Universiteit Leiden
Opleiding Informatica

Deep Mammography

Applying deep learning to whole-mammogram classification

Name: L.J. Peters

Date: July 10, 2019

Supervisor: dr. W.J. Kowalczyk
Company supervisor: drs. P. ten Have
Second reader: dr. W.A. Kosters

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In the Netherlands, women between the ages of 50 and 75 can be voluntarily screened for breast-cancer. These scans are currently manually examined by radiologists across the country. In this thesis, we explore the possibilities of using well-known neural networks for automatic classification. Using the mammograms and results from the Dutch national breast cancer screening we try to train classifiers for the problem of whole-mammogram breast cancer detection.

Contemporary research uses accuracy and receiver operating characteristic area under curve (ROC-AUC) as a performance measure. However, accuracy is not a useful measure for skewed data sets and AUC is a flawed measure for various reasons. In this paper we discuss a metric based on the sensitivity/specificity trade-off and evaluate our classifiers using this metric.

Using the well-known image classification networks AlexNet and ResNet, we create and evaluate a reusable image processing pipeline using our performance metric.

Contents

1 Introduction **1**
1.1 Mammograms . . . 2

2 Related work **3**
2.1 Mammography . . . 3
2.2 Neural networks . . . 3
2.3 Existing applications of neural networks to mammography . . . 4

3 Neural networks **5**
3.1 Applying a neural network . . . 5
3.2 Error functions . . . 6
3.3 Activation functions . . . 6
3.4 Data encoding . . . 8
3.4.1 Output encoding . . . 8
3.5 Stochastic gradient descent . . . 8
3.5.1 Variants of SGD . . . 9
3.5.2 Overfitting . . . 10
3.6 Convolutional layers . . . 10

4 Data sets **12**
4.1 BI-RADS . . . 12
4.2 iBOB data . . . 12
4.2.1 Classification . . . 13
4.2.2 Non-visual-indications . . . 14
4.3 INBreast data . . . 15

5 Methodology **16**
5.1 Preprocessing . . . 16
5.1.1 Otsu's method . . . 17
5.2 Performance metric . . . 19
5.2.1 Human performance . . . 19
5.3 Networks . . . 19
5.3.1 AlexNet . . . 19
5.3.2 ResNet . . . 20

6 Experiments **22**
6.1 Set-up . . . 22
6.1.1 Software . . . 22
6.1.2 Hardware . . . 23
6.2 Testing the pipeline . . . 23
6.3 Training . . . 23
6.3.1 Training cost . . . 23
6.3.2 Variations . . . 26
6.4 Evaluation . . . 26
6.5 Cross-validation with INbreast . . . 27
6.6 Postprocessing . . . 27

7 Conclusions **29**
7.1 Future work . . . 29

7.2 Acknowledgements 30

Bibliography **31**

A Network architectures **34**
A.1 AlexNet 34
A.2 ResNet 35

Chapter 1

Introduction

In the Netherlands, breast cancer is the most common form of cancer with it approximately 15,000 new cases annually [13]. However, if the disease is caught early enough, the chances of survival are fairly high. Because of this, Bevolkingsonderzoek Nederland invites every woman aged 50 to 75 for a biennial screening. In this screening, two mammograms are taken per breast and both are analysed by two individual radiologists and optionally by a third. Their consensus forms the basis of a recommendation to possibly perform a more invasive check, such as a biopsy. Only then can a definite conclusion be made.

According to Zorginstituut Nederland, one million women are screened for breast cancer every year. These screenings lead to the referral of 24,600 referrals, of which 6,900 cases have breast cancer and 17,700 do not. In addition, each year 2,200 cases of breast cancer are diagnosed in women who participate in the screening programme but who were not referred for further examination at their previous screening.

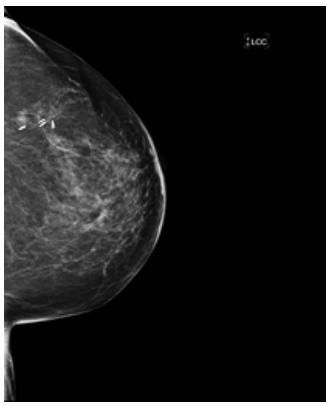
Here we can see three issues. First of all, there is the sheer number of cases to evaluate. Each mammogram needs to be viewed and analysed by at least two different radiologists. Aside from that, the current performance is less than perfect. The majority of women referred do not actually have breast cancer. These women are then commonly subject to a biopsy which is unpleasant, and in the meantime will endure the doubt of potentially having cancer. On the other hand, there are the cases of cancer in women that were cleared in the screening process. These cancers may have either developed after the screening process or have been by the radiologist. Studies suggest that half of these cancers can be seen on the mammograms on later review [37]. With these problems considered, we can see that this area is ripe for automation.

A good candidate for this automation is deep learning and deep neural networks. In recent years, these technologies have been fairly successful in classifying medical images. There have been successful studies into automatic classification of CT-scans of lungs for lung cancer [16] and retinal scans for diabetic retinopathy [9]. Additionally some work has already been done on applying neural networks to mammography, as can be seen in Chapter 2. But in order to use deep learning, we need a large collection of data which the system can learn from.

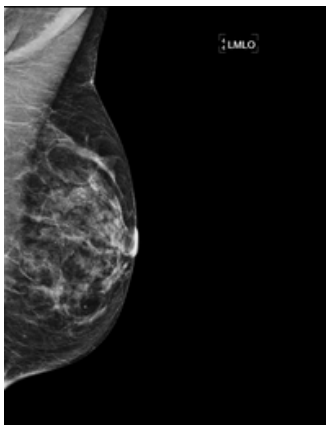
Fortunately, we have such a data set. The mammograms obtained from the Dutch national breast cancer screening have been saved since the programme went digital. In a collaboration with Bevolkingsonderzoek Noord, Bevolkingsonderzoek Zuid-West, and Zorginstituut Nederland we have been granted access to the these mammograms and the diagnoses based on the data, as well as the follow up results. In this thesis, we will look into applying the current state of the art in image classification neural networks to the problem of whole-mammogram classification. We will demonstrate our pipeline, with which we start with a raw input mammogram, transform it to a workable format, and then put it through well-known existing networks in order to reach a classification.

In the next chapter, we show an overview of the history of both mammography and neural networks. In Chapter 3 we will be going over the basics of the operation of neural networks. In Chapter 4 we will discuss the specifics of the data provided to us by Bevolkingsonderzoek Noord and Bevolkingsonderzoek Zuid-West. In Chapter 5 we will discuss our methods of preprocessing our mammograms into a workable format, and in Section 5.3 we will discuss the two existing neural networks that we will be applying to the data. In Chapter 6 we will discuss our experimental set-up and our the results of our testing. In the remainder of this introduction, we will give a brief overview into mammography and what can be seen the images.

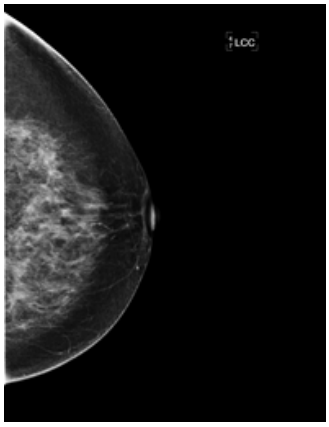
This research has been a master's thesis conducted at the Leiden Institute for Advanced Computer Science (LIACS), Leiden University in collaboration with Zorginstituut Nederland. The project was supervised by Wojtek Kowalczyk from LIACS and Pieter ten Have from Zorginstituut Nederland.



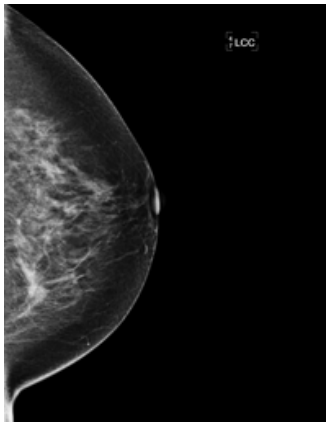
(a) CC view of a left breast. Leftover surgical staples visible are in the mammogram.



(b) MLO view of a breast.



(c) Mammogram of a breast without cancer in 2011.



(d) Mammogram of the same breast two years later.

Figure 1.1: Different example mammograms. These mammograms were taken from the iBOB data set described in Section 4.2.

1.1 Mammograms

Mammograms are x-ray photographs of breast tissue. A quick flash of x-rays is shot through the tissue and is captured on the other side. As with similar technologies, the x-rays go through the tissue and are absorbed proportionally to the density. Figure 1.1 shows two mammograms of the same breast. Here, we can see some general structures within the tissue, as well as the nipple which is slightly denser than the surrounding tissue. Skin is also slightly more dense and is therefore also clearly visible. Metal objects, such as surgical staples left in the tissue, are even brighter as the metal absorbs much more radiation than the surrounding tissue. This is clearly visible on the left side of image Figure 1.1a.

There are several views from which a mammogram can be taken. Two of these are considered the standard views: the mediolateral oblique (MLO) view and the craniocaudal (CC) view, of which MLO is the most important since it shows the most tissue. The standard views are used in regular breast exams. Depending on circumstances, other (supplementary) views may be used. However, we will focus on these two. Potential anomalies may be visible in either or both of the views. Examples of both of these views are shown in Figure 1.1.

The CC view is taken from the head looking down and may show some pectoral muscle at the edge of the image. The MLO view is taken at an approximately 45° angle looking from the side down. This is not orthogonal to the CC view which makes localization more difficult, but it covers more tissue than the mediolateral view (ML) view in the upper quadrant. This is important as this area has a higher incidence of breast cancer.

The task of classifying these mammograms is not obvious. Figure 1.1c and Figure 1.1d show the same breast in CC view, two years apart. In the two years after the first mammogram, cancer had developed and is visible in the second. The difference is hard to judge to the untrained eye, but the density in the lower left quadrant has increased significantly.

Chapter 2

Related work

Significant research has been done on mammography, neural networks, and the application of the latter to the former. In this chapter we will present an overview of the development in all three of these fields.

2.1 Mammography

Digital mammography has been around since the 1980s [8] and as the quality of the imaging improved, Computer Aided Detection and Diagnosis (CAD) became possible. As technology progressed, the results possible with digital mammography became comparable and even better than analogue mammography [26, 38], and x-ray-sensitive film made way for digital sensors. As technology improved, the Netherlands switched over from analogue imaging to digital mammography entirely. This resulted in more cancers being detected, but also in more benign tumours being referred for biopsy [27]. This improvement is considered to justify the higher cost [36] of digital mammography compared to film.

Most developed countries have a breast cancer screening programme aimed at women aged 50–69 [28, chapter 3], but the exact procedures, screened demographics, and results vary. The Netherlands uses two mammogram views (CC and MLO) and a double reading, which is the the most common procedure across European countries. The double reading means that each mammogram is evaluated by at least two radiologists which need to agree on a diagnosis. If they disagree, a third radiologist will be consulted to determine the final diagnosis.

As with other medical diagnoses, it is important to be able to compare the classification of mammograms between different studies or even different countries. In order to improve this, BI-RADS [18] was created. In this standard, a mammogram is assigned to one of seven classes, depending on the severity of the case. This is further described in Section 4.1.

Some work has also been by Moreira et al. [25] to make a standardized data set: INbreast. This data set contains mammograms in various viewpoints, annotated with the locations and shapes of anomalies. The data set is rather small, however, so studies making use of it tend to use data augmentation strategies. More details can be found in Section 4.3.

2.2 Neural networks

Neural networks, or more specifically artificial networks, are a computational model inspired by biological brains by composing relatively simple neurons in order to compute a complex function. The basic idea initially described in the 1940s by McCulloch and Pitts [22]. The technique is discussed with more detail in Chapter 3. Initial works showed that the structure could be used to compose more complex functions, but it was still did not have too many practical applications.

The next advancement came when Werbos [39] described his backpropagation algorithm in 1975. This algorithm (explained in more detail in Section 3.5) allowed networks with multiple layers to learn functions such as the exclusive-or in a feasible and efficient way. Still however, the neural networks were outperformed by much simpler and computationally cheaper methods such as support vector machines. Especially the field of image classification had to wait for GPU acceleration to be available.

MNIST [17] is a data set of 60,000 training images and 10,000 test images of handwritten digits. It was an early benchmark for the capabilities of various ways of image recognition. Initially in 1998, the best error rate achieved was 0.7% by a convolutional neural network and 0.8% by a a support-vector-machine. Since then the

best result has been lowered to 0.23% by Ciresan et al. [5] using a committee of convolutional neural networks, which is even more impressive considering the human error rate is approximately 0.2%.

Another common progress measure is the annual Imagenet Large-Scale Visual Recognition Challenge [32] (ILSVRC) in which images from 1000 categories from Imagenet [6] are classified. The initial contest in 2010 was dominated by variations on the SIFT-algorithm [20] or algorithms using hand-crafted features. For ILSVRC 2012 however, the winning implementation was a convolutional network called AlexNet [15] (described in more detail in Section 5.3.1) that achieved an top-5 error rate of 1% when the runner up achieved only 26% using a combination of SIFT and similar algorithms. Since then, most successful submissions have been convolutional neural networks with improvements to architecture such as VGG [33] in 2014 and ResNet [11] in 2015. We describe ResNet in more detail in Section 5.3.2.

2.3 Existing applications of neural networks to mammography

Kooi et al. [14] show that a network architecture similar to VGG [33] can achieve a performance level similar to radiologists when looking at small patches of the image. This however requires the training set to be annotated on a rather granular scale. Our data set, described later in Chapter 4, is on a per-breast measure. This makes their results an interesting goal for our approach, but not yet applicable to our research. Our data is labelled on a per-breast level only, which makes it impossible to use this approach.

