Application of Artificial Intelligence in Medical Thermography

Christopher Galazis

Master of Science

School of Informatics

University of Edinburgh

2018

Abstract

This master thesis will focus on using data obtained from mammary glands using microwave thermography to build a classification system to correctly predict if a patient is at low or high risk of having cancerous tumour. Initially, various preprocessing methods are evaluated on the dataset, such as normalisation against ambient temperature, feature extraction, feature selection, dimensionality reduction and oversampling techniques. The most significant step is normalisation which has shown improvement in results of unseen data. Models that will be evaluated to determine best performer are random forest, XGBoost, k-nearest neighbours, three different variations of cascade correlation neural networks, deep neural network and convolutional neural network. From out of all these models the deep neural network obtain the best performance showing strong capabilities in effectively learning the training set while also generalising to the validation set.

Acknowledgements

I would like to thank my supervisor professor Igor Goryanin for entrusting me with this project and for all the support and guidance. Additionally, I give my thanks to professor Alexander Losev and Vladislav Levshinsky for collecting and preparing the dataset and providing helpful explanations.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Christopher Galazis*)

Table of Contents

Chapter 1

Introduction

Microwave thermorgraphy has seen applications within medical diagnosis, such as breast cancer detection and monitoring (Vesnin et al., 2017). Microwave thermography is an attractive alternative to usual diagnostic systems such as mammography but also can be used to support decisions by obtaining additional information, such as the thermal activity of the tissues. In recent years, it has been approved for medical usage and has seen increased usage (Vesnin et al., 2017). One of the main reasons is that the intensified research and technological improvements has allowed for the system to obtain adequate levels of sensitivity and specificity. Additionally, it is a completely safe (no radiation), noninvasive and fast method which allows it to be used for screening of people of any age, if they are pregnant or not and with any frequency. Its largest application is for breast cancer, but can be used to detect cancerous tumours at any location.

However, despite it being used more and more, being a relative new method, it is not widely used and not many medical or clinical professionals have the required training to interpret the results. As such, diagnostic systems are being developed to help tackle both issues (Zenovich et al., 2016), by allowing professionals to obtain information from the readings it should also boost its demand at various centers across different countries. The currently optimal model introduced for such a problem was a cascade correlation neural network (Zenovich et al., 2016). Here, the network will be further expanded and improved on. Additionally, the data collected will be evaluated using *random forest*, *XGBoost*, *k-nearest neighbours* and *support vector machine* with linear and radial basis function kernels. Furthermore, architectures for a deep neural network and a convolutional network will be proposed to tackle the task.

This thesis will start off with a brief background information on microwave ther-

mography used for medical applications, description of mammary glands and cancerous tumours, summary of the base cascade correlation neural network and a briefly detailed evaluation metrics used to compare the models. Following, the previous research on this task and dataset will be briefly presented. This includes a description of the dataset that was prepared to be used for machine learning tasks, preprocessing and then the results obtained when the data was evaluated against genetic algorithms, various neural networks and a cascade correlation network. The next chapter will be focused on various preprocessing techniques, feature extraction and their overall effectiveness. After the background material is covered, non-neural network models, a deep neural network and a convolutional network will be presented and evaluated, each within their own chapter. The thesis ends with general conclusions on the models tested and some future extension.

Chapter 2

Background Material

2.1 Medical Microwave Thermography

The development of the military radar system in the 1930s was the first appearance of microwave imaging for detecting objects afar which could also been concealed or obstructed (Skolnik, 2018; Fallahpour, 2014). Its development was based on the theoretical research of James Clerk Maxwell and the experimentally verified efforts of Heinrich Hertz (Skolnik, 2018). The usage of microwave thermography for medical and clinical applications has emerged from the 1970s and early 1980s (Myers et al., 1979; Bolomey et al., 1982; Peronnet et al., 1983; Pichot et al., 1985). However, greater interest and advances were shown during the late 1990s for such applications (Conceicao et al., 2016). Such recent applications of microwave thermograhpy that have seen great success are detecting breast cancer (Vesnin et al., 2017), monitoring thermal denaturation of albumin (Ivanov et al., 2018), diagnosing carotid artery disease (Drakopoulou et al., 2018), measuring brown adipose tissue activity (Crandall et al., 2018), detecting rheumatoid arthritis (Pentazos et al., 2018), detecting inflammation levels in joints of people with axial spondyloarthritis (Laskari et al., 2018), monitoring brain temperature (Rodrigues et al., 2018), evaluating transcapillary water exchange in the lungs (Bondar et al., 2017) and

The increase in its usage for medical and clinical applications can be attributed to the fact that the technical aspects have improved, such as reduced cost of hardware components in parallel with increased computing performance (Semenov, 2009). In addition, through recent years the results obtained from microwave thermography are becoming more and more accurate and reliable (Vesnin et al., 2017). Also, it has been applied to various medical applications such as imaging of the extremities, brain

and cardiac and cancer detection and monitoring of the mammary glands and lungs (Semenov, 2009).

The are some benefits of using microwave thermography, especially for breast cancer, against other methods. The prominent advantage is that it is a completely safe technique which allows it to be used more frequently by people of any age and under any condition, like for women during pregnancy and lactation (Semenov, 2009; Vesnin et al., 2017). This is because it captures data by using low frequency microwave waveforms unlike other methods that emit ionising radiation. In combination with the fact that it can be used more often, it is a cheap and readily available method. Additionally, it is a pure noninvasive fast method allowing to capture data even without any direct contact with the skin (Vesnin et al., 2017). Lastly, it complements the data obtained from other methods, like structural information of the tissue, by providing information on the thermal activity of the cells, indication of the rate of increase of cancerous cells and the level of risk for mutagenesis of normal cells to tumorous (Vesnin et al., 2017).

Microwave thermography obtains data by capturing the levels of electromagnetic radiation omitted by the tissues in the microwave range (300 GHz − 300 MHz) (Vesnin et al., 2017). The frequency used for each problem depends on the temperature and biophysical parameters of the tissues and the desired intensity of the electromagnetic radiation to capture (Vesnin et al., 2017; Tipa and Baltag, 2006). It is able to detect cancerous tumors or produce an image of the examined area based on the variation of the dielectric properties of the various biological tissues (Semenov, 2009). Depth detection is possible because the wave lengths of microwaves can avoid significant absorption by the tissues that precede the malignant, especially compared to infrared radiation (Tipa and Baltag, 2006). It has been found that the level of water in the tissues can impact the dielectric properties of the tissues, with a significant difference between higher, such as muscle, and lower levels, like fat and bone (Gabriel et al., 1996a,b). In addition, both physiological and pathological conditions can alter the levels of dielectric properties of the tissues (Semenov, 2009). One such pathological condition is the presence of cancerous cells that are rapidly expanding (Surowiec et al., 1988).

Using microwave thermography can capture the temperature at the skin and at a depth from the surface noninvasively, which is particular useful for diagnosing and monitoring treatment progress of cancerous tumors (Vesnin et al., 2017). It has been found that tumors emit heat which is analogous of their growth rate (Gautherie, 1980). As the tumorous cells grow, they replicate them selves at a much higher rate leading

to the release of higher amounts of energy compared to neighbouring healthy cells. The tumors' ability to create new vasculature will determine its maximum volume (Schneider and Miller, 2005). At such a stage, cell growth and cell death rate reach an equilibrium. However, the growth slows down resulting to near normal temperature readings making cancer detection more difficult for such cases (Vesnin et al., 2017).

2.2 Mammary Glands

2.2.1 Anatomy

The female breasts or its medical term mammary glands is an organ that its main function is lactation for their infant and consists primarily of milk-producing glands and fatty tissues (bre, 2018). The two mammary gland tissues are symmetrically located between the collarbone, lower ribs, sternum and the armpit. They are supported by the outer chest wall and a layer of skin. While they are symmetrically positioned their shape or size might be asymmetric.

Each mammary gland contains 15 to 20 sections called lobes that form a circular formation around the nipple. Each one consists of smaller glands called lobules. Then they are further divided to clusters of acini, being the smallest functional unit. The lobules and to further extension the acini are responsible for producing milk which is transported through the milk ducts to the nipple. To each lobe there are connected around six to eight milk ducts. Additionally, the fibrous connective tissue surrounds, separates and supports the lobes which radiate out from the nipple. Furthermore, there is a pigmented area surrounding the nipple called areola. The areola contains sebaceous and sweat glands. Also, around the breasts and armpits there are lymph nodes and lymphatic vessels which carry lymph fluid and white blood cells. Lastly, for the remaining composition of the breast there is between the lobes adipose (fatty) tissue and between the pectoral muscle, found by the chest wall, and the breast tissue there is rectrommary fat.

The human female mammary glands occurs changes during puberty, menstrual cycle, pregnancy, breastfeeding and menopause. During early puberty the glands start to grow in size and volume mainly due to the hormones estrogen, progesterone and growth hormones. During the menstrual cycle morphological changes to the glands occur because of hormonal variation. When the woman is pregnant, her mammary glands change shape, are enlarged and the levels of prolactin increase to prepare for

milk production for the infant. After pregnancy hormone levels of estrogen and progesterone decrease to allow for production of milk. Finally, at the stage of menopause there is a lack of ovarian hormones which ends up causing signs of atrophy, resulting to decrease density.

2.2.2 Breast Cancer

There are different types of tumours that can affect the various parts of the mammary glands (Conceicao et al., 2016; ben, 2017). The tumours can be subdivided into two main categories benign (noncancerous) and malignant (cancerous). Benign tumors do not metastasize to other parts of the body, they might not be harmful and usually after treatment or surgical removal they do not reappear (ben, 2017). However, with malignant tumours they grow more aggressively, can be invasive and spread to nearby areas or noninvasive and be confided within the affected area and treatment is always required (Conceicao et al., 2016).

Some of the most common affected areas of the mammary glands for benign tumours are (ben, 2017):

- Most common is fibroadenomas, where fibrous and glandular connective tissue have grown faster than usual in the lobules causing an increase in size
- Fibrocystic changes impacts the milk ducts and surrounding tissues causing them to enlarge and increase in tenderness. It is also possible that the breast tissue thickens. This is caused by hormonal changes during menstrual cycles
- Simple cysts are sacs that are filled with fluid and occurs in the breast tissue between the lobes
- Intraductal papillomas tumours form at the milk ducts near the nipple and are formed from gland and fibrous tissue and blood vessels
- Traumatic fat necrosis happens when the mammary gland sustained an injury and results to an irregular lumpy formation of the fatty tissue
- For men, gynaecomastia can be observed which it causes enlargement of the breast tissue of either one or both of them

Cancerous tumours can impact the lobules and milk ducts and depending if it is invasive or not it can expand further than their basement membrane. The most frequent cancerous tumours observed are (Conceicao et al., 2016):

- Invasive ductal carcinoma, where abnormal cells are found to originate on the linings of the milk ducts
- Invasive lobular carcinoma, where abnormal cells are found in the lobules of the mammary gland
- Noninvasive ductal carcinoma, where the abnormal cells are limited within the milk ducts
- Noninvasive lobular carcinoma which in its current state is not cancerous but has high risk of developing to one

Cancerous tumours are formed when a group of altered cells grow and divide abnormally but do not follow the programmed cell death (Jemal et al., 2013). The processes when the cells are destroyed is called apoptosis. The immune system is able to handle and destroy such cells that are being formed, but if too many cells are mutated it may become overwhelmed (Conceicao et al., 2016). The rate of growth of such mutated cells indicates whether the tumour formed is benign or malignant (Jemal et al., 2013). For benign tumours, the rate of growth is more constrained so it does not expand any further, with causing harm if the tumour is causing pressure to nearby organs or if it produces harmful hormones. On the other hand, malignant tumours continue to grow at a fast rate and will expand to nearby healthy tissues and destroy them through metastases.

2.2.3 Dielectric Properties

All live biological tissues have electrical conductivity and the impedance of the current passing can be measured, hence the dielectric properties that make the usage of microwave thermography for medical purposes viable (Tipa and Baltag, 2006). A tissue is formed when similar cells, in which they consist of a cell membrane and a intracellular matrix, are grouped in conjunction with their extracellular matrix. In regards to the ability to conduct an electric current, the intracellular and extracellular matrices act as resistors because of their ionic solutions. On the other hand, due to the fact that the cell membrane contains a lipid bilayer and various proteins it can take on the role

of the capacitance, with its impedance value closely associated with the frequency of the electric signal. Because of the dependency on the frequency, the cell membrane can act either as an insulator, lower frequencies, or as a conductor, higher frequencies. At lower frequencies, the impedance becomes more resistive and so the current is limited to pass through the extracellular matrix. Oppositely at higher frequencies, the impedance allows the flow of current in the cell membrane.

Various studies having being conducted over the years to determine what if there are differences of the dielectric properties between normal and cancerous tissues, which their results are summarised in a review by Ng et al. (2008). It was concluded that measuring the impedance of the tissues presents significantly distinguishable values between them, at least for low frequency wave length. Some of the properties noted for the cancerous cells that allow to separate the two is the increased water content and sodium levels and the cell membranes had their electrochemical properties and permeability altered. Additionally, as the temperature in the cells changes so does their dielectric properties caused by the structural changes of the molecules, which result to an increase in density, orientation and rotation speed (Kyber et al., 1992; Sudsiri et al., 2007). Furthermore, from a past research (Meaney et al., 2000) it has been observed that benign and malignant tumours have somewhat similar dielectric properties making it difficult to distinguish effectively between the two.

2.3 Cascade Correlation Neural Network

The cascade correlation neural network was proposed by Fahlman and Lebiere (1990) as an alternative to the traditional multilayer perceptrons. The advantage of this network based on the authors is that it is able to build and increase dynamically its topology and number of hidden units from a minimum network. This should allow one to avoid the tedious process of predefining an architecture and optimising it. Consequently, a further benefit is that minimum (optimal) memory storage is required to run and store such models.

In addition, the authors state that it can also be trained much faster in comparison to backpropagation because it incorporates weight freezing of units in conjunction with their incremental addition. Hence, the need to propagate the error signal backwards through the network becomes redundant. This limits the network to training only single layers at a time and so they have chosen to use *quickprop* (Fahlman, 1988), which they found it converges quickly and at a good optimum. One of the reasons that they

attributed the increase in training speed is because it is able to handle the ”moving target” problem (Fahlman and Lebiere, 1990). The problem arises when weights of all the units in the network are readjusted simultaneously to try to adapt to a problem (backpropagation error) which is constantly changing (Fahlman and Lebiere, 1990; Rohwer, 1989).

Based on the description given by the authors, the network starts out with only the input and output layer, with the number of units in the output layer determined by the problem, and are fully connected between them. An iterative process follows in which the output layer updates its weights and produces a vector of predictions then a candidate hidden layer is added, containing one or more units. Each unit has input from from the initial input vector and all units that are found in previous layers, with the output layer's number of inputs grows analogous with each iteration. This processes continues until a maximum number of layers has been added to the model, the error does not decrease any more or it reached a minimum threshold. An example of a model after adding two hidden layers is shown in *Figure 2.1*.

Each of the candidate hidden layers, before being added to the network, are trained through quickprop individually in an attempt to maximise the correlation coefficient between the residual error $(predicted - true)$ of the output layer for the current iteration and the prediction of the hidden layer. Other than using the output of the output layer there are no connections yet between this hidden layer and the output. Once the correlation reaches a maximum threshold or does not improve anymore the hidden layer can be added to the network. For the candidate layer to become apart of the network, each of its units is linked with each of the output's units and has its weights on its input links frozen. This process ends a single iteration and the algorithm continues from retraining the links leading to the output layer, as for all other links have their weights frozen.

Because weights are being frozen after each iteration, output of highly erroneous hidden layers can propagate through the network hindering its training speed and achieving a minimum topology. One way to overcome this, as suggested by the authors, is to create a pool of candidate hidden layers in which all the layers are independent between each other and follow the same processes as with a single layer. Once their training is complete, a candidate hidden layer is selected with the highest correlation score from the pool. Therefore, a trade-off can be made between more optimal hidden layers, hence smaller network, or faster training as the number of layers in the pool increase or decrease respectively.

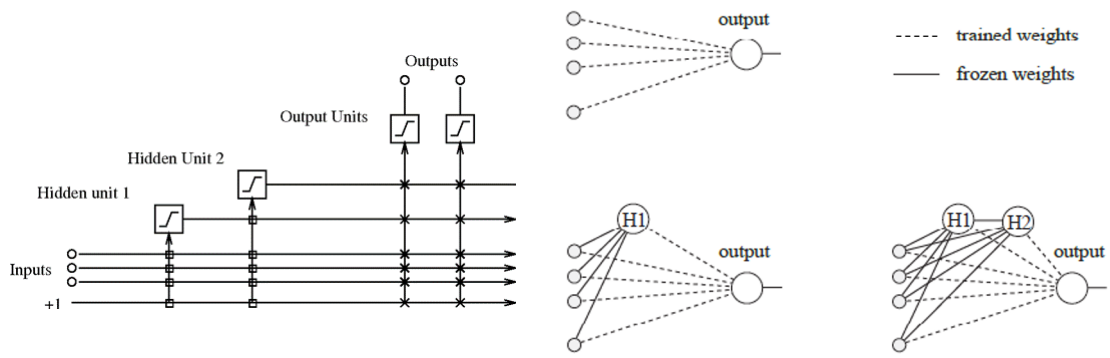

Figure 2.1: Example of a cascade correlation neural network after adding two hidden units (layers) to the network (Fahlman and Lebiere, 1990). Frozen weights are depicted with a square and a solid line and free weights with a cross and dotted line for the left and right image respectively.

2.4 Evaluation Metrics

When evaluating a medical diagnostic model it is important to assess its performance on the *specificity* and *sensitivity* metrics (Lalkhen and McCluskey, 2008). Before defining the two terms we must consider into which of the four subgroups a patient can belong to based on the output prediction of a classification algorithm:

- *True Positive* (TP): A patient is at high risk of a disease/condition and is correctly predicted as high risk
- *False Positive* (FP): A patient is at low risk of a disease/condition and is wrongly predicted as high risk
- *True Negative* (TN): A patient is at low risk of a disease/condition and is correctly predicted as low risk
- *False Negative* (FN): A patient is at high risk of a disease/condition and is wrongly predicted as low risk

Specificity, also known as the true negative rate, is the metric that quantifies the percentage of people who are truly healthy and are predicted correctly as healthy, A value closer to one indicates better classification performance in predicting healthy people. The metric is defined as such (Lalkhen and McCluskey, 2008):

$$Specificity = \frac{TN}{TN + FP} \tag{2.1}$$

Sensitivity, also known as the true positive rate, is the metric that quantifies the percentage of people who truly have the disease/condition and are predicted correctly as sick. A value closer to one indicates better classification performance in predicting sick people. The metric is defined as such (Lalkhen and McCluskey, 2008):

$$Sensitivity = \frac{TP}{TP+FN} \tag{2.2}$$

To evaluate and compare the performance between various models a more synoptic and general metric can be used like *accuracy*. It shows the percentage of people that have been correctly diagnosed against the whole evaluated population:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.3}$$

While accuracy is the most popular used metric for classification problems, its results become somewhat meaningless when there is class imbalance (Cohen et al., 2004). As an example, if the dataset contains 90% low risk patients and 10% high risk a classifier can blindly predict every sample as low risk and achieve 90% accuracy. An alternative metric that handles class imbalances is the *geometric mean* (G-Mean) of class accuracies on the positives and negatives, defined as following (Kubat and Matwin, 1997):

$$Geometric\_Mean = \sqrt{Specificity * Sensitivity} = \sqrt{\frac{TN}{TN+FP} * \frac{TP}{TP+FN}} \tag{2.4}$$

As stated by Kubat and Matwin (1997), this metric can be used by a classifier to train the data on so it will result in maximising the accuracies for the specificity and sensitivity while also maintaining a balance between them. However, when the negative samples drastically outnumber the positives the geometric mean is unable to give the intended priority to balance the classes (Cohen et al., 2004). A solution to this is to add weights to the specificity and sensitivity of the *equation 2.4*, based on the present number of positive and negative samples, so they gain equal importance and negate the imbalance (Cohen et al., 2004; Barandela et al., 2003).

A *receiver operating characteristics* (ROC) curve is a graphical representation of the accuracy of the model as the cut-off point of the threshold between deciding one class against the other varies (Hajian-Tilaki, 2013). The curve is plotted against the sensitivity (true positive rate) on the y-axis and 1-specificity (false positive rate) for the x-axis.

The optimal threshold point to obtain the highest accuracy is the most top left point on the plotted curve. However, other threshold points can be selected to refine

the results depending on preference for specificity or sensitivity. The area under the ROC curve indicates the overall predictive accuracy of the classification model with 1.0 being the optimal value and 0.5 the worst. At a value of 0.5 or below the model can be replaced with a random class selector for each sample.

Plotting the curve can enable someone to quickly and easily identify which from a set of comparing models has better performance on the test set. As an example, *Figure 2.2* shows the ROC curve plots and their respective area under the curve for two separate models. Between the two models there is a clear indication from the plots that the second model outperforms the first one. It has obtained the most top-left point between the two. Furthermore, this is also reflected on the area under the curve with a value of 0.75 comparing to 0.52. Also, by plotting the performance of a random classifier (dotted line) it becomes noticeable that the first model is just slightly an improvement, suggesting a lack of predictive capabilities.

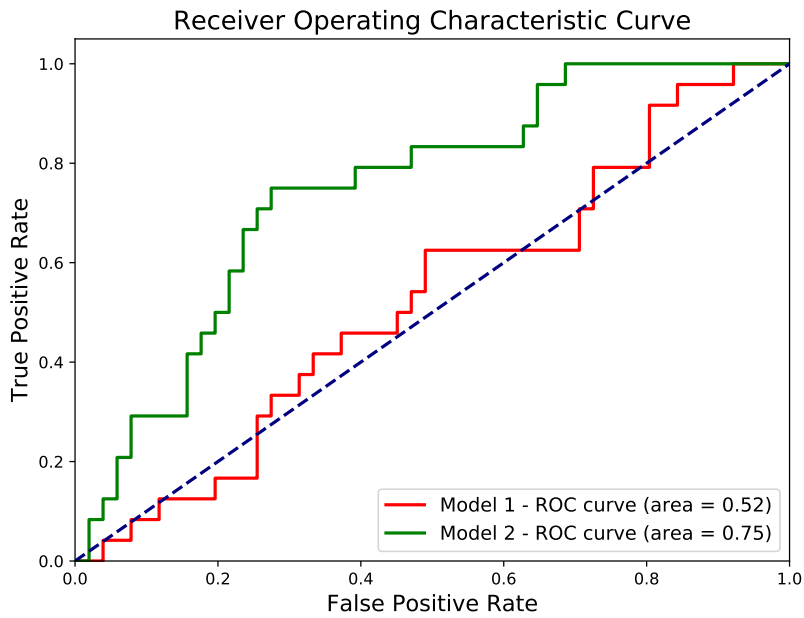

Figure 2.2: An example of a receiver operating characteristic curve for a two class problem and its calculated area under the curve for two different models. The dotted line represents the expected performance of a random binary classifier

Chapter 3

Previous Work

3.1 Dataset Description

The dataset for each sample (row) contains information about a single mammary gland. It consists of a total of 319 individual mammary glands that are indicated as healthy or low risk for presence of cancer (or otherwise labeled as *class 0* when passed to the algorithms). The remaining samples of 407 are the ones that are classified as high risk (or *class 1*), either there is presence of cancer or structural changes that are not cancerous but can not be easily distinguishable. There are an additional eight mammary glands that have previously underwent surgery, in which it distorts the temperature readings taken from thermography (Zenovich et al., 2016), and will not be included. Specifically, from the high risk samples 13 are classified as diffuse cancer, 185 as nodal cancer, 119 as diffuse changes with no presence of cancer and 90 as nodal changes with no presence of cancer.

However, to classify if a patient is at low or high risk the respective pair of mammary glands must be taken into account. When pairing them together both glands must be low risk to maintain a classification of low risk. On the other hand, if one or both are classified as high risk, then the patient will also we classified as high risk. Any gland in the dataset that happens to not have its matching pair present will be discarded. With pairing, the resulting number of samples that can be formed are 77 that are classified as low risk and 286 as high risk, with none of the glands required to be discarded.

Each of the samples, prior to pairing, consists of a feature vector the contains temperature values from various points on the mammary gland, inputs from medical professionals, meta-data for identification and a class label of either healthy or sick. Specifically, the medical professional adds for each patient their age, synoptic

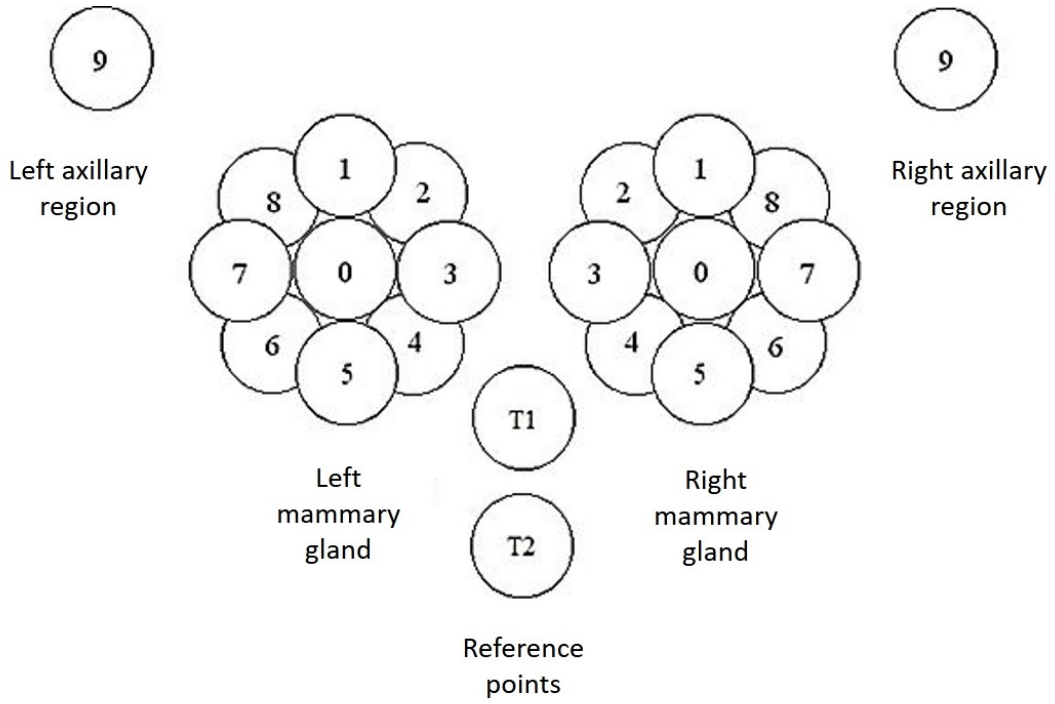

Figure 3.1: Sampling points for each mammary gland (0-8) and at the axillary point (9). Points T1 and T2 are used as reference values when normalising the values to compensate for environmental temperature differences (Zenovich et al., 2016).

history, clinical diagnosis, presence of pain, ovulation cycle for women, results from various other tests, etc. Additionally, each sample contains an identifier that associates to a specific person, while maintaining their anonymity, and which of the two glands the data sample is recorded for. These two attributes were used to pair the glands as described previously.

