FINAL THESIS

Bachelor's degree in Biomedical Engineering

LITERARY REVIEW OF ALGORITHMS FOR SEGMENTATION AND CLASSIFICATION OF ARTIFICIAL INTELLIGENCE PATHOLOGIES APPLIED TO BREAST CANCER

Report and Annexes

Author:	Ivan Ortí Fort
Director:	Christian Mata Miquel
Co-Director:	Raul Benítez Iglesias
Call:	June 2022

Abstract

According to data, breast cancer is a significant health issue and has a considerable economic impact. This clearly justifies the need for breast cancer screening. However, the current diagnostic process used in clinical settings is prone to errors. Consequently, there is a requirement for a tool that can help doctors categorize mammograms into the four BI-RADS categories.

This study presents an approach that uses deep learning. It examines the challenges and difficulties encountered and evaluates and compares its effectiveness. One dataset of mammograms was used, with experts having already classified the radiological images using the BI-RADS guidelines. The images in these datasets belong to categories 1 to 4.

The deep learning approach employed in this study is based on a Convolutional Neural Network (CNN), namely a ResNet22. The propose is to use two inputs, one for the Cranio-Caudal (CC) view and another for the Medio-Lateral Oblique (MLO) view. Each input comprises a mammogram image and two heatmaps. Consequently, we have named the architecture MammoHeatNet (MHN).

The algorithm initially processes the mammogram image by cropping it, extracting optimal centers, and obtaining the heatmaps. Once the pre-processing is complete, the inputs are fed into the model, which then classifies them into four BI-RADS categories. To obtain the best model, various parameter configurations have been tested.

The ultimate model attained a maximum accuracy of 74.19%. The process of training and testing the model was time-intensive, requiring 150 hours to obtain the best possible model.

In conclusion, the deep learning model used in this study achieve good performance. However, with the incorporation of a larger dataset for train it and various modifications to the model, even better results could be achieved. The main contribution of this work is the implementation of a deep neuronal network that process the images like a human specialist would do it, using to views of the same mammogram.

Resum

Segons les dades, el càncer de mama és un important problema de salut i té un considerable impacte econòmic. Això justifica clarament la necessitat de realitzar revisions de càncer de mama. No obstant, el procés diagnòstic actual utilitzat en entorns clínics té tendència a cometre errors. En conseqüència, és necessari disposar d'una eina que pugui ajudar els metges a classificar les mamografies en les quatre categories BI-RADS.

Aquest estudi presenta una enfocament que utilitza el "deep learning". S'examinen els desafiaments i dificultats trobades, i s'avalua i compara la seva eficàcia. S'utilitza un conjunt de dades de mamografies, amb experts que ja han classificat les imatges radiològiques utilitzant les directrius BI-RADS. Les imatges d'aquests conjunts de dades pertanyen a les categories 1 a 4.

L'algoritme de "deep learning" utilitzat en aquest estudi es basa en una Xarxa Neuronal Convolucional (CNN), concretament un ResNet22. La proposta és utilitzar dues entrades, una per a la vista Cranio-Caudal (CC) i una altra per a la vista Medio-Lateral Oblique (MLO). Cada entrada comprèn una imatge de mamografia i dues "heatmaps". Per tant, s'ha nomenat a l'arquitectura MammoHeatNet (MHN).

L'algoritme processa inicialment la imatge de mamografia, retallant-la, extraient centres òptims i obtenint les "heatmaps". Una vegada que el pre-processament està complet, les entrades es duen al model, que les classifica en les quatre categories BI-RADS. Per obtenir el millor model, s'han provat diverses configuracions de paràmetres.

El model final assolit va obtenir una precisió màxima del 74.19%. El procés d'entrenament i prova del model va requerir molt de temps, amb un total de 150 hores per obtenir el millor model possible.

En conclusió, el model de "deep learning" utilitzat en aquest estudi aconsegueix un bon rendiment. No obstant, amb la incorporació d'un conjunt de dades més gran per a l'entrenament i diverses modificacions al model, es podrien obtenir resultats encara millors. La principal contribució d'aquest treball és la implementació d'una xarxa neuronal profunda que processa les imatges com ho faria un especialista humà, utilitzant dues vistes de la mateixa mamografia.

Resumen

Según los datos, el cáncer de mama es un problema de salud significativo y tiene un impacto económico considerable. Esto justifica claramente la necesidad de realizar revisiones de cáncer de mama. Sin embargo, el proceso diagnóstico actual utilizado en entornos clínicos tiende a cometer errores. En consecuencia, es necesario disponer de una herramienta que pueda ayudar a los médicos a clasificar las mamografías en las cuatro categorías BI-RADS.

Este estudio presenta un enfoque que utiliza el "deep learning". Se examinan los desafíos y dificultades encontradas, y se evalúa y compara su eficacia. Se utiliza un conjunto de datos de mamografías, con expertos que ya han clasificado las imágenes radiológicas utilizando las directrices BI-RADS. Las imágenes de estos conjuntos de datos pertenecen a las categorías 1 a 4.

El algoritmo de "deep learning" empleado en este estudio se basa en una Red Neuronal Convolucional (CNN), concretamente un ResNet22. La propuesta es utilizar dos entradas, una para la vista Cranio-Caudal (CC) y otra para la vista Medio-Lateral Oblicua (MLO). Cada entrada comprende una imagen de mamografía y dos "heatmaps". Por tanto, se ha nombrado a la arquitectura MammoHeatNet (MHN).

El algoritmo procesa inicialmente la imagen de mamografía, recortándola, extrayendo centros óptimos y obteniendo las "heatmaps". Una vez que el preprocesamiento está completo, las entradas se entran al modelo, que las clasifica en las cuatro categorías BI-RADS. Para obtener el mejor modelo, se han probado varias configuraciones de parámetros.

El modelo final alcanzó una precisión máxima del 74,19%. El proceso de entrenamiento y prueba del modelo requirió mucho tiempo, con un total de 150 horas para obtener el mejor modelo posible.

En conclusión, el modelo de "deep learning" utilizado en este estudio logra un buen rendimiento. Sin embargo, con la incorporación de un conjunto de datos más grande para el entrenamiento y diversas modificaciones al modelo, se podrían obtener resultados aún mejores. La principal contribución de este trabajo es la implementación de una red neuronal profunda que procesa las imágenes como lo haría un especialista humano, utilizando dos vistas de la misma mamografía.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to everyone who has contributed to the completion of this final Bachelor's work. First and foremost, I would like to thank my final bachelor’s advisor, Christian Mata, for their invaluable guidance, support, and patience throughout this entire process.

I would also like to thank my college Daniel Flores at Tecnológico de Monterrey, who have provided me with a very useful insights that helps me to grow and learn. His feedback and suggestions have been essential in refining my work and improving the quality of it. Without his help I would have required much more time to finish this work.

Once again, I would like to express my heartfelt appreciation to these two people who has played a part in making this final Bachelor's work a reality. Thank you for your kindness, support, and encouragement.

Index

1. Introduction

1.1. Breast Cancer

Breast cancer is a common disease in which cells in the breast begin to multiply and grow uncontrollably. Signs of breast cancer may include a change in the breast shape, a lump in the breast, dimpling of the skin, fluid coming from the nipple or a red scaly patch of skin.

Approximately half of breast cancer develop in women who have no identifiable breast cancer risk factor other thank the gender (female) and age (over 40 years). The most important risks factors are: obesity, age, harmful use of alcohol, tobacco and family history of breast cancer.

A breast has three main parts: lobules, ducts, and connective tissue. Ducts have the function of collecting and transporting milk, which is produced by the lobules. Altogether is surrounded and held by connective tissue made primarily by fibrous and fatty tissue.

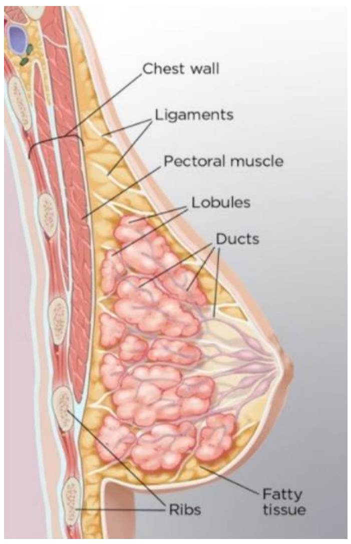


Figure 1.1: Anatomy of the breast [1]

The most common breast cancer is carcinoma, which are tumors that start in the epithelial cells that line organs and tissues throughout the body. If the carcinomas start in the breast is specific type called adenocarcinoma which stars in cells in the ducts (85%) or the lobules (15%) [2].

1.2. Cancer today

Worldwide, there were an estimated 50.3 million cancer cases in 2022. Of these, 9.3 million cases were in men and 8.8 million in women. Breast and lung cancers were the most common cancers worldwide, contributing 12.5% and 12.2%. Colorectal cancer was 10.7% followed by the prostate with a 7.8%.

In 2020, there were 2.3 million women diagnosed with breast cancer and 685000 deaths globally. As of the end of 2020, there were 7.8 million women alive who were diagnosed with breast cancer in the past 5 years, making in the world's most prevalent cancer [3].

It has been revealed that thanks to the breast cancer screening performed in the current clinical practice, mortality from this disease has significantly decreased when done at the age of over 50, which is the one with the highest incidence. In medicine, screening is looking for signs of a disease, such as breast cancer, before somebody has signs of it. The goal line of screening tests is to find cancer at an early phase when it can be treated and might be cured. However, screening is needed and essential since breast cancer is a significant public health concern with considerable medical and economic burden.

In Spain, using the narrow down the 85-95% of the breast cancer is detected if exists [4]. With this narrow down Spain reduce the total cancer costs with 9000 million of euros. A metastatic breast cancer costs 4 times more than a local breast cancer. The costs of a metastatic breast cancer can exceed the 200.000 euros per patient [5].

1.3. The mammography

Mammography is the process of using low-energy X-rays to examine the human breast for diagnosis and screening. The goal of mammography is the early detection of breast cancer, typically through detection of characteristic masses or microcalcifications.

During the procedure, the breast is compressed using a dedicated mammography unit for increase image quality by reducing the thickness of tissue that X-rays must penetrate, decreasing the amount of scattered radiation and reducing the required radiation dose. In screening mammography, both head-to-foot (CC) view and angled side-view (MLO) images of the breast are taken.

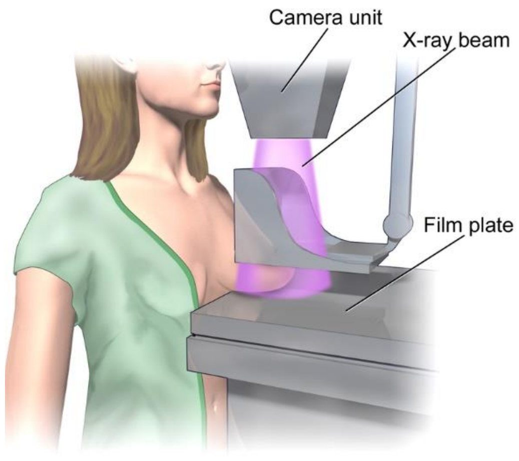


Figure 1.2: Mammography [6]

There are few handicaps affect the variability of the resulting mammographic picture. For example, there is much variation in the breast's granularity, which disturbs the radiographic density and appearance of the mammogram. In general, breast granularity decreases with increasing breast sizes, but again there can be significant differences. Breast abnormalities may appear on the mammogram as a soft tissue lesion that may be rounded or spiculated. However, sometimes the only sign of an anomaly is one or more calcifications or distortion in the breast architecture. Calcifications are crumbs of calcium hydroxyapatite or phosphate, ranging from extremely small to several millimeters. It is considered desirable to detect calcifications as small as 100 µm, which presents a significant challenge to the imaging system.

1.4. Breast Imaging Reporting and Data System (BI-RADS®)

Aiming to reduce the discordance in interpreting mammographic findings and homogenizing the terms for characterization and reporting in a standardized way, the American College of Radiology published, in 1993, the Breast Imaging Reporting and Data (BI-RADS®) [7].

This structured system aims to achieve consistency and reliability between different reports and facilitates clear communication between the radiologist and other medical professionals by providing a lexicon of descriptors. It is a reporting structure that relates assessment categories to management recommendations and a framework for data collection and auditing. The BI-RADS lexicon classifies breast imaging findings into different types:

Table 1.1. BI-RADS categories [8]

BI-RADS 1	No finding is present in the imaging modality (not even a benign finding). Symmetrical and no masses, architectural distortion, or suspicious calcifications
BI-RADS 2	A finding in this category has a 100% chance of being benign. Even though BI-RADS 1 and BI-RADS 2 represent an essentially zero probability of being malignant. BI-RADS 1 is used when the breast is unremarkable; BI-RADS 2 is used when the radiologist wants to highlight a benign finding.
BI-RADS 3	A finding is probably benign, with a shallow risk of malignancy between 0% and 2%. The density of the breast is higher than the previous categories.
BI-RADS 4	Suspicious abnormality. Lesions may not have the typical morphology of breast cancer. However, there is a high chance of malignancy. In these cases, a biopsy is recommended. The breast is very dense. The probability of malignancy is 2-94%.

There are more categories apart from the ones above. For instance, 0 indicates that additional mammograms should be taken since no conclusions can be extracted (moved or wrong taken). BI-RADS category 5 indicates a higher chance of malignancy (>95%), and BI-RADS category 6 represents a biopsy-proven malignancy. Hence, only the four levels in the table above are considered in this project.

1.5. Origin of this project and motivation

The origin of this project can be traced back to previous work from a final degree project from student of Dr. Christian Mata, the project's supervisor. Hence, this study pretends to be a continuity and improvement of a deep learning algorithm. Furthermore, create an actualized state of the art of the deep learning algorithms for the segmentation and classification of breast cancer.

The previous project was developed on Python and used machine learning techniques and deep learning to classify mammograms in the BI-RADS categories using different descriptors. The idea is implementing a new deep learning algorithm customizing an existent code consistent with our interests.

My motivation to do this project is because I like programing and deep learning has always caught my attention. Moreover, when I did the biomedical image processing, I found it very interesting and

I would have liked to go deeper. But in fact, the principal reason to do this project is contribute a little bit in the improvement of the deep learning algorithms to detect efficiently and correctly the breast cancer.

1.6. Objectives

Principal objective

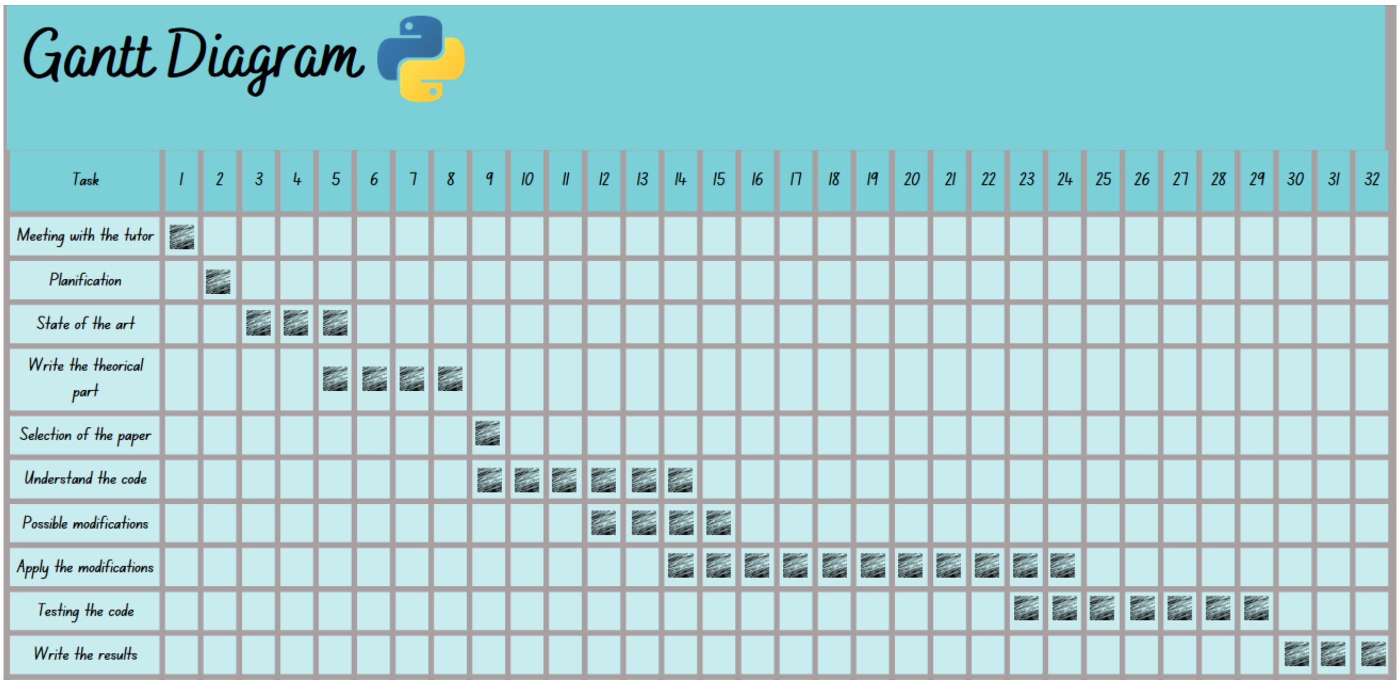
The primary goal of this project is to develop an automated tool that utilizes a deep learning algorithm to classify the four levels of BI-RADS, thereby aiding radiologists in making final decisions. Furthermore, once trained, our proposed system can serve as a reproducible process that any professional can use as a reference for analysing mammograms.

Specific objectives

- Study of the theoretical framework of breast cancer and foundation of medical imaging modality in x-ray.
- Do a literature review of recently published works on this area.
- Select a strategy to be developed in the methodology according to the database we have.
- Implementation of the proposal and discuss the results.
- Discuss which methodology has been the most appropriate and we have obtained better results.
- Limitations and problems.

1.7. Scheduled planning

The following figures aim to draw a roadmap of the project to organize the time and tasks that need to be treated. It is the final version as it was modified during the project.