Zhu et al. [42] instead use an approach similar to ours as described in Chapter 5, by looking at an entire mammogram at once. They use a version of AlexNet [15] that has been stripped of all fully connected layers, and instead use the output from the convolutional layers to create a label for a specific area. Additionally, they pre-train the network with hand-crafted convolutions designed to be sensitive to masses. Using this method, they achieve performance similar to human for the INbreast data set.

Dhungel et al. [7] split the problem into multiple stages: first detect the location of masses in the mammogram, separate the mass from the background, and finally determine whether the mass is malignant. This way, the detection network only has to classify whether a mass is malignant whilst not having to consider the mammogram as a whole. Detecting candidate-masses is less difficult, and thus can be done on the whole mammogram. This removes the need for having annotated data.

Chapter 3

Neural networks

Artificial neural networks (or just neural networks) are a method of computation inspired by the workings of biological brains. In such a brain, each neuron has some number of input connections providing it with information either from senses or from other neurons. Those incoming connections either stimulate or inhibit the neuron itself. If a neuron is sufficiently stimulated, it will in turn activate or inhibit other neurons that are connected to its output. By tuning the connections between neuron and the stimulation needed for a neuron to become active, the system can learn. By increasing the number of neurons and the connections between them, the system becomes able to learn more complex input-output relations.

We can think of neurons described above as a graph where each neuron is a node, and connections between neurons are weighted directed edges between those nodes. An illustration can be seen below in Figure 3.1. On the left, we have an input layer with some number of input nodes where we provide the data to the network, here labelled i_0 through i_2. On the right we have the output layer with nodes o_0 and o_1. Here we can read the output that the network has computed based on the values i_0, i_1, and i_2. In between are the hidden layers which help with the computation. In the following sections, we will be discussing how this structure can compute and how we can tune it to do what we want it to.

This chapter gives a very brief overview of the techniques used in artificial neural networks. However, to get a good overview of artificial neural networks, one should refer to [2], which explains everything described in this chapter and more.

3.1 Applying a neural network

The goal of a neural network is to learn (an approxiation of) some function f that transforms the input I into the desired output t. To understand how this process works, we must first look at how the network transforms the input into the output.

We start with setting the input values of the network to represent our input data. The exact methods for this are explained further in Section 3.4. Looking at our example in Figure 3.1 we now have a value in I_0 through I_2.

In order to compute the value for the other nodes in our network and reach an output, we look at the connections between the nodes, shown by the arrows. A connection from a node i to a node j has a weight ω_{ij}. Using those

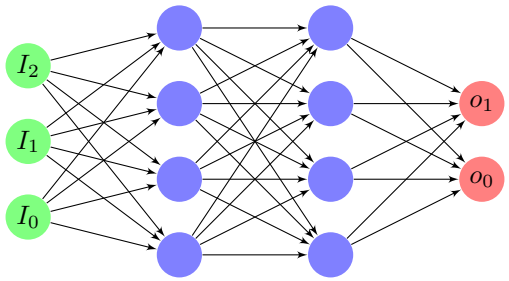


Figure 3.1: Example of a multi-layer neural network. Input on the green neurons to the left left is communicated between each fully-connected blue layer to the red output neurons on the right.

weights, we can compute the input to node j as the weighted sum of the values of all of its connected nodes:

$$x_j = \sum_i \omega_{ij} x_i$$

If we recursively apply the rules above, we can eventually compute the value of the output nodes o_0 and o_1 and reach our result.

3.2 Error functions

Right now our neural network is a machine that computes an output depending on its input. This output depends entirely on its input and the weights of the connections between the different neurons in the network. Now we want to ascribe meaning to this input output relation.

We can consider the relation between the input, the weights, and the output of the network to be a black box optimization problem. For this optimization function, we need a goal function that to minimize. We call this function E. With this function, we can simply use an optimization algorithm to solve our problem. An algorithm that works well for neural networks is (stochastic) gradient descent, which we will describe in Section 3.5.

The simplest error function we can feasibly use is the sum of squares. It is simply the sum of the square of the difference between all output nodes o_i and the values we want them to have t_i. We can write this as follows:

$$E_{\text{SOS}} = \frac{1}{2} \sum_i (t_i - o_i)^2$$

There are other functions that can be used as an error function, such as the commonly used categorical cross-entropy function. More information on these activation functions can be found in [2, chapter 6].

3.3 Activation functions

The rules for applying a neural network described in the first section have a shortcoming. The value of a node here is a linear combination of the nodes that come before it. From the composability of linear operations, we know that the concatenation of linear functions itself is a linear function. Thus, the final operation can only be used to approximate linear functions. To mitigate this, we introduce a non-linear activation function φ that transforms the input of a neuron into its output. Thus, we can rewrite our previous equation for the value of a neuron as follows:

$$x_j = \varphi\left(\sum_i \omega_{ij} x_i\right)$$

We then need to find a suitable function φ. In this section we will be looking at some mathematical functions. For reasons related to the gradient descent algorithm we will see in Section 3.5, we also need our activation functions to be differentiable and and the derivative should not be too close to zero. Therefore, we will be commenting on the derivative as well. In order to link our two definitions for the propagation of data through a network, we can define φ to be the identity function:

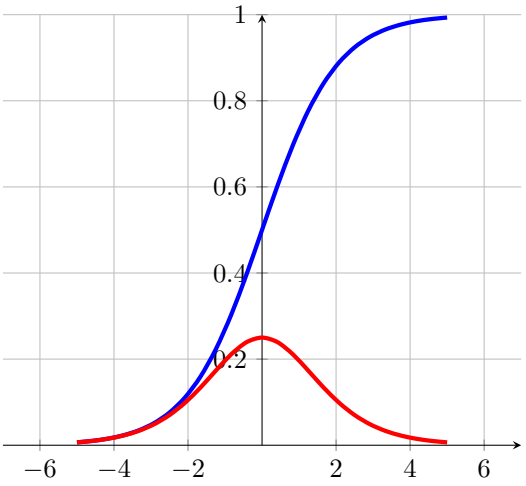
$$\varphi(x) = x \qquad\qquad \varphi'(x) = 1$$

This function is nicely differentiable, but is also a linear function, so it does not help us in making our artificial neural network non-linear. In this section, we will be discussing three commonly used alternatives: the sigmoid function shown in Figure 3.2a, the hyperbolic tangent function shown in Figure 3.2b, and rectified linear units (ReLU) shown in Figure 3.2c.

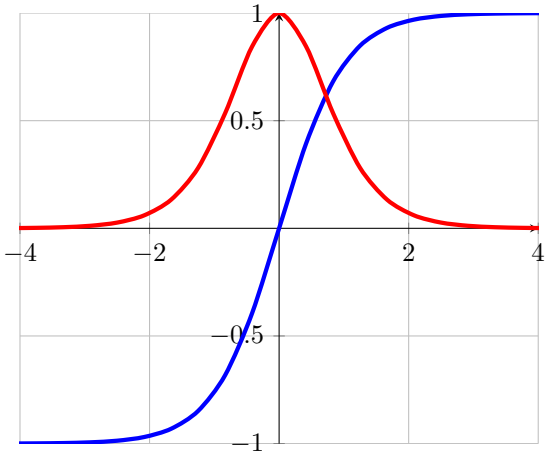
$$\varphi_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}} \qquad \varphi'_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}} * \left(1 - \frac{1}{1+e^{-x}}\right) = \varphi_{\text{sigmoid}}(x) * (1 - \varphi_{\text{sigmoid}}(x))$$

$$\varphi_{\tanh}(x) = \tanh x = 2\varphi_{\text{sigmoid}}(2x) - 1 \qquad \varphi'_{\tanh}(x) = \frac{e^{-x}}{(1+e^{-x})^2}$$

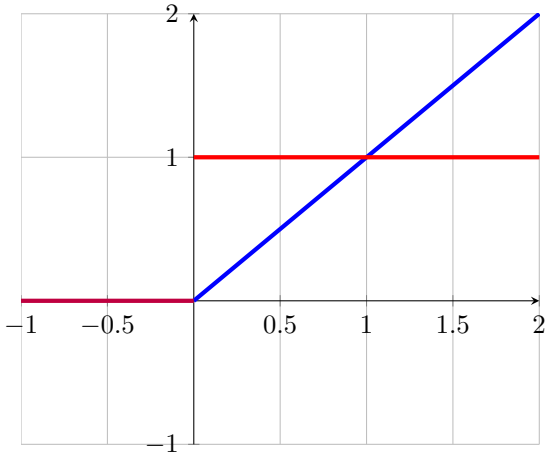
$$\varphi_{\text{ReLU}}(x) = \max(0, x) \qquad \varphi'_{\text{ReLU}}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$



(a) Sigmoid function



(b) Hyperbolic tangent function



(c) Rectified Linear Units

Figure 3.2: Illustration of different activation functions. The input level is shown on the x axis. The blue line is the activation curve, and the red line is the derivative of the activation curve. Note the differences in scale.

The sigmoid function has a nice S-curve and a well defined continuous gradient as seen in Figure 3.2a. This makes it easy to understand and nicely applicable to gradient descent. However, it also has a few issues. Its derivative is rather small compared to its response. Additionally, its derivative is near zero on around its extremes. This causes gradients to either vanish or explode when applying gradient descent recursively across many layers.

Given the problems listed above, the hyperbolic tangent function shown in Figure 3.2b is generally preferred to the sigmoid function. It has more or less the same S-curve, but it is centred around zero. Additionally, its derivative function ranges between zero and one (as seen in Figure 3.2b) instead of between zero and approximately 0.25. This improves training speed. However, it still suffers from the vanishing gradients problem as the derivative still goes to zero in its extremes.

An alternative approach is to use the ReLU function shown in Figure 3.2c. It is non-linear as values < 0 are ignored, and its derivative is trivial to compute as shown in the equations above. The function is technically not completely differentiable as the gradient is undefined at $x = 0$, but that can be mitigated by using either of the limits of $\lim_{x \to 0} \varphi'_{\mathrm{ReLU}}(x)$. The ReLU function suffers less from and rectifies the vanishing gradients quite well, and is therefore the activation function of choice for the hidden layers in deep neural networks [15, 11]. It does have problems of its own [41]. Most notably, it may result in dead neurons that will never generate an output. These neurons cannot recover from this by gradient descent, since the local gradient will always be zero.

Variations on ReLU exist that mitigate this problem. Leaky ReLU [41] adds a small negative gradient based on a parameter $0 < \alpha < 1$ to negative inputs. This works as shown below. However, as this method is more

complicated than the regular ReLU, it should only be used if the problems associated with ReLU actually occur.

$$\varphi_{\text{Leaky}}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases}$$

$$\varphi'_{\text{Leaky}}(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x < 0 \end{cases}$$

3.4 Data encoding

While the problem for a neural network can be anything, ultimately the data needs to be represented as activation levels on a number of neurons. Thus, we need a representation. There are two main ways of doing so: (normalized) continuous encoding, or one-hot encoding.

As the name suggests, continuous encoding is best applied to continuous values such as temperature or brightness. We can simply take any scalar value and set the activation of an input neuron to its value. Optionally, we can normalize the value first, making sure that all of our values range between zero and one.

If the value we need to represent is discrete, for example a choice of some specific values, one-hot encoding is more suitable. In this encoding. we have one neuron per class, and the neuron for the chosen class is set to one while the others remain zero.

3.4.1 Output encoding

As the previous section suggests, one-hot encoding is a better choice for classification problems. Machine learning has trouble however with learning the exact form of all zeros and one one in its output. Remember from before that discrete functions are hard to learn since progress cannot be gradual. We can work around this by applying the softmax (σ) activation function. It is defined as follows:

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{K-1} e^{y_j}}$$

where y is a vector of K elements starting at index 0. By taking the exponent of a value and dividing it by the sum of the exponents of all other values, it exaggerates the relative differences. This causes a distribution that is similar enough to a one-hot encoding whilst still having a well-defined derivative. Additionally, the result always sums up to one, giving the output a possible interpretation as a probability distribution.

3.5 Stochastic gradient descent

We have now shown the general structure of an artificial neural network and the error function that we want to minimize. In this section we will look at actually minimizing that error. The basic algorithm for this is stochastic gradient descent with backpropagation.

The stochastic gradient descent (SGD) algorithm is an iterative method of optimizing a function that it differentiable. It works by iteratively showing our network examples, and correcting the weights proportionally to how much they have contributed to the error E. Backpropagation means that we start computing these contributions at the output and work back to the input. The goal is to compute the gradient of the error, and walk along the slope to the local minimum. To do so, we first need to compute the derivative of the error with respect to the weights. The error itself depends on the output of the neurons, which depends on the activation of the neurons, which depends on the actual weights. If we take v_i as the input to a node i and o_i as its value after applying the activation function to v_i, we can summarize this as follows:

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial v_j}\frac{\partial v_j}{\partial \omega_{ij}} \tag{3.1}$$

We can now compute each individual derivative, and use the chain rule to compute the final derivative. We can see that the second factor of that part of the equation $\frac{\partial o_j}{\partial v_j}$ is actually the derivative of the activation function used. This is why the activation function had to be differentiable. The last factor is simply equal to o_i. This leaves the first factor.

