and repeats them until the same dimension of the UpSampling layer.

we pass through all the data transforming it (Figure 3.8). Let S be the source matrix, F be the filter and D the destination matrix:

$$D[i,j] = (S * F)[i,j] = \sum_k \sum_l F[k,l]S[i+k, j+l]. \tag{3.14}$$

The convolution is denoted by $*$, and it is equivalent to the operation z in the NNs. The learnable parameters are the elements of the filter F.

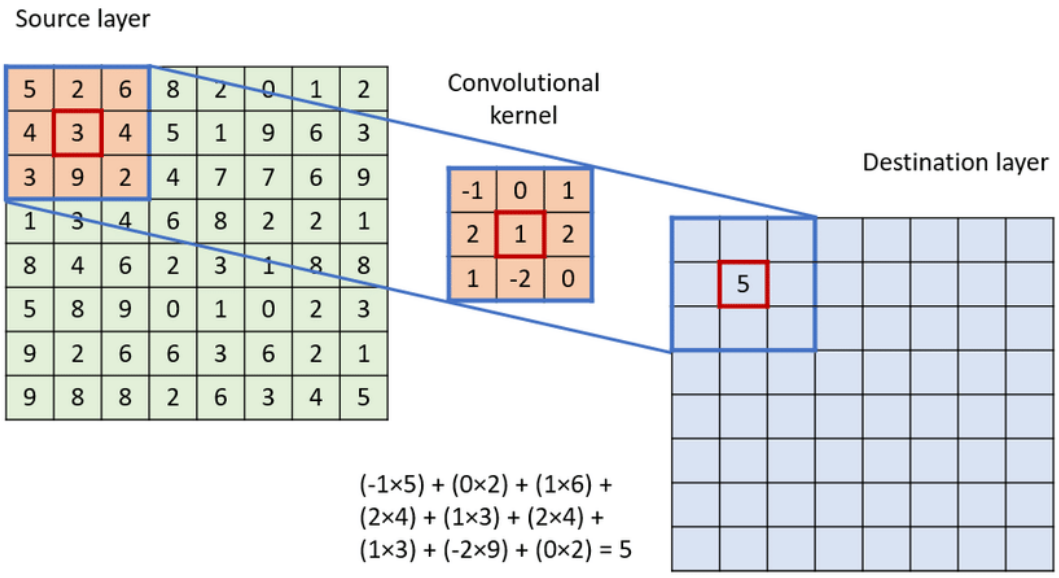

Figure 3.8: Illustration from Podareanu et al. (2019). Here, it shows a source layer, which is the layer that goes through the convolution and the destination layer that is the output of the convolution. The operation of the convolution for this example is shown here: it will take the source and maps the regions in the destination.

The filters have a size $(F_H \times F_W \times C)$, where C is always the same number of the

channels in the source (Figure 3.9). The number of channels in the destination is the number of filters. After the convolution of one filter, the destination will be a matrix with $C = 1$. If four filters make the convolution, the destination will be with $C = 4$. Each channel is responsible for the convolution in the respective channel: the channel $C = 1$ of the filter will be responsible for the convolution of the channel $C = 1$ of the source.

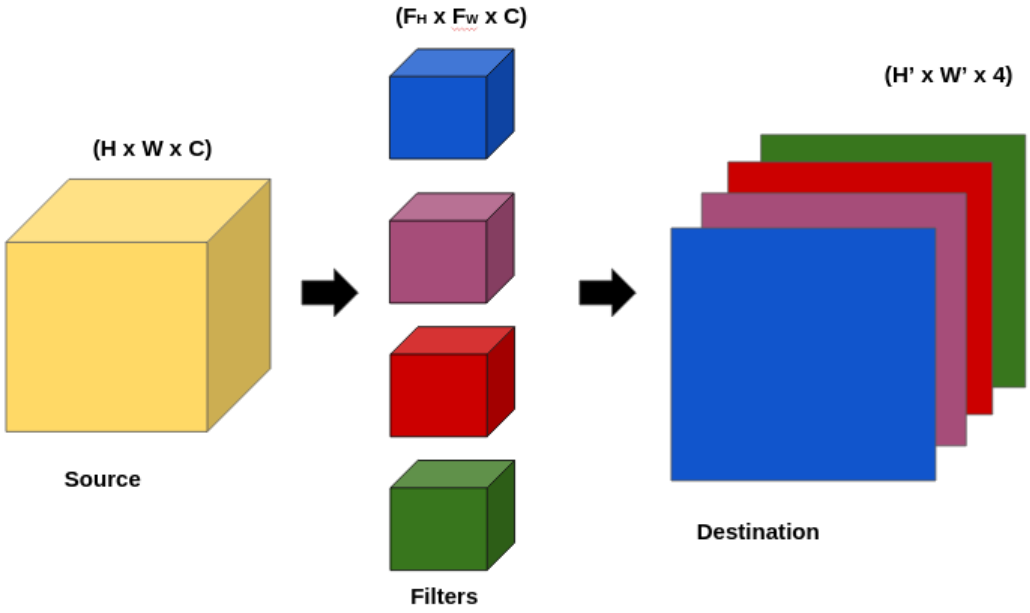

Figure 3.9: The yellow block is the source, and the four colors block is the destination block. All the filters have the same number of channels as the source. The destination block has the number of channels the same number of filters.

As we saw, the filters F goes through all the source S. We can control how they will perform the convolution. There are two properties called stride and padding. The stride controls how the filter will shift in the source, and the padding controls the size of the destination by adding zeros around the source. Typically, the strides shift 2 elements in the convolution, and the padding is determined to aim that the destination and source will be the same size after the convolution. The size of the destination $[H', W', C']$ after a convolution is given by:

$$[H', W', C'] = \left[\frac{H + 2P - F_H}{N_{stride}}, \frac{W + 2P - F_W}{N_{stride}}, N_{filters}\right], \tag{3.15}$$

where H and W are the height and width of the source, F_H and F_W are the height and width of the filters, P is the padding number (the number of rows and columns adding zeros), N_{stride} is how much it should shift and $N_{filters}$ is the number of filters.

- Learning Process

The forward propagation of the CNNs is similar to the one previously explained for NNs. The difference is that instead of a linear combination of the weights w, the operation is the convolution and the same filter is applied in all the input's points. The elements of the matrix $F[k,l]$ in Equation 3.14 represent the weights. Precisely, the filter $F[k,l]$ has two learnable parameters as the NNs: the weights $w_{(k,l)}$ and the bias $b_{(k,l)}$, where (k,l) is the element in the filter f. The forward propagation is the following:

$$z^l_{(i,j)} = \sum_k \sum_l w_{(k,l)} \cdot \sigma^{l-1}_{(i-k,j-l)} + b_{(k,l)} \tag{3.16}$$

$$\sigma^l_{(i,j)} = f(z^l_{(i,j)}). \tag{3.17}$$

The notation here will be:

- $z^l_{(i,j)}$ is the result of the convolution.
- $w_{(k,l)}$ and $b_{(k,l)}$ are the learnable weights and bias. They are the elements of the filter.
- $\sigma^l_{(i,j)}$ is the output of the layer l.
- $\sigma^{l-1}_{(i-k,j-l)}$ is the output of the layer $l-1$.
- $f(\cdot)$ is the non-linear activation function.

The process is similar to NNs, with the activation functions being the same. In the output layer L, the model will compare the prediction $\hat{Y}$ with the target Y, as explained in Equations 3.4-3.6 using the function $\mathcal{C}$. The gradient descent method is similar as in NNs, as well as the update of the parameters $w_{(k,l)}$ and $b_{(k,l)}$:

$$w_{(k,l)} := w_{(k,l)} - \eta \frac{\partial \mathcal{C}}{\partial w_{(k,l)}}. \tag{3.18}$$

$$b_{(k,l)} := b_{(k,l)} - \eta \frac{\partial \mathcal{C}}{\partial b_{(k,l)}}. \tag{3.19}$$

The only layer that is learnable is the convolutional layer in our architecture. The MaxPool and UpSampling layers do not have parameters to learn. The learning process

described for NNs and CNNs is the process of learning in our project. We use an architecture composed of convolutional layers. The activation functions are all ReLU functions, and we use Adam optimization already described. In the next section, we will describe the general architecture we use in this project.

3.3 Architecture

Our architecture, called U-Net, is based on a U-shape architecture proposed by Ronneberger et al. (2015) for image segmentation. We choose this architecture because the input and output have the same dimensions and it connects the encoder and the decoder. In Figure 3.10, we illustrate our architecture. The architecture is divided in two parts: encoder and decoder.

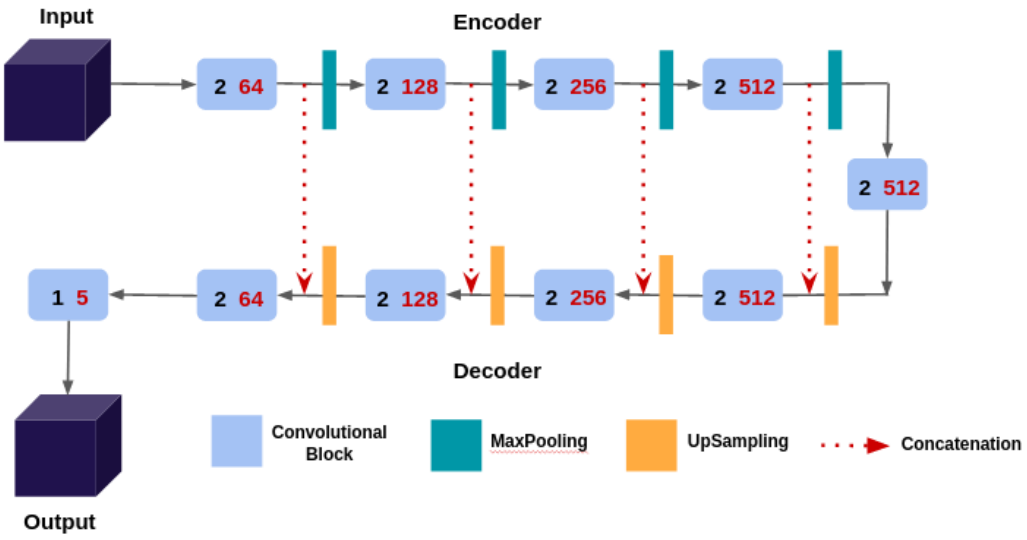

Figure 3.10: The scheme of the architecture U-Net based on Ronneberger et al. (2015). The light blue blocks are the convolutional blocks. The number in red is the number of filters in that block, and the number in black is how many convolutional layers, with 5×5 filters, are in that block. The green rectangles represent the MaxPooling layers after each convolutional block in the encoder part. The orange rectangles are the UpSampling layers before the convolutional blocks in the decoder part. The red arrow represents the concatenation between the convolutional blocks in the encoder with the convolutional blocks in the decoder. The last convolutional block is the final block, and it will output the prediction. It is the only one different block while the other ones are composed of ReLU activations, this one is composed of linear convolutional.

- Encoder

The encoder is the part of the architecture that is composed of blocks with MaxPool layers and Convolutional layers. The MaxPool layers act to resize the input aiming to low the computational cost and to keep only the relevant information of the model. The MaxPool is essential since the model can give attention to more irrelevant information, and it can affect the learning process. It corresponds to an arm of the U-shape.

In the encoder, the architecture will learn important features of the input data, and it will resize them until it reaches the smaller matrix. All the information learned by the encoder will go to the decoder, as shown in the concatenations in Figure 3.10. These concatenations provide that the decoder will remember the input data when it predicts the final output. Our problem is temporal, so the frame $N + 1$ depends on the frame N, so the model must predict new data while taking into consideration the past data.

- Decoder

The decoder is composed mainly of UpSampling and convolutional layers, but it also presents concatenations of the outputs of the encoder. Concatenations are when two arrays of the same size are connected by the same axis. In our case, the axis is the one corresponding to the channels. The decoder is the final part of the model, and it will predict output $\hat{Y}$ at its end so that it will compare with the target Y.

We built our model using Keras – a Python deep learning API – (Chollet et al., 2015) and Tensorflow – software library to build machine learning algorithm– (Abadi et al., 2016).

3.4 Hyperparameters

We choose some parameters as a feature called hyperparameters. Usually, hyperparameters do not depend on each other, so there are several possible combinations of them since they change from problem to problem. Even though they are defined beforehand, they can interfere indirectly with how the learning proceeds and its direction.

Despite changing from problem to problem and model to model, some hyperparameters are general and often used in all models. They are the batch size, learning rate η, and epochs, described in more details below:

- Batch size: it is the number of data points that will flow through the network together. If the dataset has N data points and the batch size is n, where $n < N$,

the model will take the first n data points and train the network, then it will take the next n data points and train again until it reaches N. The reason to use a batch size is due to computational cost and time: if $n = N$ it will have an expensive computational cost and if $n = 1$ it will take too long.

- Learning rate: it is the parameter that scales the magnitude of the gradient vector described in the gradient descent method.

- Epochs: it defines how many times the neural network will train. If epochs are equal to 1, the neural network will pass all the dataset only once, if epochs are equal to 100, the neural network will pass all the dataset 100 times. Too many epochs will take too much time and can cause the model to memorize the dataset causing an overfit, on the other hand if the number of epochs is low, the neural network may not learn and cause underfit.

Other common hyperparameters are the cost function, the number of layers in the architecture, the activation function, and the number of filters in each layer. In our model, we set the batch size, the loss function, and the weights of the loss function as hyperparameters. The epoch is not a hyperparameter in our problem since we use a technique called EarlyStopping (Prechelt, 1998). EarlyStopping is a tool that evaluates the learning process during training. After each epoch, it uses the value of the loss function in this epoch and compares it with previous values. If, after ten epochs, the loss values do not decrease, then the EarlyStopping stops the training. We need to avoid overfitting, which is when the model fits well for a single data set.

Finding the best hyperparameters is a problem since they may not depend on each other. For hyperparameter optimization, we use a method called a *grid search*. The grid search takes a set of the hyperparameters and a metric. For each combination, the model is trained and the metric is computed. The grid search will find the best hyperparameters after comparing the metric values of the validation set.

In our problem, the grid search is implemented in a shell script that takes combinations of hyperparameters and trains the network. The script saves the loss along with the metrics in a text file after all of them are trained. The best set is the one with the best values of the metric on the validation set, as described above.

3.5 Inference

After the model trains, we need to test using the test set. The test set is an unseen set to the model during the training. It is new data to the model, and it needs to generalize what was learned. In our case, we want to see how much the model can generalize, so the inference, i.e., the predictions of the model to this unseen data, is an iteration between the predictions. In Figure 3.11, we show the scheme of the inference.

