Strategies for Extracting Knowledge from Convolutional Deep Learning Architectures

Jack A Lees

130190116

BSc Computer Science

Dr Jaume Bacardit

Word Count: 14602

Abstract

This project aims to implement and analyse techniques to extract knowledge from convolutional neural networks. Convolutional networks are at the cutting edge for image recognition but their inner workings are still thought of as being a ‘black box’. I aim to explore techniques to change this notion and show that they function logically in a hierarchical structure through visualisations of various properties and layer output.

Declaration

"I declare that this document represents my own work except where otherwise stated"

Acknowledgments

I would like to thank my dissertation supervisor Dr Jaume Bacardit, for his invaluable advice and support over the course of this project.

Table of Contents

Table of Figures

Glossary

AI – Artificial Intelligence is an area of computer science that deals with giving machines the ability to seem like they have human intelligence or the power of a machine to copy intelligent human behaviour.

NN – An Artificial Neural Net is composed of a large number of highly interconnected artificial neurons working in unison to solve problems. Artificial neurons are simple mathematical functions based on a model of biological neurons. Throughout this paper unless otherwise specified a neural network refers to an artificial one.

DNN – A Deep Neural Network simply means a Neural Net with lots of hidden layers. Higher layers form higher levels of abstraction which are much better at recognising objects and translating speech. However, they take huge computational power to run in a reasonable time.

CNN – A Convolutional Neural Network is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the human visual cortex and are normally tailored for computer vision tasks such as recognising images.

Hyperparameter – Parameters expressing higher level properties of the model such as its complexity or how fast it should learn. Hyperparameters are normally fixed values specified by the user before training starts.

Activation Function – Every activation function takes a single number and performs a certain fixed mathematical operation on it such as ReLU, sigmoid or tanh. An activation function introduces non-linearity into a neural network, without this the network would only be able to predict linear functions.

1 Introduction

1.1 Purpose

In the last decade, deep learning techniques have dramatically improved machine learning performance in numerous tasks, I focus specifically on convolutional neural networks (CNNs). Which since [1] won the ImageNet classification competition, where they achieved a top-5 error rate of 17% compared to 26% of the second-best approach, CNNs have been at the forefront of image recognition technology and are now seeing widespread use including companies such as Google for photo search and Facebook for automatic picture tagging.

Despite their apparent success, there is still no comprehensive understanding of the internal organization of deep neural networks, and they are often criticized for being ‘black boxes’ [2]. CNNs have been used to classify lung image patches with interstitial lung disease (ILD) [3] and if the user does not understand, or cannot explain how a model makes a decision then they cannot use it, especially in areas such as this where the prediction could directly affect a human life.

My project aims to perform a qualitative and quantitative assessment of visualisation techniques designed to explain how a CNN makes a classification using different datasets and networks to compare results.

1.2 Project Aim

My project aims to explore the systematic analysis of existing techniques for understanding convolutional deep learning architectures.

1.3 Project Objectives

1. Identify existing techniques for understanding or visualising convolutional deep learning architectures.
2. Identify and implement several popular deep learning architectures.
3. Collect a set of benchmark datasets that are suitable for the architectures selected.

4. Implement the techniques on the architectures with the datasets.
5. Perform a qualitative and quantitative assessment of the technologies across architectures and datasets.
6. Summative evaluation of whether my project has achieved its goal or not.

1.4 Objective Descriptions

Objective 1. With deep learning being constantly advanced by leading researchers and developers finding techniques to use will take careful research into the latest papers, choosing those that are not only interesting and useful, but also possible for me to implement given the constraints I have in difficulty and time.

Objective 2. There are a large number of different possible convolutional deep learning architectures that have been thoroughly studied which could be implemented, so a subset of these must be chosen. These will be chosen based on the techniques chosen in objective 1, techniques are designed with a specific architecture in mind meaning this choice is not a free one.

Objective 3. Again, this choice will be made depending on objective 1, techniques to extract information from a neural network will be designed with a type of dataset in mind. There is however some freedom in the choice of specific datasets within the task, for example with classification datasets ranging from image data such as ImageNet and MNIST to multivariate problems, such as protein structure prediction from biological data or heart disease prediction from health data.

Objective 4. Implementing the techniques should take up the bulk of my time as it will require me to first set up the neural networks then to train them with the chosen datasets adjusting the parameters until they are performing well.

Objective 5. This will take place once implementation is complete, the techniques will be evaluated quantitatively to see what information, if any, they extract from the networks and qualitative analysis of its usefulness in understanding how the network functions.

Objective 6. A qualitative analysis of whether these strategies work to explain the inner workings of deep neural networks will be performed to see if my dissertation has achieved its original aims, what changes I would make if any if I were to do it again and how work in this area could be advanced further.

1.5 Report Structure

The report is structured as follows. Firstly, there is section 1, the Introduction, containing the project purpose, aim and objectives.

Next, section 2, Project Methodologies, focuses on the approach to development that I took. Including technologies used, the schedule and any problems I encountered during the project.

Section 3, Background Research, containing the prior research that went into the project. This is focused on the theory behind neural networks and specifically convolutional neural networks, going into detail on the different possible layer architectures and operations performed at each stage.

Section 4, Design Research, contains a detailed analysis of the current state of the art technology in convolutional neural network research including network architectures, machine learning frameworks, and image datasets. As well as primary research into visualisation techniques for these architectures.

Section 5, Design, goes into more detail on the chosen techniques and architectures. Going into detail on why they were chosen, what I hope to achieve from their implementation, and how they function.

Section 6, Implementation, covers how I chose to implement the chosen strategies and architectures, including the deep learning packages I decided to use. Any hyperparameters for the networks or the techniques are listed here as well as a detailed description on the code and how it functions.

Section 7, Results and Evaluation, is an analysis of my implemented techniques. Including sample results and analysis of these results.

Section 8, Conclusion, discusses whether the project met its aim and succeeded in fulfilling its goals. Finally, there are suggestions on any further work that could be done and what would advance this topic further.

2 Project Methodologies

2.1 Technology

The main technologies used in my project were Python [4] and the Keras library [5]. Keras allows for the fast prototyping of convolutional neural networks as being built on top of TensorFlow which allows for more intricate design and analysis as well as GPU support which makes training networks much quicker. The use of Python was a necessity, almost all of the deep learning frameworks are written for Python and there exists a vast amount of resources to make its use easier.

Most work was done on a machine utilising a NVIDIA GeForce GTX 760 utilising cuDNN, the NVIDIA deep neural network library [6] which provides highly tuned implementations for standard routines such as convolution, pooling, normalization, and activation layers.

Many packages were used in implementation depending on the technique and the architecture that is being used. The most common of these are MatPlotLib [7] and NumPy [8]. MatPlotLib provides functions to plot graphs and save them to disk while NumPy is the fundamental package for scientific computing with Python, integrating heavily with Keras.

2.2 Approach

I took an agile approach to the software development which was required by the research based structure of the project. As when I started I did not know what I would be implementing, only that the aim was to explain convolutional neural networks, the requirements and solutions were constantly changing to reflect my research. Early in development lots of techniques were researched and implemented as basically as possible to test their worth, these included some in other programming languages such as R and MATLAB.

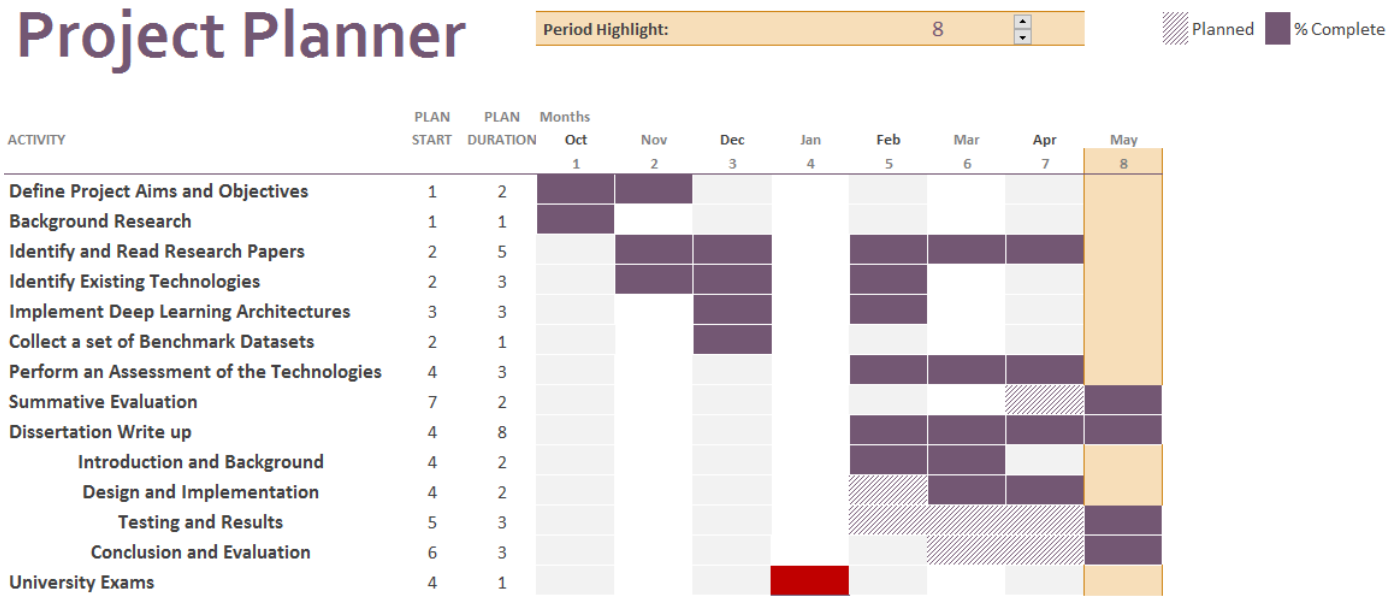
With having no final deliverable, planning my time was difficult due to a lack of an actual endpoint and I would not know if I had met my goals until the strategies were fully implemented and analysis had been done on them. This analysis being the key part of the project.

2.3 Problems

As I discussed in 2.2, a lack of a clear endpoint made time management for the project difficult and there was always more that I wanted to add. I had started with a different project in mind, one that focused on explaining the functionality of regular deep neural networks rather than convolutional ones however this proved difficult due to a lack of research in the area.

This project was my first time using Python, although not a problem, there was the added time constraint on learning its semantics and then progressing onto the deep learning packages. Over the course of the project and due to being extremely developmental in nature, the packages that I was using often updated, mainly Keras and TensorFlow, this could involve having to update all my existing codebase to reflect the update and in some cases breaking functions that have previously worked. Due to these fast updates, a lot of the example code and functions found online were out of date making development difficult.

2.4 Schedule



3 Background Research

3.1 Machine Learning

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has been behind huge advances in self-driving cars, speech recognition, web search, and has vastly improved our understanding of the human genome. Machine learning is so pervasive in modern society that the average person will use it dozens of times a day without realising [9]. Many researchers also think it is the best way to make progress towards human-level AI.

Machine Learning concerns machines improving from data, knowledge, experience, and interaction. Techniques to handle and learn from large and complex amounts of data are used by researchers in many disciplines including statistics, knowledge representation, causal inference, databases, natural language processing and machine vision [10].

Two important breakthroughs lead to the emergence of machine learning, the first being the idea that rather than teaching computers everything they need to know about data, that it might be possible to teach them to learn for themselves. The other being the emergence of the internet or big data, according to IBM 90% of the data in the world today has been created in the last two years alone. A lot of this data comes from places such as sensors that are used to gather climate information, posts to social media sites, pictures and videos, purchase records, and cell phone GPS signals [11]. Before the internet all this data was not publicly available or simply did not exist. Now it is simple to download a pre-labelled dataset with thousands of records.

3.2 Neural Networks

Defined by the inventor of the first neurocomputer, Dr Robert Hecht-Nielsen, a neural network is *'...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.'* [12]

An artificial neural network is a computational model used to approximate a function. They are based on a large collection of simple neural units called artificial neurons which are loosely

equivalent to the observed behaviour of a biological brain's axons. The neurons are connected and they interact with each other, they can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each neuron is called its activation.

Each link between neurons is associated with a weight, neural networks are capable of learning by changing these weights. How these change is dependent on the topology of the network, with the two most popular being feedforward neural networks and recurrent or feedback neural networks. In this paper, I only focus on feedforward networks. In a feedforward network the information can move in only one direction, forward. From the input nodes data goes through the hidden nodes and to the output nodes, there are no cycles or loops.

To train a feedforward network using backpropagation in conjunction with an optimisation method such as gradient descent it is a two-phase cycle of propagation and then weight update. First it is fed an input vector that propagates through the networks layers, until it reaches the final or output layer. The output of the network is then compared to the correct output using a loss function such as mean squared error; an error is calculated for each neuron in the final layer which are then propagated backwards through the network until every neuron in the network has an error value associated with it.

With backpropagation, these error values are used to work out the gradient of the loss function with respect to the weights in the network. Then this gradient is used by the optimisation method which updates the weights of each neuron, attempting to minimize the loss function. The importance of the backpropagation process is that at the end of training neurons in intermediate layers are organised as such that different neurons learn to recognise different characteristics in the input space, meaning that if a new input is presented to the trained network that contains a feature the neurons have learnt to recognise in training, even if it contains noise or is incomplete, then neurons will respond with an active output.

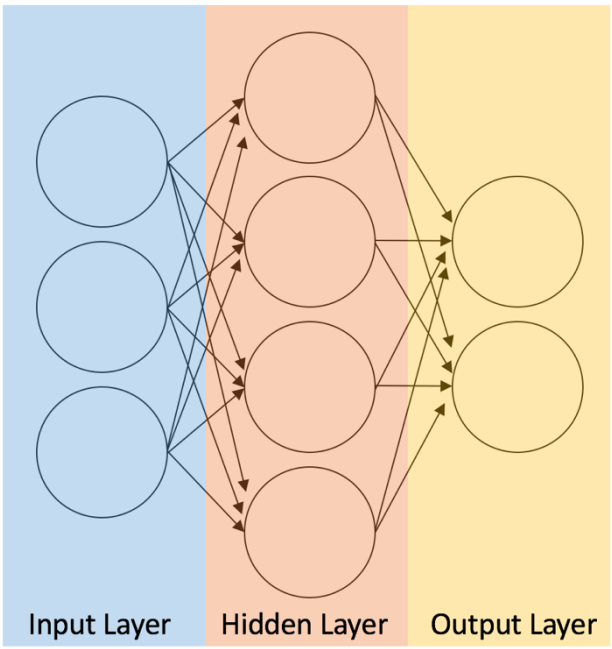


Figure 1. Neural Network with one hidden layer

3.3 Deep Learning

Deep learning is a small but rapidly developing subfield of machine learning that at the intersection of computer science and mathematics it is an approach to machine learning which has been crucial to developing the completely autonomous, self-teaching systems which are revolutionizing industries such as voice and image recognition.

Deep learning involves feeding a computational system a large amount of data which it can then use to make decisions on other data. Their inner workings are the same as a neural network using backpropagation as described before. However, we now have computers with enough computational power and large enough datasets to train very large networks. Large networks only increase in performance with the more data they are given, compared with older machine learning algorithms that reach a plateau of performance [13].

As the size and number of layers in a neural network increases the space of representable functions increases since more neurons can work together to express more complicated functions. This can be both positive and negative as if the network is too well fitted to the training data it will have failed to learn to generalize in new situations.

3.4 Convolutional Neural Network Theory

Focusing only on image classification, which is not the only use for a convolutional neural network. A convolutional neural networks task is to take an input image and output a class, or in some cases the probability of a number of classes that could best describe the image. This is a skill that comes naturally to us as humans, but machines have always struggled with. Being able to generalize from prior knowledge and adapt to difficult environments has always been very difficult for computer systems, i.e. identifying a poorly lit object or one that is partially obstructed.

When given an image as input a computer will see a matrix of pixel values, for a 28 x 28 px image in colour it will see a matrix of size 28 x 28 x 3, where the 3 stands for RGB values. A greyscale image of the same resolution would be 28 x 28 x 1, as it only needs to encode the pixel intensity of one colour, black. These are the only inputs a computer receives when it views an image so to make a classification requires some work.

The biological brain classifies images by their identifiable features such as a wing of a plane or the paws of a dog. It was shown that some individual neuron cells in the brain fired only in the presence of edges of a certain orientation, meaning some neurons would fire when exposed to vertical edges, and others horizontal or diagonal [14]. This is how a convolutional network works, it looks for low level features such as an edges or curves and builds them up through convolutional layers into more abstract features. This idea, that specific components of a system have specialised tasks is the foundation of convolutional neural networks.

A conventional neural network assumes that the position of data in the input is irrelevant, convolutional networks enforce translational weight sharing, i.e. as shown in Figure 2, data from the top left input neurons is passed to the top left of the first hidden layer and not anywhere else. In a fully connected network this information would be passed to every neuron in the next layer.