If we look at an output neuron, we can simply plug in the derivative of the error function. If we look at an inner neuron, however, we must compute the effect that particular neuron has on the error function. This is actually the sum of the effect that that it causes in all neurons it is connected to. Let us call these neurons the set L. We can then write the derivative as the following recursive expression:

$$\frac{\partial E}{\partial o_j} = \sum_{l \in L} \left(\frac{\partial E}{\partial v_l} \frac{\partial v_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial v_j} \frac{\partial v_l}{\partial o_j} \right) = \sum_{l \in L} \left(\frac{\partial E}{\partial o_l} \frac{\partial o_l}{\partial v_j} \omega jl \right) \tag{3.2}$$

We can then put these equations together to finally arrive at the derivative of the error to a particular weight:

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial v_j} \frac{\partial v_j}{\partial \omega_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial v_j} o_i \delta_j \tag{3.3}$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial v_j} = \varphi'(v_j) \begin{cases} \frac{\partial E}{\partial o_j} & \text{if } j \text{ is an output node} \\ \sum\limits_{l \in L} \delta_k \omega jl & \text{otherwise} \end{cases} \tag{3.4}$$

Now that we have a method to compute the derivative of the error to the weight, we can then start using this formula to update the weights. With the stochastic gradient descent algorithm, we show the network an example, and then compute a change to make to the gradient. This change is defined as follows:

$$\Delta \omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} = -\eta o_i \delta_j \tag{3.5}$$

Here, η is a parameter called the learning rate that controls the speed at which the convergence happens. Smaller values make the process slower to converge and more likely to settle in local minima, but larger values may make the process chaotic and not converge at all. Values can be from 0 to 1, but typical values range from 0.1 through 0.01.

With the delta we derived in Equation 3.5 we can finally update our weights and optimize our network for the task we want to use it for. If we repeat this process of showing a random example and adjusting the weights of the data, eventually the network will become better at the task.

In Equation 3.4 we still have a placeholder for the derivative of the error $\frac{\partial E}{\partial o_j}$. The actual value for this depends on the error function (see: Section 3.2) used. We can give an example here. If we use the square error loss as a function, then we can rewrite it as

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = t_j - o_j \tag{3.6}$$

We can then plug this equation in into the Equation 3.4 and do the rest of the process as normal.

3.5.1 Variants of SGD

The SGD algorithm described above works well under most circumstances, but it is not the only or even the best algorithm to tune the parameters of a network. One extension for this is to use momentum in the algorithm [31].

The momentum (see: [2, chapter 7, section 5]) in this case means that we remember the previous adjustment we made to the weights. It causes us to "remember" the direction in which we were originally adjusting the weights, reducing local noise in adjustments and possible allowing greater improvements towards a local minimum. For this, we introduce a momentum parameter m between zero and one. Our new delta rule at time θ is then as follows:

$$\Delta \omega_{ij}(\theta) = \eta \delta_j y_i + m \Delta \omega_{ij}(\theta - 1)$$

Another optimization is to use mini-batches. This way, we compute the $\Delta \omega_{ij}$ as normal, but we do not apply them directly. Instead, we keep a running sum of the change for each weight, and we only apply the combined change every n samples. This approach tends to stabilize the training, as it cancels out opposing changes. This prevents them from oscillating which can disturb the training process.

More variations exist, such as dynamically changing η over time, maintaining a second order momentum, and automatically optimizing the training parameters. Those are outside the scope of this chapter. A more thorough overview can be seen in [2, chapter 7].

−1	0	1
−2	0	2
−1	0	1

Figure 3.3: Applying a Sobel operator to an image. Values below zero are clamped to zero, as if using the ReLU activation function. Original photo taken from [40] and licensed under Creative Commons Attribution-Share Alike 3.0 Netherlands.

3.5.2 Overfitting

Applying the techniques above ensures that the output of the network will eventually produce the perfect output, for all data it was trained on. Unfortunately, this usually means that the network has learned to recognise all of the specific instances, and that it no longer applies to the general case. We will discuss two techniques used to mitigate this.

First, there is the regularization of parameters. Because of the large number of parameters, it is generally always possible to find a configuration which gives the correct results for the data the neural network is trained on, but those networks generally do not generalize to new data. The problem is similar to polynomial curve fitting. Given n points, we can always find an $n-1$th order polynomial that perfectly fits all data points. This polynomial will however have very large coefficients, and may not approximate the underlying function very well. Instead, it is usually desirable to have a lower order polynomial with smaller coefficients. In neural networks, the same principle applies, and thus the solution is similar: we punish the network by including the magnitude of the weights in the error term. This reduces the risk of overfitting.

Another technique is the use of drop-out layers [35]. When testing, these layers are simply transparent, giving exactly their input as output. In training however, they randomly set the output of some fraction of their neurons to zero. In practice, this makes the network more robust to small disturbances of the input, since no individual feature can be assumed to exist.

3.6 Convolutional layers

In order to apply a neural network to an image, we can simply hook up every pixel (or pixel channel, for non-greyscale images) and have a huge number of neurons. This results in a huge number of trainable parameters, one for every neuron to every neuron in the next layer, making training difficult. Convolutional layers make use of the locality of data in images in order to reduce those numbers, while still using the entirety of the image.

The idea behind convolutional layers is similar to the Sobel operator [34]. The Sobel operator is a matrix that we lay over our original image. We then multiply each element of the matrix with the pixel value it overlays and sum the result to get a new value for our output. We then move our matrix one pixel to the right to generate the next pixel in our output. When we reach the end of a row, we move down a pixel and repeat from the process left. An example of this is shown in Figure 3.3. The operator used produces a high response when the right side of a pixel is brighter than the left side, a low response when it is the other way around, and finally it results in approximately zero when the left and right side are similar.

The example shows the typical left-to-right edge detector with a size 3 matrix. We can include more channels (such as color) in our operator by extending it into a third dimension, applying each layer of our matrix to a channel of our image.

Moreover, we are not always using a matrix of size 3. Instead we can cover more of our original image at once. The amount that we shift by per application of the matrix is also a tunable parameter, with larger shifts resulting in a bigger reduction in resolution of the learned features.

Our example shows how you can detect edges horizontal edges. However, the learned by artificial neural networks do not necessarily mean anything to a human observer. The training process will determine what is important

by itself. Moreover, neural networks do not learn just one feature. AlexNet [15] for example learns 96 features in its first layer. And then the second layer learns 256 new features by recombining the features learned by the first layer. The resulting features are rarely recognizable as anything but noise, but they enable the network to learn many different patters while keeping the number of parameters relatively low.

Chapter 4

Data sets

For this thesis we use two data sets which we will describe in this chapter. The iBOB data set was created especially for this research and is described in Section 4.2. This data set is used for all our training and is our main target for success. Additionally, we were allowed access to the INbreast data set, described in Section 4.3 which will be used for cross-referencing results. A cursory overview of both data sets can be found in Table 4.2. More information in how we pre-process the data can be found in Section 5.1.

4.1 BI-RADS

When mammograms are examined by human radiologists, they are classified on the Breast Imaging Reporting And Data System (BI-RADS) [18] scale. This scale is a number from 0 through 6 and it classifies what can be seen in the mammogram. An overview of this can be seen in Table 4.1. The descriptions are as in [18] but modified to reflect the response in the Dutch Bevolkingsonderzoek.

In our research we want to determine automatically if there needs to be a follow-up, based on the imaging alone. Thus, we want to separate our data into no follow-up needed (BI-RADS 1, 2, or 3) and follow up suggested. (BI-RADS 0, 4, and 5) This leaves out BI-RADS 6. However, since it is impossible to arrive at this score without biopsy, our data set does not contain any instances of this score. For evaluation purposes, we add it to the follow-up suggested category.

It should be noted that the differences between adjacent BI-RADS scores are hard to define objectively. What one radiologist may classify as BI-RADS 3, another may either classify as BI-RADS 2 or BI-RADS 4.

4.2 iBOB data

Our iBOB[1] data was collected by Bevolkingsonderzoek Noord and Bevolkingsonderzoek Zuid-West from the biannual breast cancer screening in the period 2011 through 2017 and was provided to us by Zorginstituut

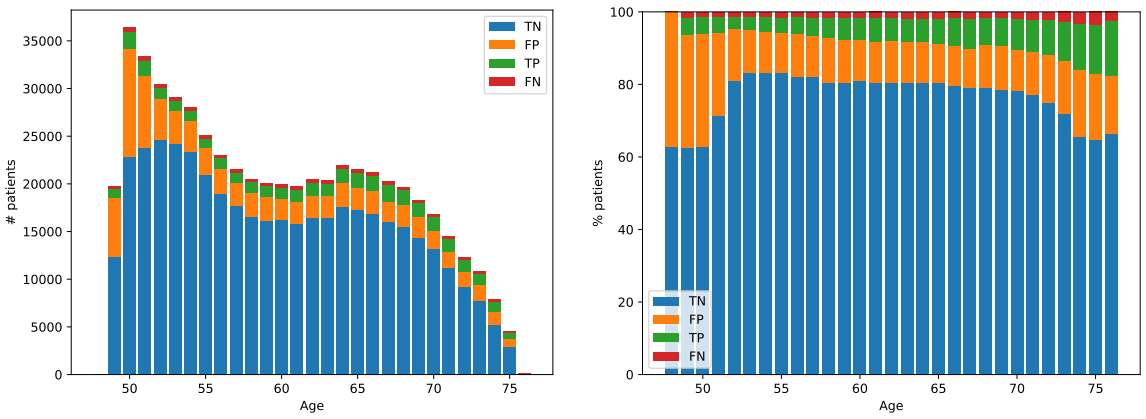
BI-RADS	Meaning	Follow-up
0	Incomplete assessment. Images may be unclear or even partially missing.	Additional imaging or biopsy
1	No findings	Regular check-up
2	Benign findings. Some spots are seen, but they are deemed non-cancerous.	Regular check-up
3	Probably benign findings.	Additional check-up
4	Suspicious findings	Biopsy
5	Highly suggestive malignant growths	Biopsy
6	Definitely malignant, proven by biopsy. This score can therefore never come from imaging alone.	n/a

Table 4.1: Meaning of the different BI-RADS scores. Taken from Liberman and Menell [18].

[1] National database for breast cancer research. In Dutch: ICT-systeem voor BevolkingsOnderzoek Borstkanker

	size	(compressed)	# images	% malignant
iBOB	10.7 TiB	3.3 TiB	557,642	7.7
INbreast	8.4 GiB	1.7 GiB	410	7.2

Table 4.2: Statistics and distribution of cancer in used data sets



(a) Relative to the population

(b) Relative to age

Figure 4.1: Distribution of different diagnoses by age

Nederland. The data consists of two parts: a collection of mammograms in Digital Imaging and Communication in Medicine [24] (DICOM) format, and a spreadsheet with the results of the results from the screening.

The DICOM files have been anonymized in various ways. All personal fields such as names, state ID number and such have been removed. Instead, the patient is referred to with a six-digit patient ID. Date of birth is not stripped entirely, but instead is rounded to the nearest January first. The date of the screening is stored in full, as well as whether the patient has a breast prosthesis and the particular settings of the machine used. Even so, we also store the data on an encrypted drive on a machine accessible only to our research group in order to ensure the privacy of the patients involved.

The results were provided as a spreadsheet containing the BI-RADS scores given per breast by the two (or three) radiologists. For every screening, the actual ground truth (whether there was cancer) was also known, and was included with the results. We use this to determine the concept of true positive/true negative.

A true positive patient has breast cancer and was diagnosed as such. A true negative has no cancer and was diagnosed as such. A false negative is a patient who has cancer but was not diagnosed, and finally a false positive is a patient who does not have cancer but was diagnosed as such. Due to the presence of interval cancers, we can only truly say a patient is truly negative if there has been a secondary check-up two years later. If the patient was however diagnosed with breast cancer in between the two screenings, she will be marked as a false negative. Figure 4.1 shows the distribution of these diagnoses and age in our data set. Here we can see that the proportion of patients with cancer increases slightly with age, that false positives are quite common, and false negatives are quire rare.

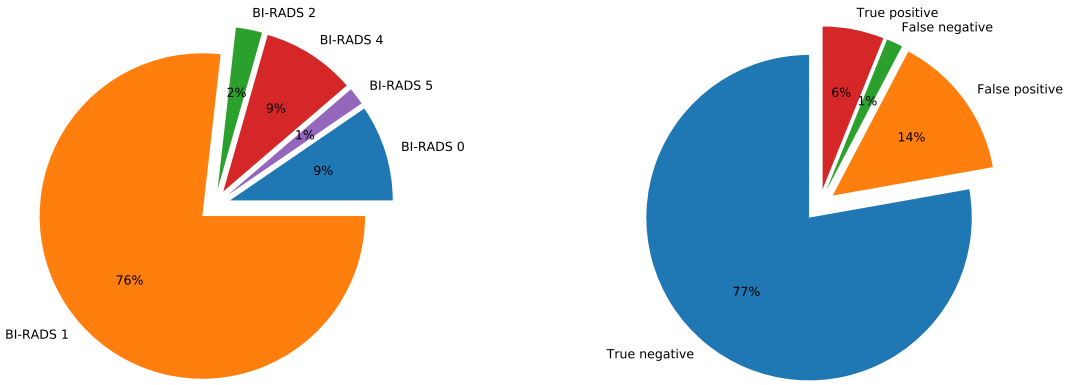
As shown in Table 4.2, the incidence of malignant results is significantly higher than the normal incidence rate of approximately 1.5% [27]. The data set has been selected to include more malignant mammograms, in order to make the data-set less skewed to non-cancer cases.

Figure 4.2 shows the distribution of the different BI-RADS scores and diagnoses in the data set. We can see that the vast majority of the data is BI-RADS 1. We can also see that according to the BI-RADS scores, 19% of patients referred, while only 7% should have been. This level of over-diagnosis is more or less consistent (proportionally) to the distribution of cases in the Netherlands.

4.2.1 Classification

Our data sources for the mammograms and the diagnoses are separate, so we need to do some work to combine them. To match a mammogram to its diagnosis, we know an anonymised patient-id, a side, and the study date. For the diagnoses however, this date has been rounded to the nearest January first. This causes some issues.