The temperature points off the mammary glands were captured through the means of medical thermography. The device RTM-01-RES[1] was used to capture the temperature values from the skin and the soft tissue at a depth of 5cm (Zenovich et al., 2016). The locations captured were at the nipples (marked as point 0), around the mammary gland with a total of eight locations (points ranging from 1 to 8), the axillary regions (point 9) and two reference points at the lower chest (points $T1$ and $T2$) as seen in figure 3.1. The two points captured below the chest are used to normalise the temperature values against the ambient environmental temperatures, which are not constant across different times and/or locations when the temperatures was sampled.

[1]http://maximed.pro/en/catalog/diagnostic/135-rtm-01-res.html

3.2 Extracted Features

The temperature measurements of the skin and within a depth allow the construction of a thermal mapping to visualise the results. However, any slight abnormalities of the temperature might not be distinguishable or apparent to the clinical professional. As such, highly informative diagnostic features can be extracted from the measurements based on a mathematical formalisation of qualitative medical features (Zenovich et al., 2016). The variation of these extracted temperature features provide information on the presence or absence of abnormalities in the readings. The features can be separated into five categories based on the information they add. Following are listed some of the mathematical functions that can be used for the specific problem of cancer diagnosis using temperature measurements in mammary glands (Zenovich et al., 2016; Losev and Lvshinskiy, 2017):
Features that describe the temperature asymmetry between the two glands. This set of features assumes mirror symmetry between the two glands for healthy patients:

- Increased thermal asymmetry between corresponding points of the mammary gland:

$$\begin{aligned} &|t_{l,i} - t_{r,i}|, \text{ for } i = 1...8, \\ &\text{where } l \text{ and } r \text{ are the left and right gland respectively} \end{aligned} \tag{3.1}$$

- Increased dispersion of the temperature difference between the mammary glands:

$$\begin{aligned} &\sum_{i=0}^{8} (\Delta t_i - \bar{t}_m)^2, \\ &\text{where } \Delta t_i = t_{li} - t_{ri} \text{ and } \bar{t}_m = \frac{1}{9}\sum_{i=0}^{8} \Delta t_i \end{aligned} \tag{3.2}$$

- The increased value of the temperature of the nipple compared with the average values of neighbouring points of the breast:

$$\begin{aligned} &t_0 - \frac{t_i + t_{(i\%8)+1}}{2}, \\ &\text{for each } i\text{=1...8} \end{aligned} \tag{3.3}$$

- The root mean squared difference between the mirrored points from the left onto the right mammary gland:

$$\sqrt{\sum_{i=0}^{8} \frac{(t_{l,i} - t_{r,i})^2}{9}} \tag{3.4}$$

- The mean difference between the same subregions of the left and right mammary glands:

$$\left|\frac{t_{l,0}+t_{l,i}+t_{l,(i\%8)+1}}{3}-\frac{t_{r,0}+t_{r,i}+t_{r,(i\%8)+1}}{3}\right|, \quad \text{where } i\text{=1..8} \tag{3.5}$$

- The difference of standard deviation between the left and right mammary glands:

$$\left|\left(\sqrt{\sum_{i=1}^{8}\frac{(t_i-t_{avg})^2}{8}}\right)_l-\left(\sqrt{\sum_{i=1}^{8}\frac{(t_i-t_{avg})^2}{8}}\right)_r\right|, \quad \text{where } t_{avg}=\frac{1}{8}\sum_{i=1}^{8}t_i \tag{3.6}$$

Features that describe the increased dispersion of temperature values for an individual gland:

- The standard deviation of all points of a single gland:

$$\sqrt{\sum_{i=0}^{8}\frac{(t_i-t_{avg})^2}{9}}, \quad \text{where } t_{avg}=\frac{1}{9}\sum_{i=0}^{8}t_i \tag{3.7}$$

- Temperature variation for each individual point against the average of a single gland:

$$t_{avg}-t_i, \quad \text{where } t_{avg}=\frac{1}{9}\sum_{i=0}^{8}t_i \text{ and } i\text{=0...8} \tag{3.8}$$

- Temperature variation of a three point neighbourhood against the average of the individual gland:

$$t_{avg}-t_i, \quad \text{where } t_{avg}=\frac{1}{9}\sum_{i=0}^{8}t_i \text{ and } i\text{=0...8} \tag{3.9}$$

Features that detect abnormal high temperature of the nipple against the rest of the points of the gland

- Increased nipple temperature in the affected mammary gland in comparison with the average temperature of the gland:

$$t_0-\frac{1}{8}\sum_{i=1}^{8}t_i \tag{3.10}$$

- Increased temperature of the nipple compared to each point of the mammary gland:

$$t_0 - t_i, \quad \text{for each } i=1...8 \tag{3.11}$$

Features that define a relationship between temperature measurements of the surface of the skin and at a depth:

- Temperature difference of the same point of the skin and at a depth of an individual gland:

$$t_{d,i} - t_{s,i}, \quad \text{for } i=0...8 \text{ and} \tag{3.12}$$

where d: measurement at a depth and s: measurements of the skin

Additionally, features can be obtained based on using methods of physico-mathematical modeling of the temperature values of the two glands.

3.3 Data Normalisation

The temperature data samples were collected under a variation of environmental temperatures ranging from 20 to 27 degrees Celsius (Losev and Lvshinskiy, 2015). In the studies by Anisimova (2013); Kobrinskiy (2008), they state that when analysing the temperature values for prediction systems they should also take into account any external conditions that can affect the results. As such, a following research by Losev and Lvshinskiy (2015) proposed and evaluated a normalisation method based on the two control points T_1 and T_2. The correlation coefficient of each of the data points t_0 to t_9 for both skin and at a depth was found to have a strong linear correlation with the two control points as the temperature increases.

This indicates the potential that the temperatures for each point can be fitted with a curve using linear polynomials against one of the two control/reference points. The temperatures can then be normalised by constructing new points in which the values are shifted towards the control points. A linear regression model was fitted to the data using the least squares approach to find the optimal line. Additionally, the normalisation method that will be described was tested using hyperbolic, logarithmic, power and exponential regression. However, the non-linear regression lines were identical to that of the linear with a slight variation of 0.1 degrees. The difference was insignificant and

so linear regression can be used for fitting. The whole process is described as follows (Losev and Lvshinskiy, 2015):

For each temperature point $t_{d,i,j}$ pair it with one of its two control temperature points $T_{c,d,j}$, where $i = 0...9$, $c \in \{1,2\}$, $j \in \{skin, depth\}$ and $d = 0...n-1$, with n the size of the dataset:

1. Find the curve that results to the smallest sum of squares from the curve to the set of points between the temperature point (y-axis) and the control temperature (x-axis) (least square fit method). Hence using linear regression, we want to find a and b such that the error is minimised on the following function:

$$t_{d,i,j} = a * T_{c,d,j} + b$$

2. Calculate the average value of the temperature point such that:

$$Tavg_{c,j} = \frac{1}{n} \sum_{d=0}^{n-1} T_{c,d,j}$$

3. Update the temperature points using the following rule:

$$t_{d,i,j} = t_{d,i,j} + a * (Tavg_{c,j} - T_{c,d,j})$$

4. Replace the control points with the average value found:

$$T_{c,d,j} = Tavg_{c,j},$$

   for $d = 0...n-1$

3.4 Data Classification

Following will be defined how the dataset was split for a previous research by Zenovich et al. (2016). The training set consisted of 74 samples from healthy participants and 302 samples from patients. The rest of the data was used as the test dataset. The whole training dataset was represented as a 2D matrix to make passing to the algorithms easier. Furthermore, each row represented a single data sample from the dataset. The columns of the matrix represent the 44 temperature readings which consists of the 10

point measurements at the skin surface and at a depth for each of the mammary glands and the two control points, again for both skin and depth measurements:

$$\begin{pmatrix} t_0^1 & \cdots & t_{43}^1 \\ \vdots & \ddots & \vdots \\ t_0^k & \cdots & t_{43}^k \\ t_0^{k+1} & \cdots & t_{43}^{k+1} \\ \vdots & \ddots & \vdots \\ t_0^n & \cdots & t_{43}^n \end{pmatrix},$$

where t_i^j is the temperature value of j-th person (sample) for temperature measurement point i, with:

$j = 1..k$: Temperatures of healthy participants

$j = k+1..n$: Temperature values of sick patients

3.4.1 Classification using highly informative features

A modeling function and a certain subset X from the whole set of values will be defined as a way to classify if a patient is of high risk (Zenovich et al., 2016; Losev et al., 2011). The multidimensional characteristic feature will determine that a patient is at higher probability to be at risk, if the value obtained from the modeling function for that patient does not belong to the subset X (Losev et al., 2011).

A modeling function of the temperatures, as defined by Losev et al. (2011):

$$f^q = f^q(t_0, t_1, \ldots, t_{43}), \tag{3.13}$$

which is considered on the set of vectors:

$$\Omega_q = \{(t_0^j, t_1^j, \ldots, t_{43}^j)\},\ \forall j = 1...n \tag{3.14}$$

is denoted as:

$$f_j^q = f^q(t_0^j, t_1^j, \ldots, t_{43}^j),\ \forall j = 1...n \tag{3.15}$$

We can now define the following vector-values function:

$$f = f^q(t_0^j, t_1^j, \ldots, t_{43}^j),\ \forall j = 1...n \tag{3.16}$$

Now the characteristic function can be defined as the pair between the vector-values modeling function f and the subset X as: (f, X), where $X \subset R^m$. $X \subset R^m$

is the maximum subset of the domain of variation of the vector-valued function f. That is $(f_j^{q_1}, f_j^{q_2}, \ldots, f_j^{q_m}) \in X$ for $j = k+1, \ldots, n$ and $(f_j^{q_1}, f_j^{q_2}, \ldots, f_j^{q_m}) \in R^m \backslash X$ for $j = 1, \ldots, k$. In other words, X is the features of f that belong in one of the classes and do not belong in the other.

Thus, the goal of a search algorithm for the characteristic features is to find a "maximum" of sub-domains (Losev et al., 2011). The temperature function values in the sub-domains must belong to the corresponding mammary glands denoted as high risk, but not belong to the values corresponding to the mammary glands classified as low risk. However, characteristic features are only able to identify patients with high risk when there are prominent temperature anomalies. This is dealt with by introducing highly informative attributes to the search algorithm (Zenovich et al., 2016).

A highly informative attribute will be represented as a tuple of three values (f, V, X), where f is a feature vector of the temperature values which models its behaviour, $V = I(f, X)$ is the informative or weight of the characteristic and X is the area of change of the vector function (Losev et al., 2011). The informativeness of the characteristic is a quantitative feature that can allow to differentiate between the various classes. The following informative characteristics have been used for detection of lower limb disorders (Anisimova et al., 2016) and of cancer in the mammary glands (Losev et al., 2015):

1. Statistical information:

$$SI(f^q, X) = -\ln \frac{C(h,k)C(s,n-k)}{C(h+s,n)} \tag{3.17}$$

2. Heuristic information:

$$HI(f^q, X) = \frac{\max(\frac{k}{h}, \frac{n-k}{s})}{\min(\frac{k}{h}, \frac{n-k}{s})} \tag{3.18}$$

3. Combined information:

$$CI(f^q, X) = \sqrt{SI(f^q, X) * HI(f^q, X)} \tag{3.19}$$

Where:

- $C(n,k)$: Combination of k items from a set of n items, with $n \geq k$
- h: The number of samples (mammary glands) for healthy participants for which $f \in X$

- s: The number of samples (mammary glands) for sick participants for which $f \in X$

3.4.2 Classification using combined features

The dataset has an imbalance between the two classes so evaluating the results against the accuracy might result in a prediction that always favours the majority class. Thus, when evaluating using combined features, and specifically for medical diagnosis, the geometric mean of the specificity P and sensitivity C is taken to indicate the importance of correct classification for both classes (Zenovich et al., 2016):

$$\Delta = \sqrt{P * C} \tag{3.20}$$

In a previous research paper by Glazunov et al. (2015a), they proposed a method in which they can find a minimum set of newly constructed features that results to a high combined informativeness. A function f is set so that it combines linearly the functions from the base set of features. Additionally, given any weight coefficients, it is possible to find a linear boundary c separating the two classes on a diagnostic feature in such a way to have the highest possible combined information. Thus, a genetic search algorithm is used to find the best linear combination of functions that will maximise the informativeness. This algorithm was used as a setup step in constructing a minimum combined set of features for all the following algorithms used in evaluation (Zenovich et al., 2016).

Glazunov et al. (2015a) gave a more specific description of the genetic algorithm used. The term *individual* will be referred to a feature thats its function f is a linear combination of functions from the initial set of features and has a boundary c in which it results to the highest combined information for specific given weights. The fitted function of the individual constitutes the combined information. A genetic algorithm is an evolutionary process in which its population of *chromosomes* (the individuals) has its *genes* (set of functions used for linear combinations) 'mutated' or combined to create new chromosomes sequentially until an optimal solution is found. A fitness function needs to be defined so the algorithm can determine if an individual should be selected for the following steps of the algorithm. The selection is based on the fitness score of the individual against all other individuals, in which for this case the score will be based off the value from the combined information.

The algorithm mentioned previously is described here (Glazunov et al., 2015a):

1. Each of the individuals have a probability of being selected to be carried on to the next iteration as a parent. The probability is determined by:

$$P(k) = \frac{CI(k)}{\sum_{i=1}^{n} CI(i)},$$

where n is the total population of individuals, k is the kth individual from the available population and $CI(i)$ is the combined information of the ith individual.

2. From the selected individuals, all possible combinations between them are used to generate the next set that will be added to the population using a random crossover point on the chromosome. The child *individual* created has equal probability of selecting a gene from either of its two parents. Also there is a chance that one of the passed genes can mutate to a random one from the possible set of genes.

3. After the new set of individuals have been created, only the ones with the highest combined information will be carried on.

4. If from the generated children there is no significant increase in the combined informativeness then the algorithm has converged to a solution and the process terminates. Otherwise repeat from the first step with the selected children added to the population.

Inspired from the algorithm developed by Anisimova et al. (2011), a variation was proposed by Zenovich et al. (2016) that also incorporated the usage of weights. This algorithm depicts the general scheme of a diagnostic model when combined with a set of features. Its execution steps are defined as followed (Zenovich et al., 2016):

1. Apply the characteristic features and if at least one of the features is satisfied then the mammary gland is classified as high risk and the algorithm terminates. Otherwise, continue with the proceeding steps.

2. For each mammary gland compute the value S as such:

$$S = \sum_{i=1}^{n} k_i c_i, \tag{3.21}$$

where k are the weight coefficients, c the combined information, n the total number of mammary glands and i the ith mammary gland.

3. If the value S obtained is greater than a specified threshold S_0, then the mammary gland is classified as high risk, otherwise as low risk.

The values k and S_0 have been selected as such so that they maximise the score value Δ on the training set.

The first variation of the algorithm (denoted as *algorithm 1*) specified by Zenovich et al. (2016) used the default genetic algorithm as described previously to evaluate the classification results on the dataset. As before, an individual of the ith mammary gland is a set consisting of the weight coefficients k_i and a threshold value of S_0. These parameters were set by the authors as such to maximise the Δ value, which will be used as the fitness function f ($f = \Delta$). The probability of the ith individual to become a parent depends on the normalised f_i against all the n individuals in the generation:

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{3.22}$$

After the pair of parents have been selected, at the crossover stage each child obtained its genes from one or the other parent. In addition, the authors set a probability of 3% for each gene position to mutate to any of the available genes in the set.

A variation of the *algorithm 1* was also proposed (denoted as *algorithm 2*) by Zenovich et al. (2016) to increase the genetic diversity of each of the child individuals. They have used two different methods of crossover, increased the probability of mutations that can also impact multiple genes at once and the number of times a pair of individuals can take part as parents in the crossover was limited. In addition, the evolution of the initial population was executed several times in parallel independent from each other with an occasional exchange of individuals between them.

Another algorithm was explored by Zenovich et al. (2016) in which it consists of two sequential algorithms, the first is the algorithm with weights presented earlier and then followed up by an algorithm without weights proposed by Anisimova et al. (2011). For the new algorithm to outperform the previous ones they state that it must correctly classify those patients who would of been otherwise misclassified by the algorithm without weights. To achieve such an improvement was accomplished by changing the fitness function used to score each individual. The fitness function was changed to the following:

$$f = \frac{\Delta + C'}{2}, \tag{3.23}$$

where C' is the sensitivity value for the algorithm with weights on the mammary glands classified as high risk, in which they were misclassified by the algorithm without weights. The statistical measure can be swapped with specificity or accuracy instead

of specificity in the fitness function. This algorithm will be denoted as *algorithm 3sen* that uses sensitivity, *algorithm 3spe* for specificity and *algorithm 3acc* for accuracy.

The results for all the different variations of the algorithms are summarised in *table 3.1* obtained by Zenovich et al. (2016). The main metric used to evaluate the algorithms was the geometric mean (Δ) of the sensitivity (P) and specificity (C) on the test set. From the results, the best performing algorithm was *algorithm 3sen* with a Δ value of 68.3%, with both sensitivity and specificity achieving high results (74.3% and 62.7%) respectively. The algorithm without weights, *algorithm 1* and *algorithm 2* achieved the lowest results with a value of Δ around 50%. Noticeably, all the algorithms, with the exception of *algorithm 3sen*, favoured classifying one class over the other despite not having such a trend on the training set. This indicates that they are not able to effectively generalise the training samples to the unseen test ones.

Algorithm	**Training Set**			**Test Set**		
	P	C	Δ	P	C	Δ
Without weights	93.2%	68.9%	80.1%	39.2%	62.7%	49.6%
Algorithm 1	93.2%	73.8%	83%	35.1%	68%	48.9%
Algorithm 2	93.2%	73.5%	82.8%	36.5%	68%	49.8%
Algorithm 3sen	98.6%	68.9%	82.4%	74.3%	62.7%	68.3%
Algorithm 3spe	93.2%	96.4%	94.8%	39.2%	93%	60.4%
Algorithm 3acc	93.3%	96.3%	94.8%	39.2%	93.7%	60.6%

Table 3.1: The results, as obtained by Zenovich et al. (2016), of the different variations of algorithms against the statistical metrics specificity (P), sensitivity (C) and the geometric mean of both of them (Δ). They were all evaluated on the training and test set.

3.4.3 Classification using neural networks

Different variations and architectures of neural networks and learning algorithms were also used to evaluate the effectiveness in classifying if a patient is at low or high risk of

cancer (Zenovich et al., 2016). Specifically, all the networks had for their output layer two neurons and as such the output class was transformed to binary categorical values. The first set of variations used were simple multilayer fully connected neural networks (NN). The NNs were tested with one and 10 hidden layers. The second set of networks used were cascade correlation neural networks (CCNN), as described in *section 2.3*, so the number of hidden layers was built dynamically. The pool size for the candidate hidden layers was set to 30 with each initialised at different weights. The one with the lowest loss value was selected and as long as it reduced the overall error of the network it was added (Zenovich et al., 2016). Also, three different learning algorithms were used to train the network which were backpropagation (Hecht-Nielsen, 1992), *Hooke-Jeeves* pattern search algorithm (Hooke and Jeeves, 1961) and *simulated annealing* (Laarhoven and Aarts, 1987).

Additionally, these configurations were tested on two different input vectors (Zenovich et al., 2016). For both cases, they included as one of the vector values a value that represented if the participant experienced pain or not in each of the mammary glands. The reset of the values of the vector for the first case compromises of the 44 temperature features, as described in 3.1. On the other hand, the second case contained values from the modeling functions of the temperature features from a minimised set, as described in *section 3.4.2* (Glazunov et al., 2015b; Losev et al., 2011).

Input Vector	**Learning Algorithm**	**Architecture (layers)**	**Test Set**		
			Accuracy	**Specificity**	**Sensitivity**
Temp	Back-propagation	Fully Connected (1)	56 %	100 %	32 %
Functions	Back-propagation	Fully Connected (1)	64 %	62 %	67 %
Functions	Back-propagation	Fully Connected (10)	56 %	60 %	52 %
Temp	Hooke-Jeeves	Fully Connected (1)	48 %	47 %	49 %
Functions	Hooke-Jeeves	Fully Connected (1)	72 %	70 %	74 %
Temp	Simulated Annealing	Fully Connected (1)	69.5 %	70 %	69 %

Input Vector	Learning Algorithm	Architecture (layers)	Test Set		
			Accuracy	Specificity	Sensitivity
Temp	Simulated Annealing	Fully Connected (10)	61 %	70 %	52 %
Temp	Simulated Annealing	Cascade Correlation	81 %	79 %	83 %
Functions	Simulated Annealing	Fully Connected (1)	73 %	72 %	74 %

Table 3.2: The results of accuracy, specificity and sensitivity on the test set on various neural network models, as found by Zenovich et al. (2016). The models varied with the input to be either the temperature values or minimised set of functions, different learning algorithms, either backpropagation, Hooke-Jeeves search algorithm or simulated annealing and various architectures being fully connected with either one or 10 hidden layers and cascade correlation neural network.

The results from the test set from the paper (Zenovich et al., 2016) are summarised in *Table 3.2* for the various configurations that were evaluated. The best performing model that had equally good results for specificity and sensitivity, hence equally good accuracy, was the cascade correlation neural network with simulated annealing that had temperature values as input. The accuracy obtained was 81%, which was the highest from all models, specificity with 79%, being the second highest and sensitivity with 83% being again the highest rating. From the setups, the fully connected with one hidden layer and backpropagation that had temperature values as input obtained the highest specificity with a value of 100%. However, it had the lowest sensitivity of 32%, pushing the accuracy to only 56%. With these percentages it shows that this model had been overfitted to the negative class, making it incapable of effectively classifying the positive class i.e. predicting sick patients as having low risk of cancer.

Chapter 4

Data Preprocessing and Feature Evaluations

4.1 Dataset Preparation

4.1.1 Vector

The dataset is transformed so that the appropriate individual mammary glands are paired, as described in 3.1, creating a vector of samples. The vector will contain the temperature values for all the points on the gland at both skin and depth level. Additionally, the class label will be kept to be used as the output prediction. All other features in the dataset will not be used at this stage.

Before pairing, each gland has a total of 24 temperature values as feature input of which one is the axillary point, 10 points on the gland and two reference points for both on the skin surface and at a depth of 5cm from the skin. After pairing, there will be 44 temperature values having a distinction between left and right mammary glands. The reference points are the same for both of the glands in the mammary gland and so can be concatenated. The output class consists of two classes labeled as 0 for patients with low risk for cancer and 1 for patient with high risk. If both of the glands are marked as 0 or 1, then when paired they will maintain prediction class of 0 or 1 respectively. However, if one of the glands is marked as 0 and the other as 1, then when paired it will be marked as 1.

Image Transformation Calculations

$L9$												$R9$
	$L8$	$\frac{L8+L1}{2}$	$L1$	$\frac{L1+L2}{2}$	$L2$		$R2$	$\frac{R2+R1}{2}$	$R1$	$\frac{R1+R8}{2}$	$R8$	
	$\frac{L7+L8}{2}$	$\frac{L0+L1+L7+L8}{4}$	$\frac{L0+L1}{2}$	$\frac{L0+L1+L2+L3}{4}$	$\frac{L2+L3}{2}$		$\frac{R3+R2}{2}$	$\frac{R0+R1+R2+R3}{4}$	$\frac{R0+R1}{2}$	$\frac{R0+R1+R7+R8}{4}$	$\frac{R8+R7}{2}$	
	$L7$	$\frac{L0+L7}{2}$	$L0$	$\frac{L0+L3}{2}$	$L3$		$R3$	$\frac{R0+R3}{2}$	$R0$	$\frac{R0+R4}{2}$	$R7$	
	$\frac{L6+L7}{2}$	$\frac{L0+L5+L6+L7}{4}$	$\frac{L0+L5}{2}$	$\frac{L0+L3+L4+L5}{4}$	$\frac{L3+L4}{2}$		$\frac{R4+R3}{2}$	$\frac{R0+R3+R4+R5}{4}$	$\frac{R0+R5}{2}$	$\frac{R0+R5+R6+R7}{4}$	$\frac{R7+R6}{2}$	
	$L6$	$\frac{L5+L6}{2}$	$L5$	$\frac{L4+L5}{2}$	$L4$		$R4$	$\frac{R5+R4}{2}$	$R5$	$\frac{R6+R5}{2}$	$R6$	

Figure 4.1: A summary of the transformation algorithm from a vector to a 2D array, where L is the left gland and R is the right gland. Any cells left blank have a value of 0. This is the same for values captured on the skin and at a depth from the skin.

4.1.2 Image

Each of the data samples can be transformed to an image representing the relational position between the captured temperature points. The transformation is conducted on the resulting vectorisation of the samples as described previously. A simple approach is to place appropriately the temperature values in an array as such they represent the same image as that seen in *Figure 3.1*. But with this format it does not represent adequately the overlap between the captured temperature points as depicted in the figure.

The proposed transformation will be an image of size $13x6$ that accommodates all points from both of the mammary glands of a pair without the reference points, which will be removed after they have been used to normalise the data. Each of the points of the mammary gland will have a one cell space between them. Gaps that are created between two original points will be filled by taking their average value. Furthermore, the inner gaps that are not neighbouring to any of the original points will obtain the average of the four closest points from the initial points. A gap between the glands will be maintained so they are distinguishable. The top left and right of the image will be used to store the axillary values of the glands. Because for each sampling point there are two levels of temperature values, the image will have two channels with the first representing the temperatures at a depth and the second temperatures captured at the skin surface. The transformation algorithm is summarised in *Figure 4.1*, which is the same for measurements for the skin and at a depth. An example of a transformation for one of the channels is shown in *Figure 4.2*.

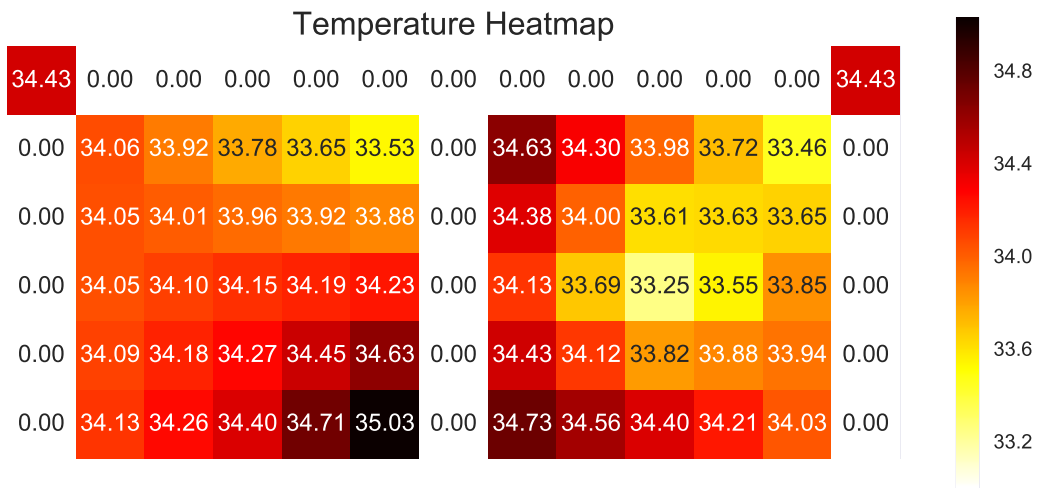

Figure 4.2: A heatmap of the results of a transformation of a single data sample vector. The temperatures captured are at a depth from the skin surface for a patient with high risk of having cancerous tumor.