Convolutional networks are also not constrained to only two-dimensional data such as images, there have been advances using one-dimensional convolutional networks on audio data for spoken language identification tasks [15], and three-dimensional networks have been used on brain extraction from magnetic resonance imaging (MRI) data with state of the art results [16].

3.4.1 Convolutional Layer

Made up of a filter (kernel) which is just a smaller numerical matrix, normally of size 3x3 or 5x5, with the same colour depth as the image. This filter slides or convolves around the input image to produce a feature map (activation map). As it does this it multiplies the values in the filter with the original pixel values of the image, thus computing element wise multiplications. These multiplications are summed up giving a single value for each possible location of the filter as shown in Figure 2.

The values of the filters are learnt by the network during the training process however some hyperparameters must be specified including the number of filters, filter size and the architecture of the network. In general, the more filters the network has the better it is at extracting features from an image. Although like with the number of neurons in a layer, this has a limit.

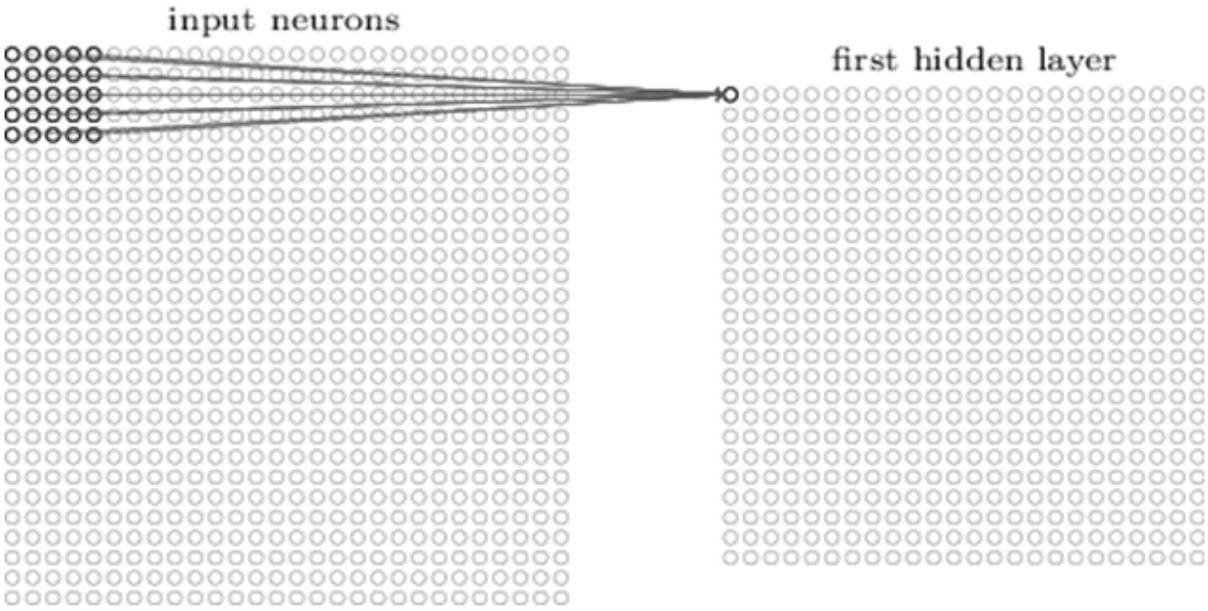


Figure 2. 5v5 filter producing first value of an activation map (Source: http://neuralnetworksanddeeplearning.com/)

3.4.2 Pooling Layer

Pooling is a form of non-linear down-sampling with max pooling being the most commonly used, as shown in Figure 3. Max pooling works by partitioning the input into a set of rectangles which do not overlap then for each rectangle, outputting the maximum. The most common form of max pooling is using filters of size 2x2 and stride = 2. This discards 75% of the activations,

due to how aggressive this down sampling is there is a trend towards using less pooling layers or none at all [17].

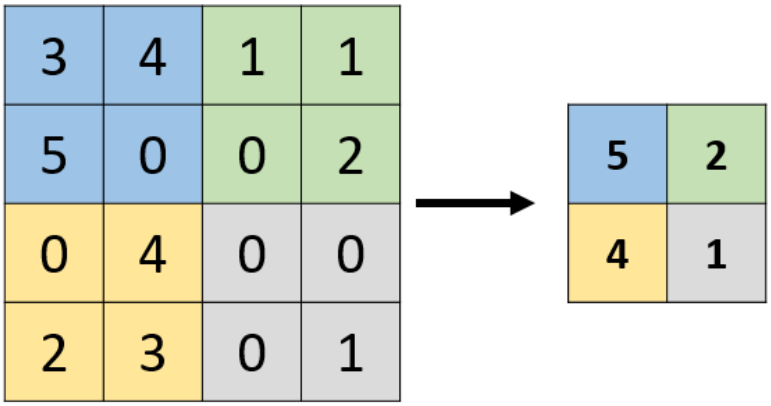


Figure 3. Max pooling with a 2x2 filter and stride = 2

3.4.3 Rectified Linear Unit Operation (ReLU)

The rectifier activation function as shown in Figure 4, is used instead of a linear activation function to add non-linearity to a neural network, using only linear functions the network would only be able to compute a linear function. The rectifier as a function, with x being the input to a neuron, is defined as:

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

A unit that uses the rectifier is called a rectified linear unit (ReLU). A ReLU layer performs a threshold operation, where any input value less than zero is set to zero i.e.

$$f(x)\begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

ReLU layers in a neural network improve the speed of training as its gradient computation is simpler than with other functions. Its use prevents the problem of vanishing gradient. The vanishing gradient problem occurs in sigmoid units, where the gradient essentially becomes 0 after a certain amount of training, this leads to sections of the network that stop learning any further. Also, shown in Figure 4 is the softplus function, defined as:

$$f(x) = \ln(1 + e^x)$$

It is largely similar to the ReLU function, apart from near 0 where it is a smooth curve. It is much harder to compute softplus and its derivative than ReLU, which happens very frequently during training, thus why it is not used.

'A 4 layer CNN with ReLUs converges six times faster than an equivalent network with tanh neurons on CIFAR-10 dataset' [18]

3.4.4 Dropout

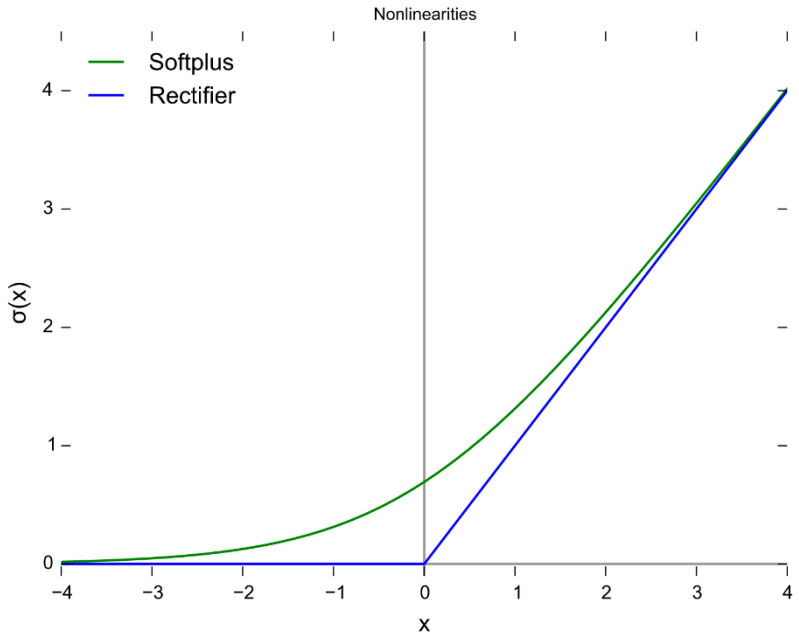


Figure 4. Plot of rectifier and softplus functions near x = 0

A regularisation technique to reduce overfitting when training a neural network. It works by randomly disabling nodes during the learning phase, this prevents neurons from co-adapting making overfitting less likely. Dropout is essentially a form of ensemble learning. Ensemble learning takes several smaller classifiers which have been trained separately and combines them, this aims to produce a stronger classifier than if they were combined then trained together as they each learn different features of the data and their mistakes are different [19].

3.4.5 Fully connected layer

Fully connected layers in a convolutional neural network are architecturally the same as a layer in a regular neural network in which neurons between adjacent layers are fully connected,

but neurons within a layer are not connected. Normally one or more of these layers is placed at the end of the network. After a fully connected layer there can be no more convolutional layers as they are not spatially located.

3.4.6 Softmax Function

Otherwise known as the normalized exponential function it is a generalization of the logistic function that takes a **K**-dimensional vector **Z** and outputs a **K**-dimensional vector σ(**Z**) of real values in the range (0, 1) that add up to 1. The function is defined as:

$$\sigma(\mathbf{Z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j} = 1, \dots, K$$

Used as the final layer in neural networks that perform multi class classification. As all the values are in the range of 0 to 1 and sum up to produce 1 they represent a true probability distribution.

4 Design Research

4.1 Convolutional Network Architectures

The most common structure for a convolutional neural network to take is several convolutional layers, each one followed by a ReLU layer, then a single max pooling layer. This pattern is then repeated one or more times until the data has been spatially reduced to a high enough degree. It is then fed into two or more fully connected layers, with the last layer having as many nodes as there are classes and using the Softmax function for multinomial logistic regression.

4.1.1 LeNet

LeNet was the first successful implementation of a convolutional neural network. Developed in the 1990's and still commonly used for simple recognition tasks, such as with the MNIST dataset [20]. Modern networks are still based partly on the LeNet model. LeNet has a very small memory footprint compared to some of the more modern networks allowing for fast development and training.

As shown in Figure 5 the lower-layers are composed to alternating convolution and pooling (sub-sampling) layers with the upper-layers being fully-connected. The input to the first fully-connected layer is the set of all features maps from the layer below, the original LeNet used the TANH activation function which was more commonly used at the time, modern implementations often use ReLU for faster training and better results.

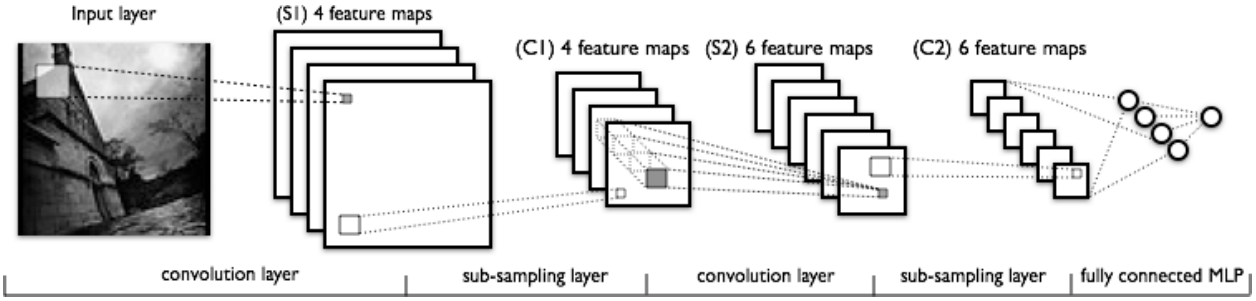


Figure 5. LeNet architecture (Source: http://deeplearning.net/tutorial/lenet.html)

Advantages:

- Small memory footprint.

- Quick prototyping and training.
- Simple structure.
- Unlikely to overfit to data due to its small size.
- Has been adapted to use modern convolutional techniques such as dropout.

Disadvantages:

- Only performs well on relatively simple recognition tasks.

4.1.2 AlexNet

The architecture that popularised convolutional neural networks. AlexNet won the ImageNet classification competition ILSVRC in 2012, where they achieved a top-5 error rate of 17% compared to 26% of the second-best approach [1]. Architecturally the network is similar to LeNet but uses more convolutional layers, 5 convolutional layers, max-pooling layers, dropout layers, and 3 fully connected layers, this time stacked one top of another.

Advantages:

- The most influential convolutional network.

Disadvantages:

- 60M parameters so slow to train.
- Outperformed by modern networks.
- Pretrained model only available in Caffe.

4.1.3 ZF Net

A refinement of AlexNet this won ILSVRC in 2013. It made the stride and filter size on the first layer smaller which preserves more information and expanded the size of the middle convolutional layers. The network contained 65M parameters.

'By using a novel visualization technique based on the deconvolutional networks of Zeiler et. al, "Visualizing and Understanding Convolutional Networks" [21] *, it became clearer what*

makes the model perform, and from this a powerful architecture was chosen. Multiple such models were averaged together to further boost performance.' [22]

Advantages:

- Arguably the most influential convolutional network.
- Pretrained models for ImageNet available in multiple deep learning environments.

Disadvantages:

- 60M parameters therefore slow to train.
- Outperformed by modern networks.

4.1.4 VGGNet

A team from the Visual Geometry Group (VGG) at Oxford University created this convolutional neural network architecture in 2014. Most popularly it has 16 or 19 weighted layers, that use 3x3 filters with a stride and pad of 1, along with 2x2 max-pooling layers. It features multiple 3x3 convolutional layers, up to three, back to back, meaning the effective receptive field is increased to 7x7. The number of filters doubles after each max-pooling layer due to the reduction of spatial dimensions. The 16 and 19 layer networks performed best and in 2014 were considered very deep though now there is the ResNet architecture which can be successfully trained with over 1000 layers on the CIFAR-10 dataset [23].

Advantages:

- Team release updated versions increasing architecture performance.
- First network to prioritise depth over the spatial dimensions of the image.
- Pretrained models for ImageNet available in multiple deep learning environments.
- Still used in many classification tasks.

Disadvantages:

- Contains over 160M parameters therefore very slow to train.

- Due to size and number of nodes the weights file is very large, taking up to 600MB of space.

4.1.5 GoogLeNet

Developed by a team at Google in 2015, GoogLeNet's improved utilization of the computing resources inside the network, achieved by a design that allows for increasing the depth and width of the network while keeping the computational budget constant. GoogLeNet managed to have over 100 layers in total but only 4M parameters thanks to using average pooling, going from a 7x7x1024 volume to a 1x1x1024 volume, and by the development of their 'inception module' which performs convolutional operations with filters sized 1x1, 3x3, and 5x5, and a pooling operation all in parallel. By itself this would produce too much data so 1x1 convolutions are used before the 3x3 and 5x5 layers and after the 3x3 max pooling as dimensionality reduction [24].

Advantages:

- Updated versions released regularly, latest is Inception 7a [25]
- Creative structure, first model to not stack convolutional layers sequentially.
- Weights file weighs under 100MB.

Disadvantages:

- Over 100 layers with many 1x1 and 3x3 convolutional layers, which does not have many parameters, but require computation and are therefore slow to train.
- Pretrained model only available in Caffe.

4.1.6 SqueezeNet

SqueezeNet succeeded in its goal to achieve AlexNet level of accuracy on ImageNet with 50x fewer parameters. Additionally, using compression techniques the team could compress SqueezeNet to less than 0.5MB which is 510x smaller than AlexNet [26].

Advantages:

- Memory footprint under 1MB.

- Pretrained models for ImageNet available in multiple deep learning environments.

Disadvantages:

- Notoriously difficult to train according to forum users [27].
- Similarly, to GoogLeNet it uses multiple 1x1 and 3x3 convolutional layers, which do not have many parameters, but require computation which slows down training to levels comparable to that of much larger networks, such as AlexNet.

4.1.7 Microsoft ResNet

Designed by Microsoft Research Asia and standing for Residual Neural Network this architecture was the winner of ILSVRC in 2015, it reformulated the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. Meaning that they take a standard feed-forward convolutional network and add skip connections that bypass several convolutional layers each time. Trained on ImageNet with up to 152 layers, up to 1000 layers with CIFAR-10. Due to its extremely deep representations the architecture managed to achieve 3.57% error on the ImageNet dataset [28].

Advantages:

- Recently, December 2015, produced state of the art results in image classification and detection.
- Difficult to program due to skipping layers.

Disadvantages:

- Very slow to train due to size. *'Using 4 NVIDIA Kepler GPUs… training took from 3.5 days for the 18-layer model to 14 days for the 101-layer model.'* [29]

4.1.8 CUImage

Current state of the art in both provided training data track and additional training data track of ILSVRC 2016 Object Detection Challenge. Used an ensemble of pre-trained models of previous winners to achieve 2.991% error on the ImageNet dataset [30].

Advantages:

- State of the art in image classification and detection.

Disadvantages:

- Very little progress from previous year's winner.
- No advances in architectural structure from previous year.
- Pretrained model available only in Caffe.

4.2 Image Datasets

4.2.1 MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset [31], created in 1998, contains images of handwritten digits written by high school students and employees of the United States Census Bureau. The MNIST database contains 60,000 training images and 10,000 test images. The current record lowest error rate of 0.21% was achieved by the Parallel Computing Centre in Ukraine was obtained in 2016 using an ensemble approach that combined 5 convolutional neural networks [32].