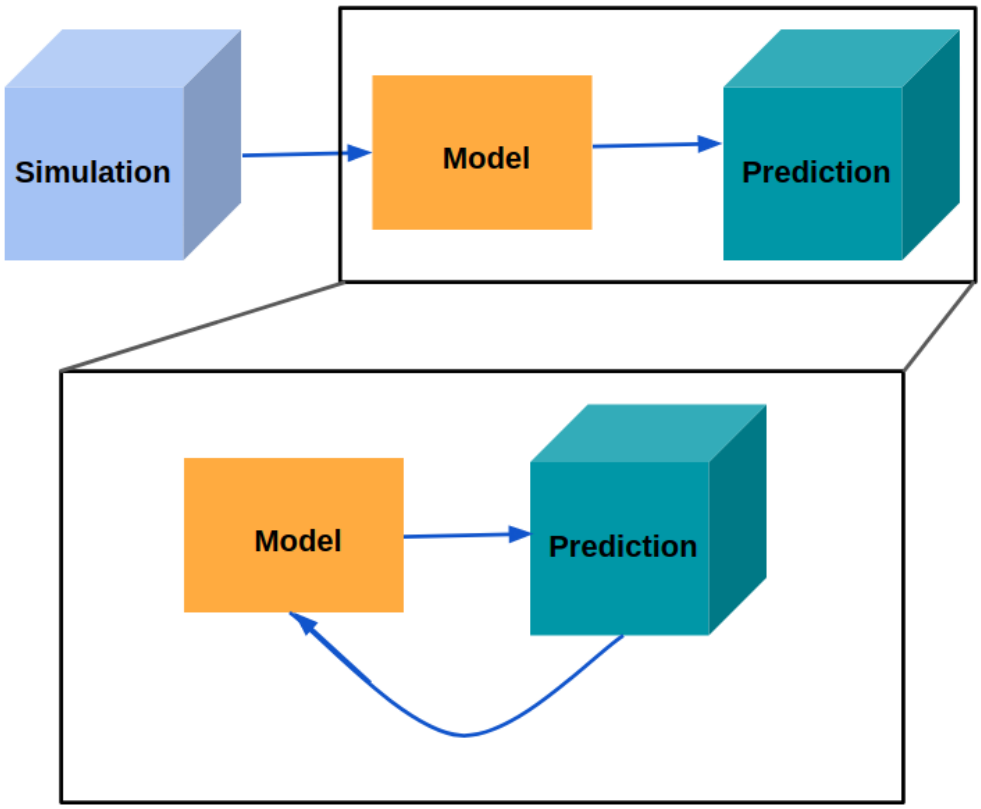

Figure 3.11: The first step is to feed the trained model with the first simulation data from the data set (light blue block). The model then generates output five steps ahead due to the way we structured the learning. We feed the prediction (green block) in the model, and then it outputs the next five frames. We iterate until we have the prediction equivalent to the last frame in the test set. The model only sees the simulation once.

We stress that we trained our model to predict five frames in the future, i.e., $\Delta t = 989.85\,GM/c^3$ times units in the future. We analyze the performance of the model when it predicts directly from the simulation (Figure 3.12).

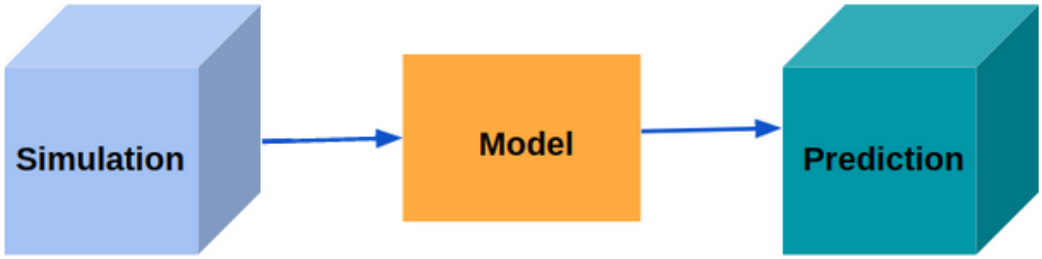

Figure 3.12: In order to analyze the proper training, we need to see how the model behaves after it predicts directly from the simulation data. The difference with respect to the previous test is that this one aims at analyzing the training itself, whereas the previous one aims at evaluating generalization.

3.6 GANs

Generative Adversarial Networks (GANs) are a model of DL proposed by Goodfellow et al. (2014). GANs consist of two adversarial neural networks: generator and discriminator. The generator receives a noise and generates an output while the discriminator is to learn what is real and what is fake — generated by the generator. The GANs follows a min-max function (Goodfellow et al., 2014). The generator's goal is to maximize the probability of the discriminator making a mistake. Meanwhile, the discriminator's goal is to get it right if the input is real or fake.

Several fields are successfully using GANs such as particle physics (de Oliveira et al., 2017) and astrophysics (Schawinski et al., 2017). GANs may present a better performance than usual NNs, due to the discriminator. Usual NNs take a loss function to evaluate the output of the model, while the discriminator may act itself as the loss function. The discriminator learns its parameters based on the properties of the target — real or fake input.

Another class of GANs are the conditional GANs (cGANs). cGANs take a condition that the generator must follow in the learning procedure. Isola et al. (2016) shows different examples of cGANs, e.g., a cGAN takes as input a satellite image and generates the map, or it takes a daylight picture and produces the same picture in the night time. The text-to-image papers (Reed et al., 2016; Qiao et al., 2019) are also using cGANs to generate images from text, i.e., it takes a sentence, and the model outputs a picture from what the

sentence describes.

Based on pix2pix and text-to-image papers, we apply cGANs in our work. Instead of it receiveing the previous snapshots, we will feed the model with a vector with the properties of each snapshot, as in Section 4.2. The vector here takes the angular momentum profile $l(R)$, viscosity profile ν, α, and the gravitational time t_G (Figure 3.13). We normalize all the properties with a min-max normalization, except for the gravitational time. We normalize the gravitational time with a logarithm min-max. The architectures of both generator and discriminator are similar to (Isola et al., 2016) with generator being the decoder of our CNN.

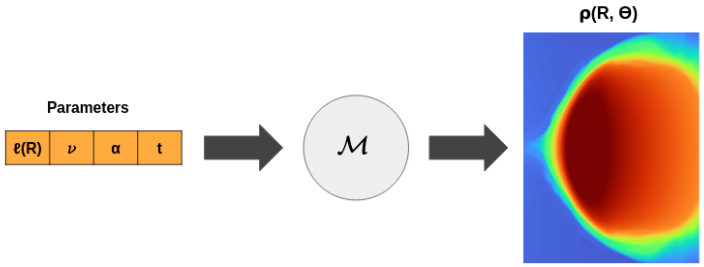

Figure 3.13: The pipeline of the cGAN model. The input is a vector x with four parameters, the model $\mathcal{M}$ is our trained generator, and the output is the density field $\rho(\theta, R)$ for each t.

3.7 Analysis

The output of the model needs to be compared with the target values (simulation data) to evaluate the model's performance, i.e., how well it can generalize and learn from observations. The analysis starts in the learning process with the metrics after each epoch.

- R^2 metric

The R^2 is commonly used for analyzing regression problems, since it is useful to compare the predictions with its ground truth. The R^2 is based on the variation of the data and gives a result on the similarity between the target and the prediction (Draper and Smith, 1966). It is also useful since the R^2 has an upper limit, i.e., when $R^2 = 1$, the prediction and the target are equal.

Here, the R^2 metric is used in the training part as an evaluation metric after each batch. R^2 is defined as:

$$R^2 = 1 - \frac{\sum_{ij}(P_{ij} - T_{ij})^2}{\sum_{ij}(P_{ij} - \overline{P_{ij}})^2}, \tag{3.20}$$

$$R^2_{ij} = 1 - \frac{(P_{ij} - T_{ij})^2}{(P_{ij} - \overline{P_{ij}})^2}, \tag{3.21}$$

where P is the prediction matrix, $\overline{P}$ is the mean of P, and T is the target matrix. The Equation 3.20 result is a real value and Equation 3.21 is a matrix the size of T and P.

- Δ, MAE and MAPE

We use the mean absolute error (MAE) and the difference (Δ) as other evaluation metrics, besides R^2.

The difference between the prediction P and the target T is $\Delta = |P - T|$. The total difference is $\sum_i \Delta$ for all data points i. MAE is then defined as:

$$MAE = \frac{\sum_{ij} |\Delta_{ij}|}{NM}, \tag{3.22}$$

where N and M is the size of the matrices.

We also use the mean absolute percentage error (MAPE):

$$MAPE = \frac{1}{NM} \sum_{ij} |\frac{\Delta_{ij}}{T_{ij}}|. \tag{3.23}$$

- 1D and 2D Mean

The mean density $\overline{\rho}$ in the plots gives us intuition about how the model is learning the density with respect to the target. We proposed an analysis of the mean in as a function of θ and function R. First, we fix a coordinate and calculate the mean through the other one. We show the equation of the density in terms of R (3.24) and in terms of θ (3.25), with N_θ being the size of the vector in θ direction and the same for N_R with the R vector size:

$$\overline{\rho}_R = \frac{\sum_\theta \rho_{R,\theta}}{N_\theta}, \tag{3.24}$$

$$\overline{\rho}_\theta = \frac{\sum_R \rho_{\theta,R}}{N_R}. \tag{3.25}$$

- Mass

Our model needs to learn the mass conservation from observations only, so it makes sense to use the mass as an additional metric. In other words, the mass will inform if the model learns mass conservation. We calculate the mass of the physical where we are training the model in spherical coordinates, as:

$$m = \int \rho dV = 2\pi \sum_{R} \sum_{\theta} \rho(R, \theta) R^2 sin(\theta) \Delta R \Delta \theta. \tag{3.26}$$

We define m_T as the mass of the target and m_P as the mass of the same region in the prediction.

3.8 Procedure

To summarize the steps involved in training our DL method:

- We prepare the data using the data preparation described in section 2.
- The architecture of the model is the U-Net described in figure 3.10.
- A shell script is created with the possible hyperparameter values.
- We run the shell script with all hyperparameters. The hyperparameters and the evaluation at the end of each training will be appended in a .txt file.
- With the best values of the evaluation using the validation set and R^2 metric, we run the inference described in figure 3.11 and figure 3.12.
- The inference will be analyzed with the methods described in section 3.7.

Chapter 4

Results

In this chapter, we describe the results of this project. We present the results after training with PNSS3 simulation in Section 4.1. We choose PNSS3 because it is one of the longest simulations and with more data available. PNSS3 presents a wide range of the density values $10^{-5} < \rho < 2$, so we customize a loss function (Equation 4.1) to embrace equally the different regions of the density map.

$$\mathcal{L} = \mathcal{L}_T + \alpha\mathcal{L}_{HD} + \beta\mathcal{L}_{AD} + \gamma\mathcal{L}_{TORUS} + \delta\mathcal{L}_{DIFF}, \tag{4.1}$$

where the weights $\alpha, \beta, \gamma, \delta$ are hyperparameters as well as the learning rate, and the batch size. Each loss term of the loss function is a MAE, where $\mathcal{L}_{HD}$, $\mathcal{L}_{TORUS}$, $\mathcal{L}_{DIFF}$, and $\mathcal{L}_{AD}$ corresponds to the higher density region, the torus region, the diffusion region, and the accretion disk region, respectively. We show each region in Figure 4.1.

In Section 4.2, we present the results of the training with all simulations. Now, we are dealing with different initial conditions for the torus, so we proposed a different loss function to deal with these differences:

$$\mathcal{L} = \mathcal{L}_{TOT} + a\mathcal{L}_{HD} + b\mathcal{L}_{LD}, \tag{4.2}$$

similarly to the previous loss for those data. $\mathcal{L}_{HD}$ is the loss for regions where $\rho > \rho_{hp}$, in the other regions the loss is $\mathcal{L}_{LD}$. a, b and ρ_{hp} are the hyperparameters of this problem. All the terms are MAE functions. The details of hyperparameters and the data splits are in Appendix A.

We also present the results of conditional generative adversarial networks (GANs) in Section 4.3. The discussion of the results will be presented in the Chapter 5.

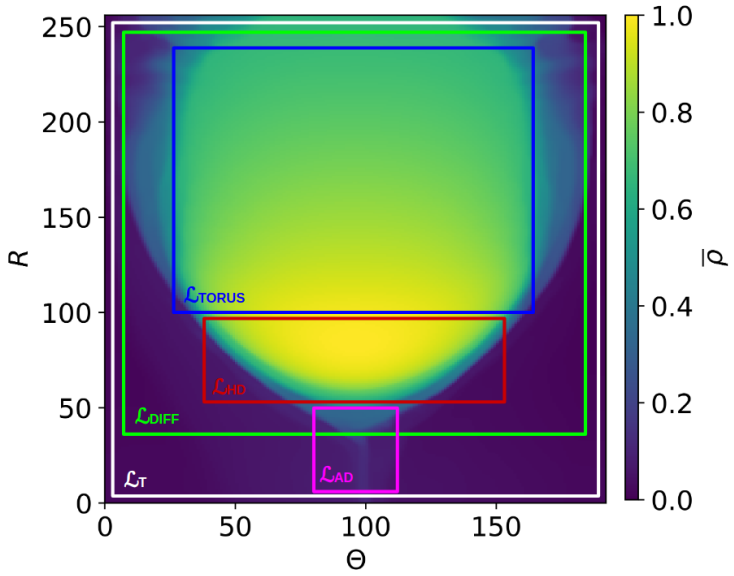

Figure 4.1: The regions of the loss function for the one simulation case. The $\mathcal{L}_{HD}$, $\mathcal{L}_{TORUS}$, $\mathcal{L}_{DIFF}$, and $\mathcal{L}_{AD}$ are associated to the higher density region, the torus region, the diffusion region, and the accretion disk region, respectively. We also have the entire region given by $\mathcal{L}_T$. All losses are MAE functions.

4.1 One simulation

We split the forecast into two different kinds: the direct and the iterative prediction. The direct predictions result after we feed a simulation frame to the trained model, and the iterative is when we iterate the predictions. In figure 4.2, we show a scheme of both kinds of forecasting.

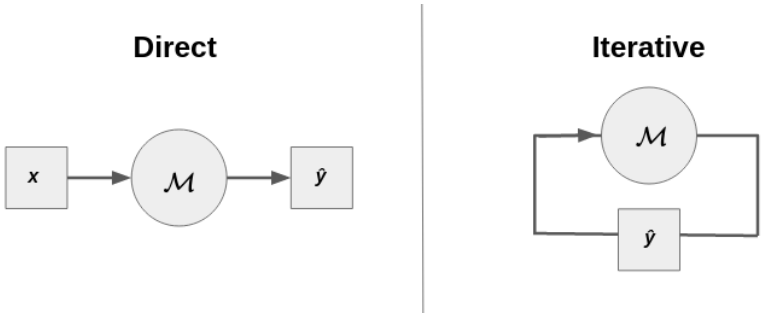

Figure 4.2: Two kinds of forecasting: direct (on the left) and iterative (on the right). The $\mathcal{M}$ is the model already trained, x is the snapshot from simulation, and $\hat{y}$ is the prediction.

