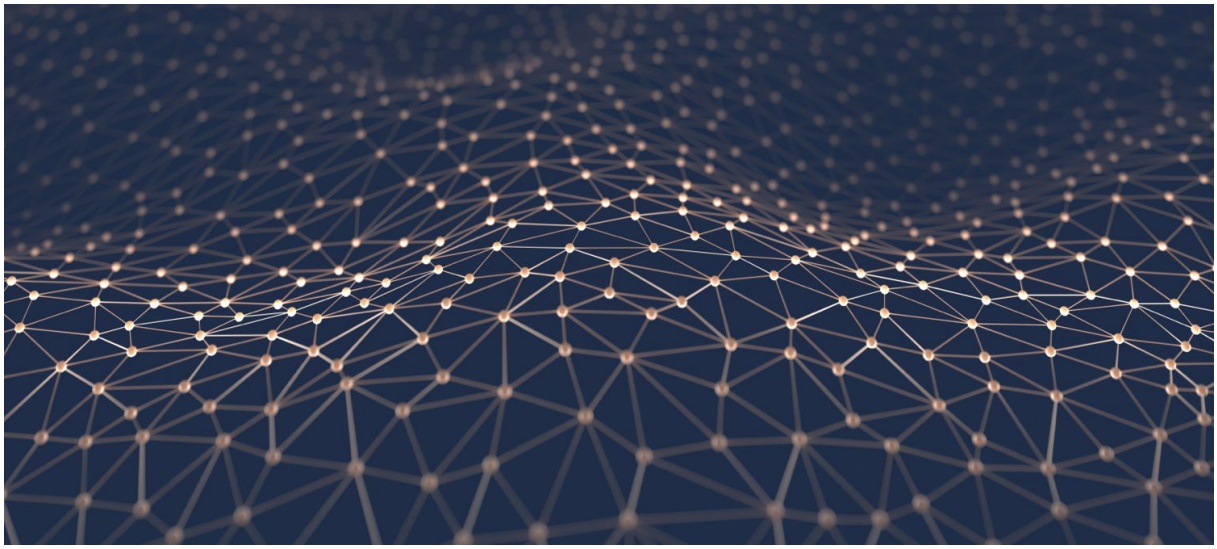

Visualisation Tool for Convolutional Neural Networks

Project written by:

Name:	Signature:
Aron MAROSSY	_____

September 7, 2018

Title:
Visualisation Tool for Convolutional Neural Networks

Theme:
Master Thesis

Project period:
June - September 2018

Project group:
Group 4.6

Members:
Aron Marossy

Supervisor:
Per Lynggaard

No. of Pages: 62

Total no. of pages: 70

Finished: September 7, 2018

Aalborg University Copenhagen
A. C Meyers Vænge 15
2450 København SV
Secretary: Maiken Keller
Telephone (+45) 9940 2471
mks@staff.aau.dk

Abstract:

The purpose of this thesis is to develop a solution for visualising convolutional neural networks. A proposed system has been developed, which gives insight into the workings of a neural network with the means of visualising its architecture and showing how it behaves with specific cases of input data. This has been achieved through a Python application which is running locally, with an interface available through a web browser.
The tool works by the user specifying where the neural network is located on their computer and then providing the dataset on which the network is to be executed.
It is also an aspect of this thesis to provide insight into how to improve the performance of the neural network, which is done by detecting typical problems in the network and by suggesting possible solutions to them.

When uploading this document to Digital Exam each group member confirms that all have participated equally in the project work and that they collectively are responsible for the content of the project report. Furthermore each group member is liable for that there is no plagiarism in the report.

Contents

1| Introduction

Neural Networks are becoming more and more accessible and popular, with out of the box software solutions being widely used for creating them. These programs, such as Torch, Tenserflow or Keras let anyone with minimal programming skills and a large enough dataset to create neural networks for a multitude of purposes, after a few minute long installation and configuration process. These software, however accessible they might be, still hold a large barrier to entry as they only provide limited feedback and analysis tools for the neural networks outputted - at least at the time of writing of this thesis.

This issue can be traced back to the origin of these software solutions, as they were originally intended for experts and enthusiasts, and because of this for most of the tools only a command line interface is available, coming without a proper graphical interface and with only minimal feedback to the users, as shown on **Figure 1.1** with one of the most popular software, which provides ony minimal information about the training process and the resulting network.

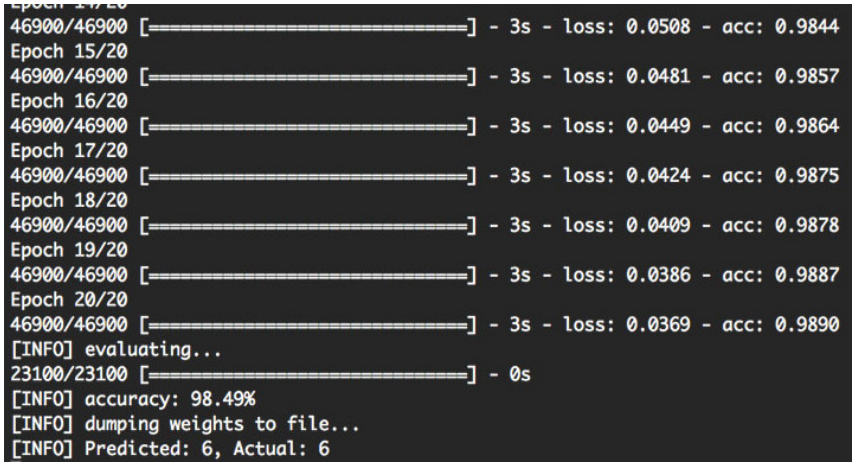

Figure 1.1: A neural network being trained for recognising digits with a commonly used over-the-shelf program called Keras. The software outputs the neural network as a matrix and gives no other feedback than the neural network's k-fold performance [1].

Such a spartan command line based interface could be "good enough" for experts, but as more and more researchers from a broad spectrum of sciences are starting to use neural networks, a more self explanatory and educative framework is due.

This issue is further complicated by the nature of neural networks, as they have originally been invented to solve complex non-linear problems, which are difficult to model and give feedback on. Because of this, the network can learn unusual or unexpected things about the dataset, which

might not originally have been the intention. As such, a neural network might be preforming well on a given set of data, but may show strange behaviour when presented with new measurements. This was the case in the 1980s, when the Pentagon in the United States ordered a new software from a research institution for identifying camouflaged tanks on images. The researchers have taken 200 pictures, half of which with tanks and the other half with just trees and sceneries. Then they trained a convolutional neural network on half of this dataset with 50 pictures of tanks and the rest without, and the resulting neural network was able to correctly identify the remaining images with high accuracy. This network was then sent to the Pentagon, who later determined the network was not performing better than random chance [2].

As it turned out, the researches have taken the pictures with tanks on them on sunny days and the pictures with scenery on cloudy days, and all the neural network did was learn how to classify pictures based on this property [2]. Because the neural networks built by algorithms are just matrices by themselves, not too descriptive to humans, it could be challenging to gain insight into the inner working of such a network.

This issue with the tanks could have been prevented with a software solution meant for analysing the output of the network, which may have provided insight for example what parts of image was the neural network taking into account for identifying the pictures. Such a solution might also educate people on the workings of neural networks, as the lack of such insights might leave some researchers treating neural networks as a mysterious black box.

This thesis aims to propose a tool to solve such problems by analysing already existing tools and methods for understanding convolutional neural networks and to lower the barrier to entry for people getting started with neural networks. This will be done by proposing a solution on how to make the behaviour of neural networks more transparent for researchers and people working in the industry.

1.1 Limitations

To limit the scope of this thesis, some limitations have been set, as the field of neural networks is rather big and quickly evolving each year. This thesis will only be focusing on convolutional neural networks, as image recognition tasks (also referred to as "computer vision") are one of the most common tasks at the time of writing of this thesis, as the amount of images and videos generated over the Internet steadily grows each year, not to mention novel uses such as self driving cars, uses for medical diagnosis and intelligent factory assembly lines.

As for keeping technical complexity from getting out of hand, this thesis will only take instance in feed-forward neural networks and static, non-changing neural network architectures.

When discussing how to improve networks, this thesis does not intend to delve into the topic on how to train neural networks in a more efficient or faster way. So when discussing on how to improve on a model, this is done with the aim of getting a better classification performance, not the training speed itself. However the training speed might be an important topic to discuss as well, it is out of scope for this thesis, and will only be mentioned an additional note at times.

1.2 Problem Formulation

To specify the aim of this project and to narrow down its scope, the following problem formulation has been created, with three additional subquestions;

How to design a tool for visualising and understanding the behaviour of convolutional neural networks?

The sub-questions are listed below. These are to ensure that the problem formulation is to be fully answered in the conclusion of this thesis. They also help to further narrow down and focus the research areas of this thesis.

- How to tell if there is an issue with the network's performance, and how to educate users about this?
- How to make neural networks more accessible to people just getting started with them?
- How to provide understanding for the input image in relation to the activations of the neural network?

1.3 Methodology

This paper was written following an agile methodology, to provide a framework for the research and development process. This framework is illustrated on Figure 1.2.

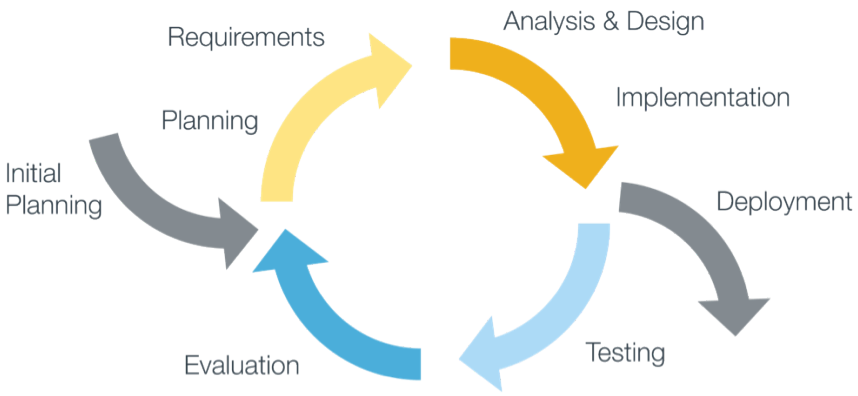

Figure 1.2: The agile methodology used for this project [3].

In the case of this thesis, after a short initial planning phase some basic initial requirements have been formulated. Based on these initial requirements, a more in-depth analysis took place, after which a mock up design has been created, which was a basis for an initial prototype. The project has gone through a number of these iterative cycles, resulting in the final product discussed in the conclusion.

There has also been some overlapping when work was done on the State of the Art, Analysis and Design chapters, as the analysis of certain technologies raised the need to investigate more into certain fields. This approach was in line with the general ideology of agile methodologies, and has been proven very useful in writing a nicely rounded thesis.

The research methodology used throughout this thesis is abductive, as it starts from the point of view of the currently existing but incomplete tools and theories, with the aim of trying to find a solution to them. It is not the intention of this thesis to find a be-all-end-all solution to the problem of visualising neural networks, such as discussed in the limitations in **Chapter 1.1**.

1.4 Expected Outcome

This thesis is to create a prototype for an application to analyse convolutional neural networks by first laying down the requirements based on an analysis of state of the art solutions and industry practices. It is not the intent of this thesis to thoroughly delve into the mathematical principles of each disciples of neural networks, although such principles will be discussed only to the extent necessary for the implementation of each part.

1.5 Challenges

Throughout the writing of this thesis there were some challenges which made the work on this thesis at times unusually difficult, and the aim of this section is to take a short mention of them.

1.5.1 Terminology

The terminology used for neural networks is fairly unstandardised, with many of the terms being used interchangeably. For example, the term for cost function is often referred to as objective function or loss [4], even though it is one of the most commonly used terms when discussing neural networks. This thesis will aim to stick for one set of terminology for the most part, but it may refer to other terminology when referring to research papers or documentation of applications, but in these cases clarification is done shortly afterwards.

1.5.2 Rapidly changing area

There are many sources and publications, with some of them being valid and used for years, and some of them going obsolete relatively quickly. During the writing of this thesis, at times it has been a challenge to see, for example if a paper was referenced in publications from a few years ago, but published in the 80s was still relevant, or has been superseded by newer developments in the field. For example [5] from 1989 is widely referenced, but its contents are very outdated. It has also been at times challenging having to decide how relevant some of the newer publications are, as the field of neural networks is being quite actively researched, and the amount of publications being released every year is quite high [6].

1.5.3 Rabbit hole of references

Another challenge when writing about neural networks is the rather long chains of references between papers. Papers often seem to be building on works that has came before, which in turn also builds on other findings and papers, and because of this, the author of this paper sometimes found himself wanting to investigate one paper and ending up by going through more than ten of them to understand a mechanism or a finer point a paper was aiming to make [7].

1.5.4 No documentation for software

Most of the software being discussed in the State of the Art chapter of this report is open source, with contributors from all over the world. The problem with such software is that the

contributors do not always write documentation for their additions, and this seems to be the case here as well. Most of the programs discussed have missing or not up to date documentation, which makes it difficult to discuss them or - in some cases - even using them can be challenging.

2| State of the Art

The aim of this chapter is to present the technologies and existing solutions related to the visualisation of convolutional neural networks, which serves as a basis for future chapters to build upon. This is done by first examining some core principles of such networks and then delving into the already existing solutions related to the problem area.

2.1 Principles of Neural Networks

This section aims to establish some terminology and principles of neural networks to be used later on in this chapter and for the rest of this thesis. Since all forms of more specialised neural networks, such as convolutional networks for image recognition - as is the main focus of this thesis - or long short-term memory networks for speech recognition are built on the same foundations, but with added specific characteristics, the principles and workings of standard "vanilla" neural networks are discussed beforehand.

The "network" part of the neural network comes from the fact that they consist of interconnected neurons. This setup is inspired by biology, as each neuron activating (or firing) in the network cascades down to other neurons which are connected to it, such as in biological brains as illustrated on the following **Figure 2.1**;

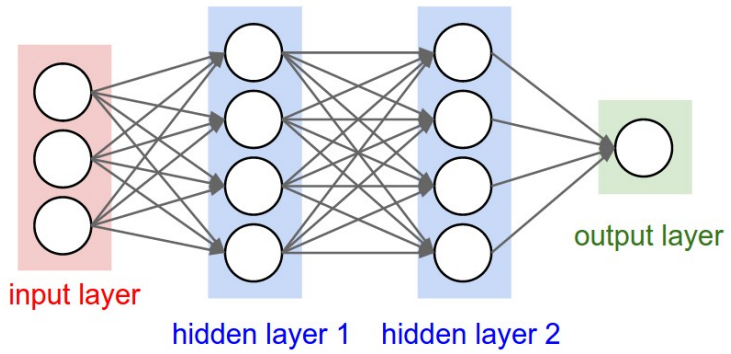

Figure 2.1: An example of a feed forward neural network [8].

As shown on **Figure 2.1**, all neural networks can be divided to three separate parts, an input layer with the inputs to the network (such as the training data), a hidden layer which does the "logic" of the neural network, and an output layer which usually outputs a number of values between 0 and 1, which corresponds to how likely the network classifies the current input to fall into a given category.

Each neuron has an activation, which is a number between 0 and 1. In the input layer, this activation changes based on the input data, and in the case of the later neurons in the network it is based on what input from the neurons in the layers before them. This input is influenced by the connections between the neurons, which are called weights, which hold a rational number. Each neuron gets an input value, which is the sum of the values of its connected neurons from the earlier layer, multiplied by their weight, as expressed the following equation;

$$x = w_1a_1 + w_2a_2 + w_3a_3 + ... + w_na_n$$

This value is then tresholded with the use of an activation function, which then can be used as a part of an input for another neuron later down the network.

2.1.1 Activation functions

Each neuron's activation after the input layer depends on their input value plotted to its activation function. Specific types of activation functions will be discussed later on in this chapter, but their working principle is that they act as a normaliser, by making sure the activation of a neuron is in a given threshold, most commonly between -1 and 1 or 0 and 1. For example, a historically commonly used activation function is the sigmoid function, which is shown on the following **Figure 2.2**;

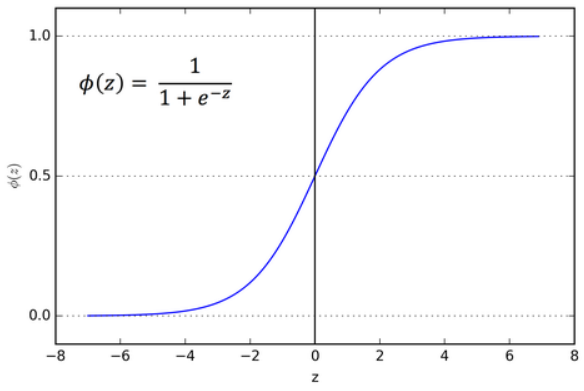

Figure 2.2: The sigmoid activation function and its formula [9]

The sigmoid function "squishes" the activation between 0 and 1, with the output of the neuron being 0 if the input is very small and 1 if the activation is very large, other values being in between. Which activation function is to be used by a specific neural network is chosen before training the network, and the use of each specific activation function has its upsides and

downsides, which will be discussed in a later section of this chapter.

