UNIVERSIDADE FEDERAL DO PAMPA
BACHARELADO EM ENGENHARIA COMPUTAÇÃO

IGNACIO PINEIRO GARCIA

DEEP LEARNING: CONVOLUTIONAL NEURAL NETWORKS FOR LUNG CANCER DETECTION AND CLASSIFICATION

Bagé
February 2023

IGNACIO PINEIRO GARCIA

DEEP LEARNING: CONVOLUTIONAL NEURAL NETWORKS FOR LUNG CANCER DETECTION AND CLASSIFICATION

Graduation Work presented in partial fulfillment of the requirements for a Bachelor's degree in Computer Engineering

Advisor: Gerson Alberto Leiria Nunes

Bagé
February 2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

G216d Garcia, Ignacio Pineiro

Deep Learning: Convolutional Neural Networks for lung cancer detection and classification / Ignacio Pineiro Garcia.

- 2023.

79 f.: il.

Advisor: Gerson Alberto Leiria Nunes

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Campus Bagé, Bacharelado em Engenharia Computação, 2023.

1. Deep learning. 2. Convolutional neural networks. 3. Lung cancer. 4. CT scans. I. Gerson Alberto Leiria Nunes. II. Título.

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

IGNACIO PINEIRO GARCIA

DEEP LEARNING: CONVOLUTIONAL NEURAL NETWORKS FOR LUNG CANCER DETECTION AND CLASSIFICATION

Graduation Work presented in partial fulfillment of the requirements for a Bachelor's degree in Computer Engineering.

Graduation Work defended and approved in: 01 of february of 2023.

Judging committee:

_____

Prof. Dr. Gerson Alberto Leiria NunesOrientador

UNIPAMPA

_____

Prof. Dr. Milton Roberto Heinen

UNIPAMPA

_____

Prof. Dr. Fabio Luís Livi Ramos

UNIPAMPA

Assinado eletronicamente por **GERSON ALBERTO LEIRIA NUNES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/02/2023, às 07:25, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO LUIS LIVI RAMOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/02/2023, às 11:47, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MILTON ROBERTO HEINEN**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 08/02/2023, às 19:06, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1049221** e o código CRC **05329B5B**.

Referência: Processo nº 23100.002573/2023-40 SEI nº 1049221

I am dedicating this thesis in memory of my grandfather, "Dodo". Rest in peace, you will always be remembered in our hearts.

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my deepest gratitude to all my friends for their unwavering support over the past five years. Although I could write about each of you, I know it would take me weeks to do so, and it still wouldn't be enough. So, to keep it brief, you guys are amazing. Thank you for cheering me up and for pretending that my bad moods were usual, especially on those stressful days. I would also like to thank all my classmates for their support throughout this process, especially Kelvin and Thiago. Thank you for all conversations, laughs and complaints shared over the last year. Additionally, I must thank Michel for being my duo in almost all my graduation, even when I was a little lost. Thank you for showing me that we can be brothers without sharing blood. Moreover, I want to express my sincere appreciation to my advisor, Gerson, for all the assistance, guidance, and support provided. I also want to thank all the professors and the university for the knowledge imparted, even being always over-stressed with the numerous assignments. Lastly, I would like to thank my family, even though I'm sure they don't understand much of what I've been studying. Sister, thank you for holding my back when I was away. I know I don't say these things frequently but, you've always been an example for me. Mom and dad, thank you for all of your support and love. You're my biggest strength fount. You've provided me with everything I needed and even more - opportunities and things you couldn't have. Thanks for keeping your faith in me. Everything that I have and everything that I will achieve is because of you. It is a great honor to be your son.

‘Do you know how to forge a sword? When you’re making a sword, you strike and strike them again to get rid of all impurities and increase the purity of the blade, so a durable sword can be forged. It’s alright to cry or go off by yourself. Just don’t ever give up. Believe in yourself. Hone it to perfection. Become the most resilient blade of all!”

— Jigoro Kuwajima

ABSTRACT

Lung cancer is a disease caused by uncontrollable tissue growth in the lungs. This kind of malignant tumor has become the cancer with the highest mortality rate worldwide. Fortunately, when lung cancer is detected in its early stages, the survival rate of patients has great improvement. Due to the Computerized Tomography (CT) scans technology, it is possible to obtain detailed internal images of the body, and this information can be employed in the training of systems to detect and classify malignant nodules. Nevertheless, CT images normally have high dimensionality, and they require a great time effort of a specialist to be analyzed. Furthermore, these professionals need to take care of diverse patients in the meantime. Consequently, factors such as fatigue can lead them to make mistakes. Computer-Aided Diagnosis Systems are extremely useful in these cases, they can assist the specialists by automating the detection of suspected nodules, and hence improve the efficiency and quality of diagnosis. For instance, this type of system uses Machine Learning techniques to create a correlation between different features of a given disease, and eventually, learn to give accurate diagnoses. Moreover, the visual recognition process is made by Convolutional Neural Networks, which are considered state-of-the-art in the object and pattern recognition area. These models present a considerable precision gain, in exchange for their great computational costs. However, the recent improvements in both software and hardware areas play an essential role in this field, allowing the systems to build more complex Artificial Neural Networks, composed of numerous specialized layers. In particular, these systems are named Deep architectures. In conclusion, they are mathematical models that can construct complex concepts by creating a composition of simpler ones. In this work, an U-Net based architecture was employed to detect lung nodules in CT Scans. The model's performance was evaluated against the Lung Nodule Analysis (LUNA16) Challenge results, ranking among the top 20 architectures in the challenge.

Keywords: Deep learning; Convolutional neural networks; Lung cancer; CT scans.

RESUMO

O câncer pulmonar é uma doença causada pelo incontrolável crescimento de tecido nos pulmões. Esse tipo de tumor maligno, representa o câncer com maior taxa de mortalidade ao redor do mundo. Afortunadamente, quando este tipo de câncer é detectado nas etapas iniciais, causa um grande incremento na taxa de sobrevivência dos pacientes. Primeiramente, graças a tecnologia de escaneamento de Tomografia Computadorizada (CT), é possível obter informação detalhada das partes internas do corpo, e essa informação pode ser utilizada no treinamento de sistemas para detectar e classificar nódulos malignos. Além disso, imagens extraídas de CTs normalmente tem uma grande dimensionalidade, assim demandando ao especialistas um grande esforço de tempo empregado durante processo análise. Ao mesmo tempo, estes profissionais precisam cuidar de diversos pacientes, com diferentes casos. Consequentemente, fatores como a fadiga podem fazer com que estes profissionais cometam erros. Sistemas de Diagnóstico Assistido por Computador se provam extremamente úteis nestes casos, visto que podem ajudar os especialistas através de uma automação na detecção dos nódulos suspeitos, e assim melhorar a eficiência e a qualidade do diagnóstico. Redes Neurais Convolucionais Profundas são consideradas o estado da arte na área de reconhecimento de objetos e padrões, e devido às melhoras apresentadas nas áreas de software e hardware, o desenvolvimento deste tipo de sistemas é algo possível. Arquiteturas profundas são modelos matemáticos com a habilidade de entender conceitos complexos através de uma composição de conceitos simples. Neste trabalho, foi desenvolvida uma arquitetura baseada em U-Net para detectar nódulos pulmonares em CT Scans. O desempenho do modelo foi avaliado em comparação aos resultados do Lung Nodule Analysis (LUNA16) Challenge, classificando-o entre as 20 melhores arquiteturas do desafio.

Palavras-chave: Aprendizado profundo, Redes neurais convolucionais, Câncer pulmonar, Tomografias computadorizadas.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
BVLC	Berkeley Vision and Learning Center
CADS	Computer-Aided Diagnosis System
CNN	Convolutional Neural Network
CSV	Comma Separated Values
CT	Computerized Tomography
CV	Computer Vision
DCNN	Deep Convolutional Neural Network
DICOM	Digital Imaging and Communications in Medicine
DNN	Deep Neural Network
DP	Deep Learning
DT	Decision Tree
FCL	Fully Connected Layer
FN	Feedforward Network
GUI	Graphical User Interface
HE	Histogram Equalization
ID	Identity
IoT	Internet of Things
KDSB	Kaggle Data Science Bowl
LIDC-IDRI	Lung Image Database Consortium and Image Database Resource Initiative
LR	Learning Rate
LUNA16	Lung Nodule Analysis 2016
MHD	Meta Image Meta Header
ML	Machine Learning

ORM	Object-Relational Mapping
ReLU	Rectified Linear Unit

CONTENTS

1 INTRODUCTION

Nowadays, Deep Learning (DL) is a Computer Science field that represents a technological tendency, being a subject of varied research due to the impressive results, among diverse applications. It can be described as a subset of Machine Learning (ML) techniques, which uses neural networks with numerous layers to learn complicated concepts, by building them out of simpler ones (GOODFELLOW; BENGIO; COURVILLE, 2016). Artificial Intelligence (AI) has completely changed the way mankind looks at computers after it tackled and solved many problems that are considered really difficult for humans, once they were formally described. Nevertheless, the real challenge to AI is to deal with tasks that are considered easy for humans, but considerably hard to describe in a formal way. In other words, problems that we tend to solve intuitively, such as recognizing faces, handwritten digits, spoken words, etc.

This kind of problem suggests that systems need to acquire their knowledge, learning directly from raw data. The field that studies this process is known as Machine Learning, and it allows computers to handle issues using knowledge of the real world, making predictions and decisions based on their experience. This was the first way to avoid the AI's drawback of dealing with more informal situations. Nevertheless, Machine Learning has to face another kind of obstacle, once the performance of the model will always be relative to the quality of the information given. For example, a simple ML algorithm named logistic regression can be used to determine whether to recommend cesarean delivery, since the important characteristics of the patient were given. Thus, in this case, the knowledge of a doctor is still necessary, to work as an interface between the system and the information. Somehow, the information has to be classified as relevant, or not (MOR-YOSEF et al., 1990). These relevant characteristics are called features, and the system needs to learn how each property is correlated with various outcomes.

However, the definition of which features should be considered important is a complex problem, and it has a massive impact on performance. Unfortunately, the task of describing exactly what a feature looks like in terms of pixel values is far away from easy. There are necessary many techniques to describe the world around in a way computers can understand. One possible solution to this downside is not only to use Machine Learning for creating a mapping between representation and output but also to understand how the representation works (GOODFELLOW; BENGIO; COURVILLE, 2016). This approach is named representation learning, and it normally results in better

performance than hand-designed representations, rapidly adapting to several tasks without much human intervention. Furthermore, extracting and understanding these features is a process that requires a high processing level. Fortunately, Deep Learning has the potential to overcome this central drawback by describing more complex representations in terms of simpler ones.

Artificial Neural Networks are the quintessential example of Deep Learning models. They can be defined as mathematical functions, mapping inputs to output values. In other words, these mappings work as approximations of real functions. Furthermore, when DL techniques are applied to neural networks, they result in a Deep Neural Network (DNN). This model is built with numerous layers, responsible for performing different tasks. In general, in the first layers more basic algorithms are applied, mostly resulting from Computer Vision (CV) techniques, then the results are fed into the next layers. Eventually, the system can construct complex algorithms, such as object detectors, based on the basic concepts learned in the previous layers. In the field of image recognition, the state-of-the-art is represented by a specialized type of Neural Network, named Convolutional Neural Network (CNN). They are based on the convolution's operation, which uses a sliding window along the input image to generate an output. When this concept is expanded to diverse sliding windows in parallel, with different sizes and values, the model is named Deep Convolutional Neural Network (DCNN).

Currently, DCNNs have virous applications among the industry tasks, once the recent improvements, in both hardware and software technologies, allowed the deep systems to overtake several obstacles caused by their elevated computational costs. For instance, since Deep Learning has presented a huge performance gain compared to common Machine Learning models, it has been a fast and evolving field, with numerous implications, especially in medical imaging. Furthermore, it has been widely used to build Computer-Aided Diagnosis Systems (CADS). Unfortunately, the amount of data frequently represents a limitation to these systems, especially in the medical field, due to patient confidentiality concerns.

Considering all the facts, many DCNNs architectures have been proposed to solve different image patterns and object recognition problems properly. However, it is essential to have a good dataset available and a detailed analysis to find the best approach to construct the learning model.

1.1 Research Problems

This section presents the research problems which motivated this work. Essentially, three main research questions were defined. They are listed following.

1. Can Convolutional Neural Networks improve the lung cancer diagnosis process?
2. Is it possible to reduce the network complexity without reducing the accuracy?
3. Which techniques are used to improve the Convolutional Neural Networks' accuracy in the lung nodule detection process?

1.2 Motivation

A Computerized Tomography scan is a medical imaging technique used in radiology, to obtain detailed body internal images. Nowadays, to detect lung cancer nodules, CT scans need to be interpreted by some specialists, such as radiologists or physicians in general.

Unfortunately, this kind of professional often has to look through large volumes of these images. This is an important issue since humans are prone to deal with fatigue and stress when subjected to excessively great amounts of work. Consequently, it can lead them to make mistakes.

In addition, there is a needing of automating this task. Automation can not only improve the accuracy of diagnosis, but it can give a better use of the time and effort given by the health professionals, letting them available to work in cases that require special treatment.

1.3 Objectives

The main objective of this project is to implement a software tool able to automate the whole process of recognition and classification of lung nodules in CT scans, using a CNN, trained using DL and CV techniques.

1.3.1 Specific Objectives

Intending to propose more detailing on the objectives of this work, the following specific objectives are proposed:

1. Learning the theory of Deep Learning.
2. Performing a study about Convolutional Neural Networks.
3. Exploring the different options of available datasets.
4. Implementing a model to train the CNN.
5. Constructing graphs to analyze the model's performance.
6. Training the CNN to detect and classify lung nodules.
7. Validating the trained model.
8. Developing a Graphical User Interface to facilitate the system's operation.
9. Testing the software.
10. Analyzing the obtained results.

1.4 Methodology

This chapter presents the methodology utilized for the execution of this work and the required tasks. It contains a description of the activities, as well as the criteria used for the selection of related works.

1.4.1 Technical Methods

Initially, was realized bibliographical research concerning Deep Learning in a general scope, besides its technical components which allow the learning process. Moreover, an investigation of Neural Networks and object detection processes in images was executed. Consequently, relevant materials about Deep Learning, Neural Networks, and the object recognition task were obtained, which will be extremely important to the project performance.

Next, a deeper study about Convolutional Neural Networks was performed, regarding the specialized operations and layers present in this specialized type of Neural Network, and also their importance to the architecture in general.

In sequence, an inquiry about the different available datasets was done, and Lung Image Database Consortium and Image Database Resource Initiative[1] (LIDC-IDRI) was chosen to be the base dataset due to the amount of validated information. At the start, it was necessary to perform experimental research aiming to understand the pylidc[2] library, which is responsible for making an object-relation mapping with the data. In this stage, the functionality of the methods was identified, together with their parameters, based on the software documentation.

Eventually, it was possible to isolate the relevant scans' parts, using this library. This information was generated and saved in a Comma Separated Values (CSV) file, containing the identification of each patient and labeled malignancy of their present nodules.

1.4.2 Selection of Related Works

The related works were selected and compared with this using a systematic literature review. The research was carried out using mainly two different databases, the IEEE Xplore[3] and the Springer[4]. At first, the research problems were defined. The keywords were extracted from them, and then the following first search string was created:

(Convolutional Neural Network) AND (Deep Learning OR Machine Learning) AND (Lung Cancer) AND (Detection OR Classification) AND (CT scans)

In the IEEE Xplore, the search returned a total of 59 results. Then, some works were selected, and the next step was performed. The abstract and conclusion were read, then their keywords were extracted, resulting in a new search string, shown in sequence.

(Lung Nodule Detection) AND (Deep Learning) AND (CT scans)

Were applied filters to reduce and select the more appropriate works inside the list, resulting in 44 works. The filters used in this process are listed following.

- Period limit from 2017 to 2022;
- Works in English;
- Computerised Tomography;
- Medical Image Processing;

[1] wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI
[2] pylidc.github.io
[3] ieeexplore.ieee.org/
[4] link.springer.com/

- Learning (Artificial Intelligence).

The same search strings were applied to the Springer database. The filters were minimally modified to adapt to the platform since the same options are not available. Nevertheless, the same ideal was followed.

Among the research results, it was searched for recent articles, with an adequate number of references. In addition, were chosen articles that presented different changes in the architectures, looking to achieve different improvements, since they had a sufficient description of the process.

1.5 Outline

The text is organized as follows: the **Chapter 1** introduces the project. The **Chapter 2** presents the main concepts related to DL and CNN. The **Chapter 3** introduces a discussion, presenting some related works. The **Chapter 4** describes all the process related to the project development. Finally, the **Chapter 5** addresses the results obtained in this work and the final considerations.

2 THEORETICAL REFERENCE

This chapter acts as a theoretical basis for the concepts and algorithms utilized and discussed in this thesis. Firstly, it sheds some light on the state-of-the-art architecture used for object detection in images, the Convolutional Neural Network. It starts presenting the idea behind the artificial neuron, named Perceptron. Next, the main operations performed along the CNN are discussed, which are responsible for classifying the hidden layers according to their function. In this part, the theory behind convolutions and filters, pooling layers, and padding are discussed. Finally, the fully connected layers used in the classification stage are shown, as well as the Dropout technique, often used to prevent overfitting the model. The second chapter focuses on the Deep Learning area, introducing some elements and algorithms that are essential to the learning theory, such as the Cost Function, Gradient Descent, and Backpropagation. Soon after, it describes the main tasks behind it, known as the regression and classification problems. The last part of Deep Learning's chapter starts discussing Deep Feedforward Networks and eventually the main techniques constructed inside this architecture's type, which are essentially based on CV's theory.