The images are centred in a 28x28px image by computing the centre of mass of the pixels then translating this point to the centre of the image. The images were originally black and white but now contain grey levels due to the anti-aliasing technique used when the images were normalized.

Advantages:

- Can quickly train a relatively small network to a very high level of accuracy with MNIST dataset.
- Readily available in most deep learning frameworks.
- Lots of tutorials on how to achieve near state of the art results.

Disadvantages:

- Too simple for the deeper networks, which are prone to overfit.

4.2.2 ImageNet

As of April 2017, 14,197,122 images in 1000 categories have been annotated by ImageNet indicating what objects are pictured [30]. Since 2010, the ImageNet project team at Stanford runs an annual contest called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where developer teams compete to accurately classify and detect objects and scenes. This competition has led to huge improvements in both the classification and detection tasks through architectures such as AlexNet and GoogLeNet which both came in 1st place in 2012 and 2015 respectively.

Advantages:

- Over 10 million annotated images.
- Many networks provide pretrained weights for this dataset.
- State of the art networks are often trained using ImageNet.

Disadvantages:

- Difficult to train a new network to an acceptable degree of accuracy with there being 1000 classes.
- Difficult to acquire the images, ImageNet does not own them. Just the third-party image URL's.

4.2.3 CIFAR-10 / CIFAR-100

The CIFAR-10 and CIFAR-100 are established computer vision datasets that are subsets of the 80 million tiny images dataset [33]. The CIFAR-10 dataset consists of 60000 32x32px colour images in 10 classes, with 6000 images per class. It has 50000 training images and 10000 test images. The record accuracy in CIFAR-10 is 96.53% achieved using Fractional Max-Pooling [34]. CIFAR-100 is set up just like CIFAR-10, apart from it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a ‘fine’ label, its class, and a ’coarse’ label, its superclass i.e.

Superclass: flowers **Class:** orchids, poppies, roses, sunflowers, tulips

Advantages:

- CIFAR-10 is one of the common benchmarks for machine learning so many networks provide pretrained weights for this dataset.

Disadvantages:

- Images are low resolution.
- CIFAR-100 is a very difficult dataset to train a network on due to the large number of classes.

4.2.4 Caltech 101 / Caltech 256

Caltech 101 contains 9146 images of objects belonging to 101 categories which were collected in September 2003 by a team at the California Institute of Technology. The size of each image is roughly 300x20px. Caltech 256 is similar however it contains 30607 images of objects in 256 categories.

Advantages:

- Images are freely available to download.

Disadvantages:

- Images are a mixture of resolutions, would require a lot of pre-processing before they could be fed into a neural network.
- No training / testing split like with other datasets.
- Caltech 256 is a difficult dataset to train a network on due to the large number of classes.

4.2.5 Common Objects in Context

A recent dataset, created by Microsoft that aims to:

'…advance the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding.' [35]

The dataset contains over 300,000 images in 80 object categories however a single image can have up to 5 captions. Recently the COCO 2016 Detection and Key Point Challenges have taken over from ILSVRC in being where the latest technology in machine vision is shown off as the best networks were reaching its true error point or the best possible with the errors in the dataset.

Advantages:

- Images are freely available to download.
- Recently created and the latest in machine vision.

Disadvantages:

- Images containing up to 5 captions are very difficult to caption and too complicated for my purpose.

4.3 Machine Learning Frameworks

4.3.1 TensorFlow

TensorFlow is an open source machine learning library developed by a team at Google Brain for internal Google use before being released to the public in 2015. The name TensorFlow stems from the multidimensional data arrays which are passed between nodes in a neural network. These multidimensional data arrays are referred to as tensors. TensorFlow uses a python API and is widely used in industry as well as in research [36].

4.3.2 Theano

Theano is a numerical computation library for Python developed by a machine learning group at the University of Montreal. It allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays using a syntax similar to that of NumPy's which is the central package for scientific computing with Python [37].

4.3.3 Keras

Keras is a high-level neural networks API, its primary author being François Chollet, a Google engineer. It is written in Python and capable of running using either TensorFlow or Theano as its backend. Its main aim is to enable fast experimentation with deep neural networks, with a focus on being minimal, modular and extensible.

4.3.4 Scikit-learn

Scikit-learn is a high-level machine learning library for Python containing mainly prebuilt solutions such as SVMs, Random Forests, Logistic Regression and a simple multi-layer perceptron model. For my purposes this will be too high level.

4.4 Visualisation Techniques

4.4.1 Effect of Convolutional Network Layers

This technique visualises the effect of each convolutional layer on a specific image by creating a model that only has a single layer then printing the output of the network as shown in Figure 6. The images appear very noisy because the network isn't trained so all the weights are randomly initialised.

This technique can show you the effect of changing the size of the kernel and the amount of information that is retained through the layers. It can also be used to show the effect of adding an activation function such as ReLU or a pooling layer simply by creating a model with one of these following a convolutional layer [38].

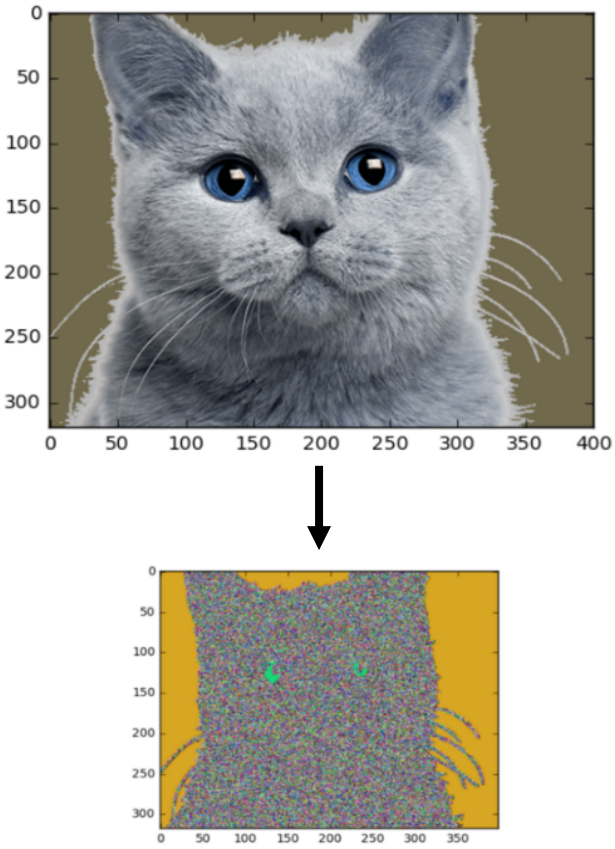


Figure 6. Effect of 1 convolutional layer with a kernel size of 3x3 and 3 filters (Source: https://medium.com/hacker-daily/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59)

4.4.2 Visualising Network Filters

This technique looks at what deep convolutional neural networks filters learn after training, and how they understand the images they are given. Input is found that maximize the activation of a unit, that is the sum of the inputs to the unit from the previous layer plus its bias, in any one of the different layers of a neural network architecture. The reasoning behind this technique is that the image that causes a unit to respond with its maximum value should be a good representation of what the unit itself is doing.

There are several ways to go about this, the first, and most simple method is to find an image from either the training or test set that produces the maximum value for its activation. This method can be extended to find a number of images that produce a high activation however from here there is the problem of how to find out what these images have in common, combining them is not a straightforward process.

The final technique works by instead of using an image from the training to test sets it starts from a grey image with random noise and runs gradient ascent with regards to the selected filters activation loss.

4.4.3 Class Model Visualisation

Given a trained convolutional neural network and a class that it has learn to detect, this method generates an image that is most representative of the specific class in terms of the networks class scoring model.

'More formally, let $S_c(I)$ be the score of the class c, computed by the classification layer of the ConvNet for an image I. We would like to find an L_2-regularised image, such that the score S_c is high:

$$\arg\max_{I} S_c(I) - \lambda||I||_2^2,$$

where λ is the regularisation parameter. A locally-optimal I can be found by the back-propagation method.'

Statement and function taken from: [39].

This operation is associated with the method of using back-propagation to optimise layer weights. However here the layer weights are fixed and the optimisation is executed with respect to the input image.

4.4.4 Visualization using a Deconvolutional Network

A deconvolutional network in its theoretical form is a network, with stacked layers and a structure like that of a neural network, which learns a set of filter masks and a set of activation matrices whose convolution would be equal to a given image patch [40]. A convolution operation of a filter mask with an image, which produces a set of activations is the reverse of a deconvolution. Which tries to predict the activations that when convolved with the filter mask would produce the original image hence the name comes from the deconvolutional operation being the opposite of a convolution operation.

A deconvolutional network can be used to invert the down sampling that takes place in a convolutional network and to revert the image to back it its original size. Deconvolutional networks were first designed to perform unsupervised learning. Here, they are not used in any learning capacity, just to probe an already trained convolutional network.

To do this first decide which filter activation to visualise. Not all filters can be used as many image patterns will cause a zero or near to zero activation, so picking a filter with a high activation is necessary to produce a useful visualisation. This can be done by selecting an activation then finding the images in the test dataset which produce the highest value for that activation when they are passed through the convolutional network.

Pass the image forward through the conv net, up to and including the layer where your chosen activation is. Zero out all the filter activations in the selected layer apart from the one you have chosen to visualise. Now use the deconvolutional network to get back to the image layer, the pattern that this produces is the discriminative pattern that the selected activation is sensitive

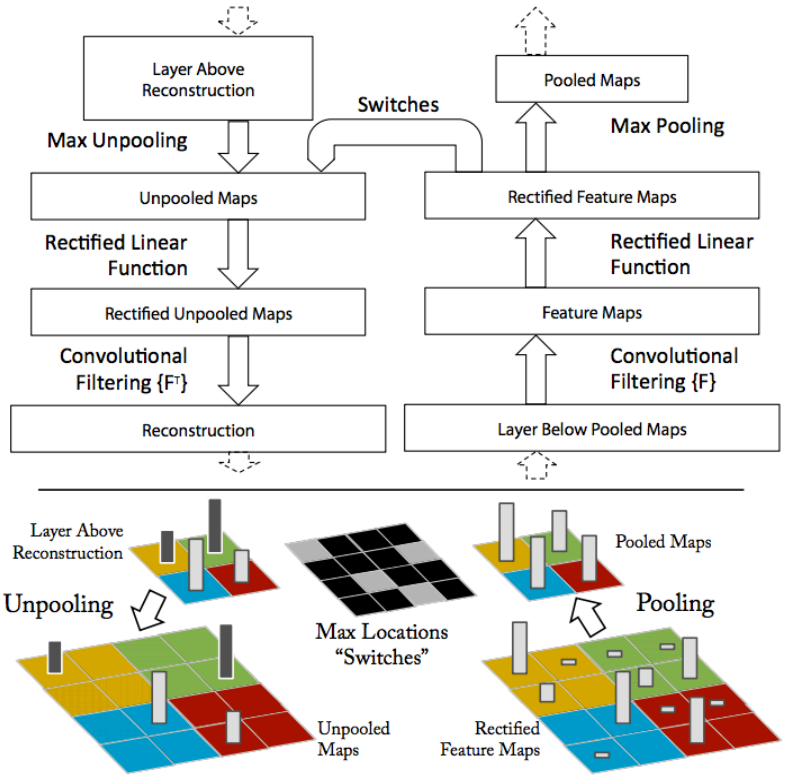


Figure 7. A deconvolutional network layer (left) attached to a convent (right) (Source: https://arxiv.org/pdf/1311.2901.pdf)

to. This is done using the inverse operations of pooling, ReLU and convolution which work as follows:

Deconvolution: Deconvolution is done by using the same filters as the corresponding convolutional layer however they are flipped horizontally and vertically.

ReLU: The inverse of the ReLU operation is performing the same ReLU operation as since convolution is applied to rectified activations during the forward pass, deconvolution should be applied to rectified reconstructed when doing a backwards pass.

Pooling: Max pooling cannot be perfectly inverted due to the lost information. The solution to this is to store in what they call a 'switch variable', the position of the max lower layer activation. Then while 'unpooling', the activation from the upper layer is copied to the position specified by the switch variable and all other activations are set to zero [41].

4.4.5 Local Interpretable Model-Agnostic Explanations (LIME)

LIME is designed to be 'model agnostic', meaning that this technique can be used no matter what the underlying model is. Its technique is to perturb the input and see how the predictions change. This can be done on convolutional networks by hiding some of the image and seeing how the predications change, but can also be used with other networks, for example a neuro-linguistic neural network that uses word embeddings by removing one or multiple words.

Shown in Figure 8 on the left is the image we want to explain its classification as a tree frog, on the right is the same image divided into its interpretable components done using superpixels [42]. To generate an explanation as to why a classifier has decided on a prediction generate a data set of perturbed instances by turning 'off' some of the interpretable components, this can be done by simply setting them to grey. Then for each instance, using the original model, get the probability of their being a tree frog in the image. Finally present the superpixels that produce the highest positive weights as an explanation, 'turning off' the rest of the image [41]

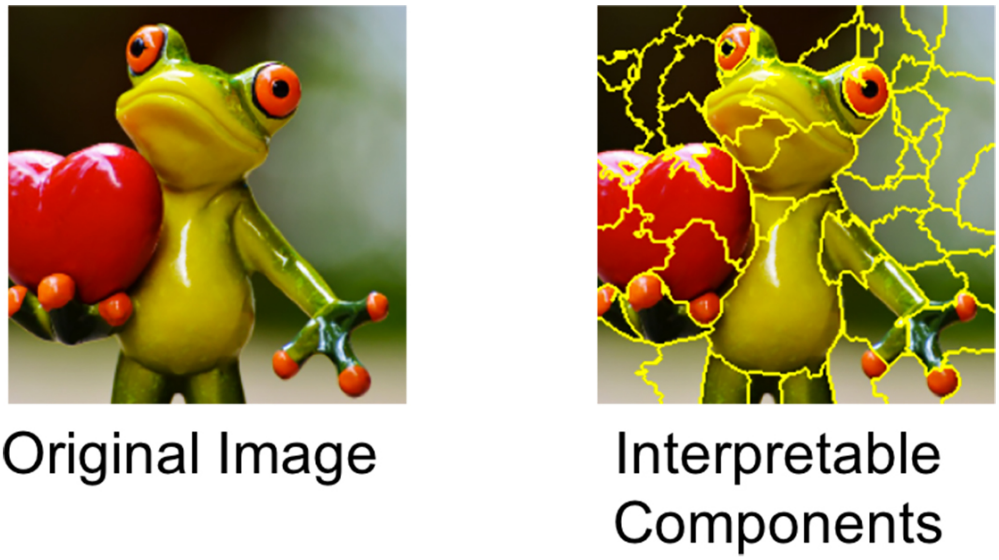


Figure 8. An image and its interpretable components (Sources: https://pixabay.com/en/love-valentine-s-day-pose-heart-903178/, https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime)

4.4.6 t-SNE

A branch of machine learning that uses unsupervised learning called manifold learning. It is an area of research to develop adaptable algorithms that can automatically discover a hidden low-dimensional structure in a high-dimensional dataset. t-Distributed Stochastic Neighbour Embedding (t-SNE) is a nonlinear dimensionality reduction technique that can be used to reduce high-dimensional datasets into a space of two dimensions. Rather than preserving the global structure like many other dimensionality reduction techniques in t-SNE similar objects are modelled by nearby points and dissimilar objects are modelled by distant points.

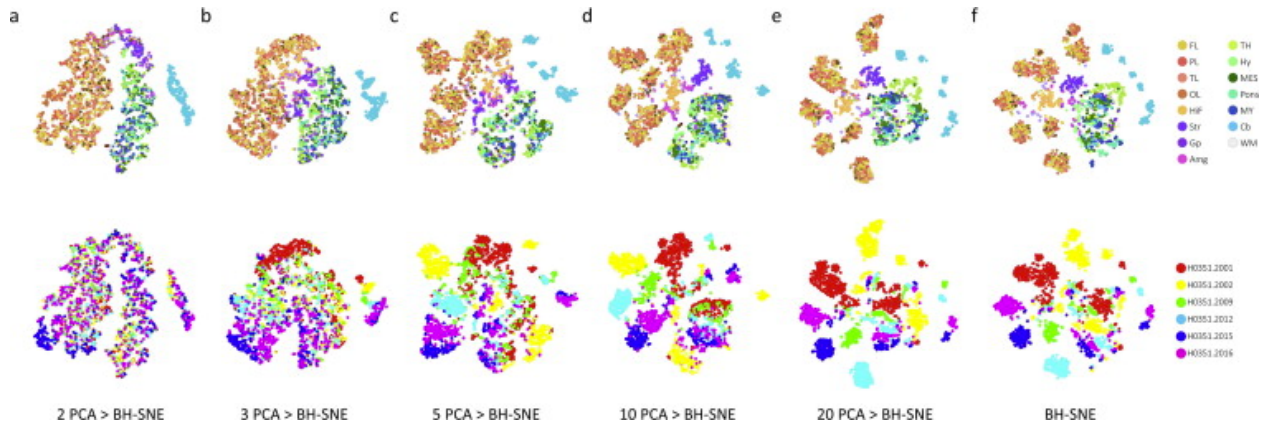


Figure 9. BH-SNE maps of the human data when using different PCA initializations (Source: http://www.sciencedirect.com/science/article/pii/S1046202314003211)

t-SNE is implemented in various languages and has been used by researchers to visualise data from Atari game states [43], to the spatial gene expression organization in the brain as shown in Figure 9 [44].