Additionally, a bias can be added to a neuron to make its activation more or less likely. This bias is trained alongside with the network, and is added to the sum before it is passed to the activation function.

2.1.2 Initialising

To set up a network, first an architecture is chosen which specifies a number of layers and a number of neurons for each of those layers. For each of these neurons an activation function is chosen, which is usually the same for every neuron in the network. This chapter is to go into detail in the effects of different architectures and activation functions later on.

Once a blank architecture is set up, the network is initialised by assigning a random value for every weight and bias in the network, with the intention of changing them later by modifying them to improve performance over time. After the initialisation, the resulting neural network will be effectively classifying the inputs randomly, and it will have to be fine tuned by the use of the training data, but to rate its performance a function for measuring how far off the network is from performing optimally is needed.

2.1.3 Cost Function

The cost function is meant to measure how well each iteration of the neural network is performing, with a higher cost meaning a worse performance of the network. A common way of calculating cost is by the use of the square of differences, where for each output of the network the output is substituted from the expected output, squared (which also avoids negative values), and these values are then summed together. This function could be thought of as a plot, where each weight corresponds to an axis, with the cost function plotted over them, where the resulting plot represents the cost function for each possible value for the weights. In reality, this is not really feasible, as this plot would have as many dimensions plus one as many weights are in the network, and it can be difficult to visualise a plot of anything above three dimensions. But for a simplified example, the cost function of a neural network with two weights could look like the function shown on **Figure 2.3**.

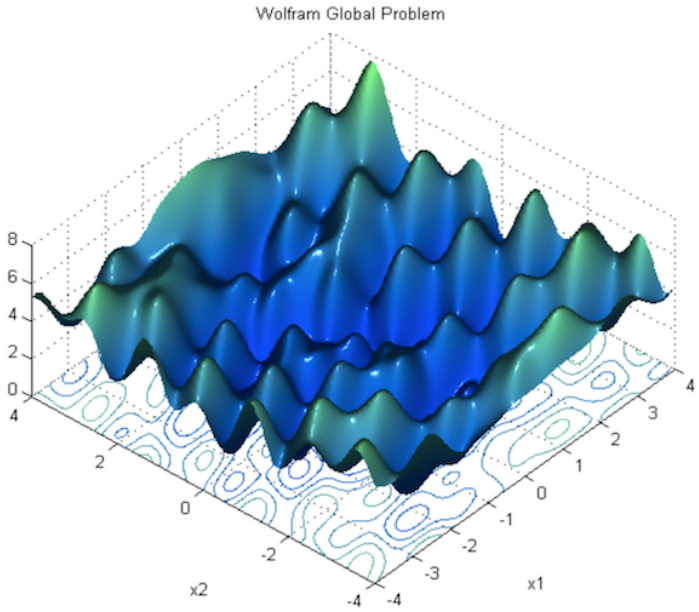

Figure 2.3: The possible cost function of an over simplified neural network with only two weights is a non-convex surface [10].

The goal when training and tweaking neural networks is to find a set of weights which leads to a neural network with the lowest possible cost function. It is important to note that the cost function of a high dimensional space can be extremely complex, and because of this finding the most optimal - or the lowest - point of it might not be feasible. Because of this a technique called gradient descent is often used for finding a local minima on this cost function.

2.1.4 Gradient Descent

Since the weights of the network are randomly initialised at the beginning, each neural network can be thought of as starting at a random point on its n dimensional cost function. It might not be straightforward to visualise this, but this n dimensional function also has a downward slope, and by using multi variable calculus, this slope can be determined on it. By adjusting the weights appropriately in the way that the network's cost function moves in the direction of the steepest descent, the network will perform more and more optimally over time. The gradient descent algorithm can be implemented in a number of ways, but these implementations should only effect the training speed of the network and not its performance [11], and as written in the problem formualtion in **Chapter 1.2**, this thesis only takes instance in the performance and behaviour of neural networks, so quickening the training process is out of scope of this thesis.

2.1.5 Backpropogation

Backpropogation is the process of updating the weights and biases of the neural network throughout the training process to find a local minima of the cost function. As of the problem formualtion in **Chapter 1.2**, this thesis takes account in already trained network and its properties, backpropogation will not be discussed in detail, but if the reader is interested reading more about it [4] and [12] are discussing it thoroughly.

2.1.6 Training Process and Overfitting

Throughout the training of the network, it is possible to overtrain it by not stopping the training process early enough. This will cause the network to perform well on the training dataset, as the network basically memorises each training example, but will cause worse performance on new instances it has not seen before [4]. This phenomena is shown on **Figure 2.4**.

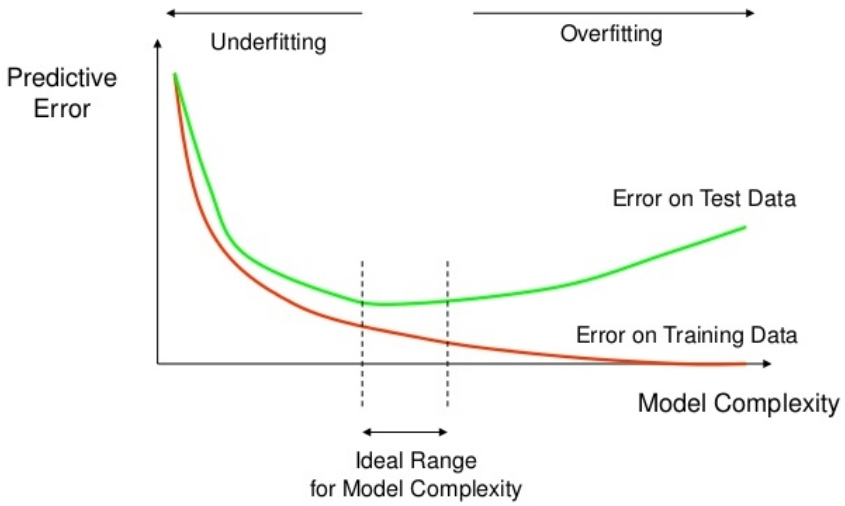

Figure 2.4: If the neural network is trained for too long it loses its ability to generalise and starts to overfit based on the training data. Because of this, there is an ideal time range when the taining should be stopped [13].

This is the reason when training neural networks a slice of the data set is kept separately from the training data, and is not used in the training of the network at all. The true performance of the network can only be measured with such test data, otherwise the neural network could simply overfit to this as well.

2.2 Principles of Convolutional Networks

As discussed in the limitations in **Chapter 1.1**, this thesis focuses on convolutional neural networks, so the aim of this section is to set the terminology for convolutional networks for the rest of this thesis.

Convolutional Networks or ConvNets are a specific type of neural networks used in the field of image recognition. They build upon the same working principles as the regular neural networks, and are also made of neurons with learnable weights and biases, but are structured according to some specific rules. This is because for a convolutional neural network each pixel of a colored input image corresponds to three input vectors. So in the case of the commonly used CIFAR-10 dataset, each of the images are 32x32 pixels in size, which results in a 3x32x32 image due to the RGB colour channels each being a separate input vector. So even in the case of such a small picture, the network has 3072 input vectors, which requires for a special architecture which supports down sampling and other specialised steps, as discussed later.

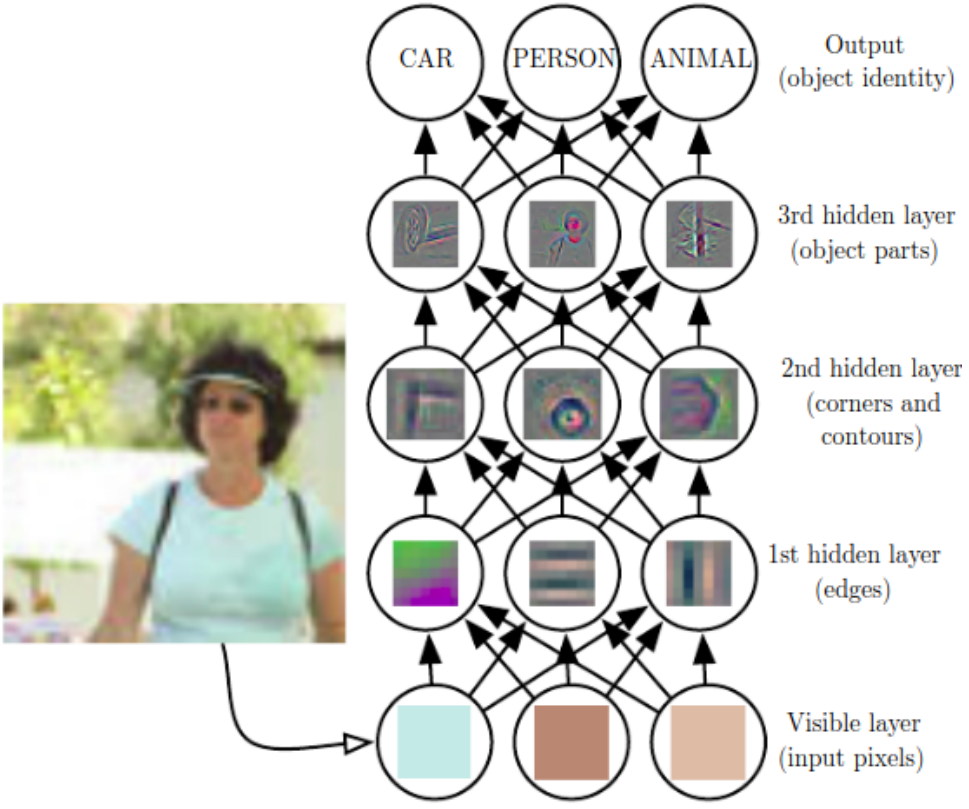

Figure 2.5: Convolutional networks follow the same principle as regular neural network, with the pixels of the picture being the input, the network learning more general and high level principles in its earlier layers, getting more and more specialised in the deeper layers, and returning a classification in the output layer [14].

For this reason, ConvNets use a technique called kerneling for reducing the amount of inputs to its first hidden layer. This is done by an n by n kernel (often 5 by 5 in practice [4]) and mapping that to a single neuron, as shown on **Figure 2.6**.

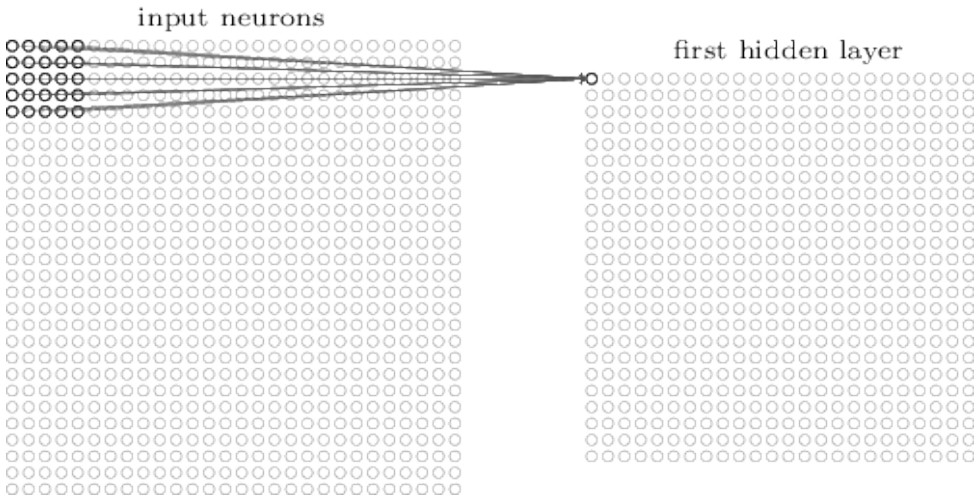

Figure 2.6: Mapping a part of the input picture with a 5 by 5 kernel to the first hidden layer, significantly reducing the amount of input being fed to the network [4].

Because of this, each filter is characterized by two properties; kernel size (25 in case of the example on **Figure 2.6**) and stride, the amount the kernel is moved after each sampling to the right or to a new row. The stride is often 1 or 2, as moving the filter by more is usually causes too big drops in the performance of the convolutional network generated [4], but as it is a hyperparameter this can be set to any value, which is to be discussed later on in this chapter.

2.2.0.1 CNN architectures

As opposed to plain neural networks, convolutional neural networks use specialised architectures for optimizing and enhancing performance. These architectures have historically been created for an annual competition called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which have been held every year since 2010 [15]. Each of these architectures is very well documented by their original authors, and have been benchmarked by numerous people since their inception. In [16] it is shown that each of the main CNN architectures have a well observable difference in their classification accuracy, as it is also shown on **Figure 2.7**

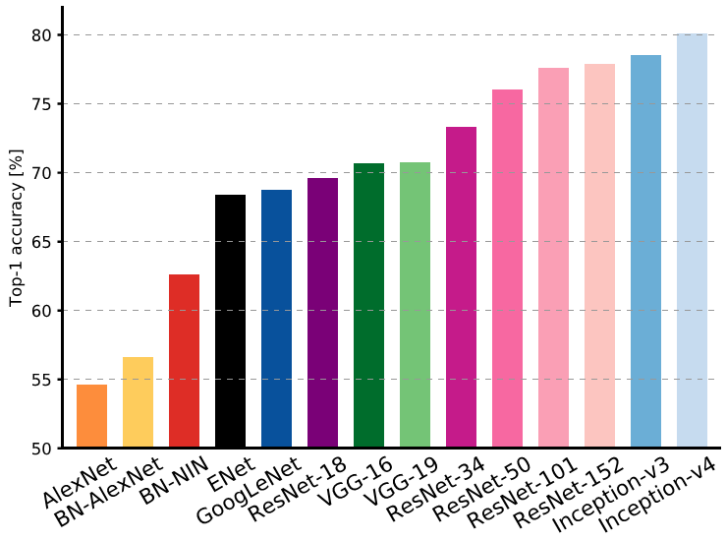

Figure 2.7: Validation accuracies of CNN architectures on the Imagenet dataset [16].

It has also been benchmarked how these architectures compare in the size of the network outputted in relation to the classification accuracy, as it is shown on **Figure 2.8**

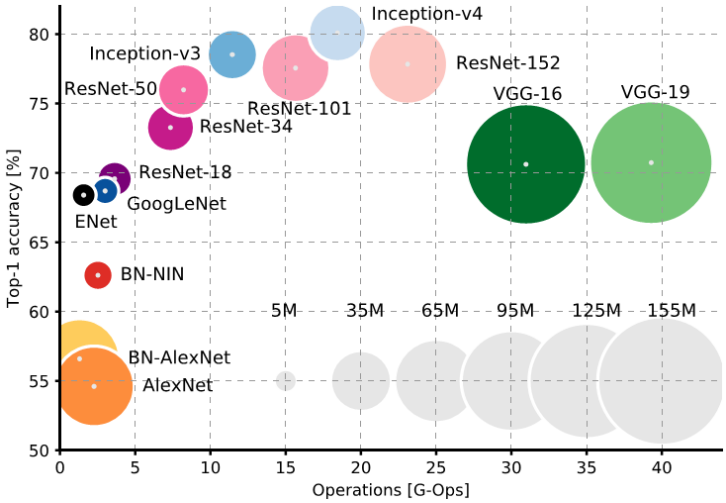

Figure 2.8: Chart showing the operations required for a single forward pass with the network (on the x-axis) in comparison to the classification accuracy of each model (y-axis). The size of each disk is proportional to the size of the neural net produced [16].

As listed on **Figure 2.7** and **Figure 2.8**, there are a number of commonly used CNNs for image classifying tasks. The following sections of this thesis will discuss the two most commonly used ones of these, in line with sub-question one of the problem formulation listed in **Chapter 1.2**.