Gantt Diagram

Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Meeting with the tutor	▨																															
Planification		▨																														
State of the art			▨	▨	▨																											
Write the theorical part					▨	▨	▨	▨																								
Selection of the paper									▨																							
Understand the code									▨	▨	▨	▨	▨	▨																		
Possible modifications												▨	▨	▨	▨																	
Apply the modifications														▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨								
Testing the code																							▨	▨	▨	▨	▨	▨	▨			
Write the results																														▨	▨	▨

Figure 1.3: Gannt diagram

2. Theorical framework

2.1. Deep Learning

2.1.1. What is deep learning?

Deep learning algorithms can be regarded both as a sophisticated evolution of machine learning algorithms. Deep learning algorithms analyze data with a logical structure similar to how a human would draw conclusions. To achieve this, these algorithms use a layered structure of algorithms called an artificial neuronal network (ANN).

The deep learning algorithms the features are extracted automatically and learns from its own errors unlike machine learning that a human is needed to manually choose features. In the **Figure 2.1** we can see clearly this difference.

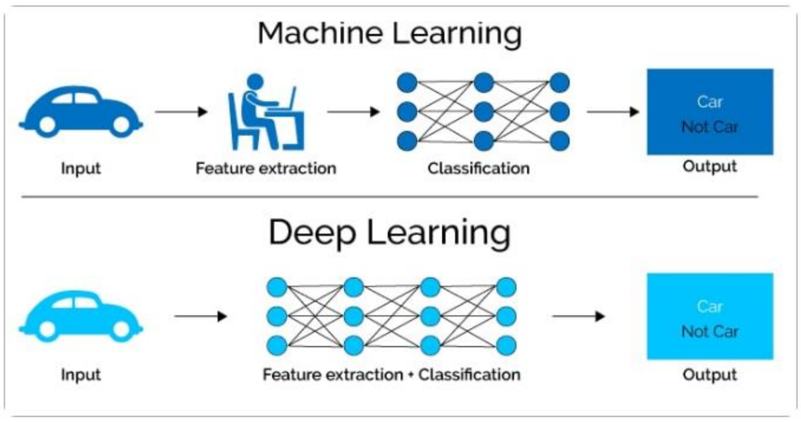


Figure 2.1: Difference between ML and DL [9]

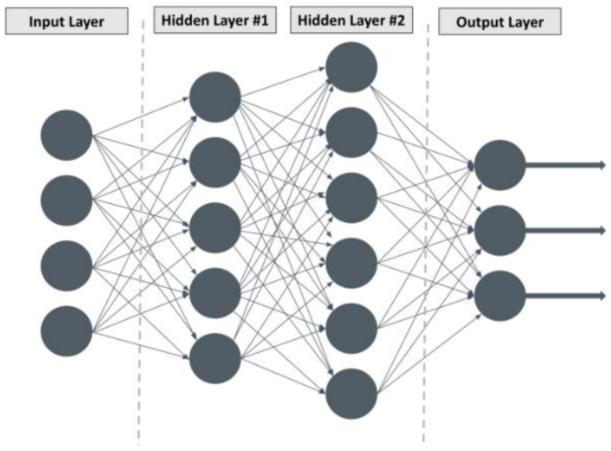


Figure 2.2: Simple ANN [9]

Consider the **Figure 2.2** an example of ANN. The leftmost layer is called the input layer and the rightmost layer is the output layer. The middle layers are called hidden layers because their values aren't observable in the training set. In simple terms, hidden layers are calculated values used by the network to obtain the results in the output layer [9].

2.1.2. How it works deep learning?

How have been seen in the previous chapter the ANN is formed to set of neurons. Let's see how a biological neuron works for translate this to a mathematical function to use it to create a deep learning algorithm.

At its core, the neuron is optimized to receive information from other neurons, process this information in a unique way, and send its result to other cells. Each of these incoming connections is dynamically strengthened or weakened based on how often it is used and its strength of each connection that determines the contribution of the input to the neuron's output. After being weighted by the strength of their respective connections, the inputs are summed together. This sum is then transformed into a new signal that's propagated along to the other neurons.

We can translate this functional understanding of the neurons in out brain into an artificial model that we can represent on our computer how we can see in the **Figure 2.3**.

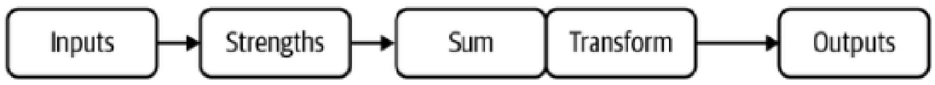


Figure 2.3: Diagram of basic function of a neuron [10]

Observing this diagram, we can create a model of a neuron that we can see in the Figure 3.4. Let's formulate the inputs as a vector x=[x1,x2… xn] and the weights of the neuron (strengths) as a w=[w1,w2…wn]. Then we can express the output of the neuron as y=f((x*w) +b), where b is the bias term. In many cases, the logit also includes a bias, which is a constant (not show in the figure). We can compute the output by performing the dot product of the input and weight vectors, adding in the bias term to produce the logit, and the applying the transformation function. While this seems like a trivial reformulation, thinking about neurons as a series of vector manipulations [10].

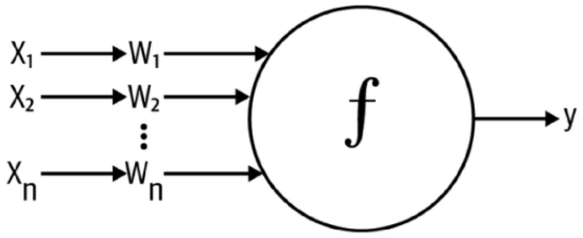


Figure 2.4: Mathematical model of a neuron [10]

In the **Figure 2.5** we can see a model with the bias term. The bias term is important because allows you to shift the activation function by adding a constant to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value [11].

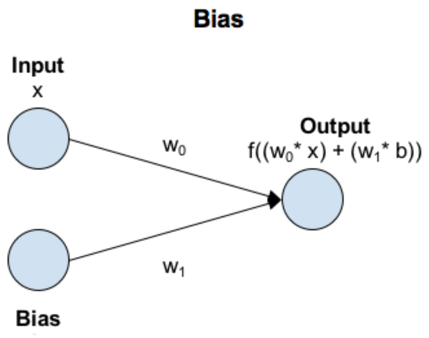


Figure 2.5: Model with the bias term [11]

There are a lot functions that are used to create different neurons, but in our case that is computer vision the best choice is the nonlinearity function named ReLU. It uses the function f (z) = max(0,z), resulting in a characteristic hockey-stick shaped response, as shown in **Figure 2.6** [10].

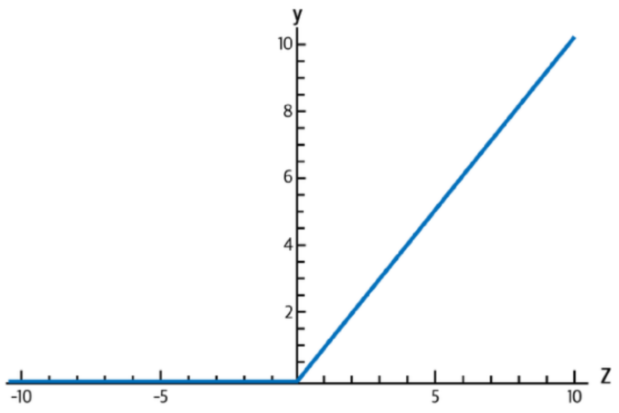


Figure 2.6: The output of a ReLU neuron as z varies [10]

2.2. Convolutional Neural Networks (CNN)

2.2.1. How it works CNN's?

In The traditional Neural Networks is used the called the multilayer perceptron (MLP). These are modeled on the human brain, whereby neurons are simulated by connected nodes are only activated when a certain threshold value is reached.

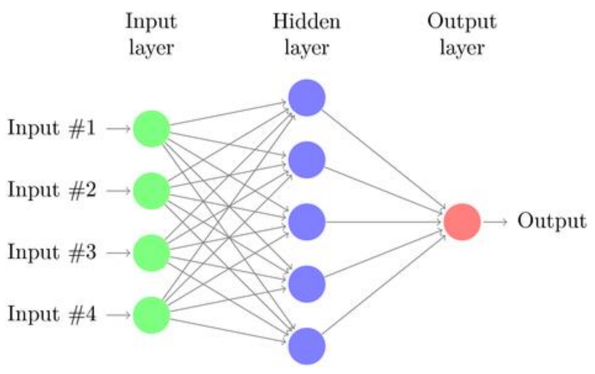


Figure 2.7: A standard multilayer perceptron [13]

There are several drawbacks of MLP's, especially when it comes to image processing. MlPs use one perceptron for each input (in the case of a pixel image, multiplied by 3 in RGB case). The number of weights rapidly becomes unmanageable for large images. For a 250 x 250 pixels image with color channels there are around 190,000 weights that must be trained. As a result, difficulties arise while training and overfitting can occur.

Another problem is that MLP's react differently to an input (images) and its shifted version, they are not translation invariant. For example, if a picture of a cat appears in the top left of the image in one picture and the bottom right of another picture, the MLP will try to correct itself and assume that a cat will always appear in this section of the image.

For these reasons for image processing are not use MLP's and use Convolutional Neural Networks (CNN).

In the case of CNN's, we analyze the influence of nearby pixels by using something called a filter. A filter is exactly what you think it is, in our situation, we take a filter of a size specified by the user (a rule of thumb is 3x3 or 5x5) and we move this across the image from top left to bottom right. For each point on the image, a value is calculated based on the filter using a convolution operation.

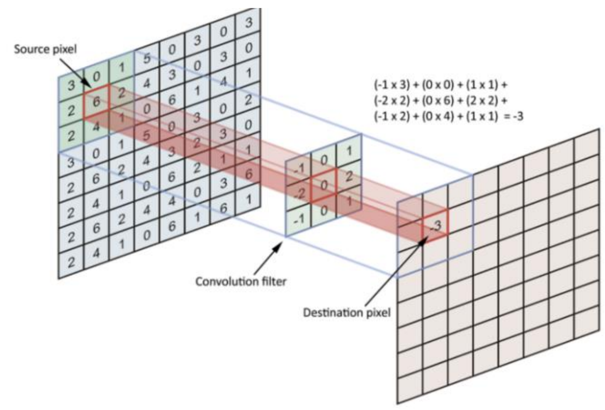

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

Figure 2.8: Convolutional operation [13]

This reduces the number of weights that the neuronal network must learn compared to an MLP, and also means that when the location of these features changes it does not throw the neural network off.

If you are wondering how the different features are learned by the network, and whether it is possible that the network will learn the same features. When building the network, we randomly specify values for the filters, which then continuously update themselves as the network is trained.

After the filters have passed over the image, a feature map is generated for each filter. These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image. We can then do a lot of things, such as adding more filtering layers and creating more feature maps, which become more and more abstract as we create a deeper CNN. We can also use pooling layers in order to select the largest values on the feature maps and use these as inputs to subsequent layers. In theory, any type of operation can be done in pooling layers, but in practice, only max pooling is used because we want to find the outliers, these are when our network sees the feature.

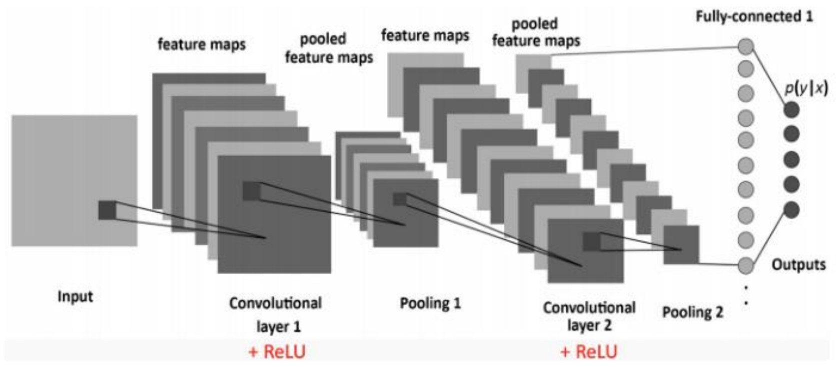


Figure 2.9: An example of CNN with two convolutional layers, two pooling layers, and a fully connected layer which decides the final classification of the image into one several categories [13].

A good question is we have many feature maps, how are these combined in our network to help us get our final result?

To be clear here, each filter is convolved with the entirely of the 3D input cube but generates a 2D feature map.

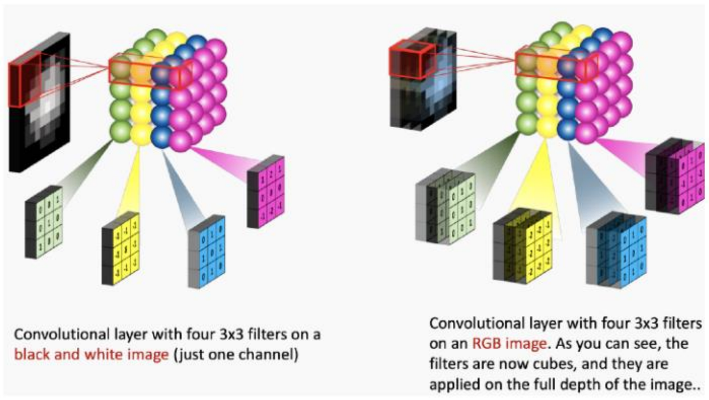

Convolutional layer with four 3x3 filters on a black and white image (just one channel)
Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

Figure 2.10: Using different filters to create more than one feature maps [13]

- Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter.
- The feature map dimension can change drastically from one convolutional layer to the next.
- Convolving the image with a filter produces a feature map that highlights the presence of given feature in the image.

In a convolutional layer, we are basically applying multiple filters at over the image to extract different features. But most importantly, we are learning those filters. One thing we're missing: non-linearity.

For solve the problem of non-linearity we use the ReLu function that explain in a previous section.

In the **Figure 2.11** we can see a standard CNN architecture.

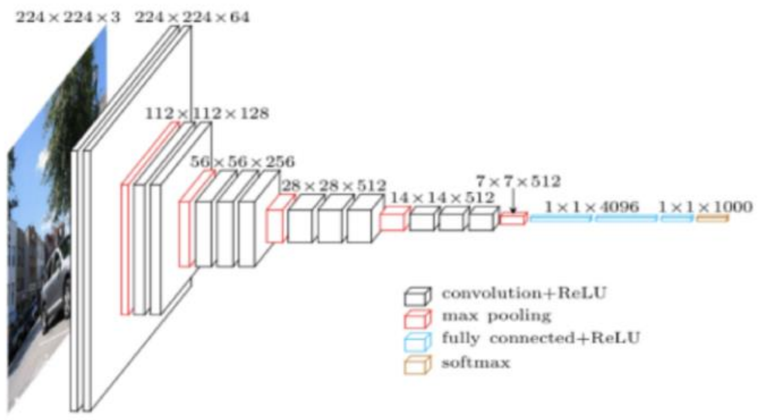


Figure 2.11: The architecture of a standard CNN [13]

One important thing is understood that each CNN layer learns filters of increasing complexity.

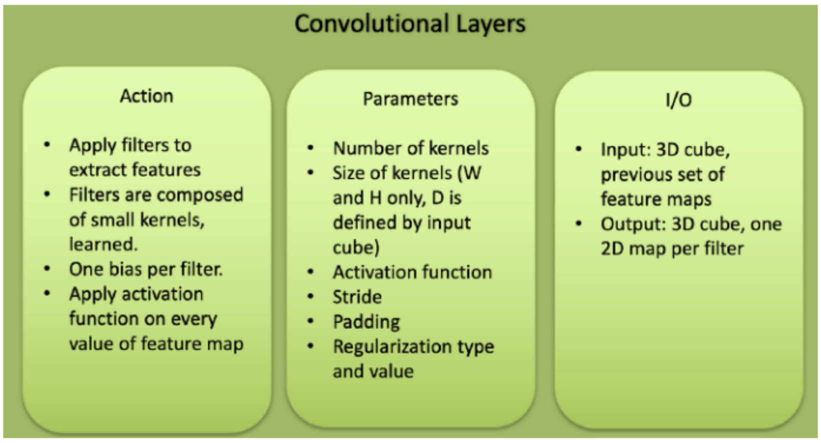
- The first layers learn basic feature detection filters: edges, corners, etc.
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.

2.2.2. Comparison of different layers

There are three types of layers in a CNN: convolutional layer, pooling layer and fully connected layer. Each of these layers has different parameters that can be optimized and performs a different task on the input data.

2.2.2.1. Convolutional layers

Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernel.



Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

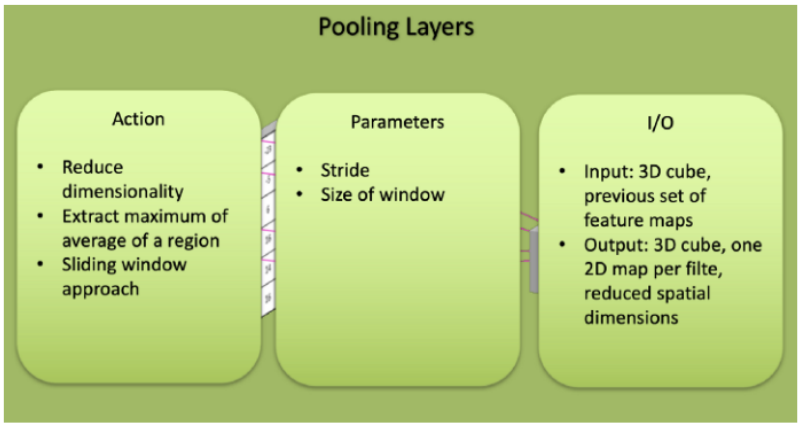
I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Figure 2.12: Features of a convolutional layer [13]

2.2.2.2. Pooling layers

Pooling layers are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region. These are typically used to reduce the dimensionality of the network.



Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

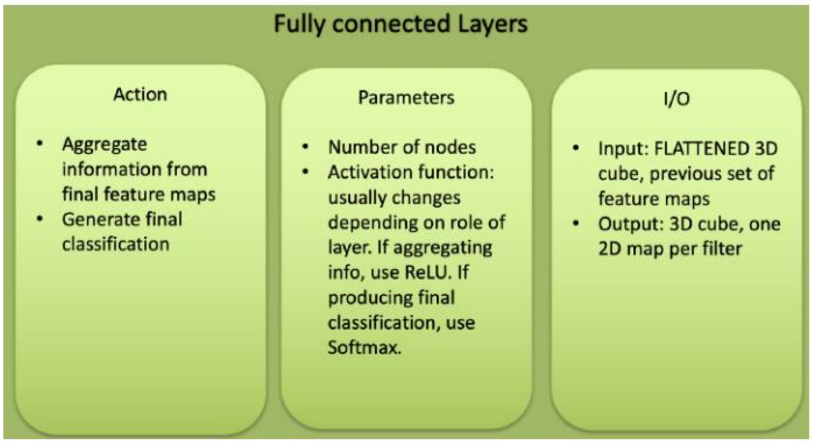
I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Figure 2.13: Features of a pooling layer [13]

2.2.2.3. Fully connected layers

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.



Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

Parameters

- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Figure 2.14: Features of a connected layer [13]

2.2.3. How it learns the deep learning algorithm?

2.2.3.1. Gradient Descent with Sigmoidal Neurons

Surely you are wondering how exactly do we figure out what parameter's vectors (the weights of the connection in out neuronal network) should be? This is accomplished by a process commonly referred to as training. During this process we show the neural net a large number of training examples and iteratively modify the weights to minimize the errors we make on the training examples. After enough examples, we expect that out neural network will be quite effective at solving the task it's been trained too.

The idea is reducing the squared error to zero because this means that our model made a perfectly correct prediction on every training example. Moreover, the closer E is to 0, the better our model is.

For explain how are selected these parameters vectors we are going to use the sigmoidal neuron as a model (this is a different kind of function that is used to create neurons like the function to ReLU neurons). For simplicity, we assume the neurons do not use the bias term [10].

Let's see the mechanism by which logistic neurons compute their output value from their inputs:

$$z = \Sigma\, \mathrm{k\, w\, k\, x\, k}$$

$$y = 1\ 1 + e - z$$

The neuron computes the weighted sum of its inputs, the logit z. It then feeds its logit into the input function to compute y, its final output. We want to compute the gradient of the error function with respect to the weights. To do so, we start by taking the derivative of the logit with respect to the inputs and the weights:

$$\partial z\ \partial w\ k = x\ k$$

$$\partial z\ \partial x\ k = w\ k$$

The derivative of the output with respect to the logit is quite simple if you express in the terms of the output:

$$dy\ dz = e - z\ 1 + e - z\ 2$$

$$= 1\ 1 + e - z\ e - z\ 1 + e - z$$

$$= 1\ 1 + e - z\ 1 - 1\ 1 + e - z$$

$$= y(1 - y)$$

We then use the chain rule to get the derivative of the output with respect each weight:

$$\partial y\ \partial w\ k = dy\ dz\ \partial z\ \partial w\ k = x\ k\ y\ (1 - y)$$

Putting all of this together, we can now compute the derivative of the error function with respect to each weight:

$\partial E\ \partial w\ k = \Sigma\ \mathrm{i}\ \ \partial \mathrm{E}\ \ \partial \mathrm{y}\ (\mathrm{i})\ \partial y\ (i)\ \partial w\ k = \ -\Sigma\ \mathrm{i\ x\ k\ (i)}\ y\ (i)\ 1 - y\ (i)\ t\ (i) - y\ (i)$

Thus, the final rule for modifying the weights becomes:

$\Delta\ w\ k = \Sigma\ \mathrm{i}\ \ \in \mathrm{x\ k\ (i)}\ y\ (i)\ 1 - y\ (i)\ t\ (i) - y(i)$

As you may notice, the new modification rule is just like the delta rule, except with extra multiplicative terms included to account for the logistic component of the sigmoidal neuron.

2.2.3.2. The Backpropagation Algorithm

Now we are finally ready to tackle the problem of training multilayer neuronal networks. To accomplish this task, we will usen and approach known as *backpropagation,* pioneered by David E.Rumelhart, Geoffrey E. Hinton and Ronald J. Williams in 1986.

So, what's the idea behind backpropagation? We don't know what the hidden units ought to be doing, but what we can do is compute how fast the error changes as we change the hidden activity. From there, we can figure out how fast the error changes when we change the weight of and individual connection. Essentially, we will be trying to find the path of steepest descent. The only catch is that we are going to be working in an extremely high-dimensional space. We start by calculating the error derivatives with respect to a single training example.

Each hidden unit can affect many output units. Our strategy will be one of dynamic programing. Once we have the error derivatives for one layer hidden units, we will use them to compute the error derivatives for the activities of the layer below. And once we find the error derivatives for the activities of the hidden units, it is quite easy to get the error derivatives for the weights leading into a hidden unit. We will redefine some notation for ease of discussion and refer to **Figure 2.15**.

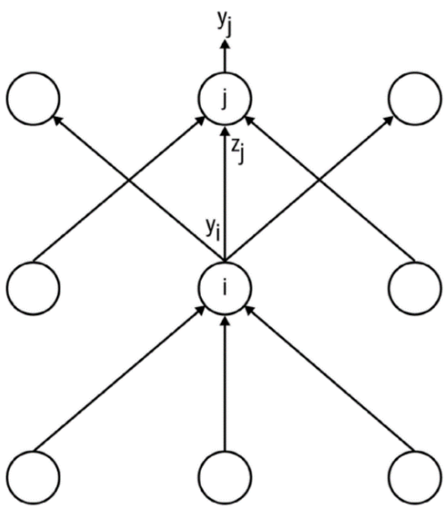


Figure 2.15: Reference diagram for the derivation of the backpropagation algorithm [10]

The subscript we use will refer to the layer of the neuron. The symbol y will refer to the activity of the neuron, as usual. Similarly, the symbol z will refer to the logit of the neuron. We start by taking a look at the base case of the dynamic programming problem.

Now we tackle the inductive step. Let's presume we have the error derivatives for layer j. We next aim to calculate the error derivatives for the layer below it, layer i. To do so, we must accumulate information about how the output of a neuron in layer i affects the logits of every neuron in layer j. This can be done as follows, using the fact that the partial derivative of the logit with respect to the incoming output data from the layer beneath is merely the weight of the connection w ij:

Once we have gone through the whole dynamic programming routine, having filled up the table appropriately with all of our partial derivatives (of the error function with respect to the hidden unit activities), we can then determine how the error changes with respect to the weights.

Finally, to complete the algorithm, just as before, we merely sum up the partial derivatives over all the training examples in our dataset.

This completes our description of the backpropagation algorithm.

2.2.3.3. Stochastic and Minibatch Gradient Descent

For a simple quadratic error surface, the backpropagation algorithm works quite well. The idea of use our entire dataset to compute the error surface and then follow the gradient to take the path of steepest descent. But in most cases, error surface may be a lot more complicated. Let's consider the scenario in **Figure 2.16**

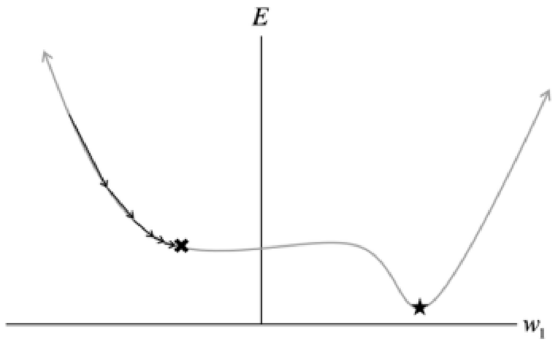


Figure 2.16: Batch gradient descent in sensitive to saddle points, which can lead premature convergence [10]

The error surface has a flat region (also known as saddle point in high-dimensional spaces), and if we get unlucky, we might find ourselves getting stuck while performing gradient descent.

Another potential approach is *stochastic gradient descent* (SGD), where at each iteration, our error surface is estimated with respect to only single example. This approach is illustrated in **Figure 2.17**, where instead of a single static error surface, our error surface is dynamic. As a result, descending on this stochastic surface significantly improves our ability to navigate flat regions [10].

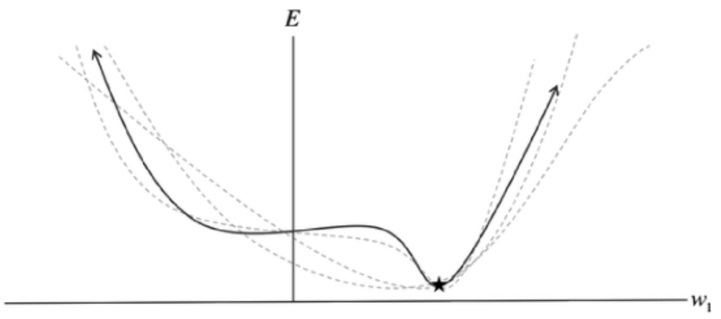


Figure 2.17: The stochastic error surface fluctuates with respect to the batch error surface, enabling saddle point avoidance [10].

The major pitfall of SGD, however, is that looking at the error incurred one example at time takes a significant amount of time. One way to combat this problem is using *minibatch gradient descent*. In minibatch gradient descent, at every iteration we compute the error surface with respect to some subset of the total dataset (instead of just a single example). This subset is called a minibatch, and in addition to the learning rate, minibatch size is another hyperparameter. Minibatches strike a balance between the efficiency of batch gradient descent and the local-minima avoidance afforded by the stochastic gradient descent.

The derived it is the same that we use in the previous section, but instead of summing over all the examples in the dataset, we sum over the examples in the current minibatch.

2.3. Test sets and validation sets

One of the major issues with artificial neural network is that models are quite complicated. For example, consider a neural network that pulls data from an image (28 x 28 pixels), feeds into two hidden layers with 30 neurons, and finally a softmax layer of 10 neurons. The total number of parameters in the network is nearly 25,000. This can be quite problematic, and to understand why, let’s consider the **Figure 2.18**:

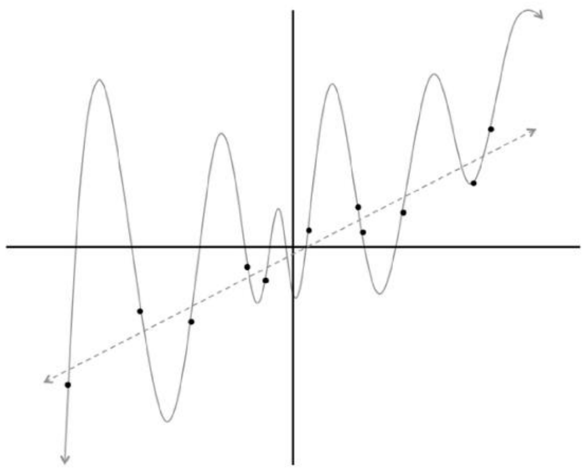


Figure 2.18: Two potential models that might describe our dataset: a linear model versus a degree 12 polynomial [10]

We are given a bunch of data points on a flat plane, and our goal is to find a curve that best describes this dataset (i.e, will allow us to predict the y coordinate of a new point given its x coordinate). Using the data, we train two different models: a linear model and a degree 12 polynomial. Which curve should we trust? The line that gets almost no training example, correct? Or the complicated curve that hist every single point in dataset? At this point we might trust the linear fit because it seems much less contrived. But just to be sure, let's add more data to our dataset. The result is shown in **Figure 2.19**.

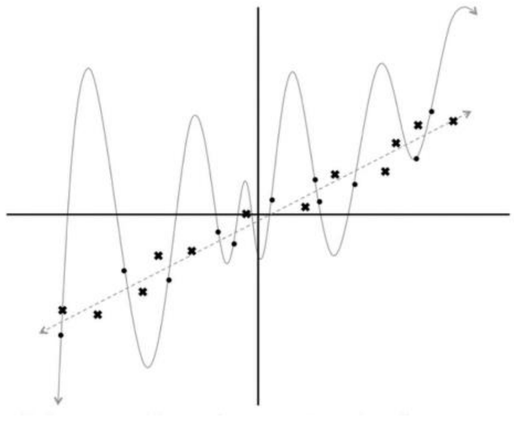


Figure 2.19: Evaluating our model on new data indicates that the linear fit is a much better model than the degree 12 polynomial [10].

Now the verdict is clear: the linear model is not only better subjectively but also quantitatively (measured using the squared error metric). This leads to an interesting point about training and evaluating machine learning models. By building a very complex model, it's quite easy perfectly fit our training dataset because we give our model enough degrees of freedom to convert itself to fit the observations in the training set. But when evaluate such a complex model on new data, it performs poorly. In other words, the model does not generalize well. This is a phenomenon called *overfitting*, and it is one of the biggest challenges that a machine learning engineer must combat. This becomes an even more significant in deep learning, because the networks have a large number of layers containing many neurons [10].

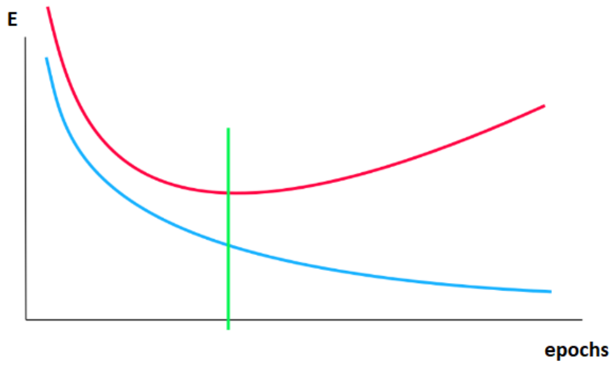


Figure 2.20: Representation of overfitting [10]

Here we have another graphic to understand better the overfitting. The red line is the error with the new data and the blue line is the error of the training data. How we can see the error in the training data is decreasing but in the new data is increasing, the model is generalizing, this is the definition of overfitting.

Let’s see how this looks in the context of neural network. We have a neural network with two inputs, a softmax output of size 2, and hidden layer with 3,6 or 20 neurons. We train these networks using for example minibatch gradient descent, and the results, visualized using ConvNetJS, are shown in **Figure 2.21.**

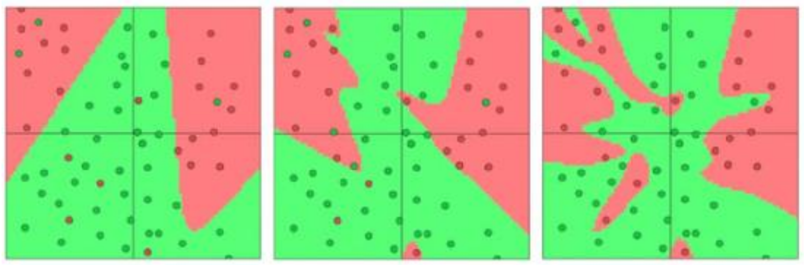


Figure 2.21: A visualization of neural networks with 3, 6 and 20 neurons (in this order) in their hidden layer [10]

It’s already quite apparent from these images that as the number of connections in our network increases, so does our propensity to overfit to the data. We can similarly see the phenomenon of overfitting as we make our neural networks deep. These results are shown in **Figure 2.22**.

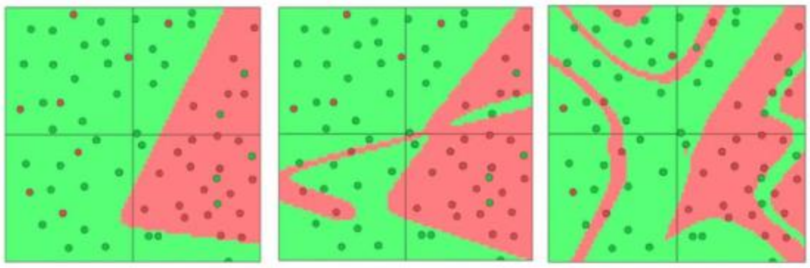


Figure 2.22: Neural networks with one, two and four hidden layers (in this order) of three neurons each [10]

This leads to three major observations. If our model is very complex (especially if we have a limited amount of data at our disposal), we run the risk of overfitting.

Second, it is misleading to evaluate a model using the data we used to train it. We need to split our data in *training set* and a *test set*. In the real world, large dataset is hard to come by. Consequently, it may be tempting to reuse training data for testing or cut corners while compiling test data. Important: If the test set isn’t well constructed, we won’t be able draw any meaningful conclusions about our model [10].

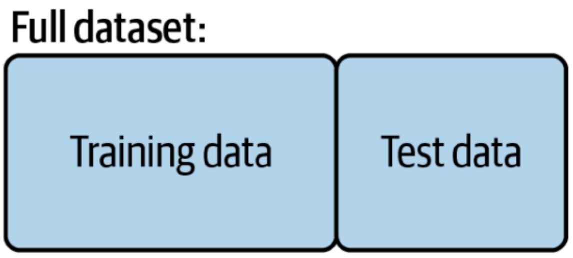


Figure 2.23: Nonoverlapping training and test set [10]

Third, it’s quite likely that while we are training our data, there is a point in time when instead of learning useful features, we start overfitting to the training set. To avoid that, we want to be able to stop the training process as soon as we start overfitting to prevent poor generalization. To do this, we divide our training process into epochs. An epoch is a single iteration over the entire training set. If we have a training set of size “x” and we are doing minibatch gradient descent with size “y”, then an epoch would be equivalent to “x” “y” model updates. At the end of each epoch, we want to measure how well our model is generalizing. To do this, we use and additional *validation set*, which is shown in **Figure 2.25**.

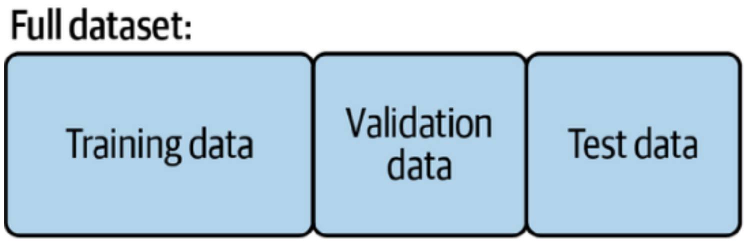


Figure 2.25: A validation set to prevent overffiting during the training process [10].

