

CBPF – Centro Brasileiro de Pesquisas Físicas

Dissertação de Mestrado

The use of Deep Learning to Automated Transient Identification

Phelipe Antonie Darc De Matos

Orientador
Dr. Clécio Roque De Bom

Rio de Janeiro, RJ
2023

Phelipe Antonie Darc De Matos

The use of Deep Learning to Automated Transient Identification

Trabalho apresentado ao Programa de Pós-Graduação no Centro Brasileiro de Pesquisas Físicas como requisito parcial para obtenção do grau de Mestre em Física.

CBPF – Centro Brasileiro de Pesquisas Físicas

Orientador: Dr. Clécio Roque De Bom

Rio de Janeiro, RJ
2023

Darc, Phelipe Antonie

The use of Deep Learning to Automated Transient Identification/ Phelipe Antonie
Darc De Matos. - 2023

90 f. : il.

Dissertação de Mestrado – CBPF – Centro Brasileiro de Pesquisas Físicas , Rio de Janeiro, RJ, 2023.
Orientador: Dr. Clécio Roque De Bom

1. (Listar palavras-chave) I. Título

CDU 02:141:005.7

AGRADECIMENTOS

Agradeço ao All might.

”Quando acordei de manhã eu sabia quem eu era,
mas acho que já mudei muitas vezes desde então”
- Alice, de Alice no País das Maravilhas

ABSTRACT

The present work proposes the use of deep learning techniques to automatically identify astrophysical transients in the Dark Energy Survey GW (DESGW) and the Southern Photometric Local Universe Survey (SPLUs) datasets. Specifically, we aim to integrate a neural network into the difference imaging pipeline of these surveys to accelerate transient detection by reducing the number of images that require visual inspection. To achieve this objective, we plan to use convolutional neural networks (CNNs), which are a type of deep learning algorithm that has proven effective in image classification tasks. The CNN will be trained on a labeled dataset of images containing both transients and non-transients, allowing it to learn the characteristic features of astrophysical transients. Once trained, the CNN will be integrated into the difference imaging pipeline, where it will analyze the images and automatically identify potential transients that will then be confirmed by visual inspection. The obtained result was that in the DESGW and SPLUs datasets, we were able to remove 95% and 97% of all non-transient examples, respectively. This leads to faster and more efficient transient detection, as the neural network reduces the number of images that need to be inspected. Thus, astronomers can focus on scientifically relevant events. Furthermore, it can help identify transients that are not obvious to the astronomer and may have been classified as false negatives.

Overall, the proposed approach has the potential to significantly improve the capability and effectiveness of transient detection in the DESGW and SPLUs surveys.

Key-Words: Multimessager Astronomy, Gravitational Waves, Machine Learning, Deep learning, Astrophysical Transients, Artificial Intelligence.

LISTA DE ILUSTRAÇÕES

Figura 1.1 – Kilonova (GW170817) a visible example of Gravitational Wave Counterpart	2
Figura 2.1 – Upper Panel: Two white dwarfs collide, causing a Type Ia supernova. Bottom panel: One white dwarf gaining mass from a companion star and eventually exploding into a Type Ia Supernova Credit: NASA/Dana Berry, Sky Works Digital.	6
Figura 2.2 – Two Neutron stars merging and eventually, creating a Kilonova and Gravitational Waves. Credit: NASA/Dana Berry, Sky Works Digital https://svs.gsfc.nasa.gov/12949	7
Figura 2.3 – DeCAm and the Blanco Telescope at the cerro tololo inter american observtatory (CTIO).	8
Figura 2.4 – Standard Bandpasses for the DECam grizY filters	9
Figura 2.5 – A schematic overview of the complete single epoch processing of DESGW pipeline	10
Figura 2.6 – On the upper left The interior of the T80-South Telescope and on the upper right T80-South Telescope dome at Cerro Tololo Inter-American Observatory (CTIO) in Chile (Bottom).	12
Figura 2.7 – 12 S-PLUS survey filter curves	13
Figura 2.8 – A schematic overview of the complete transient detection pipeline	14
Figura 3.1 – left to right, The search, template and diff image of a transient exemple from STEP.	18
Figura 3.2 – Bad subtraction Artifact - STEP	19
Figura 3.3 – Noisy Artifact - STEP	19
Figura 3.4 – Diffraction Spikes Artifact - STEP	20
Figura 3.5 – Near Field object Artifact - STEP	20
Figura 3.6 – Others Artifact - STEP	20
Figura 3.7 – Badsubtraction - DES-GW	21
Figura 3.8 – Preexisting point - DES-GW	21
Figura 3.9 – Not an obivous Transient - DES-GW	22

Figura 3.10–Dark spot - DES-GW	22
Figura 3.11–Noisy template - DES-GW	22
Figura 3.12–left to right, The search, template and diff image of a transient exemple from DESGW transient detection pipeline.	22
Figura 4.1 – The hierarchy with a resumed explanation of each branch of AI.	26
Figura 4.2 – Schematic of a neuron Network information flow. the input X are weightned by w and summed with the bias	27
Figura 4.3 – A schematic representing an example of a neural network with two inputs, three layers with three neurons in the first layer and two in the remaining layers, and two outputs. The weights of each connections are denoted as w and the bias b.	28
Figura 4.4 – A schematic dataflow complete the legend...	32
Figura 4.5 – Learning rate complete the label	33
Figura 4.6 – Example of 3D convolutional operation on a RGB input image	40
Figura 4.7 – Example of different pooling layers acting on a 5x5 feature map.	41
Figura 4.8 – This figure shows a simple example of how a CNN works. It uses an image of Jupiter to demonstrate the process of applying a convolutional filter, detecting features with an activation function, and condensing information with a pooling layer. Credits: —NASA cpy later	42
Figura 4.9 – Confusion Matrix, the rows represents the predicted labels and the columns represent the actual label	44
Figura 4.10–Confusion Matrix, the rows represents the predicted labels and the columns represent the actual label	45
Figura 4.11–Confusion Matrix, the rows represents the predicted labels and the columns represent the actual label	46
Figura 6.1 – Gráfico de Acurácia da rede ResNet50 em toda a base de dados.	53
Figura 6.2 – Gráfico da Função Custo da rede ResNet50 em toda a base de dados.	53
Figura 6.3 – Gráfico da curva ROC da rede ResNet50 em toda a base de dados.	54
Figura 6.4 – Distribuição de AUCs de acordo o tamanho do conjunto de treinamento.	57
Figura 6.5 – Esquema representativo do uso de diferentes conjuntos de treino e de validação durante o treinamento de uma rede neural utilizando validação cruzada de k folds com 4 folds.	59
Figura 6.6 – Distribuição de AUCs de acordo o tamanho do conjunto de treinamento e respectivas barras de erro.	60
Figura 6.7 – Comparação entre as AUCs com e sem aumentação de dados.	62
Figura 6.8 – Comparação entre as AUCs com e sem pré-carregamento de pesos.	64
Figura 6.9 – Comparação de desempenhos das redes ResNet50 e EfficientNetB2.	66
Figura 6.10–Comparação de desempenhos da rede ResNet50, EfficientNetB2 e do ensemble entre elas.	68

Figura 6.11–Comparação de desempenhos de duas técnicas de ensemble pré-carregadas e do ensemble por média aritmética com o das redes individuais.	69
Figura 6.12–Dois exemplos de imagens em suas bandas H, J, Y, VIS e o resultado da combinação das três bandas HJY	72
Figura 6.13–Histograma de pixels das imagens nas bandas H, J, Y e VIS	74
Figura 6.14–Histograma final de pixels das imagens nas bandas H, J, Y e VIS	75
Figura 6.15–Acurácia do Treino da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.	77
Figura 6.16–Pontos da Função Custo do Treino da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.	78
Figura 6.17–Curva ROC e AUC de cada modelo ao final da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.	78
Figura 6.18–Curva ROC e AUC de cada modelo ao final da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC, usando a banda VIS.	79
Figura 6.19–Desempenho da rede EfficientNet B2 nos dados da segunda edição do GLFC.	81
Figura 6.20–Desempenho das redes EfficientNet B2 concatenadas nos dados da segunda edição do GLFC, para diversos tamanhos de conjuntos de treino.	82

LISTA DE TABELAS

Tabela 1 – Comparação entre AUCs variando o critério de Lente Gravitacional . . . 83

LISTA DE ABREVIATURAS E SIGLAS

CBPF	Centro Brasileiro de Pesquisas Físicas
IA	Inteligência Artificial
AM	Aprendizado de Máquina
RNA	Rede Neural Artificial
RNC	Convolutional Neural Network
AUC	Área Sob a Curva
ROC	Característica de Operação do Receptor
LG	Lenteamento Gravitacional
AP	Aprendizagem Profunda
MC	Mapa de Características
ILSVRC	<i>ImageNet Large Scale Visual Recognition Challenge</i>
GLFC	<i>Gravitational Lens Finding Challenge</i>
ESA	<i>European Space Agency</i>
MECO	Método de Escalonamento Composto
SDSS	<i>Sloan Digital Sky Survey</i>
SLACS	<i>Sloan Lens ACS Survey</i>
DES	<i>Dark Energy Survey</i>
CFHTLS	<i>Canada-France-Hawaii Telescope Lens Survey</i>
DESI	<i>Dark Energy Spectroscopic Interferometer</i>

KiDS	<i>Kilo Degree Survey</i>
HST	<i>Hubble Space Telescope</i>
HDF	<i>Hubble Deep Field</i>
HUDF	<i>Hubble Ultra Deep Field</i>

SUMÁRIO

Lista de ilustrações	ix
Lista de tabelas	xiii
Sumário	xvii
1 INTRODUCTION	1
2 THE SEARCH FOR TRANSIENTS	5
2.1 Astrophysical Transients	5
2.2 Transient Detection Pipeline in DES-GW	7
2.2.1 Dark Energy Survey (DES)	8
2.2.2 DESGW - Pipeline	9
2.2.2.1 Single epoch processing	10
2.3 SPLUS Transient detection pipeline	12
2.3.1 S-PLUS Transient Extension Program (STEP)	13
2.3.1.1 Preprocessing and Reduction process of raw and template images	13
3 DIFFERENCE IMAGING	17
3.1 Artifacts	18
3.1.1 Artifacts on STEP	19
3.1.2 Artifacts on DES-GW	21
3.2 Candidate selection - trocar eventualmente	23
3.3 STEP and DESGW Resumed Pipeline - trocar nome eventualmente	23
4 A BRIEF INTRODUCTION TO ARTIFICIAL INTELLIGENCE	25
4.1 Introduction to Neural Networks	26
4.2 Activation functions and Loss functions	28
4.3 Neural Network's Learning Process:	31
4.4 Problems related to training a Artificial Neural Network	35
4.4.1 Exploding gradient and vanishing gradient	35
4.4.1.1 Overfitting and Underfitting	36
4.5 Convolutional Neural Network	37

4.5.1	Convolutional Layer	38
4.5.2	Pooling Layer	40
4.6	Metrics for Evaluation model's performance	42
4.6.1	Loss Function	43
4.6.2	Confusion Matrix (CM)	43
4.6.3	Receiver Operating Characteristics (ROC) curve	45
4.6.4	Precision-recall (PR) curve	46
5	AUTOMATED TRANSIENT DETECTION	49
6	RESULTADOS	51
6.1	Problematização	51
6.2	Desempenho da ResNet50 nos dados da primeira edição do GLFC	52
6.2.1	A investigação de um limite de poucos dados	55
6.2.2	Multiplos sets de treino/teste	58
6.2.3	Avaliação da influência da aumentação de dados	61
6.2.4	Avaliação da influência do pré-carregamento de pesos	63
6.2.5	Uso do ensemble de redes nos dados da primeira edição do GLFC	65
6.3	Classificação com dados da Segunda Edição do GLFC	70
6.3.1	Pré-processamento dos dados da segunda edição do GLFC	70
6.3.2	Desempenho do Ensemble de redes nos dados da Segunda Edição do GLFC	76
6.3.3	Treino usando a concatenação de duas EfficientNet B2	79
6.3.4	Desempenho das Efficient Net B2 concatenadas na base de dados do GLFC variando o critério da tabela-verdade	83
6.4	Conclusões e Discussão	84
7	CONSIDERAÇÕES FINAIS	87
	REFERÊNCIAS	89

CAPÍTULO 1

INTRODUCTION

Until the middle of the 20th century, the only source, or “messenger”, to observe the distant universe was the electromagnetic force carrier, or commonly called as photons, which is observed in wavelengths ranging from Radio to X-rays. This limited the understanding of the universe to what could be inferred from the electromagnetic spectrum.

However, two fundamental discoveries, the detection of neutrinos and gravitational waves (GW) from astrophysical sources, opened new windows to the Universe allowing for a more comprehensive view of the cosmos.

There are four types of what Astronomers call “messenger”:

- Electromagnetic Radiation
- Gravitational Waves
- Neutrinos
- Cosmic Rays

These observations marks the new era of Multi-messenger astronomy[1]: *The exploration of the Universe through combining information from a multitude of cosmic messengers: electromagnetic radiation, gravitational waves, neutrinos and cosmic rays.*

Different cosmic messengers, despite belonging to the same source, are produced by distinct processes and thus carry information about different mechanisms within their source, allowing us to observe the universe in a completely different way.

However, the challenge is that gravitational wave events are invisible, making it almost impossible to identify their source without any additional information. To address this challenge, astronomers have been searching for electromagnetic counterparts to gravitational wave events. One of the earliest and most well-known examples of a gravitational wave counterpart is the kilonova (GW170817) associated with the binary neutron star merger detected by LIGO in 2017 shown in 1.1.

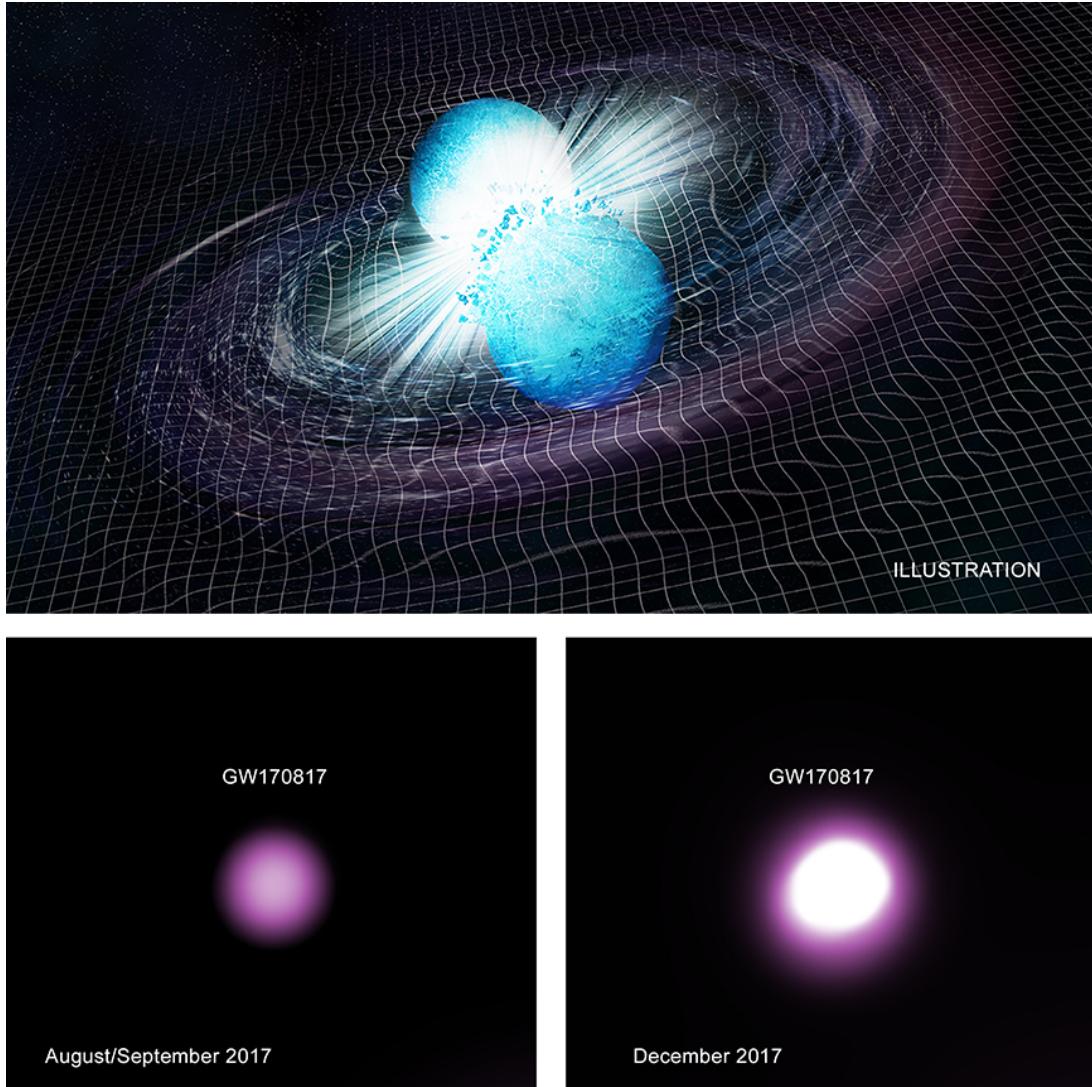


Figura 1.1 – Kilonova (GW170817) a visible example of Gravitational Wave Counterpart - Credit: NASA/CXC/Trinity University/D. Pooley et al. Illustration: NASA/CXC/M. Weiss

Searching for these optical Gravitational waves counterparts is not a simple task. The amount of data generated by O4 season on LIGO and the James Webb telescope will be massive, making it difficult for astrophysicists to analyze all of it. Therefore, an algorithm that reduces the need for visual inspection is necessary so that astrophysicists can focus on science, discovery, and projection rather than seeing different images every day. The main focus of this Thesis is search for transients, such as kilonovae or supernovae, in the electromagnetic spectrum.

The Dark Energy Survey GW (DES) and the S-PLUS project (STEP) are two surveys that have been searching for electromagnetic counterparts to gravitational wave events. In both surveys, a transient detection pipeline is used to identify possible transients. The search utilizes Difference Imaging to compare a recent image (referred to as “search”) of the sky associated with a GW or neutrino to a past image (referred to as “Template”) of

the same area of the sky. After matching the point spread functions (PSFs) to the search and template images, a subtraction of both images is made, creating a third image called a “difference image”. However, this process creates various types of false positive images. This large number of candidates makes it difficult to follow up on all of them. In this way, this thesis aims to demonstrate the use of deep learning techniques, such as convolutional neural networks, to reduce the number of candidates that require visual inspection. This dissertation is organized as follows:

Chapter 2 provides a review of the optical counterparts of gravitational waves, which are electromagnetic signals produced by astrophysical transients that are associated with gravitational wave events. The chapter explains how the transient detection pipeline works in the Dark Energy Survey GW (DESGW) survey and the Southern Photometric Local Universe Survey (SPLUs), which are two major astronomical surveys that aim to detect and study transient events. The chapter covers the basic principles and methods of transient detection, as well as some of the Artifacts and limitations of the current approach.

Chapter 3 discusses the basics of neural networks and machine learning, which are the key technologies used in the proposed approach to transient detection. The chapter explains how neural networks learn from data, and provides an overview of the different types of neural networks that are commonly used in machine learning and the most important metrics used when evaluating a binary classification, which are the focus of this study.

Chapter 4 presents a review of the state of the art algorithms for transient detection, including both classical methods and machine learning approaches, which is the current model working on DESGW. The chapter discusses the main advantages and disadvantages of each method, as well as the types of data.

Chapter five describes the results and evaluation of the proposed approach to transient detection using Convolutional Neural Network. It described the dataset used for training and testing the neural network, as well as the performance metrics used to evaluate the model.

CAPÍTULO 2

THE SEARCH FOR TRANSIENTS

The discovery of the accelerating expansion of the universe (Riess et al. 1998; Perlmutter et al. 1999) using Type Ia supernovae (SN Ia) has greatly motivated ever larger transient searches in broadband imaging surveys. The associated search pipelines have become increasingly complex in distributing enormous computing tasks needed to rapidly find new transients for spectroscopic observations, and in processing a wide range of data quality.

This chapter provides an overview of the search for Astronomical Transients, presenting the difference imaging pipeline from DES and SPLUS. This includes an explanation of how the pipeline operates, as well as the essential features that make it an effective tool for identifying transients. Specifically, this chapter focuses on the search for two main types of transients: supernovae and kilonovae.

2.1 Astrophysical Transients

The study of transients is a key area of Time-domain-Astronomy, as it focuses on the identification and characterization of objects that exhibit variability on short timescales, i.e., objects with their luminosity varying over time, such as Supernovae.

The process of supernova formation depends on the type of supernova, which is classified based on the characteristics of its optical spectra. There are two primary types of Supernovae:

- Core-Collapse Supernovae (CCSNe)
- Thermonuclear or Type Ia supernovae (SNe Ia)

Core-collapse supernovae occur when a massive star, with a mass greater than 8 times the mass of the Sun, runs out of fuel for nuclear fusion in its core and the star will no longer able to support itself by electron degeneracy pressure. This causes the core to

collapse inward, which releases a large amount of energy and leads to the formation of a neutron star or black hole. CCSNe are further classified into several subtypes based on the characteristics of their light curve and optical spectra, a graph of luminosity versus time as shown in figure xx.

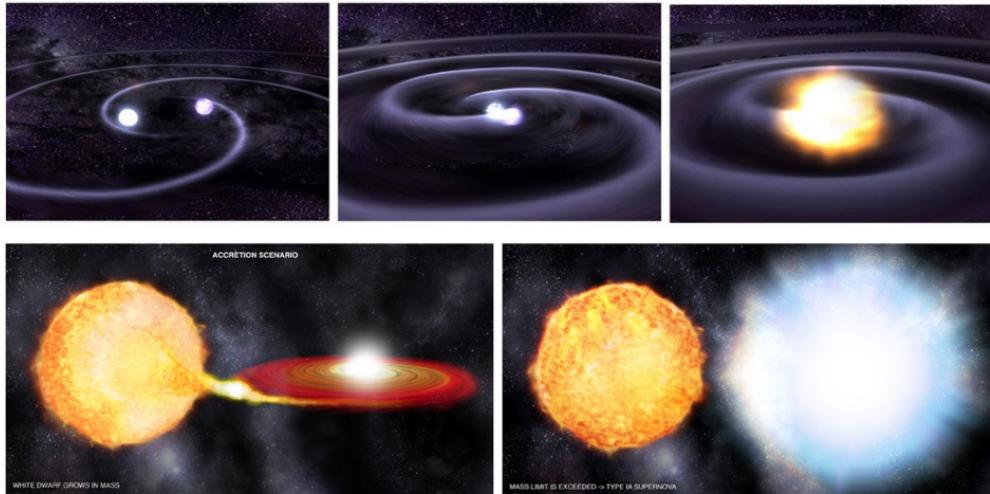


Figura 2.1 – Upper Panel: Two white dwarfs collide, causing a Type Ia supernova.
 Bottom panel: One white dwarf gaining mass from a companion star and eventually exploding into a Type Ia Supernova
Credit: NASA/Dana Berry, Sky Works Digital.

On the other hand, Type Ia supernovae occur when a white dwarf star, which is the remnant of a star that has exhausted its nuclear fuel, gains mass from a companion star (Fig.2.2 upper panel) or merges with another white dwarf (Fig.2.2 bottom pannel). There are several mechanism by which a supernova of type Ia can be created, but they all share the same physical process of As the white dwarf's mass approaches the Chandrasekhar limit of approximately 1.4 solar masses, it undergoes a runaway nuclear fusion reaction that completely disrupts the star, producing a bright explosion that can be as bright as billions of times the luminosity of our Sun, equivalent to the release of energy of 10^{44} Joules, which is equivalent to 100 octillion (10^{29}) nuclear bombs. If a supernova were to explode in our Milky Way galaxy, we would be able to see it from a distance of up to 10,000 light-years away.

On the other hand, Type Ia supernovae occur when a white dwarf star, which is the remnant of a star that has exhausted its nuclear fuel, gains mass from a companion star or merges with another white dwarf. As the white dwarf's mass approaches the Chandrasekhar limit of approximately 1.4 solar masses, it undergoes a runaway nuclear fusion reaction that completely disrupts the star (Figure ??), producing a bright explosion

that can be as bright as billions of times the luminosity of our Sun, equivalent to the release of energy of 10^{44} Joules, which is equivalent to 100 octillion (10^{29}) nuclear bombs. If a supernova were to explode in our Milky Way galaxy, we would be able to see it from a distance of up to 10 000 light-years away.

Another important type of transient event in astronomy is Kilonovae, which dates back to the 1970s when researchers first proposed that neutron star mergers could lead to observable electromagnetic radiation. However, it was not until the 2010s, with the development of advanced gravitational wave detectors such as LIGO and Virgo, that the first kilonova event was detected and confirmed in 2017.

Different from supernovae, kilonovae are caused by the merger of two compact and dense objects, such as neutron stars or a neutron star and a black hole. The merger creates a dense and rapidly rotating disk of ordinary matter, which is so hot and dense that it emits a large amount of radiation. Eventually, the binary system collides. These events produce gravitational waves and one of the most powerful electromagnetic explosions in the universe. An illustrative picture of how it might work is shown in Figure 1.1.



Figura 2.2 – Two Neutron stars collide, causing a Kilonova and Gravitational Waves. Credit: NASA/Dana Berry, Sky Works Digital <https://svs.gsfc.nasa.gov/12949>.

The goal of this thesis is to identify possible transients by integrating an AI into the current difference imaging pipeline. The next section will describe the difference imaging pipeline used to discover point-source transients in the Dark Energy survey and SPLUS.

2.2 Transient Detection Pipeline in DES-GW

In order to identify scientifically valuable transients, imaging surveys have historically adopted a manual approach, employing astronomers to visually inspect images facilitated by a difference imaging algorithm.

This work focus specifically on electromagnetic counterparts that emit in the optical wavelengths and are bright enough to be detected by existing instruments such as the Dark Energy Camera (DECam Flaugher et al. 2015) used in the DES-GW pipeline and the T80-South (T80S) used in the STEP pipeline.

2.2.1 Dark Energy Survey (DES)



Figura 2.3 – DeCam and the Blanco Telescope at the cerro tololo inter american obser-vatory (CTIO).

The Dark Energy Survey (DES) is a six-year optical imaging campaign with the goal of mapping hundreds of millions of galaxies , detecting numerous of supernovae and finding patterns of cosmic structure that could help scientists discover the nature of dark energy, which is accelerating the expansion of our universe.

DES comprises two multi-band imaging surveys:

- Wide area survey: Covers a wide-area of sky (5000 deg²) in the southern celestial hemisphere in the g,r,i,z and Y bands. Here, the exposure times are long and cumulative to detect a large number of very faint galaxies.
- Time domain survey: Covers a smaller area on the sky (27 deg²), but exposures are repeated a large number of times at regular, six night intervals over the course of the survey in the g,r,i,z bands, in order to detect Transients.

Multi-band imaging is a astronomical technique to take images of the sky across different wavelenght of light. A set of filters are placed in front of the camera to capture

light only within specific wavelength ranges, so that only the desired band of light can be observed. Each band covers a different range of wavelength spanning from 400nm to 180nm as shown in Figure 2.7.