- *Direct predictions*

Figure 4.3 shows the 2D comparison between the target and the first prediction after the ending of the training set – equivalent to gravitational time $t = 611529\ GM/c^3$. The

metrics between the two plots are $R^2 = 0.9988$ and $MAPE = 2.8\%$. It shows that the model could predict the torus form respecting the density values. We also calculated the MAPE between this prediction with previous snapshots (Figure 4.4) to show that the model does not memorize the dataset. Indeed, the MAPE is higher when calculated between this prediction and the previous snapshots. Figure 4.5 shows the mean density as a function of R and θ, indicating that the prediction from the model reproduces well the target at $t = 611529\ GM/c^3$.

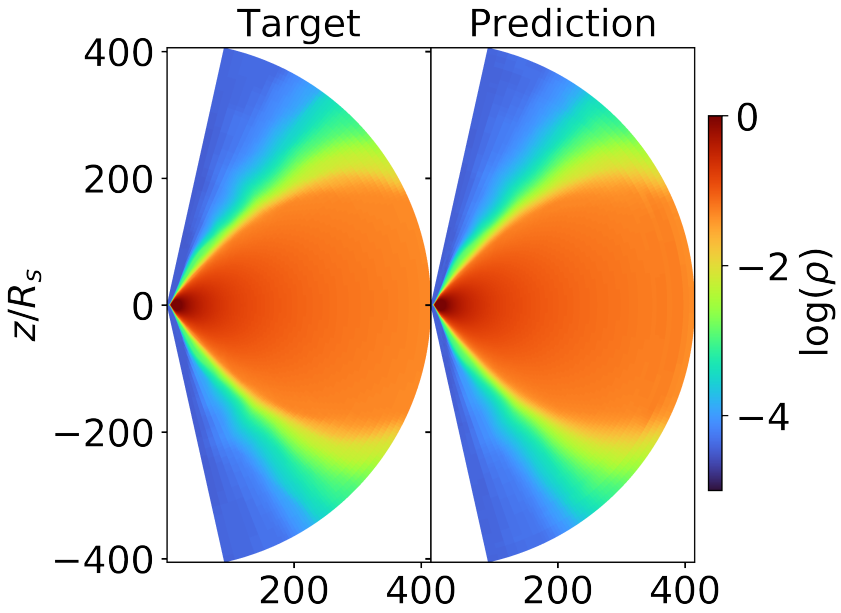

Figure 4.3: The logarithm of the density of prediction corresponding to the snapshot with gravitational time ($t = 611529\ GM/c^3$) compared with the respective target. The metrics between the two plots are $R^2 = 0.9988$ and $MAPE = 2.8\%$.

We show in Figure 4.6 the MAPE and difference between the target and the prediction at $t = 611529\ GM/c^3$. Both plots do not present discrepancies in the torus region except a slight $MAPE \sim 0.0005\%$ and $\Delta log(\rho) \sim 0.1$ difference in outer radii. We can relate this difference with the lower definition in the outer radii causing the model to learn the large grid leading to numerical errors. There is a discrepancy at smaller radii near the event horizon, especially in the higher values of θ. Still, the error in the inner radii has a MAPE lower than 0.0005% but with a difference of $10^{0.1}$. These errors do not really challenge our model, but they can lead to an accumulating error in the iterative forecast. It will lead to the interpretation that the model does predict the accretion of the black hole well. However, the model can predict the torus dynamics at larger radii with accuracy, and even

though the atmosphere part presents discrepancies, it is sufficiently small.

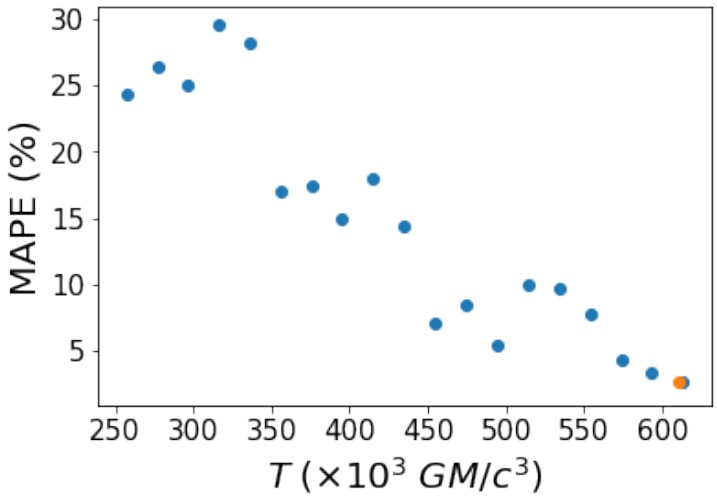

Figure 4.4: The MAPE metric between the prediction respective to $t = 611529\ GM/c^3$ and other predictions of the same simulation (blue dots) and with the respective ground truth (orange dot). We aim to show here that there are differences between the frame $t = 611529\ GM/c^3$ and previous ones, i.e., the model does not memorize the dataset.

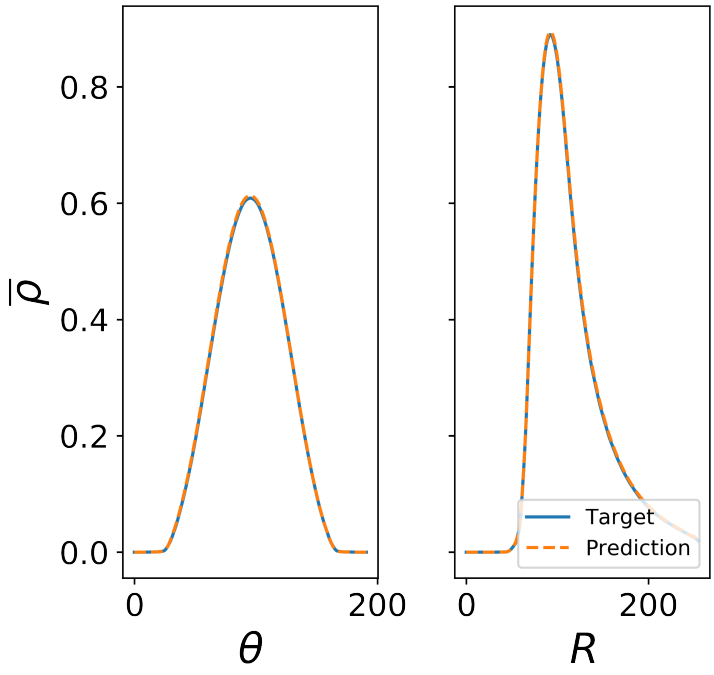

Figure 4.5: The mean density as a function of Θ (left panel) and R (right panel). The prediction fits perfectly the ground truth in the mean. The orange dashed line represents the prediction, while the solid blue line represents the ground truth.

To show the model has similar results at all time t, we fed the whole test set completely

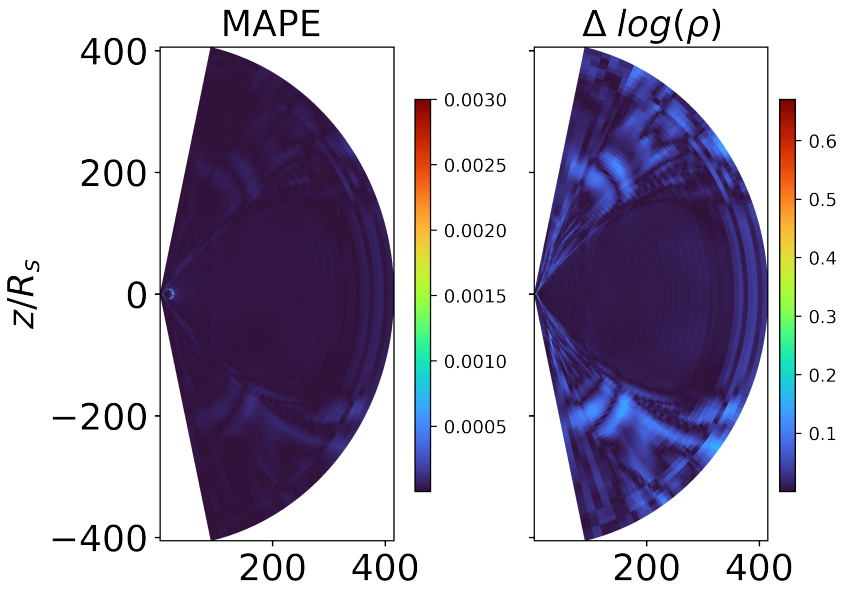

Figure 4.6: The left panel shows the mean absolute percentage error (MAPE) between the prediction and the ground truth. The right panel shows the difference of the logarithm of the ground truth and prediction ($\Delta log(\rho)$).

in the trained model and obtained the predictions, displayed in Figure 4.7. The mean error between the predictions and their ground truth is $\overline{\text{MAPE}} = 2.17\%$.

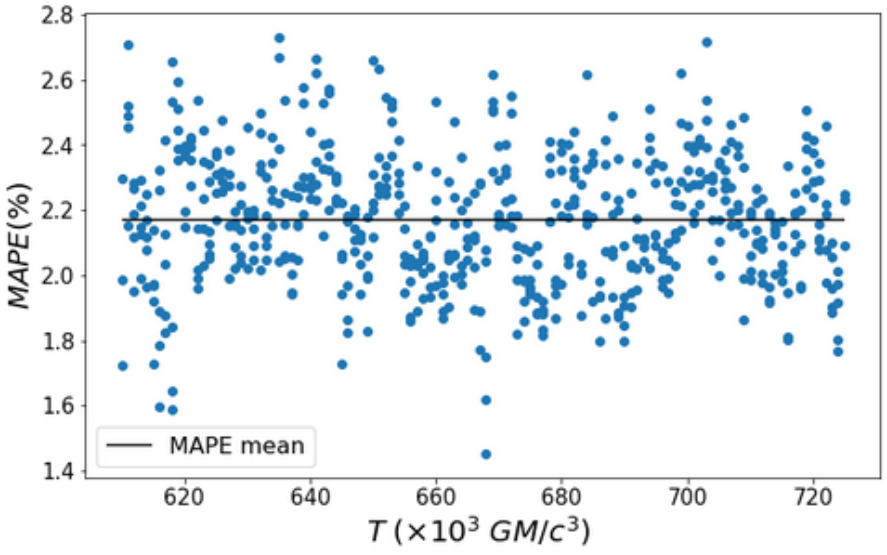

Figure 4.7: The MAPE between the prediction and its own ground truth (blue dots). The solid black line is the mean of the MAPE of all blue dots, $\overline{MAPE} = 2.17\%$. We have the maximum error $max(MAPE) = 2.73\%$ and the minimum error is $min(MAPE) = 0.07\%$.

In Figure 4.8, we compare the mass with the targets. The highest difference is $\sim 1\%$.

Here, we are computing the mass of the flow in a subset of the original full grid, therefore the computed mass is not expected to be conserved to BH accretion and inflow of the gas from outside the region we chose.

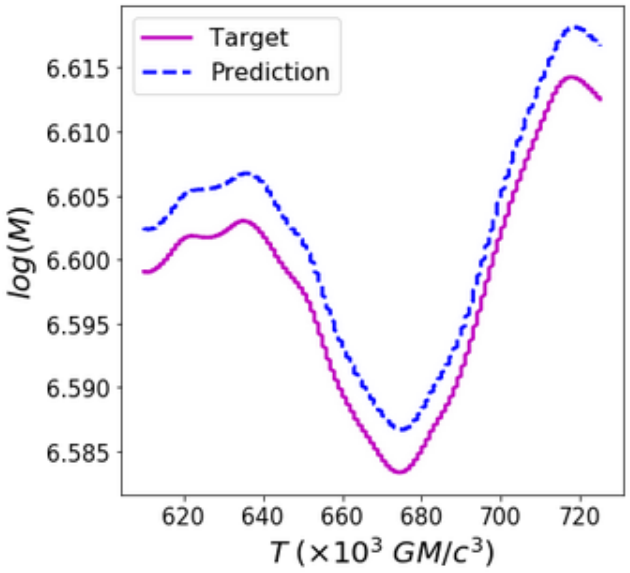

Figure 4.8: The comparison between the mass in the ground truth and the mass in the prediction. The plot shows that the model can learn the mass in the system within a difference of $\sim 1\%$.

- Iterative predictions

We want to test how much the model can generalize by itself. Instead of feeding the simulations in the model, we feed one simulation (the frame equivalent to the gravitational time $t = 610539\ GM/c^3$) and we iterate the predictions. Our goal here is to understand until what point in time the model can generalize well from what is has learned.

In Figure 4.9, we show the results after iterating the predictions 10 steps, 50 steps, and 100 steps in the future, equivalent to $\Delta t = 612519\ GM/c^3$, $\Delta t = 660032\ GM/c^3$ and $\Delta t = 709524\ GM/c^3$, respectively. After each iteration, the error is increasing, and it becomes evident near the event horizon region. In the previous analysis, we see that the model's prediction is lower in that region, so this error accumulates. Another issue is that the predictions distort the torus, i.e., the model has too much diffusion near the boundaries of the torus. An interesting aspect is that the model's results still maintain globally consistent densities: the densities decreases outwards. Larger R have lower spatial resolution and the model learned this.