First of all, this makes it impossible to match a diagnosis to a secondary mammogram taken shortly after the



(a) BI-RADS

(b) Diagnosis

Figure 4.2: Distribution of the BI-RADS scores and diagnosis in the iBOB data set.

first one. This occurs when the original mammogram was not classifiable. In this case, we match the diagnosis to whichever date was most recent, i.e. the second mammogram, as it is more likely to be the mammogram the diagnosis was based on.

Even when there is only one mammogram study to connect to its diagnosis, the matching is not straight-forward, as the diagnosis date may be a few weeks after the study date. This is why we simply brute-force the matching in this case: first, we try to match the year of the study to the year of the diagnosis. If that match does not exist, we try the next year. As the screening is biannual, this strategy manages to resolve most of the pairings. Any mammograms that cannot be matched to their respective diagnosis will be discarded for our purposes.

In our research, we will be learning to classify mammograms one at a time. This is not entirely representative of human classification, where the radiologist can look at all mammograms for a patient at once. These are from both breasts and from different angles of the same breast. Not every mammogram will cover the entire breast, so the anomalies may not be visible on a specific mammogram. For training purposes, we will however assume that they are. More discussion about using more than one mammogram can be found in Section 6.6.

Other than that, there are issues inherent to classifying mammograms. Cancer does not have one defining characteristic that separates it from benign lesions. It is seldom a round, white spot. Therefore it is difficult to classify.

There is also the issue of interval cancers, which are cancers that are found in between two screenings. When this happens, there are two options: either the cancer was present on the previous screening's mammograms but was missed by the radiologists, or it originated shortly after the previous screening. According to Van Dijck et al. [37] approximately half of all of these interval cancers were actually visible when re-evaluating the mammograms by an independent radiologist.

Unfortunately, our data was never re-examined. Therefore we will leave out all of these images as images with visible cancer may confuse the training process. This might be slightly beneficial when comparing our results to the performance of radiologists as this may potentially leaves out the more difficult cases, but we prefer having a more robust training process to the potential downside of not training on these cases.

4.2.2 Non-visual-indications

Among the meta-data of the images are some risk-factors for breast cancer. For instance, we know the age of the patient, which correlates with a higher incidence of breast cancer. Other factors, such as whether the patient has a prosthesis, whether there has been previous surgery, or if there is a potential higher risk from family history.

In our research, we disregard all of these factors, in favour of training a purely visual classifier. This is both more fair in comparison to a radiologist, and easier to reproduce with data from other sources which may not have that particular data.

4.3 INBreast data

In order to compare our results with related work, we also obtained access to the INBreast [25] data set. This data set contains hand-annotated mammograms. The data set contains for every mammogram the BI-RADS score (see Table 4.1) as well as the coordinates of all lesions. All mammograms in the data set have been verified using different means, so BI-RADS 0 is not present while BI-RADS 6 is. We will only be considering these as either true positive or true negative, since all measurements were checked.

Chapter 5

Methodology

In this chapter we will be looking over the details of our research method. First, in Section 5.1, we will discuss our preprocessing algorithms for the mammograms. In Section 5.2 we will be discussing the performance metric we chose for our evaluation. Finally, in Section 5.3 we will discuss the network architectures that we are applying to the problem.

5.1 Preprocessing

While are input images are of high resolution and quality[1], they are not yet suitable for processing. In order to make the mammograms more suitable for analysis, we run preprocessing on our images. Doing so, we want to improve two factors:

1. images should fit in GPU memory for processing,
2. images should only contain the breast, and
3. images should be cropped to the subject.

In our research, we used two different methods of preprocessing methods, each with their own strengths and weaknesses. Each method accomplishes the above tasks by doing two things: remove the scanner marker, and resize the image so that it fits in memory.

Scanner markers are burned-in pixels that indicate the type of picture taken and the angle at which is was shot, either CC or MLO. The type is always a mammogram. These markers introduce nothing but noise to our neural network, and thus we would like to remove them. Tools exist to black-out these pixels automatically based on the type of machine used, however, at the time of this thesis they proved unreliable for our data set.

Naïve method

Our first method uses the fact that scanner markers are usually of the brightest white in the image, while this does not (usually) occur anywhere else in the image. We set all of those pixels to black.

After that, the image is scaled down by a factor 8, resulting in an image of 416 by 512, or 320 by 416. For our scaling algorithm, we use an area-based method, in order to reduce the introduced noise otherwise caused by sub-sampling.

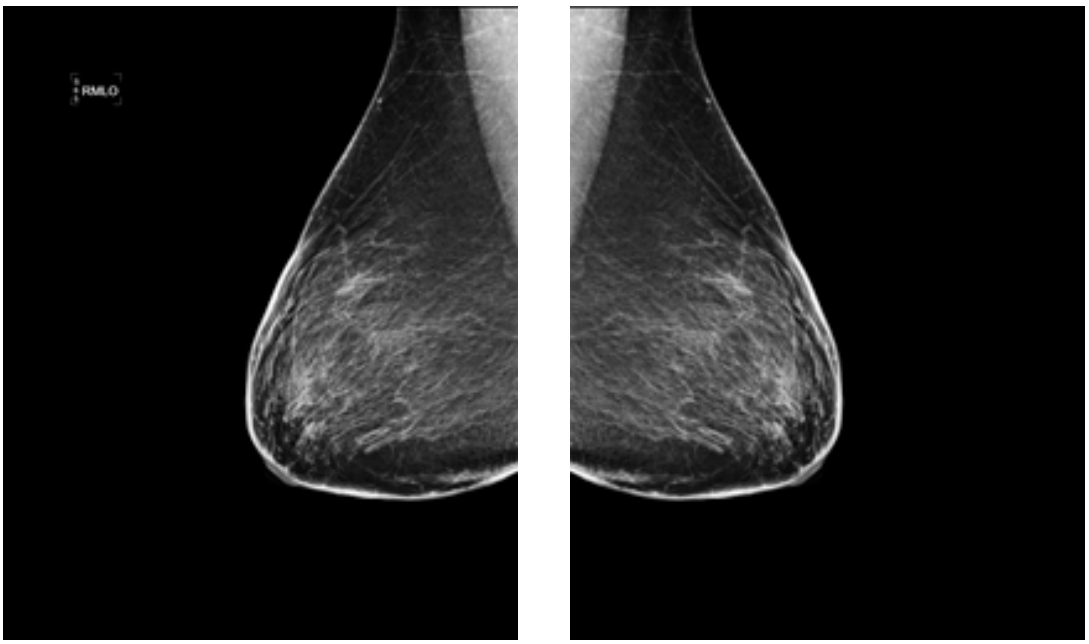
Finally, for further normalization, we ensure that the breast is always on the left side of the image. This makes for easier comparison between left and right breasts when looking at the images manually. A complete display of the transformation can be seen in Figure 5.1.

The main advantage of this method is that it is cheap and reliable. It is however not perfect; some of the older images contain different types of scanner markers, in different shades of grey.

Cropping method

While the above method works, it has two downsides: the image is still quite large to process with a convolutional neural network (more than 4 times larger than the input for the original AlexNet) and the images also still

[1] Input resolution is up to 30 megapixels in 12 bit greyscale.



(a) Original mammogram (b) Normalized mammogram

Figure 5.1: Mammogram in various stages of preprocessing

contain a lot of black space. Our second preprocessing method addresses this. In this section we will be going over the complete steps of preprocessing. A step-by-step progress of the algorithm can be seen in Figure 5.2.

The goal of the algorithm is to determine the part of the image where the breast is located. As a first step, we make a copy of the image and apply a slight Gaussian blur to that image. This smooths out irregularities in the image, so that the foreground-background separation can be done more cleanly.

After that, we apply Otsu's method [29] to the image, to determine what is the foreground and what is background noise. A description of that algorithm can be found in Section 5.1.1. When this algorithm is applied to an image without a noisy background, it will instead detect a noise threshold within the image. This can be seen in Figure 5.2b. However, this will not matter as explained below.

Using the foreground found in the previous step, we can then find bounding boxes for all foreground elements as seen in Figure 5.2c. We then select the largest bounding box by number of pixels and crop to that box. Skin will typically be very bright in the picture, so it will always be considered foreground and thus an ideal candidate for a bounding box. Finally, the resulting image is resized to 224 by 224 pixels. This size ensures that we can process these images efficiently by allowing large batch sizes. The end result can be seen in Figure 5.2e.

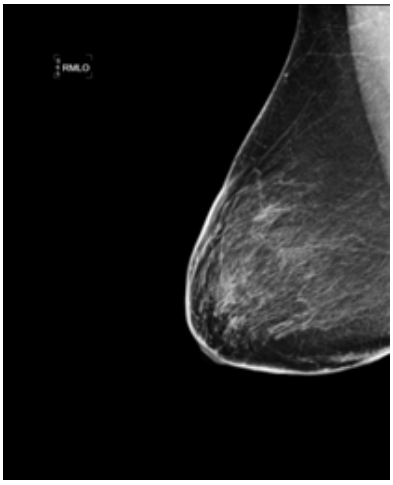
5.1.1 Otsu's method

Otsu's method [29] is an algorithm that can be used to separate foreground from background in noisy images. The assumption is that the intensity distribution of the image is bimodal, i.e. it is the sum of two different distributions. It then finds the threshold that best separates the two distributions. In actuality, it minimizes the weighted variance within the classes (intra-variance) as follows:

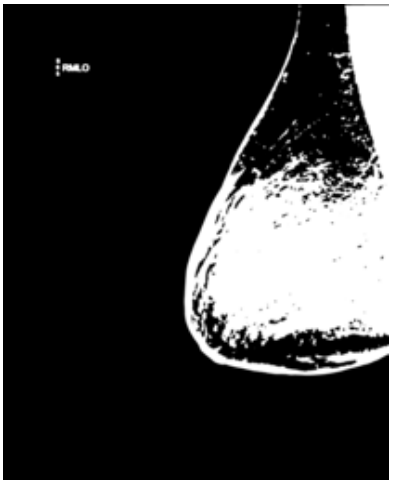
$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$$

where $\omega_i(t)$ is the probability of being in class class i with respect to threshold t, i.e. the probability of an intensity being on either side of the threshold. Otsu shows that this can be reworked as maximizing variance between the classes (inter-variance) instead:

$$\begin{aligned}\sigma_b^2 &= \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0\omega_1(\mu_1 - \mu_0)^2\end{aligned}$$

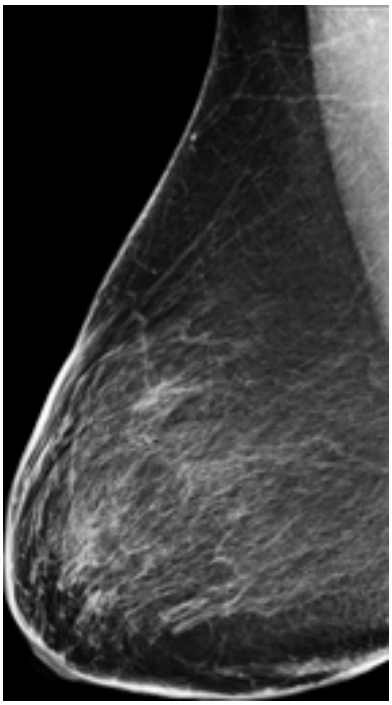


(a) Blurred image



(b) Otsu's threshholding

(c) Bounding boxes



(d) Cropped

(e) Resized

Figure 5.2: Stages of the preprocessing algorithm

where μ_T is the total mean.[2] This leaves ω_i and μ_i as the only variables needed. Both can be computed by maintaining a running sum of the buckets before- and after the threshold, allowing us to quickly compute the ideal threshold in $O(n + b)$ where n is the number of pixels and b is the number of buckets, or grey levels.

5.2 Performance metric

In order to properly review our accuracy, we need a performance metric. Accuracy is an obvious choice, but it is also very sensitive to class skew in the data set. In our case, an accuracy of more than 93% can be achieved by simply concluding *no cancer* regardless of input.

In related work, the results are commonly classified using the Receiver operating characteristic area under curve (ROC AUC, or simply AUC) metric. This is however a very misleading metric that can look very promising while the actual model is not [19, 10]. There are also cases where a difference in AUC does not correlate with a difference in actual predictive power [12]. Because of this, we decided against this metric. We do provide the metric, however, as a comparison to related work.

Instead, we will be looking at *sensitivity* and *specificity*. These metrics are shown below.

$$\textit{sensitivity} = P(\text{detected}|\text{cancer})$$
$$\textit{specificity} = P(\text{not detected}|\text{no cancer})$$

Since a binary classifier typically produces a number, we can make a trade-off between sensitivity and specificity by raising or lowering the threshold from which we classify a result as cancer for a given validation set. We can then cross-validate this by applying this threshold to an independent test set.

In order to achieve a sensitivity of n, you determine the $1 - n$ quantile of scores for all instances that should be classified as cancer. Using that threshold to determine the specificity gives a single-number performance indication. We will refer to this as the specificity measure.

Unfortunately, it is not possible to directly optimize for this performance metric directly, as it is an aggregate over multiple images. Instead, the optimizer for the neural network is set to optimize categorical cross-entropy. This is somewhat correlated with our specificity measure, but not perfectly.

5.2.1 Human performance

For any given performance measure of an automatic classification system, it is necessary to also have the same measurements for human performance. While it is hard to get the exact values for these metrics due to the way the statistics are recorded, we can make some approximation under some assumptions.

Recall from the introduction that 1,000,000 women are screened every year, resulting in 24,600 referrals. 6,900 have breast cancer, while the remaining 17,700 do not. Furthermore, each year there are an additional 2,200 cases of breast cancer among women in the screening programme. Unfortunately, these statistics are on a per patient basis, and not per breast, but we can make an estimate regardless. Assuming that half of these cases may not have been visible at the time of the screening, and counting every patient only once, we get a specificity measure of 98.2% and a sensitivity of 86.3%. If we count these cases on a per breast measure, the sensitivity will be similar, but the specificity will be slightly lower.

