Facultad
de
Ciencias

Clasificación de imágenes médicas utilizando técnicas de Deep Learning
(Classification of medical images using Deep Learning techniques)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: Carmen García Bermejo

Director: Lara Lloret Iglesias

Co-Director: Diana Tordesillas Gutiérrez

Junio - 2020

Acknowledgements

Studying Physics Degree has really been an adventure. I would like to thank to have shared this experience with such as wonderful and brilliant people that have contributed so much. Especially, I would like to thank my tutors, Lara and Diana, for being so patient and understanding these years, and for teaching me this amazing field of computer science, where I hope to continue growing.

Also, I acknowledge the support of the Advanced Computing and e-Science group at the Institute of Physics of Cantabria (IFCA-CSIC-UC).

Abstract

In this project, Deep Learning, a sub-field of Machine Learning and Artificial Intelligence, is applied to the analysis and classification of medical images. To achieve this goal, an architecture based on convolutional neural network is used. This type of network learns which are the main features from the training image dataset allowing to solve the classification problem. First of all, a summary of the state of the art both for Deep Learning and medical imaging is presented followed by the model description and the discussion of the results. A last chapter on possible improvements and future work is also included. For this project, the network has been trained using Xception model, after studying the performance of the network with other models, obtaining an accuracy of 91.7%.

KEYWORDS: medical imaging, Deep Learning, machine learning, artificial intelligence, convolutional neural networks, computer vision.

Resumen

En este proyecto, se estudia la aplicación de Deep Learning, un sub-campo del Machine Learning y de la Inteligencia Artificial, en el análisis y la clasificación de imágenes médicas. Para alcanzar este objetivo, se emplea una arquitectura basada en la redes neuronales convolucionales. Este tipo de red aprende cuáles son las características principales de las imágenes del set de entrenamiento permitiendo resolver el problema de clasificación. Inicialmente, se presenta un resumen sobre el estado del arte tanto del Deep Learning como de la imagen médica, seguido de la descripción del modelo y la discusión de los resultados. También, se incluye un capítulo final acerca de las posibles mejoras y el futuro trabajo en este campo. Para este proyecto, se ha entrenado la red usando el modelo Xception, después de haber estudiado el rendimiento de la red con otros modelos, obteniendo una precisión de 91.7%.

PALABRAS CLAVE: imágenes médicas, aprendizaje profundo, machine learning, inteligencia artificial, redes neuronales convolucionales, visión por computación.

Contents

Chapter 1

Introduction

In the last years, our society has entered a pixel era. Nowadays, a big part of the information that the technology is managing is in multimedia format, where around 85% of the Internet data is in pixels and that is why visual data has become very important. Furthermore, many people are at the same time treating this information, so it is necessary to develop a visual technology capable to understand it and work with it. This is one of the most important problems that technology is facing: the huge amount of data.

One of the consequences of this situation is suffered by hospitals and health systems, where the analysis and diagnosis of medical images (X-Ray, TACs ...) must follow a slow process due to the great number of patients.

The aim of this project is the development of an application based on Deep Learning using a Convolutional Neural Network (CNN). This type of network is based on learning the optimal parameters with a set of filters that extract the most relevant information from the images in order to classify them. These filters are convolved over the full image, extracting some particular features and creating a new representation of the initial data (also known as feature map) where only this feature is visible. This process is called feature extraction. Since many filters are used, the initial image is then decomposed into its main characteristics. The learning process is hierarchical since the first layers of the neural network learn low-level features and the last layers learn high-level features. The process of feature extraction is repeated several times over all the image dataset, so that the network is capable of correctly learning how to distinguish between the categories into play. Specifically, in this project, it distinguishes between healthy lungs and lungs with pneumonia.

The network in this work has been developed in Python, using TensorFlow and Keras. TensorFlow is an open-source software library used for machine learning that allows building and training neural networks. Keras is also an open-source library that acts as an API with Tensorflow as backend.

Initially, we will make an introduction about the historical context of artificial intelligence and computer vision and the need to use this technology to help with the massive data processing. Next, we will go deeper into the mathematical concepts that build this technology and the architecture used in this project followed by an introduction to medical imaging. Finally, we will discuss the results, ending with possible improvements and future work.

Chapter 2

Artificial Intelligence

Artificial intelligence (AI) has been the subject of science fiction for many years, from literature to cinematography, representing futuristic scenarios where civilization is dominated by machines that act and think like humans.

This technology has been sketched throughout human history, for instance, it was already present in millennial Greek myths like the story of the god Hephaestus, who manufactured creatures such Talos, a giant of bronze that protected Creta; or in the philosophy of Ramon Llull, who already in the thirteenth century anticipated that the reasoning could be automated setting the theoretical basic rules in his design of the *Ars Magna*, a machine that could prove whether a theological or scientific postulate was true or false.

However, it was not until the mid-twentieth century when it began to become part of the technical program, where a group of pioneers started to think about how machines can simulate humans. Already in 1950, Alan Turning's essay *Computing machinery and intelligence* [22] put into question the capacity of machines to think with the Turing test, where the ability of a computer is tested to check an intelligent behaviour.

Until the late eighties, the solution proposed to AI programming was a collection of methods, called *Symbolic AI*, which assumed AI could be handled through a great set of programmed rules. This worked very well for logical problems, where the rules are clear, like playing chess, but failed at solving more complex problems such as image classification or speech recognition.

AI has been increasingly developed in the last years, concretely, since the nineties it has had exponential growth, due to the increase of computers capacity and, on the other hand, to the massive quantity of data. These two factors, together with the economic investment, led to the development of Deep Blue by IBM in 1997, a supercomputer that won Gary Kaspárov in a memorable chess game, becoming the first computer that won a game of chess to a current world champion.

Artificial Intelligence is then the capacity of a machine to solve problems usually associated with human intelligence. With the symbolic AI paradigm, the system is capable, starting from some defined rules, to solve problems by performing millions of calculations in a much shorter time than a human. However, there are some problems where the rules are not clear and human intuition makes its appearance on the scene.

Let's put an example: let's imagine we want to build an application that distinguishes cats and dogs.

Figure 2.1: A dog and a cat. [1]

Our brain is able to easily distinguish a cat and a dog in Figure 2.1, even though both animals have quite similar characteristics: both have four legs, alike size, pointy ears... How can we then program the rules to distinguish them? This is an example of a problem that is trivial for a human, but that can become very tricky for a machine. It is not possible to translate this intuition problem to a defined rule. That is why Symbolic AI is not valid for this kind of problems.

2.1 Computer vision

Since it became necessary to resolve recognition problems, Symbolic AI no longer worked. We can only appeal to the most perfect visual technology existing: the human visual process.

Biologically, the vision has an evolution of more than five hundred million years [1]. The primary visual cortex is located in the occipital lobe, far from the eyes, at the back of the human brain, and is the area where a great number of neurons work in the first stage of the visual process.

The visual process can be understood as levels of layers connected by a simple structure that process the incoming information. Simple cells that are in the primary visual cortex have receptive fields that respond to the elemental characteristics of objects such as lines or edges with a defined orientation and location. They act like low-level feature detectors. More complex cells reply to the movement of these lines in a determined direction, and the hyper-complex ones respond to angles and corners that move in a defined direction or to lines with specific length and orientation that move in a determined direction.

Reached this point, we can wonder when visual computing started. In 1963, Lawrence G. Roberts began with simple structures, so the visual process interpretation is composed by layers, as it did at a biological level, which means, we think about an image and we imagine some layers forming this image. Initially, a great number of pixels are received, and they have to be identified in different groups according to their characteristics. In the '90s, computer visual process changed radically due to colour images introducing larger matrices in the reading of the image data. For example, the algorithm which locates the faces in a digital camera or in our phones learns the characteristics from these groups of pixels.

In this way, the most important part of visual computing focuses in recognition, i.e learning important characteristics that enable to recognize determined objects is required and also, it is

[1] obtained in *pharmAdeje* https://www.farmaciapharmadeje.com

essential to understand how these characteristics configure the space.

2.2 Machine Learning

Machine Learning is part of the artificial intelligence based on meaningful data transformations in order to resolve a problem. Its goal is to create programs that can generalize human behaviour from information given in the form of examples that induce knowledge to the computer, allowing it to "learn". This learning can be performed in three different ways:

- Supervised: the machine learns from a set of examples previously classified. Many problems can be solved by applying this learning technique, such as image classification, programs that delete spam emails, or speech recognition. Supervised learning is the most common algorithm nowadays.
- Unsupervised: this learning technique consists of finding common characteristics that allow separating the different classes without using labelled data. This model is commonly used in clustering algorithms or to comprehend a dataset before we start a supervised learning algorithm.
- Reinforcement: this method is the most recent. It consists in getting information from the environment and, in order to maximize some recompense, learning by choosing among a pool of possible actions. It has applications in robotics, self-driving, etc.

In this project, we are going to use a supervised learning algorithm, so our machine is going to learn from a labelled training dataset and then, create a model that is able to classify the new images that it has never seen before.

For training these kinds of models, we first need a training dataset and the corresponding labels indicating the category that the data belongs to and, also, a function to tell us how well the Machine Learning model is performing the classification, in order to have some clear metric to be optimized during the learning process.

To sum up, these algorithms automatically find operations, such as coordinate changes, translations, projections, etc., that transform data into more useful representations from predefined operations, helped by a feedback signal.

2.2.1 Classifiers before Deep Learning

One of the models used for classifying images, and also, the simplest one, is the *Nearest Neighbour Classifier* [1]. It memorizes the training dataset and, when a new image is given, it compares it to every single image of the training set. The new image is then classified with the label associated with the closest example that can be found among the training set.

In order to find the closest image, we need to first define a distance metric. One of them is the L1 distance (or sometimes called *Manhattan distance*) that compares individual pixels in both images and it is defined [1] in the following equation:

$$d(I_1, I_2) = \sum_p |I_1^p - I_2^p| \tag{2.1}$$

where d is the distance, I_1^p is the value of the pixel of the new image in the position p and I_2^p is the value of the pixel of the training image in the position p. Basically, it gets the absolute value of the difference between the pixels of each image and then it sums all these pixels up (the difference between every spatial position), adds all of them, and checks how similar both images are. If both images are identical, the final value obtained is 0, so, the lowest value will correspond to the closest training image.