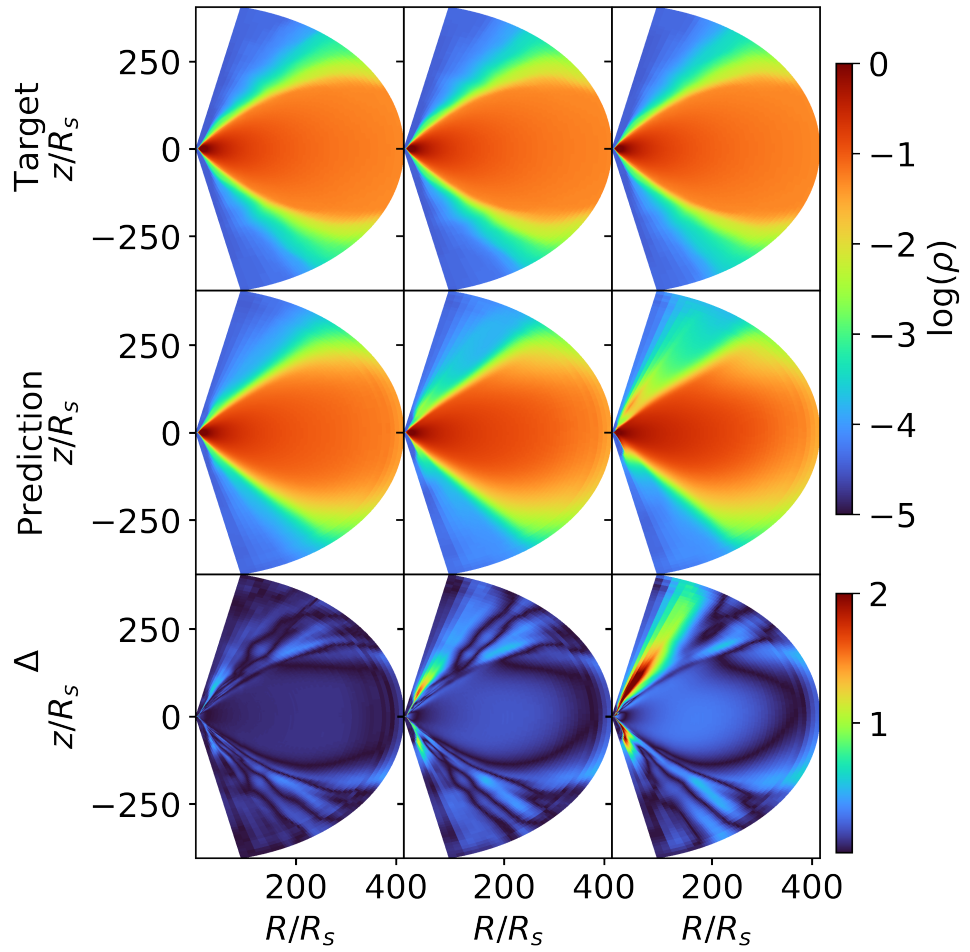

Figure 4.9: The density state for three distinct times. The first one is 10 steps in the future, the second one is 50 steps, and the third one is 100 steps. The Δt is close to $\sim 100000\ GM/c^3$. The first row shows the target while in the second row is the respective prediction. In the last row, we plot the difference between the first two ones.

In Figure 4.10, we show the mean density as a function of time. The model predicts well the mean density until $t \sim 680000\ GM/c^3$. After that point, the density starts increasing exponentially at smaller radii and near the event horizon. This may indicate that the model is not learning well the mass accretion rate. Since the model cannot deal with larger values of the density, it increases the density in that region.

We plot the mass evolution in figure 4.11. While the region with $R > 50R_s$ increases the mass slowly, the one at $R < 50R_s$ exponentially increases the mass. We compare the prediction with the target for all the grid in the right panel. While it preserves the mass, in the beginning, it rapidly exploded after $t \sim 680000\ GM/c^3$, the same point the model fails in the inner radii. Figure 4.12 shows the MAE between the mass comparison.

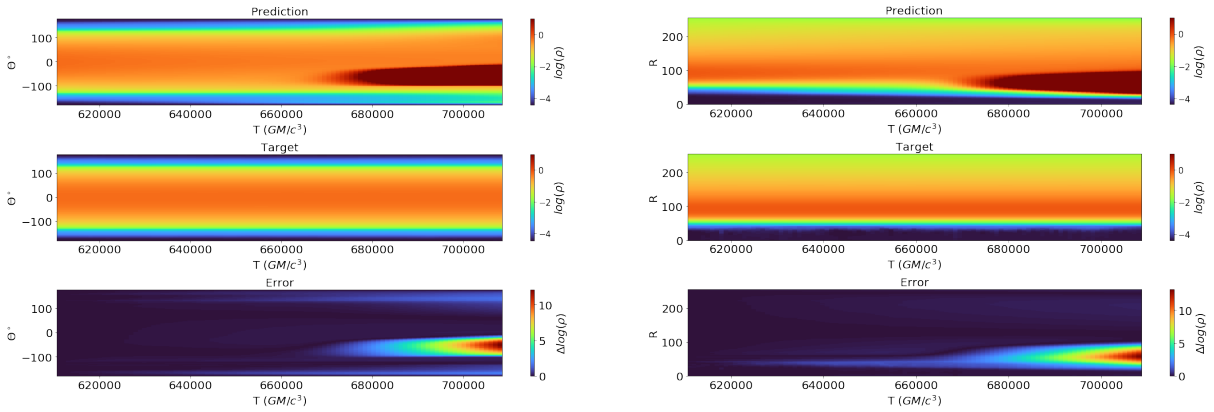

Figure 4.10: The mean density in θ (left panel) and R (right panel) in the function of the time. The model well predicts the mean until $t \sim 680000\ GM/c^3$. The error is $< 20\%$ until the region near the event horizon exploded.

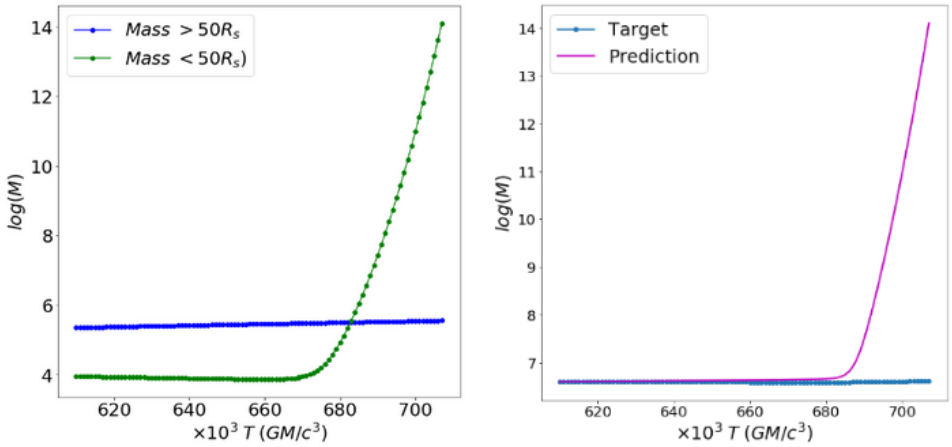

Figure 4.11: In the first panel we show the mass in two regions: $R < 50R_s$ and $R > 50R_s$. The region $R < 50R_s$ increases the mass, the mass is not conserved in these regions. The model does not fit the high region densities well. Meanwhile, the region $R > 50R_s$ has a slow increase in the mass. The comparison of the prediction with the target for all the grid is in the right panel.

4.2 All simulations

Now, we will apply the same analysis to the model trained on all simulations of (Almeida and Nemmen, 2020). We give new input to the model based on the treatment in Wang et al. (2020). As in Wang et al. (2020), we concatenate the vector $\vec{x}$ at the start of the decoder part. The $\vec{x}$ will have information on the angular momentum profile, viscosity, and α. The vector $\vec{x}$ is defined as:

$$\vec{x} = (A, B, C)$$

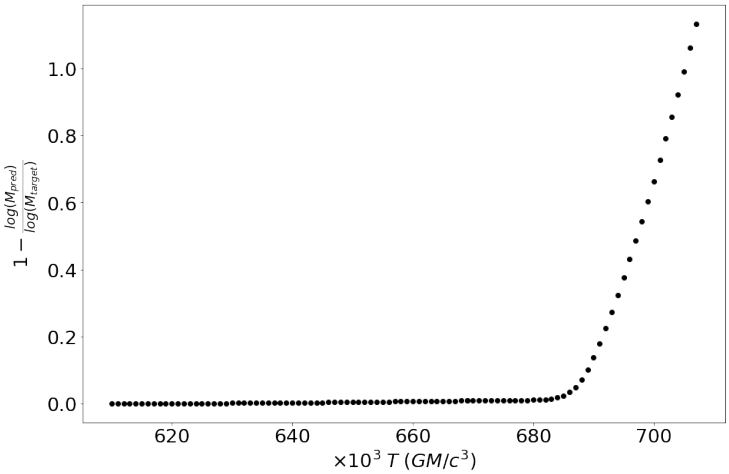

Figure 4.12: The MAE between the predictions and the respective ground truth. The error increases slightly until it reaches $t \sim 680000\ GM/c^3$. After $t \sim 680000\ GM/c^3$, the MAE increases indicating the model increases the density values.

where the following rules set the values of A, B and C:

- If the angular momentum profile is PN then $A = 0$, if it is PL then $A = 1$.
- If the viscosity profile is ST then $B = 0$, if it is SS then $B = 1$.
- The α is normalized: if α is 0.3 then $C = 1$, if α is 0.1 then $C = 0.3$ and if α is 0.01 then $C = 0$.

In Table 4.1, we show the simulations, their parameters and the cropped number of snapshots. We crop the dataset such that all simulations are with the similar number of frames, and then the model sees all simulations somehow equally.

Table 4.1 - The simulations with the parameters: angular momentum profile, viscosity profile, and the alpha parameter. In the last column, we show the number of frames used in the training of each simulation dataset.

Name	$l(R)$	ν	α	# Frames
PNST01	PN	ST	0.01	991
PNST1	PN	ST	0.1	347
PNSS1	PN	SS	0.1	990
PNSS3	PN	SS	0.3	990
PL0ST1	PL	ST	0.1	171
PL0SS3	PL	SS	0.3	709

Table 4.1 - Continuation

Name	$l(R)$	ν	α	# Frames
PL2SS1	PL	SS	0.1	763
PL2SS3	PL	SS	0.3	763

In the training part, we feed all simulations with the output being the five frames ahead except for PL0SS3. We hid the PL0SS3 to see how the model would perform with unseen parameters and unseen data. As in the previous section, we will separate this in two subsections: the direct predictions and iterative predictions. The architecture is still the same except for the $\vec{x}$ input described above.

- Direct predictions

In Figure 4.13 and Figure 4.14, we show the predictions after we feed the model with a frame of 10 steps after the end of each training set. While the model can predict and learn well the systems that have a stationary behavior, it fails in predicting large variability. For instance, PNST1 and PNST01 are both simulations showing high variability, so this model fails to learn a feature that changes continuously. However, the model can learn the torus geometry and density behavior well. It can predict a large torus with a higher density (PL2SS3) and a small torus with a gradient of density values (PNSS3). To simplify the analysis, we will now discuss the results of PNST1 and the PL0SS3, since the first one shows a significant difference and the second one is the simulation the model did not see before. Given that PL0SS3 was not part of the training set, by analysing it we can understand the generalization power of the model.

Figure 4.15 shows the difference between the $log(\rho)$ of the target and prediction for 10, 25, and 50 frames in the future. PL0SS3 does not present any prominent discrepancy even though the model does not learn with it. PNST1 shows higher differences due to the turbulent nature of this system. However, the largest disparity in the PNST1 is in the order of $10^{0.2}$ in the boundary of the torus. In Figure 4.16, we show the the mass in function of time for PNST1 and PL0SS3. To predict PL0SS3 increases the mass by a factor of $10^{0.02}$ and besides the PNST1 presenting turbulence, the model can understand how the mass varies.

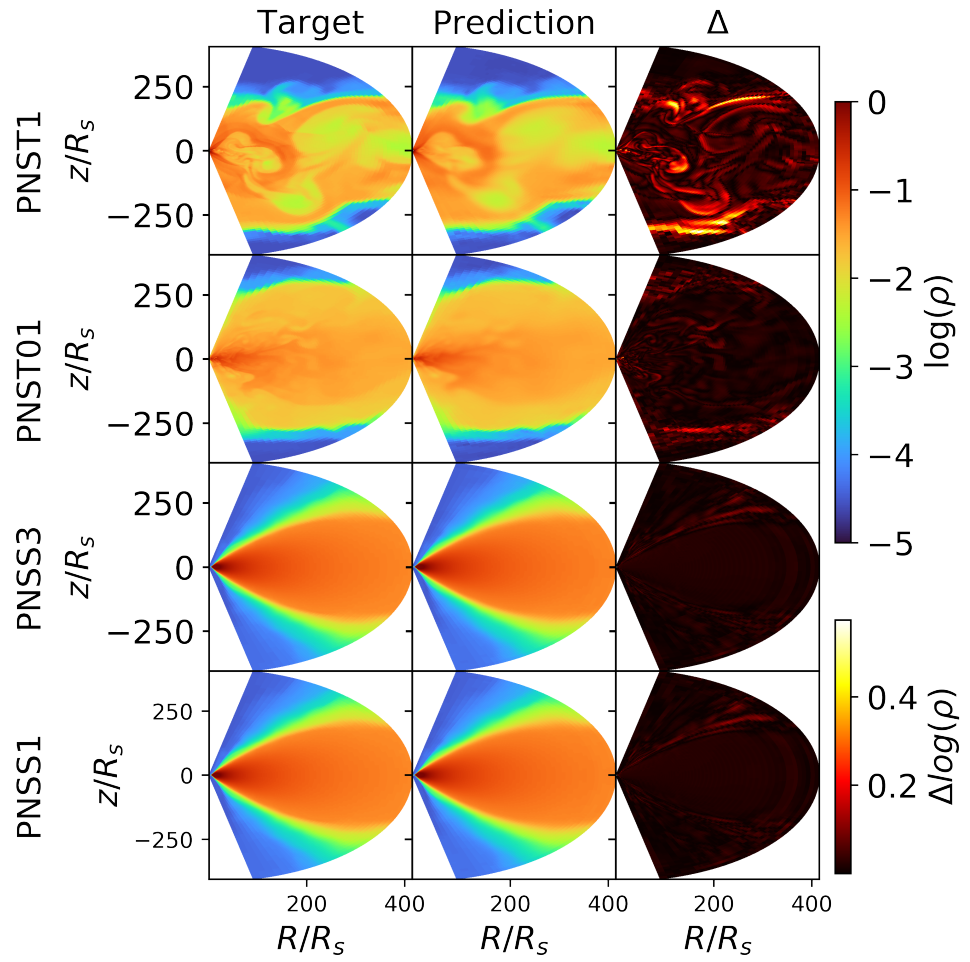

Figure 4.13: This plot is the 10th step after the training stop for all simulations. Each simulation has different gravitational time in the 10th step, but the $\Delta t = 1979.7\ GM/c^3$ is the same for all of them. The density profile of the target and the prediction as well as the difference $\Delta = |\text{Target} - \text{Prediction}|$ plot of four systems. The PNSS3 system is the one that we described alone in the previous section.

The model can predict the PL0SS3 well even this simulation was not fed during the learning process. For this reason and to quantify how much it generalizes, we will present the results in the iterative process for PL0SS3.