With these numbers, we have decided to put the sensitivity target of our tests to 90%. This score is slightly better than the radiologists score in our estimate, but the incidence of cancer is also greater in our data set than it is across the larger population.

5.3 Networks

In this thesis, we will be looking at two well-known existing networks. These networks are not specific to any kind of specific task, but are rather used for general purpose image classification.

5.3.1 AlexNet

AlexNet by Krizhevsky et al. [15] is one of the first successful deep convolutional networks, being the first to outperform the more established computer vision algorithms (generally variations on SIFT [20]) at the ImageNet Large Scale Visual Recognition Challenge [32] in 2012. Today it is outshined by more complicated architectures

[2] Some additional proof is needed to go from the first formulation of σ_b to the second. That proof can be found in the original paper but is outside the scope of this thesis.

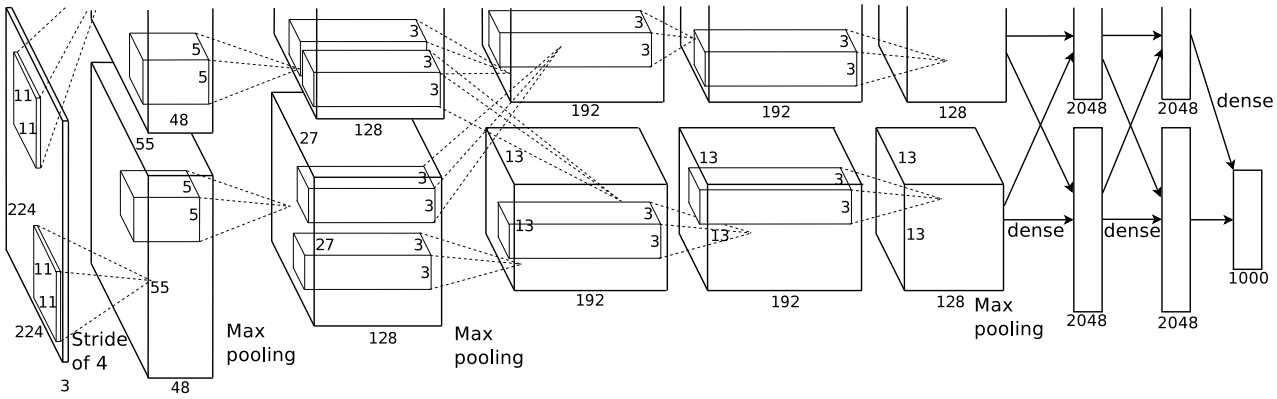


Figure 5.3: Architectural overview of the original AlexNet by Krizhevsky et al. [15]. The input to the network is a 224 by 224 greyscale image which results in a 50,176-dimensional input. We then have 5 convolutional layers, ending in two fully-connected layers and finally our output layer. For our experiments, we reduced the number of output classes to 2.

with more convolutional layers, but it still has value for being computationally very cheap, and being generally reasonably accurate. A schematic of its structure can be found in Figure 5.3.

Its architecture consists of 5 convolutional layers and 3 fully-connected layers, finally ending in a softmax layer. The convolutional part contains an explicit split in the filters so the network can be more efficiently be parallellised across multiple GPUs. This was a necessary optimization in 2011 when GPU memory was more limited. The split is no longer required, but our implementation of it still contains a single-GPU approximation of it.

Aside from layer-splitting to circumvent GPU-memory limits, AlexNet also popularized two regularization features: drop-out and local response normalization (LRN). Drop-out is a feature only active while training. It randomly disables a certain percentage of neurons when processing an image. This technique improves redundancy in the network, since no particular neuron can be counted on to fire. LRN on the other hand prevents overfitting by normalizing the response across different channels, so that no one pixel in the data will provide an extreme result. According to the authors, this mitigates the need to perform explicit regularization on the weights.

Modifications
In order to apply the network to our mammograms, we make two modifications. First, we reduce the input format from full-color to greyscale as mammograms do not have color. Additionally, we change the input size to fit the dimensions of the images produced by our two input shapes described in Section 5.1. Our cropping-based preprocessing method results in the same resolution as the original network architecture. The resizing method however results in a significantly larger input. Due to the way AlexNet is organised, this results in larger features at the end of the convolutional layers in the network. As a consequence, the number of connections in the first fully-connected layer is significantly higher. The exact architecture used can be found in Appendix A.

5.3.2 ResNet

ResNet [11] is a very deep convolutional network. In the original paper, the authors discuss and experiment with networks with as many as 1202 layers, although the results no longer improve beyond a certain point before that. For the purposes of this thesis, we will be looking at the 50-layer variant.

Training networks with large numbers of layers is difficult, since the exploding/vanishing gradients-problem [30] makes the training process rather unstable. ResNet mitigates this issue by introducing "shortcuts" in the network, allowing gradients to pass around layers if needed. These shortcuts also allow the network to operate on changes, that is, the difference between the outputs of consecutive layers. These short-cuts are used as a repeating building-block in order to create trainable networks of arbitrary depth. An example of such a building block can be seen in Figure 5.4.

The network starts with a $7 \times 7 \times 64$ and convolutional layer followed by an 3×3 max-pooling layer, both with a stride of two. After that comes a total of 16 of three-layer building blocks similar to the one shown in Figure 5.4 for a total of 50 image-like layers. The resulting output feeds into a 1000-way fully connected layer, which is then combined with a softmax activation into the final classification. The exact specification of the

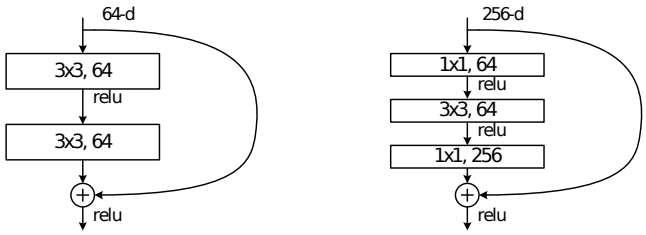


Figure 5.4: Two- and three-layer residual building blocks used in ResNet. Each building block ends in the same dimensionality as its input, and adds its input to the convolution it, combining it with a ReLU operation. Diagram taken from [11].

architecture 50 can be found in [11, Table 1].

Modifications

Like AlexNet in the previous section, we modify ResNet in order to better apply to our data. This means reducing the input channels to greyscale, and increasing the input dimensions for our resized images. For ResNet however, the convolutional layers always reduce the to the same size. Since convolutional layers have the same number of parameters regardless of input size, this means the total number of trainable parameters does not depend on the input shape for ResNet. We will see this difference in the next chapter. The exact architecture used and methods to reproduce it reliably can be found in Appendix A.

Chapter 6

Experiments

In this chapter we go over our experimental set-up in Section 6.1. In Section 6.2 we will show how we verified our pipeline by training for a simpler task.

6.1 Set-up

For our experiments we chose to work with four configurations of neural networks. Table 6.1 shows an overview of the different configurations. ResNet has the same number of trainable parameters regardless of its input size as the convolutional part of the network always results in the same output size. AlexNet on the other hand, ends its convolutional part with a size of an approximately constant fraction of the input. This means that quadrupling the number of inputs also quadruples the number of trainable parameters.

In the sections below we will go into more detail about the specific technical implementation of our set-up.

6.1.1 Software

We used the Pydicom [21] for working with the DICOM files. In order to save storage space we used `gzip`. This resulted in savings of approximately 75% We implemented our preprocessing pipeline using OpenCV [3] for most of the image processing, using its implementation of Otsu's segmentation method, edge detection, and rescaling of the images.

Our neural networks were implemented in TensorFlow [1], using the abstraction methods provided by Keras [4]. We use its CUDA bindings to run our code on a GPU. Our ResNet (see: Section 5.3.2) implementation was taken from the Keras sample networks[1] and modified to accept grayscale images.

For AlexNet, we created our own implementation based on the original paper [15] using Keras components. Two features were missing from Keras as they are no longer relevant to modern architectures and mostly superseded by different layers that are usually better or no longer needed. The split layer type was previously used to split computation of the network across multiple GPUs, but with more memory available in modern hardware this is no longer necessary. The local response normalization (LRN) layer has been deprecated in favour of other normalization techniques. We have implemented both of these features using existing TensorFlow components in order to better approximate the original network.

The weights of the network have been randomly initialized. We did not use transfer training, as doing so would have required us to generate a data set that fit our models. Training was done using SGD (see: Section 3.5) using a learning rate of 0.01. This value was chosen based on the recommendations by He et al. [11]. We used

Network	Base architecture	Preprocessing	Input size	Trainable parameters
`alexnet`	AlexNet	Crop	224×224	$46,105,282$
`alexnet_large`	AlexNet	Resize	416×512	$181,371,586$
`reduced_resnet`	ResNet50	Crop	224×224	$23,532,418$
`resnet`	ResNet50	Resize	416×512	$23,532,418$

Table 6.1: Overview of network configurations used in experimentation.

[1] See: `https://keras.io/applications/#resnet`

no additional regularization techniques. A complete overview of the network architectures used can be found in Appendix A.

In order to speed up meta analysis of our data, we created a database with an index of the diagnoses and meta-data for our images. This database was based on the SQLite database system. We combined this with Pandas [23] in order to improve the performance of reading and writing that index.

6.1.2 Hardware

Our computations were run on two twin servers. Each server is equipped with two Titan X Pascal GPUs with 12 GiB of memory each. The systems additionally had 64 GB of RAM and two Intel Core i7-6950X CPUs.

The set-up was used to train up to four neural networks at once during our experimentation phase. The resulting networks however can be run on any common hardware. A GPU is not required but can speed up the process when evaluating large batches of images, but evaluating a single image can be done in a few seconds regardless.

6.2 Testing the pipeline

Since we had some time to verify our pipeline before getting the target data we chose to train a network to predict some known value to verify that our data loading worked properly. For this we used the mammogram view position (see Figure 1.1 for the differences between them) as a target category.

Initially the images contain a marker indicating the configuration of the machine including the view position in human readable text. However, using either of the preprocessing methods described in Section 5.1 should have removed this information. Even so, the task is very simple even for laymen to mammography as the general shape is quite different.

We train the network for ten epochs to see if our pipeline works. The progress for all of our tested variations can be seen in Figure 6.1. Not shown are the loss values, since including them would make the progress in later epochs invisible. We can see that the task is expectedly easy to train for, since we achieve fairly an accuracy of more than 99% in a single epoch.

For the cropped versions, ResNet slightly outperforms AlexNet for the same number of epochs, but it is also about 1.5 times more computationally expensive. For the resized versions, the additional computations required for ResNet begin to show. The AlexNet variant still takes about the same time per training epoch (40 minutes) but the additional work in the convolutional layers balloons the training time to nearly 4 hours per epoch. Looking at the training progress in Figure 6.1 we can see that this longer training process doesn't even pay off that much, as the versions running on the cropped images perform comparably well.

6.3 Training

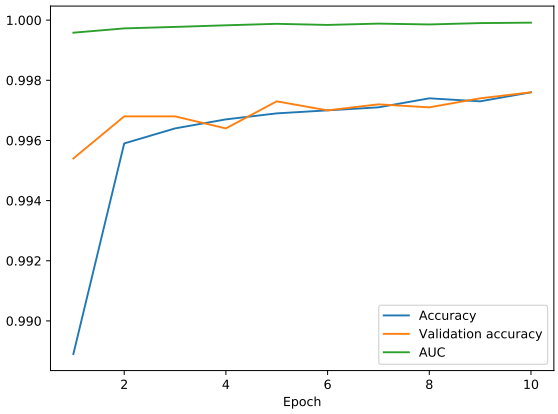
In our experiments we test the four networks shown in Table 6.1. For this, we used the training data which is 80% of the total images. Figure 6.2 shows the progress of this training. These graphs show the loss value for the training data, the loss value for the validation data, and the specificity (see Section 5.2) and the receiver operating characteristic area-under-curve (AUC) as the training progresses. Ideally, both loss and validation loss should tend to zero, while AUC and specificity should tend to one.

We can see in both Figure 6.2b and Figure 6.2d that ResNet reaches its best performance level just after a few training epochs. After this, we can see clear signs that ResNet in its cropped variant suffers from overtraining. We can see that progress is still being made on the training data, but the metrics for the validation data are rapidly worsening. This is an indication that the network is starting to memorize the individual images, rather than the patterns in them.

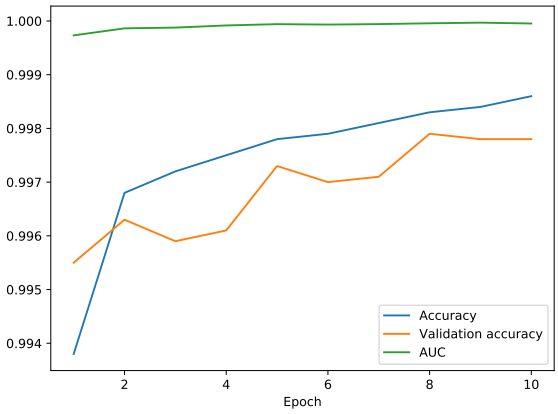
AlexNet on the other hand is a bit slower to train. After 40 epochs they (Figure 6.2a and Figure 6.2c) are barely reaching levels reached by ResNet in 5 epochs. Even with their faster training cycles, this still takes longer. However, if we continue training for 40 more epochs, we can see that the levels start to become similar, and finally overtraining effects start to show as well. We therefore decided to stop the training process.