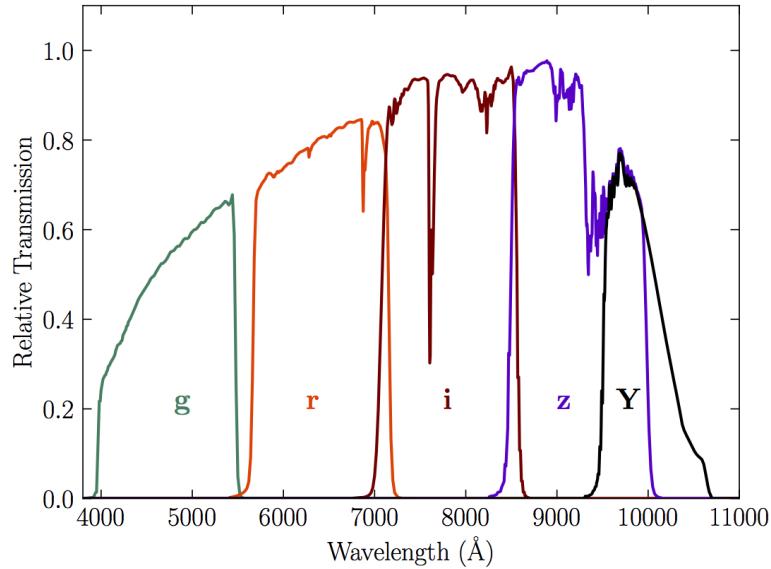


Figura 2.4 – Standard Bandpasses for the DECam grizY filters, including atmospheric transmission (airmass = 1.2) and the average instrumental response across the science CCDs. Credit: NOIRLAB SCIENTISTS' WEBSITE <https://noirlab.edu/science/programs/ctio/filters/Dark-Energy-Camera>.

Both of these multi-band imaging surveys uses an extremely sensitive 570-Megapixel digital Camera, Dark energy Camera (DECam; Flaugher et al. 2015). DECAM is mounted on the Blanco 4m telescope (2.3) at the cerro tololo inter american observatory (CTIO) and the data are processed by the DES data management system (Sevilla et al. 2011; Mohr et al. 2012; Desai et al. 2012) at the National Center for Supercomputing Applications (NCSA).

The DeCam is composed of 62 science image CCDs, each with 2.048 x 4.096 pixels and 8 CCds for guiding. The total active area covered by all working CCDs is around 2.7 deg²

2.2.2 DESGW - Pipeline

The DesGW is a group of researchers with focus on finding counterparts to the GW, the pipeline is integrated with the LIGO/VIRGO detector that triggers the pipeline into finding the electromagnetic counterpart using difference imaging. The pipeline is based on the DES Single-epoch processing pipeline (Morganson et al., 2018) and supernova difference imaging pipelines (Kessler et al. (2015)).

2.2.2.1 Single epoch processing

The first step of the image preparation is the single epoch(SE) pipeline, which occurs each night. Consists of an image correction stage and an object cataloging stage (known as FirstCut). A schematic overview of the complete single epoch processing of DESGW pipeline is shown bellow:

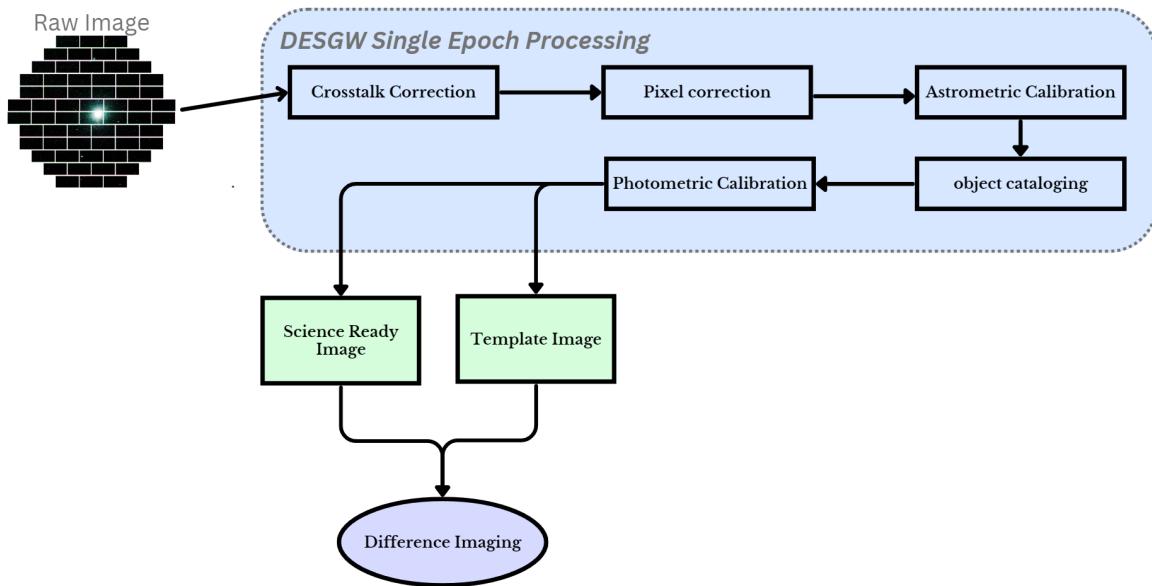


Figura 2.5 – A schematic overview of the complete single epoch processing of DESGW pipeline

1. Image correction: This steps aims to make the raw images science ready by applying a series of corrections and masking. A brief description of each correction are given bellow:
 - a) crosstalk corrections: Is an error that occurs when the electrical signals from one CCD detector leaks into an adjacent detection. This can result in errors in the pixel values of the adjacent detector affecting the quality of the data.
 - b) pixel corrections: A series of pixel-level corrections are applied to the images, including:
 - i. bad pixel masking: CCDs may contain a small number of Defective pixels that need to be masked out to prevent them from affecting the Data. In addition to masking out defective pixels, the SE pipeline also corrects for hot pixels, which are pixels with a high dark current, and for bleed trails, which are caused by bright sources saturating the CCDs.
 - ii. Flat fielding: corresponds to correcting the combined optical-system and CCD throughput at each pixel so that each pixel on the CCD would res-

- pond equally to a source with the same photon flux. Flat fielding removes the effect of the pixel-to-pixel sensitivity variations across the array as well as the effect of dust or scratches on the CCD window
- iii. a “brighter-fatter” correction: The brighterfatter effect is caused by charge building up on wellilluminated pixels and pushing away nearby charge, effectively broadening the PSF of bright objects.
 - iv. pixel nonlinearity correction: The pixel response of CCDs is non-linear at low fluxes and near saturation as determined from flat field images. A non-linearity correction is applied to the images to make the pixel response linear.
 - v. bias subtraction: The standard electronic bias corrections apply the overscan data taken at the end of each row readout and a average zero bias pattern. The overscan corrections accounts for bias variations at the time scale of individual line readouts while the zero bias correction is patterns that are static over times scales of a night. There are no suspected bias variations (often called pattern noise) on intermediate time scales. The average overscan value, with the highest and lowest excluded, is subtracted from each row.
 - vi. a conversion from DN to electrons: The images are converted from units of Digital Numbers (DN) to electrons, which is the physical unit of the CCD signal.

After this stage a catalog of the brighter objects are created using SExtractor, to be used in the Astrometric calibration.

- c) Astrometric calibration: This stage uses the positions of theses bright objects in the image to determine the opinting and orientantion of the telescope. This is done using SCAMP (Bertin, 2006), wich compares a catalog of knowing star positions to the positions of the stars in the image.
2. The FirstCut processing: Calculates an astrometric solution with SCAMP (see 6.1.2), performs bleed trail masking, fits and subtracts the sky background, divides out the star flat, masks cosmic rays and satellite trails, measures and models the point spread function (PSF), performs object detection and measurement using SExtractor, and performs image quality measurements.
 3. Photometric calibration: A photometric characterization provides a magnitude zero point (MAGZERO, MAGZPT) and photometric depth (PHOTDPTH) in the data product metadata header using the estimated PSF. (TERMINAR A DESCRIÇÃO)

The outputs of SE processing are the inputs to the difference imaging stage (diffimg). the Difference imaging will be discussed in details in the chapter ?? after we gave an

introduction to SPLUS transient detection pipeline.

2.3 SPLUS Transient detection pipeline



Figura 2.6 – On the upper left The interior of the T80-South Telescope and on the upper right T80-South Telescope dome at Cerro Tololo Inter-American Observatory (CTIO) in Chile (Bottom).

The Southern Photometric Local Universe Survey (SPLUS) is a brazillian-led imaging survey that covers a region of 9300 deg² of the celestial sphere in 12 optical bands using a dedicated 0.8m robotic telescope, T80-South. The telescope located at Cerro Tololo Interamerican Observatory (CTIO), Chile. Equiped with a large-format camera, with a field of view of 2 square degrees(The SouthernPhotometric Local Universe Survey(S-PLUS): improved SEDs, morphologies and redshifts with 12 optical filters - reference)

The Splus is divided in five sub-surveys, but only two of them are used in the Transient Extension Program (STEP):

- The main survey(MS): Covering an area of 800deg² with a single visit to each filed per filter. The MS strategy is motivated by the requirements set by the extragalactic science, with accurate photometric redshifts for objects down to $i = 21$ (Molino et al., in prep.), allowing the study of the local large-scale structure, star formation rates and stellar populations.

- Galactic Survey(GS): Covering an area of 1300 deg² in the milk Way plane in all 12 filters, including 2 galactic regions, the bulge and the disk, using the same strategy as the Main survey.

The other three sub-surveys are the ultra-short survey (USS), the marble field survey (MFS), and the variability fields survey (VFS). The SPLUS includes 12 bands, consisting of 5 Broad bands similar to those used in DES, with the additional of 7 narrow bands as displayed in Figure 2.7.

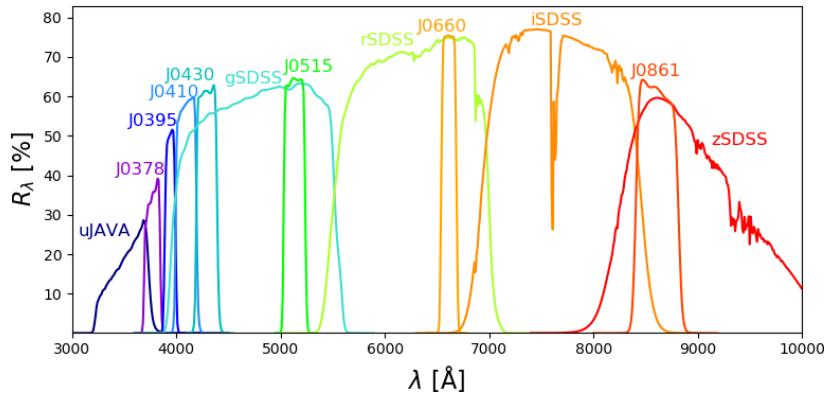


Figura 2.7 – 12 S-PLUS survey filter curves *Credit: NOIRLAB SCIENTISTS’ WEBSITE <https://datalab.noirlab.edu/splus/>.*

The telescope T80S is equipped with an optical imager, T80Cam-S, consisting of a 12-filter system. The detector used is a 9232×9216 $10\mu m - pixel$ array manufactured by the company e2v. The telescope plate scale at the detector is 0.55 arcsec/pixel and the FoV of the camera is 1.4×1.4 deg².

2.3.1 S-PLUS Transient Extension Program (STEP)

The STEP pipeline was developed with the objective of to report and characterize transients discovered by the Main Survey, using the photometric redshift as a way to obtain the luminosity distance, and other physical properties of the object together with the host galaxy and spectroscopy complementary data(when available).

The objective is to discover supernovae and other transients, for which the difference imaging algorithm, similar to the DESGW pipeline, is necessary. This chapter describes the main stages used to prepare the raw images for use in the difference imaging pipeline, making them science-ready.

2.3.1.1 Preprocessing and Reduction process of raw and template images

The pipeline starts each day by collecting raw images from S-PLUS Main Survey, captured using 4 broad band filters (g,r,i,z) and with 40 seconds of exposure time. These images are stored as fits files, and in addition to the images, a file containing R.A. and Dec.

coordinates for the center of each field retrieved is also saved on the CBPF-Server located in Rio de Janeiro, Brazil. After the images are transferred to our servers, an initial pre-preprocessing of the files is performed before they pass to the reduction process. It's important to highlight that the raw image does pass through a partial reduction pipeline in S-PLUS data flow, this partial reduction of each individual image includes overscan subtraction, trimming, bias subtraction, master flat-field correction, cosmetic corrections (e.g. satellite and aeroplane trails, cosmic rays and bad pixels masking), and fringing pattern subtraction (which is usually only necessary for the z filter). This process is performed using version 0.9.9 of the data processing pipeline jype, which was designed for the data reduction of J-PLUS and J-PAS (Cristobal- ' Hornillos et al. 2014). Besides all this corrections the image can still be called Raw, it is only after the STEP reduction pipeline that the image becomes scientifically-ready.

A schematic overview of the complete transient detection pipeline is shown below (Fig.2.8):

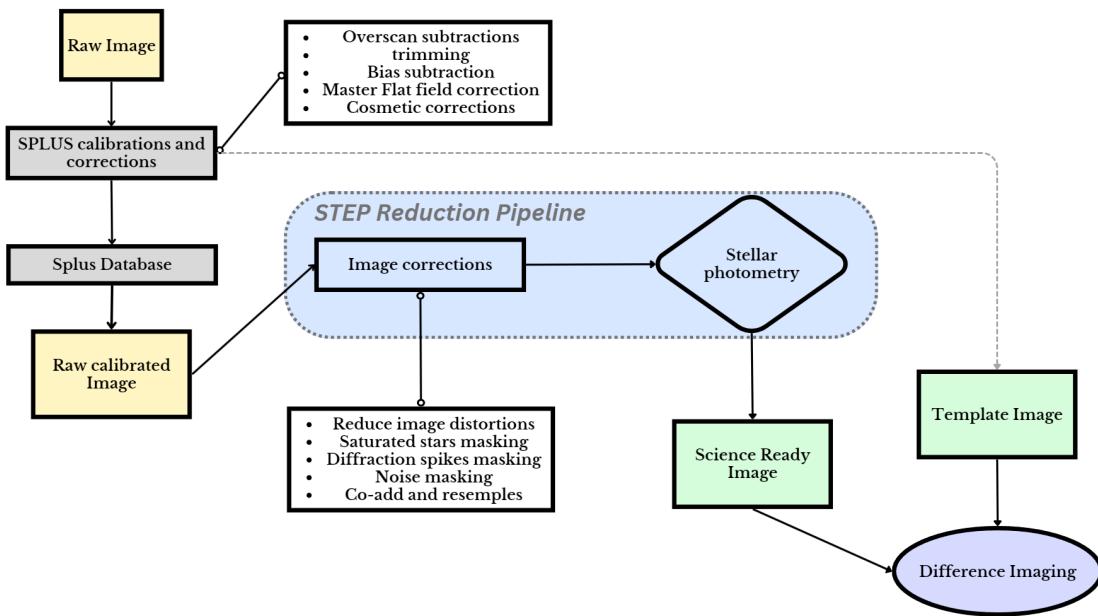


Figura 2.8 – A schematic overview of the complete transient detection pipeline

After transferring the image to the CBPF servers, the raw image passes through an initial astrometric calibration. Then, the calibrated image goes into the image corrections stage, which starts by reducing image distortions using a series of open-source software called IRAF (Tody, 1986). This includes a tool that matches a star catalog (usually obtained from the GAIA DR3 survey) with the stars in T80's camera data. Distortions in this context refer to any deviation from the ideal imaging properties of the telescope and camera system, such as geometric distortions, pixel irregularities, and variations in sensitivity across the detector. Then, we can fit a polynomial function to correct these

distortions. Once it's solved, the information of coefficients from the polynomial distortion is saved on the FITS file header.

On the next stage, for each collected image, the background levels and the flux on each pixel are measured to create a catalog of saturated stars and diffraction spikes that are presented on the images. This catalog is used to create a mask FITS image in order to remove these artifacts (Figure vv).

[Insert photo caption: "S - PLUS image with and without mask"]

After that, using the SWarp Bertin (2010), a noise mask and weight masks are created for a given image. It reassures the plate scale to be $0.55'' \text{pixel}^{-1}$ for all masks and raw images. That accounts for the fact that for pixels far away from the image center, the pixel's scale is not the same. The images are then co-added and re-sampled to a size of (10,000 x 10,000) px. These masks are used to correct and remove noise, diffraction spikes, and overbright stars. With these corrections made, the next step is to perform stellar photometry using DoPHOTS (Schechter et al., 1993).

Stellar photometry or PSF photometry is the measurement of the flux of bright objects by fitting an analytical function called the point spread function (PSF) to each bright point source object. Then, we compare this flux measurement with catalogs of other known surveys like Skymapper for known stars in order to get a zero point magnitude and calculate the absolute magnitudes of any interesting objects in our data. The process described is a key step, allowing the identification and characterization of sources in astronomical images.

Template images, as mentioned previously, are images from the sky taken in previous observations. These images are generated on a dedicated server in which we give as input a file containing a set of fields with their RA and DEC coordinates. They're made using cutout images available in public releases of different surveys. We mostly use data from the Dark Energy Survey (DES) when both overlaps; otherwise, we use data from other surveys like SKYMAPPER or PAN-STARRS. Once this process is finished, we retrieve those images back to our main server to perform the reduction process. The reduction process for template images is done in a similar way to those collected from S-PLUS, with the differences being the astrometric and distortion calibration steps. The template images already come with corrections for distortions and with astrometric calibration, so those steps in the reduction process are not necessary in this case.

Finally, the search image and the template are ready to go into the Difference imaging pipeline. A detailed discussion about the Difference imaging is presented in Chapter 3.

CAPÍTULO 3

DIFFERENCE IMAGING

There are a variety of astrophysical phenomena that can cause variations in the brightness and color of astrophysical objects, including explosive stellar death (supernovae, kilonovae, Gamma Ray Bursts, etc), less dramatic and powerful stellar variability (flares, pulsations), and variability arising from geometric effects (such as planetary transients and microlensing). The time scales for these phenomena range from seconds to years. When variations are sudden and terminal, as in the case of supernovae, or stochastic and unpredictable, as with stellar flares, they are referred to as "transients." To detect these transients, a process known as Difference Image Analysis (DIA) is employed. DIA involves comparing a nightly image with a previously observed sky image and subtracting them. This technique was first pioneered by Tomaney and Crotts (1996) and later formalized by Alard and Lupton (1998), and forms the basis of the two transient search pipelines described in chapter 2.

Templates are typically constructed as stacks of high quality sky images. These images must then be aligned with the nightly image, typically called the "search image". The alignment and calibration of these images are two critical steps that must be carefully performed to produce a high-quality subtracted image, known as a "difference image"(diff). For a careful calibration, it must account for the fact that the images used to build the template and the search images are taken principally within different atmospheric conditions (Zackay e Ofek 2017) generating variations in the quality of the images. Other complication is the construction of a proper template, typically templates are built by stacking tens of images taken under favourable sky conditions at different times. This improves the image quality but also mitigates issues related to variability in the astrophysical objects captured in the image. As a result, the template image is of higher quality than the search image, requiring it to be degraded to match the search image's point spread function (PSF) and scaled to match its brightness.

The central point of the difference imaging approach is to find a convolution kernel

“K” that matches the PSFs of two astronomical images, “I” (referred to as the search image) and T (referred to as the template). Mathematically, the equation that needs to be solved is the minimization of the equation below by solving for the kernel K.

$$\sum([T \otimes K](x, y) - I(x, y))^2 \quad (3.1)$$

To address the computationally expensive task of matching PSFs, the kernel can be decomposed in terms of simple functions, such as Gaussian functions and the method of least squares can be used to determine the best values for the kernel. The fitted solution of one search image can be determined in a short computational time.

In both the DES-GW and STEP pipelines, HOTPANTS, an open-source software that enables difference imaging on astronomical image data, is used. HOTPANTS is responsible for the alignment, calibration, and subtraction steps of the process. However, The DES-GW pipeline does the difference imaging per CCD, which means it performs a separate subtraction between the search image and each template, and then combine the difference images, rather than do a single subtraction on one combined template. While this approach is clearly slower than doing a single subtraction, it avoids potentially large PSF variations that could arise when combining templates taken in potentially very different observing conditions. On the other hand, The STEP does the subtraction in the full sized search image and after the subtraction a cut is made centralizing each candidate.

3.1 Artifacts

After the execution of difference imaging, it becomes possible to identify potential transients by selecting images that exhibit bright regions after the subtraction. An example of a transient, with its corresponding search, template, and difference images, are shown below (Figure.[?]):

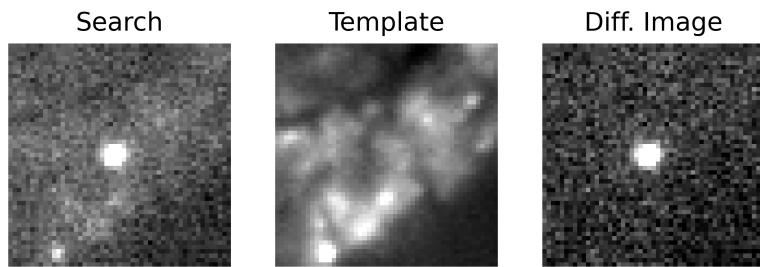


Figura 3.1 – left to right, The search, template and diff. image of a transient example from STEP.

However, this subtraction stage generates false positive candidates, also known as artifacts. Artifacts can arise due to poor PSF matching, imperfect alignment, defective pixels, near-field objects, and other inaccuracies in preprocessing.

Artifacts dominate the difference imaging data, with the majority of images produced daily being false positives. Since the primary objective of these two pipelines is to quickly find candidate objects for spectroscopic characterization, excluding these artifacts by visual inspection is not a viable option. In order to reduce the number of artifacts, each transient detection pipeline has unique steps after the difference imaging analysis. The following subsections will discuss these steps, along with the most common artifacts observed in each pipeline (DES-GW and STEP).

3.1.1 Artifacts on STEP

After the difference imaging is done, the PSF for the remaining possible sources is recalculated and the zero-point magnitude is calculated. Then, a catalog is generated containing all the remaining possible candidates, along with their unique identifiers, celestial coordinates, and magnitudes. However, this catalog is often contaminated with various types of artifacts that need to be removed.

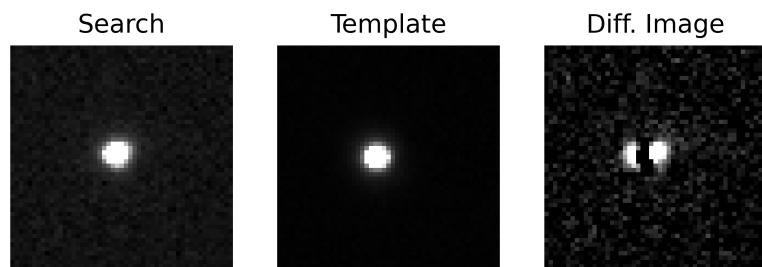


Figura 3.2 – Bad subtraction Artifact - STEP

- **Bad Subtraction:** The most common example of an artifact is known as a "bad subtraction". When the search and template images are not perfectly aligned, or if there are variations between the point spread functions (PSFs), the subtraction process can create undersubtracted and oversubtracted regions in the difference image.

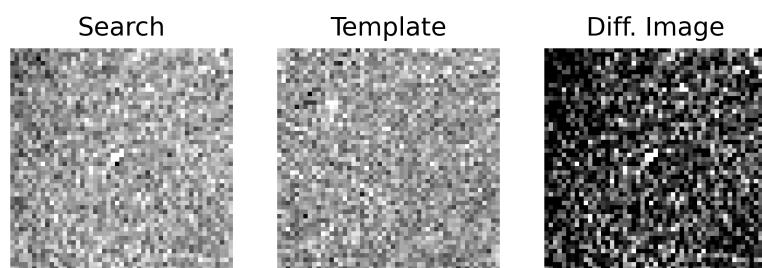


Figura 3.3 – Noisy Artifact - STEP

- Noisy search or template image: Another type of artifact occurs when realizations of Poisson noise produce groups of pixels that can resemble an object in the search image. In some cases, groups of under-fluctuations in the template image can also create the appearance of an object after subtraction.

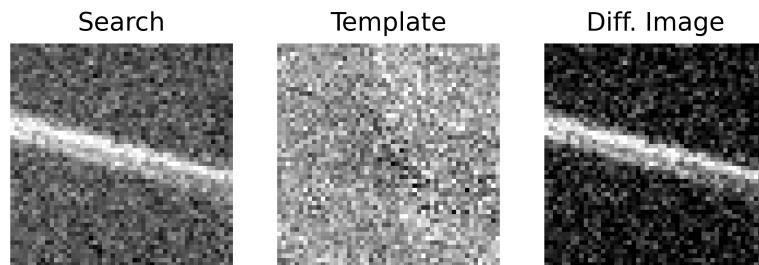


Figura 3.4 – Diffraction Spikes Artifact - STEP

- Diffraction Spikes Artifact: Another common false positive occurs due to diffraction spikes that were not eliminated in the preprocessing, and can produce a large bright line in the search image.

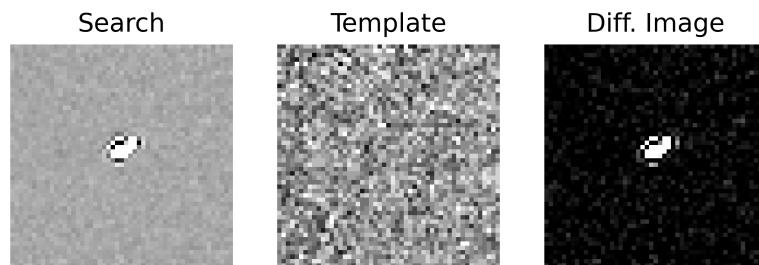


Figura 3.5 – Near Fiel object Artifact - STEP

- Near field object: A point-like source in the difference image could appear due to an asteroid that was detected in a previously empty patch of sky.

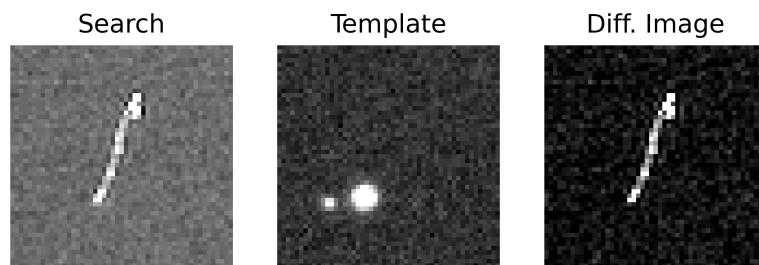


Figura 3.6 – Others Artifact - STEP

- Other Artifacts: There are others less common types of false positives, one of them are displayed, the explanation of how they occurs depend of each separated images.

3.1.2 Artifacts on DES-GW

After Difference imaging Analysis, SExtractor is used on the difference image to identify candidates. Objects detected are then filtered through a set of selection criteria listed in table 1 in K.Herner, Optical follow-up of gravitational wave triggers with DECam during the first two LIGO/VIRGO observing runs, 2020 [2].

Despite all of these selection criteria, several types of artifacts can still be labeled as "Transient candidates". Some of them are described below.

- Bad Subtraction (Badsub): As described before, the most common example of an artifact is known as a "Bad Subtraction," which can be interpreted as a real object by Source Extractor. The figure below exemplifies a bad subtraction from the DESGW pipeline.

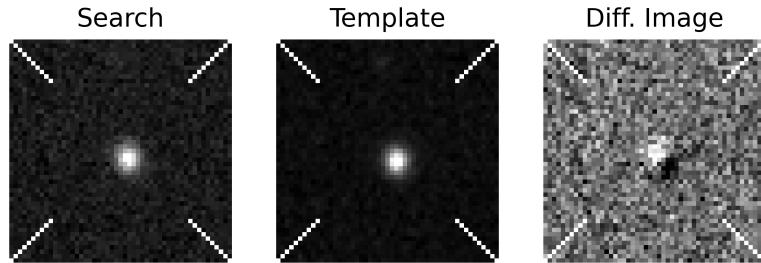


Figura 3.7 – Badsubtraction - DES-GW

- Preexisting point source: Another type of artifact is an object that was already visible in the template image but produced a bright spot in the difference image due to changing brightness (e.g., variable Milky Way stars or astrophysical transients in the template image).

