Stockholms universitet

Unveiling the inner mechanisms of deep convolutional neural networks through the lens of unsupervised learning

Hilding Köhler

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2022:6
Matematisk statistik
Juni 2022

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2022:6**
http://www.math.su.se

Unveiling the inner mechanisms of deep convolutional neural networks through the lens of unsupervised learning

Hilding Köhler*

June 2022

Abstract

To understand the world it is vital to find answers to the questions how, why and where. This is also true when modelling data where these questions are asked to better understand the model's working mechanisms. By understanding model mechanisms, it is possible to explain what causes its results. In recent years the rise of complex models such as convolutional neural networks (CNN:s) has made it possible to produce high performance models, but they are difficult to understand at first sight. CNN:s are the models in focus for this thesis because they are the model standard for image classification. CNN:s are implemented in more fields making it important to understand their working mechanisms. To understand the working mechanisms of CNN:s, this thesis aims to unveil the mechanisms of the CNN, Residual neural network 18 (ResNet-18). Studying ResNet-18 is interesting due to its complex architecture, wide use and high accuracy. Understanding its working mechanisms provides insights into the results of this and other CNN:s. ResNet-18 is in this thesis trained and tested on the benchmark dataset CIFAR-10. The unveiling of the working mechanisms is done using cluster analysis and shape aware distances called commute time distances. This method analyses image clusters based on their characteristics using a distance that respects the underlying data structure. These techniques belong to the field of unsupervised learning, a field providing methods for analysing high dimensional data without needing the data labels. The power of the methods in this thesis is that they can be applied to different mechanisms of models while also avoiding complicated model fitting. The conclusion of the thesis is that all working mechanisms in ResNet-18 serve a purpose. However, the last convolutional operation is the cause for the good image classification which exhibits itself in the low test error of 5%.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: hildingkohler@gmail.com. Supervisor: Chun-Biu Li.

Acknowledgements

I would like to thank my supervisor, professor Chun-Biu Li at the Department of Mathematics at Stockholm University for his guidance, insights and support during this project. I would also like to thank Ali Leylani, AI expert at Granditude, for the idea that sparked this project and for his support and knowledge.

Table of Contents

Symbols and notation

^{T} The transpose operation

$\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ Set of input data

$\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_N\}$ Set of class labels

$C(\boldsymbol{\theta})$ Cost function of the parameter vector $\boldsymbol{\theta}$

$a_t^{(l)}$ Activation function in node t in layer l

O_l Output function of node l in the output layer

$z_t^{(l)}$ Input to activation function in node t in layer l

$\mathbf{W}^{(l+1)}$ The matrix of the set of weights $\{w_{i,j}^{(l+1)}; i \in \{1, ..., T_l\}, j \in \{1, ..., T_{l+1}\}\}$ of T_{l+1} rows and T_l columns

$\mathbf{b}^{(l+1)} = (b_1^{(l+1)}, ..., b_{T_{l+1}}^{(l+1)})^T$ The vector of biases in from layer l to $l+1$

$(I \star K)(t)$ The convolutional operation between functions I and K

$\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)$ Matrix of data in original space

$\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N)$ Matrix of data in lower dimensional space

$\mathbf{P} = (\mathbf{p}_1, ..., \mathbf{p}_d)$ Projection matrix from original to lower dimensional space

C_X Covariance matrix of centered data in original space

$C_{\hat{X}}$ Covarinace matrix of centered data in lower dimensional space

$\mathbf{S}$ The gram matrix

$d(\cdot, \cdot)$ The distance function between two elements

$p_{j|i}$ The conditional probability of a point i choosing j as its neighbour in the ordinary dimensional space

$q_{j|i}$ The conditional probability of a point i choosing j as its neighbour in the lower dimensional space

$KL(P|Q)$ The Kullback-Leibler divergence between distributions p and q

G The weighted undirected graph with edge set E and vertex set V

$\mathbf{W}_G$ The symmetric matrix containing the edge weights of G

$\mathbf{D}$ The diagonal degree matrix

$V(G)$ The volume of the graph G

$\mathbf{L}$ The graph laplacian constructed as $\mathbf{D} - \mathbf{W}_G$

$c_{i,j}$ The commute time distance between points (vertices) i and j

1 Introduction

How, why and where are typical questions that are asked in everyday life in order to gain better knowledge of an event, object or location. These types of questions and the natural curiosity in humans are the driving forces in trying to unveil foreign concepts. In mathematical statistics, modelling is a major field focusing on explaining or reproducing relationships in data. Sometimes, especially in recent years, the models used have gotten so complex that their working mechanisms are beyond our capability to directly understand. However, it is important to understand the models used to be able to explain the results and what caused them. One typical model type that is hard to fully understand the working mechanisms of is the convolutional neural network (CNN) which is mostly used for image recognition. The reason as to why these models are difficult to understand is because they are large and comprised of hand-crafted operations. However, these models are often used in fields such as medicine, self-driving cars and e-commerce. Due to the wide use of CNN:s the aim of this thesis is to implement clustering methods and shape aware distances in order to understand their working mechanisms. This is important since it provides a way of explaining decisions based on their results.

The specific CNN model studied in this project is Residual neural network 18 (ResNet-18), for the original paper presenting this model see He (2016). This is a state-of-the-art image classification model. The idea to study this model originates from a data scientist (Ali Leylani) at an e-commerce consultant company called Granditude. At the company they use an even more complex CNN model called EfficientNet-B8 to find matches between product images at web-shops of their clients and the clients' competitors. Overall, the model has great performance, but in some cases it fails to match images even though they are a clear match. Naturally, they want to understand why this failure in matching the images occurs, whereas in much harder cases a match is found. In an attempt to understand this complex model, this thesis aims to scale it down in both the data set and model used in order to provide insights into the mechanisms of high performing CNN:s. By understanding the mechanisms, it might be possible to improve on future models and explain the results produced for the current ones.

Methods to understand the working mechanisms of models have been presented throughout the years. The most basic idea which also stems from the lack of computational power, is to use models that are easy to understand from the beginning. Examples of these are linear regression models or decision trees, see Xu (2019) *et al.* However, these are often lacking in performance compared to complex models such as CNN:s. Instead, other methods to explain models, in particular neural networks, have been presented. Two of these methods are saliency maps and local interpretable model-agnostic explanations (LIME), see Ribeiro (2016) *et al* and Adadi and Berrada (2018). The aim of saliency maps is to provide a heat map for all parts of an input where the heat is higher for parts that are important to the model output, see Adadi and Berrada (2018). LIME are used to fit simple models such as linear models that locally mimic the mechanisms of the complex model that one tries to understand, see Ribeiro (2016) *et al.* However, the issue with these two methods is that the first one does not really unveil the inner mechanisms. The second method requires that the local models are good fits and make sense in order for them to be useful. To avoid these two issues, this thesis uses cluster analysis and shape aware distances to avoid local model fitting while being able to locally study the model (ResNet-18). For further reading and an overview of explainable AI methods like the ones presented here, see Adadi and Berrada (2018) and Molnar (2022).

The upcoming section serves the purpose of detailing the theoretical aspects of the methods and models used in this thesis. First the fundamentals of neural networks and convolutional neural networks are presented. This is because the main model (ResNet-18) studied in this thesis belongs to this family of models. After neural network theory, dimensionality reduction techniques are presented in order to visualise the behaviour of the networks. Lastly, clustering and graph-based methods are presented to allow for a deeper analysis of neural networks.

2 Theory

This section is meant to provide the reader with the theoretical details to fully understand the results of the thesis. Along with proofs and methods, explanations and reasoning is provided as well.

2.1 Neural networks

All sections concerning neural networks and the specific model types along with notations and some explanations are largely inspired by Goodfellow (2016) *et al.*

Neural networks are a family of models often implemented in regression and classification problems with high accuracy ratings. This family of models originated from the idea to mimic the inner workings of the human nervous system, a highly intricate system for information processing of nerve signals. From a mathematical and statistical point of view neural networks apply numerous linear and non-linear transformations to input data. These models then try to adjust the model parameters to learn general patterns in the data.

Most neural networks belong to a larger family of models that implement supervised learning, meaning that input data is labelled. That is, for a given set of input data $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ each instance has a corresponding observed output or class $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_N\}$. The models belonging to this family learn the appropriate values of the model parameters by minimizing a cost function after applying the model to the input data. The cost function is designed in a way that it measures the difference between true and the model's predicted outputs or class in an appropriate manner. That is, a large cost is incurred for very different $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ and a small cost for similar $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$, where $\hat{\mathbf{y}}_i$ represents the predicted outputs or labels. In mathematical notation, it means minimization of the cost function $C(\boldsymbol{\theta})$ for the parameter vector $\boldsymbol{\theta}$ of the model. An example of supervised learning would be to try to find the best regression weights, $\mathbf{w}$, in multiple linear regression by minimizing the mean square error cost,
$C(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{y}_i]^2 = \frac{1}{N}\sum_{i=1}^{N}[y_i - \mathbf{w}^T\mathbf{x}_i]^2$.

In recent times, neural networks have been favoured by both the industry and academia due to their high performance. The high performance is mostly due to the abundance of data, the large number of parameters, computational tricks and flexibility available. The abundance of data allows for longer and more accurate training. This is often needed for large models such as neural networks. Mainly due to the flexibility of this family, there are many sub-families of neural networks which are more or less suitable for different tasks. This thesis will study convolutional neural networks (CNN:s), which are state-of-the-art models for image classification. To understand CNN:s it is best to first get a better understanding of the most basic sub-family called feedforward neural networks (FNN:s).

2.1.1 Feedforward neural networks

As briefly discussed in the prelude to this section the aim of a neural network is to estimate a model depending on the parameter vector $\boldsymbol{\theta}$ which minimizes the cost function $C(\boldsymbol{\theta})$. That is, a function or model that approximates the mapping between the input and the observed values or labels. Since the models in this thesis are applied to a classification problem, $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_N\}$ corresponds to the true classes of the inputs. The approximation of this mapping is done by finding the estimated parameter vector $\hat{\boldsymbol{\theta}}$ such that the cost function $C(\boldsymbol{\theta})$ is minimized. Although it is good to find an estimate of $\boldsymbol{\theta}$ that minimizes the cost function, it is also important for this estimate to generalise well to new samples. That is, the estimate is one achieving a small cost in training and a high accuracy (percentage of correct classifications) on new input data that is not found in $\mathcal{X}$. This is often tested by splitting the data $\mathcal{X}$ into two subsets $\mathcal{X}_{\text{train}}$ and $\mathcal{X}_{\text{test}}$ called the training set and test set. For $\mathcal{X}_{\text{train}}$ one tries to minimize the cost and achieve a high prediction accuracy on $\mathcal{X}_{\text{test}}$.

The name feedforward neural networks (FNN:s) is mainly due to two reasons. Reason one is that the input, $\mathbf{x}_i \in \mathcal{X}$, is transformed by applying many different functions which together form a composition. When visualizing how the FNN implements these functions to form the composition, a network with connections between input and functions emerges. Reason two is that the input is only processed forward through the network, from input to network output. That is, the FNN never processes the input or the output from the

functions in the model twice. One can think of the input as a skier going downhill, never does the skier visit the same part of the hill twice.

For a more mathematical formulation we consider the visualisation of a general FNN classifier displayed in Figure 1. In the figure three different kinds of layers are shown: the input, the hidden and the output layer. Each circle represents what is called a neuron or node, meaning that our input is of D dimensions and our output of C dimensions. The figure also shows k hidden layers which can be of different dimensions. That is, the first hidden layer can consist of three nodes and the fifth might consist of ten.

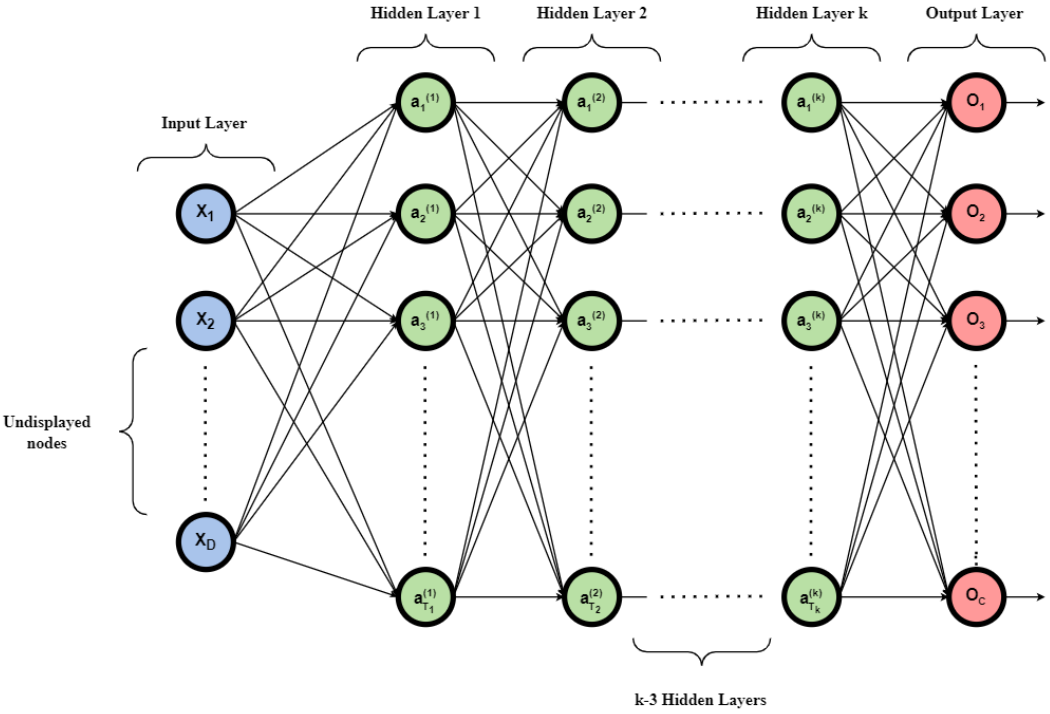


Figure 1: A schematic of the architecture of a general FNN with k hidden layers, D input units and C output units.

In each hidden layer there is a function called the activation function denoted by $a_t^{(l)}$ for $l \in \{1, ..., k\}$ and $t \in \{1, ..., T_l\}$. For the purpose of this thesis, I assume the activation function to be the same for each of the nodes belonging to the same hidden layer. That is, $a_i^{(l)}(.) = a_j^{(l)}(.)$ for $i, j \in \{1, ..., T_l\}$. Generally, it is possible for every node in the network to have unique activation functions. Furthermore, let $\mathbf{a}^{(l)} = (a_1^{(l)}, ..., a_{T_l}^{(l)})^T$ be the vector of activation functions in layer l. The calculations done by the activation function is shown in equation (1) where the ReLU (rectified linear unit) activation function is used as example. In the equation, $\mathrm{z}_j^{(l)}$, is a linear transformation of the outputs from the previous layer which is used as input to activation function j in layer l.

$$a_j^{(l)}(\mathrm{z}_j^{(l)}) = \max(\mathrm{z}_j^{(l)}, 0) \tag{1}$$

The purpose of the activation function is thus to introduce non-linearity into the model. This allows it to model more complex patterns in the data. Since this thesis will only handle classification models the introduction of non-linearity aims to find a better class separation or class decision boundaries. In complex data sets it is common that linear decision boundaries are insufficient. See Figure 2 for an example of a toy dataset of two classes which is not linearly separable.

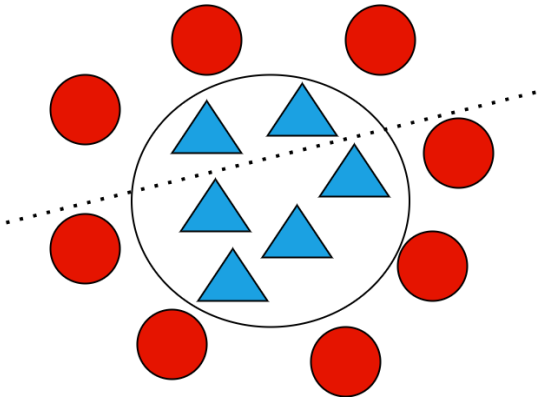


Figure 2: Non-linear (solid line) and linear (dotted line) decision boundaries for a non-linearly separable data set of two classes, red and blue shapes.

The choice of activation function is not trivial and there are many to choose from, some more suitable than others. A more detailed discussion on this is given later in the theory section. In the output layer there is a special case of activation functions denoted by O_l for $l \in \{1, ..., C\}$. These special activation functions are unique for each node in the output layer. The purpose of these output activation functions is to create the final decision boundaries making it possible to discriminate between classes in the data. The most popular choice of output activation function is the softmax function defined in equation (2) where $\mathbf{z} = (z_1^{(k+1)}, ..., z_C^{(k+1)})^T$ is the input vector to the softmax.

$$O_l(\mathbf{z}) = \frac{\exp\left(z_l^{(k+1)}\right)}{\sum_{t=1}^{C} \exp\left(z_t^{(k+1)}\right)} \tag{2}$$

To summarise the introduced notations: superscript marks layers, subscript marks features/elements in a vector, C is the number of output units (classes) and k is the number of hidden layers.