2.1 Convolutional Neural Networks

As seen in the Deep Learning Chapter, Artificial Neural Networks are architectures inspired by the way the human brain learns. They are based on multiple connected artificial neurons, creating a processing network. Furthermore, due to the fast growth of processing capacity and the improvement of Machine Learning techniques, diverse Artificial Neural Network models with several specializations have started to gain popularity. Convolutional Neural Networks is one of these specialized ANNs, focused on processing data that has a known grid-like topology (GOODFELLOW; BENGIO; COURVILLE, 2016). It has brought exceptional advances in practical applications such as taking samples in time series and detecting different object classes in images, as a result of the use of a mathematical operation called convolution, in place of general matrix multiplication.

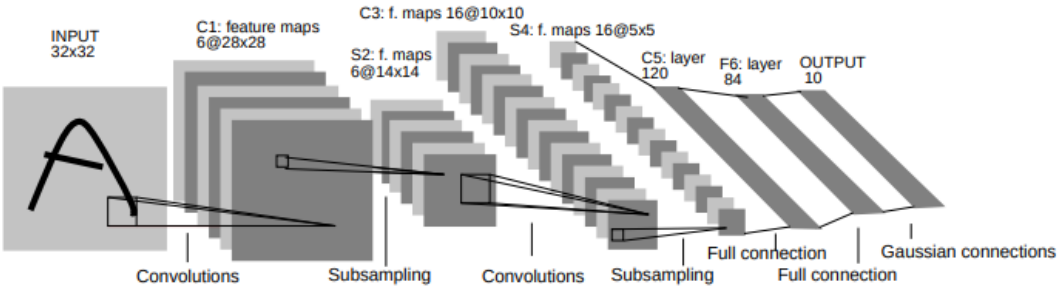
The object detection algorithm works as a complexity pyramid. The basis of this pyramid is composed of different pre-processing techniques, such as Smoothing, Blurring, Thresholding, and Segmentation, as seen in the previous subsections. In the

middle levels, there are many filters of increasing complexity, and consequently, there are really complicated filters compounded by many functions at the top, such as object detectors.

Another parameter is added to this kind of neural network, which is called stride. It is responsible for dictating the amount of movement over the image, in pixels. This property brings up similarities with the way our visual cortex works. According to the work of Hubel e Wiesel (1962), neurons in the visual cortex have a small local receptive field, which in other words means that they only process a small subsection of the entire image that a person is viewing. Eventually, these local subsections can be overlapped, creating a larger image. This guarantees that the model will only be able to find patterns between pixels in the same neighborhood, and that is why it is considered an essential element in object recognition.

However, this was the idea responsible for adapting an Artificial Neural Network to become the CNN architecture. LeNet-5 is considered the first CNN, and it was famously implemented in the work of LeCun et al. (1998), employed to classify handwritten numbers using the MNIST data set. This architecture can be seen in figure 1. There is a wide variety of architectures that were implemented for different kinds of object recognition in images. Nonetheless, almost all of them have elements in common. This section will be shading some lights in the components that are usually seen in these architectures.

Figure 1 – LeNet-5 CNN Architecture.



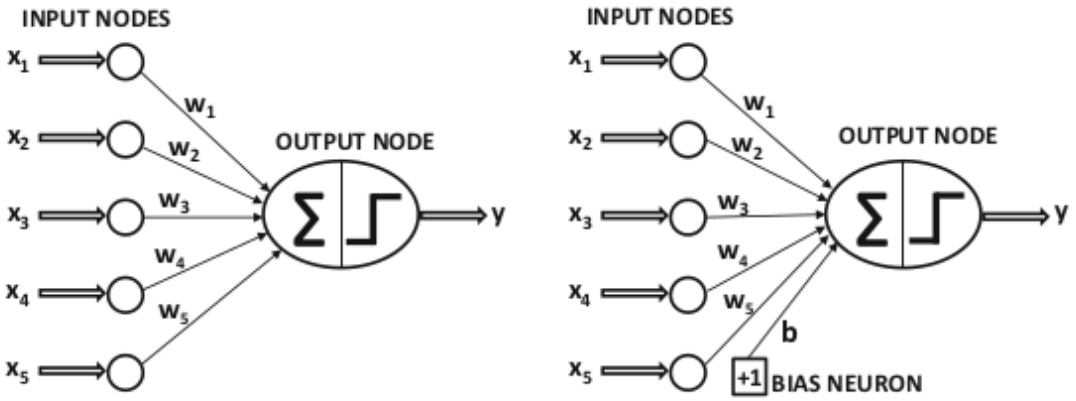
Source: LeCun *et al.* (1998)

2.1.1 The Perceptron

The neural network's groundwork is referred to as the Perceptron. For instance, it can be defined as the simplest Neural Network existent, containing a single input layer

and an output node. The Perceptron basic architecture is described in figure 2. Every Supervised Learning has the training instance of the form $(\bar{X}, y)$, where $\bar{X}$ contains the set of variable features and y contains the output value of the binary class variable. The input features set is represented by $\bar{X} = [x_1, ...x_d]$, and the output by $y \in [-1, +1]$ (AGGARWAL et al., 2018). The input layer has d nodes, responsible for introducing to the neural network d features $\bar{X} = [x_1, ...x_d]$ with weight $\bar{W} = [w_1, ...w_d]$ and mapping them to an output node. It is important to highlight that the input layer is not controlling any processing. This is performed by the output nodes. Following this, it is necessary to use an activation function to produce an output whenever specific sets of inputs arrive.

Figure 2 – The basic architecture of the perceptron.



Source: Aggarwal *et al.* (2018)

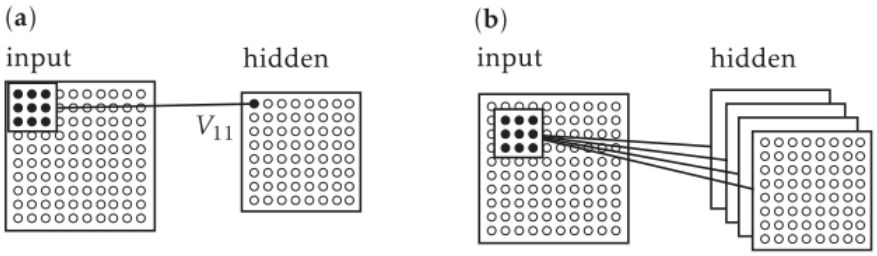
The equation 1 uses the sign function to map inputs to either $+1$ or -1, creating a binary classification. A further point is that the circumflex at the top of variable y is used to indicate that it is an output prediction rather than an already known value. Moreover, there is a part of the prediction which is invariant, referred to as the bias. This is a way to avoid invalid values produced between specific weight and feature. Note that regardless of the weight's value, if the feature is equal to zero, the output will be zero. This would drive the model to overfit. Fortunately, it can be easily prevented by using a *bias* term.

$$\bar{y} = sign(\sum_{j=1}^{d} w_j x_j + b) \tag{1}$$

2.1.2 The Operations

As introduced earlier, the CNN architecture is based on the convolution operation. In other words, different filters are applied to the input image. Convolution is an essential mathematical operation for image detection. It can be defined as a pattern-finding technique that is applied to detect diverse features. It is composed of three essential elements, an input image, kernel, and feature map. The kernels, also known as filters, are small 2-dimensional arrays that will be applied to the input image, generating a feature map. Therefore, layers constituted of different feature maps and kernels are referred to as convolution layers (MEHLIG, 2019). These filters could be customized to detect different features, such as corners and edges, for instance. For example, in the face recognition task, certainly there are numerous filters working inside the CNN's hidden layers. One perfectly possible approach is designing one of these kernels for detecting gender traces, meanwhile, another was responsible for detecting curvy or straight lines, and eventually be capable of detecting the input face. Note that this is just a hypothetical heuristic since the filters are implicitly defined by the deep network, as a result of what feature it considered more relevant. Figure 3 (a) shows the process of generating a feature map after applying a kernel to an input image. The same proceeding can be extended to different kernels, creating multiple feature maps, as shown in figure 3 (b).

Figure 3 – The convolution operation.



Source: Mehlig (2019)

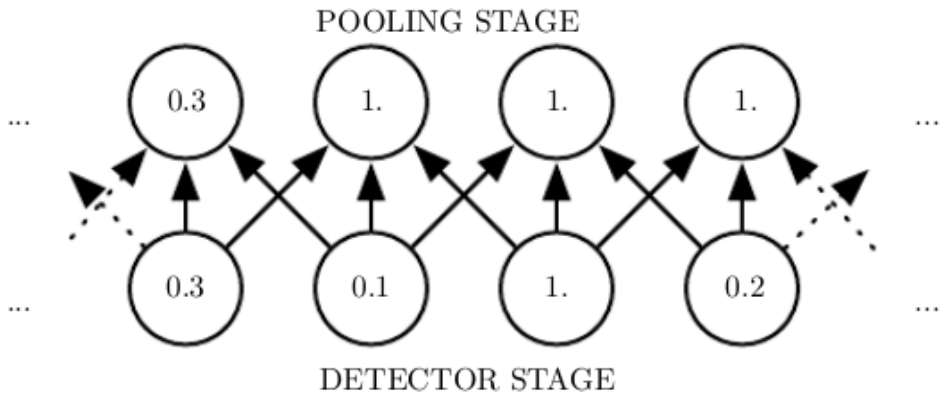
The convolution operation is shown in equation 2, being S the generated feature map, K the kernel, and I the input image. Furthermore, convolutions are often used over more than one axis at a time, and as a result, some variables become essential to represent the sum of a component's dimensions (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n). \tag{2}$$

Furthermore, there is another important operation widely used in Convolutional Neural Networks, called pooling. According to Goodfellow, Bengio e Courville (2016), a typical layer inside a CNN consists of three stages. The first stage is responsible for several convolutions in parallel to produce a set of linear activations. Following, each of these linear activations runs over a detector stage, where they are submitted through a nonlinear activation function. The third and last stage of this process is called the pooling function, where the output of the layer is adapted to the further layer.

In the pooling layer, one special technique called sub-sampling is used for reducing the input image parameters, looking for an improvement in the memory requirement, and optimization in the computer load procedure. Essentially, this function replaces the output at a certain location with a summary statistic of the nearby outputs. The max-pooling has a little similarity with the convolution operation, needing a kernel to apply through the input. Eventually, small kernels are used in this process, just like the convolution's operation. The difference is that unless of making a convolution, the max-pooling layer looks for the highest existent value according to the place where the kernel is placed, to be the output of this instance, as shown in figure 4. As an example, even in a kernel with a $2x2$ pixel size, the lost information will be representing 75% of the total data, since just the highest value will be passed to the next layer. Fortunately, this is not an important issue, as, in the first instance, the goal is not to find where the feature is, instead of that, the neuron just needs to be sure that it is there.

Figure 4 – Detector and Pooling Stage.

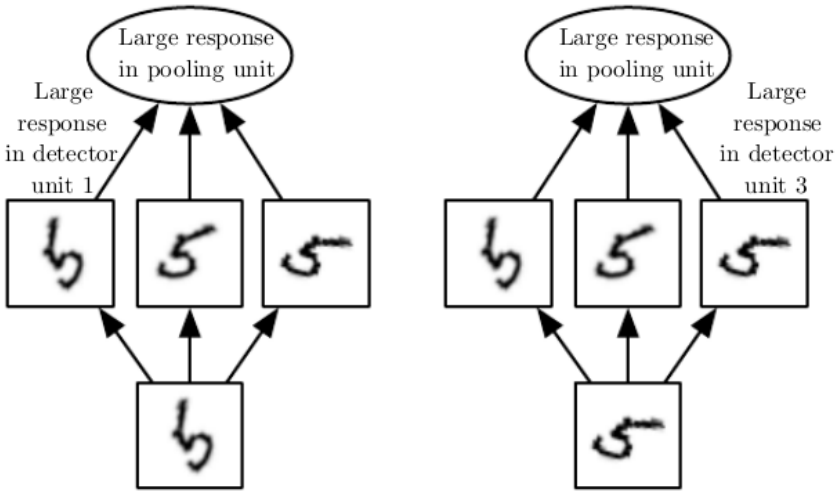


Source: Goodfellow, Bengio e Courville (2016)

Despite that, pooling helps the representation to become approximately invariant to small translations of the image, which can be an extremely useful property (GOODFELLOW; BENGIO; COURVILLE, 2016). Determining whether an image contains a face, for example, firstly it is necessary to be sure that there is an eye on

the left side and another on the right side, currently the exact location is not essential. However, this is not the essential approach to detecting objects, although the idea can be used to explain the way that pooling layers treat data. Moreover, a pooling unit can learn to be invariant to transformations of the same input, by pooling over multiple features that are learned with separate parameters. A set of filters able to be invariant to rotation of handwritten numbers is shown in figure 5.

Figure 5 – Learned invariance.



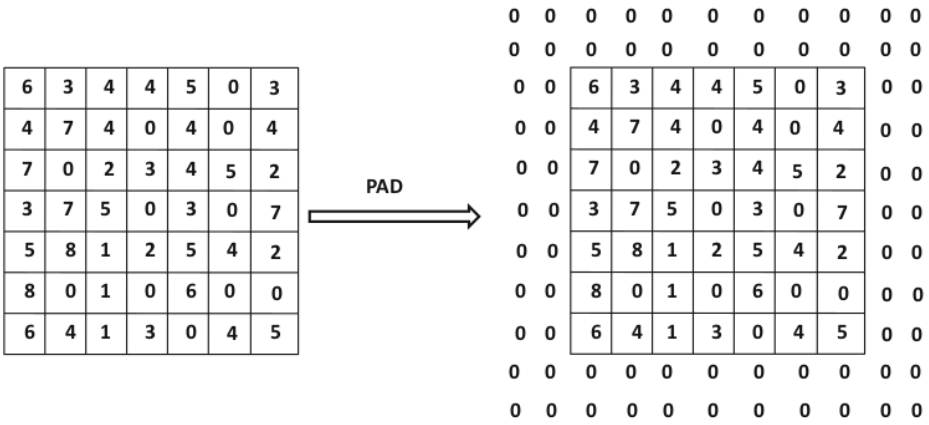
Source: Goodfellow, Bengio e Courville (2016)

One important issue caused for the convolution operation is the next layer's size reduction. This kind of reduction in size is not ideal in general, because this causes a reduction in data along the image's borders, or in the feature map. Fortunately, this problem can be resolved by using a technique named padding, which consists in adding pixels all around the feature map's borders (AGGARWAL et al., 2018). In that way, the original size of the image could be recovered after the convolution operation is performed. A further point is that the padded portions do not contribute to the final dot product, since they are initialized as zero, as shown in figure 6. Essentially, there are three different types of padding used on different occasions, the difference between them is briefly shown in table 1. The F_l and F_w are the kernel's length and width, respectively. The half-padding increases the output image by $(F_l - 1)$, and $(F_w - 1)$ correspondingly, resulting in the input image's original size. It is properly designed to maintain the spatial footprint exactly.

Moreover, when there is no padding added in the process, it is called valid padding. It ensures that the contributions of the central pixels will be higher than the pixel's information at the layer's borders, generally resulting in a precarious performance, in an experimental view. Meanwhile, another useful form of padding is the full-padding. It

increases the spatial footprint in the same dimension that the valid approach reduces. In other words, both dimensions of the input increase by $2(F_l - 1)$ and $2(F_w - 1)$, respectively. This is an essential step for diverse convolutional autoencoders, due to the increased spatial footprint.

Figure 6 – Example of padding.



Source: Aggarwal *et al.* (2018)

Table 1 – Padding Styles

Mode	*Output Size*	*Usage*
Half	$I_{\text{size}} + F_{\text{size}} - 1$	Maintain the size
Valid	I_{size}	Borders Under-represented
Full	$I_{\text{size}} + 2(F_{\text{size}} - 1)$	Convolutional autoencoders

Source: Adapted from Aggarwal *et al.* (2018)

2.1.3 Fully Connected Layers

After being submitted to several convolutional and pooling layers, each feature in the final spatial layer ends up connecting with diverse different neurons in the first Fully Connected Layer (FCL). The process used to convert all the resultant 2-Dimensional arrays from the previous operations into a single long continuous linear vector, which will be treated as an input to the next ANN, is called flattening. Aggarwal et al. (2018) claims that this layer acts exactly in the same way as a traditional FN, and in most cases, more than one FCL is employed to increase the power of the computation towards the end.

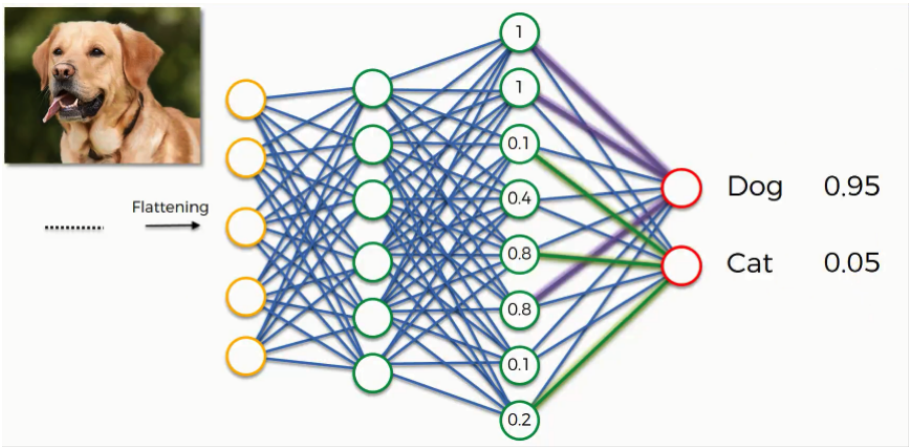
An important issue is that FCLs are excessively expensive, in terms of

computationally. Let us imagine that two of them have a sum of 8192 hidden units, a total of 4096 in each one. The number of connections between these hidden units exceeds 16 million, resulting in numerous parameters.

The CNN's output is designed to be specific to an application. In the classification problem, the output layer is fully connected to every neuron in the penultimate layer, with its weight associated. In this case, this layer might use different activation functions, such as *Logistic*, *Softmax*, *Linear Activation*, *Rectified Linear Unit*, all depending on what is the nature of the application.