- Iterative process

Figure 4.17 shows the predictions of the PL0SS3 for gravitational time 50680 GM/c^3, 65528 GM/c^3, and 90274 GM/c^3, respectively. The difference is prominent for higher values of θ, and lower radii. Though, it predicts the torus without any particular discrepancy between the target and the prediction. The model extrapolates the region near the black hole's event horizon, it decreases the density, and it causes the mass to distinguish, as

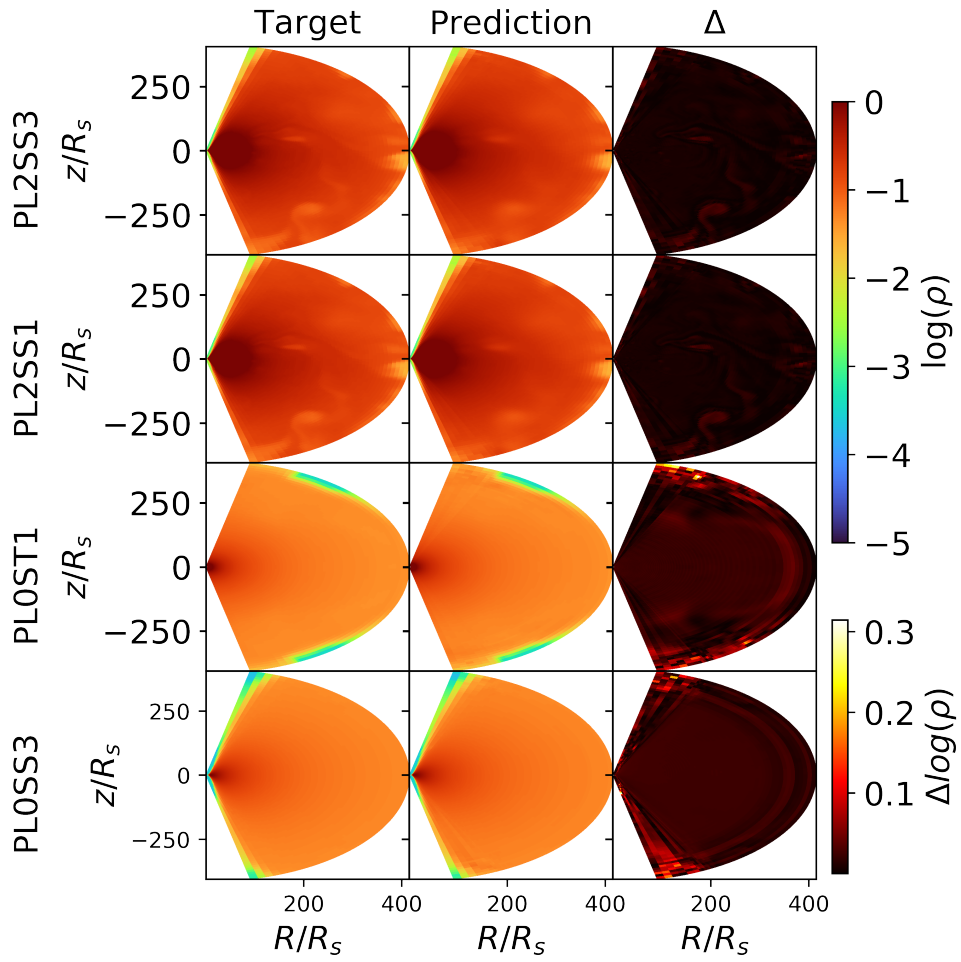

Figure 4.14: The continuation of 4.13 plot. Each simulation has different gravitational time in the 10th step, but the $\Delta t = 1979.7\ GM/c^3$ is the same for all of them. The density profile of the target and the prediction as well as the difference $\Delta = |\text{Target} - \text{Prediction}|$ plot. The PL0SS3 was not fed to the model in the training part, so the model is predicting it after learning the other seven simulations with their respective parameters.

shown in Figure 4.18. The reason is also that the model did not learn from thick torus like PL0SS3. Still, the problem is similar to the one simulation problem: the model fails to predict near the black hole's event horizon. It indicates that we may need to constrain the accretion rate in future work.

We show in Figure 4.19 the density mean in θ direction and R in the function of time. The first panel shows the density mean for θ direction, indicating that the inconsistency is mostly for higher values of θ. The second panel shows that the inconsistency is for smaller radii suggesting that the model fails when predicting near the black hole. It may be due to be a small region in comparison with the rest of the system.

However, we see the model fails in some regions and with turbulent simulations. One

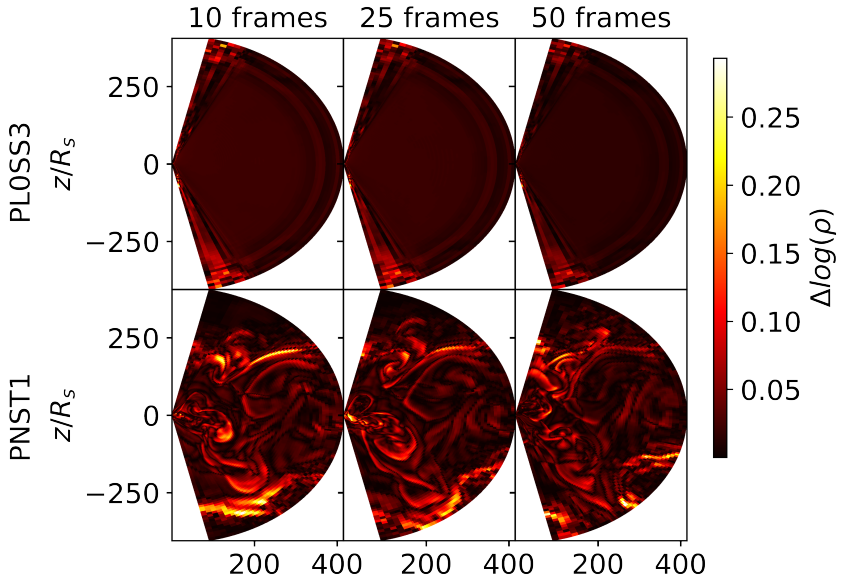

Figure 4.15: The differences for 10, 25 and 50 frames in the future for PL0SS3 and PNST1.

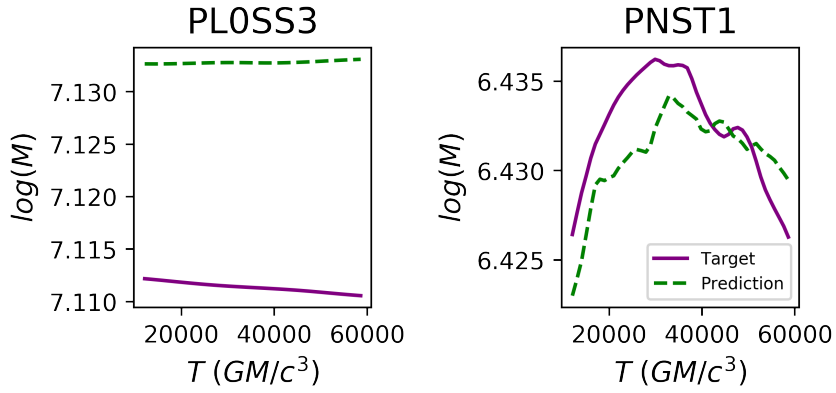

Figure 4.16: The mass analysis of PL0SS3 and PNST1. The mass of PL0SS3 presents a difference of factor $10^{0.02}$. The PNST1 presents a similar curve between predictions and targets.

reason may be the loss function that not contemplate the complexity of the systems. We want to propose in the next section another DL method that may learn the ideal loss function for the problem. We will introduce the GANs as well as the results of using these models to our work.

4.3 GANs

Here, we present the preliminary results of the work with cGANs.

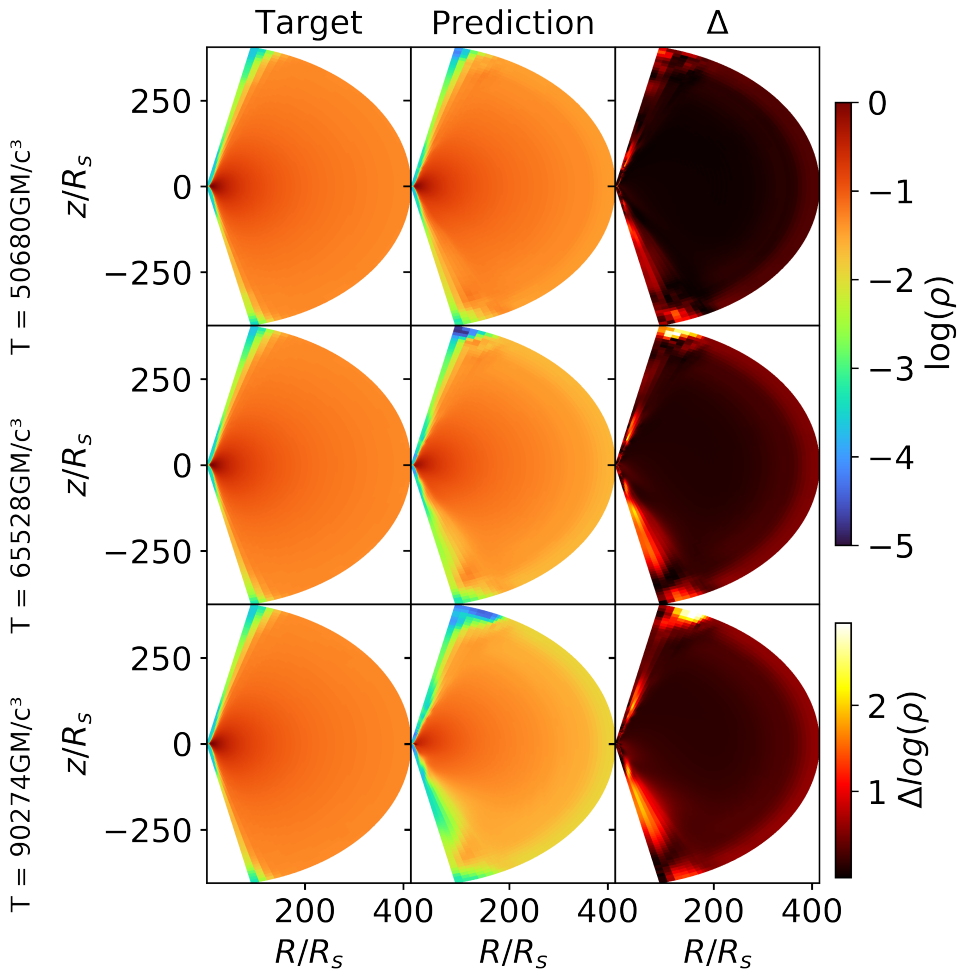

Figure 4.17: The predictions of the PL0SS3 using the iterative scheme. We have the 10, 25, and 50 steps, equivalent to times 50680 GM/c^3, 65528 GM/c^3, and 90274 GM/c^3 respectively. Δ correspons to the difference between the first two columns.

- *Data*

We use the same simulations as in Section 4.2, but instead of predicting five snapshots in the future, we feed a vector, and the cGAN generates the density field associated with that vector. The split is 80% for training, 10% for validation, and 10% for the test. Each snapshot has a vector of $l(R)$, ν, α and t. We made the inference by feeding the vector in the trained generator.

- *Results*

The results are from simulations PL0ST1, PL2SS3, PNSS3, PNST01, and PNST1. Figure 4.20 shows the results after at a time $\Delta t = 9898.5GM/c^3$ has passed. The model

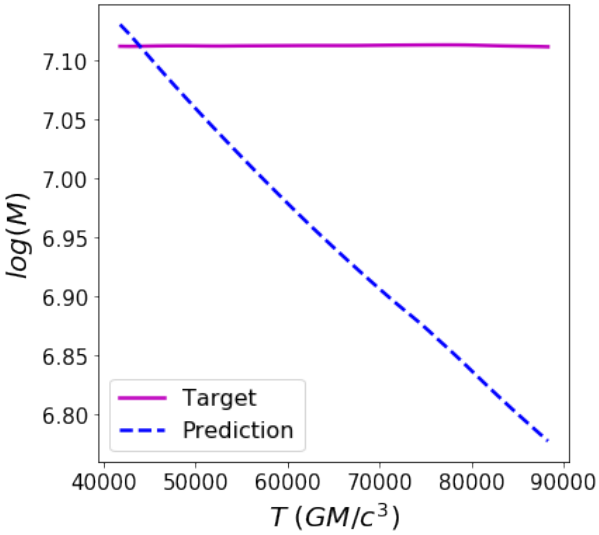

Figure 4.18: The mass of the target vs the mass of the predictions. The model does not predict the mass of the system well. It can be explained since the cropped region of all systems does not have mass conservation due to black hole accretion. The model extrapolates what it is learned.

predicts well, except for the PNST1, which shows higher variability than the others. We see the PNST1 presents higher values of $\Delta = |\text{Target} - \text{Prediction}|$ in the regions that shows outflows. The outflows changes rapidly through the simulations, becoming a hard-to-predict behavior for the model.

Figure 4.21 shows the mass of PNSS3 and PNST1. The cGAN misses the mass by a factor of $10^{0.015}$ in both systems. While the PNS3 presents a similar mass curve for prediction and target, the PNST1 shows a different curve. The model to predict the turbulent behaviour in PNST1.

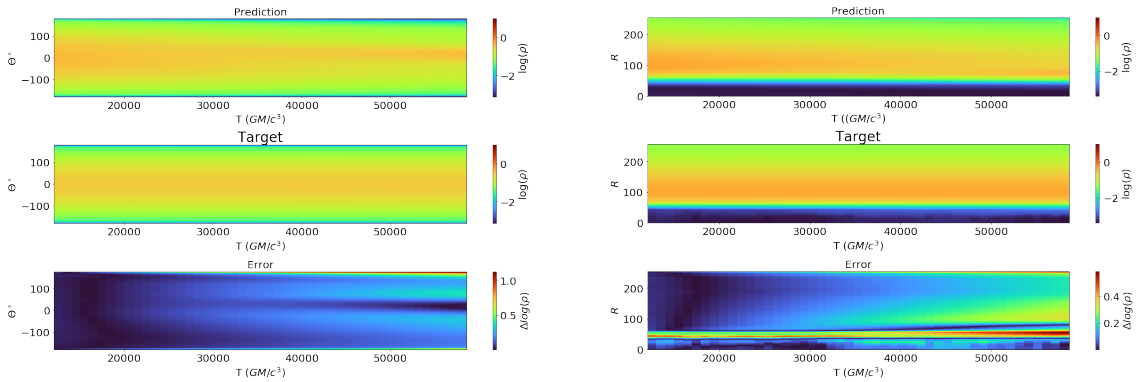

Figure 4.19: The mean of density in θ (left panel) and R (right panel) in the function of the time. The model presents problems while predicting for lower R and higher values of θ but it has better prediction values in comparison of figure 4.11.