4.2 Data Normalisation Evaluation

As described in *section 3.3*, a data normlisation algorithm was proposed to be used on this specific dataset. Bochkarev et al. (2015) showed that normalising the dataset allowed improvements on accuracy, specificity and sensitivity on the test set when used with a linear regression algorithm. They evaluated normalising using both $T1$ and $T2$ as the main reference points in which both achieved higher accuracy than without normalisation. However, when $T2$ was evaluated it showed a slightly better improvement compared to $T1$, with the total improvement from the base results to be around 4%.

The dataset was again evaluated but using this time a *random forest* (Breiman, 2001) classifier from the *sklearn* (Pedregosa et al., 2011) library, which does not require significant preprocessing to obtain better results because the decision point to split a tree remains more or less the same. Additionally, a random forest is able to compensate the class imbalance by adding more weight to the underrepresented one. The dataset was class balanced split into training, validation and test with a total percentage of 60% (low risk: 46 and high risk:171), 20% (low risk: 15 and high risk:57) and 20% (low risk: 16 and high risk:58) respectively. The training set was used to fit the weights of the algorithm, the validation set to optimise the hyperparameters and the test set to find the unbiased performance of the model on unknown data samples. When normalising the dataset, the reference point $T2$ was used, as for it outperformed

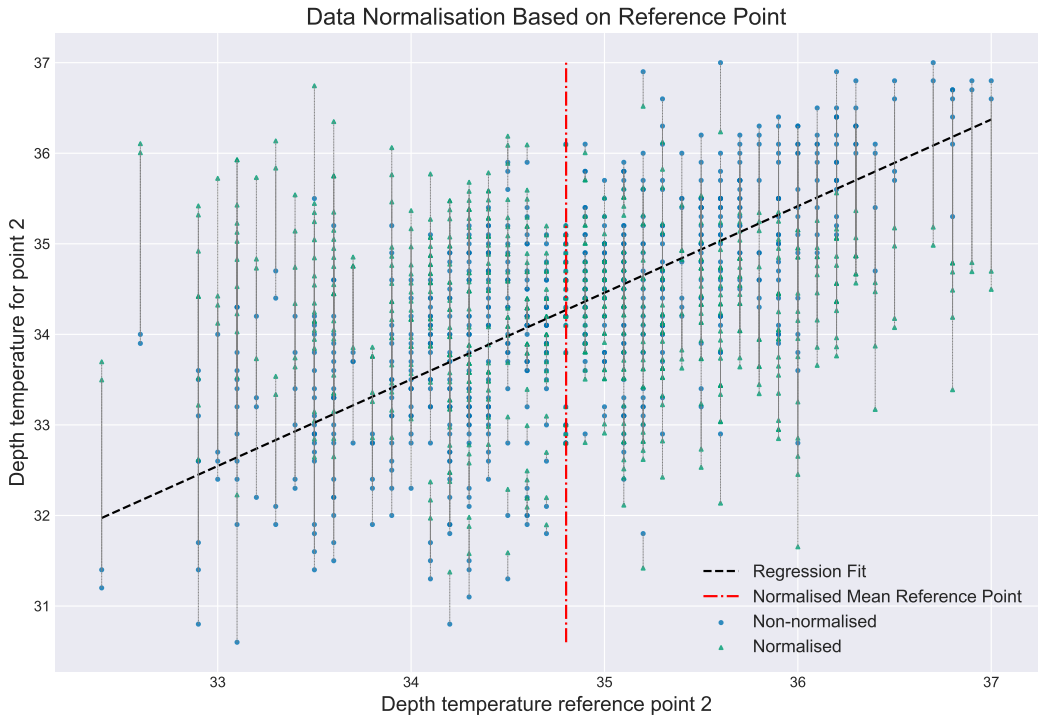

Figure 4.3: Example of fitting a linear regression line to the dataset and then normalising for a specific data point in regards to the reference point $T2$. The blue dots mark the temperature values before transformation and the red triangle (with the interconnecting dotted lines) the the new values after transforming.

$T1$. Then both the reference points $T1$ and $T2$ for skin and at a depth were removed from the vector, leaving 40 features. An example of the normalisation processing for a single temperature point can be seen in *Figure 4.3*.

The random forest classifier was optimised using the *hyperopt* (Bergstra et al., 2015) optimisation library with *tree of Parzen* (Bergstra et al., 2011) estimators as the optimiser, up to a maximum of 50 optimisation evaluations and weighted G-mean loss as the loss function. The optimal parameters found for the random forest without doing normalisation preprocessing were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False, max_depth = None, max_features = 0.1674, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_weight_fraction_leaf = 0.0 and warm_start = False. The optimal parameters for the random forest when using normalisation are: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False, max_depth = None, max_features = 0.33, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_weight_

fraction_leaf = 0.0 and warm_start = False.

Preprocessing	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
No normalisation	Train	0.9862	0.0247	0.9942	0.9565
	Validation	0.7639	0.3846	0.8597	0.4
	Test	0.7568	0.3971	0.8621	0.375
Normalisation	Train	0.9171	0.0686	0.9064	0.9565
	Validation	0.7222	0.3791	0.7895	0.4667
	Test	0.7703	0.3347	0.8448	0.5

Table 4.1: Results using a random forest on the training, validation and test set when excluding and including normalisation preprocessing on the data.

The results from the random forest classifiers when using or not normalisation preprocessing are summarised in *Table 4.1*. Based on the weighted G-mean loss value of the test set, with normalisation has better performance (0.3347) than without (0.3971). This is also reflected on the accuracy obtained which reached 77.03% when normalisation was used and 75.68% without. The notable improvement with normalisation is that it is able to classify more correctly the low risk classes with a specificity value of 0.5 against 0.375 without. However, this comes at a small cost in misclassifying samples classified as high risk, but the difference in sensitivity is less than 0.02.

The difference on the results is that without normalisation it was able to classify correctly an additional high risk sample, but at the cost of two low risk samples. From the results, one can notice that without normalisation and for all three datasets the random forest classifier favours much more predicting high risk classes rather than a balance between the two, which is not desirable.

Hence, we opt to prefer using normalisation because it has also shown improvements on linear algorithms (Bochkarev et al., 2015), in conjunction with improvements here on a non-linear one and reduces the total number of features by four to improve speed of training. Also, with normalistion the classification algorithm is less prone to overfitting and is more capable of generalising to unseen samples (normalistion has worse results on the training set, but better on the test set).

4.3 Extracted Features Evaluation

There are many features that can be extracted from this specific problem to assist in identifying if a patient is at low or high risk of having cancer in the mammary glands, as some of these features have been presented previously in *section 3.2*. However, not all will contribute the same to classifying correctly a patient and most will become redundant. So, some of these features were compared to determine how effective they are. The features that were evaluated from those presented in *section 3.2* are 3.1 (abbreviated as *dif*), 3.4 (*rmsdif*), 3.10 (*ndif*), 3.2 (*disp*), 3.11 (*comp*) and 3.3 (*reg*).

The features were evaluated using the random forest classifier with its hyperparameters optimised using the hyperopt library for each feature set. The dataset was class balanced split into training, validation and test with a total percentage of 60% (low risk: 46 and high risk:171), 20% (low risk: 15 and high risk:57) and 20% (low risk: 16 and high risk:58) respectively. The training set was used to fit the weights of the algorithm, the validation set to optimise the hyperparameters and the test set to find the unbiased performance of the model on unknown data samples.

The otimiser used for the hyperopt was tree of Parzen. Additional settings were set for the maximum number of optimisation intervals of 50 and used the weighted G-mean loss function as its loss function to minimise on. The optimal parameters found for the dif features were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = True, max_depth = 4, max_features = 'log_2', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For the rmsdif features they were: n_estimator = 400, class_weight = 'balanced', criterion = 'entropy', bootstrap = True, max_depth = None, max_features = 'log_2', max_leaf_nodes = None, min_impurity_ decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For the ndif features they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = True, max_depth = None, max_features = '*sqrt*', max_leaf_nodes = None, min_impurity_ decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For the disp features they were: n_estimator = 400, class_weight = 'balanced', criterion = 'entropy', bootstrap = True, max_depth = None, max_features = 'log_2', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_ samples_split = 2, min_weight_fraction_leaf = 0.0 and warm_start = False. For the

comp features they were: n_estimator = 400, class_weight = 'balanced', criterion = 'entropy', bootstrap = False, max_depth = None, max_features = 0.36, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 4, min_samples_split = 2, min_weight_fraction_leaf = 0.0 and warm_start = False. Lastly, for the reg features they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False, max_depth = None, max_features = 0.33, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_weight_fraction_leaf = 0.0 and warm_start = False.

Features	**Dataset**	**Accuracy**	**G-loss**	**Sensitivity**	**Specificity**
dif	Train	0.8249	0.1208	0.7836	0.9783
	Validation	0.7778	0.2907	0.8246	0.6
	Test	0.6891	0.3345	0.7069	0.625
rmsdif	Train	0.9816	0.0117	0.9766	1.0
	Validation	0.8472	0.2776	0.9298	0.5333
	Test	0.7702	0.4186	0.8966	0.3125
ndif	Train	0.6867	0.2791	0.6608	0.7826
	Validation	0.6667	0.3333	0.6667	0.6667
	Test	0.5405	0.4516	0.5345	0.5625
disp	Train	1.0	0.0	1.0	1.0
	Validation	0.8333	0.2857	0.9123	0.5333
	Test	0.6892	0.4942	0.8103	0.25
comp	Train	0.9217	0.0579	0.9064	0.9783
	Validation	0.8194	0.2129	0.8421	0.7333
	Test	0.7432	0.3772	0.8276	0.4375
reg	Train	0.7143	0.2767	0.7076	0.7391
	Validation	0.6528	0.3928	0.6842	0.5333
	Test	0.6081	0.5105	0.6897	0.3125

Table 4.2: Summary of the results of a random forest classifier with various extracted features passed as the algorithm's input.

The results of the different input feature sets are summarised in *Table 4.2*. The lowest weighted G-mean loss on the test set is achieved on the dif feature set with a value of 0.3345. However, its accuracy is the fourth highest with a percentage of 68.91% and the highest being 77.02% due to imbalance in the dataset. The dif feature

is the most informative allowing the random forest classifier to obtain a good balance between the two classes, while being able to have the highest accuracy on classifying low risk samples compared to the other features. The classifier on the disp feature set favours the most in classifying high risk samples against low risk ones, with a specificity of 0.8103 and sensitivity of 0.25 on the test set.

4.4 Feature Importance Ranking

There are a lot of features that can be extracted from the dataset for this specific problem, as also seen in *section 3.2*. With each mathematical function used to extract data will result to an increase in dimensionality of the input vector. This will have as a consequence a decreased performance, increased computational complexity and more parameters (weights) to store for the various models (Chandrashekar and Sahin, 2014). Moreover, not all of the information extracted will contribute equally to correctly classifying a patient. Additionally, most features will most likely express the same information and can be abbreviated.

A random forest classifier was used to determine the importance of each of the features based on how many times a feature is located at the leaf of all the trees in the forest (Breiman, 2001). A total of 132 features were passed to the random forest with the purpose to classify as correctly as possible the samples. From the extracted features seen previously, there are a total of 88 additional values over and above the 44 initial features. The dataset was once again class balanced split into training, validation and test with a total percentage of 60% (low risk: 46 and high risk:171), 20% (low risk: 15 and high risk:57) and 20% (low risk: 16 and high risk:58) respectively. The training set was used to fit the weights of the algorithm, the validation set to optimise the hyperparameters and the test set to find the unbiased performance of the model on unknown data samples. The optimisation through the hyperot library was achieved with setting as the optimiser the tree of Parzen algorithm, max optimisation evaluations of 50 and G-weighted mean loss function to minimise on.

The feature importance rankings for all the features is summarised in *Figure 4.4*, showing the score for each with the expected standard deviation range for each individual tree in the forest. The first 20 features have a large impact on classifying correctly the samples, with the scoring dropping dramatically with each next feature. The score reaches 0, standard deviation 0 for the last nine features, which are: left gland skin at point 6, right gland depth at point 1, left gland skin comp at point 3, skin dif at point

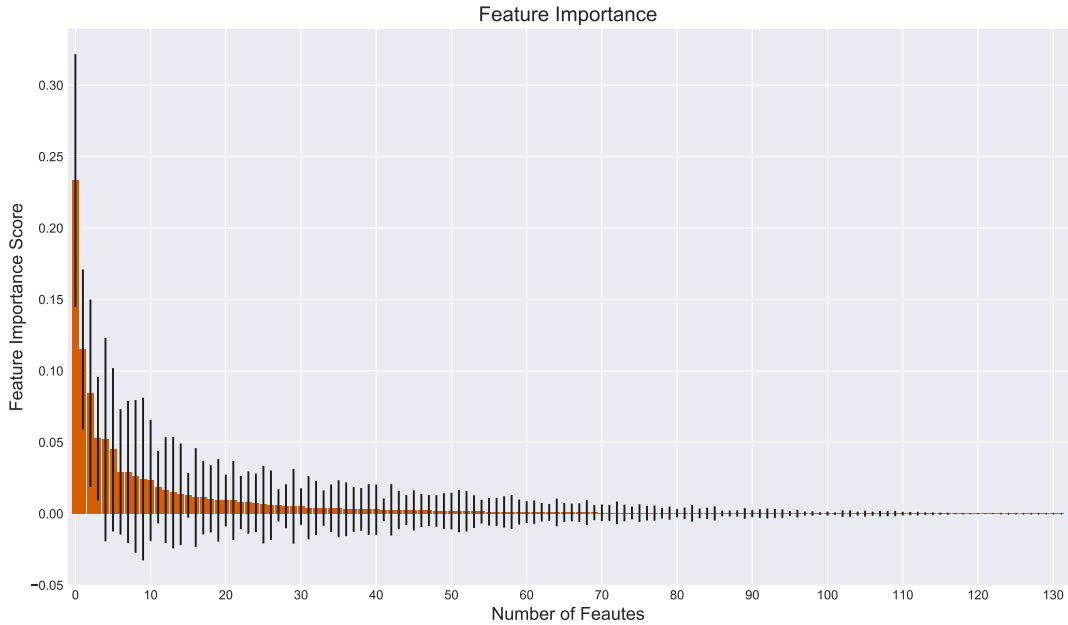

Figure 4.4: The feature importance as ranked by the random forest algorithm. The higher the score the more frequently it has been found at a leaf of a tree. For each of the features its standard deviation is shown with a black vertical line.

5, 4 and 3, skin reference point 2, right gland depth reg at point 1 and left gland depth comp at point 5. The five most prominent features are: depth rmsdif (score: 0.2333 and standard deviation: 0.0885), left gland depth reg at point 6 (score: 0.1151 and standard deviation: 0.0559), right gland depth axillary point (score: 0.0843 and standard deviation: 0.0653), depth reference point 1 (score: 0.0526 and standard deviation: 0.0433) and depth dif at point 7 (score: 0.0519 and standard deviation: 0.0712).

While the top 20 scoring features have a larger impact on classifying than the rest of the features, they still obtain quite low values, with the exemption of the first three, (ranging from 0.0526 to 0.0094). In addition, they also have high standard deviation indicating that only a few of the estimators were able to effectively use those features to split the nodes of the trees. This indicates that no single feature can be used reliably and that an increase in temperature, and hence indication of a tumor, can be found at any of the measured points. in addition, the results show that a lot of the features are either irrelevant because of the low score or redundant because of the high standard deviation in which a tree can interchange features with the same effect more freely. However, observing the full list of scores for each of the features, extracted features have an overall much higher positive impact on classification than the raw temperature features. This shows that processing and extracting features from the dataset can

improve the accuracy of the classifier by adding additional information and linking specific correlation between points.

4.5 Oversampling Comparison

The dataset is currently imbalanced in favour of the high risk class, with the total samples for each is 77 for low risk and 286 for high risk. There are algorithms that can set weights to the samples so that the underrepresented class can be considered equally important when fitting the weights. However, some algorithms do not have such capability and so will favour classifying the dominant class. To overcome this issue when evaluating different algorithms, the data from the underrepresented class was sampled from until there are in total 286.

A random forest algorithm was used having also set a weight balancing scheme to compensate for the imbalance. While the algorithm is capable of handling the class imbalance, we want to evaluate which of the oversampling techniques on the training set will not cause a reduction in performance. The oversampling techinques that were evaluated are random re-sampling with no alteration to data, *Synthetic Minority Over-Sampling Technique* (SMOTE) with regular, borderline 1, borderline 2 and Support vector Machine (SVM) variations (Chawla et al., 2002; Han et al., 2005) and *Adaptive Synthetic* (ADASYN) (He et al., 2008). The dataset was balanced split into training (60% - low risk: 46 and high risk:171), validation (20% - low risk: 15 and high risk:57) and test (20% - low risk: 16 and high risk:58). For each of the oversampling techniques the hyperparameters of the random forest were optimised using the hyperopt library with the tree of Parzen optimiser, 50 maximum optimisation intervals and weighted G-mean loss as the loss function to minimise on.

The optimised hyperparameters found without using any oversampling were: n_ estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False, max_depth = None, max_features = 0.17, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_ fraction_leaf = 0.0 and warm_start = False. For random sampling they were: n_estimator = 400, class_weight = 'balanced', criterion = 'entropy', bootstrap = False, max_depth = None, max_features = '*sqrt*', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For SMOTE regular they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False,

max_depth = None, max_features = '*sqrt*', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For SMOTE borderline 1 they were: n_estimator = 400, class_weight = 'balanced', criterion = 'entropy', bootstrap = False, max_depth = None, max_features = '*sqrt*', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 9, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For SMOTE borderline 2 they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = False, max_depth = None, max_features = 0.28, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. For SMOTE SVM they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = True, max_depth = None, max_features = 0.47, max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 36, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False. Lastly, for ADASYN they were: n_estimator = 400, class_weight = 'balanced', criterion = 'gini', bootstrap = True, max_depth = None, max_features = '*sqrt*', max_leaf_nodes = None, min_impurity_decrease = 0.0, min_impurity_split = None, min_samples_leaf = 1, min_samples_split = 2, min_ weight_fraction_leaf = 0.0 and warm_start = False.

Oversampling	**Dataset**	**Accuracy**	**G-Loss**	**Sensitivity**	**Specificity**
No oversampling	Train	0.9724	0.0577	0.9942	0.8913
	Validation	0.75	0.3925	0.8421	0.4
	Test	0.7702	0.3894	0.8793	0.375
Random	Train	0.7515	0.2485	0.8187	0.6842
	Validation	0.6944	0.3413	0.7193	0.6
	Test	0.6622	0.3994	0.7069	0.5
SMOTE regular	Train	0.9123	0.0877	0.8596	0.9649
	Validation	0.6944	0.3677	0.7368	0.5333
	Test	0.6622	0.3749	0.6897	0.5625
SMOTE borderline1	Train	0.9181	0.0819	0.9532	0.883
	Validation	0.8194	0.3533	0.9298	0.4
	Test	0.7838	0.3268	0.8621	0.5
SMOTE borderline2	Train	0.8655	0.1345	0.8947	0.8363
	Validation	0.7639	0.4159	0.8772	0.3333

Table 4.3 continued from previous page

Oversampling	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
	Test	0.7568	0.3693	0.8448	0.4375
SMOTE SVM	Train	1.0	0.0	1.0	1.0
	Validation	0.7639	0.3846	0.8448	0.4
	Test	0.7297	0.4126	0.8276	0.375
ADASYN	Train	0.9591	0.0409	0.9181	1.0
	Validation	0.7083	0.3594	0.7544	0.5333
	Test	0.7027	0.401	0.7759	0.4375

Table 4.3: Summary of the results of a random forest classifier when using oversampling on the least represented class (low risk) in the dataset so it becomes balanced.

Random re-sampling the underrepresented class (low risk) resulted in to slightly worse overall results in regards to the weighted G-mean loss value, which it obtained 0.3994 compared to 0.3894 from no oversampling. While it helped with classifying the low risk class, it had a much worse effect on classifying the high risk class. Using random forest with class balance through allocating appropriate weights to each (no oversampling) allows the samples to be considered equally important. However, with random sampling the underrepresented samples will have unequal distributed weight, giving more importance to ones that happen to be sampled more than once.

The oversampling technique that reaches close to the results that of without any oversampling but also gives slight improvement is SMOTE with borderline1 variation. The G-mean loss achieved is 0.3533 and an accuracy of 78.38%, which are both improvement to no oversampling that achieved 0.3894 and 77.02% respectively. The technique helped correctly identify two additional low risk samples (0.5 specificity) compared to no sampling (0.375 specificity), at the cost of misclassifying a single sample from the high risk class (0.8621 compared to 0.8793 sensitivity). This was achieved through the addition of statistical noise to the sampled data, allowing it to generalise better to unseen data.

The overall accuracy in predicting correctly the classes on the test set is only surpassed by SMOTE with borderline1 variation in relation to no oversampling. SMOTE borderline2 and SMOTE SVM variations, achieving 75.68% and 76.39% respectively accuracy, they come close to the accuracy with no oversampling, 77.02%, trading off better classification for the low risk class against the high risk. The rest of the methods

sacrifice too much in correctly classifying the high risk class to gain some improvement for the low risk one.

4.6 Image Generation

Seeing from the previous section (4.5), by generating additional samples that have added noise to them allow an algorithm to better generalise (prevent overfitting) to unknown data. While the technique shown was used to balance the classes, it can be extended to generate additional samples for both and increase the total data size that can be used for training. However, while they can sample from a generate distribution for each of the features, their combination might lead to inaccurate data because domain knowledge can not be considered by the generator, as mentioned in the review paper S and Thilak Chaminda (2017). They state that generating biomedical data is not as easily achievable as with other fields and more complex models, like Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), are required to generate more meaningful samples.

Another, much simpler, approach is to do standard transformations such as rotations and mirror flipping on the images created using the feature vectors, as described in *section 4.1.2*. Such an approach has been used by Vasconcelos and Vasconcelos (2017), while also including colour augmentations over and above image transformations/distortions. Consequently, they managed to improve the classification algorithm for detecting melanoma in which their data set was also quite small and unbalanced.

Taking the transformed vector of features to images as described in *section 4.1.2*, it is possible to generate additional samples through transformations. The proposed algorithm is to take the outer points of the glands (points from 1-8 as shown in *Figure 3.1*) and rotate them by one position at a time, until each point has completed a full circle. The points of the left mammary gland will be rotated clockwise and the points of the right anticlockwise simultaneously. Also, a further extension is to mirror flip all the initial and generated samples in such a way that the left mammary gland and axillary point becomes the right one and vice versa.

While this method is much simpler than that reviewed by S and Thilak Chaminda (2017), it has however an upper limit of how many samples can be generated. From each sample, it can generate a total of seven additional samples just by doing rotations on the points. This number can be doubled with plus one for the original image when also performing mirroring flip. So, the upper limit is 15 generated samples from each

one passed.

With such transformations, it is necessary to be cautious that by doing such image generations the classification algorithms will not be able to identify positional variations of cancer in the mammary glands as reliably as before. In a previous research by Veltmaat et al. (2013) they have shown that cancer is more prominent in specific positions of the mammary gland and its appearance is unequal between the two glands. They explain that this is due to most likely structural asymmetries of the two glands.

Chapter 5

Non-Neural Network Model Evaluations

We want to evaluate how non-neural network classification algorithms can compare against neural network ones. Also, it is important to identify the effectiveness of neural networks in extracting features from the dataset compared to manual mathematical functions, which can be directly passed to non-neural network models. Various preprocessing steps can be taken that will result to different possible sets of input features. Specifically, such sets were used to explore how adding extracted features to the original set, selecting optimal features and reducing their dimensionality can affect the results.

Non-neural network models are still a vital alternative to neural network ones (Wilkins et al., 1996; Lim et al., 2000). Depending on the complexity of the problem and size of the dataset, non-neural networks models are capable of training their weights with much less time than deep neural networks and without compromising results or even being able to outperform. Also, this leads to the demand of lesser computational resources allowing them to be trained on personal machines. Lastly, they require less hyperparameter tuning and setup time and do not require an architecture to be designed specifically for the problem, making them production-ready much sooner.

5.1 Feature Selection

There are many features that can be extracted from the presented dataset, with some of them presented in *section 3.2* reaching to 132 features, including the original ones. From the analysis of the features in *section 4.3* when each subset is used individually it

shows that they carry highly predictive information and in turn the extraction process suffices.

Extracting features from the dataset can add additional information to the model enabling it to achieve better outcome. However, the results from *section 4.4* show that when combining all the features a lot of the information was overlapped making them irrelevant or redundant. In addition, the increased dimensionality of the feature set makes the training process more difficult, both in time and task complexity, and removes focus from the relevant information (Blum and Langley, 1997).

It has been shown that using feature selection algorithms allow the predictive model to either maintain the same accuracy or even achieve better results using only a smaller subset of the features (Hall, 1999). This is achieved by removing the irrelevant features that do not contribute to the task, redundant features that their information can be directly obtained from the original or other extracted features and features that are highly noisy, in which the do not contribute to generalisation or prediction. However, feature selection presents risks if a highly informative feature is removed when it covers a unique subspace of the predictive area, thus reducing the overall performance of the algorithm (Hall, 1999). Despite of that, the usage of a feature selection algorithm becomes significant enough to explore for potential improvements.

The feature selection was achieved through using a *Recursive Feature Elimination and Cross-Validated* (RFECV) selection based on feature ranking (Guyon et al., 2002). The algorithm starts off with the full set of features and iteratively removes the least important feature until only one feature remains. To identify which to remove the feature set was passed to a linear prediction model and the feature with the lowest absolute value is selected. Also, at each step the cross validation score was calculated, which was used to determine the number of features that resulted to the highest validation score.

The prediction algorithm, for this case a classification algorithm, incorporated to rank the features was a *Support Vector Machine* (SVM) (Cortes and Vapnik, 1995; Burges, 1998) classification model, as initially proposed by Guyon et al. (2002), with a linear kernel. The SVM's implementation was obtained through the sklearn library and its optimal parameters were set through the usage of a grid search. The resulted parameters were: C = 0.75, penalty = 'l2', dual = False, tol = 1e-6. SVM was also selected because it can handle the class imbalance by allocating appropriate importance weights through the parameter: class_weight = 'balanced'.

The features selected were based on the highest weight G-mean score obtained

from five cross validations. With cross validation, the dataset was split into five equal folds with four taking part in training and the last one for validation. The algorithm is trained a total of five times for each step so that all folds will have been selected for validation purposes. The G-mean score was used to compensate for the imbalance of the classes by giving equal importance in predicting the low risk and high risk patients.

The data was normalised before being passed to the algorithm. As mentioned in *section 3.3* and *4.2*, there is no negative impact when used with non-linear algorithms and slight improvement when used with linear ones. So, four of the features, the reference points for skin and at a depth, can be removed from the set.