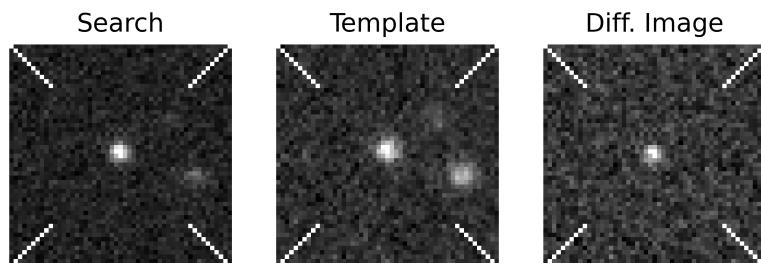


Figura 3.8 – Preexisting point - DES-GW

- Not an obvious Transient: These are images that seem to contain a host galaxy and a new object may appear in the search image, however, the resulting difference image is inconclusive. In most cases, this class contains galaxies with small variability in their centers rather than supernovae or kilonovae producing an obvious transient.

There are also other less common artifacts, two of them are Dark spot and noisy template.

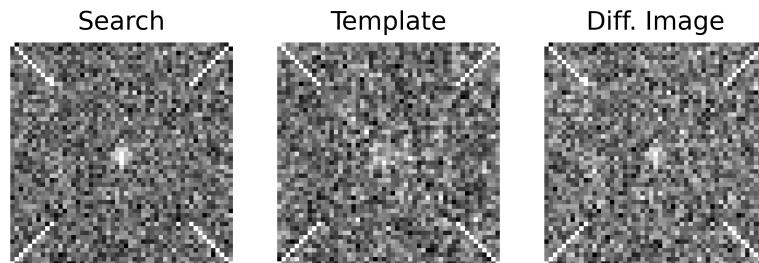


Figura 3.9 – Not an obvious Transient - DES-GW

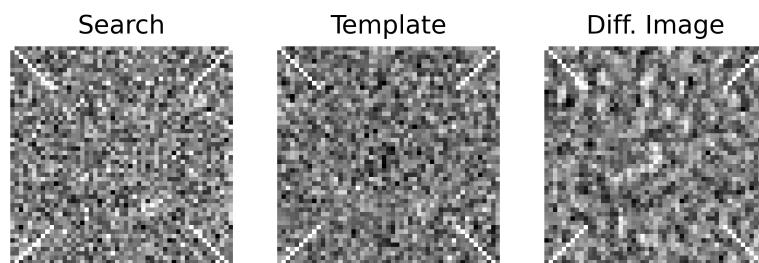


Figura 3.10 – Dark spot - DES-GW

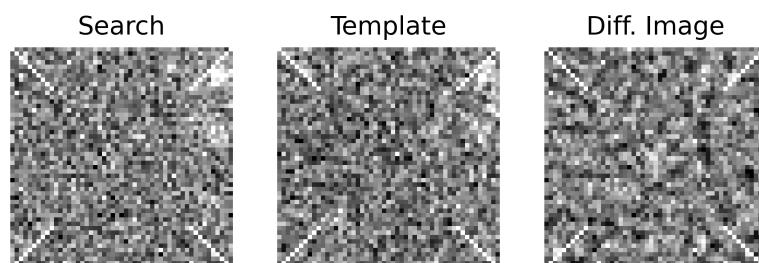


Figura 3.11 – Noisy template - DES-GW

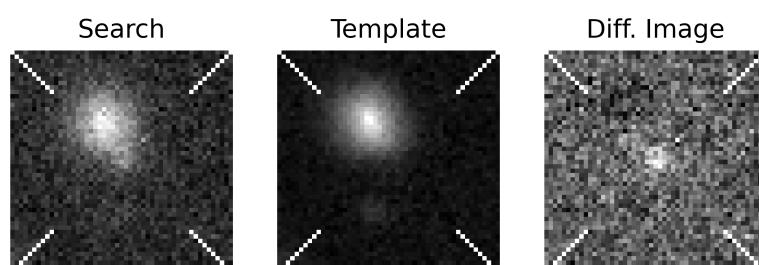


Figura 3.12 – left to right, The search, template and diff. image of a transient exemple from DESGW transient detection pipeline.

3.2 Candidate selection - trocar eventualmente

Knowing that most of the transient candidates are artifacts, it is necessary to have a step focused on removing these artifacts or at least selecting the real transients. There are two main methods used to achieve this in the pipelines of STEP and DESGW.

- Autoscan (DESGW) - Autoscan (Goldstein et al., 2015) is a machine learning tool specialized in removing the Bad Subtractions from the Dark Energy Survey dataset. which takes as input the template, search, and difference images and considers such items as 1) the ratio of PSF flux to aperture flux on the template image, 2) the magnitude difference between the detection and the nearest catalog source, and 3) the SExtractor-measured SPREAD MODEL of the detection. autoScan returns a number between 0, an obvious artifact, and 1, a high-quality detection.
- Visual inspection (STEP / DESGW) - This is the most straightforward way to select candidates. Essentially, a specialized astronomer is trained to identify real transients by visualizing the results on a Webpage.

There are some issues with relying only on the two methods mentioned above. Firstly, Autoscan only removes bad subtractions, while there are several different types of artifacts. Although bad subtraction is the most common type, with the upcoming O4 season, the number of images is expected to rise dramatically, necessitating a faster and more generalized solution. A similar issue arises with visual inspection in the STEP pipeline. Without an artifact-removing algorithm, astronomers would have to manually examine hundreds of images daily. This thesis aims to address this issue by proposing a deep learning model that can classify images as either transients or non-transients, thereby reducing the number of images that requires visual inspection.

3.3 STEP and DESGW Resumed Pipeline - trocar nome eventualmente

The transient detection pipeline in DESGW involves a complex data flow that begins with the triggers from LIGO/VIRGO, and the area of the sky with highest probability of the sky is and then the transient detection pipeline starts with

the acquisition of raw images from the DECAM and ends with the identification and characterization of potential gravitational wave (GW) counterparts. The pipeline consists of two main stages: the Single Epoch (SE) pipeline and the Difference Imaging Pipeline. The SE pipeline is performed each night and includes image correction to prepare raw images for scientific analysis, as well as cataloging of brighter objects for astrometric calibration. The Difference Imaging Pipeline uses a template image and a science-ready

image to perform a subtraction. The resulting difference image is then passed to a machine learning algorithm called Autoscan, as well as a convolutional neural network classifier (CNN) that will be presented in this thesis. The candidates identified by these methods are then selected for further analysis through visual inspection on the DESGW webpage.

The STEP pipeline begins by downloading raw calibrated images from the S-PLUS Main Survey database. These images then undergo corrections and PSF photometry in the reduction pipeline. The difference imaging analysis is performed next, which involves subtracting the template and search images in the full-sky image. The resulting images are then processed through DeepStep, which is other CNN also presented in this thesis. Finally, the candidates are selected through visual inspection on the STEP Webpage. The output of the pipeline is a list of candidates for transient astronomical phenomena. Overall, the STEP pipeline's goal is to detect transient astronomical phenomena using data from the T80 telescope.

As the field of Artificial Intelligence (AI) continues to evolve and the volume of data increases exponentially, the integration of Deep learning techniques into the pipelines has become increasingly prominent. In order to discuss the two Convolutional Neural Network classifiers developed in this thesis, a brief introduction to Deep Learning, CNNs, and the associated metrics for classifiers will be presented in the following section.

CAPÍTULO 4

A BRIEF INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is a field of computer science that aims to develop intelligent systems that can perform tasks that typically a human can do, such as learning patterns, problem solving, and even more elementary level, AI can be programmed just to tell the hours when you wake up and this act can still be considered AI.

The term "AI" was first coined in 1956 at the Dartmouth Conference. Since then, the field has grown rapidly, and AI has become one of the most important topics in today's world.

Machine learning (ML) is a subfield of AI that uses models built from Data, these models also called (Data Driven) enable computer systems to learn from data without being explicitly programmed.

Finally, Deep learning (DL) is a subset of machine learning and AI that involves the use of artificial neural networks to learn from data (Data Driven). However Deep learning is different from traditional machine learning in several ways. Traditional machine learning algorithms rely on custom-made features that are fed into the model as inputs, whereas deep learning algorithms can learn these features directly from the raw data, making them more flexible and adaptable to a wide range of tasks, including image recognition, speech recognition, natural language processing, and financial forecasting. For example, convolutional neural networks (CNNs) are commonly used for image recognition tasks because these convolutional layers have the ability to filter an image for a particular feature.

The history of Deep learning can be traced back to 1943, when Walter Pitts and Warren McCulloch created a computer model based on the neural networks([3]) (NN) of the human brain. Neural networks are a key component of deep learning. They are a type of algorithm that consists of layers of interconnected nodes, or neurons, that process and transmit information.

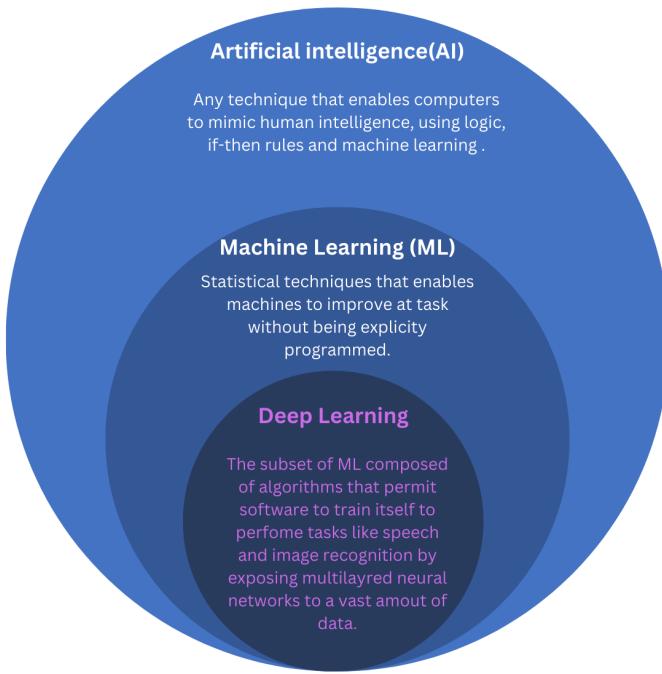


Figura 4.1 – The hierarchy with a resumed explanation of each branch of AI.

In the next subsections an introduction to the basics of NN, CNN, and the main problems faced when using Deep learning in Image classification. The Figure 4.1 illustrates the hierarchy, and a resumed explanation of each branch of AI.

4.1 Introduction to Neural Networks

Neural Networks (NN) are composed of neurons, where each neuron individually performs only a single computation. The power of a NN comes instead from the complexity of the connections these neurons can form.

The fundamental component of a Neural network is the individual neural, a diagram with a neuron (or unit) and one input are displayed below.

The input to a neuron is denoted as \mathbf{x} , and its connection to the neuron has a weight represented as \mathbf{w} . When a value flows through a connection, it is multiplied by the connection's weight. Therefore, the value that reaches the neuron from input \mathbf{x} is $w \times x$. A neural network "learns" by adjusting its weights, which includes a special kind of weight known as the bias, denoted as b . Unlike other weights, the bias does not have any input data associated with it. Instead, for this example, it is defined as 1 in the diagram and is added to the weighted input value to reach the neuron as b (since $1 \times b = b$). The bias enables the neuron to modify its output independent of its inputs.

The value outputted by the neuron is denoted as \mathbf{y} . To obtain the output, the neuron

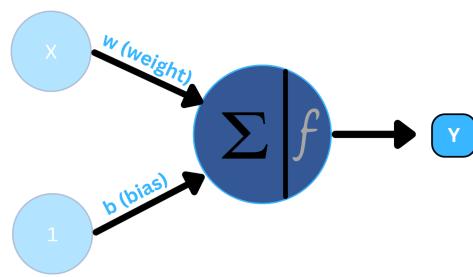


Figura 4.2 – Schematic of a neuron Network information flow. the input X are weightned by w and summed with the bias

sums up all the values it receives through its connections. Although a neuron's activation function can take on various forms, in this example, the activation function is linear, and it can be expressed as $y = w \times x + b$.

Neural networks typically organize their neurons into layers. A dense layer is formed by grouping linear neurons that share the same set of inputs. Even in a neural network with multiple neurons, the linear function is incapable of performing non-linear learning. Activation functions are utilized to add non-linearity to neurons replacing the linear function. These functions are applied to the summation and generate the artificial neuron's output signal. There are numerous activation functions available, each better suited for specific types of problems.

Ultimately, the model can be expanded to include multiple inputs, multiple layers of neurons, and different activation functions. Figure xx displays A schematic representing an example of a neural network with two inputs, three layers with three neurons in the first layer and two in the remaining layers, and two outputs.

In summary, for a multilayer neural network, each neuron has multiple input entries denoted as x_1, x_2, \dots, x_n . These inputs are weighted by their respective weights, represented as w_1, w_2, \dots, w_n , and the respective bias b_n is added. An activation function denoted as f_n is then applied to the weighted sum, and the result is passed on to the next layer. The weights are used to assign different relative importance to each input.

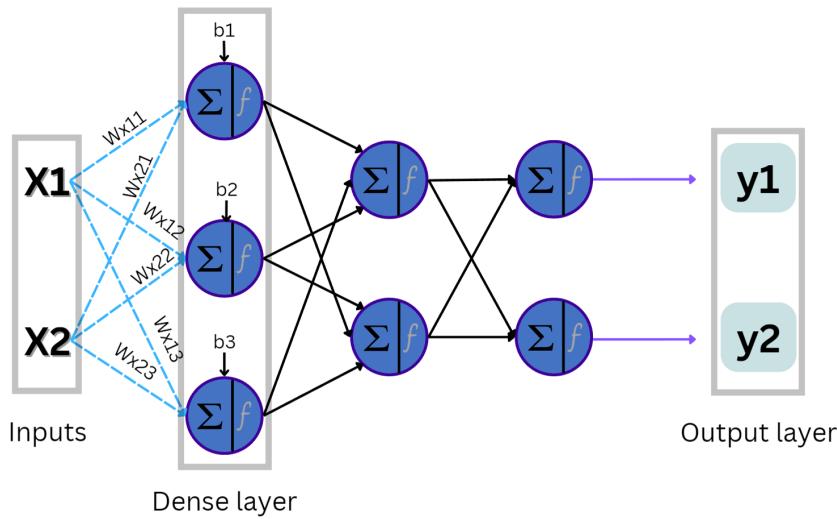


Figura 4.3 – A schematic representing an example of a neural network with two inputs, three layers with three neurons in the first layer and two in the remaining layers, and two outputs. The weights of each connections are denoted as w and the bias b .

4.2 Activation functions and Loss functions

One of the most widely used activation functions is the Rectified Linear Unit or ReLU. When a ReLU activation function is applied to a linear unit, the output becomes $\max(0, w \times x + b)$. This means that any negative value will be set to 0, and positive values will remain the same. In other words, the function is a straight line for positive inputs and 0 for negative inputs, which introduces non-linearity into the network.

There are other slight variations of the Relu function:

- SeLU (Scaled Exponential Linear Unit) is another activation function that has been found to perform well in deep neural networks. The main advantage of SELU is that we can be sure that the output will always be standardized due to its self-normalizing behavior.
- GeLU (Gaussian Error Linear Unit) is a more recent activation function that is based on the Gaussian cumulative distribution function. The GELU nonlinearity weights inputs by their percentile, rather than gates inputs by their sign as in ReLUs. Consequently the GELU can be thought of as a smoother ReLU.

Activation functions are also used in the last layer of a neural network. The activation function in the output layer shapes the output values of the network to match the specific problem the network is trying to solve. For classification tasks, the three most used activation's function:

- Sigmoid is an activation function that maps input values to a range between 0 and 1. It is often used in the output layer of a neural network for binary classification tasks, where the goal is to produce a probability value between 0 and 1.
- Softmax is an activation function that maps input values to a probability distribution over multiple classes. It is often used in the output layer of a neural network for multi-class classification tasks, where the goal is to produce a probability value for each categorical class that sums to 1 for each input vector.
- tangent hyperbolic (Tanh) is an activation function that maps input values to a range between -1 and 1. It is similar to the sigmoid function but has a symmetric range around 0.

for regression tasks, some commonly used activation functions include:

- linear activation function, also known as the identity function, represents a straightforward mapping where the output is equal to the input without any transformation or non-linearity. This activation function is commonly used when the desired output is expected to be a continuous value.
- ReLU (Rectified Linear Unit) is a widely utilized activation function in neural networks, particularly in deep learning models. It introduces non-linearity by mapping negative input values to zero and maintaining positive values unchanged.
- Leaky ReLU is a variation of ReLU that addresses the limitation of possible "dead neurons" (i.e., neurons that never activate) observed in standard ReLU. In Leaky ReLU, negative inputs are assigned a small negative value, thus a non-zero gradient.

This section explains the process of designing a neural network architecture and the principles of neuron computation. However, it does not provide any guidance on how to specify the particular problem that the network needs to solve. This crucial task is accomplished by utilizing a loss function. A loss function is a mathematical function that measures how well a neural network's predictions match the expected values (True values) of the data its is being trained on.

The value of the loss function is used to adjust the network's weights and biases during training, thus choosing the most important features that yield the best results for the specific task it was designed. There are many different types of loss functions, each designed for specific types of problems. Here are some examples of commonly used loss functions:

- Mean Squared Error (MSE) - This is a commonly used loss function for regression problems. It is defined as the average of the squared differences between the predicted output and the true output. The formula for MSE is: $MSE = \frac{1}{n} \times \sum(y_i - \hat{y}_i)^2$

where n is the number of data points, y is the true output, and y' is the predicted output.

- Mean absolute Error (MAE) - The MAE loss function is also used to evaluate the performance of regression models, and it is especially useful when dealing with outliers in the dataset, as it is not as sensitive to extreme values as other loss functions. The formula for MAE is as follows: $MAE = \frac{1}{n} \times \sum |y_i - y'_i|$ where n is the number of data points, y_i is the true output, and y'_i is the predicted value. The lower the MAE value, the better the model's predictions match the actual values.
- Binary Cross-Entropy - This is a loss function that is commonly used for binary classification problems. It measures the difference between the predicted output and the true output using the cross-entropy formula. The formula for binary cross-entropy is: $BCE = \frac{1}{n} \times (y \times \log(y') + (1 - y) \times \log(1 - y'))$ where n is the number of data points, y is the true output (either 0 or 1), and y' is the predicted output (between 0 and 1).
- Categorical Cross-Entropy - This is a loss function that is used for multi-class classification problems. It measures the difference between the predicted output and the true output also using the cross-entropy formula. The formula for categorical cross-entropy is: $CCE = \frac{-1}{n} \times \sum_i \sum_j^{nclass} (y_{ij} \times \log(y'_{ij}))$ where n is the number of data points, y_{ij} is the true output for the ith data point and the jth class, and y'_{ij} is the predicted output for the ith data point and the jth class.

The process of training a neural network is essentially finding ways to reduce the value of the loss function, as will be shown in the next section. This demonstrates the importance of choosing a loss function and an activation function of the output layer aligned with the objectives of the neural network, because all optimization will be done on it. If the loss function and the objective of constructing the network are not aligned, it is possible that the network learns very well, i.e., greatly reduces the value of the loss, but has poor results in its real application. Some usual combinations of Loss functions and activation function on the output layer for each type of problem are listed below:

- Binary Classification: For binary classification problems where the output should be a probability value between 0 and 1, the Sigmoid activation function is commonly used in the output layer with The binary cross-entropy as loss function (Sigmoid + Binary cross entropy).
- Multi-class Classification: For multi-class classification problems where the output should be a probability distribution over multiple classes, the softmax activation function is commonly used in the output layer and The categorical cross-entropy as loss function (Softmax + Categorical cross entropy).

- Regression: For regression problems where the output should be a continuous value, the linear activation function is commonly used in the output layer. The mean squared error (MSE) loss function is commonly used in this case. However, other loss functions such as mean absolute error (MAE) and mean squared logarithmic error (MSLE) can also be used depending on the nature of the problem (linear/Relu + MAE/MSE).

Overall, the choice of activation function and loss function is one of the most important part of designing an artificial neural network and should be carefully considered based on the specific requirements of the problem. For this thesis, we developed a binary classifier, then the Sigmoid + Binary cross entropy combination was used.

4.3 Neural Network's Learning Process:

In the previous section was described how choosing the activation function and loss function can set the NN to solve determinated problem, but it was not present how the NN actualy learns how to solve it. Training an Artificial Neural Network (ANN) involves iteratively updating the weights and biases of the network to minimize the loss function, which represents the difference between the predicted output and the actual output. This process is accomplished using a technique known as backpropagation, which also involves choosing an optimizer.

The training process can be broken down into four main steps, which are summarized in Figure XX:

1. **Forward propagation (Forward pass):** Starts by randomly initializing the weights and biases of the network, then During the forward pass, the input data is fed through the network, and the output is computed using the current weights and biases.
2. **Loss computation:** The difference between the predicted output and the actual output is computed using a loss function. This value is also known as the error of the model. Then the loss score is used as a feedback signal to update parameter in the backpropagation step
3. **Backward propagation (Backward pass/Backpropagation):** Backpropagation is a key algorithm for training neural networks, as it allows us to calculate the gradient of the loss function with respect to the weights of the network. This gradient is then used to update the weights using an optimization algorithm, such as SGD, Adam or RMSProp, in order to minimize the loss function (error) and improve the network's predictions. This error is then propagated **back** through the network in reverse order, with the gradient of the loss with respect to each weight

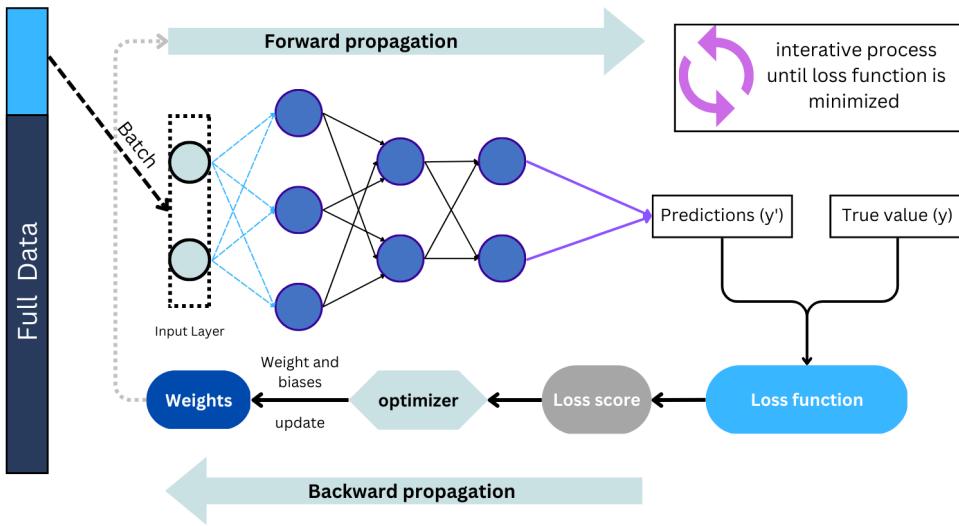


Figura 4.4 – A schematic dataflow complete the legend...

being calculated at each layer. The calculation of the gradient is done using the chain rule of calculus. The derivative of the loss function is its slope which provides the direction that is considered for updating (changing) the values of the weights and biases.

4. **Repeat:** The input data is fed again to the NN, but now the output prediction is calculated based on the new weights. This process continues until a global minimum in the gradient is found, therefore the loss function converges, and the performance of the neural network has reached a plateau thus further training will not improve the performance significantly. In practice, a stopping criterion, such as a maximum number of epochs or a minimum change in the loss function, is used to terminate the training process.

Instead of updating the weights after every input is passed through the network, we typically update the weights after a batch of inputs have been processed. This is because updating the weights after every input can be computationally expensive and can lead to overfitting. The size of the batch (also known as Batch Size) is a hyperparameter that needs to be chosen carefully, as smaller batches can lead to faster convergence but can also make the optimization process more noisy.

Another important parameter in the Training process is learning rate, which determines the step size of the weight and bias updates. A high learning rate can lead to unstable training (fig.4.5), while a low learning rate can lead to slow convergence (fig 4.5). The simplest case of the learning rate usage is on the gradient descent optimization

(GD) algorithm, a learning rate of 0.01 means that on each training iteration the weight parameters are updated only 1% of the gradient term (Equation 4.3).

$$w_i = w_i - \alpha \frac{\partial Loss}{\partial w_i}$$

$$b_i = b_i - \alpha \frac{\partial Loss}{\partial b_i}$$

In summary, The hyperparameters of batch size, learning rate, and choice of optimizer play a crucial role in the effectiveness and training performance of the backpropagation algorithm.

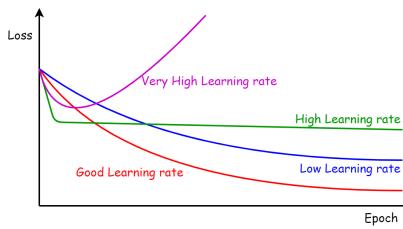


Figura 4.5 – Learning rate complete the label

An optimizer is a function or an algorithm that updates the weights, biases and learning rates of the neural network. The problem of choosing the right weights for the model is a complicated task, as a deep learning model generally consists of millions of parameters. It's necessary to choose carefully the optimization algorithms that best converges the loss to the global minima. Some of the most frequent optimizers are:

- **Gradient Descent (GD)** is the simplest optimization algorithm used to update the weights of a network by calculating the negative gradient of the loss function with respect to each weight. In this algorithm the entire training dataset is used to compute the gradient of the loss function, leading to a slow convergence.

$$w_i^{t+1} = w_i^t - \alpha \cdot g_t$$

Where $g_t = \nabla_i L(w_i^t)$, that is, the gradient of the loss function with respect to the i'th weight on step t, and α is the learning rate.

- **Stochastic Gradient Descent (SGD)** is a popular variation of Gradient descent optimization algorithm. Like GD, it updates the weights using the negative gradient of the loss function for each weight. However, instead of using the entire training dataset, SGD randomly selects a single training sample, computes the gradient based on that example, and performs a parameter update. While SGD can be faster for large datasets, it may have high variance due to the noisy updates, resulting in slower convergence.

- **Batch Gradient Descent (Batch GD)** is another variant of GD where the training dataset is divided into batches. The gradients are computed for each batch, and the model parameters are updated based on the average gradient of all the batches. In other words, Batch GD performs a single update per iteration by considering all the training examples within each batch.
- **Mini-Batch Gradient Descent (Mini-Batch GD)** is a combination of Batch GD and SGD. Instead of calculating the gradient using a single randomly selected sample (like SGD) or the entire training dataset (like GD), Mini-Batch GD computes the gradient based on a small randomly chosen subset of training data, also known as a minibatch. The model parameters are then updated based on the average gradient of the current minibatch. Mini-Batch GD combines the advantages of Batch GD (more stable convergence) and SGD (faster updates).
- **SGD with momentum** is an algorithm that improves the basic GD algorithm by using information from previous gradients to accelerate convergence towards the relevant direction and reduce fluctuations in irrelevant directions. This is achieved by introducing a new variable called the momentum vector, which is subtracted from the local gradient term. This momentum term (also known as mean or First momentum) represents the average of the previous gradients. With the objective of simulating some kind of friction and stop the momentum being too high, the algorithm introduces a new hyperparameter β which is defined between 0 (high friction) and 1 (No friction).

$$\begin{aligned} w_i^{t+1} &= w_i^t - \alpha \cdot m_t \\ m_t &= \beta \cdot m_{t-1} + (1 - \beta)g_t \end{aligned}$$

Where the initializing value of m_t is zero.

