Multiscale image representation in deep learning

by

Jean-Pierre Stander

Submitted in partial fulfillment of the requirements for the degree

Magister Scientiae

In the Department of Statistics

In the Faculty of Natural and Agricultural Sciences

Univeristy of Pretoria

October 2020

I, *Jean-Pierre Stander,* declare that this mini-dissertation (100 credits), which I hereby submit for the degree Magister Scientiae in Advanced Data Analytics at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

Signature:

Date:

Summary

Deep learning is a very popular field of research which can input a variety of data types [1, 16, 30]. It is a subfield of machine learning consisting of mostly neural networks. A challenge which is very commonly met in the training of neural networks, especially when working with images is the vast amount of data required. Because of this various data augmentation techniques have been proposed to create more data at low cost while keeping the labelling of the data accurate [65]. When a model is trained on images these augmentations include rotating, flipping and cropping the images [21]. An added advantage of data augmentation is that it makes the model more robust to rotation and transformation of an object in an image [65].

In this mini-dissertation we investigate the use of the Discrete Pulse Transform [54, 2] decomposition algorithm and its Discrete Pulse Vectors (DPV) [17] as data augmentation for image classification in deep learning. The DPVs is used to extract features from the image. A convolutional neural network is trained on the original and augmented images and a comparison made to a convolutional neural network only trained on the unaugmented images. The purpose of the models implemented is to correctly classify an image as either a cat or dog. The training and testing accuracy of the two approaches are similar. The loss of the model using the proposed data augmentation is improved. When making use of probabilities predicted by the model and determining a custom cut off to classify an image into one of the two classes, the model trained on using the proposed augmentation outperforms the model trained without the proposed data augmentation.

Acknowledgements

Firstly, I want to thank God for giving me the ability to be able to do this research.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

I want to thank my supervisor Dr Inger Fabris-Rotelli for her guidance and help through my Honours research and now with my Masters. Her insight help and motivation were invaluable and I definitely would not have been able to complete this without her.

I also want to thank my parents, Theuns and Dalene Stander for giving me the opportunity to be able to do this research. If it had not been for the incredible opportunity they gave me to go to university I would never have been able to do this research. I am also grateful that I could live with them in the lock down period where they supported me allowing me to work on and complete my research.

Lastly, I want to thank all my friends, who over the past two years supported me and understood when I had to work and missed some fun times. I especially want to thank René Kirsten who has supported me and motivated me when I most needed it.

Contents

List of Figures

Chapter 1

Introduction

In the modern age, the availability of data has increased. Data formats range from simple observed measurements to more complex formats such as images and sound. The analysis of data is important for various reasons, but importantly can be used to make predictions of unseen data by training an applicable model.

A popular technique which inputs a variety of data types is deep learning [1, 16, 30]. Being able to correctly classify images based on what they contain is useful for the automation of hand driven tasks. For example, if a model can be trained to correctly classify an apple as rotten or healthy, the process of removing the rotten apples from a batch can be automated. The model will have a look at each apple and decide whether it is rotten or healthy. This model can then tell a machine which apples to remove to ensure no rotten apples continue to the next step of production.

In this mini-dissertation it is investigated whether the use of a decomposition algorithm to represent an image at multiple scales can be useful to aid in the training process of a deep learning model. The Discrete Pulse Transform algorithm is used to decompose images into multiple scales. These images, then represented at multiple scales, are used as augmented data for training a neural network. This model is compared to a model only trained on the unaugmented images to see if any improvement is observed.

1.1 Deep learning

Deep learning is a field of research which has been enjoying notably more attention over recent years. Neural networks were first proposed in 1944 by Warren McCullough and Walter Pitts [24] who developed the Threshold Logic Unit (also called the McCullough-Pitts neuron) which takes a weighted sum as input

and provides a single value as output. This value is then compared to some threshold value and passed through a transfer function [24]. The output from this neuron and other neurons can be used as input for the next neuron forming a perceptron more commonly called a neural network [46]. Artificial neural networks can also be traced back to Frank Rosenblatt in 1958 who did a lot of work in perceptrons [14]. These perceptrons were used in the Mark I Perceptron which is a pattern recognition device and was one of the first devices which could learn on a trial and error basis. These models form a very powerful subset of machine learning which is capable of handling a variety of data types [29]. These models can do anything from regression and clustering up to determining if and where an object is in an image [39, 73].

There are a range of different types of neural networks which make up the field of deep learning. In Figure 1.1 a basic artificial neural network is shown. This model has three layers - one input, one hidden and one output layer. Here X_i, A_i and Y_i represent the values of the nodes, w_{ij} represents the weights which connect the relevant nodes from adjacent layers and b_i represents the bias of each layer. This model takes inputs X_1, X_2 and X_3. The first element in the hidden layer is calculated as a weighted sum of the inputs with the relevant weights and then adding the layer's bias. To calculate A_1 the weights w_{11}, w_{12} and w_{13} are used. The output layer's values are again calculated as a weighted sum of the four elements in the hidden layer using the relevant weights and then adding this layer's bias.

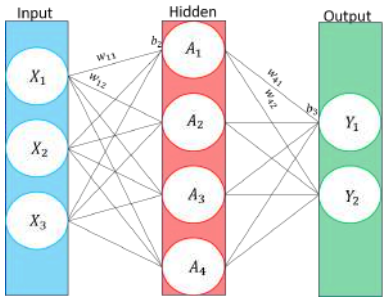


Figure 1.1: A basic artificial neural network with three layers (one input, one hidden and one output). The input layer has three nodes (X_1, X_2, X_3), the hidden layer four nodes (A_1, A_2, A_3, A_4) and the output layer two nodes (Y_1 and Y_2). The nodes in each layer is connected to the adjacent layers via weights $w_{11}, \ldots, w_{52}$. The hidden layer has bias b_2 and the output layer has bias b_3

In Figure 1.1 the input for A_1 and Y_1 are calculated as,

$$A_1^{in} = \sum_{i=1}^{3} w_{i1} X_i + b_1$$
$$Y_1^{in} = \sum_{i=1}^{4} w_{i4} A_i + b_2$$

Each layer has a pre-specified number of nodes (also called neurons). The input layer has a node for each

of the input elements in a single observation. The output layer differs depending on what the response variable is. If the response is continuous a single output node is required. This node can return any real number as the estimate. If the response variable is categorical the output layer will have a node for each category. The output value for each node is bounded between zero and one and can therefore be interpreted as the probability of the observation falling into the category represented by each node. The number of nodes in each hidden layer is chosen by the user. Each of the nodes in a layer is connected with weights to each node in the neighbouring layers. To move from one layer to the next a weighted sum of a layer's output is determined and passed to the next layer.

Since there is no bound on the value of the weights, the range of the value of the input of a layer is unbounded. We want to map these unbounded values onto a range which is known, for example $[0, 1]$. This will cause the node to 'light up' if the value is high enough, which can indicate an important part of the data or an important connection. To do this some activation function, f_1 is applied to the value which is the input of the layer. If a function such as Sigmoid is used (Figure 1.2b), values larger than zero will have an activation value of greater than 0.5 which can be seen as a lit-up node.

If it is not required for the node be lit up if it is greater than zero but rather when it is greater than some value b_1, adding a value called the bias before passing this value to the activation function can be imposed. Each of the hidden and output layers have a bias which is added to the values inputted to the layer before the activation function is applied.

Some examples of activation function are plotted in Figure 1.2 where the x-axis values are those passed to the function from the previous layer and the y-axis is what is returned by the functions. The hidden layers of the neural network can make use of any of these activations. The only limitation on the type of activation function used is in the output layer of a neural network with more than one node. In such a case an activation function such as Sigmoid (Figure 1.2b) which returns a value between zero and one should be used. The reason for this is the fact that this returned value can then be seen as the probability that the observation should be classified into the class represented by the node.

The four activation functions shown in Figure 1.2 are some of the most commonly used activation functions. All four activation functions can input any real valued number. The output range for ReLu (Figure 1.2a) is $[0, \infty)$. The ReLu is easy to work with mathematically since it is a simple function ($f(x) = \max\{0, x\}$). But ReLu outputs a zero for values inputted smaller than zero. This can be problematic since it ignores negative features [71]. The Sigmoid (Figure 1.2b) has an output range of $[0, 1]$ which can be very useful in the last layer if we have more than one output node since this value can be seen as a probability of the observation being associated with that node. A problem of the Sigmoid function is the fact that its gradient is very small for very large or small input values which can cause problems with the optimisation [67]. Hyperbolic tan (Figure 1.2c) is the same as the Sigmoid only differing in the fact that it outputs

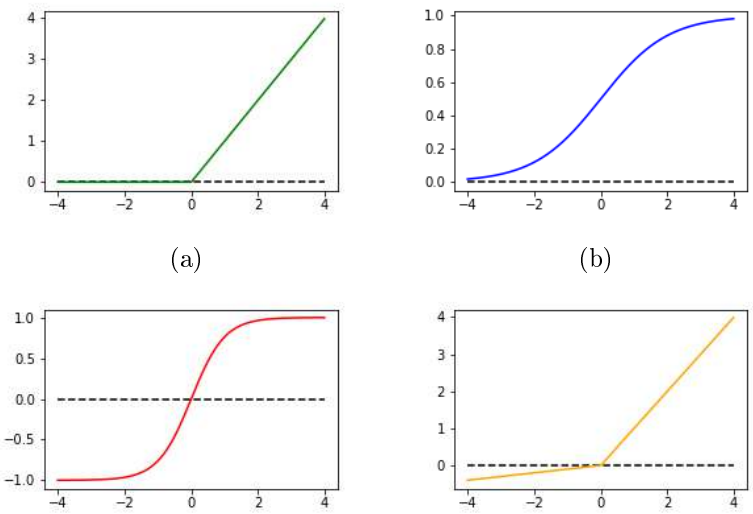


Figure 1.2: Examples of activation function [58]. a) Rectified linear unit (ReLu). b) Sigmoid. c) Hyperbolic tan. d) Leaky ReLu ($\alpha = 0.2$).

a value in the range $[-1, 1]$. Hyperbolic tan also has the problem of a very small gradient for very large and small inputs. Lastly, leaky ReLu (Figure 1.2d) has an output range of $(-\infty, \infty)$ [71]. Leaky ReLu is defined as $f(x) = \max\{\alpha x, x\}$, $\alpha \in [0, 1]$. If α is chosen as zero it is the standard ReLu and if α is chosen as one it returns exactly what was inputted. Leaky ReLu does not ignore negative features and has a non-zero gradient for its entire range.

The number of parameters in a neural network can be calculated by determining the number of weights and biases required. The number of weights between two layers is calculated as the number of nodes in the first layer times the number of nodes in the second layer, since each node in the first layer is connected to each node in the second layer. The total number of biases is the number of layers minus one since each layer except the input layer has a bias term. The number of parameters in the neural network in Figure 1.1 is thus calculated as $3 \times 4 + 4 \times 2 = 20$ weights, 2 biases resulting in a total number of parameters of 22. Since this is a very small and simple neural network, it is easy to see that the number of parameters to optimise will explode with a larger and more complex model.

The optimal estimated values for the weights and biases are required for the model to perform i.e. be trained. Due to the large number of parameters and the complexity of the data usually involved, the error surface is highly complex with possibly many local minimums [9]. For optimal performance, the parameters should be selected such that the error surface is at the global minimum. The method most commonly used for optimising the parameters of the neural network is stochastic gradient descent [9]. This is an iterative optimisation algorithm which updates the parameters of the model using an appropriate

loss function along with the estimated and actual output value. The update step of this optimisation is done as follows,

$$\boldsymbol{\beta_i} = \boldsymbol{\beta_{i-1}} - r * \nabla(\boldsymbol{X}, \boldsymbol{\beta_{i-1}}) \tag{1.1}$$

where $\boldsymbol{\beta_i}$ are the parameter values at step i, r is the learning rate and $\nabla(\boldsymbol{X}, \boldsymbol{\beta_{i-1}})$ the derivative of the loss function with respect the parameters at step $i-1$.

This algorithm is similar to the Newton-Raphson algorithm but the Hessian matrix is replaced by a constant r which is known as the learning rate. The learning rate determines the step sizes of the optimisation steps. This value needs to be selected carefully. If the learning rate is too large we risk the chance of the optimisation process stepping over the global minimum and converging to a local minimum. If the learning rate is chosen too small the optimisation step will be very small which will cause the optimisation to be time consuming. Much research has gone into creating algorithms with an adaptive learning rate to improve the performance and training time of the optimisation process. Some of these algorithms are AdaGrad [15], RMSprop [6] and Adam [55].

To train all these parameters a vast amount of labelled data is required. This can be a tedious task. For example, if we want to train a neural network to classify images of dogs and cats correctly we need to obtain 1000s of images of cats and dogs and label the images to be able to train our network.

Various augmentations such as cropping, rotating and flipping of images can be useful to create more training data [21, 65]. These augmentations are done stochastically in the sense of choosing an augmentation as well as the parameters of the augmentations, for example the angle of the rotation of an image [59]. These augmentations also help a neural network to be more robust to the angle and transformations of the object in an image [69]. Augmentations are inexpensive methods of creating extra observations while keeping the labelling accurate. It is much faster to apply these augmentations to an image than to obtain and label a new image.

A more extreme data augmentation is the use of generative adversarial networks (GANs) [68]. This is a rather complex process which involves two neural networks, see Figure 1.3. The first generative neural network uses some random noise to generate an image with the aim that this image should be similar to the training set. The generated and actual images are then passed to the second discriminating neural network. The discriminating network has to determine which images are generated and which are real images. The parameters of both these networks are updated in the training. Once the discriminating network does not discriminate well between the generated and real images the training is done. The generative network can generate images similar but different to the images in the training set [65].

The neural network most commonly used for image recognition is the convolutional neural network (CNN)

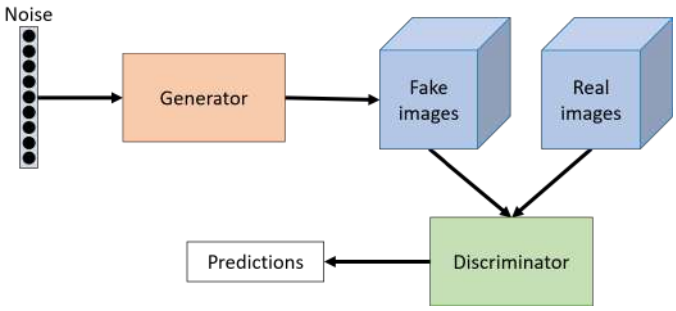


Figure 1.3: A representation of a generative adversarial network.

[41]. The reason this network works well with image recognition is because in the convolutional layer, it scans through the image with different size filters. These layers learn certain features present in the image which is important for the task at hand. The reason the CNN works well with images is that it learns these features by itself and it is not necessary for any human input as to which features are important [7].

1.2 Multiscale deep learning

The concept of multiscale image analysis is covered in detail by Lindeberg [44]. Image content covers a variety of scales, thus harnessing this information results in improved image understanding [11, 43].

Multiscale image analysis is a field of study where images are represented at different scales in order to attempt to extract feature from these images. Multiscale wavelet representation was used in [34] to enhance the features in digital mammograms. It was shown that this technique makes unseen or barely visible features more obvious without requiring additional radiation.

When working with images a fundamental step is to represent the image in a such a way that it can be used on a computer. Good image representation leads to good image analysis results. A common type of image representation is to divide the image into a range of smaller areas called pixels and then representing each pixel with one or more values. The image will then be represented by a matrix of values. In a greyscale image each pixel has one value between 0 and 255 whereas in a RGB image (colour image) each pixel has three values between 0 and 255, one value of the red spectrum, one value on the green spectrum and one value on the blue spectrum. The combination of RGB in each pixel forms a unique colour [49].

Objects in images are much more than a few pixels with a specific value. Objects are formed by lines and shapes within the image. Being able to detect these lines and shapes accurately assists in processing images.

A common problem encountered in image recognition is to detect objects of different sizes in an image [51]. Some research has gone into solving this problem. One possible solution is to use a region-based convolutional neural network (R-CNN) [22]. This model uses an algorithm such as selective search to find proposal regions which can potentially contain an object [64]. The proposal regions obtained vary in size and it is possible that many regions can represent different parts of the same object, so a hierarchical grouping algorithm is used to group regions which are very similar. These regions are then padded or resized and passed to a CNN which extract features. The extracted features (which is the output of the CNN) and labels are then used to train a classifier model such as a support vector machine to classify the detected objects into one of many possible classes. This model has been proven to work well when coming across the problem of different size objects in images [22, 23]. The optimisation of these models can be cumbersome because each component needs to be trained individually [51].

Another technique to possibly solve this problem is a you only look once (YOLO) neural network [51]. In these models the classification and detection are done as a single regression model. With the use of a single CNN multiple bounding boxes and their class probabilities are predicted. It is evident that this model will be faster and easier to optimise than a model such as R-CNN. Another advantage of YOLO is that it sees the entire image when deciding on bounding boxes and class probabilities. So instead of only looking at a local segment of the image it looks at the global setting. This can help to better classify objects, for example if an object is detected and the background is water, the object is more likely to be a boat than a house. Again, the problem of detecting small objects was encountered and so YOLOv2 and later on YOLOv3 was developed [52, 53]. Firstly, the model is trained on different size images, ranging from a resolution 320×320 to 608×608. This makes the model more robust towards image size. Secondly the model predicts boxes that may contain objects at three different scales. This is done using a similar concept to feature pyramid network [42]. In this method, a pyramid like structure of features is generated from the images at different scales (see Figure 1.4). The method works on a bottom-up and top-down pathway. The first steps extract features usually with the use of a convolutional models, containing multiple convolutional layers, creating multiple feature maps. The higher up the features map is in the pyramid the lower the resolution is, therefore the spatial resolution is reconstructed using upsampling.

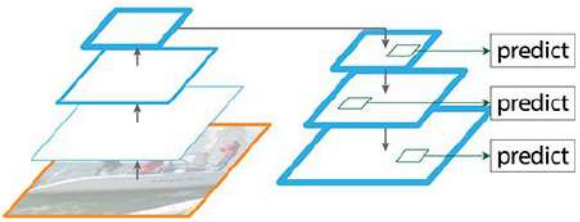


Figure 1.4: The process of extracting the pyramid feature maps at different scales and reconstructing the spatial resolution [42].

These models make use of images with objects of different sizes as well as images of different sizes as a multiscale image learning to improve the performance of a model. As far as the authors can determine, there is no work done on representing the same image at different scales using some decomposition in aiding the training of models. This will be investigated in this mini-dissertation.

1.3 Deep learning for texture analysis

Texture in images can be seen as a repetition of a visual pattern over a part of the whole image [48]. Texture can be artificial or natural (see Figure 1.5). When working with images, natural texture is more common. Examples of natural texture are water surface[1], a brick wall[2] and a grass field[3] as seen in Figure 1.5. Artificial textures form a very clear and perfect pattern. For example the texture in Figure 1.5a is a black line followed by a white line. This pattern will repeat perfectly. Natural textures also form a pattern but not as perfectly as with artificial texture. For example, the texture of a brick wall in Figure 1.5d is rectangular shapes divided by thick lines but the rectangular shapes are not the same size throughout and the dividing line thickness varies.

In human vision, texture is a meaningful aspect as it is used to distinguish between objects [48]. Therefore, it would make sense that we can use texture analysis to classify or segment images in computer vision [45].

Texture analysis is used in remote sensing [8]. For example a farmer wanting to monitor crops using remote sensing imagery, can use texture analysis to detect the parts in the image which contain the crops.

Texture analysis has been used to classify the quality of produced steel [8] and medical diagnoses of skin diseases [48]. In [28], texture analysis was used to detect collapsed buildings after an earthquake which can help emergency personnel to more productively help people that were potentially harmed in such an event. The first method calculated six different texture features of the building before and after the earthquake. A random forest classifier, using these six texture features calculated before and after the earthquake as input was trained to predict which buildings collapsed and which did not. The second method trained a CNN to predict which buildings collapsed and which did not. The features learned by the convolutional layers of the CNN are extracted. These extracted features before and after the earthquake are used as input for a random forest classifier. Although the CNN-RF method (87.6% accuracy) outperformed the Texture-RF method (83.4% accuracy) the texture-based method still performed well. This supports the fact that texture plays a very important role in image classification.

[1] Obtained from https://www.peakpx.com/452514/water-grayscale-photo on 1 April 2020
[2] Obtained from https://www.pickpik.com/brick-wall-old-dark-brick-wall-brick-wall-background-39130 on 1 April 2020
[3] Obtained from https://www.pxfuel.com/en/free-photo-xvxiy on 1 April 2020

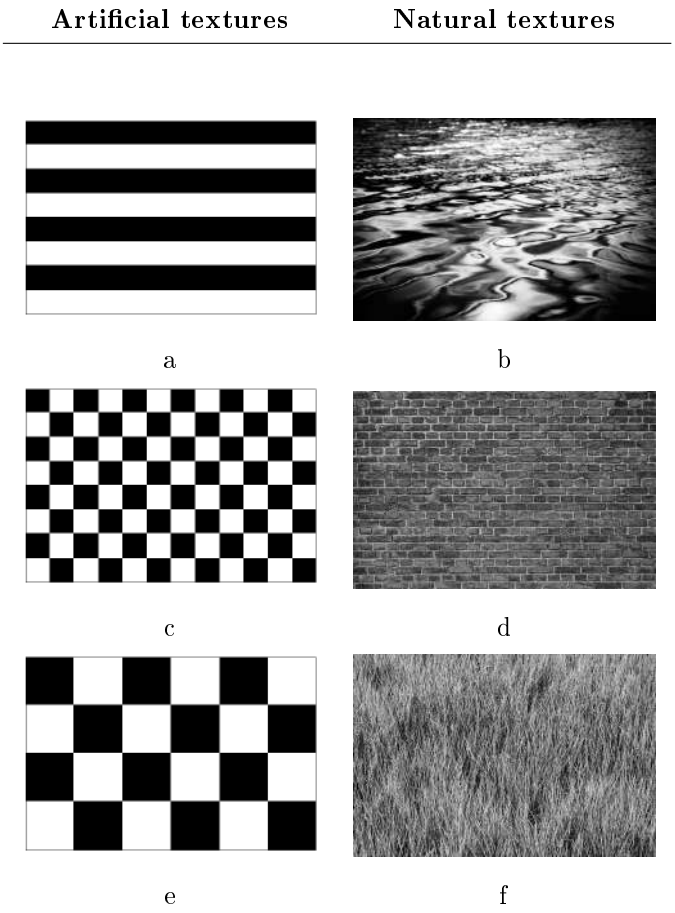


Figure 1.5: Examples of artificial and natural texture. a, c and e) Artificial textures. b) Water surface. d) Brick wall. f) Grass fields.

Deep learning has also been used for texture analysis more specifically for texture recognition, segmentation and description [12]. A problem when working with texture in deep learning is the curse of dimensionality. Consider the texture database Brodatz[4]. This database contains images of different types of textures such as a brick wall and tree bark. The dimensionality of a texture database such as Brodatz is much higher than an object recognition dataset such as MNIST [5]. A possible solution to this can be to extract features from the texture and use this to train a neural network [5].

In the rest of this mini-dissertation the basic knowledge required will be covered. The basic knowledge for deep learning and the Discrete Pulse Transform decomposition algorithm will be covered in Chapter 2. This decomposition will be used to represent images at different scales and assess whether this can help to extract/highlight features in the image. In Chapter 4 we investigate whether using these images represented at different scales can aid in the training process of deep learning model and more specifically in an image classifying convolutional neural network as a method of data augmentation. The deep learning

[4]Database available at http://sipi.usc.edu/database/database.php?volume=textures.

model is trained using a subset of all the available data to see if this DPT decomposition of an image can be a useful data augmentation method.

The objectives aimed to be met in this mini-dissertation are as follows,

- Use of the DPT decomposition to represent images at different scales to extract/highlight features in the image.
- Investigate whether the images which extract these features are useful in improving training of a deep learning model.
- Investigate whether the DPT and its DPVs can be a useful data augmentation method in the case of limited data.

Chapter 2

Background theory

A digital signal can be seen as a sequence of discrete observations of a continuous process [60]. A simple example of this is the hourly electricity usage in a town. The electricity usage is a continuous process but is observed at discrete time points, namely every hour. A digital image is also an example of a digital signal. The true colour are observed from a continuous plane which contains an infinite number of colours and when we take a picture and convert it to a digital image we observe a discrete value for the red (R), green (G) and blue (B) bands for every pixel. Each of the RGB elements can have an integer value between 0 and 255 and therefore there are $256^3 = 16777216$ possible colours which is a finite discrete set.

Digital signal processing includes processing a wide variety of signal types such as audio, visual and many more. Processing these signals is very important, for example if the hourly electricity usage can be predicted it can help the electricity producers with their planning with regards to generation of electricity. The processing of digital images can also be useful in computer vision for tasks such as object detection.

2.1 LULU

The LULU smoothers are non-linear, fully trend preserving smoothers developed in 1-dimension by Rohwer in 1989 [26]. The LULU smoothers consist of two operators namely the L_n operator and the U_n operator. These operators are repeated application of minimum and maximum operators on a subset of size n on the digital signal. First a connection has to be defined.