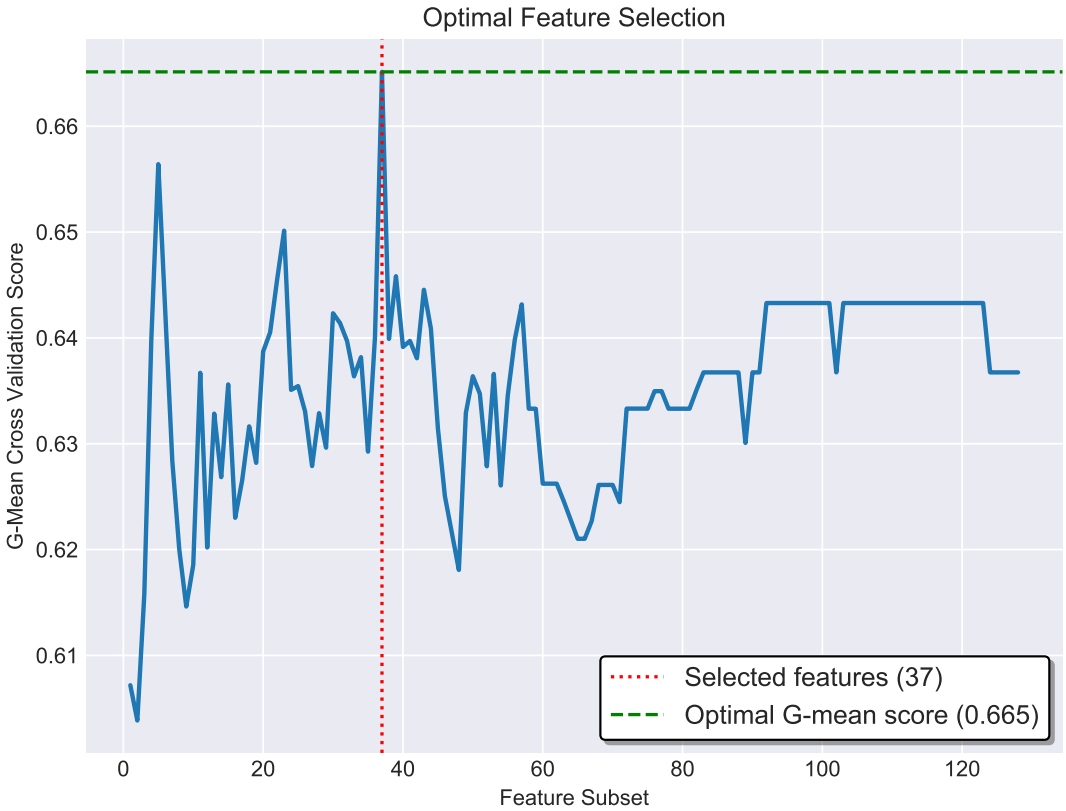

Figure 5.1: The weighted G-mean 5-fold cross-validation results for each feature subset. The optimal subset of features consists of a total of 37 features, achieved with a G-mean score average across the cross-validations of 0.665.

The optimal number of features determined by the algorithm is at 37 which was achieved with an average G-mean cross-validation score of 0.665. From 92 and up features maintained in the dataset, there was no significant change to the results, something which was expected from seeing the results in *section 4.4*, in which a lot of features did not contribute to the classification accuracy. For the rest of the number

of features that remained, there was a significant variation of the obtained weighted G-mean loss values. The second highest score was obtained with a mere five features, with a G-mean score of 0.657, which again confirms the findings from *section 4.4* that the first features have the highest contribution in correctly predicting samples. The results for each step can be seen in *Figure 5.1*.

5.2 Dimensionality Reduction

From the previous section (5.1), the number of features were reduced from 128 (with removed features from normalisation) to 77, which is still a considerable amount. One method to further reduce the dimensionality is to do a *Principal Component Analysis* (PCA) (Ian, 2014) on the set of features, which transforms them from n-dimensions to a new set of features which are m-dimension ($m < n$) depending on how linearly separable the features are. The usage of PCA is a common preprocessing technique used in classification problems that not only benefits from the reduced complexity of using a smaller set of features, but also it can improve the overall performance of various classification algorithms (Rodarmel and Shan, 2002; Wang and Paliwal, 2003).

The PCA algorithm was able to reduce the dimensionality of the dataset without losing significant information. However, determining what is considered as a significant loss of information in regards to the classification accuracy can not be achieved solely through the PCA algorithm, as for it does not take into account the output class of the set. To compensate for this a SVM classification algorithm with a linear kernel was paired to evaluate the resulting prediction capabilities of the set.

The implementations for the PCA and SVM were taken from the sklearn library. The parameters for both were determined using a grid search, which for the PCA were: whiten = True, svd_solver = 'auto', tol = 0.0, iterated_power = auto and for the SVM: C = 0.45, penalty = 'l1', dual = False, class_weight = 'balanced', tol = 1e-6. The number of features (dimensions) that were selected from the PCA algorithm were determined through the aforementioned pairing. An iterative processes was followed where the returned number of features from the PCA was incremented at each step from 1 up to 37 and the resulted feature set was piped to the SVM algorithm. For each iteration, the weighted G-mean 5-fold cross-validation score was calculated from the SVM predictor to determine which of the number of features from the PCA algorithm are optimal.

The summary of the results for each step is shown in *Figure 5.2*. The optimal number of dimensions reduced to was determined to be at 32 with a weighted G-mean

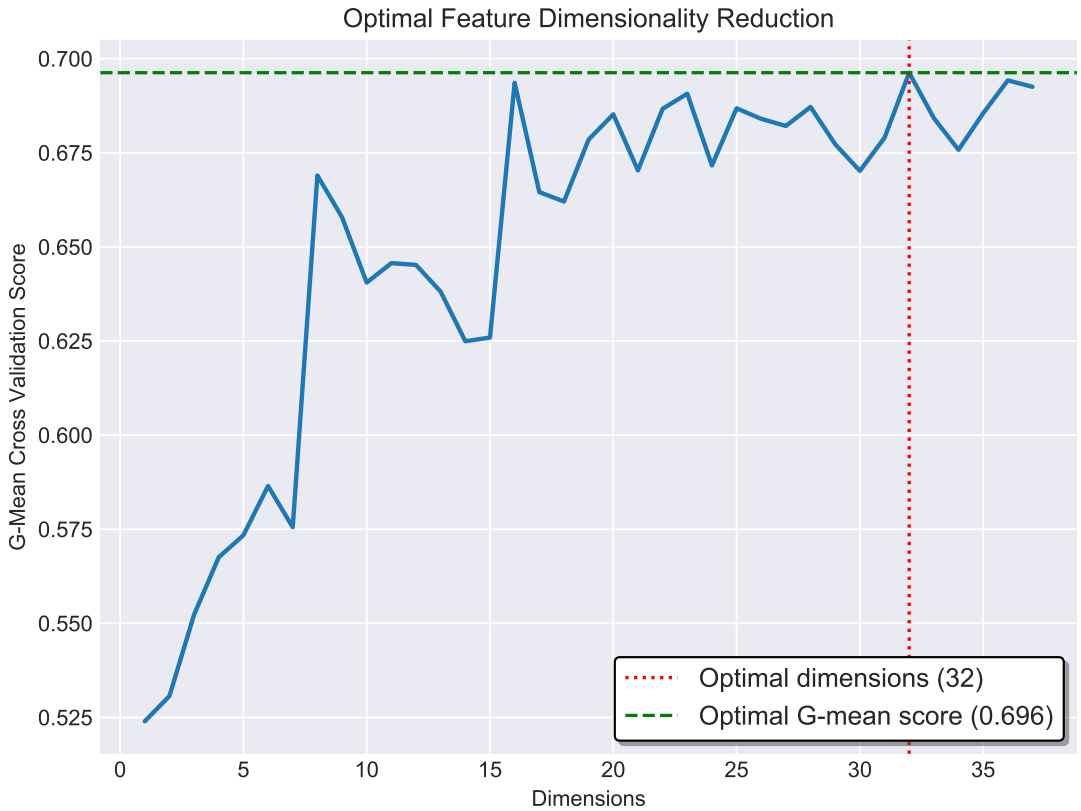

Figure 5.2: The weighted G-mean 5-fold cross-validation results for each number of dimensions. The highest G-mean score is achieved when the features are reduced to 37 dimensions with a value of 0.696.

cross-validation score of 0.696. Any other number of dimensions has a lower scoring, with a larger drop as the dimension become significantly smaller.

5.3 Results

The following algorithms presented were obtained through the sklearn library unless mentioned otherwise. Each of the models were evaluated against using only the original features, including the extracted features, using feature selection and then dimensionality reduction. The original features were normalised before proceeding with the preprocessing, as described in *section 3.3*, on the reference point $T2$ and then both reference points $T1$ and $T2$ were removed.

The dataset was balanced split into training (60% - low risk: 46 and high risk:171), validation (20% - low risk: 15 and high risk:57) and test (20% - low risk: 16 and high risk:58). In addition, the training set's samples were classed balanced using the

SMOTE algorithm with borderline 1 variation, as shown to also improve performance in *section 4.5*. The samples were only taken from the underrepresented class, the low risk. This resulted in having 171 samples for both low and high risk patients in the training set.

All of the algorithms used and for each input set they had their hyperparameters optimised through the hyperopt library against the validation set. The optimisation algorithm itself used the tree of Parzen algorithm to optimise the parameters. Also, the maximum optimisation iterations of the algorithm was set to 10, balancing between allowing the algorithm to obtain optimal results while preventing it from overfitting the parameters to the validation set and hindering its generalisation capabilities on the test set. Lastly, the loss function used, in which it determines the positive or negative change of a parameter, was the weighted G-mean loss function.

5.3.1 Random Forest Classifier

The optimal hyperparameters selected for each of the input feature sets for the random forest classifier were:

- Original set of features: 'bootstrap': False, 'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': None, 'max_features': 0.814611, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 33, 'warm_start': False

- Added extracted features: 'bootstrap': True, 'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 18, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 770, 'warm_start': False

- Feature selection: 'bootstrap': True, 'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2', 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 15, 'warm_start': False

- Dimensionality reduction: 'bootstrap': False, 'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': None, 'max_features': 0.391526, 'max_leaf_nodes':

None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 15, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 662, 'warm_start': False

Table 5.1: Random forest classification results on the four input sets

Input	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Original Data	Train	1.0	0.0	1.0	1.0
	Validation	0.7083	0.3328	0.7368	0.6
	Test	0.7027	0.4281	0.7931	0.375
Extracted Features	Train	0.7836	0.2164	0.8304	0.7368
	Validation	0.7361	0.4313	0.8421	0.3333
	Test	0.7027	0.35	0.7418	0.5625
Feature Selection	Train	0.7924	0.2076	0.8246	0.7602
	Validation	0.7083	0.3871	0.7719	0.4667
	Test	0.6757	0.4437	0.7586	0.375
Dimensionality Reduction	Train	0.6579	0.3421	0.7485	0.5673
	Validation	0.6528	0.3928	0.6842	0.5333
	Test	0.7027	0.375	0.7586	0.5

The best performance on the test set is observed with the added extracted features with a weighted G-mean loss value of 0.35. It achieves a decent sensitivity score of 0.7418 and a slight above random predictive capabilities for specificity with a value of 0.5625. Its overall accuracy on the imbalanced test set is at 70.27% which is the highest from all the input features in par with the original feature set and with dimensionality reduction. The results for all the feature input sets are shown in *Table 5.1*.

In addition, using the added extracted features results to the highest area under the curve with a value of 0.71 on the test set. This allows for the possibility to set an appropriate threshold to determine a heavier focus on either sensitivity or specificity while having a less negative impact on the other. With the reduced dimension feature set, setting to a higher threshold than 0.5 for selecting high risk classes will bring worse results than using a random classification model. This is reflected with the area under the curve value of 0.57. The ROC plots for all four feature sets are presented in *Figure 5.3*.

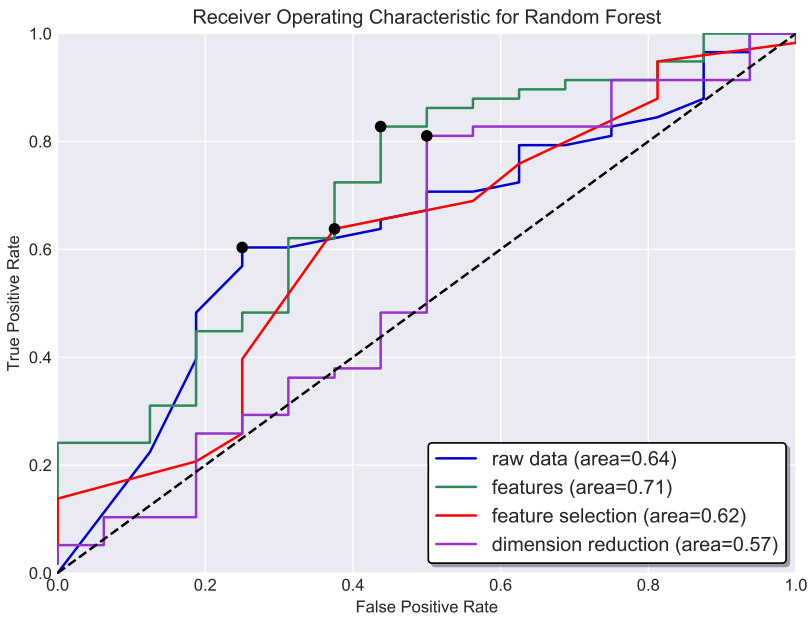

Figure 5.3: The ROC curve and the area under the curve for each of the input features when using a random forest classifier.

5.3.2 XGBoost Classifier

The *XGBoost*'s (Chen and Guestrin, 2016) implementation was taken from the XGBoost library[1]. The optimal hyperparameters selected for each of the input feature sets were:

- Original set of features: 'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 0.702684, 'colsample_bytree': 0.56484, 'gamma': 0.006109, 'learning_rate': 0.001236, 'max_delta_step': 0, 'max_depth': 8, 'min_child_weight': 14, 'n_ estimators': 3000, 'objective': 'binary:logistic', 'reg_alpha': 0.001034, 'reg_lambda': 1.133951, 'scale_pos_weight': 1, 'subsample': 0.538521

- Added extracted features: 'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 0.973668, 'colsample_bytree': 0.971151, 'gamma': 0.075752, 'learning_rate': 0.001888, 'max_delta_step': 0, 'max_depth': 6, 'min_child_weight': 4, 'n_estimators': 2600, 'objective': 'binary:logistic', 'reg_alpha': 0.001135, 'reg_lambda': 2.569113, 'scale_pos_weight': 1, 'subsample': 0.913667

- Feature selection: 'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel':

[1] *https://xgboost.readthedocs.io* "XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework"

0.653495, 'colsample_bytree': 0.687234, 'gamma': 0.000106, 'learning_rate': 0.32074, 'max_delta_step': 0, 'max_depth': 4, 'min_child_weight': 17, 'n_ estimators': 1800, 'objective': 'binary:logistic', 'reg_alpha': 0.000044, 'reg_lambda': 1.858376, 'scale_pos_weight': 1, 'subsample': 0.992918

- Dimensionality reduction: 'base_score': 0.5, 'booster': 'gbtree', 'colsample_ bylevel': 0.5503164100732001, 'colsample_bytree': 0.52901, 'gamma': 0.001356, 'learning_rate': 0.000111, 'max_delta_step': 0, 'max_depth': 7, 'min_child_weight': 2, 'n_estimators': 2200, 'objective': 'binary:logistic', 'reg_alpha': 0.000612, 'reg_lambda': 3.954649, 'scale_pos_weight': 1, 'subsample': 0.868741

Table 5.2: XGBoost classification results on the four input sets

Input	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Original Data	Train	0.8275	0.1725	0.7953	0.8596
	Validation	0.6806	0.3498	0.7018	0.6
	Test	0.6622	0.3994	0.7069	0.5
Extracted Features	Train	0.8333	0.1667	0.9239	0.7427
	Validation	0.8056	0.3019	0.8772	0.5333
	Test	0.8108	0.3109	0.8966	0.5
Feature Selection	Train	0.7135	0.2866	0.7544	0.6725
	Validation	0.625	0.3598	0.614	0.6667
	Test	0.6216	0.4725	0.6897	0.375
Dimensionality Reduction	Train	0.7573	0.2427	0.7135	0.8012
	Validation	0.6528	0.4198	0.7018	0.4667
	Test	0.6486	0.4076	0.6897	0.5

The best outcome is achieved on the extracted feature sets with a weighted G-mean loss value of 0.3109 and sensitivity and specificity values of 0.8966 and 0.5 respectively. This is furthermore reflected on the accuracy on the imbalanced test set reaching a percentage of 81.08%. However, the XGBoost classifier, regardless of the feature set, was unable to obtain a specificity score above 0.5, with feature selection achieving the lowest at a value of 0.375. It can benefit from setting a slightly higher threshold for selecting a sample as high risk to improve accuracy on low risk prediction. For the other three feature sets, the classifier resulted to fairly higher G-mean loss values ranging from 0.3994 for the original dataset to 0.4725 for the feature selection set. The complete results on all feature sets are presented in *Table 5.2*.

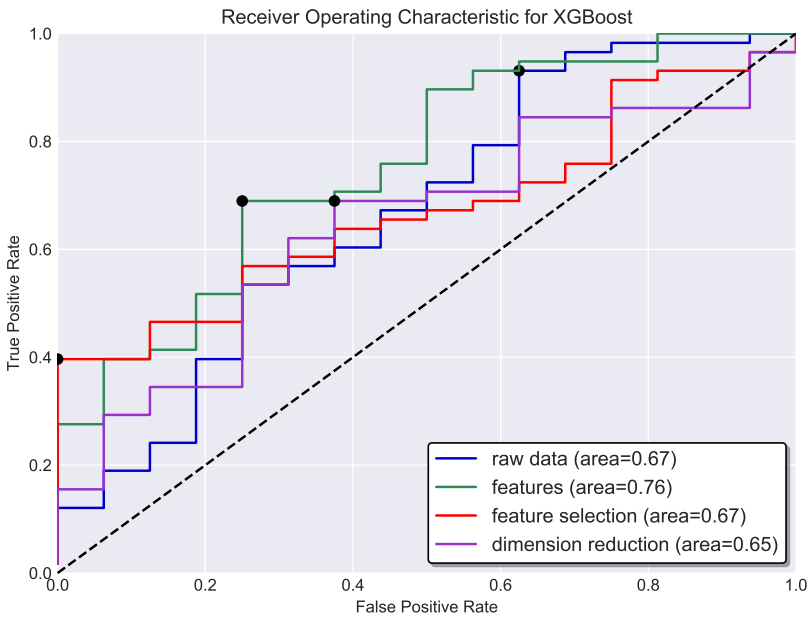

Figure 5.4: The ROC curve and the area under the curve for each of the input features when using the XGBoost classifier.

The area under the curve achieved on the extracted feature set is 0.76, allowing to set a good balance between sensitivity and specificity without increasing drastically the false classifications. The rest of the features achieve decent results that are better than random selection, with values 0.65 and 0.67. The plots for all four feature sets are shown in *Figure 5.4*.

5.3.3 K-Nearest Neighbour Classifier

The optimal hyperparameters selected for each of the input feature sets for the *K-Nearest Neighbours* (K-NN) (Cover and Hart, 2006) classifier were:

- Original set of features: 'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_neighbors': 3, 'p': 5.0, 'weights': 'uniform'
- Added extracted features: 'algorithm': 'auto', 'leaf_size': 30, 'metric': 'euclidean', 'metric_params': None, 'n_neighbors': 1, 'p': 2, 'weights': 'uniform'
- Feature selection: 'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_neighbors': 4, 'p': 4.0, 'weights': 'distance'
- Dimensionality reduction: 'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_neighbors': 4, 'p': 4.0, 'weights': 'distance'

Table 5.3: K-nearest neighbours classification results on the four input sets

Input	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Original Data	Train	0.8304	0.1696	0.7076	0.9532
	Validation	0.6389	0.3272	0.6141	0.7333
	Test	0.527	0.4829	0.5345	0.5
Extracted Features	Train	0.7953	0.2047	0.731	0.8596
	Validation	0.5833	0.3867	0.5614	0.6667
	Test	0.6081	0.3858	0.6034	0.625
Feature Selection	Train	0.6023	0.3977	0.6433	0.5614
	Validation	0.6389	0.3043	0.5965	0.8
	Test	0.527	0.5312	0.5689	0.375
Dimensionality Reduction	Train	0.5497	0.4503	0.6667	0.4327
	Validation	0.5972	0.4805	0.6491	0.4
	Test	0.5676	0.4817	0.6034	0.4375

The k-nearest neighbours model achieved the best results on the extracted features with a weighted G-mean loss value of 0.3858 on the test set. The sensitivity and specificity have near similar but low values of 0.6034 and 0.625 respectively. This is extended to low the accuracy with a percentage of 60.81%. The classifier struggles to correctly identify samples that are high risk compared to other models. The full results are shown in *Table 5.3*.

The area under the ROC curve achieved on all the feature sets is very low. On the extracted feature set, which achieved the highest, reached an area of 0.61. With raw data and dimensionality reduction achieved slightly better than a random classifier of 0.54 and 0.56 respectively. The most appalling results are achieved with the selected feature set managing to achieve an area under the curve of 0.49. Its performance is slightly worse than a random classification algorithm. The ROC curve plots for all the feature sets with the k-nearest neighbours on the test set are shown in *Figure 5.5*.

5.3.4 Support Vector Machine with Linear Kernel Classifier

The optimal hyperparameters selected for each of the input sets for the SVM with linear kernel were:

- Original set of features: 'C': 34.362729, 'cache_size': 200, 'class_weight': 'bal-

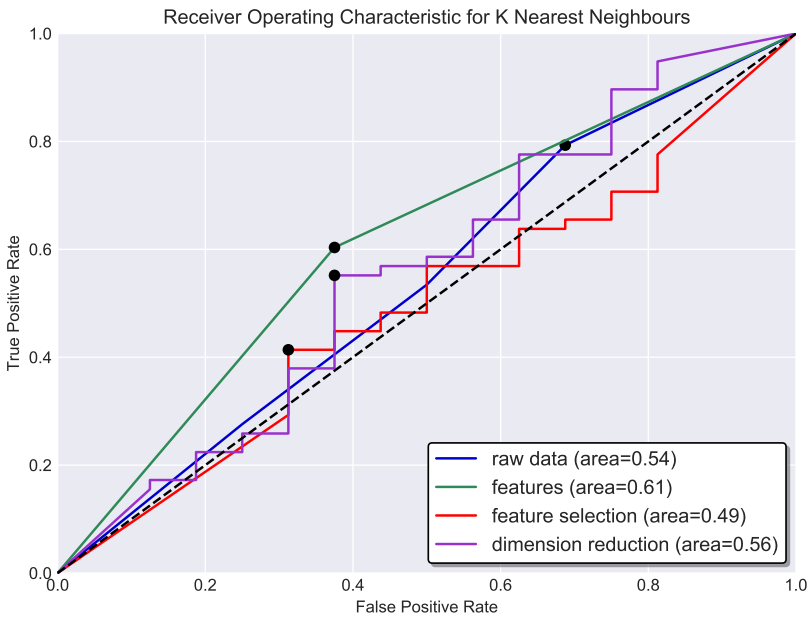

Figure 5.5: The ROC curve and the area under the curve for each of the input features when using a k-nearest neighbour classifier.

anced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 'auto', 'max_iter': 10000, 'shrinking': False, 'tol': 1e-06

- Added extracted features: 'C': 0.031607, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 'auto', 'max_iter': 10000, 'shrinking': False, 'tol': 1e-06

- Feature selection: 'C': 0.315749, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 'auto', 'max_iter': 10000, 'shrinking': True, 'tol': 1e-06

- Dimensionality reduction: 'C': 0.315749, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 'auto', 'max_iter': 10000, 'shrinking': True, 'tol': 1e-06

The best performance of the SVM with linear kernel is achieved on the extracted feature set with a weighted G-mean loss value of 0.3429 and sensitivity and specificity values of 0.6897 and 0.625 respectively. However, because of the low value, mainly sensitivity, the accuracy on the imbalanced test set remained quite low with a percentage of 67.57%. Following, with feature selection the achieved G-mean loss was slightly worse with a value of 0.3588. It however achieved better results on sensitivity

Table 5.4: SVM with linear kernel classification results on the four input sets

Input	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Original Data	Train	0.7807	0.2193	0.8012	0.7602
	Validation	0.6667	0.3844	0.7018	0.5333
	Test	0.6216	0.4241	0.6551	0.5
Extracted Features	Train	0.6842	0.3158	0.6784	0.6901
	Validation	0.6944	0.3159	0.7018	0.6667
	Test	0.6757	0.3429	0.6897	0.625
Feature Selection	Train	0.6754	0.3246	0.8655	0.4854
	Validation	0.7222	0.3511	0.7719	0.5333
	Test	0.7297	0.3588	0.7931	0.5
Dimensionality Reduction	Train	0.6784	0.3216	0.7544	0.6023
	Validation	0.6528	0.3669	0.6667	0.6
	Test	0.5946	0.4653	0.6379	0.4375

(0.7931) than specificity (0.5333) which also allowed it to obtain a higher test accuracy of 72.97%. The full results of the classifier on all the feature sets are shown in *Table 5.4*.

The obtained area under the ROC curve for all features are somewhat under par. The highest achieved is an area of 0.67 for both the added extracted features and with feature selection. With the raw data and dimensionality reduction varying the threshold might result to worse performance than a random classifier. Their area is 0.6 and 0.63 respectively for the two features. The ROC curve plots for all four features are presented in *Figure 5.6*.

5.3.5 Support Vector Machine with RBF Kernel Classifier

The optimal hyperparameters selected for each of the input sets for the SVM with *Radial Basis Function* (RBF) kernel were:

- Original set of features: 'C': 138.243011, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 0.125856, 'max_iter': 5000, 'shrinking': False, 'tol': 1e-06

- Added extracted features: 'C': 0.788815, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma':

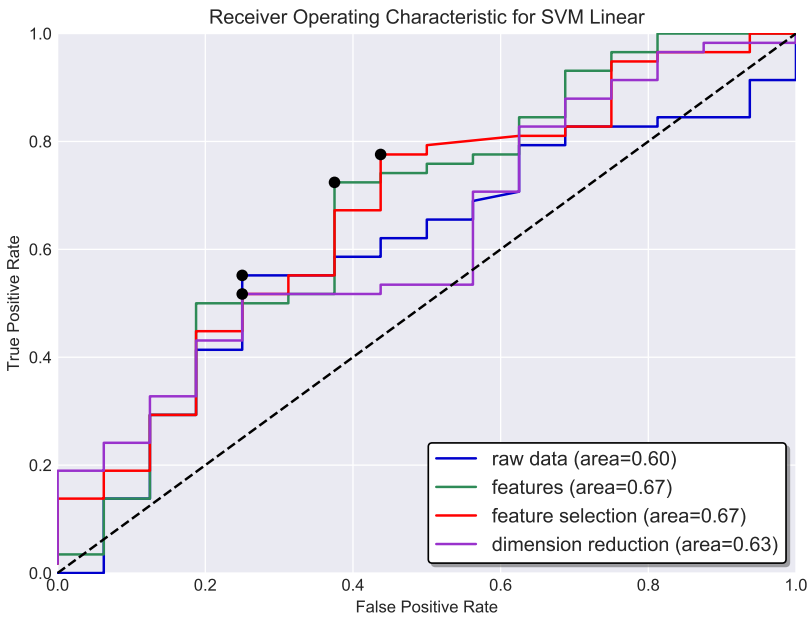

Figure 5.6: The ROC curve and the area under the curve for each of the input features when using a SVM with linear kernel classifier.

0.009116, 'max_iter': 5000, 'shrinking': False, 'tol': 1e-06

- Feature selection: 'C': 0.788815, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 0.009116, 'max_iter': 5000, 'shrinking': False, 'tol': 1e-06
- Dimensionality reduction: 'C': 0.788815, 'cache_size': 200, 'class_weight': 'balanced', 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 0.009116, 'max_iter': 5000, 'shrinking': False, 'tol': 1e-06

Overall performance on all feature sets using SVM with RBF kernel is very low. The best performance is on the extracted feature set with a weighted G-mean loss value of 0.5334. It achieved a sensitivity score of 0.8103 and a specificity score of 0.1875, with the accuracy reaching 67.57%. The classifier heavily favours predicting high risk samples with the highest specificity is at 0.1875 and the lowest is at 0.0625. The full results are shown in *Table 5.5*.