At the end of an epoch, the validation set will tell us how the model does on data it has yet to see. If the accuracy on the training set continues to increase while the accuracy on the validation set stays the same (or decreases), it is good sing that it is time to stop because we are overfitting.

For finish, let’s see the workflow we use when building and training deep learning models:

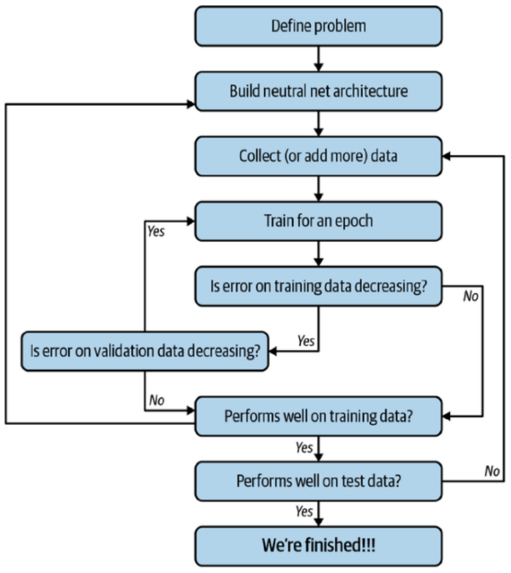

Define problem
Build neutral net architecture
Collect (or add more) data
Train for an epoch
Is error on training data decreasing?
Yes
No
Is error on validation data decreasing?
Yes
No
Performs well on training data?
Yes
Performs well on test data?
No
Yes
We're finished!!!

Figure 2.26: Detailed workflow for training and evaluating a deep learning model [10]

How we can see in the workflow is a summary of all talked in the previous sections.

3. State of the art

In recent years, various deep learning algorithms have been utilized to detect and evaluate mammograms in different BI-RADS categories. Each algorithm employs distinct features for training and specific architectures for creating neural networks. These fundamental elements serve as the foundation for developing deep learning algorithms and offer a multitude of possibilities for combinations. **Table 3.1** displays a comprehensive list of deep learning algorithms with their respective characteristics. It's worth noting that in this case, the classification column is not based on benign or malignant outcomes but rather on the four categories of BI-RADS.

Table 3.1. Overview of the deep learning literature

Reference	*Approach*	*Architecture*	*Segmentation*	*Classification*	*Dataset*
Setiawan et al. 2015 [29]	Mammogram classification using Law's texture energy measure and deep learning	ANN Classifier	✓	✓	327 images from Analysis Society (MIAS)
Gastounioti et al. 2018 [30]	Breast patterns finder associated with breast cancer risk via deep learning	Lattice-based Texture Analysis with Multi-channel CNNs	✓	✗	Unknow
Jadoon et al. 2017 [31]	Three-class mammogram classification based on descriptive CNN features (Deep learning)	Multilayered CNNs	✓	✓	Unknow
Arora et al. 2020 [32]	Benign and malignant classification via deep learning		✓	✓	Unknow

Altan et al. 2020 [33]	Three-class mammogram classification via deep learning		✓	✓	Unknow
Suh et al. 2020 [34]	Cancer detection in mammograms of various densities via deep learning		✓	✗	Unknow
Shen et al. 2019 [35]	Classification of patches in benign or malignant calcification or masses via deep learning		✓	✓	Unknow
Mohamed et al. 2018 [36]	Deep learning-based breast density three-class mammogram classifier		✓	✓	22000 images
Wang et al. 2016 [37]	Identifying metastatic breast cancer via deep learning		✓	✓	Unknow
Adedigba et al. 2019 [38]	Deep learning-based classifier for small dataset		✓	✗	Unknow
Aylet Akselrod-Ballin et al. 2019 [16]	Combination of machine and deep learning for detection of breast cancer	XGBoost algorithm	✓	✗	52936 images

Al-masni et al. 2017 [17]	Computer-aided diagnosis via deep belief network	RBM	✓	✗	2620 images from South Florida University
Yong Joon Suh et al. 2020 [18]	Deep learning model based on various densities	DenseNet-169 and EfficientNet-B5	✓	✓	3002 imgaes
Dezso Ribli et al. 2018 [19]	Deep learning model based on Regional Proposal Network	Faster R-CNN (VGG16)	✓	✓	2620 DDSM + 847 University of Budapest
Kim et al. 2021 [20]	Deep learning algorithm based on CAD model (AI-CAD)	Deep CNN, dANN and Faster R-CNN	✓	✗	Unknow
Rodriguez Ruiz et al. 2019 [21]	Deep learing model support system	AI computer sysytem Transpara	✓	✗	9000 images
Sun et al. 2019[22]	Deep learning algorithm based on multi-view CNNs	MVMDCNN	✓	✓	10400 images
Teare et al. 2017 [23]	Dual deep convolutional neural networks	Inception v3	✓	✗	6000 images
Chougrad et al. 2018 [24]	Deep CNNs trained with different architectures	VGG16, ResNet50 and Inception v3	✓	✗	5316 from DDSM + 600 from BCDR + 200 INbreast + 6116 from MD
Bandeira Diniz et al. 2018 [25]	CNNs based in bilateral analysis adapted to breast density	Deep CNN	✓	✗	2500 images from Massachusetts General Hospital

Nan Wu et al.2020 [26]	CNNs based in different images views	ResNet	✓	✓	229426 images

Table 3.1 demonstrates the multitude of strategies and algorithms available for detecting cancer in mammograms. However, only a few algorithms classify based on the different scales of BI-RADS. The majority of algorithms segment the mammogram and identify zones that may be sensitive to cancer, aiding radiologists in identifying areas of interest. Moreover, most algorithms categorize as malignant or non-malignant, but in this study, the focus is on classification based on the BI-RADS scale.

Through this review, we gain a comprehensive understanding of the various strategies and CNN architectures employed in deep learning for breast cancer detection.

4. Methodology framework

After reviewing the state of the art literature, I selected the most appropriate algorithm for implementing the classification of the four levels of BI-RADS. The algorithm that best fits my needs is described in the paper "*Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening"* [26]. This algorithm is based on CNNs and employs a ResNet architecture.

The decision to choose this algorithm was based on several technical reasons. CNNs are the most effective in classifying images, and ResNet architecture is both simple and powerful. Moreover, I found the use of two different views of the breast for inputs and heatmaps to be an interesting feature of this algorithm.

4.1. ResNet Architecture

In the case of the paper selected uses the Resnet architecture or also known as *Deep Residual Learning for Image Recognition*.

I am going to explain a little bit which problem resolves and how it works.

Deep convolutional neural networks have led to a series of breakthroughs for image classification. Deep networks naturally integrate low/mid/high-level features and classifiers in an end-to-end multilayer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth) [12].

Driven by the significance of depth, a question arises: Is learning better networks as easy as stacking more layers? The response is no. Here appears the problem of the vanishing/exploding gradients. Basically, this problem appears when the weights are very small (vanishing) or very big (exploding) which leads to a steep learning rate.

For understand this problem first of all we need to understand what a gradient is here. Basically, the gradient is the relation between the error respect the weights and this indicates to the network how is going the learning.

$$Gradient = \frac{\partial E}{\partial W}$$

The network uses this gradient to recalculate the weights using the backpropagation algorithm to improve the error.

One solution to resolve this is normalize the gradients to avoid that these gradients become small or big. This works well for networks that have tens of layers but no for networks that have cents of layers. Here is where the Resnet architecture solve this problem.

Now that we know what is the vanishing problem let's see with one example how it looks this problem:

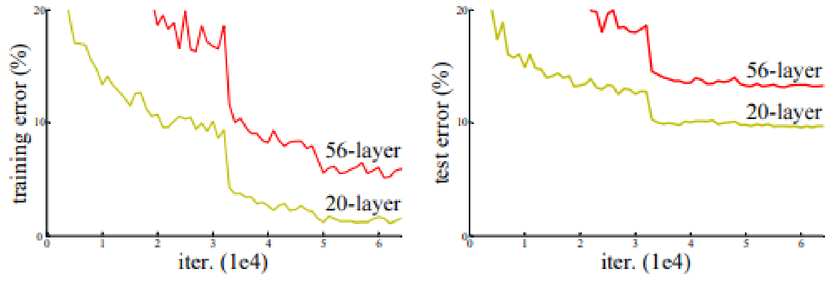
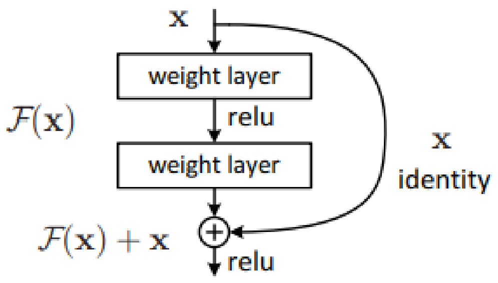


Figure 4.1: Training error (left) and test error (right) [12].

How we can see the network with more layers have more problems to learn. We can think that this is a problem of overfitting but is not the case. If it were an overfitting problem we would see that in the training error is decreasing every epoch but in the test error we would see an increment of the error.

Now we are going to explain how it works the Resnet architecture and solve this problem. This architecture uses a framework named deep residual learning and conclude that a deeper model should no produce higher training error that its shallower counterpart.

The idea is instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping that we can see in the **Figure 4.2**.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 4.2: Residual learning: building block [12]

Formally, denoting the desired underlying mapping as H(x), we let the stacked nonlinear layers fit another mapping of F(x) = H(x)−x. The original mapping is recast into F(x)+x [12].

Let's put an example to understand better. Imagine that you have a neuronal network that works well but you want to increase their depth. You can do that adding to the network the building blocks showed in the **Figure 4.2** without causing the problem of vanishing problem. The network will look like that:

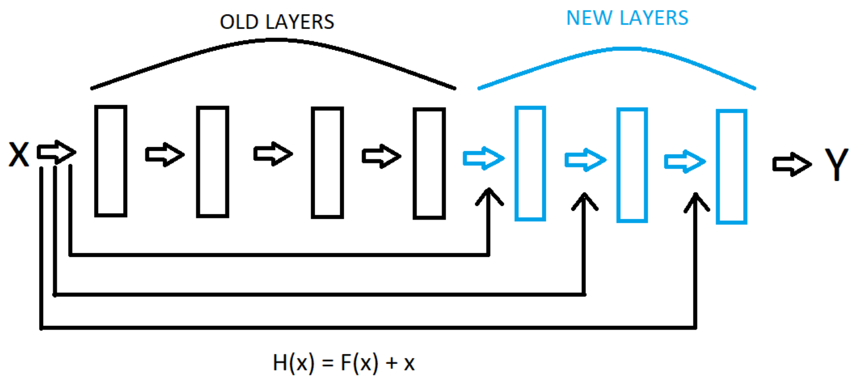


Figure 4.3: Neuronal network using ResNet architecture

Now that we have the architecture in the mind, I am going to analyse in more detail why implementing this not affects bad in the results of the network. Imagine that we have this scene:

We have an input "x" that pass with a linear combination [w'*x] and we obtain an output H' and goes through an activation function and we obtain an output Y'. And the same in the second part. We avoid bias for simplification. Imagine first that we don't have the bypass. The equations that we obtain are the following:

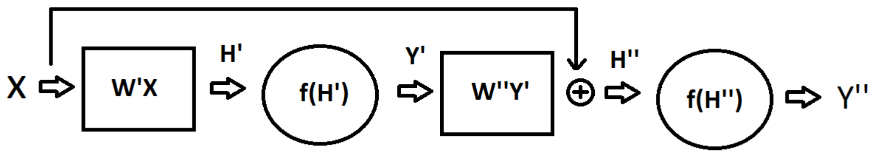


Figure 4.4: Flow of the ResNet architecture

$$Y'' = f(H'') = f(W''Y')$$

$$H'' = W''Y' = W''(f(W'x))$$

Now we consider the bypass, the only change is in H'':

$$H'' = W''Y' + x$$

In the Resnet architecture [W’’*Y’] considers the residual function and we obtain the function that is present in the **Figure 4.2**:

$$F(x) = W''Y'$$

$$H(x) = F(x) + x$$

Perfect we obtain the residual function but surely you are thinking, what is the advantage of this? Well, how we can see the residual function depends on the W’’ and this implies that H(x) also. In the case that we have the problem of vanishing gradients the residual function will be near 0 because the weight will be small and we can delete it. This would be a problem if we don’t have the bypass because we obtain a 0 but is not the case because we have the x term that is the identity and this not affects to the result of the neuronal network.

This is the reason that why we can add new layers using Resnet without negatively affecting the result of the neuronal network. In addition, this is why most of the deep algorithms use the resent architecture.

4.2. Explanation of the algorithm (View-wise algorithm)

With a basic understanding of the workings of deep learning and the different CNN architectures, we can now delve into describing the workings of the chosen algorithm, sourced from the paper *"Deep Neural Networks Improve Radiologists’ Performance in Breast Cancer Screening"* [15]. In section 4.3 we will describe the reasons for the modifications for this architecture.

The paper introduces various architectures, but we opted for the view-wise architecture, which has demonstrated the most promising results. **Figure 4.5** illustrates the architecture in detail.

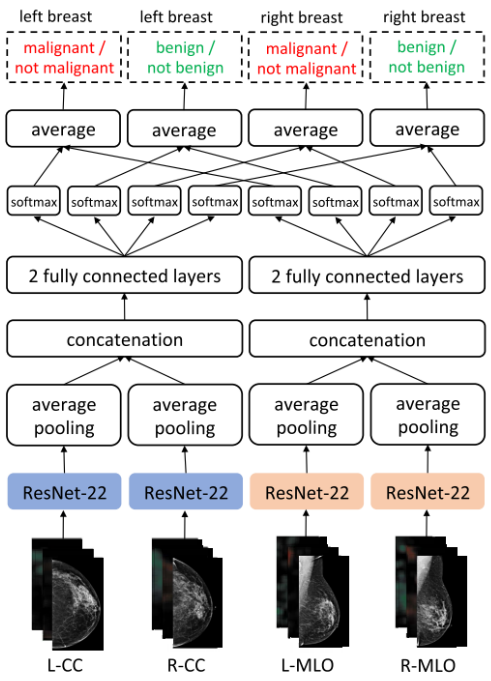


Figure 4.5: View-wise architecture [26]

To understand better the architecture, I am going to segment it and explain part by part.

4.2.1. Inputs and pre-processing

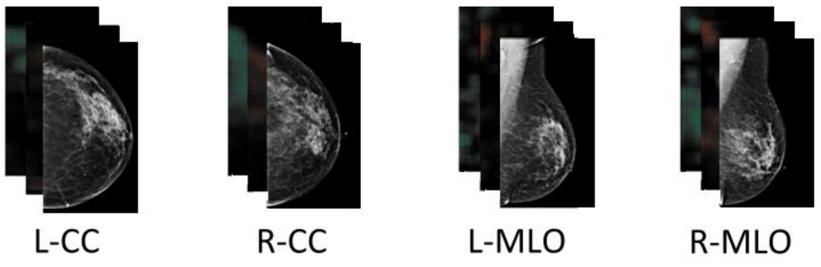


Figure 4.6: Inputs of the architecture [26]

Figure 4.6 provides an overview of the inputs used in the architecture. For the right breast, two views are considered, namely R-CC and R-MLO, while for the left breast, the views are L-CC and L-MLO. These views are typical for mammograms, and for each view, three images are generated the mammography image and two heatmaps, which we will discuss later in this text.

To generate these three images, the algorithm first crops the original image to remove any labels and pixels that do not correspond to the breast. This process adjusts the image to highlight the breast itself, as illustrated in **Figure 4.7**. Next, the algorithm extracts the optimal centers of the image to position the mammogram at the center, facilitating the model's ability to learn useful features. Once the image is centered, the algorithm generates the heatmaps using a Densenet, which is a CNN, to obtain the benign and malignant heatmaps. And that's how we obtain the inputs.

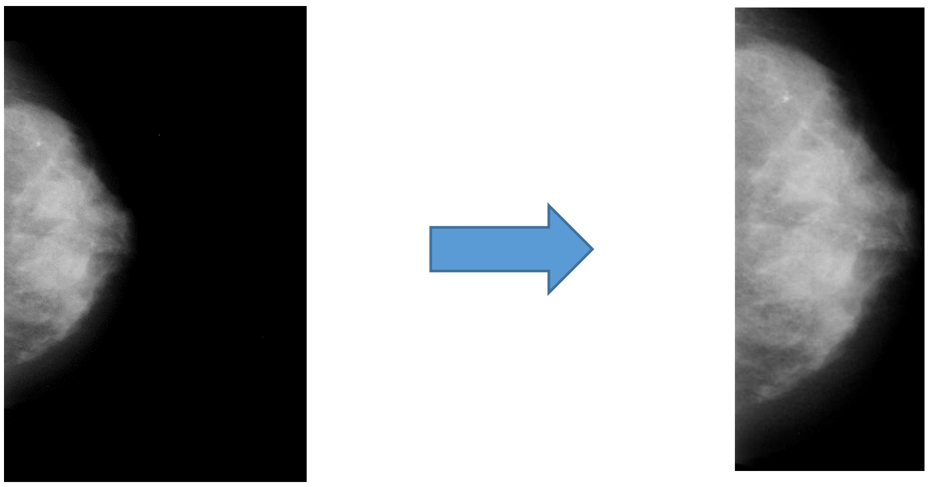


Figure 4.7: Image before cropping (right) and after cropping (left)

Let's discuss the heatmaps in detail. These heatmaps are created to reduce the time taken to classify images, as we can reduce the areas of interest. There are two heatmaps - one containing the estimated probability of finding a malignant tumor for each pixel, and the other containing the estimated probability of finding a benign tumor. This is done because working with high-resolution images would be computationally expensive, and would significantly increase the processing time required to process each pixel. By using heatmaps, we can mark the zones of major interest and remove the irrelevant zones, which allows the main classifier to benefit from pixel-level labels without requiring heavy computation to produce the pixel-level predictions each time an example is used for learning. [15]

In **Figure 4.8**, we can see an example of the heatmaps obtained using Densenet. We can observe that the two images are complementary in nature. The zones with a higher intensity of green or red indicate that this zone has a higher probability of being benign or malignant, respectively. We can also see that the zones with higher intensity of red have practically zero intensity in the benign image, and vice versa.