The arrows in Figure 1 connecting the nodes with each other represent a linear transformation of the output from the node at the origin of the arrow. This linear transformation is the input to the node at the end of the arrow. That is, for layers l and $l+1$ we let the arrows connecting nodes in layer l with node j in layer $l+1$ represent the linear transformation in equation (3).

$$z_j^{(l+1)} = \sum_{i=1}^{T_l} w_{i,j}^{(l+1)} a_i^{(l)} + b_j^{(l+1)} \tag{3}$$

In this formulation $w_{i,j}^{(l+1)}$ is the scalar weight from node i to node j and $b_j^{(l+1)}$ the scalar bias added to the weight and activation function multiplication. Each network layer can have many nodes and therefore many weights and biases connecting it to the next layer. Hence, it is convenient to introduce the following matrix and vector notation. Here $\mathbf{W}^{(l+1)}$ is the matrix of the set of weights $\{w_{i,j}^{(l+1)}; i \in \{1, ..., T_l\}, j \in \{1, ..., T_{l+1}\}\}$ of T_{l+1} rows and T_l columns and $\mathbf{b}^{(l+1)} = (b_1^{(l+1)}, ..., b_{T_{l+1}}^{(l+1)})^T$ the set of biases. This yields the following vector notation for inputs to layer $l+1$

$$\text{Input to layer } (l+1) = \mathbf{z}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)} \tag{4}$$

For a full theoretical framework, weights $\mathbf{W}^{(1)}$ and biases $\mathbf{b}^{(1)}$ connecting the input layer to the first hidden layer are needed. That is, for $i \in \{1, ..., N\}$

$$\text{Input to layer } (1) = \mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)} \tag{5}$$

This means that in this network the parameter vector $\boldsymbol{\theta}$ is equal to the following union of the two sets of weights and biases.

$$\boldsymbol{\theta} = \{\mathbf{W}^{(l+1)}; l \in \{0, 1, ..., k\}\} \cup \{\mathbf{b}^{(l+1)}; l \in \{0, 1, ..., k\}\} \tag{6}$$

The motivation for these linear transformations is to rescale and reposition the data in order to learn more complex patterns in it. In other words, the linear transformation is a straightforward way of moving the data around in a high dimensional space to gain more favourable decision boundaries. Furthermore, the transformation combines different features to create a new common feature. This is sometimes called feature fusion.

In order to find the optimal values for $\boldsymbol{\theta}$ for the model in Figure 1, the minimization problem $\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}}\ C(\boldsymbol{\theta})$ is solved. Hence, it is important to choose a function for C which accurately measures the difference in the model output $\mathbf{O}(\mathbf{x}_i) = (O_1(\mathbf{x}_i), ..., O_C(\mathbf{x}_i))^T$ and the true class labels $\mathbf{y}_i$ for $i \in \{1, ..., N\}$. Now, let the output activation functions be the softmax function as in equation (2), with the motivation being that the $O_l(\mathbf{x}_i)$:s in this case represent the class conditional probabilities $p_{\mathrm{c}_i|\mathbf{x}_i,\boldsymbol{\theta}} = \mathbb{P}(\mathbf{y}_i = \mathrm{c}_i|\mathbf{x}_i, \boldsymbol{\theta})$. The reason it can be interpreted as a class conditional probabilities is because given $\mathbf{x}_i$ and $\boldsymbol{\theta}$, the value of $O_l(\mathbf{x}_i)$ is between zero and one. With the softmax output activation function a popular choice of $C(\boldsymbol{\theta})$ is the cross-entropy cost. Cross-entropy in combination with the softmax output activation function is a suitable choice since it can alleviate the problem of vanishing gradient. A more detailed explanation of why this choice is good is given in the next section. In mathematical terms we define the cost as follows.

$$\begin{aligned} C(\boldsymbol{\theta}) &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\mathbb{I}\{\mathbf{y}_i = \mathrm{c}_j\}\log(O_j(\mathbf{x}_i)) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\mathbb{I}\{\mathbf{y}_i = \mathrm{c}_j\}\log\left(\frac{\exp\left(z_j^{(k+1)}(\mathbf{x}_i)\right)}{\sum_{t=1}^{C}\exp\left(z_t^{(k+1)}(\mathbf{x}_i)\right)}\right) \end{aligned} \tag{7}$$

Now, the method for minimizing the cost function can be chosen in several ways, most of which are gradient-based. This means that an algorithm is used that updates the model parameters iteratively by subtracting the gradient of the cost with respect to the parameter vector. For this thesis mini-batch gradient descent with momentum is employed. That is, I find the optimal values of the parameter vector $\boldsymbol{\theta}$ by the iteration scheme in equation (8) where t denotes the iteration step and $C(\boldsymbol{\theta})$ is calculated for a subset of size 128 (called a mini-batch) of the training data. This means that the parameter vector is updated $N/128$ times for each run on the training data. One run on the training data is called an epoch and training a model often involves several epochs (300 is used in thesis), which implies $300N/128$ updates or iterations. In equation (8) the parameters η (the learning rate) and $\alpha(t)$ (the momentum for iteration t) can be observed, these are set by the user. For more details on minimization and mini-batch with momentum see chapter 8 in Goodfellow (2016) *et al.*

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \eta\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \alpha(t)(\boldsymbol{\theta}^{(t-1)} - \boldsymbol{\theta}^{(t-2)}) \tag{8}$$

The reason mini-batch and momentum are implemented together with ordinary gradient descent is to avoid local minima trapping, which makes the model more accurate in theory.

For a final theoretical aspect of FNN it is needed to clarify how the gradients in $\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ are found. The calculations are done by an implementation of the multivariate chain rule called backpropagation which is a computationally effective method. By implementing multivariate techniques from calculus it is seen that each layer's derivatives depend on the derivatives in the later layers. Hence, the change in cost is spread backwards in the derivatives. The exact formulas are not of great importance to this thesis and are therefore omitted. Since these are computational aspects not related to mathematical statistics, further discussion

is omitted. For further reading on backpropagation and its implementations see chapter 6 in Goodfellow (2016) *et al.*

2.1.2 Activation function motivation

This section shortly motivates the choice of activation functions and cost functions. As mentioned previously there are many choices of activation functions for the hidden layers and the output layer. In this report the rectified linear output unit, which is often abbreviated as ReLU, is implemented. The form of the ReLU activation function is $\max(\mathbf{x}_i, 0)$. This function is almost a linear transformation of the input. One of the main motivations why the ReLU activation function is used is because it is easy to optimize with gradient descent while still introducing non-linear transformations in an interpretable way. The ReLU being easy to optimize means that differentiating the cost function with respect to the parameter vector $\boldsymbol{\theta}$ yields fast and easily computed products of partial derivatives involving differentiation of the ReLU function.

An aspect to be aware of when using the ReLU activation function is that it can both alleviate or cause the problem of vanishing gradient. This is because it has a derivative of one on the non-negative part of its support otherwise it is zero. The problem of vanishing gradient can be described as the partial derivatives in the backpropagation algorithm becoming small or zero for some dimensions of $\boldsymbol{\theta}$. When these become very small the update of these and other parameters in the algorithm can be next or equal to zero, meaning that the model stops learning from the data. Hence, the ReLU function is advantageous when the input to it tends to be positive due to the large derivative. However, when the inputs are negative the derivatives are zero causing the gradient to vanish.

Earlier, the implementation of the softmax output activation function was shortly motivated by its correspondence to the class conditional probabilities. It can be further motivated by studying the expression in equation (9). In the equation it is shown that $z_j^{(k+1)}(\mathbf{x}_i)$ has a direct additive effect on the the cross-entropy cost in equation (7). This implies that the problem of vanishing gradient can be avoided to a larger extent. This is because the gradient will have a linear contribution from the additive effect.

$$\begin{aligned} \log(O_j(\mathbf{x}_i)) &= \log\left(\frac{\exp\left(z_j^{(k+1)}\right)(\mathbf{x}_i)}{\sum_{t=1}^{C}\exp\left(z_t^{(k+1)}(\mathbf{x}_i)\right)}\right) \\ &= z_j^{(k+1)}(\mathbf{x}_i) - \log(\sum_{t=1}^{C}\exp\left(z_t^{(k+1)}(\mathbf{x}_i)\right)) \end{aligned} \tag{9}$$

Furthermore, the choice of softmax is good in combination with the cross-entropy loss. The reason for this is due to the following approximate relationship in equation (10). The approximation works well when $\mathbf{z}^{(k+1)}$ has one component that is much larger than the other. This situation occurs when the network is very confident in its class prediction.

$$\log(\sum_{t=1}^{C}\exp\left(z_t^{(k+1)}(\mathbf{x}_i)\right) \approx \max(\mathbf{z}^{(k+1)}) \tag{10}$$

The reason the approximate relationship is useful is because all small values in $\mathbf{z}^{(k+1)}$ will have close to no impact on $\log(\sum_{t=1}^{C}\exp\left(z_t^{(k+1)}(\mathbf{x}_i)\right)$ compared to the largest value (due to the exponential function). This approximate relationship implies that the inner sum of the cost function in equation (7) roughly evaluates to zero when $\max(\mathbf{z}^{(k+1)})$ corresponds to the output activation function for the correct class label. This means that a small cost is incurred for correct and confident class predictions. Otherwise this expression evaluates to a large number if $\max(\mathbf{z}^{(k+1)})$ corresponds to a wrong and confident prediction of the correct label. In other words, the softmax in combination with the cross-entropy cost continuously penalizes very wrong predictions while no cost is incurred for correct predictions. This implies that we are punishing the model for being very wrong in an appropriate way, allowing it to learn from the training data.

2.2 Convolutional neural networks

This section continues the theory of neural networks by introducing one of the main model types in this thesis, the convolutional neural network. This model type is the focus of this thesis because of its strength in analysing images by the implementation of different model-specific operations. The main operations characterising the convolutional neural networks (CNN:s) are the convolutional operations and the average pooling operations (a subsampling technique). The first operation is mainly used to reduce image dimensions by weighting image pixel values belonging to a predefined neighbourhood. The second operation, average pooling, is mainly used for making the model robust by averaging over transformed input image pixels.

The reason FNN:s are introduced in previous sections is because CNN:s work in a similar fashion, but with some distinctions. The distinctions and the architecture of the CNN are the main topics for the coming sections. The theory in this section is based on Goodfellow (2016) *et al.*

2.2.1 CNN architecture

The architecture of the CNN model is in a broad sense similar to that of the basic FNN presented in Figure 1. For the theory of CNN:s we first consider a simple, but common architecture shown in Figure 3. The figure shows that the input in this case is an image of a robot. Images of this kind can be translated into $C \times H \times W$ (with C often equal to 3) dimensional matrices containing pixel values. The matrices are often referred to as C dimensional matrices or C dimensional tensors. The notations H and W represent the height and width (spatial dimensions) respectively while C represents the channel dimension (colour dimension). Furthermore, the input and output at each layer in the CNN is a high dimensional matrix, of which images are a subtype.

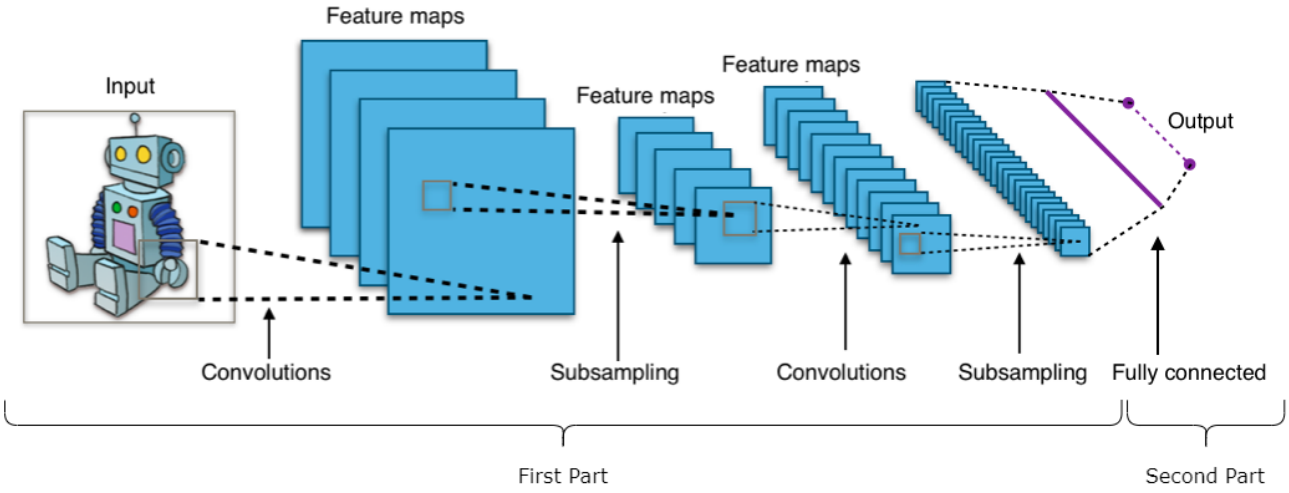


Figure 3: Schematic of the architecture of a general CNN with a FNN at the end. The image is taken from Aphex34 (2015) and edited.

As can be observed in Figure 3 the network is divided into two parts. The first part uses convolutional and subsampling operations and the second part is a FNN at the end of the last subsampling. For illustrative purposes, we assume that the FNN is the one presented in Figure 1.

The core idea of CNN:s is to first implement convolutions and average pooling (subsampling) operations to extract important features from the input. In other words, with CNN:s the aim is to extract only the relevant parts of an image. When the relevant parts of the image have been extracted they are used as the input to a FNN for classification.

The blue squares in the Figure 3 in the first part of the model can be interpreted as the convolutional

layers. Between the blue squares there are arrows symbolising weights and biases like in the FNN:s. In the blue squares there are also activation functions implemented. The reason is to introduce non-linear transformations in the convolutional layers. The weights and biases in the convolutional layers have a special interpretation in CNN:s, they represent the convolutional and average pooling operations.

It is possible to transform images to work with ordinary FNN:s only. However, this approach often involves extremely large networks (many parameters) that are prone to overfitting. Furthermore, these models tend to ignore the natural spatial organization of images. CNN:s handle images very well and in general do not need as many parameters to work well, a trait which is mainly due to sparse connectivity. Sparse connectivity means that instead of having many connections between the layers (symbolised by arrows in Figure 1) only a few are used. This is in turn an effect of the convolutional operation.

2.2.2 The convolutional operation

This section presents and discusses the main operation in CNN:s called the convolutional operation. In other fields other than neural networks the convolutional operation is sometimes referred to as the cross-correlation function. However, in this thesis the operation is referenced to as the convolutional operation.

The motivation behind the convolutional operation is to allow the neural network to extract important features from the images and forward these deeper into the network. This means that with each convolutional operation the neural network is aiming to extract remaining important information from the image. Different convolutions can have different areas of information that they deem to be important. For example, some may focus on edge detection in the image while others focus on stark colour distinctions. For a real-life analogy, the convolutions are like asking museum visitors what aspects of a painting they find the most interesting. Based on the answer one tries to create a better painting based on only the interesting parts. This means, that one could end up with a highly skewed version of the Mona Lisa. Another way of viewing the convolution is that it outputs a weighted average of the input, where the weights are described by a kernel.

For a formal definition of the convolutional operation, let I (the input function, pixel values in this case) and K (the kernel function) be two complex valued functions on $\mathbb{R}^n$. The convolutional operation is then defined as follows.

$$(I \star K)(t) \coloneqq \int I(t+\tau)\overline{K(\tau)}d\tau, \quad t \in \mathbb{R}^n \tag{11}$$

Where $\overline{K(\cdot)}$ is the complex conjugate of $K(\cdot)$. However, in this thesis only real valued functions are implemented, implying that $\overline{K(\cdot)} = K(\cdot)$. The parameter t is often viewed as a point in time which makes the integrating variable τ a time shift. In real-life applications it is not always realistic or possible to have measurements of the functions I and K at every time point, meaning that t is discrete. This yields a similar definition of the discrete time convolution. For more details on the formulas for convolution, see Papoulis (1962).

$$(I \star K)(t) \coloneqq \sum_{\tau \in \mathcal{T}} I(t+\tau)\overline{K(\tau)}, \quad t \in \mathbb{R}^n \tag{12}$$

Where $\mathcal{T}$ is the range of possible values of τ. Since discrete time convolution is expressed as a sum of products of the two functions I and K it can be written in element-by-element matrix multiplication form. This is convenient in computer applications. It is also convenient to implement convolutions using matrices when interpreting the effect of them. Figure 4 shows an illustration of a $3 \times 3 \times 3$ convolution applied to a $3 \times 32 \times 32$ matrix, which yields a $1 \times 30 \times 30$ output. The convolutional operation when applied as is, reduces the dimensions from the input matrix to the output matrix which can also be observed in the figure. Applying the matrix representation of the convolutional operation is often referred to as applying a kernel or

filter to the input, where the kernel is the matrix. Applying the kernel is equivalent to sliding it across the input matrix and performing element-by-element products to calculate the output. In this case $3 \cdot 3 \cdot 3 = 27$ element-by-element multiplications are executed each time the kernel is moved.