2.2.0.2 AlexNet

AlexNet was the first large scale neural network architecture that performed really well on the earlier mentioned ILSVRC competition, outperforming all non-deep learning based models, and starting the revolution of modern convolutional neural networks in 2012 [17]. AlexNet's architecture is shown on the following **Figure 2.9**;

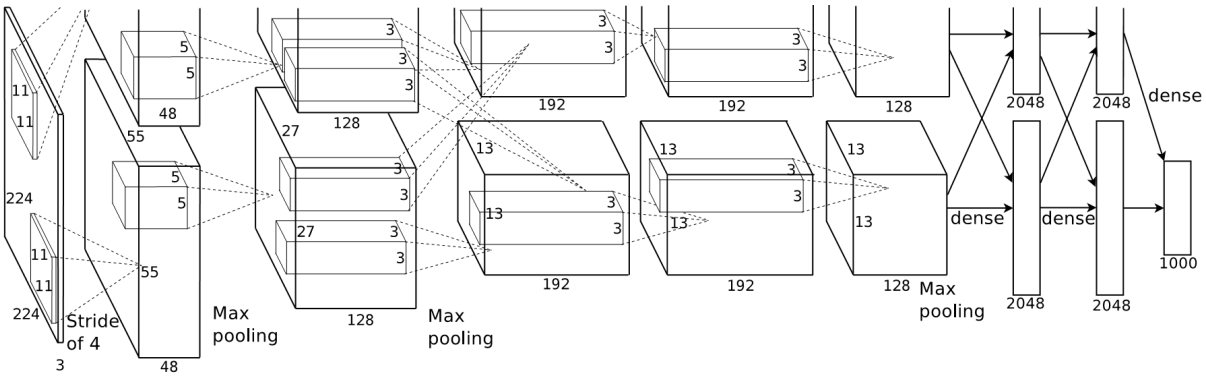

Figure 2.9: Architecture of the AlexNet, starting with a convolutional layer, leading to a pooling layer, leading to normalisation, then leading to another convolutional, normalisation and then a number of convolutional layers, finishing with a pooling layer and then three fully connected layers. The picture in the original paper is missing its top part, and is cited as such in academic sources as well [17].

Even though AlexNet has been outperformed since its inception in 2012, as shown on **Figure 2.7**, it is still widely popular as it is a very well documented and understood network, with its original paper having been cited over 27.000 times as of the writing of this thesis, half of these citations being from the past two years [18]. This, and the architecture's precalence in online tutorials shows that this architecture is still popular, and is still widely used.

2.2.0.3 GoogLeNet

GoogLeNet is a very deep neural net compared to the previous winners of the ILSVRC competition, which won the challenge in 2015 with its 22 hidden layers [19]. It has been designed to be very efficient with the use of so called "inception" modules, which applies parallel filtering operations to the same input, taking advantage of how GPUs can compute many similar calculations in parallel at a time, with one of these inception modules being shown on **Figure 2.10**.

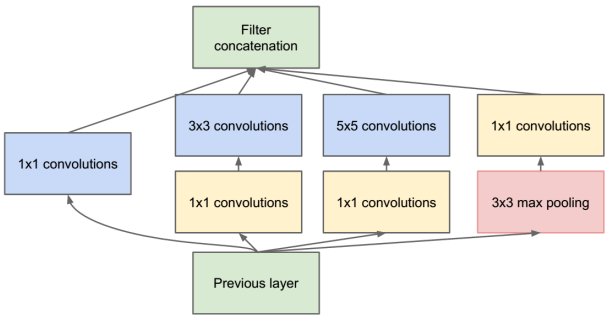

Figure 2.10: An inception module of the GoogleNet, where three seperate convolutions and one pooling is parallelized in a single module. Notice how a 3x3 convolution is preceded by a 1x1 "bottleneck" convolution, and the 3x3 pooling is followed by an additional 1x1 "bottleneck" convolution to reduce the dimensionality of the output filter [19].

The full GoogLeNet architecture is shown on **Figure 2.11**, showing how nine of the earlier mentioned inception modules connect with each other.

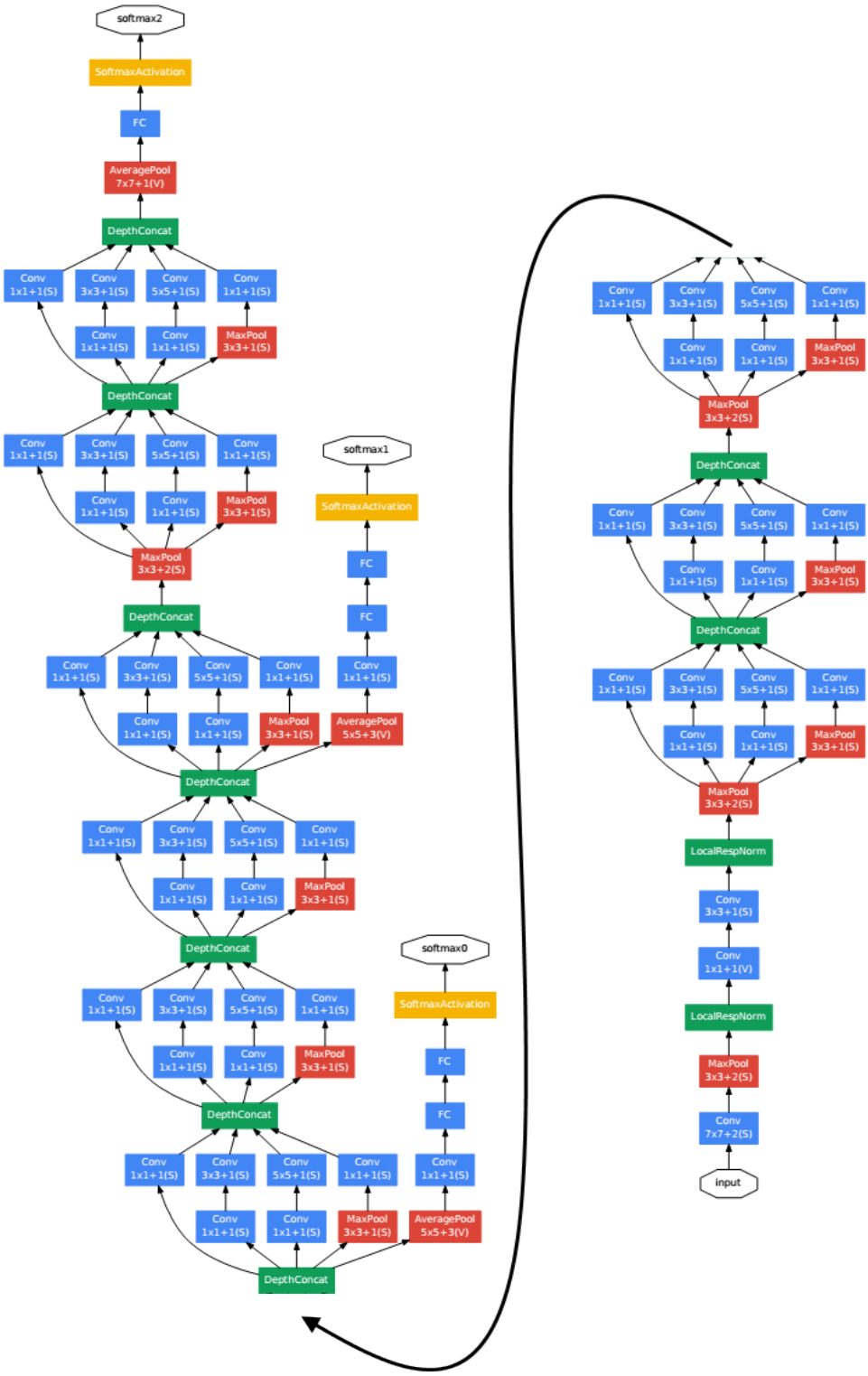

Figure 2.11: The full GoogLeNet architecture. The image has been modified from [19] to fit better on this page.

The GoogLeNet model starts out with a convolutional, a pooling and a normalisation layer, followed by two more convolutional, a normalisation and a pooling layer, after which nine inception modules follow. Then the output of the last one is pooled, which is fed to a final, fully connected layer.

2.2.1 Hyperparameters

In machine learning hyperparameters are constants, which need their values to be predefined before the models could be constructed. These parameters then will influence how the algorithm will learn about the data or how the data will be processed and treated.

This section discusses the most often used hyperparameters for convolutional neural networks, as if these are set incorrectly, they can heavily influence the performance of the resulting network, which is one of the main topics of this thesis as written in the problem formulation in **Chapter 1.2**.

- **Learning rate** - The learning rate (or sometimes called step size) is the most important parameter when training a neural network [20]. It influences how much the weights are adjusted during backpropogation, and if it incorrectly set it can cause the network to converge at a suboptimal local minima. Unfortunately there is no proper way for detecting if an already trained network has been trained with a wrongly set learning rate, and at the time of writing of this thesis for the most part the learning rate is found on a trial-and-error basis, although research is being done on finding a more optimal solution [21].
- **Batch size** - When calculating the derivative for the gradient descent algorithm, it would be too slow (or computationally expensive) to make the calculations for each and every sample, especially if the training data consists of millions of examples [20]. A common approach for addressing this issue is to calculate the gradient in batches, and the gradient calculated from these batches can be used for the gradient descent algorithm. In practice, a high batch size has been observed to reduce the resulting neural network's ability to generalise, and there have been some attempts to provide a precise answer as to why this phenomena occurs, but it as of the writing of this thesis no conclusive evidence has been found [22].
- **Activation function** - As inspired by biological neurons found in brains, each neuron of a neural network receives inputs from its connected neurons and weights, which are then used for producing an activation to be used as inputs for future neurons. This output depends on the activation function, which is a mathematical formula with the goal to normalise the output of the neuron. This activation function is specified for the neurons of the network as a hyperparameter, and usually the entire network uses the same activation

function for every neuron [4]. The most commonly used activation functions, with their respective upsides and downsides are to be discussed in the next section.

- **Kernel size and Stride** - Convolutional neural networks have two unique and important hyperparameters called kernel size and stride, which are discussed in detail in **Chapter 2.2**. These paremeters, however are set together with the choice of architecture of the CNN, and as such further discussing how to set them optimally is out of scope for this thesis, as it would not lead closer to answering the problem formulation.

2.2.2 Activation Functions

As explored in **Chapter 2.2.1**, activation functions are a hyperparameter of neural networks, the choice of which influences the performance of the neural network to a large degree. This section is to explore commonly used activation functions, and their effects on the performance of neural network, such as in line with the problem formulation in **Chapter 1.2**.

2.2.2.1 Sigmoid Activation Function

The sigmoid activation function is based on the mathematical formula of the sigmoid non-linearity, expressed by the following formula;

$$\sigma(x) = 1/(1 + e^{-x})$$

This function takes any real-valued number, and "squashes" it into the range between 0 and 1, as illustrated by the curve on **Figure 2.12**.

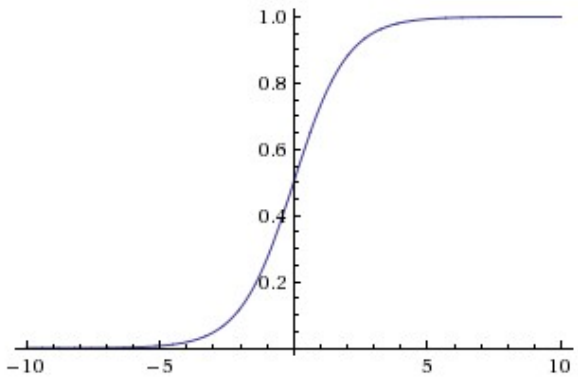

Figure 2.12: Sigmoid non-linearity function, normalizing real numbers to range between 0 and 1 [8].

The sigmoid activation function has seen frequent use historically for neural networks, as it is most similar to how biological brains work, but has two major drawbacks [8]. Firstly, in case of very high or very low inputs to the neuron, the result saturates to the 0 or 1 "tail" of the activation function, which will kill the local gradient, and no changes will be made to that part of the network at the time of backpropogation. The same applies when setting initial weights, if they are too large or too small from the beginning, the neuron will get saturated and will never change. The second downside of the sigmoid activation function is that it is not zero-centered. This can introduce zig-zag dynamics in the gradient updates, as each neuron can turn from positive to negative, and then vica-versa after each update [8].

2.2.2.2 Tanh Activation Function

Tanh, or the Hyperbolic Tangent function has historycally been used to fix some of the issues with the sigmoid function [8]. Its expressed by the following equation.

$$tanh(x) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

This in practice results in a very similar function to the sigmoid, with a comparative illustration shown on **Figure 2.13** shown below;

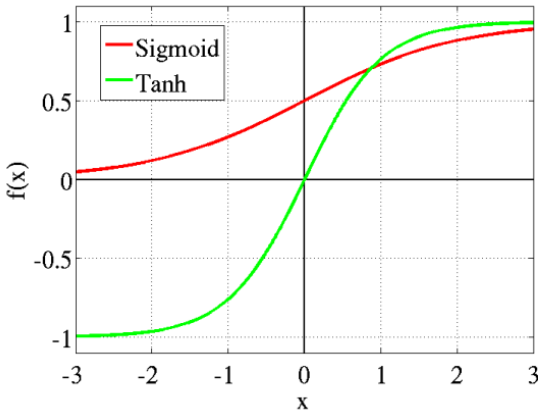

Figure 2.13: A comparison between the Tanh and the Sigmoid functions [9]

An upside of using the tanh function over the sigmoid is that the outputs are zero centered, so the gradient can be updated more efficiently, but it still kills the gradients when the inputs are saturated towards one end of the function [8]. It is mainly used in classification problems between only two classes [9].

2.2.2.3 ReLU Activation Function

The ReLU, or Rectified Linear Unit activation function aims to fix the issues with the sigmoid and tanh functions. A simple $\boldsymbol{f(x) = cx}$ linear function would not be a feasible activation function for a neural network, as then it would not matter how many hidden layer would a network have, the final activation function of the last layer would just simply be a linear function of the input of the first layer, this the neural network would not be able to solve the complex tasks it was intended to do [23]. To solve this issue, ReLU has been used, which has a property of being only partially linear, as expressed by the following equation;

$$f(x) = max(0, x)$$

ReLU has been shown to be a good approximitation for any non-linear function [24], so it is a perfect candidate for an activation function of a neural network. However, the function outputs zero for any input smaller than or equal to zero, as illustrated by its graph shown on **Figure 2.14**.

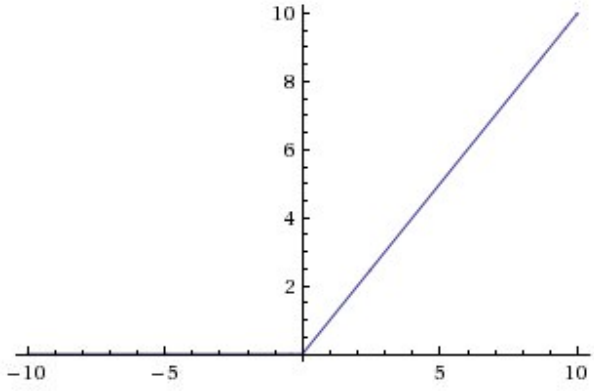

Figure 2.14: The Rectified Linear Unit Activation Function [25].

The problem with this property of ReLU is that after the network has been iniatilised randomly, half of its neurons will be dead, as small nudges to its gradient will not account to any amount of change. Some argue that this more closely models biological neurons, as in reality not all neurons take place in every computation, and that this sparsity might actually be beneficial [26], but in practice there is an activation function called leaky ReLU, which performs better in most cases for convolutional networks [27].

2.2.2.4 Leaky ReLU Activation Function

An improvement over the ReLU, the Leaky ReLU Activation Function introduces a small change. When its input is smaller than zero, it still has an effect on the activation, as shown by its function on the following equation;

$$f(x) = max(0.01x, x)$$

This results in a function which also does not have the issue of saturation as its slope stays constant no matter how much higher or lower is the input than zero. And unlike the regular ReLU, it does not have about half of the neurons being "dead" right after initialisation. A comparison between ReLU and Leaky ReLU is shown on **Figure 2.15** as follows;

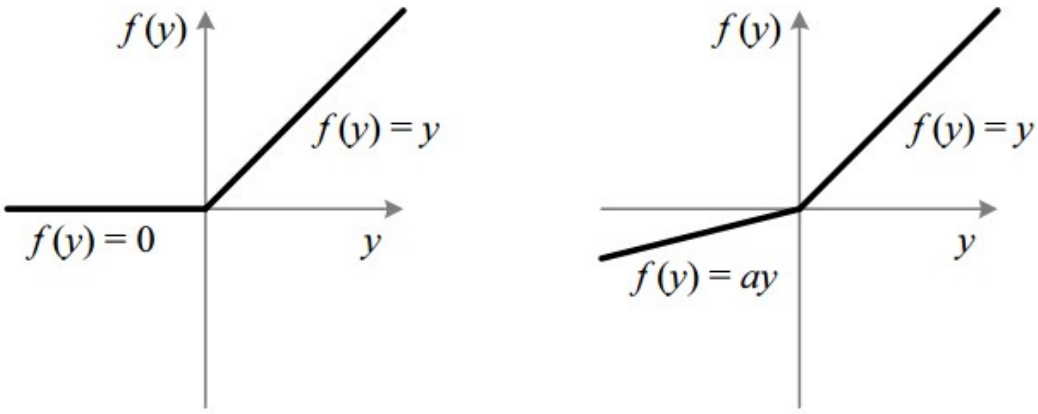

Figure 2.15: The Rectified Linear Unit Activation Function (on the left) as compared to the Leaky Rectified Linear Unit Activation Function (on the right) [9]

It should be mentioned that some other versions of leaky ReLU also exist, such as parametric ReLU, which adjusts the slope when the input is smaller than zero, and exponential linear ReLU, which replaces the smaller than zero "tail" of ReLU with an exponential function, and so on. These additional changes do not offer that significant difference in performance, and so are out of scope of this thesis.