Another set of algorithms, which are a bit more complex, are the so-called support vector machines (SVMs). These algorithms are basically focused on searching the best hyperplane for dividing a dataset into two classes [3]. The closest points to the hyperplane are known as *support vectors* and they are very important because if they are modified or deleted, the dividing hyperplane would see its position affected. In Figure 2.2, we can see an example of how SVMs work for a classification task:

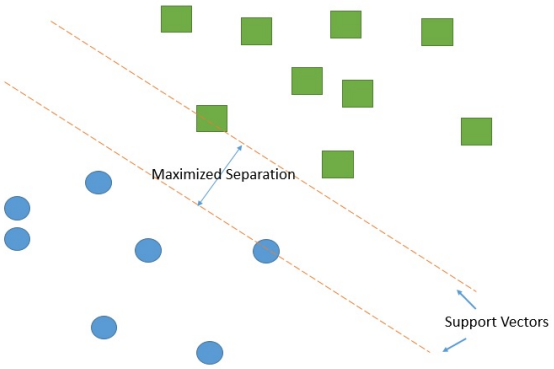


Figure 2.2: Graphical representation of the optimal separation. [2]

We can be more confident that it has been correctly classified the further away from the hyperplane the points are, which must be on the right side. Then, wherever new testing data lands, any of both sides of the hyperplane will determine the class.

As we can see, these kinds of methods are quite simple, but nowadays, classification problems need more complex algorithms. After this short review of some of the Machine Learning algorithms, let's now introduce the concept of a Neural Network.

[2] obtained in *Wikimedia Commons* https://commons.wikimedia.org/wiki/Main_Page

Chapter 3

Neural Networks

Neurons are the elementary unit of artificial neural networks. A neuron receives one or more inputs and by a non-linear function provides a single output resulted from the sum of inputs. This function is called *activation function* and, without it, it would be only possible to perform linear problems. It is the result of the summation of weighted input:

$$f(x_1, x_2, ..., x_m) = w_1x_1 + w_2x_2 + ... + w_mx_m, \tag{3.1}$$

where m is the number of inputs. Input data x will pass through the activation function in order to perform a linear transformation (rotation + translation) by applying the matrix of weights W and the b bias factor:

$$z = W^T x + b \tag{3.2}$$

As we are working in a classification scenario, the output must be the probability of an event. Let's study the case of a binary classification, where we have two classes: a_1 and a_2, for example. The probability of a_1 gives us the probability of a_2, because $P(a_1) + P(a_2) = 1$, thus, the mean value of this distribution must be between 0 and 1. As activation function, we can use the *sigmoid* $\sigma(z)$, which is widely used in binary classification:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \rightarrow \sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{3.3}$$

$\sigma(z)$ gives smooth output values, and its derivative depends on the function itself. The sigmoid function is graphically represented as:

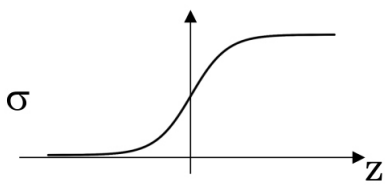


Figure 3.1: Sigmoid function.

3.1 Learning process

The learning process is a multi-step one. In this section, we will describe this process for one single neuron where we can distinguish the following steps:

3.1.1 Forward propagation

The neuron receives one or more inputs, where each one is separately weighted and their sum is passed through the activation function to produce an output.

$$input \rightarrow neuron \rightarrow output$$

So, being x the input and its prediction a the final output. That is:

$$x \rightarrow f(x, W) \rightarrow xW + b \tag{3.4}$$

parameters W and b are applied to the input data and, in the first iteration, both values are random. Taking z as:

$$z = W^T x + b \rightarrow a = \sigma(z) \rightarrow L(a, y) \tag{3.5}$$

where $L(a, y)$ is known as the loss function that will tell us how well we are performing; a is the prediction and y is the true label. This case is for a single neuron, a neural network has more neurons like this connected one to another. That is why they are called networks.

3.1.2 Gradient descent

Let $f(x)$ be any function and $f'(x)$ its derivative, the slope of $f(x)$ at point x is given by $f'(x)$. In order to know how the input has to change to get an appropriate output, we minimize $f(x)$, by [9], that is:

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{3.6}$$

Now, just by shifting x in small increments with opposite sign of $f'(x)$, it is possible to minimize $f(x)$. This method is known as the *gradient descent*.

3.1.3 Loss Function

The accuracy of the model is measured by the loss function. In summary, when the algorithm does not guess correctly, it sums high numbers; otherwise, it sums smaller values or none at all. In this way, the bigger the function gets, the worse our classification algorithm is working. This tells us in which direction we have to go the next time that we introduce the input with the new weights. If the prediction gets better, the loss function decreases and we repeat the process until an optimal value is found.

One of the most commonly used loss functions, specially in regression problems with a linear prediction function, is the mean square error [9]:

$$MSE = \frac{1}{m}\sum_{i}(a-y)_i^2 \tag{3.7}$$

where a are the predictions, y is the vector of regression targets and m is the number of example inputs used for evaluating. This technique determines the square of the difference between the predicted value and the actual one.

Through the loss function, we calculate the difference between the output we get and the right output, determining the error in the prediction. That is why it is necessary for training to use the right function, since distinct loss functions end up in different errors and therefore, different consequences in our system. If the neural network has multiple outputs, multiple loss functions can be involved, but the gradient descent process has to be based on a unique scalar loss value, so these loss functions are compounded into a single scalar value.

Since we have non-linear prediction functions (due to *sigmoid* function), we can use another different loss function for a binary case (m=2), the cross-entropy:

$$L(y,a) = -[ylog(a) + (1-y)log(1-a)] \tag{3.8}$$

Then, we can have two cases:

- If $y = 1$, loss function is $L(y,a) = -log(a)$. Then, we have to minimize $L(y,a)$ in order to maximize a.
- If $y = 0$, loss function is $L(y,a) = -log(1-a)$. We have to minimize $L(y,a)$ in order to minimize a.

Hence, performing wrong predictions contributes positively to the loss function, making it bigger, so that minimizing loss function leads to better predictions. This is actually the learning process.

3.1.4 Backpropagation

In the first iteration, the values of W and b are random, but they are not anymore in the next iterations. Both values change in order to get the smallest loss function possible. Therefore, we use an algorithm that searches for the minimum value of the weights and the bias.

Consequently, we have to minimize the loss function through gradient descent. First, derivatives of W and b are calculated and the values are updated in order to follow the direction of the minimum. Once the first iteration is done, W can be greater or smaller than the minimum value of W that minimizes the loss function. If it is smaller, the slope at that point and the derivative of dL/dW, is negative, in the opposite case where W is greater, the slope is positive. The same process can be applied for b. Both values are updated by the following formula using the chain rule:

$$\left.\begin{aligned} \tfrac{dL}{dw} = dW = \tfrac{dL}{da}\tfrac{da}{dz}\tfrac{dz}{dw} \\ \tfrac{dL}{db} = db = \tfrac{dL}{da}\tfrac{da}{dz}\tfrac{dz}{db} \end{aligned}\right\} \begin{aligned} W &= W - \alpha dW \\ b &= b - \alpha db \end{aligned} \tag{3.9}$$

Where α is the learning rate hyperparameter, which gives the magnitude of the step to take in the direction of the minimum. The smaller α, the slower the learning process is because we are taking smaller steps to approach the minimum. This process is known as *backpropagation* and it is repeated with all inputs, updating for each iteration W and b.

To sum up, as we can see in Figure 3.2, in order to decrease the loss score, weights and biases are tuned slightly in the right direction every time that the network processes an example. This is the "training loop", which, repeated enough times, makes the predicted output closer to the real targets and therefore, the network gets trained.

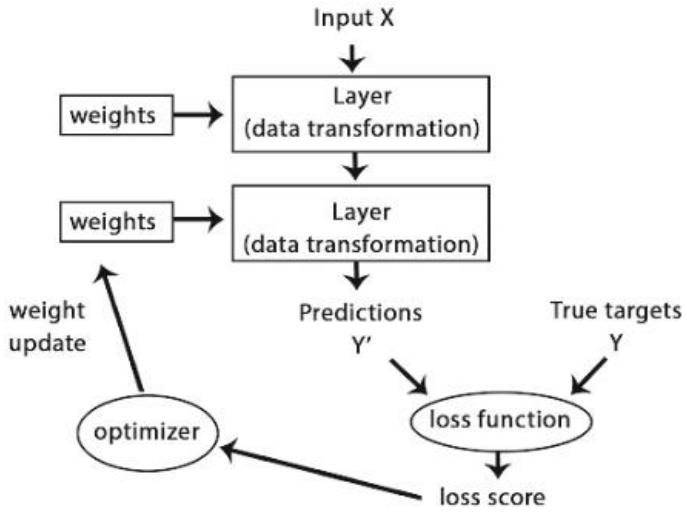


Figure 3.2: Backpropagation scheme [3]

3.2 Multiple classification

Moving from a binary scenario to a multiple one, we have to customize the learning algorithms.

Softmax regression is used to classify into multiple categories, so the output layer has the same number of units as categories. It is defined by [9]:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.10}$$

Softmax returns values between 0 and 1 that can be interpreted as probabilities of a given result. Then, the probability distribution of each label over m different labels is obtained. The sample will be classified as the label corresponding to the highest probability.

3.2.1 Loss function

The cross entropy loss function is frequently used for classification problems:

$$L(y, a) = -\sum_j y_j log a_j \tag{3.11}$$

where y is the true label and a the predicted value. As we want $L(y, a)$ to be as small as possible, we have to minimize it, then, the probability of the right category is maximized:

$$Cost = \frac{1}{m} \sum_{i=1}^{m} L(y, a) \tag{3.12}$$

3.2.2 Activation functions

The sigmoid activation function presents several problems that make it unstable when going to Deep Learning. Having a small derivative implies that systems with many layers will obtain a gradient so small that they won't learn almost anything. This problem is called *vanishing gradient*. We must then introduce another function with bigger derivative values.