- **Adaptive Moment Estimation (ADAM)** is an optimizer that computes adaptive learning rates for each parameter in the network, unlike SGD that maintains a single learning rate through training, ADAM optimizer updates the learning rate for each network weight individually. It uses the first momentum (m) and the second momentum (v , also known as uncentered variance) of the gradients to adapt the learning rate. The second momentum uses the uncentered variance from previous gradients to adapt the learning rate for each parameter by considering both the magnitude and the variation of the gradients. This is achieved by introducing a new hyperparameter, the decay rate β_2 , which controls the weight given to the current squared gradient compared to the historical squared gradients. A smaller value of β_2 leads to faster decay of the estimates, while a larger value assigns more importance to older squared gradients. By maintaining an adaptive learning rate based on the

second moment estimate, ADAM can converge faster, and handle sparse gradients encountered in many machine learning tasks. The weight update is given by:

$$\begin{aligned} w_i^{t+1} &= w_i^t - \alpha \cdot \frac{m'_t}{v'_t} \\ m'_t &= \frac{m_t}{1 - \beta^t} \\ v'_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned}$$

m'_t and v'_t are corrections to the first and second momentum, which are defined as:

$$\begin{aligned} m_t &= \beta \cdot m_{t-1} + (1 - \beta)g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2)g_t^2 \end{aligned}$$

Accordingly to [4], the suggested values are $\beta = 0.9$, $\beta_2 = 0.999$ and $\alpha = 0.002$. the initializing value of m_t and v_t are, by default, zero.

4.4 Problems related to training a Artificial Neural Network

Identify transients using AI is a complex task that requires a deep neural network with multiple layers, which in turn involves a large number of derivatives. Training a Deep Neural Network can led to several training related problems, In this section, It will be described each problem and possible solutions.

4.4.1 Exploding gradient and vanishing gradient

here are two main problems that may arise during training that are related to weight derivatives. In a network of n hidden layers, n derivatives will be multiplied together. If the derivatives are large, then the gradient will increase exponentially as it propagates down the model, resulting in the problem of the exploding gradient. Alternatively, if the derivatives are small, then the gradient will decrease exponentially as it propagates through the model, resulting in the vanishing gradient problem.

In the case of exploding gradients, the accumulation of large derivatives leads to a very unstable model that cannot learn effectively. The large changes in the model's weights make the network highly unstable, and extreme weight values may cause overflow, resulting in NaN weight values that can no longer be updated. On the other hand, the accumulation of small gradients leads to a model that is incapable of learning meaningful insights. The weights and biases of the initial layers, which tend to learn the core features from the input data (X), will not be updated effectively. In the worst-case scenario, the gradient will be zero, which will stop the network from further training.

To better understand these gradient problems, consider a deep neural network with L layers. The output of the i -th layer is denoted as h_i , and the weights of the i -th layer

are denoted as W_i . The activation function of the i -th layer is denoted as f_i and f'_i its derivative. The output of the Neural Network is denoted as y .

To update W_i its necessary to compute its gradient with respect to the loss function.

$$\begin{aligned}\frac{\partial \text{Loss}}{\partial w_i} &= \frac{\partial \text{Loss}}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_i} \\ h_i &= f(W_i * h_{i-1} + b_i) = f(z_i) \\ h_{i-1} &= f(W_{i-1} * h_{i-2} + b_{i-1}) = f(z_{i-1})\end{aligned}$$

W_i and b_i are the weight matrix and bias vector of the i -th layer, respectively, and h_{i-1} is the output of the $(i-1)$ -th layer. Using the chain rule, we can rewrite this as:

$$\begin{aligned}\frac{\partial \text{Loss}}{\partial w_i} &= \frac{\partial \text{Loss}}{\partial h_i} \cdot \frac{\partial h_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial w_i} \\ \frac{\partial \text{Loss}}{\partial w_i} &= \frac{\partial \text{Loss}}{\partial h_i} \cdot \left(\frac{\partial h_i}{\partial h_{i-1}}\right) \cdot \left(\frac{\partial h_{i-1}}{\partial h_{i-2}}\right) \cdot \dots \cdot \left(\frac{\partial h_2}{\partial h_1}\right) \cdot \left(\frac{\partial h_1}{\partial w_1}\right)\end{aligned}$$

As we can see, the gradient is a product of the gradients of the activation functions of each layer. If the activation function has a derivative that is less than 1, then the gradient will become very small as it is multiplied by each subsequent layer, leading to the problem of vanishing gradients. On the other hand, if the activation function has a derivative with high values, then the gradient will increase until it explodes to very high values.

One solution to the problem of vanishing gradients is to use activation functions that have a derivative that is greater than 1, such as the Rectified Linear Unit (ReLU) function. Another solution is to use normalization techniques, such as batch normalization, which can help to reduce the problem of vanishing gradients. The problem of exploding gradients can be addressed by using techniques such as gradient clipping, which involves clipping the gradients to a maximum value, or by reducing the learning rate during training.

4.4.1.1 Overfitting and Underfitting

Overfitting and underfitting are two of the most common problems in deep learning. Overfitting occurs when the model is trained excessively on the training data, causing it to learn irrelevant and specific patterns from the training dataset. As a result, the model performs very well on the training data, but poorly on the test data because it was unable to generalize the learned patterns to unseen data. One indication of overfitting is when the training error value is much lower than the Validation error, indicating that the model is losing the capacity to generalize to the validation set. This often caused when there are too many model parameters, or not enough training data, or when the model is trained for too many epochs.

Underfitting occurs when the model is too simple to capture the underlying patterns or features in the data. In this case, the model may perform poorly on both the training

and test data, since it is unable to capture the complexity of the data. Underfitting is often caused by having too few model parameters, not enough training data, or training the model for too few epochs. There are a few solutions to address these problems:

- L1 regularization, also known as Lasso regularization. L1 regularization adds a penalty term to the Loss function that is proportional to the absolute value of the model's weight coefficients. The penalty term has the objective of reducing the magnitude of the model's weights, effectively eliminating the impact of irrelevant or noisy features.

$$\text{Loss}(L1) = \text{Loss} + \lambda \sum_{j=1}^N |W_j| \quad (4.1)$$

- L2 regularization, also known as Ridge regularization. L2 regularization adds a penalty term to the cost function that is proportional to the squared magnitude of the model's weight coefficients. The penalty term encourages the model to reduce the magnitude of its weights, but unlike L1 regularization, it does not typically result in weights becoming exactly zero. Instead, it shrinks the weights towards zero, effectively reducing their impact on the model's predictions.

$$\text{Loss}(L2) = \text{Loss} + \lambda \sum_{j=1}^N (W_j)^2 \quad (4.2)$$

- Dropout: A dropout layer randomly drops out some of the neurons in the model during training, effectively creating an ensemble of models that share weights.
- Data augmentation: increases the size of the training set by applying random transformations to the input data, such as cropping, scaling, or rotating the images.

In summary to mitigate the overfitting problem one common solution is to use regularization techniques such as L1 or L2 regularization, or dropout. Regularization helps to prevent overfitting by adding a penalty term to the loss function that encourages the model to have smaller weights or biases, which can help to reduce the complexity of the model and prevent it from memorizing the noise in the data. In order to avoid the Underfitting the most common solutions is to increase the complexity of the model by adding more layers, neurons, or using a more powerful architecture.

4.5 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of deep neural network (DNN) that have proven to be effective in a wide range of image and video recognition tasks, including image classification, face recognition, object detection, and more. The main difference between a CNN and an ordinary NN is that the CNN was specifically developed

to process images as input. The neurons used in a CNN are arranged in three dimensions: width, height, and depth. It is important to note that the term "depth" here refers to the third dimension of an activation volume, rather than the total number of layers in a neural network.

The idea behind using a CNN is based on the principle of convolutions, which is a mathematical operation used for signal processing. By applying convolutional filters to the input image, the CNN is able to extract features such as edges, corners, and textures. These feature extractions are based on the three main types of layers used to build a ConvNet:

- Convolutional Layer: Filters an image for a particular feature and, by using a ReLU activation, can detect that feature within the filtered image.
- Pooling Layer: Condenses the image to enhance the features.
- Fully-Connected Layer: Learns how these features can be used to solve the determined problem.

The convolutional layer is the first layer while the FC layer is the last. From the convolutional layer to the FC layer, the complexity of the CNN increases. The layers are arranged in such a way that they detect simpler patterns first, such as lines and curves, and gradually move towards more complex patterns, such as faces and objects.

4.5.1 Convolutional Layer

The convolutional layer is the core building block of the CNN. It carries the main portion of the network's computational load and is responsible for filtering and detecting features from an input image. In the Convolutional Layer, the input image is first convolved with a set of filters, also known as kernels, designed to detect specific patterns in the image. The filter slides over the input image, and at each position, it computes the dot product between the filter and the corresponding region of the input image. The final output from the series of dot products is known as a feature map or convolved feature.

The filter is typically a small matrix, such as a 3x3 or 5x5 matrix, and the number of filters used in the Convolutional Layer is a hyperparameter that can be adjusted during the training process. Each filter in the Convolutional Layer learns to detect a different pattern in the input image, such as edges, corners, or textures.

To illustrate this convolved operation, let's assume we have an input image of size 51x51x3 (where 51 is the width and height of the image, and 3 is the number of color channels: red, green, and blue). We also have a Convolutional Layer with a filter size of 5x5, and let's say we are using a single filter. To apply the filter to the input image, we slide the filter over the image one pixel at a time, computing the dot product between the filter and the corresponding 5x5 region of the input image at each position. We then

store the result of each dot product in a new matrix, which represents the output of the Convolutional Layer.

Here's an example of how this would work for the first few positions of the filter on the input image:

- At position (0,0), we compute the dot product between the filter and the 5x5 region of the input image starting at (0,0), which gives us a single output value.
- We then move the filter one pixel to the right and compute the dot product between the filter and the 5x5 region of the input image starting at (1,0), which gives us another output value.
- We continue sliding the filter one pixel at a time until we reach the end of the first row of the input image.
- We then move the filter down one row and repeat the process until we have computed the dot product for every position of the filter on the input image.

After we have applied the filter to the input image, we obtain a new matrix with dimensions 47x47x1. This is because each position of the filter produces a single output value, and we slide the filter over the input image until we reach the edges. The resulting matrix represents the output of the Convolutional Layer for this single filter. Note that in practice, a Convolutional Layer typically uses multiple filters to extract multiple features from the input image. Each filter produces its own output matrix, and these matrices are stacked together to form the final output of the Convolutional Layer. Additionally, the output of the Convolutional Layer is often passed through an activation function, such as the ReLU function, to introduce non-linearity into the model and detect these extracted features.

Figure 4.6 demonstrates how the convolutional operation for two 3x3x3 filters act on an Input image (tensor). The input tensor of shape (w,h,3) is convolved with two kernels, the filter slides over the input image, and at each position, it computes the dot product between the filter and the corresponding region of the input image producing two values on the feature map, one value for each filter that are stacked together.

Some of the most important parameters to choose when using a convolutional neural network are:

- Number of filters: The number of filters in a Convolutional Layer is a hyperparameter that determines the number of patterns or features that the layer can learn from the input image. Increasing the number of filters increases the capacity of the network to learn complex features, but also increases the number of parameters in the network, which can lead to overfitting. Therefore, the number of filters should be chosen based on the complexity of the task and the available computational resources.

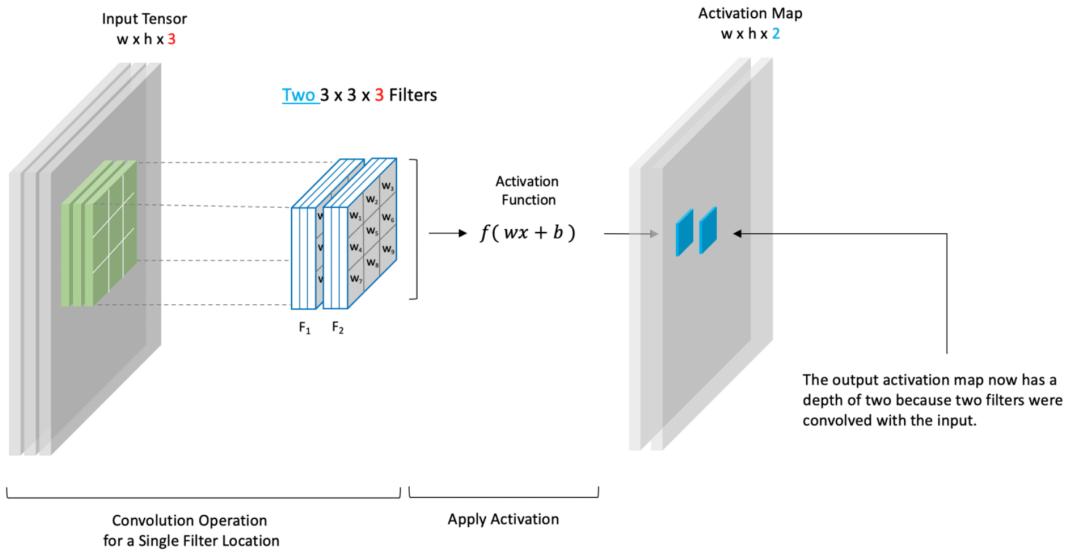


Figura 4.6 – Example of 3D convolutional operation on a RGB input image

- **Stride:** The stride is the step size used by the filter when sliding over the input image (in our example the stride was set to one). A larger stride reduces the spatial size of the output feature map, which can be useful for reducing the computational cost of the network. However, a larger stride also reduces the spatial resolution of the output feature map and can lead to loss of important features. A smaller stride can preserve more details in the feature map, but increases the computational cost of the network.
- **Padding:** Padding refers to the amount of zeros added to the border of the input image to ensure that the filter can slide over the entire image. Without padding, the output feature map will be smaller than the input image, which can lead to loss of information at the borders. There are two types of padding: same padding and valid padding. Same padding adds zeros to the border of the input image so that the output feature map has the same spatial size as the input image. Valid padding, on the other hand, does not add any padding and the output feature map is smaller than the input tensor.

One advantage of using these layers is that there is no need to pre-process the image to identify patterns that can help the model, as CNN can do this automatically. The weights connecting each of these features determine the most important features for solving the presented task.

4.5.2 Pooling Layer

After a convolutional layer and ReLU activation function, the resulting feature map is often permeated with zeros and dead pixels that do not add any additional information. To address this, a pooling layer is used to reduce the spatial dimensions of the feature map while retaining the most important features. There are several types of pooling, including Max Pooling and Average Pooling.

tion to the problem, thereby costing time and memory computacionality. In practice, the Convolutional Layer is often followed by a Pooling Layer, which is used to reduce dimensionality of the feature maps, thus condensing the filtered (detected) information, and helps to make the model more robust to variations in the input image.

The most common type of pooling is max pooling, which takes the maximum value within a small region of the feature map and replaces the entire region with that maximum value. This reduces the spatial size of the feature map while preserving the most important features.

The other common type of pooling is average pooling, which instead of taking the maximum value as the max pooling, it takes the average value within a small region of the feature map and replaces the entire region with that mean value.

Global pooling is a pooling layer that operates globally over the entire input feature map, instead of using a local window like max pooling or average pooling. Global pooling can be either Global max pooling or Global average pooling. Global max pooling selects the maximum value across the entire feature map, while global average pooling takes the average value across the entire feature map. Global pooling is effective in reducing the dimensionality of the feature map while retaining the most important features.

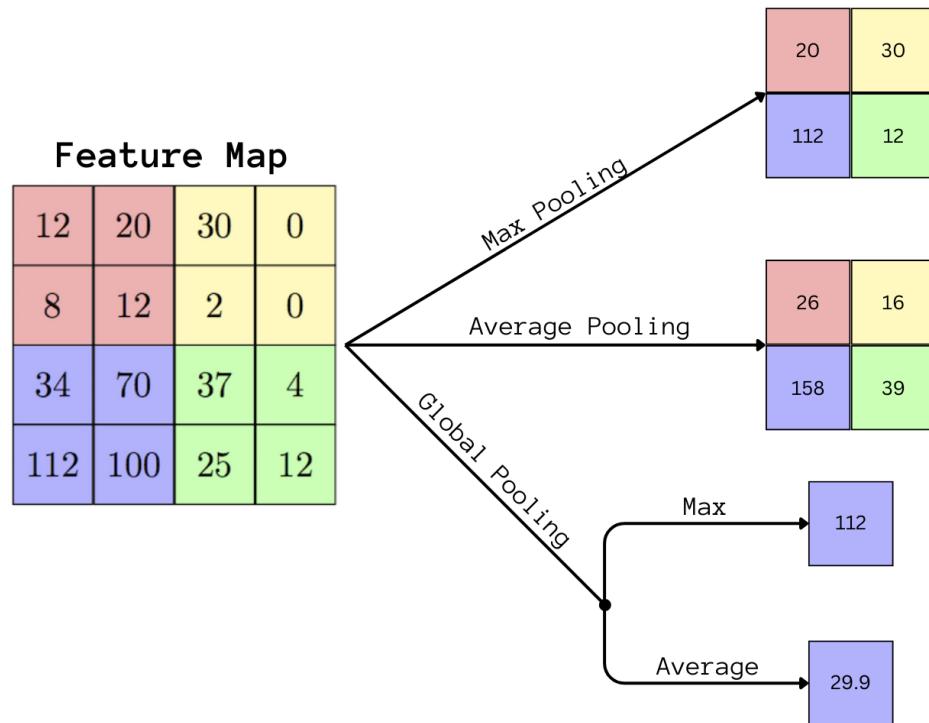


Figura 4.7 – Example of different pooling layers acting on a 5x5 feature map.

Figure 4.7 demonstrates an example of a max pooling, an average pooling, a Global max pooling and global average pooling layer acting on a 5x5 feature map, resulting in a

condensed matrix. As described above, max pooling takes the maximum value within a region, while average pooling takes the average value.

For a better understanding of these layers, Figure 4.8 illustrates how an image changes inside a simple convolutional block. An image of Jupiter from the James Webb telescope is used to exemplify the process of applying a convolutional filter, then an activation function is applied to detect the extracted features, followed by a pooling layer that condenses the information. This figure demonstrates how three different kernels acting on the input image can lead to different features being extracted in a convolutional neural network. This is a simple example of how a CNN works, as there are several different types of CNNs with different features and purposes.

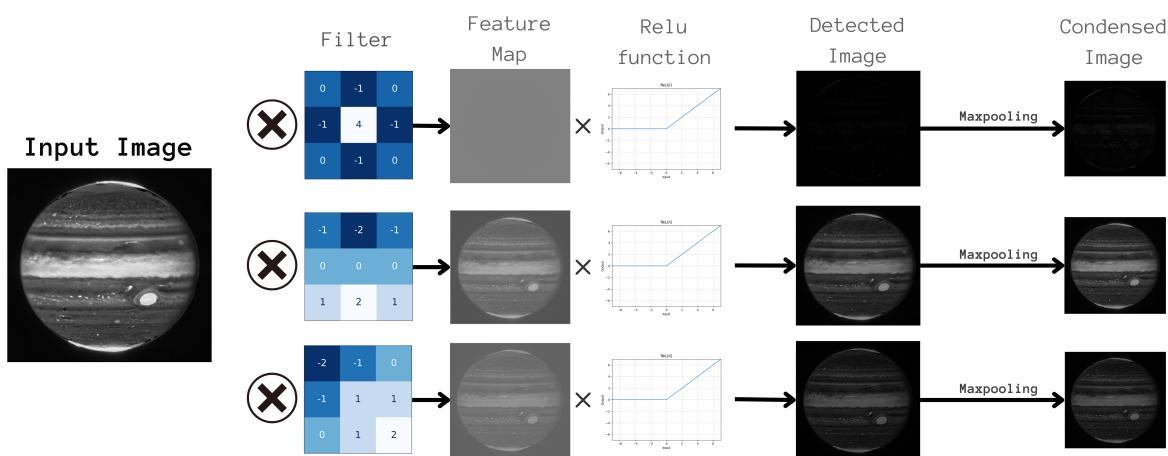


Figura 4.8 – This figure shows a simple example of how a CNN works. It uses an image of Jupiter to demonstrate the process of applying a convolutional filter, detecting features with an activation function, and condensing information with a pooling layer. Credits: —NASA cpy later

4.6 Metrics for Evaluation model's performance

In deep learning, after a training process, it is essential to validate the performance of a trained model using a variety of metrics to ensure its application in real-world science. Typically, the dataset is divided into three subsets: training, validation, and test. The training set is actively used on the training process, as described in section 4.2 . The validation set is used to tune the hyperparameters of the model and prevent overfitting(section ??). Overfitting occurs when a model performs well on the training data but poorly on the validation data. It happens when a model is too complex and “memorizes” the training set. That’s why a test set composed of unseen data is used to evaluate the model’s performance. The test dataset is used to determine the model’s accuracy, recall, and other metrics, ensuring that the model is ready for scientific usage.

This thesis focused on a binary classification problem, where the main objective is to use images as input and the model outputs a probability of this image belonging to a specific class. In this section, we will describe the most common metrics used in this type of problem.

4.6.1 Loss Function

One way to evaluate the predictive capacity of a model as a classifier during training is by using a Loss function. A commonly used error function for binary classifiers is the Binary Cross-Entropy (BCE) loss function, as described in Chapter ???. This loss function measures how well the model predicts the correct label.

The BCE loss function compares the output probability for a given image with its true label (either 1 or 0), and penalizes the model for predictions that deviate from the true value. The formula for the BCE loss function is given by Equation ???. Which is also used as a metric to ensure that the model does not overfit by monitoring the loss function value on a validation set. If the model's error function on the validation set is significantly higher than its error on the training set, it may indicate that the model is overfitting the training data. While the loss function is a useful metric for evaluating the model's performance, it is insufficient if used alone, because it only provides information about the overall performance of the model and not about the prediction of individual classes. Therefore, it is essential to use additional metrics such as accuracy, precision, recall, and F1-score to evaluate the model's performance on each class separately.

4.6.2 Confusion Matrix (CM)

The confusion matrix (CM) provides a visual information about the model's predictions and the true labels for each class, facilitating for a more detailed analysis of the model's performance. The confusion matrix is a matrix that displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class. The rows represent the predicted labels, and the columns represent the true labels (Figure 4.9), and the elements of the matrix are the number of instances that fall into each category.

- True positives (TP): The actual value was positive, and the Deep learning model predicted a positive value
- True Negative (TN): The actual value was negative, and the Deep learning model predicted a negative value
- False Positive (FP): The actual value was negative, but the deep learning model predicted a positive value

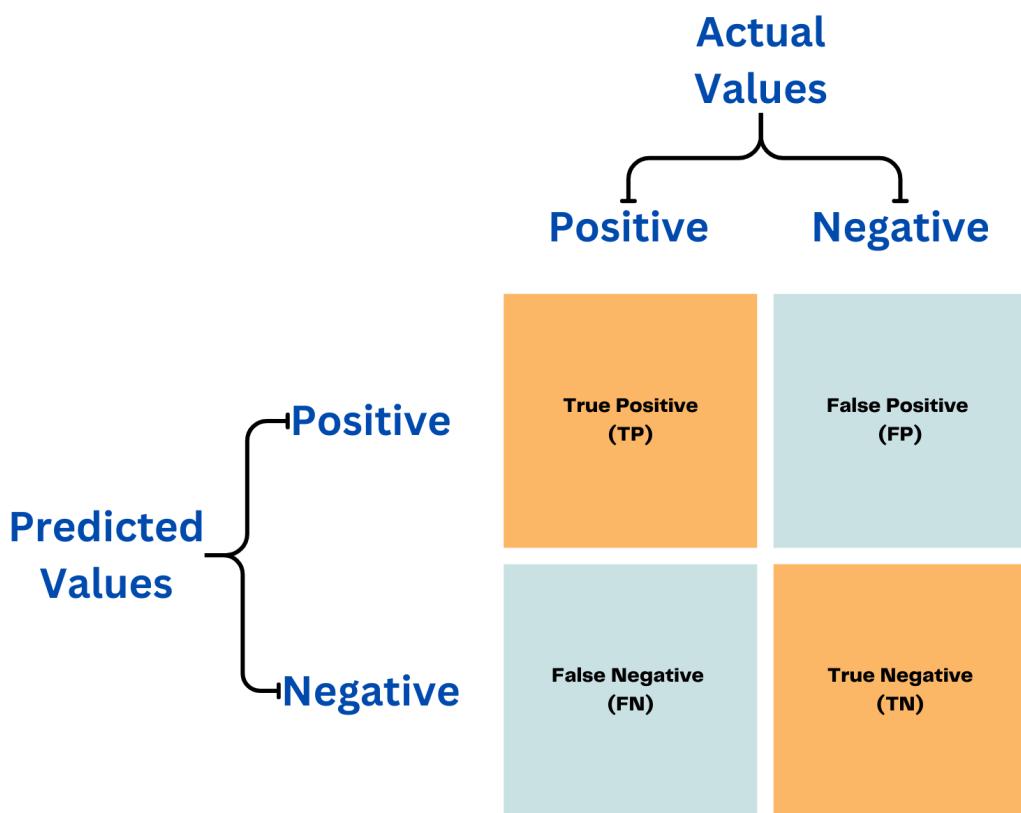


Figura 4.9 – Confusion Matrix, the rows represents the predicted labels and the columns represent the actual label

- False Negative (FN): The actual value was positive, but the deep learning model predicted a negative value

Using the values in the confusion matrix, we can calculate several useful metrics, including accuracy, precision, recall, and F1-score. These metrics provide insightful information about the model's ability to identify transients and artifacts.

- **Accuracy:** The percentage of the total predictions that were correct. It is calculated as the number of true positives plus true negatives divided by the total number of predictions, t defines how often the model predicts the correct output. For example, if the model correctly predicts 95 instances out of 100, then the accuracy is 95%.
- **Precision:** : The proportion of the total positive predictions were correct. It provides a metric to calculate the model's ability to classify positive samples. It is calculated as the number of true positives divided by the sum of true positives and false positives. For example, if the model predicts that 100 instances are positive, out of which 90 are actually positive (true positive), then the precision is 90%.

- **Recall (sensitivity or True positive rate):** The proportion of the total actual positives values were correctly classified. It measures the model's ability to detect positive instances. Recall is calculated as the number of true positives divided by the sum of true positives and false negatives. For example, if there are 100 positive instances in the dataset and the model correctly identifies 90 of them, then the recall is 90%.
- **False Positive Rate (FPR):** The fraction of false positive samples to all actual negative samples. It tells us how often the model incorrectly predicts a negative instance as positive. For example, if there are 100 negative instances in the dataset, and the model predicts that 10 of them are positive, then the FPR is 10%.
- **F1-score:** is the harmonic mean of precision and recall, and captures the contribution of both of them in a single score. This means that if a model has high precision but low recall, or vice versa, the F1 score will be low. The higher the F1-score, the better the model. The F1-score equation is:

4.6.3 Receiver Operating Characteristics (ROC) curve

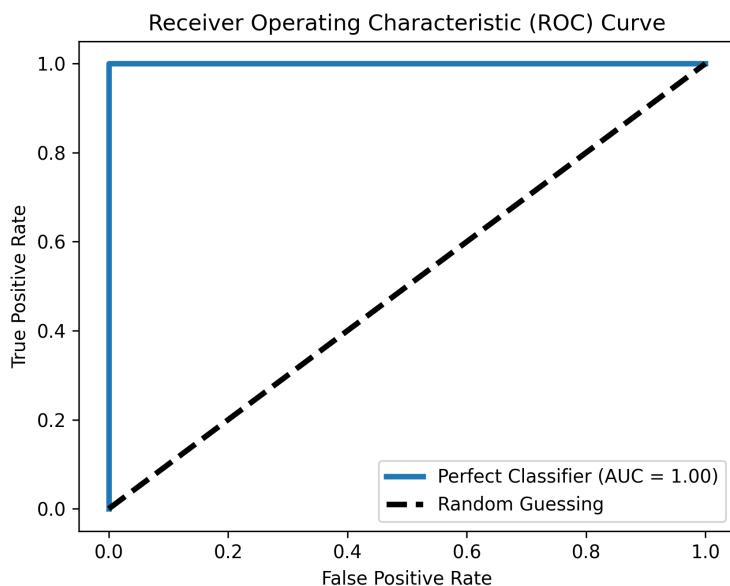


Figura 4.10 – Confusion Matrix, the rows represents the predicte labels and the columns represent the actual label

A Receiver Operating Characteristics (ROC) curve is a plot of the true positive rate with respect to the false positive rate at different classification thresholds for the output probability. It shows the trade-off between the TPR and FPR of the model as the threshold is varied. If our model did not learn how to separate artifacts and transients, the line would be a constant, and changing the threshold would have no impact on the curve,

as the model would essentially be guessing every image. On the other hand, if the model is a perfect classifier, the TPR would be 1 and the FPR zero for all different thresholds, meaning that the model would find every positive case.