2.3 Theories for Visualizasing Convolutional Neural Networks

The following chapter discusses already existing theories for visualising convoluttional neural networks. These techniques were mainly part of academic papers exploring the area of visuali-

sation, and although pieces of code exist for most of the techniques discussed in this section, it is either published on Github with the need to compile it beforehand or is just partially available.

There have been some works concerning the visualization of convolutional neural networks, most of them focusing on the first layer of the network, where projecting the activations to the pixel space is the most straightforward [28]. This section discusses approaches taken for more comprehensive visualisation techniques, as it has been shown that the first layer usually only handles general features, which are not that significant from the perspective of the problem formulation written in **Chapter 1.2**. It has been shown that these layers even work interchangeably between convolutional neural networks trained to do different tasks, as it has been shown that they always learn features similar to Gabor filters and color blobs, which do not appear to be specific to a particular dataset or task [29].

2.3.1 Partial Occlusion Technique

One of the most cited papers about visualising neural networks is by Matthew D. Zeiler and Rob Fergus from the New York University [30], which discusses a novel visualisation technique for understanding what patterns from the training set activate the feature map of the network. This is done by systematically covering up parts of an image from the training set, and then seeing how this influances the network's classification score for a given class. The following image shows three of these heatmaps generated by the researchers at Stanford University for a demo.

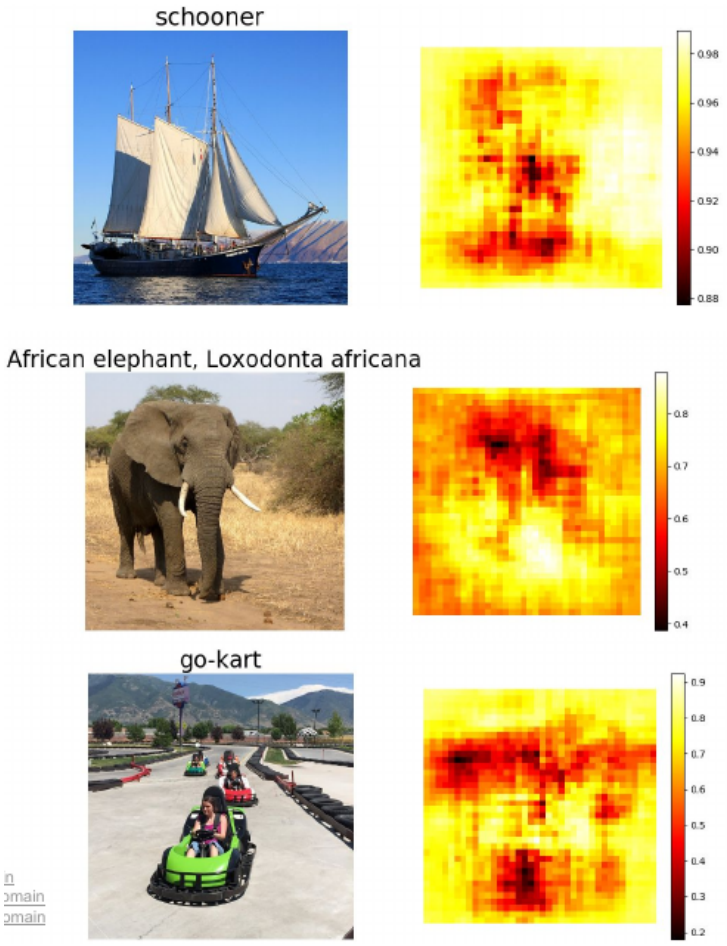

Figure 2.16: Heatmaps created with partial occlusion technique, showing which parts of the image influenced the classification the most [31]

This technique can be used to just simply generate heatmaps as shown above, one for each class which the network is meant to identify, or to create advanced visualisations, such as a single picture showing the highest activations for each possible class.

2.3.2 Saliency Mapping

Saliency mapping is a technique used for convolutional neural networks to see which parts of the image are being analyzed by the network. This is achieved by ranking each pixel's significance in the input image by computing the gradient of the predicted class score in respect to the pixels in the original image [32].

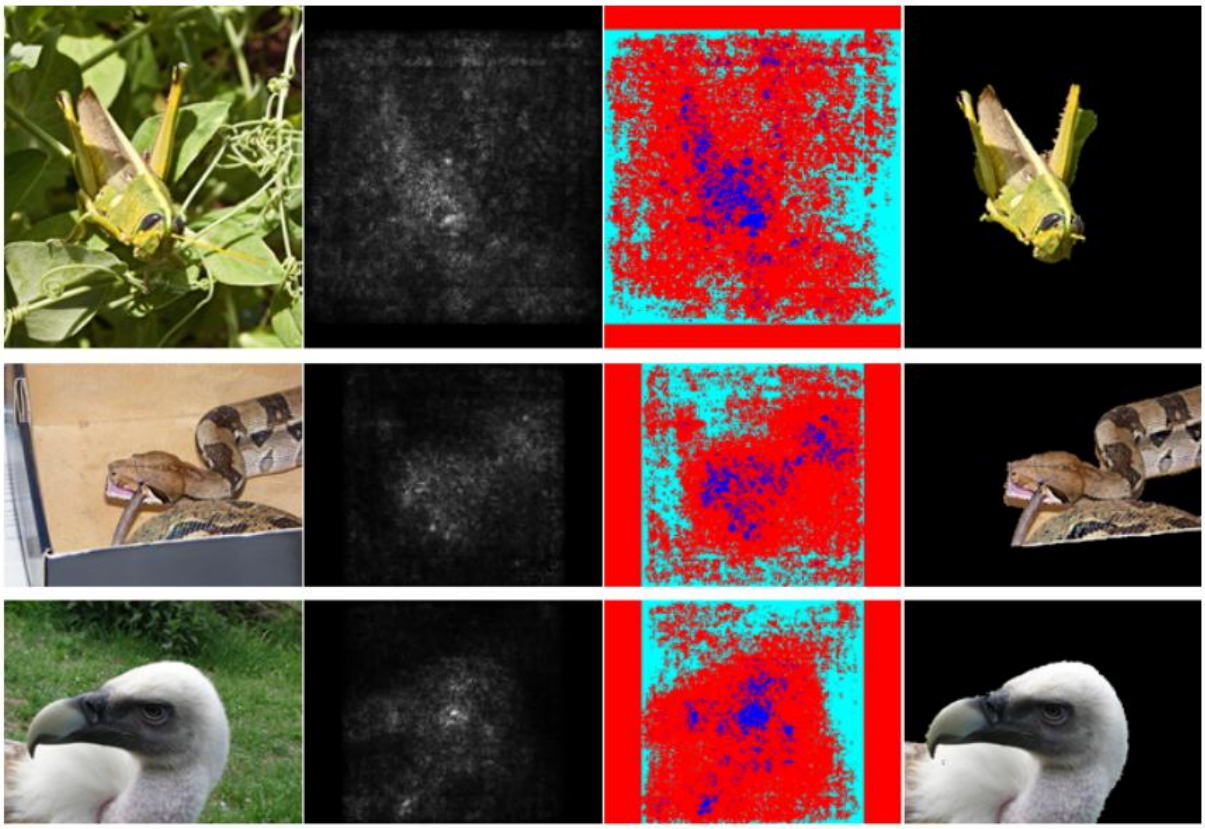

Figure 2.17: Saliency mappings of a neural network trying to identify animals. First a saliency map is created (black and white), then a threshold saliency map is created, where a pre-defined algorithm separates the foreground (shown in blue) from the background (red and cyan), and then a cropping algorithm is used to separate the animal from the rest of the image [32].

This is originally done with the intention to see which parts of the image is the network doing computations on, but as shown on the image above it can also be used to do automatic cropping of images. Because of this additional property, saliency mapping is often used for semantic segmentation tasks, where the locations of objects have to be detected on images.

2.4 Existing Software Solutions for Visualizasing and Understanding Convolutional Neural Networks

The following section discusses already existing software solutions for visualising and understanding convolutional neural networks. This section focuses on tools which are visualising and educating users about the workings of a trained neural network, as it is in line with the problem formulation in **Chapter 1.2**.

2.4.1 Picasso

Picasso is a specialised, open-source tool for rendering visualisations for analysing convolutional neural networks. It is written in Python and is published under the Eclipse Public License, and lets its users to visualise neural network architectures by specifying a visualisation code and an HTML template, and has support for convoultional networks created only in Keras and Tenserflow [33].

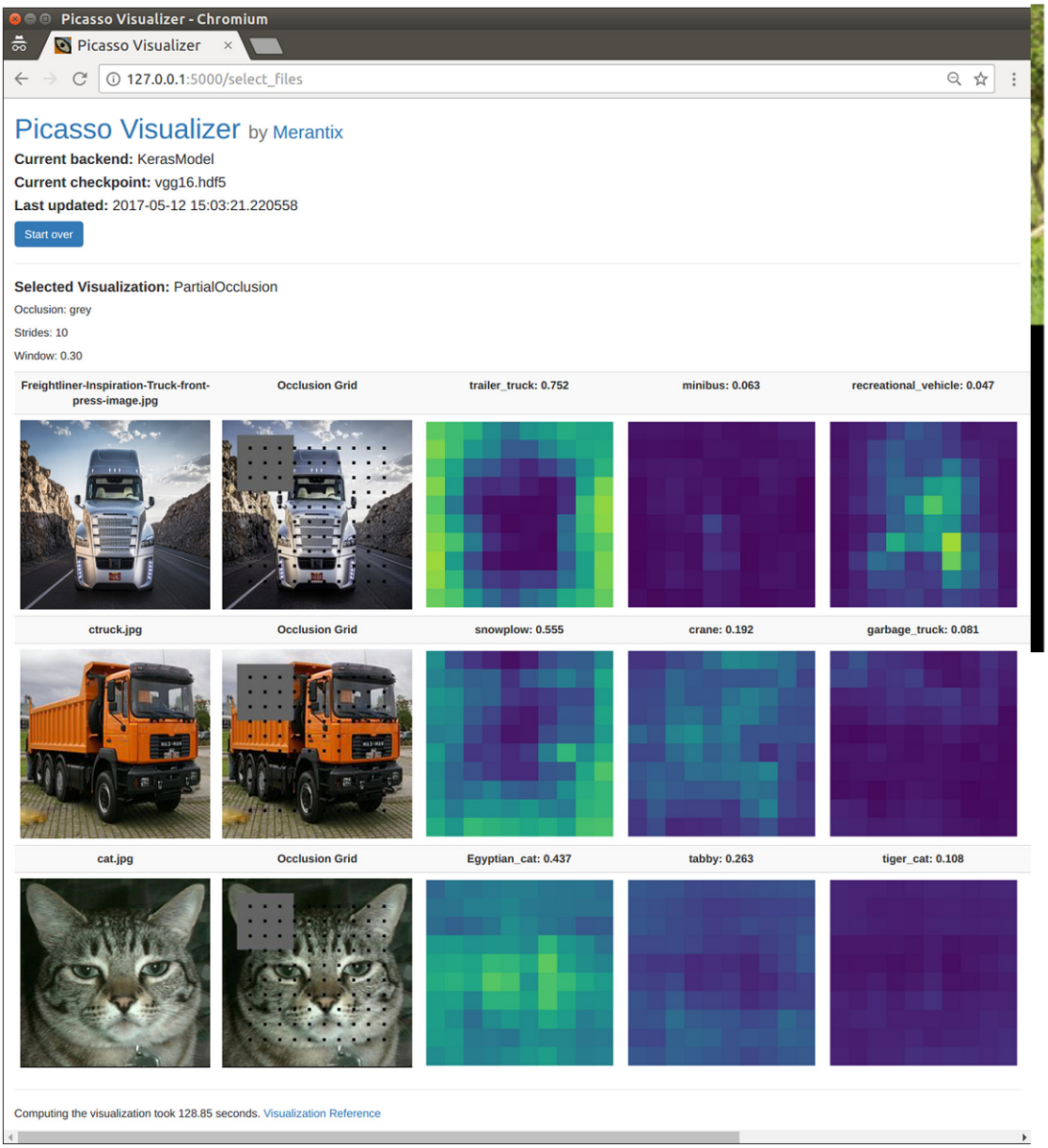

Figure 2.18: A screen capture of a custom Picasso dashboard, set to do partial occlusion over the image. On the capture, three images are shown with their corresponding heatmaps of a neural network's three highest class scores [33].

The visualisations in Picasso are possible by the combination of two things; implementing the partial occlusion technique from **Chapter 2.3.1** and saliency mapping discussed in **Chapter 2.3.2**. This sounds like a feature rich solution, but in practice to use Picasso one needs fairly extensive knowledge in Python and Javascript, and its documentation is quite incomplete at parts. For example, the documentation in its tutorial[1] simply says "For more complex visualizations, see the examples in the visualizations module" which then just points at a hundreds of lines long Python source code on Github.

At the time of writing of this thesis, Picasso seems like the most commonly used third party visualisation tool for convolutional neural networks, without any real contender.

2.4.2 Visualisation built into Keras

Keras is a high-level, open source neural network framework, written in Python. It is the most well known and widely used tool for training neural networks [34], and it has some built in tools for visualising the neural networks it generates. It allows for a summarized view of each layers roles and their properties, but it does not provide any further insight to the workings of the network. One such visualisation is shown on **Figure 2.19** for a convolutional network trained with the LeNet architecture.

Use this somewhere in Analysis, mention how this is not enough or something

[1] https://picasso.readthedocs.io/en/latest/visualizations.htmlconfigure-the-html-template

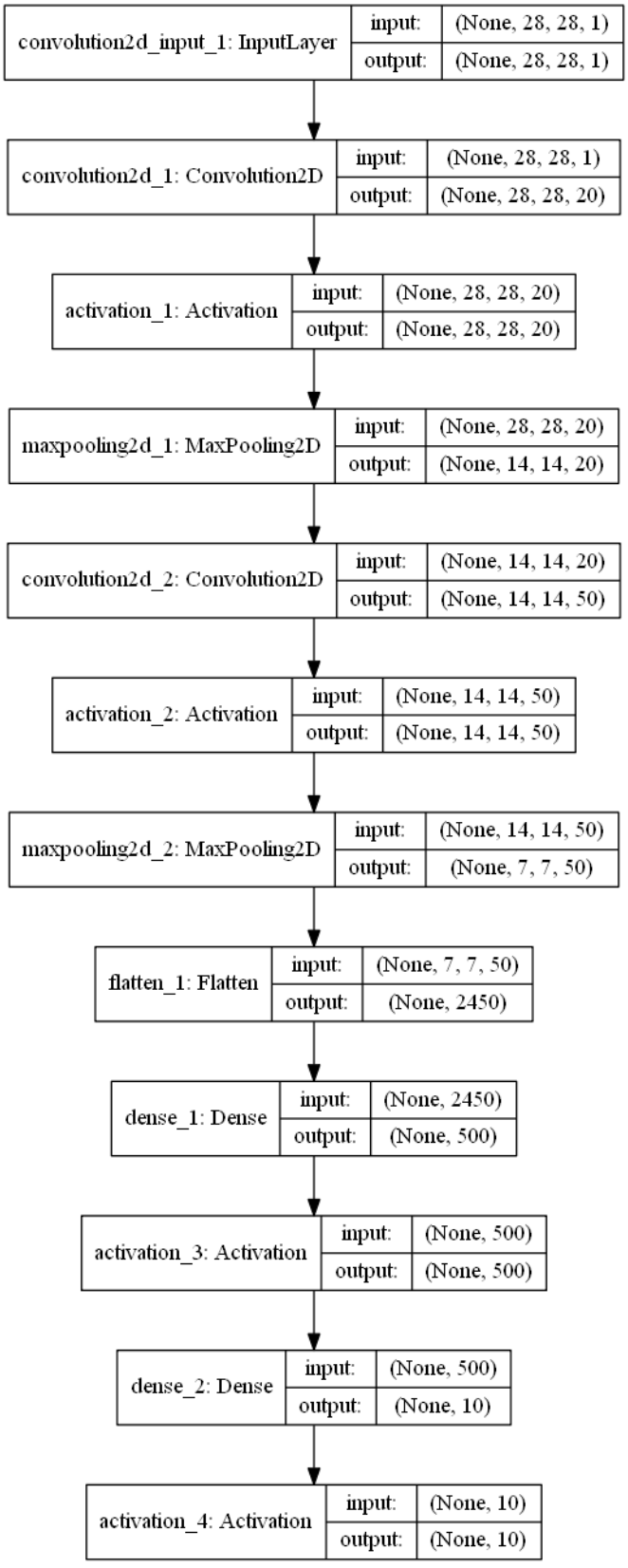

Figure 2.19: The model summary as provided in Keras with the keras.utils.vis_utils function. This image contains information about each layer's role, their output shape and their number of parameters [35].