After all this procedure, the ANN final layer will be able to perform the classification task, calculating the input's probability to belong to different classes. Figure 7 shows a fully connected network applied to classify an input as dog or cat. Note that the flattening function is arriving as the input of this network.

Figure 7 – Fully connected network.



Source: SuperDataScience Team (2018)

2.1.4 Overfitting and Regularization

Deep Neural Networks are likely to deal with a problem during the training process. It is natural to divide the dataset information into two different parts, resulting in a split between training and testing datasets. Aware of this, usually the networks present different performances in both datasets. The first division is applied in the training stage, meanwhile, the second one is kept only for testing the performance of the model. This means that this data is unknown to the network.

Overfitting is a common issue that appears when the model is performing overly well with already known information, which in theory, is something desirable, due to the

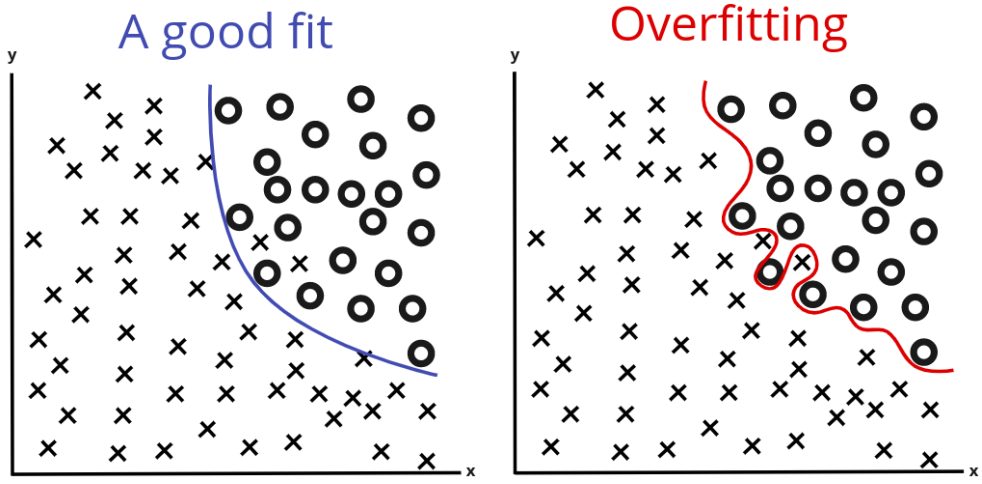
ML's main goal. Although, when the model is subjected to unknown data, the general network performance notably decreases.

This is a central drawback in ML since the models must perform well not just on the training data, but also in information that they do not know. Otherwise, the network even being excellent at classifying known information, would not be able to make unsupervised predictions. Many strategies are employed, looking to reduce the test error, which sometimes means an increased training error. These strategies are known as regularization. Figure 8 shows how regularization techniques can avoid overfitting.

The dropout is a common technique that is deployed with Convolutional Neural Networks to evade this issue. It can be thought of as a technique that helps to prevent overfitting. For that reason, randomly, units are dropped along with their connections (AGGARWAL et al., 2018). The dropout is useful to avoid units from excessively co-adapt the training information. Besides, this provides a computationally inexpensive and powerful method of regularization (GOODFELLOW; BENGIO; COURVILLE, 2016). If they are not present, the first training samples will influence the learning with unnecessary priorities. This operation is utilized in the dropout layer, which can be defined as a mask that nullifies the contribution of some specific neurons, without modifying all others. It can be applied both to the input vector and to the hidden neurons.

Therefore, the possible future features would have their learning affected if this layer were missing, and consequently reduce the entire network's performance. Looking in another way, the network weights' quantity will be increased more than normal, cause of this technique's use. Furthermore, this works especially well in practice, usually replacing another regularization technique's usage.

Figure 8 – Regularization's effects



Source: Author (2022)

2.2 Deep Learning

Deep learning is a subset of Machine Learning, which is a field dedicated to the development of systems that can learn with their experience. Furthermore, it plays an important role in industry solving practical tasks in a variety of fields (TRASK, 2019). The work of Goodfellow, Bengio e Courville (2016) affirms that the fact that Deep Networks are formed of numerous layers provides them a powerful framework for supervised learning, creating a system with increasing complexity. It makes it possible to reach more complex functions through simpler ones.

To make the learning process possible, it is necessary a considerable amount of validated data. The model precision is directly related to the number of information available, and hence, the technological resources play an essential role in this field, making this kind of system become really attractive. Meanwhile, the rising technologies with the Internet of Things (IoT) focus, and in addition, the plenty of sensor's quantity being used around the world, have tremendously increased data creation.

Due to the technology increasing, the DL gets a huge advantage in the process of understanding information from the real world. However, to get these benefits, the system must extract the right features, detect information patterns, and eventually be able to create a relation between inputs and outputs. After all, if the model achieves a considerable knowledge level in relation to the features, it would be able to predict accurate outputs, even to inputs that it has never seen before.

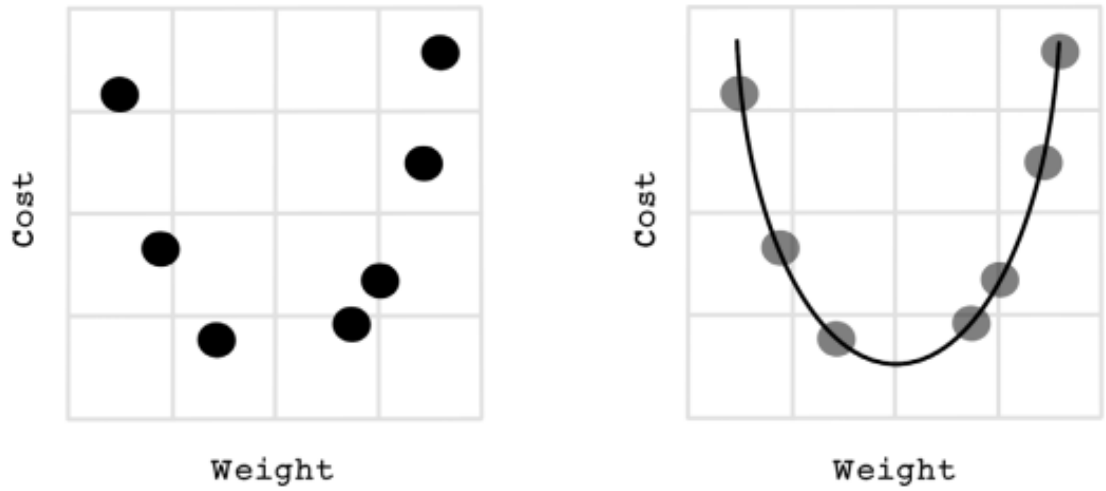
2.2.1 The Learning Process

Since the main DL models' goal is to create approximations of certain functions, the systems must have information as inputs. The information would be composed of numerous independent variables. Each variable is multiplied by a specific weight, concerning to find the correct relevance of the input to the output, these weights are adjusted iteratively. In other words, the model tries to reach values that best represent how relevant each variable is to the output. Three different algorithms are essential to make this procedure of learning possible.

According to Goodfellow, Bengio e Courville (2016), the choice of the Cost Function is an essential aspect when designing a deep neural network. In most cases, a distribution $p(y|x;\theta)$ is defined by the model, and then it is only needed to use the

principle of maximum likelihood. In other words, the definition of the Cost Function is the cross-entropy between the training data and the predictions of the model. Besides, regularization techniques are usually applied to help in this process. The Cost function is nothing but a way to measure the model, evaluating how wrong it is in finding the relations between features and outputs. As a result, it represents how badly the system is predicting. The model might have to deal with different variables that it does not know. These changeable parameters can influence how the model behaves. In other words, the Cost Function is employed for fitting and optimizing the model. Furthermore, it can be defined as each layer error's sum. This value can be found by calculating the error in each layer and then summing the individual errors. Finally, the representation of a Cost Function's example is shown in figure 9.

Figure 9 – Cost Function.



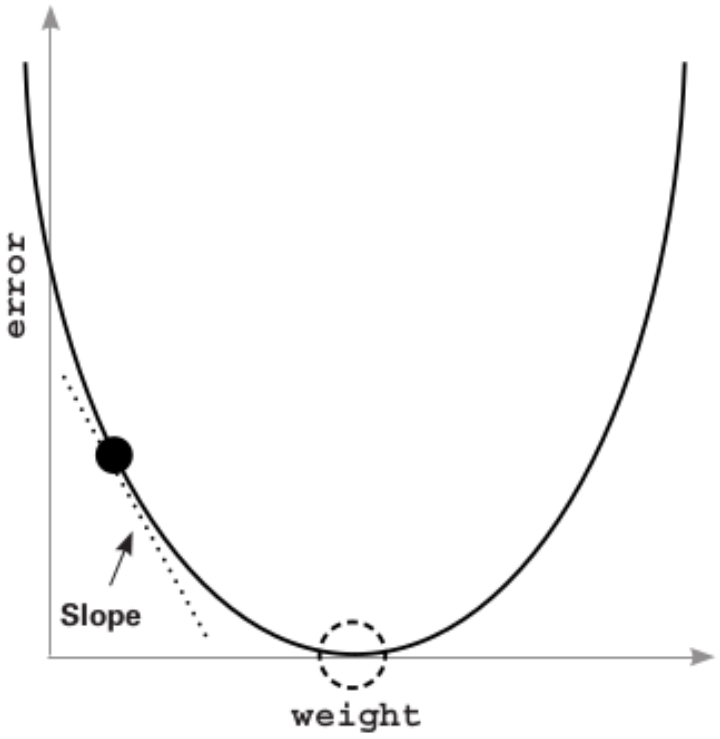
Source: Hurbans (2020)

Goodfellow, Bengio e Courville (2016) claim that the Gradient Descent is an iterative algorithm that aims for the model's better weights, looking to minimize the Cost Function. Mathematically, it looks for a point in the loss function where the derivative will result in a zero. If we expand this concept to multiple variables, the Gradient Descent is nothing but a matrix containing the Cost Function's partial derivatives. The derivative is a crucial point for this algorithm, for the reason that it measures the sensitivity to change for that function. For example, in a physics view, the acceleration of an object is defined by the speed derivative in relation to time. In conclusion, derivatives can find the slope at a specific point in a function, and Gradient Descent uses that knowledge to determine the direction of the next weights (HURBANS, 2020). If the goal is to maximize a function,

it is possible by following the Gradient Descent. Anyway, in the case of a loss function, the goal will always be to minimize, it means that it is necessary to follow the opposite way. Each iteration that the algorithm runs is called an epoch. As an example, figure 10 shows how a specific point's derivative in the cost function determines its slope, which can be used to move the network weights in the direction of rather minimal or maximal points. In this case, the black point represents the current cost value, and the slope shown is determined by its derivative. The dotted circle represents the error function's global minimum. Note that the slope is directed to the second circle.

There is another essential task that must be done to make the learning process possible, besides the Cost Function's calculation and the Gradient Descent's analysis. Even knowing the direction that the Cost Function should follow, the weights of the neurons still need to be updated. This is precisely Backpropagation's main goal. It is responsible for quickly adjusting the optimal parameters and weights, across the entire network. It works by calculating the error at the output and then distributing it back through the network layers. The process happens from the output to the input neurons, hence the origin of the backpropagation name. In summary, the three components work together to make the learning process possible for the system.

Figure 10 – Gradient Descent analysis.



Source: Trask (2019)

2.2.2 Deep Learning Tasks

As a subset of Machine Learning, Deep Learning is nothing but an approximation problem. In contrast, looking at it as a geometry problem, DL tasks can be mainly divided into two classes of problem, regression, and classification problem. In other words, this approximation, in fact, is the process of fitting a straight line in a plane, finding the best way to rather describe a given function, or using it to divide the samples into two classes. Both types of issues will be introduced in this section.

Regression is a statistical model which can be used to measure the probability to happen of a specific phenomenon. It uses the relation between different features to create a line that better describes the event's behavior. According to Raschka (2015), the goal is to find a relation between a given number of input variables and a continuous output variable. Consequently, it is possible to use equation 3 to minimize the average distance to the sample points.

$$y = mx + b \tag{3}$$

Russell e Norvig (2021) affirms that this procedure of adjusting the line is made by an iteration algorithm, which will keep making changes in the values of m and b, moving to the minimal point of the cost function. Let this line's error function be defined by the Mean Squared Error, represented in equation 4.

$$MSE = \sum_{i=1}^{n} (reg_i - y_i)^2 \tag{4}$$

Where $y_i = f(x_i)$ and reg_i is the predicted value. It is important to highlight that equation 4 is a quadratic function to avoid a possible cancellation between higher and lower points in the regression line. The learning algorithm will try to decrease the error in each iteration, which means that the line will be fitting better little by little. The direction of the steps is guided by derivatives. The next value of m is defined by equation 5.

$$m = m - \alpha \frac{\partial MSE}{\partial m} \tag{5}$$

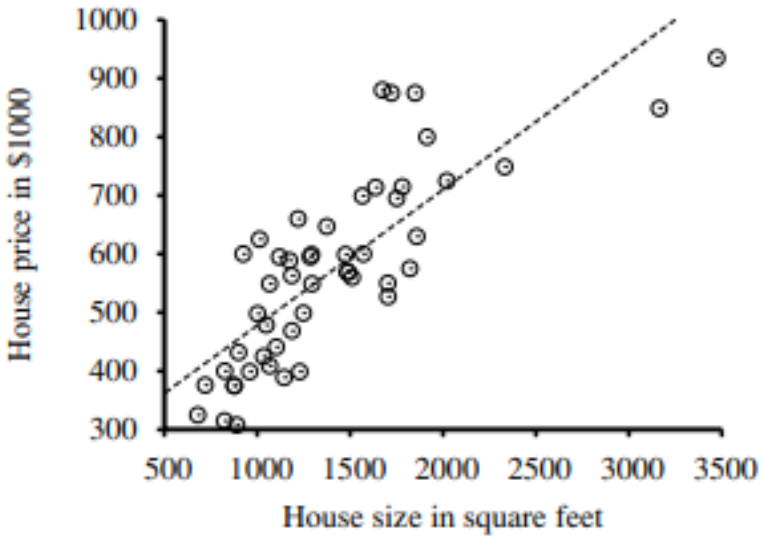
Where α is propriety called Learning Rate (LR), which is responsible for determining the size of these steps. The MSE equation has a relation of dependence with reg_i, which at the same time has a dependency on m. To determinate this derivative, it is necessary to

use the Chain Rule as shown in equation 6.

$$\alpha \frac{\partial MSE}{\partial m} = \frac{\partial MSE}{\partial (reg_i - y_i)^2} \cdot \frac{\partial (reg_i - y_i)^2}{\partial m} \tag{6}$$

The same idea of (5) can be expanded to find the better b value, using the difference between the current variable and the derivative of (4) related to b. The line will be tending to the minimal point of the cost function. As a result of numerous iterations, it will be fitted in the best way possible, describing the average behavior of the problem. Figure 11 shows a linear regression problem, with a straight line representing the average behavior of the price of houses as a function of their floor space.

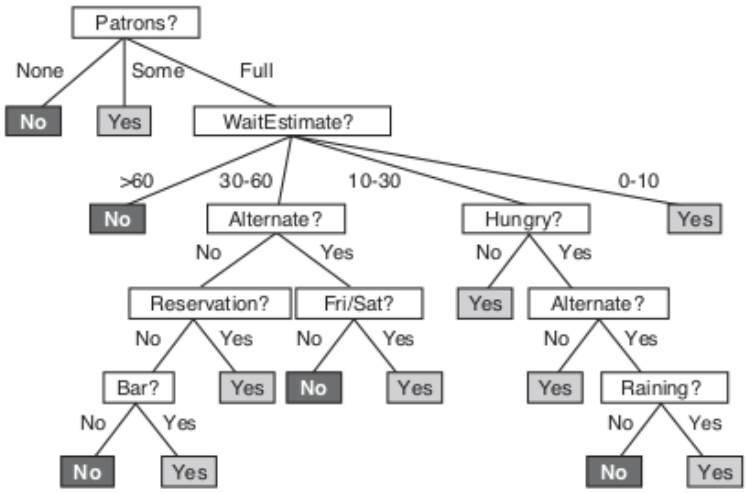
Figure 11 – Data points of price versus floor space of houses for sale in Berkeley.



Source: Russell e Norvig (2021)

Classification is another Machine Learning problem example. The goal is to classify the inputs into different classes or labels. The classes can be divided into binary sides, like zero or one, cat or dog, dark or white, nodule or non-nodule, etc. Charbuty e Abdulazeez (2021) affirms that one of the most powerful methods commonly used to classify objects is the Decision Tree (DT). They are composed of Nodes and Branches, each node being a feature in a category, and the subsets of these nodes represents the different values that could be taken by the node. As an example, figure 12 shows the constitution of a DT employed to decide weather to wait in a restaurant.