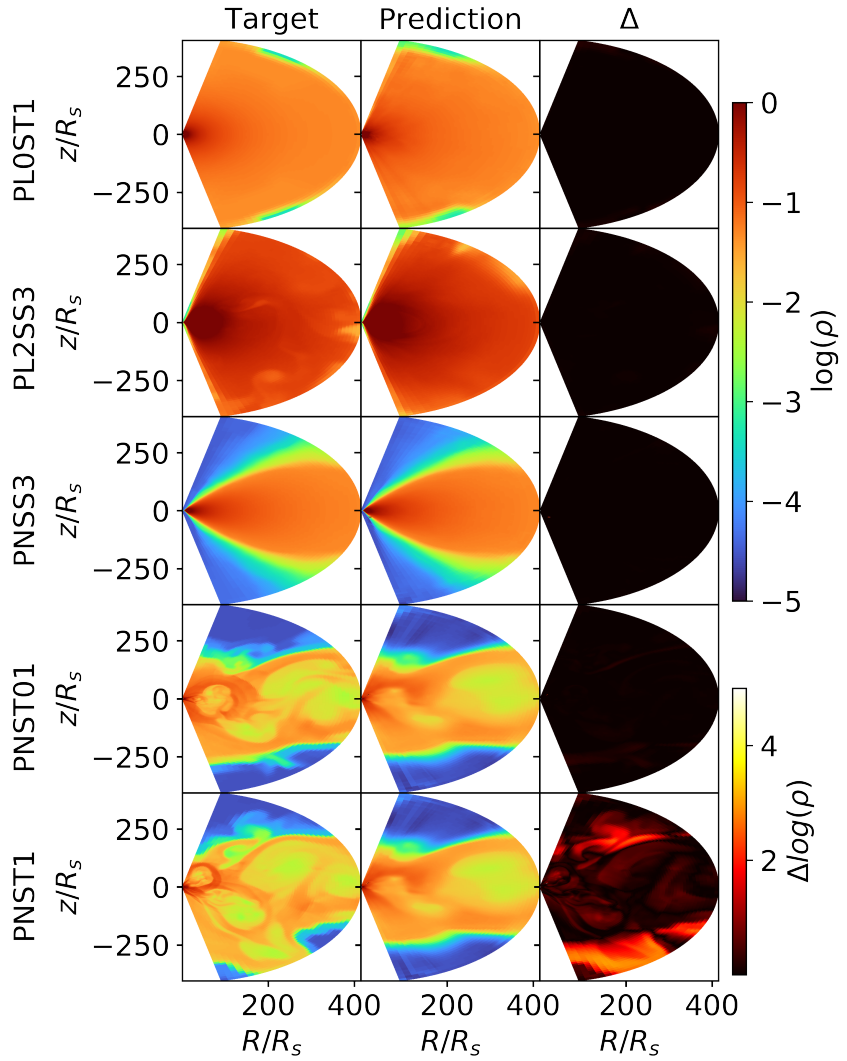

Figure 4.20: The results are from simulations PL0ST1, PL2SS3, PNSS3, PNST01, and PNST1 from the GAN model. All the results are from 50 steps in the future, equivalent to $\Delta t = 9898.5GM/c^3$ after the training set is over.

Figure 4.22 shows the PNSS3 mean density for θ and R directions in function of time. The differences in the plots are not as prominent as in the CNNs results. Since the cGAN receives as input the vector and does not have an iterative process, the errors do not sum up during the predictions. In other words, the error is the same for all the outputs since it does not depend on other predictions. We show the same analysis for the PNST1 in Figure 4.23. Even though the chaotic character of the system, the density mean of the prediction follows somehow the same profile of the target.

Since the cGANs can generate the density profiles right after receiving the parameters, this is a useful tool to obtain new density profiles with new settings without needing to

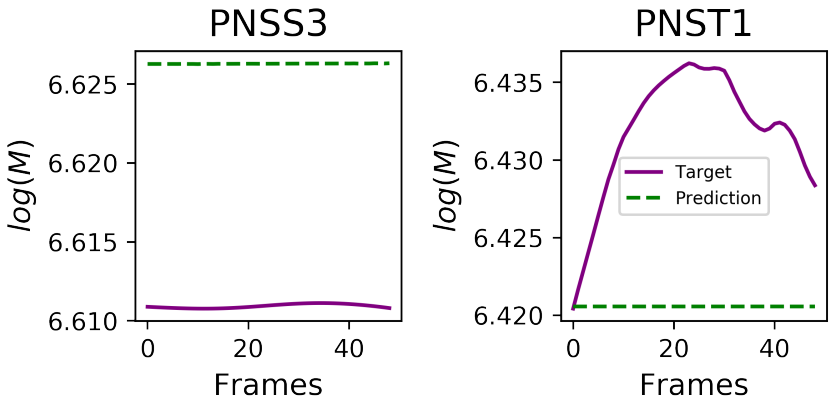

Figure 4.21: The panels shows the $log(M)$. The PNSS3 predictions follow the same behavior of the target with a difference of $10^{0.015}$. The PNST1 does not maintain the same behavior, but the highest contrast is also $10^{0.015}$.

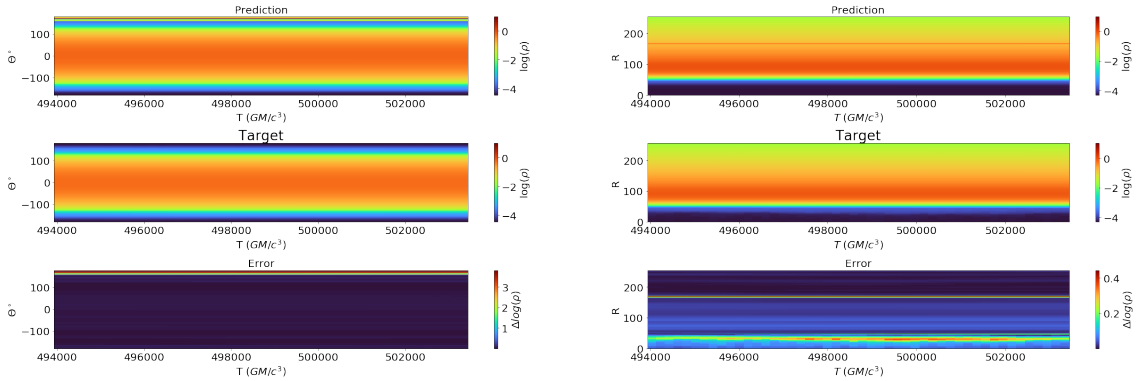

Figure 4.22: The PNSS3 density mean in θ (left panel) and R (right panel) in the function of the time for the cGAN. The cGAN's results are softer than the CNN's results. It does not present any significant contrast, as in the previous results.

simulate. The speed-up of the cGANs is similar to the CNNs. But the advantage is that since the next prediction does not depend on the previous one, the error does not accumulate during the predictions.

4.4 *Speed-Up*

Here we will expand the results of the speed-up achieved in our work. In Table 4.2, we show the resources used in our work to train the model and to predict.

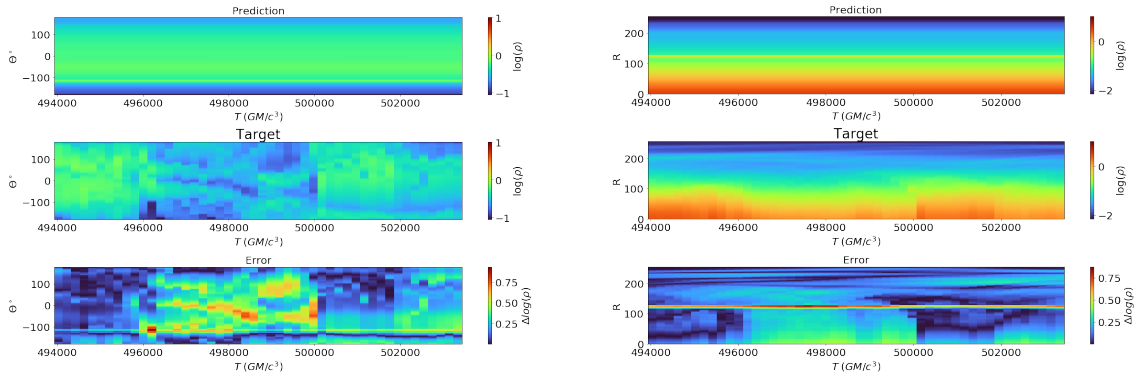

Figure 4.23: The PNST1 density mean in θ (left panel) and in R (right panel) in the function of the time for the cGAN. Even though it presents differences between the prediction – that is slightly softer than the target – the model still maintain the density mean profile.

Table 4.2 - The resources we use in our works are listed below.

Resource	TFLOPS	Cores
Quadro P6000	12.60	3840
Quadro GP100	10.34	3584

Almeida and Nemmen (2020) simulated the dataset using 400 cores and 7.6 TFLOPS from the AGUIA cluster containing Intel Xeon E7- 2870 CPUs. Each snapshot took 141 seconds to create in the numerical simulations.

Basing on the resources described, we will show the speed-up for the one simulation case and all simulations case.

- *One simulation*

The model for one simulation case can predict the system with physical accuracy until $t \sim 680000\ GM/c^3$, we want to explore the speed-up obtained using the DL method until that point. In other words, we will compare how much time each method takes to simulate until $t \sim 680000\ GM/c^3$. In Figure 4.24, we show the bars comparing the time of each method. In A is considering the training time, B is when we already have a trained model. In the case where we do not consider the training time, the speed-up here is $\sim 10^{4.5}\times$. If we have a trained model, the model will get in only 15 seconds the same results, that a simulation would take five days.

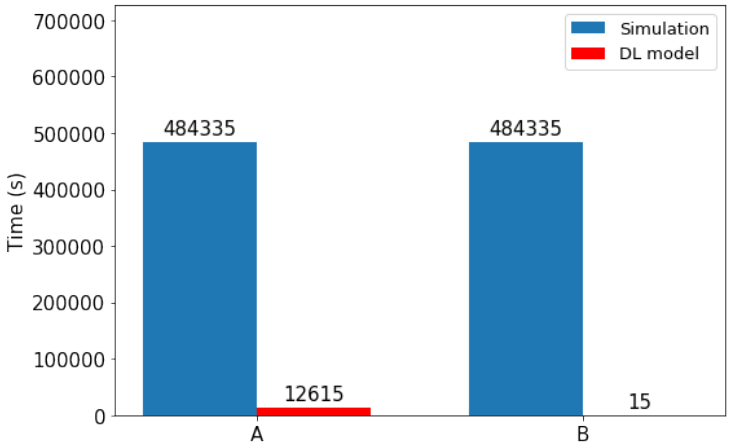

Figure 4.24: Duration of regular simulation and DL model. A shows the result when considering the training time, this is a 38x speed-up. If we have a trained model as in B, there is a 32289x speed-up.

Since the computational resources affect the running speed, it is useful to compare the CPU-hours and the FLOP between the DL and the simulation. We show the comparisons in both plots of the Figure 4.25. Like in the previous plot, A is the time considering the training time, B is when we already have a trained model.

- *All simulations*

With all simulations, the results of a prediction from a single system are similar to the previous ones. The new result here is to consider the speed-up of PL0SS3 since the model

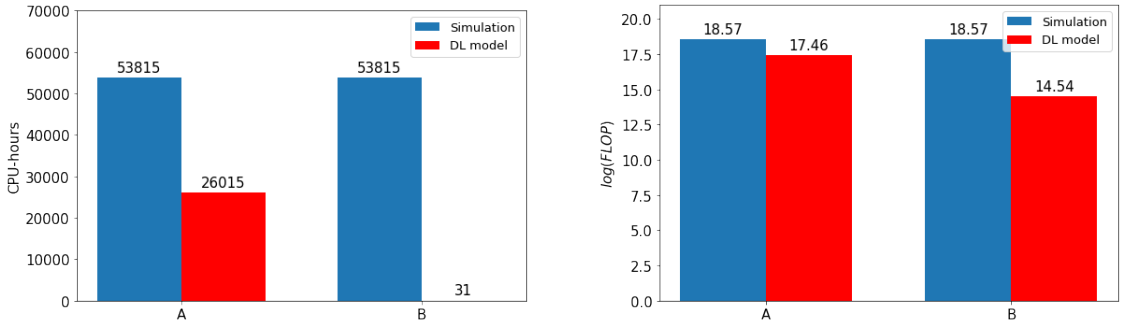

Figure 4.25: Temporal comparison the simulations in simulation and in DL model. In the first panel, we compare the CPU-hours between both methods. In the second panel, we compare the FLOP number between them. A means the comparison when considering the training time + test time and B is considering only the test time.

simulates it without seeing it before. In Figure 4.26, we compare the simulation time with DL time. A shows the comparison considering the training time, B without considering the training, and C is comparing only the PL0SS3 simulation if we have a model trained able to predict it alone. Figure 4.27 shows the results using CPU-hours and FLOP numbers.

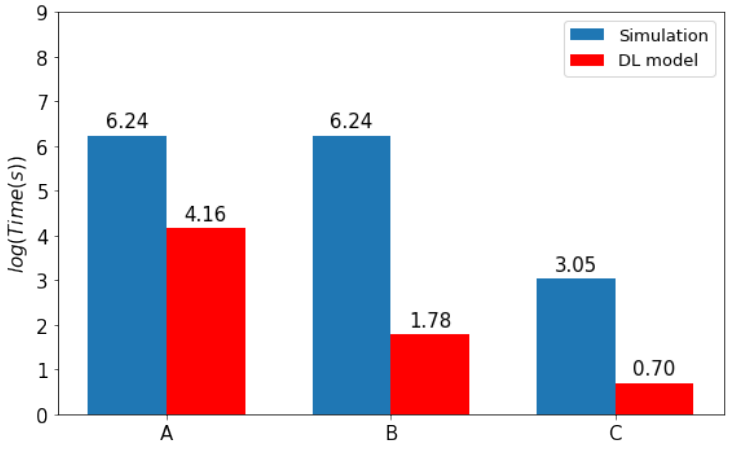

Figure 4.26: Duration of all simulations and DL model. A shows the result when considering the training time, this is a $\sim 10^{2.1}$ speed-up. B shows without considering the training time, with a $\sim 10^{4.5}$ speed-up. C is the case with only PL0SS3, giving a $\sim 10^{2.4}$ speed-up.

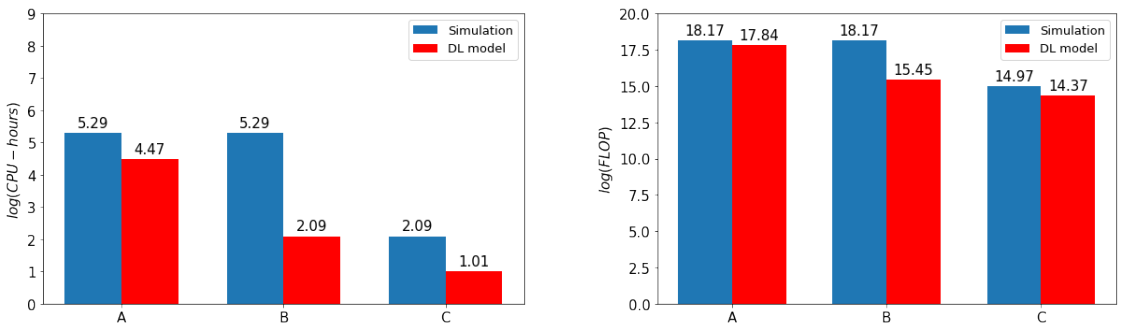

Figure 4.27: Processing time comparison the simulations in simulation and in DL model for all simulations. In the first panel, we compare the CPU-hours between both methods. In the second panel, we compare the FLOP number between them. A means the comparison when considering the training time + test time, B is considering only the test time, and C considering only the PL0SS3 simulation.

