Predicting malignancy in breast cancer using Deep Learning

Maxence Boels

Report for the Master of Science in Computer Vision, Robotics and Machine Learning

from the

University of Surrey

Department of Electronic Engineering

Faculty of Engineering and Physical Sciences

University of Surrey

Guildford, Surrey, GU2 7XH, UK

August 2020

Supervised by Dr Kevin Wells

DECLARATION OF ORIGINALITY

I confirm that the project dissertation I am submitting is entirely my own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of my report to the JISC anti-plagiarism software resource, I confirm that my work does not contravene the university regulations on plagiarism as described in the Student Handbook. In so doing I also acknowledge that I may be held to account for any instances of uncited work detected by the JISC anti-plagiarism software, or as may be found by the project examiner or project organiser. I also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then I may forfeit any credit for this module, or a more severe penalty may be agreed.

Predicting malignancy in breast cancer using Deep Learning

Maxence Boels

Author Signature Date: 24/08/2020

Supervisor's name: Dr Kevin Wells

WORD COUNT

Number of Pages: 70

Number of Words with code included: 17366

Number of Words in report: 13913

ABSTRACT

Although breast cancer has come a long way over those past decades, women are still facing a lot of anxiety, expensive and unnecessary invasive biopsy due to a high false positive rate. For instance, out of 400 patients with a screening assessment, 330 had a biopsy and only 3.3% of those women had a malignancy [1]. Meaning, for over 96% of those women, a simple "wait and see" method rather than a stressful and expensive biopsy could have been recommended. So, there is a strong need to reduce unnecessary burdensome emotional stress for women. Meanwhile curbing the false positive curve with more accurate diagnosis, the need to reassure women with low chances of malignancy, but still referred for biopsy based on abnormal screenings, is necessary.

Therefore, this project is driven by the need to provide women with reassuring information regarding their abnormal screening while going through the stressful waiting period that precedes the concluding biopsy results, to reduce the emotional burden.

This can be done by providing women, that had an abnormal screening result from an x-ray mammogram, a likelihood of having a malignant cancer.

Regarding the data, all 628 screening mammograms in this project have been classified as abnormal by 2 radiologists and thus require a biopsy. Radiologists are cautious during screenings, the consequences of having a false negative push them to send women for biopsy if there is the slightest doubt for them to have cancer. As explained, this results in a high number of false positive since the 'cost' is lower than having a false negative. The dataset used in this project comes from the OPTIMAM Medical Image Database, which collects NHS Breast Screening Programme (NHSBSO) images in the UK.

A deep learning approach is used to classify abnormal screenings as either malignant or benign cancer with a certain probability. Transfer Learning makes it possible to obtain high performances on small datasets.

This project achieved a ROC of 80%, 86% sensitivity, and 77% NPV, which were reached with a pre-trained ResNet50v2, a state-of-the-art neural network optimized through fine-tuning hyperparameters and data pre-processing.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Breast Cancer is the second leading death cause of death from cancer in women, but early detection and treatment can considerably improve outcomes [1]. Despite the surge in mammography screenings by women, detecting cancer in images remains a challenging task. Thus, there is still improvement to be done to reduce the number of false positives which leads to stress and unnecessary biopsy. Moreover, false negatives at screening are very problematic since they are often identified at an advanced stage and thus, less curable during treatment.

Computer-aided detection (CAD) software for mammography was introduced in the 1990s and have failed to meet medical expectations. However, deep learning has recently been able to fulfil this need for improvement with the ability to exceed the performance of human experts on medical-image analysis [1].

This project evaluates the performance of a machine learning system to predict the likelihood of breast cancer based on screenings. The dataset comes from the OPTIMAM Medical Database (OMI-DB) and is collected by the NHS Breast Screening Program (NHSBSP) which gathers screening images from different breast cancer centers across the UK [2].

To conclude, a lot of efforts have been made to improve the deep learning architecture and parameters fine-tuning to build neural networks with lower false positive and false negative rates.

1.1 Background and Context

This section is divided into 4 sections and covers the prior necessary knowledge to understand the motivations and concepts behind this health issue. Firstly, statistics on *breast cancer* are mentioned. Then, the 3 most used *medical imaging* technologies are briefly presented. As third point, *Radiomics* harnesses the capabilities of deep learning. Finally, the *dataset* and its splits are introduced.

1.1.1 Breast Cancer

"*One in seven women* in the UK will develop breast cancer in their lifetime. *Eight out of 10 cases* of breast cancer are diagnosed in women aged 50 and over. Breast cancer is the *fourth most common cause* of cancer death in the UK. Breast cancer is a *leading cause of death in women under 50 in the UK*" [3].

"Almost nine in 10 women survive breast cancer for five years or more. Breast cancer survival is improving and has doubled in the past 40 years in the UK due to a combination of improvements in treatment and care, earlier detection through screening and a focus on targets, including faster diagnosis" [3].

Mammogram are used for screening breast cancer. Figure 1.1 illustrates the different textures in

breasts and its corresponding mammogram.

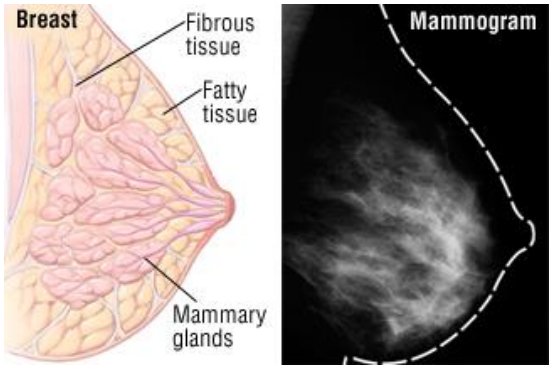

Figure 1.1 – Breast anatomy and Mammogram [4]

The mammograms are used to detect anomalies in breasts. In this case, the doctor will need to do a biopsy to check if the anomaly is malignant (cancerous) or benign (not cancerous) by extracting cells with a needle and proceeding further examinations with the microscope [4].

As depicted on Figure 1.2, the two images of 2 pairs of breasts are taken with planar X-rays imaging technique, which captures fine details of the different textures in breasts as 2 dimensional images despite the breast density. The brighter the white tissues, the denser they are since white areas capture more electromagnetic waves than darker areas.

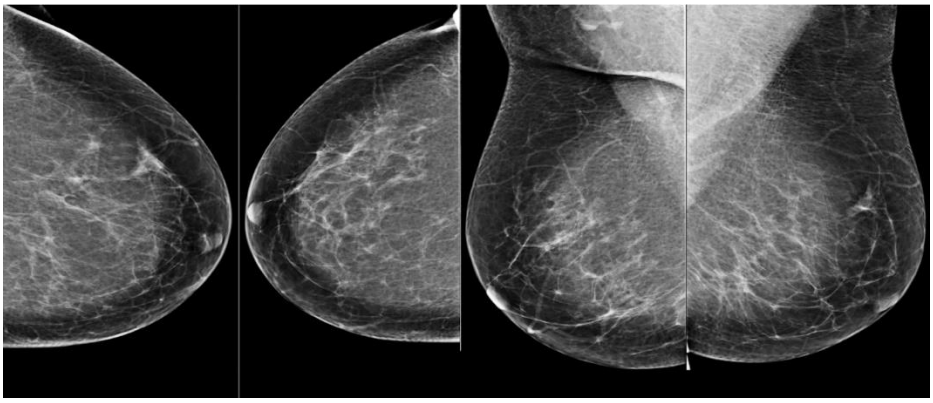

Figure 1.2 - Screening mammograms [1]

Figure 1.3 illustrates how women are positioned during the X-ray image acquisition with a compression plate keeping the breast static.

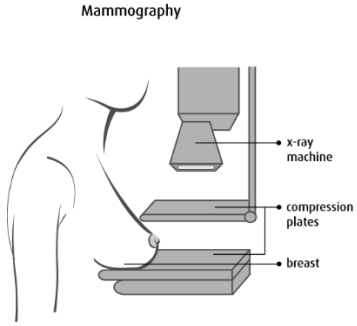

Figure 1.3 - X-rays mammography [5]

1.1.2 Medical Imaging

Three of the most used medical imaging techniques are X-rays, Computed Tomography (CT) and Magnetic Resonance Imaging (MRI).

X-ray imaging is a transmission-based technique that uses *X-rays* from an emitter (source), goes through the patient, and are absorbed by a surface that detects the quantity of electromagnetic particles received and transform the energy to an image. A planar *X-ray* image, as shown in Figure 1.4, is an X-ray image that stack the depth of the breast to form a 2 dimensional image. This can make it difficult for radiologists to interpret the overlapping layers of soft tissues and calcifications. This issue is fixed with X-ray computed tomography (CT) scans [6]. Note that, X-rays imaging use harmful electromagnetic radiations that can take a radiography of the density of body parts e.g. mammography.

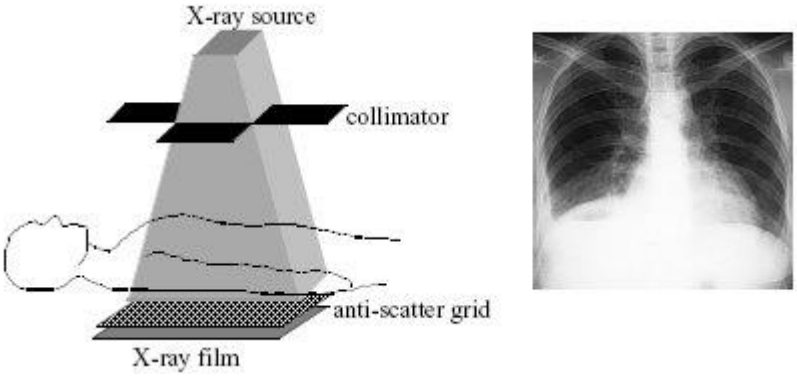

Figure 1.4 - Planar X-ray Imaging [6]

Computed Tomography (CT) are also using x-rays technology but adding one more dimension to have 3d volumes that are a stack of many 2-dimensional slices. The emitting energy source and detector are simultaneously rotating around the patient to capture the image from different angels as illustrated on Figure 1.5.

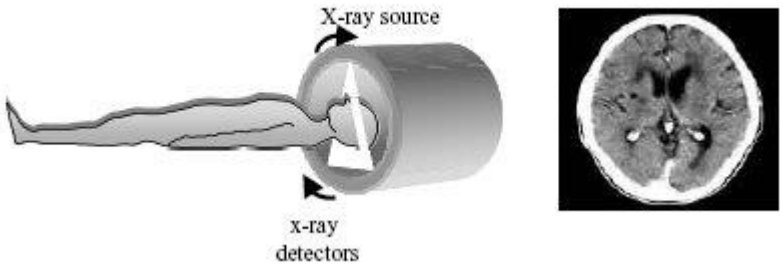

Figure 1.5 - X-ray Computed Tomography (CT) and patient's slide of head [6]

Magnetic Resonance Imaging uses magnets to detect protons movements. It is not damaging and is well suited for non-bony parts or soft tissues e.g. shoulder ligaments or the brain. However, CT scans are more expensive than X-rays imaging.

1.1.3 Radiomics using Deep Learning

Radiomics uses machine learning and image processing techniques to extract quantitative textural metrics from radiological images. By examining those features, it is possible to diagnose pathologies and interpret fine tissues characteristics. *Radiomics* can be used for discriminating benign and malignant breast tumors [7].

Deep learning (DL) has confirmed its effectiveness at competing with radiologist's performances [1]. For example, using radiomics based predictive models on pre- and post- treatment computed tomography (CT) images to indicate if the chemotherapy response on bladder cancer is complete. Pattern recognition methods such as deep learning convolutional neural networks (DL-CNN) are competing with radiologists [8].

1.2 Breast Cancer Detection [9]

Improving breast cancer detection at an early stage yields the following advantages:

- Improved patients' outcome by reducing the false negatives, which delay diagnosis and treatment, would help detect breast cancer at an early stage.
- Reduced healthcare costs by reducing false positives, which lead to unnecessary procedures.

1.2.1.1 Consequences of Dense Breast Classification

Women with increased breast density have a higher volume of glandular tissue, which makes it difficult to detect non-calcified lesions, and is comparable as finding a polar bear in a snowstorm.

1.2.1.2 Challenges Faced by Both Healthcare Providers and Patients

When facing abnormal screenings, healthcare providers need to decide on either "wait and see" or further examinations. This leads to a higher sensitivity but could also mean a lower specificity with a rise in anxiety for patient without cancer. Another challenge are the anxious patients in the "gray zone" insisting on biopsy, which increases false positives. Also, the lack of solutions for patients with dense breasts. And reducing the number of high-risk invasive surgical procedure when biopsy is not enough.

1.2.1.3 The Impact of Overdiagnosis (False Positive and False Negative results)

"The financial burden of further diagnostic workup associated with false positive mammograms, including out-of-pocket expenses, contributes to the emotional stress and anxiety imposed on a patient and their family" [10]. Moreover, overdiagnosis of breast cancer exposes patients to the harm of overtreatments where in fact the breast lesion is unlikely to become malignant. However, excepted the highly increased chances of survival for early detected cancers, the costs of treatment are also much lower between a stage 1 and 4 breast cancer in the US.

The emotional impact of overdiagnosis has, in addition from the health anxiety, an indirect cost such as missed work, and financial stress. False negative has also an emotional impact since a stage 4 is more stressful than early detected cancers with high chances of survival.

1.3 Objectives

The following objectives have been defined to reach clear targets:

1. Explore new methods and latest achievements of deep learning models on medical imaging.
2. Get a baseline result with pre-trained Neural Networks i.e. AlexNet using Transfer Learning.
3. Build and test other types of neural networks and compare results.
4. Improve results with hyperparameters fine tuning and regularization techniques.
5. Evaluate and comment the results
6. Present limitations and suggest paths of improvement.

1.4 Achievements

The content of this report gathers the work that has been achieved which can be summarized as:

1. Read the literature and become familiar with the breast cancer, biomedical and radiology domains.
2. Accessing the data and data pre-processing.
3. Design and run an end-to-end predictive model as baseline.
4. Use transfer learning to improve model's performance
5. Select the best model and fine-tune its hyperparameters to improve classification.
6. Evaluate and comment on the results.

1.5 Overview of this Report

This report is articulated into 4 chapters. First, it introduces background information as the prerequisites domain knowledge to understand the basics of medical imaging and breast cancer. Secondly, the in-dept literature review has been summarised into 4 sub-sections, namely *neural networks, convolutional neural networks, transfer learning and evaluation metrics.* As third chapter, the *methodology and developments* of modelling the neural networks are presented. Finally, this report explains in a detailed way the *data engineering, data modelling and evaluation* that has been applied.

2 BACKGROUND THEORY AND LITERATURE REVIEW

This chapter aims at providing the prerequisite knowledge to understand the theoretical concepts behind the practical applications in the next chapter and is divided into 3 sections. First, *Neural Networks'* key building blocks are described in detail. Secondly, *Convolutional Neural Networks* are a special type of deep neural networks used with images, in this case medical imaging. As third section, the motivations of using *Transfer Learning,* when working with small datasets, are presented. Finally, *Evaluation Metrics* are required to assess classifiers' performances, especially in machine learning.

This chapter is highly inspired from the Computer Science Course from Stanford University which includes the complete in-depth online course notes and the detailed syllabus from 2015 to 2020 [9].

2.1 Neural Networks

Neural Networks can be described as a function approximator inspired by the human brain [10]. It relies on many concepts presented in the following sections.

2.1.1 Loss function

The *loss* is a quantity to track during training, it measures how bad your model is performing. A *function*'s output can be chosen as the loss.

2.1.1.1 Softmax function

This project aims at predicting the likelihood of a patient with abnormal screenings to have cancer or not. This likelihood is obtained with normalized class probabilities. The softmax function is a generalization of the sigmoid binary function to multiple classes. Here, a brief explanation of the concepts and intuitions behind this function are given.

First, let us define a *loss function* to compute the classification error. The softmax classifier uses the *logistic function* as loss function but generalized to multiple classes. Not to confuse with the *sigmoid function* which is the binary version of the logistic function.

$$Sigmoid(x)\ or\ \sigma(x) = \frac{1}{1 + e^{-x}} \qquad 2.1$$

Where x is the output value from the linear function $f(x_i; W) = Wx_i + b$ that takes as input W for the weights, x as the input/output value of each neuron and b as the bias.

This function provides a probabilistic interpretation as the probability of classifying an image with the true or false label where the sum of both probabilities equals one.

2.1.1.2 Cross-entropy or log loss

Measures the performance of a classifier by calculating the difference between the predicted output's value, which ranges between 0 to 1, and the true label value 0 or 1. The shape of the log loss function strongly penalizes wrong predictions that are very confident. For example, predicting class probability 0.01 whereas 1 is the true class [11].

The formula for the Cross-entropy or log loss:

$$\mathcal{L}_{CE}(y, \hat{y}) = -[y.log\ \hat{y} + (1 - y)\ log(1 - \hat{y})] \qquad 2.2$$

Where y is the true label and $\hat{y}$ the predicted probability class.