These new activation functions that will be widely used in Deep Learning architectures are ReLU and Leaky ReLU:

- ReLU. From *Rectified Linear Unit*, is defined by: $y(x) = max(0, x)$, where x is the input. More accurate results are obtained with this function.
- Leaky ReLU. It is an alternative to ReLU and it is defined by the following expression:

$$y(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \tag{3.13}$$

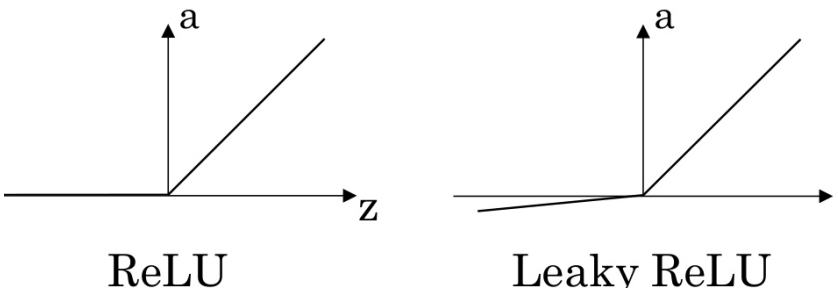


Figure 3.3: Graphic representation of ReLU and Leaky ReLU.

With these functions, we are able to avoid the vanishing gradient problem.

Chapter 4

Deep Learning

Deep Learning is a subfield of Machine Learning. It consists of a set of algorithms that try to configure high-level abstractions in data using architectures made of multiple non-linear transformations. Today, we can find many examples of Deep Learning applications to many fields such as particle physics, cybersecurity, medical research, etc. So, we can place Deep Learning inside Machine Learning and this, inside AI:

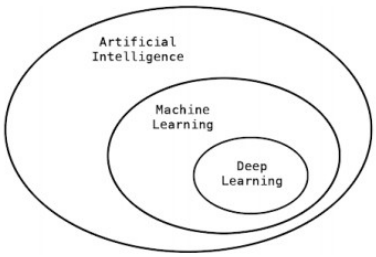


Figure 4.1: Scheme of AI, Machine Learning and Deep Learning. [3]

Deep Learning describes an automatic searching system of finding the best representations for the target problems using many layers of neurons. Convolutional neural networks are the most promising algorithms used in Computer Vision nowadays. A Deep Learning neural network is basically a mathematical framework to get many useful representations hierarchically. We can see a scheme in the following picture:

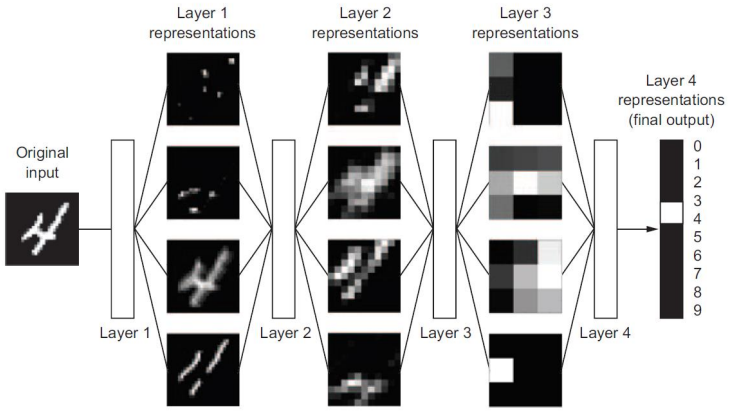


Figure 4.2: Representation of a classification model of handwritten numbers. [3]

In this example, the network has to distinguish handwritten numbers from 0 to 9. The original input is inserted in the first layer *Layer 1*. Each layer uses as input the output of the previous layer that is its representations or feature maps and all of them form a hierarchy of filters whose parameters can be trained as weights. Finally, the representations of the last layer will tell us what number is the original input, which is the final input. In this case, it is number 4.

4.1 Image Classification

Image classification is a core task in computer vision. From the point of view of a computer, an image is composed of pixels that are represented by different values. Every image is made of pixels forming a rectangular matrix. We can then define a pixel as an image element, being the smallest homogeneous unit of a digital image.

We can differentiate two types of images according to their colour:

- Black and white. The image is a matrix with values between 0 and 255 depending on the grayscale. It is 256 possible values corresponding to 2^8 for 8 bits images.
- Coloured image. It has three different matrices with values between 0 and 255. These matrices correspond to the RGB colours: red, green and blue, respectively. The final image is obtained by mixing these three colour channels.

This big amount of numbers supposes a problem called semantic gap, that means, the same object with two descriptions can have different meanings. Any change like zoom, rotate or just taking the picture in a different position with the camera of the image where an object, for example, is represented, even if it represents the same object, for the computer it is completely different, because every pixel has new different values, so we have the same object representation with two distinct arrays and we need the algorithms to be robust to this.

This situation, this viewpoint problem, is also repeated with the illumination, deformation, occlusion (where we just see a part of the object in the image), the background, or just the intraclass variation where the object appears two or more times in the image. All these situations create new representations for our algorithm. The correct handling of this situation is very important for visual recognition.

There are different models for classifying images, but in each supervised learning image classifier, there are three datasets, each of them composed of images and their corresponding labels.

- *Training set*: as it is indicated in its name, it is used for the training part, where the algorithm is learning.
- *Development set*: also known as *dev set*, it is the validation dataset, where we will be testing the model while it is training to detect possible problems such as overfitting.
- *Test set*: is used to determine the final accuracy of the classifier.

To decide which hyperparameter fits better in our classifier, we have to split the data into training, validation and test or if we have a small amount of data we can do cross-validation, where we separate data into folders, try each one as validation and test the results.

4.2 Convolutional Neural Networks

Convolutional Neural Networks, or CNN, can be understood as an artificial neural network where neurons are receptive fields similar to the neurons that are in the primary visual cortex in a biological brain. They consist of multiple layers of convolutional filters of one or more dimensions. After each layer, one function is added to make non-linear mapping.

Layers are the heart of neural networks. They are a kind of filter that receives some data, extracts its representations out and converts them into a more meaningful set. In a CNN, layers are placed one after another, each time more refined, following the example in Figure 4.3:

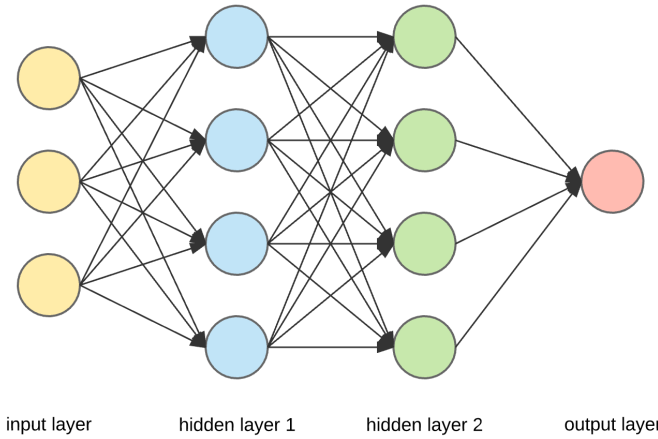


Figure 4.3: Layers scheme [4]

Each layer is composed of neurons (or nodes) where each one is connected to the others in the following layer. These neurons take the weighted sum of the input, they receive and pass this data into an activation function that follows the equation Eq 3.1. The result is the output of the neuron that will become the input of another neuron in the next layer.

Different layers are involved in a network as we can see in the previous Figure 4.3. *Input layers* (or *visible*) contain the initial variables that will be processed by the rest of the layers, that is why it is always set at the beginning of the system; *output layers* are the last ones and they provide the classification result of the network; and *hidden layers* take as input the outputs of previous layers and by an activation function, converts it in the input of the following layer. They are called *hidden* because they are between the input and the output layers. Each hidden layer is getting more meaningful representation, for example, the first hidden layer identifies the edges, the second one can find the corners and contours; the third, one piece of a determined object, and so. The more number of hidden layers, the deeper the network becomes.

In summary, training of a neural network works around layers, input data and targets, a loss function and an optimizer that show how the learning advances. In this way, Deep Learning uses CNN as a tool to learn the representations in layers that are combined in order to form a network. These learn patterns that are invariant to translations, which means that, if they recognize a certain pattern in a region of an image, the network will find it in any other place, so they have a great capacity for generalization. Likewise, convolutional neural networks can learn the spatial hierarchy of the images, so the first convolutional layer learns to extract low-level features (such as lines and edges) and the second one will combine them to create more complex concepts.

4.2.1 Variance and bias

For the study of the precision of our network, we need to investigate different kinds of errors that can be found: bias, variance and irreducible errors.

We have to note that irreducible error (also called *noise*) alludes to its name, it will never disappear, we are not going to create a machine with 100% hits, that is not possible. But, on the other hand, *bias* and *variance* can be reduced.

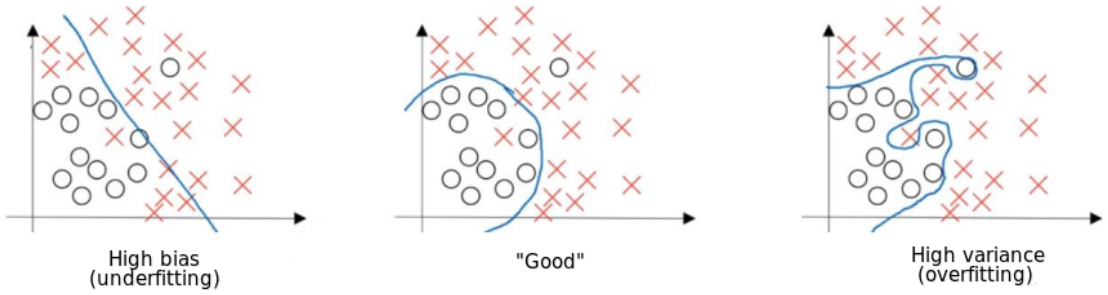


Figure 4.4: Graphic bias and variance [1]

Bias shows how far is our estimated value from the right one. A low bias indicates that our estimations are close to the target and, on the other hand, high bias shows that our algorithm is losing some relevant information between features and targets, oversimplifying the model. This leads to high error and *underfitting*, this means that we probably have a small network that does not have enough flexibility to properly learn from the training data. Underfitting can be solved with a greater network or training longer.