Chapter 5

Discussion

Here, we will discuss the results presented in the previous chapter. We start by discussing the results of "one simulation model" in section 5.1. In section 5.2, we discuss the trained model that takes into account all simulations. We briefly discuss the results of the GANs work in section 5.3.

5.1 One simulation

We trained the model for one system only – PNSS3. In the direct prediction, we conclude that the model can predict well after we feed a simulated snapshot to it. It returns the torus' structure and the density values with an $R^2 = 0.9988$. But we found two main issues:

(i) At higher radii, the predictions present the formation of an anomalous structure in the polar direction.

(ii) We see the atmosphere presents variations higher than the torus when we calculate the difference between prediction and simulation.

The (i) is due to the poor resolution of the original simulation at the outer radii. As we see in figure 2.3, the inner radii have higher resolution than the outer ones. We have more cells in the inner region, and this is more information on CNNs. The opposite happens at the high radii, because we have fewer cells, leading to less information. We can solve this by feeding data with similar and high-resolution in all grid. Another possibility is to propose the use of other techniques such as GlobalMaxPooling (Christlein et al., 2019) in the encoder or Transpose Convolution (Dumoulin and Visin, 2016) in the decoder. These techniques may lead to keeping the information accurately in the regions with larger

and fewer cells. The (ii) is because the values of the atmosphere are small, even with normalization and the fact that we set low loss' weights in this region. It leads to a biased dataset inclined to learn the higher values of the system. The atmosphere defines the boundaries of the torus, and it plays an important role when we deal with winds and diffusion in this region. But in this case, the system is stationary without much variability in the boundaries.

For the "one simulation model", the error in the atmosphere becomes a problem in the iterative process. In the iterative process, the torus starts to lose its structure in the boundaries getting a triangular-like aspect. It indicates the problem in the loss function to contemplate every aspect of the system. We need another loss function, as we proposed in the "all simulations case" and the cGANs.

In the iterative process, the issues (i) and (ii) got worse, and there is an additional (iii) issue:

(iii) The region near the event horizon increases the density after each iteration that raises the error to 10^2.

This also leads to an accumulation of mass near the event horizon. It indicates the model struggles to understand the accretion rate by the black hole. We aim to constrain the accretion rate in future works using networks such as Lagrangians NNs and Hamiltonians NNs (Greydanus et al., 2019). The iterative process shows that the errors are summing after each iteration, so we propose the use of cGANs that do not depend on the previous snapshot.

5.2 All simulations

Figure 4.13 and figure 4.14 shows the result of the direct predictions from all simulations. The model can predict all the simulations with a lower Δ, except for the PNST1. PNST1 presents a turbulent behavior, and it has less data in comparison with the other simulations. The number of data is an essential component in the learning process since the DL model is data-hungry. Another main aspect is that PNST1 presents high variability, and the model cannot learn the region that changes fast. Networks such as RNNs and LSTMs may be techniques to overcome this problem since these are networks created to deal with time-series and sequences. RNNs and LSTMs present results in predicting

turbulence, as seen in the work of Pathak et al. (2018) and Mohan et al. (2019). GNNs are also new techniques to deal with fluid dynamics, as proposed in the work of Cranmer et al. (2020). We suggest the use of these techniques as well as hybrid models in future work.

The "all simulations case" motivated us to see how the trained model behaves with a never-seen system. We feed the PL0SS3 to the model and iterate the results to see how the model simulates the system by itself. The iteration confirms (ii) and (iii) issues: the model struggles to learn the atmosphere and the accretion rate misleading when predicting near the event horizon. It that the learned mass evolution does not follow the simulation. The model decreases the density during the iteration, causing the loss of the mass. Although we base the loss function in the density values, the loss still does not contemplate the density of lower radii properly. It indicates the loss does not consider the accretion rate well, so that it may need a new loss. Except for the lower radii, the model can predict other regions of the torus well for a never seen system before. The solutions proposed, such as constrain the accretion rate and the use of a new loss function, hold here. Also, it is interesting to feed more simulated data to the model since we only trained with seven simulations — 7 different initial conditions.

We see that we need an appropriate loss function that can contemplate the complexity of the problem. In the one simulation case, we use a geometrically based loss, and in all simulations case, we use a density-based loss function. To find an appropriate loss is not an easy task, so we propose the use of cGANs since the discriminator acts as a loss, and it can learn with the data. In this case, we do not need to specify a loss.

5.3 cGANs

Figure 4.20 presents the preliminary results of the cGANs. We feed a parameter vector to the model, and it generates a density field. After feeding a set of parameters that are consecutive in time, we obtain the mass profile for both simulations in figure 4.21, as well as the density mean in figure 4.22 for PNSS3 and figure 4.23. With those results, we see that cGANs show as a promising DL model to simulate black holes. The main advantage of using cGANs is the presence of a discriminator, as well as the results, do not depend on the previous one, i.e., the errors do not sum up during the iterative process.

As shown in the plots of Cartesian-coordinates and of the density mean, we see that cGANs return the system evolution more similar to the simulations than the previous networks. This result is mainly due to the discriminator that can learn the properties of each system differently of the fixed loss function in the CNNs. However, mass evolution is still an issue even to the cGANs. We need to propose a constrain in the accretion rate of the model and the mass conservation. We also suggest the use of a hybrid model that combines cGANs with LSTMs.

Chapter 6

Conclusions

In this work, we proposed a DL method that could learn the physics of an accretion flow around a black hole, i.e., make black hole weather forecasting from a simulated dataset. We proposed a CNN architecture fed with one simulation at first and then, fed with seven simulations. The CNN could predict the future states correctly until $t \sim 680000\ GM/c^3$ in the case for "one simulation". In the "seven simulation"case, it could predict a never-seen system, showing some discrepancies in the atmosphere at lower radii. We also proposed the use of cGANs to simulate the system since the cGANs do not need a customized loss function, and the errors do not accumulate during the iterative process.

We feed our model with simulated data from Almeida and Nemmen (2020). In their work, they performed HD simulations of BH accretion flow in the RIAF mode. They investigated how the properties such as angular momentum profile and the viscosity affect the flow. Motivated by recent interest in exploring ML, data-driven techniques for modeling spatiotemporally chaotic systems (Pathak et al., 2018; Mohan et al., 2019), we proposed to use DL technique to perform HD simulations of a BH accretion flow in the RIAF mode. Our goal was to obtain a DL model capable of learning and predict future states of the system after learning from observations of the data. Our dataset is 400×200 arrays, so it indicates to used CNNs as our leading architecture. We treated the problem as a computer vision problem: each matrix of density can be interpreted as an image matrix of $256 \times 192 \times 1$. We also considered the channels as a temporal axis creating a $256 \times 192 \times 5$ array. The main conclusions are:

(i) For the models, we found that the CNNs approach was able to forecast well the system for a duration of $\Delta t \sim 80000 GM/c^3$. During this period, the maximum difference

is $\Delta log(\rho) = 2$. The error builds up and becomes $\Delta log(\rho) = 12$ after the time mentioned above.

(ii) For the GAN approach, we found that this approach was able to forecast the system well for a duration of $\Delta t \sim 100000 GM/c^3$. During this period, the maximum difference is $\Delta log(\rho) = 3$ in the atmosphere and $\Delta log(\rho) = 0.4$ in the torus. The error is constant during the entire prediction for the PNSS3 system. In the case of PNST1, the maximum difference is $\Delta log(\rho) = 0.75$.

(iii) We found some shortcomings with the approach: excessive diffusion and the distortion of the geometry of accretion flow. We believe this is due to a biased dataset and low weights in the loss function.

(iv) Our approach is promising for generalizing for a never before seen system. For instance, when we feed all the simulations, we could predict an unseen system only with observations of other systems. The model could correctly predict the PL0SS3 system for $\Delta t \sim 40000\ GM/c^3$ without seeing it before. The maximum difference is $\Delta log(\rho) = 1$ during the iterative process.

(v) We also obtain a 32289x speed-up using DL method — considering only the time to achieve the results and we heave a trained model available. The speed-up considering the training time is 38x. But when comparing the CPU-hours, we get a 1736x speed-up and a 2x speed-up considering training time. In this work, we used two GPUs: Quadro P6000 and Quadro GP100, with 3840 and 3584 CUDA-cores, respectively.

(vi) In "all simulations"case, we obtain a $10^{4.46}$x speed-up in practice — considering only the prediction time. Considering the training time, we obtain a $10^{2.1}$x speed-up. But when comparing the CPU-hours, we get a $10^{3.2}$x speed-up and a $10^{0.82}$x speed-up considering training time.

In conclusion, we believe that a data-driven, equations-free ML approach is very promising. It could potentially be a faster alternative to directly solving the nonlinear partial

differential equations that correspond to the conservation laws of the flow.

6.1 Future Perspectives

In this section, we delineate future directions for work in this field which if pursued should improve the performance of the learning models. They are divided in the following categories: (i) generate more and better training data, (ii) explore other neural network architectures, better adapted for time-domain studies and (iii) improve the loss function, and (iv) investigate in more details the generalization properties of the learned models.

Below, we develop these directions in more details:

(i) **Generate more — and better — training data and few-shot learning**

The DL method usually benefits from a larger dataset. Generating more data is a direction to have a better performance of such models as CNNs and cGANs. Also, the quality of data must be taken into consideration since, in our model, we see that the poor resolution at outer radii presents a limitation. However, few-shot learning techniques are increasing in the DL field. Few-shot learning techniques are useful when only a few examples are available (Sung et al., 2017). Recently, Brown et al. (2020) proposed to use few-shot learning in the natural languagem processing (NLP) field, creating a powerful tool called generative pre-training transformer (GPT-3). The GPT-3 benefits from few-shot learning, and it is already used to NLP problems such as generating architecture by itself and creating human-level stories. We suggest the use of few-shot learning techniques to simulate accretion flows.

(ii) **Explore other neural network architectures, better adapted for time-domain studies**

To deal with time-series problems, RNNs may be a better option. Originally, RNNs were proposed to NLP problems, since the sentences depend on previous words. RNNs present gates that learn the sequences and the prediction depend on previous information. Here, we suggest some of networks:

- LSTMs: The long short term memory networks (LSTMs) are networks widely used in NLP and to forecast time-series problems (Gers et al., 2002; Wen et al., 2015). It

is a possible tool to make physical forecasting, especially the Conv2DLSTMs (Karim et al., 2018) that are combinations of CNNs and LSTMs. Th main difference that LSTMs bring is because it is composed of gates: learning/output gate, state gate and forget gate. Those gates combined can remember features of previous data and it may be possible to forget useless information for the learning process.

- Attention: the attention network was proposed by (Vaswani et al., 2017) and has attention modules as well as channels that would "pay attention" to certain regions of the data. It is also possible to create hybrid-models such as combinations of Attention with LSTMs (Bin et al., 2019).

- GANs: we used a GAN as a test in this project and it already shows encouraging results. As explained above, GANs are useful when we do not have a loss that is able to contemplate the complexity of the systems and the error does not sum up in. So a hybrid-model as GANs combination with LSTMs and Attentions may contemplate the complexity of the system.

(iii) **Improve the loss function**

One of the main issues we found in our work is the loss function. We needed a loss function able to contemplate all the configurations and the regions of the system. To solve that, we proposed two loss functions, but both presented limitations. To get better results, we need a function that can interpret all regions of the system. However, such function may not exist or can be too complex to use. We proposed the use of networks that does not need a specific loss function, such as cGANs.

(iv) **Investigate in more details the generalization properties of the learned models**

A goal is to obtain a model that can generalize better what it learns. Also, we aim to use more realistic configurations of accretion flows. We want to test the ability of the model to learn GRMHD models. It is worth investigating the ability to predict GRMHD models to quantify how much of the black hole physics the DL models can learn. GRMHD systems are complex and present turbulence due to the presence of magnetic fields, and they can explain the formation of jets.

Bibliography

Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., et al., Tensorflow: A system for large-scale machine learning , 2016, p. 265

Abbott B. P., Abbott R., Abbott T. D., Abernathy M. R., Acernese F., Observation of Gravitational Waves from a Binary Black Hole Merger, Phys. Rev. Lett., 2016, vol. 116, p. 061102

Abramowicz M. A., Fragile P. C., Foundations of Black Hole Accretion Disk Theory, Living Reviews in Relativity, 2013, vol. 16, p. 1

Akeret J., Chang C., Lucchi A., Refregier A., Radio frequency interference mitigation using deep convolutional neural networks, Astronomy and Computing, 2017, vol. 18, p. 35

Almeida I., Nemmen R., Winds and feedback from supermassive black holes accreting at low rates: hydrodynamical treatment, MNRAS, 2020, vol. 492, p. 2553

Balbus S. A., Enhanced Angular Momentum Transport in Accretion Disks, ARA&A, 2003

Balbus S. A., Hawley J. F., A Powerful Local Shear Instability in Weakly Magnetized Disks. I. Linear Analysis, ApJ, 1991, vol. 376, p. 214

Beckman J. E., The Nearest Active Galaxies, 1993

Bellman R., Dynamic programming, Science, 1966, vol. 153, p. 34

Bin Y., Yang Y., Shen F., Xie N., Shen H. T., Li X., Describing Video With Attention-Based Bidirectional LSTM, IEEE Transactions on Cybernetics, 2019, vol. 49, p. 2631

Blandford R. D., Znajek R. L., Electromagnetic extraction of energy from Kerr black holes., MNRAS, 1977, vol. 179, p. 433

Bolton C. T., Identification of Cygnus X-1 with HDE 226868, Nature, 1972, vol. 235, p. 271

Breen P. G., Foley C. N., Boekholt T., Portegies Zwart S., Newton versus the machine: solving the chaotic three-body problem using deep neural networks, MNRAS, 2020, vol. 494, p. 2465

Brown T., Mann B., Ryder N., Subbiah M., Language Models are Few-Shot Learners, 2020

Chollet F., et al.,, 2015 keras

Christlein V., Spranger L., Seuret M., Nicolaou A., KrÃ¡l P., Maier A., Deep Generalized Max Pooling, 2019