These results are also reflected on the ROC curve which can be seen in *Figure 5.7*. Decent area under the curve is only achieved on the raw dataset with a value of 0.61, allowing for some threshold balancing for preferring between sensitivity and specificity. Following, the extracted features have only an area of 0.53 and outperform a random classifier at a lower threshold. The last two have appalling results that they

Table 5.5: SVM with RBF kernel classification results on the four input sets

Input	Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Original Data	Train	1.0	0.0	1.0	1.0
	Validation	0.8611	0.33	0.8615	0.4
	Test	0.7432	0.5687	0.7826	0.0625
Extracted Features	Train	0.6521	0.3479	0.8655	0.4386
	Validation	0.7639	0.3264	0.8246	0.5333
	Test	0.6757	0.5334	0.8103	0.1875
Feature Selection	Train	0.5614	0.4386	0.7778	0.345
	Validation	0.6806	0.4034	0.7368	0.4667
	Test	0.6216	0.5629	0.7414	0.1875
Dimensionality Reduction	Train	0.5702	0.4298	0.7836	0.3567
	Validation	0.7083	0.4468	0.807	0.3333
	Test	0.6216	0.5629	0.7414	0.1875

can be replaced by a random classifier to obtain better results. With feature selection it obtains an area of 0.49 and with dimensionality reduction an area of 0.42.

5.4 Conclusions

All the algorithms obtain their best performance based on the weighted G-mean loss value of the test set on the extracted feature set. This shows that the added features provide extra information for the algorithms to better distinguish between the low risk and high risk classes. The top performing algorithm is the XGBoost with G-mean loss of 0.3109 and accuracy of 81.08%. Following, is the SVM with a linear kernel in which it obtained a G-mean loss of 0.3429, with however a more balanced score on sensitivity and specificity being 0.6897 and 0.625 respectively. The worst performance on all of the features was obtained solely from SVM with RBF kernel. One of the possible reasons for this is that the distribution of the data was not centred to have a mean of zero, which the RBF kernel assumed to be so (Phienthrakul and Kijsirikul, 2005).

A general conclusion is that the algorithms are able to perform better without any feature selection. Despite during feature selection, using SVM with a linear kernel on the whole dataset to evaluate the selection showed improvement on the results. However, this was not reflected when splitting the dataset to smaller subset of training,

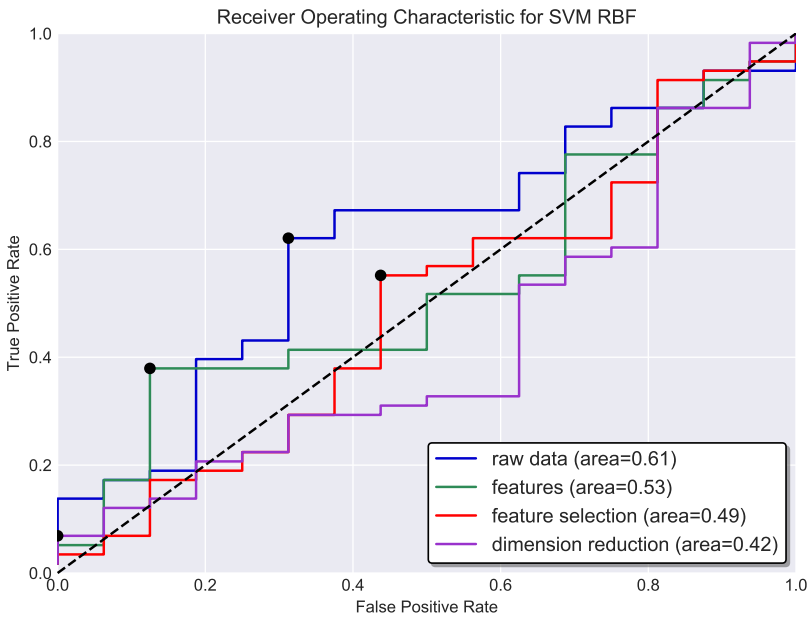

Figure 5.7: The ROC curve and the area under the curve for each of the input features when using a SVM with RBF kernel classifier.

validation and test. The results obtained on the training set were also lower indicating that there was a decent amount of information loss. This shows the high variance in location of where cancer can be detected and the uniqueness of characteristics in the dataset. Even features that have very small informativeness can help with generalisation. In addition, random forest and XGBoost classifier, being ensembles of trees (Breiman, 2001; Chen and Guestrin, 2016), they can rank the features based on their importance and so make the feature selection step redundant.

Although using feature selection decreased the performance of the algorithms, using dimensionality reduction showed slight improvements. Using PCA allowed to condense the features and obtain more informative and higher quality features in their place, aiding the classification algorithms. In spite of that, the results are still not comparable with either the original features or with the additional extracted ones.

While the best performance in classification on the data was achieved through the added extracted features we also want to evaluate on how they perform on the original dataset. One of the goals is to be able to generalise easily from classifying cancer in mammary glands to detecting cancer at any location using microwave thermography. These means no added extracted features and no extended preprocessing because for each location a new set of mathematical functions has to be defined. The best performer on the original dataset is again the XGBoost classification algorithm with a weighted

G-mean loss of 0.3994, sensitivity of 0.7069, specificity of 0.5 and accuracy of 66.22% on the test set.

Chapter 6

Cascade Correlation Neural Network Evaluation

All the following models were built using *Keras* (Chollet et al., 2015) version 1.5 with *TensorFlow* (Abadi et al., 2016) backend version 1.1 and the source code for the cascade and deep networks are available at *appendix B.1* and *B.2* respectively. The only preprocessing conducted on the dataset was data normalisation on the reference point 2, as described in *section 3.3*. Then the reference points for skin surface and at a depth were removed after normalisation. Then the dataset was balanced split into training (60% - low risk: 46 and high risk:171), validation (20% - low risk: 15 and high risk:57) and test (20% - low risk: 16 and high risk:58).

6.1 Base Model

6.1.1 Model Description

A non-weighted G-Mean batch-wise loss function was implemented to monitor the network during training of the output layer. It was not used to directly optimise on because it was not possible to obtain a differentiable global G-mean loss function. Hence, it was used solely as an early stopping criteria based on the validation set (i.e. exiting before the maximum determined epochs if the validation set has no improvement). Early stopping was also constraint to allow up to 50 epochs from when the improvement on the validation loss last stopped before truly terminating. The obtained validation loss at a new epoch must be less than 0.00001 from the current optimal to be considered as an improvement.

The optimisation loss function used instead was a categorical cross entropy (de Boer et al., 2005). By using a categorical loss function this means that the class output must be represented with binary values in a vector. This was achieved by one-hot encoding the output classes so that we have as a result all samples with output class 0 transformed to (1, 0) and for class 1 to (0, 1). The cross entropy loss is a probabilistic function and as such requires to be fed a vector that sums to one and its individual values are within the range of [0, 1]. This was achieved by using a softmax activation function (Bishop, 2006) for the output layer of the network.

Having an imbalanced dataset requires to either use oversampling techniques, similarly used in *section 5*, or attribute appropriate weights to the loss function used to give equal importance to the underrepresented class. Because, as mentioned previously, we are not able to obtain the global G-mean loss value during batch-wise computations and allocate the appropriate balancing weights this was compensated by using class weights over oversampling. The categorical cross entropy is able to take into account the class weights and maintain a balance between the two classes when updating the weights.

Each hidden layer added used as its activation function a sigmoid function, as was used in the original paper of the cascade correlation algorithm by Fahlman and Lebiere (1990). Additionally, the loss function for the hidden layers used was a correlation coefficient one to update the weights so that they minimise the residual error obtained from the output class (Fahlman and Lebiere, 1990). As with the output layer, the hidden layers also incorporated an early stopping criteria on the validation loss function. It allowed for a grace period of 50 epochs before stopping if the validation loss has not decreased more than 0.00001. The maximum number of epochs that it executed for was also 1000. Lastly, the candidate pool size was set to 16 and the candidate with the lowest validation loss was selected.

Early stopping with patience was expanded from the individual epochs to also each iteration of hidden layers being added. The criteria was used on the validation G-mean loss obtained from the output layer. The grace period was set to 50 iterations and anything higher than 0.0001 of decrease was considered as an increase in performance. When the lowest G-mean validation loss was achieved on the current model (with its added hidden layers) then the obtained weights/bias were stored and treated as the final ones.

The weights for the output and each candidate hidden layer were set randomly within a normal distribution. The mean sampling point was set to 0 with a standard

deviation of 0.5. The bias of each layer was initialised to 0. The output layer's weights were reinstantiated after every iteration to avoid being stuck at bad local minimums.

From a previous research (Zenovich et al., 2016), they used *simulated annealing* (SA) (Kirkpatrick et al., 1983) as an optimisation function for the problem. However, for this base model *stochastic gradient descent* (SGD) was used in its place (Bottou, 2010). Based on the findings of Bottou (1991), SGD behaves very similarly to SA. SA is able to set the weights so that a global minimum of the error is found. On the other hand, SGD can obtain a very good local minimum error rate in which it approaches that found by SA. Obtaining a good local minimum can be better than obtain a global one on the training set, in which it will allow for some generalisation to the validation and in turn to the test set. The learning rate of the SGD optimiser for the output layer was set to 0.00001 and for the hidden candidate layers to 0.000005, after exploring various values.

6.1.2 Results

The optimal network built included 62 hidden layers with each containing two neurons. The output layer ended with having an input vector of size 164 in which 40 of the features are the initial data passed. It achieves on the test set a weighted G-mean loss of 0.5889, sensitivity of 0.4483, specificity of 0.375, which is also reflected on the unbalanced accuracy which resulted to 56.91%. However, on the validation set the network is able to obtain more decent results achieving a weighted G-mean loss of 0.3512. This indicates the lack of ability for the network to generalise to unseen data. Another note is that the performance on the training set is much worse than that on the validation set, with G-mean loss of 0.4681. Each time the network attempts to extract features that will aid in classifying the samples in the training set hinder that in the validation, showing that there are unique characteristics between the data splits. The full results of the base network are presented in *Table 6.1*.

Table 6.1: Results for base cascade correlation neural network.

Dataset	**Accuracy**	**G-Loss**	**Sensitivity**	**Specificity**
Train	0.5253	0.4681	0.5205	0.5435
Validation	0.5694	0.3512	0.5088	0.8
Test	0.4324	0.5889	0.4483	0.375

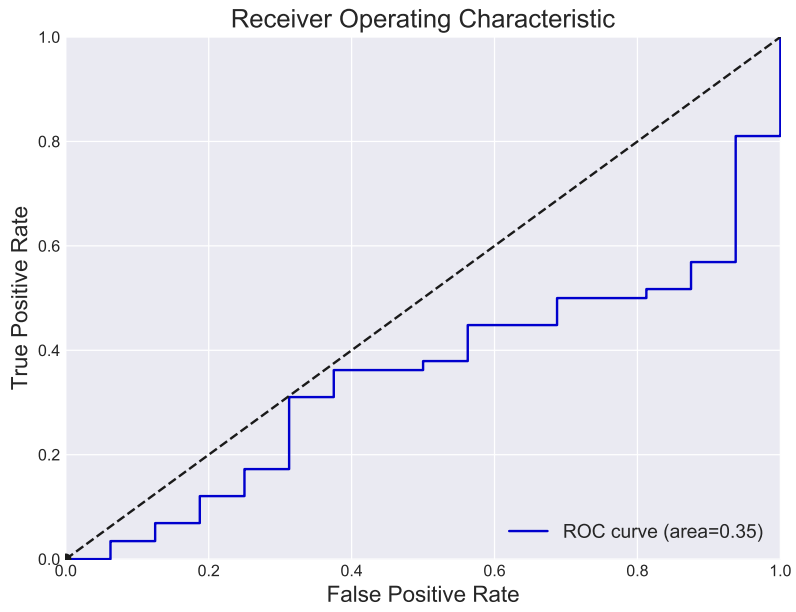

Figure 6.1: The ROC curve and the area under the curve for the test set on the base cascade network.

Having such a low accuracy the results of the ROC curve are also appalling. The curve is plotted in *Figure 6.1*. As seen in the figure, it is a lot worse than a random binary classifier for any possible threshold. This is also concluded from the area under the ROC curve which it obtained a value of 0.35.

The base network is highly unstable and it is ineffective in extracting features to distinguish between the two classes. For the majority of the iterations in adding hidden layers the network classifies all samples as one class or the other. Occasionally it is able to make some correct predictions for both classes, but returns to shifting weights to a bad optimum. The accuracy and the average batch-wise G-mean loss of the output layer as hidden layers are added is shown in *Figure 6.2*.

6.2 Improved Model

6.2.1 Model Description

The model for the improved version was carried over from the base model as described in *section 6.1.1*. Here only changes or additions to the base model will be included and everything else remains unchanged. The improvements were based on the results presented previously in (*section 6.1.2*).

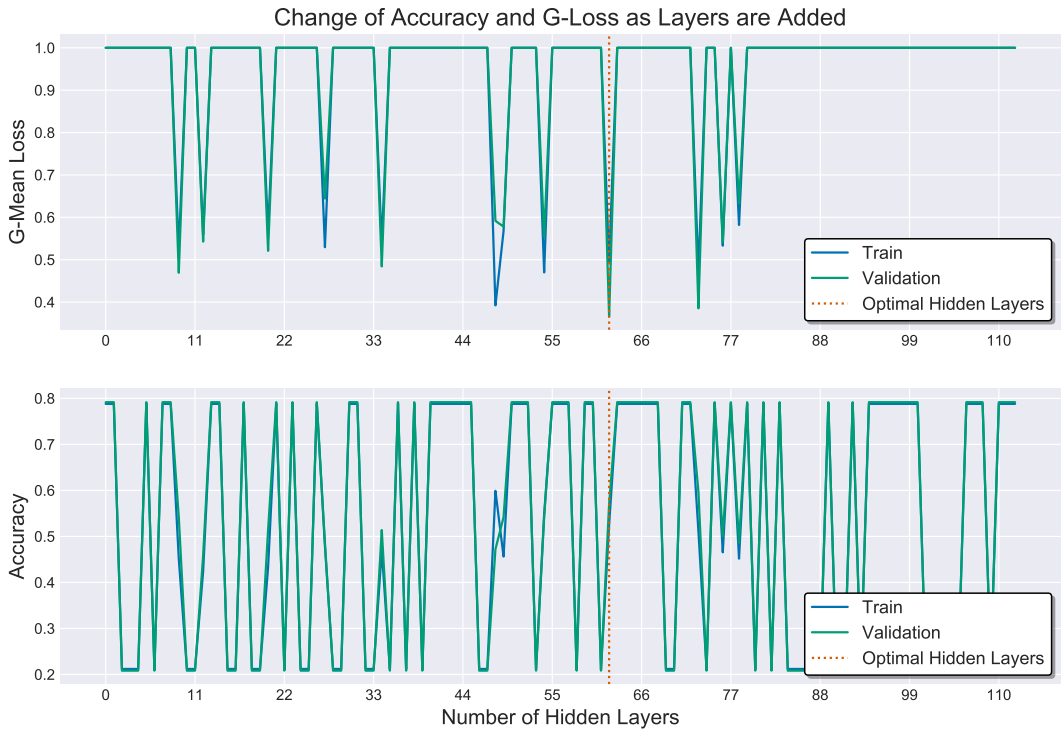

Figure 6.2: The average batch-wise G-mean loss and accuracy of the base network as hidden layers are added.

One of the main issues from the base model is that it spikes constantly between predicting all samples as positive or as negative. The range in which the weights are initialised is very important because if the weights are too small then there will be slow convergence or the input will lead to 0 and if they are too high then the network will not be able to converge (Thimm and Fiesler, 1995). So, from the random normal distribution the weight initialisation scheme was changed to the *Xavier* initialiser (Glorot and Bengio, 2010) sampling from a normal distribution, in which the range of the weights will be based on the fan-in and fan-out number of neurons.

After various tests, the optimiser was also changed to *Adam* (Kingma and Ba, 2014) which combines the principals from *RMSProp* (Hinton et al., 2012) and *ADAGRAD* (Duchi et al., 2011) as extensions to SGD. Incorporating Adam as the optimiser will often result to better and faster convergence than other optimisation algorithms (Ruder, 2016). Through various trials the algorithm's parameters were set to 0.00001 and 0.000005 learning rate for the output and hidden layers respectively and for both the rest of the parameters were set as beta1 (decay of first-order gradient) to 0.9, beta2 (decay of second-order gradient) to 0.999 and epsilon to 1e-08, as recommended by Kingma and Ba (2014). Because of the change of the optimisation algorithm which

incorporates decay of the learning rate, the network did not fully converge at 1000 epochs as used previously. As such, for both the candidate hidden layers and output layer the epochs were increased to 10000, while maintaining early stopping to prevent overfitting.

A more resent research (Krizhevsky et al., 2012) showed that modern techniques improve training speed and are able to push to a better optimal, at least for classification problems. One such method is *Rectified Linear Units* (ReLUs) (Nair and Hinton, 2010) which have shown some improvement over sigmoid activation functions. In addition, it is computationally non intensive while also capable of boosting training speed of each individual hidden candidate. Also, with ReLUs it promotes sparsity within the added layers by setting some of the weights of the neurons to 0. This allows to indirectly remove one or even both of the neurons that will be added to a layer. In addition, if one of the previous hidden layers generates highly erroneous output it can be negated and prevented from propagating through the network.

Previously, for each iteration the output layer had its weights reinitialised based on the defined weight scheme. This was done to allow the algorithm to escape from bad local minimums that was created in combination from the way the weights were initialised and the way they were update through SGD. With the previously mentioned changes for the improved model, warm-start of the output layer weights was used, a similar technique used in ensemble machine learning algorithms (Feurer et al., 2015). Warm-start was used because of the presence of a low learning rate with momentum and decay in which it should boost performance and speed up training (Senior et al., 2013). When the model reaches the currently lowest G-mean validation loss the weights of the output layer were stored. These weights were used to set the respective links for all future instantiated output layers. The new links created from the added hidden layers were initialised as normal until they contribute to minimising the G-mean loss.

6.2.2 Results

The improved cascade network reached a total of 47 hidden layers with each containing two neurons to obtain its optimal results on the validation set. The output ended with an input size of 134. On the test set it obtained a weighted G-mean loss of 0.5417, sensitivity of 0.6207, specificity of 0.3125 and a biased accuracy of 55.41%. This is a slight improvement from the previous model shown in *section 6.1.2* in regards to

the G-mean loss value. However, this model achieved slightly worse output in regards to specificity, but compensated significantly with its results in sensitivity. There is a similar trend as previously were the model is able to obtain better results on the validation set compared to the training set. This again indicates an issue with the small dataset in which there are not enough samples for a characteristic found in one to be generalisable or useful to others. The full results on all three datasets are shown in *Table 6.2*.

Table 6.2: Results for improved cascade correlation neural network.

Dataset	**Accuracy**	**G-Loss**	**Sensitivity**	**Specificity**
Train	0.6313	0.3851	0.6433	0.587
Validation	0.6944	0.2677	0.6667	0.8
Test	0.5541	0.5417	0.6207	0.3125

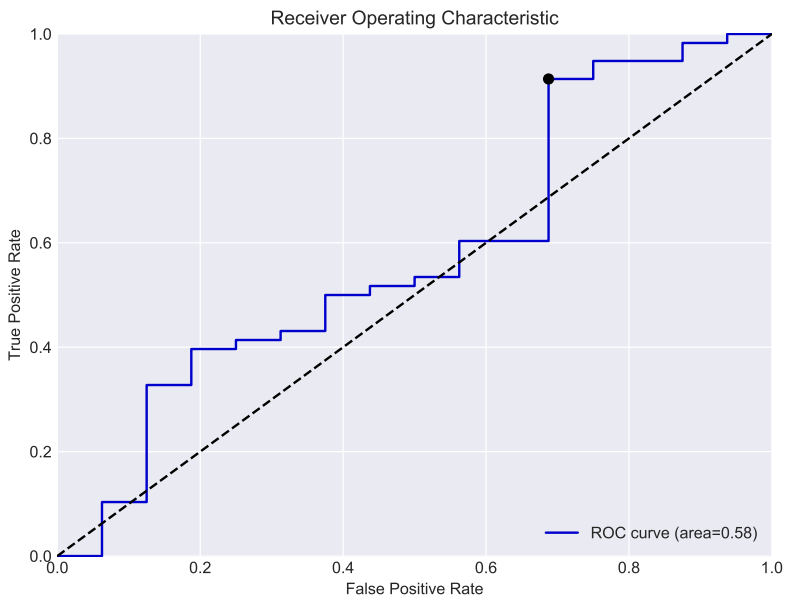

Figure 6.3: The ROC curve and the area under the curve for the test set on the improved cascade network.

The ROC curve for the test set is also plot and is shown in *Figure 6.3*. The area under the ROC curve achieved by the network was at 0.58 which is slightly better than a random binary classifier, but still very low to be regarded as a decent classifier. Also, it allows for some shift of the threshold between preference for low or high risk classes as compared with the base model.

Figure 6.4: The average batch-wise G-mean loss and accuracy of the improved network as hidden layers are added.

The different weight initialisation scheme has helped the network to start from a more favourable position. After the first weight update of the output layer it ended in a decent local minimum, managing to obtain a weighted G-mean loss of about 0.6 on both the training and validation set. In addition, it has a more gradual improvement as hidden layers are added until it reaches its optimal size, avoiding classifying all samples as one class or the other. The G-mean loss and the accuracy as hidden layers are add can be seen in *Figure 6.4*.

6.3 Extended Model

6.3.1 Model Description

Here an extension to the cascade correlation neural network's dynamic architectural construction is proposed. Seeing the results from the previous two models in *sections 6.1.2* and *6.2.2*, they were unable to effectively generalise the samples in the training set to the validation. Beyond a point, each time useful information would of been extracted from the training set it would hinder the validation set and so was disregarded.

So, to make the network more generalisable with each hidden candidate unit it will create a set of possible combinations of a Gaussian noise, a dropout (Srivastava et al., 2014) and a batch normalisation (Ioffe and Szegedy, 2015) layer.

Each hidden candidate layer had the following format:

- Gaussian noise layer, with a mean of 0 and a standard deviation of 0.5
- Dense layer
- Batch normalisation layer, with momentum at 0.99, epsilon at 0.00001 and a trainable beta value as an added offset to the normalised results
- Dropout layer, which it will randomly drop one of the two neurons by setting its weights to 0

During each hidden layer addition it generates all possible combinations of candidate layers, eight in total, of the dense layer and the different regularisers as shown above. However, this does increase the training time and computational demand and so the pool size was reduced to two for each generated candidate so that the total pool size was maintained to 16. Also, a combination of *Lasso Regression* (L1) and *Ridge Regression* (L2) (Ng, 2004) were added to the hidden and output layers but did not bring any improvements to the results and so was not further used. The rest of the model is carried over as is, as described in *section 6.2.1*

6.3.2 Results

The algorithm resulted in constructing a network that consisted of 36 hidden layers, having all possible combinations with Gaussian and dropout layer. However, batch normalisation was not selected in any of the hidden layers as for it did not bring the lowest loss value from the candidate layers. This should be expected because, other than the first hidden layer, the layers had as input the feature vector of the original data which consists of values around 36 and a feature vector from the output of previous layers where its values range from [0, inf).

It obtained a weighted G-mean loss of 0.5495, sensitivity of 0.6034, specificity of 0.3125 and accuracy of 54.05% on the test set. Even with the extended version of the cascade network, it presents the same issue as previously in which it is unable to effectively generalise to the unknown test set. While it obtained slight worse results than that of the improved model on the test set (*section 6.2.2*) it obtains much better results

on the validation set. The G-mean loss on the validation set is at 0.1578, sensitivity at 0.7544 and specificity at 0.9333.

Table 6.3: Results for extended cascade correlation neural network.

Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Train	0.6129	0.3851	0.6141	0.6087
Validation	0.7917	0.1578	0.7544	0.9333
Test	0.5405	0.5495	0.6034	0.3125

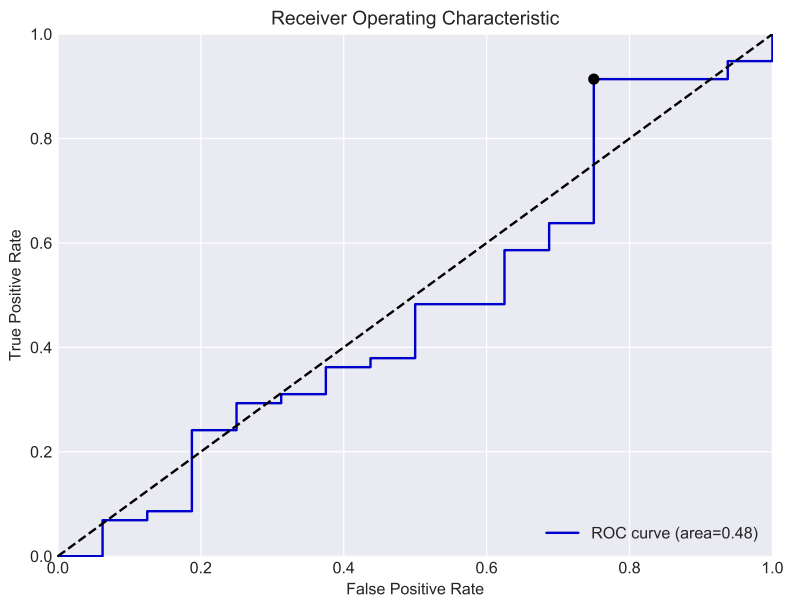

Figure 6.5: The ROC curve and the area under the curve for the test set on the extended cascade network.

From the ROC curve, shown in *Figure 6.5*, performs poorly on the test set. It obtains worser performance than a random binary classifier with an area under the ROC curve of 0.48. At only a high threshold it allows the network to obtain better performance than a random classifier.

With smaller number of hidden layers there is higher disparity between the training and validation loss. As the number of layers increase the precedence of noise in the network is mitigated allowing a closer relationship between the training and validation set. This results to an improved performance on the training set but hampers that of the validation set, resembling closer to the results obtained on the improved model at a high number of layers. The full results of the accuracy and the average batch-wise G-mean loss as hidden layers are added is show in *Figure 6.6*.

Figure 6.6: The average batch-wise G-mean loss and accuracy of the extended network as hidden layers are added.

Increasing the generalisation ability of the network from the training set to the validation set allowed it to improve performance on the validation set. However, the results of the test set did not show a proportional improvement as was expected with the improvement on the validation. By dynamically building the network that included Gaussian noise and dropout layers it allowed it to overfit to the validation set. Reaching the same conclusion, there are unique characteristics split between the three datasets and the cascade network is unable to generalise enough to predict unknown outliers in the test set.

6.4 Deep Neural Network

6.4.1 Model Description

A deep neural network was also built to compare its results with that from the previous cascade networks. In addition, it was used to evaluate the obtained results in relation to the overall training speed. The network was built taking in mind the results from the three previous models, especially in regards to the fact that the network should

incorporate techniques to boost generalisation capabilities.

The hidden layers used ReLU as their activation function and the output layer used softmax. The optimiser used is Adam with a learning rate of 0.00005, beta1 at 0.9, beta2 at 0.999 and epsilon at 1e-8. Also, categorical cross entropy was utilised as the loss function to optimise on for up to a maximum of 2000 epochs. Again, because no oversampling techniques were used to balance the training set, appropriate class weights were passed to the loss function. Like before, early stopping on the validation set was used with the batch-wise G-mean validation loss as the metric to monitor on. In addition, early stopping had a grace period before truly terminating of 500 epochs. The network consists of the input layer, five hidden layers and the output layer. The architecture of the network is summarised in *appendix A.1*.