Another important metric that measures the overall performance of a classifier is the area under the ROC curve (AUC) or AUROC value. As the name suggests, it is simply the area measured under the ROC curve. A higher value of AUC represents a better classifier, i.e, If the AUC is 1, the model has perfect discrimination capacity, while an AUC of 0.5 indicates that the model is guessing the images and is not a good classifier. The AUC can be interpreted as follows: for example, an AUC of 0.95 means that the model classifies two randomly chosen test images correctly with a probability of 95%.

4.6.4 Precision-recall (PR) curve

Similar to the ROC curve, the precision-recall (PR) curve is another graphical representation of the performance of a binary classification model. It is a plot of the "precision" with respect to the "recall" at different classification thresholds. It shows the trade-off between precision and recall of the model as the threshold is varied. Ideally, a perfect classifier would have a PR curve that passes through the top right corner of the plot, which represents a precision of 1 and a recall of 1. The closer the PR curve is to the top right corner, the better the model performs (Figure 4.11).

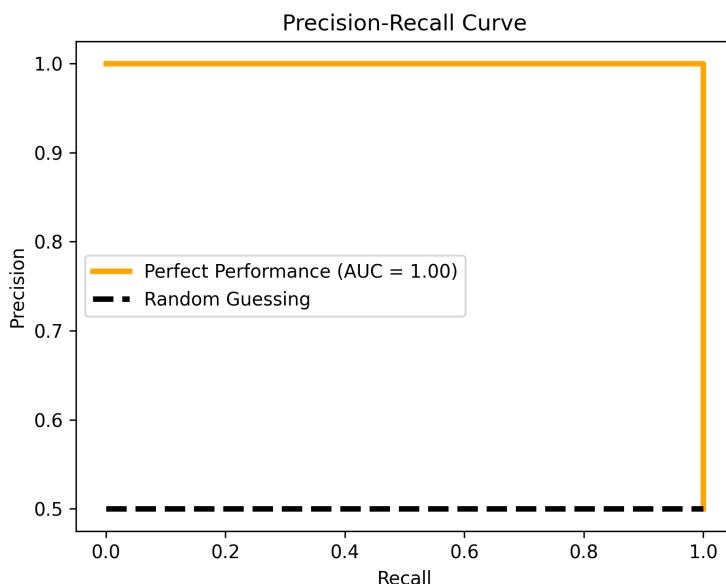


Figura 4.11 – Confusion Matrix, the rows represents the predicted labels and the columns represent the actual label

The area under the precision-recall curve (AUCPRC) is also a metric used to evaluate model performance. It summarizes the plot in a single number, where an AUPRC closer to 1 indicates better performance. Using only a confusion matrix to evaluate the performance

of a model is not the best approach. This is because when a CM is built, a threshold t is defined, which is usually set to 0.5. However, better results can be obtained with other thresholds. That's why the ROC and PR curves are important for analysis. They compare classifiers for several or all possible thresholds, and then we can choose the best threshold that fits our objective.

 CAPÍTULO 5 

AUTOMATED TRANSIENT DETECTION

lalala

a a a

a a a a a a a

CAPÍTULO 6

RESULTADOS

Neste capítulo apresentamos o processo investigativo executado durante a pesquisa, acerca do problema de realizar classificação de um conjunto de imagens de lentes gravitacionais e, em particular, o regime de poucos dados. Por isso, está organizado de modo que cada seção seja um novo passo dado na investigação do problema, de acordo as lacunas encontradas no passo anterior ou, de modo a dar mais substância ao resultado obtido anteriormente.

6.1 Problematização

Conforme exposto no capítulo 4, já houveram duas competições recentes de inteligência artificial para a classificação de lentes gravitacionais, os *GLFC*. A proposta era de criar o melhor classificador de uma base de dados de imagens simuladas de telescópios *ground-based* (no primeiro GLFC) e uma banda *space-based*. Os dados incluíram a adição de ruído e outros efeitos observacionais pela própria natureza dos equipamentos de observação, com base numa tabela verdade¹. Por isso, consistem de bom material para início do estudo. O fato de esses dados já haverem sido classificados por redes neurais, já estarem prontos para uso e tabelados contornam o problema de classificar imagens reais em que, seria necessário realizar cortes nas imagens que isolassem bem os objetos e, organizá-los de acordo a sua natureza. Um classificador que seja eficiente em detectar imagens reais de lenteamento gravitacional deve ter facilidade para detectar o mesmo fenômeno em imagens simuladas. Esta é a base de dados sobre a qual foram realizados os primeiros testes neste trabalho, até o momento em que obtivemos os resultados desejados.

A problematização inicial, portanto, consiste de utilizar uma arquitetura de rede neural para realizar classificação sobre essa base de dados. A escolhida foi a vencedora da edição

¹ A base de dados e a tabela verdade das duas edições pode ser obtida em http://metcalf1.difa.unibo.it/blf-portal/gg_challenge.html

do *ILSVRC* de 2015, a arquitetura ResNet50 [5], já abordada na subseção ???. Além de toda a validação que o seu uso em larga escala para diversos projetos supracitados poderia fornecer, a sua implementação é bastante simples na linguagem de programação escolhida para os nossos estudos - o *python*. A rede pode ser importada com apenas uma linha de código de uma biblioteca de acesso aberto - o *tensorflow-keras*.

6.2 Desempenho da ResNet50 nos dados da primeira edição do GLFC

Para utilizar a base de dados *ground-based* da primeira edição do *GLFC*, que consiste de 20000 objetos, fez-se necessário um pré-processamento dos dados. Na entrada do CBPF da primeira edição GLFC, verificou-se que utilizar diferentes estratégias para realizar a inserção dos dados na rede retornava diferentes pontuações de AUC [6]. Tanto no caso *ground-based* quanto no caso *space-based*, a melhor alternativa de pré-processamento entre as possibilidades testadas consistiu de executar um ajuste de contraste e um filtro de Wiener[7] em cada uma das três bandas utilizadas (U, G e R).

Conforme o sugerido para um problema de classificação (ver capítulo 3, faz-se necessário estabelecer uma divisão dessa base de dados em conjuntos de treino, teste e validação. A divisão utilizada na primeira execução foi de reservar 10% dos dados para validação, 10% para teste e os 80% restantes para treino. Recuperando as métricas de avaliação dos resultados já abordadas na seção ??, nas primeiras execuções da rede buscou-se analisar o custo e a acurácia do treino ao longo das épocas de execução e, seu desempenho em um teste cego e um teste aberto.

Escolhemos realizar o treino com 50 épocas, com uma taxa de aprendizado de 0,01 e de modo a otimizar uma função custo de entropia binária cruzada. Dado que acurácia e o custo da rede são formadas por valores que são calculados e registrados à cada época, ao final de toda a fase de treino é possível visualizar um registro das flutuações destes índices. Para a acurácia, este registro se encontra na figura 6.1.

Enquanto a linha azul se refere a acurácia da classificação sobre o próprio conjunto de treino da rede, a linha laranja se refere à acurácia sobre o conjunto de validação. É possível verificar que, no caso da acurácia, no início das épocas enquanto a acurácia de treino começa a partir de um certo valor, a acurácia de validação começa do zero. Ao longo das épocas ocorre um rápido crescimento da acurácia de validação, o que indica que há um rápido aprendizado da rede já no início. O esperado é que ao longo das épocas haja uma convergência cada vez maior das duas linhas, até que idealmente elas se encontrem. Neste caso, isto não acontece evidentemente, mas observando os intervalos do gráfico é possível ver que os valores dos dois oscila entre 0.8 e 0.9. Isto é, estão relativamente próximos. Além disso, cabe lembrar que esta é uma primeira tentativa de classificação dos dados da rede, que ainda será bastante refinada.

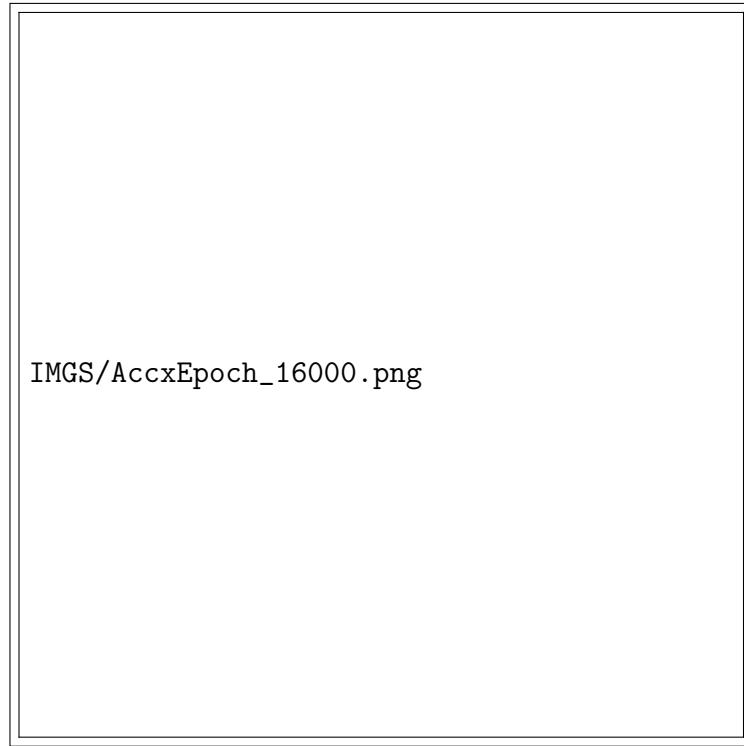


Figura 6.1 – Gráfico de Acurácia da rede ResNet50 em toda a base de dados.

Já no caso da função custo, conforme já abordado no capítulo 3, o esperado é que seja uma tendência de minimização. Neste contexto, o gráfico gerado está presente na figura 6.2.

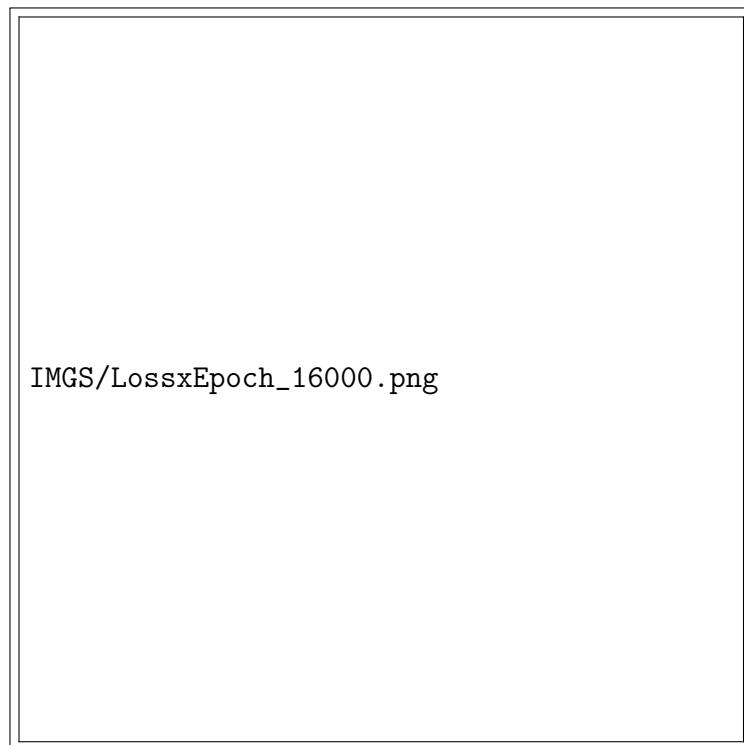


Figura 6.2 – Gráfico da Função Custo da rede ResNet50 em toda a base de dados.

Verifica-se na figura 6.2 a reprodução de um comportamento bastante comum para

o gráfico da função custo em problemas de classificação, que se trata do início quase assintótico na primeira época para, em seguida, descer até atingir uma convergência para zero. Da mesma maneira que no gráfico da acurácia, a linha azul é referente ao custo calculado sobre os dados de treino, enquanto a amarela é sobre os dados de validação. No gráfico reproduzido, o custo de validação não mostra uma convergência tão clara quanto o custo de treino. Se verifica uma forte tendência de overfitting a partir da época ~ 20 , sugerindo que devemos usar a rede treinada em torno dessa época apenas.

Por fim, a última métrica adotada a fim de avaliar a classificação é a curva ROC. Consiste nada mais do que a avaliação do modelo em um teste cego, em que é registrado a cada limiar entre as possibilidades de classificação a taxa de falsos positivos com a de positivos verdadeiros (ver subseção ??). Diferentemente das métricas anteriores, a ROC é calculada após o treino e assim, utilizando o modelo final da rede, temos para a curva o gráfico da figura 6.3.



Figura 6.3 – Gráfico da curva ROC da rede ResNet50 em toda a base de dados.

Evidentemente, há uma reprodução do comportamento característico para curvas ROC já abordado: a subida bastante abrupta da taxa de falsos positivos inicialmente, seguida da convergência para o máximo de falsos positivos posterior conforme o limiar se torna alto demais. A sigla "AUC" (*Area Under Curve*) na legenda pode ser traduzida literalmente como "área sob a curva", que indica em unidades ao quadrado a área do gráfico que se mantém abaixo da curva ROC. A linha tracejada indica a linha que teria que ser descrita pela curva para que a AUC fosse de 0.5. Na classificação que resultou na figura 6.3, o resultado para a AUC foi de 0.965, valor bastante próximo de 1.

Esse resultado da ROC nos indica que a ResNet50 já se mostrou ser bastante eficiente na classificação da base de dados da primeira edição do GLFC. Verifica-se que o valor de AUC se aproxima do que foi obtido pelas redes que melhor performaram na primeira edição da competição, de modo que ficaria em quarto lugar na tabela de classificação (ver [6]). O seu sucesso já era conhecido, mas reproduzir esse dado confere a certeza da replicabilidade deste. Enquanto as acurácia e custo se mantiveram em convergência e sob controle respectivamente, a AUC se aproximou bastante do ideal valor de 1.0. Isto, à princípio, resolve o problema de encontrar uma rede neural que execute a classificação nessa base de dados.

6.2.1 A investigação de um limite de poucos dados

Os resultados da seção anterior mostram a performance em todo o conjunto de dados. No entanto, conforme já abordado no capítulo 4, um problema comum na implementação de uma busca automatizada de arcos gravitacionais é a raridade do evento. Seria muito conveniente ter uma base de dados reais tão grande quanto o tamanho dessa base de dados simuladas, no entanto, não existe um número tão grande de registros desse fenômeno até o presente momento. Por isso, se torna relevante o desenvolvimento de um buscador de arcos gravitacionais, que pudesse ser treinado com um conjunto restrito de dados reais. Dessa forma o método não dependeria de um modelo de lente pré-estabelecido ou de alguma hipótese adicional sobre os dados. Quanto menor for este número, a rede neural receberá menos informações acerca da distinção entre imagens de galáxias comuns e imagens onde está ocorrendo o fenômeno de lenteamento. Assim, uma vez confrontada com imagens do fenômeno não vistas previamente, a rede neural possuirá uma dificuldade maior em discernir o fenômeno, uma vez que a sua capacidade de generalização poderá ficar prejudicada devido a baixa variedade.

No intuito de prosseguir com essa motivação, começamos a reduzir a quantidade de dados de entrada na rede, fixamos o tamanho do conjunto de validação e, acrescentamos o restante ao conjunto de teste. Uma vez treinando com menos dados, espera-se que a rede perca um pouco da sua capacidade de predição. A adição dos dados restantes ao conjunto de teste não impacta o desempenho da rede neural, poderíamos manter a mesma quantidade de dados de teste usadas na seção anterior. Possuir um conjunto de teste maior apenas adiciona mais pontos às curvas de predição no teste cego e, portanto, nos permite ter um pouco mais de precisão no desempenho da rede com novos dados. Seguindo essa estratégia, escolhemos alguns intervalos de valores para ir diminuindo a eficiência do treino até que a classificação se tornasse a pior possível - próximo do regime aleatório. A avaliação deste desempenho da classificação será feita os métodos de avaliação de resultados já discutidos previamente, mas em especial observando a AUC.

Uma preocupação importante a partir desse ponto é a randomização dos dados de entrada. Conforme formos diminuindo o tamanho do conjunto de treino, a influência

individual de cada imagem da base de dados, na eficiência da rede como um todo, aumenta. Portanto, podem haver grandes variações na capacidade de predição da rede se houver qualquer viés na escolha desses dados. Um pequeno conjunto de dados onde o fenômeno a ser classificado esteja claramente discernível pode performar melhor do que uma base de dados maior, com dados mais confusos. Uma maneira de se contornar esse problema, à princípio, é sempre randomizar a base de dados na entrada, de modo a garantir que não haja privilégio na escolha das imagens. Essa randomização deve vir, inclusive, antes da separação da base de dados em conjuntos de treino, validação e teste, para garantir que também não ocorra o mesmo problema nestas outras duas etapas.

Portanto, o procedimento para a classificação diminuindo o tamanho do conjunto de treino foi o seguinte: à cada execução,

- embaralhe as imagens e a tabela verdade correspondente;
- separe uma certa quantidade de dados para validação;
- use o restante dos dados como conjunto de teste;
- construa um novo modelo e treine-o utilizando o conjunto de treino e validação;
- realize as previsões com o conjunto de teste (ainda não visto pelo modelo).

Seguindo este procedimento de validação cruzada, começamos a realizar o trabalho de treino e obtenção de resultados no intervalo de dados de nosso interesse. Por motivos de simplificação do trabalho realizado, nos focaremos neste primeiro momento em observar o valor da AUC do modelo final da rede. O resultado deste estudo está presente no gráfico da figura 6.4, em que o eixo horizontal corresponde ao conjunto do tamanho de treino, o vertical à AUC. A linha tracejada que corta o gráfico corresponde à AUC de 0.5 em cada altura, a fim de observar a proximidade do valor de AUC do correspondente valor de treino à esse valor. Conforme já mencionado na subseção ??, o valor de 0.5 para AUC corresponde a um regime de completa aleatoriedade na classificação, portanto quanto mais próximo desse valor, pior é o poder de predição da rede no teste cego.

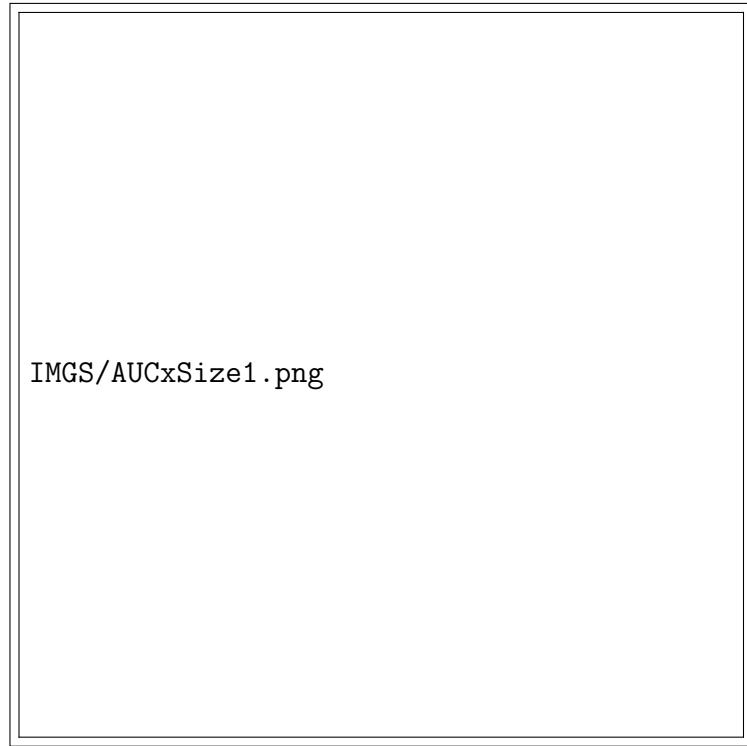


Figura 6.4 – Distribuição de AUCs de acordo o tamanho do conjunto de treinamento.

Como mostra a figura 6.4, na análise que foi realizada o valor máximo de AUC obtido no teste cego foi atingido na marca de 16000 objetos de treinamento, com AUC de aproximadamente 0.963. No contexto desta execução, foram utilizados 2000 objetos para validação e 2000 para teste, correspondentes à 10% cada da base de dados total, o que contribui para justificar a escolha dessa proporção de dados no contexto da seção anterior. Realizamos testes com 17000 e 18000 objetos de treino conforme mostra a referida figura, no entanto, o fato de isto exigir uma diminuição dos conjuntos de validação e de teste os torna não completamente comparáveis.

Ao começarmos a avançar para o lado esquerdo do gráfico, percebemos que o desempenho da classificação se mantém bastante alto conforme diminuímos o tamanho do conjunto de treino, até uma porcentagem bem pequena do tamanho total da base de dados. Somente por volta dos 500 à 600 objetos em que a AUC começa a cair para abaixo de 0.8. Cabe apontar também que a continuidade do gráfico se deve apenas à ligação de cada par de pontos vizinhos, e não à ocorrência de treinos para toda a quantidade possível de objetos dentro desse intervalo.

De acordo com o critério sugerido por Hosmer (2013) [8], um regime “ótimo” para classificação seja o limiar de 0,8 para AUC, temos portanto que a faixa de 500 à 600 objetos de treino é uma faixa de quantidade de dados onde ainda é possível obter uma boa classificação em testes cegos. Deste modo, no problema de classificação de lentes e não lentes, seriam necessários portanto 600 objetos entre lentes e não lentes para ser possível afirmar que o modelo é eficiente em classificar os dados considerados. Esta

quantidade, apesar de ser pequena em comparação ao tamanho da base de dados total utilizada neste problema, ainda seria um número bem grande de registros de fenômenos de lenteamento gravitacional forte para uma mostra homogênea de sistemas confirmados espectroscópicamente.

6.2.2 Multiplos sets de treino/teste

Embora não abordado na seção 6.2.1, conforme diminui-se o tamanho do conjunto de imagens de treinamento, pode-se verificar que sucessivos treinos com a mesma quantidade de dados pode gerar resultados com desvios significativos. Isto ocorre por conta do passo de randomização dos dados antes de sua separação, e também pelo componente estocástico do treinamento das redes, por exemplo, o uso de camadas de dropout e a inicialização dos pesos. Os conjuntos de treino que possuírem uma melhor variedade e quantidade de objetos que poderão ser mais eficazes em realizar as predições e, portanto, resultarão em valores de AUC maiores. Esse conjunto ideal não deve ser composto apenas por objetos facilmente identificados como lentes gravitacionais mas também objetos difíceis, de modo que haja a possibilidade de aprender pelo equilíbrio. Qualquer um dos casos diferentes pode acabar dificultando o treino - o que também justifica a randomização.

Pode-se concluir portanto, que quando diminui-se o tamanho do conjunto de treino, existe uma tendência a diminuir a variedade, o que pode prejudicar o desempenho da rede. Dado um conjunto fixo de simulações e reduzindo suficientemente, surge de modo natural a possibilidade de criar mais de um conjunto de treino, com imagens totalmente distintas, sem grandes prejuízos no conjunto de validação e de teste. Seria possível, por exemplo, dividir as 20000 imagens do GLFC de modo a usar 2000 para validação, 2000 para teste e, os 16000 restantes poderiam ser divididos em 10 conjuntos de treino de 1600 dados, permitindo assim, a realização de 10 treinos utilizando os mesmos conjuntos de treino e validação.

Um dos primeiros benefícios dessa estratégia é a análise do quanto divergentes podem ser os resultados de acordo com os dados que entram na rede, isto é, verificar o quanto sensível está à classificação à escolha dos dados de entrada. Isto leva à natural consequência da possibilidade de realização de uma estatística sobre os resultados da rede, como o cálculo da média e/ou desvio padrão. Particularmente, podemos realizar essa estatística sobre a AUC do teste cego, uma métrica de bastante peso na avaliação final do modelo. Desta forma, torna-se possível calcular uma AUC média com barras de erro, calculadas apartir do desvio padrão. Outra utilidade para a medida inclui a possibilidade de verificar a ocorrência de *overfitting*, um problema comum na aprendizagem de máquina em que a rede neural consegue reconhecer bem os dados do conjunto de treino mas, tem dificuldades para generalizar os resultados para dados nunca antes vistos.

Como já mencionado, podemos adotar esta técnica no momento em que há uma quantidade suficientemente pequena de dados de treino entrando na rede neural, em um con-

junto grande de simulações. Enquanto isto não ocorre, no entanto, ainda é possível definir uma estratégia para encontrar uma estatística do nosso modelo, inspirada num recurso comumente empregado na aprendizagem profunda de máquina conhecido por *validação cruzada*. Uma das versões mais conhecidas da técnica de validação cruzada é o que se conhece por validação cruzada de *k-folds*, onde k é o número de subdivisões da base de dados. Nesta técnica, divide-se toda a base de dados que seria utilizada para treino e validação em k *folds*, ou k subconjuntos de mesmo volume de dados. Cada permutação destes conjuntos comporá a base de dados usada no treino do modelo, enquanto o restante é usado para validação.

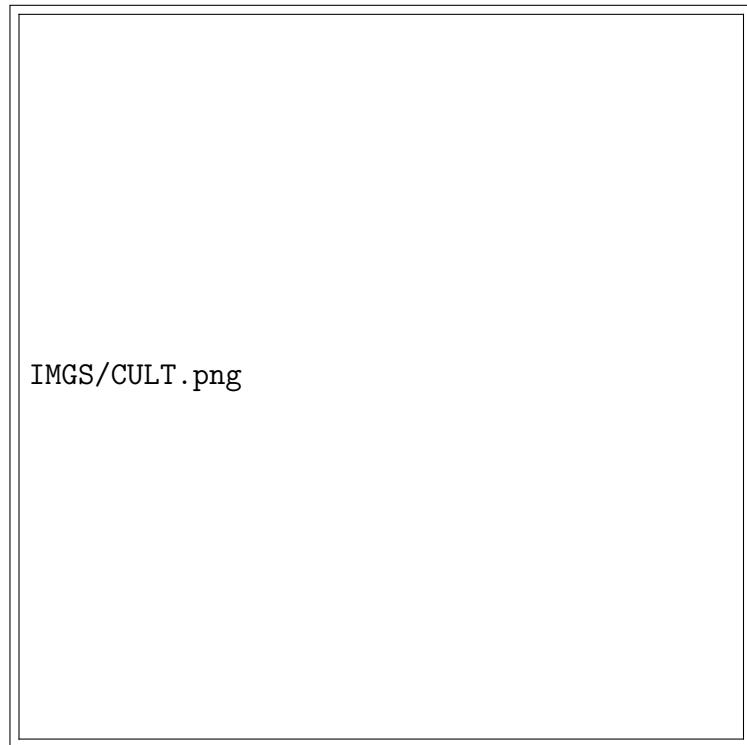


Figura 6.5 – Esquema representativo do uso de diferentes conjuntos de treino e de validação durante o treinamento de uma rede neural utilizando validação cruzada de k folds com 4 folds.