Çiçek Ö., Abdulkadir A., Lienkamp S. S., Brox T., Ronneberger O., 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation , Springer International Publishing, Cham, 2016, p. 424

Cranmer M., Greydanus S., Hoyer S., Battaglia P., Spergel D., Ho S., Lagrangian Neural Networks, 2020

de Oliveira L., Paganini M., Nachman B., Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis, Computing and Software for Big Science, 2017, vol. 1

De Villiers J., Hawley J. F., Three dimensional Hydrodynamic Simulations of Accretion Tori in Kerr Spacetimes, The Astrophysical Journal, 2002, vol. 577, p. 866

Draper N., Smith H., Applied regression analysis. Wiley series in probability and mathematical statistics, Wiley, 1966

Dumoulin V., Visin F., A guide to convolution arithmetic for deep learning, 2016

Event Horizon Telescope Collaboration First M87 Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole, ApJ, 2019, vol. 875, p. L1

Fabbro S., Venn K. A., O'Briain T., Bialek S., Kielty C. L., Jahandar F., Monty S., An application of deep learning in the analysis of stellar spectra, MNRAS, 2017, vol. 475, p. 2978

Ferrarese L., Ford H., Supermassive Black Holes in Galactic Nuclei: Past, Present and Future Research, Space Sci. Rev., 2005, vol. 116, p. 523

Fragile P. C., Anninos P., Hydrodynamic Simulations of Tilted Thick Disk Accretion onto a Kerr Black Hole, The Astrophysical Journal, 2005, vol. 623, p. 347

Frank J., King A., Raine D., Accretion Power in Astrophysics. Cambridge University Press, 2002

Gensler A., Henze J., Sick B., Raabe N., Deep Learning for solar power forecasting â An approach using AutoEncoder and LSTM Neural Networks , 2016, p. 002858

Gers F. A., Eck D., Schmidhuber J., Applying LSTM to Time Series Predictable Through Time-Window Approaches , Springer London, 2002

Gilfanov M., X-Ray Emission from Black-Hole Binaries. vol. 794, 2010, 17

Goldsmith J., An algorithm for the unsupervised learning of morphology, Natural Language Engineering, 2006, vol. 12, p. 353

Goodfellow I., Bengio Y., Courville A., Deep Learning. The MIT Press, 2016

Goodfellow I. J., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., Generative Adversarial Nets , NIPS14, MIT Press, Cambridge, MA, USA, 2014, p. 2672

Gopinath R., Burrus C., On Upsampling, Downsampling, and Rational Sampling Rate Filter Banks, Trans. Sig. Proc., 1994, vol. 42, p. 812

Greydanus S., Dzamba M., Yosinski J., Hamiltonian Neural Networks, 2019

Grover A., Kapoor A., Horvitz E., A Deep Hybrid Model for Weather Forecasting , 2015, p. 379

Hausen R., Robertson B., Morpheus: A Deep Learning Framework For Pixel-Level Analysis of Astronomical Image Data, 2019

He S., Li Y., Feng Y., Ho S., Ravanbakhsh S., Chen W., Póczos B., Learning to predict the cosmological structure formation, Proceedings of the National Academy of Sciences, 2019, vol. 116, p. 13825

Hon M., Stello D., Yu J., Deep learning classification in asteroseismology, MNRAS, 2017, vol. 469, p. 4578

Huertas-Company M., Rodriguez-Gomez V., Nelson D., Pillepich A., Bottrell C., Bernardi M., Dominguez-Sanchez H., Genel S., Pakmor R., Snyder G. F., Vogelsberger M., The Hubble Sequence at z $\sim$ 0 in the IllustrisTNG simulation with deep learning, MNRAS, 2019, vol. 489, p. 1859

Igumenshchev I. V., Abramowicz M. A., Narayan R., Numerical Simulations of Convective Accretion Flows in Three Dimensions, The Astrophysical Journal, 2000, vol. 537, p. L27

Isola P., Zhu J. Y., Zhou T., Efros A. A., Image-to-Image Translation with Conditional Adversarial Networks, 2016

Jain P., Meka R., Dhillon I. S., Simultaneous Unsupervised Learning of Disparate Clusterings, Stat. Anal. Data Min., 2008, vol. 1, p. 195

Karim F., Majumdar S., Darabi H., Chen S., LSTM Fully Convolutional Networks for Time Series Classification, IEEE Access, 2018, vol. 6, p. 1662

Kerr R. P., Gravitational Field of a Spinning Mass as an Example of Algebraically Special Metrics, Phys. Rev. Lett., 1963, vol. 11, p. 237

King R., Hennigh O., Mohan A., Chertkov M., From Deep to Physics-Informed Learning of Turbulence: Diagnostics, 2018

Kingma D. P., Ba J., Adam: A Method for Stochastic Optimization, 2014

Kirby M., Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns. John Wiley Sons, Inc. USA, 2000

Krizhevsky A., Sutskever I., Hinton G. E., ImageNet Classification with Deep Convolutional Neural Networks, 2012, p. 1097

Landau L. D., Lifshitz E. M., Fluid mechanics, 1959

LeCun Y., Bengio Y., Convolutional Networks for Images, Speech, and Time Series, 1998, p. 255

Lloyd S., Least squares quantization in PCM, IEEE Transactions on Information Theory, 1982, vol. 28, p. 129

Lukosevicius M., Jaeger H., Reservoir computing approaches to recurrent neural network training, Computer Science Review, 2009, vol. 3, p. 127

Lydia A., Francis S., Adagrad - An Optimizer for Stochastic Gradient Descent, 2019, vol. 6, p. 566

McCulloch W. S., Pitts W., A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics, 1943, vol. 5, p. 115

McKinney J. C., Tchekhovskoy A., Blandford R. D., General relativistic magnetohydrodynamic simulations of magnetically choked accretion flows around black holes, Monthly Notices of the Royal Astronomical Society, 2012, vol. 423, p. 3083

Mathuriya A., Bard D., Mendygral P., Meadows L., Arnemann J., Shao L., He S., KÃrnÃ T., Moise D., Pennycook S. J., Maschhoff K., Sewall J., Kumar N., Ho S., Ringenburg M. F., Prabhat P., Lee V., CosmoFlow: Using Deep Learning to Learn the Universe at Scale. In SC18: International Conference for High Performance Computing, Networking, Storage and Analysis , 2018, p. 819

McKinney J. C., Gammie C. F., A Measurement of the Electromagnetic Luminosity of a Kerr Black Hole, ApJ, 2004, vol. 611, p. 977

McKinney J. C., Narayan R., Disc-jet coupling in black hole accretion systems - I. General relativistic magnetohydrodynamical models, MNRAS, 2007, vol. 375, p. 513

Meier D. L., Black Hole Astrophysics: The Engine Paradigm. Springer Science Business Media, 2012

Misner C. W., Thorne K. S., Wheeler J. A., Gravitation. W. H. Freeman San Francisco, 1973

Mohan A., Daniel D., Chertkov M., Livescu D., Compressed Convolutional LSTM: An Efficient Deep Learning framework to Model High Fidelity 3D Turbulence , 2019

Mohan A., Livescu D., Chertkov M., Physics-Constrained Convolutional LSTM Neural Networks for Generative Modeling of Turbulence. In APS Division of Fluid Dynamics Meeting Abstracts , APS Meeting Abstracts, 2019, p. C17.002

Moscibrodzka, M. Falcke, H. Noble, S. Scale-invariant radio jets and varying black hole spin, A&A, 2016, vol. 596, p. A13

Murtagh F., Multilayer perceptrons for classification and regression, Neurocomputing, 1991, vol. 2, p. 183

Nagi J., Ducatelle F., Di Caro G. A., CireÅan D., Meier U., Giusti A., Nagi F., Schmidhuber J., Gambardella L. M., Max-pooling convolutional neural networks for vision-based hand gesture recognition , 2011, p. 342

Nieto D., Brill A., Feng Q., Humensky T. B., Kim B., Miener T., Mukherjee R., Sevilla J., CTLearn: Deep Learning for Gamma-ray Astronomy, 2019

Paczyński B., Wiita P. J., Thick accretion disks and supercritical luminosities., A&A, 1980, vol. 500, p. 203

Pathak J., Hunt B., Girvan M., Lu Z., Ott E., Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach, Phys. Rev. Lett., 2018, vol. 120, p. 024102

Pathak J., Wikner A., Fussell R., Chandra S., Hunt B. R., Girvan M., Ott E., Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model, Chaos: An Interdisciplinary Journal of Nonlinear Science, 2018, vol. 28, p. 041101

Pearson W. J., Wang, L. Trayford, J. W. Petrillo, C. E. van der Tak, F. F. S. Identifying galaxy mergers in observations and simulations with deep learning, A&A, 2019, vol. 626, p. A49

Penna R. F., Kulkarni, Akshay Narayan, Ramesh A new equilibrium torus solution and GRMHD initial conditions, A&A, 2013, vol. 559, p. A116

Penrose R., Gravitational Collapse: the Role of General Relativity, Nuovo Cimento Rivista Serie, 1969, vol. 1, p. 252

Perraudin N., Srivastava A., Lucchi A., Kacprzak T., Hofmann T., Refregier A., Cosmological N-body simulations: a challenge for scalable generative models, 2019

Podareanu D., Codreanu V., Aigner S., Leeuwen C., Weinberg V., Best Practice Guide - Deep Learning, 2019

Porth O., et al., The Event Horizon General Relativistic Magnetohydrodynamic Code Comparison Project, Astrophys. J. Suppl., 2019, vol. 243, p. 26

Prechelt L., Early Stopping-But When? , Springer-Verlag, Berlin, Heidelberg, 1998, p. 55

Proga D., Begelman M. C., Accretion of Low Angular Momentum Material onto Black Holes: Two-dimensional Hydrodynamical Inviscid Case, The Astrophysical Journal, 2003, vol. 582, p. 69

Qiao T., Zhang J., Xu D., Tao D., MirrorGAN: Learning Text-to-image Generation by Redescription, 2019

Reed S., Akata Z., Yan X., Logeswaran L., Schiele B., Lee H., Generative Adversarial Text to Image Synthesis, 2016

Rezaie M., Seo H.-J., Ross A. J., Bunescu R. C., Improving Galaxy Clustering Measurements with Deep Learning: analysis of the DECaLS DR7 data, 2019

Ribli D., Ãrmin Pataki B., Csabai I., An improved cosmological parameter inference scheme motivated by deep learning, 2018

Ronneberger O., Fischer P., Brox T., U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015

Rosenblatt F., Perceptron Simulation Experiments, Proceedings of the IRE, 1960, vol. 48, p. 301

Roy N., Bundy K., Cheung E., Rujopakarn W., Cappellari M., Detecting Radio AGN Signatures in Red Geysers., 2018

Ruder S., An overview of gradient descent optimization algorithms, 2016

Salian I., , 2018 SuperVize Me: What is the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning? https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

Schawinski K., Zhang C., Zhang H., Fowler L., Santhanam G. K., Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit, MNRAS, 2017, vol. 467, p. L110

Schmidt M., 3C 273 : A Star-Like Object with Large Red-Shift, Nature, 1963, vol. 197, p. 1040

Schodel R., et al., A Star in a 15.2 year orbit around the supermassive black hole at the center of the Milky Way, Nature, 2002, vol. 419, p. 694

Shakura N. I., Sunyaev R. A., Black holes in binary systems. Observational appearance., A&A, 1973, vol. 500, p. 33

Shallue C. J., Vanderburg A., Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90, The Astronomical Journal, 2018, vol. 155, p. 94

Siemiginowska A., Eadie G., Czekala I., Feigelson E., The Next Decade of Astroinformatics and Astrostatistics, 2019

Silver D., Huang A., Maddison C., Guez A., Sifre L., Driessche G. v. d., Schrittwieser J., Antonoglou I., Panneershelvam V., Mastering the game of Go with deep neural networks and tree search, Nature, 2016

Stone J. M., Pringle J. E., Magnetohydrodynamical non-radiative accretion flows in two dimensions, MNRAS, 2001, vol. 322, p. 461

Stone J. M., Pringle J. E., Begelman M. C., Hydrodynamical non-radiative accretion flows in two dimensions, MNRAS, 1999, vol. 310, p. 1002

Sung F., Yang Y., Zhang L., Xiang T., Torr P. H. S., Hospedales T. M., Learning to Compare: Relation Network for Few-Shot Learning, 2017

Tchekhovskoy A., McKinney J. C., Prograde and retrograde black holes: whose jet is more powerful?, MNRAS, 2012, vol. 423, p. L55

Turner N., Stone J., Krolik J., Sano T., Local three-dimensional simulations of magneto-rotational instability in radiation-dominated accretion disks, Astrophys. J., 2003, vol. 593, p. 992

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L. u., Polosukhin I., Attention is All you Need, 2017, p. 5998

Wagner R. M., Starrfield S., Cassatella A., Gonzalez-Riestra R., Kreidl T. J., Howell S. B., Hjellming R. M., Han X.-H., Shrader C., Sonneborn G., The 1989 Outburst of V404 Cygni: A Very Unusual X-Ray Nova, International Astronomical Union Colloquium, 1990, vol. 122, p. 429

Wang Y., Bilinski P., Bremond F. F., Dantcheva A., ImaGINator: Conditional Spatio-Temporal GAN for Video Generation, 2020

Wen T.-H., Gasic M., Mrksic N., Su P.-H., Vandyke D., Young S., Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems, 2015

Yuan F., Narayan R., Hot Accretion Flows Around Black Holes, ARA&A, 2014, vol. 52, p. 529

Yuan F., Quataert E., Narayan R., Nonthermal Electrons in Radiatively Inefficient Accretion Flow Models of Sagittarius A*, ApJ, 2003, vol. 598, p. 301

Zeiler M. D., Adadelta: An Adaptive Learning Rate Method, 2012

Appendix

Appendix A

Details

Table A.1 - Here are the split of the dataset. Above each percentage, there is the number of frames corresponding as well the ΔT (GM/c^3) of the range.

Train	Validation	Test
67.5%	12.5%	20%
1606 frames	178 frames	594 frames
$317939\,GM/c^3$	$34644\,GM/c^3$	$117594\,GM/c^3$

Table A.2 - Hyperparameters found by the grid search method for the one simulation case. These are the hyperparameters with the best results on the R^2 metric during the evaluation and the best analysis results.

Hyperparameter	Values
Batch Size	64
Learning Rate	0.0005
α	8.0
β	5.0
γ	10.0
δ	4.0

Table A.3 - Hyperparameters found by the grid search method for the one simulation case. These are the hyperparameters with the best results on the R^2 metric during the evaluation and the best analysis results.

Hyperparameter	Values
Batch Size	64
Learning Rate	0.0005
ρ_{hp}	0.8
a	10.0
b	1.0