6.4.2 Results

Table 6.4: Results for deep neural network.

Dataset	**Accuracy**	**G-Loss**	**Sensitivity**	**Specificity**
Train	1.0	0.0	1.0	1.0
Validation	0.7639	0.2728	0.7895	0.6667
Test	0.7703	0.2843	0.8103	0.625

The network was able to do a perfect fit of the training set but was still able to generalise onto the validation and test set effectively. It achieved a weighted G-mean loss of 0.2843, sensitivity of 0.8103, specificity of 0.625 and accuracy of 77.03% on the test set. The validation set achieved very similar results to the test set with a G-mean loss of 0.2728. The final results are shown in *Table 6.4*. The improvement of the validation set as the network is trained is closely related to the training set. Further improvement of the validation set is restricted by the fact that there are no more features for the network to extract from the training set. The plot of the average batch-wise G-mean loss and accuracy during the network's training is shown in *Figure 6.8*.

Also, plotting the ROC curve of the test results shows that different thresholds can be used to balance between the specificity and sensitivity without hindering heavily one or the other. Its strong performance is also reflected on the area under the ROC curve which it obtained a value of 0.72. The model is able to significantly outperform a random binary classifier on the problem. The ROC curve is plotted in *Figure 6.7*.

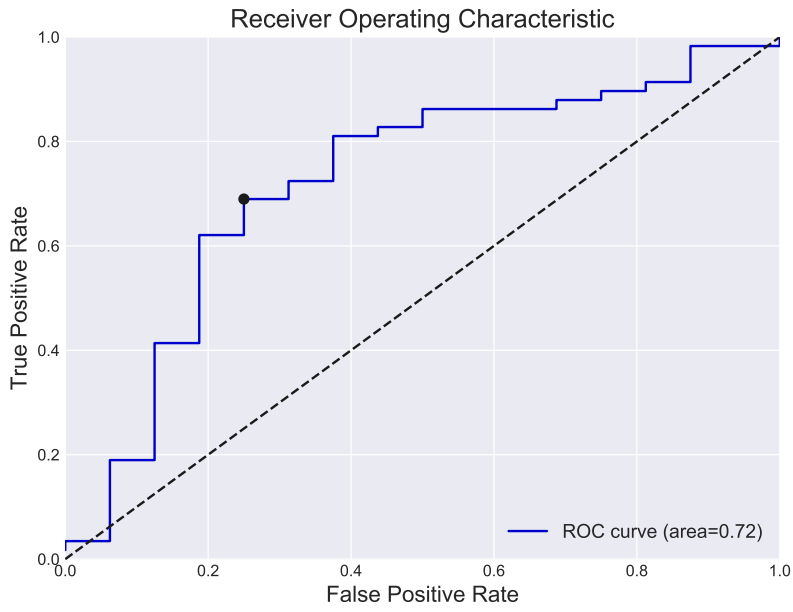

Figure 6.7: The ROC curve and the area under the curve for the test set on the deep neural network.

Figure 6.8: The average batch-wise G-mean loss and accuracy of the deep neural network as it is trained through more epochs.

6.5 Conclusions

From the cascade correlation neural networks, the best performer on the test set was the improved version with a weighted G-mean loss of 0.5417 and with the extended

model slightly trailing behind. Their performance is quite low and there is a significant difference from the top performing non-neural network, the XGBoost algorithm, regardless of the input feature set passed.

With cascade correlation neural networks they attempt to build an optimal model that optimise the performance on the validation set by extracting features from the training set. In this case, having a small dataset with unique characteristics between the sets quickly hinders any improvement on the validation set as its further fitted to the training set and so stagnates any further weight updates and layer additions. For this specific problem, by adding noise to the network it allowed it to obtain a huge boost in performance on the validation set. However, this lead to overfitting on the validation set as seen with its final performance on the test set. With the added noise the network was most likely also able to select the hidden layer candidate that distorted the input of the training set in such a way that it allowed it to become very close to that of the validation. Hence, it propagated the problem from generalsing from the training set to generalising from the validation set.

A deep neural network was also built to compare with the cascade networks. The proposed model in *section 6.4.1* can be trained in a very short time and much faster than the cascade networks. With the cascade a lot of the processing to train the hidden layer candidates is discarded. While being faster it also outperform the cascade networks drastically, obtain a weighted G-mean loss value of 0.2843 on the test set. However, it does use significantly more memory to fit the whole network, but still of an insignificant amount in todays terms.

The deep network does not only outperform the cascade networks but also the non-neural network models, especially the top performer XGBoost. It is able to outclass the XGBoost algorithm regardles if it was passed the additional extracted features, with G-mean loss of 0.3109, or the original feature set, with G-mean loss of 0.3994. This indicates that neural networks are highly capable of extracting useful features from the dataset that are better than individually defining methods, which is tied to the problem at hand, and is able to detect unconsidered relations (Mao and Jain, 1995). The deep net was roughly built to obtain some initial results on its capabilities against dynamically built networks and so has potential for further improvements, as for optimisations have not being fully exhausted.

Chapter 7

Convolutional Neural Network

7.1 Model Description

Convolutional neural networks have been used successfully for various other problems in breast cancer detection, such as Cireşan et al. (2013); Spanhol et al. (2016); Arajo et al. (2017). In combination with the successful results seen from using a deep neural network, a convolutional network will also be explored here in the hopes to obtain further improvements. Like previously, the implementation of the network was focused around the ability to generalise to unseen data.

The dataset was normalised, as described in *section 3.3*, on the reference point 2 to take into account environmental ambient temperature. On the deep neural network, by adding at the beginning a batch normalisation layer it helped boost the network's performance. However, transforming the vector to an image with added 0 values would not bring the desired results with such normalisation. So, an additional preprocessing step was taken to centre the data before transforming. To avoid distorting outliers a robust scaler based on the interquartile range was used. The normalised and centred data was then transformed to an image with two channels, as described in *section 4.1.2*. From the image transformation, two different variance were used to test on the convolutional network. The first is without any changes from the original method and the second splits each gland with its associated axillary point for both skin and at a depth readings to its own channel, with a total of four (image size: 6x6 with four channels). Furthermore, with and without image generation on the training set was explored, as described in *section 4.6*. The goal with the generated images was to determine if it can bring improvements to the network by adding more sensible noise to the network in an attempt to boost generalisation capabilities.

The dataset was again balanced split into training (60% - low risk: 46 and high risk:171), validation (20% - low risk: 15 and high risk:57) and test (20% - low risk: 16 and high risk:58) sets. For the additional images, the training set was first balanced using SMOTE with borderline1 variance. Following, the image generation took place which resulted to a total of 5472 samples (2736 low risk and 2736 high risk).

Various activations and optimisation functions with different learning rates were tested and the ideal ones were ReLU activation functions for the hidden layers and Adam optimiser with a learning rate of 0.0000005, beat1 at 0.9, beta2 at 0.999 and epsilon at 1e-8. The output layer used a softmax activation function. Similarly with previous networks, the loss function used is a categorical cross entropy and early stopping on the validation set using batch-wise G-mean loss. Early stopping was allowed for a much higher patience before terminating of 1000 epochs. Finally, the weights of all the layers were initialised using Xavier uniform distribution algorithm.

The network was designed using Keras version 2.2.2 with TensorFlow backend version 1.9 and the source code is available in *appendix B.3*. It was primarily built using convolutional (Lecun et al., 2015), dense, separable convolution (Chollet, 2016), max pooling (Lecun et al., 2015), global average pooling (Lin et al., 2013), spatial dropout (Tompson et al., 2014), dropout, batch normalisation and Gaussian noise layers. Designing the architecture of the network was heavily limited by the dimensions of the transformed images. The final architecture is listed in *appendix A.2*, after experimenting with various versions.

7.2 Results

The results of only the best performing variance out of the four (two or four channel images in combination with or without image generation) will be presented. The best performer based on the test set results is using two channel images (the mammary gland pairs are maintained) with image data generation.

From the different input variations, the best performer on the test set is with two channels and image generation. Closely in second came the input that had the mammary glands split across four channels and again with image generation. The two other variances which did not have image generation performed substantially worse. The two channel with image generation obtained a weighted G-mean loss of 0.3637, sensitivity of 0.5862, specificity of 0.6875 and a biased accuracy of 60.81%. The network was prevented from obtaining better performance on the training set by adding large

Table 7.1: Results for convolutional neural network with two channel images and additional image generated.

Dataset	Accuracy	G-Loss	Sensitivity	Specificity
Train	0.8392	0.1608	0.8845	0.7939
Validation	0.5972	0.3106	0.5263	0.8667
Test	0.6081	0.3637	0.5862	0.6875

amount of regularisation. This however helped sufficiently enough to improve results on the validation set and consequently on the test set. The effect of regularisation can be seen in *Figure 7.2*, which there is a high amount of variation between one epoch to the other on the training set. Because of the unique characteristics in the validation set compared to the training, this is further amplified with the validation data having a larger variance. The full results on the three datasets can be seen in *Table 7.1*.

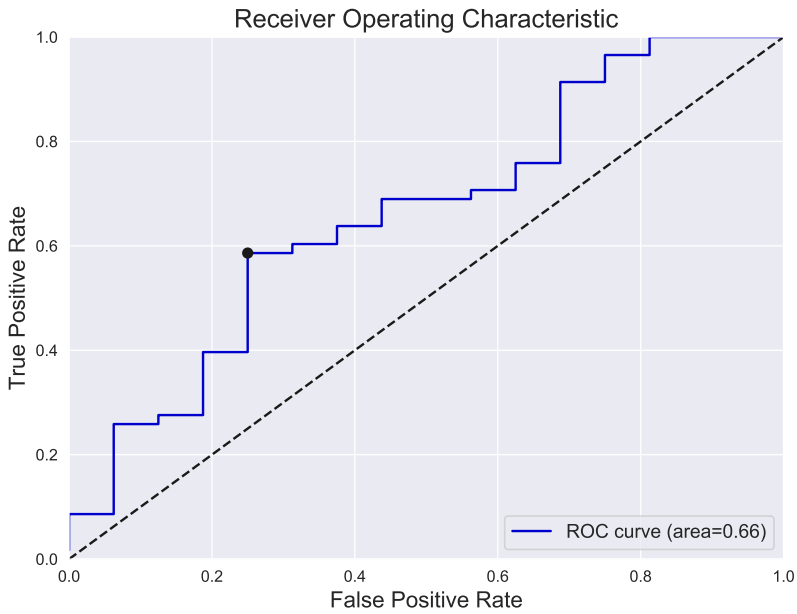

Figure 7.1: The ROC curve and the area under the curve for the test set on the convolutional neural network with two channels and image generation.

From the ROC curve, the model is able to outperform a random binary classifier at any threshold level. This allows it be more flexible when balancing between sensitivity and specificity. It obtained a decent area under the curve of 0.66. The ROC curve is plotted in *Figure 7.1*.

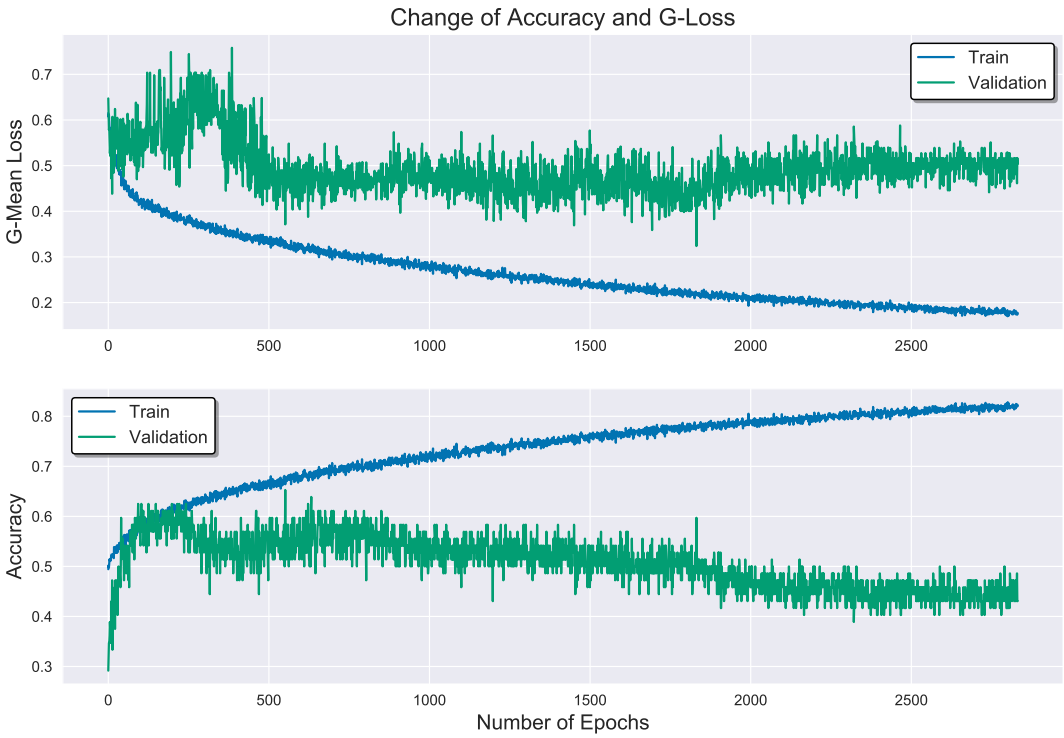

Figure 7.2: The average batch-wise G-mean loss and accuracy of the convolutional neural network with two channels and image generation as it is trained through more epochs.

7.3 Conclusions

By adding oversampling through image transformations, it allowed the network to further improve its performance. It acted as an extra regularisation method preventing it from overfitting to the training set. Additionally, it added enough variance and noise to allow it to improve its generalisation ability. This allowed the model to take advantage of its capability to extract positional information from each of the points in the image. Although, even with the added generalisation it was still unable to correctly classify unique samples that are split between the training, validation and test set, like with previous models.

The convolutional network was not able to outperform the top non-neural network model, XGBoost, on the extracted feature set which it achieved a G-mean loss of 0.3109. But it did manage to obtain better results than that on just the original set of features which it achieved a G-mean loss of 0.3994. This indicates some potential for the network to automatically extra features from the dataset. On the other hand, it was sufficiently able to outperform all three variances of the cascade correlation neural

networks.

However, again the top performer remains the deep neural network which it achieved a G-mean loss of 0.2843 on the test set. Because of the improved performance when using image generation on the convolutional network, a similar data generation was used for the deep network and was reevaluated. But there were no improvements and so the results are omitted.

Chapter 8

Conclusions

Out of all the models presented, the proposed deep neural network obtained the best performance on the test set with a weighted G-mean loss of 0.2843. However, its improvement was achieved through a momentarily 'spike' during training. This is possibly caused by the added noise and dropout layers which altered the training data to ideally match that of the validation set and to an extension the test set. This can be resolved by adding layers to the network so that it will be able to learn the noisy data and to an extension improve validation performance. But, as seen with the cascade networks, it will more likely hinder the network's ability to further generalise to the test set due to unique characteristics.

From the dataset, the temperature values and reference points were only used so that the effectiveness of microwave thermography alone can be evaluated. Despite the small sample of data the network achieved decent results in classifying high or low risk for cancer with just data from microwave thermography. But a recommendation system must consider not only the external conditions of the variance in temperature measurements, but also the physiological condition of each individual recorded by a clinical professional (Zenovich et al., 2016). Additional features in the dataset can be utilised by constructing various models and combining them as ensembles to further improve the results.

One of the issues with the dataset is the small number of both positive and negative classes. The insufficient amount of data hinders any kind of model to effectively generalise to new samples. This was also observed from the experiments where a few unique samples (outliers) were split to the validation and test sets and were unable to be predicted from what information was available in the training set. Using various oversampling techniques and image transformations and rotations helped the models

from overfitting on such a small training set, but not sufficient enough to generalise to this extend. For future work, generative adversarial networks should also be evaluated in comparison to the methods used here because it has shown that it is better for generating samples for bioinformatic type of datasets (S and Thilak Chaminda, 2017).

From the previous research by Zenovich et al. (2016) there are some differences that do not allow for immediate comparison between the results obtained here. Their dataset was split into train and test which it does not accurately represent the unbiased future performance (Russell and Norvig, 2009) and subsequently there were more samples available to training on. Furthermore, they included an additional feature, a pain indicator for each of the mammary glands. There is a good correlation between presence of pain and existence of a tumour which should allow for improved results. It was not included so that solely the effectiveness of microwave thermography can be determined.

The model results presented in *section 5* had their hyperparameters optimised using the tree-structured Parzen estimator algorithm (Bergstra et al., 2011) implemented in the hyperopt library. The domain spectrum of each of the hyperparameters are represented as a probabilistic distribution based on a maintained history of previous executions. It can obtain good local optimum values but they may not be necessary the global ones depending on the ratio between exploration and exploitation. Nonetheless, the results obtained are still better than that using a random search algorithm or grid search both in aspects of time to tune and the final results obtained (Bergstra et al., 2011).

Currently, the models presented when given a new sample they can make a classification prediction of low or high risk with an associated confidence percentage. A clinical professional can follow up with further examination for cases that result to high risk prediction. However, the information returned is very limited and gives no insight on its reasoning. Clinical or medical professionals require additional information to evaluate the results from the model and make informed decisions and determine next steps, as for the models act as a supporting tool.

Another possible extension is to expand the problem from a binary to a multi-class one. This though is only feasible when sufficiently descriptive and large amount of data is collected. When such data becomes available, the healthy and sick classes can be expanded to subcategories. The classes can be split to benign, malignant or various noncancerous tumours, inflammation, infection and no pathological indication. While this makes for a much more difficult machine learning task, it does offer more

information to professionals.

The improvements proposed for the cascade correlation network can be further extended. An additional combination of settings can be used where the model will try against various activation functions. This will result to a non homogeneous model. This can be further improved to do dynamic hyperparameter tuning for each of the hidden layers which will be based on a grid search method. However, this will require proportionally more computational resources and will increase the training time of the network.

Appendix A

Model Architectures

A.1 Deep Neural Network

Deep neural network's architecture used in experiments:

1. Input layer
2. Batch normalisation layer
3. Gaussian noise layer with standard deviation of 0.2
4. Dense layer with 1000 units
5. Batch normalisation layer
6. Dropout layer with 20% of the units dropped
7. Gaussian noise layer with standard deviation of 0.2
8. Dense layer with 200 units
9. Batch normalisation layer
10. Dropout layer with 20% of the units dropped
11. Gaussian noise layer with standard deviation of 0.2
12. Dense layer with 200 units
13. Batch normalisation layer
14. Dropout layer with 20% of the units dropped
15. Gaussian noise layer with standard deviation of 0.2
16. Dense layer with 200 units
17. Batch normalisation layer
18. Dropout layer with 20% of the units dropped
19. Dense layer with 200 units
20. Dense output layer with 2 units

A.2 Convolutional Neural Network

Convolutional neural network's architecture used in experiments:

1. Input layer
2. Convolutional with 64 units, kernel size of 3x3, stride of 1, padding same and no bias
3. Batch normalisation
4. ReLU activation function
5. Convolutional with 64 units, kernel size of 3x3, stride of 1, padding same and no bias
6. Batch normalisation
7. ReLU activation function
8. Gaussian noise with standard deviation of 0.01
9. Max pooling with a pool size of 3x3 and padding same
10. Convolutional with 128 units, kernel size of 3x3, stride of 1, padding same and no bias
11. Batch normalisation
12. ReLU activation function
13. Convolutional with 128 units, kernel size of 3x3, stride of 1, padding same and no bias
14. Batch normalisation
15. ReLU activation function
16. Gaussian noise with standard deviation of 0.001
17. Max pooling with a pool size of 3x3 and padding same
18. Convolutional with 256 units, kernel size of 3x3, stride of 1, padding same and no bias
19. Batch normalisation
20. ReLU activation function
21. Convolutional with 256 units, kernel size of 3x3, stride of 1, padding same and no bias
22. Batch normalisation
23. ReLU activation function
24. Gaussian noise with standard deviation of 0.001

25. Spatial dropout of 20%
26. Max pooling with a pool size of 3x3 and padding same
27. ReLU activation function
28. Separable convolution with 512 units, kernel size of 3x3, stride of 2, padding same and no bias
29. Gaussian noise with standard deviation of 0.0001
30. Batch normalisation
31. ReLU activation function
32. Separable convolution with 512 units, kernel size of 3x3, stride of 2, padding same and no bias
33. Gaussian noise with standard deviation of 0.0001
34. Batch normalisation
35. ReLU activation function
36. Separable convolution with 512 units, kernel size of 3x3, stride of 2, padding same and no bias
37. Gaussian noise with standard deviation of 0.0001
38. Batch normalisation
39. ReLU activation function
40. Separable convolution with 512 units, kernel size of 3x3, stride of 2, padding same and no bias
41. Gaussian noise with standard deviation of 0.0001
42. Batch normalisation
43. ReLU activation function
44. Separable convolution with 512 units, kernel size of 3x3, stride of 2, padding same and no bias
45. Gaussian noise with standard deviation of 0.0001
46. Batch normalisation
47. ReLU activation function
48. Global average pool
49. Dense layer with 512 units
50. Batch normalisation
51. ReLU activation function