Figure 12 – Example decision tree



Source: Russell e Norvig (2021)

Entropy and Information Gain are essential metrics used to measure the purity of the data and the priority of each node in the decision tree, respectively. The entropy is defined by equation 7. Being S the set of classification and P_i the probability of each variable of the subset. The value of entropy will be at its maximum point when the probability of each class is balanced. In other words, when there is a class that is predominant, the entropy will tend to extremes.

$$Entropy(S) = \sum_{i=1}^{c} P_i log 2^{P_i} \tag{7}$$

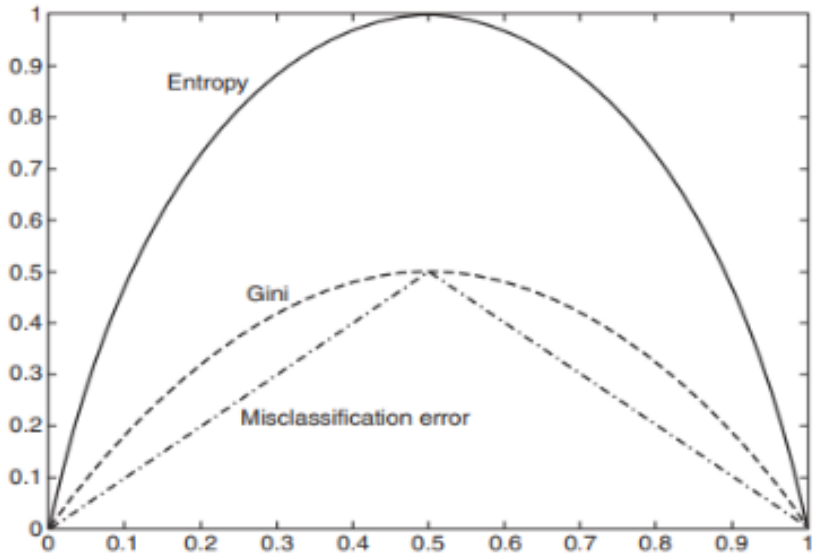
The Entropy function is shown in figure 13. Another important metric used for segmentation is information gain. It works opposite to the entropy, the higher metric represents the better value. Charbuty e Abdulazeez (2021) claims that the information gained intuitively informs how much knowledge of a random variable's value. The information gain is given by equation 8.

$$Gain(S,A) = \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{8}$$

Being S_v a subset of the set S equal to the value of the attribute v, and A is an attribute whose range is $V(A)$. In conclusion, the DT is simple to comprehend and has a good capability of classification in both categorical and numerical outcomes. In contrast, it has an increasing complexity and a bad representation of the mechanism can lead the system

to take wrong decisions.

Figure 13 – The Entropy function



Source: Charbuty e Abdulazeez (2021)

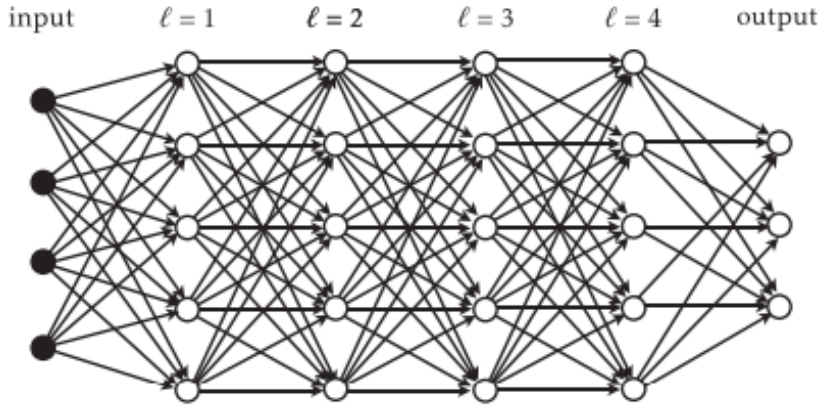
2.2.3 Deep Feedforward Networks

Feedforward Networks (FN) represent an important role in Machine Learning and their commercial applications. For example, most of the advance in the recent CV field is because of the good performances in object recognition achieved by Convolutional Neural Networks, which is a specialized type of FN. Feedforward Neural Networks are named networks because they are typically represented as a composition of different layers, that can also be seen as functions (GOODFELLOW; BENGIO; COURVILLE, 2016).

The first and last layers of a neural network are named input and output layers, respectively. The layers contained among these layers are called hidden layers, as the training data does not contain the correct answer for them. Finding those answers is one of the responsibilities of the learning algorithm. In addition to the layers, each layer is responsible for learning concepts and feeding them forward to the next layers. For example, in object recognition, the first layer could be responsible for detecting edges, the following layer for detecting borders, the next for recognizing the content, and eventually, the output layer would be able to classify them into different classes of objects. Of course, this described system is just hypothetical, since DNNs work as a black box, deciding on their own the features that they consider important to detect. In other words, the goal of

a Deep Feedforward Network is given x and y, being input and output correspondingly, to find some approximated function $f^*(x) = y$. It defines a mapping $y = f(x;\theta)$ and learns values of θ that result in the best approximation (GOODFELLOW; BENGIO; COURVILLE, 2016). A Feedforward Network's example is shown in figure 14.

Figure 14 – Fully Connected Feedforward Network.



Source: Mehlig (2019)

They are called networks because they were inspired by neuroscience, and feedforward because each layer has the previous layer's output as an input, and there are no feedback connections, the results are only passed forward. It is possible to represent the output layer as a chain of functions, as defined by equation 9.

$$f^n(x) = f^{n-1}(f^{n-2}(f^{n-3}...f^1(x)) \tag{9}$$

The overall length of this chain gives the model depth. Moreover, the model width is determined by the number of neurons in these hidden layers. The neurons in the same layer can be seen as different processing units that act in parallel, each one with its activation value. Neuroscience has presented the use of multiple neurons concerning to compute data, but it is essential to understand that the goal of neural networks is not to perfectly model the brain. They were created to be thought as approximation machines designed to achieve statistical generalization, loosely inspired by the way our brain works (GOODFELLOW; BENGIO; COURVILLE, 2016). Linear models are undoubtedly attractive to feedforward networks. They may be fit efficiently and also can be reliable. The problem is that this type of model cannot understand the relation between two or more input variables. That is why they are limited to only linear functions $f(x)$.

Although, these models can be really useful to understand nonlinear models. It is possible to expand their concept to represent nonlinear functions, describing x in terms

of the features. As a result, we have a new representation $\phi(x)$. It is important to highlight that choosing a good mapping ϕ is a difficult task, and it has a massive impact on the model's performance. Goodfellow, Bengio e Courville (2016) affirms that there are mainly three possible options for this:

1. Using a very generic ϕ. It can be a good option if it is available enough capacity to fit the training set, whereas the generalization to the test set may remain poor. Anyway, very generic feature mappings do cannot encode enough prior information, which turns impossible to solve advanced problems;
2. To manually engineer ϕ. Nevertheless, it is impossible to affirm that it is always possible. It could take a large amount of time and effort from different specialized practitioners. Besides, generally, it is necessary to relate strongly to different areas;
3. The approach of Deep Learning, that tries to learn ϕ. Being $y = f(x;\theta,w) = \phi(x;\theta)^T w$ a model in which θ represents the parameter used to learn ϕ from different classes of functions and w is a map that leads ϕ to the desired output. The representation is parametrized as $\phi(x;\theta)$ and an optimization algorithm is used to find the θ that corresponds to a good representation. It is important to know that this third approach can be complemented by the other two. Specialized humans can encode their knowledge to improve the generalization by designing multiple descriptions of $\phi(x)$ Also, it is possible to reach a highly generic level in this procedure, using a big family $\phi(x;\theta)$.

In summary, Feedforward Networks are essential to understand the concept of deterministic learning. In sequence, it will be introduced how the concept basis is built inside the early layers, using CV techniques.

2.2.4 Computer Vision in Deep Learning

It is pertinent to say that among the five senses of humans, the vision is the one that provides the most information. According to Davies (2017), the data rates for continuous viewing exceed 10 Mbps, but much of this information is compressed and processed by millions of neurons inside the visual cortex. Therefore, it is necessary to describe the world in a way that computers can understand, and this is precisely CV's primary goal. This section aims to get across some concepts that are essential to object detection and act inside Deep Learning's black box.

The objects need to be expressed in terms of their properties, such as shape, illumination, and color distribution. According to Szeliski (2010), often the complexity of this perceptual problem is underestimated. Initially, it was believed that the AI's logical and planning parts were more important and difficult, which is a misperception. Moreover, the deep neural network's hidden layers work as the visual cortex, and they are responsible for applying a great variation of filters, and just then, it will be able to recognize and classify objects in the output layer. A considerable volume of theories that are necessary to build up these simpler filters is straightly based on CV's techniques. One essential procedure that anticipates the training stage is the pre-processing step. Images are adjusted and optimized to facilitate the model's recognition in this process.

In the edge detection procedure, it is crucial to reduce the image's noise to help the algorithm to get rid of minimal details and focus on the general task since they likely consider excessive edges in high-resolution images. This detail reduction is made by applying some blurring or smoothing to the images. Figure 15 shows the difference in the number of detected edges between both normal and blurred images, respectively.

Figure 15 – Blurring's difference in edge detection.

Source: Author (2022)

Besides, Histogram Equalization is another common technique used in the pre-processing step. It works as an image's intensity adjuster, enhancing the contrast of the inputs. The work of (ASUNTHA; SRINIVASAN, 2020) claims that given a CT scan image represented as a matrix I, with dimension I_x by I_y and intensity range of pixels between 0 and 256, the normalized histogram bin N of the image is denoted by equation 10, where $\{0 \leq n \leq 255\}$.

$$I_N = \frac{\text{Number of pixels with available intensity n}}{\text{Total number of pixels}} \tag{10}$$

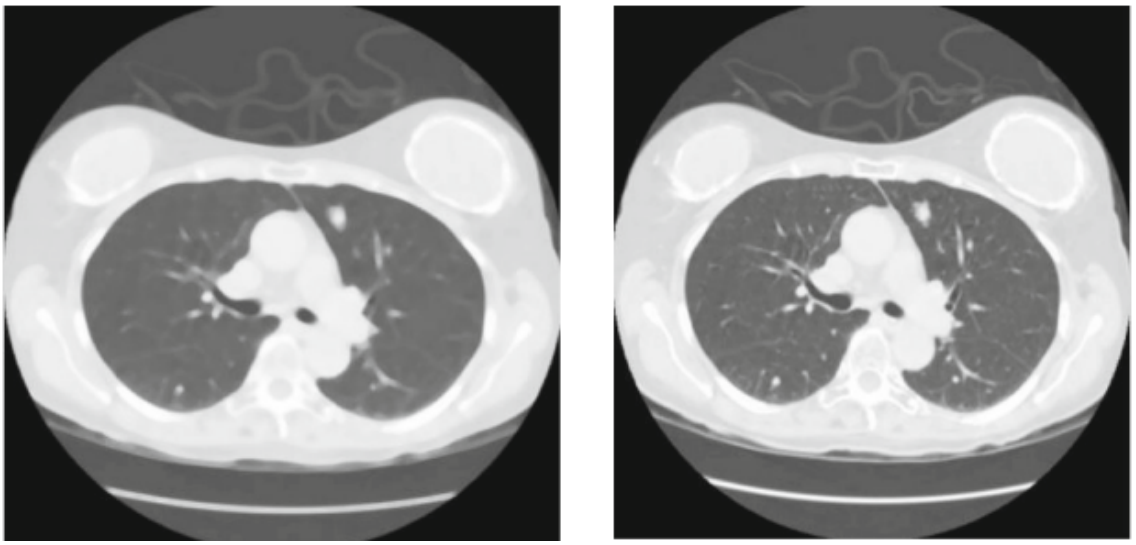
The contrast of the image is recovered by dividing the image into small kernels and transforming the intensity of their pixels to make a balance between the colors. The

result of this process is shown in figure 16.

Furthermore, Image Thresholding is another fundamental method in CV. It has a simple goal, segmenting the image into different parts. It is used to simplify images, converting them to consist of only two values, white or black. Furthermore, it has a massive impact on the compression and complexity of the images, and it also works as a basis for object detection algorithms. For example, because of this, it is possible to do Edge Detection work, as long as an edge is nothing but a difference in the intensity of color between pixels.

This algorithm needs a threshold value to work, and it works as a limiter between the two values. Its efficiency is heavily dependent on the value given, and there is a variance of methods focused on finding the best size for it. If the original pixel value is greater than the threshold value, it will be defined as black, and otherwise, it will be white. On one hand, if the value is too big, the output image may be noisy, and on the other hand, if it is too small, it may have an excessive abstraction of details. Concluding, the pre-processing step is an exhaustive process that leads the Convolutional Neural Networks to eventually understand and construct more complex filters, aiming to detect potential features. Nonetheless, to make this procedure possible, the system must be able to master the simpler concepts and use them as a basis.

Figure 16 – CT scan before and after applying HE.



Source: Asuntha e Srinivasan (2020)

3 RELATED WORKS

This section is dedicated to present relevant studies on the research topic. Currently, there are numerous studies that describe different models for detecting and classifying lung nodules in CT scans. Firstly, the study of Wang et al. (2017) Wang, Xu e Sun (2017) will be presented. This study focuses on developing a model for detecting lung nodules using less complex networks, such as AlexNet and ResNet. Next, the study of Shukla et al. (2021) Shukla et al. (2021) will be discussed. The objective of this study was to create a model that can be run on low-memory devices, such as embedded systems. The authors aimed to achieve a good performance while keeping the computational costs low. Subsequently, the study of Sreekumar et al. (2020) Sreekumar et al. (2020) will be discussed. This study introduced a 3D architecture and a special preprocessing stage that resulted in a notable increase in accuracy, but at the cost of increased computational cost due to the 3-dimensional layers. Additionally, the study of Tekade et al. (2019) Tekade e Rajeswari (2018) will be presented. The authors proposed a combination of two architectures: a 3D-Net for feature extraction and nodule classification and a U-Net network for segmentation. Furthermore, the study of Sori et al. (2021) Sori et al. (2021) will be discussed. This study introduced an innovative two-path architecture containing variable kernel sizes, aimed at enhancing the detection of both global and local features. Lastly, a summary of the studies presented will be provided to give a clear overview of the current state of the art in this field.

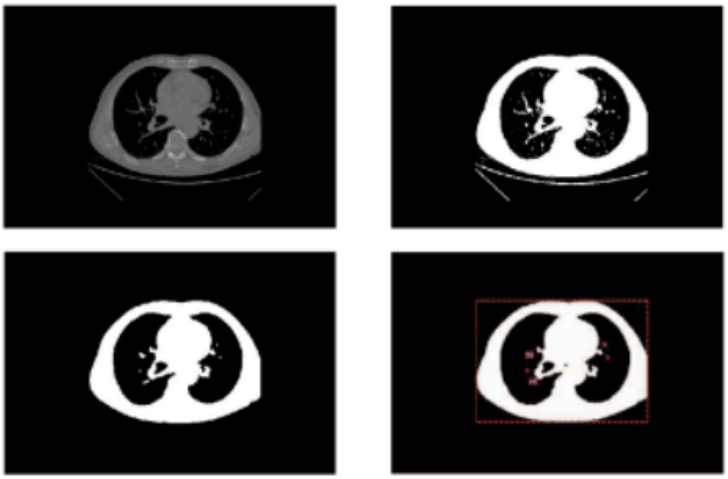
3.1 Work of Wang *et al*

The study of Wang, Xu e Sun (2017) presents a further examination of nodule detection using the AlexNet and ResNet architectures. The first step in this approach is image segmentation, where the pulmonary parenchyma is extracted from the CT images using the Otsu method, which is an adaptive threshold method that tries to maximize the separation between the background and target classes by adjusting the threshold value. After segmentation, noise is eliminated by an open operation, and finally, the regions of interest are extracted from the images. Figure 17 illustrates the image results during the preprocessing step.

The data used in this study were low-dose lung CT images in Meta Image Meta Header (MHD) format. Each scan consisted of a series of axial sections of the thoracic

cavity, with a three-dimensional representation. The lung slices were processed by the model, and the suspected lung nodules were extracted and labeled accordingly. After that, the regions of interest defined in the different slices were expanded and divided into training and testing sets. Finally, the information was transformed by the Caffe framework [5] to be suitable as input for the networks.

Figure 17 – Image preprocessing results



Source: Wang, Xu e Sun (2017)

In conclusion, two different architectures were selected for testing in the experiment: AlexNet and ResNet, both using the same data. As shown in table 2, the AlexNet architecture performed relatively well in the experiment. Additionally, the different deep neural networks were able to automate the detection of pulmonary nodules in the CT scans, but the AlexNet model showed better adaptation to the dataset than the ResNet model.

Table 2 – Training results of Wang *et al*'s work

Network name	*Accuracy*
AlexNet	0.76
ResNet	0.58

Source: Wang, Xu e Sun (2017)

[5] caffe.berkeleyvision.org/

3.2 Work of Shukla *et al*

The Shukla et al. (2021) work focuses on detecting lung nodules using a 2D SqueezeNet model on the Lung Nodule Analysis 2016 dataset [6] (LUNA16). The primary goal is to detect the disease in its early stages, thus reducing the risk of the patient entering more advanced stages.

This approach has a strict preprocessing step since it is necessary to transform the images into the input format required by the 2D network. This process utilizes the SciPy [7] and SimpleITK [8] libraries for image handling since the images are mostly represented in MHD and raw format. Three essential preprocessing steps are taken: first, the information is converted to a viewable form, then the image noise is removed, and finally, it goes through a normalization operation.

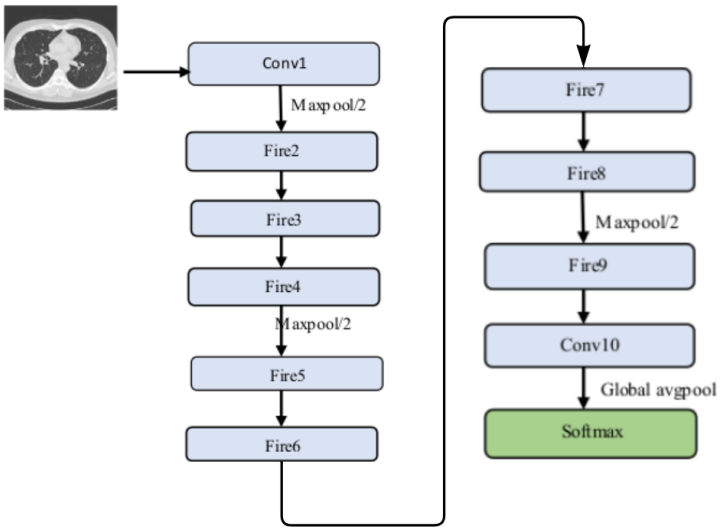
After that, the lung nodules are segmented from the image. Due to the class imbalance, it is necessary to use some method of data augmentation. Then, minor changes were applied to extend the nodule class, such as image flips and rotations. At this point, the information is ready to be fed into the network.