With $\hat{y} = \sigma(w.x + b)$ the predicted class probability from the sigmoid function defined in:

$$\text{Sigmoid(x) } or\ \sigma(\text{x}) = \frac{1}{1 + \text{e}^{-\text{x}}} \qquad 2.3$$

Sigmoid(x) or $\sigma(x) = \frac{1}{1 + e^{-x}}$ 2.1 Now, the *cross-entropy* or *log loss* formula is written as,

$$\mathcal{L}_{CE}(w, b) = -[y.log\sigma(w.x + b) + (1 - y).log(1 - \sigma(w.x + b))] \qquad 2.4$$

To conclude, the cross-entropy is broadly used for evaluating the classification error in neural networks because it uses log probability as the sigmoid function does. The *Hinge* function is also often used for classification and worth to mention as another loss function.

2.1.2 Stochastic Gradient Descent

Now we need to optimize the weights to minimize the classification error from the loss function. This is done by finding the gradient i.e. the slope or derivative related the loss function's search space. For multiple dimensions we refer to partial derivatives forming the gradient vector. Gradient Descent is the fact of computing iteratively the derivate and update the weights parameters. Using batches of data updates the weights more often and refers to Stochastic Gradient Descent (SGD).

2.1.3 Backpropagation

Backpropagation is the backbone of gradient descent for neural networks. It relies on the *Chain Rule* which is a set of gradients multiplications used to update the weight parameters after each feedforward propagation. These mathematical expressions are critical to understand how Neural Networks can learn from their classification errors during training. This thesis project will not cover the chain rule but encourage those that are not familiar to it to read more about it.

2.1.4 Data Pre-processing

Mean Subtraction is often used for translating the data to the origin e.g. subtract the mean for every pixel in an image.

Normalization is very useful for putting different features on the same scale if it makes sense to

do so. It is also computationally less expensive to work with floats between 0 and 1 than in range from 0 to 255 with regards to images. Thus, it is common to divide pixel values by 255.

PCA (Principal Component Analysis) is a feature extraction that finds the most discriminative dimensions and is often very useful during pre-processing to reduce the variance in the data and leads to less complexity in the model. Note that PCA is not relevant for ConvNets since Neural Networks include the feature extraction during training, and thus is only mentioned for completeness.

Finally, pre-processing of the data *must only be calculated on the training set* and applied on validation and test sets i.e. it is a mistake to compute the subtracted mean on the whole data.

2.1.5 Weights Initialization

It is necessary to use random *weights initialization* to avoid computing all the same gradients and parameter updates. Therefore, it is recommended to use a specific variance of neurons with ReLU activation functions as described by He et al [12].

2.1.6 Activation functions

The reason we need to use *activation functions* is because of the linearity of the formula to compute neuron's input as the dot product between an input vector and a weight matrix such as,

$$\mathrm{Z} = \sum_{i=1}^{n} x_i w_i \qquad 2.5$$

Neural Networks need to capture complex patterns in data that are non-linear. Thus, we need to insert non-linear functions in the neural network. Moreover, activation function's outputs are continuous values which makes it easier to compute the gradient and fixed range to keep the model stable and prevent it to explode with very high or low values.

2.1.7 Batch Normalization

This step is critical for image classification models as described by Ioffe and Szegedy [13]. *Batch Normalization* addresses the problem of nonlinearities of layer's inputs by first normalizing each pre-activation in each layer according to the mean and standard deviation across the mini batch. Then, normalized pre-activations are scaled and shifted by a per-feature coefficient γ and bias β. This gives significant better accuracy on results as demonstrated by Jonathan Frankle et al [14].

2.1.8 Regularization hyperparameters

Regularization hyperparameters refers to a set of techniques that prevents Neural Networks to overfit.

2.1.8.1 Common Regularization parameters

L2 regularization penalizes prominent weights and ensure weights update linearly. *Dropouts* retain

some neurons out of the network with some probability p (hyperparameter) Srivastava et al. [15]. Finally, *L1 regularization*, *max norm constrains,* and *Bias regularization* are other effective regularization hyperparameters worth to know about.

2.1.8.2 Epochs and Batch size

The number of *epochs* tells how many times an observation has been seen during training. Epochs are usually set as the x-axis when fine tuning hyperparameters. Whereas the *batch size* is the number of training samples to feedforward through the network at a time. Thus, the number of batches is equal to the number of samples in the training dataset divided by the batch size. Finally, once all training batches have been propagated through the network one time, one epoch has been completed.

Advantages of using batches are to use *less memory* since the machine needs to store fewer samples' parameters and *training faster* since the networks updates weights at every propagation i.e. number of batches in one epoch.

2.1.8.3 Learning rate and Optimizers

The *learning rate* tells how much to update the weights at each backpropagation. A high rate gives a lot of speed to the particle and will likely rush into the first local sub optimal valleys of the loss function's search space. On the other hand, a low learning rate will miss the global optimum valley. Thus, we need a trade-off which is to decay the learning rate while searching for the best solution.

Step decay reduces the learning rate by a constant value at every certain number of epochs. The following techniques are in practice more effective. They use a local parameter update, like the *RMSprop* which is based on the adaptive learning rate method [16]. Finally, *Adam* is also an effective optimizer largely inspired from the RMSprop but with momentum [17].

2.1.8.4 Momentum

This parameter can be described as the coefficient of friction to reduce the velocity of the particle moving in the loss function domain, looking for the global optimum. In practice, the *momentum* gives more velocity to a direction that has consistent gradient. The *Nesterov momentum* is now recognize as giving better performance than the classic momentum and is worth to mention for completeness.

2.1.8.5 Hyperparameters optimization

As demonstrated by Bergstra and Bengio, *random search* leads to better results rather than grid search [18]. If the final best parameter value is on the edge of the random interval, it is advised to expand this range to search for more optimal values. Moreover, going from a large to narrow range makes it easier to find the global optimum.

2.1.8.6 Ensembles

Combining different models' outputs and take their average as the final prediction is a very effective way to improve performances regarding the generalization ability of the model especially, neural networks ensembles for melanoma classification as argued by Fábio Perez et al [19].

To conclude about regularization hyperparameters, the most used ones are the *learning rate value* and its *decay* mode, the *L2* penalty and *Dropouts.* Regarding the parameter's optimizers, *SGD* with *Nesterov Momentum* and *Adam* are the currently best to start with. Next, use *random search* for fine tuning hyperparameters and funnel the interval. Finally, ensembles models will slightly increase your performance.

2.2 Convolutional Neural Networks

Convolutional Neural Networks are the backbone of visual recognition with deep learning. This section presents the concept of *Convolution* in image processing, then the *architecture of CNNs and the different* hyperparameters. Finally, *Backpropagation* is critical to understand how CNNs can learn.

2.2.1 Convolution

Generally, convolution is used for extracting features from images. This is performed with a small window called kernel, sliding through the image to find some lines, shapes, objects, …, which can form cancers. The kernel is a matrix that multiplies pixel values to form an output image with highlighted features.

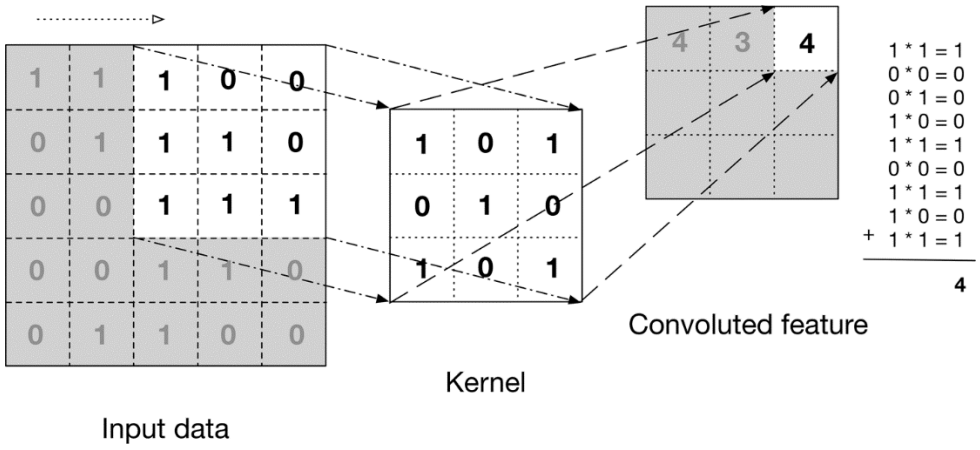

Figure 2.1 – Convolution with kernel [20]

For example, Figure 2.2 shows 12 filters per convolution layer after training. We can observe the increase in complexity from left to right as the neural network becomes deeper (more layers). In particular, the first layer captures the colour information while the last 2 layers represent shapes.

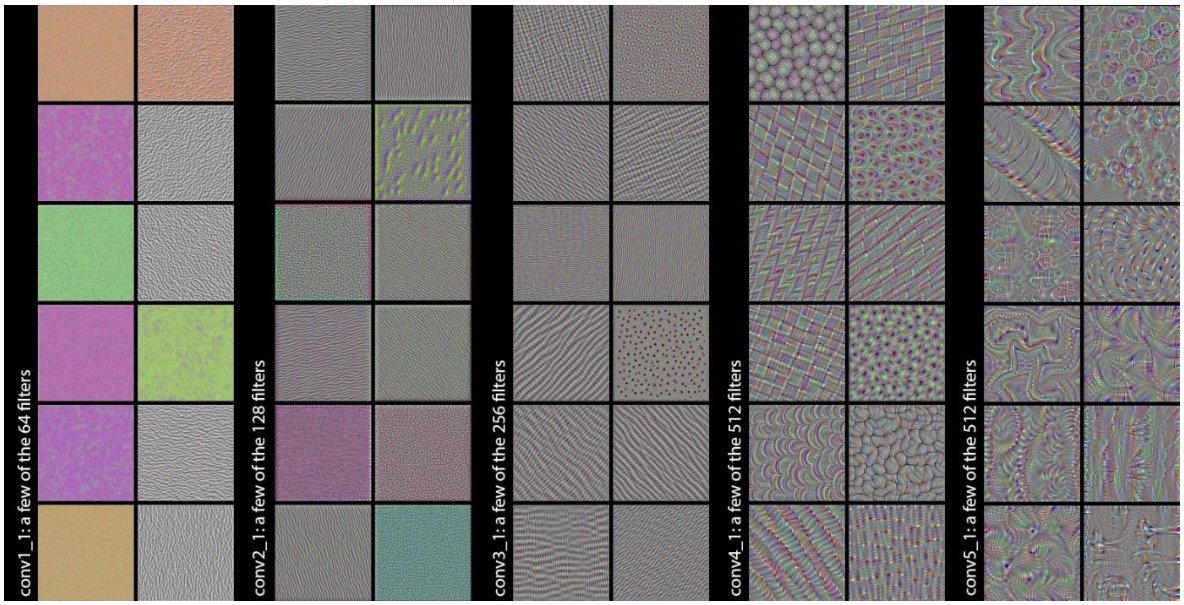

Figure 2.2 - How convolutional neural networks see the world [21]

2.2.2 Architecture of a CNN

ConvNets are a special type of Neural Networks, they assume the input are images. Weights are not scalars between neurons but filters between convolutions, neurons from a hidden layers are not fully connected to all previous neurons but instead are 3D volumes with dimensions: width, height and depth, and layers are not a pile of neurons but a stack of different sub-layers: Convolution Layer, Pooling Layer and Fully-Connected Layers.

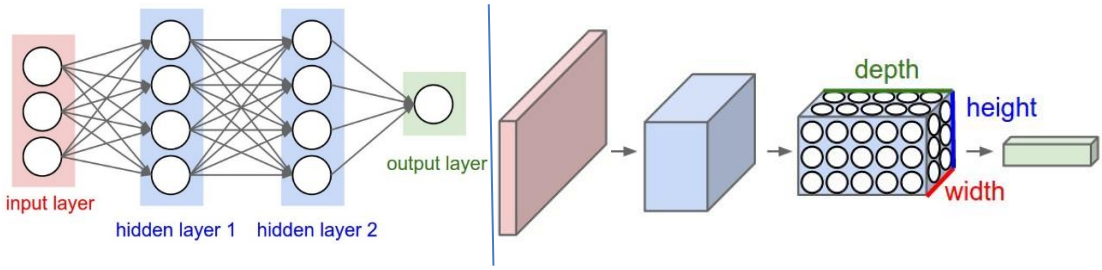

Figure 2.3 - Regular Neural Network and ConvNet [9]

2.2.2.1 Input Layer

The input layer receives as input an image which should have an input size divisible by 2 many times regarding its width and height.

2.2.2.2 Filters, Neurons and Weights

Neurons are the filters and the weights are the values in those filters which are learned during training. Filters have 3 dimensions and have same depth as the input volume to convolve through its full depth. Using dot product between the input volume and filters, we get a 2D activation map. By stacking all those activation maps for each filter, we create a new output volume of depth equal to the number of filters [9].

2.2.2.3 Convolution Layers

Computes the output by convolving a kernel through the image as depicted on Figure 2.1. The number of filters used will determine the depth of the output volume.

2.2.2.4 ReLU Layer

The Rectified Linear Unit activation function such as the $max(0, x)$ keeps the same input and output size and outputs positive values.

2.2.2.5 Pool Layer

This layer reduces the width and height of the 3D volume and keeps the depth unchanged. Empirically, the best results are found by selecting the maximum values in a 2 by 2 sliding window. This enable to reduce the number of parameters to 25% and control overfitting. Note that the pooling occurs on every activation map independently.

2.2.2.6 Fully Connected layers

Neurons in this layer are connected to all the neurons in the previous layer, similarly to a classical NN. Often added as last layer to classify the input with one of the labels. The output volume is a vector of depth n-classes. Where each class is fully connected to all the previous neurons i.e. filters.

2.2.2.7 Stacking all those Layers

This pattern is typical for ConvNets and can be modified with the parameters N, M and K. Usually, the FC layer is applied when the output volume has shrunk to a small window.

INPUT -> [[CONV -> RELU] *N -> POOL] *M -> [FC -> RELU] *K -> FC

Figure 2.4 – CNN Layers.

Where N <= 3, M >= 0 and K > 3.

To conclude, the main difference between classical neural networks and CNNs is the number of parameters i.e. weights to compute between each neuron. ConvNets are using sliding filters as output neurons instead of having fully-connected pixels $[width\ x\ height\ x\ depth]^2$ between 2 layers. Whereas a CNN has in its n^{th}-volume: [width x height x depth] neurons and each has filter: [width x height x depth] weights. Which is high but still less than a fully connected NN. Note that in practice we use *parameter sharing* to keep weights the same in each depth layer and are updated only after each filter, depending on the objects to classify.

2.2.3 Layer Hyperparameters and Layer Sizing

In the previous sections we presented what convolutions are and the architecture of a CNN, this part explains how the size of the output volume is determined by the hyperparameters.

2.2.3.1 Depth, Stride and Padding

Those 3 hyperparameters are defining the size of the output volume.

The *depth* of the output is set by the number of filters that are convolved on the input volume. Each filter creates an activation map that is stacked along the 3 axes.

The *stride* corresponds to the number of steps of the sliding filter when convolving e.g. stride 1 does the dot product for every pixel whereas stride 2, skip one out of two pixels and thus, divide the input width and height by 2.

Finally, the *padding* enables the filter to convolve on the borders of the input by adding zeros rows and columns to preserve the width and height of the input.
The formula to computes the output shape is given by *(Width – Filter + 2xPadding) / Stride + 1*. Note that the result must be an integer to be valid.

2.2.4 Backpropagation

This algorithm makes it possible for the neural network to learn from the data by updating its weights while the data is trying to classify each input. For a CNN, the weights of the layers are local neurons in a volume which are in fact stacked activation maps.

2.3 Transfer Learning

Datasets are often too small to extract representative class features without overfitting the data. Since training does also take enormous amount of time despite the increasing computing power available, it is a good practice to start with pre-trained weights rather than starting from random weights. Hopefully, learned weights trained on huge datasets with thousands of classes have shown great results when transferred to other datasets. This is due to the similarity between most features extracted at early layers regardless of the classified objects. However, there are a few rules to be aware of before using transfer learning.

On the one hand, it is strongly recommended to use transfer learning if your dataset is small and the classes to predict are present in the dataset that trained the transferred weights. On the other hand, if your dataset is totally different and does not contain similar classes e.g. medical imaging for tumours, microscopic data, etc., it is great to have a reference dataset with pre-trained weights on malignant or benign tumours.

Although the architecture of the pre-trained model can lead to constraints if the number of layers is different, the input volume is valid with the transferred weights if the kernels' stride fits the convolution.

It is possible to freeze layers during training i.e. prevent weights to update with the gradient, since it is supposed that the weights are already optimal. In practice the last layers are not frozen to let the neural network adjust the weights to its input dataset. Note, that the last fully connected layer must stay trainable to discriminate the classes.

Finally, the learning rate should be cautiously set to a low value since we consider to uploaded weights as almost optimal and do not want to radically changes them. Therefore, the learning rate should slightly fine tune the weights by being set to low.