4.4.7 Activation Visualisation

This technique involves plotting the activation values for the neurons in each layer of a convolutional neural network in response to an image. Unlike fully connected neural networks which are not spatially informative, in a convolutional network the filters are applied such that activations on subsequent layers are arranged spatially. This technique can be used to determine which features are learnt at each layer of a convolutional network. Shown in Figure 10 is a visualisation of the activations of a convolutional neural network that has been trained to recognise faces, whether they are animal or human. As you can see the activations are high in the corresponding place to the faces in the images.

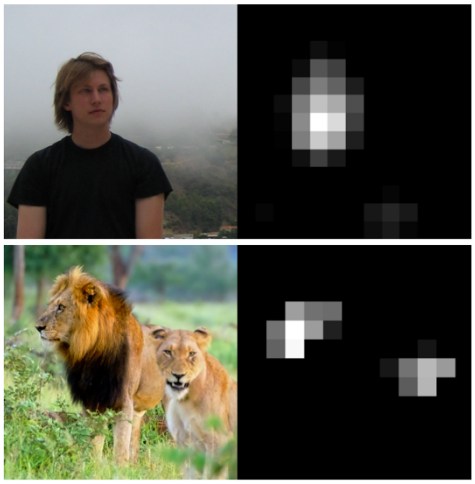


Figure 10. A view of activations of on the conv5 layer of a deep neural network trained on ImageNet (Source: https://arxiv.org/pdf/1506.06579.pdf)

5 Design

5.1 Chosen Techniques and Architectures

The decision on which of the techniques I had researched to implement and analyse was made based on which of them I believed would provide the most information on the inner workings of a convolutional neural network. I would have liked to be able to implement all the strategies that I researched but I was limited in my choice by the time that I had to complete the work and the lack of documentation or code examples with some techniques.

The architectures I decided on reflect my choice of techniques. They must suitable for the chosen technique but also be of varying structure as to provide interesting and contrasting results. I was limited by my ability with neural network packages, some networks such as CUImage were beyond my capabilities to implement in the timeframe, and I was limited by the availability of pre-trained data available. Training a network such as Microsoft's ResNet with its full 152-layer architecture would take roughly 3 weeks on four NVIDIA Tesla K80 GPUs [45], and as I had access to only a single NVIDIA Tesla K40 GPU I had to use pre-trained networks for the larger architectures such as VGG16 and GoogLeNet.

5.1.1 Neuron Visualisation

I chose to implement neuron visualisation as showing what image will maximally activate neurons in each layer of a trained network will allow me to compare what complexity of features is learnt at each level in a CNN. It should provide a lot of information about where in the network a classification is made. It is thought that the structure of the filters changes with the depth of the network, the lower hidden layers are simple features such as edge finding and colour recognition, which the higher layers combine into more abstract and complex features.The technique works by instead of using an image from the training to test sets it starts from a grey image with random noise and runs gradient ascent with regards to the selected filters activation loss. The output from this can be converted into an image showing the different properties of filters in different layers of the network. This function is given by the statement:

'Let θ denote our neural network parameters (weights and biases) and let $h_{ij}(\theta,x)$ be the activation of a given unit i from a given layer j in the network; h_{ij} is a function of both θ and the

input sample x. Assuming a fixed θ (for instance, the parameters after training the network), we can view our idea as looking for:

$$x^* = arg \quad max \quad h_{ij}(\theta, x).$$
$$x\ s.t.\ \left||x|\right| = p$$

Statement and function taken from [46].

There has been some work in this area lately attempting to improve the visualisations that the gradient ascent method produces, using techniques such as gaussian blur and L2 decay, [47], which I will analyse. I decided to implement neuron visualisation in Keras with the VGG16 network architecture which has been pre-trained with ImageNet datasets weights, as well as on the LeNet network on the MNIST dataset so that I can compare the results with those of my other methods that use the same architecture and dataset. With ImageNet containing images from 1000 classes, lots of which are very similar such as the Indian and African Elephant, the representation learnt by the network and therefore the neurons in the network should be free from noise and as close to a perfect representation of the classes as it is possible to achieve with the architecture.

This technique should show that as you get deeper into the network the filters become more intricate as they search for more abstract features [48].

5.1.2 Activation Visualisation

I implemented layer activation visualisation to allow me to produce a visualisation of the feature representation that each layer is learning. Visualising the activations on each layer in a convolutional network offers you a detailed investigation into exactly what the network is doing with a given input, and what it is seeing in each image that allows it to classify that image. Activation visualisation can plot any of the layers in a model, including pooling and normalization layers. Visualizing these layers provides information on how they function and what their effect is on the image.

I have implemented activation visualisation with the LeNet architecture on the MNIST dataset, the VGG16 architecture on the ImageNet dataset, and on a modified VGG16 network

called VGG4 which was trained on the CIFAR10 dataset. Implementing it on three different datasets and architectures will allow me to compare them to each other as well as give me a baseline for which to compare other techniques.

5.1.3 t-SNE

Because CNN's layers produce high dimensional arrays of data, their weights and activations, which are naturally very difficult to interpret by themselves, I implemented t-SNE with the aim of being able to make sense some of the data by reducing its dimensions down to 2. Arrays that are similar in their high-dimensional space will be represented by two-dimensional arrays that are close to each other, arrays that are dissimilar by arrays that are distant.

t-SNE functions by trying to minimise the Kullback-Leibler divergence between joint probabilities of the high dimensional data and low dimensional representation. Kullback-Leibler divergence is the measure of the non-symmetric difference between two probability distributions P and Q over the same variable x, or the information lost when $Q(x)$, which in this case is the low dimensional representation, is used to approximate $P(x)$, which here is the graph the t-SNE produces.

I have applied t-SNE to the MNIST dataset on the LeNet network and to the CIFAR10 dataset on the VGG4 network architecture. Implementation on VGG16 was not possible due to the dataset not being publicly available for download.

5.1.4 Local Interpretable Model-Agnostic Explanations

I have applied LIME to the GoogLeNet network architecture, specifically Inception v3, which will highlight contiguous patch of similar pixels (a super pixel) with either positive or negative weight towards a specific class explaining the model's predictions.

LIME's formal definition is given by: *"Let the model being explained be denoted $f : R d \rightarrow R$. In classification, $f(x)$ is the probability (or a binary indicator) that x belongs to a certain class. We further use $\pi_x(z)$ as a proximity measure between an instance z to x, so as to define locality around x. Finally, let $L(f, g, \pi_x)$ be a measure of how unfaithful g is in approximating f in the locality defined by πx. In order to ensure both interpretability and local fidelity, we must*

minimize L(f, g, π_x) while having Ω(g) be low enough to be interpretable by humans. The explanation produced by LIME is obtained by the following:"

$$\varepsilon(x) = \underset{g \in G}{argmin}\ L(f, g, \pi_x) + \Omega(g)$$

Where g ∈ G defines the explanation of the model, Ω(g) is a measure of complexity of the explanation g ∈ G. Statement and function taken from [41].

Essentially what LIME does is generate an explanation of an image by approximating the underlying model with a simple interpretable one, meaning a linear one that has few non-zero coefficients, which has been learned on a dataset of perturbations of the original image. This is a form of local approximation rather than global which is much harder to achieve. This will give me an insight into the superpixels of a picture that a model deems as most important to making its prediction, and those that impact negatively on the prediction. These areas can be compared to areas that humans use to recognise an image to see if they match, helping to provide insight into whether a CNN does work in a natural way like the human cerebral cortex does.

6 Implementation

6.1 Deep Learning Environment

Implementation was done entirely in python. Where possible the techniques were implemented using Keras with a TensorFlow backend. In some cases, implementation with Keras was not possible due to it not having the required features as it is designed for the implementation and development of neural networks, not research into their inner workings, so in these cases TensorFlow was used.

6.2 Convolutional Networks

6.2.1 LeNet

LeNet_mnist.py contains the code to train the LeNet convolutional network on the MNIST dataset using Keras which achieves a test accuracy of 98.92% and contains 113,386 trainable parameters.

MNIST is provided by Keras so is simply imported then shuffled between train and test data in an 80/20 split. Because the neural network requires input to be in the format (image height in pixels, image width in pixels, colour depth), each of the images in the dataset it reshaped adding on the extra dimension which in this case is simply a 1 as MNIST is greyscale. The pixel values are currently integers in the range of 0-255 so they are converted to the type float32 then divided by 255 so that they are in the range of 0-1.

Hyperparameter	Value
Epochs	12
Batch Size	128
Learning Rate	Adaptable
Rho	0.95
Epsilon	10^{-8}
Pooling Size	2x2
Kernel Size	3x3

Table 1. LeNet Hyperparameters

The class labels are converted into one-hot vectors meaning they have a single high bit, a 1, and all the others are 0. The networks architecture is then defined as a sequential model with the structure shown in Figure 5. The model is compiled using categorical cross entropy as the loss function and ADADELTA as the optimiser [49], ADADELTA is an adaptive learning rate method that dynamically adapts over time. After compilation, the model is fit to the training data and then saved in HDF5 format to *LeNet.h5*, which is designed to store and organize large amounts of data and can also be opened by Keras.

6.2.2 VGG4

VGG4_cifar10.py implements and trains a smaller version of the VGG16 network which is trained on the CIFAR10 dataset achieving an accuracy on the test data of 76.15%.

Implementation is like that of the LeNet network with regards to the loading of data and the shuffle between train and test data. However, the VGG4 network is deeper and wider, with 1,080,470 trainable parameters, almost ten times that of LeNet. VGG4's structure is a simplified version of VGG16's structure, with some of the intermediate layers and one of the fully connected layers removed to improve training speed. Its architecture is structured as follows:

Zero Padding > Conv > ReLU > Zero Padding > Conv > ReLU > Max Pooling > Zero Padding > Conv > ReLU > Zero Padding > Conv > ReLU > Max Pooling > Flatten > Dense > ReLU > Dropout > Dense

Hyperparameter	Value
Epochs	30
Batch Size	128
Learning Rate	RMSPROP
Rho	0.000001
Epsilon	10^{-8}
Pooling Size	2x2
Kernel Size	3x3

Table 2 VGG4 Hyperparameters

VGG4 is compiled using categorical cross entropy as the loss function and RMSPROP as the optimizer which uses the magnitude of recent gradients to normalise the gradient of the learning rate. VGG4's hyperparameters are displayed in Table 2. The model uses the Keras ImageDataGenerator for real time data augmentation of the training data, it can randomly shift images horizontally and vertically as well as flipping them horizontally. This can help to avoid overfitting on the data, as no two batches are ever the same and the network isn't being shown the same images over and over. Once trained, the network is saved in HDF5 format to *VGG4_CIFAR10.h5.*

6.2.3 VGG16

Displayed in Table 3 is VGG16's structure, where conv stands for convolutional and FC for fully connected. The number following is the number of neurons in that layer. The network achieves a 7.32% top-5 error on the ImageNet dataset. The VGG16 network contains 138,357,544 parameters, most of which are in the two large fully connected layers with 4096 neurons each.

Stage				
1	Conv-64	Conv-64	Max Pooling	
2	Conv-128	Conv-128	Max Pooling	
3	Conv-256	Conv-256	Conv-256	Max Pooling
4	Conv-512	Conv-512	Conv-512	Max Pooling
5	Conv-512	Conv-512	Conv-512	Max Pooling
6	FC-4096	FC-4096	FC-1000	

Table 3 VGG16 Structure

6.2.4 Inception v3

The Inception architecture makes use of inception modules, which allow the network to forgo the decision on which operation to perform at each layer. Instead it performs them all in parallel

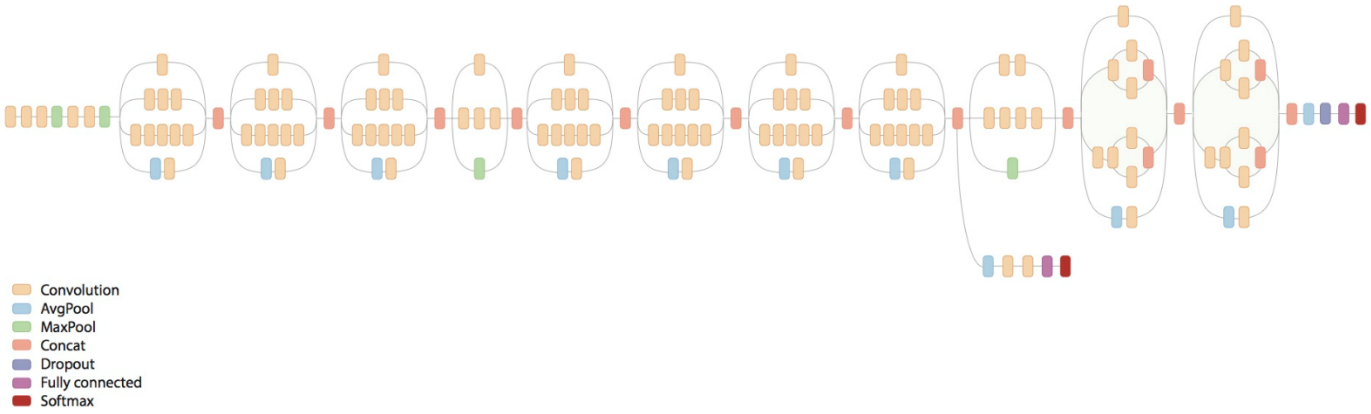


Figure 11. Inception v3 Architecture

(Source: https://github.com/tensorflow/models/blob/master/inception/g3doc/inception_v3_architecture.png)

and concatenating the resulting feature maps. This architecture allows the model to retain both local features with smaller convolutions and the more abstract larger features with large convolutions.

6.3 Visualisation Techniques

6.3.1 Neuron Visualisation

The idea behind neuron visualisation is to produce an input image that would produce the largest output from a neuron in one of the layers of a network. This is done by performing backpropagation from the output of the filter we have selected, back to an input image, giving us the gradient of the output of the filter with respect to the input pixels. This is then used to perform gradient ascent to find the input pixels that maximise the output of the selected filter. This gradient-based visualisation approximates visualisation using a deconvolutional network which is described in section 3.4.4.

This is done in the same way for both VGG16 and LeNet with the code contained in *VGG16_nv.py* and *LeNet_nv.py* respectively. First the network is loaded in either from the HDF5 file for LeNet or using the VGG16 network provided by Keras. The dimensions of the picture to be generated are then input, the name of the layer to visualise which can be found by using model.summary() and the number of filters in the layer to run gradient ascent on is selected.

The filters are then processed one by one. First a loss function is created that maximizes the activation of the selected filter of the layer that is being processed then the gradient of the input with regards to the loss is calculated. Then the gradient is normalised, by its L2 norm, of the pixels of the input image. This avoids very small and very large gradients which ensures a smooth gradient ascent and has an impact on the speed of convergence and stability. A Keras function that returns the loss and gradient given the input picture is used to run gradient ascent in the input space, with regard to the filter activation loss. This process is repeated for the number of filters selected. Filters that get stuck at 0 are skipped saving time. The filters are then sorted by their loss, the highest first, and the best are kept. These are then de-processed and saved as a PNG format.

Hyperparameter	**Value**
Steps	500
Step Size	0.1

6.3.2 Activation Visualisation

Implemented in the same way for the three networks, LeNet, VGG4 and VGG16 and datasets MNIST, CIFAR10 and ImageNet with code contained in *LeNet_av.py, VGG4_av.py* and *VGG16_av.py* respectively. Activation visualisation uses a Keras function to return the activations from an intermediate layer of a model in response to an image chosen by the user. These values are plotted using the matplotlib pyplot package with the neuron with the highest activation in that layer being labelled max. An image is created and saved for each layer and is saved in PNG format.

The only differences between the different architectures implementation is the number of layers that can be displayed, LeNet has 6 displayable layers, VGG4 has 9 and VGG16 has 18, and the number of neurons that can be displayed at each layer. Also, LeNet and VGG4 use the networks that I have created and trained while VGG16 uses the Keras provided pretrained network.

6.3.3 t-SNE

Implemented in *LeNet_tsne.py* and *VGG4_tsne.py* and using the sklearn t-SNE package first a Keras function is defined which returns the output of an intermediate layer of the network, here the user can select which layer of the network they want to visualise with t-SNE. The dataset is loaded into memory and then each value in the test data is reshaped, prefixed with another dimension, and fed into the Keras function defined earlier. What is returned is the activation of the selected layer in response to the image, these activations are saved in a high dimensional array which is fed into the t-SNE algorithm to reduce the dimensions down to two.