Definition 1. *Connectivity [57, 19]*

Let B be a non-empty set. A subset $\mathcal{C}$ of B is a connection if:

1. $\emptyset \in \mathcal{C}$,

2. $\{x\} \in \mathcal{C}$ *for each* $x \in B$ *and*

3. *for any family* $\{C_i\} \subseteq \mathcal{C}$ *we have*

$$\bigcap_{i \in I} C_i \neq \emptyset \implies \bigcup_{i \in I} C_i \in \mathcal{C}$$

Then any $C_i \in \mathcal{C}$ *is called a connected set.*

Consider $f(x)$ a digital image signal defined on $\mathcal{A}(\mathbb{Z}^d)$, the domain where the signal is defined (where $\mathcal{A}(\cdot)$ is a vector lattice). The set of all connected sets of size $n+1$ which includes the point x, for any given $x \in \mathbb{Z}^d$ and $n \in \mathbb{N}$, is given by $\mathcal{N}_n(x) = \{V \in \mathcal{C} : x \in V,\ \text{card}(V) = n+1\}$, where $\text{card}(\cdot)$ is the cardinality of a set, namely a measure of the number of elements in the set. The LULU operators [2] are defined for $x \in \mathbb{Z}^d$ and $f(\cdot) \in \mathcal{A}(\mathbb{Z}^d)$ as,

$$(L_n f)(x) = \max_{V \in \mathcal{N}_n(x)} \min_{y \in V} f(y), \tag{2.1}$$

$$(U_n f)(x) = \min_{V \in \mathcal{N}_n(x)} \max_{y \in V} f(y). \tag{2.2}$$

This definition of the LULU operators is defined in d-dimensions.

A more intuitive explanation follows. Let us consider a 1-dimensional signal $x \in \mathbb{Z}$. We assume that the signal is connected to an infinite signal of zero.

L_n operator on $f(x) \in \mathcal{A}(\mathbb{Z})$

Apply L_n to the signal $f(x) = \{f(x_1), \ldots, f(x_N)\}$. For each element $f(x_j)$ in the signal.

- Create $n+1$ subsequences $s_1, \ldots, s_{n+1}$ of size $n+1$.
- Each s_i should consist of $n+1$ consecutive values from the signal and each should contain $f(x_j)$ i.e. $f(x_j) \in s_i$ for $i = 1, \ldots, n+1$, for example if $n = 2$ the subsequences of $f(x_1)$ will be $s_1 = \{0, f(x_1)\}$ and $s_2 = \{f(x_1), f(x_2)\}$.
- Obtain the minimum of each subsequence, $\min(s_1), \ldots, \min(s_{n+1})$.
- Obtain the maximum of the minimums, $\max\{\min(s_1), \ldots, \min(s_{n+1})\}$

For example consider the signal $\{f(x_1), \ldots, f(x_{j-1}), f(x_j), f(x_{j+1}), \ldots, f(x_N)\}$. To apply the L_2 operator to $f(x_j)$ we construct three subsequences. The three subsequences will be $s_1 = \{f(x_{j-2}), f(x_{j-1}), f(x_j)\}$, $s_2 = \{f(x_{j-1}), f(x_j), f(x_{j+1})\}$ and $s_3 = \{f(x_j), f(x_{j+1}), f(x_{j+2})\}$. Then the smoothed value is obtained as the maximum of the minimums of these subsequences, $(L_2 f)(x_j) = \max\{\min(s_1), \min(s_2), \min(s_3)\}$.

The U_n operator is defined exactly the same as the L_n but we take the minimum of all the maximums of the subsequences. The L_n operator smooths out all the bumps of size n and smaller in the signal and the U_n operators smooths out the pits of size n and smaller. If n is chosen to be two, only bumps and pits of size two and smaller would be smoothed out and larger bumps and pits not.

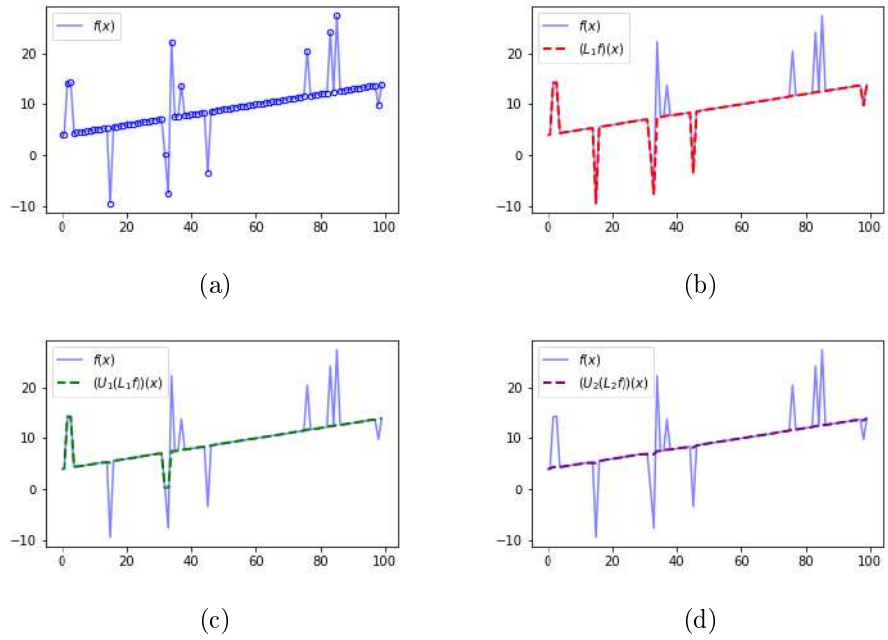
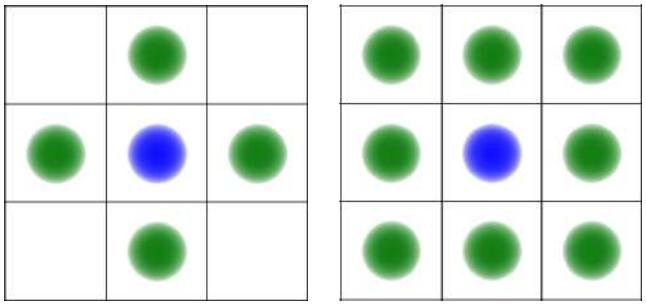


Figure 2.1: Example or LULU operators applied to a signal. a) The original signal $f(x)$. b) $(L_1 f)(x)$ the smoothed signal after the bumps of size one are removed. c) $(U_1(L_1 f))(x)$ the smoothed signal after the bumps and pits of size one are removed. d) $(U_2(L_2 f))(x)$ the smoothed signal after bumps and pits of size two and smaller are removed.

In Figure 2.1a, a signal $f(x) = \{4, 4.1, 14.2, 14.3, 4.4, \dots\}$ is plotted. In Figure 2.1b, L_1 is applied to the signal and we see that only the bumps of size one are smoothed out, bumps of size two and larger are not altered. For the first of the n elements two subsequences of size two are created, $s_1 = \{0, f(x_1)\} = \{0, 4\}$ and $s_2 = \{f(x_1), f(x_2)\} = \{4, 4.1\}$. The smoothed value for $f(x_1)$ is then $\max\{\min(s_1), \min(s_2)\} = \max\{0, 4\} = 4$, this is repeated for the other $n - 1$ elements. In Figure 2.1c, L_1 and U_1 is applied to the signal. We see that both bumps and pits of size one are smoothed out. After L_1 is applied to all the elements as explained above, U_1 is applied to all the elements. Again for the first of the n elements, which was already smoothed with L_1, the two subsequences of size two are $s_1 = \{0, f(x_1)\} = \{0, 4\}$ and $s_2 = \{f(x_1), f(x_2)\} = \{4, 4.1\}$. The smoothed value for $f(x_1)$ is then $\min\{\max(s_1), \max(s_2)\} = \min\{4, 4.1\} = 4$, which is again repeated for the other $n - 1$ elements. In Figure 2.1d L_2 and U_2 are applied to the signal and we see that all bumps and pits of size two and smaller are smoothed out.

The LULU operators have been extended to multidimensional signals [19]. In the 1-dimensional case we had a look at the observation of interest's n neighbours. This included n observations to the left and

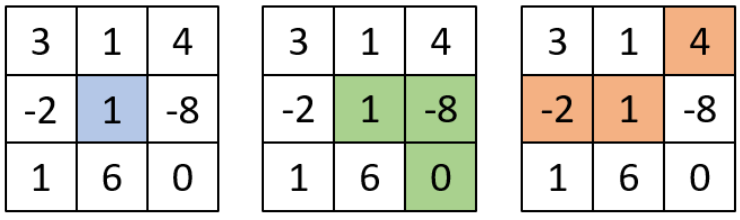
right of the observation of interest. When we have a 2-dimensional case there are not only observations to the left and right of the observation of interest but also above and below the observation. To be able to apply LULU to higher dimensions, a connected region is required as defined in Definition 1. Examples of different neighbourhoods are shown in Figure 2.2 which are special cases of Definition 1. The observation marked in blue is the observation of interest and the observations marked green are all the observations considered as its neighbours.



4-Connectivity 8-Connectivity

Figure 2.2: Graphical representation of 4- and 8-connectivity for the centre blue elements.

We see that with 4-connectivity each element has four possible neighbours and with 8-connectivity each element has eight possible neighbours. A connected region in $N_n(x)$ is a collection of elements where each element in the region is the neighbour of at least one other element in the region. For example, if we have a 2-dimensional array as shown in Figure 2.3a and we want to find connected sets of size two for the centre element, marked blue. If we use 4-connectivity a possible connected region as shown by the green region in Figure 2.3b. If we were to use 8-connectivity a possible connected region is shown by the red region in Figure 2.3c. It is important to note that not all the elements of a connected region does necessarily have the same value.



(a) (b) (c)

Figure 2.3: Examples of 4- and 8-connectivity connected regions. a) The element of interest who's connected region of size three needs to be determined. b) A possible connected region of size three of the centre element when 4-connectivity is used. Therefore, this is an element of $\mathcal{N}_3(x)$ when using 4-connectivity. c) A possible connected region of size three of the centre element when 8-connectivity is used therefore this is an element of $\mathcal{N}_3(x)$ when using 8-connectivity.

To be able to apply the multidimensional LULU operators we need to define a local minimum and maximum set.

Definition 2. *Local minimum and maximum set [19]*

Let $V \in \mathcal{C}$ be a connected set. A point x is adjacent to V if $x \notin V$ and $V \cup \{x\} \in \mathcal{C}$. The set of all points adjacent to V is denoted by $adj(V)$ i.e.

$$adj(V) = \{x : x \notin V, V \cup \{x\} \in \mathcal{C}\}.$$

Then a connected set V is called a local maximum set of f if

$$\sup_{y \in adj(V)} f(y) < \inf_{x \in V} f(x).$$

set V is a local minimum set of f if

$$\inf_{y \in adj(V)} f(y) < \sup_{x \in V} f(x).$$

Therefore, a local minimum (maximum) set of size n is defined as a connected region of size n where all the points in the region have values less (greater) than all their neighbours that are not contained in the connected region, that is all the elements adjacent to the region.

In Figure 2.4 the four possible connected regions of size two in $N_2(x)$ where x is the middle element as shown. This will be the subregions for the multidimensional LULU operators.

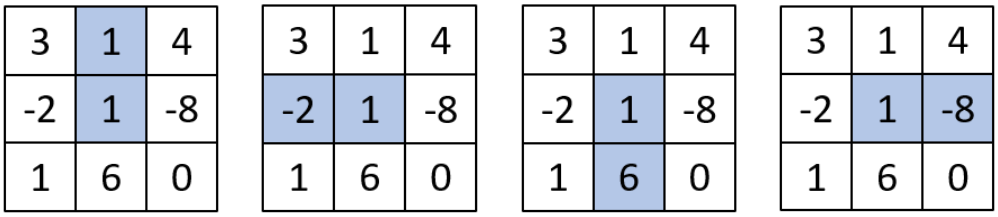
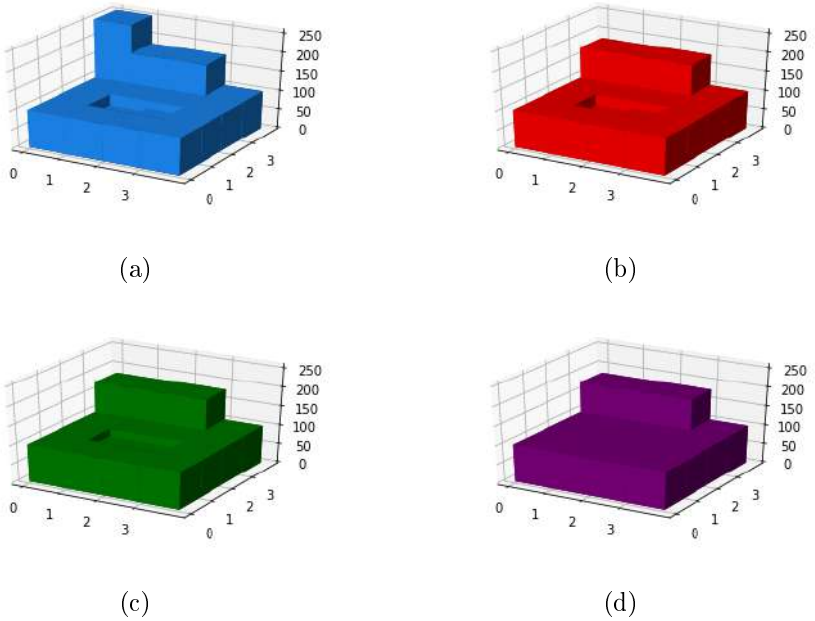


Figure 2.4: The four possible neighbourhoods, $V \in N_2(x)$ for a 2-dimensional signal (image) for the middle element x.

These connected regions, if they are either a local minimum of maximum set, are the bumps and pits the multidimensional LULU operators act on, as in Equation (2.1) and (2.2).

In Figure 2.5 an example of how the LULU operators would smooth a 2-dimensional signal is shown. Now $x \in \mathbb{Z}^2$ and $f(\cdot) \in \mathcal{A}(\mathbb{Z}^2)$ In Figure 2.5a the original signal is shown and Figure 2.5b shows the smoothed signal when L_1 is applied. In Figure 2.5c the smoothed signal when L_1 and U_1 is applied is shown and lastly Figure 2.5d shows the smoothed signal when L_2 and U_2 is applied.



(a) (b)

(c) (d)

Figure 2.5: An example of the LULU operators applied to a 2-dimensional signal. a) The original 2-dimensional signal $f(x)$. b) $(L_1 f)(x)$ the smoothed signal after the bumps of size one are removed. c) $(U_1(L_1 f))(x)$ the smoothed signal after the bumps and pits of size one are removed. c) $(U_2(L_2 f))(x)$ the smoothed signal after the bumps and pits of size two and smaller are removed.

To explain the steps taken to obtain the smoothed signal, Figure 2.6 shows the signal in matrix form and each of the elements are named $f(x_1), \ldots, f(x_{16})$ and the values given in brackets.

$f(x_1)$ (254)	$f(x_2)$ (180)	$f(x_3)$ (180)	$f(x_4)$ (100)
$f(x_5)$ (100)	$f(x_6)$ (100)	$f(x_7)$ (100)	$f(x_8)$ (100)
$f(x_9)$ (100)	$f(x_{10})$ (70)	$f(x_{11})$ (80)	$f(x_{12})$ (100)
$f(x_{13})$ (100)	$f(x_{14})$ (100)	$f(x_{15})$ (100)	$f(x_{16})$ (100)

Figure 2.6: A matrix view of the 2-dimensional signal which is smoothed using the multidimensional LULU operators.

To obtained the results in Figure 2.5b L_1 was applied to the signal. First the entire signal is padded with zeros in 2-dimensions. For element $f(x_1)$ four subregions are created $s_1 = \{0, f(x_1)\} = \{0, 254\}, s_2 = \{0, f(x_1)\} = \{0, 254\}, s_3 = \{f(x_1), f(x_2)\} = \{254, 175\}, s_4 = \{f(x_1), f(x_5)\} = \{254, 90\}$, which are all the possible subregions since 4-connectivity is used. The smoothed value $(L_1 f)(x_1)$ for $f(x_1)$ is then $\max\{\min(s_1), \min(s_2), \min(s_3), \min(s_4)\} = \max\{0, 0, 175, 90\} = 175$. This is repeated for the other 15

elements. To obtain the results in Figure 2.5d, U_2 is applied to the smoothed signal after L_1 and U_1 are applied to the signal. For $f(x_1)$ subregions of size three was created. Due to 4-connectivity being used there will be 12 subregions each unique and containing $f(x_1)$. The smoothed value for $f(x_1)$ is then $\max\{\min(s_1), \ldots, \min(s_{16})\}$.

2.2 Discrete Pulse Transform

When applying the LULU operators to a signal some information is lost because the local minimum and maximum sets are simply smoothed. This information can be very important therefore it will be useful if this information can be stored and used later on. This is what the Discrete Pulse Transform algorithm does.

The Discrete Pulse Transform (DPT) is a decomposition algorithm [54]. DPT decomposes a signal f by recursively applying the L_n and U_n operators for $n = 1, ..., N$, where N is the number of elements in the signal. The local minimum and maximum sets which are smoothed are stored of each iteration as pulses, therefore no information is lost with the decomposition. The DPT decomposition is also mean and variance preserving [2]. This results in a decomposition as follows [20],

$$DPT(f) = \{D_1(f), D_2(f), ..., D_N(f)\}. \tag{2.3}$$

The discrete layers are obtained as follows,

$$\begin{aligned} D_1(f) &= (I - P_1)(f) \\ D_2(f) &= (I - P_2) \circ P_1(f) \\ D_n(f) &= (I - P_n) \circ Q_{n-1}(f) \end{aligned}$$

where $P_n = L_n \circ U_n$ or $U_n \circ L_n$ and $Q_n = P_n \circ \cdots \circ P_1$.

A pulse is defined in Definition 3.

Definition 3. *Pulse [2]*

A function $\phi \in \mathcal{A}(\mathbb{Z}^2)$ is called a pulse if there exists a connected set V and a real number α such that

$$\phi(x) = \begin{cases} \alpha & \text{if} \quad x \in V \\ 0 & \text{if} \quad x \in \mathbb{Z}^2 \backslash V \end{cases}$$

Each of these discrete layers, $D_n(f)$ consists of all the pulses of size n i.e. $D_n(f) = \sum_{j=1}^{\gamma(n)} \phi_{jn}$ where $\gamma(n)$ is the number of pulses of size n and ϕ_{jn} is pulse j of size n, n being a non-zero value. A pulse is an array of the same dimension as the original signal consisting of only zeros except for some connected regions were the pulse is some constant real value. The value of the constant is the difference between the height of the connected region, which can be negative in the case of a local minimum set, in the signal and their closest neighbour.

In each passing, the LULU operators will remove the local minimum and maximum sets of size $n = 1, ..., N$ and these sets which were cut off are stored in the pulses. All the edges of the array are connected to an infinite zero signal. This decomposition is done such that the original signal can be obtained by adding all the discrete layers together as follows,

$$f = \sum_{n=1}^{N} D_n(f) = \sum_{n=1}^{N} \sum_{j=1}^{\gamma(n)} \phi_{jn}.$$

A simple example of DPT applied to a 1-dimensional array is now given. A graphical representation of the results is shown in Figure 2.7. Figure 2.7a is the original signal. Figure 2.7b, d, f and h is the smoothed images after these local minimum/maximum sets has been removed. Figure 2.7c, e, g and i is the pulses that were removed from the image in each step. As we can see in the first step the local maximum sets of size one were removed and stored as pulses. Note that the values of the right-hand side of the graphic is not a local minimum of size one because the edge of the array is connected to an infinite zero signal, thus this cannot be a local minimum as 150 is less than 170 and greater than zero. As we can see if we add all these pulses together we will obtain the original signal.

In Figure 2.8 an example of a 2-dimensional signal being decomposed using DPT is shown. Figure 2.8a is the original signal. Figure 2.8b, d, f and h is the smoothed images after these local minimum/maximum sets has been removed. Figure 2.8c, e, g and i is the pulses that were removed from the image in each step. In the last step the local maximum set has a size of 16. This is a local maximum set since the signal is connected to an infinite signal of zeros on each side.

When smoothing a signal with methods such as the LULU operators some information is lost because we throw away the values which are removed when smoothing. The advantage of DPT decomposition is that this information is not lost, it is stored and can be used later on. The number of elements in the pulses range from 1 to N where N is the number of elements in the signal, the number of elements which make up the pulse in known as the scale of the pulse. If we were to extract all the pulses of scale two we will extract all the pulses consisting of two elements. These pulses at their various scales can either contain noise or valuable information. If the right range of scales can be selected it can represent the structure of the image well, especially for texture rich images [18].

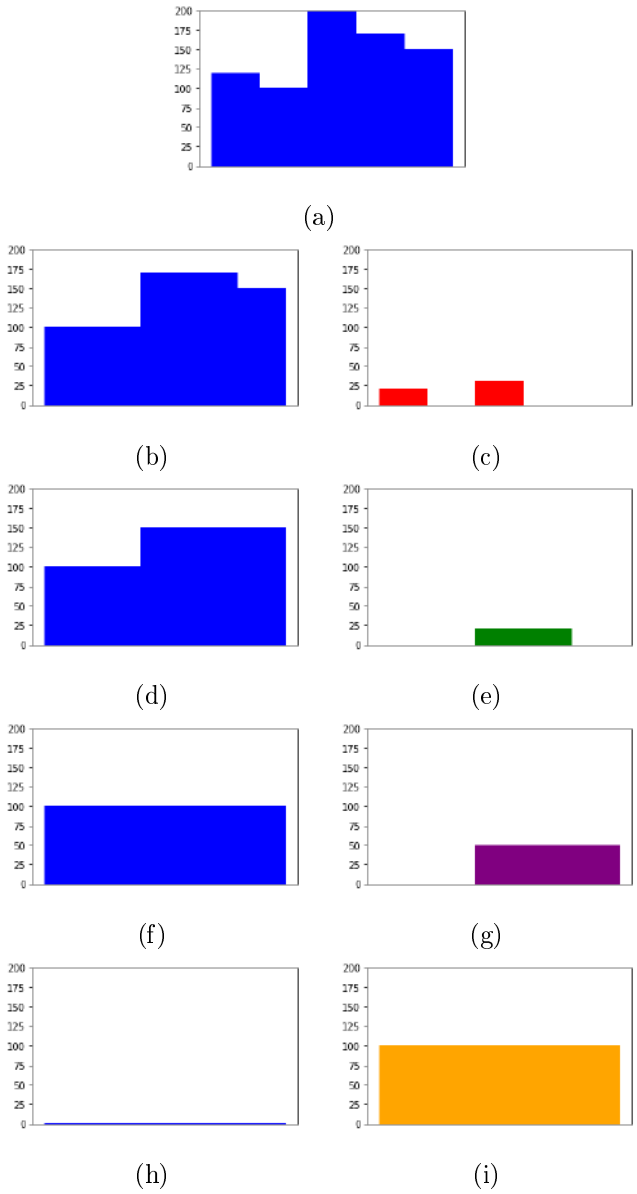


Figure 2.7: An example of DPT applied to a 1-dimensional signal showing the smoothed signals and extracted pulses. a) The original signal. b) $Q_1(f)$ the smoothed signal after applying P_1. c) $D_1(f)$ the extracted pulses of size one. d) $Q_2(f)$ the smoothed signal after applying $P_2 \circ P_1$. e) $D_2(f)$ the extracted pulses of size two. f) $Q_3(f)$ the smoothed signal after applying $P_3 \circ P_2 \circ P_1$. g) $D_3(f)$ the extracted pulses of size three. h) $Q_4(f)$ the smoothed signal after applying $P_4 \circ P_3 \circ P_2 \circ P_1$. i) $D_4(f)$ the extracted pulses of size 4.