6.3.1 Training cost

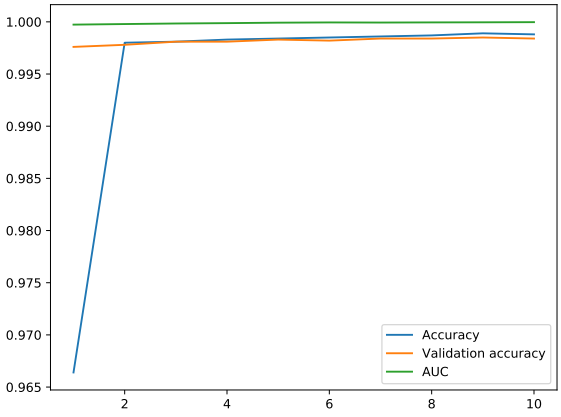
The different networks have significantly different costs to train per epoch. In general, ResNet is more complicated than AlexNet and requires significantly more operations per image processed. This does not impact the training process for the smaller cropped images too much, as AlexNet and ResNet take approximately 41 and



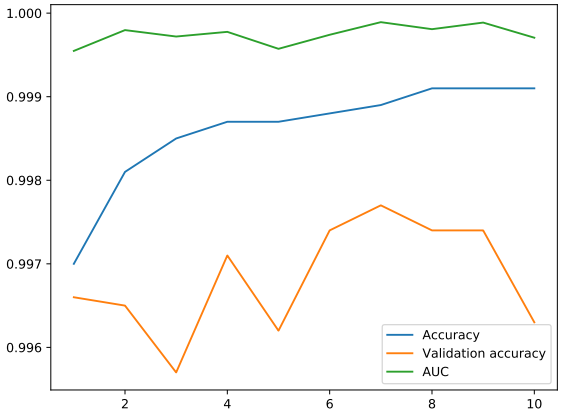
(a) AlexNet with cropped images



(b) ResNet with cropped images

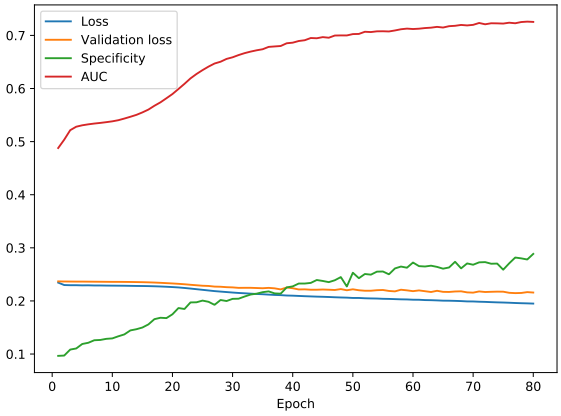


(c) AlexNet with resized images

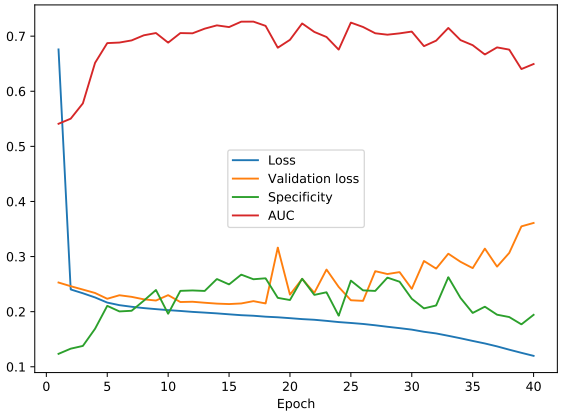


(d) ResNet with resized images

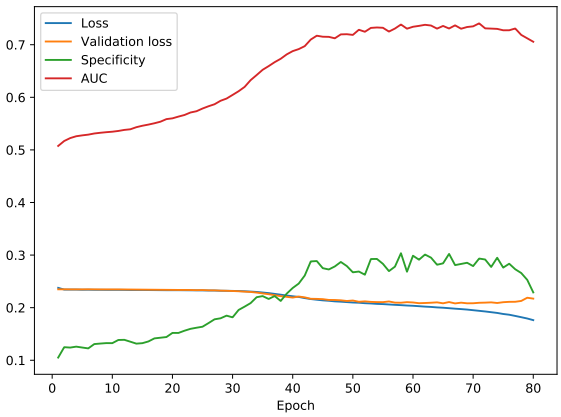
Figure 6.1: Training progress for learning the difference between MLO and CC mammograms.



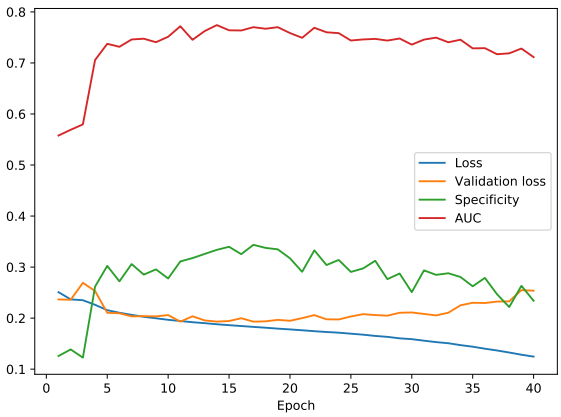
(a) AlexNet with cropped images



(b) ResNet with cropped images



(c) AlexNet with resized images



(d) ResNet with resized images

Figure 6.2: Training progress for ResNet on cropped images

	Time/epoch	File size
`alexnet`	41:16	352 MiB
`alexnet_large`	54:33	1.4 GiB
`reduced_resnet`	54:17	181 MiB
`resnet`	3:38:50	181 MiB

Table 6.2: Cost to train and save the different network configurations. Time is an average across all training documented here.

	Epochs	Val AUC	Val Spec	Test AUC	Test Sen	Test Spec	Test Acc/T-Acc
`alexnet`	40	68.0%	21.4%	69.8%	91.3%	21.1%	94.0% / 25.3%
	80	72.2%	28.1%	74.1%	90.8%	28.0%	94.1% / 31.7%
`alexnet_large`	40	68.8%	23.4%	68.4%	90.8%	23.2%	94.2% / 27.1%
	80	73.7%	28.1%	73.4%	90.3%	27.9%	94.2% / 31.5%
`reduced_resnet`	40	71.6%	25.0%	74.2%	91.7%	25.1%	94.4% / 29.1%
`resnet`	40	76.0%	29.5%	76.0%	90.6%	29.0%	94.6% / 32.5%

Table 6.3: Evaluation results of the training process. Validation sensitivity is 90% by definition. All other values are computed by evaluating the network on the validation set to determine a cut-off point, and then cross-referencing with the test set. Accuracy and T-accuracy (threshold) are the prediction accuracy with the threshold set at the $\frac{1}{2}$ or the determined cut-off-point needed to achieve the sensitivity, respectively.

55 minutes per training epoch. We can however see the difference in utilization, as ResNet keeps the system mostly busy while AlexNet is constantly loading new data.

The difference really starts to show when we consider the resized images. AlexNet now takes 51 minutes per epoch to train, but ResNet now balloons up to over three and a half hours per epoch. This is mainly caused by the fact that the convolutional part of the network now has more than 4 times as many pixels to deal with. Since ResNet has more convolutional layers than AlexNet, this affects the former more proportionally.

The lower cost of evaluation for AlexNet as well as its tendency to overtrain later has allowed us to do more training epochs for our two variations. The performance however is rather similar. It should be noted that both AlexNet variants have a significantly larger number of trainable parameters and therefore require a bit more memory to run. An overview of the statistics mentioned in this section can be seen in Table 6.2.

6.3.2 Variations

We have experimented with several variations on the training process in order to improve the results or its efficiency. None of them had a significant impact on the results of the network, but for completeness we will list them here.

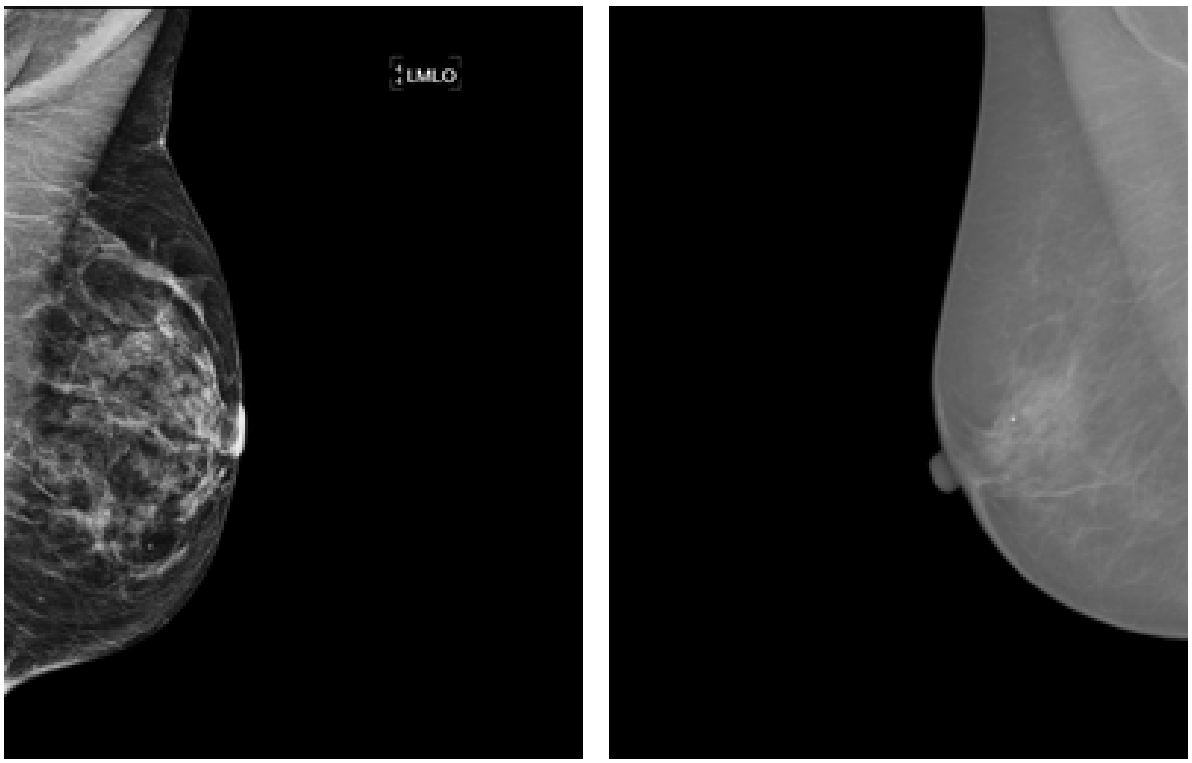
Oversampling of the cancer cases lead to a faster training process arriving at its plateau earlier, which was slightly lower than it was without oversampling.

We have also attempted removing parts of our data that may be difficult to train on, as they either look very different from regular mammograms (such as mammograms with a visible prosthesis) or that are difficult to classify for human observers, such as BI-RADS 0. This also did not significantly change the results of the training process.

6.4 Evaluation

After training the networks until they no longer improve, we evaluate their performance using the validation- and test set. We first use the validation set to determine the cut-off point to achieve a 90% sensitivity, and then evaluate the performance on the test set using that threshold. The resulting figures can be see in Table 6.3. For our evaluation, we use a snapshot from the training process of each network when its validation loss was lowest. This lets us ignore the effects of overtraining that may have degraded the final network. For completeness, we also show the state of both AlexNet variations after 40 training epochs.

We can see that the sensitivity on the test set is at least 90%. This suggests that the threshold set using the validation set is a good generalizes to the new data. The resulting AUC and specificity are also very similar between the two data sets. For the remainder of this thesis, we will therefore only discuss the results on the test set rather than both for brevity.



(a) iBOB image (b) INbreast image

Figure 6.3: Differences in image characteristics between iBOB and INbreast mammograms. Both are MLO images and normalized to maximum intensity.

6.5 Cross-validation with INbreast

As discussed in Section 4.3 we have been granted access to the INbreast data set. However, when applying our trained networks to the INbreast data, the results were bad to non-existent, leading to a sensitivity below 1%. We consider these results to be sufficiently low that they are non-significant. On the other hand however, the specificity is near 100%. This is merely the result of our networks classifying all instances in the data as non-cancerous.

We attribute this to the differences between the iBOB data and the INbreast data: the contrast levels between them for example are vastly different. We can see these differences in Figure 6.3. The first thing that stands out is that the contrast levels are wildly different between the two images. The iBOB image on the left shows al sorts of structures internal to the breast. In the INbreast image on the right, they are also visible, but with way less contrast.

In order to make the INbreast image even visible, we have had to boost the levels significantly. This should not make a difference in processing as this levels normalization is also done as a step in loading the data. It should be noted that DICOM-specific tools handle the files well, but general image processing tools do not.

As a result of these findings, we have chosen to leave out the exact results achieved on the data set. Future work may include some preprocessing to bring the contrast levels of the images more in line with what we see in the iBOB data, in order to better cross-validate the results.

6.6 Postprocessing

Up until now we have applied our trained networks to single mammograms and used the output to create a diagnosis about the breast. However, in practice, there are at least two mammograms per breast taken from different view positions, as mentioned in Figure 1.1. We can combine the results from the different mammogram views to create a better classifier. In this section we will evaluate the results when using the minimum, maximum, and mean score across the different view positions.

For our evaluation, we combine the different scores of the different view positions. Then, we follow the same procedure as outlined in Section 6.4 in order to determine a cut-off and then compute the resulting AUC and specificity. We only use best instance of each of the network architectures trained.

	Baseline		Minimum		Maximum		Mean	
	AUC	Spec	AUC	Spec	AUC	Spec	AUC	Spec
`alexnet`	74.1%	28.1%	75.1%	28.1%	76.6%	31.1%	77.2%	29.4%
`alexnet_large`	73.4%	27.9%	74.1%	30.3%	76.5%	34.2%	76.9%	34.5%
`reduced_resnet`	74.2%	25.1%	74.0%	23.2%	77.6%	27.6%	77.9%	27.6%
`resnet`	76.0%	29.0%	77.2%	32.0%	77.2%	33.6%	79.0%	33.1%

Table 6.4: Performance difference by aggregating classification scores across view positions. Baseline is taken from Table 6.3.