52. Dropout with 20%
53. Gaussian noise with standard deviation of 0.1
54. Dense output layer with 2 units

Appendix B

Source Code

B.1 Cascade Correlation Neural Network

```
#!/usr/bin/env python
import tensorflow as tf
from tensorflow.python.keras.models import Sequential, Model, load_model
from tensorflow.python.keras.layers import Input, InputLayer, Dense, concatenate
from tensorflow.python.keras.layers import Dropout, GaussianNoise, BatchNormalization
from tensorflow.python.keras.utils import to_categorical, plot_model
from tensorflow.python.keras import callbacks
from tensorflow.python.keras import initializers, regularizers, optimizers
from tensorflow.python.keras import backend as K
from sklearn.utils import class_weight
import matplotlib.pyplot as plt
import numpy as np
import itertools
import threading
import warnings
import os


class CascadeCorrelation(object):

    def __init__(self, one_hot_output=True, num_classes=2):
        """ Class initialiser

         :param one_hot_output: Boolean, whether to do one-hot encoding
         of the output classes (default: False)
         :param num_classes: int, the number of unique classes in the
         dataset (default: 2)
         """
        self.model = None
        self.hidden_models = []
        self.one_hot_output = one_hot_output
        self.num_classes = num_classes

    @staticmethod
    def geometric_loss(y_true, y_pred):
        """ Calculates the batch-wise geometric mean loss
```

```
        :param y_true: Actual output classes
        :param y_pred: Predicted output classes
        :return: float, geometric loss
        """
        def sensitivity(y_true, y_pred):
            """
              Calculates the batch-wise sensitivity
              """
            true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
            possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
            sensitivity = true_positives / (possible_positives + K.epsilon())

            return K.maximum(sensitivity, K.epsilon())

        def specificity(y_true, y_pred):
            """
              Calculates the batch-wise specificity
              """
            true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
            possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
            specificity = true_negatives / (possible_negatives + K.epsilon())

            return K.maximum(specificity, K.epsilon())

        y_true = y_true[:, 1]
        y_pred = y_pred[:, 1]

        return 1 - K.sqrt(sensitivity(y_true, y_pred) * specificity(y_true, y_pred))

    @staticmethod
    def correlation_coefficient_loss(y_true, y_pred):
        """ Calculates the Pearson correlation coefficient loss

        :param y_true: Actual output classes
        :param y_pred: Predicted output classes
        :return: Loss value
        """
        mx = K.mean(y_true)
        my = K.mean(y_pred)
        xm, ym = y_true-mx, y_pred-my
        r_num = K.sum(tf.multiply(xm,ym))
        r_den = K.sqrt(K.maximum(tf.multiply(K.sum(K.square(xm)), K.sum(K.square(ym))), K.constant(1e-07)))
        r = r_num / (r_den + K.constant(1e-07))
        r = K.maximum(K.minimum(r, 1.0), -1.0)
        return 1 - K.square(r)

    @staticmethod
    def __getInitialiser(initialiser='rand_normal'):
        """ Returns the initialisation function - used to also obtain identical
              parameters throughout the model

        :param initialiser: Valid inputs: 'rand_normal', 'rand_uniform',
        'glorot_normal', 'glorot_uniform', 'lecun_normal', 'lecun_uniform',
        'he_normal' and 'he_uniform' (default: 'rand_normal')
        :return: Initialiser
        """
        if initialiser == 'rand_normal':
            return initializers.RandomNormal(mean=.0, stddev=0.5)
```

```
        elif initialiser == 'rand_uniform':
            return initializers.RandomUniform(minval=-0.5, maxval=0.5)
        elif initialiser == 'glorot_normal':
            return initializers.glorot_normal()
        elif initialiser == 'glorot_uniform':
            return initializers.glorot_uniform()
        elif initialiser == 'lecun_normal':
            return initializers.lecun_normal()
        elif initialiser == 'lecun_uniform':
            return initializers.lecun_uniform()
        elif initialiser == 'he_normal':
            return initializers.he_normal()
        elif initialiser == 'he_uniform':
            return initializers.he_uniform()
        else:
            warnings.warn('Invalid input string ("' + initialiser + '") for weight initialisation.' +
                          'Defaulting to "rand_normal"')
            return initializers.RandomNormal(mean=.0, stddev=0.5)

    @staticmethod
    def __getOptimiser(optimiser='sgd', lr=None):
        """ Returns the optimsation function - used to also obtain identical
            parameters throughout the model

         :param optimiser: Valid inputs: 'sgd', 'rmsprop', 'adagrad', 'adadelta',
         'adamax' and 'adam' (default: 'sgd')
         :param lr: Learning rate to use for the optimiser. None to use predefined
         value (default: None)
         :return: Optimiser function
         """
        if optimiser == 'sgd':
            return optimizers.SGD(lr=0.000005) if lr is None else optimizers.SGD(lr=lr)
        elif optimiser == 'rmsprop':
            return optimizers.RMSprop(lr=0.000005) if lr is None else optimizers.RMSprop(lr=lr)
        elif optimiser == 'adagrad':
            return optimizers.Adagrad(0.000005) if lr is None else optimizers.Adagrad(lr=lr)
        elif optimiser == 'adadelta':
            return optimizers.Adadelta(0.000005) if lr is None else optimizers.Adadelta(lr=lr)
        elif optimiser == 'adamax':
            return optimizers.Adamax(0.000005) if lr is None else optimizers.Adamax(lr=lr)
        elif optimiser == 'adam':
            return optimizers.Adam(lr=0.000005) if lr is None else optimizers.Adam(lr=lr)
        else:
            warnings.warn('Invalid input string ("' + optimiser + '") for optimiser.' +
                          'Defaulting to "sgd"')
            return optimizers.SGD(0.000005) if lr is None else optimizers.SGD(lr=lr)

    @staticmethod
    def __gaussianNoiseLayer(name='GaussianNoiseLayer'):
        """ Returns a Gaussian noise layer  - used to also obtain an identical
            layer when reconstructing

         :param name: The layer's name (default: 'GaussianNoiseLayer')
         :return: Gaussian noise layer
         """
        return GaussianNoise(0.5, name=name)

    @staticmethod
    def __batchNormalizationLayer(trainable=True, name='BatchNormalizationLayer'):
```

```
        """ Returns a batch normalisation layer  - used to also obtain an identical
            layer when reconstructing

         :param trainable: Whether the weights can be trained (default: True)
         :param name: The layer's name (default: 'BatchNormalizationLayer')
         :return: Batch normalisation layer
         """
        return BatchNormalization(epsilon=1e-5, trainable=trainable, name=name)

    @staticmethod
    def __dropoutLayer(name='DropoutLayer'):
        """ Returns a dropout layer - used to also obtain an identical layer when
            reconstructing

         :param name: The layer's name (default: 'DropoutLayer')
         :return: Dropout layer
         """
        return Dropout(rate=0.5, name=name)

    @staticmethod
    def __getBalancedClassWeights(y):
        """ Calculates the weights for each of the classes so that when passed to
            the training algorithm they will be treated with equal importance

         :param y: Actual output classes
         :return: Dictionary, key: class label, value: weight
         """
        # If one-hot encoding of y, then transform it back to a single column
        # array
        if y.ndim == 2:
            y = [y_row.argmax() for y_row in y]

        class_weights = class_weight.compute_class_weight('balanced',
                                                          np.unique(y),
                                                          y)
        return dict(enumerate(class_weights))

    def __hiddenLayerCandidate(self, x_train, y_train, x_val, y_val, units,
                               outputDim, inputDim, cache, index, batch=10,
                               epochs=100, weightInitialiser='rand_normal',
                               optimiser='sgd', activation='sigmoid', l1=0.,
                               l2=0., dropout=False, noise=False, norm=False):
        """ Training session for an individual hidden layer. Tt will be trained
            using correlation coefficient loss function based on the residual
            error of the output layer. Based on the settings passed a hidden
            layer will have the following format: noise, dense, normalisation
            and dropout

         :param x_train: Training feature set
         :param y_train: Training actual output classes
         :param x_val: Validation feature set
         :param y_val: Validation actual output classes
         :param units: Number of hidden layer units
         :param outputDim: Number of output units
         :param inputDim: The input size
         :param cache: Dictionary, in which the output of the thread is stored
         :param index: The threads index
         :param batch: Batch size for training (deafult: 10)
         :param epochs: The maximum epochs to execute on (default: 100)
```

```
    :param weightInitialiser: Which weight initialisation method to use
    (default: 'sgd')
    :param optimiser: Which optimiser to use (default: 'rand_normal')
    :param activation: Which activation function to use for the hidden layer
    (default: 'sigmoid')
    :param l1: Amount of l1 regularisation for the hidden layer (default: 0.0)
    :param l2: Amount of l2 regularisation for the hidden layer (default: 0.0)
    :param dropout: Whether to use a dropout layer (default: False)
    :param noise: Whether to use a Gaussian noise layer (default: False)
    :param norm: Whether to use a batch normalisation layer (default: False)
    :return: Dictionary, with results, weights and model/layers
    """
    with tf.Session(graph = tf.Graph()):
        layers = []
        # Built a model only for the hidden layer to train and obtain optimal weights
        model = Sequential()
        model.add(InputLayer(input_shape=(inputDim,), name='InputLayer'))

        if noise:
            model.add(self.__gaussianNoiseLayer())
            layers.append('noise')

        if l1 > 0 or l2 > 0:
            model.add(Dense(outputDim, input_shape=(inputDim,), activation=activation,
                            kernel_initializer=self.__getInitialiser(weightInitialiser),
                            kernel_regularizer=regularizers.L1L2(l1=l1, l2=l2),
                            name='DenseLayer'))
        else:
             model.add(Dense(outputDim, input_shape=(inputDim,), activation=activation,
                            kernel_initializer=self.__getInitialiser(weightInitialiser),
                            name='DenseLayer'))
        layers.append('dense')

        if norm:
            model.add(self.__batchNormalizationLayer())
            layers.append('norm')

        if dropout:
            model.add(self.__dropoutLayer())
            layers.append('dropout')

        model.compile(optimizer=self.__getOptimiser(optimiser), loss=self.correlation_coefficient_loss)
        history = model.fit(x_train, y_train, validation_data=(x_val, y_val),
                            batch_size=batch, epochs=epochs, verbose=0,
                            callbacks=[callbacks.EarlyStopping(monitor='val_loss',
                                                               min_delta=1e-5,
                                                               patience=200),
                                       callbacks.TerminateOnNaN()])
        loss = model.evaluate(x_val, y_val, batch_size=batch, verbose=0)

        cache[index] = {
            'layers': layers,
            'settings': {'weight': weightInitialiser,
                         'l1': l1,
                         'l2': l2,
                         'optimiser': optimiser},
            'loss': loss,
            'weights': model.get_layer(name='DenseLayer').get_weights(),
            'batchparam': model.get_layer(name='BatchNormalizationLayer').
```

```
                get_weights() if norm else None,
            'train_output': model.predict(x_train),
            'val_output': model.predict(x_val)}

def __outputLayer(self, outputDim, inputDim, weightInitialiser, l1=0., l2=0.,
                  name='DenseLayer'):
    """ Returns the output layer - used to also obtain an identical layer when reconstructing

     :param outputDim: Number of output units
     :param inputDim: Input dimensions
     :param weightInitialiser: The weight initialisation method
     :param l1: Amount of l1 regularisation for the hidden layer (default: 0.0)
     :param l2: Amount of l2 regularisation for the hidden layer (default: 0.0)
     :param name: Name of the output layer (default: 'DenseLayer')
     :return:
     """
    if l1 > 0 or l2 > 0:
        layer = Dense(outputDim, input_shape=(inputDim,), activation='softmax',
                      kernel_initializer=self.__getInitialiser(weightInitialiser),
                      kernel_regularizer=regularizers.L1L2(l1=l1, l2=l2),
                      name=name)
    else:
        layer = Dense(outputDim, input_shape=(inputDim,), activation='softmax',
                      kernel_initializer=self.__getInitialiser(weightInitialiser),
                      name=name)

    return layer

def __getHiddenCandidate(self, x_train, y_train, x_val, y_val, units, outputDim,
                         inputDim, batch, epochs, weightInitialiser, optimiser,
                         activation, l1=0., l2=0., dropout=False, noise=False,
                         norm=False, poolSize=10, regularisationSearch=False):
    """ Returns the best hidden layer from the candidate pool. Each of the hidden
          layers in the pool will be executed in an individual thread with the passed
          arguments

     :param x_train: Training feature set
     :param y_train: Training actual output classes
     :param x_val: Validation feature set
     :param y_val: Validation actual output classes
     :param units: Number of hidden layer units
     :param outputDim: Number of output units
     :param inputDim: The input size
     :param batch: Batch size for training
     :param epochs: The maximum epochs to execute on
     :param weightInitialiser: Which weight initialisation method to use
     :param optimiser: Which optimiser to use
     :param activation: Which activation function to use for the hidden layer
     :param l1: Amount of l1 regularisation for the hidden layer (default: 0.0)
     :param l2: Amount of l2 regularisation for the hidden layer (default: 0.0)
     :param dropout: Whether to use a dropout layer (default: False)
     :param noise: Whether to use a Gaussian noise layer (default: False)
     :param norm: Whether to use a batch normalisation layer (default: False)
     :param poolSize: Determines the pool size - how many times a hidden layer
     will be executed independently (default: 10)
     :param regularisationSearch: Whether to conduct all possible combinations
     between dropout, noise and batch
     normalisation layers. If true the pool size becomes the number of
     executions for each individual combination
```

```
        :return: Best candidate hidden layer
        """
        threads = []
        cacheCandidatePool = {}

        if regularisationSearch:
            threadIdx = 0
            for reg in itertools.product([False, True], [False, True], [False, True]):
                for i in range(poolSize):
                    thread = threading.Thread(target=self.__hiddenLayerCandidate,
                                              args=(x_train, y_train, x_val,
                                                    y_val, units, outputDim,
                                                    inputDim, cacheCandidatePool,
                                                    threadIdx, batch, epochs,
                                                    weightInitialiser, optimiser,
                                                    activation, l1, l2, reg[0],
                                                    reg[1], reg[2]))
                    threads.append(thread)
                    thread.start()
                    threadIdx += 1
        else:
            for threadIdx in range(poolSize):
                thread = threading.Thread(target=self.__hiddenLayerCandidate,
                                          args=(x_train, y_train, x_val,
                                                y_val, units, outputDim,
                                                inputDim, cacheCandidatePool,
                                                threadIdx, batch, epochs,
                                                weightInitialiser, optimiser,
                                                activation, l1, l2, dropout,
                                                noise, norm))
                threads.append(thread)
                thread.start()

        for t in threads:
            t.join()

        return min(cacheCandidatePool.values(), key=lambda x: x['loss'])

    def __appendHiddenLayer(self, model_layers, model_layer_weights, inputLayer,
                            selectedLayer, outputUnits, layerNum, activation,
                            l1=0., l2=0.):
        """ Reconstructs the candidate hidden layer and appends it to the passed model.
              Sets the weights found in the candidate layer

        :param model_layers: List of layers added thus far
        :param model_layer_weights: Respective weights if the layers in the list
        'model_layers'
        :param inputLayer: Input layer for the candidate hidden layer (contains
        original input and previous hidden layers' outputs)
        :param selectedLayer: Best candidate hidden layer
        :param outputUnits: Number of output units
        :param layerNum: The id number of the layer in the model
        :param activation: Which activation function used
        :param l1: Amount of l1 regularisation for the hidden layer (default: 0.0)
        :param l2: Amount of l2 regularisation for the hidden layer (default: 0.0)
        :return: Concatenated inputLayer
        """
        model_layers.append(inputLayer)
        # Only used to pass weights to model and not to recreate it
```

```
        model_layer_weights.append(None)
        l = inputLayer

        for layer in selectedLayer['layers']:
            if layer == 'noise':
                l = self.__gaussianNoiseLayer(name='Gaussian_Noise_Hidden_Layer_' + str(layerNum))(l)
                model_layers.append(l)
                model_layer_weights.append(None)
            elif layer == 'dense':
                if l1 > 0 or l2 > 0:
                    l = Dense(outputUnits,
                              activation=activation,
                              trainable=False,
                              kernel_regularizer=regularizers.L1L2(l1=l1, l2=l2),
                              name='Dense_Hidden_Layer_' + str(layerNum))(l)
                else:
                    l = Dense(outputUnits,
                              activation=activation,
                              trainable=False,
                              name='Dense_Hidden_Layer_' + str(layerNum))(l)
                model_layers.append(l)
                model_layer_weights.append(selectedLayer['weights'])
            elif layer == 'norm':
                l = self.__batchNormalizationLayer(trainable=False,
                                                   name='Batch_Normalisation_Hidden_Layer_' + str(layerNum))(l)
                model_layers.append(l)
                model_layer_weights.append(selectedLayer['batchparam'])
            elif layer == 'dropout':
                l = self.__dropoutLayer(name='Dropout_Hidden_Layer_' + str(layerNum))(l)
                model_layers.append(l)
                model_layer_weights.append(None)

        return concatenate([inputLayer, l], name='Concatenate_Layer_' + str(layerNum))

    def fit(self, x_train, y_train, x_val, y_val, batch=50, hiddenEpochs=100,
            epochs=100, poolSize=10, warmStart=True, weightInitialiser='rand_normal',
            lr=0.1, l1=0., l2=0., hiddenActivation='sigmoid', optimiser='sgd',
            hiddenOptimiser='sgd', regularisationSearch=False, delta=1e-4,
            patience=5, minIterations=None, verbose=0):
        """ Builds dynamically a neural network based on the cascade correlation neural
              network.  The training data is used to train the weights of the model and
              the validation data to determine the network's number of hidden layers and
              for early stopping.

          :param x_train: Training feature set
          :param y_train: Training actual output classes
          :param x_val: Validation feature set
          :param y_val: Validation actual output classes
          :param batch: Batch size for training (default: 50)
          :param hiddenEpochs: The maximum number of epochs for each candidate hidden
          layer (default: 100)
          :param epochs: The maximum number of epochs for the output layer (default: 100)
          :param poolSize: Number of hidden candidate layers for each iteration
          (default: 10)
          :param warmStart: Whether to store the optimal hidden layer's weights for the
          following iterations (default: True)
          :param weightInitialiser: Which weight initialisation method to use. Valid
          inputs: 'rand_normal', 'rand_uniform', 'glorot_normal', 'glorot_uniform',
          'lecun_normal', 'lecun_uniform', 'he_normal' and 'he_uniform'
```

```
 (default: 'rand_normal')
 :param lr: Learning rate for the layers (default: 0.1)
 :param l1: Amount of l1 regularisation for the layers (default: 0.0)
 :param l2: Amount of l2 regularisation for the layers (default: 0.0)
 :param hiddenActivation: Which activation function to use for the candidate
 hidden layers (default: 'sigmoid')
 :param optimiser: Which optimisation function to use for the output layer.
 Valid inputs: 'sgd', 'rmsprop', 'adagrad', 'adadelta', 'adamax' and 'adam'
 (default: 'sgd')
 :param hiddenOptimiser: Which optimisation function to use for the candidate
 hidden layers. valid inputs: 'sgd', 'rmsprop', 'adagrad', 'adadelta', 'adamax'
 and 'adam' (default: 'sgd')
 :param regularisationSearch: Whether to conduct all possible combinations of
 Gaussian noise, dropout and batch normalisation for the candidate hidden layers
 (default: False)
 :param delta: Minimum error difference (default: 1e-4)
 :param patience: Number of iterations allowed to find a better model before
 terminating through early stopping (default: 5)
 :param minIterations: Number of iterations allowed for the network before
 patience counter starts (default: None)
 :param verbose: Whether to print progress output (default: 0)
 :return:
 """
trainLossResults = []
valLossResults = []
trainAccResults = []
valAccResults = []
trainGResults = []
valGResults = []

minLossVal = float('inf')
curPatience = patience
warmStartWeights = None
hiddenFeatTrain = np.copy(x_train)
hiddenFeatVal = np.copy(x_val)
hiddenLayerCount = 0

if self.one_hot_output:
    y_train = to_categorical(y_train, self.num_classes)
    y_val = to_categorical(y_val, self.num_classes)

classWeights = self.__getBalancedClassWeights(y_train)

self.model = None

model_layers = []
model_layer_weights = []

inputLayer = Input(shape=(x_train.shape[1],), name='Input_Layer')
dataInputLayer = inputLayer

optimalLayerCount = 0
optimalLayers = 0
optimalOuputLayerWeights = None
finalInputLayer = inputLayer

outputDimOutputLayer = hiddenLayerUnits = y_train.shape[1]

layerCounter = 0
```

```
# Start dynamic hidden layer addition and only terminate when there
# is no more improvement to the model
while True:
    if verbose == 1:
        print('run: ', layerCounter)

    inputDimOutputLayer = hiddenFeatTrain.shape[1]

    # Create a temp output layer, which will have input from all previous
    # layers
    outputModel = Sequential()
    outputModel.add(self.__outputLayer(outputDimOutputLayer, inputDimOutputLayer,
                                       weightInitialiser, l1, l2))
    outputModel.compile(optimizer=self.__getOptimiser(optimiser, lr),
                        loss='categorical_crossentropy',
                        metrics=['accuracy', self.geometric_loss])
                        # Use of g_loss as a metric instead of a loss function
                        # because it isn't differentiable

    # Add the weights from the previous training in addition to the
    # weights of the new inputs
    if warmStart and warmStartWeights:
        weights = warmStartWeights[0]
        bias = warmStartWeights[1]
        newWeights = outputModel.layers[-1].get_weights()
        newWeights[0][:len(weights)] = weights
        newWeights[1] = bias
        outputModel.set_weights(newWeights)
    # Training the output layer to determine if execution should continue
    # and find the residual error for the candidate hidden layers
    history = outputModel.fit(x=hiddenFeatTrain,
                    y=y_train,
                    validation_data=(hiddenFeatVal, y_val),
                    batch_size=batch,
                    epochs=epochs,
                    verbose=verbose,
                    class_weight=classWeights,
                    callbacks=[callbacks.EarlyStopping(monitor='val_geometric_loss',
                                                       mode='min',
                                                       min_delta=1e-5,
                                                       patience=200),
                               callbacks.TerminateOnNaN()])

    if verbose >= 2:
        print(history.history['val_geometric_loss'])
        plt.plot(history.history['val_geometric_loss'])
        plt.show()
    # Evaluate the training and validation results to be used as part
    # of the output history for each iteration. Additionally, calcualtes
    # the residual error which will be fed to the candidate hidden layers
    predTrain = outputModel.predict_proba(hiddenFeatTrain)
    predVal = outputModel.predict_proba(hiddenFeatVal)
    residualTrain = y_train - predTrain
    residualVal = y_val - predVal
    lossTrain, accTrain, gLossTrain = outputModel.evaluate(hiddenFeatTrain, y_train, verbose=0)
    lossVal, accVal, gLossVal = outputModel.evaluate(hiddenFeatVal, y_val, verbose=0)

    trainLossResults.append(lossTrain)
    valLossResults.append(lossVal)
```

```
trainAccResults.append(accTrain)
valAccResults.append(accVal)
trainGResults.append(gLossTrain)
valGResults.append(gLossVal)
if verbose >= 1:
    print(accTrain, accVal)
    print(lossTrain, lossVal)
    print(gLossTrain, gLossVal)

# If there is no more or very small decreaese in validation loss
# then terminate dynamic training with some level of patience, if
# future iterations do not result to an improvement
if gLossVal + delta >= minLossVal:
    if minIterations is None:
        curPatience -= 1
    else:   # Allow for some additional explorative period before
            # starting the patient's variable count down
        minIterations -= 1
        if minIterations <= 0:
            minIterations = None

    if curPatience <= 0:   # Truly terminate
        # Recreate the model based on the stored hidden layers and
        # the final output layer. The model's layer will have its
        # weights set based on the weights stored during training
        outputLayer = self.__outputLayer(outputDimOutputLayer,
                                         None,
                                         weightInitialiser,
                                         l1, l2,
                                         name='Output_Layer')(finalInputLayer)
        self.model = Model(inputs=dataInputLayer, outputs=outputLayer)
        # Go through each individual layer and set its weights,
        # based on the stored ones, if applicable
        for layeridx in range(optimalLayers):
            if model_layer_weights[layeridx] is None:
                continue
            self.model.layers[layeridx].set_weights(model_layer_weights[layeridx])
            self.model.layers[layeridx].trainable = False

        self.model.layers[-1].set_weights(optimalOuputLayerWeights)
        self.model.compile(optimizer=self.__getOptimiser(optimiser, lr),
                           loss='categorical_crossentropy',
                           metrics=['accuracy', self.geometric_loss])
        break
else:   # If better validation geometric mean loss is obtained the store
        # the output layer's weights and
        # reference points
    if minIterations is not None:
        minIterations -= 1
        if minIterations <= 0:
            minIterations = None
    # Store the current hidden layers' weights that resulted to
    # the current best results
    if warmStart:
        warmStartWeights = outputModel.get_weights()

    minLossVal = gLossVal
    curPatience = patience
```

```
            optimalLayers = len(model_layers)
            optimalLayerCount = layerCounter
            optimalOuputLayerWeights = outputModel.layers[-1].get_weights()
            finalInputLayer = inputLayer

        # Trains all the candiadte hidden layers in the pool
        # Then returns the model that results to the smallest validation
        # geometric loss value from the pool
        hiddenModel = self.__getHiddenCandidate(hiddenFeatTrain, residualTrain,
                                                hiddenFeatVal, residualVal,
                                                hiddenLayerUnits,
                                                outputDimOutputLayer,
                                                inputDimOutputLayer, batch,
                                                hiddenEpochs, weightInitialiser,
                                                optimiser=hiddenOptimiser,
                                                activation=hiddenActivation,
                                                l1=l1, l2=l2,
                                                poolSize=poolSize,
                                                regularisationSearch=regularisationSearch)

        hiddenLayerCount += 1

        if verbose == 1:
            print('hidden layer(s):')
            print(hiddenModel['layers'])

        # Adds the hidden layer to the model. It is concatenated with the previous
        # input and hidden layers, allowing it to be readily used for the next
        # iterations output layer and when reconstructing the complete model
        inputLayer = self.__appendHiddenLayer(model_layers, model_layer_weights, inputLayer,
                     hiddenModel, hiddenLayerUnits, hiddenLayerCount, hiddenActivation,
                     l1=l1, l2=l2,)

        # Concatenate the newly added hidden layer's outputs to the set for the next layer
        # Cache the results from the previous layers so it can be directly
        # fed to the temp output layer
        hiddenFeatTrain = np.concatenate([hiddenFeatTrain, hiddenModel['train_output']], axis=1)
        hiddenFeatVal = np.concatenate([hiddenFeatVal, hiddenModel['val_output']], axis=1)
        layerCounter += 1

    return {'Train Loss': trainLossResults, 'Val Loss': valLossResults,
            'Train Acc': trainAccResults, 'Val Acc': valAccResults,
            'Train G-Loss': trainGResults, 'Val G-Loss': valGResults,
            'layers': optimalLayerCount}

def saveModel(self, directory, fileName, overwrite=True):
    """ Saves the model to file. Will be saved with  a '.h5' extension

     :param directory: String, directory of the folder to store the file in
     :param fileName: String, name of the model file
     :param overwrite: Boolean, whether to overwrite any existing saved model
     (default: True)
     :return: True if successful, otherwise False
     """
    if self.model is None:
        return False

    os.makedirs(directory, exist_ok=True)
    if not fileName.endswith('.h5'):
```

```
            fileName = fileName + '.h5'
        self.model.save(os.path.join(directory, fileName), overwrite)

        return True

    def loadModel(self, directory, fileName):
        """ Loads a stored model from file. The file must end with a '.h5' extension

         :param directory: String, directory of the folder to load the file from
         :param fileName: String, name of the model file
         :return: True if the model is loaded successfully, otherwise False
         """
        if not fileName.endswith('.h5'):
            fileName = fileName + '.h5'

        filepath = os.path.join(directory, fileName)
        if os.path.isfile(filepath):
            self.model = load_model(filepath, custom_objects={'geometric_loss': self.geometric_loss})
            return True

        return False

    def predict(self, X, batch=50):
        """ Predicts the class outputs based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class predictions or None if no model fitted
         """
        if self.model is not None:
            return np.argmax(self.model.predict(X, batch_size=batch), axis=1)

        return None

    def predict_proba(self, X, batch=50):
        """ Predicts the class probabilities based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class probabilities or None if no model fitted
         """
        if self.model is not None:
            return self.model.predict(X, batch_size=batch)[:, 1]

        return None

    def evaluate(self, X, y):
        """ Predicts the passed feature set 'X' and produces statistics based on
             the actual outputs 'y'. Obtains the accuracy, categorical loss value
             and geometric mean loss

         :param X: Feature set
         :param y: Actual output classes
         :return: Evaluations or None if model not fitted
         """
        if self.model is not None:
            if self.one_hot_output:
                y = to_categorical(y, self.num_classes)
            return self.model.evaluate(X, y, batch_size=None, verbose=0, sample_weight=None)
```

```
        return None

    def plotModel(self, directory, fileName='model', showShapes=False,
                  showLayerNames=True, rankdir='TB'):
        """ Plots and saves the constructed model to file

         :param directory: String, the directory of the output folder
         :param fileName: String, output file name (default: 'model')
         :param showShapes: Boolean, whether to display shape information
         (default: False)
         :param showLayerNames: Boolean, whether to display layer names
         (default: True)
         :param rankdir: String, indicates whether to create a vertical
         ('TB') or horizontal ('LR') plot (default: 'TB')
         :return: None
         """
        def printToFile(s):
                with open(os.path.join(directory, fileName + '_summary.txt'),'w+') as f:
                    print(s, file=f)

        if self.model is not None and directory is not None:
            print(self.model.summary())
            self.model.summary(print_fn=printToFile)
            plot_model(self.model, to_file=os.path.join(directory, fileName + '.png'),
                       show_shapes=showShapes, show_layer_names=showLayerNames,
                       rankdir=rankdir)
```

B.2 Deep Neural Network

```
#!/usr/bin/env python
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, GaussianNoise, BatchNormalization
from keras.utils import to_categorical
from keras import callbacks
from keras import optimizers
from keras import backend as K
from sklearn.utils import class_weight
import numpy as np
import os


class DeepNet(object):

    def __init__(self, one_hot_output=False, num_classes=2):
        """ Class initialiser

         :param one_hot_output: Boolean, whether to do one-hot encoding
         of the output classes (default: False)
         :param num_classes: int, the number of unique classes in the
         dataset (default: 2)
         """
        self.one_hot_output = one_hot_output
        self.num_classes = num_classes
        self.model = None
```

```
@staticmethod
def geometric_loss(y_true, y_pred):
    """ Calculates the batch-wise geometric mean loss

     :param y_true: Actual output classes
     :param y_pred: Predicted output classes
     :return: float, geometric loss
     """
    def sensitivity(y_true, y_pred):
        """
          Calculates the batch-wise sensitivity
          """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        sensitivity = true_positives / (possible_positives + K.epsilon())

        return K.maximum(sensitivity, K.epsilon())

    def specificity(y_true, y_pred):
        """
          Calculates the batch-wise specificity
          """
        true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
        possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
        specificity = true_negatives / (possible_negatives + K.epsilon())

        return K.maximum(specificity, K.epsilon())

    y_true = y_true[:, 1]
    y_pred = y_pred[:, 1]

    return 1 - K.sqrt(sensitivity(y_true, y_pred) * specificity(y_true, y_pred))

@staticmethod
def getBalancedClassWeights(y):
    """ Calculates the weights for each of the classes so that when passed
          to the training algorithm they will be treated with equal importance

     :param y: Actual output classes
     :return: Dictionary, key: class label, value: weight
     """
    # If one-hot encoding of y, then transform it back to a single column
    # array
    if y.ndim == 2:
        y = [y_row.argmax() for y_row in y]

    class_weights = class_weight.compute_class_weight('balanced',
                                                      np.unique(y),
                                                      y)
    return dict(enumerate(class_weights))

def fit(self, x_train, y_train, x_val, y_val):
    """ Trains the weights of the model based on the training data. Uses the
          validation data to find the best weights and do early stopping with
          500 patience. The metric monitored is the validation geometric mean
          loss. Executes for a maximum of 2000 epochs

     :param x_train: Training feature set
```

```
        :param y_train: Training actual output classes
        :param x_val: Validation feature set
        :param y_val: Validation actual output classes
        :return: Epoch history
        """
        if self.one_hot_output:
            y_train = to_categorical(y_train, self.num_classes)
            y_val = to_categorical(y_val, self.num_classes)

        classWeights = self.getBalancedClassWeights(y_train)

        self.model = None

        self.model = Sequential()
        self.model.add(BatchNormalization())

        self.model.add(GaussianNoise(0.2))
        self.model.add(Dense(1000, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(rate=0.2))

        self.model.add(GaussianNoise(0.2))
        self.model.add(Dense(200, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(rate=0.2))

        self.model.add(GaussianNoise(0.2))
        self.model.add(Dense(200, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(rate=0.2))

        self.model.add(GaussianNoise(0.2))
        self.model.add(Dense(200, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(rate=0.2))

        self.model.add(GaussianNoise(0.2))
        self.model.add(Dense(200, activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(rate=0.2))

        self.model.add(Dense(200, activation='relu'))


        self.model.add(Dense(y_train.shape[1], activation='softmax'))
        self.model.compile(optimizer=optimizers.Adam(lr=0.00005),
                            loss='categorical_crossentropy',
                            metrics=['accuracy', self.geometric_loss])

        history = self.model.fit(x=x_train,
                        y=y_train,
                        validation_data=(x_val, y_val),
                        batch_size=50,
                        epochs=2000,
                        verbose=1,
                        class_weight=classWeights,
                        callbacks=[callbacks.EarlyStopping(monitor='val_geometric_loss',
                                                           mode='min',
                                                           min_delta=1e-5,
```

```
                                                    patience=500),
                                callbacks.ModelCheckpoint(
                                        filepath='./tmp_deep_weights.hdf5',
                                        monitor='val_geometric_loss',
                                        save_best_only=True,
                                        save_weights_only=True,
                                        mode='min')])
        # Load the early stopping weights
        self.model.load_weights('./tmp_deep_weights.hdf5')
        os.remove('./tmp_deep_weights.hdf5')

        return history.history

    def saveModel(self, directory, fileName, overwrite=True):
        """ Saves the model to file. Will be saved with  a '.h5' extension

         :param directory: String, directory of the folder to store the file in
         :param fileName: String, name of the model file
         :param overwrite: Boolean, whether to overwrite any existing saved model
         (default: True)
         :return: True if successful, otherwise False
         """
        if self.model is None:
            return False

        os.makedirs(directory, exist_ok=True)
        if not fileName.endswith('.h5'):
            fileName = fileName + '.h5'
        self.model.save(os.path.join(directory, fileName), overwrite)

        return True

    def loadModel(self, directory, fileName):
        """ Loads a stored model from file. The file must end with a '.h5' extension

         :param directory: String, directory of the folder to load the file from
         :param fileName: String, name of the model file
         :return: True if the model is loaded successfully, otherwise False
         """
        if not fileName.endswith('.h5'):
            fileName = fileName + '.h5'

        filepath = os.path.join(directory, fileName)
        if os.path.isfile(filepath):
            self.model = load_model(filepath, custom_objects={'geometric_loss': self.geometric_loss})
            return True

        return False

    def predict(self, X, batch=50):
        """ Predicts the class outputs based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class predictions or None if no model fitted
         """
        if self.model is not None:
            return np.argmax(self.model.predict(X, batch_size=batch), axis=1)
```

```
        return None

    def predict_proba(self, X, batch=50):
        """ Predicts the class probabilities based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class probabilities or None if no model fitted
         """
        if self.model is not None:
            return self.model.predict(X, batch_size=batch)[:, 1]

        return None

    def evaluate(self, X, y):
        """ Predicts the passed feature set 'X' and produces statistics based on the
             actual outputs 'y'. Obtains the accuracy, categorical loss value and
             geometric mean loss

         :param X: Feature set
         :param y: Actual output classes
         :return: Evaluations or None if model not fitted
         """
        if self.model is not None:
            if self.one_hot_output:
                y = to_categorical(y, self.num_classes)
            return self.model.evaluate(X, y, batch_size=None, verbose=0, sample_weight=None)

        return None
```