Variance reveals the deviation from the expected value, the shorter it gets, the more accurate is the estimated value. High variance means that the model pays to much attention to the training data and is no generalizing new data correctly, leading to high error rates on test data, therefore, in *overfitting*. The first solution is to find more data, but it is not always possible. Fortunately, there are more ways to solve it. One of them is by using *regularization*.

Regularization

There are many ways of using regularization when we are facing overfitting.

One example is *Data Augmentation* that, basically, generates more training data by using meaningful transformations on the existing training data so that this is slightly modified. These transformations are random and different every time a new sample is analyzed. In this way, our network will not be seeing exactly the same training data twice.

There is another method we can use to fight overfitting, it is called *Regularization Ridge*, also known as *L2*. Its goal is to reduce as much as possible the error in the training set and in the prediction in order to avoid overfitting. For example, linear regression:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||w||^2$$

[1] obtained in *Packt* in Machine Learning Quick Reference: https://subscription.packtpub.com/

where $w \epsilon R^{n_x}$, $b \epsilon R$. Alluding to section 3.2.1 *Loss Function*, w are the weights, $L(a^{(i)}, y^{(i)})$ is the loss function where a are the predictions, y is the vector of regression and m is the number of example inputs used for evaluating. Also, λ is the regularization parameter that is chosen in the development set and $||w||^2 = \sum w_j^2 = wTw$. The first element of $J(w, b)$ has to fit the best as possible the model and the second element, making w as low as possible. This will make the network generalize better by simplifying the model as much as possible while trying to fit the training data.

4.2.2 Padding and Pooling

While working with CNN there are a couple of concepts that we should be familiar with:

Padding

As we said before, the larger amount of hidden layers, the deeper the network will become, but, this can also imply that images can be reduced too much by convolutional layers and if the system has many layers, the image can become too small. So, the information provided by the pixels in the corners and edges is underrepresented. To solve this situation, we introduce an additional layer in the corner of the image, this technique is called *Padding*.

For example, if we have a grid of dimension $5x5$, there are only 9 different ways of getting a $3x3$ grid. As the output is going to be $3x3$, it will get shrunk in $2x2$. To solve this loss information, one more layer is added in the corners of the image in order to obtain the same dimension for the input (in this case $5x5$) and for the output ($3x3$ in the example). So, this new layer has the appropriate number of columns and rows making possible to fit every input. Continuing our example, we can see another for a $7x7$ in the following picture:

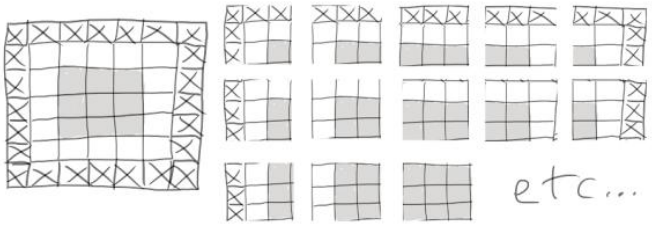


Figure 4.5: Procedure of padding. [3]

Padding is done until input and output are the same size. Then, the number of layers added, p, is obtained by:

$$\text{image } x \text{ filter} \rightarrow nxn * fxf$$
$$(n + 2p - f + 1) \text{ x } (n + 2p - f + 1)$$
$$n + 2p - f + 1 = n$$

where f is the size of the filters and nxn the dimension of the matrix. Then:

$$p = \frac{f - 1}{2} \tag{4.1}$$

Size of the filters f is usually odd, allowing us to have a middlemost pixel. If there is no padding done, it will get as:

$$(n - f + 1)x(n - f + 1) \text{ image}$$

Another solution is *strided convolution*, where instead of shifting the filter in steps of one pixel, we do it in steps of *s* pixels. Following the same mathematical operation as padding:

$$[\frac{n+2p-f}{s} + 1]x[\frac{n+2p-f}{s} + 1]$$

We have to note that if $\frac{n+2p-f}{s}$ is not an integer, it is rounded to the nearest integer below.

Pooling

As with each convolution we are having a hidden layer with more neurons than the previous one, we need more performance, so, in order to reduce the dimensionality of the feature map without losing the most important characteristics, pooling layers are commonly used. This consists of a function that makes the representation invariant to small translations of the input, so it doesn't change the values of most of the pooled outputs when a translation is applied.

Pooling has no parameters to learn. We can see a diagram of how pooling works in Figure 4.6:

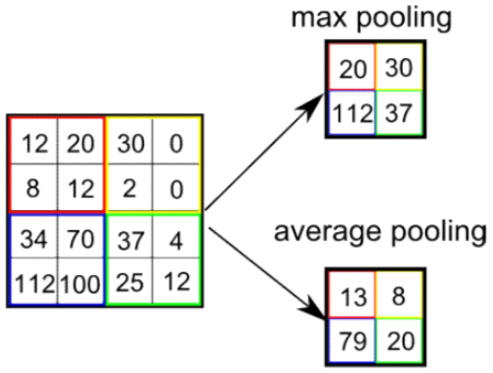


Figure 4.6: Scheme of the operation of pooling. [2]

4.3 Models

There are different architectures of neural networks relying on information treated. Here, we focus on the convolutional neural networks in the field of imaging processing, classification and object recognition. Let's explain the most common ones used in this project:

[2] obtained in *Medium* in Deep Dive into Convolutional Neural Networks: https://medium.com/

- Xception model is 71 layers deep where spatial correlations and the ones existing between channels are not mapped together because they are free enough. Xception maps separately using an operation known as *depthwise separable convolution* that consists of doing independently one spatial convolution for each channel, so, first, it searches for spatial correlations two-dimensional and then, for one-dimensional ones.

- VGG architecture follows common CNN model: some convolutions, activation layers, max pooling for reducing volume size and some classification layers (softmax) at the end. This model uses 3x3 convolutional layers one of top of another. VGG16 model basically uses 16 weight layers in the network and VGG19 employ 3 more layers.

- ResNet50 model is a subclass of ResNet models that employs 50 layers. This model is known for using a skip connection that allows the input to jump overweight layers that the network finds less relevant and goes through the identity function, this offsets vanishing gradient by introducing additional layers.

4.4 State of art

In the last years, since the development of Artificial Intelligence became part of the technological schedule, it has been moved to many research fields in order to improve technology invaded by the massive amount of data. Nowadays, the development of Deep Learning is constantly growing in many fields: particle physics, cybersecurity, speech recognition, search engine, medicine...

In particle physics, for example, convolutional neural networks are used to analyze particle collisions at LHC [7] by classifying images of the events; or to reconstruct the trace of the particles since the quantity of traces and collisions became impossible to treat with current methods [10].

For cybersecurity, Deep Learning is still in development, but it performs very well on detecting intruders in network attacks [23] by using convolutional neural networks combined with autoencoders. Also, Deep Learning is used to find threats [21] by using unsupervised models in order to check anomalies.

It is clear that Deep Learning is becoming an important branch of research in many fields, in this project, we focus on the study of the application of Deep Learning in medicine, which has become one of the most important fields where Deep Learning and AI, is being developed. For instance, Deep Learning models are successfully working for the classification of medical images for diagnosis purposes and for classification of DNA sequences. This kind of algorithm can even lead to the discovery of new treatments and new prevention methods. Our work will be focused on medical diagnosis using images.

Chapter 5

Diagnostic imaging

Diagnosis in medical imaging consists of the visualization of the internal structure of the human body through different techniques, where an anatomic image is obtained for medical diagnosis and research. These techniques can be developed by using X-Ray, ultrasonography, radio-isotopic emissions and magnetic resonance.

A medical image is a human body representation taken by a set of techniques and determined processes, whose principal goal is the diagnosis of illness or the study of human anatomy. In this project, we are focus on medical images obtained by X-Ray techniques that are, usually, taken using projectional radiography or by computed tomography scan (also called CT scan).

As structural elements with different attenuation coefficients compose the body region to be studied, the image of a projectional radiography appears as a two-dimensional projection of a three-dimensional object, similar to a shadow of the bones with a grayscale representation [8].

Dispersion and absorption attenuate the photon beam of X-Rays that spread linearly along different lineal paths that continue through the object, depending on the thickness, mass density and region composition. Emergent X-Ray beam provides information in the form of flux intensity variations in a transversal slice of the beam.

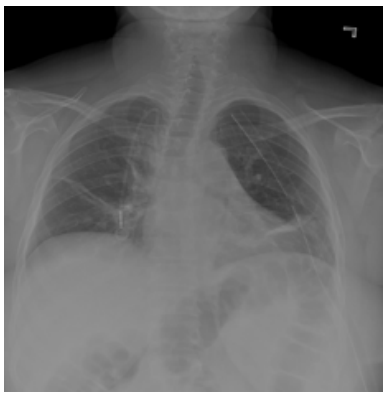


Figure 5.1: Projectional radiography of lungs from the dataset.

In the case of the CT scan, this is an imaging technology that generates 2D anatomical images by using X-Ray radiation taken from different angles in order to get cross-sectional images. CT image is a map of the spatial distribution of calculated attenuation coefficients of radiological attenuation, thus, it can be read as a chart of mass densities of the tissues.

5.1 Medical Images format

Every image, as we said before, is made of pixels forming a rectangular matrix. Medical images are stored, principally, in two different formats: NIFTI and DICOM. The difference between both depends on the way they are used.

5.1.1 NIFTI

NIFTI (Neuroimaging Informatics Technology Initiative [11]) is a standard representation of images that it is commonly used as an analytic file. These images are 3-D arrays of numbers, so they form a 3-D image of the whole part of the body and they are saved in *.nii* format. NIFTI images don't contain medical information about the patient or the hospital, but they contain the imaging metadata like, for example, pixel dimension.

Even though NIFTI images are easier to analyze than DICOM ones, the inconvenience of NIFTI format is that images obtained directly from the MRI instruments are DICOM, so it is necessary to convert these to NIFTI.

5.1.2 DICOM

DICOM (from Digital Imaging and Communications in Medicine [5, 15]) is a standard representation of medical images and the format of files obtained from a scanner or a hospital archive. Moreover, it is the recognized standard for the exchange of medical images worldwide, that allows to handle, store and visualize them. It enables medical imaging information to be operated by two o more systems in order to exchange information and use it.

DICOM allows an unequivocal identification of the objects due to each image having a unique identifier (UID), making medical images information interoperable and, moreover, it is constantly updated. Thus, there is no way to have two UID identical displaying different information and also, it is not possible to separate the image of this information.