2.5 Visualising the filters learned by a Convolutional Network

Another approach for visualising convolutional neural networks is to show each of the filters learned by the network. This returns somewhat meaningful information on the early layers when the network is picking up edges and other general features, as shown on the following figure with a network trained on Stanford University's online demo tool at [36].

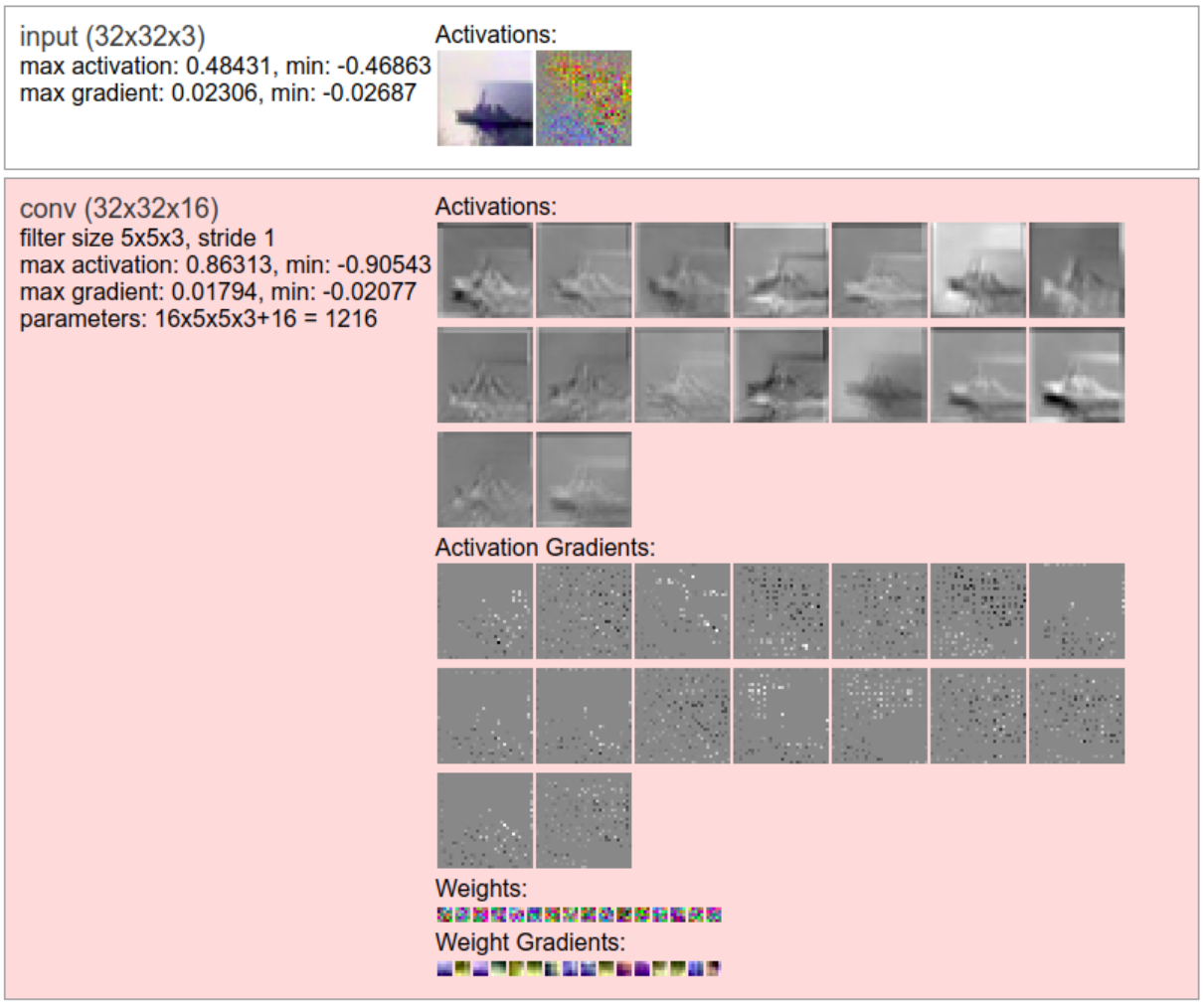

Figure 2.20: An example of the filters learned by the first convolutional layer of a neural network, trained on the CIFAR-10 dataset and ran through the visualisational demo of Stanford University at [36].

This works fairly well on the first convolutional layer, but the output gets less and less meaningful as it progresses to the deeper layers of the network, as shown on the following figure.

Figure 2.21: An example of the filters learned by the third convolutional layer of a neural network, trained on the CIFAR-10 dataset and ran through the visualisational demo of Stanford University at [36]. The input image is the same ship as on Figure 2.20

In the case of this visaulisation the input is less straightforward to interpret, as the image is scaled down because of the convolutional layers in the network, and the weights learned by the filters are not colorcodeable in a meaningful way. This is because the image being processed is not consisting of three layers anymore (corresponding to the RGB color channels) but has the inherited dimensionality of the filters instead summed together.

2.5.1 Neural Network Playground by Tenserflow

Neural Network Playground by Tenserflow is an open source educatory tool for getting insight to neural networks, a screen capture of which is shown on **Figure 2.22**. It allows users to "play around" by creating nodes in an ANN by defining the number of hidden layers and the number of nodes for each layer. It also allows the user to change the input features and some properties

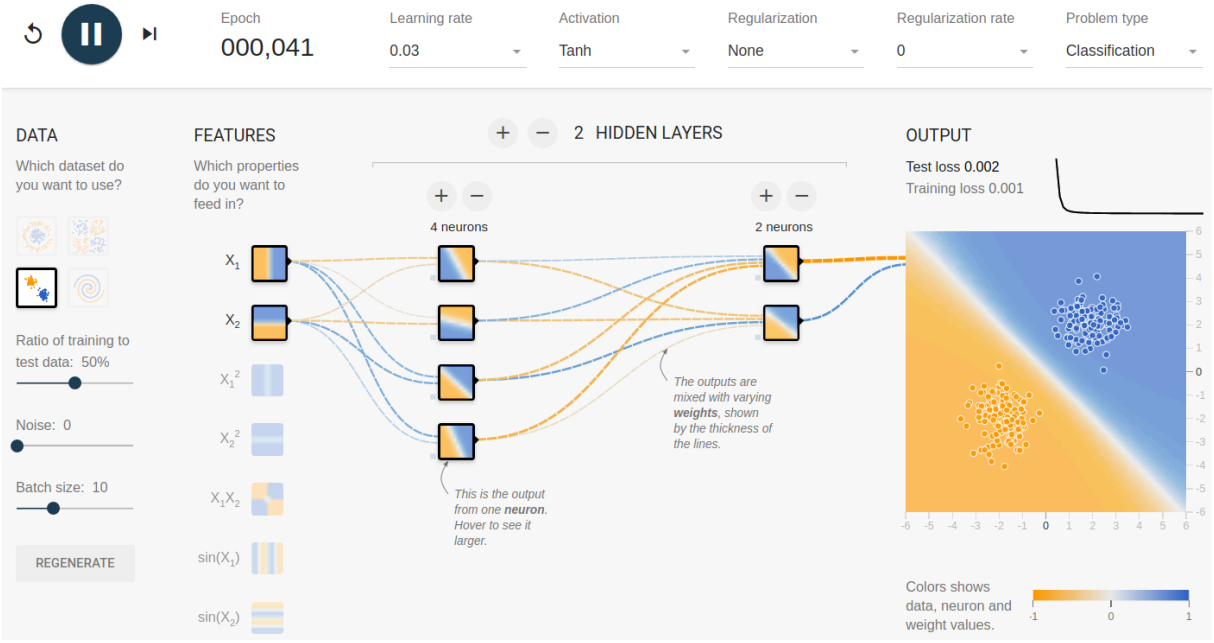

Figure 2.22: A neural network being trained on the Neural Network Playground

of the dataset, such as ratio of dataset used for k-fold validation, the noise ratio and the batch size. It also allows for setting the following hyperparameters when training the neural network:

- Learning rate
- Activation function (with options for Relu, Tanh, Sigmoid and Linear)
- Regulasiton type

After having chosen the desired settings, the user can click a run button to begin training the network. While the network is being trained, a counter shows real time how many epochs have passed, while the weights and biases being learned by the network are shown live in the browser, with a visual output of the resulting decision boundaries.

The basis for the implementation of this application is a library called "Deep playground" published by Tensorflow under the Apache 2.0 license, and is made available on Github[2].

[2]https://github.com/tensorflow/playground

2.6 Possible Problems causing Neural Networks to function sub-optimally

One of the problems this thesis is aiming to find a solution for is to provide insight into why a neural net is not performing optimally, as written in the problem formulation in **Chapter 1.2**. This could be due to a number of reasons, with some of the most common ones being as follows:

2.6.1 Too high learning rate

Neural networks are usually trained with the use of gradient descent algorithms, which has many implementations, as discussed in **Chapter 2.1.4**, but most of them has one thing in common; they allow for a learning rate and a batch size to be specified. When training the network, this parameter will decide how far the weights will be moved in the direction of the determined gradient for each mini batch.

If the learning rate is set too high then the network can overshoot local minimums and if it is set too high, then the training will not even converge [37]. If this is not the case however, the network will finish training with inefficient weights, which will cause the network to perform suboptimally [37].

2.6.2 Bad labels and wrong data

Another reason for a neural network to behave unexpectedly and provide incorrect results is if the training data is wrong or if it has incorrect labels. In this case the network will still learn something based on the training data, but when presented with real data its output will be seemingly random, as it has been shown in [38] where the researchers shuffled the labels of the dataset and the network still managed to correctly identify the "classes", proving that no matter what dataset the network is presented with, it will always learn its features, even if it means that the it is effectively memorising the dataset.

2.6.3 Too many neurons

A common issue with neural networks is overfitting, which often happens because too many neurons have been added to the neural net. A symptom of this issue is when the network is being trained it performs very well o the training set, but when it is measured against new data the performance drops, such as on the following example when a network is supposed to learn to approximate the underlying sine wave from noisy measurements [39].

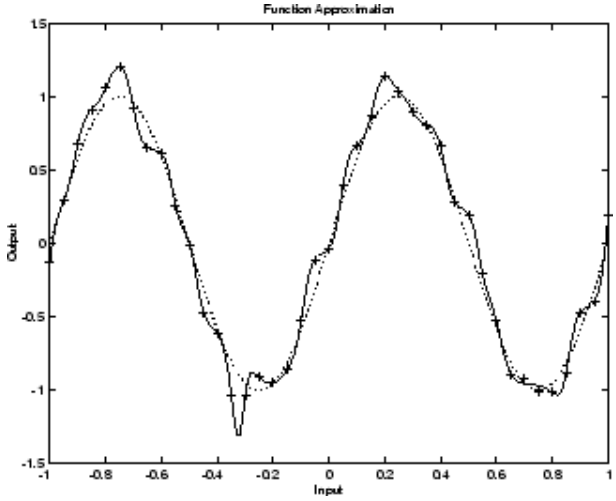

Figure 2.23: A 1-20-1 neural network attempting to approximate the underlying sine wave. The dotted line shows the underlying sine function, the noisy measurements are marked by + symbols an the response of the neural network is marked as the solid line [39].

In this case, the network would perform very well on the training set, but loses accuracy with new data because it is most likely trying to find a complicated pattern for a single case. This is because the larger the network is, the more complex functions it can create, so the most desirable outcome is when the network is small enough to solve problem, but is not big enough yet to have enough power to overfit [39].

2.6.4 Not enough training data

Not enough training data will cause the network to overfit, which will lead to problems when it comes to generalisation, as discussed in **Section 2.1.6**.

3| Analysis

This chapter aims to put into perspective the points discussed in the previous State of the Art chapter in relation to the problem formulation of this thesis found at **Chapter 1.2** and analyse how they relate to the creation of a visulasisational tool for convolutional neural networks. This is done by separately discussing each design principle and then creating requirements from these findings. These requirements are then to be prioritised with the means of a MoSCoW analysis for the final product of this project.

3.1 Categories of Visualisation Techniques for Neural Networks

In the previous State of the Art chapter, the solutions for visualising neural networks have been categorised as theoretical approaches mainly discussed in research papers, but without a proper implementation or being embedded in off the shelf software, or as part of a software solution readily available to use for people working with neural nets. For the purposes of this chapter this thesis will organise possible ways for visualisation techniques from two main perspectives; either from the data's point of view or from the learned neural network itself.

3.1.1 Visualising the network from the dataset's point

As the dataset is the origin point for the creation of the neural net, the person training the network has most likely already familiarised themselves with it, and because of this it is a good starting point for providing insight into the learned behaviour of the neural network itself. This can be done by showing the activations and ways of behaviour for each individual object in the dataset, which also allows for a more visual approach by plotting this to the chosen image itself.

This data centric approach helps to see how the network performs on each class type on a case-by-case basis. A straightforward way of showing the dataset is through a browser menu, which allows for selecting an image which will be passed through the neural network, and then the activations and the behaviour of the network can be shown for this specific case.

3.1.1.1 Partial Occlusion

An approach for providing insight into the workings of the neural network is by treating the network as a black box, and only looking at the input image in relation to the output of the network. A way of doing this is through sliding a box which is colored as the mean color

value of the image gradually over the picture and calculating how significant each pixel is for the classification of the image, as discussed in **Section 2.3.1**. This can be used to generate a heatmap, which will show which parts of the chosen image is the neural network "looking at" when classifying it as whatever it is classifying as, such as the one shown on **Figure 2.16**.

This information from the heatmap can be especially useful in two cases. When the network is incorrectly identifying the image as something else, then the knowledge of which parts of the image are causing this can be a good start for an investigation. For example, if the network would incorrectly be identifying a picture of a standing dog as a cat, and it turns out that the network is only looking at the dog's legs, it could be that the training dataset does not have many pictures of standing dogs but has pictures of standing cats, so it has learned that the presence of four straight legs means that it is a picture of a cat.

Another case when such a heatmap can be useful is when the network is identifying a picture correctly, but for the wrong reasons. For example the case also mentioned in the introduction of this thesis, when the Pentagon ordered a nerual network for recognising if there was a camouflaged tank on pictures or not, performed well on the dataset by recognising the pictures of tanks, but as they later found out, all the pictures of tanks were taken in bad weather and all the other pictures were taken in good weather [2].

Figure 3.1: Two images used for training the neural network for the Pentagon in the 1980s, on the left an image without a tank in good weather, on the right an image with a tank with cloudy weather [2]

In this case the network could correctly classify every single image, but for the wrong reasons. Progressive occlusion could have been used, and then the researchers would have realised that there would have been no significant activations for parts of the pictures with the tanks on them, but the blue or grey sky would have been highlighted on the heatmaps.

3.1.1.2 Saliency Mapping

Another approach for getting a visual feedback of which pixels of the input matter in the classification of the image by a convolutional neural network is through saliency mapping, as discussed in the State of the Art in **Section 2.3.2**. The difference between saliency mapping and progressive occlusion is that it does not treat the neural network as a black box, and instead looks at which pixels of the image are used by the network for the classification, and because of this it could highlight different issues than progressive occlusion. For instance, it can detect if a given pixel is dramatically influencing the classification [33], and because of this it could be possible to use it to see if the network is overly sensitive for an outline or a very limited part of the image, which could be a sign of an overfitted network. As an example, two saliency maps of two images belonging to a convolutional neural network trained to classify animals is shown on the following **Figure 3.2**;

Figure 3.2: The saliency maps of two neural networks trained to classify animals [32]. The fairly even activations on both of the pictures means that the network is using most of the pixels in the image to recognise the animals, but in case the picture on the left it also uses the surrounding scenery in the classification process.

3.1.1.3 Other Benefits of Data Centric Analysis of a Neural Network

Providing insight from a data centric point of view has other benefits as well, such as the tool proposed by this thesis could be used as a way to quickly scan through the pictures used to train or validate the network on. It would also allow for grouping the pictures by their classes, and together with an uploaded excecutable neural network, these pictures could be sorted by how accurately the network recognises them. This could be used to find outliers, tricky images or even incorrectly labelled data.