2.4 Evaluation Metrics

In machine learning, the performances of predictive models and classifiers are evaluated with a set of specific metrics that makes it possible to compare results. This section is divided into 4 parts. First, a *Confusion Matrix* is an easily computed and explainable table that aggregates the classifiers' predictions in four categories that can be used to compute other metrics. Secondly, the sensitivity and specificity among other metrics are presented for *Binary Classification*. Finally, the *AUC-ROC Curve* provides a single metric and a plot that uses the Confusion Matrix elements.

2.4.1 Confusion Matrix

A Confusion Matrix aggregates all the predictions in the data set into four classes by crossing the actual class with the predicted class as illustrated in Table 2.1.

The four classes are described as:

- *True Positives*: Correctly classified as positives.
- *True Negatives*: Correctly classified as negatives.
- *False Positives*: Incorrectly classified as positives (negative class).
- *False Negatives*: Incorrectly classified as negatives (positive class).

Thus, the first argument represents the correctness of the prediction and the second the predicted value as illustrated in Table 2.1.

	Predicted Positive Class	*Predicted Negative Class*
Actual Positive Class	True Positive	False Negative
Actual Negative Class	False Positive	True Negative

Table 2.1 - Confusion Matrix

2.4.2 Binary Classification

Binary classification metrics uses the elements of the Confusion Matrix as inputs and are listed in the Table 2.2. All those metrics are proportions expressed as ratios and should be converted to percentages. Typically, Sensitivity and Specificity are chosen to evaluate a binary classifier.

Metrics	*Formula*	*Description*
Accuracy	$\frac{TP + TN}{TP + FP + TN + FN}$	Correct predictions over the total classifications.
Error Rate	$\frac{FP + FN}{TP + FP + TN + FN}$	Wrong predictions over the total classifications. Error Rate = 1 - Accuracy
Precision	$\frac{TP}{TP + FP}$	Positives correctly classified over total predicted as positives.
Recall *Sensitivity / TPR*	$\frac{TP}{TP + FN}$	Positives correctly classified over total positives.
Specificity	$\frac{TN}{TN + FP}$	Negatives correctly classified over total negatives.
F-Measure	$\frac{2\ x\ Precision\ x\ Recall}{Precision\ +\ Recall}$	Equal importance between Precision and Recall.

Table 2.2 - Classification Metrics for Evaluation

For example, if all observations are classified as positives, then the *Sensitivity* equals 100% because all positive values have been found and there are no false negatives. Whereas, if all observations are classified as negatives, then the *Specificity* equals 100% since all negative values have been found and there are no false positives.

The *F-Measure* balances the importance of Precision and Recall. It is the case where the *F(β)-Measure* uses a coefficient β equal to 1. This coefficient is useful if having false negatives is not as critical as false positives, and inversely. Thus, we want to give more weights to either Precision or Recall.

2.4.3 AUC - ROC Curve

Binary classification problems are very often evaluated with the AUC–ROC Curve. Where the *AUC (Area Under the Curve)* is a performance metric computed with the *ROC (Receiver Operating Characteristics) Curve*. This section aims at explaining this metric and the relations between Sensitivity, Specificity and Thresholds.

2.4.3.1 Terms and Relations

The *Threshold* is a value that corresponds to the decision boundary used to classify observations with class 0 or 1. This boundary is a probability value between 0 and 1 that splits the 2 probability densities functions (pdf) in two predicted class labels, one pdf for each class label in binary classification. This decision boundary can be calculated with the optimal *Bayes error,* but we will not cover this theorem in this project. Thus, the values in the confusion matrix depends on this threshold which are the input values for *Sensitivity* and *Specificity.* As a result, the *Sensitivity* and

Specificity change inversely proportionally when we move the continuous value of the threshold from 0 to 1.

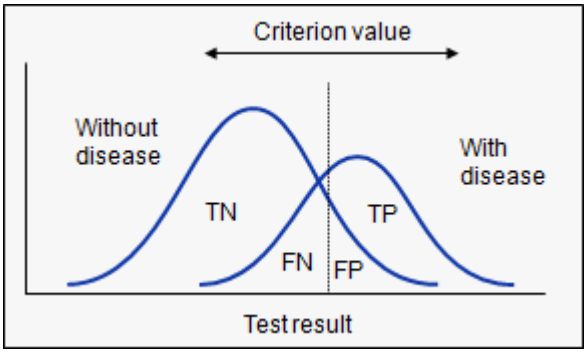

Figure 2.5 - Probability densities function for 2 classes and the decision boundary.

For example, as illustrated in Figure 2.5, when we decrease the threshold (criterion value), we have larger positive area 'with disease' than negative area 'without disease' which increases sensitivity, and inversely when increasing the threshold.

A good model has an AUC close to 1 which means it classifies almost without misclassifications. On the contrary, a bad model has an AUC near 0 which means it classifies almost all observations with the opposite true class. If the AUC is 0.5, then it is a random binary classifier.

As illustrated on Figure 2.6, Sensitivity decreases, and Specificity increases when the criterion value increases. For instance, this can be explained by larger True Negative and smaller False Positive values for the Specificity. This results in a perfect inversely proportional relation between Sensitivity and Specificity when the 2 classes are two equiprobable and equivariant gaussians.

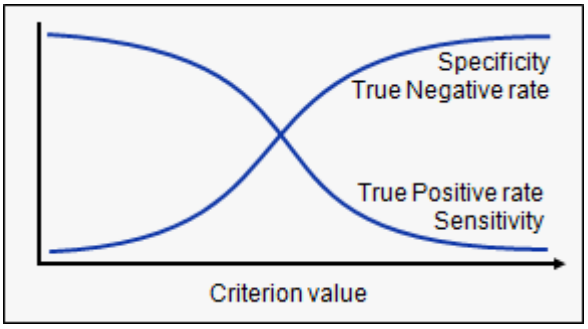

Figure 2.6 - Inverse relation between Sensitivity and Specificity.

Finally, as depicted on Figure 2.7, the blue line should stay close to the upper left corner to maximize the *Sensitivity* and *Specificity* simultaneously which can be interpreted as having very low False Positive and False Negative values in the *Confusion Matrix* for a given *Threshold*.

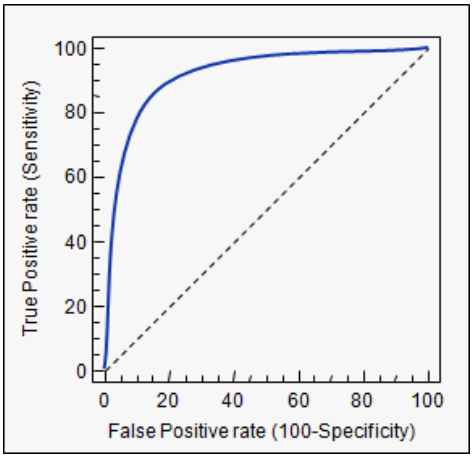

Figure 2.7 - AUC – ROC curve

Sensitivity is on the vertical axis and 100-Specificty on the horizontal axis. The dotted line is the random classifier or 0.5 AUC.

2.5 Evaluation Summary

To conclude, this chapter 2 first introduced the fundamentals of *Neural Networks,* then covered the *Convolutional Neural Networks* building blocks for visual recognition by defining a convolution, the network architecture, the hyperparameters and backpropagation. Next, *Transfer Learning* makes it possible for neural networks to get good results despite small datasets by using already trained weights on huge datasets with thousands of classes. Finally, Sensitivity and Specificity were illustrated with other metrics as it is critical to use the pertinent *Evaluation Metrics* when assessing results for a particulate classification problem.

3 DATA ACCESS, METHODOLOGY AND TOOLS

This chapter covers the *data access, methodology, and tools* for this project. It is divided into 5 sections. First, *accessing* the *data* can sometimes be challenging especially when it is done remotely, on a secured server and inherited by another user. Secondly, the technical choices for *Hardware and Software* are stated. Next, the *Neural Network design and training* baseline for further improvement is presented. As third point, *transfer learning* is showing better results on the dataset. Fourthly, *fine tuning hyperparameters* increases the model's performances. Finally, the *best model and hyperparameters* are selected based on evaluation metrics.

3.1 Data

The first steps were to access the dataset on the server from a personal laptop while being off campus. Those steps are often underestimated when building the project's timetable. Indeed, launching or carrying out someone else's work is usually not a smooth as an IT project comes with its firewalls and inherited legacies.

3.1.1 Accessing the data

3.1.1.1 VPN

It is interesting to understand the motivation for using a Virtual Private Network and being able to use it for future projects, especially when working on sensitive data. Thus, it was necessary to install the VPN software on the laptop and access the University's secured network to connect on the server as if the laptop was on-campus. Indeed, to access the server it is required to be connected to the university's local internet network. Hence, a secured tunnel was required.

3.1.1.2 Server

Once the connection through the VPN was established, it was possible to access the servers on the terminal with the "ssh" command which is also a secured encrypted connection between 2 hosts. After that, finding the dataset on the server took time since the right person who knew its location was not met yet, but then a meeting to get to the data was planned. Unfortunately, it turned out that the access to the full path was secured and required authorizations. Hopefully, the IT services did a great job at helping everyone despite the tensed situation due to the Covid-19 crisis.

As a result, learning to work remotely on secured channels and using the terminal with all its Linux commands are very useful skills to master.

3.1.2 Importing the Data

The raw images were formatted as TIF files (Tagged Image File) with no compression and with red, blue, and green (RGB) colour space. This format uses the '.tif' extension, which is a common-

ly used format in biomedical imaging, and is imported using the tiff file python library.

The dataset size has been increased with a combination of rotations and flips from the original patches which are patches extracted from mammograms where each patch has a patient id, and are named as either bilateral craniocaudal (CC) or mediolateral oblique (MLO) which are standard views in mammography screenings.

The reason the images are dark is because the density of the structure of the object is very low as air appears black, bones are white and soft tissues in-between i.e. grey.

3.1.3 Data Pre-Processing

The TIF files are encoded as 16-bit integer which makes it necessary to normalize the pixel values in the arrays by 65,535 rather than 255.

Two approaches have been considered for building the dataset with class labels for each image. The first one is using one hot encoding, the second one is using binary classes which has been retained as it is more adapted for a binary classification problem with a Sigmoid activation function in the output layer and having one threshold for the only output neuron, instead of 2.

Since the patches from the original images are already extracted, it is not required to use patch extraction techniques.

To zoom in may causes in some cases to miss the class object in the patch, they are already a close-up version of the mammograms.

3.2 Deep Learning Hardware and Software

Although PyTorch has a clean API, a growing adoption, number of users and recognition among the research community. To get a quick start with an intuitive language and a large online community, TensorFlow was chosen over PyTorch. The Keras API is particularly user friendly and has all the necessary features for this project.

Being able to use the laptop's GPU speeds up the running time on the machine. The Spyder IDE (Integrated Development Environment) was used to import and explore the data as well as to design the neural networks architecture. This IDE is built for Python and is very convenient for debugging. Another way to increase the training speed is to use the Google Colab's free GPUs and TPUs which are Graphics Processing Units and Tensor Processing Units, respectively. For this project, Google Colab was selected as an easy to use, powerful and free platform for developing.

3.3 Preliminary Designs and Training Neural Networks

This section focuses on the *design and the training* of multiple neural networks to reproduce the state-of-the-art with well-known architectures. As listed in Table 3.1, these neural networks have been designed and trained:

Neural Networks	*Description*
AlexNet	1st significant improvement on the ImageNet dataset in 2012. It Has 5 Convolutional (Conv) Layers and 3 Fully Connected Layers.
VGG16, VGG19	Small kernels with 16-19 Conv Layers outperforming AlexNet in 2014.
ResNet50	Hundreds to thousands of Conv Layers and no "vanishing" gradient in 2015.

Table 3.1 - Neural Networks [22]

AlexNet and VGG16-19 are easily designed since they are reasonably shallow networks compared with ResNets which can be imported from the Keras library. For this experiment, weights were randomly initialized.

During training, an early stopping condition has been set with patience equal to 3 by monitoring the validation loss progress at each epoch.

3.4 Transfer Learning

When using *transfer earning* the performances increase significantly as the weights are already optimal for extracting most of the interesting features. Thus, weights learned on the Imagenet dataset were loaded in the networks and "frozen" but not for every layer. Indeed, the last fully connected layers should stay trainable to fit the model to the class labels from the mammogram dataset (benign and malignant) instead of the Imagenet classes. The weights are stored in an open source '.h5' file and are easily added to the model by mentioning Imagenet as an argument. An h5 file is an HDF (Hierarchical Data Format) file containing multidimensional arrays of data i.e. arrays of weights for each layer.

3.5 Model Evaluation and Selection

The best model will be evaluated and selected based on the AUC-ROC Curve as it considers the Sensitivity and Specificity. The F-β-Measure might also be considered as it makes it possible to give more weights to the Recall or Sensitivity. Remember that False Negatives are very dangerous when predicting or evaluating the likelihood of having a disease i.e. having a malignant cancer, we want to avoid this scenario.

4 DATA ENGINEERING, MODELING AND EVALUATION

This chapter presents first the *data engineering* steps which are the *data exploration, data splitting,* dealing with class imbalance, *data pre-processing* and *data augmentation*. Next, the *modelling* part focuses on building and selecting the most suitable neural network architecture for this classification task. Then, the *hyperparameters* have been *fine-tuned* through the training and validation of the model yielding the best performances on the training and validation set. Finally, the model's performances are *evaluated* and explained.

4.1 Preventing Data Leakage

The training, validation and test sets have been created based on "patient_id" and while checking for patient overlap between training and test sets. It is necessary to avoid data leakage between the training and test sets since it would memorize the class answer for a certain patient during the learning phase and recognize the same patient during the testing. Therefore, patients have been allocated to only one of the 3 subsets.

4.2 Data Splitting

The data splits have 50% of the positive class in the validation and test sets. The classifier's performance is then not influenced by the probalistic distribution of the class labels during evaluation. For example, if the dataset has 90% of benign scans and the classifier learns to predict only negative scans then the accuracy is high but would perform weakly if the test set is balanced. Therefore, it is critical to balance to validation and test sets to have an unbiased performance. Figure 4.1 and Figure 4.2, illustrate the extracted raw patches from 25 benign and 25 malignant mammograms.

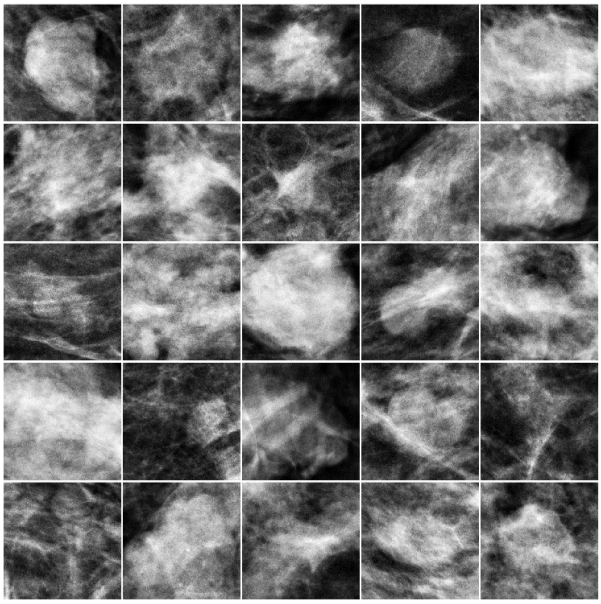

Figure 4.1 – Random images from the negative class.

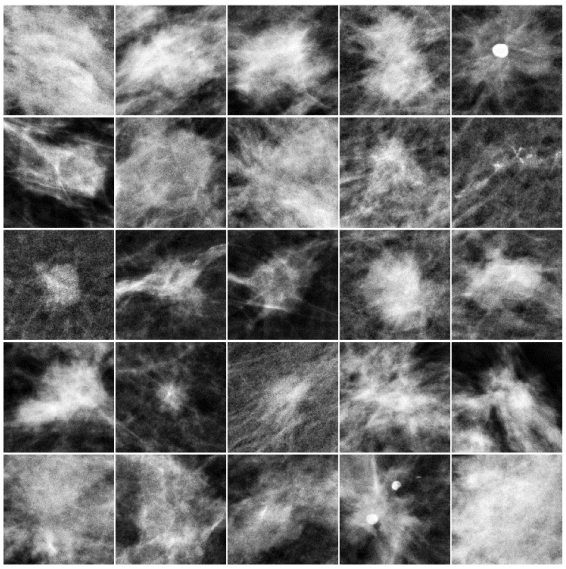

Figure 4.2 – Random images from the positive class.

The data is split with balanced class label proportion in the validation and test sets. The rest is put into the training set which is imbalanced.

Datasets	Training	Validation	Test	Total
Negative	37	25	64	126
Positive	411	25	65	501
Total	448	50	129	627

Table 4.1 - Data Splitting. The dataset splits are 72% training, 8% validation, 20% test. Choosing a large test set makes evaluation more reliable.

4.3 Class Imbalance

Class imbalance occurs when class labels are not equally distributed. This causes the loss to be biased during training and testing. Three techniques have been developed to deal with this issue: a weighted loss function, class weights and oversampling.