The architecture used was based on SqueezeNet. The reason for choosing a smaller CNN is that they have fewer connections between neurons, making them low-memory organizations. This makes them suitable for running on embedded systems and other equipment with limited memory. Since the goal is to keep the model simple, the SqueezeNet is similar to AlexNet, but with a few changes. The 3x3 channels are replaced with 1x1 channels, and the number of input channels was decreased to 3x3 channels. Additionally, fire modules were designed to feed a squeeze layer into an expand layer, which contains a combination of 1x1 and 3x3 convolution filters. Techniques from MobileNet were also applied to decrease the computation and model size, which is particularly beneficial for this type of application. The network model is shown in Figure 18.

In conclusion, an optimal solution was achieved. The final model can achieve an accuracy above 70 percent in a low-memory model, and the sensitivity and specificity criteria, which are often neglected, are also satisfied, reaching a value of 88 percent specificity.

[6] luna16.grand-challenge.org/Data/
[7] scipy.org/
[8] simpleitk.org/

Figure 18 – Shukla *et al*'s proposed architecture



Source: Adapted from Shukla *et al.* (2021)

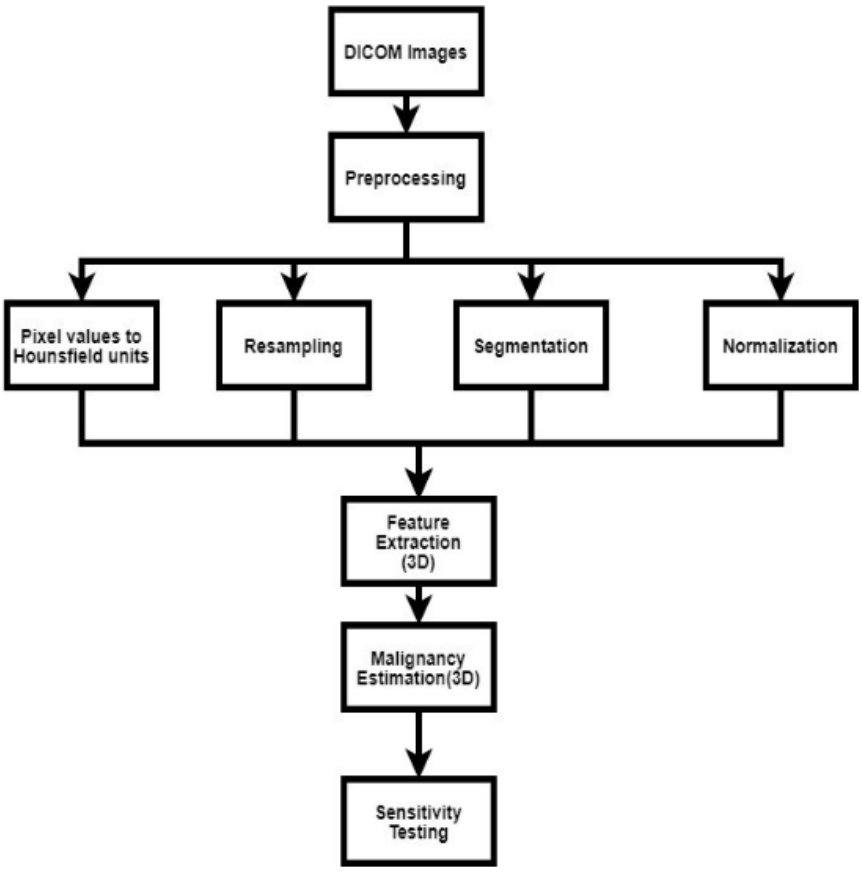
3.3 Work of Sreekumar *et al*

The Sreekumar et al. (2020) work utilizes a Deep Convolutional Neural Network (DCNN) and various preprocessing techniques to automate the prediction and classification of lung cancer malignancy in CT scans. It proposes a model that primarily uses the LIDC-IDRI dataset [1] along with resources from the LUNA16 challenge [6] to reduce false positives. Consequently, the model is able to detect and demarcate pulmonary nodules in the scans. The proposed system is shown in Figure 19.

Since this dataset is composed of Digital Imaging and Communications in Medicine (DICOM) files, a preprocessing step is necessary before feeding the images into the neural network. Four operations were applied in this step: first, the different slices of the scan were loaded into a list, and an isomorphic resampling was performed on the pixel spacing, transforming them into a certain isotropic resolution. Then, the pixels were converted into a specific radio-density measure known as Hounsfield units.

Additionally, the lungs were segmented out of the CT scan to reduce the space problem and increase the effectiveness of feature extraction. Finally, the scans were normalized using a threshold value. After this step, the system is able to start the feature extraction and malignancy detection procedure. However, at this stage, the data had a significant imbalance in classes, and it was necessary to use some data augmentation techniques, reducing the ratio from 1 : 1000 to 1 : 20 for nodules and non-nodules, respectively.

Figure 19 – Sreekumar *et al*'s proposed system



Source: Sreekumar *et al.* (2020)

The model was trained using more than 1000 patient scans, and the architecture was constructed using the BVLC Caffe framework, which is a modifiable system for creating and training CNNs and other deep models (JIA et al., 2014). The network's architecture was adjusted to support 3-dimensional convolutional layers, and it is shown in table 3. This results in a performance gain during the training, testing, and fine-tuning stage when using 3-dimensional convolutional layers. Additionally, the malignancy's metadata extracted from the scans was used to train an estimator ranging from 1 to 5, representing not malignant and malignant, respectively.

Initially, the model was trained with 800 patient scans and was able to reach a sensitivity of 76 percent for detecting nodules and predicting their malignancy after training and minimizing the cost function using stochastic gradient descent. Later on, more scans were added to the training, resulting in a 7 percent increase in sensitivity. Additionally, an adjustment in the image's size was made, which resulted in an extra 3 percent increase in sensitivity. In conclusion, the additional training and advanced preprocessing techniques applied have resulted in a 10 percent increase in sensitivity for

detecting malignant lung nodules, achieving a total of 86 percent in the final model.

Table 3 – Network Architecture of Sreekumar *et al*

Layer	*Parameters*	*Activation*	*Output*
Inputs			32x32x32, 1
AveragePooling3D	2x1x1		16x32x32, 1
Convolution3D	3x3x3	ReLu	16x32x32, 64
MaxPooling3D	1x2x2		16x16x16, 64
Convolution3D	3x3x3	ReLu	16x16x16, 128
MaxPooling3D	2x2x2		8x8x8, 128
Convolution3D (2x)	3x3x3	ReLu	8x8x8, 256
MaxPooling3D	2x2x2		4x4x4, 256
Convolution3D (2x)	3x3x3	ReLu	4x4x4, 512
MaxPooling3D	2x2x2		2x2x2, 512
Convolution3D (Bottleneck)	2x2x2	ReLu	1x1x1, 64
Convolution3D (Nodule Detector)	2x2x2	Sigmoid	1x1x1, 1
Convolution3D (Malignancy Predictor)	2x2x2	None	1x1x1, 1

Source: Sreekumar *et al.* (2020)

3.4 Work of Tekade and Rajeswari

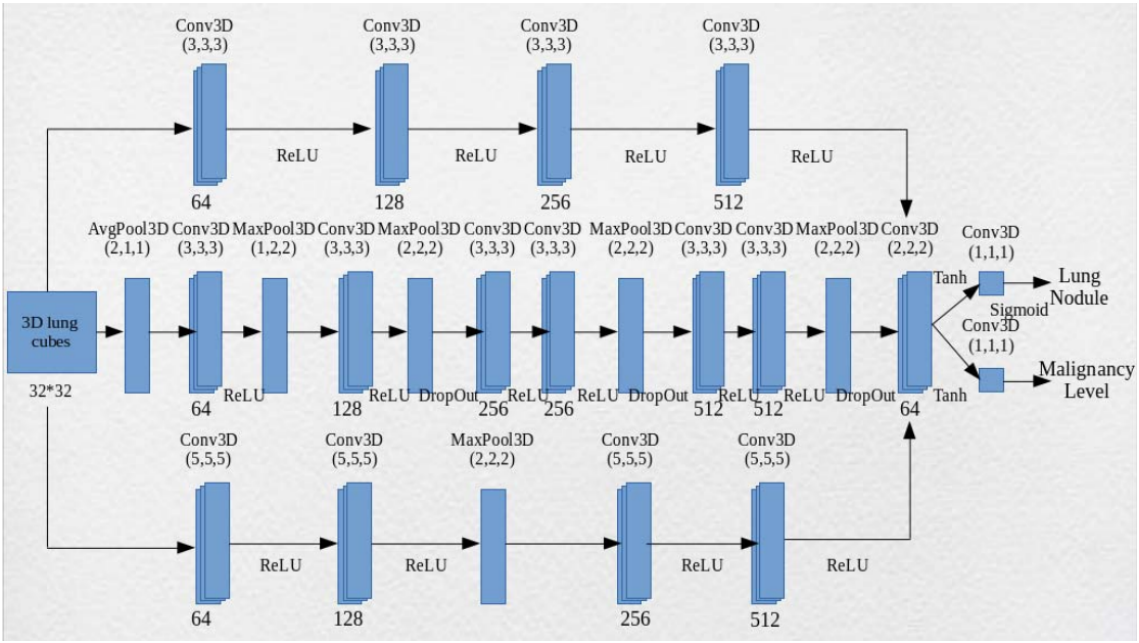
The Tekade e Rajeswari (2018) work used a combination of three different datasets to train and evaluate the network. As a basis, the LIDC-IDRI[1] dataset was used, containing more than 1010 patient cases and 1018 thoracic CT scans in DICOM format. These scans were evaluated by four different radiologists, who classified the nodules into four levels of increasing malignancy. The LUNA16[6] dataset was a competition focused on screening CT scans images from the LIDC-IDRI dataset. The patient IDs' list and lung nodules' location were provided in this competition, and any redundant annotations were removed. This dataset contains a total of 888 CT scans in raw and MHD format. The Kaggle Data Science Bowl 2017[9] (KDSB) was another competition held in 2017, which aimed to increase the efficiency of algorithms used for the classification and detection of lung nodules in CT scans. This dataset contains 1595 patient cases with 286380 CT scans in DICOM format. The three datasets were combined to detect lung nodules' location in CT scans and classify them into different types of cancer.

This process uses two different deep learning architectures. An adapted CNN was employed for feature extraction and nodule classification, as well as a U-Net architecture

[9] www.kaggle.com/c/data-science-bowl-2017

for segmentation. 3D convolutions were used to extract features, while two different types of pooling layers, max pooling and average pooling, were used to select the important ones. The activation function used in the neural network is Rectified Linear Unit (ReLU), which is responsible for passing forward positive values and converting negative inputs to zero (AGARAP, 2018). A classifier was applied to calculate the probabilities of classes, and the difference between the prediction and the correct output gives the loss function. Besides this, optimizers were applied to select better weights. This work introduces a 3D multi-path architecture that uses a VGG-16-based structure, due to its light weights and faster training procedure (SIMONYAN; ZISSERMAN, 2014). The architecture is shown in figure 20.

Figure 20 – Tekade and Rajeswari proposed network.



Source: Tekade e Rajeswari (2018)

The model has fully convolutional layers working simultaneously, which are concatenated in the final output. The classification is done in the last layer, using the Softmax classifier and Adam optimizer. The U-Net architecture generates segmentation masses for each scan, resulting in another nodule's detection approach (RONNEBERGER; P.FISCHER; BROX, 2015). Finally, the result of both models was combined to achieve more accurate results in the nodule detection and malignancy prediction procedures. The model's final results are shown in table 4.

Table 4 – Results of Tekade and Rajeswari's work

Accuracy	*Loss*	*Dice coefficient*	*Predicting log loss*
95.66%	0.09	90%	38%

Source: Tekade e Rajeswari (2018)

3.5 Work of Sori *et al*

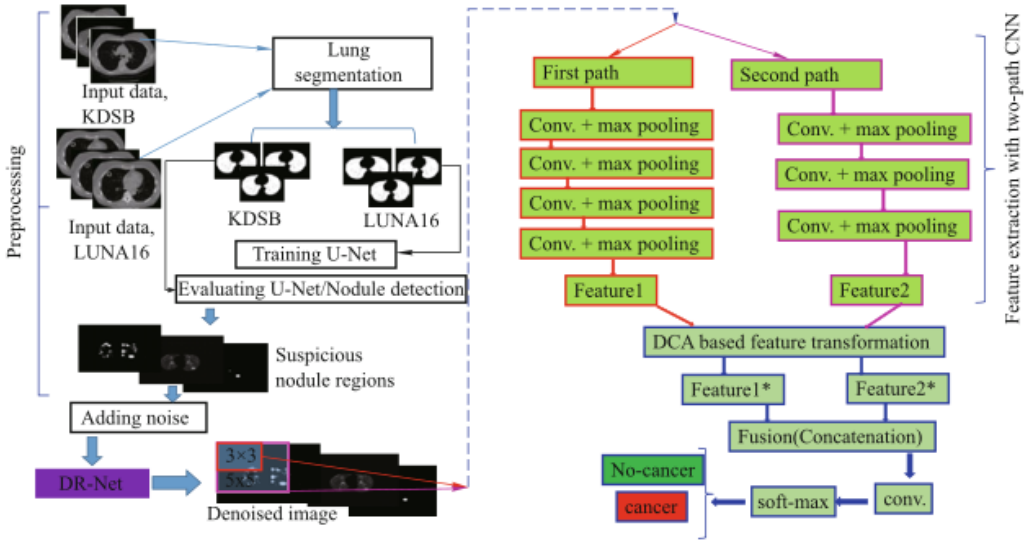
The work of Sori et al. (2021) proposes a different approach using the KDSB and LUNA16 datasets. The CT scans first pass through a denoising step using a residual learning model named DR-Net, which was introduced by their research group (JIFARA et al., 2019). Next, the output is fed into a two-path convolutional neural network to detect the lung nodules. Each path employs different filter sizes to model local and global dependencies, then discriminant correlation analysis is used to concatenate more representative features. Moreover, a retraining technique is applied to overcome difficulties associated with the image label imbalance.

This work uses a 2D-two-path convolutional network to characterize and learn different contextual information about nodules. This approach differs from conventional lung cancer detection methods as it focuses on the integration of local and global dependencies, rather than treating them separately.

Furthermore, a two-stage training is proposed to avoid the critical class difference problem. In this step, a fine-tuned process is applied to the parameters after the first training stage, and a retraining strategy is employed in the output layer. This network is named DFD-Net, and the architecture is shown in figure 21.

Finally, it can be concluded that the two-path architecture performed better than other commonly used approaches in terms of accuracy, sensitivity, and specificity. The architecture was tested on a set of 300 samples, which were evenly divided into larger and smaller nodules. Additionally, this work only used 2-dimensional layers, resulting in lower computational costs. Table 5 shows the results achieved by the network.

Figure 21 – DFD-Net framework



Source: Sori *et al.* (2021)

Table 5 – Predictions of the model

Number of Samples	*Correctly predicted (%)*	*Wrongly predicted (%)*
Bigger Nodules (150)	96.66	3.33
Smaller Nodules (150)	95.33	4.66

Source: Adapted from Sori *et al.* (2021)

3.6 Summary

In this chapter, the work briefly described some CNN architectures for detecting lung nodules in CT scans. Among the different works presented, models that can be considered state-of-the-art solutions to the various problems faced by deep learning models have been highlighted.

However, it is important to note that the works were focused on achieving improvements in different metrics, making it a challenging task to compare them. Despite this, researching different architectures is crucial in determining the CNN model that will be used in this work. Beside that, they also provided a description of the techniques used and the results achieved. Table 6 summarizes the main objectives and results of the presented works.

Table 6 – Summary of Related Works

Work	*Architecture based*	*Main feature*	*Accuracy*
Wang	AlexNet, ResNet	Simplicity	76%
Shukla	SqueezeNet, MobileNet	Low memory cost	72%
Sreekumar	BVLC Caffe	Pre-processing, 2 Datasets	86%
Tekade	VGG-16	3D convolutions, 3 Datasets	95.66%
Sori	DFD-Net	2D-2path architecture	Up to 96.66%

Source: Author (2022)

4 DEVELOPMENT OF A CNN FOR LUNG CANCER DETECTION

This chapter presents the implementation details of the experimental project carried out in this thesis. The focus will be on the software development process, including the requirements, design, and training of the proposed system. The chapter will also provide an overview of the tools and technologies used in the project. In addition, the chapter will present the obtained results from the experiments carried out using the developed system.

4.1 Functional Requirements

1. Create a function to import and process numpy array files;
2. Implement a model for detection of lung nodules in CT scans;
3. Develop a user-friendly interface for easy navigation and analysis of results;
4. Generate reports for further analysis.
5. Achieve high sensibility in lung nodules detection;
6. Reduce false negatives and false positives.

4.2 Nonfunctional Requirements

1. Reduce the training time;
2. Reduce the processing time;

4.3 Potential Challenges and Limitations

1. Large size of the dataset;
2. Substantial amount of time required for training;
3. Necessity for an additional preprocessing step to establish nodule masks;
4. Model's architecture complexity.

4.4 Project Management

This section presents the project management methodology that was employed in this work. Scrum, an agile development methodology widely used in the software industry, was implemented. It consists of an incremental and iterative process, resulting in an effective development environment that aims to meet the different requirements proposed in the project (SRIVASTAVA; BHARDWAJ; SARASWAT, 2017).

The task organization was divided into sprints, each lasting for two weeks. Weekly meetings were organized to plan and define tasks as well as to prepare for subsequent sprints. The task management was systematized using the Kanban methodology, which involves the use of cards to achieve better assignment management (AHMAD; MARKKULA; OIVO, 2013). The sprints were controlled using the Trello[10] platform. Additionally, the project has an online repository available on the GitHub[11] platform. The Git software was utilized throughout the project's development for version control.