B.3 Convolutional Neural Network

```
#!/usr/bin/env python
from keras.models import Model, load_model
from keras.layers import Input, Dense, Dropout, GaussianNoise, BatchNormalization
from keras.layers import Activation, Conv2D, SeparableConv2D, MaxPooling2D
from keras.layers import GlobalAveragePooling2D, SpatialDropout2D
from keras.utils import to_categorical
from keras import callbacks
from keras import optimizers
from keras import backend as K
from sklearn.utils import class_weight
import numpy as np
import os


class ConvNet(object):

    def __init__(self, one_hot_output=False, num_classes=2):
        """ Class initialiser

         :param one_hot_output: Boolean, whether to do one-hot encoding of
         the output classes (default: False)
         :param num_classes: int, the number of unique classes in the dataset
         (default: 2)
```

```
        """
        self.one_hot_output = one_hot_output
        self.num_classes = num_classes
        self.model = None

    @staticmethod
    def geometric_loss(y_true, y_pred):
        """ Calculates the batch-wise geometric mean loss

         :param y_true: Actual output classes
         :param y_pred: Predicted output classes
         :return: float, geometric loss
         """
        def sensitivity(y_true, y_pred):
            """
              Calculates the batch-wise sensitivity
              """
            true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
            possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
            sensitivity = true_positives / (possible_positives + K.epsilon())

            return K.maximum(sensitivity, K.epsilon())

        def specificity(y_true, y_pred):
            """
              Calculates the batch-wise specificity
              """
            true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
            possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
            specificity = true_negatives / (possible_negatives + K.epsilon())

            return K.maximum(specificity, K.epsilon())

        y_true = y_true[:, 1]
        y_pred = y_pred[:, 1]

        return 1 - K.sqrt(sensitivity(y_true, y_pred) * specificity(y_true, y_pred))

    @staticmethod
    def getBalancedClassWeights(y):
        """ Calculates the weights for each of the classes so that when
              passed to the training algorithm they will be treated with
              equal importance

         :param y: Actual output classes
         :return: Dictionary, key: class label, value: weight
         """
        # If one-hot encoding of y, then transform it back to a single column
        # array
        if y.ndim == 2:
            y = [y_row.argmax() for y_row in y]

        class_weights = class_weight.compute_class_weight('balanced',
                                                          np.unique(y),
                                                          y)
        return dict(enumerate(class_weights))

    def getModel(self, input_shape, classes):
        """ Returns the convolutional model consisting of convolutional,
```

```
        pooling and separable convolutional blocks with integrated
        batch normalisation and gaussian noise layers

    :param input_shape: vector, shape of the input that will be passed
    to the model with the following format: (rows, columns, channels)
    :param classes: int, the number of unique classes in the dataset
    :return: Model
    """
    K.set_image_dim_ordering('tf')
    #input
    img_input = Input(shape=input_shape, name='input')  # (rows, cols, channels)
    # block 1
    x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(64, (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    # pooling 1
    x = GaussianNoise(0.01)(x)
    x = MaxPooling2D(pool_size=(3, 3), padding='same')(x)
    # block 2
    x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(128, (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    # pooling 2
    x = GaussianNoise(0.001)(x)
    x = MaxPooling2D(pool_size=(3, 3), padding='same')(x)
    # block 3
    x = Conv2D(256, (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(256, (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    # pooling 3
    x = GaussianNoise(0.001)(x)
    x = SpatialDropout2D(0.2)(x)
    x = MaxPooling2D(pool_size=(3, 3), padding='same')(x)
    # separable block 4
    x = Activation('relu')(x)
    x = SeparableConv2D(512, (3, 3), padding='same', use_bias=False)(x)
    x = GaussianNoise(0.0001)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(512, (3, 3), padding='same', use_bias=False)(x)
    x = GaussianNoise(0.0001)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(512, (3, 3), padding='same', use_bias=False)(x)
    x = GaussianNoise(0.0001)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(512, (3, 3), padding='same', use_bias=False)(x)
    x = GaussianNoise(0.0001)(x)
    x = BatchNormalization()(x)
```

```
        x = Activation('relu')(x)
        x = SeparableConv2D(512, (3, 3), padding='same', use_bias=False)(x)
        x = GaussianNoise(0.0001)(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        # pooling 4
        x = GlobalAveragePooling2D()(x)
        # block 5
        x = Dense(512)(x)
        x = BatchNormalization()(x)
        x = Activation('relu')(x)
        x = Dropout(0.2)(x)
        x = GaussianNoise(0.1)(x)
        # output
        x = Dense(classes, activation='softmax')(x)

        # Create model
        model = Model(img_input, x)

        return model

    def fit(self, x_train, y_train, x_val, y_val):
        """ Trains the weights of the model based on the training data. Uses the
             validation data to find the best weights and do early stopping with
             1000 patience. The metric monitored is the validation geometric mean
             loss. Executes for a maximum of 5000 epochs

         :param x_train: Training feature set
         :param y_train: Training actual output classes
         :param x_val: Validation feature set
         :param y_val: Validation actual output classes
         :return: Epoch history
         """
        if self.one_hot_output:
            y_train = to_categorical(y_train, self.num_classes)
            y_val = to_categorical(y_val, self.num_classes)

        classWeights = self.getBalancedClassWeights(y_train)

        self.model = None
        self.model = self.getModel(input_shape=(x_train.shape[1:]),
                                   classes=self.num_classes)

        self.model.compile(optimizer=optimizers.Adam(lr=0.0000005),
                           loss='categorical_crossentropy',
                           metrics=['accuracy', self.geometric_loss])

        history = self.model.fit(x=x_train,
                                 y=y_train,
                                 validation_data=(x_val, y_val),
                                 batch_size=50,
                                 epochs=5000,
                                 verbose=1,
                                 class_weight=classWeights,
                                 callbacks=[callbacks.EarlyStopping(monitor='val_geometric_loss',
                                                                    mode='min',
                                                                    min_delta=1e-5,
                                                                    patience=1000),
                                            callbacks.ModelCheckpoint(
```

```
                                                filepath='./tmp_weights.hdf5',
                                                monitor='val_geometric_loss',
                                                save_best_only=True,
                                                save_weights_only=True,
                                                mode='min')])
        # Load the early stopping weights
        self.model.load_weights('./tmp_weights.hdf5')
        os.remove('./tmp_weights.hdf5')

        return history.history

    def saveModel(self, directory, fileName, overwrite=True):
        """ Saves the model to file. Will be saved with  a '.h5' extension

         :param directory: String, directory of the folder to store the file in
         :param fileName: String, name of the model file
         :param overwrite: Boolean, whether to overwrite any existing saved model
         (default: True)
         :return: True if successful, otherwise False
         """
        if self.model is None:
            return False

        os.makedirs(directory, exist_ok=True)
        if not fileName.endswith('.h5'):
            fileName = fileName + '.h5'
        self.model.save(os.path.join(directory, fileName), overwrite)

        return True

    def loadModel(self, directory, fileName):
        """ Loads a stored model from file. The file must end with a '.h5' extension

         :param directory: String, directory of the folder to load the file from
         :param fileName: String, name of the model file
         :return: True if the model is loaded successfully, otherwise False
         """
        if not fileName.endswith('.h5'):
            fileName = fileName + '.h5'

        filepath = os.path.join(directory, fileName)
        if os.path.isfile(filepath):
            self.model = load_model(filepath, custom_objects={'geometric_loss': self.geometric_loss})
            return True

        return False

    def predict(self, X, batch=50):
        """ Predicts the class outputs based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class predictions or None if no model fitted
         """
        if self.model is not None:
            return np.argmax(self.model.predict(X, batch_size=batch), axis=1)

        return None
```

```
    def predict_proba(self, X, batch=50):
        """ Predicts the class probabilities based on the based input features

         :param X: Feature set
         :param batch: int, batch size of passed input when predicting (default: 50)
         :return: Class probabilities or None if no model fitted
         """
        if self.model is not None:
            return self.model.predict(X, batch_size=batch)[:, 1]

        return None

    def evaluate(self, X, y):
        """ Predicts the passed feature set 'X' and produces statistics based on the
              actual outputs 'y'. Obtains the
              accuracy, categorical loss value and geometric mean loss

         :param X: Feature set
         :param y: Actual output classes
         :return: Evaluations or None if model not fitted
         """
        if self.model is not None:
            if self.one_hot_output:
                y = to_categorical(y, self.num_classes)
            return self.model.evaluate(X, y, batch_size=None, verbose=0, sample_weight=None)

        return None
```

Bibliography

(2017). Benign breast lumps. *WebMD*. https://www.webmd.com/breast-cancer/benign-breast-lumps, Access date: 01/08/2018.

(2018). Anatomy of the female breast information. *myVMC*. Available at https://www.myvmc.com/anatomy/breast/, Access date: 02/08/2018.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association.

Anisimova, E., Zamchnik, T., Losv, A., and E.A., M. (2011). O nekotorykh kharakternykh priznakakh v diagnostike zabolevaniy nizhnikh konechnostey metodom kombinirovannoy termografii [on some characteristic features in the diagnosis of diseases of the lower extremities by a combined thermography]. *Vstnik novykh mditsinskikh tkhnologiy*, 18(2):329–330.

Anisimova, E. V. (2013). Intllktualnyy analiz dannykh i algoritmy klassifikatsii v diagnostik vnoznykh zabolvaniy po dannym kombinirovannoy trmomtrii: avtorf. dis. ... kand. tkhn. nauk [data mining and classification algorithms in the diagnosis of venous diseases according to the combination of thermometry. abstract of diss. and. of technical sciences]. *Volgograd*, page 16.

Anisimova, E. V., Zamchnik, T. V., Losev, A. G., and Mazpa, E. A. (2016). O nkotorykh kharaktrnykh priznakakh v diagnostik zabolvaniy nizhnikh konchnosty mtodom kombinirovannoy trmografii [on some characteristic features in the diagnosis of diseases of the lower extremities by a combined thermography]. *Vstnik*

Volgogradskogo gosudarstvnnogo univrsitta. Sriya 1, Matmatika. Fizika [Science Journal of Volgograd State University. Mathematics. Physics], 18(2):149–160.

Arajo, T., Aresta, G., Castro, E., Rouco, J., Aguiar, P., Eloy, C., Polnia, A., and Campilho, A. (2017). Classification of breast cancer histology images using convolutional neural networks. *PLOS ONE*, 12(6):1–14.

Barandela, R., Snchez, J., Garca, V., and Rangel, E. (2003). Strategies for learning in class imbalance problems. 36:849–851.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA. Curran Associates Inc.

Bergstra, J., Yamins, D., and Cox, D. D. (2015). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Blum, A. L. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1):245 – 271. Relevance.

Bochkarev, O., Zenovich, A., and Losev, A. (2015). Regressionnaya model diagnostiki patologiy molochnykh zhelez po dannym mikrovolnovoy radiotermometrii [regression model for diagnosis of breast pathology according to microwaves radiometry data]. *Vestnik Volgogradskogo gosudarstvennogo universiteta. Seriya 1. Mathematica. Physica*, 6(31):72–82.

Bolomey, J. C., Izadnegahdar, A., Jofre, L., Pichot, C., Peronnet, G., and Solaimani, M. (1982). Microwave diffraction tomography for biomedical applications. *IEEE Transactions on Microwave Theory and Techniques*, 30(11):1998–2000.

Bondar, S. S., Terekhov, I. V., Voevodin, A. A., Leonov, B. I., and Khadartsev, A. A. (2017). Assessment of transcapillary water exchange in the lungs by active radiometry. *Biomedical Engineering*, 51(3):211–214.

Bottou, L. (1991). Stochastic gradient learning in neural networks.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Lechevallier, Y. and Saporta, G., editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg. Physica-Verlag HD.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Chollet, F. (2016). Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357.

Chollet, F. et al. (2015). Keras. https://keras.io.

Cireşan, D. C., Giusti, A., Gambardella, L. M., and Schmidhuber, J. (2013). Mitosis detection in breast cancer histology images with deep neural networks. In Mori, K., Sakuma, I., Sato, Y., Barillot, C., and Navab, N., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 411–418, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cohen, G., Hilario, M., and Geissbhler, A. (2004). Model selection for support vector classifiers via genetic algorithms. an application to medical decision support. pages 200–211.

Conceicao, R., O'Halloran, M., and Mohr, J. (2016). *An Introduction to Microwave Imaging for Breast Cancer Detection*.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3):273–297.

Cover, T. and Hart, P. (2006). Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27.

Crandall, J. P., O, J. H., Gajwani, P., Leal, J. P., Mawhinney, D. D., Sterzer, F., and Wahl, R. L. (2018). Measurement of brown adipose tissue activity using microwave radiometry and 18f-fdg pet/ct. *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*, 59(8):12431248.

de Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67.

Drakopoulou, M., Moldovan, C., Toutouzas, K., and Tousoulis, D. (2018). The role of microwave radiometry in carotid artery disease. diagnostic and clinical prospective. *Current Opinion in Pharmacology*, 39:99 – 104. Cardiovascular and renal.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159.

Fahlman, S. E. (1988). An empirical study of learning speed in backpropagation networks. Technical Report CMU-CS-88-162, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Fahlman, S. E. and Lebiere, C. (1990). Advances in neural information processing systems 2. chapter The Cascade-correlation Learning Architecture, pages 524–532. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Fallahpour, M. (2014). *Synthetic aperture radar-based techniques and reconfigurable antenna design for microwave imaging of layered strutures*. Missouri University of Science and Technology.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

Gabriel, S., Lau, R. W., and Gabriel, C. (1996a). The dielectric properties of biological tissues: Ii. measurements in the frequency range 10 hz to 20 ghz. *Physics in Medicine and Biology*, 41(11):2251.

Gabriel, S., Lau, R. W., and Gabriel, C. (1996b). The dielectric properties of biological tissues: Iii. parametric models for the dielectric spectrum of tissues. *Physics in Medicine and Biology*, 41(11):2271.

Gautherie, M. (1980). Thermopathology of breast cancer: Measurement and analysis of in vivo temperature and blood flow. *Annals of the New York Academy of Sciences*, 335(1):383–415.

Glazunov, V., Znovich, A., and Losv, A. (2015a). Geneticheskiye algoritmy opredeleniya vysokoinformativnykh priznakov zabolevaniy molochnykh zhelez [genetic algorithms for determining highly informative signs of breast diseases]. *Vstnik Volgogradskogo gosudarstvnnogo univrsitta. Sriya 1, Matmatika. Fizika [Science Journal of Volgograd State University. Mathematics. Physics]*, 5(30):72–83.

Glazunov, V. A., Znovich, A. V., and Losev, A. G. (2015b). Gntichski algoritmy oprdlniya vysokoinformativnykh priznakov zabolvaniy molochnykh zhlz [genetic algorithms for determining highly informative signs of breast diseases]. *Vstnik Volgogradskogo gosudarstvnnogo univrsitta. Sriya 1, Matmatika. Fizika [Science Journal of Volgograd State University. Mathematics. Physics]*, 5(30):72–83.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA. MIT Press.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1):389–422.

Hajian-Tilaki, K. (2013). Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. 4:627–635.

Hall, M. A. (1999). Correlation-based feature selection for machine learning. Technical report.

Han, H., Wang, W.-Y., and Mao, B.-H. (2005). Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I*, ICIC'05, pages 878–887, Berlin, Heidelberg. Springer-Verlag.

He, H., Bai, Y., Garcia, E. A., and Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328.

Hecht-Nielsen, R. (1992). Neural networks for perception (vol. 2). chapter Theory of the Backpropagation Neural Network, pages 65–93. Harcourt Brace & Co., Orlando, FL, USA.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Lecture 6a overview of minibatch gradient descent. *Coursera Lecture slides [Online]*.

Hooke, R. and Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *J. ACM*, 8(2):212–229.

Ian, J. (2014). *Principal Component Analysis*. American Cancer Society.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Ivanov, Y., Kozlov, A. F., Galiullin, R. A., Tatur, V. Y., Ziborov, V. S., Ivanova, N. D., Pleshakova, T. O., Vesnin, S. G., and Goryanin, I. (2018). Use of microwave radiometry to monitor thermal denaturation of albumin. *Frontiers in Physiology*, 9:956.

Jemal, A., Simard, E. P., Dorell, C., Noone, A.-M., Markowitz, L. E., Kohler, B., Eheman, C., Saraiya, M., Bandi, P., Saslow, D., Cronin, K. A., Watson, M., Schiffman, M., Henley, S. J., Schymura, M. J., Anderson, R. N., Yankey, D., and Edwards, B. K. (2013). Annual report to the nation on the status of cancer, 19752009, featuring the burden and trends in human papillomavirus (hpv)associated cancers and hpv vaccination coverage levels. *JNCI: Journal of the National Cancer Institute*, 105(3):175–201.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kobrinskiy, B. A. (2008). Konsultativny intllktualny mditsinski sistmy: klassifikatsiya, printsipy postroniya, effktivnost [consulting intelligent medical systems: Classification, principles of construction, efficiency]. *Volgograd*, (2):38–47.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

Kubat, M. and Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann.

Kyber, J., Hansgen, H., and Pliquett, F. (1992). Dielectric properties of biological tissue at low temperatures demonstrated on fatty tissue. *Physics in Medicine and Biology*, 37(8):1675.

Laarhoven, P. J. M. and Aarts, E. H. L., editors (1987). *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA.

Lalkhen, A. G. and McCluskey, A. (2008). Clinical tests: sensitivity and specificity. *Continuing Education in Anaesthesia Critical Care and Pain*, 8(6):221–223.

Laskari, K., Pitsilka, D., Pentazos, G., Siores, E., Tektonidou, M., and Sfikakis, P. (2018). Sat0657 microwave radiometry-derived thermal changes of sacroiliac joints as a biomarker of sacroiliitis in patients with spondyloarthropathy. *Annals of the Rheumatic Diseases*, 77(Suppl 2):1178–1178.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lim, T.-S., Loh, W.-Y., and Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

Losev, A. G., Khoprskov, A. V., Astakhov, A. S., and Sulymanova, K. M. (2015). Problmy izmrniya i modlirovaniya tplovykh i radiatsionnykh poly v biotkanyakh: analiz dannykh mikrovolnovoy radiotrmomtrii [problems of measurement and modeling of thermal and radiation fields in biological tissues: Analysis of microwave thermometry data]. *Vstnik Volgogradskogo gosudarstvnnogo univrsitta. Sriya 1, Matmatika. Fizika [Science Journal of Volgograd State University. Mathematics. Physics]*, 6(31):31–71.

Losev, A. G. and Lvshinskiy, V. V. (2015). Regressionnaya model diagnostiki patologiy molochnykh zhelez po dannym mikrovolnovoy radiotermometrii [regression model for diagnosis of breast pathology according to microwaves radiometry data]. *Vestnik Volgogradskogo gosudarstvennogo universiteta. Seriya 1. Mathematica. Physica [Science Journal of Volgograd State University. Mathematics. Physics]*, 6(31):72–82.

Losev, A. G. and Lvshinskiy, V. V. (2017). Intellektualnyy analiz termometricheskikh dannykh v diagnostike molochnykh zhelez [the thermometry data mining in the diagnostics of mammary glands]. *UBS*, 70:113–135.

Losev, A. G., Znovich, A. V., Bochkarv, O. A., and Lvshinskiy, V. V. (2011). Intllktualnyy analiz mnogomrnykh trmomtrichskikh dannykh v mditsinskoy diagnostik [multidimentional thermometric data mining in medical diagnostics]. *Vstnik novykh mditsinskikh tkhnologiy*, 5(36):329–330.

Mao, J. and Jain, A. K. (1995). Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, 6(2):296–317.

Meaney, P. M., Fanning, M. W., Li, D., Poplack, S. P., and Paulsen, K. D. (2000). A clinical prototype for active microwave imaging of the breast. *IEEE Transactions on Microwave Theory and Techniques*, 48(11):1841–1853.

Myers, P. C., Sadowsky, N. L., and Barrett, A. H. (1979). Microwave thermography: Principles, methods and clinical applications. *Journal of Microwave Power*, 14(2):105–115.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA. ACM.

Ng, E. Y. K., Sree, S. V., Ng, K. H., and Kaw, G. (2008). The use of tissue electrical characteristics for breast cancer detection: A perspective review. *Technology in Cancer Research & Treatment*, 7(4):295–308. PMID: 18642968.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830.

Pentazos, G., Laskari, K., Prekas, K., Raftakis, J., P. Sfikakis, P., and Siores, E. (2018). Microwave radiometry-derived thermal changes of small joints as additional potential biomarker in rheumatoid arthritis: A prospective pilot study. 24:1.

Peronnet, G., Pichot, C., Bolomey, J. C., Jofre, L., Izadnegahdar, A., Szeles, C., Michel, Y., Guerquin-Kern, J. L., and Gautherie, M. (1983). A microwave diffraction tomography system for biomedical applications. In *1983 13th European Microwave Conference*, pages 529–533.

Phienthrakul, T. and Kijsirikul, B. (2005). Evolutionary strategies for multi-scale radial basis function kernels in support vector machines. In *GECCO*.

Pichot, C., Jofre, L., Peronnet, G., and Bolomey, J. (1985). Active microwave imaging of inhomogeneous bodies. *IEEE Transactions on Antennas and Propagation*, 33(4):416–425.

Rodarmel, C. and Shan, J. (2002). Principal component analysis for hyperspectral image classification.

Rodrigues, D. B., Stauffer, P. R., Pereira, P. J. S., and Maccarini, P. F. (2018). *Microwave Radiometry for Noninvasive Monitoring of Brain Temperature*, pages 87–127. Springer International Publishing, Cham.

Rohwer, R. (1989). The 'moving targets' training algorithm. In *Proceedings of the 2Nd International Conference on Neural Information Processing Systems*, NIPS'89, pages 558–565, Cambridge, MA, USA. MIT Press.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

S, S. and Thilak Chaminda, H. (2017). Generate bioinformatics data using generative adversarial network: A review. *Conference: 2nd International Conference on Information Technology Research*.

Schneider, B. P. and Miller, K. D. (2005). Angiogenesis of breast cancer. *Journal of Clinical Oncology*, 23(8):1782–1790. PMID: 15755986.

Semenov, S. (2009). Microwave tomography: Review of the progress towards clinical applications. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 367(1900):3021–3042.

Senior, A., Heigold, G., Ranzato, M., and Yang, K. (2013). An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6724–6728.

Skolnik, M. I. (2018). Radar. *Encyclopdia Britannica*. Available at https://www.britannica.com/technology/radar/History-of-radar, Access date: 27/07/2018.

Spanhol, F. A., Oliveira, L. S., Petitjean, C., and Heutte, L. (2016). Breast cancer histopathological image classification using convolutional neural networks. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2560–2567.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Sudsiri, J., Wachner, D., and Gimsa, J. (2007). On the temperature dependence of the dielectric membrane properties of human red blood cells. *Bioelectrochemistry*, 70(1):134 – 140. Bioelectrochemistry 2005.

Surowiec, A. J., Stuchly, S. S., Barr, J. R., and Swarup, A. (1988). Dielectric properties of breast carcinoma and the surrounding tissues. *IEEE Transactions on Biomedical Engineering*, 35(4):257–263.

Thimm, G. and Fiesler, E. (1995). Neural network initialization. In Mira, J. and Sandoval, F., editors, *From Natural to Artificial Neural Computation*, pages 535–542, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tipa, R. and Baltag, O. (2006). Microwave thermography for cancer detection. *Romanian Journal of Physics*, 51(3-4):371377.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2014). Efficient object localization using convolutional networks. *CoRR*, abs/1411.4280.

Vasconcelos, C. N. and Vasconcelos, B. N. (2017). Increasing deep learning melanoma classification by classical and expert knowledge based image transforms. *CoRR*, abs/1702.07025.

Veltmaat, J., Ramsdell, A., and Sterneck, E. (2013). Positional variations in mammary gland development and cancer. 18(2):179–188.

Vesnin, S., Turnbull, A., Michael Dixon, J., and Goryanin, I. (2017). Modern microwave thermometry for breast cancer. 7.

Wang, X. and Paliwal, K. K. (2003). Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition. *Pattern Recognition*, 36(10):2429 – 2439.

Wilkins, M. F., Boddy, L., Morris, C. W., and Jonker, R. (1996). A comparison of some neural and non-neural methods for identification of phytoplankton from flow cytomery data. *Bioinformatics*, 12(1):9–18.

Zenovich, A. V., Glazunov, V. A., Oparin, A. S., and Primachenko, F. G. (2016). Algoritmy prinyatiya resheniy v konsultativnoy intellektualnoy sisteme diagnostiki molochnykh zhelez [algorithms of decision-making in the advisory intellectual system of diagnostics of mammary glands]. *Mathematical physics and computer modeling*, 6:129–142.