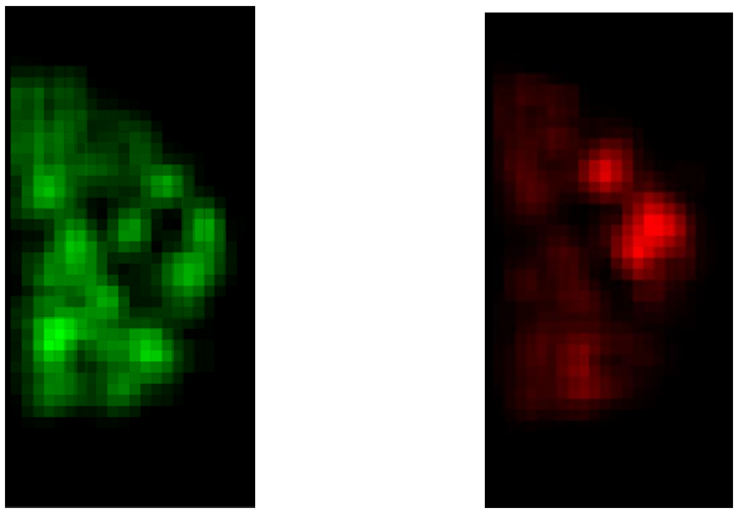


Figure 4.8: Benign heatmap (green) and malignant heatmap (red)

In the **Figure 4.9** we can see the image with the heatmaps and how the algorithm will process the pixels image for optimization of the computational power.

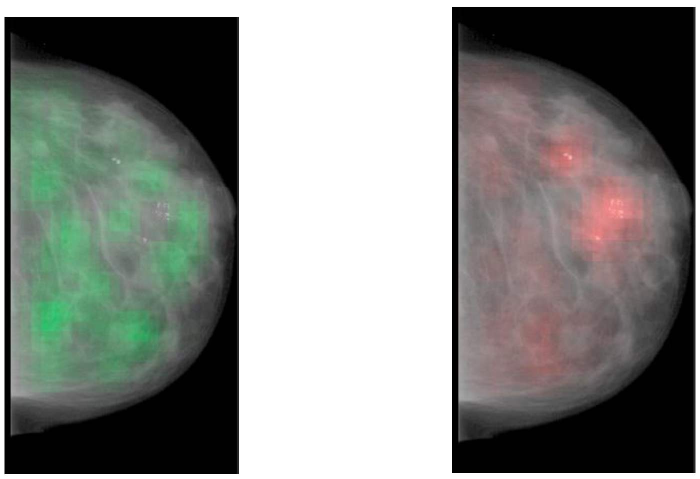


Figure 4.9: Image + Heatmaps

4.2.2. CNN

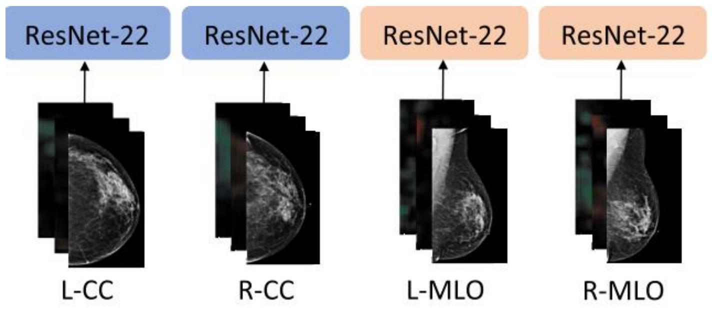
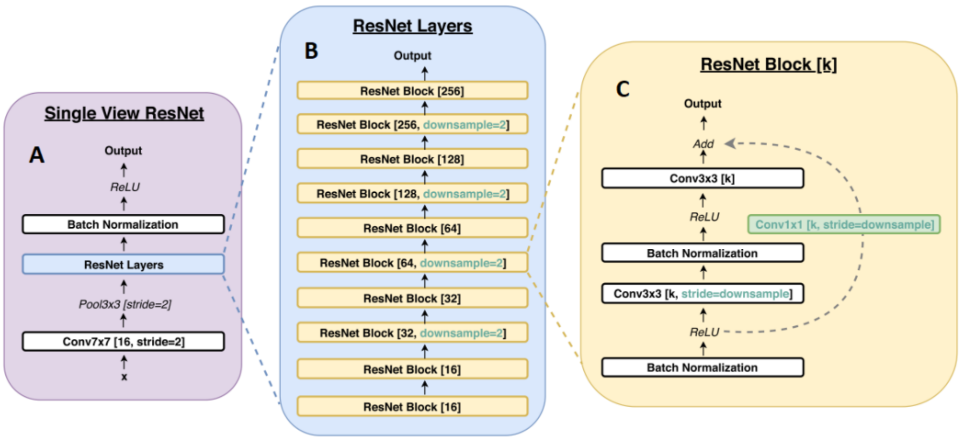


Figure 4.10: Inputs of the CNN [26]

When we have, the inputs there are passed to the CNN, in this case is a ResNet-22 that we talk in the previous points.

The architecture of this ResNet is based as columns computing 256 dimensions hidden representations vector of each view and each ResNet block consist of two 3x3 convolutional layers. Each ResNet Layer starts with 16 channels and each ResNet Block doubles the number of channels for a final hidden representation of 256 channels. In the **Figure 4.11** we can see the evolution of the channels starting at 16 and ending in 256.



Conv7x7	$1339\times971\times16$
ResBlock 1	$670\times486\times16$
ResBlock 2	$335\times243\times32$
ResBlock 3	$168\times122\times64$
ResBlock 4	$84\times61\times128$
ResBlock 5	$42\times31\times256$

Figure 4.11: ResNet architecture and ResNet Blocks. A) which is a brunch of the model, B) ResNet Layers and C) ResNet Block [26]

4.2.3. Average pooling and full connected layers

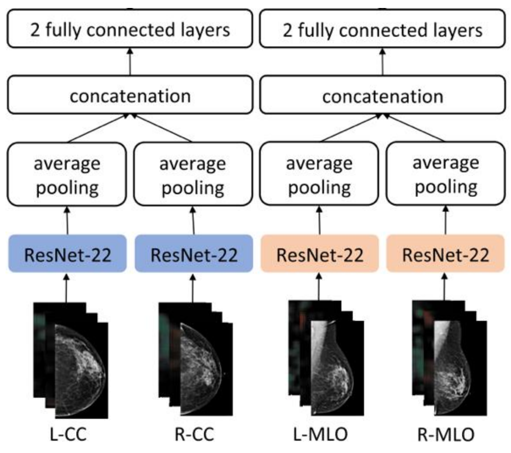


Figure 4.12: Average pooling, concatenation and fully connected layers [26]

At the output of the ResNet is done an average pooling and a concatenation for each view to unification the result to pass the result to the two fully connected layers.

This is done because using two fully connected layers before the output layer allow more complex interactions between the different views.

4.2.4. Softmax, average and final results

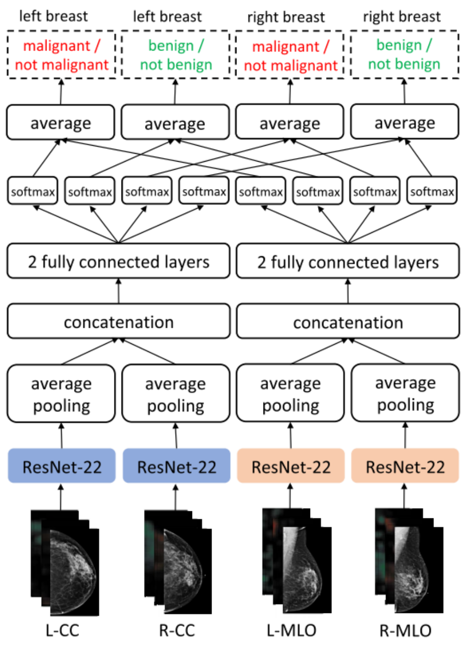
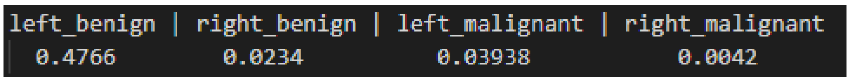


Figure 4.13: Softmax and averages [26]

The final step of the algorithm is to apply a two-step softmax for each view, which includes one for the benign/malignant and another for the not benign/malignant result. The output layer's highest value is chosen using the softmax function. Next, an average is calculated for each view by merging the information from both the left and right breasts. This produces the final results, which are then saved in a CSV file. **Figure 4.14** provides an example of the results obtained.



```
left_benign | right_benign | left_malignant | right_malignant
|  0.4766        0.0234          0.03938          0.0042
```

Figure 4.14: Results

4.2.5. Training

Now that we understand how the algorithm works, let's delve into the training process. The algorithm was pretrained using transfer learning, where a model pre-trained on a different task is used as a starting point for the target model. This approach leverages the learned representations from the pretraining task, improving the performance of the model. In this case, the architecture was pretrained on the BI-RADS classification.

Next, the entire model was fine-tuned using the Adam optimization algorithm, which is a stochastic gradient descent optimization algorithm. The learning rate used was 10-5, and the minibatch size was set to 4. The dataset used for training consisted of 229,426 images.

5. Customizing the algorithm

At the beginning of the project, I conducted a thorough review of state of the art algorithms and selected the one I described in the previous section based on its impressive results. I found the use of heatmaps to optimize computational cost and the utilization of different views for the input to be particularly intriguing. However, our objective is not only to classify an image as benign or malignant; we want to classify it into one of the four BI-RADS classes.

To accomplish this, we made some modifications to the algorithm. Specifically, we modified the fully connected layers and used two inputs, namely, two different views (CC and MLO) for the same breast (right or left), instead of four inputs as used in the paper. This is because our dataset does not have all four views for each breast and mammograms normally are requested by one breast at the time. The specialist commonly uses two views for the same breast to diagnose a patient. Hence, our model process mammograms in the same fashion as a specialist.

We did not modify the ResNet because it was pretrained on BI-RADS and we believe that the results obtained with it are satisfactory. The fact that it was pretrained on BI-RADS is encouraging, as the network has already learned features that can be used to classify the different BI-RADS types we are interested in. You can see the modified points in the **Figure 5.1**.

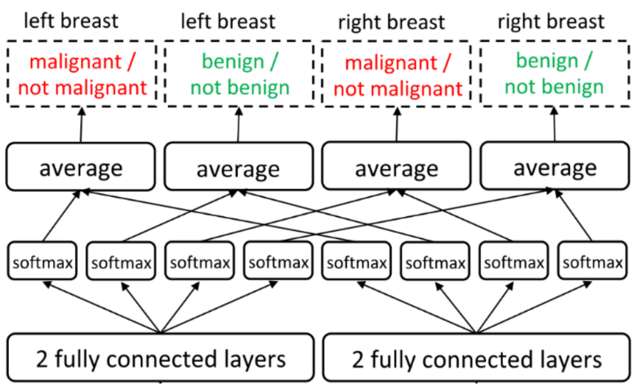


Figure 5.1: Modified points [26]

5.1.1. Dataloader

One of the challenges we faced while modifying the algorithm was developing a custom dataloader since we need to process two inputs simultaneously.

To overcome this, we looked towards the Multiview investigation field, which involves the classification of multiple input images at the same time. This served as my inspiration, and I was able to develop a dataloader that allowed me to use two different views (CC and MLO) of the same breast as inputs.

5.1.2. Retraining the fully connected layers

To modify the architecture for the specific task of classifying BI-RADS into four categories, we needed to retrain the fully connected layers. Since the original code did not have any training files, we adapted implementations and dataloaders from other models.

We opted to use two fully connected layers instead of three or one, as it worked better in our empirical analysis.

Retraining the layers was challenging, but we persevered and used a total of 1819 images, with 1632 for training and 187 for testing.

5.1.3. Softmax

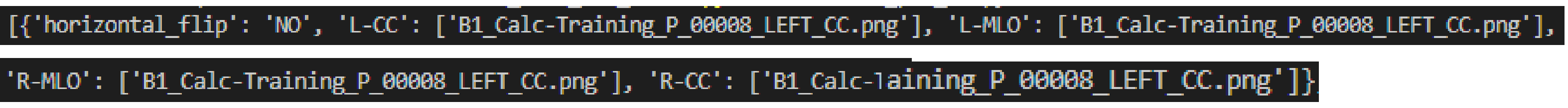
In our implementation, we utilized only one softmax layer. The output layer produces an array with four logits, one for each class of BI-RADS. The softmax layer then takes this array and transforms the logits into probabilities (normalized to 1) and selects the highest value, thereby giving us the final result.

5.3.3 Preparing for the preprocessing

In our case, we performed the same preprocessing as the original paper, but since we needed image pairs and our dataset contained images without pairs, we developed a code that searches for the corresponding images. We used a naming convention for the images, such as "B1_Calc-Test_P_00141_LEFT_CC". Firstly, we arranged all the images in numeric order, then we checked if the next image had the same number, and if not, we deleted it because it did not have a peer.

After filtering the images, we needed to create an array of dictionaries and save it to a file because the original code used it for preprocessing. We recreated the same dictionary using two keys and two values for each image. The first key indicated if we wanted a horizontal flip, and in our case, all the values for this key were "NO". The second key indicated the view of the image, with possible values of "L-CC", "L-MLO", "R-CC", or "R-MLO". We created four dictionaries for each image, one for each view.

In **Figure 5.2**, we provide an example of one image, where the view of the image is not important since it is only used for preprocessing. We have four dictionaries for each image because the original code has four views.



```
[{'horizontal_flip': 'NO', 'L-CC': ['B1_Calc-Training_P_00008_LEFT_CC.png'], 'L-MLO': ['B1_Calc-Training_P_00008_LEFT_CC.png'],
'R-MLO': ['B1_Calc-Training_P_00008_LEFT_CC.png'], 'R-CC': ['B1_Calc-1aining_P_00008_LEFT_CC.png']}
```

Figure 5.2: Array of dictionaries

Now we are ready to start the preprocessing. First crop the images, extract the optimal centers and generate the heatmaps.

5.3.4 Inputs

As explained in previous sections, three inputs per channel are required: the mammography image, the benign heatmap, and the malignant heatmap. To combine these inputs into a single RGB image, we assigned each input to a different color channel. Specifically, we assigned the mammography image to the blue channel, the benign heatmap to the green channel, and the malignant heatmap to the red channel. These inputs were obtained during the preprocessing phase.

We have done like this because we need to charge the inputs to the dataloader in a unique input. You can see an example in the **Figure 5.3**.

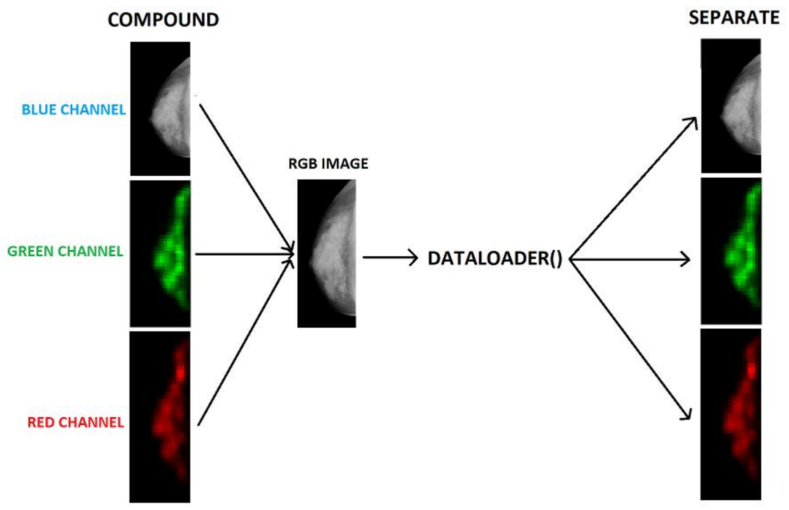


Figure 5.3: Creation of the input image and using to the dataloader to charge the inputs to the model

In the **Figure 5.4** we can see an example of a combined image:

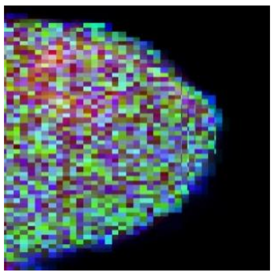


Figure 5.4: Combined image

As we combined all the images, we encountered a problem where some of the images were in the right position, which is problematic for the model because it expects the inputs to be in the left position.

To tackle this issue, we developed a code that calculates the average of the first 15% of the columns of the image. If this average is close to 0, it indicates that the image is in the incorrect position because most of the pixels are black. In such a case, we flip the image to the correct position.

Figure 5.5 shows a schematic representation of how this approach works.