4.5 Project Environment

This project will be implemented using the Python[12] programming language. The main library that will be used to build the CNN is PyTorch[13]. Additionally, the OpenCV library will be used to handle images, while the PyQt5[15] library will be used to develop the Graphical User Interface (GUI).

The JupyterLab[16] platform will primarily be utilized as the development environment. In case any performance issues arise during the training stage, the Google Colaboratory Pro[17] platform will be employed to access hardware with improved performance.

The information for this project will be extracted from the LIDC-IDRI[1] dataset, since it has 125 GB of CT Scans, totaling 1018 patient cases. The Pylidc library will be used to assist with Object-Relational Mapping (ORM), making it easier to handle information during the training stage.

[10] trello.com
[11] github.com
[12] python.org
[13] pytorch.org
[15] pypi.org/project/PyQt5
[16] jupyter.org
[17] research.google.com/colaboratory

The PyTorch library will be used during the training stage. It is an open-source ML library used for building and training neural networks in Python. It is particularly popular in the field of computer vision and natural language processing. PyTorch provides a variety of tools for building and training neural networks, including support for dynamic computation graphs, which allows for greater flexibility in building and modifying models. Additionally, PyTorch provides a number of pre-built modules for common tasks such as convolutional layers and pooling layers, making it easy to build and train CNNs. It also has a large and active community, which means that it is easy to find support and tutorials online.

4.6 Lung Nodule Detection and Segmentation

In this section, the process for detecting and segmenting lung nodules is presented. The LIDC-IDRI dataset1 was selected as the primary dataset, as it contains a large amount of information, with a total of 1018 cases and over 130 GB of data. The cases are available in DICOM format, which is widely used to store medical imaging. DICOM files contain a significant amount of metadata, and thoracic information is expressed as a composition of slices, resulting in large files in terms of memory usage.

The Pylidc library was utilized to create a relational mapping between the dataset and application. This allows for querying the data in an SQL-like manner, converting the information into an object with various additional functionalities. This was an important step, as it required further research to understand the library. The library was installed using pip and a configuration file, which contained the path to the dataset, was required to be created.

Jupyter Notebooks were created to conduct this research and are currently available in a GitHub repository[21]. The initial objective was to comprehend the Scan class, which enables performing initial queries on the dataset and retrieving individual scans using the patient ID, as well as customizing the results using filters. The Scan class has several essential attributes that can be beneficial. For instance, this class enables the retrieval of the scan, the total number of nodules, and the corresponding annotations, using a patient ID.

In the LIDC-IDRI dataset, each scan was evaluated by four different radiologists, resulting in a maximum of four annotations per nodule. The annotation class is

[21] github.com/ignssj/lung-cancer/tree/main/pylidc-notebooks

responsible for storing the characteristic values assigned by the specific annotating radiologist. Additionally, annotations can be filtered in the same way as scans by using the appropriate properties in the query.

Another important function of the annotation class is the ability to obtain the diameter, surface area, and volume attributes of the nodules. This enables the determination of their bounding box, which is crucial in the preprocessing stage. Additionally, this class allows for the creation of masks and 3D nodule visualizations, which can be very useful.

After studying Pylidc, it was possible to begin preprocessing the LIDC-IDRI dataset. This step had three main objectives: creating the nodule's masks for training the model, classifying the nodules according to their malignancy, and dividing the dataset into train and test sets.

The first objective is essential for the training process, as it is crucial to define the desired output of the segmentation architecture, represented by the masks. In this task, some code from the Full Preprocessing Tutorial[16] provided by the Kaggle team was used.

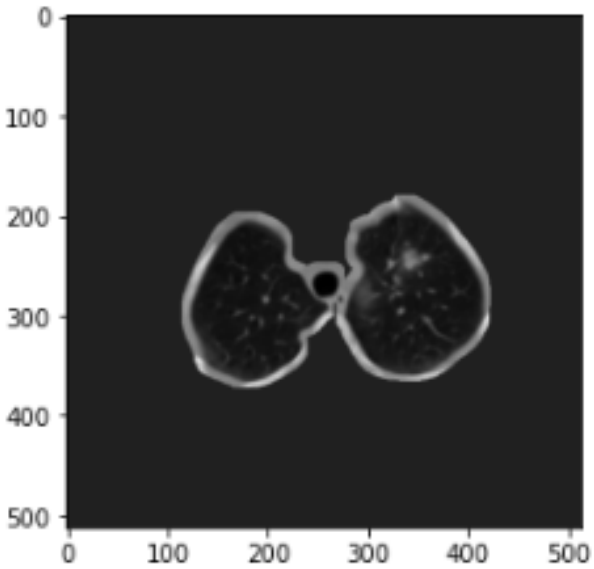
A function was created to determine the malignancy of the nodules. This function calculated the median score of the annotations provided. Nodules with a median score greater than 3 were labeled as positive for cancer, while those with a median score of exactly 3 were considered ambiguous. Nodules with a median score less than 3 were labeled as negative for cancer. The extracted information was then saved in a CSV file for use in the Convolutional Neural Network's training.

Finally, a for loop was employed to iterate through all the patients. Inside this loop, another loop was utilized to iterate through every annotation in the patient's scan. For each nodule found, the malignancy function is called and the corresponding value is written in the CSV file. Additionally, patients who do not have nodules were saved in a different file and will be useful for validating the model. Lastly, a function that utilizes the K-Means algorithm to segment the lungs was applied. Figure 22 illustrates the result of lung segmentation in a specific slice.

Moreover, a data splitting operation was executed, resulting in a split of 80% for the training set and 20% for the test set. As a result, it was possible to accomplish the three main objectives of the preprocessing stage. The lungs were segmented, nodule masks were created, nodules were classified based on their malignancy, and the data was divided into training and testing sets.

[16]https://www.kaggle.com/code/gzuidhof/full-preprocessing-tutorial

Figure 22 – Segmented lungs.



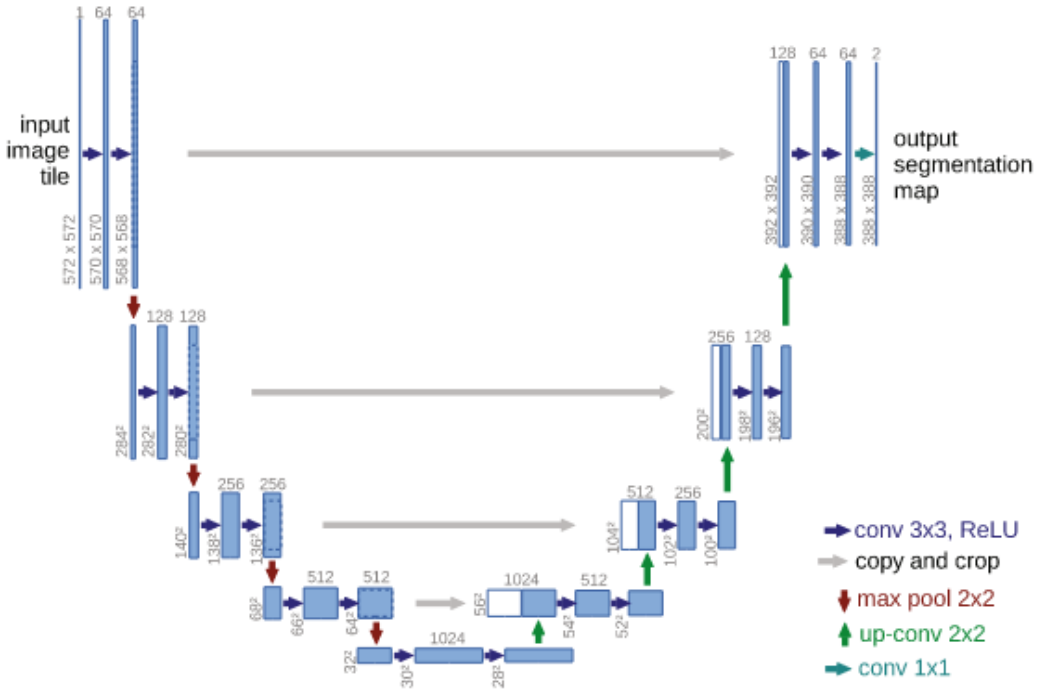
Source: Author (2022)

The next step is to define the architecture model for the detection of lung nodules. The U-Net architecture was chosen for this task as it is currently considered the state-of-the-art method in medical imaging, being employed in Sori et al. (2021) and Tekade e Rajeswari (2018)'s work. The U-Net is a type of FCNN that utilizes an encoder-decoder structure, which is able to learn from a limited number of training samples as noted in (RONNEBERGER; P.FISCHER; BROX, 2015). The general approach is to use downscaling operations to detect an object and then replace the pooling operations for upscaling operations, increasing the resolution of the output and predicting a mask. The U-Net architecture is depicted in figure 23.

The network implementation was sourced from the Github repository[17]. This project adheres to the general concept of U-Net for semantic segmentation by utilizing techniques such as downscaling, upscaling, and double convolutions. The encoder employs multiple convolutional layers to extract and downsample useful features from the input. Meanwhile, the decoder upsamples these extracted features using transpose convolution to generate predicted masks as output. Additionally, the architecture employs a technique known as skip connections, which allows for the direct transfer of selected features from the encoder to the decoder, ultimately resulting in more accurate masks. The properties and connections of all layers can be found in the *UNetModel* module, which is presented in annex A.

[17]https://github.com/milesial/Pytorch-UNet

Figure 23 – U-Net architecture.



Source: Ronneberger, P.Fischer e Brox (2015)

The *UNetParts* module has four distinct classes that describe various operations. Specifically, the *DoubleConv* class is responsible for the implementation of the double 3x3 convolution, batch normalization and reLU activation in the top-left portion of the architecture. Additionally, in the top-right section of the architecture, it is required to perform a convolution with a 1x1 kernel, which is implemented in the *OutConv* class.

In the middle architecture section, downscaling and upscaling techniques are employed. The encoder commences with multiple downscaling operations utilizing the max-pool operation to increase data's resolution, and this is followed by double convolutions. This procedure is repeated four times, at which point in the architecture, the system is able to detect the target. Subsequently, the decoder aims to predict a mask and applies four upscalings followed by double convolutions, to decrease the spatial resolution. All code related to the *UNetParts* module is presented in Appendix B for reference.

At the training stage, the model had been defined and all images and masks generated during the preprocessing phase were ready for use. The model was initially trained using a Nvidia RTX 2060 and the model weights were saved every time the Dice coefficient improved. The Dice coefficient is calculated using equation 11, where X represents the predicted pixels set and Y represents the ground truth.

$$DC = \frac{2 * |(X \cap Y)|}{|X| + |Y|} \tag{11}$$

However, due to this GPU limited memory capacity, with only 6 GB VRAM, the batch size had to be adjusted. As a result, the U-Net validation performance was substantially low, barely reaching 30% training's performance after 50 epochs. This suggests that the model has been adapted too well to the training data and is unable to generalize well to new, unseen data, which is evidence of overfitting. However, these results will be discussed in the Results and Discussion section.

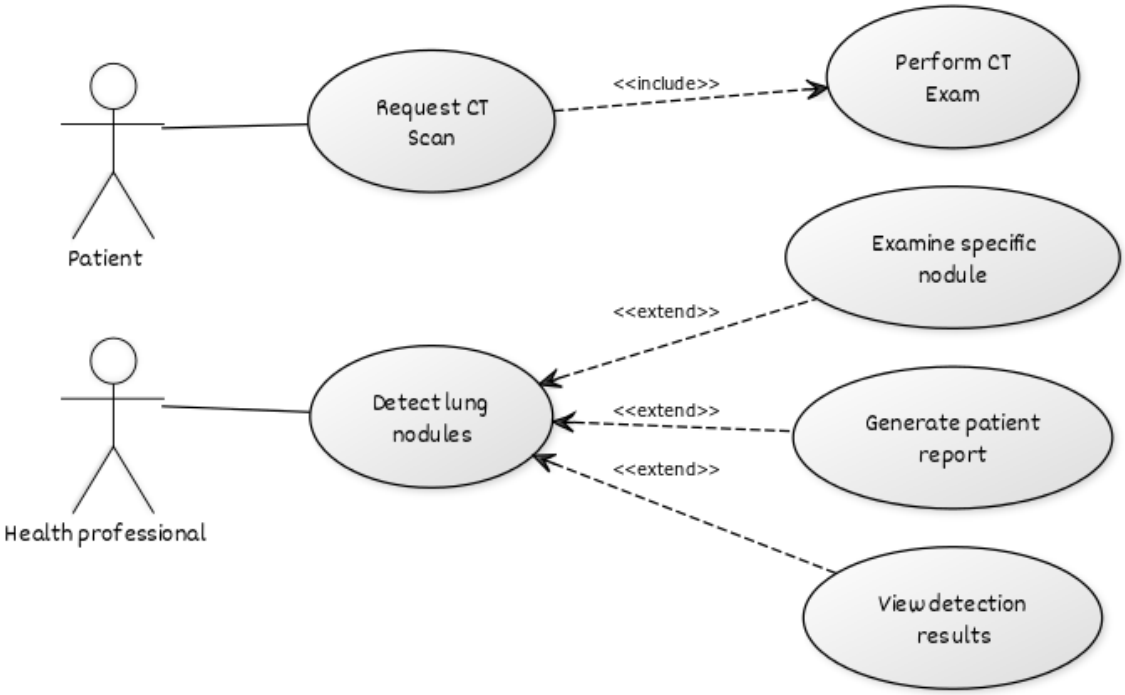
Moreover, it was necessary to upgrade the hardware to improve validation results. The system was retrained for 150 epochs using the Google Colab Pro platform, which provided an A100-SXM4-40GB GPU. Furthermore, the batch size was increased to 24, the GPU's maximum capacity, and the validation Dice coefficient improved significantly, reaching a maximum value near 0.58. To further improve the model's performance, the training process was continued for more 80 epochs, while maintaining the Dice coefficient at or above 0.58. This approach ensured that the model was not overfitting to the training data, and it was able to generalize well to new, unseen data.

4.7 Software Development

The software primary objective is to provide healthcare professionals with a user-friendly and objective workspace, aimed at facilitating the suspicious lung nodules detection. This is achieved by directing attention towards the CT Scan most critical slices. In order to clarify this concept, a use case diagram has been created, as depicted in Figure 24, to clearly outline the context and system requirements. It is imperative to emphasize that the software was specifically designed to aid healthcare professionals in their activities.

The system has several key classes, namely GraphicalUserInterface, UNetModel, DoubleConvolution, DownOperation, UpOperation, and OutConvolution. The GraphicalUserInterface class is responsible for the user interface, including the results presentation and the scans selection. It is connected to a DatasetHandler class, employed to generate metrics and evaluate the CNN performance.

Figure 24 – Software use case diagram.



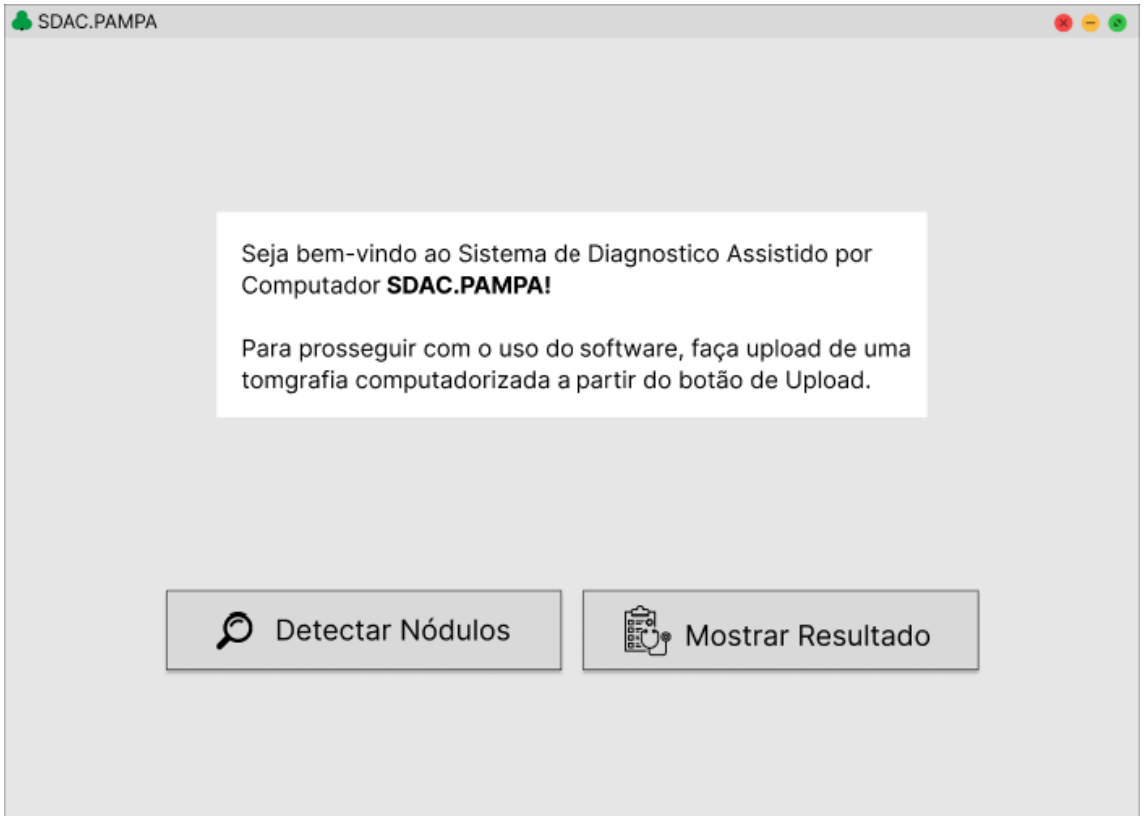
Source: Author (2023)

The UNetModel module contains DoubleConvolution, DownOperation, UpOperation and OutConvolution classes. The DoubleConvolution class employs PyTorch neural network modules to execute double convolution operations, while the DownOperation and UpOperation classes perform down-sampling and up-sampling operations, respectively. The OutConvolution class performs output convolution operations.