The most important hyperparameter for t-SNE is the perplexity, otherwise known as epsilon, the perplexity is the measure for the information that is defined as 2 to the power of the Shannon entropy [50]. In the t-SNE algorithm it defines how to balance attention between local and global aspects of your data governing the number of effective nearest neighbours for each value.

Hyperparameter	Value
Perplexity	30
Learning rate	1000
Iterations	1000

Table 4 t-SNE Hyperparameters

6.3.4 LIME

The InceptionV3 network as well as its weights are contained in the TF-Slim library. The LIME visualisation technique is implemented in *Inception_lime.py*, first two functions are defined, the first *transform_img* takes an image path as input and returns a 3-D float Tensor containing an appropriately scaled image to be input into the InceptionV3 architecture, *top5_predictions* takes an image as input and returns the top 5 predictions from the network. The

ImageNet class labels are retrieved as well as the activations which are passed through a TensorFlow softmax function. The weights are assigned to the Inception network from the Slim library.

The image which has been selected to have its prediction explained is then converted to the right format and dimensions and its top 5 predictions are printed out. The LIME package which is used to display the predictions takes as input the class number to get the explanation for, whether to only display positive influences on the decision, the number of super pixels to display and if to make the non-explanation part of the return image grey. It returns the im and mask where im is the 3D image array and mask is a 2D array containing the bounding boxes of superpixels used to explain the prediction. These values can then be used as input to skimage.segmentation.mark_boundaries which returns the input image with the boundaries between labelled regions, given by mask, highlighted.

7 Results and Evaluation

7.1 Evaluation Methodology

The networks are evaluated based on their accuracy which is compared with the state of the art result for that dataset and network. This is to ensure that the visualisation techniques can discern the most possible knowledge on how a convolutional network functions internally. Using a visualisation technique on a network that has been poorly trained or misfit to data will provide misleading information on how it functions.

The images produced by the visualisation techniques that I implemented will be evaluated qualitatively based on the information that can be discerned on how the underlying convolutional network functions. With the output of all techniques being images of various types quantitative analysis is very difficult.

7.2 Convolutional Networks

7.2.1 LeNet

My LeNet architecture achieves a test accuracy of 98.92% on the MNIST dataset test data and therefore a test error of 1.08%. The state of the art for MNIST is a test error of 0.21% meaning the difference between the two is 0.87%. I believe this to be an acceptable difference for my purposes.

7.2.2 VGG4

My VGG4 architecture achieves a test accuracy of 76.15% on the CIFAR10 dataset and therefore a test error of 23.85% which although not comparable to current state of the art of 3.47%, is comparable to the best achievable with a network of similar size.

[51] details a similarly sized convolutional network that was trained on multiple image datasets including MNIST and CIFAR10. While they could achieve state-of-the-art results on the MNIST dataset, 0.37% error, on CIFAR10 they believe that due to CIFAR-10 having relatively few training points, training the convolutional filters in Stage 1 so that they extract features that

are more discriminative for the specific dataset will be the most likely enhancement for improving results on CIFAR-10.

7.2.3 VGG16

This is the network that was submitted to the ILSVRC 2014 competition where it achieved a top-5 error rate of 7.32% on the ImageNet dataset. State of the art which was achieved by Inception v4 in 2016 is 3.08%. So, while not state of the art VGG16 provides me with a very high performance convolutional network with a relatively simple structure that has been applied to many machine learning tasks with great success.

7.2.4 Inception v3

Inception v3 achieved a top-5 error rate of 3.46% on the ILSVRC2012 classification task. State of the art is the latest version Inception v4 which achieves 3.08% a difference of only 0.38%, a very marginal amount. It was released in 2015 by Google along with its weights and provides a cutting-edge network to examine.

7.3 Improving Networks

To achieve a lower error rate, a more advanced architecture would have to be used such as a recurrent convolutional neural network. Most newly developed architectures are deciding to forgo using pooling due to the data that is lost due to its down sampling. Now that GPU computations have sped up training time its removal is possible, it would improve accuracy due to more information being retained by later layers but still slow down training.

Other techniques such as DropConnect rather than dropout could also be used to improve performance, when training with Dropout, a randomly selected subset of activations are set to zero within each layer. DropConnect instead sets a randomly selected subset of weights within the network to zero.

The ensemble approach to CNN's which has been steadily improving on the state of the art results in many categories is another way I could increase performance. Ensemble learning involves taking several smaller classifiers which have been trained separately and combining

them, this aims to produce a stronger classifier than if they were combined, then trained together as they each learn different features of the data and their mistakes are different.

7.4 Neuron Visualisation

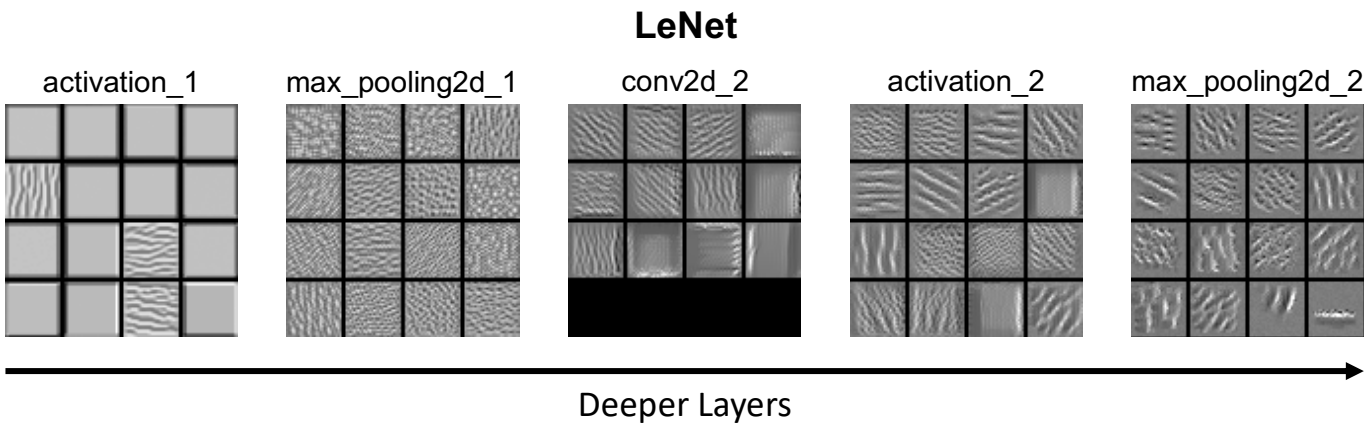
7.4.1 LeNet

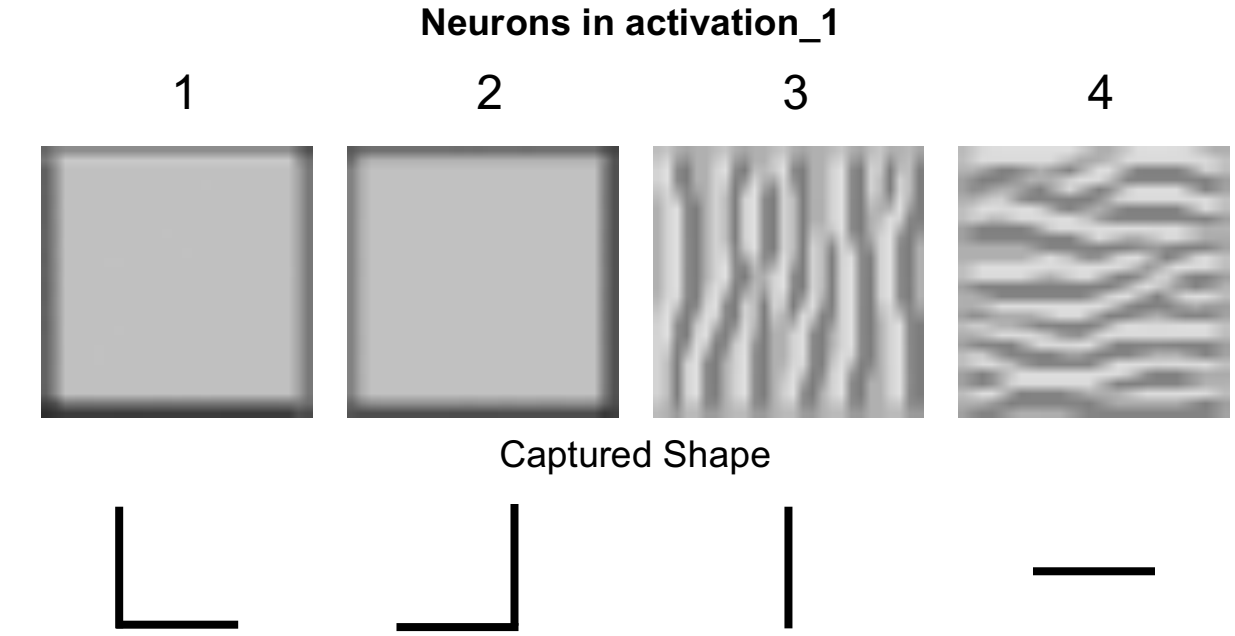
The visualisations produced by using this technique on the LeNet network doesn't seem to show a progression of more complicated features being represented the deeper the layer being represented is in the network. Shown at the top of Figure 12 is the neuron visualisation of all the possible layers of LeNet. The main drawback of this algorithm is that it cannot be run on every layer of the network. Once the network has been flattened, it is no longer possible to do gradient ascent on the layers so some of the deeper layers are missing, including the fully connected one. These may contain structures that are more recognisable as numerals.

Figure 12 shows four neurons taken from the first layer 'activation_1'. Neurons one and two are corner detectors, as indicated by the dark edges of the neuron. Three and four are edge detectors which appear to function like Gabor filters which are generally used in texture analysis, edge detection and feature extraction. They respond maximally to edges and change of texture.

The neurons in the layer 'activation_2' are from a deeper layer, the maximal activation for these appears more blurred and noisy than the lower layer. This may indicate that their weights, and therefore what they activate for is less clear.

Only the first layer appears to contain neurons that activate for corners in the input. The deeper layers activate for edges in different orientations with lots of neurons being copies of others but rotated, normally by 90 degrees. A network architecture that could rotate neurons might be able to be a lot smaller while also achieving the same accuracy.





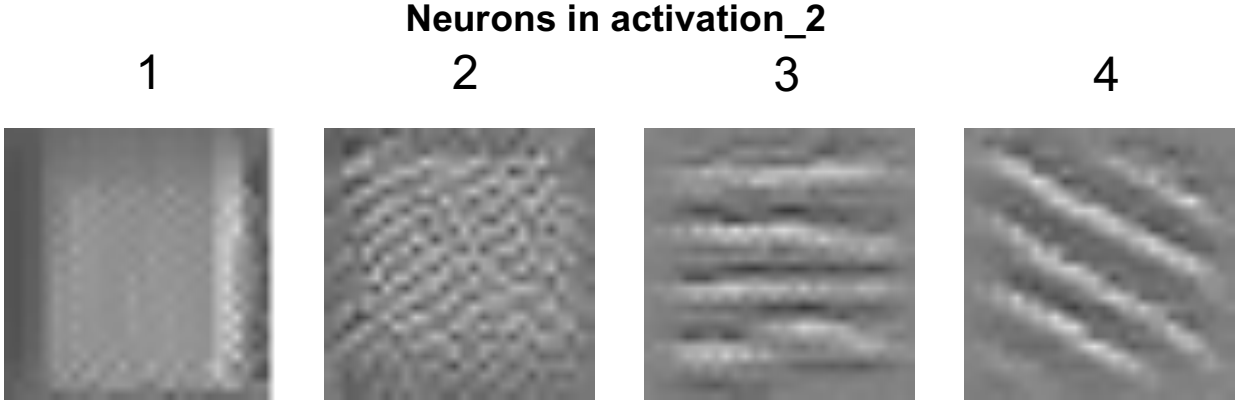


Figure 12. LeNet Neuron Visualisation

7.4.2 VGG16

Visualisations produced by neuron visualisation on VGG16 show a clear hierarchical model of the layers in the network. The first layers simply encode for colour and direction of edges. These colours and edges are combined by deeper layers into basic textures, such as grids and spots. These textures are then combined to create features that we might recognise such as eyes and dogs, although they are difficult to pick out.

Shown in Figure 13 are images that have been generated using gradient ascent to maximally activate neurons from the first block of the VGG16 architecture containing two convolutional layers and one max pooling. The first layer, block1_conv1, is composed entirely of neurons that encode for colour, edges and corners. Some of the colour and edge encoding neurons are repeated, some up to four times. This happens because, as the layers are not yet fully connected, information is not passed between neurons in the same layer and there is no function in the networks training to stop neurons learning the same feature. This could mean that if you were able to view this visualisation during training and you were able to lock the weights of certain neurons and reset others you could build a very efficient network, utilising all available space.

There is already a slight change from the first to the second convolutional layer, block1_conv2, as seen in Figure 13. It still encodes for colour, edges and corners, but now there are some slightly more complex features, what's interesting is how some of these filters appear to already be encoding for composite features as can be seen especially in neuron 3 in Figure 13. It is already becoming difficult to discern exactly what the filter is encoding for, neurons 1 and 2 are quite similar but it's hard to tell what they would activate for, I would have to guess for patches of colour.

Interestingly a large change occurs from the convolutional to pooling layers, there are still neurons which encode for edges but no longer neurons which encode only for colour. Instead there are more textures such as spots which can be seen in neurons 5 and 6 of Figure 13. This change is probably due to the dimensionality reduction that occurs due to the pooling operation, which might cause the earlier features to start combining into more abstract shapes.

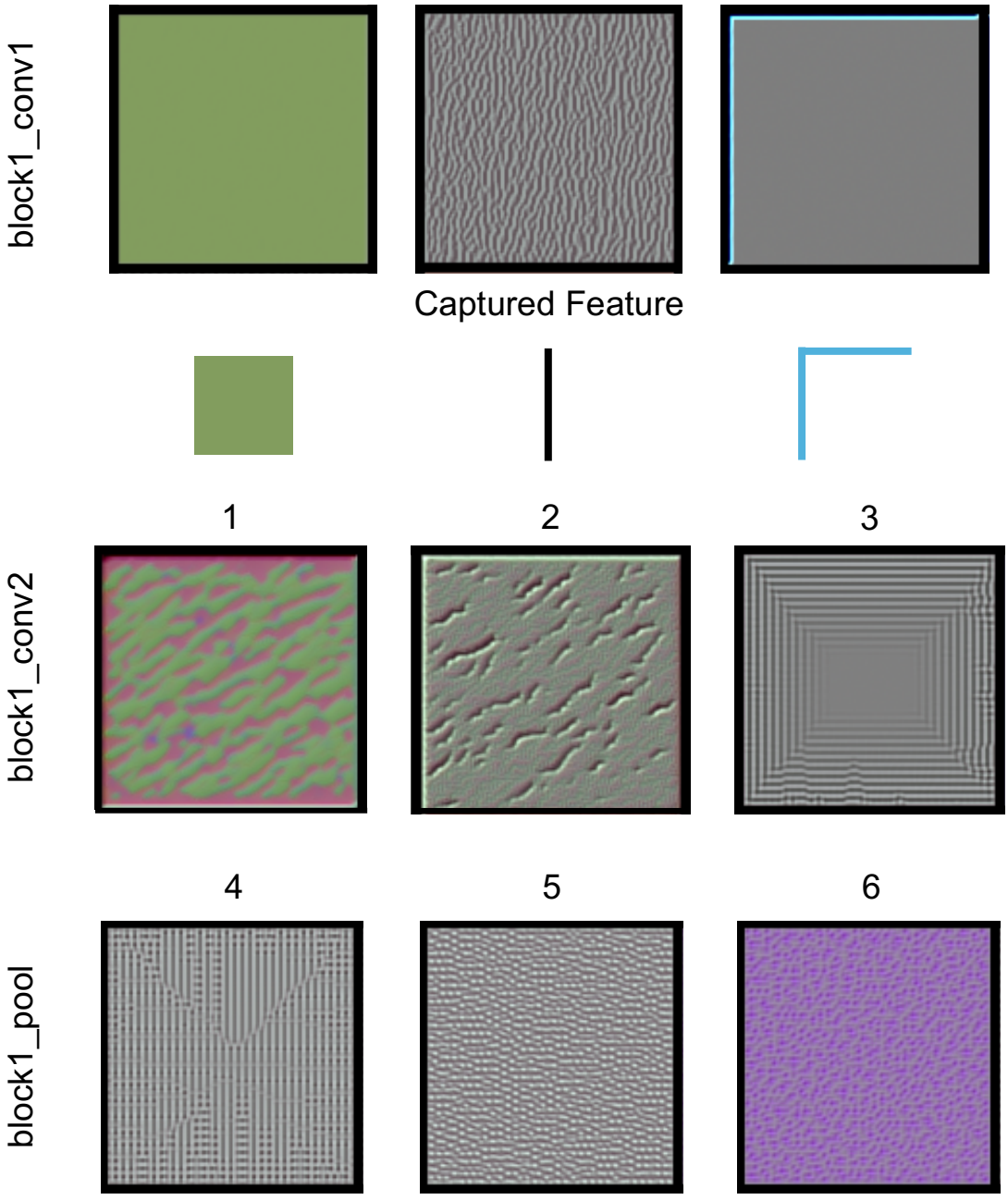


Figure 13. VGG16 Neuron Visualisation of First Block

Block 2 and 3 continue to search for textures and more complicated features, although none that are yet recognisable, I have not included any images from these however the 8x8 generated images for each layer are available in the supplementary material.