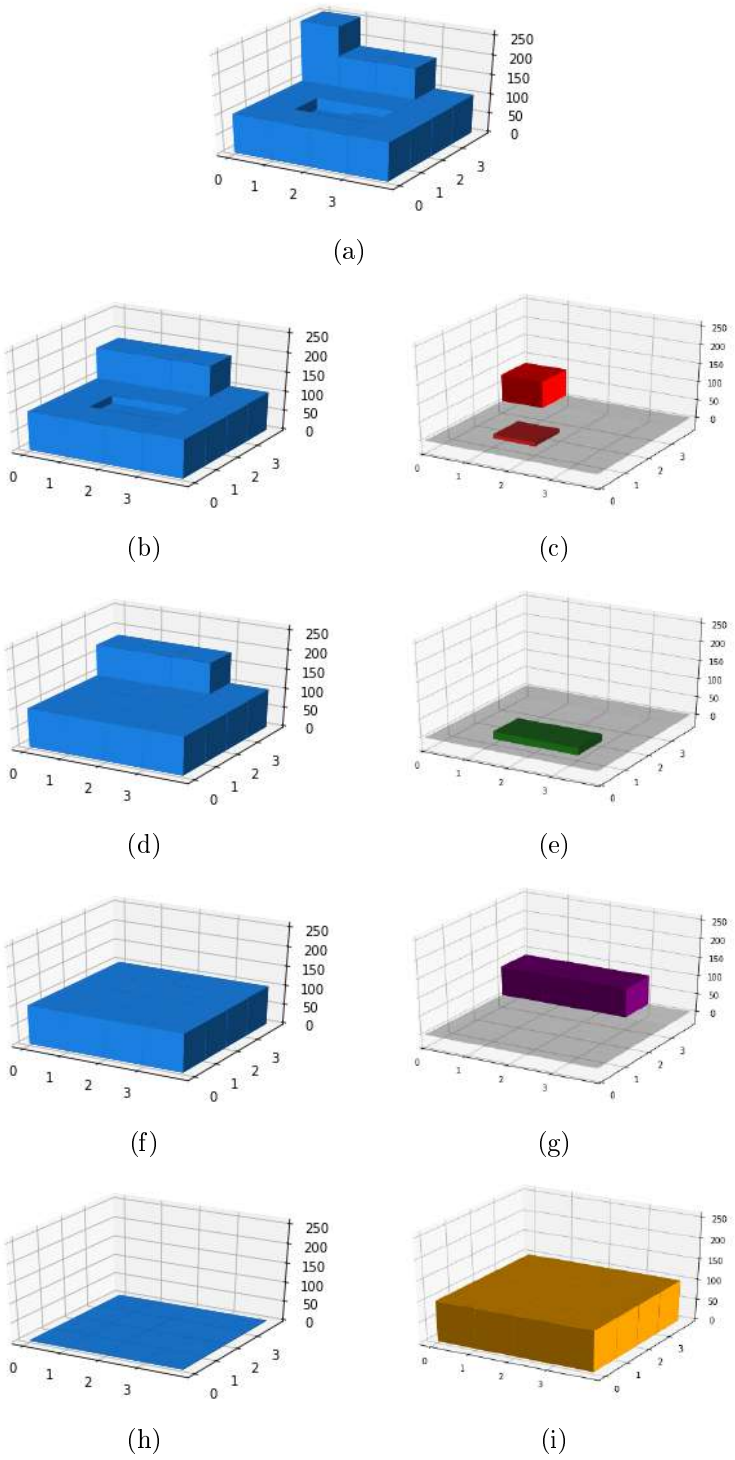
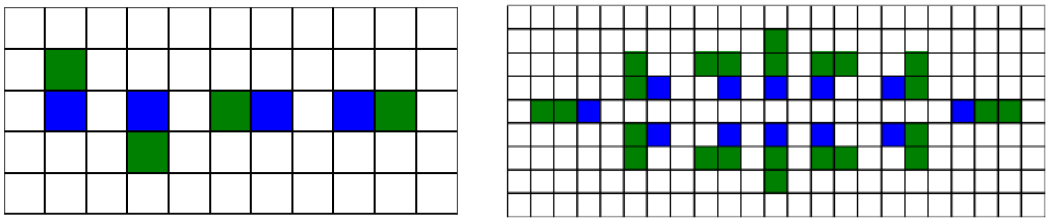


Figure 2.8: An example of DPT applied to a 2-dimensional signal showing the smoothed signals and extracted pulses. a) The original signal. b) $Q_1(f)$ the smoothed signal after applying P_1. c) $D_1(f)$ the extracted pulses of size one. d) $Q_2(f)$ the smoothed signal after applying $P_2 \circ P_1$. e) $D_2(f)$ the extracted pulses of size two. f) $Q_3(f)$ the smoothed signal after applying $P_3 \circ P_2 \circ P_1$. g) $D_3(f)$ the extracted pulses of size three. h) $Q_{16}(f)$ the smoothed signal after applying $P_{16} \circ P_{15} \circ \cdots \circ P_1$. i) $D_{16}(f)$ the extracted pulses of size 16.

A problem encountered when applying DPT using the LULU operators is the amount of memory required to store the pulses. The amount of memory required to store an array in memory is directly related to the number of elements which needs to be stored. If we have an image of size 200×200 that results in 40 000 elements. If we were to decompose this using DPT the possible range of pulses would be $1, \ldots, 40\,000$. Therefore the number of elements which needs to be stored in memory is $200 \times 200 \times 40001 \approx 1.6$ billion elements. It is easy to see how this can use up a lot of memory.

Another problem encountered when applying DPT using the LULU operators is the fact that it is a computationally intensive algorithm. If we use 4-connectivity as shown in Figure 2.3b the number of neighbours for each element is shown in Figure 2.9 for $n = 1$ and $n = 2$. For $n = 1$ each element has four possible connected sets and for $n = 2$ each element has 12 possible connected sets. Therefore, it is easy to see that this algorithm will be computationally intensive if we were to apply it to an image with thousands of pixels since all possible connected regions for size $n = 1, \ldots, N$ needs to be found. Applying the DPT algorithm using the LULU operators to a signal with N elements results in a process with $\mathcal{O}(\mathcal{N}^3)$ complexity [13].



$n = 1$ $n = 2$

Figure 2.9: All possible connected sets using 4-connectivity for $n = 1$ and $n = 2$. The blue squares are the elements of interest and the green squares are the neighbours forming the possible connected sets, which is $N_n(x)$ for $n = 1$ and $n = 2$.

Up to now all the examples were implemented as per definition of the LULU operators using the minimum and maximum operators. But as shown in Figure 2.9 the number of subregions grows exponentially as the size of n increases. Laurie suggested an algorithm to address the high complexity of the DPT algorithm in [36]. This algorithm is called the Roadmaker's algorithm. This algorithm searches for all connected regions (connected components) and then determines which of these connected regions are local minimum or maximum sets. If a connected region is a local minimum or maximum set then the LULU operators are applied to these regions only and not to all connected sets of size n, namely $N_n(x)$. It was proven in [36] that the Roadmaker's algorithm is equivalent to the iterated LULU operators in 1-dimension and in [35] for multiple dimensions. In [2] it is also proven the Roadmaker's algorithm holds on $\mathcal{A}(\mathbb{Z}^d)$.

Applying the Roadmaker's algorithm to a signal can be compared to smoothing a rough dirt road. The signal of interest is the dirt road, we assume that there is an infinitely long road of height zero connected

to each end of the road. In the first pass over the signal all the bumps of size one are removed. This is done by subtracting a value such that each bump is the exact same height as the highest of all its neighbours. Then all the pits of size one is removed. This is done by adding a value such that the height of each pit is exactly the same height as the lowest of all its neighbours. The extracted pulses are then stored. This process is repeated for bumps and pits of size 2 to N. Although this implementation is faster than the DPT using the LULU operators it still requires a lot of memory since an array of the same size as the signal is created for each size pulse. Fabris-Rotelli and van der Walt [19] proposed a more memory efficient implementation of this algorithm where only the value of the pulse and its location in terms of the row and column is stored, this is known as the compressed sparse row format. The Roadmaker's algorithm has $\mathcal{O}(\mathcal{N})$ complexity [36], which is a big improvement from the DPT algorithm making use of the LULU operators and sets $N_n(x)$ directly.

2.2.1 Roadmaker's Pavage

In 2014, Stoltz proposed a graph based DPT algorithm called the Roadmaker's Pavage [62]. This algorithm is based on the Roadmaker's algorithm but uses a more usable format to extract the pulses. The following notation will be used for the Roadmaker's Pavage algorithm,

f	–	The d-dimensional signal of size N.
N	–	Number of elements in the signal.
$C_k(i)$	–	Connectivity function
W_G	–	The working graph.
P_G	–	The pulse graph.
$V_{G,i}$	–	Node i of graph G
$FeatTable$	–	The feature table.

Hence $f \in \mathcal{A}(\mathbb{Z}^d)$ is the signal we wish to decompose. This is a d-dimensional signal with N elements.

The connectivity needs to be defined, namely $C_k(i)$ which is the k^{th} neighbour of element i. When working with a 2-dimensional signal such as an image, we flatten the signal into a 1-dimension. To do this we give each element of the signal a unique and logical label. We are able to use these labels to determine each element's position and neighbours based on the chosen connectivity. We then need to define our connectivity in 1-dimension.

In Figure 2.10a, a simple example of a 2-dimensional signal flattened into 1-dimension. In this example the number of columns (n_1) is three and the number of rows (n_2) is also three. In the images the black

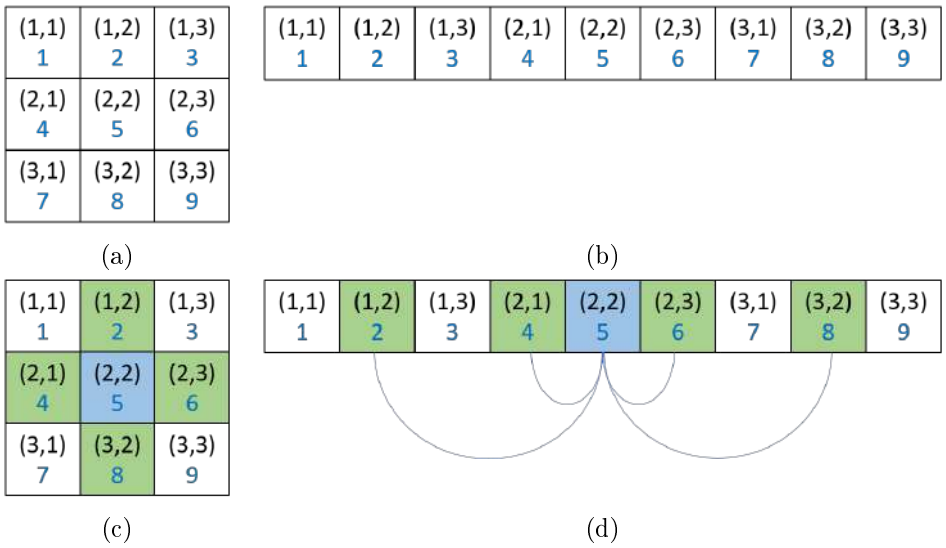


Figure 2.10: Example of how a 2-dimensional signal is flattened into 1-dimensional and neighbours using 4-connectivity. The black text shows the indexing in 2-dimensions and the blue text the indexing in 1-dimension. a) The original 2-dimensional signal. b) The signal flattened into 1-dimension. c) Element five of the signal (blue element) and its neighbours (green elements) using 4-connectivity. d) The flattened signal showing again element five of the signal and its neighbours.

text in each element is the indexing in 2-dimensions, so the row and column index is used. The blue text in each element is the indexing in 1-dimension. If we have a look at element five (blue element in Figure 2.10c) we can see that its neighbours are elements 2, 4, 6, 8. These can be determined by using the index of the element in question and the number of columns ($n_1 = 3$) as follows $C_1(5) = 5 - n_1 = 2$, $C_2(5) = 5 + n_1 = 8$, $C_3(5) = 5 - 1 = 4$ and $C_4(5) = 5 + 1 = 6$. Therefore, we can easily define the connectivity function for this specific case as follows,

$$\text{Top neighbour:} \quad C_1(i) = \begin{cases} 0 & i = 1,2,3 \\ i - n_1 & \text{otherwise} \end{cases}$$

$$\text{Bottom neighbour:} \quad C_2(i) = \begin{cases} 0 & i = 7,8,9 \\ i + n_1 & \text{otherwise} \end{cases}$$

$$\text{Left neighbour:} \quad C_3(i) = \begin{cases} 0 & i = 1,4,7 \\ i - 1 & \text{otherwise} \end{cases}$$

$$\text{Right neighbour:} \quad C_4(i) = \begin{cases} 0 & i = 3,6,9 \\ i + 1 & \text{otherwise} \end{cases}$$

This connectivity function can be written more generally as follows,

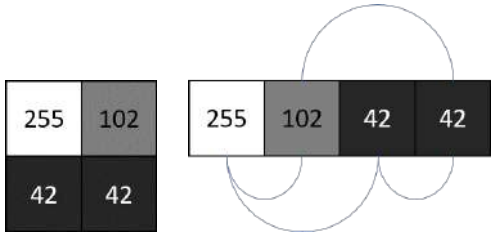
$$C_1(i) = \begin{cases} 0 & i \leq n_1 \\ i - n_1 & \text{otherwise} \end{cases} \tag{2.4}$$

$$C_2(i) = \begin{cases} 0 & i > (n_2 - 1)n_1 \\ i + n_1 & \text{otherwise} \end{cases} \tag{2.5}$$

$$C_3(i) = \begin{cases} 0 & i = kn_1 + 1 \ \forall \ k = 0, \ldots, n_2 - 1 \\ i - 1 & \text{otherwise} \end{cases} \tag{2.6}$$

$$C_4(i) = \begin{cases} 0 & i = kn_1 \ \forall \ k = 1, \ldots, n_2 \\ i + 1 & \text{otherwise} \end{cases} \tag{2.7}$$

This connectivity function will be used to determine neighbours within the signal. Consider the following example of a greyscale image to which DPT is applied using the Roadmaker's Pavage algorithm. In Figure 2.11a the original image is shown. The image is flattened into 1-dimension and the neighbours of each element shown in Figure 2.11b.



(a) (b)

Figure 2.11: Roadmaker's Pavage example. a) The signal to which DPT should be applied using the Roadmaker's Pavage algorithm. b) The flattened signal showing the neighbours of each element.

The neighbours for each element is,

Element index	$C_1(i), C_2(i), C_3(i), C_4(i)$
1	0, 3, 0, 2
2	0, 4, 1, 0
3	1, 0, 0, 4
4	2, 0, 3, 0

The next step is to create a working graph, W_G. The working graph represents the elements of the signals and connections between the elements which are neighbours. The working graph consists of $N + 1$ nodes where N is the number of elements in the signal. Each node has two attributes namely value and scale. The scale of the node is the number of elements which it represents and the value of the node is the value of the elements which it represents. A node $V_{W_G,i}$ is created in the working graph for each of the elements

in the signal, each node has a scale of one and a value equal to the value of the element it represents. Then a node with scale ∞ and value zero is created, since in DPT we require the elements at the edge of the signal to be connected to an infinite signal of zeros. The initial working graph for the signal in Figure 2.11 is shown in Figure 2.12.

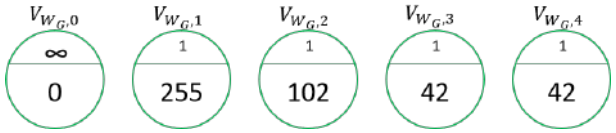


Figure 2.12: The initial working graph of the signal to which DPT is being applied to.

Now we need to create edges between the nodes of which the elements are neighbours. Using the connectivity function defined in Equations (2.4) and (2.5) we determine the edges of the graph. For example, node $V_{W_G,1}$ in Figure 2.12 has three neighbours which are nodes $V_{W_G,0}$, $V_{W_G,2}$ and $V_{W_G,3}$. These edges are shown in the working graph in Figure 2.13.

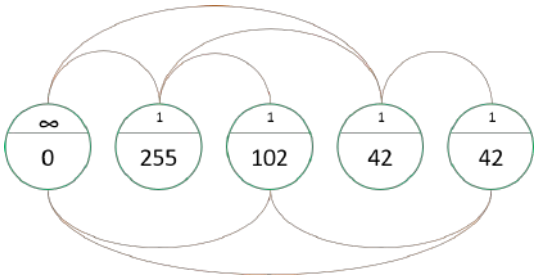
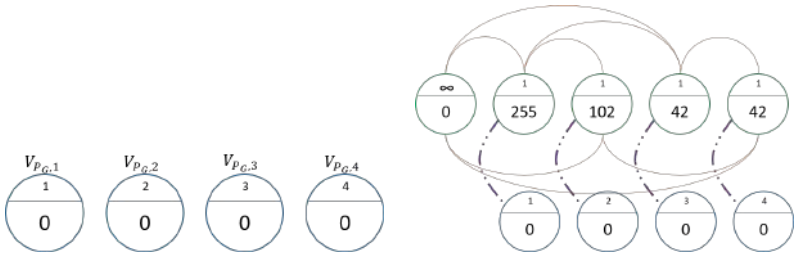


Figure 2.13: The initial working graph with the edges connecting all the elements with their neighbours.

Now a pulse graph P_G is created. The pulse graph also has nodes but these nodes represent the pulses extracted and have arcs instead of edges which represents the links between the pulses. The pulse graph will help to determine which pulses belong to which element in the signal. To initialise the pulse graph we create a node for each element in the signal. Each node has a value of zero and its scale is the index of the element it is associated with. This is done for reference purposes. When the decomposition is done and we want to extract pulses we will start at the pulses with the desired scales. The arcs will be followed up in the graph to these initial nodes and we use the value in the scale to determine which element in the original signal forms the pulses. The node in the pulse graph is connected to the corresponding node in the working graph with virtual edges. For the example the pulse graph has four nodes since we have four elements in the signal. The pulse graph is shown in Figure 2.14a. The virtual edges connecting the nodes from the pulse graph to the corresponding node in the working graph are shown in Figure 2.14b.

The nodes in the working graph with the same values are now merged into one node. This node's scale will be the sum of the scales of all the nodes being merged. All the edges of the nodes being merged are moved to the one remaining node and then the rest of the nodes are deleted. In Figure 2.15 the new



(a) (b)

Figure 2.14: Roadmaker's Pavage example continued. a) The initial pulse graph. b) The working graph (green nodes) and pulse graph (blue nodes) and the virtual edges connecting the nodes from the pulse graph to the corresponding nodes in the working graph.

working graph and connections are shown were nodes $V_{W_G,3}$ and $V_{W_G,4}$ were merged and all edges moved to node $V_{W_G,3}$. When this is done there will be one edge for each neighbour of the entire connected region and not edges for each element in the connected region.

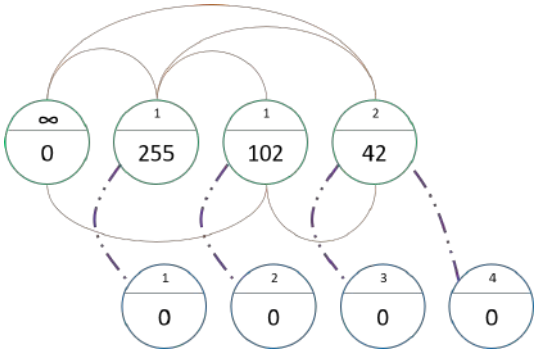


Figure 2.15: The working and pulse graphs after the nodes of same value are merged and all edges moved to the remaining node.

The feature table is created next. In order to do this, we first need to define a feature. A node is defined as a pit if the node is strictly smaller than all the node's neighbours, also known as a local minimum set. A node is defined as a bump if the node is strictly larger than all of the node's neighbours, also known as a local maximum. These are the same as defined in Definition 2. A pit or bump can have a size of $1, \ldots, N$. If a node is defined as a pit or a bump it is known as a feature. Each node is now checked to determine whether it is a feature. The feature table is created by adding a reference to every node which is identified as a feature. This table will be used to keep track of the features as they change and disappear. Therefore, the decomposition algorithm will repeat itself until there are no features left in the feature table.

In the example, the feature table will only contain one reference and this is a reference to node $V_{W_G,1}$. Since this node has edges connecting it to nodes with values zero and 102 it is a bump of scale one and

therefore, a feature. The rest of the nodes have edges connecting it to at least one node with a value larger than itself and at least one node with a value smaller than itself. For example, node $V_{W_G,2}$ with value 102 has edges with nodes $V_{W_G,1}$, $V_{W_G,3}$ and $V_{W_G,0}$ with values 255, 42 and 0 respectively. Thus, this node is not strictly greater than or less than all of its neighbours and is therefore not a feature. Equation (2.8) is the feature table with a reference to all the features which in this case is only one node.

$$FeatTable = \{V_{W_G,1}\} \tag{2.8}$$

The next step will be the actual DPT decomposition. For this we define two operators $P_n = L_n \circ U_n$ or $U_n \circ L_n$ and $Q_n = P_n \circ \cdots \circ P_1$. These are operators which operate on the nodes created in the graphs. P_n flattens all local minimum and maximum sets of size n and smaller. Q_n also flattens all local minimum and maximum sets of size n and smaller but it is done sequentially which allows for the flattened features to be stored for each scale $n = 1, \ldots, N$.

In the first pass all bumps of scale one in feature table are flattened. After this is done all the pits of scale one are flattened. When this feature is flattened it creates a node which will have a neighbouring node of the same value. Therefore, the process of clustering node with the same value has to be repeated after each decomposition step. This decomposition is done for $n = 1, \ldots, N$.

For the example when applying P_1, the feature at node $V_{W_G,1}$ needs to be flattened. This node will be flattened to this node's nearest neighbour which has a value of 102. This value which was cut-off needs to be added to the pulse graph at the relevant nodes. In Figure 2.16a the working and pulse graph after this feature has been flattened can be seen. In Figure 2.16b the working and pulse graphs after the nodes with the same values has been merged is shown.

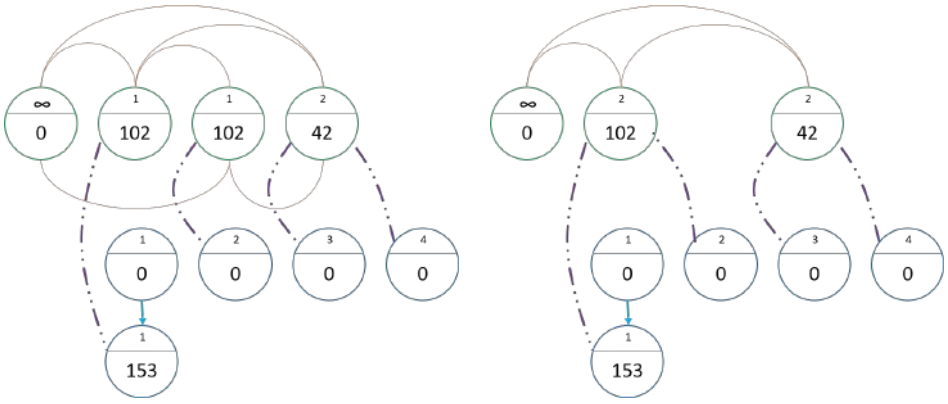


Figure 2.16: Roadmaker's Pavage example continued. a) The working and pulse graphs after the feature at node $V_{W_G,1}$ has been flattened. b) The working and pulse graph after the nodes with the same values have been merged.

The feature table now needs to be updated but since the new feature is at the same node as in the previous step it will remain the same. The next step will be to apply P_2. This will again flatten the feature at $V_{W_G,1}$ since it is a bump of size two. The resulting working and pulse graph after this feature are flattened and the pulse created is shown in Figure 2.17a. The feature table again is unchanged as the new feature remains at node $V_{W_G,1}$. In Figure 2.17b the working and pulse graph after all pulses has been extracted is shown. At this point there are no features remaining so the feature table will be empty and the decomposition process is over. To get to this final step P_1, P_2 and P_4 had to be applied. It is important to note that although not present in this simple example it is possible to obtained pits (local minimum sets), when these features are flattened it will result in a pulse with a negative value.

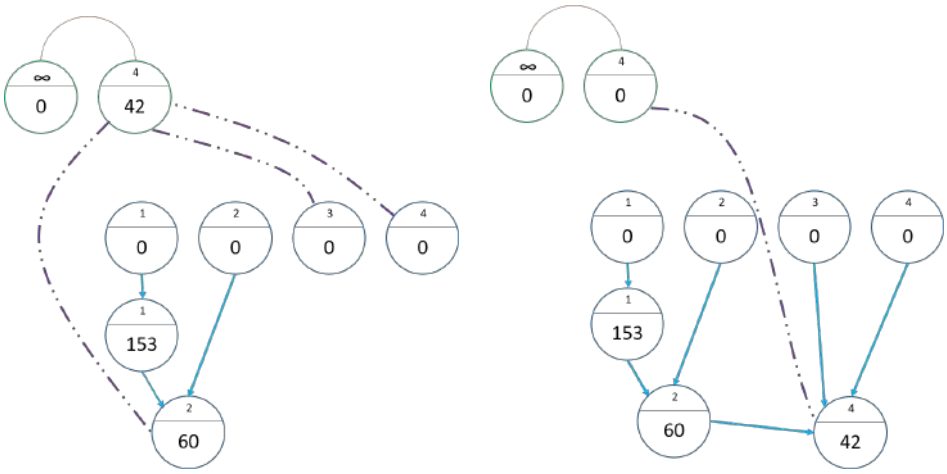
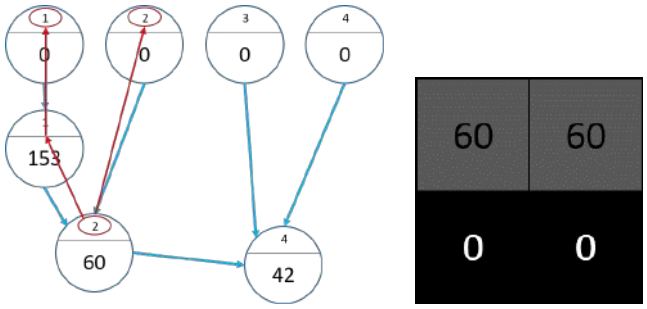


Figure 2.17: Roadmaker's Pavage example continued. a) The working and pulse graphs after P_2 was applied. b) The working and pulse graph after the decomposition is done.

We can see that the final pulse graph has a tree type structure. It is a special case of a Directed Acyclic Graph [62]. This is a type of graph where the nodes are connected with arcs, the arcs are always directed to the node's parent. In this case the arcs are directed from the child nodes to the parent nodes. This can be used to extract the pulses. For example, if we want to extract the pulses of scale two we find all the nodes in the pulse graph with a scale of two and a value unequal to zero. We can then follow these arcs to their parent node and so on until we reach a node that does not have a parent. Since the nodes in the pulse graph that were created in the first step has a scale of the index of the element it represents we can use this to determine where in the original signal this pulse is present. This is known as root reconstruction [62]. In Figure 2.18a, this path that needs to be followed through the pulse graph as well as the index it represents is shown when extracting pulses of scale two. In Figure 2.18b the extracted pulse of scale two is shown.

Using this pulse graph, we can also determine to which size pulses a certain pixel belonged to through



(a) (b)

Figure 2.18: Roadmaker's Pavage example continued. a) The path that needs to be followed through the pulse graph to extract pulses of scale two using root reconstruction. b) The extracted pulse of scale two.

the whole decomposition process. We can do this by starting at the node in the pulse graph with value zero and scale the index of the element in question. We then go to this node's child node and so on each time extracting either the size or scale at each node depending on what is needed. This is known as point reconstruction [62]. An example of this path is shown in Figure 2.19. As we can see element one was present in pulses of size 1, 2 and 4.