The PyTorch neural network modules, such as batchNorm, reLu, 2dConv, upSample, and downSample, are utilized by the DoubleConvolution, DownOperation, and UpOperation classes to execute more complex operations. The UNetModel class also includes the unetLayers method, which is responsible for the overall architecture of the U-Net model.

The graphical user interface (GUI) developed for this project uses the PyQt5 library, a set of Python bindings for the Qt libraries, and a U-Net model trained to detect lung nodules in CT scans. The GUI allows the user to select a folder containing a CT scan and uses the selected folder path to create a list of image paths. The CNN is then used to detect any suspicious lung nodules in the CT scan slices.

Figure 25 – GUI's final design.

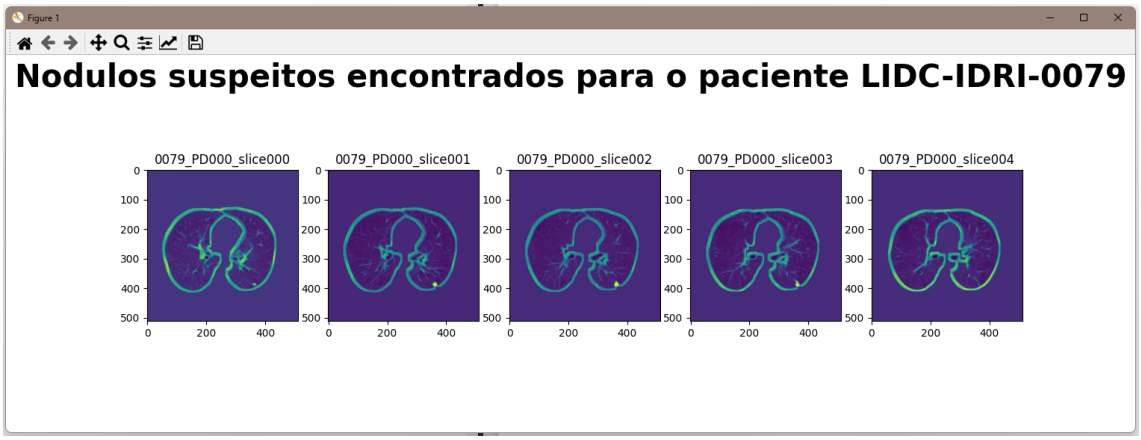


Source: Author (2023)

The detection process outcomes are displayed in a text-editing box on the GUI. The GUI also includes two buttons, one for initiating the detection process, and another for opening the results folder. The layout of the GUI has been designed to be visually appealing, with clear and easy-to-understand instructions and incorporated icons. The GUI was initially designed using the software Figma, and its final version is illustrated in Figure 25.

The GUI has been designed with simplicity, effectiveness and ease of use in mind. Users can easily select a CT scan and initiate the detection process. The detection results are displayed in a new window and automatically saved in an output folder, named according to the patient ID. Figure 26 illustrates the result screen that appears after detection. The screen displays the original image with the predicted mask overlaid, as well as other relevant information, such as the patient ID and scan slice, making it easy for the user to identify and understand the detected nodule location and characteristics.

Figure 26 – Results screen.



Source: Author (2023)

5 RESULTS AND DISCUSSION

This chapter presents a discussion based on the research study findings, which aimed to investigate the convolutional neural networks performance in the lung cancer diagnosis process. The methodology and study results are presented in the previous chapters. The discussion in this chapter is based on the results obtained from the U-Net metrics and is divided into three sections. The initial section presents an overview for the primary research findings. The subsequent section delves into the objectives that were accomplished. The third section examines the study limitations and proposes ideas for future work.

5.1 Model Evaluation

This section presents the study results, which aimed to evaluate a CNN trained to detect lung nodules in CT scans. The model's performance evaluation was based on two metrics: the Dice coefficient and the Binary Cross-Entropy Dice Loss (BCE Dice Loss) metric.

The Dice coefficient is a commonly used metric in the medical imaging field for evaluating the similarity between two sets of data, such as the predicted segmentation and the ground truth segmentation. It ranges from 0 to 1, with a value of 1 indicating a perfect match between the predicted and ground truth segmentations. It can be quantified through the previously presented equation 11.

The Binary Cross-Entropy Dice Loss (BCE Dice Loss) is a combination of two loss functions, the binary cross-entropy loss and the dice loss. This metric is used to optimize the model's parameters during the training process, it ensures that the model learns to classify the images correctly while also learning to segment the nodules correctly. Combining the two methods allows for some diversity in the loss, while benefitting from the BCE's stability. The BCE Dice Loss is calculated using equation 12, where N is the total number of pixels in the image, y is the ground truth mask, x is the predicted mask and DSC is the Dice coefficient function.

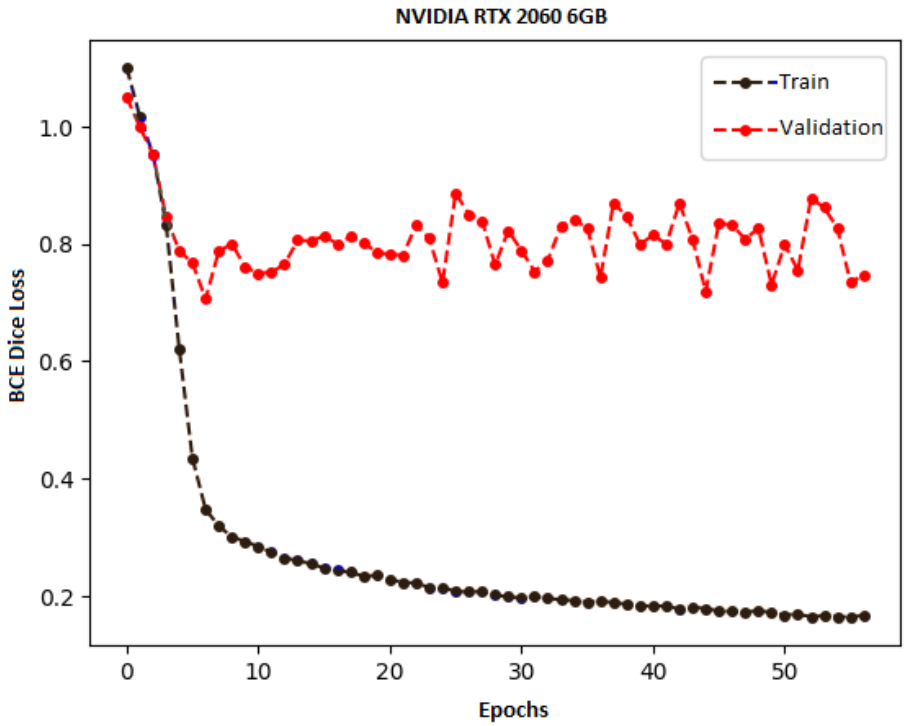
$$L_{BCEDice} = -\frac{1}{N}\left[y\log(x) + (1-y)\log(1-x)\right] + \frac{1 - DC(x,y)}{2} \tag{12}$$

In summary, the Dice coefficient provides a model's segmentation accuracy

measure, while the BCE Dice Loss metric is used to optimize the model during the training measuring the difference between the predicted and actual outputs. In the following sections, the model's performance results on the test dataset will be presented, and their implications will be discussed.

Firstly, the U-Net model was trained locally for 50 epochs using a NVIDIA RTX 2060 6GB. Owing to the GPU limited memory capacity, the batch size was reduced to 2 in order to fit within the GPU's VRAM, and the process took 27 hours. At this stage, the model achieved 0.20 training and 0.70 validation in the BCE Dice Loss measures, approximately. The validation Dice Coefficient also reached a value close to 0.38, whereas the training measure increased until 0.93. It is evident that the model performed notably better on the training set, which suggests that the model was overfitting to the training data. The BCE Dice Loss and Dice Coefficient results, obtained from the model trained locally, are illustrated in Figures 27 and 28, respectively.

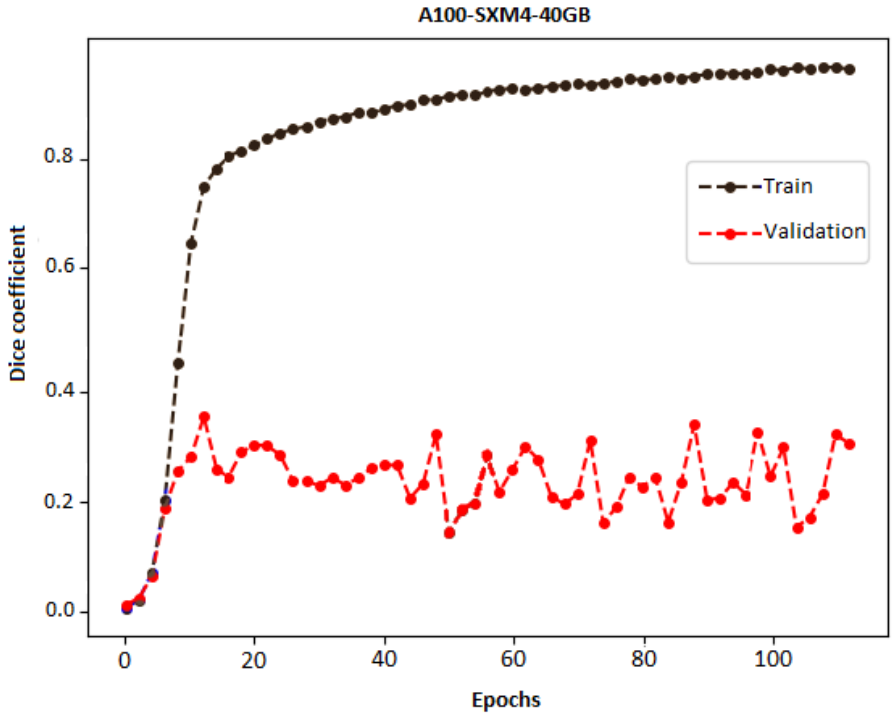
Figure 27 – Local BCE Dice Loss.



Source: Author (2023)

Therefore, the model was retrained on the Google Colab platform using a A100-SXM4-40GB GPU. The batch size was increased to 24, and the system was trained for 150 epochs. Due to this change, the model attained 0.46 validation BCE Dice Loss, while the training measure decreased to nearly 0.05. Similarly, the training and validation Dice coefficient increased to 0.95 and 0.58, correspondingly.

Figure 28 – Local Dice Coefficient.

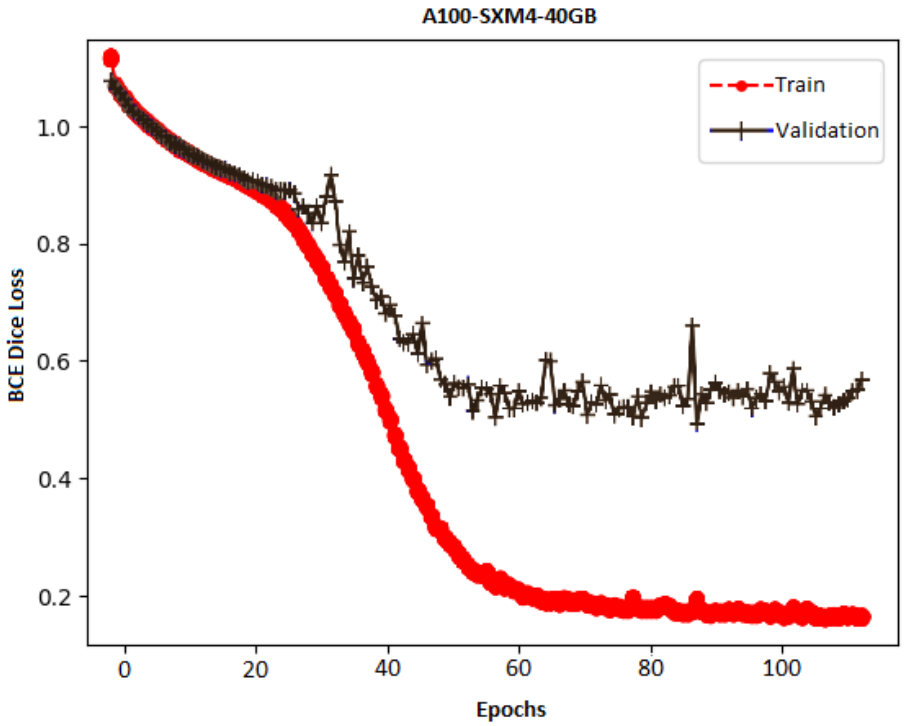


Source: Author (2023)

The results suggest that using a smaller batch size negatively impacts the model's performance, particularly on the validation dataset. However, using a more powerful GPU and larger batch size helped the model to generalize better to unseen data, resulting in better overall performance. The BCE Dice Loss and Dice Coefficient results obtained by the model trained on Google Colab Pro's platform are presented in figures 29 and 30, respectively.

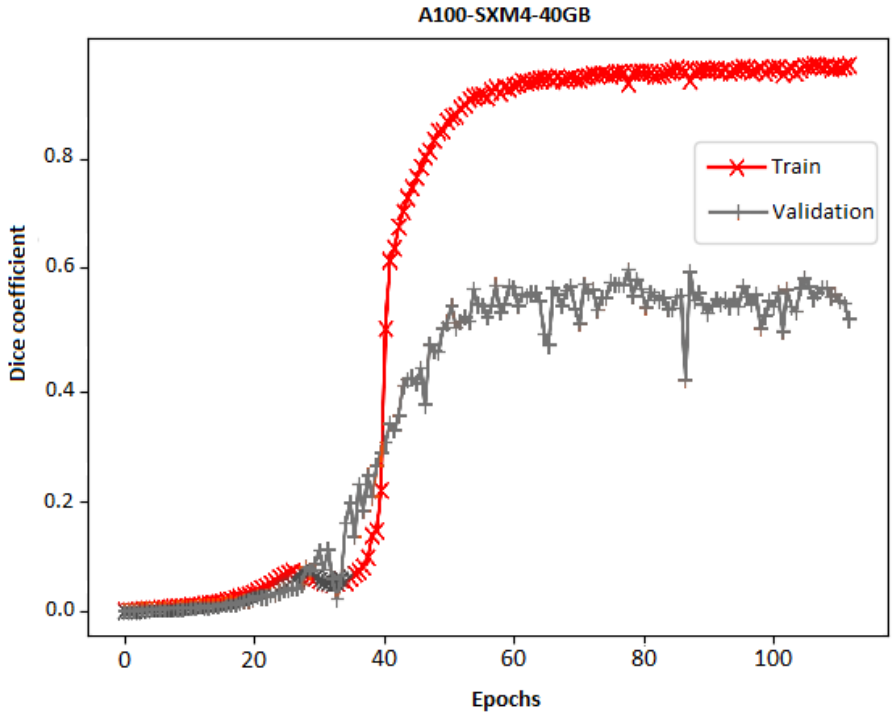
Another important metric that was calculated is the confusion matrix, which is a tool used to evaluate the performance of a binary classification model by determining the number of correct and incorrect predictions for each class. It has four elements: true positives, true negatives, false positives, and false negatives. In this study the confusion matrix is particularly important, as it is a known issue that U-Net architectures tend to produce a high number of false positives. This is a common problem that is caused because the U-Net architecture can only be trained using real target masks. Figure 31 shows the proposed model confusion matrix, calculated in the LIDC-IDRI dataset.

Figure 29 – Google Colab Pro BCE Dice Loss.



Source: Author (2023)

Figure 30 – Google Colab Pro Dice Coefficient.



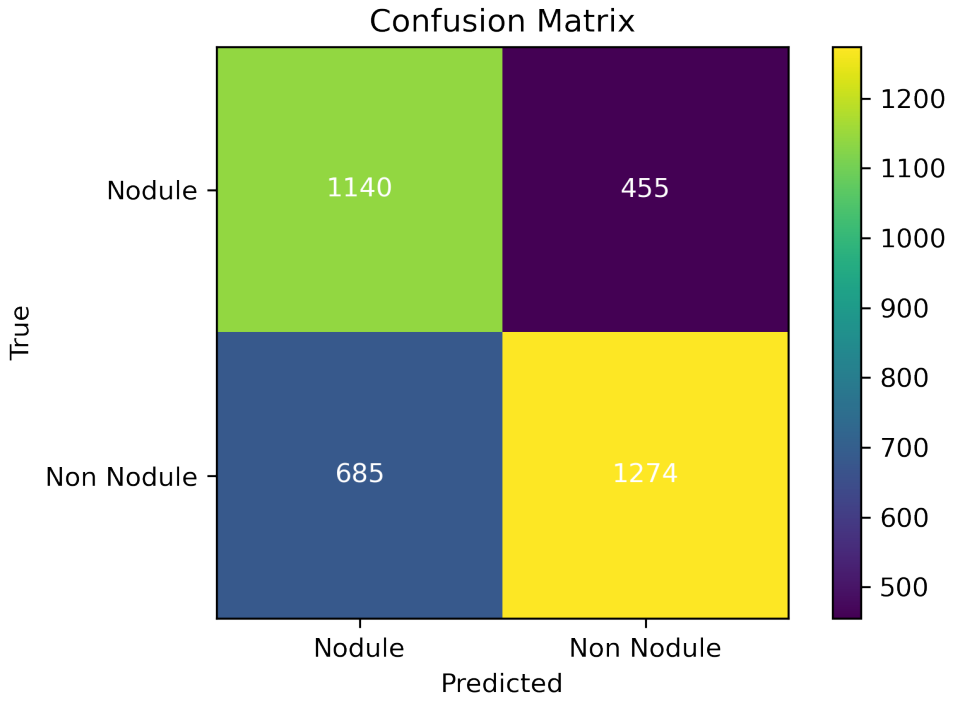
Source: Author (2023)

Furthermore, calculating the confusion matrix turns possible to compare this model with the best architectures in the LUNA16 Challenge. In this challenge, the main evaluation metric used was the sensibility, which basically determines the true positives proportion in relation to all positive cases. The sensibility is given by equation 13.

$$Sensibility = TruePositive/(TruePositive + FalseNegative) \tag{13}$$

In other words, this metric is employed to measure how good the model is at detecting a nodule when one definitely exists. The model trained in this work achieved a total sensitivity of 0.624, ranking $20th$ in the LUNA16 Challenge. Table 7 presents a comparison between the proposed architecture and the results from the challenge. As demonstrated, the proposed model demonstrated a capability to detect lung nodules with sufficient sensitivity, earning a ranking among the top 20 architectures in the LUNA16 Challenge. However, it remains inferior to the challenge's state-of-the-art models.