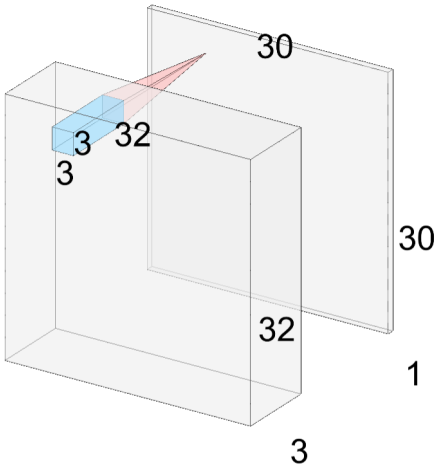


Figure 4: Visualisation of convolution applied to a $3 \times 32 \times 32$ matrix represented by a $3 \times 3 \times 3$ matrix (kernel). The convolution outputs a new $1 \times 30 \times 30$ matrix by sliding the $3 \times 3 \times 3$ matrix over the input matrix. The sliding starts at the top left corner with the center of the $3 \times 3 \times 3$ matrix being moved one element in each slide. When the right edge of the input has been reached the kernel is repositioned vertically in the same manner.

This is the most basic way of implementing the convolutional operation to higher-dimensional matrices. However, the more common way of implementing the convolutional operation is to apply several convolutions to the same matrix ($3 \times 32 \times 32$ in this case). In other words, let the matrix the convolution is applied to have elements $I_{i,(j,k)}$ (where indices represent channel, height and width) and the l:th convolution or kernel applied be represented by a matrix with elements $K_{t,(s,v),l}$ with l fixed. Then the output matrix has elements $Z_{l,(j,k)}$ given by equation (13) which yields a high dimensional matrix. The sum is over the indices for which the convolutional operation is valid. Indices inside of the parentheses represent the spatial dimensions, while the other indices outside represent the channel dimensions where t and l need not be the same.

$$Z_{l,(j,k)} = \sum_t \sum_s \sum_v I_{i+t,(j+s,k+v)} \cdot K_{t,(s,v),l} \tag{13}$$

For a visual interpretation, this corresponds to making the operation in Figure 4 k times with k different kernels and then fuse the output to form one large matrix output. By using the numbers in Figure 4, a matrix of size $k \times 30 \times 30$ is outputted by applying k unique $3 \times 3 \times 3$ kernels.

As shown in Figure 4, the convolution can be represented by a matrix or kernel that is slid across the image to produce an output. The matrix that is slid across contains the $3 \cdot 3 \cdot 3 = 27$ weights for this convolutional layer in the CNN model. This means that, instead of connecting each element to a unique set of weights (as in FNN:s), the same weights are reused across the same layer to connect it to the next layer. This is called sparse connectivity and weight sharing, which is a good property since it makes the CNN less prone to overfitting. This is due to a less complex model and that each element is only connected to some parts of the next layer. The weights contained in the matrix representation are learned by gradient-based optimization methods as in the FNN:s.

Although the weights of the convolutional operations in a CNN do not need to be specified by the user (they are trained via backpropagation) the size of them needs to be. By size, we refer to the dimensions of the matrix or kernel representing the convolutional operation in equation (12). In this thesis the size

of the kernel is always set to be $C \times 3 \times 3$ or $C \times 1 \times 1$ with 3×3 and 1×1 as the spatial dimensions. The reason for the $C \times 3 \times 3$ size is due to it being symmetric around a centre element. The symmetry is a satisfying property since elements around the centre pixel contain information about local dissimilarities and similarities around it. If the kernel is not symmetric, then there is no clear centre and it will not collect information from the whole local area. The $C \times 1 \times 1$ are often used to summarise over channels only, since they will not consider the spatial coordinates of elements. Hence, this can be implemented to reduce channel dimensions or extract channel specific information.

There are a few more methods that can be applied together with the convolutional operation. The two main methods that are often seen together with the convolutional operation are strides and padding.

By adding a stride of size s to the convolutional operation, the kernel or matrix is only allowed to be applied to every s:th element in the input. This makes the output dimensions smaller and contain less information from the input, due to the smaller number of times the convolution is applied. However, the reduction in information and dimensions can be traded for faster computations. This is especially important in large networks and in layers where applying the convolutional operation to every element instead of every s:th element does not matter too much.

The second addition that can be made to the convolutional operation is the padding of size p. This method adds a frame of width p of zero valued elements around the input matrix. This allows for the convolutional operation or kernel to be applied to the input more times than compared to the input without padding. The reason for the padding is to avoid downsampling in the output matrix which allows for deeper model architectures. However, the downside of this is that the computational cost can become large since the convolutional operation is applied a larger number of times.

It is important to note that the convolutional operation is a linear operation. It is possible to introduce non-linearity into the model to be able to learn more complex patterns in the data. This is done in this thesis by forwarding convolution output into a ReLU together with an added bias as done in FNN:s.

2.2.3 The average pooling operation

This section introduces and discusses another key operation for CNN:s called average pooling. There are several reasons why it is desirable to implement the average pooling operation in some layers of a CNN model. The main reason is to introduce robustness to changes in the layers in the model. In this case robustness can be interpreted as the model producing the same results despite some changes in the layer that average pooling is applied to. There are other types of pooling available, but average pooling is the operation used in this thesis. To understand the average pooling operation it is illustrative to view it as applying a kernel similar to the procedure shown in Figure 4.

Suppose the input to the average pooling operation is a matrix of size $C \times H \times W$. As in the convolutional operation, pick a kernel size for the average pooling kernel, for generalization pick k. This kernel has all weights equal to $\frac{1}{k}$. The average pooling is done by applying the kernel with a stride to the input. This operation results in averaging over spatial pixel values in a neighbourhood of size k pixels. For a visualization of how average pooling works see Figure 5.

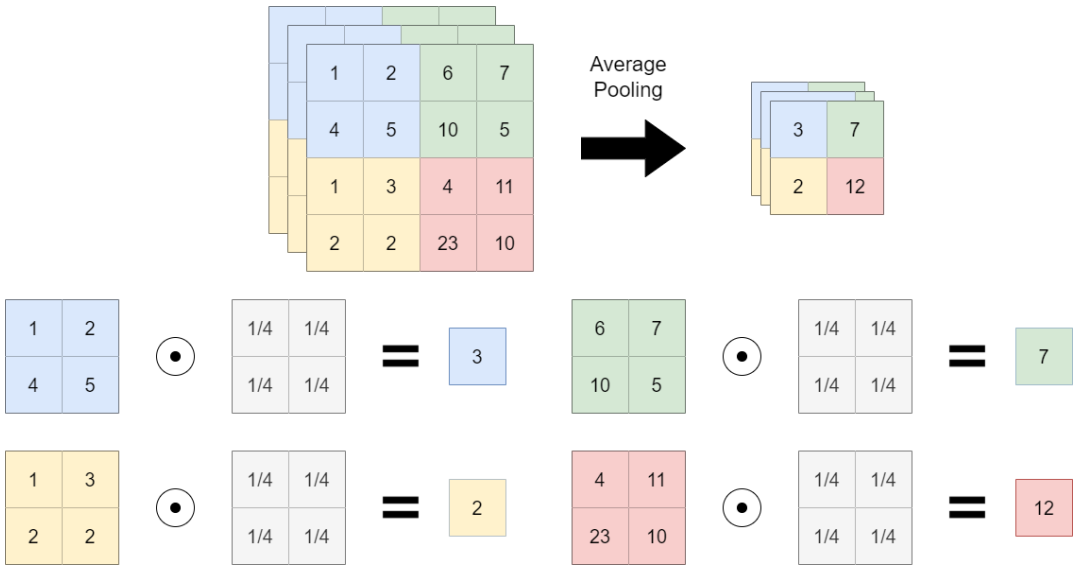


Figure 5: A visualisation of a $1 \times 2 \times 2$ average pooling applied to a $3 \times 4 \times 4$ image. The convolution outputs a new $3 \times 2 \times 2$ matrix by sliding the $1 \times 2 \times 2$ average pooling matrix over the input image. The sliding starts at the top left corner with the top left corner of the $1 \times 2 \times 2$ matrix matching the top left corner of the current channel. Then the kernel is moved such that the left edge of the repositioned kernel is adjacent to the right edge of the kernel at its previous position. When the right edge of the input has been reached the kernel is repositioned vertically in the same manner. This is equivalent to applying the average pooling 4 times per channel C. In the lower part of the figure the calculations done in the average pooling for the first channel are shown explicitly where outputs are color-coded.

As is shown in Figure 5, the average pooling is weighting the pixel values in local neighbourhoods defined by the kernel size, for each channel. This means that the output from the average pooling channel does not change a lot if only one element in a neighbourhood changes. This implies that the later layers in the network will also not be affected too much by local changes. Hence, by introducing the average pooling operation the CNN becomes robust to local changes in matrix elements. This property of being insensitive to small shifts or changes in pixel values is often called translational invariance. It is also common to view average pooling operations as creating a blurred or fuzzy version of the input matrix. This is because elements in a local neighbourhood tend to be somewhat similar. That is, by averaging, some features from each element are extracted to create a common representative in the output from the neighbourhood. Lastly, average pooling can alleviate the computational costs of the CNN since it reduces the dimensions in the output. This property is called downsampling.

2.2.4 Skip connection

This particular section is inspired by He (2016) *et al.*

This section handles a special feature of CNN:s called skip connections. This is a relatively new feature that is not present in all CNN:s. The mechanism behind the skip connection is to allow inputs to certain layers to partially avoid being processed by the layer. This might seem a bit counterintuitive since it is often desirable for the model to learn from the output of previous layers. However, this method has been shown to actually boost the performance of the CNN:s.

The skip connection is initially introduced in the paper *"Deep Residual Learning for Image Recognition"*, in which the authors motivate the implementation alongside a specific type of CNN:s called ResNet (residual neural networks), see He (2016) *et al.* This model family is more thoroughly presented in the model part of this thesis. The founders of this network design motivate it by prevention of the degradation problem and vanishing gradient.

The degradation problem is not really a theoretical problem, but rather an empirical observation in very deep CNN:s. The degradation problem is observed when training CNN:s of different depths (different number of layers with the same structures) and where shallower architectures perform better than deep architectures in terms of both training and test accuracy. The lower test accuracy for deeper architectures might not be too surprising due to deeper architectures being prone to overfitting as an effect of more flexibility (more parameters). However, more flexible models should have a higher training accuracy compared to less flexible (shallower CNN:s). Hence, a very counterintuitive observation has emerged, lower training and test accuracy for more flexible models. For a graphical illustration of this observation see the copy-righted figures Fig 1 and Fig 4 in He (2016) *et al.*

Why the degradation problem occurs can be explained by deeper architectures' inability to learn identity layers. By learning identity layers it is meant that the CNN is unable to mimic a true identity mapping in parts of the network. An identity mapping means that the input to the layer is the same as the output to the layer. That is, if the input to a layer is $\mathbf{x}$ and the output is $M(\mathbf{x})$ where M represents the true mapping, then $M(\mathbf{x}) = \mathbf{x}$. This means that no transformation is done. The inability to learn identity mappings explains why shallower architectures have better accuracies. This is because deeper architectures with the same layer structure should be able to perform equally good if the additional layers in them approximate an identity mapping well. In other words, if the additional layers in the deeper architecture perform no transformations then they are equivalent to the shallower architectures. Why deeper architectures fail to learn identity mappings (which are linear) in the additional layers might be because of the non-linear transformations introduced by the activations functions, see He (2016) *et al* for further details.

To alleviate the problem of deep CNN:s failing to approximate linear mappings in some of its layers, the skip connection is introduced. The skip connection is also called residual connection in some applications. To understand how the implementation of a skip connection is done it is illustrative to study the graphical illustration in Figure 6. In the figure, the input $\mathbf{X}$ is split into two parts: one part which transforms the input with weight operations and an activation function; a second part which skips these transformations. The output from part one is $\mathrm{F}(\mathbf{X})$ and the output from part two is just $\mathbf{X}$. The two parts are then added together and used as input to an activation function. Without the skip connection the input to the last activation function would be $\mathrm{F}(\mathbf{X})$. By letting $M(\mathbf{X})$ be the true mapping between the input and the input to the activation function, we can approximate it by $M(\mathbf{X}) \approx \mathrm{F}(\mathbf{X}) + \mathbf{X}$. This means that one models $\mathrm{F}(\mathbf{X}) = M(\mathbf{X}) - \mathbf{X}$ which uses network weights to approximate the difference between the true mapping and the input. Hence, if a linear mapping is appropriate for this part of the network, the network can learn to put the weights for $\mathrm{F}(\mathbf{X})$ to zero. This means that $M(\mathbf{X}) = \mathbf{X}$ is what this layer represents.

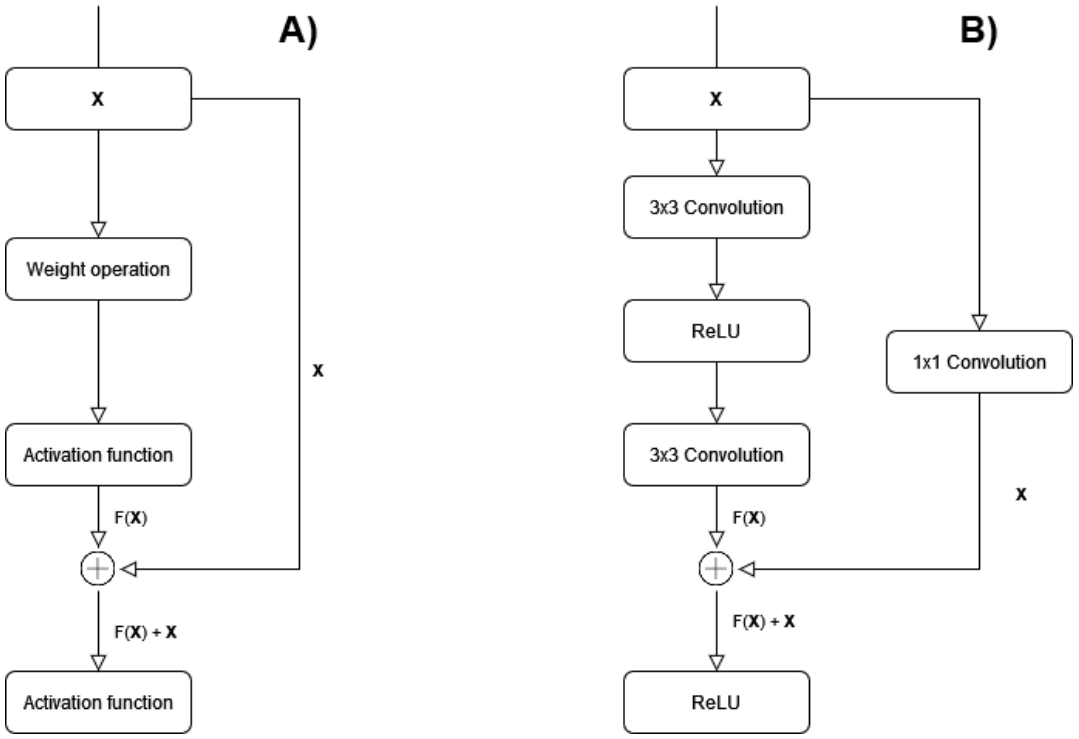


Figure 6:
A): A graphical illustration of a general implementation of a skip connection
B): A graphical illustration of the skip connection implemented as done in the model used in the application section of this thesis.

As is shown in B) in Figure 6, the weights in F($\mathbf{X}$) represent convolutions and the input to part two is actually transformed by a $C \times 1 \times 1$ convolution. The reason for this is that this convolution ensures that the dimensions of $\mathbf{X}$ and F($\mathbf{X}$) are the same. The difference in dimensions is caused by the $C \times 3 \times 3$ convolutions in part B) of the diagram.

Another motivation behind the skip connection is that it also helps in avoiding the problem of vanishing gradients. This can be explained by the skip connections making direct additions to the gradient with respect to the weights of the layers.

With the important theoretical aspects of neural networks presented, the next logical step is to present the necessary theory for visualisation of the network layers. The necessary theory belongs to a field called dimensionality reduction. This field allows high dimensional layers in the network to be reduced down to lower dimensions, often two, while retaining necessary information. The end goal is two find a way of interpreting the behaviour of the networks.