Intuitively one might say that the maximum of all different scores would be the best classifier, as if an anomaly is visible in a single mammogram and obscured in another, we should detect it nonetheless. In practice, it turned out that taking the average of the different measurements was more productive. The improved results are shown in Table 6.4. For brevity, we only show the AUC and specificity, but the other metrics also changed similarly.

We can see that all three aggregate functions sometimes provide an improvement, but an improvement, but the mean was consistently the largest improvement. This suggests that the ideal function is not this simple, but instead a more complex separation would be more useful. A secondary classifier may be trained to determine the ideal function, but that is outside the scope of this thesis.

One last thing to note is that AUC and specificity do increase by a similar amount depending on the the aggregate function used. Also, `reduced_resnet` produces a fairly high AUC value, but that is not really reflected in the specificity value. We consider this an indication that indeed the AUC metric is not an ideal metric in classification problems such as this.

Chapter 7

Conclusions

In this thesis we have looked at applying two well-known neural networks for image classification to the problem of whole-mammogram classification. Here, we have shown that it is possible to train a rudimentary classifier for the problem. However, the classifier itself is not yet a useful replacement for or even comparable to human classification.

Additionally, we have created a preprocessing pipeline for mammograms in order to more easily classify them. This pipeline may be useful for future expansions of this research, with more advanced neural networks and better computational capabilities. Larger networks, such as ResNet, are still rather expensive to train on large data sets of high resolution images.

In this thesis, we also discussed using a different performance metric. Contemporary research publishes accuracy and receiver operating characteristic area under curve, but neither captures the actual quality of the classifier very well. By aiming at a specific sensitivity and computing the specificity, the score resembles more closely of the way we intuitively rank the performance of mammogram classification.

While our trained classifiers fail to meet human classification in quality or even compare to similar research, we did notice that an increase in resolution leads to an improvement in detection quality. We are confident that with more experimentation and more work, it is definitely possible to improve the results we achieved.

7.1 Future work

As mentioned in the section above, a higher resolution can suggest an improvement to the quality of the results. In our research we presented two methods of preprocessing, with one having a better final resolution and preserving the aspect ratio. However, it would also be good to examine the results when using the cropping method (described in Section 5.1) with a larger final resolution. With current hardware, it should be possible to at least double the current dimensions.

Another direction for future work is to see if the multi-stage approaches discussed in Chapter 2 could work for this data set. If we can first identify the interesting areas of the image with a less expensive method, we can then further analyse those areas with the current network input sizes. This has the potential to greatly improve performance and reduce the memory requirements.

In our experimentation, we have not looked into additional regularization of the trained networks as the authors of both AlexNet and ResNet claim that their architecture should prevent overfitting by other means. However, after a relatively low number of training epochs, we do observe these effects. It might be interesting to look at the effects of additional regularization on the training process.

Aside from training these networks, there are various other image classification networks. The VGG networks are an obvious candidate, as are the architectures that have dominated the more recent ILSVRC 2017 competition. AlexNet (winner 2012) and ResNet are still quite good, but can no longer be considered the state of the art.

The performance metric we have introduced is related to, but not the same as the AUC metric. Since accuracy is not useful for skewed data sets, AUC is the most reliable performance indicator for competing research into mammography. It would however be interesting to see how well other research compares in these metrics. Reproduction of the studies mentioned in Chapter 2 with our performance metric might give some insight as to how close the current state-of-the-art is to human performance.

Finally, as most competing research has either been trained or at least evaluated itself on the INbreast data set, it would be interesting to see why our classifiers are this incompatible with the data. Visual inspection suggests a contrast issue, but perhaps this is merely a difference in encoding between Dutch and Portuguese mammography.

7.2 Acknowledgements

This thesis would not have been possible without the cooperation of both the Rijksinstituut voor Volksgezondheid en Milieu, Bevolkingsonderzoek Noord, and Bevolkingsonderzoek Zuid-West. We would like to thank them for the opportunity and their trust to work with the data from the Dutch mammogram screening. We also would like to thank Annemieke van der Steen in particular for her support regarding our questions into the Dutch screening programme and in arranging additional funding for the data.

Additionally we would like to explicitly thank Pieter ten Have from Zorginstituut Nederland for creating the project. His medical expertise has helped the project tremendously in the initial design phases as well as helped inform the final specification of the performance metrics.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[7] Neeraj Dhungel, Gustavo Carneiro, and Andrew P. Bradley. A deep learning approach for the analysis of masses in mammograms with minimal user intervention. *Medical Image Analysis*, 37:114–128, 2017.

[8] U. Fischer, K. P. Hermann, and F. Baum. Digital mammography: current state and future aspects. *European Radiology*, 16(1):38–44, Jan 2006.

[9] G. G. Gardner, D. Keating, T. H. Williamson, and A. T. Elliott. Automatic detection of diabetic retinopathy using an artificial neural network: a screening tool. *British Journal of Ophthalmology*, 80(11):940–944, 1996.

[10] David J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning*, 77(1):103–123, Oct 2009.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Jin Huang and C. X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, March 2005.

[13] Nederlandse Kankerregistratie. `https://www.cijfersoverkanker.nl/`, January 2019.

[14] Thijs Kooi, Albert Gubern-Merida, Jan-Jurre Mordang, Ritse Mann, Ruud Pijnappel, Klaas Schuur, Ard den Heeten, and Nico Karssemeijer. A comparison between a deep convolutional neural network and radiologists for classifying regions of interest in mammography. In Anders Tingberg, Kristina Lång, and Pontus Timberg, editors, *Breast Imaging*, pages 51–56, Cham, 2016. Springer International Publishing.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Jinsa Kuruvilla and K. Gunavathi. Lung cancer classification using neural networks for ct images. *Computer Methods and Programs in Biomedicine*, 113(1):202–209, 2014.

[17] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits, 1999. URL `http://yann.lecun.com/exdb/mnist/`. Accessed 2019-05-31.

[18] Laura Liberman and Jennifer H. Menell. Breast imaging reporting and data system (BI-RADS). *Radiologic Clinics*, 40(3):409–430, 2002.

[19] Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography*, 17(2):145–151, 2008.

[20] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, page 1150. IEEE, 1999.

[21] Darcy Mason et al. Read, modify and write dicom files with python code. `https://pydicom.github.io/`, 2015. Accessed 2019-05-31.

[22] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[23] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[24] Peter Mildenberger, Marco Eichelberg, and Eric Martin. Introduction to the dicom standard. *European Radiology*, 12(4):920–927, April 2002.

[25] Inês C. Moreira, Igor Amaral, Inês Domingues, António Cardoso, Maria João Cardoso, and Jaime S. Cardoso. INbreast: Toward a Full-field Digital Mammographic Database. *Academic Radiology*, 19(2): 236–248, 2012.

[26] H. W. Nab, N. Karssemeijer, L. J T. H. O van Erning, and J. H. C. L Hendriks. Comparison of digital and conventional mammography: A ROC study of 270 mammograms. *Medical informatics = Médecine et informatique*, 17:125–31, 04 1992.

[27] J. Nederend, L. E. M. Duijm, M. W. J. Louwman, J. H. Groenewoud, A. B. Donkers-van Rossum, and A. C. Voogd. Impact of transition from analog screening mammography to digital screening mammography on screening outcome in The Netherlands: a population-based study. *Annals of Oncology*, 23(12):3098–3103, 06 2012.

[28] World Organization. *Breast cancer screening*. IARC, International Agency for Research on Cancer, Lyon, France, 2016. ISBN 978-92-832-3017-5.

[29] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[30] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2, 2012.

[31] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[34] Irvin Sobel. Neighborhood coding of binary images for fast contour following and general binary array processing. *Computer Graphics and Image Processing*, 8(1):127–135, 1978.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

[36] Anna NA Tosteson, Natasha K Stout, Dennis G Fryback, Suddhasatta Acharyya, Benjamin Herman, Lucy Hannah, Etta Pisano, DMIST Investigators, et al. Cost-effectiveness of digital mammography breast cancer screening: results from acrin dmist. *Annals of internal medicine*, 148(1):1, 2008.

[37] Jos A. A. M. Van Dijck, André L. M. Verbeek, Jan H. C. L. Hendriks, and Roland Holland. The current detectability of breast cancer in a mammographic screening program. a review of the previous mammograms of interval and screen-detected cancers. *Cancer*, 72(6):1933–1938, 1993.

[38] Elisabeth M.S.J. van Gennip and Jan L. Talmon. *Assessment and evaluation of information technologies in medicine*, volume 17. IOS Press, 1995.

[39] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[40] Wikipedia, the free encyclopedia. Goliath poldermolen, 2012. URL `https://en.wikipedia.org/wiki/File:Goliath_Poldermolen.jpg`. [Online; accessed July 1st, 2019].

[41] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL `http://arxiv.org/abs/1505.00853`.

[42] Wentao Zhu, Qi Lou, Yeeleng Scott Vang, and Xiaohui Xie. Deep multi-instance networks with sparse label assignment for whole mammogram classification. In Maxime Descoteaux, Lena Maier-Hein, Alfred Franz, Pierre Jannin, D. Louis Collins, and Simon Duchesne, editors, *Medical Image Computing and Computer Assisted Intervention − MICCAI 2017*, pages 603–611, Cham, 2017. Springer International Publishing.

Appendix A

Network architectures

A.1 AlexNet

Our AlexNet implementation is mostly based on the original paper by Krizhevsky et al. [15]. Existing implementations existed before this thesis but we did not find a version that worked with the most recent versions of TensorFlow. Please contact the author for source code of the implementation if needed. Both of the variants discussed in this thesis are described in the table below.

Layer	Type		`alexnet`	`alexnet_large`
input_1	`InputLayer`	Input shape	$224 \times 224 \times 1$	$512 \times 416 \times 1$
zero_padding2d	`ZeroPadding2D`	Input shape	$224 \times 224 \times 1$	$512 \times 416 \times 1$
conv1	`Conv2D`	Input shape	$228 \times 228 \times 1$	$516 \times 420 \times 1$
			$11 \times 11 \times 1 \times 96$	
		Trainable	96	
		Stride	(4, 4)	
pool1	`MaxPooling2D`	Input shape	$55 \times 55 \times 96$	$127 \times 103 \times 96$
		Stride	(2, 2)	
		Pool size	3×3	
lrn_layer	`LRNLayer`	Input shape	$27 \times 27 \times 96$	$63 \times 51 \times 96$
zero_padding2d_1	`ZeroPadding2D`	Input shape	$27 \times 27 \times 96$	$63 \times 51 \times 96$
split_layer	`SplitLayer`	Input shape	$31 \times 31 \times 96$	$67 \times 55 \times 96$
split_layer_1	`SplitLayer`	Input shape	$31 \times 31 \times 96$	$67 \times 55 \times 96$
conv2_1	`Conv2D`	Input shape	$31 \times 31 \times 48$	$67 \times 55 \times 48$
			$5 \times 5 \times 48 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
conv2_2	`Conv2D`	Input shape	$31 \times 31 \times 48$	$67 \times 55 \times 48$
			$5 \times 5 \times 48 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
concatenate	`Concatenate`	Input shape	$27 \times 27 \times 128$	$63 \times 51 \times 128$
pool2	`MaxPooling2D`	Input shape	$27 \times 27 \times 256$	$63 \times 51 \times 256$
		Stride	(2, 2)	
		Pool size	3×3	
lrn_layer_1	`LRNLayer`	Input shape	$13 \times 13 \times 256$	$31 \times 25 \times 256$
zero_padding2d_2	`ZeroPadding2D`	Input shape	$13 \times 13 \times 256$	$31 \times 25 \times 256$
conv3	`Conv2D`	Input shape	$15 \times 15 \times 256$	$33 \times 27 \times 256$
			$3 \times 3 \times 256 \times 384$	
		Trainable	384	
		Stride	(1, 1)	
zero_padding2d_3	`ZeroPadding2D`	Input shape	$13 \times 13 \times 384$	$31 \times 25 \times 384$
split_layer_2	`SplitLayer`	Input shape	$15 \times 15 \times 384$	$33 \times 27 \times 384$
split_layer_3	`SplitLayer`	Input shape	$15 \times 15 \times 384$	$33 \times 27 \times 384$
conv4_1	`Conv2D`	Input shape	$15 \times 15 \times 192$	$33 \times 27 \times 192$
			$3 \times 3 \times 192 \times 192$	
		Trainable	192	

Layer	Type		alexnet	alexnet_large
		Stride	(1, 1)	
conv4_2	Conv2D	Input shape	$15 \times 15 \times 192$	$33 \times 27 \times 192$
			$3 \times 3 \times 192 \times 192$	
		Trainable	192	
		Stride	(1, 1)	
concatenate_1	Concatenate	Input shape	$13 \times 13 \times 192$	$31 \times 25 \times 192$
zero_padding2d_4	ZeroPadding2D	Input shape	$13 \times 13 \times 384$	$31 \times 25 \times 384$
split_layer_4	SplitLayer	Input shape	$15 \times 15 \times 384$	$33 \times 27 \times 384$
split_layer_5	SplitLayer	Input shape	$15 \times 15 \times 384$	$33 \times 27 \times 384$
conv5_1	Conv2D	Input shape	$15 \times 15 \times 192$	$33 \times 27 \times 192$
			$5 \times 5 \times 192 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
conv5_2	Conv2D	Input shape	$15 \times 15 \times 192$	$33 \times 27 \times 192$
			$5 \times 5 \times 192 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
concatenate_2	Concatenate	Input shape	$11 \times 11 \times 128$	$29 \times 23 \times 128$
pool5	MaxPooling2D	Input shape	$11 \times 11 \times 256$	$29 \times 23 \times 256$
		Stride	(2, 2)	
		Pool size	3×3	
flatten	Flatten	Input shape	$5 \times 5 \times 256$	$14 \times 11 \times 256$
dense1	Dense	Input shape	$6,400$	$39,424$
			$6,400 \times 4,096$	$39,424 \times 4,096$
		Trainable	$4,096$	$4,096$
dropout	Dropout	Input shape	$4,096$	
dense2	Dense	Input shape	$4,096$	
			$4,096 \times 4,096$	
		Trainable	$4,096$	
dropout_1	Dropout	Input shape	$4,096$	
dense	Dense	Input shape	$4,096$	
			$4,096 \times 2$	
		Trainable	2	
prediction	Activation	Input shape	2	

A.2 ResNet

The ResNet50 implementation that we used is part of Keras. One can reproduce our models from the Keras application samples with the following code snippet, or by consulting the structural description table below.