Block 4 starts to generate recognisable features. As can be seen in Figure 14, block4_conv3 contains multiple neurons which encode for eyes in different positions and orientations, open and closed, including eyelid and without. These are features that the network has learnt on its own in order to make its classifications, they are not classes included in ImageNet. This tells you that rather than just looking at patterns and textures to make predictions, like how some people predicted convolutional networks worked. Instead they build up from simple edges and colours, into features that would be recognisable to humans, like these eyes.

Block 5 is seemingly where the features and textures from the previous layers are combined to make the classes that featured in ImageNet. Clearly visible in Figure 14, block5_conv1 contains recognisable animal faces and bodies. With more work on the regularisation of the pixels and techniques to generate more lifelike images, I believe that most neurons in these layers would produce easily recognisable classes from the dataset.

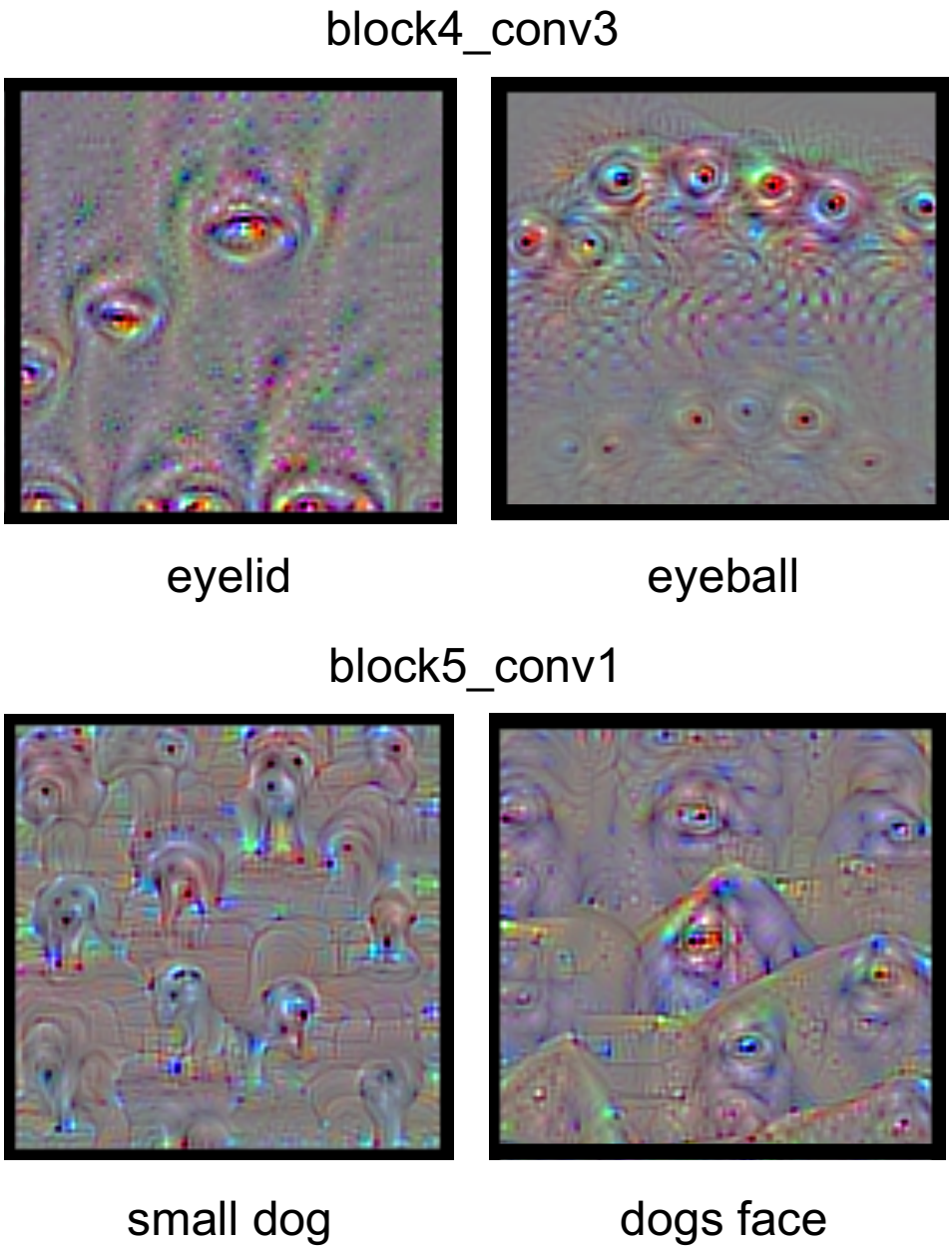


Figure 14. VGG16 Neuron Visualisation

7.5 Activation Visualisation

Activation visualisation was performed on three networks each with a different dataset, it provided a good insight into what areas of the image are causing neurons to activate in each layer as well as visualising the effect each layer has on the dimensional reduction of the picture.

Shown in Figure 18 is the activation visualisation for a digit from the MNIST testing dataset that has been passed through the LeNet architecture, Figure 16 contains the original image. Shown are 6 layers, although the 5th and 6th layers are the same, this is because it's displaying a layer that has no visual effect on the output of the previous layer.

This visualisation technique allows you to see what it is the network is doing with the image when it is input to the network. As you can see each layer transforms data from the previous layer into a higher-level representation of itself, meaning it compacts it, reducing the dimensions of the image. This can be seen with this visualisation as the dimensions of each image in pixels is plotted on each neuron. The size of the image is first cut in half by the second layer, from 26x26px to 13x13px, then again by the third layer down to 11x11px, it is reduced once more by the fifth layer down to 5x5px. This means there are only 1/27th the original number of pixels left at the final layer.

Also shown is the neuron that has the highest activation in the layer which can provide information on how it tells the figures apart. In layer 4 of Figure 18 you can see that it's the head and tail of a two that it uses to recognise it here which is similar to how a human would tell it apart from any other number, as these are its unique features. When two of the same numbers are passed through and compared the same neurons are often maximally activated, indicating that they are neurons which have learnt to recognise the unique features of this number. Interestingly in the first two layers there is a neuron that for each image activates only for the background, clearly it has learnt to activate only for the colour white.

Activation visualisation applied to the VGG4 network which was trained on the CIFAR10 dataset is similar to LeNet. The images are very low resolution, as can be seen in Figure 17, so as are the visualisations created. Shown in Figure 19 are all the layers where convolution or pooling

take place. All 9 layers can be viewed in supplementary material although these include the padding layers, which don't affect the image other than to add pixels around the edge.

As with LeNet the resolution of the image is reduced by the pooling layers, from 32x32px down to 8x8px. In Figure 15, which contains neurons from the third layer of the activation visualisation, you can see there are neurons which search for edges of different orientations. The first for vertical edges, the second for horizontal.

Activation visualisation on VGG16 is again, like both previous. It starts at a much higher resolution but this is quickly reduced, ending at just 7x7px, smaller even than the final representation of CIFAR10 on VGG4. Shown in Figure 20 are six layers from the possible 18 which are contained in the supplementary materials. Again, like LeNet, there are neurons that search for edges of a certain orientation, which can be seen especially in layer 15.

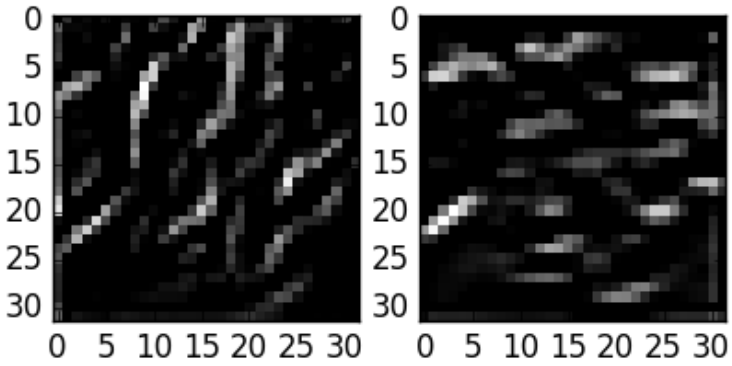


Figure 15. Neurons from layer 3 of VGG4 Activation Visualisation

Figure 16. Input to Figure 19.