2.3 Dimensionality reduction techniques

2.3.1 Principal component analysis (PCA)

The sections relating to PCA is inspired by Lee and Verleysen (2008) with some slight change of notations.

The simplest method of dimensionality reduction and data visualisation is PCA, short for principal component analysis. PCA falls into a family of methods called linear dimensionality reduction since it performs a linear projection of centred data onto a new and smaller vector space. This new vector space is constructed in such a way that the new basis vectors make the data uncorrelated (removing redundant information). Furthermore, the new vector space is constructed in such a way that the data has the largest sample variance

along the direction of the first basis vector. It has the second largest sample variance along the direction of the second basis vector and so on. Hence, PCA can be viewed as a way of finding a set of basis vectors maximizing sample variance and minimizing the correlation of the data. It can be shown that PCA is equivalent to finding the eigenvectors and eigenvalues of the sample covariance matrix of the centred data. The reason PCA seeks to maximize sample variance and minimize correlation is because variance can be viewed as information and correlation as a measure of redundant information.

2.3.1.1 Mathematical formulation

For a mathematical formulation of the PCA method, let $\mathbf{X}$ be the matrix of D rows and N columns containing our data. That is, $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)$ where $\mathbf{x}_k = (x_{k,1}, ..., x_{k,D})^T$ for $k \in \{1, ..., N\}$. Since PCA makes use of the sample covariance matrix we introduce the following formula for this matrix, where $\mathbf{1}_N$ is the N-dimensional column vector containing ones everywhere.

$$C_X = \frac{1}{N-1}(\mathbf{X} - \frac{1}{N}\mathbf{X}\mathbf{1}_N\mathbf{1}_N^T)(\mathbf{X} - \frac{1}{N}\mathbf{X}\mathbf{1}_N\mathbf{1}_N^T)^T \tag{14}$$

Further assume the data to have zero sample mean in every dimension, then the sample covariance matrix simplifies to the following expression in equation (15). That is if $\frac{1}{N}\mathbf{X}\mathbf{1}_N\mathbf{1}_N^T = \mathbf{0}$ holds.

$$C_X = \frac{1}{N-1}\mathbf{X}\mathbf{X}^T \tag{15}$$

Now let $\mathbf{P} = (\mathbf{p}_1, ..., \mathbf{p}_d)$ be a matrix of D rows and d columns, such that the vectors $\mathbf{p}_k = (p_{k,1}, ..., p_{k,D})^T$ for $k \in \{1, ..., d\}$ are all orthonormal to each other. This implies that $\mathbf{P}^T\mathbf{P} = \mathbf{I}_d$, where $\mathbf{I}_d$ is the identity matrix of d rows and columns. Let us now introduce the lower dimensional representation of $\mathbf{X}$ as the new matrix $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N)$ of d rows and N columns. Where $\hat{\mathbf{x}}_k = (\hat{x}_{k,1}, ..., \hat{x}_{k,d})^T$ for $k \in \{1, ..., N\}$. We further specify $\hat{\mathbf{X}}$ to be the representation of $\mathbf{X}$ in a new vector space (of lower dimension) spanned by the orthonormal columns of $\mathbf{P}$. In a mathematical expression, $\hat{\mathbf{X}} = \mathbf{P}^T\mathbf{X}$. Since it is assumed that each dimension of $\mathbf{X}$ has zero sample mean, it directly follows that $\hat{\mathbf{X}}$ is also centred. This leads us to define the sample covariance matrix of the data in the lower dimensional vector space as.

$$C_{\hat{X}} = \frac{1}{N-1}\hat{\mathbf{X}}\hat{\mathbf{X}}^T \tag{16}$$

Since the projection matrix $\mathbf{P}$ was introduced with basis vectors $\mathbf{p}_k$ for $k \in \{1, ..., d\}$, it follows that the above expression can be rewritten as.

$$\begin{aligned} C_{\hat{X}} &= \frac{1}{N-1}[\mathbf{P}^T\mathbf{X}][\mathbf{P}^T\mathbf{X}]^T \\ &= \frac{1}{N-1}\mathbf{P}^T\mathbf{X}\mathbf{X}^T\mathbf{P} = \mathbf{P}^T C_X \mathbf{P} \end{aligned} \tag{17}$$

Equation (17) is very useful since it provides an initial relationship between the covariances in the two vector spaces. This is because one can retrieve either one of the matrices from the other, by applying the matrix $\mathbf{P}$ and its transpose.

Equation (17) can be further simplified by performing an eigenvalue decomposition of $C_X = \mathbf{V}\Lambda\mathbf{V}^T$ with $\mathbf{V}$ being the matrix of D rows and columns of orthonormal eigenvectors $\mathbf{v}_k$ for $k \in \{1, ..., D\}$. This eigenvalue decomposition is possible since C_X is a covariance matrix and is thus a real and symmetric matrix. The matrix Λ of D rows and columns is a diagonal matrix with the diagonal elements being the size ordered eigenvalues λ_k for $k \in \{1, ..., D\}$ corresponding to the eigenvectors $\mathbf{v}_k$. The simplification of equation (17) is shown in equation (18).

$$C_{\hat{X}} = \mathbf{P}^T[\mathbf{V}\Lambda\mathbf{V}^T]\mathbf{P} \tag{18}$$

Now, without violating the stated assumptions on $\mathbf{P}$, choose $\mathbf{P}$ such that it contains the first d eigenvectors $\mathbf{v}_k$ corresponding to the first d largest eigenvalues. This choice leads to the following simplification of equation (18) in equation (19). This equation shows that the eigenvalues of C_X correspond to the variances of the lower-dimensional representation of the data. This is because the diagonal of a covariance matrix corresponds to the variances.

$$C_{\hat{X}} = \mathbf{I}_{d\times D}\Lambda\mathbf{I}_{D\times d} \tag{19}$$

Since $C_{\hat{X}}$ is diagonal, the covariances in the lower dimension are zero which implies that the projection minimizes the redundant information. This is because when the covariances are zero there is no linear relationship between the variables. That is, no dimension contains information about the others.

Now, for a full understanding of PCA it is important to know that this choice of $\mathbf{P}$ also maximizes the variance in the lower dimension. This is shown by not imposing any constraints on $\mathbf{P}$ (other than orthonormality) and solving the maximization problem in equation (20). For a full solution see equation (2) to (4) in the appendix.

$$\max_{\mathbf{P}} \ \text{Trace}(C_{\hat{X}}) \ \text{ subject to } \ \mathbf{P}^T\mathbf{P} = \mathbf{I}_d \tag{20}$$

Furthermore, it is important to study the total variance of $\hat{\mathbf{X}}$ defined as in equation (21) and with $\mathbf{P}$ chosen as above. For a full motivation see equation (1) in the appendix.

$$\sigma_{\hat{X}} = \sum_{i=1}^{d} \lambda_i \tag{21}$$

This equation provides a measure of how much variation is captured in the low dimensional representation of $\mathbf{X}$, since it is the sum of the diagonal elements of $C_{\hat{X}}$. The reason for this is because the total variation in the original data is given by equation (22). (In the proof in equation (22) the cyclic property of trace is used. It is also used that the matrix $\mathbf{V}$ is orthonormal.)

$$\begin{aligned} \text{Trace}(C_X) &= \text{Trace}(\mathbf{V}\Lambda\mathbf{V}^T) \\ &= \text{Trace}(\mathbf{V}^T\mathbf{V}\Lambda) \\ &= \text{Trace}(\Lambda) = \sum_{i=1}^{D} \lambda_i \end{aligned} \tag{22}$$

In equation (21) only the d largest components in the sum in equation (22) are used. It was also shown that no other choice of $\mathbf{P}$ can yield a higher total variance in $\hat{\mathbf{X}}$. This means that by choosing the columns of $\mathbf{P}$ to be the d first eigenvectors corresponding to the d largest eigenvalues, the maximum fraction of the global variation is preserved. That is, $\sum_{i=1}^{d} \lambda_i / \sum_{i=1}^{D} \lambda_i$ cannot be larger for any other choice of columns of $\mathbf{P}$.

It is illustrative to think of PCA as finding a low dimensional representation of $\mathbf{X}$ such that the fraction of the total global variance is maximal in the lower dimension. This is because variance in a sense corresponds to information since a large variance implies a large range of values in the low dimensional. That is, when there is a large range of values in the data, one covers more of its support.

Although it is possible to make the low dimension the same dimension as the original data, it is desirable to make the dimensions two or even three for visualisation purposes. Since PCA selects the dimensions with

the highest variances, it is interesting to study how much of the global variance each dimension accounts for. This is often done by dividing the corresponding eigenvalue by the total global variance (sum of eigenvalues).

The PCA method of dimensionality reduction turns out to be equivalent to another method called classical metric MDS. However, the equivalence only holds under the certain conditions that the pairwise distances between the data points are defined by the Euclidean distance.

2.3.2 Classical metric multidimensional scaling (classical metric MDS)

This section is inspired by Lee and Verleysen (2008) with some slight change of notation.

Classical metric MDS is another method for dimensionality reduction and visualisation of data. Like PCA it is a linear dimensionality reduction due to the implementation of a linear projection of data onto a new vector space. The key feature of classical metric MDS is that it is scalar product preserving between data points. Since scalar products induce norms which in turn induce distances, classical metric MDS also preserves distances and angles. Hence, classical metric MDS can be motivated by the fact that it maps high dimensional data into a lower dimension while preserving the scalar product structure in the high dimension.

2.3.2.1 Mathematical formulation

To put classical metric MDS into a mathematical framework we assume the same notations as in the mathematical formulation of PCA. That is, $\mathbf{X}$ is the matrix containing the original data of D rows and N columns, $\hat{\mathbf{X}} = \mathbf{P}^T\mathbf{X}$ the low dimensional representation of $\mathbf{X}$ of d rows and N columns. Furthermore, we assume both of these matrices to have zero mean in each dimension. Now, introduce the Euclidean scalar product notation between columns $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ in $\hat{\mathbf{X}}$ as in equation (23). Note that it is possible to use other scalar products, but for this thesis only the Euclidean is used.

$$s_{\hat{\mathbf{x}}}(i,j) = s(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) = \langle \hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j \rangle \tag{23}$$

Furthermore, write the following matrix, called the Gram matrix, in the following notation.

$$\mathbf{S} = [s_{\hat{\mathbf{x}}}(i,j)]_{1 \leq i,j \leq N} \tag{24}$$

$$= \hat{\mathbf{X}}^T\hat{\mathbf{X}} \tag{25}$$

$$= [\mathbf{P}^T\mathbf{X}]^T[\mathbf{P}^T\mathbf{X}] = \mathbf{X}^T\mathbf{X} \tag{26}$$

Next we perform an eigenvalue decomposition of $\mathbf{S} = \mathbf{U}\Lambda\mathbf{U}^T = [\Lambda^{1/2}\mathbf{U}^T]^T[\Lambda^{1/2}\mathbf{U}^T]$ where $\mathbf{U}$ is an orthonormal matrix of N rows and columns and Λ is an diagonal matrix of N rows and columns containing the eigenvalues. This is possible since $\mathbf{S}$ is real and symmetric. By using this eigenvalue decomposition $\hat{\mathbf{X}}$ can be retrieved in the following manner since $\mathbf{I}_{d\times N}$ selects the first d columns of $\Lambda^{1/2}\mathbf{U}^T$.

$$\hat{\mathbf{X}} = \mathbf{I}_{d\times N}\Lambda^{1/2}\mathbf{U}^T \tag{27}$$

As mentioned before this theory section there is a link between PCA and classical metric MDS. This can now be shown by starting from this last expression. From the theory section of PCA it holds that.

$$\hat{\mathbf{X}}_{\text{PCA}} = \mathbf{P}^T\mathbf{X} = \mathbf{I}_{d\times D}\mathbf{V}^T\mathbf{X} \tag{28}$$

This is due to PCA maximizing the global variance when $\mathbf{P}$ is set to have columns equal to the first d eigenvectors of C_X corresponding to the d largest eigenvalues. We now make a singular value decomposition of $\mathbf{X} = \mathbf{V}\Lambda^{1/2}\mathbf{U}^T$ which implies that the following holds.

$$\begin{aligned}\hat{\mathbf{X}}_{\text{MDS}} &= \mathbf{I}_{d\times N}\Lambda^{1/2}\mathbf{U}^T \\ &= \mathbf{I}_{d\times N}\mathbf{V}^T[\mathbf{V}\Lambda^{1/2}\mathbf{U}^T] \\ &= \mathbf{I}_{d\times N}\mathbf{V}^T\mathbf{X} = \hat{\mathbf{X}}_{\text{PCA}}\end{aligned} \tag{29}$$

This proves that both classical metric MDS and PCA maximize the global variance in the lower dimensional space. This relationship between classical MDS and PCA only holds when the Euclidean inner product is used. When any other inner product is used for the Gram matrix, then it is not equal to $\mathbf{X}^T\mathbf{X}$, which is necessary for PCA to be equal to classical metric MDS. This relationship between PCA and classical metric MDS is important since it allows for computational gains in some scenarios and for a maximization problem interpretation of classical metric MDS.

It is important to note that classical metric MDS works on inner products. It is not rare that data is given as pairwise distances between points rather than the inner products. To convert the distances into the inner products for classical metric MDS, we use a technique called double centring. For details on double centring, see chapter 4 in Lee and Verleysen (2008).

2.3.3 SNE and t-SNE (Stochastic neighbour embedding)

The two following sections are inspired by Van der Maaten and Hinton (2008). Furthermore, deeper and better understanding on the subject was provided by reading the thesis of Wängberg (2020). The theory involving information theoretic basics is inspired by Cover and Thomas (2006).

To further study dimensionality reduction techniques and data visualization, non-linear and more advanced techniques are presented. However, it should be noted that more advanced techniques are generally not always better. Linear techniques are fast, easy to implement and powerful when the mapping between the higher dimension and the lower one is linear. The non-linear techniques of this section are called SNE and t-SNE, where t stands for the t distribution. According to the original founders of these two methods, their strength stems from the attempt to preserve the local neighbourhood of points in the high dimension rather than preserving all distance relationships in the lower dimension. However, since SNE and t-SNE focuses on preserving the local structures of the high dimension in the lower one, the hierarchy structure between global structures is not reliable. To really understand the mathematics behind t-SNE, it is good to first get an overview of the theoretical framework for the SNE.

2.3.3.1 Mathematical formulation: SNE

Let $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ be the original data points in a high dimensional space. This is the space for which we want to find a lower dimensional representation $\hat{\mathcal{X}} = \{\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N\}$. For the sake of consistent notation we assume that the higher dimensional space is of D dimensions and the lower of d. Furthermore, introduce the following notation for the scaled squared Euclidean distance between the two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ as

$$d(\mathbf{x}_i, \mathbf{x}_j)^2 = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2} \tag{30}$$

It is good to note that this is only one type of distance and the choice of other distances (such as the Manhattan distance) will yield other results. The reason this is the distance for original SNE is because it is fast and easy to calculate compared to other distances. This is advantageous when this distance is appropriate for the underlying manifold that is locally Euclidean. However, it should be noted that the use of the Euclidean distance between points might not be useful if the manifold that the data points reside in is naturally non-linear on a larger scale. For example, if the manifold is a curved plane then the Euclidean distance easily creates short cuts between points on the opposite sides of the curve. It is called a short cut since the natural distance should follow the curve of the manifold. These short cuts are not too problematic on a very local scale, but when they are used over far distances, global structures between points are distorted. To avoid this, it can be useful to implement shape aware distances that preserve local and global structures.

One such distance is the commute time distance presented in later sections. For an illustration of short cuts in an S-curved manifold see Figure 7.

Lastly, the parameter σ_i in the expression will be chosen to some appropriate value which is discussed later in this section, for now we assume it to be a known value. For the time being it is enlightening to think of σ_i as a parameter controlling the local neighbourhood size.

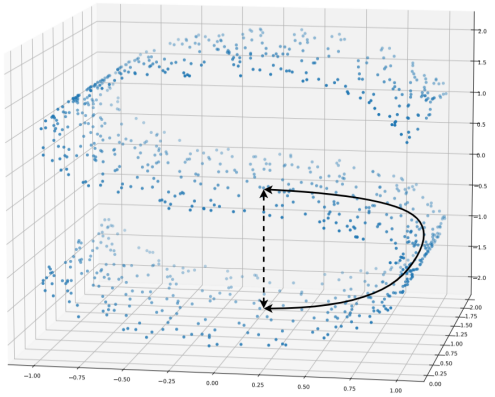


Figure 7: A data set residing on a S-curved manifold. The dotted line shows a shortcut created when using Euclidean distance, while the full curved line shows the true distance when following the manifold structure.