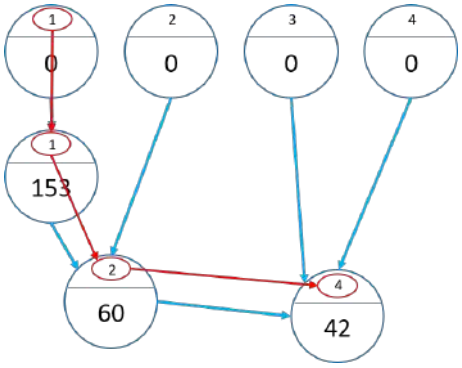
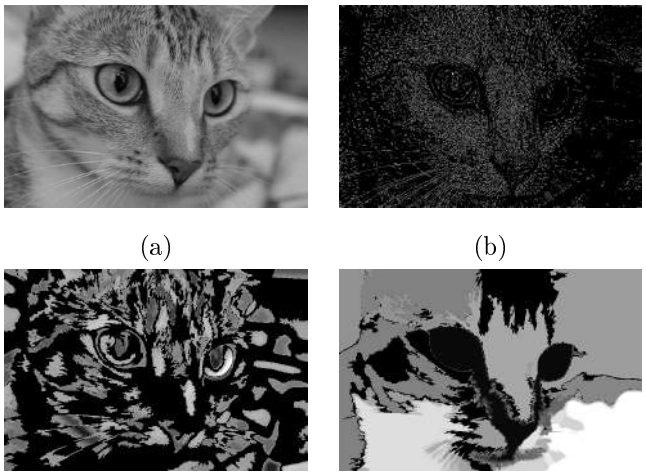


Figure 2.19: The path followed through the pulse graph to determine in which size pulses element one was in using point reconstruction.

The Roadmaker's Pavage algorithm has the same complexity as the Roadmaker's algorithm, $\mathcal{O}(\mathcal{N})$, but it is much more memory efficient since the extracted pulses do not have to be stored in matrices. Only when the extracted pulses are used to reconstruct the image does a matrix have to be created.

As an example, the DPT decomposition is applied to a greyscale image and some scale pulses extracted to see what it yields. It is important to note that some pulses with negative values were extracted. In order to be able to visualise something as a greyscale image all the pixel values need to be between 0 and 255. Therefore the array representing the pulses was scaled so that all the pixel values are between 10 and 255 and then all the pixels that were zero in the pulse is kept zero. This is only done for visualisation purposes, in all calculations the original extracted pulses are used, negative or positive.



(a) (b)

(c) (d)

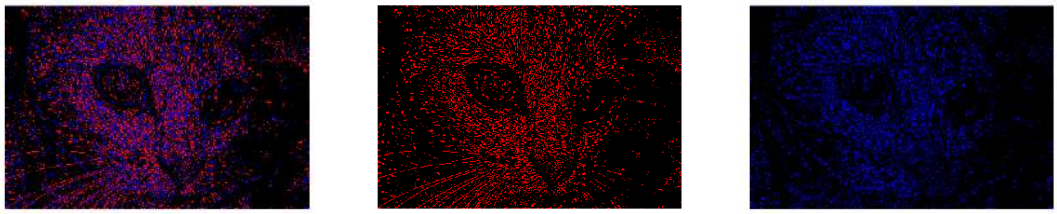
Figure 2.20: An example of DPT applied to a greyscale image. a) The image being decomposed. b) The DPT pulses of scale 1 to 10 extracted. c) The DPT pulses of size 100 to 500 extracted. d) The DPT pulses size 10000 to 20000 extracted

As we can see the smaller scale pulses tend to extract smaller features in the images. For example, in Figure 2.20b we can see that very small features of the cat are extracted such as the whiskers and hairs of the cat. In Figure 2.20c we can see the larger size pulses extract larger features such as the cat's eye shapes. In Figure 2.20d we can see large features such as the shape of the cat's face is extracted.

In order to highlight the fact that some negative features are also extracted the images in Figure 2.21 was created. Here RGB images was created were all the positive extracted pulses are shown on the red spectrum and all the negative extracted features are shown on the blue spectrum. The absolute values of the negative features are shown on the blue spectrum. From the images in Figure 2.21 we see that the negative pulses extracted also yield valuable information since we can see the object present in the image by only looking at the negative valued pulses (see Figure 2.21c). The positive and negative values pulses together represents the structure of the object well.

2.3 Neural networks

Neural networks are a very powerful subset of deep learning. These are models which can input a variety of data types. A simple example of a neural network is shown in Figure 2.22. A neural network consists of a certain number of layers of which there are three types. Firstly, an input layer, which is the very first layer of the neural network and is how the data is inputted to the network. Secondly there are hidden layers. The number of hidden layers can be chosen. Lastly there is an output layer, which is the very last layer of the network and yields the output the network has generated. Each layer has a certain number of



(a) (b) (c)

Figure 2.21: The extracted pulses from a greyscale image where the positive features are shown in red and the negative features in blue. a) Both the positive and negative features are shown. b) Only the positive extracted features are shown. c) Only the negative extracted features are shown.

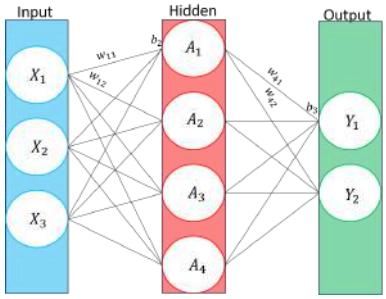


Figure 2.22: A basic artificial neural network with three layers (one input, one hidden and one output). The input layer has three nodes (X_1, X_2, X_3), the hidden layer four nodes (A_1, A_2, A_3, A_4) and the output layer two nodes (Y_1 and Y_2). The nodes in each layer is connected to the adjacent layers via weights $w_{11}, \ldots, w_{72}$. The hidden layer has bias b_2 and the output layer has bias b_3

nodes, the input layer's number of nodes is the number of elements in a single observation. For example, if we what to classify greyscale images of size 200×200 pixels there will be an input node for each pixel, therefore 40 000 input nodes. Each hidden layer's number of nodes is chosen by the model builder. The output layer has the same number of nodes as the possible outcomes of the model. For example, if we are creating a neural network to predict the stock price of a commodity for the following day it will have a single node in the output layer which predicts a continuous unbounded value. If we are creating a neural network to classify an image into one of ten classes, the output layer will have ten nodes each predicting a probability of the observation falling in the class represented by the node.

Each of the nodes in a specific layer is connected to each of the nodes in the adjacent layers with weights. Each hidden and output layer typically has a bias which is added to all the values passed to the layer. A layer where all the nodes is connected to all the nodes in the adjacent layers is called a fully connected layer. So, the input for a node which is not in the input layer is the weighted sum of the previous layers output plus the bias of that specific layer and then an activation function (Examples shown visually in Figure 1.2 and formulas given in Table 2.1) is applied to this value. The activation function at each

Name	**Function**
Sigmoid	$f(x) = \frac{\exp^x}{\exp^x + 1}$
Hyperbolic tan	$f(x) = \tanh(x)$
Rectified linear unit (ReLu)	$f(x) = \max\{0, x\}$
Leaky Rectified linear unit (Leaky ReLu)	$f(x) = \max\{\alpha x, x\}$, $\alpha \in [0, 1]$

Table 2.1: Some activation functions [58].

layer adds some non-linearity to the model. The non-linearity is good since there is not always a linear relationship between the explanatory and response variable. The activation function also shows important information by returning a higher value for the important nodes which will have a more significant impact on the optimisation since it is a bigger value. These activation functions can be different for each layer. The only layer which has restrictions on the activation function used on it is the output layer of a classification neural network with more than one output node. This layer must have an activation function which returns a value between zero and one (Sigmoid or SoftMax is most commonly used) [27]. The reason for this is that if each of the output nodes have a value between zero and one it can be seen as the probability of the observation falling in the class which that node represents. Using a function such as SoftMax also ensures that all the outputted probabilities sum to 1. Then the class it is predicted in is the class whose node has the highest probability. It is important that we must be able to determine the derivative of an activation function for the optimisation.

In order for a neural network to perform optimally the optimal weights and biases should be obtained. This is done by using a labelled training sample to minimize a chosen loss metric. A relevant loss metric needs to be chosen for the specific type of data on hand. For example, if we want to train a neural network to correctly classify the MNIST[1] hand-written digits dataset we would require a number of images of hand written digits which are correctly labelled. A loss metric such as cross entropy loss [72] which can measure the performance of a classification model is also required. Images are inputted into the network and the network makes a prediction. The predicted and true labels are then used to calculate the loss. The weights and biases are then updated to minimize the loss. The optimisation of the network is most commonly done using stochastic gradient descent [9] along with backpropagation [56]. This is a gradient based optimisation method. This can be cumbersome to do since the network has so many nested functions. Since at each layer the input values are passed through an activation functions the output of this layer is then used as input for the next layer were it is again passed through an activation function. Some notation for the neural networks which will be used is defined in Table 2.2.

If we take the network shown in Figure 2.22 the weights connecting the input and hidden layers are $w_{11}, \ldots, w_{34}$ and create W_1 as follows,

[1] Data available at https://www.kaggle.com/c/digit-recognizer

x_{ij}	-	Element i of observation j of the input observation.
X_j	-	The matrix with all the elements of observation j.
y_i	-	The true label of observation i.
Y	-	Array of true label of observations.
f_i	-	The activation function applied to the i^{th} layer's input
w_{ij}	-	The weight connecting node i from the previous layer to node j of the current layer.
W_i	-	The matrix of weights connecting layer i to layer $i+1$.
b_j	-	The bias for layer j.
a_{ij}^{in}	-	Input for node j of layer i (before the activation function is applied).
A_i^{in}	-	The vectors of nodes inputs in layer i
a_{ij}^{out}	-	Output of node j of layer i (after the activation function is applied).
A_i^{out}	-	The vectors of nodes outputs in layer i
$\hat{y}_i$	-	Predicted label for observation i.
$\hat{Y}$	-	The array of predicted labels for a sample (This will be the final A_i^{out})

Table 2.2: Notation for neural networks.

$$W_1 = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix}.$$

Then we can calculate A_1^{in}, A_1^{out}, A_2^{in} and $\hat{Y}$ when X_i is inputted as follows, $A_1^{in} = X_i W_1 + b_2$, $A_1^{out} = f_1(A_1^{in})$, $A_2^{in} = A_1^{out} W_2 + b_2$ and $\hat{Y} = f_2(A_2^{in})$ where b_1 and b_2 are the bias terms for the hidden and output layer and f_1 is the activation function of the hidden layer and f_2 the activation function of the output layer. Therefore, $\hat{Y}$ in terms of X is

$$\hat{Y} = f_2(f_1(XW_1 + b_2)W2 + b_3). \tag{2.9}$$

Next a loss function needs to be decided on. There are many possible loss functions which can be used of which only two will be defined now for a sample of size n. The input for the model is $X_1, \ldots, X_n$ and corresponding labels $y_1^k, \ldots, y_n^k$ where y_i^k is one if observation i falls in class k and zero otherwise. The estimated labels which are estimated by the model is represented by $\hat{y}_1^k, \ldots, \hat{y}_n^k$. $\hat{y}_i^k$ are probabilities of observation i falling in class k. The cross-entropy loss is most commonly used for classification [72] and mean squared error which is most commonly used for continuous prediction [70]. For this reason, we only look at these two loss functions.

The Cross entropy loss (used for classification) is calculated as

$$L(Y, \hat{Y}) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_i^{(k)} \log(\hat{y}_i^{(k)}) \tag{2.10}$$

where $y^{(k)}$ is one if class label k in the correct classification and zero if not, with K possible classes. $\hat{y}_1^k$ is the probability of observation i falling into class k.

The Mean Squared Error (used for regression) is calculated as

$$L(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{2.11}$$

The objective is to use the loss function along with the model and labelled data to determine the optimal weights and biases of the neural network. To do this a gradient based optimisation algorithm namely stochastic gradient descent (SGD) can be used. SGD is an iterative optimisation algorithm which updates the parameters based on the loss function. The reason SGD is used is because the model can have thousands if not millions of parameters. Due to this the error surface of the loss function is highly complex. In order for the model to perform optimally we need to find the global minimum and not a local minimum of the error surface. SGD is more likely to find a global minimum [9]. SGD also lightens the high computational burden associated with high-dimensional optimisation problems. The reason being that SGD only estimates the gradient using a sample of the data instead of determining the exact gradient using all the data as is done with gradient descent [9].

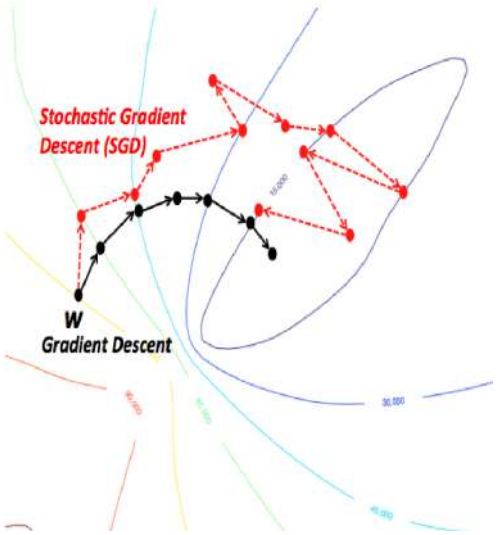


Figure 2.23: The path of optimisation of gradient descent (black path) and stochastic gradient descent (red path).

The fact that the gradient is only estimated causes the parameter values to jump around a lot more (see Figure 2.23[2]) and because of this it is less likely to get stuck in a local minimum. Using an appropriate

[2]Obtained from https://wikidocs.net/3413 on 14 July 2020

loss function the loss of a specific sample can be calculated and along with this the gradient of the loss function with respect to each parameter in the model. The parameters are then updated as follows,

$$\boldsymbol{\beta_i} = \boldsymbol{\beta_{i-1}} - r * \nabla(\boldsymbol{X}, \boldsymbol{\beta_{i-1}}) \tag{2.12}$$

where $\boldsymbol{\beta_i}$ are the parameter values at step i, r is the learning rate and $\nabla(\boldsymbol{X}, \boldsymbol{\beta_{i-1}})$ the derivative of the loss function with respect the parameters at step $i-1$. This updating is done until convergence. When looking at Equations (2.9) - (2.11) it can be seen that to get the derivative of all the weights and biases in this very simple neural network is going to be tedious. Therefore, the algorithm backpropagation [56] was developed which allows us to easily calculate the gradient of the loss function with respect to the weights and biases. This algorithm makes use of the chain rule of differentiation to obtain the derivatives of the weights and biases. Let's consider an example where the variable z depends on the variable y which in turn is depend on the variable x. Then the derivative of z with respect to x can be calculated using the chain rule as follows,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}.$$

Keeping in mind that $\hat{Y}$ is the output (predicted label) from the model, A_i^{out} is the output of layer i, W_i is the matrix of weights connecting layer $i-1$ and i and b_i is the bias terms of layer i. Using this rule we can obtain the gradient of the loss function L with respect to the weights and the biases of the last layers since the output of the last layer $\hat{Y}$ is dependent on the output of the second layer, A_2^{out} which is dependent on the weights between the second and third layer, W_2 and the third layer's bias b_3. Thus,

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial W_2}, \tag{2.13}$$

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial b_3}. \tag{2.14}$$

To obtain the gradient of the loss function with respect to the weights, W_1 and biases, b_2 in the first layer we can apply this rule again,

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial A_1^{out}}\frac{\partial A_1^{out}}{\partial A_1^{in}}\frac{\partial A_1^{in}}{\partial W_1} \tag{2.15}$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{Y}}\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial A_1^{out}}\frac{\partial A_1^{out}}{\partial A_1^{in}}\frac{\partial A_1^{in}}{\partial b_2} \tag{2.16}$$

We see from the second and third term of Equations (2.13), $\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial W_2}$ and the second and third term of Equation (2.15), $\frac{\partial \hat{Y}}{\partial A_2^{in}}\frac{\partial A_2^{in}}{\partial A_1^{out}}$ and the fourth and fifth terms of Equation (2.15), $\frac{\partial A_1^{out}}{\partial A_1^{in}}\frac{\partial A_1^{in}}{\partial W_1}$, that a pattern

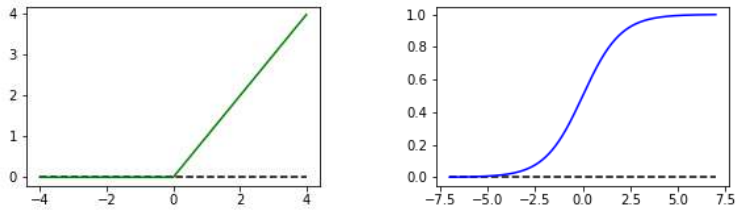
is emerging. The derivative of the input of layer two with respect to the output of layer two is followed by the derivative of the output of layer one with respect to the input of layer one. This pattern will continue for how many layers are in the neural network. We use this pattern to create an algorithm to calculate the gradient with respect to all the parameters in a neural network with any number of layers.

Consider an example of a neural network with I layers - one input, one output and $I-2$ hidden fully connected layers. This neural network will have $I-1$ weight matrices and $I-1$ biases. These weights and biases need to be optimised using backpropagation and stochastic gradient descent. As we can see from Equations (2.13) and (2.14) we have five types of derivatives (see Table 2.3).

$\frac{\partial L}{\partial \hat{Y}}$	=	$\dots$	-	The derivative of the loss function with respect to $\hat{Y}$, this is dependent on the chosen loss function.
$\frac{\partial A_i^{\hat{out}}}{\partial A_i^{in}}$	=	$f_i'(A_i^{in})$	-	The derivative of the output of layer i with respect to the input of layer i.
$\frac{\partial A_i^{in}}{\partial A_{i-1}^{out}}$	=	W_i	-	The derivative of the input of layer i with respect to the output of layer $i-1$.
$\frac{\partial A_i^{in}}{\partial W_i}$	=	A_{i-1}^{out}	-	The derivative of the input of layer i with respect to the weights connecting layer i to $i-1$.
$\frac{\partial A_i^{in}}{\partial B_{i+1}}$	=	1	-	The derivative of the input of layer i with respect to the bias of layer i.

Table 2.3: All possible derivatives which will be needed for the backpropagation algorithm.

Using the derivatives as shown in Table 2.3 and the updating step shown in Equation (2.12) the process to update the weights can be described as shown in Algorithm 1.



(a) (b)

Figure 2.24: Examples of activation function which experiences the vanishing gradient problem. a) Rectified linear unit (ReLu). b) Sigmoid.

We can see that for a function such as ReLu [58] or Sigmoid (see Figure 2.24), if the input value is too small (or too big in the case of Sigmoid) the node's output and gradient will either be very small or zero. This is problematic since if only one of the terms in an equation such as Equation (2.13), is zero or very close to zero the whole term will become zero or very close to zero. This is known as the vanishing gradient problem [25]. In a case such as ReLu all negative valued features are ignored, since this function returns zero for an input of less than zero which will result in a loss of information [71]. A node only returning a zero is known as a dead node and does not contribute to the learning process [47]. Because of this a

Algorithm 1 Neural network optimisation using SGD and backpropagation: Consider a fully connected neural network with I layers. W_i is the matrix of weights connecting layer $i-1$ and i for $i = 2, \ldots, I$. b_j is the bias term for layer $j = 2, \ldots, I$

Required: $r \leftarrow$ *Learning rate.*
Required: $\epsilon \leftarrow$ *Tolerance.*
Required: $f_i \leftarrow$ *Activation function for each layer (derivatives for these functions also required).*
Initial: W_i *for* $i = 2, \ldots, I \leftarrow$ *Initialise values for all weights of the model.*
Initial: b_j *for* $j = 2, \ldots, I \leftarrow$ *Initialise values for all bias terms of the model.*
    While *error* $> \epsilon$ ***do***
        $Chng = 0$
        $X^{(n)}, Y^{(n)} \leftarrow$ *Draw a random sample of size* n *from the data.*
        $\hat{Y}^{(n)} \leftarrow$ *Predict the labels of the sample using current estimates.*
        $PrevDer \leftarrow$ *Calculate the derivative of the loss function with respect to* $\hat{Y}^{(n)}$.
        For $i = I, \ldots, 2$ ***do***
            $dA_{out}dA_{in} = f_i'(A_i^{in})$
            $dA_{in}dA_{out} = W_i$
            $dAdW = A_{i-1}^{out}$
            $PrevDer = PrevDer \times dA_{out}dA_{in} \times dA_{in}dA_{out}$
            $W_i^{new} = W_i^{old} - r \times (PrevDer \times dAdW)$
            $b_i^{new} = b_i^{old} - r \times PrevDer$
            $Chng = Chng + abs(W_i^{new} - W_i^{old}) + abs(b_i^{new} - b_i^{old})$
        $error = Chng$
    Return W_i, b_i *for* $i = 2, \ldots, I \leftarrow$ *optimised weights and biases.*

function such as leaky ReLu [58] was developed which has a non-zero output and gradient for small input values [71]. This will give a node a chance to become non-zero again if it does yield valuable information.

A popular type of neural network used for image classification is a convolutional neural network (CNN) [41]. This network has a predetermined number of convolutional layers. These layers each have a filter of size $n \times m$ where n and m is chosen by hand. An example of a convolutional layer is shown in Figure 2.25. As shown in Figure 2.25 the convolutional filter is essentially a linear combination for data in a 2-dimensional setting. The first value in the feature map is a weighted sum of the first four values in the image using the values of the filter as the weights. The area on which the filter is applied is then moved on. The size of the feature map will be smaller than the input image since multiple values in the input image makes one value in the feature map. The reason these convolutional filters works so well is the fact that the filter's parameters can be optimised. Thus, the neural network will learn the features which are important for the particular task by itself using the provided data.

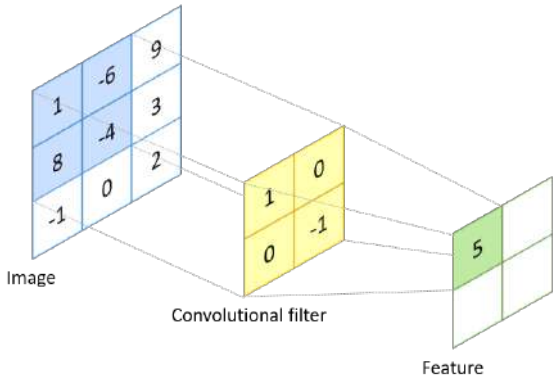


Figure 2.25: Convolutional layer of a convolutional neural network.

The size of the filter depends on the size of the features it should learn, large features will require a larger filter and smaller features would require smaller filters. This filter is then applied to the output of the previous layer. Therefore, a convolutional filter applied to the greyscale image will have a dimension of $n \times m \times 1$ since the image only has one channel. A convolutional layer which is applied to the output of another convolutional layer has filters with a dimension equal to the number of output channels from the previous layer. So, if the previous layer outputs 16 channels the filters will have a dimension of $n \times m \times 16$ where n and m are chosen by the model builder. More than one convolutional filter is usually used together since a large feature can be created from a range of smaller feature [38]. For example, a face is a feature which consists of smaller features such as eyes, nose and mouth, where eyes are also made up of smaller features such as that it's a round shape with most commonly a patch of hair above it. If the filter is the same size as the array it is being applied to, it will return a scalar value. If the filter is smaller than the array it is being applied to, it will return a feature map which is smaller than the initial array. The values which make up this filter are also parameters which can be optimised [33]. A layer commonly

followed by the convolutional layer is the max pooling layer. In this layer a kernel of size $n \times m$ is chosen and applied to the output of the previous layer. This layer moves over the output of the previous layers in blocks and returns the maximum value in this block (see Figure 2.26). Pooling represents the information compactly as well as achieving invariance to transformations on images [10]. Average and min pooling are two other possible layers. Since a greyscale image's pixel values range from 0 (completely black) to 255 (completely white) the max pooling works for extracting information where the feature is light with regards to the background. Min pooling will be useful to extract features which are dark with regards to the background.

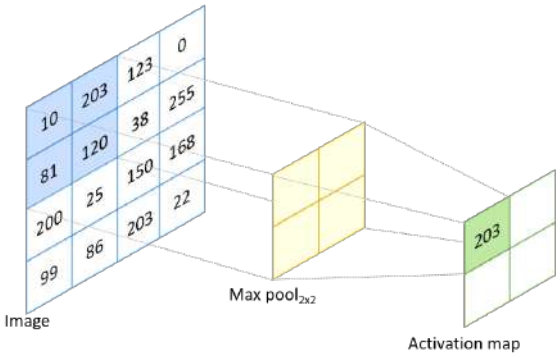


Figure 2.26: An example of max pooling with a 2×2 size kernel being applied to an image.

.

These different pooling strategies are shown in Figure 2.27. Figure 2.27a is an image where the feature is bright and the background dark. In Figure 2.27b max pooling was applied to this image. In Figure 2.27c min pooling was applied to the original image. In Figure 2.27d average pooling was applied. Figure 2.27e is an image where the feature is dark and the background bright. In Figure 2.27f max pooling was applied to this image. In Figure 2.27g min pooling was applied to the original image. In Figure 2.27h average pooling was applied.