Its format extension is *.dcm* and it represents one medical image. Unlike NIFTI images, DICOM ones contain medical information about the patient and the hospital, and also about the scanning parameters used to obtain the image and its properties. So, a DICOM image has two principal components:

- IOD (Information Object Definition) object. This is made up of the image and IEs (Information Entities) that is its associated information, about the patient, equipment, image...
- Image(s) and image data.

One more thing is the grayscale image, in order to different monitors or print systems have identical grayscale, the DICOM grayscale standard display function (GSDF) has been developed to show digitally assigned pixel values.

It is important to notice that due to *The General Data Protection Regulation (EU) (GDPR)* [6] all information stored in these DICOM files must be completely anonymous.

5.2 State of Art of Deep Learning on medical data

Since Deep Learning has shown its efficiency as a solution to handle huge quantities of data, many experts have applied this knowledge and methods to medical research. There are many examples where we can find Deep Learning in medical research:

In patient categorization, revealing new classes of treatable conditions by providing a meaningful approach [2].

Also, in electronic health record (EHR) [16], Deep Learning algorithms can analyze parts of free text from pathology reports in order to identify tumours' primary site and their laterality.

This project, Deep Learning in medical images, is focused on the treatment of medical images. For this part of medical research, there are many studies of the use of this technology in medical diagnoses, such as the study of the stages in schizophrenia [17] or different chest pathologies using radiographs [14].

Chapter 6

The data

6.1 DataSet

At the end of 2019, Covid-19 disease, produced by SARS-CoV-2 virus, was first detected in the city of Wuhan in China. During the next few months, the number of patients experimented a huge increase, leading to the World Health Organization in March of 2020 to declare an outbreak of Covid-19 a global pandemic, where nearly all countries started taking different measurements.

In order to avoid the spread of the virus and so, new infections, doctors prefer the use of projectional radiographs to detect pneumonia in patients' lungs, instead of using other methods such as TACs. Projectional radiographs allow keeping a security distance between the X-Ray machine and the patients, situation that is not repeated in TACs or MRI, where patients are introduced in the machine, making it more difficult to carry out prevention and hygiene tasks.

Health systems began to collapse, prompting hospitals to take steps to speed up, among other things, diagnostic processes. Thousands of medical images, most of them chest X-rays, were taking every day. It was necessary to expedite the diagnosis of pneumonia.

Our dataset is composed of 17374 lung images, 6507 diagnosed with pneumonia.

6.2 Preprocess

We have a binary classification net with 17374 images divided in training set and validation set:

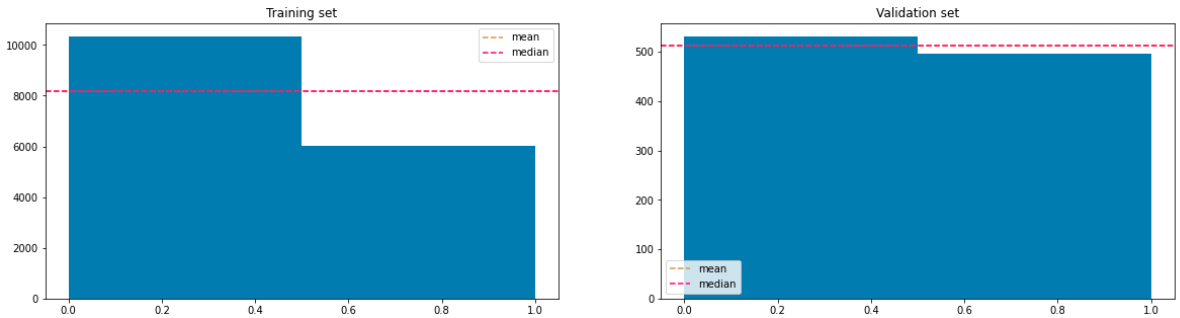


Figure 6.1: On the right, training set and, on the left, validation set.

For the training set, we have 17374 images, where 6012 are positives and 10336 are negatives. In the case of the validation set, we have 1026 images, 495 positives and 531 negatives.

As we can observe in Figure 6.1, our dataset is imbalanced between positive and negative cases, especially in the training set. With the aim of avoiding the net to over-learn from negative cases harming positive class which is the minority one, it is possible to compensate it by using a parameter that allows the net to balance the training data in order to avoid over-training in specific cases of the dataset. It consists of adjusting metrics in the algorithm for the purpose of balance the minority class by putting more emphasis on it by penalizing the majority class.

6.2.1 From DICOM to Python

Images from the dataset are in DICOM format. Image information is saved in structures as vectors and matrix, being numpy format the chosen solution to treat this information. So, to extract the image, we have developed a script that converts dicom data to numpy array.

6.2.2 Normalization of the input

First step running the net is normalization RGB (red, blue, green) of the input by calculating the mean of colours, in order to avoid colour changes due to lighting. This process consists of dividing each value of the pixel by the sum of all of them on all channels. That is, if there is a pixel with R, G and B intensities [20]:

$$(R', G', B') = \left(\frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B}\right) \tag{6.1}$$

Since we have black and white images, they have only one channel so, in order to use the pre-trained net, the black and white channel is repeated three times.

6.2.3 Weights initialization

As it is previously explained, CNNs follow a hierarchy where first layers focus on extracting lines and edges, that is, low-level features that are common in all images. Through the so-called *Transfer learning* technique, it is possible to use a neural network previously trained, keep the first layers frozen and only let the last layer weights vary during the training process.

We have then used a pre-trained model, trained on the ImageNet dataset (with 14.000.000 images with labels prepared for learning), where optimal weights are calculated for general image classification. Then, we have taken these weights as a starting point, frozen the first layers, and retrained only the last ones. This transfer learning method is known as fine-tuning [12] since it does not train the whole neural network from scratch, but only fine-tunes the relevant high-level features associated with the problem to be solved. It saves time and computer resources, allowing us to skip a big part of the training.

6.2.4 Mini-Batch Gradient Descent

We divide the training dataset into small batches since CNNs do not handle the whole data at once. This mini-batch allows our network to calculate a specific gradient descent update for every mini-batch, instead of doing it only once every epoch. The number of epochs is the amount of times that the network reads the full training set. The batch size is usually chosen to be a power of 2 to assist with GPU memory allocation.

Our neural network configuration uses a batch size of 16 (2^4). It is necessary to differentiate the batch from the epoch.

6.3 Data Augmentation

So as to increase training data, using meaningful transformations in existing data, we can obtain new training samples with the appropriate labels (we already have them). So, for us, it is the same image but flipped or with different illumination or even with some black patches, but for the network, it is, to some extent, a new image. One example from the dataset is:

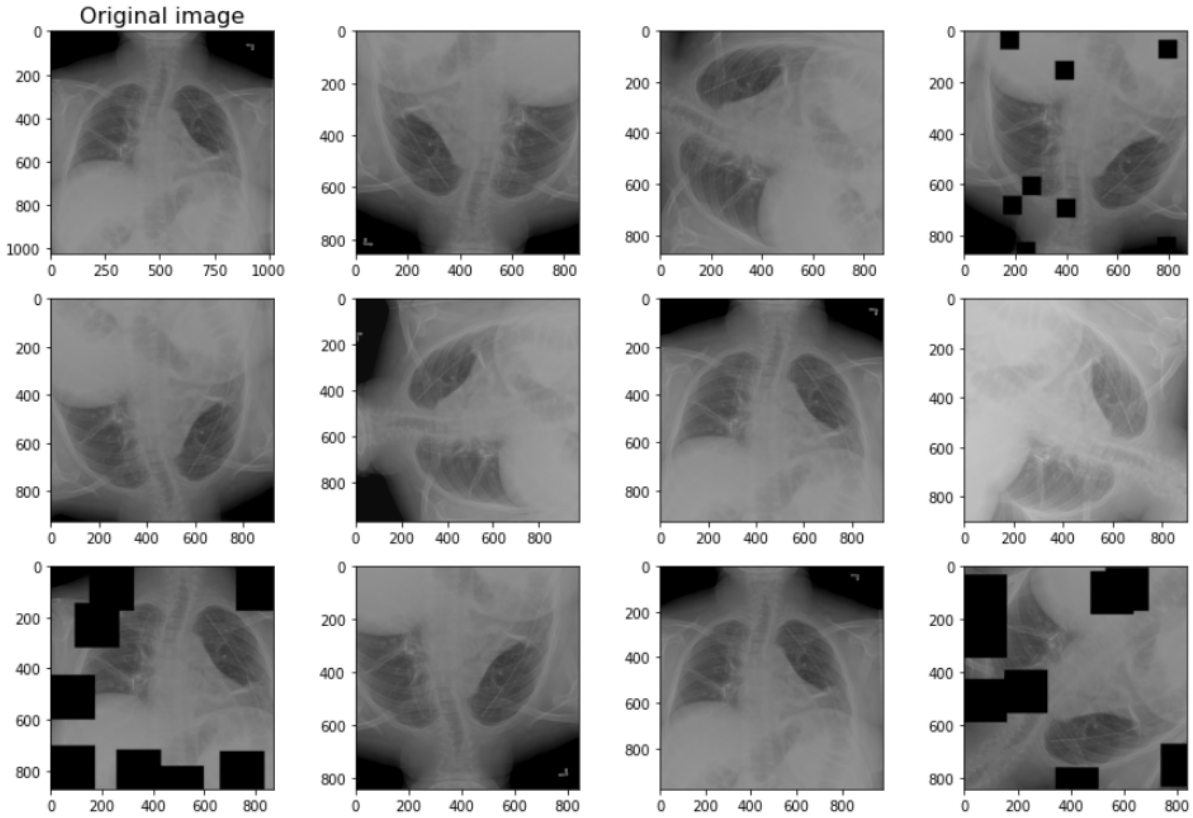


Figure 6.2: Data Augmentation on one image from the dataset.

Also, the use of black patches can allow the network to learn not focusing on specific characteristics.