Listing A.1: Loading ResNet from Keras

```
import keras
# or: from tf import keras

ResNet50 = keras.applications.resnet50.ResNet50

reduced_resnet = ResNet50(
        input_shape=(224, 224, 1),
        weights=None,
        classes=2)

resnet = ResNet50(
        input_shape=(512, 416, 1),
        weights=None,
        classes=2)
```

Layer	Type		reduced_resnet	resnet
input_1	InputLayer	Input shape	$224 \times 224 \times 1$	$512 \times 416 \times 1$

Layer	Type		reduced_resnet	resnet
conv1_pad	ZeroPadding2D	Input shape	$224 \times 224 \times 1$	$512 \times 416 \times 1$
conv1	Conv2D	Input shape	$230 \times 230 \times 1$	$518 \times 422 \times 1$
			$7 \times 7 \times 1 \times 64$	
		Trainable	64	
		Stride	(2, 2)	
bn_conv1	BatchNormalization	Input shape	$112 \times 112 \times 64$	$256 \times 208 \times 64$
			64	
		Trainable	64	
activation	Activation	Input shape	$112 \times 112 \times 64$	$256 \times 208 \times 64$
pool1_pad	ZeroPadding2D	Input shape	$112 \times 112 \times 64$	$256 \times 208 \times 64$
max_pooling2d	MaxPooling2D	Input shape	$114 \times 114 \times 64$	$258 \times 210 \times 64$
		Stride	(2, 2)	
		Pool size	3×3	
res2a_branch2a	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$1 \times 1 \times 64 \times 64$	
		Trainable	64	
		Stride	(1, 1)	
bn2a_branch2a	BatchNormalization	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			64	
		Trainable	64	
activation_1	Activation	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
res2a_branch2b	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$3 \times 3 \times 64 \times 64$	
		Trainable	64	
		Stride	(1, 1)	
bn2a_branch2b	BatchNormalization	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			64	
		Trainable	64	
activation_2	Activation	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
res2a_branch2c	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$1 \times 1 \times 64 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
res2a_branch1	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$1 \times 1 \times 64 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn2a_branch2c	BatchNormalization	Input shape	$56 \times 56 \times 256$	$128 \times 104 \times 256$
			256	
		Trainable	256	
bn2a_branch1	BatchNormalization	Input shape	$56 \times 56 \times 256$	$128 \times 104 \times 256$
			256	
		Trainable	256	
add	Add	Input shape	$56 \times 56 \times 256$	$128 \times 104 \times 256$
activation_3	Activation	Input shape	$56 \times 56 \times 256$	$128 \times 104 \times 256$
res2b_branch2a	Conv2D	Input shape	$56 \times 56 \times 256$	$128 \times 104 \times 256$
			$1 \times 1 \times 256 \times 64$	
		Trainable	64	
		Stride	(1, 1)	
bn2b_branch2a	BatchNormalization	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			64	
		Trainable	64	
activation_4	Activation	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
res2b_branch2b	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$3 \times 3 \times 64 \times 64$	
		Trainable	64	
		Stride	(1, 1)	
bn2b_branch2b	BatchNormalization	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			64	
		Trainable	64	
activation_5	Activation	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
res2b_branch2c	Conv2D	Input shape	$56 \times 56 \times 64$	$128 \times 104 \times 64$
			$1 \times 1 \times 64 \times 256$	
		Trainable	256	

Layer	Type		reduced_resnet	resnet
		Stride	(1, 1)	
bn2b_branch2c	BatchNormalization	Input shape	56 × 56 × 256	128 × 104 × 256
			256	
		Trainable	256	
add_1	Add	Input shape	56 × 56 × 256	128 × 104 × 256
activation_6	Activation	Input shape	56 × 56 × 256	128 × 104 × 256
res2c_branch2a	Conv2D	Input shape	56 × 56 × 256	128 × 104 × 256
			1 × 1 × 256 × 64	
		Trainable	64	
		Stride	(1, 1)	
bn2c_branch2a	BatchNormalization	Input shape	56 × 56 × 64	128 × 104 × 64
			64	
		Trainable	64	
activation_7	Activation	Input shape	56 × 56 × 64	128 × 104 × 64
res2c_branch2b	Conv2D	Input shape	56 × 56 × 64	128 × 104 × 64
			3 × 3 × 64 × 64	
		Trainable	64	
		Stride	(1, 1)	
bn2c_branch2b	BatchNormalization	Input shape	56 × 56 × 64	128 × 104 × 64
			64	
		Trainable	64	
activation_8	Activation	Input shape	56 × 56 × 64	128 × 104 × 64
res2c_branch2c	Conv2D	Input shape	56 × 56 × 64	128 × 104 × 64
			1 × 1 × 64 × 256	
		Trainable	256	
		Stride	(1, 1)	
bn2c_branch2c	BatchNormalization	Input shape	56 × 56 × 256	128 × 104 × 256
			256	
		Trainable	256	
add_2	Add	Input shape	56 × 56 × 256	128 × 104 × 256
activation_9	Activation	Input shape	56 × 56 × 256	128 × 104 × 256
res3a_branch2a	Conv2D	Input shape	56 × 56 × 256	128 × 104 × 256
			1 × 1 × 256 × 128	
		Trainable	128	
		Stride	(2, 2)	
bn3a_branch2a	BatchNormalization	Input shape	28 × 28 × 128	64 × 52 × 128
			128	
		Trainable	128	
activation_10	Activation	Input shape	28 × 28 × 128	64 × 52 × 128
res3a_branch2b	Conv2D	Input shape	28 × 28 × 128	64 × 52 × 128
			3 × 3 × 128 × 128	
		Trainable	128	
		Stride	(1, 1)	
bn3a_branch2b	BatchNormalization	Input shape	28 × 28 × 128	64 × 52 × 128
			128	
		Trainable	128	
activation_11	Activation	Input shape	28 × 28 × 128	64 × 52 × 128
res3a_branch2c	Conv2D	Input shape	28 × 28 × 128	64 × 52 × 128
			1 × 1 × 128 × 512	
		Trainable	512	
		Stride	(1, 1)	
res3a_branch1	Conv2D	Input shape	56 × 56 × 256	128 × 104 × 256
			1 × 1 × 256 × 512	
		Trainable	512	
		Stride	(2, 2)	
bn3a_branch2c	BatchNormalization	Input shape	28 × 28 × 512	64 × 52 × 512
			512	
		Trainable	512	
bn3a_branch1	BatchNormalization	Input shape	28 × 28 × 512	64 × 52 × 512
			512	
		Trainable	512	
add_3	Add	Input shape	28 × 28 × 512	64 × 52 × 512
activation_12	Activation	Input shape	28 × 28 × 512	64 × 52 × 512
res3b_branch2a	Conv2D	Input shape	28 × 28 × 512	64 × 52 × 512

Layer	Type		reduced_resnet	resnet
			$1 \times 1 \times 512 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3b_branch2a	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	
activation_13	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3b_branch2b	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$3 \times 3 \times 128 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3b_branch2b	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	
activation_14	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3b_branch2c	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$1 \times 1 \times 128 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn3b_branch2c	BatchNormalization	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			512	
		Trainable	512	
add_4	Add	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
activation_15	Activation	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
res3c_branch2a	Conv2D	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			$1 \times 1 \times 512 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3c_branch2a	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	
activation_16	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3c_branch2b	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$3 \times 3 \times 128 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3c_branch2b	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	
activation_17	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3c_branch2c	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$1 \times 1 \times 128 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn3c_branch2c	BatchNormalization	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			512	
		Trainable	512	
add_5	Add	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
activation_18	Activation	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
res3d_branch2a	Conv2D	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			$1 \times 1 \times 512 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3d_branch2a	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	
activation_19	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3d_branch2b	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$3 \times 3 \times 128 \times 128$	
		Trainable	128	
		Stride	(1, 1)	
bn3d_branch2b	BatchNormalization	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			128	
		Trainable	128	

Layer	Type		reduced_resnet	resnet
activation_20	Activation	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
res3d_branch2c	Conv2D	Input shape	$28 \times 28 \times 128$	$64 \times 52 \times 128$
			$1 \times 1 \times 128 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn3d_branch2c	BatchNormalization	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			512	
		Trainable	512	
add_6	Add	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
activation_21	Activation	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
res4a_branch2a	Conv2D	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			$1 \times 1 \times 512 \times 256$	
		Trainable	256	
		Stride	(2, 2)	
bn4a_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_22	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4a_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4a_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_23	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4a_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	1,024	
		Stride	(1, 1)	
res4a_branch1	Conv2D	Input shape	$28 \times 28 \times 512$	$64 \times 52 \times 512$
			$1 \times 1 \times 512 \times 1,024$	
		Trainable	1,024	
		Stride	(2, 2)	
bn4a_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			1,024	
		Trainable	1,024	
bn4a_branch1	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			1,024	
		Trainable	1,024	
add_7	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_24	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res4b_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4b_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_25	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4b_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4b_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_26	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4b_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	1,024	
		Stride	(1, 1)	
bn4b_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$

Layer	Type		reduced_resnet	resnet
			1,024	
		Trainable	1,024	
add_8	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_27	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res4c_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4c_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_28	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4c_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4c_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_29	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4c_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	1,024	
		Stride	(1, 1)	
bn4c_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			1,024	
		Trainable	1,024	
add_9	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_30	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res4d_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4d_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_31	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4d_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4d_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_32	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4d_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	1,024	
		Stride	(1, 1)	
bn4d_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			1,024	
		Trainable	1,024	
add_10	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_33	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res4e_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4e_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_34	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4e_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$

Layer	Type		reduced_resnet	resnet
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4e_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_35	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4e_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	$1,024$	
		Stride	(1, 1)	
bn4e_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1,024$	
		Trainable	$1,024$	
add_11	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_36	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res4f_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4f_branch2a	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_37	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4f_branch2b	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$3 \times 3 \times 256 \times 256$	
		Trainable	256	
		Stride	(1, 1)	
bn4f_branch2b	BatchNormalization	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			256	
		Trainable	256	
activation_38	Activation	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
res4f_branch2c	Conv2D	Input shape	$14 \times 14 \times 256$	$32 \times 26 \times 256$
			$1 \times 1 \times 256 \times 1,024$	
		Trainable	$1,024$	
		Stride	(1, 1)	
bn4f_branch2c	BatchNormalization	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1,024$	
		Trainable	$1,024$	
add_12	Add	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
activation_39	Activation	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
res5a_branch2a	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$
			$1 \times 1 \times 1,024 \times 512$	
		Trainable	512	
		Stride	(2, 2)	
bn5a_branch2a	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_40	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5a_branch2b	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$3 \times 3 \times 512 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn5a_branch2b	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_41	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5a_branch2c	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$1 \times 1 \times 512 \times 2,048$	
		Trainable	$2,048$	
		Stride	(1, 1)	
res5a_branch1	Conv2D	Input shape	$14 \times 14 \times 1,024$	$32 \times 26 \times 1,024$

Layer	Type		reduced_resnet	resnet
			$1 \times 1 \times 1,024 \times 2,048$	
		Trainable	$2,048$	
		Stride	(2, 2)	
bn5a_branch2c	BatchNormalization	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$2,048$	
		Trainable	$2,048$	
bn5a_branch1	BatchNormalization	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$2,048$	
		Trainable	$2,048$	
add_13	Add	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
activation_42	Activation	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
res5b_branch2a	Conv2D	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$1 \times 1 \times 2,048 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn5b_branch2a	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_43	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5b_branch2b	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$3 \times 3 \times 512 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn5b_branch2b	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_44	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5b_branch2c	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$1 \times 1 \times 512 \times 2,048$	
		Trainable	$2,048$	
		Stride	(1, 1)	
bn5b_branch2c	BatchNormalization	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$2,048$	
		Trainable	$2,048$	
add_14	Add	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
activation_45	Activation	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
res5c_branch2a	Conv2D	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$1 \times 1 \times 2,048 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn5c_branch2a	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_46	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5c_branch2b	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$3 \times 3 \times 512 \times 512$	
		Trainable	512	
		Stride	(1, 1)	
bn5c_branch2b	BatchNormalization	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			512	
		Trainable	512	
activation_47	Activation	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
res5c_branch2c	Conv2D	Input shape	$7 \times 7 \times 512$	$16 \times 13 \times 512$
			$1 \times 1 \times 512 \times 2,048$	
		Trainable	$2,048$	
		Stride	(1, 1)	
bn5c_branch2c	BatchNormalization	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
			$2,048$	
		Trainable	$2,048$	
add_15	Add	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
activation_48	Activation	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
avg_pool	GlobalAveragePooling2D	Input shape	$7 \times 7 \times 2,048$	$16 \times 13 \times 2,048$
fc1000	Dense	Input shape	$2,048$	

Layer	Type		`reduced_resnet`	`resnet`
			$2,048 \times 2$	
		Trainable	2	