Next, one introduces a measure of similarity between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ as the following conditional probability, $p_{j|i} = \mathbb{P}(\{\mathbf{x}_i \text{ chooses } \mathbf{x}_j \text{ as neighbour}\})$. A similarity measure is a way of quantifying how similar two elements are. This means that a similarity measure is larger or equal to zero and smaller for elements that are different. Now, SNE sets this conditional probability to be represented by a Gaussian distribution centred at $\mathbf{x}_i$, meaning that the similarity is measured by the density of this Gaussian. The reason for the choice of a Gaussian distribution is due to the rotational symmetry and exponential decay in density. In other words, the similarity is invariant under rotation and strongly favours local neighbours in terms of giving them a much higher density (given a reasonable value on the standard deviation, σ_i). In mathematical notations introduce equation (31) where the sum is acting as a normalizing constant.

$$p_{j|i} = \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_k\|^2}{2\sigma_i^2}\right)} \tag{31}$$

For the lower dimensional representation, SNE defines a similarity measure between $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ as the following conditional probability with the same interpretations and properties, but with $\sigma_i = 1/\sqrt{2}$ for $i \in \{1, ..., N\}$

$$q_{j|i} = \frac{\exp\left(-\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_k\|^2\right)} \tag{32}$$

Since SNE is only concerned with what the neighbours of each point is, set $p_{i|i} = q_{i|i} = 0$ for $i \in \{1, ..., N\}$, meaning that a point cannot be its own neighbour.

What is left for a complete formulation of SNE, is to define a way to choose σ_i. First, note that the σ_i is the standard deviation in the Gaussian distribution. This means that for very large values on σ_i the density is more evenly spread out over the support. For small values the density is concentrated to the points (support) close to the mean. In other words, the choice of σ_i controls the density of the neighbours (close points) of the point in consideration. Since there are as many σ_i:s to choose as there are points in the data set, one needs a systematic way of choosing the different σ_i:s such that they give large densities to approximately the

same number of points. That is, the σ_i:s should be selected in a way such that they consider approximately the same number of points as close neighbours.

One method of doing this uses information theory. Hence, it is necessary to introduce some of the theoretical framework from information theory, namely Shannon entropy. Shannon entropy is a measure of uncertainty in a random variable. For some intuition it is easier to consider a discrete random variable, since in this case the Shannon entropy amounts to the expected number of binary questions needed to know the outcome of this random variable. Following this intuition, the Shannon entropy for discrete random variables is the largest for the uniform distribution since all outcomes are equally likely. This means Shannon entropy also is a measure of uncertainty since the uniform distribution gives no indication of what outcome is more likely. That is, no outcome is more favourable to expect.

For a mathematical formulation, consider a discrete random variable Z with probability distribution $p_Z(z) = \mathbb{P}(Z = z)$ with support $\mathcal{Z}$. Then the Shannon entropy for Z is given by equation (33).

$$H(Z) := -\sum_{z \in \mathcal{Z}} p_Z(z) \log_2(p_Z(z)) \tag{33}$$

If Z is continuous the entropy is defined by the integral over the support of Z, the density function of Z and with $\ln(.)$ instead of $\log_2(.)$. Switching the base of the logarithm changes the unit in which the entropy is measured. However, one can translate between these units since $\ln(x) = \log_2(x)/\log_2(e)$. When we study the entropy for continuous random variables, the entropy is the largest for the Gaussian distribution for a given variance. It is also known that increasing the variance will increase the entropy for the Gaussian, which is intuitive since a higher variance increases the uncertainty.

With Shannon entropy defined we can select σ_i such that the perplexity of the conditional distribution $p_{j|i}$ is equal to a set value. That is, choose σ_i such that the following holds for some fixed constant C.

$$\text{Perplexity}(p_{j|i}) = \text{Perp}(p_{j|i}) = 2^{-\sum_{j=1}^{N} p_{j|i} \log_2(p_{j|i})} = C \tag{34}$$

One can loosely view the perplexity (which is a monotonically decreasing function) as a measurement of the effective number of neighbours in the local neighbourhood of $\mathbf{x}_i$. To understand why, note that the following holds

$$\begin{cases} -\sum_{j=1}^{N} p_{j|i} \log_2(p_{j|i}) = -1 \log_2(1) = 0 & \text{if } p_{j|i} = 1 \text{ for some } j \\ -\sum_{j=1}^{N} p_{j|i} \log_2(p_{j|i}) = N \frac{1}{N} \log_2(N) = \log_2(N) & \text{if } p_{j|i} = \frac{1}{N} \ \forall j \in \{1, ..., N\} \end{cases} \tag{35}$$

This in turn implies that the following in equation (36) holds where the upper bound occurs when $\sigma_i \to \infty$ and the lower bound when $\sigma_i \to 0$. Now, recall that σ_i is the standard deviation of a Gaussian distribution. Hence, equation (36) can be understood in the following way: increasing the standard deviation increases Shannon entropy. Since $p_{j|i} = \frac{1}{N} \ \forall j \in \{1, ..., N\}$ corresponds to the case when Shannon entropy is maximized, the upper bound must correspond to the case of maximal standard deviation. In other words, when σ_i increases the perplexity goes towards the total number of data points at the same time as the density of the Gaussian distribution is spread out over its support. To be specific, each point is given the same density and thus considered to be a local neighbour. That is, increasing σ_i makes more points local neighbours (which can maximally be N) while it forces the perplexity to converge towards its upper bound which is N. On the other hand, decreasing σ_i makes fewer points considered local neighbours while the perplexity approaches 1.

$$1 \leq \text{Perp}(p_{j|i}) \leq N \tag{36}$$

To summarise, σ_i controls the size of the neighbourhood of each point in the high dimension. Since SNE sets the neighbourhood sizes in the high dimension it is unnecessary to account for the neighbourhood size in the low dimension. Furthermore, by choosing σ_i to satisfy equation (34) one ensures that the neighbourhood

of each point consists of the same number of neighbours and adjusts the Gaussian distribution accordingly. This makes different neighbourhoods comparable.

The core idea of SNE is to minimize the difference between the conditional probabilities $p_{j|i}$ and $q_{j|i}$, since this attempts to preserve the neighbourhood structures in the high dimension when mapped into the lower one. As with all optimization problems a cost function is needed to measure the difference between $p_{j|i}$ and $q_{j|i}$. From information theory, a common way to measure differences between two distributions is the Kullback-Leibler divergence, which is going to be the cost function. Now, we introduce the definition of the Kullback-Leibler divergence as in equation (37).

$$Cost = KL(P|Q) = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{j|i}\log_2\left(\frac{p_{j|i}}{q_{j|i}}\right) \geq 0 \tag{37}$$

In equation (37), equality holds if and only if $P = Q$. An intuitive way of thinking of the Kullback-Leibler divergence is to view it as a measure of the inefficiency of assuming that the distribution is Q when the true distribution is P. As a cost function it has some desirable properties. It yields a large cost for data points that are close in the high dimension but are mapped far apart in the lower dimension. That is, to use a small $q_{j|i}$ for a large $p_{j|i}$. A small cost is caused when using a large $q_{j|i}$ for a small $p_{j|i}$. This cost function punishes mappings that do not preserve local structures in the higher dimension, which is desirable.

For a full solution to the optimization problem, it is needed to find the minima of the cost function, $KL(P|Q)$, defined above. To find this minima, gradient descent is implemented. For this to be a viable approach, the partial derivative of $KL(P|Q)$ are needed with respect to $\hat{\mathbf{x}}_i$ for all $i \in \{1, ..., N\}$. The partial derivative is given as in equation (38). To start the optimization the points in the lower dimensional representation are randomly initialized using a Gaussian distribution.

$$\frac{\partial KL(P|Q)}{\partial \hat{\mathbf{x}}_i} = 2\sum_{j=1}^{N}(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j). \tag{38}$$

To avoid local minima trapping, add a momentum term to the ordinary gradient descent iteration procedure. That is, we use an extra term to the gradient descent update which adds an exponentially decaying sum of gradients from previous iterations. The full iteration scheme is shown in the equation below where $\hat{\mathbf{X}}^{(t)}$ is the solution at iteration step number t, η the learning rate and $\alpha(t)$ the momentum at iteration t.

$$\hat{\mathbf{X}}^{(t)} = \hat{\mathbf{X}}^{(t-1)} - \eta\frac{\partial KL(P|Q)}{\partial \hat{\mathbf{X}}} + \alpha(t)[\hat{\mathbf{X}}^{(t-1)} - \hat{\mathbf{X}}^{(t-2)}] \tag{39}$$

When the optimization has been done with a satisfactory result, the lower-dimensional representation is found in the last $\hat{\mathbf{X}}^{(t)}$. This is the solution found to minimize the Kullback-Leibler divergence between the higher and lower dimensional distributions.

2.3.3.2 Mathematical formulation: t-SNE

With ordinary SNE detailed, I will discuss the theory of the t-SNE method in this section. Although ordinary SNE is a good method to achieve dimensionality reduction and data visualisation, it has a few weaknesses that are resolved by t-SNE. The main aspect that makes t-SNE more useful is that it allows for better separation of distant neighbours while preserving the distance between close neighbours. Hence, t-SNE focuses even more on preserving local structures in the higher dimension when mapping to the lower one.

The key difference that sets t-SNE apart from ordinary SNE is the use of joint probability distributions instead of conditional distributions in high and low dimensions. This means that one redefines the cost function (Kullback-Leibler divergence) as the following for t-SNE.

$$Cost = KL(P|Q) = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{j,i}\log_2\left(\frac{p_{j,i}}{q_{j,i}}\right) \geq 0 \tag{40}$$

This new cost function introduces symmetry in neighbourhood distributions since $p_{i,j} = p_{j,i}$ and $q_{i,j} = q_{j,i}$ for $i, j \in \{1, ..., N\}$. We now let $p_{j|i}$ and $q_{j|i}$ represent the conditional distributions presented in the theory section for ordinary SNE. Furthermore, make the following definition of $p_{i,j}$ which does not violate the symmetry condition discussed above.

$$p_{i,j} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{41}$$

With this definition it is ensured that no $p_{i,j}$ is too small when either data point $\mathbf{x}_i$ or $\mathbf{x}_j$ is an outlier. An outlier causes the probabilities $p_{i,j}$ to be close to zero if defined as in equation (42). Too small probabilities in the high dimensional space will cause the low dimensional representation corresponding to the outlier to have close to no effect on the cost function. That is, the lower-dimensional mapping does not affect the cost. This property defeats the purpose of the method since the low dimensional mapping of outliers is then random. To see why this is the case, we observe that all $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ will be large for outliers $\mathbf{x}_i$, which causes $p_{i,j}$ to be very small if naively defined as follows.

$$p_{i,j} = \frac{\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)}{\sum_{k\neq l}\exp\left(-\frac{\|\mathbf{x}_l - \mathbf{x}_k\|^2}{2\sigma^2}\right)} \tag{42}$$

Now, by using the definition in (41) instead, it is observed that no $p_{i,j}$ is too small since $\sum_{j=1}^{N} p_{i,j} > \frac{1}{2N}$, meaning that each point contributes to the cost function.

It is now necessary to define the probability $q_{i,j}$ for the lower dimension. It is possible to use a similar Gaussian structure as in equation (42), but with $\sigma = \frac{1}{\sqrt{2}}$. However, instead of applying this approach, the probability is chosen to be the longtailed t-distribution with one degree of freedom. In other words, we define $q_{i,j}$ as follows.

$$q_{i,j} = \frac{(1 + \|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\hat{\mathbf{x}}_k - \hat{\mathbf{x}}_l\|^2)^{-1}} \tag{43}$$

This causes a mismatch between the probabilities in the high dimensional space and the low dimensional space at large distances. With the t-distribution one has similar probabilities as in the Gaussian for small distances. For moderate distances, a larger probability is given by the t distribution compared to the Gaussian. That is, two moderately distant points that are equidistant in low and high dimensional space will be given quite different probabilities due to the mismatch in distribution. This means that the neighbourhoods in the lower dimensional space have neighbours that are already moderately far apart in the high dimension even farther apart in the low dimension. The reason for this is because in order to minimize the cost function, far away points in the high dimension need to be modelled even farther away in the low dimension to achieve matching probabilities. An effect of this is that the t-distribution allows for better separation of distant neighbours while preserving the distance between close neighbours. Furthermore, it alleviates the crowding problem which is caused by smaller dimensions being too small to faithfully preserve close distance relations in higher dimensions. Despite this new t-distribution, the cost function is still affected in the same way as in the ordinary SNE. That is, a large cost is incurred for modelling small distances in high dimensional space with large distances and small costs are incurred to model large distances with small ones. This means that t-SNE focuses more on preserving local distances rather than distances between distant points.

With both the high dimensional and low dimensional probabilities defined and motivated, one can solve the optimization problem by gradient descent. As before, first find the partial derivatives which are given by the following expressions and then plug these into equation (39).

$$\frac{\partial KL(P|Q)}{\partial \hat{\mathbf{x}}_i} = 4\sum_{j=1}^{N}(p_{i,j} - q_{i,j})(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j)(1 + \|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|^2)^{-1} \tag{44}$$

By presenting PCA, classical MDS and t-SNE, three methods for data visualisation and dimensionality reduction are available for studying the layers in the neural networks. For the next section cluster analysis is the main focus. The reason this section is needed is because it yields a framework for analysing how well neural networks can group data points of the same class labels.

2.4 Cluster analysis

The brief overview of the method of clustering is inspired by Hastie (2009) *et al.* All theory involving graph Laplacians and commute time distances is inspired by Von Luxburg (2007). The section on silhouette theory is inspired by Tan (2019) *et al.*

This section gives a brief overview of cluster analysis and how to evaluate cluster quality. The aim of cluster analysis is to place elements into groups such that elements in the same group exhibit similar characteristics, while elements of different groups do not. In order to achieve this it is necessary to be able to measure how similar elements are to each other. This is often measured by some predefined measure called similarity measure. Hence, for a good clustering the elements assigned to the same cluster should have a high similarity measure between each other, but a low measure for elements assigned to other clusters. Alternatively, one can define clustering using dissimilarity measures since it is the opposite of similarity measures. There are many popular choices of dissimilarity, one of which is the Euclidean distance. This is often used when the clustering elements can be represented by vectors in a vector space.

As with most fields in mathematical statistics, cluster analysis has its challenges. One of the main challenges for a lot of the existing clustering algorithms is to define how many clusters to assign elements to. As an example, given a bag of candies one can allocate candies into two clusters, chocolate and liquorice. However, there might be further and more accurate clusters, such as chocolate, fruity and liquorice. When clustering is applied to the data in this thesis, the data provides the luxurious information of how many clusters that are actually present.

For a mathematical formulation of this theory section and cluster evaluation we need to introduce some notation. Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be two elements in the data set, $\mathcal{X}$, that clustering is applied to. Denote the dissimilarity measure between the two elements by $d(\mathbf{x}_i, \mathbf{x}_j)$. Furthermore, let N_c denote the number of clusters and C_i for $i \in \{1, ..., N_c\}$ the set of clusters.

2.4.1 Graph Laplacian

This section concerns graph Laplacians and their properties. It serves as a preliminary to the next section on a dissimilarity or distance metric called commute time distance. Graph Laplacians are used to represent graphs which essentially are data structures explaining how different data points are related to each other via connections called edges. Furthermore, graph Laplacians have some interesting properties which accurately unveil interesting features of the graphs. Two of these properties are the number of connected components and the commute time distances of the graphs. To extract this information about the graph it is enough to study the eigen-spectrum of the graph Laplacian. That is, even for large and impossible to visualize data sets, one can unveil properties of a graph representation of the data by studying the eigen-spectrum. With this motivation I now introduce the mathematics of the graph Laplacian restricted to undirected graphs.

2.4.1.1 Mathematical formulism

This section introduces the theoretical aspect of the graph Laplacians for undirected graphs. The section focuses on undirected graphs since these are later used for symmetric similarities between the vertices in the graph. First let $G = (V, E)$ denote the undirected graph constituted of the vertex set $V = \{\mathrm{v}_1, ..., \mathrm{v}_N\}$ and the edge set E. In the application part of this thesis the vertices represent data points which are high dimensional matrices. Furthermore, $\mathbf{W}_G$ represents the weight matrix of N rows and columns with elements

$w_{i,j} = w_{j,i} \geq 0$ due to the graph being undirected. When $w_{i,j} = w_{j,i} = 0$ there is no edge between vertices v_i and v_j. This means that $\mathbf{W}_G$ is a symmetric matrix. Now denote the diagonal degree matrix by $\mathbf{D}$ with $d_i = \sum_{j=1}^{N} w_{i,j}$ at element (i, i). A subset $U \in V$ is called a connected component if there is no edge between vertices in U and vertices in its complement, U^c, and if there exists a path of vertices in U between any two vertices also in U.