Chapter 7

Results

7.1 Computing environment

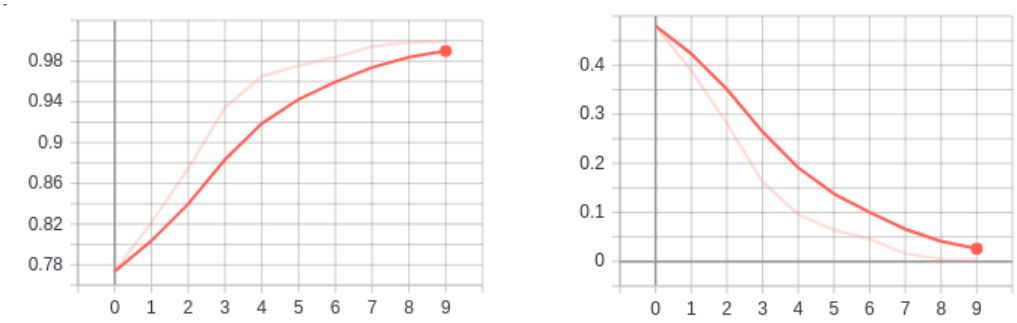
The network is trained using a Graphic Design Unit (GPU) and a docker container. These containers are very similar to virtual machines, but are better at handling the computing resources. This allowed creating a closed environment with all the packages and software needed to run a neural network without having to worry about dependencies, or local software installation.

7.2 Why Xception?

Our model has been trained with and without early stopping. This is a regularization method that avoids overfitting in CNNs by ending the training when the validation metrics start being considerably worse than the training metrics.

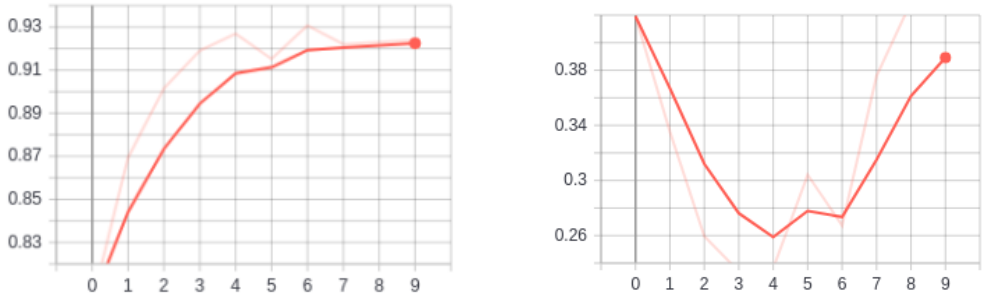
In order to find the most accurate model, we have tested the training of the net with different models: Xception without early stopping, Xception, VGG16, VGG19 and ResNet50 using early stopping, resulting in the following results represented in graphs, where we can observe two curves: one no-smoothed (light) and other smoothed (dark) obtained using the exponential moving average to analyze more easily the trend that follows. The latter is taken as a reference.

- Training using Xception with 10 epochs and without using early stopping:



(a) (b)

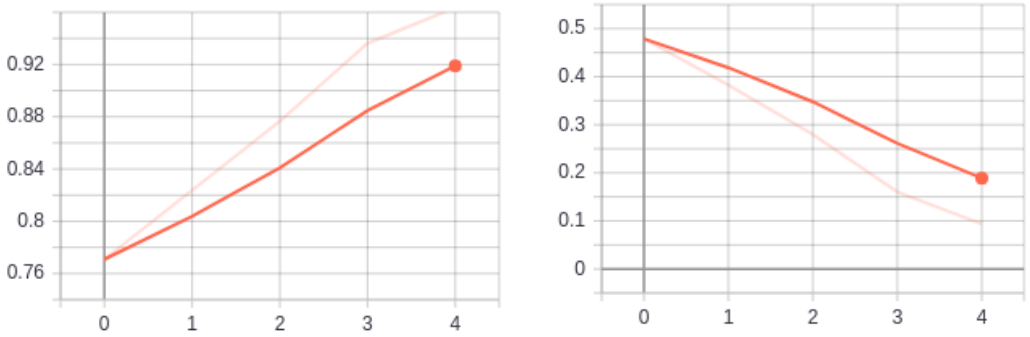
Figure 7.1: Accuracy (a) and loss function (b) per epoch using Xception model.



(a) (b)

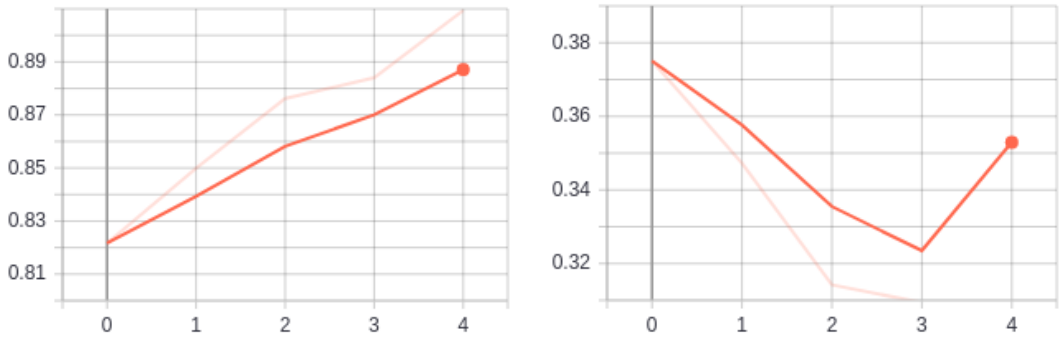
Figure 7.2: Validation accuracy (a) and validation loss (b) per epoch using Xception.

- Training using Xception with early stopping:



(a) (b)

Figure 7.3: Accuracy (a) and Loss Function (b) per epoch using Xception model with early stopping.



(a) (b)

Figure 7.4: Validation accuracy (a) and validation loss (b) per epoch using Xception model with early stopping.

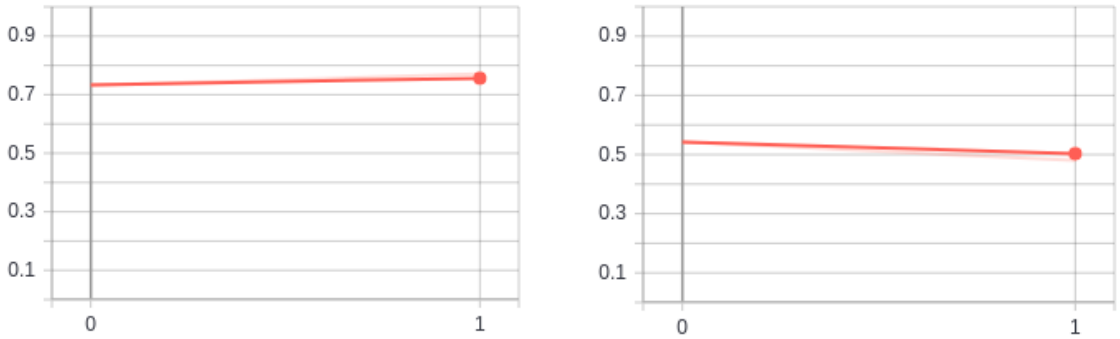
From these results, we can conclude that our net achieves overfitting at the fifth epoch. In Figure 7.1a and 7.2a, where training accuracy and validation accuracy are represented respectively, as early stopping is not used, we observe that the training curve is better than

validation accuracy in epoch 4, where the model is not generalizing new data correctly anymore since it is focusing too much to training data, that is why in Figures 7.3 and 7.4, where early stopping is used, the last epoch is 4.

On the other hand, we can compare both accuracy values: when using early stopping this value rounds 0.92, but without early stopping it is more than 0.98. It does not mean that not using early stopping is necessarily better, it is due to overfitting, because the network is performing too well training data, leading to high error rates on new data.

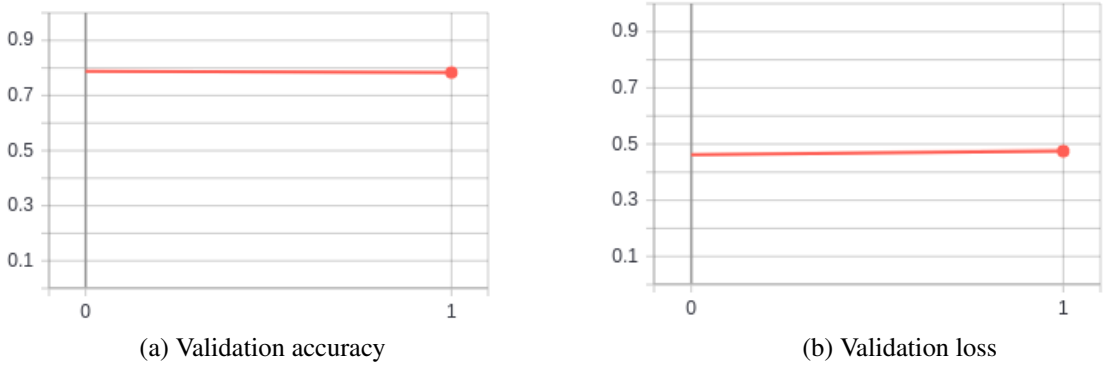
We can also observe the effects of overfitting when using Xception without early stopping: while loss function in Figure 7.1b is decreasing, validation loss in Figure 7.2b is highly increasing, and it is happening the same when using early stopping, in Figures 7.3b and 7.4b, it in this case, since early stopping is used, by the time overfitting is detected, training stops and that the last epoch is the fifth.

- Training using VGG16 model with early stopping:



(a) Accuracy (b) Loss Function

Figure 7.5: Accuracy (a) and Loss Function (b) per epoch using VGG16 model with early stopping.

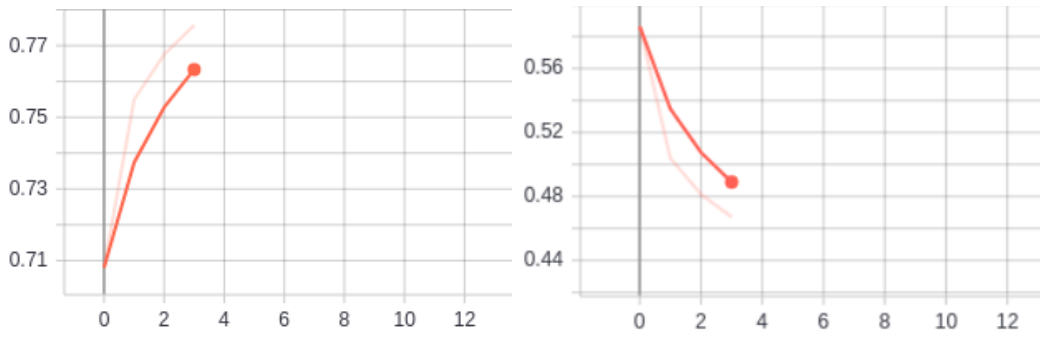


(a) Validation accuracy (b) Validation loss

Figure 7.6: Validation accuracy (a) and validation loss (b) per epoch using VGG16 model with early stopping.