Figure 31 – Confusion matrix.



Source: Author (2023)

Table 7 – LUNA16 Nodule Detection Results

Rank	Team	*Score*
1	PAtech (PA_tech)	0.951
2	JianPeiCAD (weiyixie)	0.950
3	LUNA16FONOVACAD (zxp774747)	0.947
4	iFLYTEK-MIG (yinbaocai)	0.941
5	zhongliu$_xie$($zhongliu.xie$)	0.922
6	iDST-VC (chenjx1005)	0.897
7	qfpxfd (qfpxfd)	0.891
8	CASED (CASED)	0.887
9	3DCNN$_NDET$($lishaxue3$)	0.882
10	Aidence (mjharte)	0.871
11	junxuan20170516 (chenjx1005)	0.865
12	MEDICAI (bharadwaj)	0.862
13	Ethan20161221 (ethanhwang2012)	0.856
14	resnet (QiDou)	0.839
15	CCELargeCubeCnn (Intel$_wuhui$)	0.833
16	ZNET (gzuidhof)	0.811
17	MOT$_M5Lv1$($elopez69$)	0.742
18	VisiaCTLung (jacobsc)	0.715
19	etrocad (jefvdmb2)	0.676
20	**This proposed model**	0.624
21	M5LCADThreshold0.3 (atraverso)	0.608

Source: Author (2023)

5.2 Completed Tasks

This section presents the objectives that were completed during this work. Firstly, a Deep Learning study was conducted, which helped to understand the concepts and algorithms that work in this field, as well as the process that enables models to recognize patterns in data and learn from them.

Additionally, a CNNs study was conducted and it was determined that they are the state-of-the-art in the object recognition field. They have the capability to recognize a wide range of classes and different architectures can be used to achieve performance gains in various tasks such as memory usage, accuracy, reducing false positives and negatives, and computational cost. However, it was also noted that these deep models require a large data amount and high computational power to train.

A systematic review of related literature was also carried out to understand the CNNs application in the lung nodules detection process. It was found that this type of network can accurately detect and classify lung cancer, making it a viable option for creating a system to assist healthcare professionals in the diagnosis process.

Furthermore, the different available datasets for training models for detecting lung nodules in CT scans were explored. The LIDC-IDRI, LUNA16, and KDSB datasets were identified as the main sources of data. The LIDC-IDRI dataset was downloaded and used for the initial experiments. The Pylidc tool was studied to interact with the CT scans, and preprocessing steps were performed to organize the DICOM files and create masks of the nodules within the scans. The data was also split into train, test, and validation datasets for further analysis.

Following, the U-Net model was defined as the architecture for the detection and segmentation process, since it is considered the state-of-the-art method in medical imaging. The model was trained, looking for improvements of the dice coefficient, whereas information were collected to create graphs and analyze the model's performance. The model was validated through a comparision between preprocessed masks and predicted masks. In the last part of this work, an GUI for the software was implemented. Finally, the project is available in this github repository[18].

5.3 Future Work

This section presents a comprehensive overview of the tasks that were completed during the course of this research project. The primary objective of this work was to develop a CNN for detecting lung nodules in CT scans. In order to improve the performance of the model, further research can be conducted to reduce the rate of false-positives. To address this issue, suggestions for improvement have been proposed. These include training another ANN to reduce the false-positive rate, implementing a feature to generate patient reports, creating a separate screen to examine specific nodules, and implementing the GUI in a different thread to improve the software's overall efficiency. These recommendations aim to optimize the performance and usability of the model, making it more practical for healthcare professionals in the lung cancer diagnosis process.

[18] github.com/ignssj/sdac.pampa

6 FINAL CONSIDERATIONS

In conclusion, this study evaluated the performance of a CNN trained to detect lung nodules in CT scans. The model's performance was mainly evaluated using two metrics: the Dice coefficient and the sensibility. The results showed that the model performed well in terms of overall accuracy, however it was found to generate a high false positives number.

Initially, the model was trained locally with a smaller batch size, which resulted in low validation results. To address this issue, the model was retrained on the Google Colab platform using an A100-SXM4-40GB GPU, with a larger batch size and a longer training period. The results indicate that using a more powerful GPU and larger batch size assisted the model in generalizing better to unseen data, resulting in improved overall performance.

Another important metric that was calculated is the confusion matrix, which is a tool used to evaluate the binary classification model performance, by determining the correct and incorrect predictions for each class. In this study the confusion matrix was particularly important, to confirm the high false positives occurrences.

It is important to highlight that there is still room for improvement. One suggestion to improve this work would be to train another ANN to reduce the false-positive rate. Additionally, implementing a functionality to generate patient reports and creating a separate screen to examine a specific nodule could be beneficial. Furthermore, implementing parallelism to run the GUI in a different thread could improve the software's efficiency.

Overall, this study has demonstrated that CNNs have the potential to achieve high accuracy in detecting lung nodules in CT scans, making them a valuable tool to assist healthcare professionals in the diagnostic process. However, further research is required to address the issue of high false positive rates and to enhance the overall performance of the model.

REFERENCES

AGARAP, A. F. Deep learning using rectified linear units (relu). **arXiv preprint arXiv:1803.08375**, 2018.

AGGARWAL, C. C. et al. Neural networks and deep learning. **Springer**, Springer, v. 10, p. 978–3, 2018.

AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: IEEE. **2013 39th Euromicro conference on software engineering and advanced applications**. [S.l.], 2013. p. 9–16.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. **2017 international conference on engineering and technology (ICET)**. [S.l.], 2017. p. 1–6.

ASUNTHA, A.; SRINIVASAN, A. Deep learning for lung cancer detection and classification. **Multimedia Tools and Applications**, Springer, v. 79, n. 11, p. 7731–7762, 2020.

CHARBUTY, B.; ABDULAZEEZ, A. Classification based on decision tree algorithm for machine learning. **Journal of Applied Science and Technology Trends**, v. 2, n. 01, p. 20–28, 2021.

DAVIES, E. R. **Computer vision: principles, algorithms, applications, learning**. [S.l.]: Academic Press, 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT press, 2016.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, Wiley-Blackwell, v. 160, n. 1, p. 106, 1962.

HURBANS, R. **Grokking Artificial Intelligence Algorithms**. [S.l.]: Manning Publications, 2020.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. **arXiv preprint arXiv:1408.5093**, 2014.

JIFARA, W. et al. Medical image denoising using convolutional neural network: a residual learning approach. **The Journal of Supercomputing**, Springer, v. 75, n. 2, p. 704–718, 2019.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LITJENS, G. et al. A survey on deep learning in medical image analysis. **Medical image analysis**, Elsevier, v. 42, p. 60–88, 2017.

MAJAJ, N. J.; PELLI, D. G. Deep learning—using machine learning to study biological vision. **Journal of vision**, The Association for Research in Vision and Ophthalmology, v. 18, n. 13, p. 2–2, 2018.

MEHLIG, B. Machine learning with neural networks. **arXiv preprint arXiv:1901.05639**, 2019.

MOR-YOSEF, S. et al. Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. **Obstetrics and gynecology**, v. 75, n. 6, p. 944–947, 1990.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. **the Journal of machine Learning research**, JMLR. org, v. 12, p. 2825–2830, 2011.

RASCHKA, S. **Python machine learning**. [S.l.]: Packt publishing ltd, 2015.

RONNEBERGER, O.; P.FISCHER; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: **Medical Image Computing and Computer-Assisted Intervention (MICCAI)**. Springer, 2015. (LNCS, v. 9351), p. 234–241. (available on arXiv:1505.04597 [cs.CV]). Disponível em: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach, eBook**. [S.l.]: Pearson Higher Ed, 2021.

SHUKLA, V. V. K. et al. Lung nodule detection through ct scan images and dnn models. In: IEEE. **2021 6th International Conference on Inventive Computation Technologies (ICICT)**. [S.l.], 2021. p. 962–967.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SORI, W. J. et al. Dfd-net: lung cancer detection from denoised ct scan image using deep learning. **Frontiers of Computer Science**, Springer, v. 15, n. 2, p. 1–13, 2021.

SREEKUMAR, A. et al. Malignant lung nodule detection using deep learning. In: IEEE. **2020 International Conference on Communication and Signal Processing (ICCSP)**. [S.l.], 2020. p. 0209–0212.

SRIVASTAVA, A.; BHARDWAJ, S.; SARASWAT, S. Scrum model for agile methodology. In: IEEE. **2017 International Conference on Computing, Communication and Automation (ICCCA)**. [S.l.], 2017. p. 864–869.

SZELISKI, R. **Computer vision: algorithms and applications**. [S.l.]: Springer Science & Business Media, 2010.

TEKADE, R.; RAJESWARI, K. Lung cancer detection and classification using deep learning. In: **2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)**. [S.l.: s.n.], 2018. p. 1–5.

TOM, M. Mitchell: Machine learning. **1997 Burr Ridge**, IL McGraw Hill, v. 45, n. 37, p. 870–877, 1997.

TRASK, A. W. **Grokking deep learning**. [S.l.]: Simon and Schuster, 2019.

WANG, Z.; XU, H.; SUN, M. Deep learning based nodule detection from pulmonary ct images. In: IEEE. **2017 10th International Symposium on Computational Intelligence and Design (ISCID)**. [S.l.], 2017. v. 1, p. 370–373.

APPENDIX A – GUI SDAC.PAMPA

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog
import torch
import torch.backends.cudnn as cudnn
import torch.nn as nn
from Unet.unet_model import UNet
from dataset import MyLidcDataset
import os
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
import yaml

OUTPUT_DIR = "output/"
PATIENT_ID = ""
INPUT_PATIENT = ""
PATIENT_IMG_PATHS = ""

class Ui_MainFrame(object):
    def setupUi(self, MainFrame):
        MainFrame.setObjectName("MainFrame")
        MainFrame.resize(592, 425)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(".\\/Icons/unipampa-icon.png"),
        QtGui.QIcon.Normal, QtGui.QIcon.Off)
        MainFrame.setWindowIcon(icon)
        MainFrame.setLocale(QtCore.QLocale(QtCore.QLocale.Portuguese,
        QtCore.QLocale.Brazil))
        self.pushButton = QtWidgets.QPushButton(MainFrame)
        self.pushButton.setGeometry(QtCore.QRect(120, 320, 161, 41))
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap(".\\/Icons/lupa.png"),
        QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.pushButton.setIcon(icon1)
        self.pushButton.setIconSize(QtCore.QSize(18, 18))
        self.pushButton.setObjectName("pushButton")
        self.textEdit = QtWidgets.QTextEdit(MainFrame)
        self.textEdit.setEnabled(True)
        self.textEdit.setGeometry(QtCore.QRect(80, 100, 441, 101))
```

```
        self.textEdit.setAutoFillBackground(False)
        self.textEdit.setReadOnly(True)
        self.textEdit.setObjectName("textEdit")
        self.pushButton_2 = QtWidgets.QPushButton(MainFrame)
        self.pushButton_2.setGeometry(QtCore.QRect(320, 320, 161, 41))
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap(".\\/Icons/diagnosis.png"),
        QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.pushButton_2.setIcon(icon2)
        self.pushButton_2.setIconSize(QtCore.QSize(20, 20))
        self.pushButton_2.setObjectName("pushButton_2")

        self.retranslateUi(MainFrame)
        QtCore.QMetaObject.connectSlotsByName(MainFrame)


    def open_folder_dialog(self):
        folder_path = QFileDialog.getExistingDirectory()


        if folder_path:
            global INPUT_PATIENT
            INPUT_PATIENT = folder_path
            PATIENT_MASK = INPUT_PATIENT.replace("Image","Mask")


            input_slices = os.listdir(INPUT_PATIENT)
            full_image_paths = [os.path.join(
            INPUT_PATIENT, slice) for slice in input_slices]

            global PATIENT_IMG_PATHS
            PATIENT_IMG_PATHS = full_image_paths

            mask_slices = os.listdir(PATIENT_MASK)
            full_mask_paths = [os.path.join(
            PATIENT_MASK, slice) for slice in mask_slices]

            list_image_paths = list(full_image_paths)
            list_mask_paths = list(full_mask_paths)

            # extrai o ID do paciente
```

```
            global PATIENT_ID
            PATIENT_ID = os.path.basename(
            os.path.dirname(full_image_paths[0]))

            # Directory to save U-Net predict output
            print("Salvando resultados em {}".format(
            OUTPUT_DIR+PATIENT_ID))
            os.makedirs(OUTPUT_DIR+PATIENT_ID,exist_ok=True)

            test_dataset = MyLidcDataset(list_image_paths,
            list_mask_paths)
            test_loader = torch.utils.data.DataLoader(
            test_dataset,
            batch_size=config['batch_size'],
            shuffle=False,
            pin_memory=True,
            drop_last=False,
            num_workers=6)

            counter = 0
            with torch.no_grad():

                for input, target in test_loader:
                    input = input.cuda()
                    target = target.cuda()

                    output = model(input)



                    output = torch.sigmoid(output)
                    output = (output>0.5).float().cpu().numpy()
                    output = np.squeeze(output,axis=1)

                    for i in range(output.shape[0]):
                        label = list_image_paths[counter][-23:]
                        label = label.replace('NI','PD')
                        np.save(OUTPUT_DIR+PATIENT_ID+'/'+label,
                        output[i,:,:])
                        counter+=1

```

```
122     def open_result_folder(self):
123         folder_path = QFileDialog.getExistingDirectory()
124 
125         if folder_path:
126             outputs = os.listdir(folder_path)
127             full_result_paths = [os.path.join(
128             folder_path, output) for output in outputs]
129 
130             fig, axs = plt.subplots(1, len(full_result_paths))
131             fig.suptitle(
132             'Nodulos suspeitos encontrados para o paciente'+
133             PATIENT_ID,fontsize=30,fontweight='bold')
134 
135             for i, array_file in enumerate(full_result_paths):
136                 original_ct = np.load(PATIENT_IMG_PATHS[i])
137                 detected_nodule = np.load(array_file)
138                 axs[i].imshow(original_ct+detected_nodule)
139                 axs[i].set_title(os.path.basename(array_file)
140                 .split(".")[0])
141 
142             plt.show()
143             plt.get_current_fig_manager().window.showMaximized()
144 
145     def retranslateUi(self, MainFrame):
146         _translate = QtCore.QCoreApplication.translate
147         MainFrame.setWindowTitle(_translate("MainFrame", "SDAC.PAMPA"))
148         self.pushButton.setText(_translate("MainFrame",
149         "Detectar Nodulos"))
150         self.pushButton.clicked.connect(self.open_folder_dialog)
151         self.textEdit.setHtml(_translate("MainFrame",
152         "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
153         \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"
154 "<html><head><meta name=\"qrichtext\" content=\"1\" />
155 <style type=\"text/css\">\n"
156 "p, li { white-space: pre-wrap; }\n"
157 "</style></head><body style=\" font-family:\'MS Shell Dlg 2\';
158 font-size:8.25pt; font-weight:400; font-style:normal;\">\n"
159 "<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
160 margin-right:0px; -qt-block-indent:0; text-indent:0px;\">
161 <span style=\" font-family:\'Ubuntu\'; font-size:11pt;\">
162 Seja bem-vindo ao Sistema de Diagnostico Assistido por Computador
```

```
</span><span style=\" font-family:\'Ubuntu\'; font-size:11pt;
font-weight:600;\">SDAC.PAMPA</span><span style=\" font-family:
\'Ubuntu\';font-size:11pt;\">!</span></p>\n"
"<p style=\"-qt-paragraph-type:empty; margin-top:0px;
margin-bottom:0px; margin-left:0px; margin-right:0px;
-qt-block-indent:0; text-indent:0px; font-family:\'Ubuntu\';
font-size:11pt;\"><br /></p>\n"
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;\">
<span style=\" font-family:\'Ubuntu\'; font-size:11pt;\">
Para prosseguir com o uso do software, selecione uma tomografia a
partir do botao Detectar Nodulos.</span></p></body></html>"))
        self.pushButton_2.setText(_translate("MainFrame",
        "Mostrar Resultado"))
        self.pushButton_2.clicked.connect(self.open_result_folder)

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainFrame()
    ui.setupUi(MainWindow)
    MainWindow.show()

    with open('E:\Projetos\Python\sdac.pampa/config.yml', 'r') as f:
        config = yaml.safe_load(f)
    model = UNet(n_channels=1, n_classes=1, bilinear=True)
    state_dict = torch.load("E:\Projetos\Python\sdac.pampa/model.pth")
    model.load_state_dict(state_dict['model_state_dict'])
    model = model.cuda()


    sys.exit(app.exec_())
```

ANNEX A – UNET MODEL

```python
import torch.nn.functional as F
from .unet_parts import *

class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits
```

ANNEX A – UNET PARTS

```python
i""" Parts of the U-Net model """

import torch
import torch.nn as nn
import torch.nn.functional as F


class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3,
            padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3,
            padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)


class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

```

```
    def forward(self, x):
        return self.maxpool_conv(x)


class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear',
            align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels,
            in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels ,
            in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)


    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = torch.tensor([x2.size()[2] - x1.size()[2]])
        diffX = torch.tensor([x2.size()[3] - x1.size()[3]])

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)


class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels,
        kernel_size=1)

    def forward(self, x):
        return self.conv(x)
```