As we can see from Figure 2.27a - d when the images have a dark background with bright features the max pooling is more useful to extract information. Furthermore, from Figure 2.27e-h we can see that when the images have a bright background with dark features the min pooling is more useful to extract information.

The reason a CNN is so powerful in image classification is the fact that these convolutional layers allows the network to learn features and structures in the images by itself without need for manually created features [7]. CNNs have been used with great success in various fields. See for example [1, 3, 4, 28, 37, 50].

In this chapter the decomposition algorithm Discrete Pulse Transform was explained from its origin of the LULU operators to a more efficient algorithm called Roadmaker's Pavage which is the algorithm used for the decomposition in this mini-dissertation. The basic knowledge regarding deep learning such

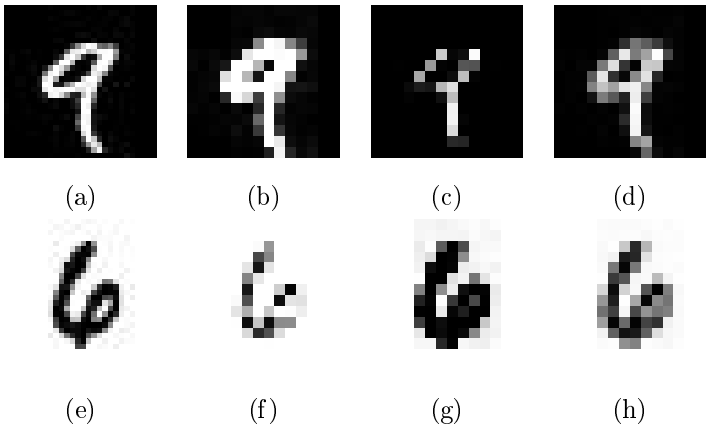


Figure 2.27: Examples of maximum, minimum and average pooling. a and e) The original images. b and f) The images obtained if max pooling with a kernel of size 2×2 is applied to the images. c and g) The images obtained if min pooling with a kernel of size 2×2 is applied to the images. d and h) The images obtained if average pooling with a kernel of size 2×2 is applied to the images.

as the model architecture and optimisation was also discussed. In the rest of the mini-dissertation a proposed method for data augmentation will be given. A CNN will then be trained using the original and augmented data as well as a CNN with the same architecture on only the unaugmented data. These two CNN's performance will then be compared to determine if the proposed method is useful.

Chapter 3

Methodology

When decomposing an image using the DPT, an important question to ask is whether the pulses contain information. To answer this question, we decompose a number of images and then have a look at the distribution of the pulses. As can be seen in Figure 2.9, the number of different shape pulses of a 2-dimensional signal grows exponentially as the size of the pulses increase. Therefore it makes more sense to look at the distribution of two shape features namely compactness and elongation rather than looking at the proportion of every unique shape because as can be seen in Figure 3.1 there are a vast number of shapes of pulses.

3.1 Compactness and Elongation

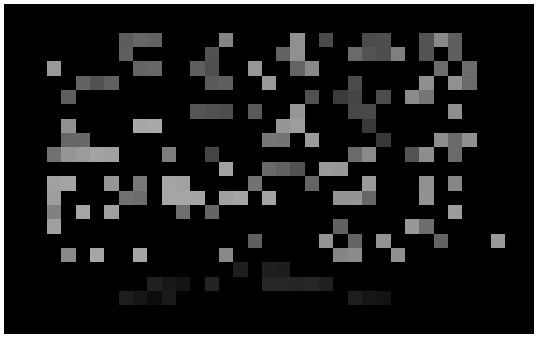


Figure 3.1: Some of the pulses obtained when decomposing an image, showing a number of different pulse shapes which can be obtained.

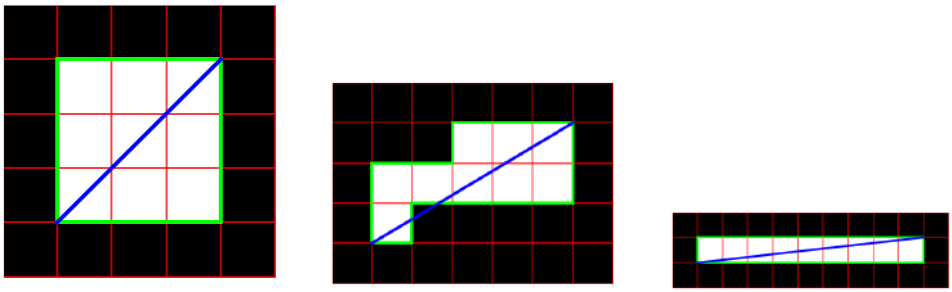
The compactness and elongation [40] of a pulse are defined as,

$$cm = 2\frac{\sqrt{\pi a}}{p},$$
$$eln = \frac{a}{l^2,}$$

where a is the area of the pulse, p is the perimeter and l is the major axis of the polygon which is the length of the longest possible line which can be drawn through the pulse.

Pulses consist of n pixels at scale n. This can be very useful to calculate the perimeter, area and the major axis length. Since pixels are squares with a width and height of one unit, the area of a single pixel is 1 unit^2. Therefore n pixels will have an area of n unit^2 regardless the shape of the connected region of pixels. It is important to note that the size and area of a pixel is relative to the image size and not the objects within the image. Therefore, it is possible for a pixel to have an area of 1 cm^2 or 1 m^2 relative to the object in the image. Regardless of the area of the pixels relative to an object in the image the pixel has an area of 1 unit^2 relative to the image. The perimeter can be calculated using 4-connectivity. For each pixel in the pulse obtain its four neighbours and then the amount contributed to the shapes perimeter is the number of neighbours with a value different from the pixel since all pixels in a pulse have the same height. The length of the major axis of the pulse can be calculated by determining the smallest enclosing circle using the corners of each pixel as all the point to be in the circle. The length of the major axis will be the diameter of this enclosing circle.

The area of the pulses in Figure 3.2 are shown in white. The perimeter is shown as the green line and the length of the major axis as the blue line.



Compactness: 0.886
Elongation: 0.5
(a)

Compactness: 0.665
Elongation: 0.265
(b)

Compactness: 0.532
Elongation: 0.110
(c)

Figure 3.2: Examples of pulses showing different compactness and elongation. a) An example of a very compact pulse. b) A less compacted pulse. c) An example of a very elongated pulse.

Consider the pulses in Figure 3.2. The pulse in Figure 3.2a forms a perfect square this will be a very compact pulse. The pulse shown in Figure 3.2c forms a line-like structure and this will be a very elongated

pulse. In the compactness formula a function of the area is divided by the perimeter of the shape, therefore the bigger the area of the pulse and the smaller the perimeter is the higher the compactness will be. The elongation is the area divided by the length of the major axis in the shape, therefore the smaller the area of the shape and the greater the major axis of the shape is the smaller the elongation will be. The compactness and elongation can be proven to be bounded for pulses.

Consider a shape pulse such as in Figure 3.2c. This pulse forms a line-like structure which is the least compact a pulse can be. Suppose the pulse of this shape consists of n pixels. The area of such a pulse will be n as this pulse consists of n pixels. The perimeter of this pulse is then $2n + 2$. The compactness can then be calculated as,

$$\begin{aligned} cm &= 2\frac{\sqrt{\pi a}}{p} \\ &= 2\frac{\sqrt{\pi n}}{2n+2} \\ &= 2\frac{\sqrt{\pi n}}{n+1}. \end{aligned}$$

Now take the limit as n tends to infinity since the longer the pulse is the less compact it will be, so that $\lim_{n\to\infty} 2\frac{\sqrt{\pi n}}{n+1} = 0$.

Therefore, the lower bound for the compactness of a pulse is 0.

The most compact a pulse can be is a pulse which best represents a perfect circle. The circle will be represented better the more pixels are used. The more pixels are used the more perfect the circle would be. Therefore, as the number of pixels increase, the pulse closer represents a perfect circle with radius r. Therefore the pulse's area and perimeter will tend to that of a circle. Therefore, as n increases the area tends to πr^2 and the perimeter to $2\pi r$. Therefore, the compactness can be calculated as,

$$\begin{aligned} cm &= 2\frac{\sqrt{\pi a}}{p} \\ &= 2\frac{\sqrt{\pi \pi r^2}}{2\pi r} \\ &= \frac{\sqrt{\pi^2 r^2}}{\pi r} \\ &= 1. \end{aligned}$$

Therefore, the upper bound for the compactness of a pulse is 1. Thus $cm \in (0, 1)$

Consider a shape pulse such as in Figure 3.2c. This pulse forms a line-like structure which is the most elongated a pulse can be. Consider a pulse of this shape consisting of n pixels. The area of such a pulse

will be n as this pulse consists of n pixels. The major axis of the shape can be calculated using Pythagoras' theorem since this axis will run from the one corner of a pixel to another in opposite corners of the pulse. Therefore, the squared length of the major axis is $l^2 = n^2 + 1$. The elongation can then be calculated as,

$$\begin{aligned} eln &= \frac{a}{l^2} \\ &= \frac{n}{n^2+1}. \end{aligned}$$

Taking the limit as n tends to infinity of this shape pulse, since the longer the pulse is the more elongated it will be yields, $\lim_{n\to\infty} \frac{n}{n^2+1} = 0$. Therefore, the lower bound for elongation of a pulse is 0.

Consider a pulse which most closely represents a perfect circle. This is the least elongated a pulse can be. As n increase the pulse would better represent a circle with radius r. The shape's area tends to πr^2 and the squared length of the major axis will tend to $4r^2$. The elongation is then calculated as,

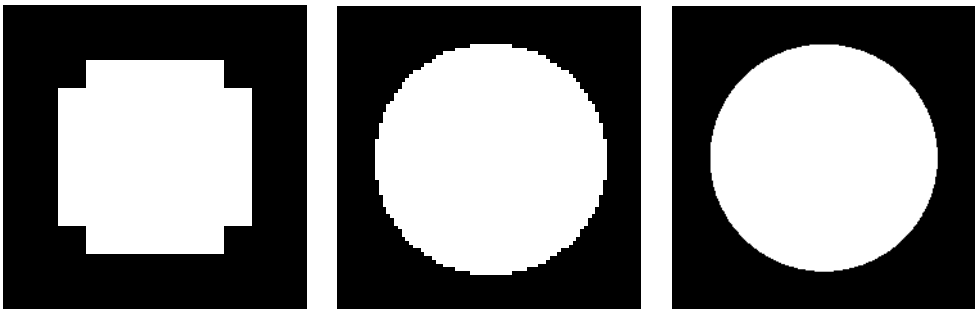
$$\begin{aligned} eln &= \frac{a}{l^2} \\ &= \frac{\pi r^2}{4r^2} \\ &= \frac{\pi}{4} \end{aligned} \tag{3.1}$$

Therefore, the upper bound for elongation is $\frac{\pi}{4}$.

Thus $eln \in (0, \frac{\pi}{4})$.

These bounds do not include the end points since a perfect circle cannot be obtained using pixels. As we can see in Figure 3.3 as the number of pixels increase we get a better circle with more pixels being used but a perfect circle is not obtainable. Also, n can become very large yielding values close to the end point but will never be exactly this value.

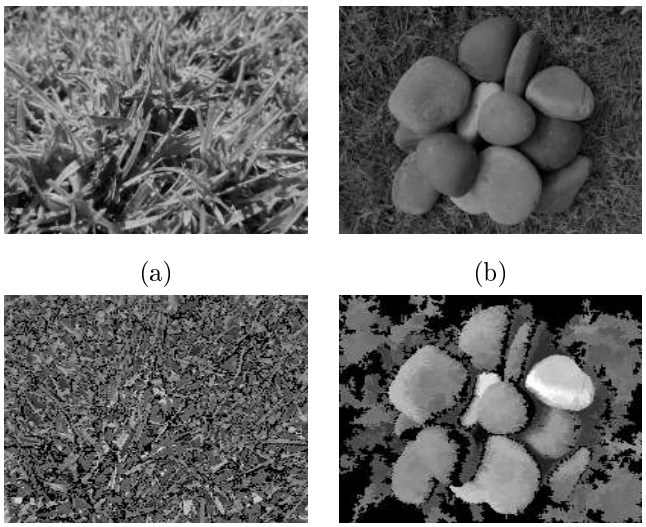
A simple example will now be done to show that the pulses of the DPT decomposition contains useful information. In this example an image of grass (Figure 3.4a) and an image of rocks (Figure 3.4b) is decomposed using DPT. Grass is made up of many small blades and therefore it would make sense to use smaller scales to represent the image. Rocks on the other hand are large structures and are not made up of many smaller structures but rather larger features. Therefore, it would make sense to use large scales when reconstructing the image as more pixels are needed to represent these larger features. The scales chosen for the image of grass is 1 to 100 and the scale chosen for the rocks is 100 to 5000. To better visually represent the extracted pulses the values of the entire image containing the pulses were rescaled into the range 10 to 255. A greyscale image cannot have negative valued pixels and it is possible to obtain negative pulses and this is why the values are rescaled. The values which were 0 in images containing the



(a) (b) (c)

Figure 3.3: Examples of circles created using pixels for different numbers of pixels to show a perfect circle cannot be obtained using pixels. a) A circle made of 45 pixels. b) A circle made of 2 933 pixels. c) A circle made of 283 325 pixels.

extracted pulses were kept as zero. The lower bound of the rescaled values are chosen as 10 because if it is chosen as 0 the pulses with the smallest values will become 0 and will not be visible in the image. The rescaling is only done for visual purposes.



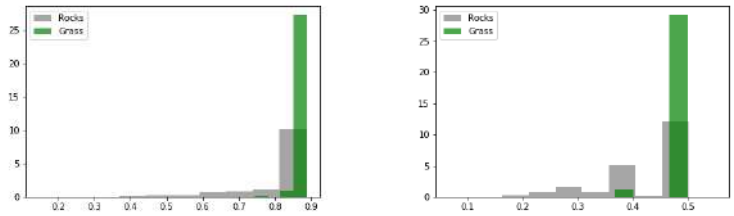
(a) (b)

(c) (d)

Figure 3.4: An example of DPT pulses extracted from images of grass and rocks. a) The original image of grass. b) The original image of rocks. c) The extracted pulses of scale 1 to 100 of the image of grass. d) The extracted pulses of scale 100 to 5000 of the image of rocks.

For each of the images pulses of different scales were extracted. For each pulse in each image the compactness and elongation were calculated. The distribution of the compactness and elongation metrics are shown in Figure 3.5.

We can see that the densities of the compactness and elongation for both images are concentrated are more or less the same location. But it can be seen that for the pulses extracted from the image of grass



(a) (b)

Figure 3.5: a) The distribution of the compactness of the pulses for an image of grass and an image of rocks. b) The distribution of the elongation of the pulses for an image of grass and an image of rocks.

most of the pulses has a very high compactness (between 0.8 and 0.886). The pulses extracted from the image of rocks also has very high compactness but the compactness is spread out a lot more than for the image of grass. The same can be seen for the elongation distribution. To compare the spreads of the compactness and elongation the coefficient of variation was calculated. These metrics can be seen in Table 3.1. The coefficient of variation for both compactness and elongation of the images of rocks is very high. This indicated that the values of these metrics for all the pulses are spread out. The coefficient of variation of the compactness and elongation measures of the image of grass is low indicating that the values are more concentrated and closer to the mean.

	Grass	Rocks
Compactness	0.0134	0.1381
Elongation	0.0404	0.1922

Table 3.1: The coefficient of variation for the extracted pulses from images of grass and rocks for compactness and elongation.

The purpose of the DPT decomposition is to extract salient features within in the image. Visual saliency is a broad term which is discussed by Kadir and Brady in [31] as the fact that in a scene there are some regions which pop-out. This means that there are parts of the image which differ from its surroundings which can be defined as a feature. Features can differ from their surroundings with regards to a number of attributes for example shape and colour. Examples of visual saliency can be seen in Figure 3.6.

The type of visual saliency focused on when decomposing an image using DPT is the difference based on colour such as Figure 3.6a. This is since the pulses are the difference between the feature (local minimum/maximum set) and its closest neighbours. If a certain region of an image is present in multiple scales of the decomposition it is intuitive that this region is an important feature, since it forms part of multiple features. The scale of the pulse also contributes to the saliency since if a feature differs from its neighbours in colour it is a more prominent feature. Therefore, it would be useful if we could have a look at each pixel in an image and determine some sort of metric to represent its saliency.

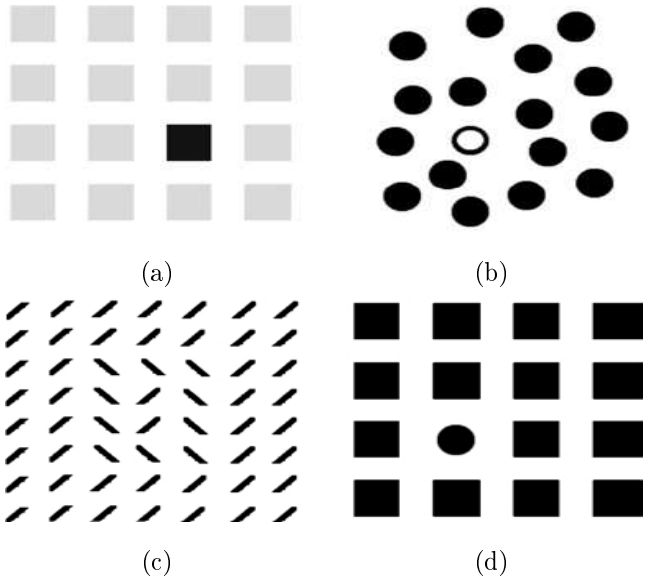


Figure 3.6: An example visual saliency based on different attributes [31]. a) The feature differs from its neighbours based on colour. b) The feature differ from its neighbours based on density. c) The features differs from its neighbours based on orientation. d) The feature differs from its neighbours based on shape.

A possible metric we can use to represent each pixel's saliency is the Discrete Pulse Vectors (DPV) [17].

3.2 Discrete pulse vectors

Each pixel i in the image belongs to k_i pulses $\{\phi_1, \phi_2, \ldots, \phi_{k_i}\}$ in the DPT decomposition. These pulses have scales $\{n_1, n_2, \ldots, n_{k_i}\} \subseteq \{1, 2, \ldots, N\}$ [17]. A vector can thus be constructed of all the scales of pulses to which a pixel belongs as $p_i = [\phi_1, \phi_2, \ldots, \phi_{k_i}]$, $i \in \mathbb{Z}^2$. A feature's saliency should increase proportionally to the number of pulses in which it is present. The saliency should also increase proportionally to the scale of the pulses in which it is present. For example an element with a DPV of $p_1 = [-1]$ is less likely to be an important feature than an element with a DPV of $p_2 = [10, 158, 20]$. The lengths of these DPV are $|p_1| = \sqrt{(-1)^2} = 1$ and $|p_2| = \sqrt{10^2 + 158^2 + 20} = 159.6$. We can thus conclude that the length of the DPV can be used to determine which pixels form part of an important feature. Consider again the 2×2 greyscale image decomposed in Figure 2.11a. We will now determine their DPVs as well as the length of these vectors. Pixel one (top left-hand pixel with a value of 255) was present in a pulse of size one and scale 153, a pulse of size two and scale 60 and a pulse of size 4 and scale 42. The DPV P_1 is [153, 60, 42]. All the DPVs and their lengths are shown in Table 3.2. Therefore, we can conclude that the feature represented by pixel one would be the most prominent feature within the image.

Using the image in Figure 2.20 which was decomposed using DPT, the Discrete Pulse Vectors and the length of these vectors of each pixel in the image were determined. All the pixels which had a DPV length

	Vector	Length
P_1	$[153, 60, 42]$	$\sqrt{153^2 + 60^2 + 42^2} = 169.6$
P_2	$[60, 42]$	$\sqrt{60^2 + 42^2} = 73.2$
P_3	$[42]$	$\sqrt{42^2} = 42$
P_4	$[42]$	$\sqrt{42^2} = 42$

Table 3.2: The Discrete Pulse Vectors and their lengths of each pixel of the image in Figure 2.11a which was decomposed using DPT.

greater than the median DPV length were extracted from the image and the rest of the pixels were made zero. The resulting images are shown in Figure 3.7a.

(a) (b) (c)

Figure 3.7: The images obtained when the pixels with a specific DPV length are extracted. a) The original image. b) Pixels with DPV lengths greater than the median DPV length are extracted and the rest of the pixels set to zero. c) Pixels with DPV lengths in the upper 10% of DPV lengths are extracted and the rest of the pixels set to 0.

We can see from Figure 3.7b that the DPVs with lengths greater than the median length extract the important features in the image. In Figure 3.7c only the pixels with DPV lengths in the upper 10% is extracted. We see that few pixels are extracted but all these pixels form part of the cat structure which is the important part in the image. In the image in Figure 3.7c only 4055 of the 40000 pixels were extracted and we can still see the cat in the image. This shows that the longer the DPV vectors are the more important feature it represents in the image, thus measuring saliency.

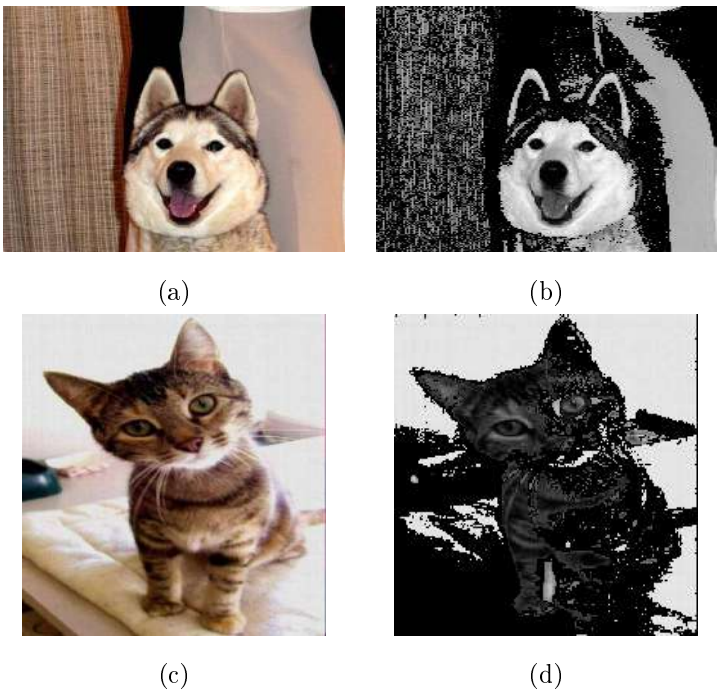
3.3 Proposed augmentation

The proposed data augmentation is given in Algorithm 2. In Figure 3.8 examples of images as well as the augmented images is shown. From the images in Figure 3.8 we can see that important features in the images such as the cat and dog is still present and less important features such as the background is not.

The DPT algorithm is edge preserving [2], therefore the edges of the object are also still visible in the augmented image.

Algorithm 2 Proposed data augmentation

1: *Decompose each image using the DPT algorithm.*
2: *For each pixel in each image, determine the DPV and its length.*
3: *For each image determine the median DPV length.*
4: *For each image, extract the pixels with DPV length greater than the median DPV length.*
5: *For each image, the extracted pixels' original values are obtained while the rest of the pixels are set to zero. This is the augmented image to add to the training set.*



(a) (b) (c) (d)

Figure 3.8: Examples of the proposed data augmentation applied to images. a) An image of a dog. b) The image of a dog after the proposed data augmentation has been applied. c) An image of a cat. d) The image of a cat after the proposed data augmentation has been applied.

In this chapter it was investigated whether the pulses extracted when using the DPT decomposition yield valuable information of the structures in the image. Two shape measures namely compactness and elongation were used to investigate if the shapes of the pulses differ from images with different size features. The DPVs of the pixels in the image were discussed and calculated. The proposed data augmentation method using the DPVs of each pixel was also shown and an example of this augmentation on example images given. The DPVs extract important structures in the image.

Chapter 4

Application

The code used in this chapter can be found on GitHub at https://github.com/JP0201/MSc-Multiscale-image-representation-in-Deep-Learning-code.git.

In this chapter we investigate whether the use of DPT decomposition is useful as a method of data augmentation in a deep learning model. The objective is to train a model which can correctly classify an image as containing either a dog or a cat. The dataset used was obtained from a Kaggle competition[1]. This is a dataset consisting of 25 000 labelled images of different sizes. There are 12 500 images of cats and 12 500 images of dogs. An example of a cat and dog image from the dataset is shown in Figure 4.1. All the images are resized to 200×200 resolution and converted to greyscale to train the model. Each pixel then has a single value between 0 and 255. These values will be divided by 255, for a range between zero and one which is easier to work with in a neural network.

Dog Cat

Figure 4.1: Two example images from the Kaggle dataset being used.

[1] Data available at https://www.kaggle.com/c/dogs-vs-cats/data

The network architecture followed is the so called SmallNet proposed by Wang and Perez in [65]. This architecture was shown to perform well with this dataset [65] and has 3 convolutional layers, two of which directly follow each other (layers 4 and 5). With this the network extracts useful information present in the data. The architecture of the neural network is given below.

Input : Gray scale image of size 200×200 pixels (40 000 input nodes).

1 : Convolutional layer with 16 channels, 3×3 filter size and ReLu activation.

2 : Batch normalisation.

3 : Max pool layer, 2×2 filter size and strides of 2×2.

4 : Convolutional layer with 32 channels, 3×3 filter size and ReLu activation.

5 : Convolutional layer with 32 channels, 3×3 filter size and ReLu activation.

6 : Batch normalisation.

7 : Max pool layer, 2×2 filter size and strides of 2×2.