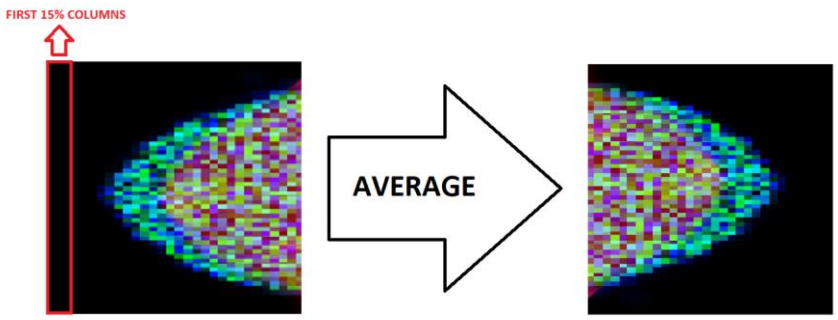


Figure 5.5: Turning scheme

5.3.5 Final model, MammoHeatNet (MHN)

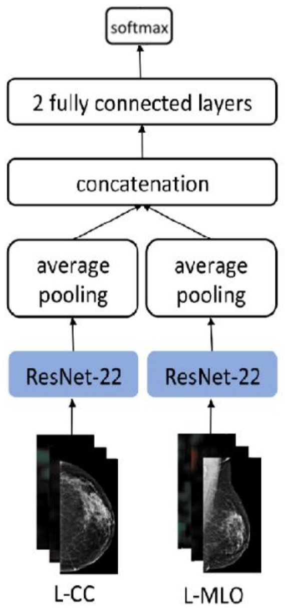


Figure 5.6: Schema of the MammoHeatNet (MHN)

The visual schematic of the MHN is presented in **Figure 5.6**. The model consists of two inputs, one for each view, which are composed of three images (the mammography and the two heatmaps).

These inputs are fed into two ResNet-22 models followed by average pooling. To combine the two inputs, we have used concatenation. Finally, the model is composed of one fully connected layer with ReLu activation and one fully connected output layer. The softmax function is applied to obtain the maximum logit predicted by the model, which is used to determine the BI-RADS classification.

6. Implementation and results

6.1 Mammographic database

In the present work, digital mammograms already classified and labeled are used to train and assess the approaches developed. The database consists of the following image types:

Tabla 6.1: Dataset

	TRAINING	TEST
BI-RADS 1	326	32
BI-RADS 2	462	56
BI-RADS 3	480	58
BI-RADS 4	364	41
TOTAL	1632	187

The database come from the University of Arkansas for Medical Sciences (UAMS) [14]. A set of experts has manually classified the databases following the guidelines in the Breast Imaging Reporting and Data System (BI-RADS).

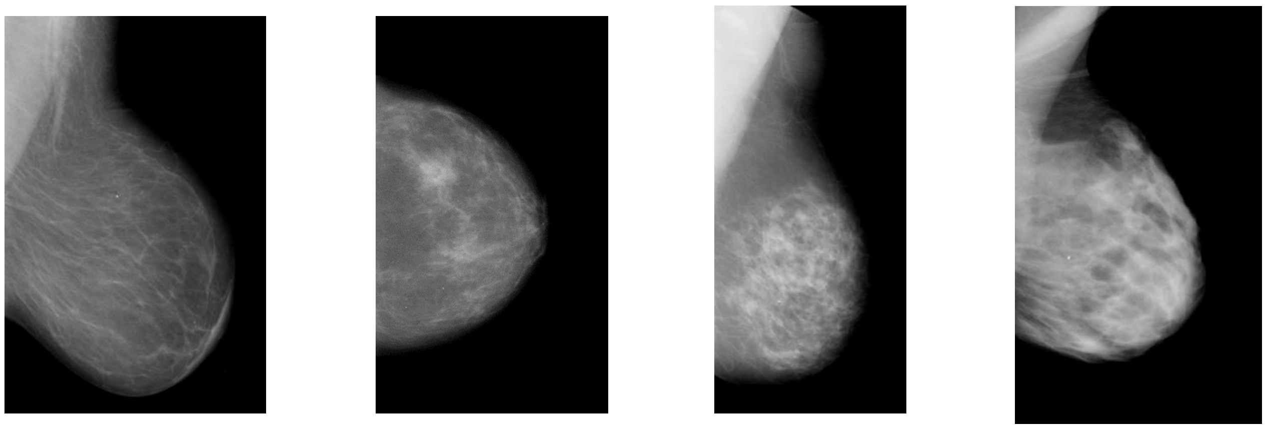


Figure 6.1: Example of BI-RADS 1 to 4 starting with the BI-RADS 1

6.2 Hyper-parameters configuration

We have completed the model and obtained the necessary inputs to train it. It is now time to explore the various configurations to determine which one works best for the model. In our case, we have identified 7 hyper-parameters that can be modified. These are as follows:

- Data augmentation
- Initialization
- Batch size
- Learning rate
- Dropout
- Learning rate step size
- Image size

I am going to explain a little bit each to understand how can affect to the model.

Data augmentation: Refers to the technique of artificially increasing the size of a training dataset by creating modified copies of the existing data. These modifications are applied using Pytorch library and include padding of 50 pixels, increasing the contrast 1.5 times, gaussian blur with kernel value of 5 sigma = (0.1, 2.0), rotation between ± 45 degrees, translation in x and y of ± 20% and scale between 80-100% to help to diversify the training set and improve model performance.

Initialization: Is an important hyperparameter that can affect how quickly a model converges and the accuracy it ultimately achieves. In our case, we explored initializing the model with Kaiming initialization, a mathematical technique that calculates the best initial weights for the neural network. This can save time during training and help the model reach maximum accuracy more quickly.

Batch size: Is another important hyperparameter that indicates to the model how many images to process in each epoch. Learning rate is another critical hyperparameter that regulates the weights of the neural network based on the loss gradient.

Learning rate: Refers to the step size at which the model updates its parameters during training. Specifically, the learning rate determines how much the parameters are adjusted with respect to the loss gradient calculated during backpropagation.

Dropout: Is a regularization function that discourages the model from overfitting by temporarily disabling some neurons during training.

Learning rate step size: Another regularization technique involves reducing the learning rate over a set number of epochs. By making smaller changes to the weights of the model, this approach can improve the model's generalization performance. It also helps prevent overshooting, which is when the model's weights are updated too drastically and fail to converge to a minimum.

Image size: Refers to the dimensions of the input images used to train the model. This can affect model performance and training time, so it is important to explore different sizes and find the optimal size for the given task.

6.3 Training

This is the final step before obtaining the results. Now that we have pre-processed inputs and a customized model, we move on to the training phase. To achieve this, I created a code that generates all possible combinations of the parameters that were explained earlier, and trains the model for each combination. Additionally, the code compiles information for each epoch and generates graphics to study and analyse the training behaviour later.

The maximum number of epochs for each training is 100, but conducting all possible combinations would take a lot of time. Therefore, I set a patience value of 10, which means that if the model fails to achieve a better accuracy in the next 10 epochs after the best accuracy, the training stops and moves to the next configuration.

I ran the training three times to characterize the statistical performance behaviour of the architecture and obtain the best configurations based on the accuracy of the model. The following are the values for each parameter:

- **Batch size:** [8 16 24 32 36 48 54 64]
- **Learning rate:** [1e-2 1e-3 1e-4 1e-5]
- **Learning rate step size:** [16 14 12 10 8 6]
- **Image size:** [224, 256, 350, 512]

We explored a batch size range of 8-64 because the literature [27] suggests a minimum batch size of 8 and a maximum of 64 due to hardware limitations.

For the learning rate, we chose values in the range of 1e-3 and 1e-4, as these are common values in the literature [27]. However, we also explored extreme values of 1e-2 and 1e-5 to observe their impact on the model.

The learning rate step size values were chosen based on our patience of 10. We experimented with values close to this patience to observe their effect on the model.

Regarding the image size, we selected a range of values based on several factors. Firstly, the original images were 224x224 and the ResNet was pre-trained on images of this size. Secondly, 256 is a common resolution for images. Finally, we chose a maximum of 512 as our images have a maximum resolution of 600x600.

It's worth noting that this results in a lot of combinations. In this case, there are 12 values, which means that there is 23^2 = 529 possible configurations. Conducting all of them would take 36 hours. This demonstrates that the process is computationally and time-intensive.

6.4 Metrics

The model's performance was evaluated using the following metrics:

- **Accuracy:** These measures how well the model predicts the correct class labels for a given set of input data. It was used for early stopping and to choose the best configuration.
- **Precision:** These measures how well the model correctly predicts positive class labels (true positives) out of all the samples it has predicted as positive (true positives and false positives).
- **Recall:** This measures the ratio between the number of positive samples correctly classified as positive to the total number of positive samples.
- **F1 score:** This metric combines the precision and recall scores of a model.

Although the precision, recall, and F1 score were calculated, they were not used in this study. These metrics are commonly used to compare different deep learning algorithms, and were included for the purpose of comparison with other studies.

6.5 Choosing the best combination

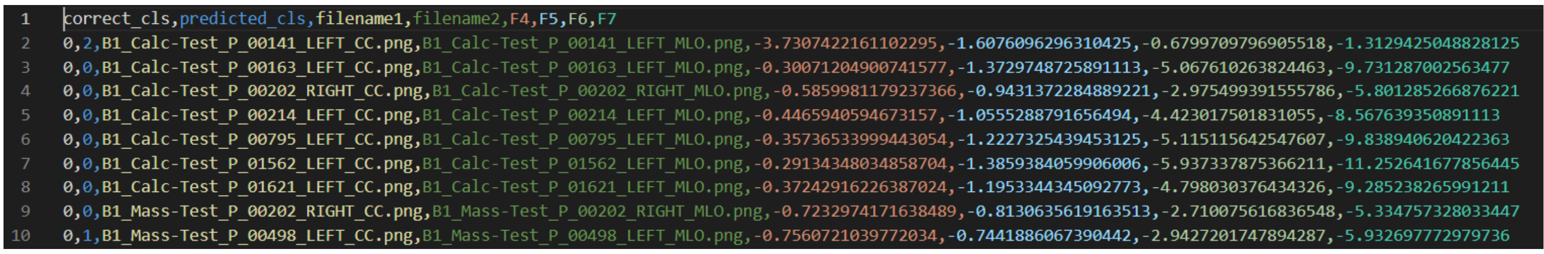
To begin with, after running all the possible combinations, I selected the top five configurations based on their accuracy. To choose the best configuration among them, I ran each of them ten times to eliminate the effect of random initialization and obtain an average accuracy for each configuration.

Apart from the accuracy, I compiled additional information to compare the configurations and observe their behavior. This includes the loss function, learning rate schedule, and accuracy over the epochs. This information helps to analyze the model's learning process and identify potential issues.

6.6 Graphics

To analyze the behavior of each model configuration, I collected important data to create a set of graphics including:

Data: I saved the classification results of the best epoch for both the training and test sets. This involved storing the correct output classification, predicted classification, input image names, and logits for the four possible classes. **Figure 6.1** shows an example of this data.



```
correct_cls,predicted_cls,filename1,filename2,F4,F5,F6,F7
0,2,B1_Calc-Test_P_00141_LEFT_CC.png,B1_Calc-Test_P_00141_LEFT_MLO.png,-3.7307422161102295,-1.6076096296310425,-0.6799709796905518,-1.3129425048828125
0,0,B1_Calc-Test_P_00163_LEFT_CC.png,B1_Calc-Test_P_00163_LEFT_MLO.png,-0.30071204900741577,-1.3729748725891113,-5.067610263824463,-9.731287002563477
0,0,B1_Calc-Test_P_00202_RIGHT_CC.png,B1_Calc-Test_P_00202_RIGHT_MLO.png,-0.5859981179237366,-0.9431372284889221,-2.975499391555786,-5.801285266876221
0,0,B1_Calc-Test_P_00214_LEFT_CC.png,B1_Calc-Test_P_00214_LEFT_MLO.png,-0.4465940594673157,-1.0555288791656494,-4.423017501831055,-8.567639350891113
0,0,B1_Calc-Test_P_00795_LEFT_CC.png,B1_Calc-Test_P_00795_LEFT_MLO.png,-0.35736533999443054,-1.2227325439453125,-5.115115642547607,-9.838940620422363
0,0,B1_Calc-Test_P_01562_LEFT_CC.png,B1_Calc-Test_P_01562_LEFT_MLO.png,-0.29134348034858704,-1.3859384059906006,-5.937337875366211,-11.252641677856445
0,0,B1_Calc-Test_P_01621_LEFT_CC.png,B1_Calc-Test_P_01621_LEFT_MLO.png,-0.37242916226387024,-1.1953344345092773,-4.798030376434326,-9.285238265991211
0,0,B1_Mass-Test_P_00202_RIGHT_CC.png,B1_Mass-Test_P_00202_RIGHT_MLO.png,-0.7232974171638489,-0.8130635619163513,-2.710075616836548,-5.334757328033447
0,1,B1_Mass-Test_P_00498_LEFT_CC.png,B1_Mass-Test_P_00498_LEFT_MLO.png,-0.7560721039772034,-0.7441886067390442,-2.9427201747894287,-5.932697772979736
```

Figure 6.1: Example of compiled data

Graphics: I created graphics showing the accuracy and loss for each epoch of the training and test sets, boxplots to visualize the average and dispersion for each configuration, and confusion matrices for the training and test sets.

In the **Figure 6.2** you can see an example.

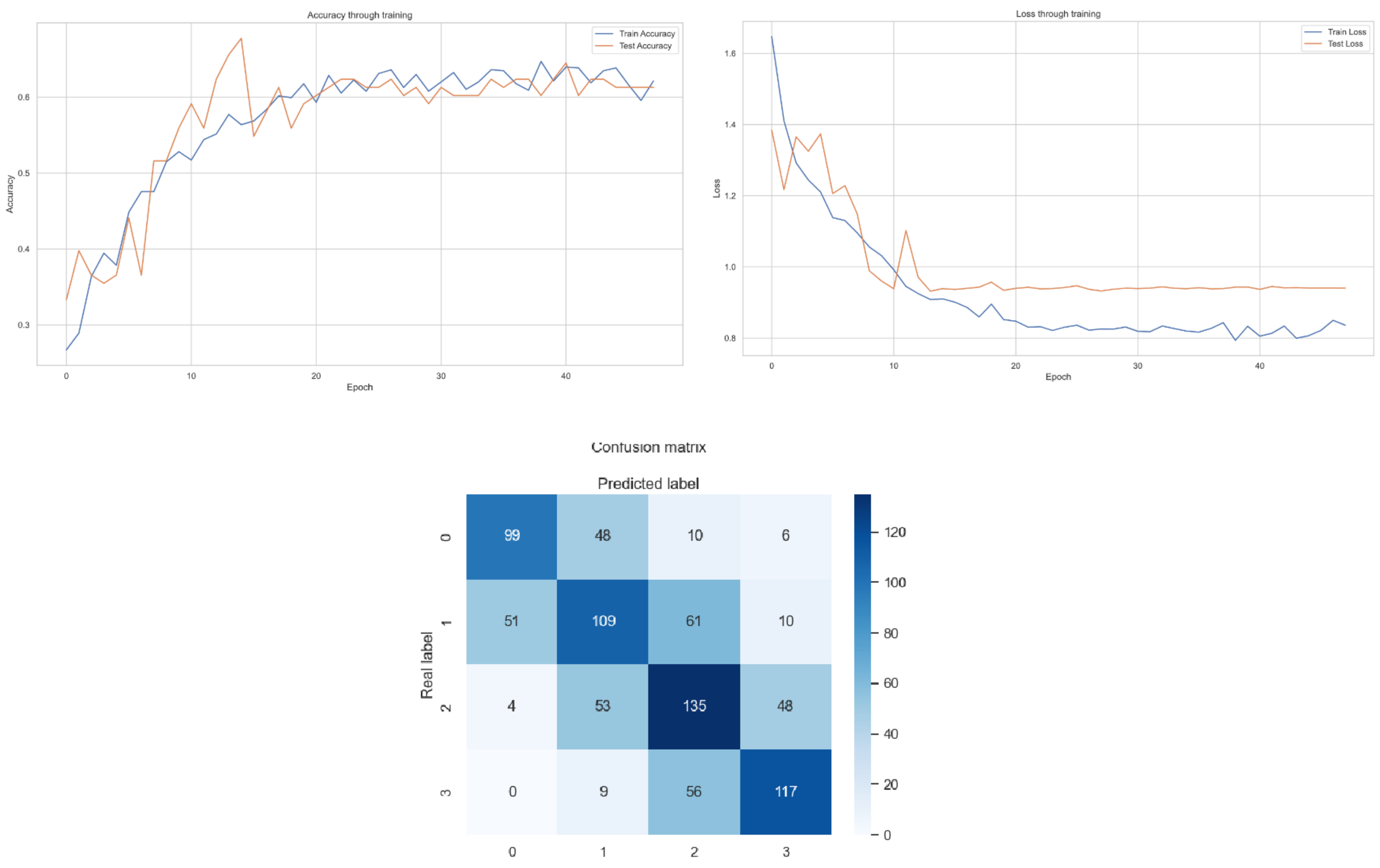


Figure 6.2: Graphics for epoch for accuracy, loss and confusion matrix

7. Results

In this section we are going to see a table with the top five configurations with their configuration, test accuracy and deviation. The *baseline i* is the model in its simplest form using the best architecture configuration, without any additional optimizations such as Kaiming initialization, data augmentation, or dropouts. The *baseline ii* is the same as *baseline i* but without the pre-trained weights. The purpose of establishing a baseline is to examine the model's performance without any improvements, and compare it with the results obtained after applying these three enhancements.

With the initial configuration achieves near a 47% of accuracy. This reflects the importance of doing the hyper-parameter search and training configuration.

Table 7.1: Best configurations found

MHN	Batch size	Learning rate	Lr. Step size	Image size	Accuracy
Base Line i	48	0.001	10	224	57.42 ± 4.34
Base Line ii	48	0.001	10	224	56.77 ± 4.26
Conf 5	16	0.0001	6	224	63.66 ± 2.42
Conf 4	32	0.0001	8	224	62.15 ± 2.95
Conf 3	48	0.0001	8	224	62.69 ± 2.43
Conf 2	8	0.0001	14	256	64.27 ± 2.96
Conf 1	48	0.001	10	224	65.49 ± 4.43

How you can see in the **Table 7.1** the best configuration is the first. In this case we have the following parameters:

- **Batch size:** 48
- **Learning rate:** 0.001
- **Weight decay:** 10
- **Image size:** 224

The best performance achieved by this configuration was an accuracy of 74.19%. As discussed in the methodology section, we experimented with different values for the learning rate and image size, and found that the optimal range for the learning rate was between 0.001 and 0.0001, and for the image size, between 224 and 256.

In the upcoming sections, we will present various graphics to better understand the model's behavior. We will focus on exploring the run that produced the maximum accuracy, and only present graphics for the best run to avoid overwhelming the reader with too much information.