For such a solution, a browser would have to be implemented as part of the visualisational tool, with support for the loading and handling of large amounts of data, as neural networks are often trained on thousands or millions of images, such as the commonly used imagenet dataset having (at the time of writing of this thesis) about 14 million images for training [40].

3.1.2 Visualising the network from the neural network's point

Another approach for providing insight is to look at the trained convolutional neural network in general without a specific training example, and use techniques such as gradient ascent to show the behaviour of the network for a specific class.

This approach is called class model visualisation, and the goal of it is to generate an image with the highest possible for a given class. For example, the following images were generated to get the highest possible scores for a neural network trained to recognise classes from the imagenet dataset.

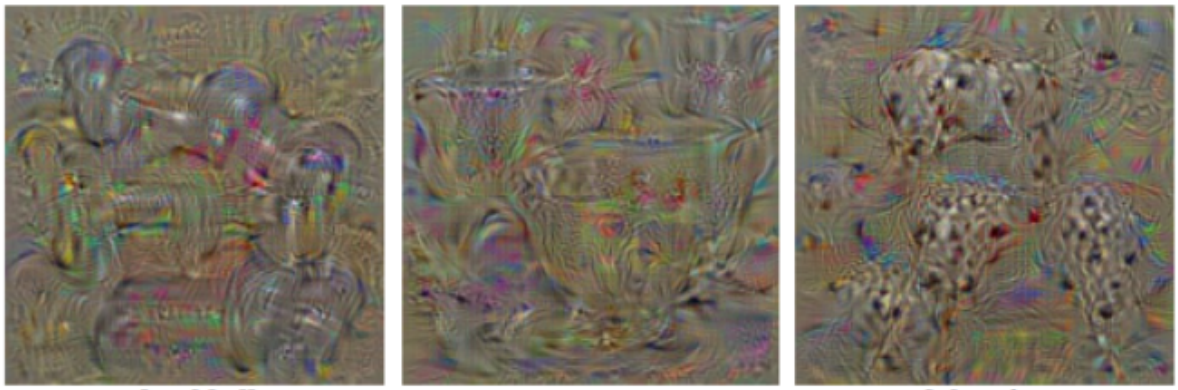

Figure 3.3: Class model visualisations for the dumbbel, kettle and dalmatian classes of the imagenet dataset [32]

This approach is less useful for the purposes of this thesis, as using the network to generate outputs for a specific class is a good way to gain an understanding of what the network does, but it is not necessarily useful in terms of getting insight on the performance of the neural network. In the case of the example above, the network may be giving a specifically high score to cups with round bottoms and handles, but this does not mean it would not recognise a straight sided handleless mug as a cup. Because of this, class model visualisations may be interesting, but are not necessarily useful for understanding why a network performs as it does.

3.1.2.1 Visualising Activations and Weights of the network

Another way to get insight into the working of the network while taking the trained network into account is to show each of the activations and weights learned by the network, as discussed in **Section 2.5**. This approach is useful for the early layers where the dimensionality mathces the RGB input space of the original image, such as shown on **Figure 2.20**, but is less intuitive as it gets to the deeper layers, as it is shown on **Figure 2.21**. This is because the activation maps produced at the hidden layers always have as many "channels" as many filters have been applied to the previous activation from layer one before.

These activations can be visualised as as many grayscale images as many channels they have, and that is what is happening on **Figure 2.20**. Seeing this however most likely does not give insight into the workings of a neural network that would help someone in improving its performance or to understand why its working differently, as these projections of higher dimensional activations will just seem like random noise. Maybe with some advanced analytical function some additional information could be extracted of them, but this is out of scope for this thesis.

The earlier filters however could be useful in theory, as they can be shown as proper colored images, but as they are early in the network they only do very general tasks, as it has been shown that these layers can even be transformed between different networks, which have been trained to do completely different tasks [29]. Because of this, showing these learned attributes does not lead to any significant insight either.

3.2 Representing the Layers and Nodes of the Network

One of the biggest issues with neural networks is that they cannot be comprehended in an easy way when they are shown in a plain matrix form. Because of this, it is imperative that a visualization tool starts out by plotting the neural net to some graphical abstraction. As shown in the State of the Art chapter, there are multiple ways of doing this, such as creating a layer-by-layer representation and displaying all nodes seperately like Tenserflow's neural network playground or as a summary for each layer, as shown on **Figure 2.19**.

However such a distant visaulisation by itself might not be as useful, as the user might already know the details of their chosen convnet architecture, which are very well documented in their original papers, as discussed in **Chapter 2.2.0.1**.

To provide a more dynamic view and in line with the problem formulation written in **Chapter 1.2**, on top of showing the neural network a closer view could be shown to the user by letting them to choose one of the layers of the architecture, which could expland and show how the network has been trained in this specific case. This way, all nodes can have their weights and

biases displayed in a meaningful way. An example of such a fanned out neural network is show on the following **Figure 3.4**;

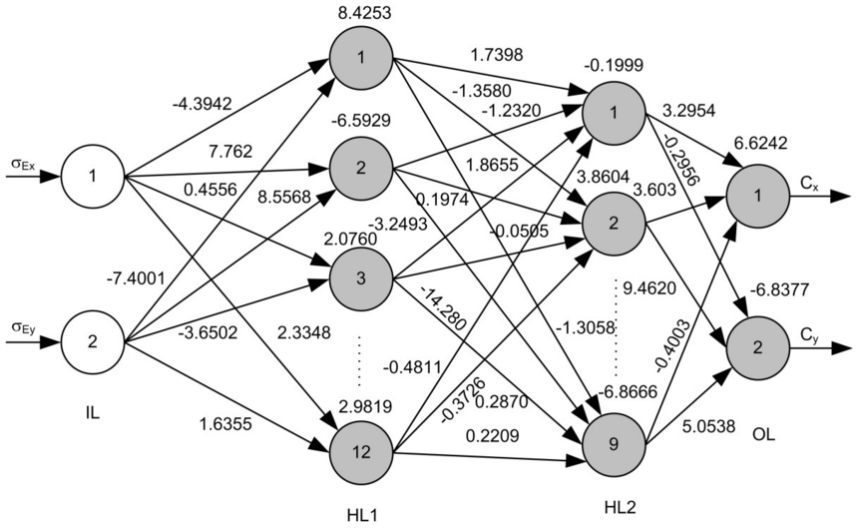

Figure 3.4: A neural network with all of its weights and biases displayed [41]

3.3 Visualising Node and Layer Parameters of the Network

A further way of providing insight into the workings of a convolutional neural network is by taking each convolutional layer individually, and presenting the filters learned by them to get a sense of what features the network has learned to look for. An approach to this is discussed in **Chapter 2.5**, where it is concluded that this is not feasible for answering the problem formulation of this thesis. However, using these layers in addition to a kind of visualisation as the one shown on **Figure 3.4**, with the filters laid over the nodes a similar visualisation could be created as the one used by the Neural Network Playground discussed in **Chapter 2.5.1**. The filters would still not help to educate the user about how to increase the efficiency of the network, but it could be argued that such an interface with the filters being shown for each node would result in a more positive user experience.

Another possible solution is to emphasize significant weights and nodes with higher biases, as these are more significant for the possible outputs of the neural network. These "more important" nodes then could be color coded in a meaningful way, and shown on a map such as the one presented on **Figure 3.4**.

3.4 Performance Analysis of Neural Networks

A way to provide performance analysis is to see how the efficiency of the neural network has changed over time during training. From this data it can be learned if something has gone wrong during the training period, which could cause the neural network to perform suboptimally. For example it can be learned if the network has started to overfit because it has been running on the same limited dataset for too long, along with other information that can be inferred, as discussed in **Chapter 2.1.6**.

3.5 Giving Insight into Possible Problems causing Neural Networks to function suboptimally

As discussed in **Chapter 2.6**, there are some typical issues which could happen when training neural networks. As discussed in the problem formulation in **Chapter 1.2**, one of the intentions of this thesis is designing a tool which makes neural networks more accessible for beginners. Because of this, a feature which detects common issues and shortly discusses possible solutions to them could be highly useful. This section is to discuss how can insight be given into these problems, and if there is a straightforward way to educate users about them.

3.5.1 Dead Neurons

As discussed in **Chapter 2.2.1** in the State of the Art, some activation functions have a tendency of killing or saturating neurons, which makes that part of the network not change throughout the backpropogation, seriously hindering performance. With an attached training history, the user of a visualisation tool can be informed with a statistic of how many of the neurons are not performing.

This feature has been requested for the most commonly used tool used for training neural networks called Keras, discussed in **Chapter 2.4.2**, but since has only been partially implemented and with poor documentation [42]. Other than this, at the time of writing of this thesis no other out of the box dead neuron detector seems to exist.

3.5.2 Activation Function Detection

Another information that can be obtained from neurons is their activation function, as discussed in **Chapter 2.1.1**. This can be important in terms of performance, as some activation functions can achieve better results on certain types of tasks [4]. This could be communicated to the user by showing a prompt telling the user about the pros and cons of other type of activation functions

if it is detected that a more efficient one could be used in this case.

3.5.3 Analysis of Training History

Other than looking at the neural network and its current performance, insight can also be obtained from looking at older versions of the network during training and how the neurons have changed, as discussed in **Chapter 2.1.6**. This information can be used to see if the network has begun overfitting at some point, or if it did not converge properly on a local minima, or if has been causing it to perform suboptimally because of other reasons originating from the training process.

3.6 Visiualisational Tool Scenarios

To better understand what requirements the visualisational tool has to fulfill, this section is to introduce multiple scenarios, describing users' interactions with the system, exploring possible cases in order to help to answer the problem formulation written in **Chapter 1.2**. These scenarios are meant to describe human activities in the form of a short story to allow for the exploration of needs and requirements to be filled by the tool discussed in this thesis. A the end of a brainstorming process, the following scenarios have been identified;

- **Scenario 1** - A researcher wants to make sure his well performing neural network is behaving as expected
- **Scenario 2** - An engineering student wants to understand why is his network performing so poorly
- **Scenario 3** - A professional developer wants to see why his network always identifies only one of the classes as another one
- **Scenario 4** - A researcher has trained a neural network which performs fairly well, but she wants to see if her network could do better than this
- **Scenario 5** - A student is learning about convolutional networks, and wants to see how does a state of the art network work

3.6.1 Scenario 1 - A researcher wants to make sure his well performing neural network is behaving as expected

A researcher has trained a convolutional neural network for recognising tumors based on the CT scan of patients, which has been performing with a higher than 99% accuracy on the validation

dataset. However before he would publish his findings he would like to get an insight into what is the network basing its classification decisions on, and consult with some experts in the field. He downloads a tool from his research institution's website, which he then installs on his private computer. Then he follows the instructions on his institution's website, and accesses the tools' interface in his web browser, which is prompting him to select a path for the trained network and its corresponding data. After having loaded the network, the researches sees some statistics about it showing on the screen, which matches with what he was previously seen. Then he selects a number of images belonging to positive cancer patients, and the tool highlights the areas of the image where the network was identified signs of a tumor. The researcher then prints a number of these images with highlighting to bring to a consultation with an expert later on.

3.6.2 Scenario 2 - An engineering student wants to understand why is his network performing so poorly

An engineering student has trained a convolutional neural network, which has performed very well on the training dataset, but only has slightly better results on the validation dataset than random. He downloads the visualisational tool he has used with his class, and opens its interface from his web browser. He selects a path for the trained network and its corresponding data, and he also uploads the training history to the tool. The tool takes some time to process this data, and then after displaying its regular interface, the student notices that a part of it is highlighted in red. The visualisational tool is pointing out how the training loss has started to climb over time, and is showing a small prompt explaining that this is usually a sign of overfitting. This reminds the student how letting the network train too long will make the network less able to generalise, which explains its poor performance on the validation dataset. With this information he begins training his new network in a separate software.

3.6.3 Scenario 3 - A professional developer wants to see why his network always identifies only one of the classes as another one

A developer working for a big international firm developed a convolutional neural network for classifying pictures of postal stamps, but the network always classifies stamps from China as Danish ones. He downloads a tool from a software database, installs it, and accesses its interface from his web browser, as written in the software's guide. He then gives a path for the network and its dataset to the software, which then shows him a default menu. In the menu he chooses to only see data points from the Chinese stamps, after which the program loads all 120 pictures corresponding to the category, and it shows that most data points are classified as Danish, with only a bit smaller scores for the Chinese classification. Then the developer chooses to see every data point corresponding to the Danish stamps, after which the software loads all 3400

pictures corresponding to that category, most of them correctly being identified as stamps from Denmark. The developer realises that the network might have this issue because there are not enough training examples of Chinese stamps, and he begins to look for more to train a better working version of the neural network.

3.6.4 Scenario 4 - A researcher has trained a neural network which performs fairly well, but she wants to see if her network could do better than this

A researcher has trained a convolutional neural network for identifying breeds of dogs, and it is performing at about a 90% accuracy, but she has read that others managed to get better performance on the same dataset. She downloads a tool from the website of her institution, installs it on her laptop, and accesses its interface from her web browser, as written in the software's instructions. She then gives a path for the network and its dataset when the software prompts her to do so, and the network shows her samples of her dataset in relation of the architecture of the neural network. The software shows with a yellow marking that the selected neural network has been trained with the GoogLeNet architecture, and the researcher clicks on the prompt for more details. On the following page, the program shows a chart with comparisons between convolutional neural networks for classification problems, and the researcher notices that the GoogLeNet architecture that she used is outperformed by some other architectures on this kind of classification tasks. Armed with this new information, she then attempts to train a new network in a separate software.

3.6.5 Scenario 5 - A student is learning about convolutional networks, and wants to see how does a state of the art network work

A university student has learned about convolutional neural networks using the AlexNet architecture in one of her courses, and she would like to better understand what goes on "under the hood" in an example, state of the art network that she downloaded from the Internet. She installs a tool from her institution's website, launches it and opens it from her laptop's browser. She then chooses the path for the network and for the dataset used by the network. On her screen she sees a part of the dataset, how well the network is performing on it and the layout of the neural network, which the software correctly identified as AlexNet. Then she chooses a part of the network she would like to more closely examine, and the software zooms in on it, with the weights and activations color coded and highlighted based on the currently selected datapoint. Then she chooses some other images from the dataset and the coloring and highlighting of the visualisation changes accordingly. She closes the program, satisfied after having seen the inner workings of the neural network.

3.7 Requirements

To further limit the scope of the project for the upcoming design chapter, some requirements have been set. The aim of these requirements is to specify what the proposed solution is to do, and to set constraints on the resources and the design of the solution. These requirements are presented grouped into functional and non-functional requirements, and then they will be prioritised with a MoSCoW analysis described by Sommerville in [43]. This is done by scoring each requirement with either a score of must, should, could or won't, depending on how important they have been found for the creation of a prototype of the solution discussed in this thesis.

Each requirement is numbered, which is used for reference in later chapters. For all of the requirements, a short function is summarised, followed by a more detailed description of the function, then a rationale is given to justify the requirement and then the MoSCoW priority of the given requirement is shown.

3.7.1 Functional requirements

This section lists the functional requirements set for the system. These requirements are to describe what the system should do and behave and what kind of services should it provide.

Table 3.1: Functional Requirements

Requir.	Function	Description	Racionale	Priority
FR_1	Specify the Network's location	Select where the network is located on the computer	The network is needed for a network centric analysis discussed in 3.1.2	Must
FR_2	Specify the location of the Dataset	Select where the dataset is located on the computer	The dataset is needed for a data centric approach discussed in section 3.1.1	Must
FR_3	Specify the Training History's location	Select where the training history is located on the computer	The training history is required for analysing it, as discussed in section 3.5.3	Must

FR_4	Execute the neural network	Run the network on one or more instances of the training data and show its output	This is needed for seeing how the network classifies each data point as discussed in 3.1.1	Must
FR_5	Browser for dataset	A viewer which displays the dataset in a grid style view	Needed for a dataset centric approach, as discussed in 3.1.1	Must
FR_6	Filtering for the dataset	A dropdown menu, filtering the datased based on each image's real class	As the dataset is used due to FR_2 , which can have many thousand datapoints, for which filtering is essential to make it manageable	Must
FR_7	Sorting the dataset	Sort the dataset based on classification accuracy	It is important to show on which data points is the network performing best and worst to understand its behaviour, as discussed in 3.1.1	Must
FR_8	Identification of typical problems	Identify typical problems with neural networks and show a short description of it	Informing users if their network is victiom of a typical problem is important, as discussed in 3.5	Must
FR_9	Installer	Installs the visualisational tool	Needed for the solution to be more accessible, as also discussed in the scenarios, found at 3.6	Must

FR_10	The system should recognise the convnet's architecture	Identify which type of convnet architecture has been used (if any) and display statistics about it	Convnet architectures are very well studied and documented, as discussed in 3.1.2.1, which can be used to display additional information about it. Also discussed in scenario 4 found at 3.6.4	Could
FR_11	Comparison between other convnet architectures	Display comperative information between the network's convnet architecture and other convnets	The choice of convnet architecture significantly influances the network's performance, as discussed in 3.6.5	Could
FR_12	Detect activation function	Detect and display information about the activation function(s) used by the network	The activation function is one of the most determining characteristic of a neural network, as discussed in 3.5.2	Should
FR_13	Show comparison between activation functions	Show how the activation function used by the network compares with other ones	The choice of activation function can significantly affect performance, so feedback on is is very important as discussed in 3.5.2	Could

FR_14	Color code weights and and activations	The weights and activations should be color coded for better insight	Color coding the inner workings of the network can help insight, as explored in scenario 3.6.5	Should
FR_15	Show dead neurons	Show neurons which have died and were not contributing when trainign the network	Dead neurons can hinder performance, as discussed in 3.5.1	Won't
FR_16	Heatmaps to show activations	Heatmaps to show which part of the image is being processed by the network for each class	Heatmaps give useful insight into which part of the image is the network looking at, as discussed in 3.1.1.2	Must

3.7.2 Non-functional requirements

This section is to list the non-functional requirements of the solution proposed in this thesis. They are to describe how well the solution is to perform its functions and to set specific constraints on it.