8 : Fully connected layer with 1024 nodes, ReLu activation and Dropout with a rate of 0.5.

Output : Fully connected layer with two nodes and SoftMax activation.

Neural networks trained on a small amount of data run the risk of overfitting the data [61]. This is addressed by dropout. A dropout with a rate of 0.5 applied to a layer means that with each training step of the neural network, 50% of this layer's nodes are randomly dropped-out. Therefore, the network is updated without these nodes and all of the weights associated with them. This changes the architecture of the neural network in each step and therefore prevents the neural network from overfitting easily [61]. As an added benefit, dropout reduces the computation required to update the model parameters as these nodes are not present in the model in the update step.

Batch normalisation re-scales and re-centres the inputted data [66]. This is done by subtracting the mean and dividing by the square root of the variance plus some term ϵ which helps stabilize the term when the variance is very small. This normalisation is done along each dimension of the data. This is done for each drawn sample in each update step.

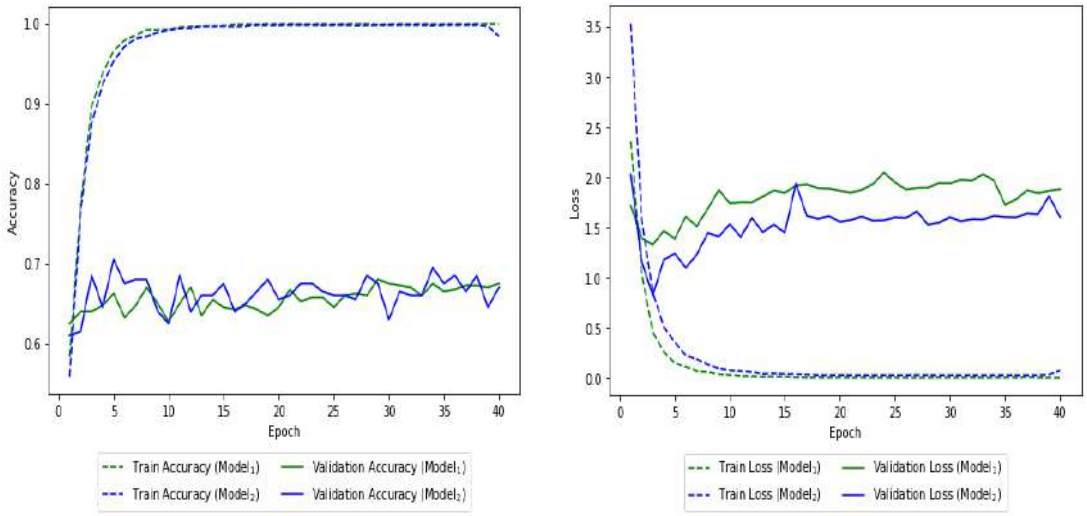
The model is optimised using the Adam optimiser with a learning rate of 0.0001 with 40 epochs [65]. The Adam optimiser is an extension of stochastic gradient descent and is derived from the term adaptive moment estimation. This algorithm uses estimates for the first and second moments of the gradient of the loss to update the parameters [32]. The loss metric used is binary cross entropy. Binary cross entropy loss is used, as each observation can fall into one of two classes.

$$L(Y, \hat{Y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i))$$

where y_i is the actual label of observation i and can be either zero or one and $p(y_i)$ is the predicted class probability for observation i.

Two models are trained. The first model (model1) is trained on a subsample of the data due to the computational intensity of the DPT algorithm and to investigate how this method performs in a scenario where little data is available. The sample consists of 1000 images of dogs and 1000 images of cats. The second model (model2) is trained on the same as well as the augmented images created as explained in Algorithm 2. Therefore, model2 is the model using the proposed data augmentation. These extra images are only used for the training of the model and not the evaluation, the evaluation is only done on the original images. The datasets are split into a training (700 of each), testing set (200 of each) and validation set (100 of each). The validation set is used to evaluate the model after each update step of the model. The testing set is used for an unbiased measure of performance since it is unseen data. The images used in the training and validations sets' decomposed images are not used in the training set as this will lead to biased results.

Both models training accuracy and losses as well as the validation accuracy and losses for the 40 epochs of optimisation are shown in Figure 4.2.



(a) (b)

Figure 4.2: The training and validation accuracy and loss of both models. a) Training (dotted lines) and validation (solid lines) accuracy for model1 (green lines) and model2 (blue lines). b) Training (dotted lines) and validation (solid lines) loss for model1 (green lines) and model2 (blue lines).

From the training accuracy and loss plots in Figure 4.2, it can be seen that the model only using the

unaugmented data performs better based on the training sample and seems to converge quicker. However, looking at the validation accuracy plot (see Figure 4.2a solid lines) we can see that the models seem to perform more or less the same with model2 outperforming model1 by a small margin. Looking at the validation loss we can see that the loss of model2 is consistently lower than that of model1.

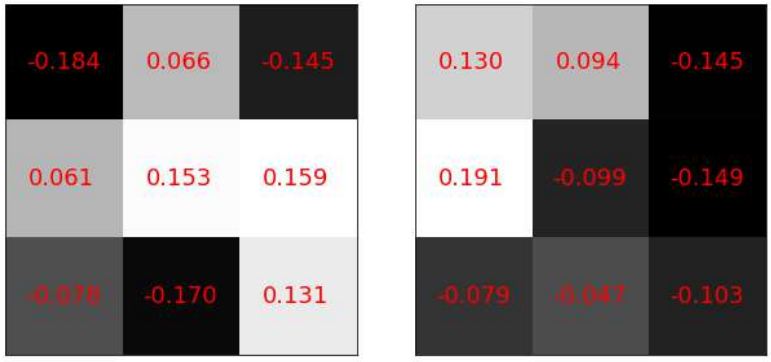
Therefore, it seems that model2 generalises better than model1. Thus, it seems that using the proposed data augmentation does assist the neural network learn the features in the images better.

Testing these images on a test set which it has never seen results in an accuracy of 67% for both model1 and model2. The testing loss is 1.843 for model1 and 1.797 for model2. Therefore, it would seem that model2 does perform better. The loss for model2 is lower since for the observations it is predicting incorrectly the probabilities of being predicted into the correct class is bigger although it is still smaller than the incorrect class. The larger probability for the correct class is good since the observation is classified into the class with the largest probability.

If we extract the predicted probabilities of the observations having the label one (dog) and determine a value $c \in [0, 1]$ when the predicted probability is greater than c the predicted class is one and if not the predicted class is zero. Then using the validation set and these probabilities we can determine an optimal value for c. We use the validation set to ensure the results obtained for the test set is unbiased. The value for c in model1 is 0.94 which results in a validation accuracy of 65.5%. The value of c for model2 is 0.91 which results in a validation accuracy of 67.5%.

Each convolutional layer has a number of convolutional filters. For example, the first layer of the model has 16 channels. Each of these channels require a convolutional filter each of which learns to pick up important features in the image. These filters can be visualised to see which features they pick up. The values of each of the elements of the filter are unbounded therefore to visualise it on a greyscale we rescaled the values such that all the values of the filters are between 0 and 255. Some examples of each model's filters are shown in Figure 4.3 and the actual value of each element is shown. The convolutional filters in the second and third convolutional layer have a dimension for each of the input channels. Therefore, the second layer's filters will have a dimension of $3 \times 3 \times 16$ and the third layer's filter $3 \times 3 \times 32$. These filters cannot be visualized due to the high dimensionality but the feature maps obtained from the layer can be visualized.

From Figure 4.3a we can see that this filter learned to pick up both vertical and horizontal edges. Edges of objects are visible in images due to the colour difference. For example, a dark object with a light background. The dark part will have low pixel values and the light part high pixel values. The filter having a high number such as 0.159 next to a low number such as -0.145 will pick up an edge since this will either result in a very high or very low value depending on if the object is dark with a light background or vice versa. From Figure 4.3b we can see that this particular filter learned to pick up diagonal edges



(a) (b)

Figure 4.3: Visual representation of the filters trained by the two models. a) A filter from the first convolutional layer of model2 (proposed data augmentation). b) A filter from the first convolutional layer of model1.

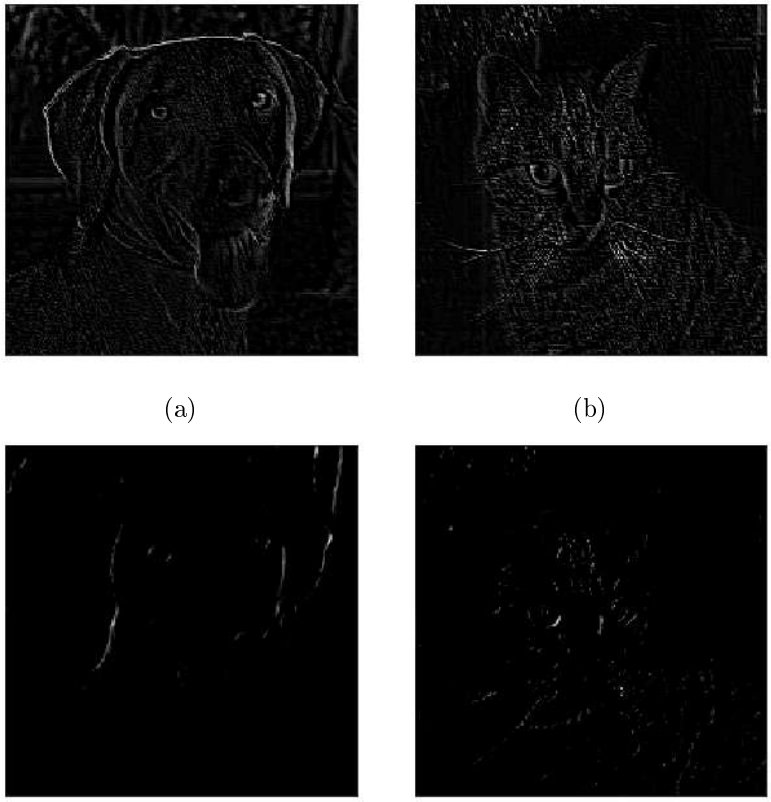
due to the arrangement of the low and high filter values. The DPT decomposition is edge preserving [2], which means that after decomposition and partial or full reconstruction the edges in the image are not lost. As can be seen from the examples in Figure 4.4, the model using the proposed data augmentation picks up the edges of the objects better than the model that only uses the unaugmented images.

Each filter in a convolutional layer is applied to the input data and generates a feature map. Therefore, a feature map is generated for each channel in the convolutional layer. The input data for the first convolutional layer is the image after batch normalisation was done. The input data for the second and third convolutional layers is the feature maps obtained from previous layer. Examples of these obtained feature maps are shown in Figure 4.4.

In Figure 4.4a and b, the feature maps obtained from applying the filter in Figure 4.3a to both images in Figure 4.1 is shown. This is one of the output channels of the first convolutional layer of model2. The small vertical and horizontal edges picked up by the filter yields a feature map were the general shape of the object is extracted well.

In Figure 4.4c and d, the feature maps obtained from applying the filter in Figure 4.3b to both images in Figure 4.1 is shown. This is one of the output channels of the first convolutional layer of model1. The diagonal edges picked up by the filter is visible. Although the feature maps do not represent the image very well it is important to note that this feature map along with the other feature maps is passed to further convolutional layers which will use these small extracted larger features [38].

All the convolutional filters and feature maps obtained for these two models using the two images in Figure 4.1 as input can be seen in the Appendix.



(a) (b)

(c) (d)

Figure 4.4: The obtained feature maps by the two models. a) A feature map obtained after after applying the filter in Figure 4.3a to the images in Figure 4.1a. b) A feature map obtained after applying the filter in Figure 4.3a to the images in Figure 4.1b. c) A feature map obtained after applying the filter in Figure 4.3b to the images in Figure 4.1a. d) A feature map obtained after applying the filter in Figure 4.3b to the images in Figure 4.1b

In this chapter the proposed data augmentation method was used to train a CNN to discriminate between images of cats and dogs. Various measures of performance were used to compare this model to a model trained on only the unaugmented image. The model using the proposed data augmentation perform better when looking at testing accuracy (using the predicted probabilities with a custom cut off value to classify the images) and loss as well as validation loss. Some of the filters learned by the two models and the feature maps they generate was also shown and discussed.

Chapter 5

Discussion and Conclusion

In this mini-dissertation the use of DPT decomposition as data augmentation for a deep learning model was investigated. When looking at the trained models (one with only the unaugmented data and one with the proposed data augmentation as well) the model trained using the data augmentation performs better than the model without augmentation. The features map which are generated by the model trained on the augmented data as well does represent the images better. The DPT being a edge preserving decomposition algorithm seems to help the model pick up the edges of the objects in the image. The aims set out in the beginning of the mini dissertation were all met. The DPT was used to extract some features of the images. These extracted features of the images did aid in the training process of the deep learning model. Therefore, it seems that the proposed data augmentation is useful in a deep learning scenario where large data sets are not available. The decomposition of these images is quite computationally intensive. The optimisation of these deep learning models are already quite computationally intensive so adding this does lead to longer training time. The DPT could be applied to different images in parallel cutting down on the computational intensity however. The advantage of using the Roadmaker's Pavage algorithm is the fact that the decomposed image is stored as a graph object, therefore once the decomposition is done once different scales and the DPVs can be extracted with ease. The proposed data augmentation method was only compared to a training only on the original image. The method was not compared to other data augmentation techniques. Looking at the feature maps shown in Figure 4.4 it looks like the DPT decomposition does help to train these filters to learn the features present in the images.

In this mini-dissertation, we achieved the following aims,

- Use of the DPT decomposition to represent images at different scales to extract/highlight features in the image.
- Investigate whether the images which extract these features are useful in improving training of a

deep learning model.

- Investigate whether the DPT and its DPVs can be a useful data augmentation method in limited data cases.

These aims were achieved through the method proposed to use the extracted pulses in order to extract/highlight features in the image. When comparing the models (one trained with the proposed data augmentation and one without) the model using the augmentation does perform better in a scenario where large data sets are not available.

In this mini-dissertation this data augmentation was only tested on one specific neural network architecture. The use of this data augmentation approach can be investigated for more architectures and different classification scenarios.

The data augmentation method can be compared to other data augmentation methods.

It can also be investigated if various shape measures such as compactness and elongation can be used as additional input to the neural network to extract further shape and scale information from the images as has been shown successful in [63]. Thiede et al. [63] show that they are useful measures of the pulses extracted using the DPT and this could further enhance a small data set. Herein we only investigate their bands and leave their implementation for future research.

This proposed data augmentation technique for training a convolutional neural network shows promise in training with structure and scale in mind. It is a first step in combining the Discrete Pulse Transform with deep learning and shows that the multiscale decomposition extracts meaningful constructs from training of a convolutional neural network.

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[2] Roumen Anguelov and Inger Fabris-Rotelli. LULU operators and Discrete Pulse Transform for multidimensional arrays. *IEEE Transactions on Image Processing*, 19(11):3012–3023, 2010.

[3] Manfredo Atzori, Matteo Cognolato, and Henning Müller. Deep learning with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands. *Frontiers in Neurorobotics*, 10:9, 2016.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[5] Saikat Basu, Supratik Mukhopadhyay, Manohar Karki, Robert DiBiano, Sangram Ganguly, Ramakrishna Nemani, and Shreekant Gayaka. Deep neural networks for texture classification: A theoretical analysis. *Neural Networks*, 97:173–182, 2018.

[6] Yoshua Bengio and Montreal CA. RMSprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*, 2015.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[8] Manish H Bharati, J Jay Liu, and John F MacGregor. Image texture analysis: methods and comparisons. *Chemometrics and Intelligent Laboratory Systems*, 72(1):57–71, 2004.

[9] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991.

[10] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118, 2010.

[11] Lars Bretzner and Tony Lindeberg. Qualitative multiscale feature hierarchies for object tracking. *Journal of Visual Communication and Image Representation*, 11(2):115–129, 2000.

[12] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, and Andrea Vedaldi. Deep filter banks for texture recognition, description, and segmentation. *International Journal of Computer Vision*, 118(1):65–94, 2016.

[13] Mark De Lancey and Inger Nicolette Fabris-Rotelli. Effective graph sampling of a nonlinear image transform. In *Proceedings of FAIR 2019*, volume 2540, pages 185–195. Cape Town, 3-6 December 2019.

[14] Stuart E Dreyfus. Artificial neural networks, back propagation, and the kelley-bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, 13(5):926–928, 1990.

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[16] Michael Egmont Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks: a review. *Pattern Recognition*, 35(10):2279–2301, 2002.

[17] Inger Fabris-Rotelli. The Discrete Pulse Transform for images with entropy-based feature detection. In *Proceedings of the 22nd Annual Symposium of the Pattern Recognition Association of South Africa*, pages 43–48, 2011.

[18] Inger Fabris-Rotelli and Alfred Stein. Inhomogeneous spatial modelling of DPT pulses for marine images. *Spatial Statistics*, 28:257–270, 2018.

[19] Inger Fabris-Rotelli and Stefan J Van der Walt. The Discrete Pulse Transform in two dimensions. *Pattern Recognition Association of South Africa*, 2009.

[20] Inger Nicolette Fabris-Rotelli. *Discrete Pulse Transform of images and applications*. PhD thesis, University of Pretoria, 2013.

[21] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[22] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[23] Georgia Gkioxari, Bharath Hariharan, Ross Girshick, and Jitendra Malik. R-CNNs for pose estimation and action detection. *arXiv preprint arXiv:1406.5212*, 2014.

[24] Sumit Goyal and Gyandera Kumar Goyal. Simulated neural network intelligent computing models for predicting shelf life of soft cakes. *Global Journal of Computer Science and Technology*, 11(14):29–33, 2011.

[25] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 582–591, 2018.

[26] Maria Dorothea Jankowitz. *Some statistical aspects of LULU smoothers*. PhD thesis, University of Stellenbosch, 2007.

[27] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[28] Min Ji, Lanfa Liu, Runlin Du, and Manfred F Buchroithner. A comparative study of texture and convolutional neural network features for detecting collapsed buildings after earthquakes using pre- and post-event satellite imagery. *Remote Sensing*, 11(10):1202, 2019.

[29] Luyang Jing, Ming Zhao, Pin Li, and Xiaoqiang Xu. A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox. *Measurement*, 111:1–10, 2017.

[30] Radiša Ž Jovanović, Aleksandra A Sretenović, and Branislav D Živković. Ensemble of various neural networks for prediction of heating energy consumption. *Energy and Buildings*, 94:189–199, 2015.

[31] Timor Kadir and Michael Brady. Saliency, scale and image description. *International Journal of Computer Vision*, 45(2):83–105, 2001.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Jayanth Koushik. Understanding convolutional neural networks. *arXiv preprint arXiv:1605.09081*, 2016.

[34] Andrew F Laine, Sergio Schuler, Jian Fan, and Walter Huda. Mammographic feature enhancement by multiscale analysis. *IEEE Transactions on Medical Imaging*, 13(4):725–740, 1994.

[35] Dirk P Laurie. The Roadmaker's algorithm for the Discrete Pulse Transform. *IEEE Transactions on Image Processing*, 20(2):361–371, 2010.

[36] Dirk P Laurie and Carl H Rohwer. Fast implementation of the Discrete Pulse Transform. In *In Proceedings of theInternational Conference of Numerical Analysis and Applied Mathematics*, pages 15–19. Weinheim, Germany, 2006.

[37] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[39] Hong-Ze Li, Sen Guo, Chun-Jie Li, and Jing-Qi Sun. A hybrid annual power load forecasting model based on generalized regression neural network with fruit fly optimization algorithm. *Knowledge-Based Systems*, 37:378–387, 2013.

[40] Mengmeng Li, Alfred Stein, Wietske Bijker, and Qingming Zhan. Region-based urban road extraction from VHR satellite images using binary partition tree. *International Journal of Applied Earth Observation and Geoinformation*, 44:217–225, 2016.

[41] Wei Li, Guodong Wu, Fan Zhang, and Qian Du. Hyperspectral image classification using deep pixel-pair features. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2):844–853, 2016.

[42] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[43] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(1-2):225–270, 1994.

[44] Tony Lindeberg. *Scale-space theory in computer vision*, volume 256. Springer Science & Business Media, 2013.

[45] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1):7–27, 2001.

[46] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[47] Carles Roger Riera Molina and Oriol Pujol Vila. Solving internal covariate shift in deep learning with linked neurons. *arXiv preprint arXiv:1712.02609*, 2017.

[48] Ranjan Parekh. Using texture analysis for medical diagnosis. *IEEE MultiMedia*, (2):28–37, 2010.

[49] Frédéric Patin. An introduction to digital image processing. *online]: http://www. programmersheaven. com/articles/patin/ImageProc. pdf*, 2003.

[50] Md Mahbubar Rahman, MAH Akhand, Shahidul Islam, Pintu Chandra Shill, and MH Rahman. Bangla handwritten character recognition using convolutional neural network. *International Journal of Image, Graphics and Signal Processing (IJIGSP)*, 7(8):42–49, 2015.

[51] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[52] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7263–7271, 2017.

[53] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[54] CH Rohwer and DP Laurie. The Discrete Pulse Transform. *SIAM Journal on Mathematical Analysis*, 38(3):1012–1034, 2006.

[55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[56] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[57] Jean Serra. A lattice approach to image segmentation. *Journal of Mathematical Imaging and Vision*, 24(1):83–130, 2006.

[58] Sagar Sharma. Activation functions in neural networks. *Towards Data Science*, 6, 2017.

[59] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[60] Grahame Smillie. *Analogue and digital communication techniques*. Elsevier, 1999.

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

[62] George Gene Stoltz. Roadmakers pavage, pulse reformation framework and image segmentation in the Discrete Pulse Transform. Master's thesis, University of Pretoria, 2014.

[63] RN Thiede, IN Fabris-Rotelli, Alfred Stein, Pravesh Debba, and M Li. Uncertainty quantification for the extraction of informal roads from remote sensing images of south africa. *South African Geographical Journal*, 102(2):249–272, 2020.

[64] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[65] Jason Wang, Luis Perez, et al. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*, 11, 2017.

[66] Shui-Hua Wang, Chaosheng Tang, Junding Sun, Jingyuan Yang, Chenxi Huang, Preetha Phillips, and Yu-Dong Zhang. Multiple sclerosis identification by 14-layer convolutional neural network with batch normalization, dropout, and stochastic pooling. *Frontiers in Neuroscience*, 12:818, 2018.

[67] Xin Wang, Yi Qin, Yi Wang, Sheng Xiang, and Haizhou Chen. Reltanh: An activation function with vanishing gradient resistance for SAE-based DNNs and its application to rotating machinery fault diagnosis. *Neurocomputing*, 363:88–98, 2019.

[68] Eric Wu, Kevin Wu, David Cox, and William Lotter. Conditional infilling GANs for data augmentation in mammogram classification. In *Image Analysis for Moving Organ, Breast, and Thoracic Images*, pages 98–106. Springer, 2018.

[69] Hang Yu, Aishan Liu, Xianglong Liu, Gengchao Li, Ping Luo, Ran Cheng, Jichen Yang, and Chongzhi Zhang. PDA: Progressive data augmentation for general robustness of deep neural networks. *arXiv preprint arXiv:1909.04839*, 2019.

[70] Mohamed M Zahra, Mohamed H Essai, and Ali R Abd Ellah. Performance functions alternatives of MSE for neural networks learning. *International Journal of Engineering Research & Technology (IJERT)*, 3(1):967–970, 2014.

[71] Xiaohu Zhang, Yuexian Zou, and Wei Shi. Dilated convolution neural network with LeakyReLU for environmental sound classification. In *2017 22nd International Conference on Digital Signal Processing (DSP)*, pages 1–5. IEEE, 2017.

[72] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*, pages 8778–8788, 2018.

[73] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019.