Conforme mostra a figura 6.5, na primeira iteração um dos folds é reservado para validação, enquanto os demais folds são agrupados para compôr o conjunto de treino. Já na segunda, outro fold é designado como conjunto de validação, enquanto os demais são reagrupados para compôr o conjunto de treino. Este processo é repetido k vezes até que todos os folds já tenham sido utilizados para validação em alguma das iterações. Consequentemente, o treinamento da rede foi executado k vezes, com diferentes conjuntos de treino de acordo as permutações possíveis no agrupamento de folds. À cada início de nova iteração é preciso reiniciar os pesos da rede, no intuito de obter diferentes resultados com cada agrupamento de folds.

É importante mencionar que, no presente trabalho, utilizamos este método apenas para obter diferentes conjuntos de treino. O conjunto de teste sempre foi separado antes da etapa de treinamento da rede e portanto, da subdivisão dos dados em folds. O motivo

reside no interesse de ter um conjunto de teste de tamanho sempre fixo, e não dependente do tamanho dos folds. Neste trabalho, adotamos 10 como número fixo de folds para todas as ocasiões e, adotamos a estratégia da subdivisão dos dados de treino em folds nos contextos em que haviam mais de 1500 objetos no conjunto de treinamento da rede. Para este limiar e abaixo, é possível obter 10 conjuntos de treino totalmente disjuntos dentro do dataset de 20.000 dados (por isso utilizamos a subdivisão dos dados em folds com, por exemplo, 10.000 dados de treino). Por isso, julgamos adequado demarcar graficamente de alguma maneira a mudança de regime de múltiplos treinos, indicado na figura 6.6 por diferentes cores. Os pontos e barras demarcados em azul correspondem ao método de múltiplos treinos, enquanto os marcados por vermelho ao método inspirado na validação cruzada.



Figura 6.6 – Distribuição de AUCs de acordo o tamanho do conjunto de treinamento e respectivas barras de erro.

Exceto pelas barras de erro, podemos ver que o gráfico da figura 6.6 é semelhante ao da figura 6.4, o que é de esperar visto que não adicionamos melhorias à classificação. No entanto, os pontos do gráfico 6.6 agora correspondem à mediana das AUCs obtidas pela predição da rede em cada uma das execuções, sobre o mesmo conjunto de teste. Apesar de muito pequenas, existem barras de erro também nos pontos em vermelho, porém, as margens estão tão próximas do valor original que podem à princípio parecerem ausentes. Poderia ser argumentado que isto se deve por conta de, nesse diferente regime de escolha dos conjuntos para os treinos consecutivos, os conjuntos de treino serem bastante parecidos e por isso, não serem muito divergentes um dos outros durante cada execução. Isto minimizaria bastante a margem de erro porque, usando conjuntos de treino bastante

parecidos, não seriam esperadas grandes divergências no desempenho. No entanto, pode-se observar da figura que, nos pontos onde se mantém o regime de conjuntos de treino disjuntos próximos aos pontos onde ocorre a mudança de regime, as margens de erro já são bastante pequenas e já se confundem com os próprios pontos. Seria esperado, portanto, que mesmo que os conjuntos de treino com tamanhos superiores ao limiar de transição de regime fossem disjuntos, ainda teríamos pequenas margens de erro.

Por outro lado, conforme se prossegue para o lado esquerdo do gráfico é possível notar que as barras de erro vão aumentando quase progressivamente, até atingirem escalas que abrangem toda a margem de dados ou um dos lados. Portanto, na hora de observar esse limiar de otimização, podemos ver que na faixa dos 500 a 600 objetos as barras de erro ainda estão bem pequenas, apesar de haver um ponto dissidente no ponto dos 450 objetos. Podemos assumir assim, que a escolha prévia de um regime razoável de classificação de 500 a 600 objetos se mantém, visto que nessas faixas as barras de erro ainda são pequenas.

Verifica-se portanto, o benefício do uso dos múltiplos treinos na classificação dos dados. A partir das divergências na classificação dados os diferentes conjuntos de treino, somos capazes de definir margens de erro para as execuções. Fica bastante clara a distribuição de regiões onde os resultados são bastante confiáveis dos locais onde estes não o são. Lançando mão deste recurso, podemos introduzir outras técnicas, agora, que ajudem efetivamente no treinamento do modelo e avaliar a sua contribuição para o problema em questão.

6.2.3 Avaliação da influência da aumento de dados

Depois de se verificar a existência de um limite em que podemos, por convenção, classificar o desempenho da detecção como satisfatório, podemos pensar em alternativas que forcem ainda mais esse limiar. Até aqui já fizemos a estatística com múltiplos treinos, mas cabe a adoção de recursos que possamos inserir dentro da própria rede neural a fim de melhorar o desempenho da classificação. Um destes recursos é o que se chama por *aumentação de dados*.

O recurso da *aumentação de dados* consiste de aumentar a quantidade de dados de entrada na rede, criando cópias com modificações sutis dos dados já existentes. Entre essas modificações estão a rotação da imagem, transposição, ampliação, filtragem, entre outros. Essas cópias entram na rede neural junto às imagens originais como se houvessem mais dados de entrada do que o previamente estabelecido. Por continuarem sendo o mesmo dado com uma simples modificação visual, a tabela verdade para os novos dados continuam sendo a mesma. Isso é bastante útil principalmente na ausência de grandes quantidades de dados, pois alimenta a rede com novos dados que, serão tão eficazes quanto os antigos para o treino. Por consequência, é de se esperar um aumento na capacidade de generalização da rede em testes cegos, uma vez que, tendo sido alimentada com mais

variações de um fenômeno, poderá se tornar mais fácil identificar novas ocorrências do fenômeno em diferentes padrões ou posições.

Nesse processo de investigar o desempenho da rede com poucos conjuntos de dados, o recurso da aumentação de dados pode ser um aliado forte no aumento da precisão da rede em suas previsões. Fizemos o teste com o intervalo de dados sobre o qual já trabalhamos até agora. No entanto, apenas para fim de clarificar a exposição, optou-se por plotar graficamente apenas no pequeno intervalo de dados. Os dados da aumentação escolhidos foram gerados a partir de:

- Rotações aleatórias das imagens em ângulos de 90° ;
- Espelhamento horizontal;
- Espelhamento Vertical.

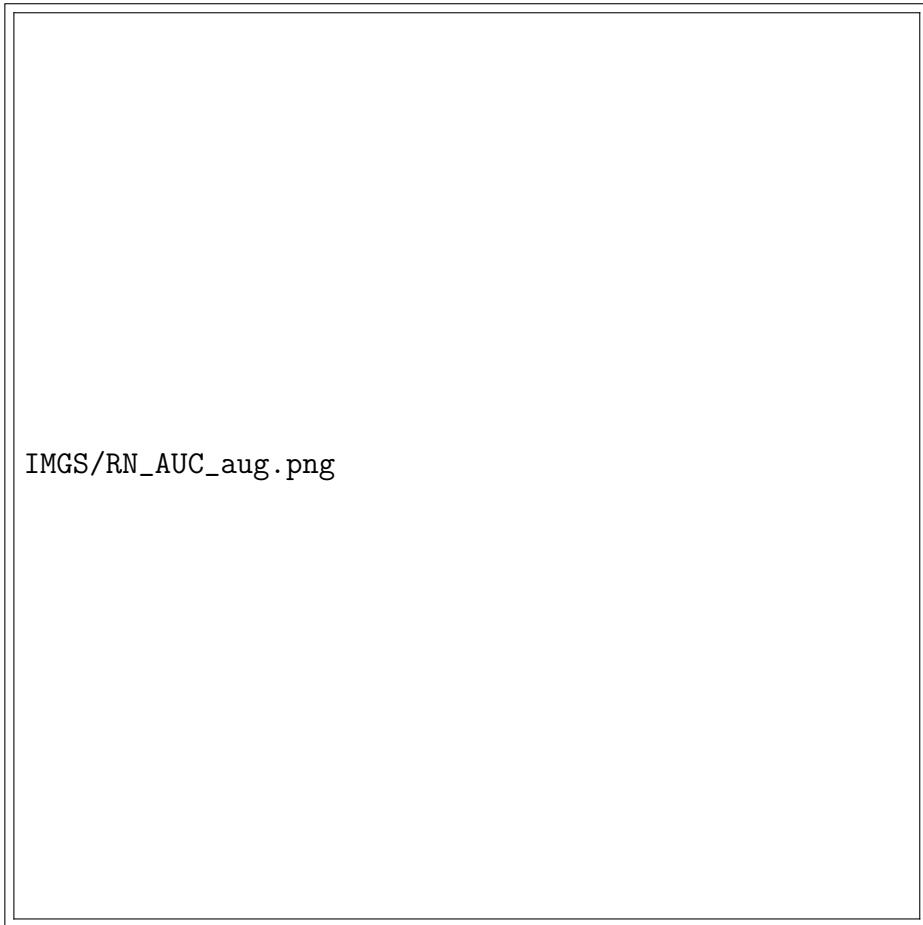


Figura 6.7 – Comparação entre as AUCs com e sem aumentação de dados.

Da figura 6.7, temos a justificativa para o por quê limitar a análise a poucos dados. É possível visualizar que mesmo nos maiores valores do gráfico, os pontos que demarcam as AUCs com aumentação e sem aumentação se fazem bastante próximos até que, para um número de cerca de 200 ou mais, já chegam a se sobrepor.

6.2.4 Avaliação da influência do pré-carregamento de pesos

O treinamento de uma rede neural, como já vimos no capítulo 3, consiste, resumidamente, na otimização de seus parâmetros. Antes de começar a fase de treino, na construção do modelo aos seus parâmetros são atribuídos valores aleatórios que, de acordo as informações retiradas dos dados de entrada, vão sendo otimizados época à época de modo a se adequarem aos dados que estão entrando na rede. É possível que essa otimização dos parâmetros leve bastante tempo ou sequer ocorra, como ocorre no problema dos mínimos locais. O pré-carregamento de pesos consiste, apenas, de fornecer antes do treino da rede, valores otimizados em outra base de dados aos parâmetros iniciais da rede. Funciona como a importação do aprendizado da rede com outra base de dados, servindo de pontapé inicial para desenvolver aprendizado em uma nova base de dados. Mesmo que o objeto da classificação seja diferente ou ausente dos objetos presentes na base de dados treinada anteriormente, pode haver um impacto positivo na detecção da nova base.

Precisamente, no caso do problema em questão, aproveitando as facilidades de importação presentes nas bibliotecas do python, é possível importar os pesos pré-carregados do treino da rede sobre uma base de dados bastante conhecida: o ImageNet. Retreinamos a rede em todos os intervalos de dados usados previamente, usando o recurso de múltiplos treinos, a fim de verificar o quanto o pré-carregamento dos pesos impacta na classificação. A comparação feita está presente na figura 6.8, em que os pontos em vermelho indicam as medianas das AUCs sobre 10 conjuntos na ocasião anterior, sem pré-carregamento de pesos e, em azul, usando o recurso de pré-carregamento dos pesos. O eixo vertical consiste das AUCs com suas respectivas margens de erro, enquanto o horizontal constitui o valor do tamanho do conjunto de treino utilizado ou, para os casos em que isto é possível, o tamanho das conjuntos(*folds*) utilizadas em cada treino.

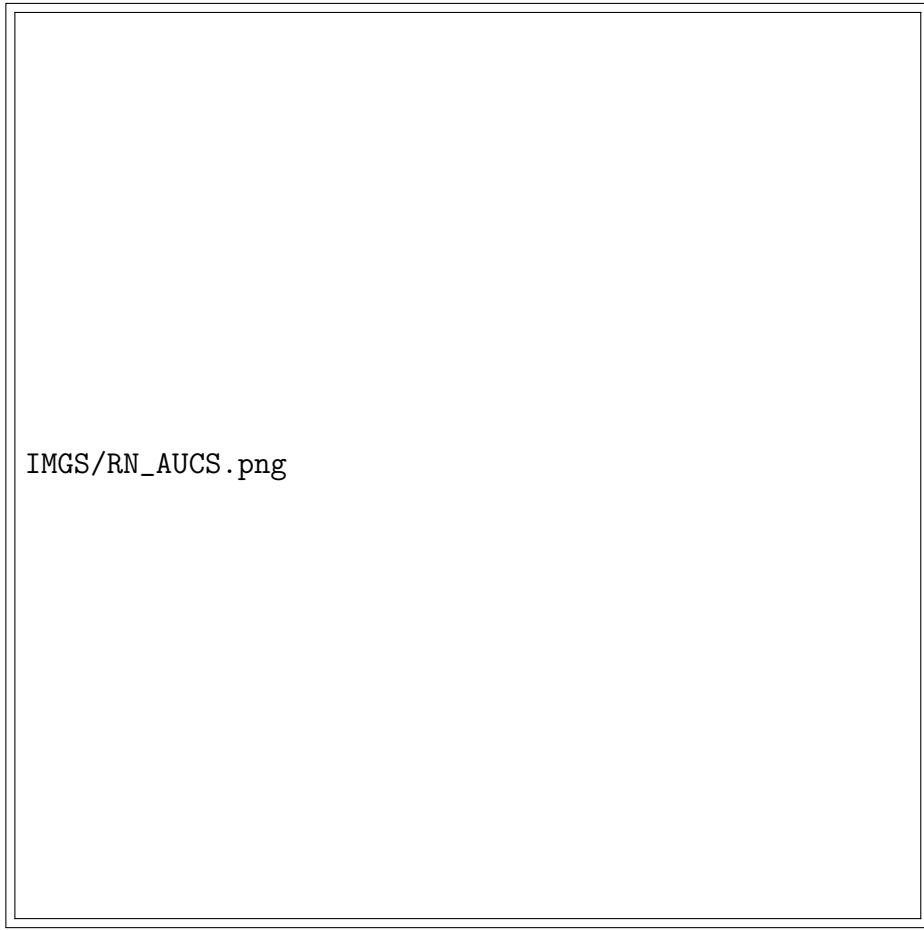


Figura 6.8 – Comparação entre as AUCs com e sem pré-carregamento de pesos.

Enquanto no canto direito do gráfico em que se atinge a totalidade dos dados de treino há uma evidente proximidade e até sobreposição dos valores de AUC, conforme diminui-se o tamanho da base de dados usada para treino surge uma evidente disparidade. Essa distância entre os valores de AUC só cresce até o mínimo de objetos de treino utilizados que, enquanto para o caso dos dados sem pré-carregamento de pesos atinge o limiar de 0.5 na faixa dos 150 dados de treino, para os treinos com o pré-carregamento esse limiar sequer é atingido na mediana, porém já é consistente dentro dos erros por volta de 60. Além disso, as barras de erro para o caso sem pré-carregamento de pesos mantém-se comportadas até por volta dos 500 dados de treino, no caso dos pesos pré-carregados estas margens se mantém próximas ao valor da mediana até a faixa de cerca de 70 dados de treino.

Estas informações mostram que, ao contrário da estratégia da aumentação de dados, dentro dos testes considerados nesta base de dados, o recurso do pré-carregamento dos pesos mostra-se indispensável para a melhora dos resultados atingidos, sobretudo nos casos em que o número de dados de treinamento é reduzido. Além disso, por conta de reduzir consideravelmente o tamanho das barras de erro, garante que as redes treinadas sejam mais estáveis. Tomando apenas por convenção, já discutida em seções anteriores, que o valor de 0.8 para a AUC é um limite de confiabilidade para a predição num teste

cego, vimos que sem o pré-carregamento dos pesos esse valor definia o uso de cerca de 450 dados para treino como mínimo necessário. Usando o pré-carregamento dos pesos podemos levar esse limite um pouco mais além, exigindo um mínimo de cerca de 200 dados para treino que tornem a classificação aceitável, considerando as barras de erro.

Nesse limite seria possível obter 80 amostras de dados reais, em uma amostra uniforme em compatíveis com surveys recentes, para serem usadas no treinamento da rede, misturadas com 80 dados de galáxias comuns. No entanto, continuamos a nossa investigação buscando verificar se ainda é possível reduzir esse limite ainda mais.

6.2.5 Uso do ensemble de redes nos dados da primeira edição do GLFC

Ao invés de substituir o método atual por outra arquitetura, podemos aproveitar o desempenho da arquitetura que temos e tentar encontrar outra que desempenhe tão bem quanto. Descobrindo uma arquitetura nova que desempenhe ainda melhor que a anterior, poderíamos efetuar apenas a substituição simples dessa nova arquitetura pela antiga mas, existe a possibilidade de ir além. Nesta seção avaliamos a possibilidade de combinar o resultado de duas arquiteturas diferentes, com intuito de superar a performance das duas, individualmente através de um *ensemble* de redes.

Neste trabalho resolvemos prosseguir o estudo com o ensemble de duas redes neurais, a ResNet, já utilizada previamente, e a EfficientNetB2. Fizemos uma breve abordagem destas na seção ???. A motivação para escolha da EfficientNetB2 se deu por conta de ser uma rede bastante atual, no estado do conhecimento da época, e pela agilidade na sua execução. Por possuir menos parâmetros do que a rede ResNet50 (quase 8 milhões, em comparação aos respectivos mais de 23 milhões), o tempo para treiná-la é ainda menor do que o tempo que leva o treino da própria ResNet50. Por isso, julgamos que seria interessante verificar como a EfficientNetB2 desempenha nessa base de dados estudada. Na figura 6.9, apresentamos os resultados de ambas as redes. Realizamos o treino da EfficientNetB2 utilizando os mesmos recursos para incremento das previsões usados na ResNet50, como a aumentação de dados e o pré-carregamento de pesos da base de dados *ImageNet*. O que temos na figura 6.9 são as medianas das AUCs devido aos múltiplos treinos da nova rede, nos mesmos intervalos de dados que já utilizamos para a ResNet50. Após o treino, fizemos o modelo executar um teste cego de classificação na base de dados utilizada à cada etapa do multitreino e, por fim, depois retiramos as medianas.

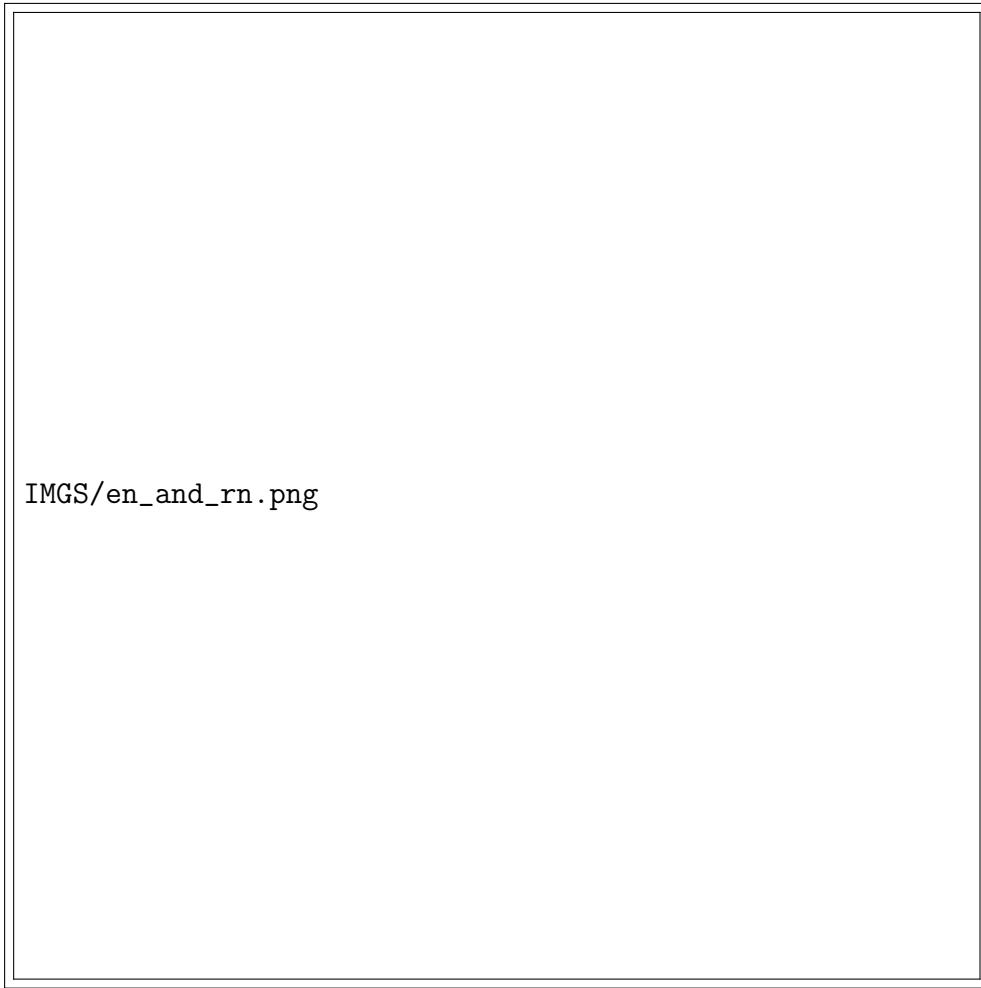


Figura 6.9 – Comparação de desempenhos das redes ResNet50 e EfficientNetB2.

Como pode-se verificar pela figura 6.9, a EfficientNetB2 (em azul) na média, performa para quase todos os pontos melhor do que a rede ResNet50, no entanto, as discrepâncias são consistentes dentro da margem de erro. Comprova-se, desta maneira, a usabilidade da rede EfficientNetB2 na estratégia de classificar essa base de dados, conjuntamente à arquitetura que utilizamos antes.

Inicialmente, a primeira estratégia para se realizar um ensemble das redes foi a mais imediata que poderia se imaginar: uma média aritmética das previsões. As previsões de um modelo de rede neural sobre um conjunto de dados constituem de pontuações para aqueles dados, que podem ter diferentes valores de acordo com a quantidade de classes para se classificar e, o tipo de função de ativação na saída da rede. No caso das redes que utilizamos, as saídas da rede foram as funções pré-definidas chamadas por *softmax*, que retornam para cada classe um número de 0 à 1, normalizado em 1 para todas as classes. Por conta de termos um problema de classificação binária, estão envolvidas apenas duas probabilidades: a chance de estar ocorrendo um lenteamento gravitacional ou não. Assim, a saída da rede neural retorna dois números com as probabilidades preditas de estar havendo lenteamento ou não, cuja soma deve ser 1. Cada arquitetura prevê este conjunto

de probabilidades para cada imagem do conjunto de teste. Portanto, calcular a média aritmética destas previsões seria calcular a média entre as probabilidades de estar havendo lenteamento e a de não estar, resultando em dois novos valores de probabilidades para aquele dado. Repetindo este mesmo processo para todos os dados de teste, o conjunto das previsões constitui uma espécie de previsão conjunta das duas arquiteturas.

Para verificar se o uso do ensemble produz um impacto positivo no desempenho das redes, além do treino que já realizamos nas imagens da base de dados utilizando a rede ResNet50, faz-se necessário repetir o treino com todos os intervalos mencionados com a rede EfficientNetB2 para, à cada caso, calcular a média dos valores das previsões. Assim como fizemos ao longo dos passos anteriores deste trabalho, também lançamos mão do recurso do múltiplos treinos, da aumentação de dados e do pré-carregamento dos pesos. A rede EfficientNetB2 também é uma rede pré-carregada pela biblioteca keras e assim, oferece uma função que carrega a rede com a opção de pré-carregar os pesos da base de dados *imagenet*, conforme fizemos. Apresentamos os resultados na figura 6.10. Na legenda, "RN" corresponde à mediana das AUCs dos múltiplos treinos prevista pela rede ResNet50 (em vermelho), "EN" à respectiva mediana prevista pela rede EfficientNetB2 (em azul) e, "(RN+EN)/2" corresponde ao ensemble entre elas (em verde).



Figura 6.10 – Comparação de desempenhos da rede ResNet50, EfficientNetB2 e do ensemble entre elas.

Pode-se observar pela figura 6.10 que a rede EfficientNetB2 teve um desempenho superior, na média, à rede ResNet50, mas consistente dentro da incerteza. O mesmo ocorre com o ensemble, a média parece superar as redes individuais. Essa diferença é pouco perceptível nos casos em que o conjunto de treino individual é o maior possível, mas vai se acentuando conforme vamos reduzindo o tamanho do conjunto de treino, principalmente em relação à rede ResNet50. A conclusão dessa análise, no entanto, seria que é possível reduzir esse limiar de dados para cerca de 80 dados, apesar das barras de erro estarem bem grandes. Esse é o mínimo de dados que seria necessário para reproduzir um resultado de 0,8 através do ensemble de redes.

Existe toda uma gama de funções internas de pacotes do python com alternativas para o desenvolvimento destes ensembles, usando métodos bastante utilizados na literatura. Fizemos uma análise comparativa do desempenho de alguns destes métodos de se realizar *ensemble* de redes, a fim de verificar qual método poderia ser mais eficiente na classificação da nossa base de dados. Observando o pacote *deepstack*, carregamos a função para ensemble nele já pré-definida *DirichletEnsemble*, e da biblioteca *scikit-learn* importamos a função *LogisticRegression* para observar o seu desempenho. Realizamos o

treino para todas as faixas de treino utilizadas, obtendo o gráfico expresso na figura 6.11.

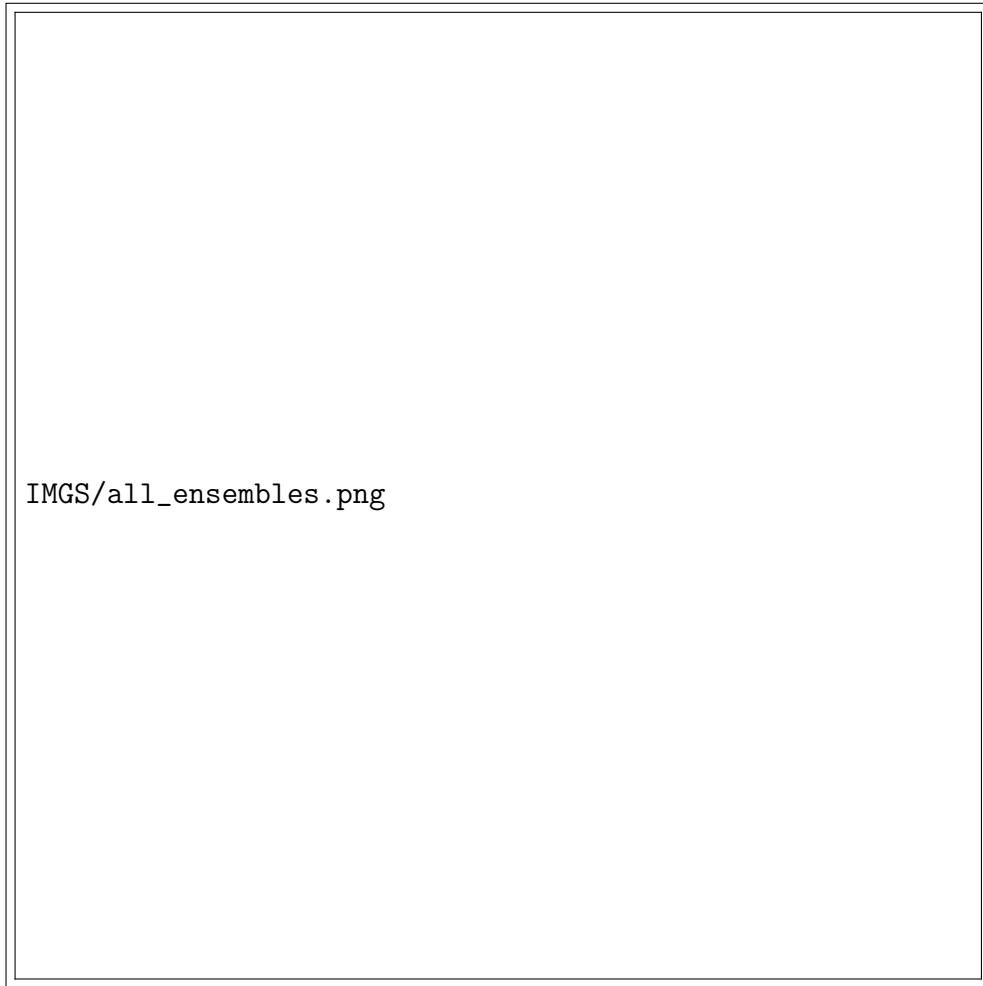


Figura 6.11 – Comparação de desempenhos de duas técnicas de ensemble pré-carregadas e do ensemble por média aritmética com o das redes individuais.