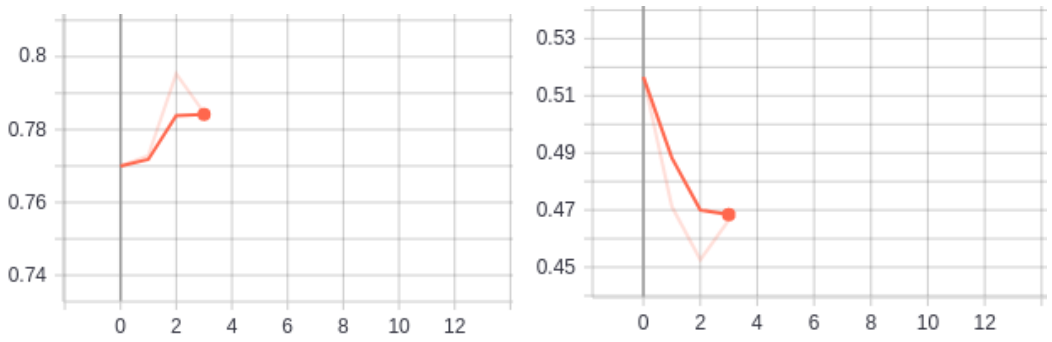
Using VGG16, the maximum accuracy obtained is close to 0.8 both for training and validation, less than the 0.91 achieved by Xception model with early stopping. At the same time, loss function in Figures 7.5b and 7.6b are higher than ones obtained in Figures 7.3b and 7.4b.

- Training using VGG19 with early stopping, we obtained the following results:



(a) Accuracy (b) Loss Function

Figure 7.7: Accuracy (a) and Loss Function (b) per epoch using VGG19 model with early stopping.

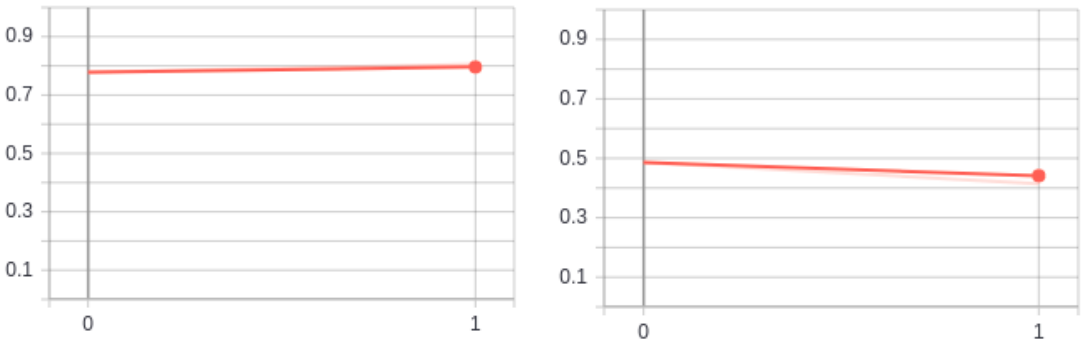


(a) Validation accuracy (b) Validation loss

Figure 7.8: Validation accuracy (a) and validation loss (b) per epoch using VGG19 model with early stopping.

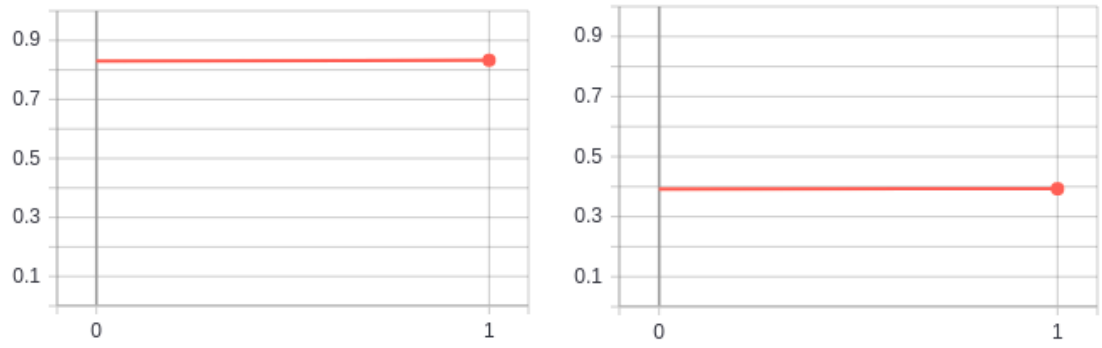
The value obtained for the of accuracy is close to 0.77 for training and 0.78 for validation, even less than the 0.8 achieved by VGG16 model and then also, by Xception model.

- Training using ResNet50 with early stopping, we obtained the following results:



(a) Accuracy (b) Loss Function

Figure 7.9: Accuracy and Loss Function per epoch using ResNet50 model with early stopping.



(a) Validation accuracy (b) Validation loss

Figure 7.10: Validation accuracy (a) and validation loss (b) per epoch using ResNet50 model with early stopping.

Using ResNet50, it starts overfitting before the third epoch. From the results, one can see that the maximum accuracy obtained is a bit higher than 0.80, both for training and validation, better than the one with VGG19, but it is still less than the 0.91 from the Xception model. We can conclude that the best option, in this case, is to use Xception model.

7.3 Results

After comparing several architectures and checking that Xception was the one giving the most promising results, we have decided to use this architecture for the final results. We show here the results with Xception, tuning the network hyperparameters to optimize the results. Our net has been trained using early stopping and class-weights balance, obtaining the following results:

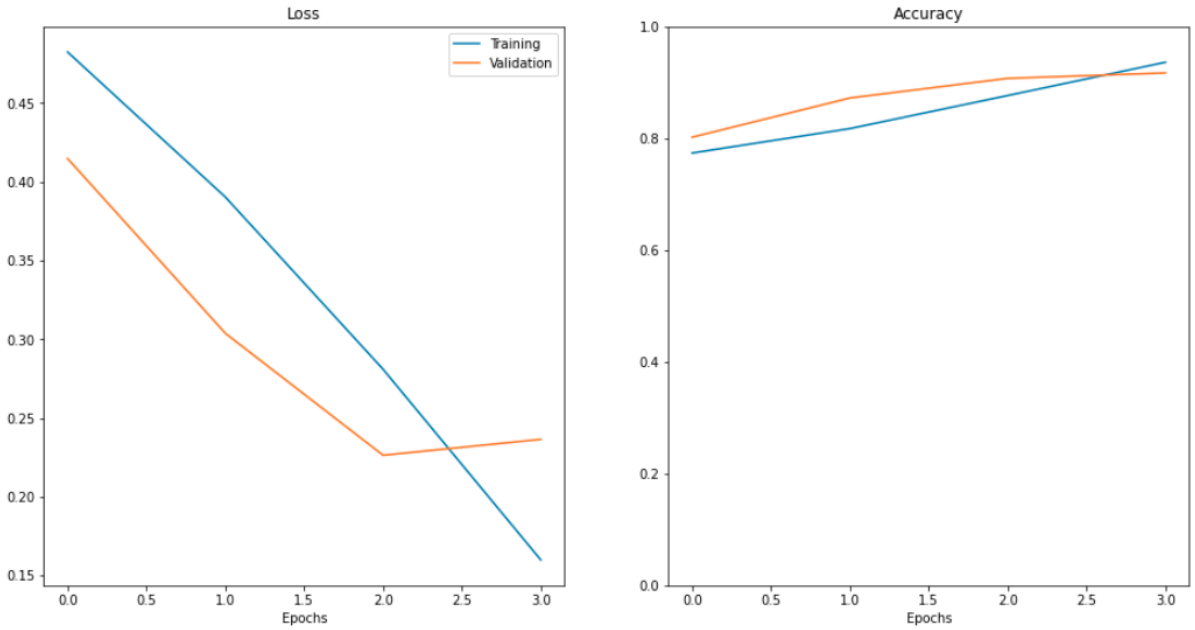


Figure 7.11: On the left, loss function and, on the right, accuracy per epoch using Xception model. In blue, training's curve and in orange, validation's curve.

From Figure 7.11, from the crossing of both curves, training and validation, we can observe that overfitting is achieved before the fourth epoch, it concretely starts once the third epoch is completed, where the validation loss curve increases and intersects with training loss in the halfway between the third and fourth epochs. Same situation is repeated with the accuracy, on the right of 7.11, where the validation curve starts to decrease from epoch 2 (the third one) and crosses over the training accuracy curve in the halfway between the third and the fourth epoch.

7.4 Metrics

Here are the metrics from the results obtained from the training:

7.4.1 Confusion matrix

Confusion matrix is a great tool in order to evaluate the network performance with supervised learning. Each row represents instances in real class and columns show each class prediction numbers. Confusion matrix for this project is illustrated in the following figure:

- True Negative refers to properly predicted labels with no pneumonia (that is *nopato* in the matrix).
- False Positive means incorrect answer for pneumonia.
- False Negative also stands for wrong guessed.
- True Positive is the correct prediction for pneumonia.