Appendix

Model1

Convolutional filters

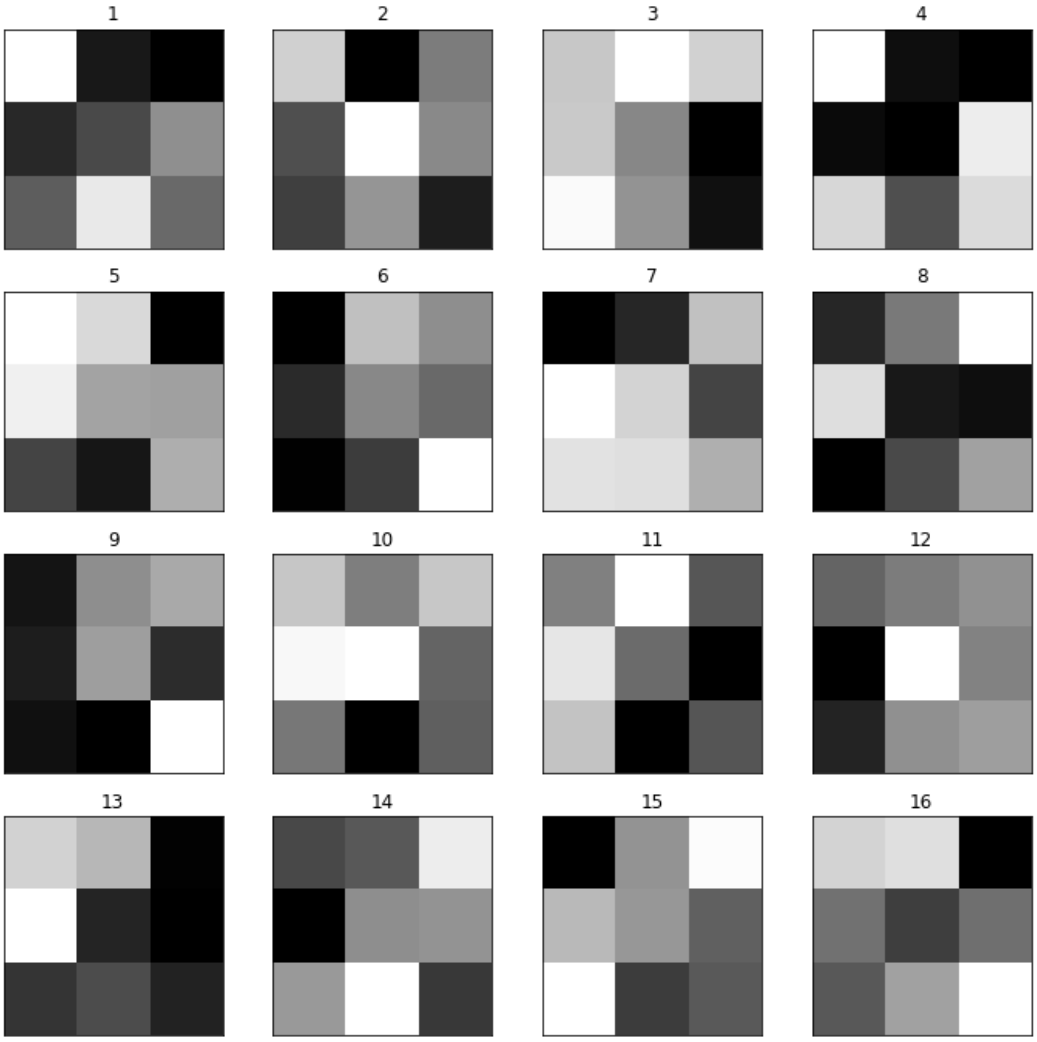


Figure 5.1: Convolutional filters of the first convolutional layer in model1.

Feature maps

Cat

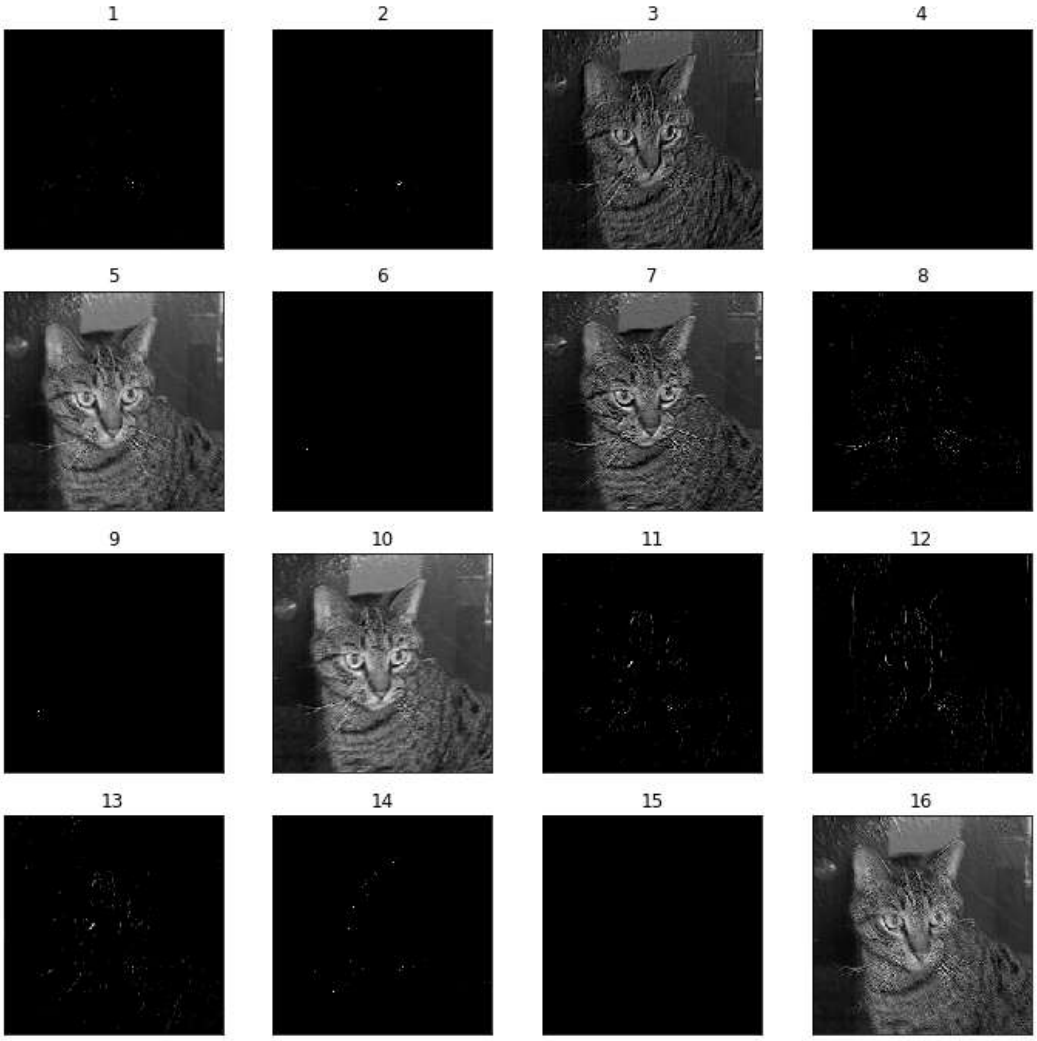


Figure 5.2: The 16 output feature maps after the first convolutional layer in model1 using an image of a cat as input to the model.

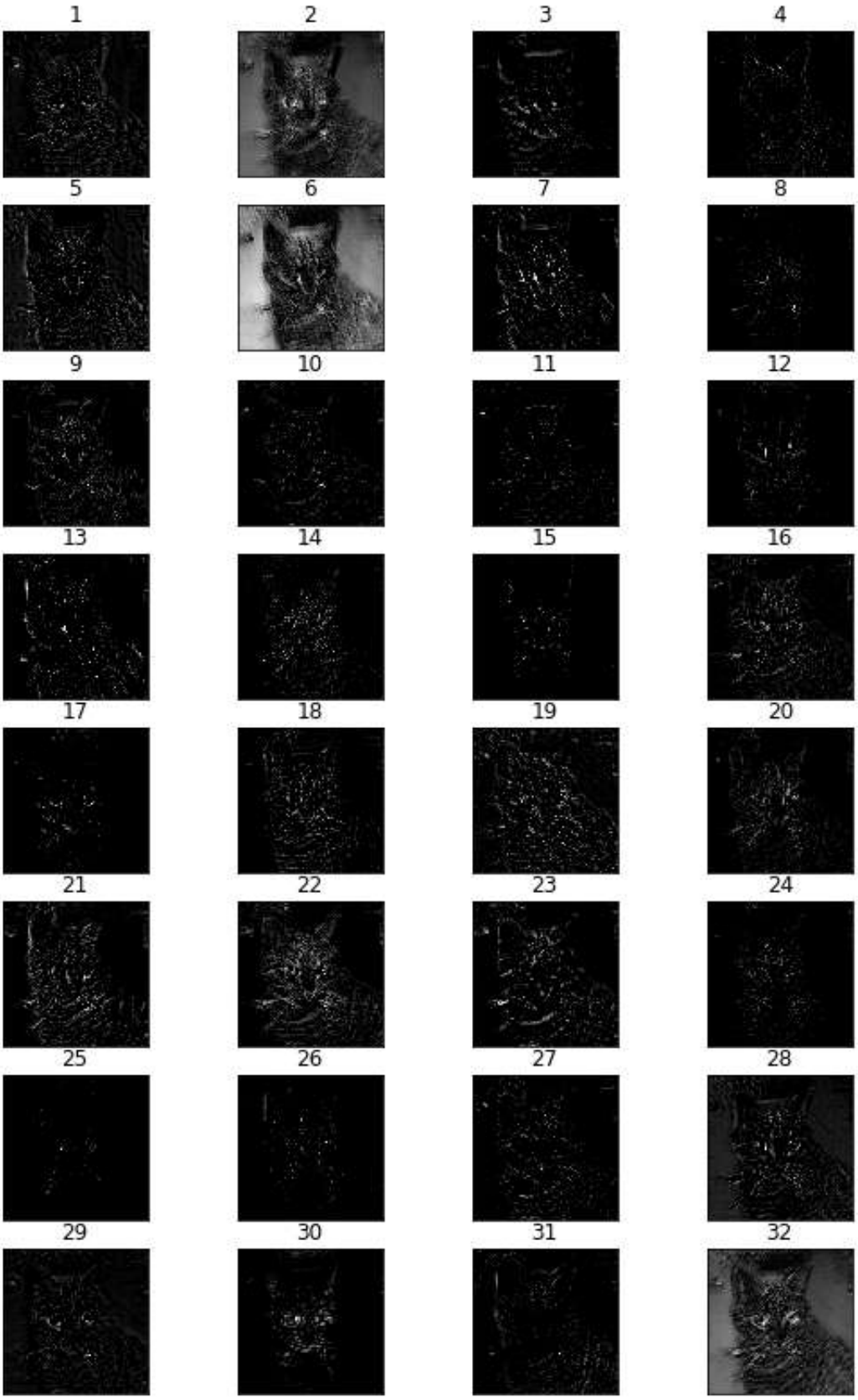


Figure 5.3: The 32 output feature maps after the second convolutional layer in model1 using an image of a cat as input to the model.

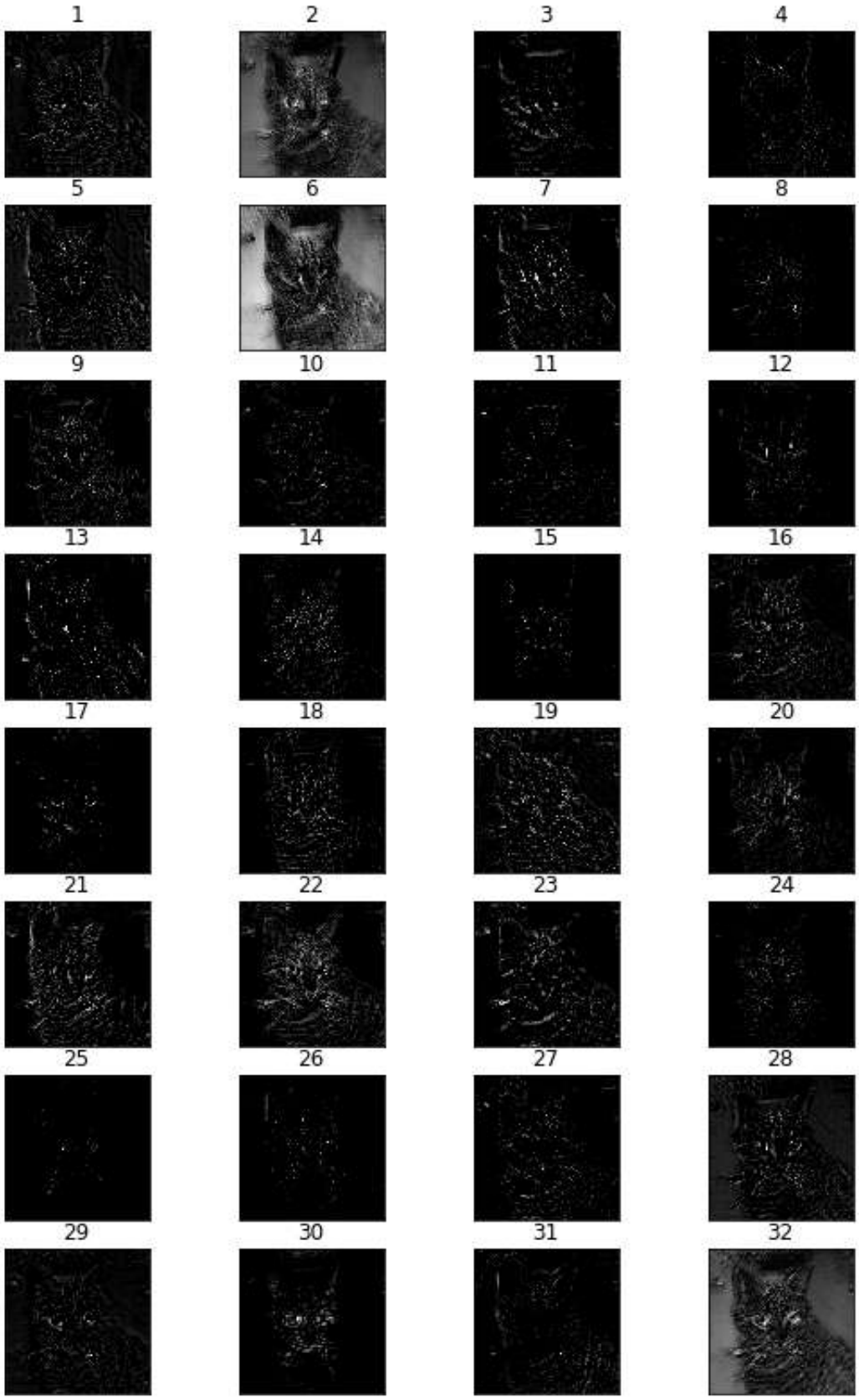


Figure 5.4: The 32 output feature maps after the third convolutional layer in model1 using an image of a cat as input to the model.

Dog

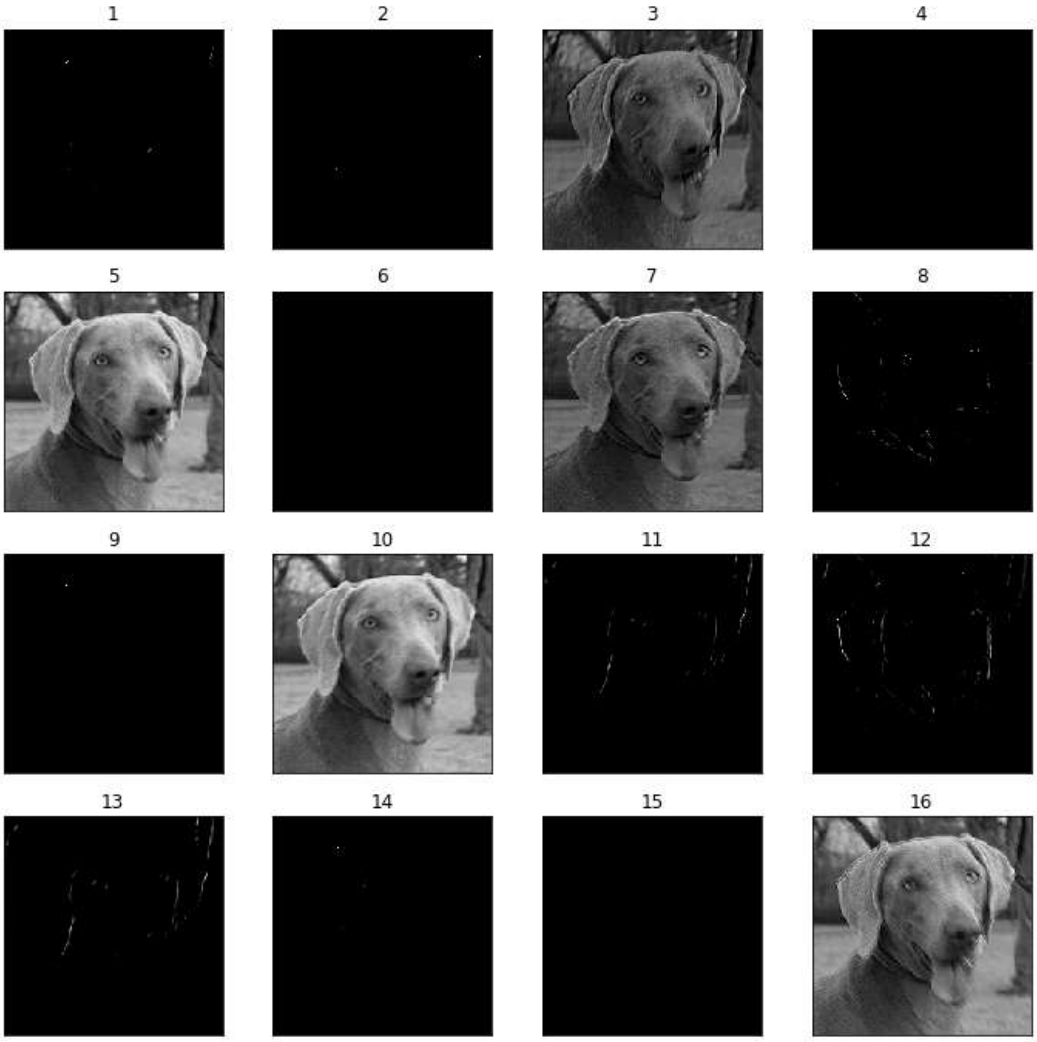


Figure 5.5: The 16 output feature maps after the first convolutional layer in model1 using an image of a dog as input to the model.

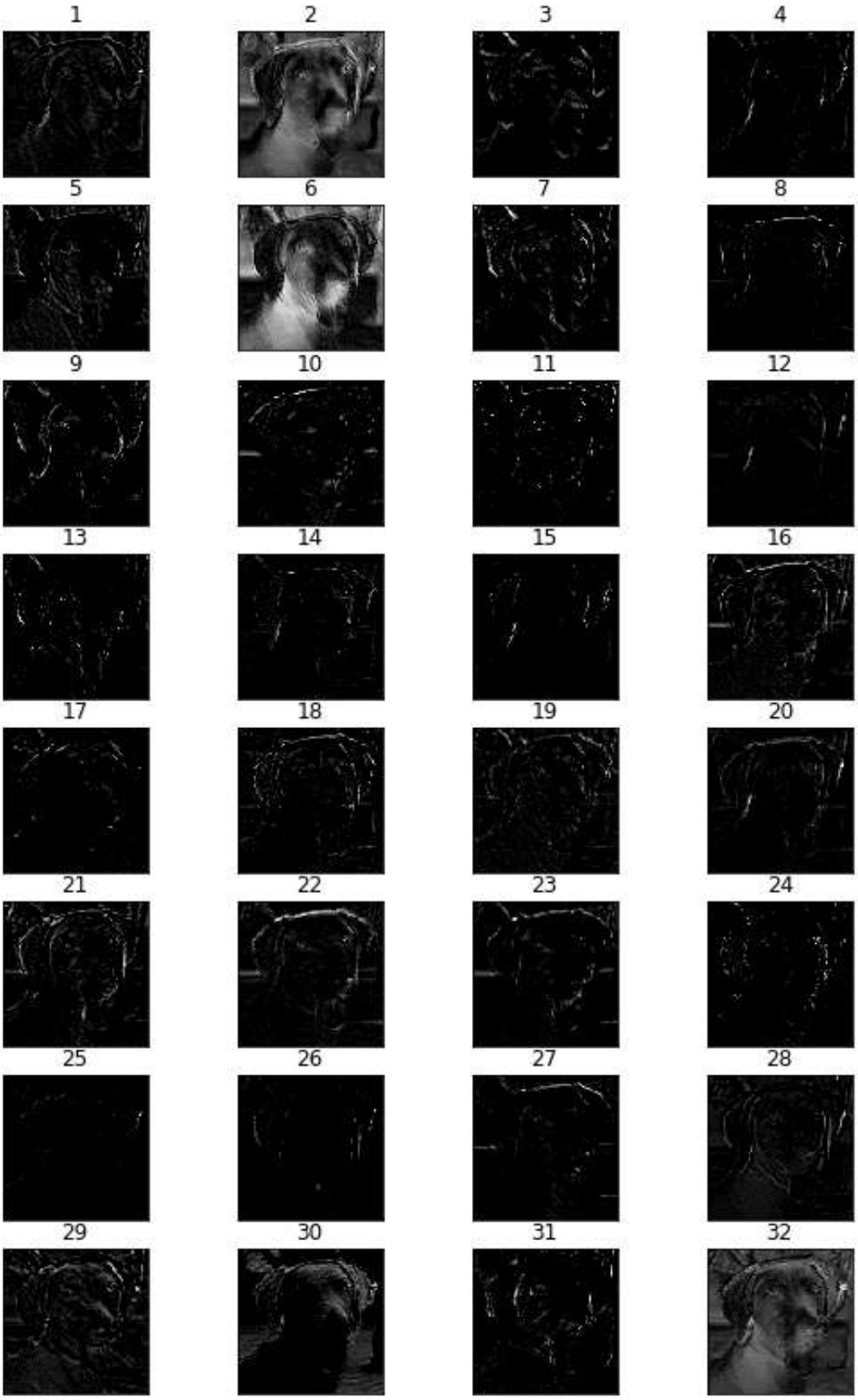


Figure 5.6: The 32 output feature maps after the second convolutional layer in model1 using an image of a dog as input to the model.

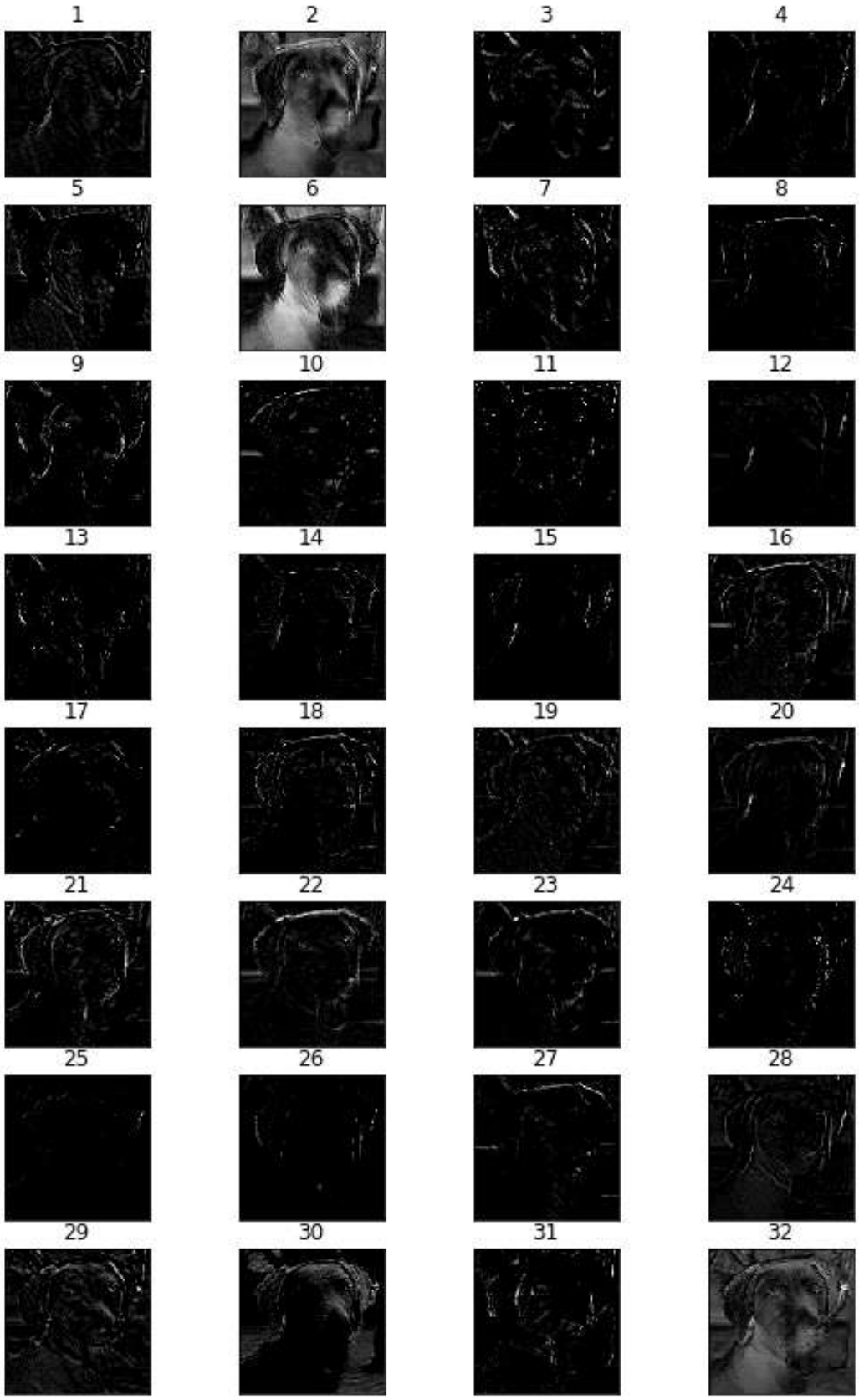


Figure 5.7: The 32 output feature maps after the third convolutional layer in model1 using an image of a dog as input to the model.