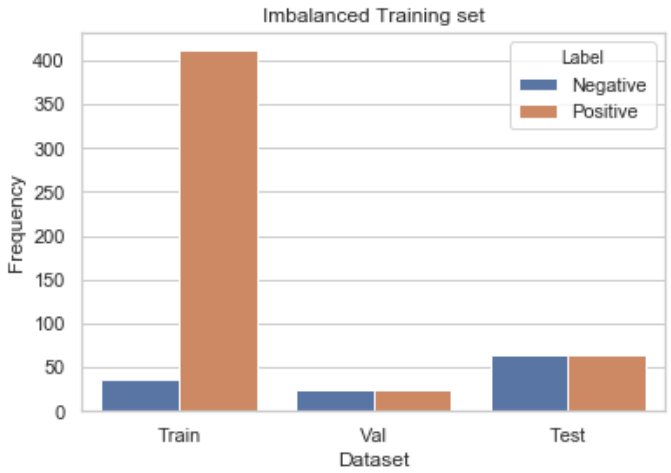

Figure 4.3 - Imbalanced Class Frequency. The validation and test sets are well-balanced with 25/25 and 64/65 observations, whereas the training set is highly imbalanced with 37 and 411, negatives and positives, respectively.

4.3.1 Custom Weighted Loss function

Since even balanced dataset have negative and positive classes that contribute equally to the loss. As the formula, illustrates that for any training case, either $y_i = 0$ or else $(1 - y_i) = 0$, so only one of the two terms contributes to the loss as the other is multiplied by zero and becomes zero.

$$\mathcal{L}_{cross-entropy}(x_i) = -\big(y_i \log\big(f(x_i)\big) + (1 - y_i) \log\big(1 - f(x_i)\big)\big), \qquad 4.1$$

where x_i and y_i are the input features and the label, and $f(x_i)$ is the output of the model, i.e. the probability that it is positive.

The cross-entropy loss equation over the whole training set $\mathcal{D}$ of size $\mathcal{N}$ can be rewritten as follows:

$$\mathcal{L}_{cross-entropy}(\mathcal{D}) = -\frac{1}{\mathcal{N}}\left(\sum_{postives} \log\left(f(x_i) + \sum_{negatives} \log\left(1 - f(x_i)\right)\right)\right) \qquad 4.2$$

Therefore, the need to introduce weights to balance the contribution of the fewer training examples:

$$\mathcal{L}_{w\ cross-entropy}(x_i) = -\big(w_p y_i \log\big(f(x_i)\big) + w_n (1 - y_i) \log\big(1 - f(x_i)\big)\big) \qquad 4.3$$

This weighted loss function is an efficient way to tackle class imbalance problems.

4.3.2 Class Weights

The objective is to give more importance to the under-represented class by assigning it a larger weight. Each class receives a certain weight based on its frequency in the whole dataset. The minority class receives a greater weight which is commonly estimated as the fraction of the majority class and the total observations. Keras has a weight parameter in its fit function.

As shown in Table 4.2, the computed class weights are:

Class	Weight
Negative (benign)	6.05
Positive (malignant)	0.55

Table 4.2 - Class Weights. The negative weight is 11 times larger than the positive weight. Which will give more importance to the loss related to the negative class by factor 6.05.

4.3.3 Oversampling the minority class

This method resamples the dataset to increase the minority class until an equally balanced training set is reached. However, it inserts duplicated images in the training set which artificially reinforce the importance of certain observations.

4.3.4 Comparaison between Oversampling and Class Weights

For both models, all parameters are fixed except the *method* for dealing with class imbalance. The initial results are listed in Table 4.3.

Models	Method	AUC
Vgg19	Oversampling	0.591
	Class weight	0.593
Resnet50v2	Oversampling	0.730
	Class weight	0.724

Table 4.3 - Comparison between Oversampling and Class Weights. Both methods yield the same AUC result. Resnet50v2 is clearly the best model to pick between both.

Although the AUC is not very different from one another, the loss and accuracy curves have different shapes. Oversampling curves are much smoother than the class weight curves.

To conclude, the difference between the oversampling and class weight methods have been tested on two different models and is clearly not significative on each of them. We can assume that these two methods have the same performance when dealing with imbalance datasets. Therefore, from now on the *class weight*s method will be used.

4.4 Data Pre-Processing

4.4.1 Import and Data Structure

Only the images without augmentation have been imported since it is better to do the data augmentation on the fly per batch. The png images have been read and stored in a “pandas data frame” with their filename, patient id and class label as illustrated in Table 4.4.

images_png	patient_id	malignancy
demd53857_CC_Right.png	53857	0
demd1565_MLO_Right.png	1565	1
demd53857_MLO_Right.png	53857	0
demd1545_CC_Right.png	1545	1
demd53599_MLO_Right.png	53599	0
…	…	…

Table 4.4 - Data Frame. This data structure is broadly used in Python from the Pandas package.

4.4.2 Raw format and data type

All the 628 raw images in the dataset are 16-bits tiff files with pixel values ranging from 0 to 65,535. Each image shape is (227, 227, 3) although the 3 channels are repeated values and so images are in grayscale. The data type is “uint16” which refers to 16-bits integer values as illustrated in Figure 4.4.

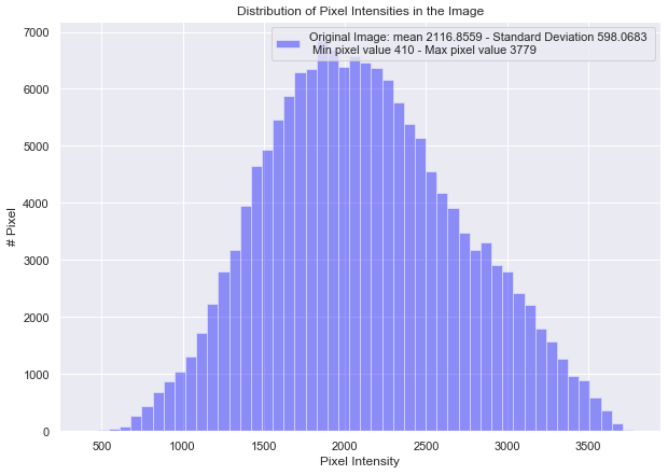

Figure 4.4 – Pixel values histogram of a raw image. This raw image has pixel values ranging from 410 to 3,779. Where each pixel has 3 channels as depth and thus has 3x16-bits values. So, those tiff images are not normalized across their full pixel intensity range.

4.4.3 Models' Input format

It is a good practice to pre-process images before feeding it to a neural network. Normalization makes it easier for the model to process the data since it avoids the possibility of exploding gradients because of the high range of pixels like [0, 255] or [0, 65,535], and thus improve the convergence speed. Centred images with zero mean and a rescaled range between [-1, 1] helps the neural network converging since a sigmoid activation function takes as input negative and positive values. Also, the magnitude of the input values has an impact on the contribution of the learning rate. Therefore, the data has first been rescaled from [0, 65,535] to [0, 255] and saved as png files as shown in Table 4.4. Then, png images were normalized as float32 between 0 and 1 (blue), and finally with zero mean (red) as depicted in Figure 4.5.

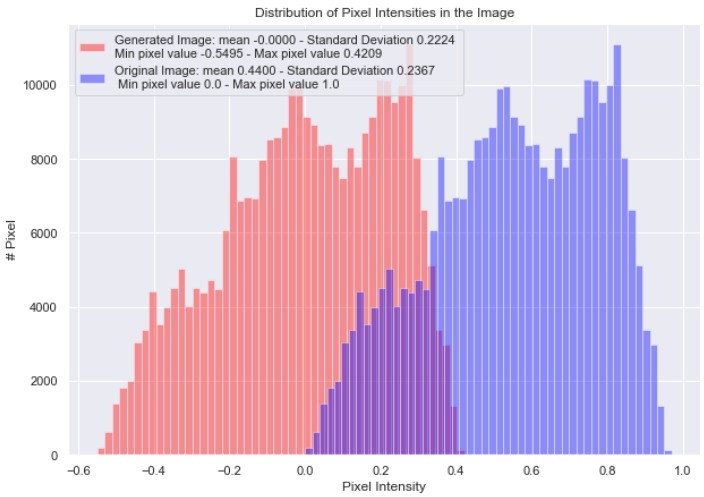

Figure 4.5 – Pixel value histogram of pre-processed images. In blue: normalized images [0, 1]. In red: centred to zero mean.

Finally, the target input shape has been changed to (batch, 128, 128, 3) to have a proper input shape for the convolutional layers.

4.4.4 Transfer Learning and Pre-trained weights

When using transfer learning, it is necessary to use the same input format and range as the one used to train the pre-trained network since weights are determined by the input values.

The same input range as the one from the pre-trained model has been used for the mammogram dataset. Since the weights have been learned from a certain range of value, it is necessary to keep this range identical. Keras has the preprocess _input function for each of its transfer learning models. It is advised to use the rescale parameter from the ImageDataGenerator function if the model is trained from scratch. However, if the model uses the pre-trained weights from Imagenet (transfer learning), the preprocess_input function should be used. This function is pre-processing the data in the same way as the Imagenet data for a certain model. It is still possible to use its own data augmentation from the ImageDataGenerator function since it is performed before the pre-processing.

Note that this method is not suited for all transfer learning models when working with grayscale images since Imagenet is a colour dataset. Some preprocess_input functions may introduce colours in the dataset as depicted on Figure 4.6, which turns out to yield poor classification results.

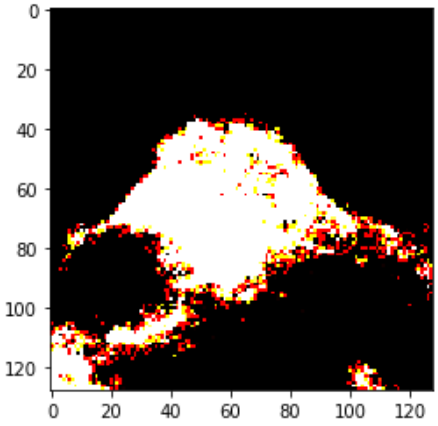

Figure 4.6 - Resnet50 pre-processing. This pre-processed image from Resnet50 inserts colour in our mammograms' patch. Therefore, this model is rejected as possible candidate for transfer learning since it introduces colours in our grayscale images which yields poor performances.

So, using Imagenet for transfer learning on a grayscale dataset is not always effective since Imagenet is an RGB colour dataset. However, some models are performing well on grayscale datasets such as Densenet121 and Resnet50v2.

To conclude, *transfer learning* is very useful when limited data is available, which makes it difficult to train models that generalizes well from scratch, especially in the medical industry. But *transfer learning* would be more efficient if weights were learned from grayscale images as proven in this paper [23].

4.5 Data Augmentation

Real-world datasets are often too large to fit into memory, and require data augmentation to increase the generalization ability and reduce overfitting.

A common misbelieve is that the augmented dataset fed per epoch is larger than the original dataset size. For example in this project, the original training set size has 448 observations (Table 4.1). Data Augmentation applies changes on the original data while keeping the same training set size. Thus, the model will see 448 training images per epoch. But those images are different for each epoch since they are augmented and thus, no more unique. So, if we have 10 epochs with 448 training images, the model will be trained on 4.480 different images instead of using just the 488 original images on the whole training. This is performed by using the off-the-shelf ImageDataGenerator object from the Keras Framework which can augment the data with translations, rotations, resizing, flipping, zooming, etc. on the fly.

This method enables also to normalize the input and divide by its standard deviation. Changing the channels format, shuffling and resizing the inputs are also built-in parameters. Thus, the ImageDataGenerator function is performing data augmentation on each batch until an epoch is reached. This approach is suitable for large datasets that require to use batch data augmentation and is also very convenient to use with the flow_from_dataframe or flow_from_directory function to access the data.

Note that we use a different ImageDataGenerator for training, validation and test sets since normalization is using per batch statistics which cannot be common between training and testing (data leakage).

Finally, it's requied to set the steps_per_epoch argument of the fit method to n_samples divided by the batch_size where n_samples is the training set size. For this experiment, the batch size for the training is set to 10. So, there are 448 / 10 = 45 batches or steps per epoch.

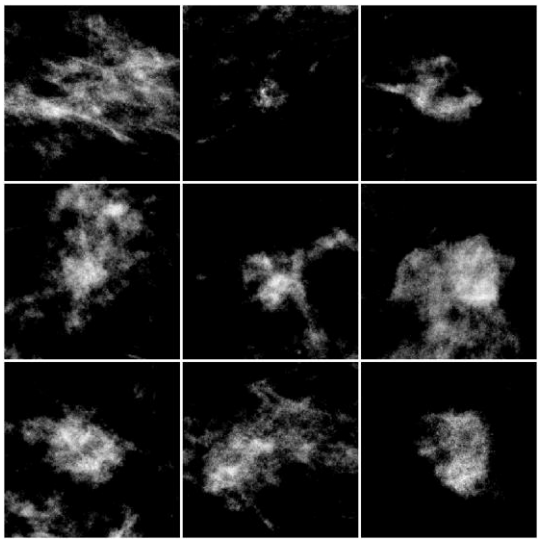

Figure 4.7 – Nine pre-processed and augmented images with Resnet50v2. The grayscale is preserved, and contrast increased.

4.6 Data Modelling

To compare models and find the best hyperparameters, it is necessary to use the same data. Changing data splits while fine-tuning hyperparameters could lead to wrong interpretations of the models' performance. For this reason, to compare the trained models, the images in the datasets have been kept identical.

Four pre-trained models have been considered for transfer learning as listed in Table 4.5.

Models	Size	Top-1 Accuracy*	Top-5 Accuracy*	Parameters
Xception	88 MB	0.790	0.945	22,910,480
ResNet50V2	98 MB	0.760	0.930	25,613,800
DenseNet121	33 MB	0.750	0.923	8,062,504
VGG19	549 MB	0.713	0.900	143,667,240

Table 4.5 - Models Characteristics using Imagenet database classes.

*The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Based on the number of parameters, memory size and top-1 and top-5 performance, 3 different models have been selected for initial training. From those 3 models, one will be selected for fine-tuning its hyperparameters.

Training Parameters	Xception	ResNet50V2	DenseNet121
Learning rate	1e-2	1e-2	1e-2
Batch Size	10	10	10
Optimizer	Adam	Adam	Adam
Loss Function	Binary Crossentropy	Binary Crossentropy	Binary Crossentropy
Epochs	20	20	20
H/V flipping	Yes	Yes	Yes
Rotation Range	0-360°	0-360°	0-360°
Rescaling	No	No	No
Pre-processing input	Yes	Yes	Yes

Table 4.6 – Base line parameters for training the pre-selected models. In this table, the default parameters are listed to compare the models' performances on the same data splits.

Based initial performances of each model, the Resnet50v2 has been selected from the top-3 models and thus, we will search for its optimal hyperparameters.

4.7 Fine-Tuning Hyperparameters

Some of the following hyperparameters have been fine-tuned during training and validation, but not yet tested as listed in Table 4.7. It is important to remind that the purpose of the validation set is to fine-tune the model's hyperparameters and to identify where the model starts overfitting to stop the training at the ideal number of epochs. Whereas the test set has not been used for building the model. It is only used for evaluation.

All those parameters have been tested independently on the same base line, and then combined in descending order as shown in Table 4.7.

Hyperparameters	*Mode/Values*	*Best Result*
Learning rate	[0.001, 0.0001, 0.00001]	0.001
Batch Size	[5, 10, 20]	10
Optimizers	SGD, Adam, RMSprop, Adagrad	Adam
Dropouts rates	[0.0, 0.10, 0.20, 0.50]	0.1
L-Regularization	L1, L2	L1
Decay	Step, Exponential,	Step
Epochs	[0:200]	40

Table 4.7 – Fine-tuning hyperparameters and best results. The learning rate start converging at 0.0001. Using a L1, regularization and a step learning rate decay increases performances.

To illustrate the fine-tuning methodology that has been applied for every parameter, two different learning rates have been depicted in Figure 4.8.

Figure 4.8 – Left: the train/val losses decrease sharply with a learning rate of 0.0001, reaching a plateau after 100 epochs with no sign of overfitting. Right: the accuracy is higher with the smaller learning rate with nearly 70% accuracy after 40 epochs, but then starts to overfit.

Based on the validation loss curves, the best learning rate is 0.0001. The other hyperparameters values are listed in the aforementioned Table 4.7.

4.8 Evaluation of Diagnostic model

In this section, the *evaluation metrics* used to predict malignancy of patients with abnormal breast cancer screenings have been presented. The results of the best performing model are included in the evaluation Table 4.8 to give a global interpretation of the results. Note that, the evaluation has been performed on the test set, which remained unseen while fine-tuning the hyperparameters.

4.8.1 ROC Curve and Thresholds

By varying the threshold, the ROC Curve changes the assumption that predictions equal or above a fixed threshold of 0.5 are classified as positives and otherwise negative if below. This ROC Curve plots the true positive rate against the false negative rate at various thresholds. The ideal spot is on the top left corner where the TPR is 1 and the FPR is 0. The AUC is the area under the ROC Curve and is referred in the medical field as the probability of a randomly picked patient with a disease to have a disease as illustrated on Figure 4.9.