With the graph G defined as above, introduce the graph Laplacian as the matrix $\mathbf{L}$ of N rows and columns in equation (45). Since both $\mathbf{W}_G$ and $\mathbf{D}$ are symmetric matrices of the same dimensions, $\mathbf{L}$ is also symmetric.

$$\mathbf{L} = \mathbf{D} - \mathbf{W}_G \tag{45}$$

The graph Laplacian has several properties that are of interest to this thesis. The main property of the graph Laplacian that is needed is the result in equation (46), for a detailed proof see equation (5) in the appendix. This also proves that the graph Laplacian is positive semi-definite. Since it is now known that $\mathbf{L}$ is positive semi-definite it must be that the following holds for its eigenvalues $0 = \lambda_1 \leq ... \leq \lambda_N$. See equation (6) in the appendix for a complete proof. The smallest eigenvalue must also be exactly equal to zero since the graph Laplacian is not of full rank, which is seen by summing over any row which will yield zero. This implies that the kernel of $\mathbf{L}$ is not the empty set.

$$\mathbf{x}^T L \mathbf{x} = \frac{1}{2}\left[\sum_{i,j=1}^{N} w_{i,j}(x_i - x_j)^2\right] \geq 0, \quad \mathbf{x} = (x_1, ..., x_N)^T \in \mathbb{R}^N \tag{46}$$

With these key properties one can also prove an interesting result for the number of connected components in G. The number of connected components in the graph corresponds to the multiplicity of the eigenvalue zero of $\mathbf{L}$. For a full proof of this property, see equations (7) and (8) in the appendix.

2.4.2 Commute time distance

This section outlines the theory behind commute time distances on undirected weighted graphs. CTD, short for commute time distance, is a distance based on random walks on graphs. A random walk on a graph can be described as traversing the graph between vertices by using existing edges. At each vertex the next step or edge chosen to travel is based on a probability proportional to the edge weight. That is, each step in the traversal is chosen based on a distribution reciprocating the edge weights. The transition probabilities of an undirected weighted graph are described by the following probabilities from vertex v_i to v_j in the set E. With these probabilities, we may view the random walk as a Markov chain using these transition probabilities.

$$P(\mathrm{v}_j|\mathrm{v}_i) = \frac{w_{i,j}}{\sum_{j:\mathrm{v}_j \text{ connected to } \mathrm{v}_i} w_{i,j}} \tag{47}$$

The CTD between two vertices is given by the expected time it takes for a random walk to travel from one vertex to another and then back. The reason it is desirable to use CTD over other distances such as the Euclidean, is because it takes non-linear and cluster structures in the data of the graph into consideration. To clarify, when there is a certain cluster structure in the data, data points should be considered similar to each other when they belong to the same cluster and dissimilar when they belong to different clusters. When there is a non-linear structure in the data it means that the manifold that the data resides in does not have the global properties of a Euclidean space. That is, when traversing the data over long distances the path between the data points is not a linear path. These properties of the data points might not be captured with the Euclidean distance since it only measures the shortest absolute path between two points. It is a bit similar to the short cut example shown in Figure 7.

To exemplify, two points A and B in the same cluster might be far away in Euclidean distance and close to a point, C, in another cluster. However, despite the shorter distance to C from A and B, it might be

harder to reach C. It might be because there are many more connections that one can use to travel from A to B compared to A to C or B to C. That is, since there probably are many more points in the same cluster between A and B there are many more edges to use to travel between A and B compared to the one edge from A to C and B to C. CTD uses the abundance of edges when travelling between points to measure distances. In other words, when a point has many routes to another point the CTD will be smaller. Since points in the same cluster tend to have more routes in-between compared to points in different clusters, the CTD within clusters tend to be smaller. That is, CTD is a distance that respects the underlying cluster structure or hierarchy. Furthermore, since CTD sums up the many and short edges when measuring the distance between points it also respects the non-linear structure in the data. To be more specific, it will not sum long short cuts between points to measure distances since these edges are not common in the graph. For a graphical illustration see Figure 8.

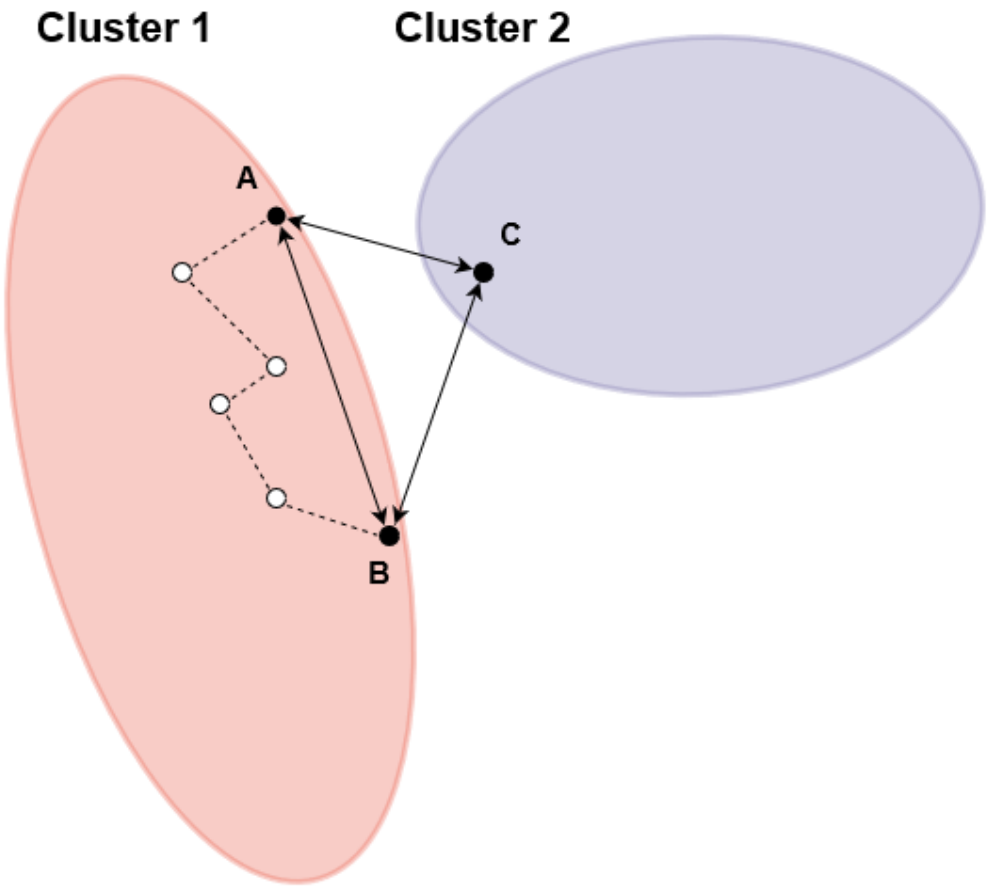


Figure 8: Example of points A and B in the same cluster being farther from each other in Euclidean distance, but close to point C in another cluster. The CTD uses points in the same cluster as A and B to reveal that they are in fact closer to each other than to point C, due to the more accessible route via the white points. The CTD will also follow the non-linear route between A and B when summing up the distances between them. This respects the underlying manifold structure.

2.4.2.1 Mathematical formulism

The CTD can be quite tedious to calculate using the theory of random walks directly. However, with the graph Laplacian it is possible with the use of standard eigenvalue computations to calculate the CTD:s on the graph. Now, let $c_{i,j}$ denote the CTD between vertices v_i and v_j in the graph G. The CTD is then given

by the formula stated in equation (48) where λ_i:s are the eigenvalues of the graph Laplacian $\mathbf{L}$ and $v_{i,k}$ the k:th component of the i:th eigenvector of $\mathbf{L}$.

$$c_{i,j} = \left[\sum_{i=1}^{N} d_i\right]\left[\sum_{k=2}^{N} \frac{1}{\lambda_k}(v_{i,k} - v_{j,k})^2\right] = \sum_{k=2}^{N}\left[v_{i,k}\sqrt{\frac{V(G)}{\lambda_k}} - v_{j,k}\sqrt{\frac{V(G)}{\lambda_k}}\right]^2 \tag{48}$$

Equation (48) is a simplification of a formula given in proposition (6) in Von Luxburg (2007). The proof of the formula presented in proposition (6) in this article can be found in Klein and Randić (1993). Since the proof is very technical and not a main result of this thesis, it is omitted.

The constructed graph G is in this report constructed using undirected k nearest neighbour graphs (kNN). That is, an undirected edge is created between two vertices if one of them is in the kNN neighbourhood of the other. The distance to measure closeness of vertices is the Frobenius distance, $F_{i,j}$, which is presented in equation (49). The reason for using this approach is due to the graph Laplacian requiring a symmetric weight matrix. Furthermore, this approach is also desirable since it for small neighbourhoods do not create a lot of short cuts between vertices. Avoiding short cuts in this setting means that it makes it harder to traverse the graph between clusters or between points which have more natural passages through existing clusters. This is desirable since it takes the cluster structure in the data into consideration, since passages and therefore small CTD:s are promoted inside clusters. A sensible way of choosing the size of the neighbourhood is presented later on.

Furthermore, the weight matrix, $\mathbf{W}_G$, consists of similarity measures between all the data points. The similarity measure used in this case is the t-distribution similarity with the Frobenius distance given in the first line of equation (49). The Frobenius distance is used since it is a generalisation of the Euclidean distance in higher dimensions. In mathematical terms this equates to the following where a is a constant, $\mathbf{x}_j$ and $\mathbf{x}_k$ are points in some arbitrary matrix space, $\mathbb{R}^{d_1 \times d_2 \times d_m}$ with components $x_{j,i_1,\ldots,i_m}$, (where m equals to 2 in ordinary matrix data, see sections 2.2.1 and 2.2.2 for details on matrix data) corresponding to vertices in the graph.

$$\begin{aligned} w_{j,k} = w_{k,j} &= \frac{1}{1 + \frac{F_{j,k}^2}{a^2}} \\ F_{j,k} &= \sqrt{\sum_{i_1=1}^{d_1} \ldots \sum_{i_m=1}^{d_m} (x_{k,i_1,\ldots,i_m} - x_{j,i_1,\ldots,i_m})^2} \end{aligned} \tag{49}$$

The reason we use the t distribution similarity as weights in the graph Laplacian is because it only requires the estimation of one parameter, the constant a, for all data points. This can be compared to the Gaussian similarity which has a local parameter σ_i^2 for each point in the graph. Both the Gaussian and the t distribution similarity increase the number of parameters to estimate, but the Gaussian similarity also defeats the purpose of the kNN graph. Since the kNN graph already creates a local neighbourhood for each vertex in the graph it does not make much sense to also assign neighbourhood-based similarities. The reason is that local similarity might alter the underlying neighbourhood created in the kNN which is desirable to preserve.

Furthermore, much like the Gaussian similarity used in the t-SNE method it has an exponential decay meaning that it is good at separating vertices close to the centre. However, the downside of the t distribution is that the weights for distances (even though being largely different) far out in the tail are hardly different due to the slow decay. To resolve this issue, it is good to choose a value for the constant a in a clever way.

The method for choosing the constant a in this thesis is to solve equation (50) where $dist_{90\%}$ is the empirical 90% percentile of the distances to the k:th farthest neighbour of the vertices in the graph. In the equation 9.472139 corresponds to the 90% percentile of the squared t distribution with one degree of freedom.

$$\frac{dist_{90\%}}{a} = 9.472139 \tag{50}$$

First, the reason for using the square of the t distribution is because the t distribution similarity is using squared distances divided by a squared constant a. Since the t distribution is for a non-squared variable one can account for this mismatch by instead considering the square of the t distribution. By using the table for the non-squared t distribution with one degree of freedom, one can see that the 90% percentile is equal to 3.077684. By using the equivalence in equation (51) it can be derived that 9.472139 is the 90% percentile of the squared t distribution.

$$\mathbb{P}(T \leq x) = 0.9 \;\Leftrightarrow\; \mathbb{P}(T^2 \leq x^2) = 0.9 \tag{51}$$

The reason why choosing the constant a in this way is desirable is because it allows for greater differences in similarities for distances far out in the tail of the t distribution similarity. That is, if the empirical 90% percentile of the distances of the k:th farthest neighbour is too small compared to the one of the squared t distribution then $a < 1$, else $a > 1$. The effect of this is that one gets more different similarities for larger distances (except the extreme outlier cases), which might not be achievable due to the slow decay of the tail in the t distribution similarity. This is achieved by the constant a shifting the distances in order to spread them out over the part of the t distribution similarity where the decay is rapid. Using a larger percentile than 90% is possible but can lead to the solution being too affected by outliers. That is, the separation might only be good for very extreme distances which are very few in comparison. To summarize, the end goal of choosing a in this way is to get better separation far out in the tail of the t distribution similarity, meaning that it is easier to separate vertices.

For the choice of k in the kNN graph, one uses the result for the graph Laplacian regarding the number of connected components. More specifically, one makes the choice of k to be equal to the smallest value for which the multiplicity of the eigenvalue zero is equal to one. In other words, the first time k yields one single connected component in the graph. This means that it is possible to create edges between all vertices while being sure not to create too many connections between vertices.

2.4.3 Cluster evaluation: Method of silhouette coefficients

This section presents a method for evaluating the quality of a clustering by combining two measures called cohesion and separation. For a data point $\mathbf{x}_i \in \mathcal{X}$ perform the calculations in equations (52a) to (52c). $a(\mathbf{x}_i)$ in equation (52a) is called cohesion since it measures the average similarity between the elements in the cluster $\mathbf{x}_i$ belongs to. $b(\mathbf{x}_i)$ is called separation since it measures the smallest average dissimilarity between $\mathbf{x}_i$ and all points belonging to a separate cluster. $s(\mathbf{x}_i)$ is called the silhouette coefficient and is a value between -1 and 1, where more negative values indicate that $\mathbf{x}_i$ is harder to accurately assign a cluster. This is because it is negative when the average dissimilarity is large between points in the same cluster and the average dissimilarity to points in the closest cluster is low. Lastly one can evaluate the silhouette coefficient for a whole cluster by calculating $\frac{1}{|C_i|}\sum_{\mathbf{x}_k \in C_i} s(\mathbf{x}_k)$ and for the whole clustering by calculating $\frac{1}{N}\sum_{\mathbf{x}_k \in \mathcal{X}} s(\mathbf{x}_k)$.

$$a(\mathbf{x}_i) = \frac{1}{|C_i| - 1} \sum_{\mathbf{x}_j \in C_i; i \neq j} d(\mathbf{x}_i, \mathbf{x}_j) \tag{52a}$$

$$b(\mathbf{x}_i) = \min_{k \neq l} \left[\frac{1}{|C_k|} \sum_{j \in C_k} d(\mathbf{x}_i, \mathbf{x}_j) \right] \quad \text{where} \quad x_i \in C_l \tag{52b}$$

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max(a(\mathbf{x}_i), b(\mathbf{x}_i))} \tag{52c}$$

It is interesting that the silhouette coefficient is close to zero when the correct clustering structure is equivalent to random clustering. That is, if the true generating clustering distribution assigns equal probability to a new sample belonging to any of the clusters, then the silhouette coefficient is close to zero. The reason for this is because points are about the same distance from each other no matter the assigned cluster. This leads to similar values of the equations (52a) and (52b). By similar reasoning does a silhouette coefficient equal to minus one corresponds to the case when one actively put points in the most dissimilar cluster.

It is often the case when using silhouette coefficients for cluster evaluation that the true number of clusters in the data is unknown. This also implies that the true cluster memberships of the data points are unknown. However, in this thesis the true number of clusters and cluster memberships are known since for each data point the true class label is given. Despite knowing the true number of clusters and cluster memberships, the silhouette coefficient is still a relevant measure. The only change is that the silhouette coefficients respect and evaluate the true cluster memberships of the data points. That is, in this thesis cluster evaluation is equivalent to evaluation of the grouping of data from different classes.

3 Prequel

The following sections are specific to the data and model used in this thesis. To understand them, a good knowledge of the theory section is needed. The idea is to present the specific data and model in detail. Then the model trained on the data is studied and explained in the preliminary study and result section. This part is concluded by a section describing further studies and improvements of the results and method.

4 Data

The aim of this thesis is to unveil the inner working mechanisms of the state-of-the-art CNN model called ResNet-18. The model is trained and tested on the well-established benchmark dataset called CIFAR-10. For the original source and details on the CIFAR-10 dataset see chapter 3 in Krizhevsky (2009). CIFAR-10 is an image data set consisting of RGB images of size 32×32 pixels (elements), where RGB stands for the standard colour channels used for images. In the notation used in this thesis, each image is viewed as a $3 \times 32 \times 32$ matrix.