Table 3.2: Non-functional Requirements

Requir.	**Function**	**Description**	**Racionale**	**Priority**
NF_1	Run on an average modern laptop	Be able to run on a modern laptop with 4 Gb of RAM and with a processor with integrated graphics	The platform should be accessible to as many people as possible, so it should not have too high system requirements	Should

NF_2	Platform independency	The system should be platform independent	The platform should be platform independent to be as widely accessible as possible	Should
NF_3	Show in web browser	The interface of the system platform be accessible from a web browser	The state of the art solutions discussed in chapter 2 ara all displayed in the browser, and this part of them works just fine. This is also explored in the scenarios at 3.6	Must
NF_4	Run from current version of Chrome	The platform should support Chrome version 68.0.3440	The platform should support the latest version of Chrome at the time of writing this project, as this is a free browser accessible on all major platforms, which is needed due to NF_2 and NF_3	Must
NF_5	Work offline	The platform should work offline	The platform should work offline, further incrasing accesibility	Must

NF_6	Colorblind mode	Support for colorblind users	If the platform is using colors for its functions which are not accessible for colorblind people, a feature should be implemented to fix this	Won't

This chapter analysed the solutions from the State of the Art chapter and based on what has been gathered, in conjunction with the problem formulation, five scenarios have been formulated. This analysis together with the scenarios have been used to formulate the funcional and non-functional requirements shown in **Table 3.1** and **Table 3.2**, which are to be used for the following Design chapter.

4| Design

This chapter is to discuss the design decisions made based on the State of the Art and Analysis chapters, with the focus on fulfilling the requirements set in **Table 3.1** and **Table 3.2**. For this thesis a prototype is to be created, which is to implement the "Must" requirements of the aforementioned tables. The requirements from these tables will be referred to by their identifiers found in the first column of these tables respectively.

4.1 Menu Layout

Before the development of the solution could have begun, a flowchart has been created to make it easier to visualise how the functions and flows of the application interconnect. The following figure shows this flowchart, with each of the points on it corresponding to a menu of the application.

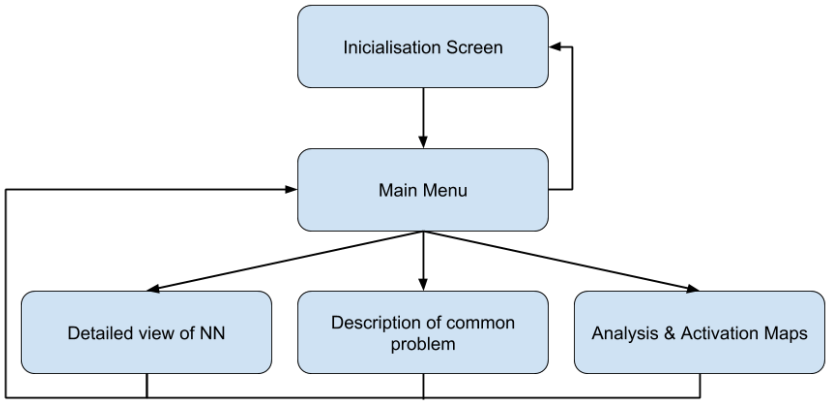

Figure 4.1: Flowchart, showing how the menus of the application connect to each other

Each of these menus are to hold the following features, fulfilling their respective requirements:

- **Initialisation Screen** - lets the user choose the location of the dataset, the location of the trained neural network and (optionally) the training history's location in order to fulfill requirements **FR_1**, **FR_2** and **FR_3**. This is the first screen that is shown when the application is launched, and when a "next" button is pressed after all the necessary information has been given, the main menu is shown.
- **Main Menu** - houses as a main hub for the application, where a browser for the dataset

and a quick view for the neural network's architecture can be found, fulfilling requirements **FR_5** and **FR_10**. The user can go back to the initialisation screen with a back button.

- **Detailed view of the Neural Network** - Shows a more detailed view of the neural network's architecture, where statistics and additional information is shown about it as well. This section is to fulfill requirements **FR_11**, **FR_13** and **FR_15**. This menu is shown if the user clicks on the picture of the neural network in the main menu, and it is also possible to go back to the main menu from here with a back button.
- **Description of common problem** - In case the system detects the neural network is suffering from a typical issue, a colorful indicator is to be shown in the main menu. If the user clicks on it this window is displayed, as required by **FR_8**.
- **Analysis and Activation Maps** - This window is shown when a user clicks on an image from the dataset, providing analysis on how the network classifies it into one of the possible classes. A visualisation is shown, along with a heatmap for each of the top classes the network recognises the image as. At the bottom of the window the neural network is shown, color coded with its activations for the chosen datapoint. This is done in order to fulfill **FR_14** and **FR_16**.

4.2 Conceptual Designs

Based on the requirements in **Chapter 3.7** and the menu layout discussed previously, the following conceptual designs have been created in preparation to the development of the software solution. The sketches have been created with the free version of the online sketching tool Figma[1]. The sketches contain only simple design elements, as to be functional and to the point. Additional design elements and colors can always be added later on after a working prototype has been created.

4.2.1 Installer

To make the system discussed in this thesis more accessible for non-experts, the installation process should be streamlined as much possible, such as discussed in **FR_9**. Because of this, the prototype of the solution is to be bundled in an installer for the Windows platform, which installs all the required parts for the solution to work. This also includes a detection to see if commonly used software such as Python and Chrome is already installed on the system (because of requirement **NF_4**), and in case they are missing, their corresponding launcher is initiated.

The installation process should be as streamlined as possible to not "reinvent the wheel" and by

[1]Figma is available at `https://www.figma.com/`

this making the procedure straightforward as possible. For this, the installer should be created with a widely used tool such as NSIS, which is the most widespread packager for open source applications [44]. An installer created by NSIS is shown on **Figure 4.2**, which is also the installer for NSIS itself.

Figure 4.2: A typical installation screen for the Windows operating system [44]

4.2.2 Initialisation Screen

This is the welcome screen of the application, where the user can specify where the neural network, the training dataset and the training history is located on their computer, as discussed in **Chapter 4.1**. A sketch of the initialisation screen is shown on **Figure 4.3**.

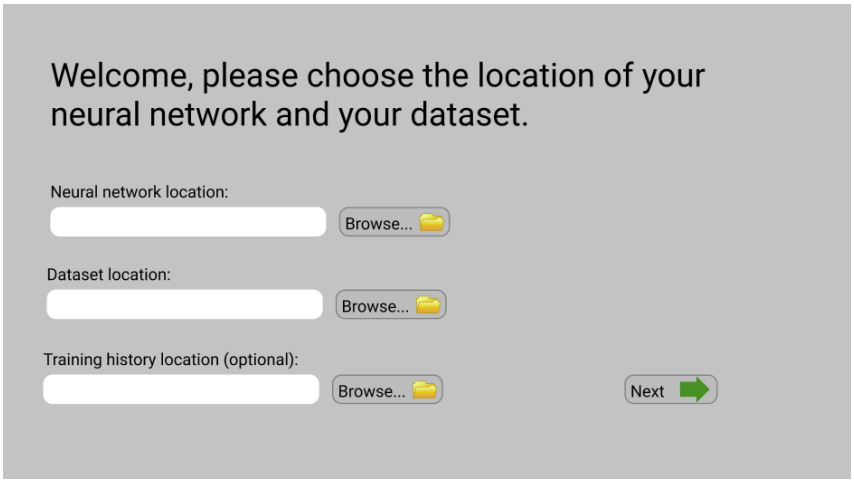

Figure 4.3: A sketch of the initialisation screen of the application

If the user clicks the button for choosing the location for either of the prompts, a native browser of their operating system will be shown to them for choosing the folder containing the needed files.

4.2.3 Main menu

The main menu is available after having given the necessary details on the initialisation screen, as discussed in **Chapter 4.1**. A sketch of the main menu is shown on **Figure 4.4**.

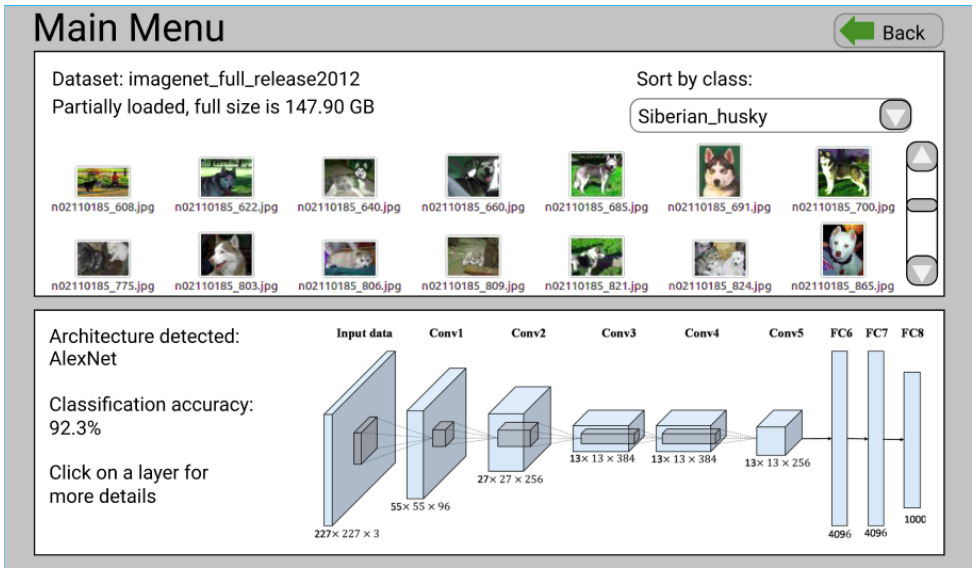

Figure 4.4: A sketch of the main menu of the application. The picture used to represent the architecture of the neural network has been taken from [45].

This screen is divided into two main parts, one for the browser of the dataset and one for a summary of the neural network's architecture. The section with the dataset contains information about the name of the dataset, to show which one has been loaded in, and shows the size of the dataset. On **Figure 4.4** the pictures have been sorted by the Siberian_husky class of the dataset as an example to show the solution fulfills requirement **FR_7**, but when the menu is loaded for the first time is would say "Show all", displaying examples from all classes of the dataset. Then the images of the dataset are displayed, alongside with a scrollbar if they could not all fit on the screen.

On the bottom half of the screen a summary of the loaded neural network is shown. The software displays the detected architecture and its corresponding image representation, and classification accuracy of the network. If the user clicks a layer of the network, a more detailed view of it is shown.

4.2.4 Screen Describing a Common Problem of Neural Networks

In case the system has detected that a common issue could be causing the neural network to not perform optimally, this screen is shown, fulfilling requirement **FR_8**. It describes the issue and also suggests measures in order to prevent it, such as it is shown on **Figure 4.5**

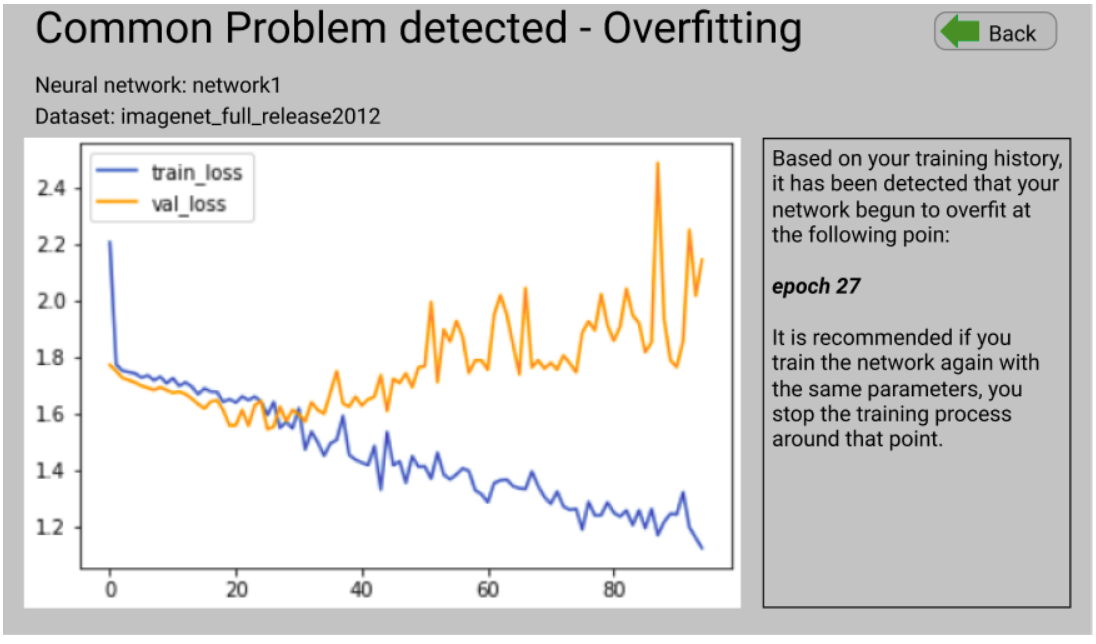

Figure 4.5: A sketch of the menu describing common problems with neural networks in the application. As an example, this sketch shows a case when the neural network started overfitting.

This screen would show the details of the currently detected problem. In the case of **Figure**

4.5 it shows an example of the neural network beginning to overfit on the training dataset as it has been let to train for too long, with a relevant graph generated for this example with Keras shown on the left of the image. It is also explained on the right of the screen in plain language what could be causing the issue with the network, and a possible solution to the issue.

4.2.5 Analysis and Activation Maps Screen

The analysis and activation maps screen is shown when the user clicks on an image from the dataset to get a better insight into how the network decided to classify it as it did, which is to fulfill requirements **FR_4** and **FR_16**. A sketch of the screen is shown on **Figure 4.6**.