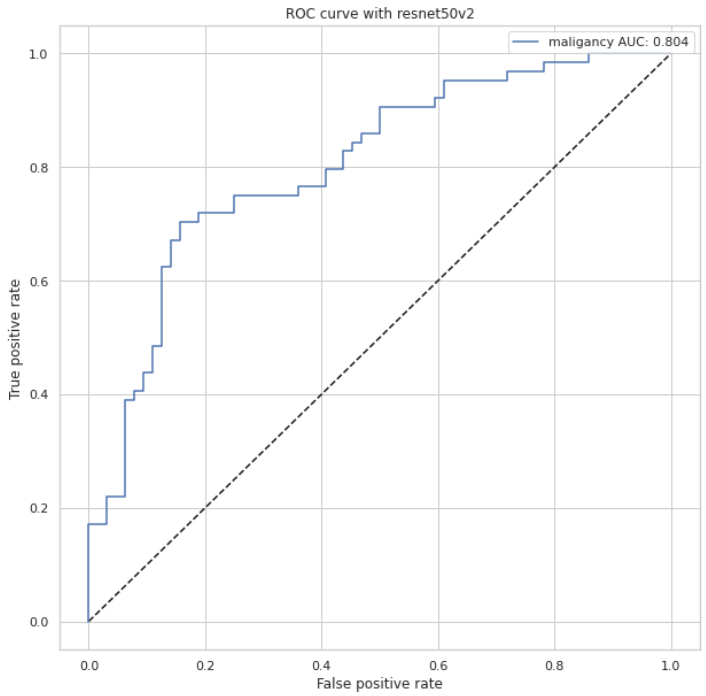

Figure 4.9 - ROC Curve and AUC. The ROC Curve depicts a centred top left curve with an AUC score reaching 0.804. Based on this plot, the performance of the model to predict malignancy of abnormal screenings can be evaluated as good through different thresholds.

Looking at the ROC curve it is not possible to select a certain threshold and know how it will impact patient's diagnosis. To achieve this, it is necessary to simulate for each threshold a confusion matrix and derive the other evaluation metrics. Then, by establishing a cost function regarding the false negative and false positive results it is possible to pick a threshold value that satisfies certain predetermined conditions.

4.8.2 Selecting a Threshold

For predicting malignancy of abnormal screenings, the evaluation metrics in Table 4.8 have been calculated based on a balanced test set of 129 mammograms with 50% of positive cases.

TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1	Threshold
65	0	64	0	0.504	0.504	1.0	0.0	0.504	Nan	0.765	0.67	0.1
65	1	63	0	0.512	0.504	1.0	0.016	0.508	1.0	0.765	0.674	0.2
64	5	59	1	0.535	0.504	0.985	0.078	0.52	0.833	0.765	0.681	0.3
63	18	46	2	0.628	0.504	0.969	0.281	0.578	0.9	0.765	0.724	0.4
56	30	34	9	0.667	0.504	0.862	0.469	0.622	0.769	0.765	0.723	0.45
53	37	27	12	0.698	0.504	0.815	0.578	0.662	0.755	0.765	0.731	0.5
28	54	10	37	0.636	0.504	0.431	0.844	0.737	0.593	0.765	0.544	0.6
7	64	0	58	0.55	0.504	0.108	1.0	1.0	0.525	0.765	0.194	0.7
1	64	0	64	0.504	0.504	0.015	1.0	1.0	0.5	0.765	0.03	0.8
0	64	0	65	0.496	0.504	0.0	1.0	Nan	0.496	0.765	0.0	0.9

Table 4.8 - Evaluation Table. A threshold at 0.45 gives a sensitivity of 86%, and a specificity of 47%. Also, a 77% confidence in not having the disease when the test is negative (NPV) which will reassure patients with abnormal screening during the waiting period that separates the screening and biopsy.

There is a trade-off between the number of False Negatives and False Positives when selecting a threshold. This trade-off can be decided by computing a weighted sum of the false positives and false negatives as the cost of misclassifying, where each term has its own weight cost associated. Setting those weights values is subjective and depends on what we want to prioritize between either reducing the number of patients sent to biopsy with no malignancy (false positives) or missing a cancer diagnosis (false negatives).

With regards to breast cancer screening, the related costs of false negatives and false positives have been discussed in section 1.2. Therefore, those two metrics are used to decide on the appropriate threshold.

The FNR is the number of people who have the disease, but are classifier as not having the disease:

$$\text{False Negative Rate} = \frac{False\ Negatives}{False\ Negatives + True\ Negatives} \qquad 4.4$$

The FPR is the number of people who do not have the disease but are classifier as having the disease, divided by the total number of people who do not have the disease:

$$\text{False Positive Rate} = \frac{False\ Positives}{False\ Positives + True\ Negatives} \qquad 4.5$$

A threshold at 0.40 would mean that 10% of the people with abnormal screenings that have a malignant tumour would be classified as not having a disease (false negative rate). And that 70% of people with an abnormal screening that does not have a malignant tumour would still be classified as having the disease (false positive rate).

Whereas a threshold at 0.45 would yield a larger false negative rate of 23% but a better false positive rate of 53%.

Thus, setting a threshold at 0.45 to avoid having 70% of people with abnormal screenings without disease classified as malignant. We want to avoid creating fake financial burden for patients and unnecessary emotional anxiety while waiting for their results. However, this threshold will increase the number of patients with abnormal screening classified with benign tumours although their biopsy will be positive. This could lead to delays for families in preparing themselves financially and emotionally.

It is important to remind that this classifier's objective is to output a likelihood of having a malignant tumour based on patients with an abnormal screening, and that false negatives are only temporary until the results of the biopsy are disclosed. Thus, this classifier's purpose is more to relieve patients from stress than to create unnecessary. So, the priority is on a low false positive rate and a high confidence in truly not having a disease.

4.8.3 Confusion matrix

After selecting our threshold, the confusion matrix indicates how images where classified compared with their actual class label.

	Predicted Positive Class	*Predicted Negative Class*
Actual Positive Class	56 (True Positives)	9 (False Negatives)
Actual Negative Class	34 (False Positives)	30 (True Negatives)

Figure 4.10 - Confusion Matrix. This confusion matrix gives 56 true positives, 30 true negatives, 34 false positives and 9 false negatives which are used to compute the other evaluation metrics.

In breast cancer diagnosis, reducing the number of false negatives is a major challenge since breast cancer screenings are usually performed once a year. However, reducing false positives would eliminate many unnecessary surgeries, biopsies and radiations which are harmful, stressful, and expensive for patients.

4.8.4 Positive Predicted Value (PPV) and Negative Predicted Value (NPV)

In this context of Diagnostic, sensitivity and specificity are less relevant since they presume that the patient already has or not the disease. Instead, the PPV and NPV give an indication on whether a patient has the disease and thus, diagnostically speaking it makes more sense.

4.8.4.1 Positive Predicted Value (PPV):

- Probability of truly having a disease given that the test output is positive.

$$PPV = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad 4.6$$

PPV	0.662

Table 4.9 – PPV. 66% of patients with a positive screening test are true positives.

4.8.4.2 Negative Predicted Value (NPV):

- Probability of truly not having a disease given that the test output is negative.

$$NPV = \frac{True\ Negaives}{True\ Negatives + False\ Negatives} \qquad 4.7$$

NPV	0.76

Table 4.10 - NPV. 76% of patients with a negative screening test are true negatives.

So, the Negative Predicted Value has a higher diagnostic performance than the PPV which means that it is more reliable at classifying patients as not having a condition than at diagnosing someone with it. This exactly what we want when reassuring patients while waiting for biopsy.

4.8.5 Sensitivity and Specificity

Sensitivity and Specificity are major evaluation metrics for assessing machine learning models. Especially, *sensitivity* since it considers the number of false negatives which are the undetected positives to minimize. Whereas *specificity* checks the number of patients without malignant cancer sent to biopsy (false positives).

Sensitivity	$\frac{TP}{TP + FN}$	*0.862*
Specificity	$\frac{TN}{TN + FP}$	0.469

Table 4.11 - Sensitivity and specificity. 85% of positives where correctly classified as positives and 0.47% of negatives where correctly classified as negatives. So, this threshold gives a low specificity and a high sensitivity.

4.8.6 Precision-Recall Curve

This metric is useful when classes are very imbalanced. Precision is like the Positive Predicted Value which gives the probability of finding true positives in the group of patients classified as positive. Whereas *recall* which is also referred as *sensitivity* is the probability of detecting the positive cases.

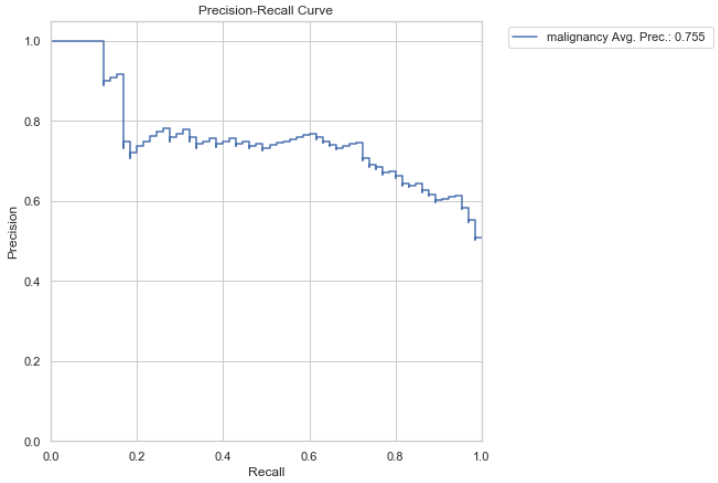

Figure 4.11 - Precision-Recall Curve. The curve depicts the evolution of the score with a moving threshold and an average score of 0.755.

4.8.7 F1 score

The F1 score is the mean of the precision and recall, where the score reaches its best value at 1 and worst at 0. The formula is as follows:

$$F1\ score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \qquad 4.8$$

This f1 score is 0.723 and is also presented in Table 4.8.

4.8.8 Calibration curve

By plotting the Calibration Curve, one can observe how the model's generated probabilities are aligned with the real probabilities (fraction of positives). A well-calibrated model has a curve that is almost aligned with the x=y line. The blue dashed-line scenario occurs when all the negatives have a prediction of 0 (fraction=0), and all the positives a predicted value of 1 (fraction=1).

Thus, this curve is built by assigning all the predicted values into a fixed number of bins on the x-axis and taking the mean value per bin, i.e. 14 bins for 129 predictions, and computing the fraction of positive labels in each bin.

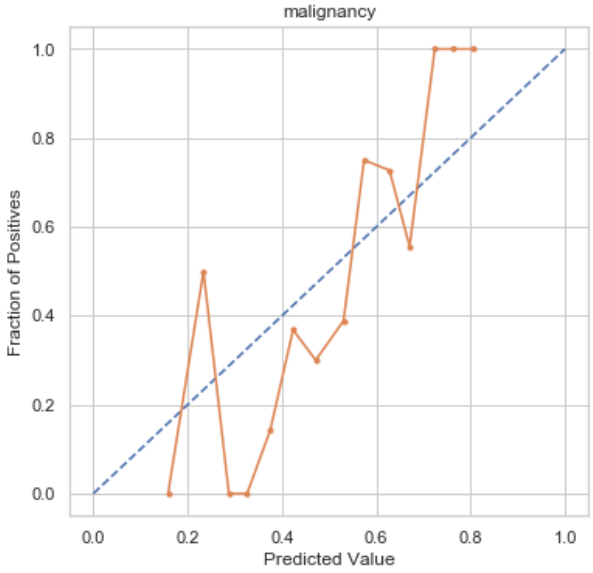

Figure 4.12 - Calibration Curve. The dashed line represents a perfectly calibrated classifier since the fraction of positives in each bin is equal to the mean predicted value in each bin. The solid orange line is the calibration curve for our trained binary classifier Resnet50v2, and is diagonally oriented as it should be. We can observe the predicted probabilities ranging from 0.158 to 0.806 which indicates a good confidence on the predictions but not stretched to its maximal. The starting and end points are showing good results with no positives in the first bin and 100% of positives in the last 3 bins. However, the S-shape indicates an under-confident classifier since it under-predicts the positive class for negative labels and over-predicts the positive class for high class probabilities.

Thus, the classifier will over-estimate the number of positives (malignant tumours) and underestimating the number of negatives (benign tumours).

4.9 Visualizing Learning with GradCAM to increase interpretability.

The Class Activation Map shows what the model is looking at when classifying an image. Depicting the activated pixels as a heatmap of the malignant region. This is performed by computing the gradients of an image at the final batch normalization layer and taking the mean value as the weights and multiplying those weights with the predicted output per pixel. Then, resizing the class activation map to match the original image and stacking both images together.

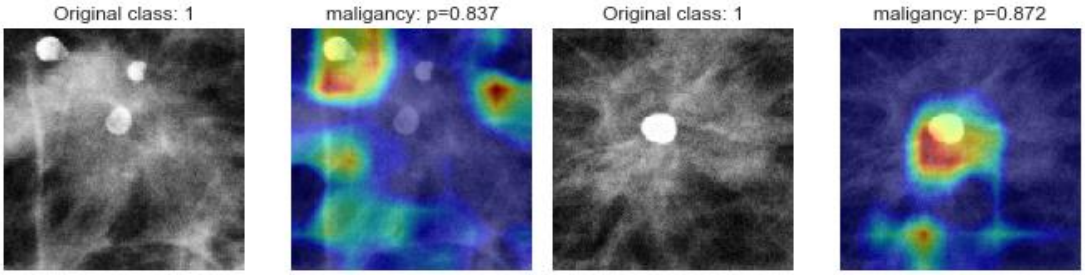

Figure 4.13 - GradCAM. Those 2 coloured images are a representation of what the model has learned to detect on images. The highlighted areas are responsible for the classification outcome.

5 CONCLUSION

This report presented 4 chapters, the first one introduced the fundamentals of *breast cancer, medical imaging, deep learning,* and the optimam *dataset* used for this project. Secondly, an in-depth literature review is summarized covering *neural networks*, *convolutional neural networks, transfer learning* and *evaluation metrics*. As third chapter, the *data access, methodology, and tools* have been described in detail. The fourth chapter presented a detailed report of the empirical testing and logical decision path that led to the final neural network classifier.

5.1 Results

The objective was to provide an accurate probability likelihood on the available mammogram patches using a deep learning approach to classify abnormal screenings that have been sent to biopsy as either benign or malignant. With this regard, the objective has been reached and explored in details, which has led to obtain a classifier able to predict malignancy from abnormal breast cancer screening with an AUC recorded at 81%, a sensitivity of 86%, specificity of 47%, and a NPV of 77% with a threshold selected at 0.45.

By extracting the predicted value from a mammogram send into the neural network classifier, it is now possible to share to the patient a likelihood of malignancy before going to biopsy and while waiting for the results. This will relieve women with a low likelihood score, reducing anxiety and avoid unnecessary expensive further examinations.

5.2 Looking Ahead for Progress, Limits and Responsibilities

Future work would focus on automated patch extraction from mammograms and detecting abnormal features to help professional practitioners localize potential malignant breast tumours.

Transfer learning showed its limits at reaching better evaluation metrics despite the efforts to deal with class imbalance, augmenting the dataset, and fine-tuning hyperparameters. Imagenet has also demonstrated its limits for medical imaging datasets. A possible way of improvement worth exploring is using transfer learning with models trained on a large grayscale dataset with biomedical images.

It is now our responsibility to increase the adoption of this technology to assist clinics, medical practitioners, hospitals, pharmacology companies and other potential users to increase early breast cancer detection, and also reducing false negatives and false positives.

Regarding the covid-19 crises, deep learning has powerful predictive capabilities able to cut costs, accelerate the dug development process, and monitor chest CT damages.

REFERENCES

[1] Z. Z. C. J. Bar RG, "Probably Benign Lesions at Screening Breast US," *RSNA,* pp. 701-712, 2013.

[2] M. T. S. J. G. Scott Mayer McKinney, "International evaluation of an AI system for breast cancer screening," *Nature,* January 2020.

[3] OPTIMAM Mammography Image DataBase, OMI-DB, 2020. [Online]. Available: https://medphys.royalsurrey.nhs.uk/omidb/.

[4] "Breast Cancer Now," 2019. [Online]. Available: https://breastcancernow.org/about-us/media/facts-statistics.

[5] H. H. Publishing, "Mammography," Harvard Medical School, 2019. [Online]. Available: https://www.health.harvard.edu/medical-tests-and-procedures/mammography-a-to-z.

[6] Canadian Cancer Society, "Breast Cancer Screening," 2019. [Online]. Available: https://www.cancer.ca/en/cancer-information/cancer-type/breast/screening/?region=on.

[7] G. C. S. Z. W. D. K. Burak Erem, "Interactive Deformable Registration Visualization and Analysis of 4D Computed Tomography," in *International Conference on Medical Biometrics*, 2007.

[8] M. A. J. Vishwa S. Parekh, "Integrated radiomic framework for breast cancer and tumor biology using advanced machine learning and multiparametric MRI," Nature research journal, 2017.

[9] L. H. H.-P. C. A. Z. W. Kenny H. Cha, "Bladder Cancer Treatment Response Assessment in CT using Radiomics with Deep-Learning," *Nature,* August 2017.