The data has a total of 60000 images with 10 classes corresponding to everyday objects (the classes are: car, truck, plane, ship, dog, cat, bird, frog, horse and deer). For each class there are 6000 images, meaning that the data set is class balanced. Furthermore, the data set is divided into a class balanced test set of 10000 images and a class balanced training set of 50000 images. To get an idea of the type of images in the data set see Figure 9.

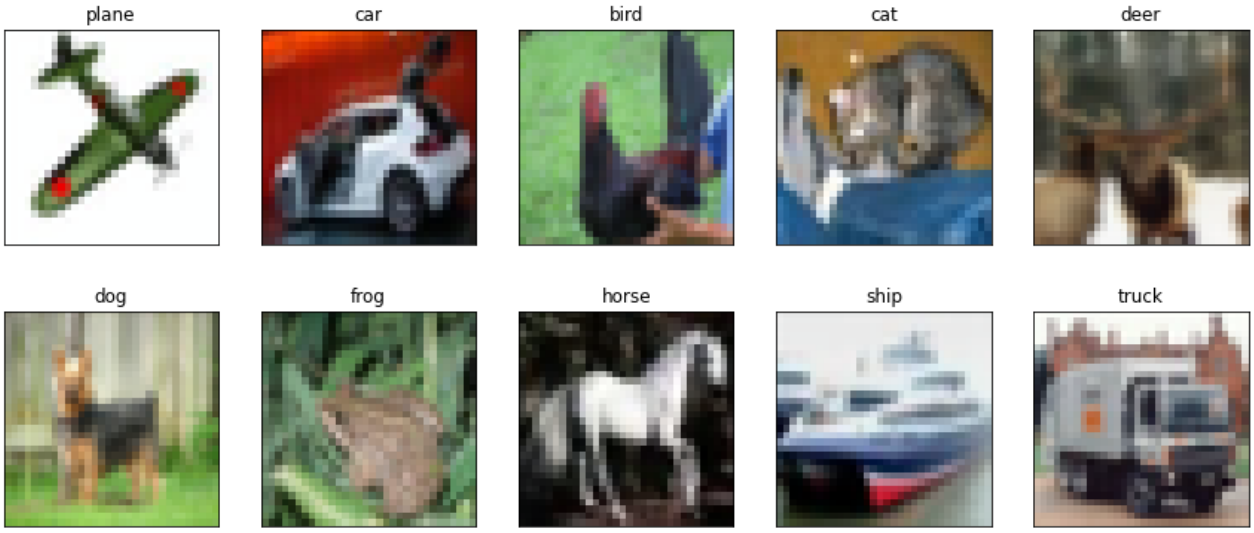


Figure 9: Example images from the CIFAR-10 data set, one from each class.

As mentioned briefly in the theory section, the network is trained in 300 epochs with mini-batch size 128. This means that the model has its parameters updated 117188 times. Lastly, some data augmentation is done for each mini-batch, meaning that each mini-batch is somewhat different from the others. The data augmentation done is called random horizontal flipping, random cropping and image normalization. Image normalization means making a channel-wise normalization which forces the pixel values to be between 0 and 1. In total, the 50000 images in the training set are augmented for each epoch. This gives $50000 \cdot 300 = 15000000$ augmentations. For details on this specific type of data augmentation see PyTorch (2022a).

5 Model description

This short section is meant to give a brief overview of the architecture of the ResNet-18 model that was briefly mentioned earlier. ResNet-18 is the smallest in the family of ResNet models. This collection of CNN:s use a repeating pattern of a building block implementing the skip connection presented in the theory section of this thesis, see section 2.2.4 in the theory section. All ResNet models are well established and achieve high accuracies on benchmark datasets when trained appropriately.

As with the rest of the ResNet family, ResNet-18 also has a block type structure which repeats itself. That is, the architecture of the model is a repeating sequence of CNN operations. In total the model has four blocks that have the same general structure and operations. Each block uses four convolutional operations, two skip connections and the ReLU activation function four times. The idea of this architecture is to be able to create a deep CNN with many convolutions while still avoiding the degradation problem and vanishing gradient. For a quick reminder, the degradation problem is the phenomena of deeper networks failing to create linear mappings and therefore achieving counterintuitive performances compared to shallower models. In total the ResNet-18 model in this thesis has 11 173 962 parameters that are trained.

An overview of the ResNet-18 architecture can be seen in Figure 10. The different blocks and additional operations are shown in this figure along with some markings. The markings are important when analysing the results in this thesis. The markings are positioned after the major blocks and important operations, which enables us to study their effects. At each marking the input image has been transformed in some way, meaning that each marking considers a representation of the input image in a different space.

The output for Block 1 is a matrix of dimensions $64 \times 32 \times 32$, $128 \times 16 \times 16$ for Block 2, $256 \times 8 \times 8$ for Block 3 and $512 \times 4 \times 4$ for Block 4. The dimension after the average pooling is $512 \times 1 \times 1$ and after the fully connected layer it is $10 \times 1 \times 1$. This means that the spatial dimensions are reduced in each block while the

channel dimension is increased. The use of many convolutions extracts more information and concentrates the information by reducing spatial dimension and increasing channel dimensions.

The training was performed using the Python code created by Liu (2021), which is made to replicate the model that is originally presented in He (2016) *et al.* The Python code created by Liu (2021) for fitting of ResNet-18 is mainly based on the commonly used package torch and library torchvision as a part of the PyTorch library, see PyTorch (2022b) and PyTorch (2022c). For this thesis the code for training is only changed to use 300 epochs instead of the original 200 epochs. The final test accuracy of the model with the specific weights from the training in this thesis is 95.36%. This means that 464 images in the test data are misclassified while the remaining 9536 images are correctly classified.

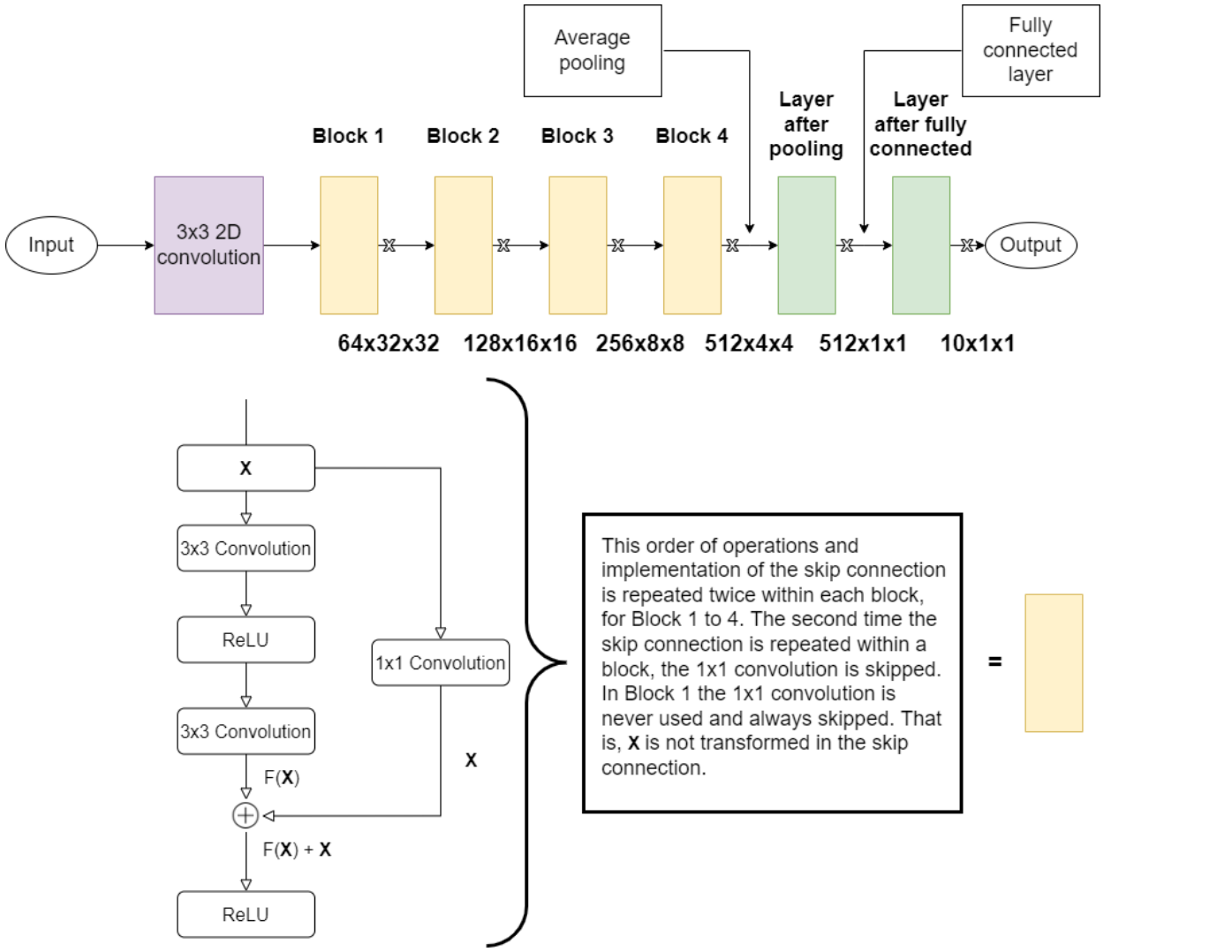


Figure 10: Overview of the ResNet-18 architecture. The bold white x:s mark the checkpoints in the model with the corresponding names in dimensions stated below. The stated dimensions are also the names of these checkpoints. The 1×1 convolution is implemented with the same number of channel dimensions as $\mathbf{x}$ has. The purpose of the operation is to make the dimensions of $\mathbf{x}$ match the dimensions of $\mathrm{F}(\mathbf{x})$ to make the addition $\mathrm{F}(\mathbf{x}) + \mathbf{x}$ possible.

To finish the model section it is later important to know more about the different parts of Block 4 in the model. The overview of the architecture of Block 4 can be seen in Figure 11. The most important part here is to get an understanding of the different checkpoints of the model which are studied later on.

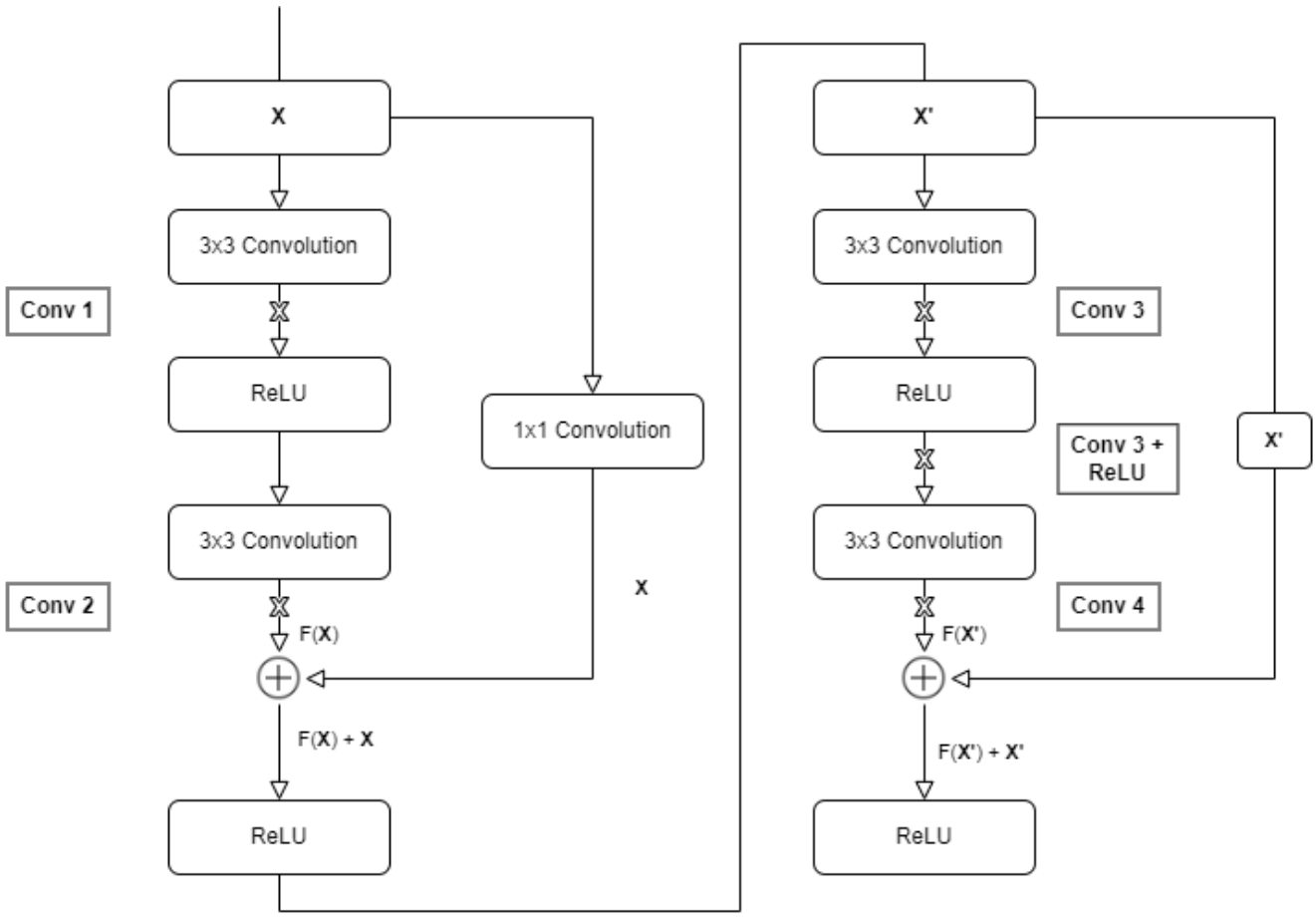


Figure 11: An architectural overview of Block 4 in the ResNet-18 architecture. The bold white x:s mark checkpoints in the model with the corresponding names in the white boxes.

6 Method

This section gives an overview of the method for unveiling the inner mechanisms of ResNet-18. It also gives a better understanding of how all the sections in the theory part come together as a tool for analysing ResNet-18.

First the ResNet-18 network (the CNN model to study in this thesis) is trained on CIFAR-10 to achieve a 95.36% accuracy on the test set (10000-464=9536 correctly classified images). After this, the unveiling of the inner mechanisms can start. Even though the accuracy is high, about 5% of the test images are still misclassified. The aim is to understand and explain more about the high performance and errors of ResNet-18. **This means that all numbers, figures and results presented below are entirely based on the test set of CIFAR-10**.

To achieve this, a preliminary study is done on the $10 \times 1 \times 1$ and $512 \times 1 \times 1$ checkpoints in the model, see Figure 10 for a reminder of the checkpoints. The preliminary study is done using classical metric MDS and t-SNE in order to get a better understanding of the clustering and class structures in these spaces. Then with the use of Euclidean distances in these spaces, silhouette plots are created to evaluate the grouping of data from different classes.

After the preliminary study the main part of this thesis and analysis can start. The idea is to use commute time distances (CTD) in order to account for non-linear structures in the data. That is, instead of using conventional distances, CTD is implemented to avoid short cut distances to respect the manifold structure.

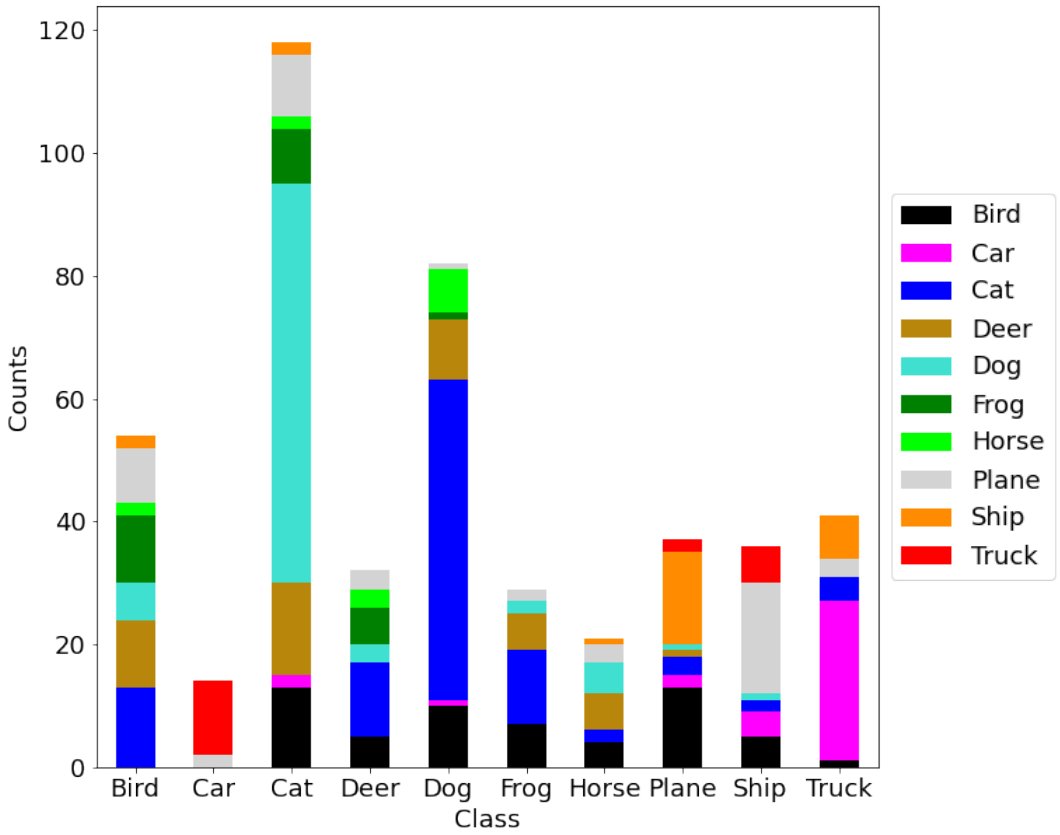
The CTD is implemented for the different checkpoints in ResNet-18 to understand where the largest increase in silhouette coefficients is gained. After this checkpoint is located an explanation using summarising metrics and histograms is formulated. For a reminder of all the checkpoints studies in the analysis, see Figures 10 and 11.