Resolvemos deixar na figura 6.11 o desempenho das redes individuais a fim de verificar o comparativo das estratégias de ensemble em relação aos resultados de cada uma. Fica evidente no gráfico que a curva azul possui a maior parte de seus pontos com valor bem menor que as demais. Esta curva corresponde ao desempenho do ensemble calculado a partir da função *DirichletEnsemble* do pacote *deeplearn*, que executa o cálculo do ensemble das redes através da técnica chamada por ensemble de Dirichlet-Markov. Apesar de ser uma técnica estatística complexa, se mostrou pior do que o desempenho das redes individuais e portanto, não serve para refinar o resultado conforme queremos.

Presente na figura 6.11 ainda está o resultado obtido via função *LogisticRegression*, do pacote *scikit-learn*, que executa uma regressão logística dos valores imputados. A regressão sobre os dados das previsões apresentou uma boa performance como meio de ensemble mas, manteve-se similar ao desempenho da EfficientNetB2. No topo do gráfico ainda está, na maior parte dos pontos, o ensemble calculado pela simples média aritmética das previsões, ainda que consistente em 1σ com a EfficientNetB2. Poderiam ainda ser testadas outras funções pré-carregadas para o ensemble, mas esses resultados nos in-

dicaram que a média se mostrou já uma estratégia de ensemble razoável e de simples implementação.

6.3 Classificação com dados da Segunda Edição do GLFC

Depois de explorar diversas alternativas para melhorar a detecção automatizada de lentes gravitacionais numa mesma base de dados, analisamos uma base de dados nova. Pois, apesar de todo o sucesso atingido, a base de dados utilizada ainda é uma base de dados simulados. Apesar de terem sido inspirados em imagens de dados reais obtidos por telescópios, constituem, como qualquer outra, de uma base de dados com os seus padrões de geração. Nem sempre transferir a capacidade de predição de uma rede em certa base de dados para outra base de dados é tarefa simples e, nesse intuito, daremos prosseguimento à nossa análise com outro conjunto de dados.

Uma outra base de dados simulados de lente gravitacional disponível na rede é a fornecida na segunda edição do GLFC, de 2020. Desta vez a base de dados de imagens de treino consiste de 100.000 (cem mil) objetos simulados em que pode haver ou não efeito forte de lente, de modo a tentarem reproduzir imagens do levantamento de galáxias do consórcio Euclid. Por sua vez, as imagens foram geradas de modo a tentar reproduzir as observações retiradas nas bandas VIS (espectro visível) e NISP J, Y e H (obtidos por fotometria de espectro infravermelho). Portanto, cada objeto da base de dados possui uma imagem em 4 bandas, sendo as imagens na banda VIS de dimensão 200×200 pixels enquanto nas bandas NISP J, Y e H, de dimensões 66×66 pixels. À princípio o primeiro desafio do uso dessa base de dados é o fato de haverem 4 bandas e, uma delas possuir imagens de dimensões diferentes das demais. Isso dificulta a entrada simultânea desses dados numa rede neural, mas foram realizados primeiros testes dessa base de dados, apenas normalizando as imagens onde foi obtido um *underfitting*.

Neste estágio do trabalho, já eram conhecidos os ganhadores desta segunda edição da competição: o time de inteligência artificial do CBPF CAST (*CBPF Arc Search Team*). Portanto, foram adotados os mesmos passos realizados pelo grupo para classificar a base de dados, e a mesma rede [9]. À princípio, cabe mencionar o tratamento das imagens que se fez necessário para que a rede neural conseguisse começar a executar a classificação. Diferentemente dos dados da primeira edição do GLFC, que requiseram apenas um pré-processamento simples, estes dados representaram um verdadeiro desafio para se descobrir qual a melhor maneira de pré-processá-los. No entanto, podemos refazer o caminho das estratégias que se mostraram mais eficazes.

6.3.1 Pré-processamento dos dados da segunda edição do GLFC

Observamos que 9 imagens que não continham todas as 4 bandas. Optamos por retirar completamente estes objetos da base de dados, uma vez representando um número

bastante pequeno em comparação ao tamanho total da base de dados.

O primeiro passo do pré-processamento da base de dados da segunda edição do GLFC foi a construção da tabela verdade para os objetos fornecidos. Diferentemente da primeira edição do GLFC, o catálogo dos objetos não possuía uma resposta pronta a respeito do seu caráter de ocorrer ou não lenteamento mas, ao invés, consiste de um arquivo de parâmetros das imagens simuladas. Entre os aspectos dos objetos descritos neste catálogo estão as (os):

- coordenadas x e y de suas curvas críticas;
- desvio pro vermelho da fonte e da lente;
- magnitude da fonte não lenteada presente no catálogo Euclid VIS;
- número de curvas críticas;
- distância da fonte ao centro de uma cáustica em radianos;
- área de largas curvas críticas em radianos quadrados;
- área da cáustica em radianos quadrados;
- número de pixels na imagem da fonte lenteada acima de 1σ ;
- fluxo total destes pixels em unidades de σ ;
- número de pixels na lente principal acima de 1σ ;
- fluxo total destes pixels em unidades de σ ;
- número de grupos separados de pixels da fonte;
- magnificação efetiva da fonte contabilizada em todas as bandas;
- contraste de brilho superficial médio entre a lente e a fonte em escala de pixels acima de um certo limiar;
- diferença de cor entre a lente e a fonte;
- massa do halo da lente principal;
- massa estelar da lente principal;
- magnitude da lente principal na banda de referência;
- número de fontes adicionadas;

Um critério sugerido pelos avaliadores no momento da classificação é tentar classificar como caso positivo de efeito forte de lente os objetos que possuem: número de grupos de pixels separados da fonte maior que zero; magnificação efetiva da fonte em todas as bandas maior que 1.6 e; número de pixels da fonte maior que 20. Seguindo este critério como definidor dos casos em que ocorre lenteamento, dentro do catálogo há um número de 49213 objetos que podem ser considerados como lentes.



IMGS/imgs_step_BOTH_af_pp_v_0_0_npp_4.png

Figura 6.12 – Dois exemplos de imagens em suas bandas H, J, Y, VIS e o resultado da combinação das três bandas HJY.

Uma vez definida a tabela verdade dos objetos que entrarão na rede, cabe agora fazer uma análise das imagens propriamente ditas. Na figura 6.12 temos alguns exemplos de imagens em cada banda, já modificadas pelas normalizações efetuadas em nosso código. No sentido de torná-las mais visíveis ao olho humano, durante o processo de gerar a visualização das imagens, foram realizados outros tipos de processamentos. Portanto, as cores das imagens não refletem exatamente as cores originais dos objetos diretamente

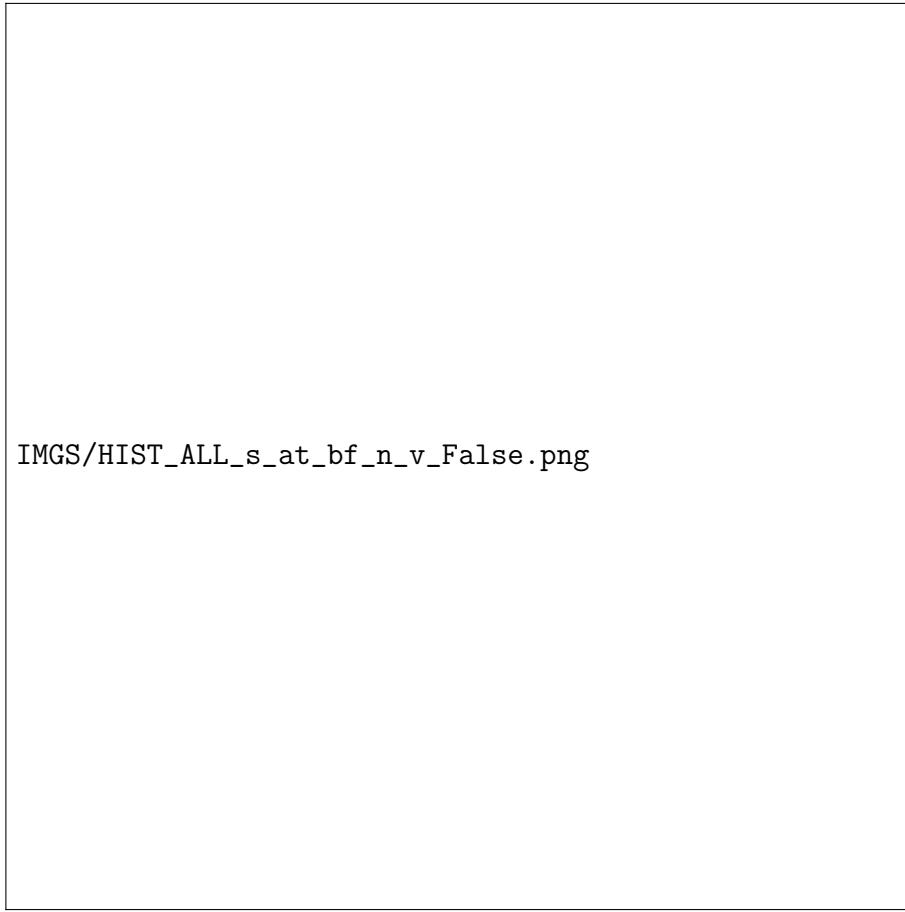
extraídos do tensor de dados. A primeira função utilizada nesse processamento é um ajuste de contraste. Funciona através de um mapeamento dos valores de pixels das imagens em uma escala de cinza, eliminando os valores abaixo e acima de 1% do mínimo e máximo respectivamente, para executar uma normalização e por fim retornar esse valores de pixel à matriz de dados. O efeito final desse ajuste é aumentar o contraste dos pixels da imagem. É durante esse processo que são geradas as figuras nas bandas H, J, Y e VIS da figura 6.12, ganhando cores. Depois disso, foi usada a subrotina `toimage`² para converter a matriz de dados numa única imagem resultante da sobreposição dos canais. O processo é análogo para a conversão de imagens nos canais R, G, e B em RGB, resultando na imagem da coluna "Result HJY".

É possível verificar que as imagens ainda possuem bastante ruído, não sendo possível verificar com precisão, nestes dois exemplos, se ocorre o efeito forte de lente ou é apenas uma sobreposição de galáxias próximas. No entanto, neste exemplo, nas imagens da primeira linha está presente o efeito forte de lente enquanto, nas imagens da segunda, não. Sabendo que o primeiro exemplo apresenta um caso positivo de lente, é possível identificar as manchas foscas ao redor da mancha branca central como imagens múltiplas de uma fonte longínqua.

No pré-processamento dos dados, um passo bastante comum é realizar a normalização dos dados. E a primeira abordagem para realizar essa normalização é colocar os dados entre zero e 1. Desta maneira, diminui-se bastante as escalas de valores para que um valor muito discrepante não provoque um grande desequilíbrio nos pesos da rede. Isto pode ser feito identificando o valor máximo de pixel das imagens e, em seguida, dividindo todos os valores dos pixels por esse máximo. Desta maneira, os pixels mais próximos do máximo atingirão um valor próximo de 1 enquanto aos menos próximos, cabe um valor próximo de zero. Antes de qualquer coisa, Cabe observar o histograma da distribuição de pixels das imagens 6.13.

Nesta figura, enquanto o gráfico abaixo mostra a distribuição de pixels para as imagens da banda VIS, a primeira mostra a distribuição das imagens nas bandas H, J, e Y, representadas nas cores azul, marrom e verde respectivamente, sobrepostas. A distribuição da banda VIS não foi inclusa às demais por conta de seus valores estarem uma ordem de grandeza acima, dado que as imagens possuem maior resolução, potencialmente dificultando a visualização das distribuições de pixels nas demais bandas. Como pode-se verificar, existe um considerável contingente de imagens que possuem pixels de valores bastante destoantes de todo o resto. O fato de haver um número grande de valores de sinais opostos também é apresenta uma dificuldade para a normalização direta. Estando a distribuição de valores centrada em zero, a maior parte dos dados ao ser dividida diretamente pelo máximo resultará em um número ainda mais próximo de zero, o que irá

² Por sua vez, esta função é uma subrotina para a conversão de uma tabela de dados em uma imagem visível da biblioteca *pillow*. Essa função está presente na versão 1.0.0 da biblioteca *scipy*, mas foi removida em versões mais recentes.



IMGS/HIST_ALL_s_at_bf_n_v_False.png

Figura 6.13 – Histograma de pixels das imagens nas bandas H, J, Y e VIS (abaixo).

afinar a curva. O efeito disto na visualização da imagem é um escurecimento, tornando ainda mais difícil a visualização a olho nu e, por conseguinte, distinção virtual da rede neural.

Uma saída natural decorrente desse problema é buscar, de algum modo, limitar esses valores de mínimo e máximo de modo a se manterem próximos do restante da amostra. Isso pode ser feito facilmente em python utilizando a função *clip* do pacote *numpy*, que limita os valores de um conjunto de dados à duas bordas do intervalo transformando todos os valores além dessas bordas (seja acima da superior ou abaixo da inferior) para valores da borda. Desta maneira, os máximos e mínimos globais são limitados em certos pontos. Resta saber que valores de máximo e mínimo seriam interessantes de se retirar das amostras. Uma saída comumente adotada em estatística é calcular um dos percentis dos valores de dados, como o 98º por exemplo, e tomá-lo como o máximo da distribuição. Desta forma, estamos excluindo uma minoria dos valores de pixel acima de um máximo comum ao resto e, com o auxílio da função para clipar dados, podemos transformar esses valores excluídos no valor deste percentil calculado.

Quanto ao mínimo, um problema que pudemos identificar foi a presença de valores negativos dentro da base de dados, o que fazia com que o ponto médio da distribuição da maioria dos valores se situasse num ponto próximo ao zero. De modo a evitar escolher

um mínimo entre esses valores e acabar tendo um mínimo de valor negativo, o modo de contornar esse problema foi tomado o negativo do percentil de índice 99,9 da distribuição de pixels e, em seguida, tomamos o negativo desse número. Esse constituiu o valor mínimo de nossos dados. Finalmente, encontrando esse máximo e mínimo o processo de normalização dos dados foi executado da seguinte forma: para cada imagem i de um objeto no canal C, a nova imagem normalizada será tal que

$$C_i = \frac{C_i - L_{inferior}}{L_{superior} - L_{inferior}} \quad (6.1)$$

onde $L_{inferior}$ constitui o percentil que corresponde ao mínimo utilizado e $L_{superior}$ o percentil que corresponde ao máximo. Repetido esse procedimento para cada canal de imagens, temos por fim as imagens normalizadas, prontas para entrar na rede neural, cujo histograma está presente na figura 6.14.



Figura 6.14 – Histograma final de pixels das imagens nas bandas H, J, Y e VIS.

Graças aos percentis serem calculados independentemente para cada canal de imagens, agora nessa figura os pixels de cada uma das bandas se distribuem em picos levemente diferentes - o que não constitui de exigência neste processo. O maior benefício dessa normalização, no entanto, está no fato de agora termos os dados distribuídos no intervalo de zero a um positivo. Isto poderá facilitar a entrada dos dados na rede neural, uma vez

que sinais diferentes na entrada podem mudar os sinais dos pesos durante o processo de retropropagação.

6.3.2 Desempenho do Ensemble de redes nos dados da Segunda Edição do GLFC

Uma vez que os dados estão processados e preparados, caberia agora inserí-los num classificador. O algoritmo desenvolvido para classificar a base de dados da primeira edição consiste de um ensemble das redes ResNet50 e EfficientNetB2. Cada uma das duas, pelo modo como estão pré-definidas por biblioteca, aceitam no máximo um tensor de dados de imagem com três canais. Sendo o caso da base de dados do II GLFC um tensor com quatro canais (H, J, Y e VIS), e tendo em mente um deles de maiores dimensões em relação ao anterior (VIS), fez-se necessário pensar sobre como esses dados poderiam entrar na rede de ensemble. À princípio, a ideia principal seria selecionar canais específicos ignorando os demais, dado que cada canal é referente ao mesmo objeto em diferentes filtros (e dimensões da imagem, no caso da banda VIS). Por isso, como primeira abordagem com essa nova base de dados experimentamos a classificação utilizando apenas os dados das camadas H, J e Y, que possuem dimensões semelhantes, como entradas das redes ResNet50 e EfficientNetB2.

Quanto ao tamanho da amostra, optamos por reproduzir os mesmos número de objetos para o treino 16000 (dezesseis mil) utilizados na classificação da base de dados da primeira edição, aumentando o conjunto de validação para 3000 objetos e o de teste como 3000 também. Valor que sequer chega a metade de toda a base de dados, de 10^5 (cem mil) objetos, o que nos permite fazer algo inédito em relação a base de dados anterior: gerar mais de um conjunto disjunto de treino de 16000 objetos. No entanto, um dos pontos principais nesse trabalho seria buscar verificar se é possível definir um limite inferior fixo para a quantidade de dados necessária neste problema. Pela base de dados da primeira edição do GLFC, 80 seria um número mínimo necessário de dados que se fariam disponíveis, tendo em mente uma classificação com AUC de pelo menos 0,8. Foi mantido o número de épocas da execução, tamanho do lote de dados que entra na rede à cada etapa do treino, escolha de otimizadores e a estrutura. Dado o tamanho da base de dados, realizamos a execução em 5 pastas de dados de treino mas, no intuito de evitar a poluição visual no trabalho, optamos por mostrar os gráficos referentes apenas à primeira de cada uma das duas redes. Para a acurácia, temos os gráficos da figura 6.15.

Nos gráficos de ambas as redes ocorre o seguinte fenômeno: enquanto a acurácia de treino em azul converge para 1, a acurácia de validação em laranja se mantém próxima ao aleatório, flutuando por volta de 0,5. Fica evidente nos gráficos, portanto, uma distância entre as acuráncias, um espaçamento que apenas cresce conforme a acurácia de treino con-

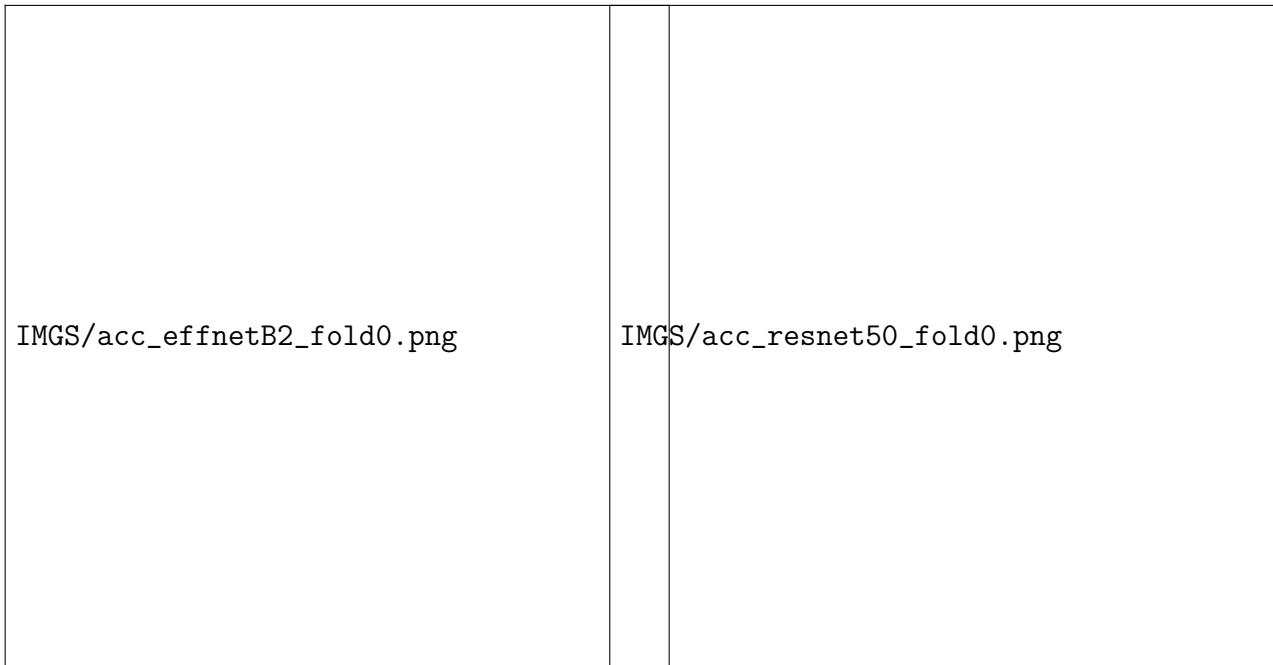


Figura 6.15 – Acurácia do Treino da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.

verge para 1,0. Esse fenômeno estatístico é conhecido como sobreajuste³. O sobreajuste comumente ocorre no ramo do aprendizado de máquina quando o modelo de classificação se especializa excessivamente nos dados de treino. Significa que, enquanto a classificação sobre os próprios dados de treino torna-se ótima, o modelo adquire uma grande dificuldade de executar um teste cego. Esse comportamento está bem reproduzido na figura 6.15: enquanto a linha azul que refere-se ao desempenho do modelo nos dados de treino demonstra uma ótima acurácia, a linha laranja diz respeito a mesma acurácia no teste cego sobre os dados de validação: um regime de classificação indistinguível da escolha aleatória. Esse comportamento também pode ser verificado na função custo do treino das duas redes, presente na figura 6.16.

Neste gráfico da função custo o sobreajuste fica ainda mais evidente. Enquanto o custo de treino cai convergindo para valores próximos de 0,0, o custo de validação cresce rapidamente, atingindo patamares muito maiores que 1,0, omitidos nesse gráfico para não dificultar a visualização. Diferentemente dos gráficos de função custo para classificações com sucesso, as linhas de custo de treino e validação não se encontram. Por fim, como último recurso de nossa avaliação, cabe verificar qual foi o desempenho em termos da curva ROC de ambas as redes, presentes na figura 6.17.

Aqui verifica-se o efeito final do sobreajuste na classificação de dados. A linha pontilhada se refere a uma AUC de 0,5, um regime de classificação próximo a escolha aleatória. Mas na figura, em ambos os casos a ROC fica apenas um pouco acima desse valor. O modelo não conseguiu realizar o treino recebendo em sua entrada as imagens das cama-

³ Tradução direta do termo comumente usado em inglês *overfitting*.



Figura 6.16 – Pontos da Função Custo do Treino da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.

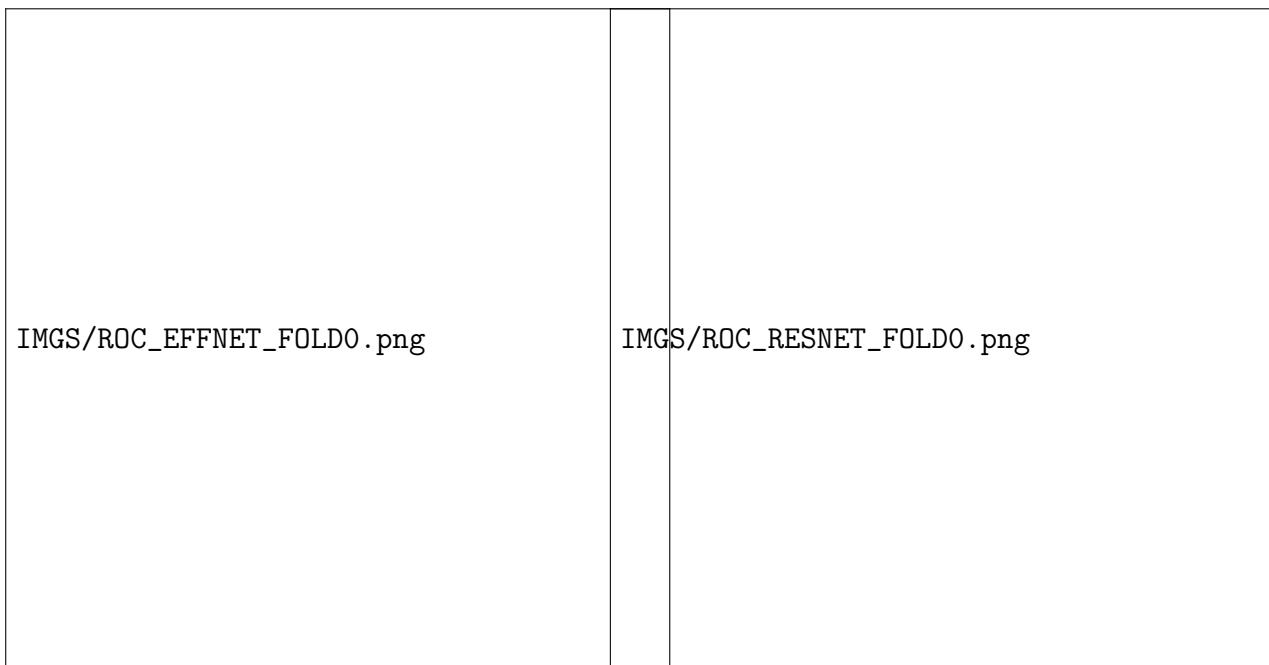


Figura 6.17 – Curva ROC e AUC de cada modelo ao final da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC.

das H, J, e Y. Resta ainda verificar como se dá esse desempenho nas imagens da banda VIS, uma vez que possuem resolução maior. Para isto, como as redes do modo como estão arquitetadas precisam de tensores de três canais na entrada, preparamos um tensor com três canais, sendo cada um deles o próprio canal VIS. Por fim, executamos as redes nas mesmas condições, usando a mesma quantidade de objetos e, para as duas redes, obtivemos as curvas ROC da figura 6.18.