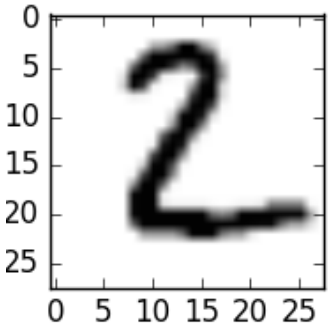


Figure 17. Input to Figure 18

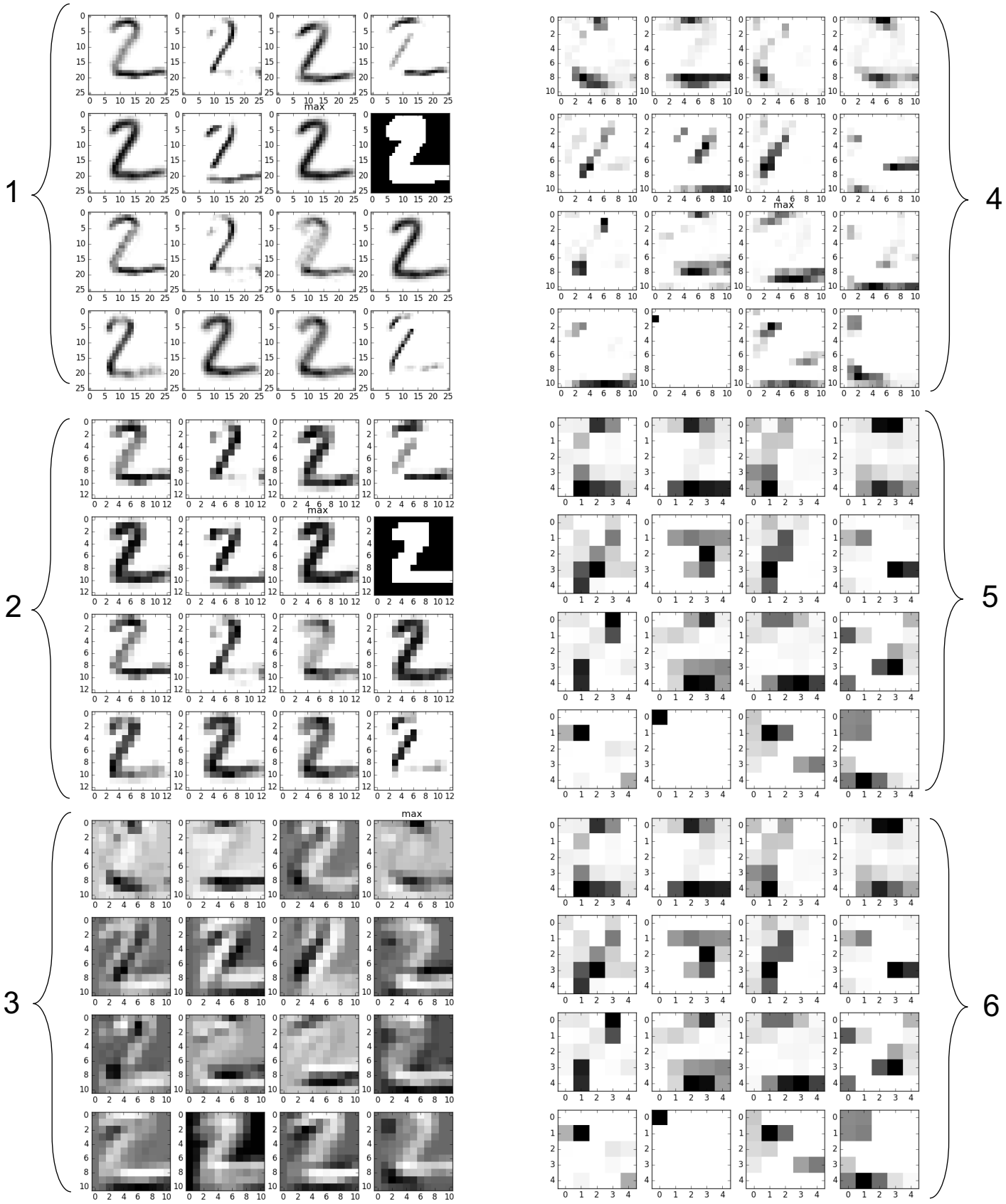


Figure 18. Part of LeNet Activation Visualisation

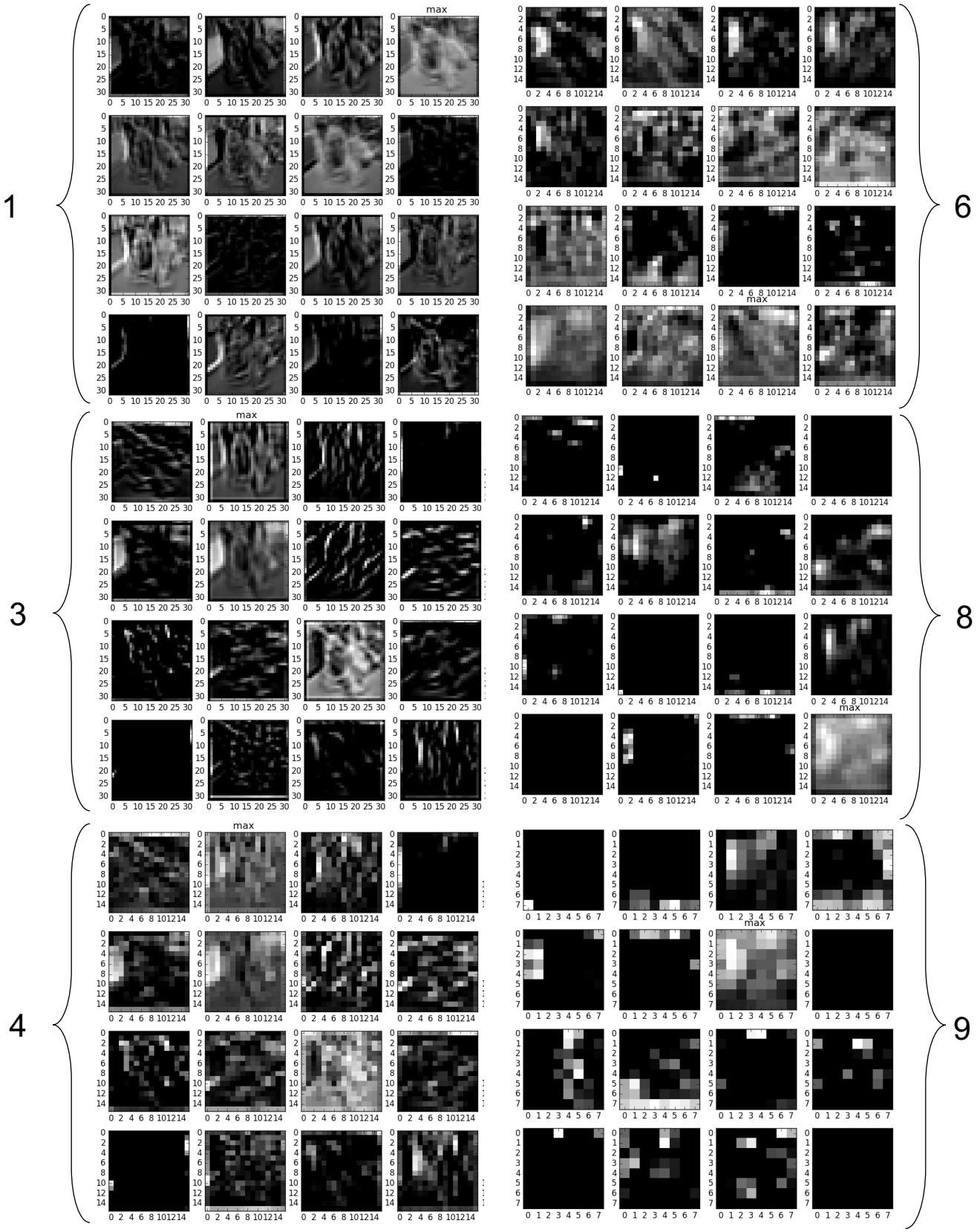


Figure 19. VGG4 Activation Visualisation

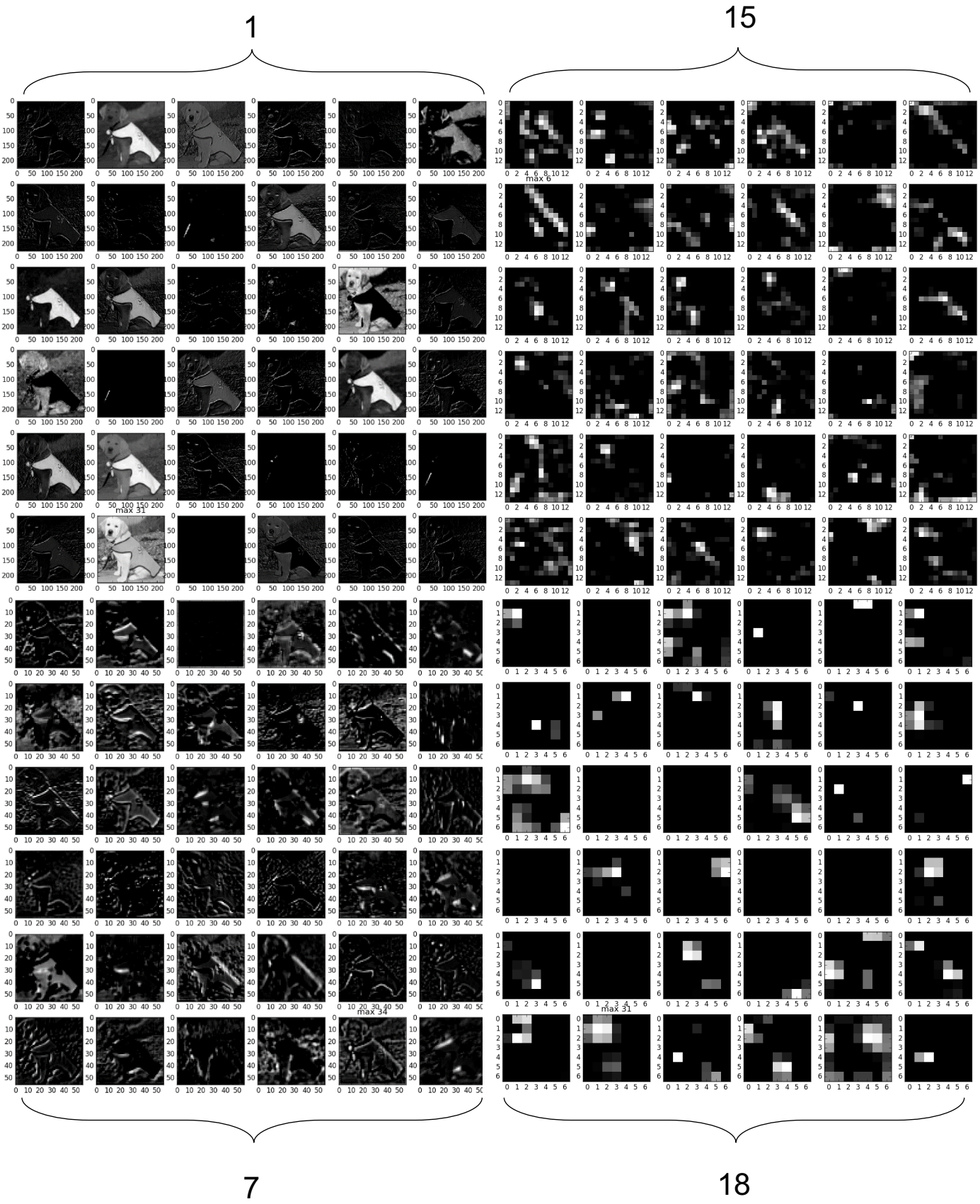


Figure 20. VGG16 Activation Visualisation

7.6 t-SNE

The t-SNE plots that I produced were the networks representation of the dataset at that point in training. As can be seen in Figure 21, initially there are some small groupings due to t-SNE being able to separate the dataset by itself, but over the course of training the digits form their own groups. This technique has often been used on the MNIST dataset so these results are not unique. What would be interesting would be to be able to see the images from the dataset, through an interface that allowed you to see for example if similar ones were close together, and which it had been unable to classify correctly. Contained in the supplementary material is a gif that shows the training process, which shows the slow grouping of similar numbers.

More interesting is the t-SNE of the CIFAR10 dataset on the VGG4 network architecture. Figure 22 shows the first layer's representation of the dataset. There are some small groupings but data seems to mainly be random, this makes sense as it indicates the network has not or cannot decide on the class yet. The small groupings seem to be made up of mainly 0's, indicating a picture of an airplane as the key in Table 5 shows. This could be because there is a feature in the first layer that strongly indicates that the class is going to be 0, I would guess that it's an abundance of blue sky in the picture, as the first layers mainly encode for colour and edges as neuron visualisation has shown.

Figure 23 shows the final layer's representation of the CIFAR10 dataset and while not as good as the final layer of LeNet on MNIST there are some clear groupings. Even with the relatively low training accuracy of this network it still produces a very good visualisation of the networks representation of the classes. Another interesting feature is that the classes that humans would deem similar, such as the automobile and the truck, and the deer and the horse are also grouped together in places.

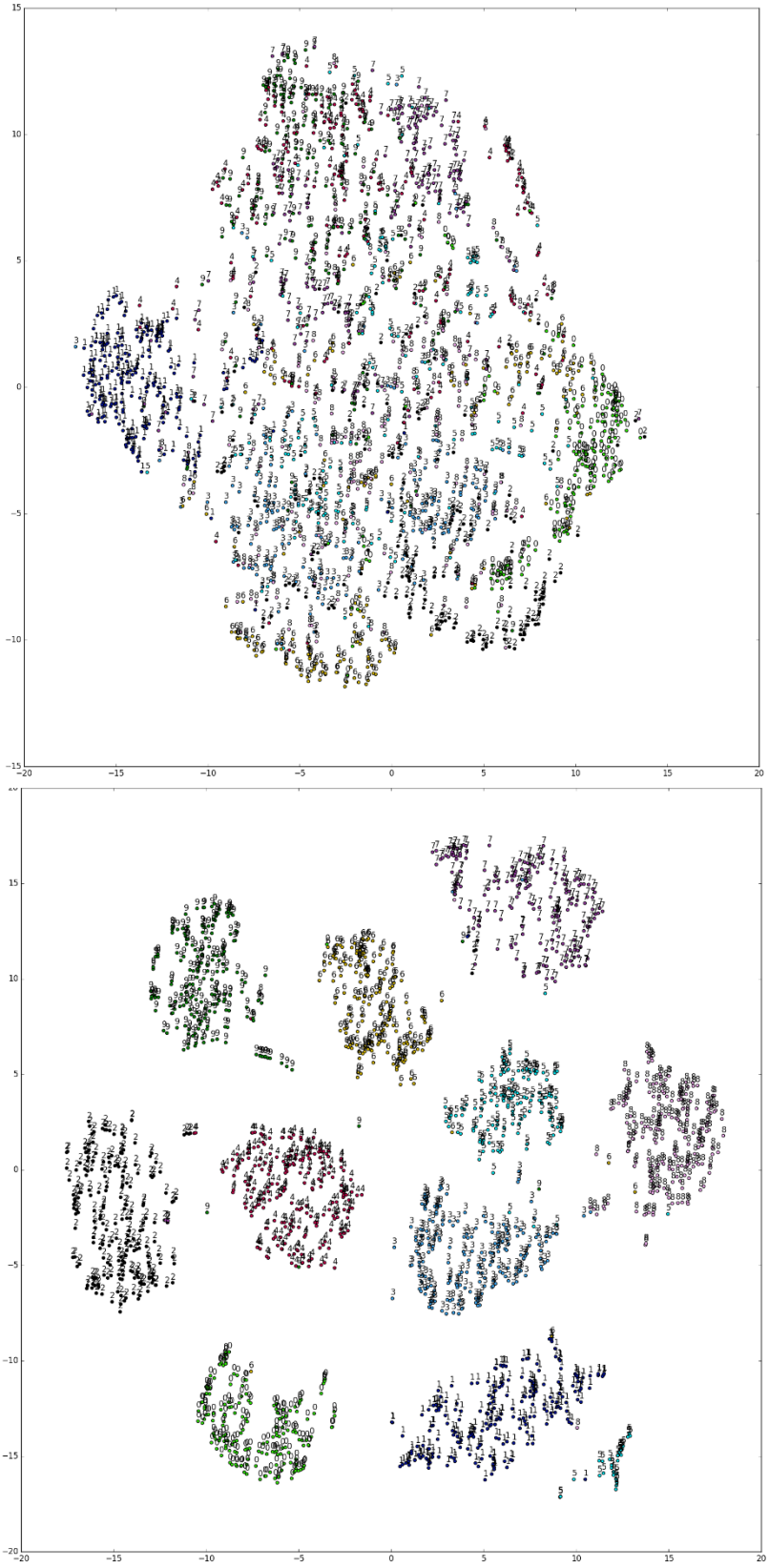


Figure 21. t-SNE on MNIST. Start and End of Training

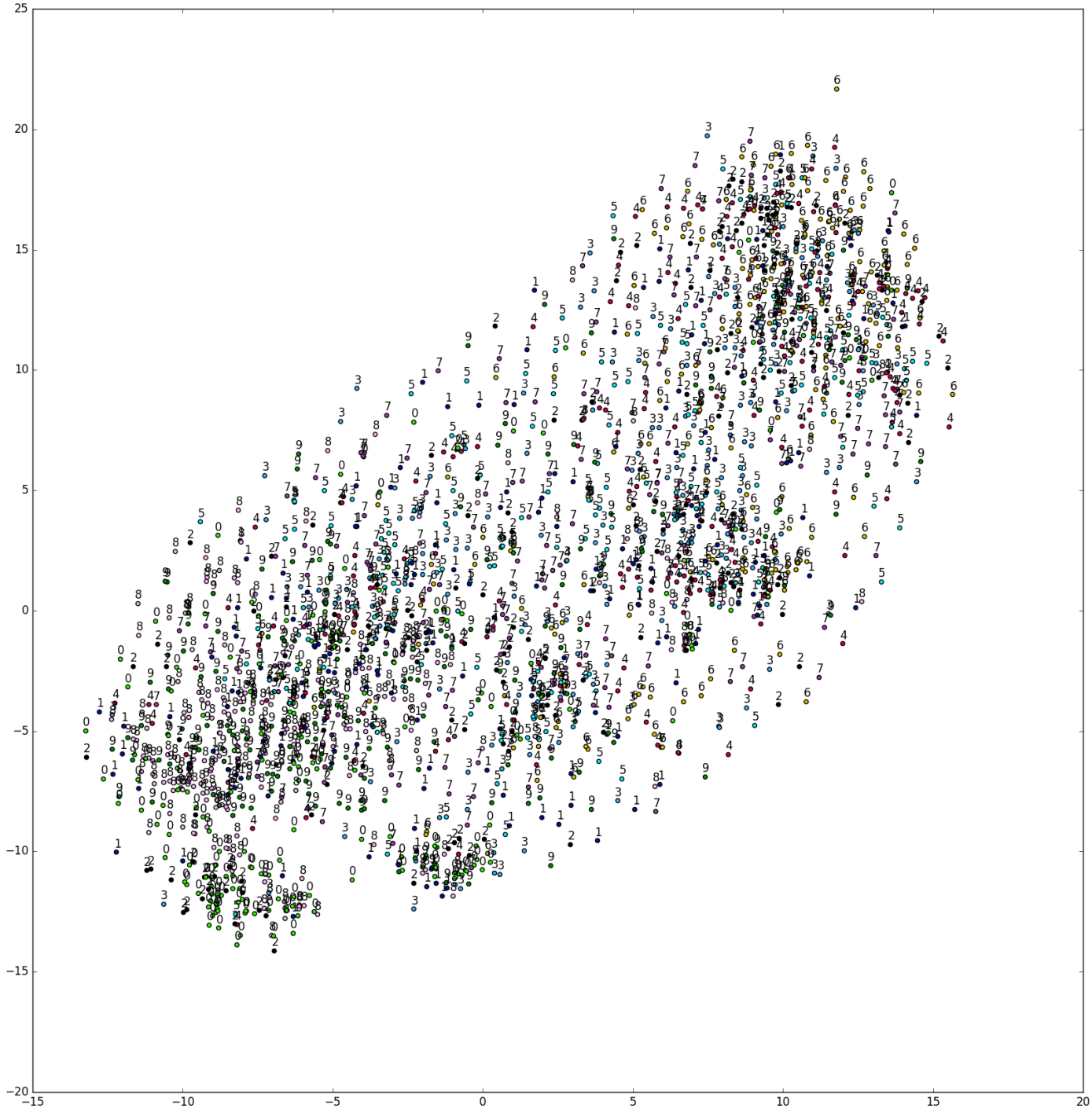


Figure 22. t-SNE on CIFAR10. First Layer Representation

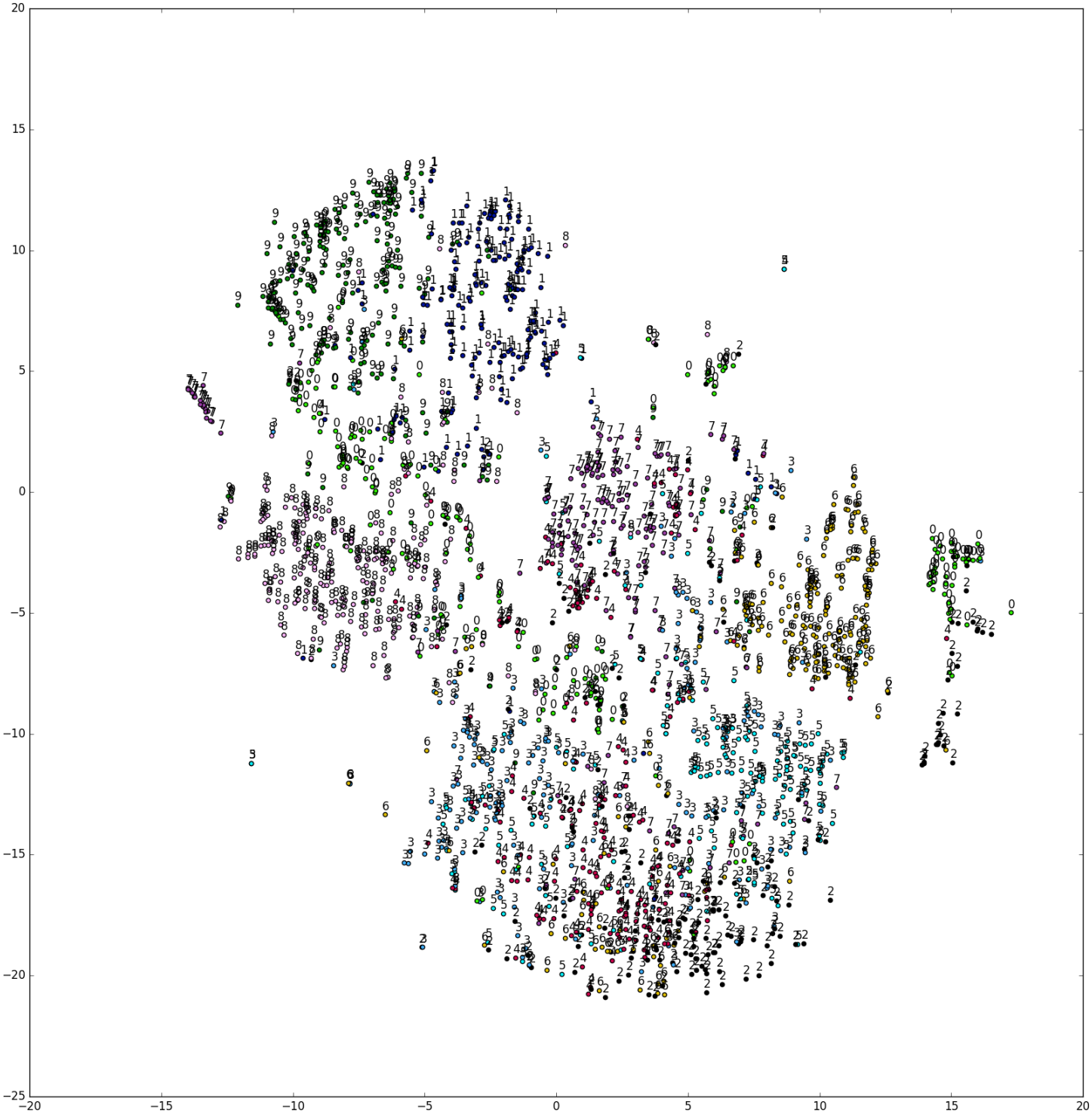


Figure 23. t-SNE on CIFAR10. Final Layer Representation

Key	Class
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

Table 5. Key for Figure 22 and 23

7.7 LIME

LIME visualisation, which was implemented on the Inceptionv3 network provided some very information rich explanations for the model's predictions. In Figures 24 and 25, the images are ordered: Original image, the top 5 super pixels that are most positive towards the class, the top 5 super pixels with rest of image shown, the top 10 super pixels with the most positive or negative influence on the selected class and super pixels with weight of at least 0.1.