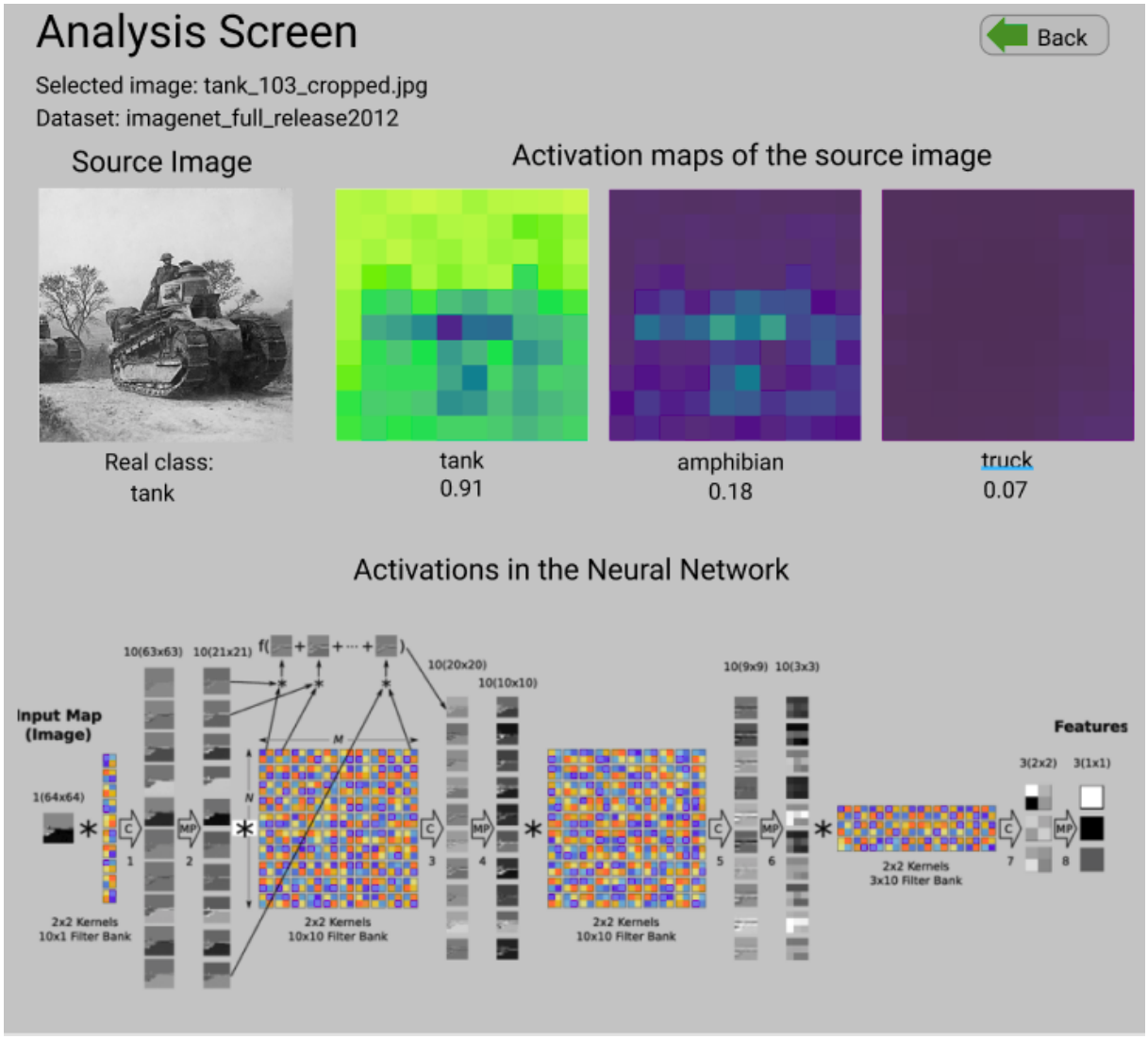

Figure 4.6: A sketch of the analysis screen of the application. The activation map has been taken from [46]

On the sketch of the analysis screen shown on **Figure 4.6** the user has chosen an image of a tank from the dataset, which is shown on the upper left corner of the screen. To the right of the source image, the activation maps of the three classes are shown, which were ranked highest by the neural network. The network correctly identifies this image as a tank, which has the highest activation of 0.91, followed by amphibian and truck with and activation of 0.18 and 0.07 respectively. The heatmap of the image for these classes shows which part of the network has taken into account for the classification. On the bottom of the picture the activation map of the network is shown, with an emphasis of the convolutional layers, which would fulfill requirement **FR_14**.

4.3 Choices for Development

This section is to discuss the choices that have been made for the development process. There are two parts discussed, the background logic with Python scripts and the hosting of the website and the connection to Python with NodeJS.

4.3.1 Python

For implementing the background logic of the prototype, Python has been chosen as the programming language. The reason for this is that most of the open source machine learning libraries are written in Python, along the ones discussed in **Chapter 2.4** and their usage reduces the time and complexity of the development significantly. Another reason for the choice of Python is that it can be included in the installer discussed in **Chapter 4.2.1**, which makes the deployment of the solution easier.

4.3.2 Flask

As of requirement **NF_3**, the solution's interface is to be made accessible in a web browser. For handling the logic of hosting the pages and communicating with the background processes, Flask has been chosen as it is open source and can handle calls to Python scrips, which makes it an ideal choice for the development of the solution discussed in this thesis. Flask is a micro web framework written in Python, which makes it not require any additional tools or libraries to function, and is also platform independent, which makes it fulfill requirement **NF_2** as well.

5| Implementation

The purpose of this chapter is to present how the requirements from **Chapter 3.7** have been implemented for a prototype. However not all of the requiriements will be implemented, only those which have been prioritised with must throught the MoSCoW analysis in tables **Table 3.1** and **Table 3.2** as those have been deemed the most important requirements. This chapter covers how the neural network is executed, how the data is loaded into the browser of the application and how the web interface is managed.

5.1 Executing the Neural Network

As discussed in **Chapter 3.1.1**, the neural network has to be executed by the application to get the information on how it classifies the image. An efficient way to do this is through the Python deep learning library called Keras, which is open source and is well documented[1] for tasks like this. Keras can be imported to the Python script discussed in **Chapter 4.3.1**.

To execute the neural network and then show its results on a page hosted by Flask, first the necessary libraries have to be imported and then the network has to be loaded. It is assumed that before this point the network's and the dataset's location has been specified, as it has been discussed in the flow of the processes in **Chapter 4.1**. A code snippet showing how the Flask application and the model for Keras is set up is shown on **Listing 5.1**.

```
1  from keras.applications import InceptionV3
2  from flask import Flask
3  import io
4
5  # initialise the Flask application and the Keras model
6  app = flask.Flask(__name__)
7  model = None
8
9  def load_model():
10     # load the model of the neural network into Keras
11     model = InceptionV3(weights="imagenet")
```

Listing 5.1: Python code for loading the Flask application and the neural network into the memory

The code snippet shown on **Listing 5.1** first imports Keras' InceptionV3 library, as in this case

[1]Keras' documentation is available at https://keras.io/

it is assumed that the neural network specified earlier has been trained earlier using the inception V3 architecture, and if the network has been trained based on another architecture this part of the code should be changed to that accordingly. Then Flask is imported as the framework of choice for locally hosting the application's website, as discussed in **Chapter 4.3.2**.

After the network has been imported and an image has been chosen from the dataset by the user from the browser, the network is to be executed on it to rank the images by their activation scores, such as the case shown on **Figure 4.6**.

```
1  # load the image through Flask
2  image = flask.request.files["image.jpg"].read()
3  image = image.resize(224,224)
4
5  # the input image is classified and the list of predictions is initialised
6        predictionsTemp = model.predict(image)
7        results = imagenet_utils.decode_predictions(predictionsTemp)
8        data["predictions"] = []
9
10       # the results are looped over and are added to the list of predictions
11       for (imagenetID, label, probability) in results[0]:
12            r = {"label": label, "probability": float(probability)}
13            data["predictions"].append(r)
```

Listing 5.2: Python code for executing the neural network to classify an image

The code snippet on **Listing 5.2** loads an image to be classified by the neural network, which in the case of the live application is to be selected by the user. Then this image is resized to be 224 by 224 pixel in size, after which it is ready to be added to be processed by the neural network. The image is passed to a method set up by Keras, inherited by the *model* variable from the code snippet shown on **Listing 5.1**.

After this, the predictions are decoded with an utility called imagenet_utils from the Keras module, which is put into an array called *results*. As it is more convenient to display lists in Flask than arrays, this *results* array is then parsed over with a for loop, and its contents are added to a previously initialised list called *predictions*, which is then ready to be shown in Flask to the user.

5.2 Hosting the Website Locally

As discussed in **Chapter 4.3.2**, the application's interface is to be made available to the user through the use of the Flask framework. This chapter is to show how this is done, and will assume that the microframework has already been deployed on the host with the installer discussed in **Chapter 4.2**.

Provided that the sketches presented in **Chapter 4.2** all have their corresponding HTML templates deployed on the client, they can be referred to through Flask with dynamic attributes being passed to and from through the framework. For example, in the case of hosting the initialisation screen shown on **Figure 4.3** this is done as shown on **Listing 5.3**.

```
1  @app.route('/', methods=['POST', 'GET'])
2  def next():
3    error = "None"
4
5    # query the location of the network, the dataset and the training history
6    networkPath = request.form['networkForm']
7    dataPath = request.form['dataForm']
8    historyPath = request.form['historyForm']
9
10   # check if the network path and dataset path fields are filled, if they are
         not, display the same page with an error message, and if they are, load
         the main menu (the history field is not manditory)
11   if networkPath == "" or dataPath == "":
12     error = 'No path provided for the network or for the data.'
13     return render_template('initialisationMenu.html', error=error)
14   else:
15     return render_template('mainMenu.html', networkFolder=networkPath,
           dataFolder=dataPath, historyFolder=historyPath)
```

Listing 5.3: Python code for the dynamic functionality of the initialisation screen

The code snippet on **Listing 5.3** shows a function which is set to the index page running on the default directory, so in the case of the application being run on the client of the user, it would be accessible in the browser either at localhost or by going to a loopback IP address, such as 127.0.0.1. The function declared in the code snippet is to be called when the next button is pressed on the initialisation screen, which is to look like as the one on **Figure 4.3**. When the button is pressed, the forms on the page containing the locations of the network, dataset and the training history are queried by a request method implemented by Flask, which are then stored in three variables. The contents of the first two of these variables are then checked, and

if they are not empty, the main menu is loaded with the location of the folders passed to it as arguments. If the network's path or the dataset's path was not given however, the initialisation screen is shown again with an error message.

6| Conclusion

The aim of this thesis was to develop a tool for visualising convolutional neural networks. Based on this, a problem formulation was defined, which is as follows;

How to design a tool for visualising and understanding the behaviour of convolutional neural networks?

To further narrow down the topic, three sub-questions were defined, which are as follows;

How to tell if there is an issue with the network's performance, and how to educate users about this?

Typical problems have been identified and discussed throughout this thesis, which are then built into the prototype of the resulting solution. The solution is to check the uploaded neural network and its training history for signs of these problems that might cause inefficiency, and then the user is informed of possible measures that can be taken in order to improve performance.

How to make neural networks more accessible to people just getting started with them?

A number of visualisational techniques have been discussed in this thesis, which are to make the network more understandable and less like a "black box" for the users who are just getting started with neural networks. An easy to use prototype has been constructed, with features for analysing the inputted image and visualising which features of the network is analysing when making the classification. The architecture of the neural network is also shown in the prototype, with the activations of the neural network projected to it, with the intention of making neural network more approachable and less mysterious.

How to provide understanding for the input image in relation to the activations of the neural network?

It has been explored both from the image's and for the neural network's point of view what kind of activations take place based on the input image, and techniques such as partial occlusion, saliency mapping, filter visualisation and class modeling has been discussed. These methods have been compared, and it has found that the most insightful of them is the heatmap generated after applying the partial occlusion technique, which is implemented in the prototype resulting from this thesis.

6.1 Future perspectives

The tool proposed in this thesis could be further developed and integrated as a part of a software meant for training neural networks. This way information about the network, the training process and the dataset would already be available, so the visualisations could be shown right away, even while the network is being trained.

Such a graphical interface would make neural networks more accessible and it could be further improved with additional features to also show insight into the training process as well.

Bibliography

[1] A. Rosebrock. (2016). LeNet – Convolutional Neural Network in Python, [Online]. Available: https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/ (visited on 09/07/2018).

[2] N. Fraser. (1998). Neural Network Follies, [Online]. Available: https://neil.fraser.name/writing/tank/ (visited on 09/07/2018).

[3] NSW Department of Education. (2018). HSC Agile Software Development, [Online]. Available: http://web1.muirfield-h.schools.nsw.edu.au/mahara/view/view.php?id=6967 (visited on 09/07/2018).

[4] M. A. Nielsen, "Neural Networks and Deep Learning - Chapter 1, Chapter 3, Chapter 6", *Determination Press*, 2015.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems (MCSS)*, 1989.

[6] P. Lucidarme. (2018). How popular are neural networks over the years?, [Online]. Available: https://www.lucidarme.me/popular-neural-networks-years/ (visited on 09/07/2018).

[7] A. Prieto, B. Prieto, E. M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas, "Neural networks: An overview of early research, current frameworks and new challenges", *Journal of Neurocomputing*, 2016.

[8] A. Karpathy. (2017). Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition - Neural Networks Part 1: Setting up the Architecture, [Online]. Available: http://cs231n.github.io/neural-networks-1/ (visited on 09/07/2018).

[9] S. Sharma. (2017). Activation functions: Neural networks - towards data science, [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (visited on 09/07/2018).

[10] Y. Bengio. (2017). Deep Learning: Theoretical Motivations, [Online]. Available: http://videolectures.net/site/normal_dl/tag=983679/deeplearning2015_bengio_theoretical_motivations_01.pdf (visited on 09/07/2018).

[11] S. Ruder, "An overview of gradient descent optimization algorithms", *CoRR*, 2016.

[12] A. Karpathy. (2017). Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition - Backpropagation, [Online]. Available: http://cs231n.github.io/optimization-2/ (visited on 09/07/2018).

[13] (2017). Over-fitting vs Complexity of Models - StackExchange, [Online]. Available: https://stats.stackexchange.com/

questions/292283/general-question-regarding-over-fitting-vs-complexity-of-models (visited on 09/07/2018).

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] Stanford Vision Lab. (2017). ImageNet Large Scale Visual Recognition Challenge (ILSVRC), [Online]. Available: http://www.image-net.org/challenges/LSVRC/ (visited on 09/07/2018).

[16] A. Canziani, A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications", *CoRR*, 2016.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Curran Associates Inc.*, 2012.

[18] Google Scholar. (2017). Alex Krizhevsky's publications, [Online]. Available: https://scholar.google.com/citations?user=xegzhJcAAAAJ&hl=en (visited on 09/07/2018).

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions", *CoRR*, 2014.

[20] A. Karpathy. (2017). Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition - Optimization: Stochastic Gradient Descent, [Online]. Available: http://cs231n.github.io/optimization-1/ (visited on 09/07/2018).

[21] L. N. Smith, "No More Pesky Learning Rate Guessing Games", *CoRR*, 2017.

[22] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima", *CoRR*, 2016.

[23] A. Sharma. (2017). Understanding Activation Functions in Neural Networks, [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (visited on 09/07/2018).

[24] J. M. Klusowski and A. R. Barron, "Approximation by Combinations of ReLU and Squared ReLU Ridge Functions Controls", 2016.

[25] D. Becker. (2018). Rectified Linear Units (ReLU) in Deep Learning, [Online]. Available: https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning (visited on 09/07/2018).

[26] D.-C. Liu. (2017). A Practical Guide to ReLU, [Online]. Available: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7 (visited on 09/07/2018).

[27] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical Evaluation of Rectified

Activations in Convolutional Network", *CoRR*, 2015.

[28] D. Erhan, Y. Bengio, and A. Courville, "Visualizing higher-layer features of a deep network", *University of Montreal*, 2009.

[29] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?", *CoRR*, 2014.

[30] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks", *CoRR*, 2013.

[31] F.-F. Li, J. Johnson, and S. Yeung. (2017). Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition - Neural Networks Part 12: Visualizing and Understanding, [Online]. Available: `http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture12.pdf` (visited on 09/07/2018).

[32] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", *Visual Geometry Group, University of Oxford*, 2014.

[33] R. Henderson and R. Rothe, "Picasso: A Modular Framework for Visualizing the Learning Process of Neural Network Image Classifiers", *Open Research Software*, 2017.

[34] I. den Bakker. (2017). Battle of the Deep Learning frameworks, [Online]. Available: `https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750` (visited on 09/07/2018).

[35] T. Amaratunga. (2017). Visualizing Model Structures in Keras, [Online]. Available: `https://www.codesofinterest.com/2017/02/visualizing-model-structures-in-keras.html` (visited on 09/07/2018).

[36] A. Karpathy. (2016). ConvNetJS CIFAR-10 demo, [Online]. Available: `https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html` (visited on 09/07/2018).

[37] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks", *CoRR*, 2015.

[38] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization", *CoRR*, 2016.

[39] (2018). Improve Neural Network Generalization and Avoid Overfitting, [Online]. Available: `https://www.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html` (visited on 09/07/2018).

[40] ImageNet. (2018). About ImageNet, [Online]. Available: `http://image-net.org/about-stats` (visited on 09/07/2018).

[41] A. B. Raj, J. A. V. Selvi, K. Durairaj, and R. Singaravelu, "Intensity Feedback based Beam Wandering Mitigation in

Free Space Optical Communication using Neural Control Technique", *EURASIP Journal on Wireless Communications and Networking*, 2014.

[42] (2017). Feature proposal: detect dead ReLUs, [Online]. Available: `https://github.com/keras-team/keras/issues/7057` (visited on 09/07/2018).

[43] I. Sommerville, "Software Engineering. 9th Edition. Chapter 1", *Pearson*, 2011.

[44] NSIS. (2018). Nullsoft Scriptable Install System, [Online]. Available: `https://sourceforge.net/projects/nsis/` (visited on 09/07/2018).

[45] X. Han, Y. Zhong, and L. Zhang, "An Efficient and Robust Integrated Geospatial Object Detection Framework for High Spatial Resolution Remote Sensing Imagery", *Remote Sensing*, 2017.

[46] J. Koutník, J. Schmidhuber, and F. Gomez, "Evolving Deep Unsupervised Convolutional Networks for Vision-based Reinforcement Learning", in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, 2014.