[10] P. Diagnostics, Breast-Cancer-Detection, 2016.

[11] M. M. K. Ong, National Expenditure for False-Positive Mammograms and Breast Cancer Overdiagnoses Estimated at $4 Billion a Year, Health Affairs, 2015.

[12] R. K. D. X. A. B. Fei-Fei Li, "CS231n: Convolutional Neural Networks for Visual Recognition," Stanford University, 2020. [Online]. Available: http://cs231n.stanford.edu/.

[13] S.-C. Wang, "Artificial Neural Network," in *Interdisciplinary Computing in Java Programming*, Boston, MA, The Springer International Series in Engineering and Computer Science, 2003, pp. 81-100.

[14] D. J. &. J. H. Martin, "Chapter 5: Logistic Regression," in *Speech and Language Processing*, 2019.

[15] X. Z. S. R. J. S. Kaiming He, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” Computer Vision and Pattern Recognition, 2015.

[16] C. S. Sergey Ioffe, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” arXiv:1502.03167, 2015.

[17] D. J. S. A. S. M. Jonathan Frankle, “Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs,” in *arXiv.org*, 2020.

[18] G. H. A. K. I. S. ,. R. S. Nitish Srivastava, “Dropout: A Simple Way to Prevent Neural Networks from,” Journal of Machine Learning Research, 2014.

[19] G. Hinton, “Coursera Lecture 6e rmsprop”.

[20] J. B. Diederik P. Kingma, “Adam: A Method for Stochastic Optimization,” 2015.

[21] Y. B. James Bergstra, “Random Search for Hyper-Parameter Optimization,” *Journal of Machine Learning Research 13,* 2012.

[22] S. A. E. V. Fábio Perez, “Solo or Ensemble? Choosing a CNN Architecture for Melanoma Classification,” in *CVPR 2019*, 2019.

[23] L. Yin, “A Summary of Neural Network Layers,” Medium, 2018.

[24] F. Chollet, “Deep Learning,” Keras, 2015.

[25] A. Z. Karen Simonyan, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *CVPR*, 2014.

[26] D. R. Yiting Xie, “Pre-training on Grayscale ImageNet Improves Medical Image Classification,” in *ECCV 2018*, 2018.

[27] B. J. M. James A. Hanley, “The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve,” *Radiology,* pp. 143, 29–36., 1982.

[28] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms.,” in *Pattern recognition*, 1997, pp. 1145-1159.

APPENDIX 1 – PYTHON PROGRAMMING

THERE ARE 3 PARTS:

FIRST PART: IMPORT, DATA PRE_PROCESSING AND DATA EXPLORATION.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 23 16:52:15 2020
@author: boels
"""

pandas as pd
import numpy as np
import seaborn as sns
import tifffile as tiff
import matplotlib.pyplot as plt
import cv2
import os
from PIL import Image

# python file with all the created functions
import util


## IMPORTING DATA

# read file names
train_list_files_benign, train_list_files_cancerous, test_list_files_benign, test_list_files_cancerous = util.read_filenames()

# extract the images without augmentation
train_b = util.not_augmented_files(train_list_files_benign)
train_c = util.not_augmented_files(train_list_files_cancerous)
test_b = util.not_augmented_files(test_list_files_benign)
test_c = util.not_augmented_files(test_list_files_cancerous)

# list files
filelists = [train_b, train_c, test_b, test_c]

# create dataframes
df_trb, df_trc, df_tb, df_tc = util.create_dataframes(filelists, train_b, train_c, test_b, test_c)

# print the number of unique patients ids in each set.
print(f'There are {len(df_trb)} unique Patient IDs in the training negative set')
print(f'There are {len(df_trc)} unique Patient IDs in the training positive set')
print(f'There are {len(df_tb)} unique Patient IDs in the test negative set')
print(f'There are {len(df_tc)} unique Patient IDs in the test positive set')


## DATA SPLITTING

# create training and test sets
df_train, df_test = util.create_datasets(df_trb, df_trc, df_tb, df_tc)

# prevent data leakage
df_train = util.drop_overlapping_patients_train(df_train, df_test)

# path to folder
train_path = 'Data/raw train/'
test_path = 'Data/raw test/'

# add tiff files arrays to the data frame
df_train = util.df_add_arrays(train_path, df_train)
df_test = util.df_add_arrays(test_path, df_test)


## DATA EXPLORATION

# plot class labels frequency
util.plot_label_frequency(df_train, df_test)
```

```
# choose a folder
local_folder = 'Data/raw train/'

# visualize a random selection of images from the train dataset.
util.show_tif(local_folder, df_train)
# plot a histogram of the distribution of the pixels
util.plot_pixel_values(df_train)


## DATA PRE-PROCESSING AND DATA AUGMENTATION

# print data type of images
print(f'The data type of the pixels for each image is: {df_train.array[0].dtype}')

# normalize images and save as png files in the folder and dataframe
util.add_png_arrays(df_train)
util.add_png_arrays(df_test)

# add png filenames to dataframe
util.add_png_filenames(df_train)
util.add_png_filenames(df_train)


## DATA VISUALIZATION

# extract numpy values from column in data frame
images = df_train['images_png'].values

# extract 9 random images from the column
random_images = [np.random.choice(images) for i in range(9)]

# image directory
img_dir = 'Data/all/'
print('Display Random Images')

# adjust the size the image
plt.figure(figsize=(20,20))

# iterate and plot random images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(img_dir, random_images[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

# adjust subplot parameters to give specified padding
plt.tight_layout()


## INVESTIGATE A SIGNLE IMAGE

# get the first image that was listed in the train_df dataframe
sample_img = df_train.images[4]
raw_image = plt.imread(os.path.join(img_dir, sample_img))
plt.imshow(raw_image, cmap='gray')
plt.colorbar()
plt.title('Raw breast X Ray Image')
print(f"The dimensions of the image are {raw_image.shape[0]} pixels width and {raw_image.shape[1]} pixels height, with
{raw_image.shape[2]} color channels")
print(f"The maximum pixel value is {raw_image.max():.4f} and the minimum is {raw_image.min():.4f}")
print(f"The mean value of the pixels is {raw_image.mean():.4f} and the standard deviation is {raw_image.std():.4f}")