Figure 24 shows the explanations for three separate images, showing the areas of the image that impact positively and negatively on the prediction. This provides a lot of information, about the features that the model uses to decide on the class. For the cat, you can see that the area that has the biggest impact is the cat's eye, which makes sense as the cats pupil is a different shape to other animals. Also, shown to have an impact is the cat's whiskers and fur colour.

Similarly, for the second set of images in Figure 24, the dogs ears, nose and paw impact most heavily on the prediction and for the third set, the goldfish's tail, fins and open mouth are what influences the decision. These are all the features that a human would pick out as unique for those animals, showing that convolutional networks do learn the same features as humans do, even if they might not function in the same way.

Most interesting is when comparing the LIME for two similar classes on one image, in Figure 25 this has been done on an image of an African Elephant. The top row shows the LIME

for the African Elephant class, the bottom shows the LIME for the Indian Elephant class. These two animals are very similar with only a couple differences such as the African elephant having bigger ears and longer tusks. For the second LIME, when trying to get the explanation as to why it didn't pick the Indian Elephant class, it clearly picks out the larger ears. Showing that convolutional networks can not only combine features to make up a more abstract feature like an Elephant, but that the presence of some features can affect the prediction negatively as well.

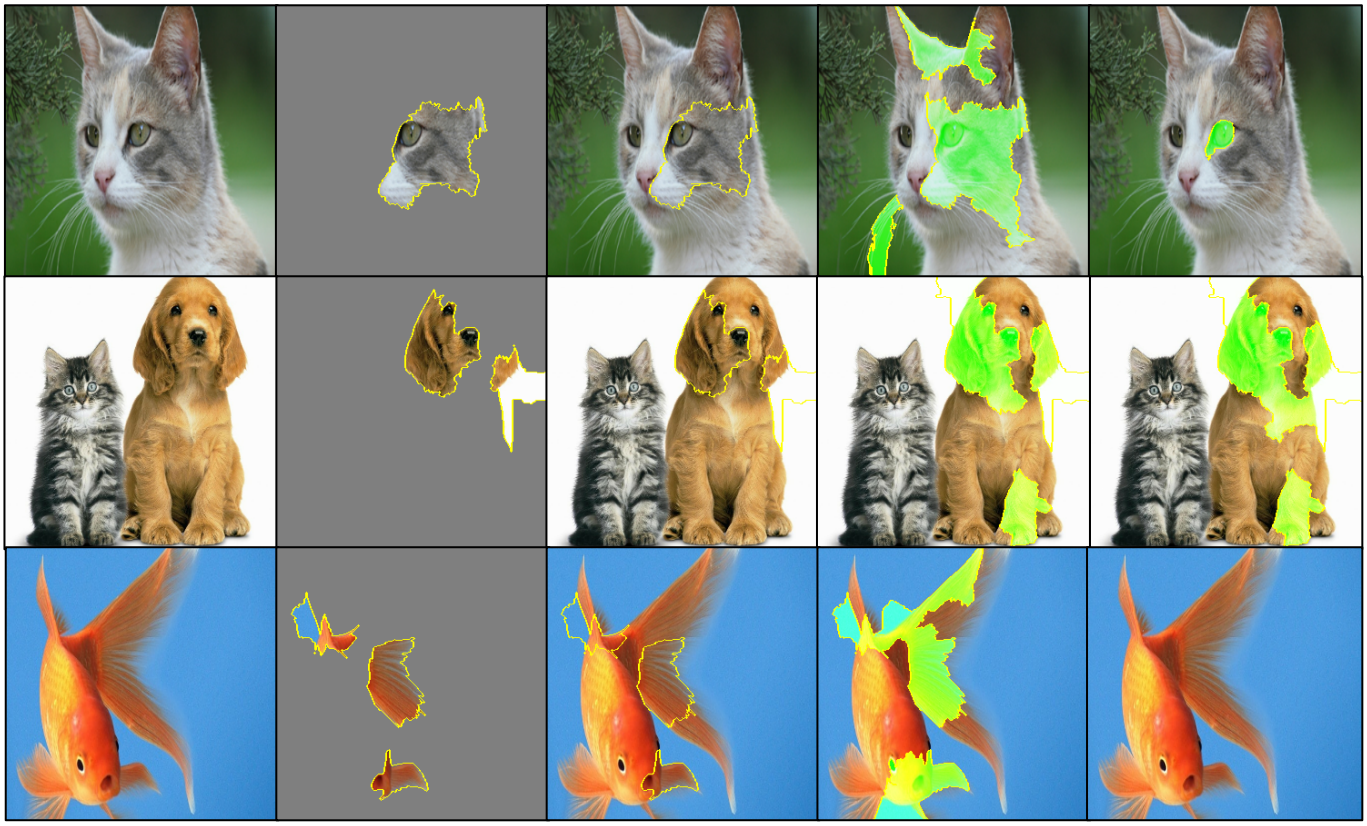


Figure 24. LIME for three images.
(Sources: http://www.thedailymash.co.uk/images/stories/cat2_425.jpg, https://blogs.thegospelcoalition.org/kevindeyoung/files/2014/08/Cat_and_Dog-1024x899.jpg, http://i.telegraph.co.uk/multimedia/archive/02779/goldfish_2779563c.jpg)

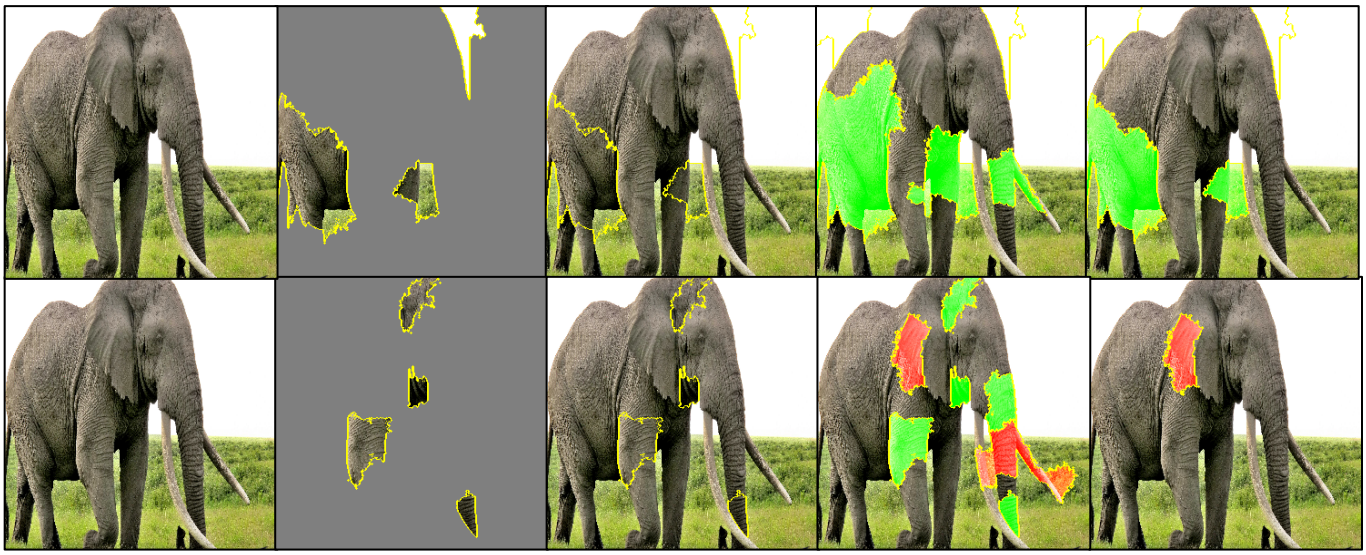


Figure 25. African Elephant LIME (Source: http://www.shunvmall.com/data/out/74/47832472-elephant-image.jpg)

8 Conclusion

The four techniques that I have researched, analysed and implemented have all been successful in providing a lot of information about how convolutional deep learning architectures function. Neuron visualisation lets you visualise what a neuron maximally activates for, creating some very interesting images, especially on the VGG16 network. Showing in detail the hierarchical structure of the network, how simple features like edges and colours are combined into more complicated grids and textures, which are further combined into increasingly abstract features such as eyes and dogs. Using this technique would make it easier to check your network for problems, to see if the features that have been learnt are logical and clear, undertraining should be visible in the features learnt, they would be unclear or contain random noise.

Activation visualisation is useful in showing how an image is passed through the network, the filters that are applied to it and the features that specialised neurons pick up on more than others, and it can demonstrate the amount of dimensionality reduction that the images input to the network are subject to. However, when compared with the other techniques it doesn't succeed at extracting knowledge from the network, it performs better as a learning aid, demonstrating how a convolutional network works. The technique is very generalizable, meaning its relatively easy to apply it to different networks and architectures.

t-SNE performed excellently on the data it was supplied, with very little fine tuning and no pre-processing it can be fed the intermediate, high-dimensional, layer output of a network and convert it into a comprehensible 2-dimensional graph. These graphs show to the extent that the network has separated the classes by the separation of the clusters and which classes it believes to be similar. Which could provide information that would allow you to improve the training process.

LIME is the most useful technique at explaining a model's predictions and it provides the most information about the features of an image that the network has used to classify an image. It's a prebuilt tool so its functionality is limited to what has been implemented by the team, however I can't see any possible improvements to it.

Overall I think that the project has succeed in its goal of implementing and analysing strategies for extracting knowledge from convolutional deep learning architectures though there is still a lot of work that could be done in this area. In my project objectives, I had ‘perform a quantitative and a qualitative assessment of the technologies across architectures’, which turned out to be only qualitative due to the nature of the results. Apart from this all the other goals were achieved, datasets were collected, various architectures were implemented, I would have preferred to train all the networks myself but this would not have been possible with some of the larger ones due to a lack of time and computational power.

8.1 Further Work

I believe that there is a vast amount of work that can be done in this field, both to advance these and other techniques and to develop new ones. Foremost I would like to create a comprehensive tool that can be used on any architecture of convolutional network, featuring the techniques that I have covered here as well as others. Which would allow data scientist to analyse their network using more than just the accuracy.

Further work on the techniques I have implemented is also necessary, tuning the neuron visualisation technique to better display the later layers. While adding more functionality such as displaying the images which maximally activate each neuron or class visualisation, using the same method of gradient ascent to find an image which the network believes to be the perfect representation of a class.

All these techniques could be implemented on more networks to observe the results but I don’t believe that this would provide any more information than it already has done. I would also like to analyse some of the intermediate layers of the VGG16 network, to see what the patters that it is searching for look like in images as currently they seem very abstract.

References

[1] A. Krizhevsky, I. Sutskever and E. G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25,* 2012.

[2] A. Shrikumar, P. Greenside, A. Shcherbina and A. Kundaje, "Not Just a Black Box: Learning Important Features Through Propagating Activation Differences," eprint arXiv:1605.01713, 2016.

[3] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng and M. Chen, "Medical image classification with convolutional neural network," *International Conference on Control Automation Robotics & Vision,* 2014.

[4] Python Software Foundation, "Python Programming Lanuguage," 2017. [Online]. Available: https://www.python.org/.

[5] F. Chollet, *Keras,* 2017.

[6] Nvidia, "cuDNN," 2017. [Online]. Available: https://developer.nvidia.com/cudnn.

[7] The MathWorks, Inc., 2017. [Online]. Available: https://matplotlib.org/.

[8] NumPy Developers, "NumPy," 2017. [Online]. Available: http://www.numpy.org/.

[9] A. Ng, Interviewee, *Interview with Andrew Ng.* [Interview]. 18 February 2017.

[10] M. Veloso, “What is Machine Learning?,” 2015. [Online]. Available: https://www.ml.cmu.edu/.

[11] IBM, “What is big data?,” 2017. [Online]. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[12] M. Caudill, “ Neural networks primer, part I,” *AI Expert,* vol. 2, no. 12, pp. 46-52, December 1987.

[13] A. Ng, “What data scientists should know about deep learning,” in *ExtractConf 2015*, 2015.

[14] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” *The Joural of Physiology,* pp. 106-154, 1962.

[15] H. Harutyunyan, “Spoken language identification with deep convolutional networks,” October 2015. [Online]. Available: http://yerevann.github.io/2015/10/11/spoken-language-identification-with-deep-convolutional-networks/.

[16] J. Kleesiek, G. Urban, A. Hubert, D. Schwarz, K. Maier-Hein, M. Bendszus and A. Biller, “Deep MRI brain extraction: A 3D convolutional neural network for skull stripping,” *NeuroImage,* vol. 129, p. 460–469, 2016.

[17] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller, “Striving for Simplicity: The All Convolutional Net,” 2014.

[18] A. Krizhevsky, I. Sutskever, and G. . E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Stanford University, 2015.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[20] Y. LeCun, "Gradient-based learning applied to document recognition," 1998.

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

[22] Stanford Vision Lab, "Results of ILSVRC 2013," 2014. [Online]. Available: http://www.image-net.org/challenges/LSVRC/2013/results.

[23] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

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

[25] C. Szegedy, J. Ibarz and V. Vanhoucke, "Scene Classification with Inception-7," Google, 2016.

[26] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016.

[27] B. Ludwiczuk, "SqueezeNet disappointingly slow," 2016. [Online]. Available: https://groups.google.com/d/msg/torch7/1vK_i920TtU/SbdAFhtZDAAJ.

[28] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," December 2015.

[29] S. Gross and M. Wilber, “Training and investigating Residual Nets,” February 2016. [Online]. Available: http://torch.ch/blog/2016/02/04/resnets.html.

[30] UNC Vision Lab, “ImageNet,” 2016. [Online]. Available: http://image-net.org/challenges/LSVRC/2016/results.

[31] Y. LeCun, C. Cortes and C. J. Burges, “The MNIST database,” 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[32] V. Romanuke, “Parallel Computing Center (Khmelnitskiy, Ukraine) represents an ensemble of 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate.,” November 2016. [Online]. Available: https://drive.google.com/file/d/0B1WkCFOvGHDddElkdkl6bzRLRE0/view?usp=sharing.

[33] “The CIFAR datasets,” 2017. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[34] B. Graham, “Fractional Max-Pooling,” 2014.

[35] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, “Microsoft COCO: Common Objects in Context,” 2014.

[36] The TensorFlow Team, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2016.

[37] The Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” 2016.

[38] "Visualizing parts of Convolutional Neural Networks using Keras and Cats," January 2017. [Online]. Available: https://medium.com/hacker-daily/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59.

[39] K. Simonyan, A. Vedaldi and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," *arXiv:1312.6034,* 2013.

[40] H. Noh, S. Hong and B. Han, "Learning Deconvolution Network for Semantic Segmentation," *ICCV,* 2015.

[41] M. T. Ribeiro, S. Singh and C. Guestrin, ""Why Should I Trust You?" Explaining the Predictions of Any Classifier," 2016.

[42] X. Ren and J. Malik, "Learning a Classification Model for Segmentation," *ICCV,* pp. 326-333, 2003.

[43] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. B. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, L. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature 518,* p. 529–533, 2015.

[44] A. Mahfouza, M. v. d. Giessena, L. v. d. Maatena, S. Huismana, M. Reindersb, M. J. Hawrylyczc and B. P. Lelieveldt, "Visualizing the spatial gene expression organization in the brain through non-linear similarity embeddings," *Methods,* vol. 73, pp. 79-89, 2015.

[45] "Training Deep Neural Networks on ImageNet Using Microsoft R Server and Azure GPU VMs," November 2016. [Online]. Available: https://blogs.technet.microsoft.com/machinelearning/2016/11/15/imagenet-deep-neural-network-training-using-microsoft-r-server-and-azure-gpu-vms/.

[46] D. Erhan, Y. Bengio, A. Courville and P. Vincent, “Visualizing Higher-Layer Features of a Deep Network,” June 2009.

[47] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs and H. Lipson, “Understanding Neural Networks Through Deep Visualization,” *ICML Deep Learning Workshop 2015,* 2015.

[48] F. Chollet, “How convolutional neural networks see the world,” January 2016. [Online]. Available: https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html.

[49] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” 2012.

[50] L. v. d. Maaten, “t-SNE,” 2017. [Online]. Available: https://lvdmaaten.github.io/tsne/.

[51] M. D. McDonnell and T. Vladusich, “Enhanced Image Classification With a Fast-Learning Shallow Convolutional Neural Network,” 2015.

[52] M. Tulio Ribeiro, S. Singh and C. Guestrin, “Why Should I Trust You?": Explaining the Predictions of Any Classifier,” *eprint arXiv:1602.04938 ,* 2016.

[53] B. Graham, “Fractional Max-Pooling,” 2015.