7.1. Accuracy

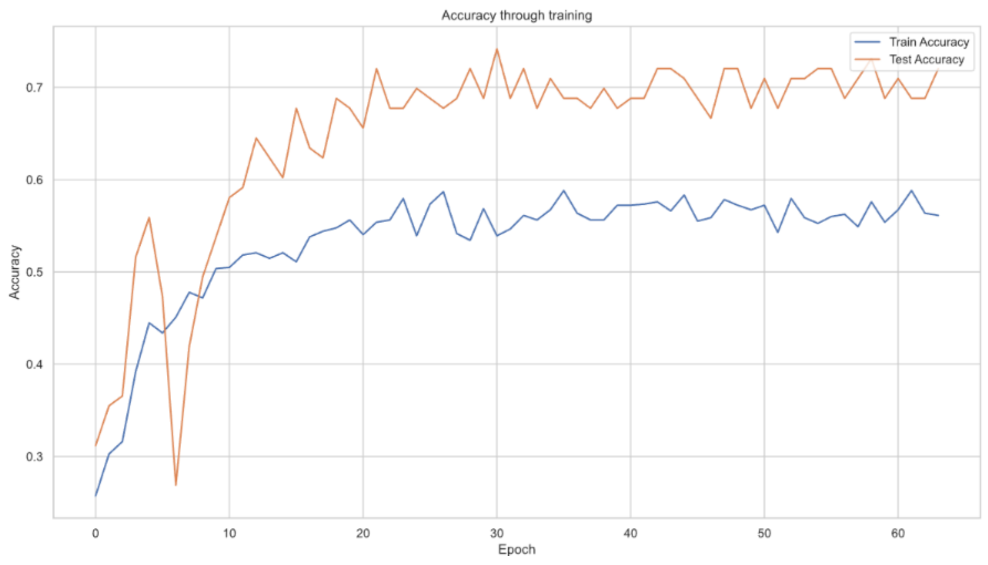


Figure 7.1: Graphic of accuracy respect epochs

In **Figure 7.1**, we can observe the evolution of the accuracy of the test and training during the training phase. It is worth noting that the test accuracy obtained during training does not affect the training of the model.

We can see that the best accuracy is achieved in epoch 30, and the training concludes in epoch 63. This is because, in this case, I increased the patience to 33 epochs to allow the model more time to achieve a better result.

As shown, we are obtaining better accuracy in the test set than in the training set, which is not typical. In this case, I am obtaining approximately 15% more accuracy in the test set. This difference becomes noticeable from epoch 8 and remains constant until the end of the training phase. One explanation for this is that the model may be over-regularized, leading to instances where the training performance is below the test performance, as seen in **Figure 7.1**. However, I am presenting this graphic because it represents the best result obtained. For a normal behavior, the same training configuration can be seen in **Figure 7.2**.

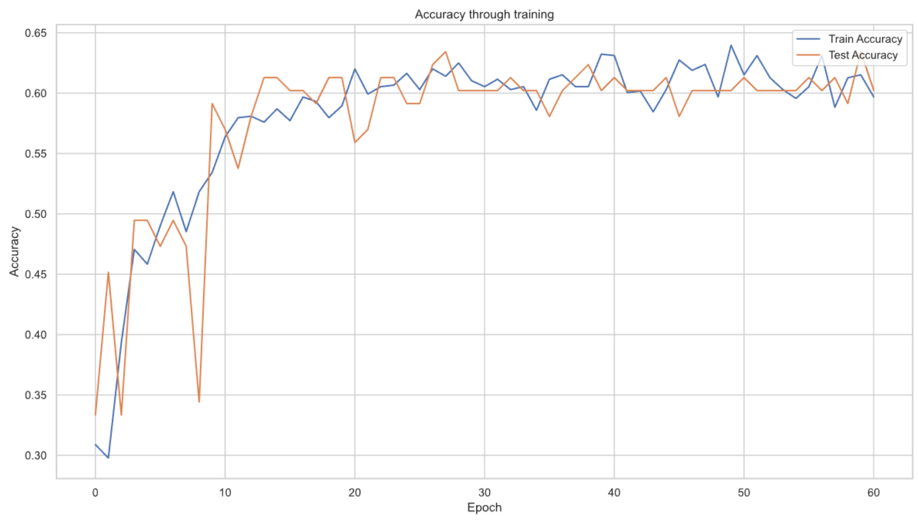


Figure 7.2: Graphic of a normal behavior of accuracy with the same configuration

7.2. Loss function

In this particular case, we opted to utilize a cross-entropy loss function. This choice stems from its prevalence in the literature, as well as its demonstrated effectiveness in achieving optimal training outcomes. Cross-entropy is a well-established penalization function, which makes it a reliable choice for training purposes. Additionally, it is simple to implement and is specifically tailored to optimize models with multiple categorical outputs.

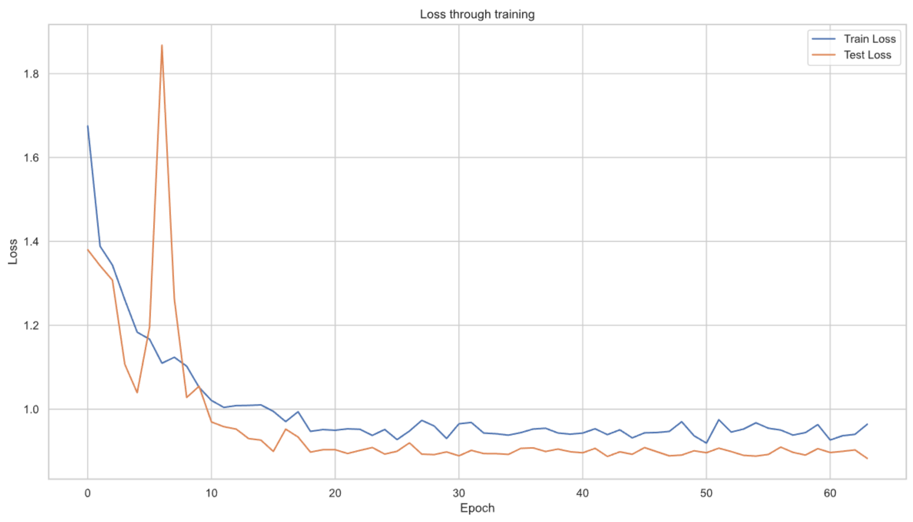


Figure 7.3: Graphic of loss respect epochs

In **Figure 7.3**, we can see the evolution of the loss throughout the training phase. This corresponds to the training configuration shown in **Figure 7.1** of the previous section. We can observe that there is no overfitting in this case. The training and test losses are parallel, which indicates that the model is not overfitting to the training data.

As mentioned earlier, in **Figure 7.1**, we saw a significant difference between the test and training accuracies, where the test accuracy was much higher. This suggests that we may be over-regularizing the model, and this is reflected in the difference between the test and training performances. However, in **Figure 7.3**, we can see that there is no overfitting, which further supports the idea that we may be over-regularizing the model.

In **Figure 7.4**, we can see the evolution of the accuracy and loss during the training phase for a more typical behavior of the model, without overregulation. As we can observe, the training and test accuracies are more aligned and gradually increase throughout the epochs until they converge at around epoch 60. The loss also decreases consistently throughout the training phase, without showing any significant fluctuations or overfitting. This indicates that the model is properly learning and generalizing from the data.

Overall, **Figure 7.4** provides a more reliable representation of the model's performance compared to the previous figures, which showed some atypical behaviors.

The loss has a horizontal behavior after epoch 20 and not presents overfitting.

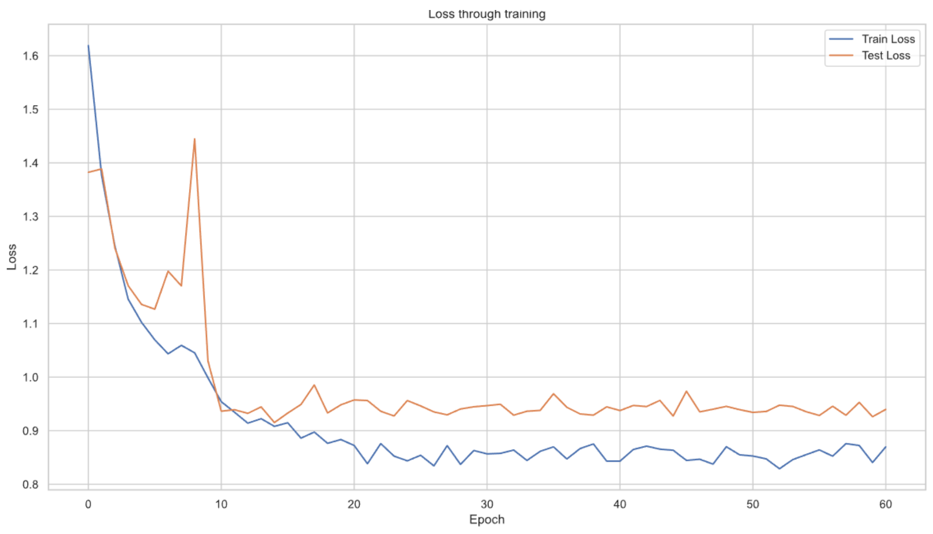
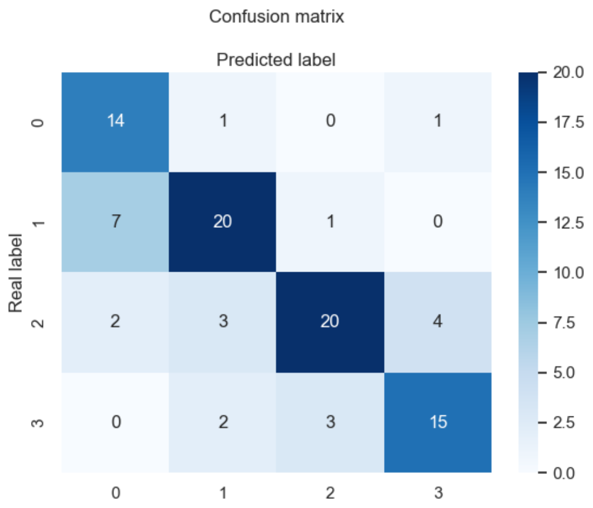


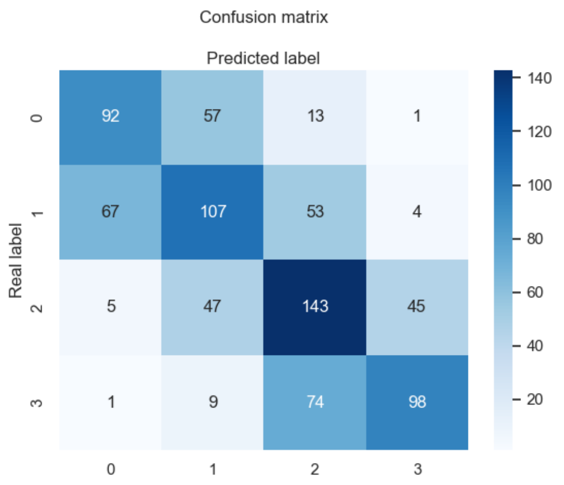
Figure 7.4: Graphic of a normal behavior of loss with the same configuration

7.3. Confusion matrix



Real \ Predicted	0	1	2	3
0	14	1	0	1
1	7	20	1	0
2	2	3	20	4
3	0	2	3	15

Figure 7.5: Test confusion matrix. The confusion matrix numeration (0,1,2,3) corresponds to BI-RADS (1,2,3,4).



Real \ Predicted	0	1	2	3
0	92	57	13	1
1	67	107	53	4
2	5	47	143	45
3	1	9	74	98

Figure 7.6: Train confusion matrix. The confusion matrix numeration (0,1,2,3) corresponds to BI-RADS (1,2,3,4).

These figures show the confusion matrix for the train and test sets, with **Figure 7.5** being the most relevant as it reflects the expected behavior of the model when predicting BI-RADS. As seen, the model is not confused with the extreme BI-RADS (1 with 4), as it only confuses them once each. However, the most challenging task for the model is correctly classifying BI-RADS in the medium range, specifically the 2s and 3s, which are the most similar and difficult to differentiate. The model makes 74 errors when trying to differentiate between BI-RADS 3 and 4, predicting a BI-RADS 3 when the actual value is 4.

For BI-RADS 1 and 2 or 2 and 1, the values are similar, indicating that the model has a consistent error in differentiating between them. The same happens with BI-RADS 2 and 3, but with fewer errors. To better understand the model's specific errors, **Table 7.2** calculates several values.

Table 7.2: General precision, recall and error of the model for configuration 1 of MHN

Predicted / Real	BI-RADS 1	BI-RADS 2	BI-RADS 3	BI-RADS 4
BI-RADS 1	92	57	13	1
BI-RADS 2	67	107	53	4
BI-RADS 3	5	47	143	45
BI-RADS 4	1	9	74	98
Precision	55.76	48.64	50.53	66.22
Recall	56.44	46.32	59.58	53.85
Error	44.24	51.36	49.47	33.78

As shown in **Table 7.2**, the model encounters the most difficulty in classifying class 2, with an error rate of 51.36%. Conversely, the classes with the least difficulty are class 4, with an error rate of 33.78%, followed by class 1, with an error rate of 44.24%. This trend makes sense as the extreme classes have clear indications of either having cancer or not, whereas the middle classes are more diffuse and less clear.

It is also worth noting the errors in contiguous classes to identify which classes the model confuses the most, in the **Table 7.3** we can see it:

Table 7.3: Error for contiguous classes

	Error
BI-RADS 1-2 and 2-1	38.39
BI-RADS 2-3 and 3-2	28.57
BI-RADS 3-4 and 4-3	33.06

As shown in **Table 7.3**, the MHN has difficulty to differentiate the BI-RADS 1-2 or vice versa.

7.4. Box-plots

Figure 7.7 presents a representation of the top five configurations using a box-plot. As previously mentioned, the top five configurations were run ten times to observe their behavior. Each box represents ten data points, which correspond to the accuracy obtained from each run. The black line inside each box indicates the mean accuracy of the configuration. The red line represents the average accuracy of the five configurations, while the blue line represents the highest accuracy achieved among the five.

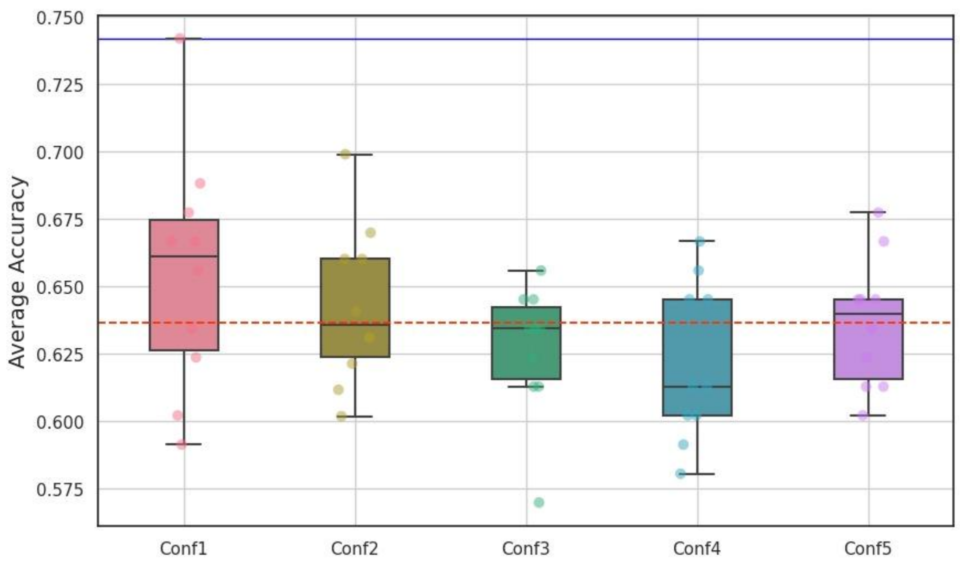


Figure 7.7: Box-plots for each configuration

By visually inspecting the deviation, we can see that the configuration 1 configuration has the highest accuracy average, but also the largest deviation. This indicates that the accuracy of this configuration is highly dependent on each run and may result in vastly different accuracies. In contrast, configuration 3 has the smallest accuracy but the smallest deviation as well, suggesting that we can expect relatively consistent results regardless of the runs.

Furthermore, we can conclude that configuration 4 is the worst-performing configuration, with the smallest average accuracy. However, it is worth noting that at certain points, it achieved accuracies that surpassed those of configuration 3.

8. Discussion and further works

Among the different parameters we explored, the learning rate and image size stand out as the most crucial ones, as their values exhibit very little variation across different configurations. This indicates that these parameters have a strong influence on the model's performance.

Interestingly, we consistently found that the image size was set to 224x224, even though this resolution may seem inadequate for capturing fine details. We hypothesize that this is because the ResNet was pretrained on images of this size, which made it easier for the model to extract relevant features for BI-RADS classification.

As for the learning rate, we found that a value of 0.0001 yielded the best results, which is lower than the commonly used threshold of 0.001 in the state of the art. This suggests that our model is more sensitive to changes in the learning rate and benefits from a more cautious approach. The results shows that we need more data samples to achieve a better accuracy. This is reflected in the best configuration tendency to overregularization. Moreover, the dataset is challenging domain task since the variability of the results obtained is quite high.

The *baseline ii* reflects the importance of having a pre-trained weights on a similar task or same type of images. Despite of the limitations of the proposal we consider valid the realized work as assistant tool for the specialist diagnosis because the training data is limited the performance results will be improved having more data. One important contribution of the work is sharing publicly the original dataset proposed for this work and its processed version used for its training.

Finally, a little summary of the main points of this work are the following:

Difficulties:

- Understand a code that you don't create it is a difficult task and I need a lot of time for achieve this point.
- I require a significant amount of time to execute all of the configurations for the model.
- Attempt to customize a code is not only modify the current code, I need to adapt the customized code for all the python libraries and this is not trivial.
- One big problem is that I used a lot of different libraries and sometimes there are not compatible and I need to adapt it. The biggest challenge of this part is creating a customized dataloader.
- In the original code it doesn't have a training file, for this reason I need to implement all the code for training the model.
- I need to implement all the code for the pre-processing of the images.