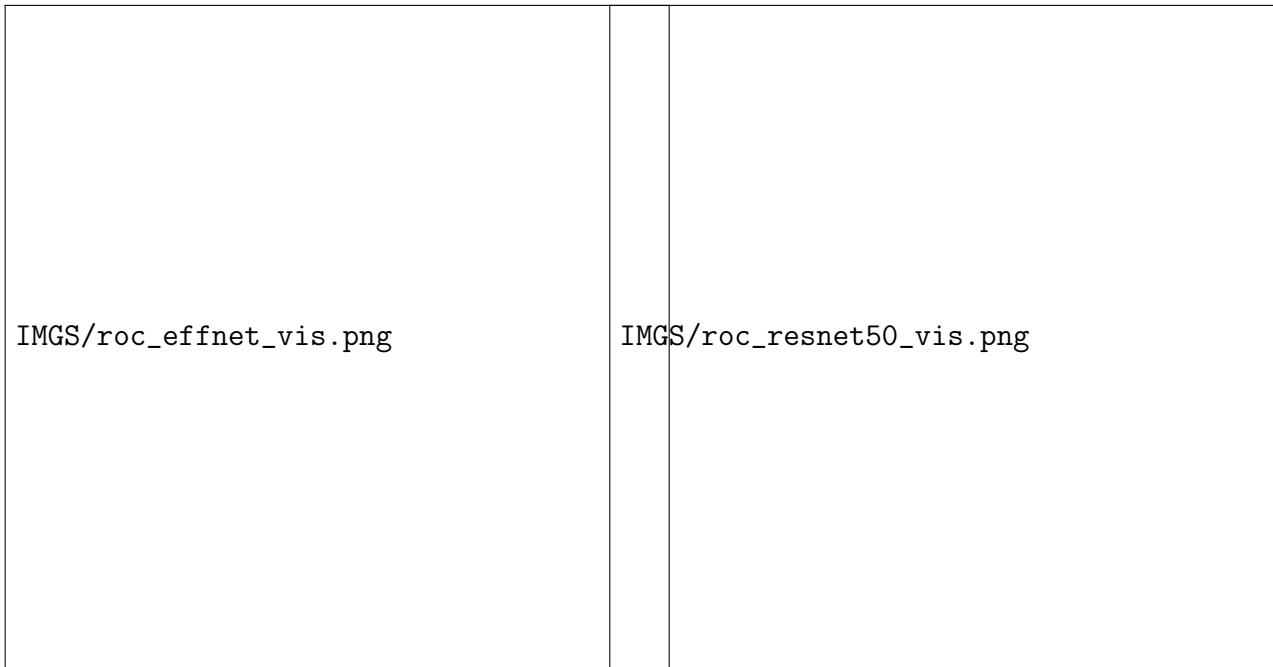


Figura 6.18 – Curva ROC e AUC de cada modelo ao final da primeira pasta das redes EfficientNet B2 (à esquerda) e ResNet50 (à direita) no Ensemble de Redes sobre a segunda base de dados do GLFC, usando a banda VIS.

Novamente, temos as duas redes apresentando uma área sob a curva próxima do valor do regime aleatório de 0,5. Apesar de, para o caso de usarmos a banda VIS não ser evidenciado nesse trabalho os gráficos de acurácia e custo, não se verifica que ocorre sobreajuste. Pelo contrário, neste caso as acurácia e custo de treino seguem com valor praticamente constante, indicando que nenhum aprendizado sequer foi extraído dos dados. Não é possível afirmar, portanto, que esse modelo é capaz de realizar previsões confiáveis num teste cego acerca da ocorrência ou não do fenômeno de lente. Cabe relembrar que foram utilizados 16000 objetos para treino na execução que retornou os resultados das figuras 6.15, 6.16 e 6.17. Existem técnicas computacionais a fim de diminuir ou retirar o sobreajuste, como a regularização da função custo, mudar o otimizador, dentre outras, mas não obtivemos sucesso. Este resultado também se mostrou presente ao realizar o treino com os dados sem o pré-processamento e com o pré-processamento. A melhor alternativa é escolher uma nova arquitetura para a rede neural, no intuito de que ela possa se adaptar melhor à base de dados. Desta forma, descartamos a possibilidade de utilizar o ensemble das redes ResNet50 e EfficientNet B2 na expectativa de encontrar um bom desempenho na classificação com apenas 80 objetos de treino, diferentemente da primeira base de dados.

6.3.3 Treino usando a concatenação de duas EfficientNet B2

Uma vez que a nossa base de dados está devidamente normalizada, pronta e tendo passado nos primeiros testes, o passo seguinte na classificação é o de buscar encontrar redes

neurais que sejam eficientes para lidar com o caso que temos em mãos. A nossa base de dados, à título de recapitação, consiste de cem mil imagens em quatro bandas, sendo três bandas de imagens com dimensões 66x66 e uma extra com imagens em dimensões 200x200. Normalmente seria interessante redimensionar as imagens 200x200 para o mesmo tamanho das imagens das demais bandas, no entanto, ao fazê-lo estaríamos abrindo mão da maior resolução da imagem na banda VIS, que poderia contribuir na classificação dos objetos. Por outro lado, se fosse feito o redimensionamento das imagens 66x66 para o tamanho 200x200, isso poderia distorcer as imagens e gerar artefatos. O desafio era, portanto, tentar encontrar uma maneira de imputar na rede as imagens em todas as bandas que nos foram fornecidas. Nesse intuito, o modo como se obteve a decisão final acerca do melhor procedimento que poderia ser utilizado seguiu de modo experimental. Primeiro testamos a rede numa arquitetura já conhecida e, caímos num problema de sobreajuste. O passo seguinte consiste de encontrar uma outra rede neural que seja capaz de resolver o nosso problema.

Reconhecendo o pré-processamento dos dados feito pelo grupo CAST vencedor da segunda edição do GLFC, uma primeira aposta possível para a rede neural também poderia ser a rede neural utilizada pelo grupo: a EfficientNet B2. O fato de ser uma rede com poucos parâmetros e, por conseguinte, cujo treino pode ser bem mais rápido do que outras a tornou bastante interessante para a competição. Dadas as pretensões de realizar várias execuções neste trabalho, investigando sempre o problema de classificar uma base de dados com o mínimo possível de objetos.

Quanto ao problema do tensor de imagens possuir quatro dimensões, uma delas maior que as demais, a solução adotada pelo grupo consistiu de utilizar duas EfficientNet B2. Em uma delas entra parte dos dados, na outra entra apenas os dados na banda VIS e, por fim, a última camada dessas duas redes estará concatenada de modo que a previsão final receba contribuições de cada uma das entradas. Isto é, as duas EfficientNet B2 serão treinadas ao mesmo tempo, com suas diferentes entradas, motivo pelo qual iremos nos referir a essas redes como Efficient Net B2 concatenadas. Precisamente, em uma das EfficientNet B2 a entrada consiste em um tensor com 3 canais de imagens 66x66, sendo um deles o canal de imagens na banda Y e, os dois canais seguintes, matrizes nulas de dimensão 66x66. Já na outra EfficientNet B2, a entrada consiste de um tensor de 3 canais de imagens 200x200, o primeiro deles com as imagens na banda VIS e os dois seguintes também consistindo de matrizes nulas de dimensão 200x200. Essa escolha se deu por conta de estas serem as bandas que se provaram mais influentes na classificação da base de dados.

Para entrar na prática, cabe por fim colocar este modelo à prova. Quanto a divisão dos dados, foi utilizada toda a base de dados de 100.000 objetos, fazendo uma divisão de 80.000 para treino, 10.000 para validação e 10.000 para o último teste cego. Utilizamos o otimizador RAdam (*Rectified Adam* [10]), que consiste de uma versão do otimizador

Adam⁴ em que é introduzido um termo para retificar a variância da taxa de aprendizado, resolvendo alguns problemas de convergência [10]. Foram mantidos o tamanho do lote e o valor da taxa de aprendizado. Nestas condições, o resultado obtido em termos de curva ROC foi a performance de um teste cego de acordo a figura 6.19.



Figura 6.19 – Desempenho das redes EfficientNet B2 concatenadas nos dados da segunda edição do GLFC.

A curva em azul representa a ROC para as previsões dos casos tabelados como lente, para a qual a área sob a curva fica por volta de 0,84, sendo esta a pontuação da classificação. Tal pontuação já demonstra que a arquitetura utilizada adquiriu certa capacidade de discernimento entre os conjuntos de dados, diferentemente dos testes anteriores, o que valida o uso da arquitetura proposta.

A partir deste resultado, utilizamos esta arquitetura para definir múltiplos treinos a fim de investigar o limite de poucos dados para uma classificação ótima nessa base de dados. Foram mantidos o tamanho dos conjuntos de validação e de teste como 10.000, dada a grande quantidade de dados disponível. Lançando mão do recurso dos múltiplos treinos em *folds*, fizemos as execuções reduzindo cada vez mais o tamanho dos conjuntos de treinamento e registramos as médias dos *folds*, resultando na figura 6.20.

Nesta figura, tem-se no eixo horizontal o tamanho dos conjuntos de treino enquanto o vertical refere-se aos valores de AUC. A linha em azul legendada como "Train" refere-se

⁴ Um método estocástico de otimização de descida de gradiente proposto em [4]

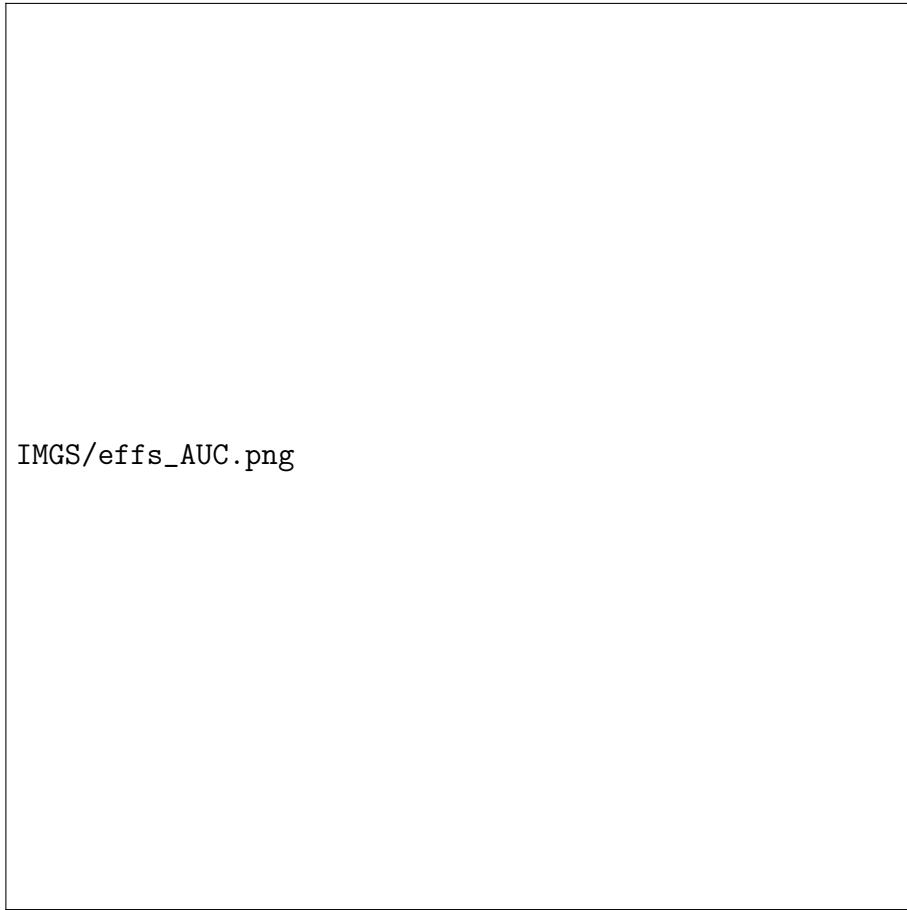


Figura 6.20 – Desempenho das redes EfficientNet B2 concatenadas nos dados da segunda edição do GLFC, para diversos tamanhos de conjuntos de treino.

aos valores médios de AUC no caso de um teste do modelo em sua última época sob os próprios dados de treino. A linha em vermelho ”Teste” indica a performance do modelo em sua última época num teste cego. E por fim, a linha em verde indica a performance do modelo em sua melhor época, de acordo a função custo de validação calculada.

Estes três casos foram colocados no gráfico da figura 6.20 a fim de observar como podem influenciar as possibilidades de tomarmos um limite de poucos dados. Observando a curva do melhor modelo, seria possível afirmar que uma quantidade mínima de dados de treino necessários para se atingir um limite de poucos dados seria 9.000 ou 10.000 dados. Já pela curva de teste, esse valor precisaria ser um pouco mais alto, apresentando um pico em 15.000 acompanhado de baixas até um pico um pouco maior em 20.000 dados. Portanto, enquanto de acordo o raciocínio da base de dados da primeira edição teríamos 15.000 como limite inferior aqui, é possível diminuir um pouco esse limite se fizermos a seleção do modelo que melhor performou ao longo das épocas para 10.000 ou até 9.000 objetos. Este será, portanto, o parâmetro a ser utilizado como último resultado destes testes, no que se refere ao limite de poucos dados.

A queda abrupta nos valores iniciais expõe mais uma vez o regime de sobreajuste: enquanto o aprendizado com dados de treino atingem uma alta taxa de acerto, as previsões

de testes cegos tornam-se indistinguíveis do regime aleatório na classificação binária.

6.3.4 Desempenho das Efficient Net B2 concatenadas na base de dados do GLFC variando o critério da tabela-verdade

Um teste que pode se mostrar promissor para trabalhos futuros, além de no sentido de explorar o potencial de classificação puro da Efficient Net B2 concatenada nesta base de dados é explorar como melhora a classificação de acordo o critério de seleção dos casos de lente. Ao introduzir esta nova base de dados, exploramos de modo geral quais são as variáveis disponíveis na base de dados e quais delas se mostraram relevantes para definir a tabela verdade. De acordo a seção 6.3.1, seriam considerados casos positivos de lenteamento os objetos que demonstrassem: número de grupos de pixels acima de 1σ do *background* maior que zero; magnificação efetiva da fonte em todas as bandas maior que 1.6 e; número de pixels da fonte maior que 20. Por esse critério, entre os 100.000, 49.213 objetos foram tabelados como lentes.

No entanto, este é um critério mínimo de objetos que serão adotados como lentes, isto é, existem muitos casos de objetos que são difíceis de se discernir como lentes, entre alguns em que o fenômeno é bem evidente. Alterando esses critérios, pode ser possível filtrar com maior precisão os resultados da classificação. Resolvemos fazer as seguintes flutuações para este estudo: magnificação efetiva da fonte em todas as bandas no intervalo 1,1-1,6-2,0 e número de pixels da fonte de 10-50-100. Durante as execuções utilizamos 20.000 imagens de treino, 10.000 para validação e 10.000 para testes, e os demais parâmetros que já se provaram úteis nesta base de dados. Na tabela 1, as AUCs foram escolhidas como parâmetro para verificar as eficárias dos treinos em cada caso, destacando o desempenho do modelo final com os dados de treino e num teste cego, e um teste cego usando o melhor modelo. A linha destacada em amarelo corresponde à performance adotando o critério de lente proposto no desafio.

Tabela 1 – Comparaçāo entre AUCs variando o critério de Lente Gravitacional

Mag.	Nº Pixels	Nº Lentes	AUC		
			Final - Treino	Final - Cego	Melhor - Cego
1,1	10	67102 (67,1 %)	0,83	0,78	0,78
1,1	50	36961 (37 %)	0,93	0,89	0,89
1,1	100	22647 (22,6 %)	0,94	0,91	0,94
1,6	10	57887 (57,9 %)	0,80	0,75	0,77
1,6	20	49213 (49,2 %)	0,86	0,79	0,80
1,6	50	33065 (33 %)	0,92	0,87	0,88
1,6	100	20834 (20,8 %)	0,95	0,91	0,93
2,0	10	45684 (45,7 %)	0,80	0,76	0,78
2,0	50	28040 (28 %)	0,93	0,88	0,88
2,0	100	18528 (18,5 %)	0,95	0,92	0,92

Na tabela, “Mag.” refere-se à magnificação mínima, “Nº Pixels” se refere ao número mínimo de pixels da fonte. “Nº Lentes” refere-se ao número de objetos da base de dados que atende a este critério e, portanto, constituem de casos positivos de ocorrência de lenteamento gravitacional. As colunas sob o termo “AUC” referem-se aos valores de AUC encontrados ao final do treino para cada um dos seguintes casos: “Final - Treino” referentes ao modelo final utilizando dados de treino, “Final - Cego” referentes ao modelo final num teste cego e, “Melhor - Cego” referindo-se ao melhor modelo escolhido com base na melhor época. No caso do critério já adotado previamente, de magnificação acima de 1,6 e número de pixels maior que 20, os seguintes valores de AUC foram obtidos: 0,84 para o final com dados de treino, 0,78 para o final no teste cego e 0,80 para o melhor modelo num teste cego.

Deste teste é possível verificar que as situações que apresentam maiores valor de AUC correspondem aos valores de magnificação maior que 1,1 e número de pixels de 50 e 100, maior que 1,6 e número de pixels de 50 e 100, e de magnificação maior que 2,0 com número de pixels maior que 50 e 100. Em especial, os casos de número de pixels maior que 100 foram os que tiveram melhor desempenho, alcançando um desempenho do melhor modelo num teste cego de 0,93. Este, portanto, pode ser dito como o fator determinante para uma maior AUC no modelo. Este valor está bem acima do valor utilizado na análise do número de samples, de 0,80. Esta análise sugere, portanto, que utilizar esse critério de lente para o número de objetos é capaz de melhorar consideravelmente a performance da classificação do modelo, em relação ao critério de lente descrito na proposição da segunda edição do GLFC.

6.4 Conclusões e Discussão

Considerando tudo o que foi exposto no capítulo, podemos realizar algumas considerações.

- No início da seção 6.2, vimos que a ResNet50 por si só, sem grandes adequações, é capaz de reproduzir os resultados competitivos da primeira edição do GLFC. A AUC encontrada foi de 0,965 enquanto o valor mais alto em Metcalf et al. é de 0,98 [6]. Este resultado foi obtido usando 16000 dados de treino, 2000 de validação e 2000 de teste, numa execução única.
- Ao começar a realizar múltiplos treinos em dez pastas na subseção 6.2.2, foi possível adicionar ao gráfico barras de erro e tomar o valor pela média. Começamos também a verificar o desempenho da classificação diminuindo o tamanho da base de dados de treino. Vimos que aqui, o regime ótimo de classificação de acordo Hosmer (2013) [8] seria atingido por volta dos 450 objetos, na figura 6.6. Significa que em média, 450 amostras de lentes e não-lentes seriam necessários para atingir uma qualidade

de classificação próxima a 0,8. Idealmente, um número próximo a metade destes deveriam ser casos positivos de lenteamento, isto é, 225. Este valor já é menor do que o tamanho da base de dados do *Master Lens Database*, podendo resultar na execução até de mais de um *fold*.

- Observamos o efeito da aumentação de dados e do pré-carregamento de pesos, nas respectivas subseções 6.2.3 e 6.2.4. Enquanto a aumentação de dados pouco contribuiu, inicializar os pesos evitando o regime aleatório chegou a reduzir o número de recortes de lentes e não lentes para 160. Significaria ter 80 casos de lentes conhecidos, o que reduz ainda mais esse limiar necessário.
- Já na subseção 6.2.5, este limite foi reduzido para 80 objetos na base de dados, através da estratégia do ensemble de redes pela média simples. Isto aponta um indício de que o ensemble de redes, portanto, é uma ferramenta promissora. Este é semelhante ao obtido em uma das pulições relacionadas a dissertação, Castro & Teles et al. (2021) [11], entretanto, no referido trabalho, algumas escolhas metodológicas fizeram as barras de erro menores. De qualquer modo, os valores concordam dentro do erro. Significaria precisar de 40 casos de efeito de lente a fim de fazer uma base de dados que pode ser treinada. Tomando os 675 casos do *Master Lens Database*, seria possível fazer 10 pastas com dados de lentes (resultando em 400 lentes utilizadas) e ainda sobrariam 275 para compor um conjunto de validação.
- Cabe enunciar, no entanto, que esse resultado é válido apenas para a rede neural de ensemble da ResNet50 e EfficientNet B2, nos dados da primeira edição do GLFC. Nos dados da segunda edição, como vimos na subseção 6.3.2, esse mesmo ensemble de redes não consegue recuperar performance similar, se tornando indiferente da classificação aleatória. Para esta base de dados foi necessária uma outra rede neural, como a concatenação de duas Efficient et B2, que tratam a situação de imagens com resoluções diferentes.
- Na subseção 6.3.3, vimos o desempenho da EfficientNetB2 na base de dados da segunda edição. Seria bastante satisfatório conseguir reproduzir o limite mínimo de 80 imagens de treino necessárias, no entanto, a arquitetura utilizada não foi capaz de tanto na nossa condição de uso. Neste trabalho, atingimos o limiar com cerca de 10.000 dados, valor bem alto em relação a quantidade de registros de eventos de lenteamento já catalogados. Não se pode esquecer, no entanto, que a base de dados da segunda edição é muito mais complexa.
- Por fim, na subseção 6.3.4 tivemos o uso de um recurso que poderia ser usado para refinar os resultados do uso das duas EfficientNetB2 concatenadas nos dados da segunda edição: a restrição do critério de lente. Verificamos que restringir principal-

mente o número de pixels da fonte para acima de 100 teve efeitos bastante positivos no aumento da AUC.

Estas foram as conclusões que puderam ser retiradas do trabalho, e que poderão nortear esforços futuros.

CAPÍTULO 7

CONSIDERAÇÕES FINAIS

Neste trabalho verificamos algumas estratégias que podem ser usadas para melhorar a classificação de lentes gravitacionais nas bases de dados do *Gravitational Lens Finding Challenge*. Alcançamos o limiar de 80 objetos na base de dados da primeira edição do GLFC, como número que deveria contemplar amostras contendo lentes gravitacionais e não, enquanto para os dados da segunda edição a marca atingida foi de 10.000 objetos. Cabe observar o efeito da escolha da arquitetura neste resultado final, e a importância da otimização das variáveis utilizadas na classificação. À princípio, não se imaginava que um esquema de rede neural que performou tão bem nas bases de dados da primeira edição, como o ensemble de uma ResNet50 e EfficientNet B2, precisaria ser revisto na base de dados da segunda edição, desde que se executasse o treino sem uso da primeira base [9].

Por fim, a não obtenção do mesmo limiar de dados obtidos com a base de dados da primeira edição do GLFC nos apontou que ainda havia trabalho a ser feito. Provavelmente há alternativas metodológicas que poderão melhorar esse limiar de 10.000 para estas bases de dados. De fato, para bases de dados restritas a um pequenos número de exemplos, a capacidade de generalização da rede é essencial e portanto não encontramos uma solução única, devendo cada caso ser discutido.

No presente, um gargalo para o desenvolvimento deste tipo de técnica costuma ser tempo de máquina. No caso da ResNet50, por exemplo, a execução inicial exige o aguardo de um treino durante 50 épocas. Separar os dados em dez pastas exigiria aguardar este tempo de 50 épocas dez vezes, e treinar com diferentes tamanhos a base de dados de treino exigiria aguardar 50 épocas dez vezes para cada tamanho de conjunto de treino. Cada novo recurso incorporado à rede aumentava ainda mais o tempo de execução necessário, e no final ainda havia uma rede a mais pra ser treinada em cada um destes passos, quando optamos pelo ensemble. Já no caso da concatenação de EfficientNet B2, apesar da EfficientNet possuir menos parâmetros, as camadas convolucionais da rede eram largas devido a maior resolução das imagens, o que também atrasou o tempo de treino. O uso de

GPUs é capaz de fazer este tipo de análise em curto intervalo de tempo. No entanto, nem sempre GPUs e clusters de GPUs não se encontram largamente disponíveis na comunidade acadêmica.

Como sugestão de trabalhos futuros, tem-se:

- Como consequência direta da descoberta da melhora da performance das EfficientNet B2 concatenadas na base de dados da segunda edição do GLFC, por conta da mudança do critério de lente, conforme a subseção 6.3.4, executar novas execuções reduzindo o tamanho do conjunto de treinamento. Do mesmo modo que foi feito nas seções 6.3.3 e 6.2.1, a fim de investigar qual seria o limiar de poucos dados alcançados com o melhor (ou cada) critério.
- Verificar como o uso de outras classes de EfficientNet concatenadas poderia alterar o resultado da classificação na base de dados da segunda edição do GLFC.
- Verificar como as EfficientNet B2 concatenadas se sairiam na base de dados da primeira edição do GLFC.
- Treinar tanto as EfficientNet B2 concatenadas quanto o ensemble da ResNet50 e EfficientNet B2 nas respectivas bases de dados em que melhor performam e, realizar um teste cego com ambas utilizando dados reais (do *Master Lens Database*, por exemplo).
- E como consequência do acima, utilizar uma base de dados reais como acervo de treino destes modelos de RNC supracitados, a fim de avaliar o seu desempenho de classificação. Diferentes levantamentos poderiam oferecer diferentes desafios nessa classificação de imagens. Não seria de se surpreender que as diferenças de filtros e instrumentos utilizados nos levantamentos fosse o suficiente para viabilizar uma arquitetura de RNC em um deles e, inviabilizar em outro.

REFERÊNCIAS

- [1] Imre Bartos and Marek Kowalski. Multimessenger astronomy. In Multimessenger Astronomy, 2399-2891, pages 1–1 to 1–18. IOP Publishing, 2017.
- [2] K. Herner, J. Annis, D. Brout, M. Soares-Santos, R. Kessler, M. Sako, R. Butler, Z. Doctor, A. Palmese, S. Allam, D. L. Tucker, F. Sobreira, B. Yanny, H. T. Diehl, J. Frieman, N. Glaeser, A. Garcia, N. F. Sherman, K. Bechtol, E. Berger, H. Y. Chen, C. J. Conselice, E. Cook, P. S. Cowperthwaite, T. M. Davis, A. Drlica-Wagner, B. Farr, D. Finley, R. J. Foley, J. Garcia-Bellido, M. S. S. Gill, R. A. Gruendl, D. E. Holz, N. Kuropatkin, H. Lin, J. Marriner, J. L. Marshall, T. Matheson, E. Neilson, F. Paz-Chinchón, M. Sauseda, D. Scolnic, P. K. G. Williams, S. Avila, E. Bertin, E. Buckley-Geer, D. L. Burke, A. Carnero Rosell, M. Carrasco-Kind, J. Carretero, L. N. da Costa, J. De Vicente, S. Desai, P. Doel, T. F. Eifler, S. Everett, P. Fosalba, E. Gaztanaga, D. W. Gerdes, J. Gschwend, G. Gutierrez, W. G. Hartley, D. L. Hollowood, K. Honscheid, D. J. James, E. Krause, K. Kuehn, O. Lahav, T. S. Li, M. Lima, M. A. G. Maia, M. March, F. Menanteau, R. Miquel, A. A. Plazas, E. Sanchez, V. Scarpine, M. Schubnell, S. Serrano, I. Sevilla-Noarbe, M. Smith, E. Suchyta, G. Tarle, W. Wester, and Y. Zhang. Optical follow-up of gravitational wave triggers with DECam during the first two LIGO/VIRGO observing runs. Astronomy and Computing, 33:100425, October 2020.
- [3] Frederic B. Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. The Journal of Symbolic Logic, 9(2):49–50, 1944.
- [4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
- [5] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5353–5360, 2015.

- [6] R Benton Metcalf, MASSIMO Meneghetti, Camille Avestruz, Fabio Bellagamba, Clécio R Bom, Emmanuel Bertin, Rémi Cabanac, F Courbin, Andrew Davies, Etienne Decencière, et al. The strong gravitational lens finding challenge. *Astronomy & Astrophysics*, 625:A119, 2019.
- [7] Norbert Wiener, Norbert Wiener, Cyberneticist Mathematician, Norbert Wiener, Norbert Wiener, and Cybernéticien Mathématicien. *Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications*, volume 113. MIT press Cambridge, MA, 1949.
- [8] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.
- [9] CR Bom, BMO Fraga, LO Dias, P Schubert, M Blanco Valentin, C Furlanetto, M Makler, K Teles, M Portes de Albuquerque, and R Benton Metcalf. Developing a victorious strategy to the second strong gravitational lensing data challenge. *arXiv preprint arXiv:2203.09536*, 2022.
- [10] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.
- [11] Icaro Castro, Kayque Teles, Clécio Bom, and Tatiana Escovedo. Desenvolvendo um ensemble de redes profundas para identificação de lentes gravitacionais: Aplicação em regime de poucos dados. *NOTAS TÉCNICAS*, 11(1), 2021.