Model2

Convolutional filters

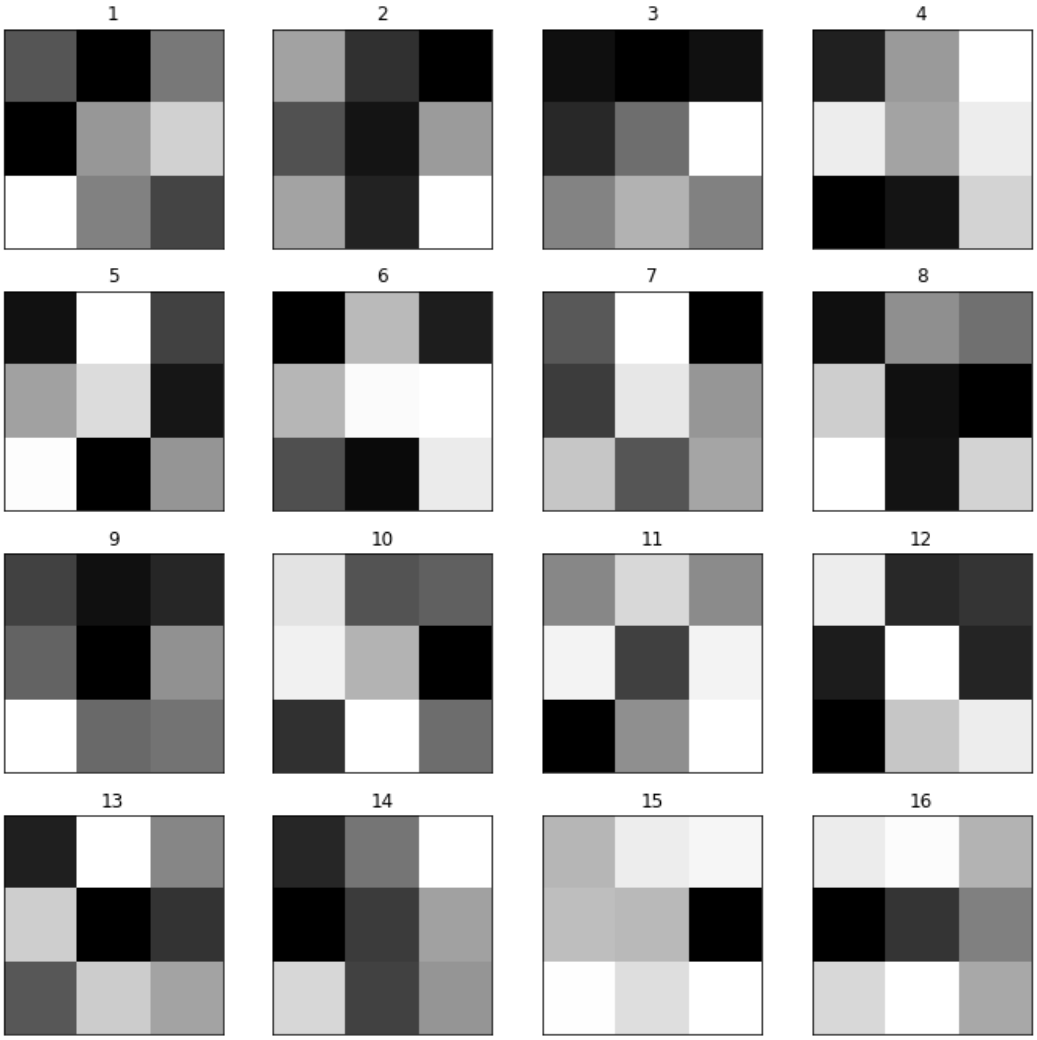


Figure 5.8: Convolutional filters of the first convolutional layer in model2.

Feature maps

Cat

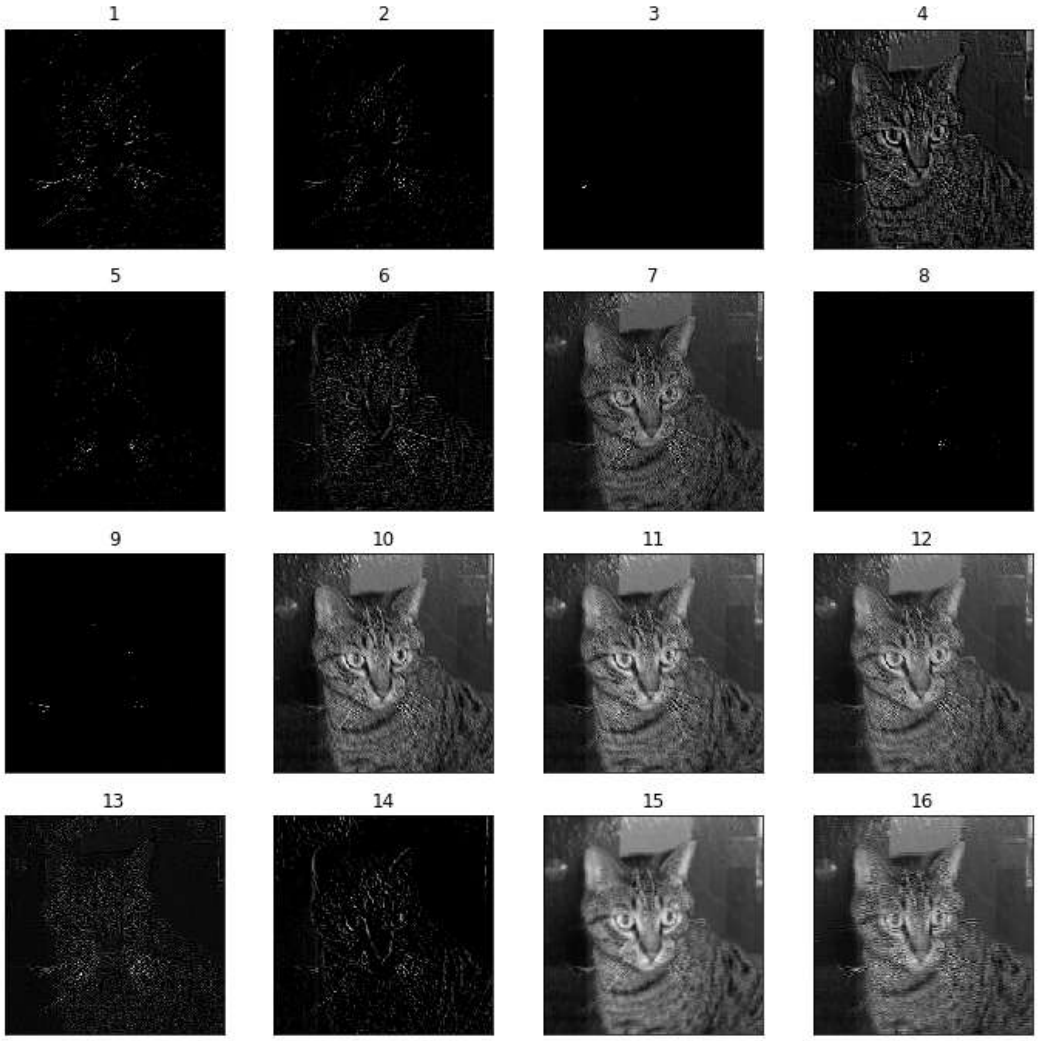


Figure 5.9: The 16 output feature maps after the first convolutional layer in model2 using an image of a cat as input to the model.

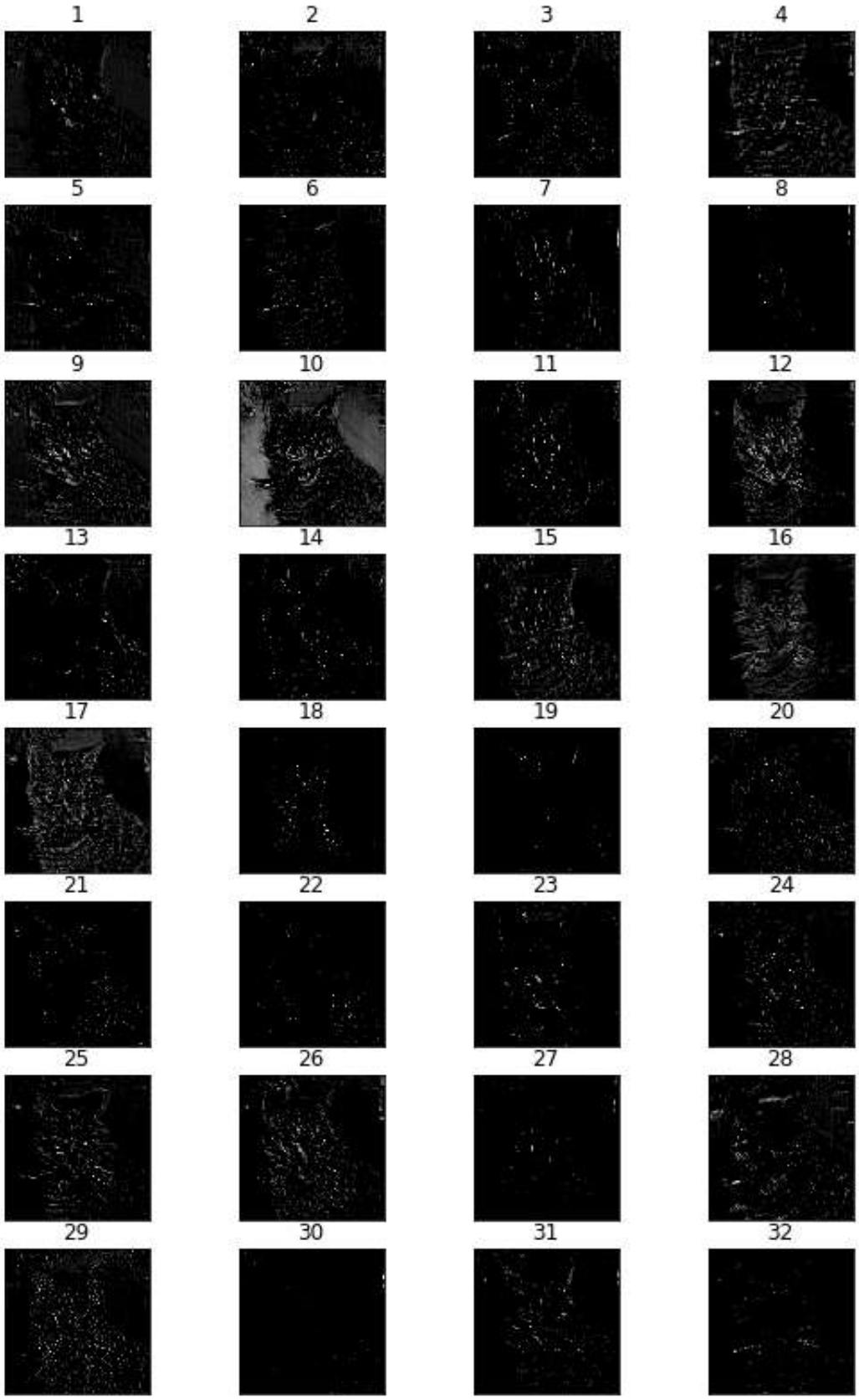


Figure 5.10: The 32 output feature maps after the second convolutional layer in model2 using an image of a cat as input to the model.

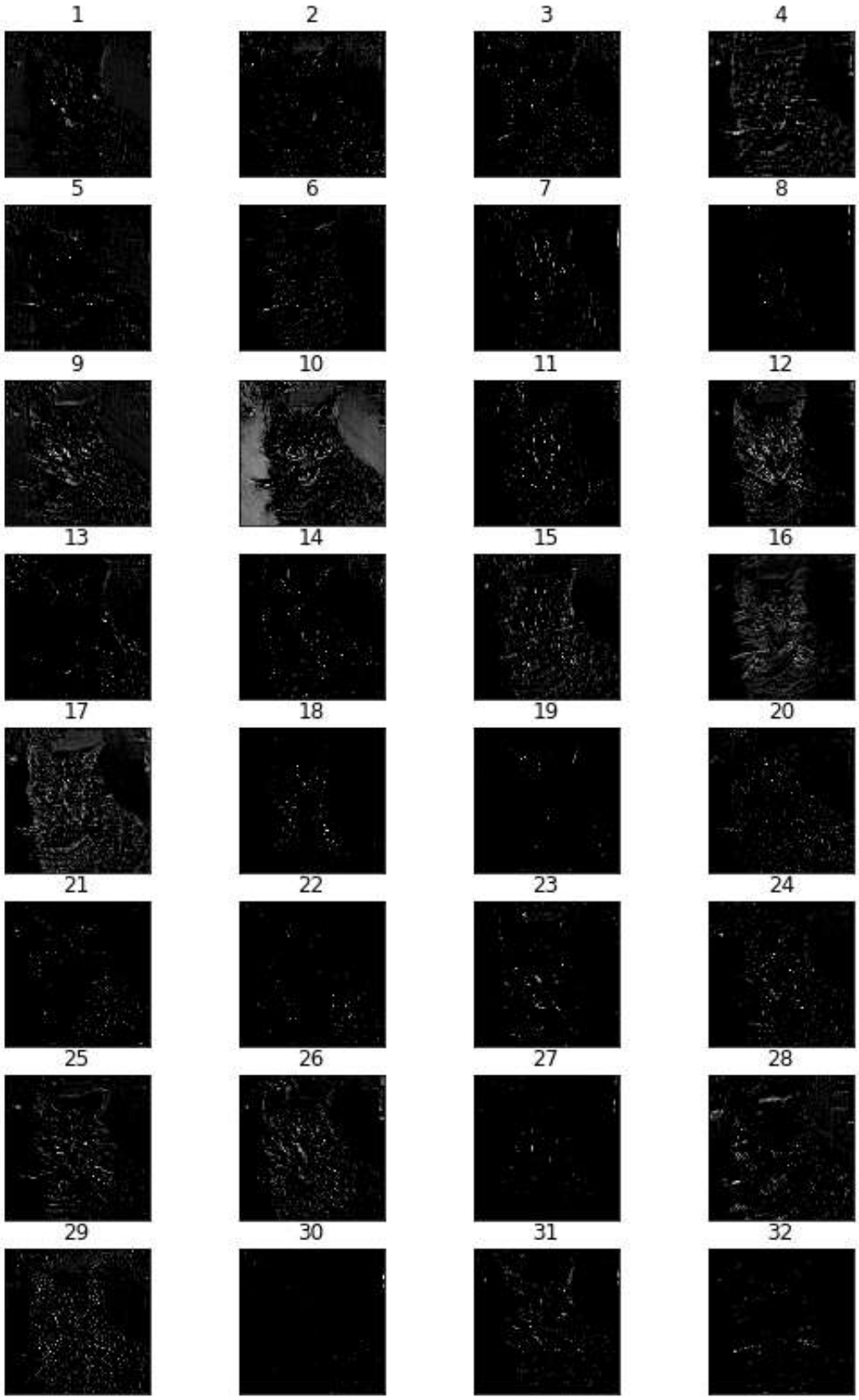


Figure 5.11: The 32 output feature maps after the third convolutional layer in model2 using an image of a cat as input to the model.

Dog

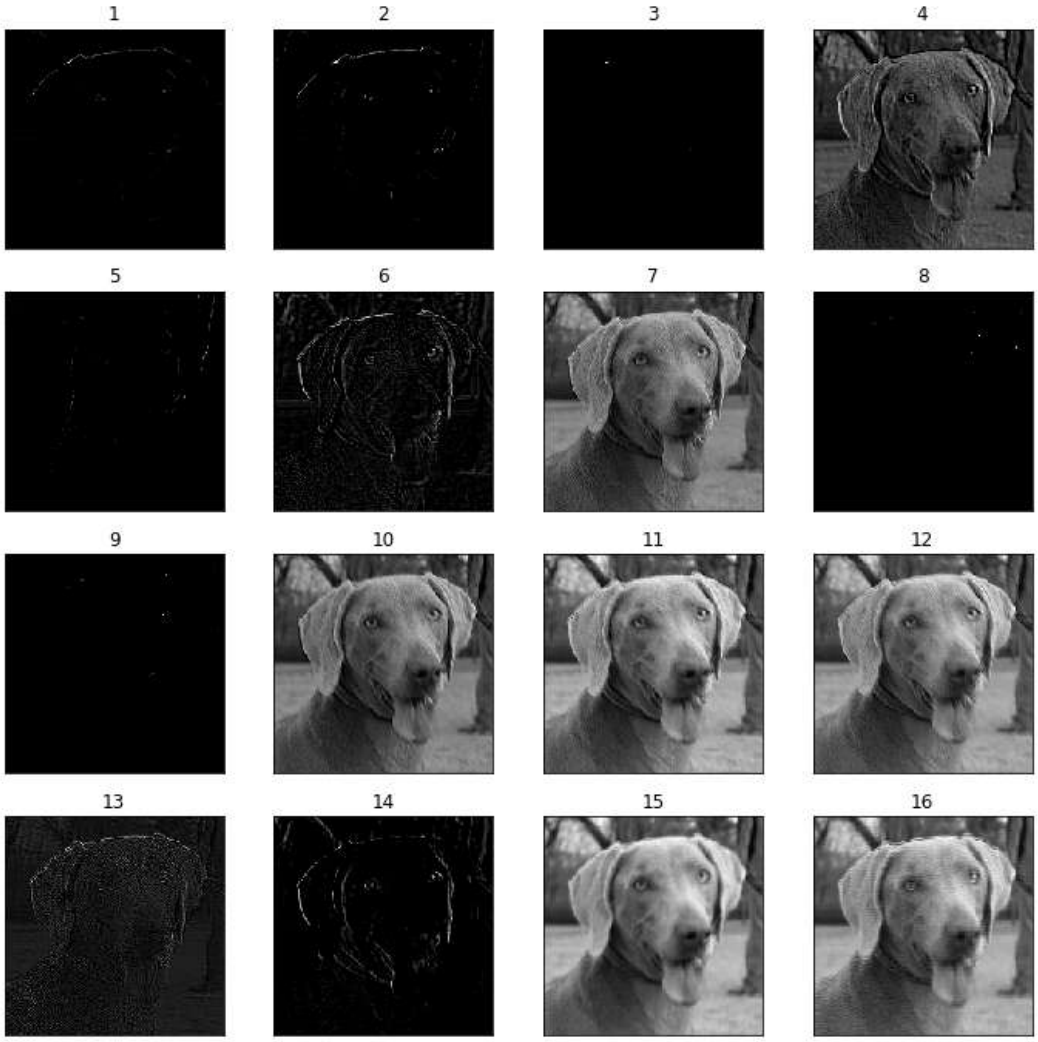


Figure 5.12: The 16 output feature maps after the first convolutional layer in model2 using an image of a dog as input to the model.

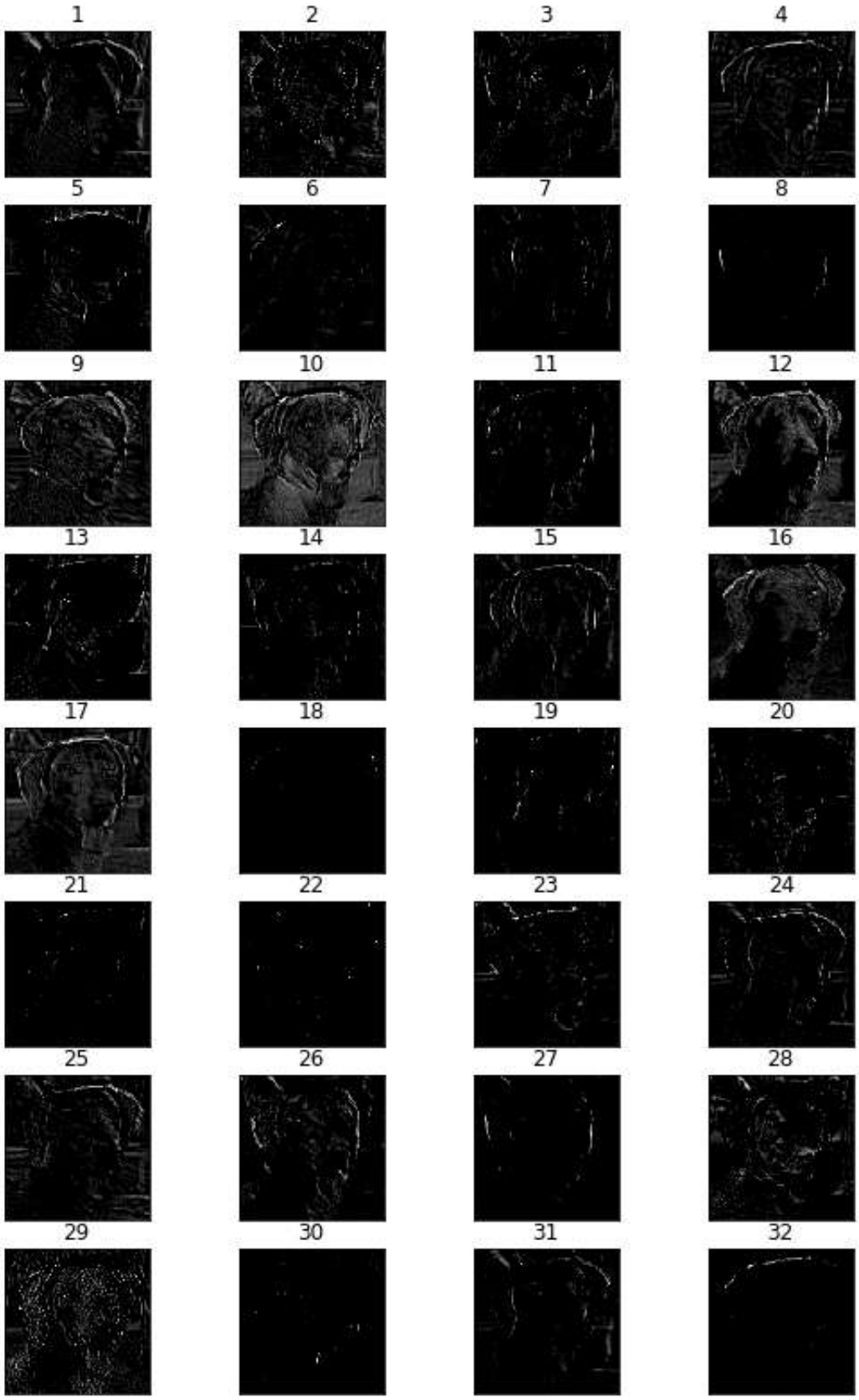


Figure 5.13: The 32 output feature maps after the second convolutional layer in model2 using an image of a dog as input to the model.

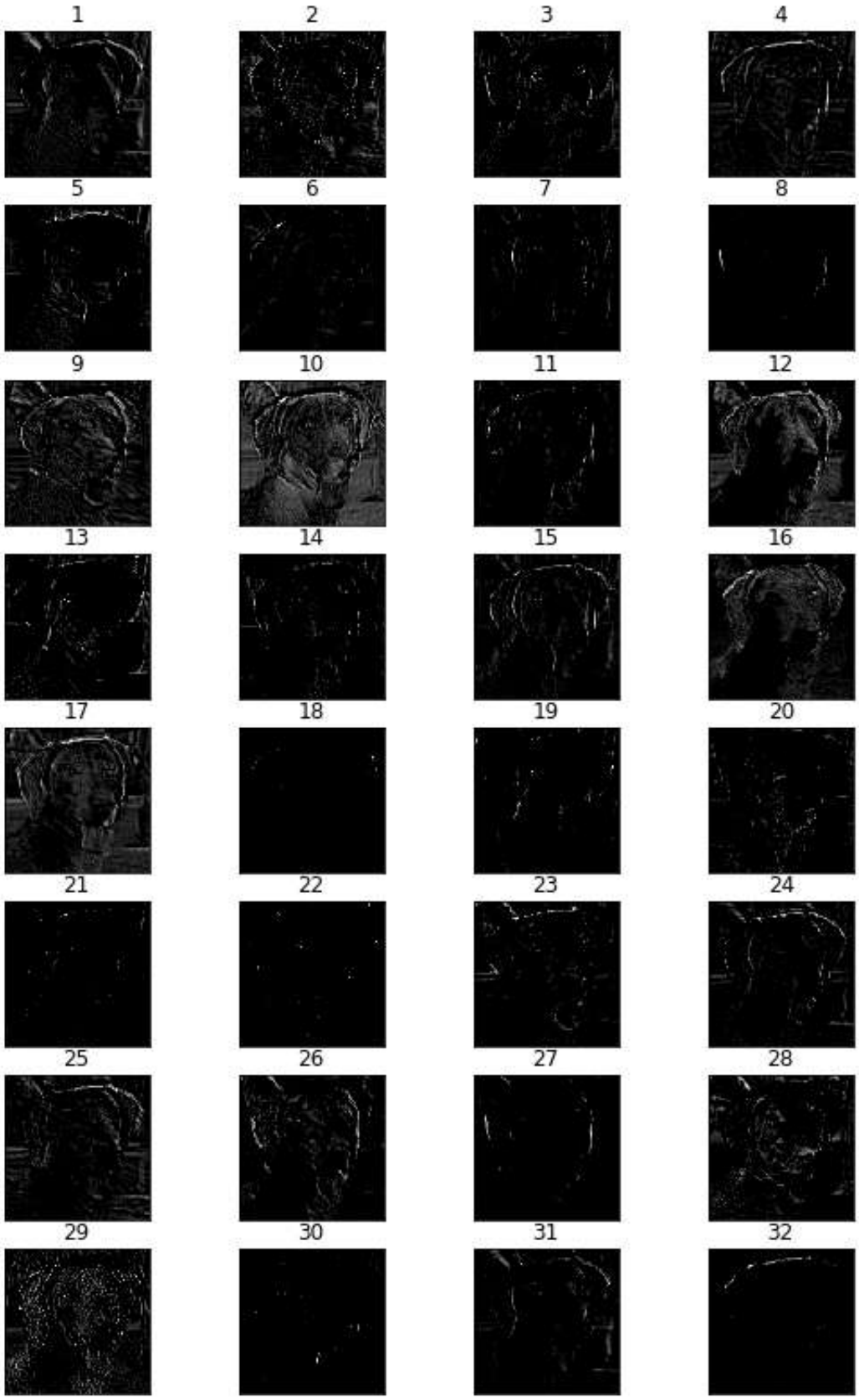


Figure 5.14: The 32 output feature maps after the third convolutional layer in model2 using an image of a dog as input to the model.