Limitations:

- The original dataset of the paper was public but I can't access it [26]. For this reason, I contacted with them but they don't respond me. Also, 2 weeks later they eliminate the web for access to the dataset.
- Having a lot of parameters makes difficult to optimize the training procedure correctly, easily and reaching the optimal configuration fast.
- Our dataset was so big but didn't contain enough cases of four different views per patient with the BI-RADS classification. This is the reason why we only use two inputs instead of four. Due this limitation of the dataset we need to adapt the architecture for only work with two input images. Also, because our task is in the BI-RADS classification the outputs of the model have to be modified.

  If the I would have had a biggest dataset surely, I can achieve better results.

- In summary I can group all of these problems in 3 points.
  - Implementation of the original code.
  - Creation of the dataset.
  - Customize the code for my goal.

Future improvements:

- Try to use another CNN or use the trained ResNet to create the heatmaps.
- Find the best images for training analyzing which images the model usually fails to classify.
- Try to use different CNN for the model architecture.
- Use all the data to do a deep analysis for search the best configuration for the model.
- Use new parameters for configurate the model.

9. Conclusions

In this project, I developed a deep learning algorithm that exhibits a high level of accuracy in classifying the four categories of BI-RADS. However, the process required significant effort and dedication on my part.

The work involved a thorough exploration of the current state-of-the-art, as well as a detailed understanding of the principles and mechanics of deep learning. This project was an equal combination of theoretical and practical aspects.

The development of the deep learning algorithm involved a considerable amount of customization to ensure that it could successfully classify both benign and malignant tumors, as well as the four classes of BI-RADS. The task involved modifications to the image pre-processing stage, which is a critical component of the algorithm since it is from this stage that all features are extracted.

Furthermore, I had to adapt the architecture of the algorithm to incorporate two inputs and combine multiple images into one. The creation of the data loader and training file presented a significant challenge, as did the resolution of incompatibilities with various libraries.

Once the algorithm produced results, the most intriguing and enjoyable aspect of the project was fine-tuning the model by modifying the architecture and hyperparameters. Given that deep learning algorithms are considered a "black box," it was necessary to run numerous experiments and analyze results to obtain the optimal model. This stage provided valuable insights into the thinking process of the deep learning algorithm.

After conducting a series of trial-and-error experiments, I developed an algorithm that exhibited commendable results, given the four possible outcomes. With additional data, the algorithm could potentially achieve even greater accuracy.

Throughout the project, I gained a deeper understanding of deep learning technology and programming, specifically in the Python language. I also gained familiarity with numerous libraries, which proved to be powerful tools for development. From a medical perspective, I learned about the intricacies of mammography and the complexities of differentiating between BI-RADS classes.

In conclusion, this project provided me with a valuable introduction to the world of deep learning, a field that I find fascinating and expect to achieve remarkable breakthroughs in the coming years. I intend to continue studying and developing deep learning algorithms, as programming is a passion of mine, and I am eager to explore the potential of these algorithms further.

10. BUDGET

This section outlines the hypothetical expenses associated with the development of the project. The budget is divided into two primary categories: personnel costs and material costs.

10.1. Personnel cost

Table 0.1 presents the costs associated with hiring a junior engineer to complete the tasks outlined in Figure 1.3 for the project's development. The average salary for this position is estimated to be 13 € per hour [30]. Assuming a standard workday of 4 hours and a project timeline of 32 weeks, the total amount of work hours required is estimated to be 896, resulting in a total cost of 11648 €.

Table 10.1: Personnel cost

Task	Working hours	Cost (€)
Planification	17	221
Introducing to the topic	15	195
Delve deeper into deep learning	35	455
Delve deeper into Python	40	520
Literary review	25	325
Bibliographic research	40	520
Data preprocessing	50	650
Customizing model	200	2600
Creation of extra files	60	780
Dataloader creation	30	390
Training the model	186	2418
Results obtention	50	650

Testing configurations	68	884
Discussion	20	260
Writing	60	780
Final review	40	520
TOTAL	896	11648

10.2. Material cost

The unique material cost has been the computer.

Table 10.2: Material cost

Hardware	Cost (€)
Computer	3500
TOTAL	3500

In conclusion, the total cost of the project is 11648 € for the junior engineer's salary, plus an additional 3500 € for materials, resulting in a total cost of 15148 €. It's worth noting that this estimation does not include the cost of project supervision or implementation.

11. ENVIRONMENTAL IMPACT ANALYSIS

The impact on the environment stemming from this project can be widely debated by thoroughly examining all the components directly or indirectly involved in its development, leading to numerous points of discussion. However, it should be noted that the project primarily consists of a series of codes created entirely with Python 3.8, meaning that no physical item has been manufactured or produced. Additionally, while the acquisition of radiological images may result in significant energy waste and environmental impact, it can be justified both ethically and economically. It is important to clarify that the focus of this report is solely on the environmental impact caused by the computer's electricity consumption and not on the image acquisition process.

Approximately 32 weeks were employed to develop the project. Considering an average of 28 hours of work per week using a computer, it can be estimated that electricity was consumed for a total of

896 hours plus the almost 186 hours required for training the model and obtain the best configuration. These parts were done using Alienware PC with an average consumption of 45W and NVIDIA GeForce GTX 1070 Ti with an average consumption of 150 W, according to the GPU's specifications [28].

$$Energy\ consumption = 896 * 45 + 186 * 150 = 68220Wh$$

Therefore, the total consumption of the computer for develop the whole project is 68.2 kWh.

In Catalonia, it is estimated that each kW produce generates 321 CO_2 [29].

$$CO_2\ produced = 68.2 * 321 = 22\ Kg\ CO_2$$

Bibliography

[1] Memorial Sloan Kettering Cancer Center (2023). Memorial Sloan Kettering Cancer Center. Obtained from:
https://www.mskcc.org/cancer-care/types/breast/anatomy-breast
[2] Cancer.org (2023). Obtained from:
https://www.cancer.org/cancer/breast-cancer/about/types-of-breast-cancer.html
[3] World Cancer Research Fund International (2023). Obtained from:
https://www.wcrf.org/cancer-trends/worldwide-cancer-data/
[4] Cribado poblacional de cáncer de mama en España (2023). Obtained from:
https://ingesa.sanidad.gob.es/ciudadanos/suSalud/mujer/docs/inform13.pdf
[5] Asociación española contra el cáncer (2023). Obtained from:
https://blog.contraelcancer.es/cribado-cancer/

[6] Wikipedia (2023). https://en.wikipedia.org/wiki/Mammography#Procedure
[7] Wikipedia (2023). Obtained from:
https://es.wikipedia.org/wiki/Mamograf%C3%ADa#:~:text=En%201993%2C%20el%20Colegio%20Estadounidense,dosis%20m%C3%A1s%20bajas%20de%20radiaci%C3%B3n.
[8] Radioterapia (2023). Obtained from:
https://radiopaedia.org/articles/breast-imaging-reporting-and-data-system-bi-rads?lang=us
[9] Levity (2023). Obtained from:
https://levity.ai/blog/difference-machine-learning-deep-learning#:~:text=Machine%20Learning%20means%20computers%20learning,documents%2C%20images%2C%20and%20text.
[10] Fundamentals of deep learning, Nithin Budama (2023). Obtained from:
https://books.google.es/books?id=2e5vEAAAQBAJ&printsec=frontcover&hl=es&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[11] Pico (2023). Obtained from: https://www.pico.net/kb/the-role-of-bias-in-neural-networks/

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015. Doi: https://doi.org/10.48550/arXiv.1512.03385

[13] Medium, Matthew Stewart (2023). Obtained from https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

[14] Wikipedia (2023). Obtained from:

https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=22516629

[16] Akselrod-Ballin A, Chorev M, Shoshan Y, Spiro A, Hazan A, Melamed R, Barkan E, Herzel E, Naor S, Karavani E, Koren G, Goldschmidt Y, Shalev V, Rosen-Zvi M, Guindy M. Predicting Breast Cancer by Applying Deep Learning to Linked Health Records and Mammograms. Radiology. 2019 Aug;292(2):331-342. doi: 10.1148/radiol.2019182622. Epub 2019 Jun 18. PMID: 31210611.

[17] Abdel-Zaher, A. M., Eldeib, A. M., & Salem, A. B. (2018). An Automatic Computer-Aided Diagnosis System for Breast Cancer in Digital Mammograms via Deep Belief Network. Journal of Medical Systems, 42(3), 48. doi: 10.1007/s10916-018-0915-5

[18] Ozturk, S., Ozkaya, U., & Barstugan, M. (2020). Automated Breast Cancer Detection in Digital Mammograms of Various Densities via Deep Learning. Journal of Medical Systems, 44(4), 84. doi: 10.1007/s10916-020-01557-2

[19] Ribli D, Horváth A, Unger Z, Pollner P, Csabai I. Detecting and classifying lesions in mammograms with Deep Learning. Sci Rep. 2018 Mar 15;8(1):4165. doi: 10.1038/s41598-018-22437-z. PMID: 29545529; PMCID: PMC5854668.

[20] Yoon JH, Kim EK. Deep Learning-Based Artificial Intelligence for Mammography. Korean J Radiol. 2021 Aug;22(8):1225-1239. doi: 10.3348/kjr.2020.1210. Epub 2021 May 4. PMID: 33987993; PMCID: PMC8316774.

[21] Alejandro Rodríguez-Ruiz, Elizabeth Krupinski, Jan-Jurre Mordang, Kathy Schilling. Detection of Breast Cancer with Mammography: Effect of an Artificial Intelligence Support System. doi: https://doi.org/10.1148/radiol.2018181371

[22] Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., & Summers, R. M. (2019). Multi-view convolutional neural networks for mammographic image classification. Computers in biology and medicine, 110, 54-63. doi: 10.1016/j.compbiomed.2019.05.018

[23] Philip Teare, Michael D C Fishman, Oshra BenzaquenOshra Benzaquen.Malignancy Detection on Mammography Using Dual Deep Convolutional Neural

Networks and Genetically Discovered False Color Input Enhancement. Doi: 10.1007/s10278-017-9993-2. June 2017.

[24] Hiba ChougradZouaki, HamidZouaki Hamid, Omar Alheyane. Deep Convolutional Neural Networks for Breast Cancer Screening.Doi:10.1016/j.cmpb.2018.01.011. January 2018.

[25] Bandeira Diniz JO, Bandeira Diniz PH, Azevedo Valente TL, Corrêa Silva A, de Paiva AC, Gattass M. Detection of mass regions in mammograms by bilateral analysis adapted to breast density using

similarity indexes and convolutional neural networks. Comput Methods Programs Biomed. 2018 Mar;156:191-207. doi: 10.1016/j.cmpb.2018.01.007. Epub 2018 Jan 11. PMID: 29428071.

[26] Wu N, Phang J, Park J, Shen Y, Huang Z, Zorin M, Jastrzebski S, Fevry T, Katsnelson J, Kim E, Wolfson S, Parikh U, Gaddam S, Lin LLY, Ho K, Weinstein JD, Reig B, Gao Y, Toth H, Pysarenko K, Lewin A, Lee J, Airola K, Mema E, Chung S, Hwang E, Samreen N, Kim SG, Heacock L, Moy L, Cho K, Geras KJ. Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening. IEEE Trans Med Imaging. 2020 Apr;39(4):1184-1194. doi: 10.1109/TMI.2019.2945514. Epub 2019 Oct 7. PMID: 31603772; PMCID: PMC7427471.

[27] "An Empirical Study of Batch Size on Large-Scale Image Classification" by Shallue et al. (2018): https://arxiv.org/abs/1804.07612

[28] Xataka (2023). Obtained from: https://www.xataka.com/analisis/nvidia-gtx-1070-ti-analisis-decidirse-en-la-gama-alta-de-nvidia-es-cuestion-de-pocos-fps#:~:text=La%20nueva%20Nvidia%20GTX%201070,W%20en%20su%20hoja%20t%C3%A9cnica.

[29] Canviclimatic Gencat (2023). Obtained from: https://canviclimatic.gencat.cat/web/.content/04_ACTUA/Com_calcular_emissions_GEH/guia_de_calcul_demissions_de_co2/190301_Practical-guide-calculating-GHG-emissions_OCCC.pdf

[30] Salario medio para Ingeniero Junior en España 2023 (2023). Obtained from: https://es.talent.com/salary?job=Ingeniero+junior

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, A. Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. 2019. Doi: 10.1109/MCSE.2019.2901468

[32] Y. Bengio, A. Courville and P. Vincent, "Representation Learning: A Review and New Perspectives," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1798-1828, Aug. 2013, doi: 10.1109/TPAMI.2013.50.

[33] R. F. Ribeiro et al., "Deep learning methods for lesion detection on mammography images: a comparative analysis," 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), Glasgow, Scotland, United Kingdom, 2022, pp. 3526-3529, doi: 10.1109/EMBC48229.2022.9871452.

Annex

A. Confidentiality

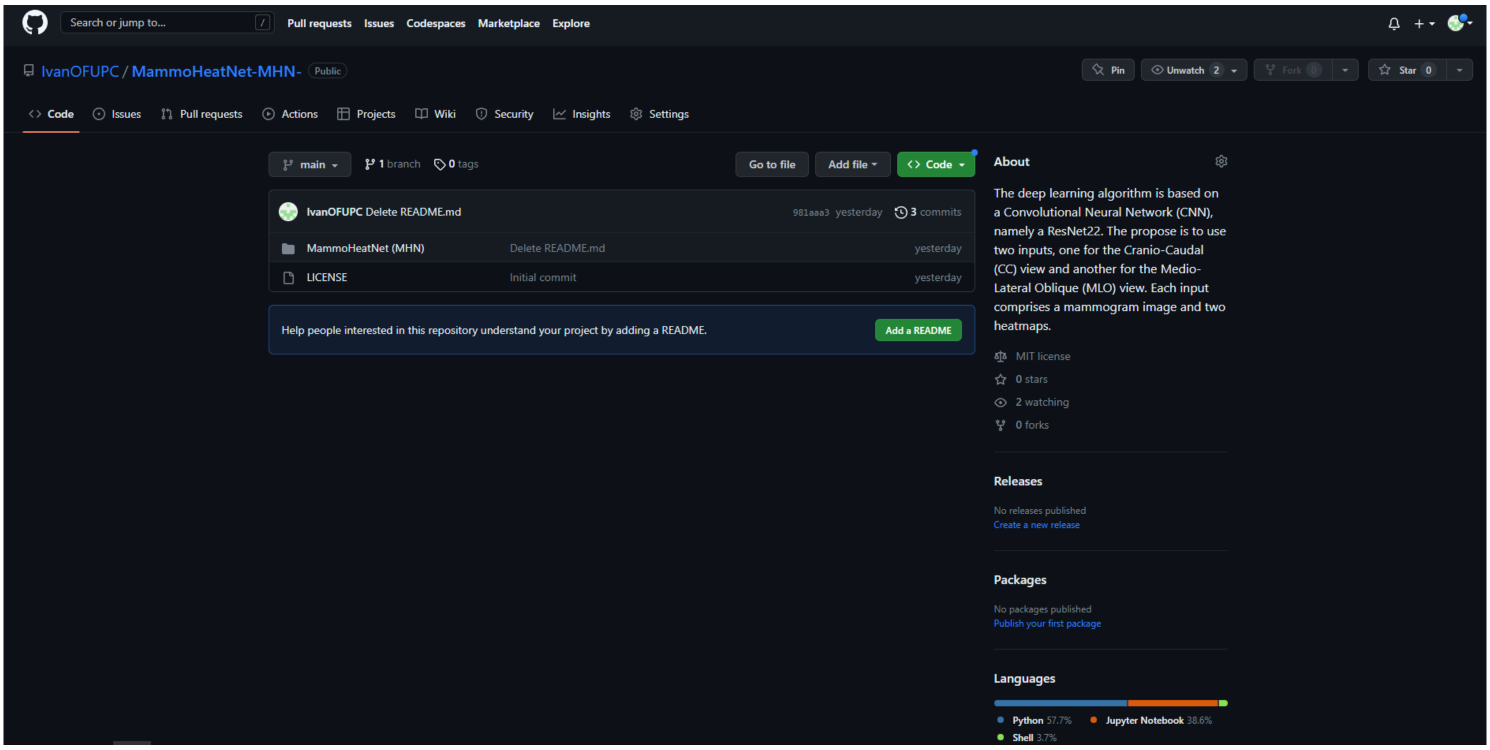
To ensure the privacy and confidentiality of sensitive information, this project has been designated as a confidential undertaking. The dataset used in this project contains potentially sensitive information that must be protected from unauthorized access and use. Additionally, as we plan to publish an article based on our findings, it is imperative that we maintain strict confidentiality to avoid compromising the integrity of the project and the privacy of individuals involved.

B. GitHub repository

All the implementation of this work is uploaded in a GitHub repository. Due to the confidentiality this link is unable for users and the access will be studied according to the scientific purposes and future improvements of the algorithm.

GitHub Project:

https://github.com/IvanOFUPC/MammoHeatNet-MHN-/blob/main/MammoHeatNet%20(MHN)/flipping.py



IvanOFUPC / MammoHeatNet-MHN- Public

1 branch 0 tags

IvanOFUPC Delete README.md 981aaa3 yesterday 3 commits

MammoHeatNet (MHN)	Delete README.md	yesterday
LICENSE	Initial commit	yesterday

Help people interested in this repository understand your project by adding a README.

About

The deep learning algorithm is based on a Convolutional Neural Network (CNN), namely a ResNet22. The propose is to use two inputs, one for the Cranio-Caudal (CC) view and another for the Medio-Lateral Oblique (MLO) view. Each input comprises a mammogram image and two heatmaps.

MIT license
0 stars
2 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● Python 57.7% ● Jupyter Notebook 38.6%
● Shell 3.7%