## INVESTIGATE PIXEL VALUE DISTRIBUTION.

# plot a histogram of the distribution of the pixels
raw_image = df_train.array[4]
sns.distplot(raw_image.ravel(),
             label=f'Pixel Mean {np.mean(raw_image):.4f} & Standard Deviation {np.std(raw_image):.4f}', kde=False)
plt.legend(loc='upper center')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

```
## IMAGE PREPROCESSING IN KERAS

# set parameters
IMAGE_DIR = "Data/out/train/"
target_w = 320
target_h = 320
BATCH_SIZE_TRAINING = 10
BATCH_SIZE_VALIDATION = 5
BATCH_SIZE_TESTING = 1

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet_v2 import preprocess_input

# generate augmented data per batch
data_generator = ImageDataGenerator(horizontal_flip = True,
                                    vertical_flip = True,
                                    preprocessing_function=preprocess_input)

# look at the data specified in the dataframe
train_generator = data_generator.flow_from_dataframe(
        dataframe=df_train,
        directory=IMAGE_DIR,
        x_col='images_png',
        y_col=['malignancy'],
        class_mode="raw",
        batch_size=BATCH_SIZE_TRAINING,
        shuffle=True,
        seed=1,
        target_size=(target_w,target_h))

# extract a batch
train_generator_keras, label = train_generator.__getitem__(4)


## IMAGE PREPROCESSING SELF DEFINED

generator_center_norm = util.get_train_generator(df_train, 'Data/out/train/', 'images_png', \
                                                 'malignancy', shuffle=True, batch_size=8, seed=1, \
                                                 target_w = 320, target_h = 320,
                                                 samplewise_center= True, samplewise_std_normalization= False,
                                                 rescale= 1/255.0
                                                 )
generator_norm = util.get_train_generator(df_train, 'Data/out/train/', 'images_png', \
                                                 'malignancy', shuffle=True, batch_size=8, seed=1, \
                                                 target_w = 320, target_h = 320,
                                                 samplewise_center= False, samplewise_std_normalization= False,
                                                 rescale= 1/255.0
                                                 )
# show a processed image
sns.set_style("white")
raw_image, label = generator_normaliz.__getitem__(4)
plt.imshow(raw_image[0], cmap='gray')
plt.title('Raw Breast X Ray Image')
print(f"The dimensions of the image are {generated_image_norm.shape[1]} pixels width and {generated_image_norm.shape[2]} pixels
height")
print(f"The maximum pixel value is {generated_image_norm.max():.4f} and the minimum is {generated_image_norm.min():.4f}")
print(f"The mean value of the pixels is {generated_image_norm.mean():.4f} and the standard deviation is {generat-
ed_image_norm.std():.4f}")


## HISTOGRAM TO COMPARE RAW AND PREPROCESSED IMAGES

# get same batches
generator_cent_norm, label = generator_center_norm.__getitem__(4)
generator_norm, label = generator_normaliz.__getitem__(4)

# include a histogram of the distribution of the pixels
sns.set()
plt.figure(figsize=(10, 7))

# plot histogram for generated image
```

```
sns.distplot(generator_cent_norm[0].ravel(),
        label=f'Generated Image: mean {np.mean(generator_cent_norm[0]):.4f}-Standard Deviation
{np.std(generator_cent_norm[0]):.4f} \n'
        f'Min pixel value {np.min(generator_cent_norm[0]):.4} - Max pixel value {np.max(generator_cent_norm[0]):.4}',
        color='red',
        kde=False)
sns.distplot(train_generator_keras[2].ravel(),
        label=f'Original Image: mean {np.mean(generator_norm):.4f} - Standard Deviation {np.std(generator_norm):.4f} \n '
        f'Min pixel value {np.min(generator_norm):.4} - Max pixel value {np.max(generator_norm):.4}',
        color='blue',
        kde=False)

plt.legend(loc='best')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixel')



## SECOND PART: CLASS IMBALCE, DATA AUGMENTATION, MODELLING, TRAINING, FINE-TUNING, PREDICT AND
GRADCAM.


## CLASS IMBALANCE

# weighted loss function

# compute the class frequency
freq_pos_train, freq_neg_train = util.compute_label_freq(df_train)
freq_pos_val, freq_neg_val = util.compute_label_freq(df_val)
freq_pos_test, freq_neg_test = util.compute_label_freq(df_test)

# visualization class frequency
sns.set(style="whitegrid")
data = pd.DataFrame({"Dataset": ['Train', 'Train', 'Val', 'Val', 'Test', 'Test'],
                "Label": ['Negative','Positive','Negative','Positive','Negative','Positive'],
                "Frequency": [freq_neg_train, freq_pos_train,
                        freq_neg_val, freq_pos_val,
                        freq_neg_test, freq_pos_test]})
f = sns.barplot(x="Dataset", y="Frequency", hue="Label" ,data=data)
plt.title('Imbalanced Training set')

# calculate the positive weight as the fraction of negative labels
pos_weights = freq_neg_train
# calculate the negative weight as the fraction of positive labels
neg_weights = freq_pos_train

# oversampling
#df_train = util.oversampling(df_train)

# weighted loss function
# see util.py

# class weights
class_weight = util.get_class_weights(df_train)


## DATA AUGMENTATION

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet_v2 import preprocess_input

# set parameters
IMAGE_DIR = "Data/out/train/"
target_w = 320
target_h = 320
BATCH_SIZE_TRAINING = 10
BATCH_SIZE_VALIDATION = 5
BATCH_SIZE_TESTING = 1

# use Resnet50v2 preprocessing input
data_generator = ImageDataGenerator(horizontal_flip = True,
                        vertical_flip = True,
                        preprocessing_function=preprocess_input)
```

```
train_generator = data_generator.flow_from_dataframe(
        dataframe=df_train,
        directory=IMAGE_DIR,
        x_col='images_png',
        y_col=['malignancy'],
        class_mode="raw",
        batch_size=BATCH_SIZE_TRAINING,
        shuffle=True,
        seed=1,
        target_size=(target_w,target_h))

valid_generator = data_generator.flow_from_dataframe(
      dataframe=df_val,
      directory= 'Data/out/validation/',
      x_col='images_png',
      y_col=['malignancy'],
      class_mode="raw",
      batch_size=BATCH_SIZE_VALIDATION,
      shuffle=False,
      seed=1,
      target_size=(target_w,target_h))

test_generator = data_generator.flow_from_dataframe(
      dataframe=df_test,
      directory='Data/out/test/',
      x_col='images_png',
      y_col=['malignancy'],
      class_mode="raw",
      batch_size=BATCH_SIZE_TESTING,
      shuffle=False,
      seed=1,
      target_size=(target_w,target_h))

# plot train image in batch
plt.figure(figsize=(40,40))
for i in range(9):
   plt.subplot(3, 3, i + 1)
# Adjust subplot parameters to give specified padding
plt.tight_layout()


## MODELLING: RESNET50V2

from keras.applications.resnet_v2 import ResNet50V2

# set hyperparameters
L_REG = 'l1'
LR = 0.0001
OPTIMIZER = keras.optimizers.Adam(learning_rate=LR)
DROPOUTS = 0.1

# path to weights without top layer
resnet_weights_path = 'C:\\Users\\boels\\.keras\\models\\resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5'

# create the base pre-trained model
base_model = ResNet50V2(weights=resnet_weights_path,
                  include_top=False)

# don't train first layer model as it is already trained
for layer in base_model.layers:
   layer.trainable = False

# x is the output
x = base_model.output

# add a global spatial average pooling layer and dropouts
x = GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(DROPOUTS)(x)  # Regularize with dropout

# add a logistic layer
predictions = Dense(1, activation="sigmoid",
                kernel_regularizer=L_REG)(x)
model = Model(inputs=base_model.input, outputs=predictions)

# use custom weighted loss function or give class weigths in fit_generator
```

```
model.compile(optimizer= OPTIMIZER,
            loss= 'binary_crossentropy',
            metrics=['accuracy'])
## TRAINING

EPOCHS = 40
PATIENCE = 40
STEPS_PER_EPOCH_TRAINING = len(train_generator)
STEPS_PER_EPOCH_VALIDATION = len(valid_generator)

early_stopping = EarlyStopping(monitor='val_loss',
                          patience=PATIENCE)

# fit the parameters to the model
history = model.fit(train_generator,
                  validation_data= valid_generator,
                  steps_per_epoch=STEPS_PER_EPOCH_TRAINING, # batches
                  validation_steps=STEPS_PER_EPOCH_VALIDATION,# batches
                  epochs = EPOCHS,
                  callbacks= [early_stopping],
                  class_weight= class_weight)

## FINE-TUNING HYPERPARAMETERS

# Loss plot
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training loss', 'Validation loss'], loc='upper right')
plt.ylabel("loss")
plt.xlabel("epoch")
plt.title("Training and Validation Loss Curves with "+ base_model.name)
plt.show()

# Accuracy plot
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Train accuracy', 'Val accuracy'], loc='center right')
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.title("Training Accuracy and Validation Accuracy with "+ base_model.name)
plt.show()

# load best weights trained on google colab
model.load_weights('Data/weights/my_model_weights_Epochs40.h5')


## PREDICTION AND EVALUATION ON TEST SET

# predict
predicted_vals = model.predict_generator(test_generator, steps = len(test_generator))
print(f' std: {predicted_vals.std()}\n min: {predicted_vals.min()}\n max: {predicted_vals.max()}')

# roc and auc
labels = ['maligancy']
auc_rocs = util.get_roc_curve(labels, predicted_vals, test_generator, base_model)

# export results for evaluation
df_test_eval = df_test[df_test.columns[[4, 1, 3]]]
# add predictions to df_test and save as .csv file
df_test_eval['predictions'] = predicted_vals
df_test_eval.to_csv("Output/df_test.csv", index=None)


## GRADCAM: VISUALIZATION OF LEARNING

# select batch normalization layer
layer_name = 'post_bn'

# select dataframe and path
df = df_test
IMAGE_DIR = "Data/out/test/"

# only show the labels with top 4 AUC
labels_to_show = np.take(labels, np.argsort(auc_rocs)[::-1])[:4]
```

```
# candidate images
images = ['demd117_CC_Right.png', 'demd122_MLO_Right.png',
          'demd1501_CC_Right.png', 'demd50449_MLO_Right.png']
# loop over candidate images
for image in images:
util.compute_gradcam(model, image, IMAGE_DIR, df, labels,
                                labels_to_show,
                                layer_name= layer_name
                                )


## EVALUATION AND INTERPRETATION

# choose thresholds and number of thresholds for evaluation table
N_THRESHOLDS = 10
THRESHOLDS = [0.10, 0.20, 0.30, 0.40, 0.45, 0.50, 0.60, 0.70, 0.80, 0.90]

# create list of string
class_labels = ['malignancy'] * N_THRESHOLDS
pred_labels = ['predictions'] * N_THRESHOLDS

# import test results
test_results = pd.read_csv("Output/df_test.csv")

# ground truth
y = test_results[class_labels].values
# predicted labels
pred = test_results[pred_labels].values
test_results[np.concatenate([class_labels, pred_labels])]

# get all evaluation metrics in one tabke
df = util.get_performance_metrics(y, pred, class_labels,
                                        acc=util.get_accuracy,
                                        prevalence=util.get_prevalence,
                                        sens=util.get_sensitivity,
                                        spec=util.get_specificity,
                                        ppv=util.get_ppv,
                                        npv=util.get_npv,
                                        auc=util.roc_auc_score,
                                        f1=f1_score,
                                        thresholds= THRESHOLDS)
# get roc curve
util.get_curve(y, pred, class_labels)

# get precision-recall curve
util.get_curve(y, pred, class_labels, curve='prc')

# get calibration curve
util.plot_calibration_curve(y, pred, class_labels)

# save evaluation table to csv
df.to_csv("Output/df_Evaluation_Metrics_10.csv", index=None)



## LAST PART: MY_FUNCTIONS

# this part gathers all the created functions called as ‘util.get_function’ in part one and two.


def read_filenames():

    local_path = ''
    folder_training_benign = 'Data/training/benign/'
    foler_training_cancerous = 'Data/training/cancerous/'

    files_benign = local_path+folder_training_benign
    files_cancerous = local_path+foler_training_cancerous

    train_list_files_benign = []
    train_list_files_cancerous  = []

    # create list benign train
    for file in os.listdir(files_benign):
        if file.endswith('.tif'):
```

```
            train_list_files_benign.append(file)

    # create list cancerous train
    for file in os.listdir(files_cancerous):
        if file.endswith('.tif'):
            train_list_files_cancerous.append(file)


    # testing images
    local_path = ''
    folder_testing_benign = 'Data/testing/benign/'
    foler_testing_cancerous = 'Data/testing/cancerous/'

    files_benign = local_path+folder_testing_benign
    files_cancerous = local_path+foler_testing_cancerous

    test_list_files_benign = []
    test_list_files_cancerous  = []

    for file in os.listdir(files_benign):
        if file.endswith('.tif'):
            test_list_files_benign.append(file)

    for file in os.listdir(files_cancerous):
        if file.endswith('.tif'):
            test_list_files_cancerous.append(file)


    return train_list_files_benign, train_list_files_cancerous, test_list_files_benign, test_list_files_cancerous


def not_augmented_files(filenames):
    """
    Returns the selected images that ends with "Left.tif" or "Right.tif"
    Args:
        filenames (list): list of filenames.
    Returns: the raw filenames for each patient_id.
    """
    unique_files = [ids for ids in filenames if ids.endswith('Left.tif')\
                                        or ids.endswith('Right.tif')]
    return unique_files


def create_dataframes(filelists, train_b, train_c, test_b, test_c):
    """
    Returns dataframes with all the images names and their labels.
    """
    df1 = pd.DataFrame({'images': train_b,
                'patient_id': np.nan,
                'array': np.nan,
                'malignancy': [0] * len(train_b)})

    df2 = pd.DataFrame({'images': train_c,
                    'patient_id': np.nan,
                    'array': np.nan,
                    'malignancy': [1] * len(train_c)})

    df3 = pd.DataFrame({'images': test_b,
                    'patient_id': np.nan,
                    'array': np.nan,
                    'malignancy': [0] * len(test_b)})

    df4 = pd.DataFrame({'images': test_c,
                    'patient_id': np.nan,
                    'array': np.nan,
                    'malignancy': [1] * len(test_c)})
    df = [df1, df2, df3, df4]

    for count, file in enumerate(filelists):
        temp = [sub.split('d')[2] for sub in file]
        df[count]['patient_id'] = [sub.split('_')[0] for sub in temp]

    return df1, df2, df3, df4
```

```
def create_datasets(df1, df2, df3, df4):
    """
    Returns: test set with at least 50% of positives cases.
    """
    # merge df train labels
    df_train = pd.concat([df1, df2], ignore_index=True)

    # merge df test labels
    df_test = pd.concat([df3, df4], ignore_index=True)

    # merge all data and shuffle to balance the labels
    df = pd.concat([df_train, df_test], ignore_index=True)

    duplicates = df[df.duplicated('images')]
    df = df.drop_duplicates('images')
    print(f'We found and dropped {len(duplicates)} duplicated values in the data set which are:')
    print(f'{duplicates}')

    # shuffle all the data
    df_shuffled = df.sample(frac=1)

    # create test set
    df_test_p = df_shuffled.loc[df_shuffled['malignancy'] == 1].sort_values(['patient_id'])[:65]
    df_test_n = df_shuffled.loc[df_shuffled['malignancy'] == 0].sort_values(['patient_id'])[:64]
    df_test = pd.concat([df_test_p, df_test_n])

    # create training set
    df_train_p = df_shuffled.loc[df_shuffled['malignancy'] == 1].sort_values(['patient_id'])[65:]
    df_train_n = df_shuffled.loc[df_shuffled['malignancy'] == 0].sort_values(['patient_id'])[64:]
    df_train = pd.concat([df_train_p, df_train_n])
    # same 'patient_id' for MLO and CC at split [65:66] and [64:65].

    df_train.reset_index(drop=True, inplace=True)
    df_test.reset_index(drop=True, inplace=True)

    print('')
    print('Training and Testing DataFrame have been created.')
    print('')

    return df_train, df_test


def drop_overlapping_patients_train(df_train, df_test):

    patient_overlap = check_leakage(df_train, df_test, 'patient_id')

    # Find and Drop overlapping indexes
    overlap_index = []

    for idx in range(len(patient_overlap)):
        overlap_index.extend(df_train.index[df_train['patient_id'] == patient_overlap[idx]].tolist())

    # Drop the overlapping rows from the training set
    df_train.drop(overlap_index, inplace=True)

    # Reset indexes
    df_train.reset_index(drop=True, inplace=True)

    print(f'We dropped overlapping patients ids from the training set patient:{patient_overlap}')

    return df_train


def df_add_arrays(path, df):
    """
    Adds the .tiff files arrays.
    -------
    df : data frame
    """
    column = 'images'
    files = df[column].values.tolist()
    array_list = []

    for file in files:
        im = tiff.imread(path+file)
```

```
        array_list.append(im)

    df['array'] = array_list

    return df


def add_png_arrays(df):
    """
    Adds the normalized arrays.
    """
    files = df['images']
    array_list = []

    for file in files:
        # locate each array with the corresponding file name.
        mammogram = df.loc[df['images'] == file, 'array'].iloc[0]
        mammogram_uint8_by_cv2 = cv2.normalize(mammogram, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        array_list.append(mammogram_uint8_by_cv2)

    # create new columns in df and add the lists.
    df['norm_mammo'] = np.nan
    df['norm_mammo'] = array_list

    return


def add_png_filenames(df):

    """
    Adds new colmun 'norm_mammo' of the dataframe (df).
    """
    files = df['images']
    file_names_list = []

    for file in files:
        file = file[:-3]+'png'
        file_names_list.append(file)

    # create new columns in df and add the lists.
    df['images_png'] = ""
    df['images_png'] = file_names_list

    return


def show_normalized_mammo(df):

    idx = np.random.randint(len(df))
    mammogram_uint8_by_cv2 = df.norm_mammo[idx]

    plt.imshow(mammogram_uint8_by_cv2, interpolation='nearest')
    plt.show()

    return

def oversampling(df_train):

    print(df_train.malignancy.value_counts())

    max_size = df_train['malignancy'].value_counts().max()
    lst = [df_train]
    for class_index, group in df_train.groupby('malignancy'):
        lst.append(group.sample(max_size-len(group), replace=True))
    df_train_oversampled = pd.concat(lst)

    print(df_train_oversampled.malignancy.value_counts())

    return df_train_oversampled

def get_class_weights(df):

    neg, pos = np.bincount(df['malignancy'])
    total = neg + pos
```

```
    # get weights
    weight_for_0 = (1 / neg)*(total)/2.0
    weight_for_1 = (1 / pos)*(total)/2.0
    # get dict
    class_weight = {0: weight_for_0, 1: weight_for_1}

    print('Weight for class 0: {:.2f}'.format(weight_for_0))
    print('Weight for class 1: {:.2f}'.format(weight_for_1))

    return class_weight


def get_weighted_loss(pos_weights, neg_weights, epsilon=1e-7):
    """
    Return weighted loss function given negative weights and positive weights.
    """
    def weighted_loss(y_true, y_pred):
        """
        Return weighted loss value.

        Args:
            y_true (Tensor): Tensor of true labels, size is (num_examples, num_classes)
            y_pred (Tensor): Tensor of predicted labels, size is (num_examples, num_classes)
        Returns:
            loss (Float): overall scalar loss summed across all classes
        """
        # initialize loss to zero
        loss = 0.0


        # for each class, add average weighted loss for that class
        loss += -1 * (K.mean(pos_weights * y_true * K.log(y_pred + epsilon) \
                    + neg_weights * (1 - y_true) * K.log(1-y_pred + epsilon)))

        return loss

    return weighted_loss


def check_leakage(df1, df2, patient_col):
    """
    Returns:
        leakage (bool): True if there is leakage, otherwise False
    """
    df1_patients_unique = set(df1[patient_col].values)
    df2_patients_unique = set(df2[patient_col].values)
    patients_in_both_groups = list(df1_patients_unique.intersection(df2_patients_unique))
    print(f'These patients are in both the training and test datasets: {patients_in_both_groups}')
    n_overlap = len(patients_in_both_groups)
    print(f'There are {n_overlap} Patient IDs in both the training and test sets')
    print('')
    leakage = n_overlap > 0

    return patients_in_both_groups

def plot_pixel_values(df):

    idx = np.random.randint(len(df))
    im = df.array[idx]
    sns.distplot(im.ravel(),
                 label=f'Pixel Mean {np.mean(im):.2f}',
                 kde=False)

    plt.legend(loc='best')
    plt.title('Distribution of Pixel Intensities in the Image')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('# Pixels in Image')

    print('Histogram of pixel distribution from a random image')

def plot_label_frequency(df1, df2):

    """
    Returns the label frequency for each dataset
    """
```

```
    train_freq_pos = [df1.loc[df1['malignancy']==1].shape[0]]
    train_freq_neg = df1.loc[df1['malignancy']==0].shape[0]
    test_freq_pos = df2.loc[df2['malignancy']==1].shape[0]
    test_freq_neg = df2.loc[df2['malignancy']==0].shape[0]

    data = pd.DataFrame({"Dataset": "train", "Label": "Positive", "Images": train_freq_pos})
    data = data.append({"Dataset": "train", "Label": "Negative", "Images": train_freq_neg}, ignore_index=True)
    data = data.append({"Dataset": "test", "Label": "Positive", "Images": test_freq_pos}, ignore_index=True)
    data = data.append({"Dataset": "test", "Label": "Negative", "Images": test_freq_neg}, ignore_index=True)

    f = sns.barplot(x="Dataset", y="Images", hue="Label" ,data=data)
    f.set_title('Label Frequency')

    print(f'There are {train_freq_pos} positive and {train_freq_neg} negative training labels.')

    return

def show_tif(local_folder, df):

    idx = np.random.randint(len(df))
    file_name = df.images[idx]
    im = tiff.imread(local_folder+file_name)
    cv2.imshow('Title', im.astype(np.float32)/np.max(im))
    cv2.waitKey(0)

    return

def show_cv2(local_folder, file_name):

    im = cv2.imread(local_folder+file_name, -1)
    cv2.imshow('Title', im.astype(np.float32)/np.max(im))
    cv2.waitKey(0)

    return

def save_to_png(df, folder):
    """
    Parameters
    ----------
    df : data frame
        train or test data frame.
    folder : string
        train or test destination folder.
    """
    files = df['images']

    for count, file in enumerate(files):
        array = df.norm_mammo[count]
        img = Image.fromarray(array,'RGB')
        img.save('Data/out/'+folder+'/'+file[:-3]+'png')

    return

def create_df_val(df_train):

    # filter by positives and negatives labels and sort by patient (avoid overlap)
    df_val_p = df_train.loc[df_train['malignancy'] == 1].sort_values(['patient_id'])[:25]
    df_val_n = df_train.loc[df_train['malignancy'] == 0].sort_values(['patient_id'])[:25]
    df_val = pd.concat([df_val_p, df_val_n])

    df_train_p = df_train.loc[df_train['malignancy'] == 1].sort_values(['patient_id'])[25:]
    df_train_n = df_train.loc[df_train['malignancy'] == 0].sort_values(['patient_id'])[25:]
    df_train = pd.concat([df_train_p, df_train_n])

    # shuffle
    df_val = df_val.sample(frac=1).reset_index(drop=True)
    df_train = df_train.sample(frac=1).reset_index(drop=True)

    return df_val, df_train


def show_png_images(df, label, IM_DIR):

    df_class = df.loc[df.malignancy == label]
```

```
    # Extract numpy values from Image column in data frame
    images = df_class['images_png'].values

    # Extract 16 random images from it
    random_images = np.random.choice(images, size=25, replace=False).tolist()
    # Location of the image dir
    img_dir = IM_DIR

    print(f'Display Random Images from class: {label}')

    # Adjust the size of your images
    plt.figure(figsize=(40,40))

    # Iterate and plot random images
    for i in range(25):
        plt.subplot(5, 5, i + 1)
        img = plt.imread(os.path.join(img_dir, random_images[i]))
        plt.imshow(img, cmap='gray')
        plt.axis('off')

    # Adjust subplot parameters to give specified padding
    plt.tight_layout()

    return

def find_duplicates(my_list):
    import collections
    print([item for item, count in collections.Counter(my_list).items() if count > 1])
    return

def get_train_generator(df, image_dir, x_col, y_cols, shuffle=True, batch_size=8,
                        seed=1, target_w = 128, target_h = 128,
                        samplewise_center=True, samplewise_std_normalization=None,
                        rescale= None):
    """
    Returns:
        train_generator (DataFrameIterator): iterator over training set
    """
    print("getting train generator...")
    # normalize images either between -1 and 1 or between 0 and 1.
    image_generator = ImageDataGenerator(
        samplewise_center= samplewise_center, #Set each sample mean to 0.
        samplewise_std_normalization= samplewise_std_normalization, # Divide each input by its standard deviation
        rotation_range = 360,
        horizontal_flip= True,
        vertical_flip= True,
        rescale= rescale # Normalize between [0:1].
        )


    # flow from directory with specified batch size
    # and target image size
    generator = image_generator.flow_from_dataframe(
            dataframe=df,
            directory=image_dir,
            x_col=x_col,
            y_col=[y_cols],
            class_mode="raw",
            batch_size=batch_size,
            shuffle=shuffle,
            seed=seed,
            target_size=(target_w,target_h))

    return generator


def get_test_and_valid_generator(valid_df, test_df, train_df, x_col, \
                             y_cols, sample_size=100, batch_size=8, seed=1, \
                            target_w = 320, target_h = 320,
                            samplewise_center=True, samplewise_std_normalization=True,
                            rescale= None):
    """
    Returns:
        test_generator (DataFrameIterator) and valid_generator: iterators over test set and validation set respectively
```

```
    """
    print("getting train and valid generators...")
    # get generator to sample dataset
    raw_train_generator = ImageDataGenerator().flow_from_dataframe(
        dataframe= train_df,
        directory='Data/out/train/',
        x_col='images_png',
        y_col= ['malignancy'],
        class_mode="raw",
        batch_size=sample_size,
        shuffle=True,
        target_size=(target_w, target_h))

    # get data sample
    batch = raw_train_generator.next()
    data_sample = batch[0] # length = sample_size

    # use sample to fit mean and std for test set generator
    image_generator = ImageDataGenerator(
        samplewise_center= samplewise_center, #Set each sample mean to 0.
        samplewise_std_normalization= samplewise_std_normalization, # Divide each input by its standard deviation
        rotation_range = 0,
        horizontal_flip= False,
        vertical_flip= False,
        rescale= rescale # Normalize between [0:1].
        )

    # fit generator to sample from training data
    image_generator.fit(data_sample)

    # get test generator
    valid_generator = image_generator.flow_from_dataframe(
            dataframe=valid_df,
            directory= 'Data/out/validation/',
            x_col=x_col,
            y_col=[y_cols],
            class_mode="raw",
            batch_size=batch_size,
            shuffle=False,
            seed=seed,
            target_size=(target_w,target_h))

    test_generator = image_generator.flow_from_dataframe(
            dataframe=test_df,
            directory='Data/out/test/',
            x_col=x_col,
            y_col=[y_cols],
            class_mode="raw",
            batch_size=batch_size,
            shuffle=False,
            seed=seed,
            target_size=(target_w,target_h))

    return valid_generator, test_generator


def compute_label_freq(df):
    # labels in training set
    freq_pos = np.sum(df['malignancy'] == 1, axis = 0)
    freq_neg = np.sum(df['malignancy'] == 0, axis = 0)

    return freq_pos, freq_neg

def get_roc_curve(labels, predicted_vals, generator, base_model):
    auc_roc_vals = []
    for i in range(len(labels)):
        try:
            gt = generator.labels[:, i]
            pred = predicted_vals[:, i]
            auc_roc = roc_auc_score(gt, pred)
            auc_roc_vals.append(auc_roc)
            fpr_rf, tpr_rf, _ = roc_curve(gt, pred)
            plt.figure(1, figsize=(10, 10))
            plt.plot([0, 1], [0, 1], 'k--')
            plt.plot(fpr_rf, tpr_rf,
```

```
                label=labels[i] + " AUC: " + str(round(auc_roc, 3)))
            plt.xlabel('False positive rate')
            plt.ylabel('True positive rate')
            plt.title('ROC curve with '+ base_model.name)
            plt.legend(loc='best')
        except:
            print(
                f"Error in generating ROC curve for {labels[i]}. "
                f"Dataset lacks enough examples."
            )
    plt.show()
    return auc_roc_vals

def compute_gradcam(model, img, image_dir, df, labels, selected_labels,
                    layer_name):
    preprocessed_input = load_image(img, image_dir, df)
    predictions = model.predict(preprocessed_input)
    label = df.loc[df['images_png'] == img, 'malignancy'].iloc[0]

    print("Loading original image")
    plt.figure(figsize=(15, 10))
    plt.subplot(151)
    plt.title(f"Original class: {label}")
    plt.axis('off')
    plt.imshow(load_image(img, image_dir, df, preprocess=False), cmap='gray')

    j = 1
    for i in range(len(labels)):
        if labels[i] in selected_labels:
            print(f"Generating gradcam for class {labels[i]}")
            gradcam = grad_cam(model, preprocessed_input, i, layer_name)
            plt.subplot(151 + j)
            plt.title(f"{labels[i]}: p={predictions[0][i]:.3f}")
            plt.axis('off')
            plt.imshow(load_image(img, image_dir, df, preprocess=False),
                       cmap='gray')
            plt.imshow(gradcam, cmap='jet', alpha=min(0.5, predictions[0][i]))
            j += 1


def load_image(img, image_dir, df, preprocess=True, H=320, W=320):
    """Load and preprocess image."""
    img_path = image_dir + img
    mean, std = get_mean_std_per_batch(img_path, df, H=H, W=W)
    x = image.load_img(img_path, target_size=(H, W))
    if preprocess:
        x -= mean
        x /= std
        x = np.expand_dims(x, axis=0)
    return x


def get_mean_std_per_batch(image_path, df, H=320, W=320):
    sample_data = []
    for idx, img in enumerate(df.sample(100)["images_png"].values):
        # path = image_dir + img
        sample_data.append(
            np.array(image.load_img(image_path, target_size=(H, W))))

    mean = np.mean(sample_data[0])
    std = np.std(sample_data[0])
    return mean, std


def grad_cam(input_model, image, cls, layer_name, H=320, W=320):
    """GradCAM method for visualizing input saliency."""
    y_c = input_model.output[0, cls]
    conv_output = input_model.get_layer(layer_name).output
    grads = K.gradients(y_c, conv_output)[0]

    gradient_function = K.function([input_model.input], [conv_output, grads])

    output, grads_val = gradient_function([image])
    output, grads_val = output[0, :], grads_val[0, :, :, :]
```

```
    weights = np.mean(grads_val, axis=(0, 1))
    cam = np.dot(output, weights)

    # Process CAM
    cam = cv2.resize(cam, (W, H), cv2.INTER_LINEAR)
    cam = np.maximum(cam, 0)
    cam = cam / cam.max()
    return cam


## METRICS

def get_true_pos(y, pred, th=0.5):
    pred_t = (pred > th)
    return np.sum((pred_t == True) & (y == 1))


def get_true_neg(y, pred, th=0.5):
    pred_t = (pred > th)
    return np.sum((pred_t == False) & (y == 0))


def get_false_neg(y, pred, th=0.5):
    pred_t = (pred > th)
    return np.sum((pred_t == False) & (y == 1))


def get_false_pos(y, pred, th=0.5):
    pred_t = (pred > th)
    return np.sum((pred_t == True) & (y == 0))



def get_performance_metrics(y, pred, class_labels, tp=get_true_pos,
                            tn=get_true_neg, fp=get_false_pos,
                            fn=get_false_neg,
                            acc=None, prevalence=None, spec=None,
                            sens=None, ppv=None, npv=None, auc=None, f1=None,
                            thresholds=[]):
    if len(thresholds) != len(class_labels):
        thresholds = [.5] * len(class_labels)

    columns = ["", "TP", "TN", "FP", "FN", "Accuracy", "Prevalence",
               "Sensitivity",
               "Specificity", "PPV", "NPV", "AUC", "F1", "Threshold"]
    df = pd.DataFrame(columns=columns)
    for i in range(len(class_labels)):
        df.loc[i] = [""] + [0] * (len(columns) - 1)
        df.loc[i][0] = class_labels[i]
        df.loc[i][1] = round(tp(y[:, i], pred[:, i], thresholds[i]),
                             3) if tp != None else "Not Defined"
        df.loc[i][2] = round(tn(y[:, i], pred[:, i], thresholds[i]),
                             3) if tn != None else "Not Defined"
        df.loc[i][3] = round(fp(y[:, i], pred[:, i], thresholds[i]),
                             3) if fp != None else "Not Defined"
        df.loc[i][4] = round(fn(y[:, i], pred[:, i], thresholds[i]),
                             3) if fn != None else "Not Defined"
        df.loc[i][5] = round(acc(y[:, i], pred[:, i], thresholds[i]),
                             3) if acc != None else "Not Defined"
        df.loc[i][6] = round(prevalence(y[:, i]),
                             3) if prevalence != None else "Not Defined"
        df.loc[i][7] = round(sens(y[:, i], pred[:, i], thresholds[i]),
                             3) if sens != None else "Not Defined"
        df.loc[i][8] = round(spec(y[:, i], pred[:, i], thresholds[i]),
                             3) if spec != None else "Not Defined"
        df.loc[i][9] = round(ppv(y[:, i], pred[:, i], thresholds[i]),
                             3) if ppv != None else "Not Defined"
        df.loc[i][10] = round(npv(y[:, i], pred[:, i], thresholds[i]),
                              3) if npv != None else "Not Defined"
        df.loc[i][11] = round(auc(y[:, i], pred[:, i]),
                              3) if auc != None else "Not Defined"
        df.loc[i][12] = round(f1(y[:, i], pred[:, i] > thresholds[i]),
                              3) if f1 != None else "Not Defined"
        df.loc[i][13] = round(thresholds[i], 3)
```

```
    df = df.set_index("")

    return df

def get_accuracy(y, pred, th=0.5):
    """
    Compute accuracy of predictions at threshold.
    """
    accuracy = 0.0

    TP = get_true_pos(y, pred, th)
    FP = get_false_pos(y, pred, th)
    TN = get_true_neg(y, pred, th)
    FN = get_false_neg(y, pred, th)
    accuracy = (TP + TN) / (TP + FP + TN + FN)

    return accuracy


def get_prevalence(y):
    prevalence = 0.0
    prevalence = sum(y) / len(y)

    return prevalence

def get_sensitivity(y, pred, th=0.5):
    sensitivity = 0.0

    TP = get_true_pos(y, pred, th)
    FN = get_false_neg(y, pred, th)
    sensitivity = TP / (TP + FN)

    return sensitivity

def get_specificity(y, pred, th=0.5):
    specificity = 0.0
    FP = get_false_pos(y, pred, th)
    TN = get_true_neg(y, pred, th)
    specificity = TN / (TN + FP)

    return specificity

def get_ppv(y, pred, th=0.5):
    PPV = 0.0
    TP = get_true_pos(y, pred, th)
    FP = get_false_pos(y, pred, th)
    PPV = TP / (TP + FP)

    return PPV

def get_npv(y, pred, th=0.5):

    NPV = 0.0
    TN = get_true_neg(y, pred, th)
    FN = get_false_neg(y, pred, th)
    NPV = TN / (TN + FN)

    return NPV

def get_curve(gt, pred, target_names, curve='roc'):
    for i in range(len(target_names)):
        if curve == 'roc':
            curve_function = roc_curve
            auc_roc = roc_auc_score(gt[:, i], pred[:, i])
            label = target_names[i] + " AUC: %.3f " % auc_roc
            xlabel = "False positive rate"
            ylabel = "True positive rate"
            title = "ROC Curve and AUC"
            a, b, _ = curve_function(gt[:, i], pred[:, i])
            plt.figure(1, figsize=(7, 7))
            plt.plot([0, 1], [0, 1], 'k--')
            plt.plot(a, b, label=label)
            plt.xlabel(xlabel)
            plt.ylabel(ylabel)
            plt.title(title)
```

```
            plt.legend(loc='upper center', bbox_to_anchor=(1.3, 1),
                       fancybox=True, ncol=1)
        elif curve == 'prc':
            precision, recall, _ = precision_recall_curve(gt[:, i], pred[:, i])
            average_precision = average_precision_score(gt[:, i], pred[:, i])
            label = target_names[i] + " Avg. Prec.: %.3f " % average_precision
            plt.figure(1, figsize=(7, 7))
            plt.step(recall, precision, where='post', label=label)
            plt.xlabel('Recall')
            plt.ylabel('Precision')
            plt.title("Precision-Recall Curve")
            plt.ylim([0.0, 1.05])
            plt.xlim([0.0, 1.0])
            plt.legend(loc='upper center', bbox_to_anchor=(1.3, 1),
                       fancybox=True, ncol=1)

def plot_calibration_curve(y, pred, class_labels):
    plt.figure(figsize=(20, 20))
    for i in range(len(class_labels)):
        plt.subplot(4, 4, i + 1)
        fraction_of_positives, mean_predicted_value = calibration_curve(y[:,i], pred[:,i], n_bins=20)
        plt.plot([0, 1], [0, 1], linestyle='--')
        plt.plot(mean_predicted_value, fraction_of_positives, marker='.')
        plt.xlabel("Predicted Value")
        plt.ylabel("Fraction of Positives")
        plt.title(class_labels[i])
    plt.tight_layout()
    plt.show()
```