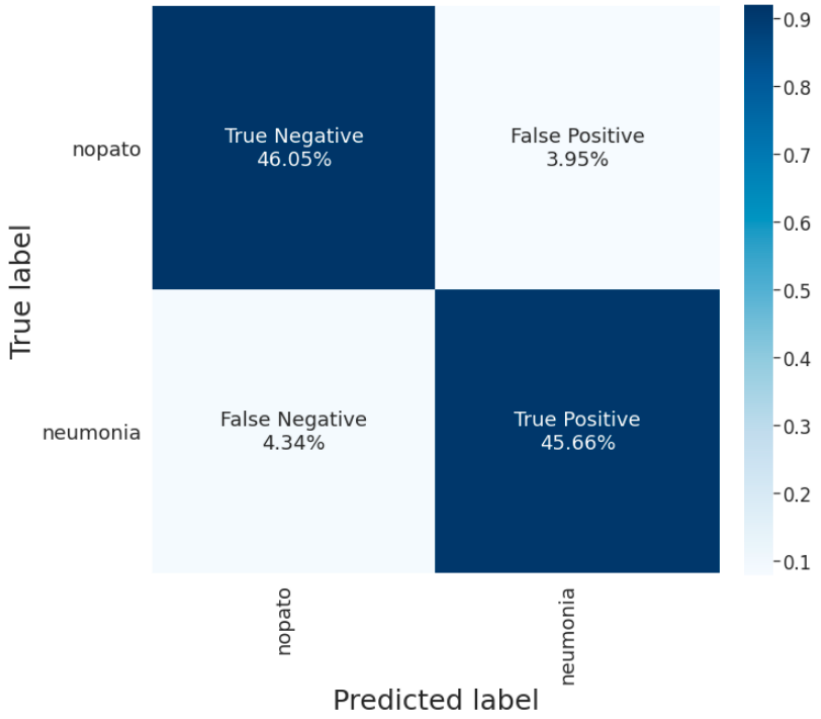


Figure 7.12: Standard confusion matrix with the results obtained for our network.

As we can see, the 45.66% is correctly labelled as pneumonia, this means that 91.32% of the images labelled with pneumonia are properly identified. On the other hand, the algorithm does not so well lungs with no pneumonia predictions with a 46.05%, that is, 92.10% of no pneumonia cases.

7.4.2 Accuracy

The accuracy can be defined as the success rate for the predicted label [18]:

$$\text{accuracy} = \frac{\text{TrueNegative} + \text{TruePositive}}{\text{TrueNegative} + \text{TruePositives} + \text{FalseNegative} + \text{FalsePositive}} \tag{7.1}$$

Here, the value obtained for the accuracy is 91.7%.

7.4.3 Recall or sensitivity

Recall or sensitivity is the ratio of true positives between all true positives and false negatives, that is, how good the model detects each class. So, it is the network capacity to obtain all the positive samples. It can be defined as:

$$\text{recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}} \tag{7.2}$$

For the network, we have obtained a value of 91.7 %.

7.4.4 Precision

Precision is the ratio of true positives between all positives (both false an true), so it tells us how reliable is the model classifying an instance in a class:

$$\text{precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \tag{7.3}$$

For the network, we have obtained a value of 91.7 %.

7.4.5 F1 score

F1 score is an statistical measure of the accuracy in a binary classification, being 1 its best value and 0 the worst. It is determined by calculating the weighted average of the precision and recall:

$$\text{F1 score} = 2 \cdot \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right) \tag{7.4}$$

For the network, we have obtained a value of 0.9.

As the precision and the recall have both high values, we can conclude that our model performs well for the classification. If instead of having precision and recall with high values, we had a

low recall and high precision, the model wouldn't detect correctly the classes, but when it does, it is correct. On the other hand, if the recall is high but precision is low, classes are correctly detected but it includes instances from other classes. Of course, the worst case would be low recall and precision, which means the model does not achieve a correct classification.

If we haven't used the class weights parameter to balance the dataset, probably we would have obtained a model with high precision but a low recall. Since we have used this option, we obtained both values high.

7.4.6 Comparison with the current state of the art on pneumonia using deep learning

Deep Learning techniques have also been studied for pneumonia detection in other researches, such as [13], published in 19^{th} of March of 2020, which distinguishes pneumonia on chest CT in order to detect COVID-19. Comparing the results obtained in this article with our results, we have obtained a recall of 91.7% compared to the 90% of the study, and for the precision, 91.7% versus the 96% of [13].

7.5 Code

Scripts developed to prepare the dataset and to obtain the results are in:

`https://github.com/CarmenGBM/TFG-Neumonia`

Chapter 8

Conclusion and Future

A CNN has been trained for the classification of medical images, concretely, between lungs with pneumonia and healthy lungs, obtaining a result for the accuracy of 91.7%. For this purpose, the variance has been studied in the results of the first training, observing the necessity of using early stopping method in order to avoid overfitting, which has been achieved at the fourth epoch. Also, different models based on fine-tuning techniques have been studied and compared: Xception, VGG16, VGG19 and ResNet50. Once the model which fits the best with our network was found, in this case, Xception model; a more thoughtful training has taken place.

Moreover, the use of the class weighting technique to balance the dataset and data augmentation to increase training data has contributed avoiding overfitting. The result obtained can be improved by using a bigger dataset with more samples of lungs with pneumonia and healthy lungs of different patients covering, preferably, all ages for both sexes and different physiological profiles: smokers, athletes, asthmatics...

Following the growth of Deep Learning and, especially, of CNN in the last years, one possible scenario to work could be saliency maps [19]. This technique is focused on unique image features which are the most relevant to classify an image with a certain label.

This could help in the future to achieve better results and even to find new characteristics not considered as relevant by medical professionals.

Bibliography

[1] ANDREJ KARPATHY. 2016. In: *Youtube*. Standford University. CS231n: Winter 2016. [Online video]. Andrej KARPATHY, Justin JOHNSON and Fei-Fei LI. [Viewed: 04/04/2019]. Available in: https://www.youtube.com/playlist?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC 2.1, 2.2.1

[2] CHING, T. [et al]. 2017. Opportunities and obstacles for deep learning in biology and medicine. In: *Journal of The Royal Society Interface*. Available in: https://doi.org/10.1098/rsif.2017.0387 5.2

[3] CHOLLET, F. 2017. *Deep Learning with Python*. MEAP Edition. Manning Publications. ISBN: 9781617294433. 2.2.1, 3.2, 4.1, 4.2, 4.5

[4] DERTAT, Arden. 2017. Applied Deep Learning - Part 1: Artificial Neural Networks. In: *Medium, towards data science*. August 7, 2017. [Viewed: 13/08/2019]. Available in: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6 4.3

[5] DICOM Standard Committee. Digital imaging and communications in medicine. [Viewed: 01/11/2019]. Available in: https://www.dicomstandard.org 5.1.2

[6] EUROPEAN UNION. Regulation (EU) 2016/679. Complete guide to GDPR compliance. [Viewed: 1/11/2019]. Available in: https://gdpr.eu/ 5.1.2

[7] FERNÁNDEZ MADRAZO, C. [et al]. 2018 Application of a convolutional neural network for image classification to the analysis of collisions in high energy physics. EPJ Web Conf. Available in: https://doi.org/10.1051/epjconf/201921406017 4.4

[8] FLECKESTEIN, P. TRANUM-JENSEN, J. 2001. *Bases anatómicas del diagnóstico por imagen*. pp 20-27. Segunda edición. ISBN: 84-80866-162-2. 5

[9] GOODFELLOW, I. BENGIO, Y. COURVILLE, A. 2016. *Deep learning*. MIT press. ISBN: 9780262337373 3.1.2, 3.1.3, 3.2

[10] Institut de Física Corpuscular, IFIC. 2019. Inteligencia Artificial para reconstruir el rastro que dejan las partículas en el LHC. April 2, 2019. [Viewed: 6/02/2020] Available in: https://webific.ific.uv.es/web/content/inteligencia-artificial-para-reconstruir-el-rastro-que-dejan-las-part%C3%ADculas-en-el-lhc 4.4

[11] Johns Hopkins University. 2018. En *Coursera*. Introduction to Neurohacking In R. [Online video]. Elizabeth SWEENEY, John MUSCHELLI and Ciprian M. CRAINICEANU. Available in: https://www.coursera.org/learn/neurohacking 5.1.1

[12] LERCH, Daniel. 2018. Fine-tuning en reconocimiento de imágenes mediante Deep Learning. in: *Medium*. February 21, 2018. [Viewed: 15/05/2020] Available in: https://medium.com/neuron4/fine-tuning-en-reconocimiento-de-im%C3%A1genes-mediante-deep-learning-c656ae728d73 6.2.3

[13] LIN, L. [et al]. 2020. Artificial Intelligence Distinguishes COVID-19 from Community Acquired Pneumonia on Chest CT. *Radiology*. 200905 (2020). [Viewed: 12/06/2020]. Available in: https://doi.org/10.1148/radiol.2020200905 7.4.6

[14] PARAS, L.; BASKARAN, S. 2017. Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. *Radiology*. 284(2) pp 574-582. Available in: https://doi.org/10.1148/radiol.2017162326 5.2

[15] PIANYKH, O.S. 2008. *Digital Imaging and Communications in Medicine (DICOM): A Practical Introduction and Survival Guide*. pp 3-27. Boston: Springer. ISBN: 9783540745709. 5.1.2

[16] RAJKOMAR, A. [et al]. 2018. Scalable and accurate deep learning with electronic health records. *npj Digital Med* 1(18). Available in: https://doi.org/10.1038/s41746-018-0029-1 5.2

[17] SCHNACK, H.G. [et al]. 2017. Multi-center mri prediction models: Predicting sex and illness course in first episode psychosis patients. *Neuroimage* 145(Pt B) pp 246–253. Available in: https://doi.org/10.1016/j.neuroimage.2016.07.027 5.2

[18] Scikit learn. sklearn.metrics: Metrics. [Viewed: 15/05/2020] Available in: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics 7.4.2

[19] SIMONYAN, K. VEDALDI, A. and ZISSERMAN, A. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In: *arXiv preprint*. Available in: https://arxiv.org/abs/1312.6034 8

[20] SINHA, Utkarsh. Normalized RGB What is normalized RGB?. In: *ALl Shack*. [Viewed: 12/05/2020]. Available in: https://aishack.in/tutorials/normalized-rgb/ 6.2.2

[21] TUOR, A. [et al]. 2017. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. *arXiv preprint*. Available in: https://arxiv.org/abs/1710.00811 4.4

[22] TURING, A.M. 1950. Computing machinery and intelligence. *Mind*. LIX(236), pp 433-460. Available in: https://doi.org/10.1093/mind/LIX.236.433 2

[23] XIN, Y. [et al]. 2018. Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access*. 6, pp. 35365-35381. Available in: https://doi.org/10.1109/ACCESS.2018.2836950 4.4