APPENDIX 2 - WORK PLAN

The objectives decided with Dr. Wells for the interim report have been successfully completed which makes it possible to move on to the second part of the thesis project that will start after the exam period.

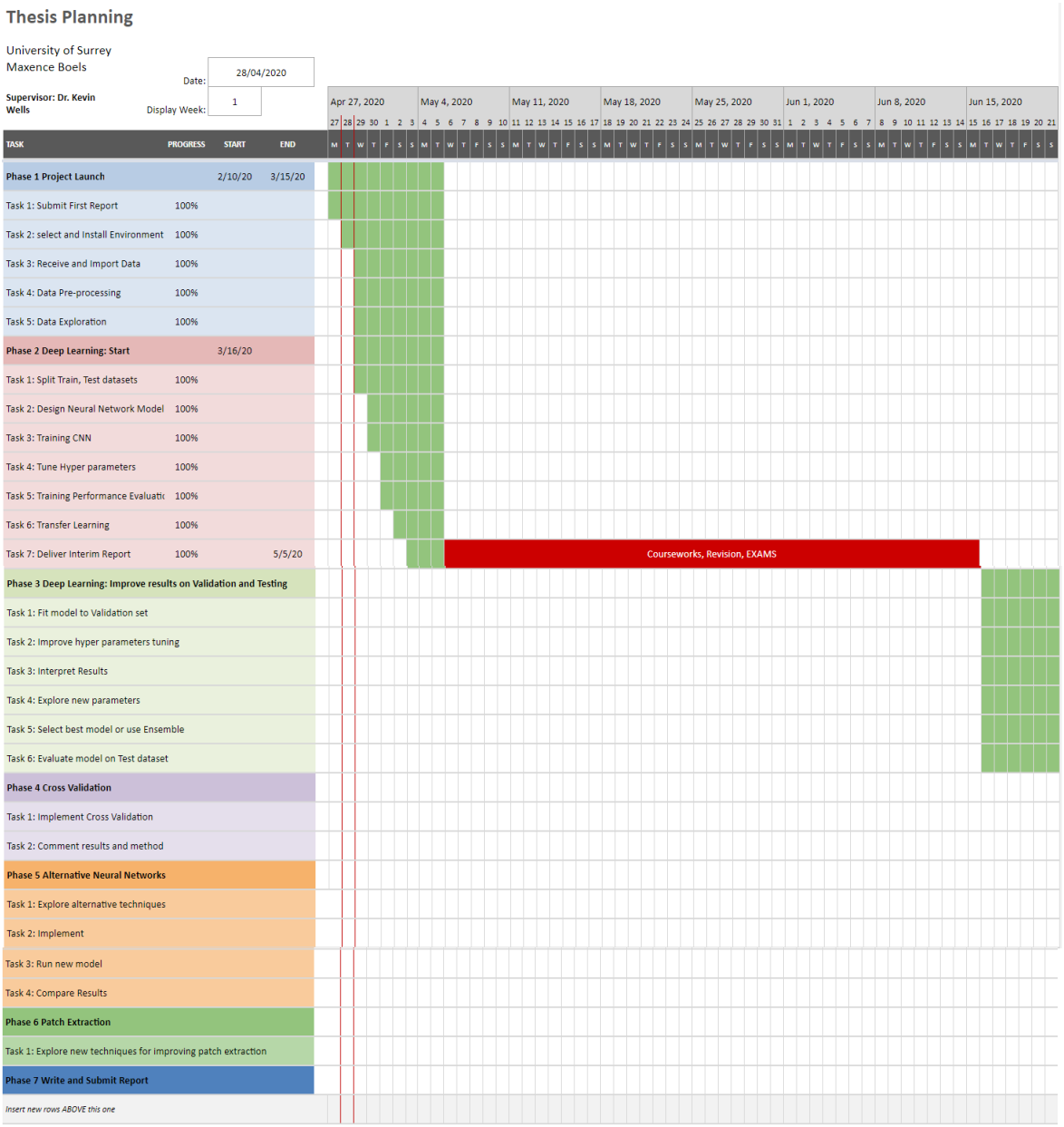

Figure 0.1 - Gantt chart

APPENDIX 3 – TRAINING SUMMARY

Professional Skills

Professional Skill	Training needed Y/N?	Current status of training. If not complete, what is the plan to complete it. If training was not needed, state how it had been completed previously.
Report writing	N	
Research methods, literature reviewing (training provided by the library later in the semester – see timetable)	N	
Oral presentation skills	N	
Time planning/Project planning	N	
Plagiarism training (Go to SurreyLearn and find it in The Student Common Room)	N	

Specialist Skills

Note you should adapt this table specifically to your project, this template is just given as a guidance.

Specialist skill	Training required Y/N?
Programming skills – e.g. MATLAB, C++, Python Note MATLAB online learning and introductory session in intro week: http://info.ee.surrey.ac.uk/Teaching/Courses/matlab/	N
How have I undertaken and will undertake training in programming if required?	
Mathematics – Refreshing or developing knowledge in maths. e.g. Maths Drop In Centre available: http://personal.ee.surrey.ac.uk/Personal/W.Wang/MathsDropInCentre.html The "Casual Drop-In Use" is available to MSc students.	N
How have I undertaken and will undertake training in mathematics if required?	
Hardware programming, e.g. Arduino, Raspberry PI, FPGA Note that the Electronics and Amateur Radio Society is a great extracurricular way of broadening your skills in firmware programming with devices like Arduino. Why not go and join them and attend the courses that students deliver to students? More info at: https://www.ussu.co.uk/ClubsSocieties/societies/ears/Pages/home.aspx	N
How have I undertaken and will undertake training in programming if required?	/
Practical skills, e.g. soldering, test and measurement, PCB design You will need to discuss more specifically the details with your supervisor if you undertake a practical project of this nature and also you should ensure you have completed a risk assessment for specialist health and safety.	N
How have I undertaken and will undertake training in practical skills if required?	/
Specialist simulation packages. Is there a specialist simulation package that you need to learn use? If you do need to learn a specific simulation package you will certainly need to discuss this with your supervisor and find out how you will learn it.	N
How have I undertaken and will undertake training in programming if required?	/
Specialist test and measurement equipment. Some projects involve undertaking a number of practical measurements and may require learning specifically how to use the equipment required for the task.	N
How have I undertaken and will undertake training in programming if required?	/

Specific knowledge

Note you should adapt this table specifically to your project, this template is just given as a guidance.

Background state of the art
What are the key words or phrases that you need to search for in the literature regarding the state of the art in relation to your project? You should have discussed this with your supervisor. There may be more than five key words or phrases so you can add more if you need to. 1. CNNs 2. Neural Networks 3. Breast Cancer 4. Training and Testing a model
Who are the key scholars in the field where research papers they produce are of relevance to read? You should make note of the key publications you have been reading and have identified as an important resource in completing the literature review of your project that you should have now done. See References

Theoretical knowledge for your project
What are the key items of theory you need to understand in order to undertake the work relevant to your project? Note that there may be more than five. If so you need to insert them. How have you progressed with this theory? 1. CNNs 2. Neural Networks 3. Breast Cancer 4. Training and Testing a model
How have you obtained this knowledge, is it essential for you to study particular module(s) to ensure you have the knowledge you require to do your project? What textbooks or other resources are there available to help deepen your learning of the background theory? How is the best way to learn what you need to know? This is something you should have established with your supervisor: Literature Review Online Courses University's courses Library books …