7 Preliminary study

The aim of this section is to do a preliminary study of the vector spaces at the checkpoints $10 \times 1 \times 1$ and $512 \times 1 \times 1$ in ResNet-18. This preliminary study is done to visualise and understand how ResNet-18 performs on the test set of CIFAR-10 as well as what types of errors can occur. For the implementation of t-SNE, silhouette plots and classical metric MDS the modules manifold, metric and decomposition in the Python library Scikit-learn are used, see Pedregosa (2011).

Firstly, it is interesting to study how the 464 misclassified images in the test set are distributed across the different classes. From the bar plot and the confusion matrix in Figure 12 some classes are more common for ResNet-18 to misclassify. For example birds, cats and dogs are the three most commonly misclassified classes. The classes car, horse and frog are the top three classes to correctly classify. For most classes the prediction errors (the wrong class) seem to be evenly distributed over classes. However, the prediction errors for the cat class and the dog class are often cat misclassified as a dog or dog misclassified as a cat. Based on this it can be concluded that the model fails to distinguish dogs and cats to a greater extent compared to other pairs of classes. A speculative explanation for this is that cats and dogs are of similar shapes, sizes, colours and are commonly seen in similar scenery.

Furthermore, for the inanimate classes (car, plane, ship and truck) the confusion is mainly contained within the inanimate classes, except for planes which are commonly mistaken for birds. Interestingly, birds are not commonly misclassified as planes. The explanation might be that animals and inanimate objects tend to have vastly different colours and shapes. Planes might be confused with birds since they are often pictured flying in the air in similar ways as birds. Birds on the other hand might be easier to separate from planes since it is hard to get good images of them when in the air compared to sitting still. That is, images of birds are often of them sitting still and not in the air as planes.



	Bird	Car	Cat	Deer	Dog	Frog	Horse	Plane	Ship	Truck	Total
Bird	0	0	13	11	6	11	2	9	2	0	54
Car	0	0	0	0	0	0	0	2	0	12	14
Cat	13	2	0	15	65	9	2	10	2	0	118
Deer	5	0	12	0	3	6	3	3	0	0	32
Dog	10	1	52	10	0	1	7	1	0	0	82
Frog	7	0	12	6	2	0	0	2	0	0	29
Horse	4	0	2	6	5	0	0	3	1	0	21
Plane	13	2	3	1	1	0	0	0	15	2	37
Ship	5	4	2	0	1	0	0	18	0	6	36
Truck	1	26	4	0	0	0	0	3	7	0	41

Figure 12: Bar plot and confusion matrix for the errors in the test set. The rows in the confusion matrix represent the true classes while the columns are the confused classes.

Since the true class for each image is known the silhouette coefficients can be calculated for each of them in order to further understand the performance of ResNet-18. In Figure 13 the silhouette plots for the two checkpoints of interest can be seen. These plots show the distribution of the silhouette coefficients for each class in the dataset. The thicker the smooth and coloured shapes are, the more common is the silhouette coefficient for the class. There is a difference between the two plots when going from the vector space at $512 \times 1 \times 1$ to $10 \times 1 \times 1$. In the vector space at checkpoint $10 \times 1 \times 1$ the left tails are longer and thinner while the right tails are longer and thicker. The reason for this is that the fully connected layer, which is the difference between these two vector spaces, in general better separates the classes or makes them more cohesive. The longer left tail indicates that some images are getting worse, but the thinness means that in general images are better clustered. Interestingly since the smaller vector space of ten dimensions has a better clustering, it seems like a lot of the dimensions in the larger vector space do not contain a lot of

relevant information about the classes. Not too surprising, the misclassified images marked by crosses in the figure mostly have negative silhouette coefficients. The interpretation is that they are most likely too far away from the cluster centre of its correct class and/or too close to some incorrect class.

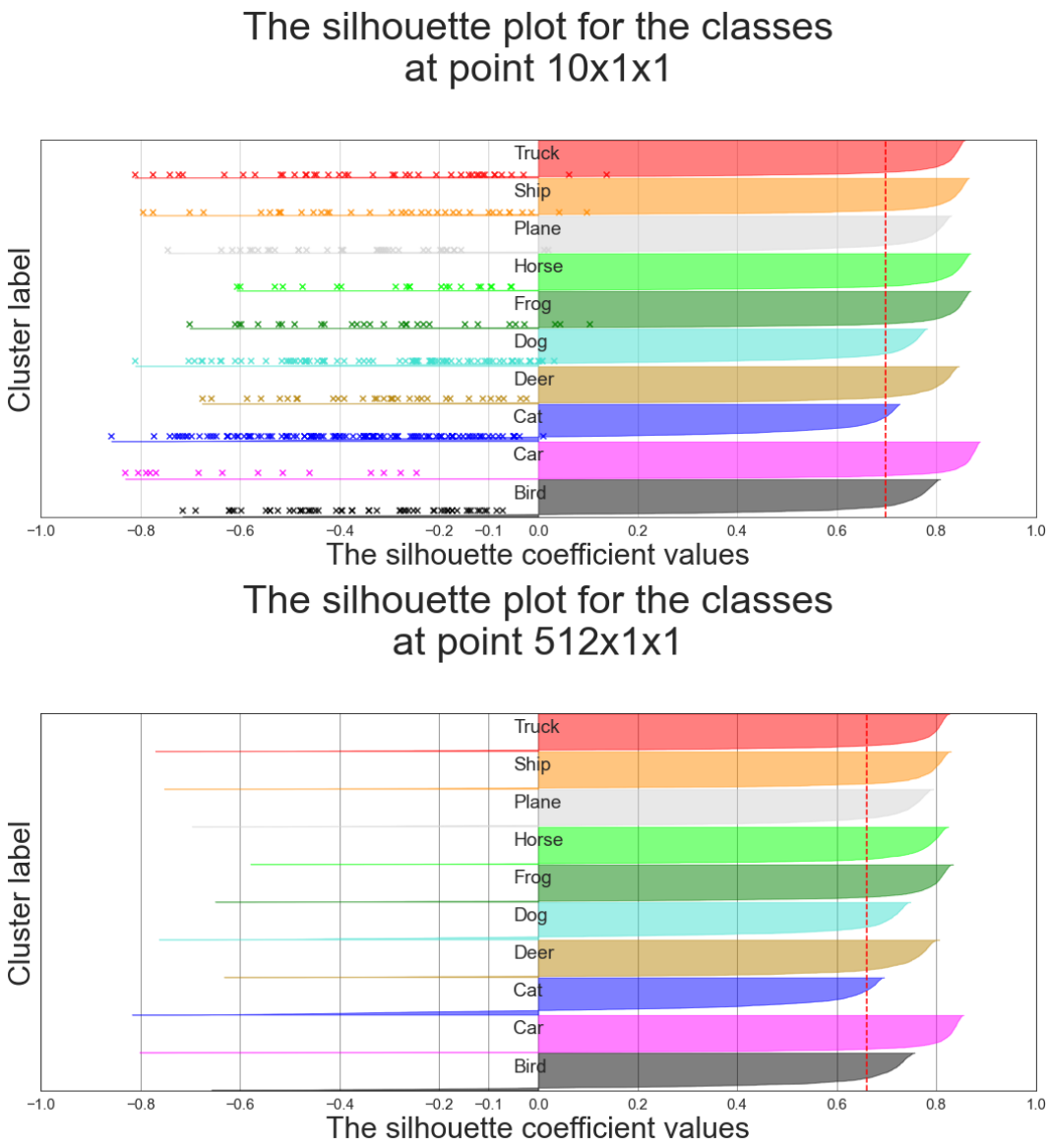


Figure 13: Illustration of the image wise silhouette coefficient for the classes at the last two checkpoints of interest. The crosses symbolise the images in the test set that are misclassified. The dashed red lines represent the overall mean silhouette coefficient for all classes. The thicker the smooth and coloured shapes are, the more common is the silhouette coefficient for the class.

The confusion matrix and the silhouette coefficients are only summarising statistics of the performance of ResNet-18. To get a better understanding of the cluster structure created by ResNet-18 it is informative to implement classical metric MDS as well as t-SNE. In Figure 14 the implementation of two dimensional (classical metric) MDS shows that in this low dimensional representation the clusters are spherical in both spaces. When MDS is applied to the smaller vector space at checkpoint $10 \times 1 \times 1$ the clusters seem to have the same cohesion, but the separation between them has increased. More specifically, the separation between inanimate and animal images seems to have improved the most. The interpretation of this is that

the fully connected layer seems to capture between-class differences rather than within-class similarities. It is important to remember that classical metric MDS tries to preserve Euclidean dot products from the high dimensional space for all images. This means that if the underlying manifold in the high dimensional space is not at all fit for Euclidean dot products then the MDS representation is misleading.

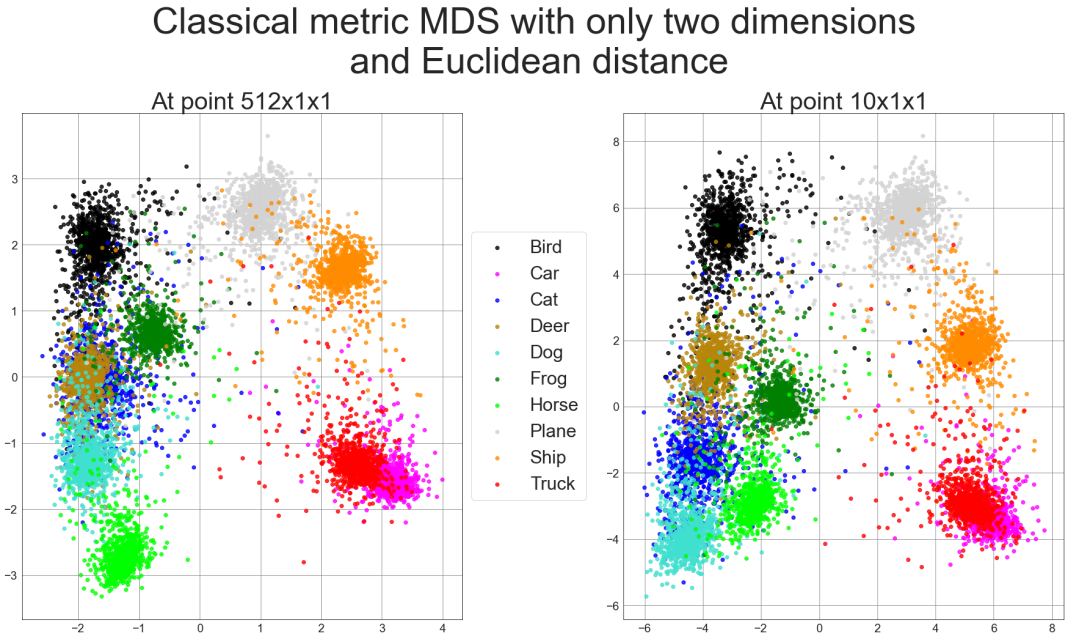


Figure 14: Two dimensional classical metric MDS for the test data at the two different checkpoints in ResNet-18.

To finalize the preliminary study of ResNet-18 the t-SNE method is implemented in the spaces of the checkpoints of interest. The resulting low dimensional representation is shown in Figure 15. In the two plots it is visible that some classes are mixed-up at the edges of the wrong clusters. This pattern is present in both plots, but at the checkpoint $10 \times 1 \times 1$ the mixed up points tend to be mapped even more towards the edges of the wrong clusters. This indicates a better between-class separation in this space. Interestingly, in both plots, one can find the patterns consistent with the confusion matrix. For instance there is a group of points with true class as planes, but are located at the edge of the cluster for bird. The large mix up between cats and dogs is also clearly present. When interpreting these plots it is very important to note that the global clustering structure is possibly very distorted. However, the local clustering structure around points is much more reliable. The reason for these two important notes is that they are an effect of the mismatch in distributions in the t-SNE method, see theory section 2.3.3.2 for the t-SNE method. In other words, the t-SNE method as is, focuses on preserving local structures in the high dimension, but is very prone to distort global structures and farther distances in the lower dimension.

Another important result when studying the MDS and t-SNE plots is that they both show pretty clear structures in terms of class-specific clusters. This is consistent with the low test-error. If the test error would have been very high the points would be scattered randomly across the lower dimensional spaces.

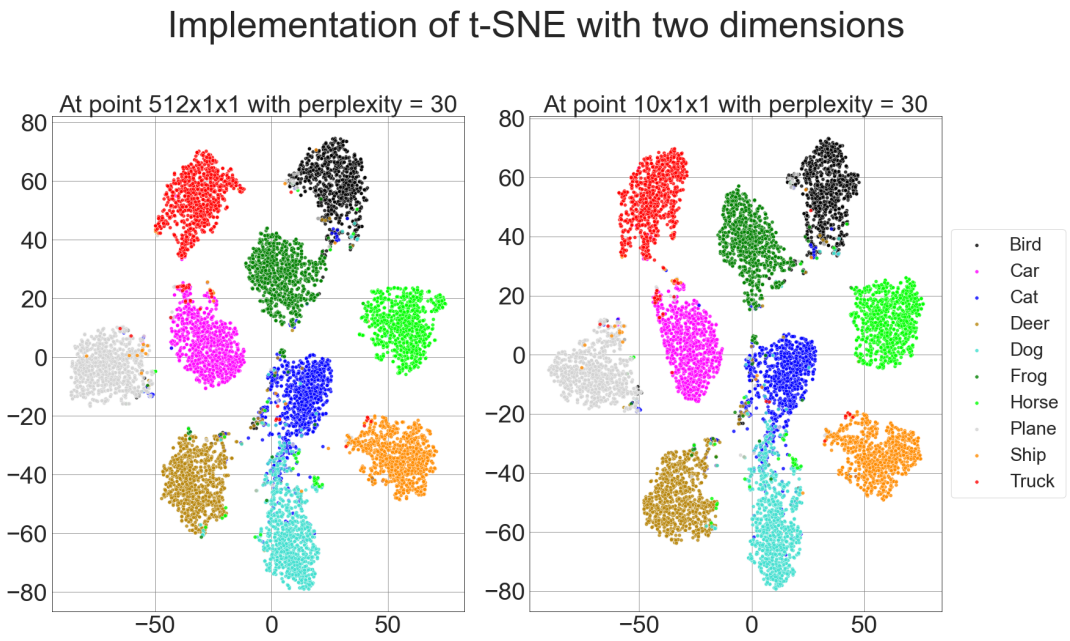


Figure 15: Two dimensional t-SNE for the test data at two different checkpoints in ResNet-18.

To summarise this section it is clear that ResNet-18 is more prone to making classification errors of certain types. It is hard for the model to discriminate between inanimate objects as well as between animals. The hardest task for the model is to separate cats and dogs. This pattern is also shown in the MDS and t-SNE dimensionality reduction plots. These plots also show how well ResNet-18 is able to create clear clustering structures for the different classes. It is also appearant when studying these two different checkpoints of interest that the model is making layer-wise improvements in classification due to better separation. This trend is also observed in the silhouette coefficient plots.

With this preliminary study discussed, it is time to dig even deeper into the network to diagnose and understand its working mechanisms on a more detailed level. This is done with the use of silhouette coefficients and commute time distances to respect the underlying non-linear structures at different checkpoints of interest in ResNet-18.

8 Results

As mentioned in previous sections, ResNet-18 is organized into a block structure with certain predefined operations repeated within each block. Since the aim of the thesis is to unveil the inner working mechanisms of the model it is a good start to study the vector space after each block. For the same reason it is also interesting to study the spaces at the input, after average pooling and after the fully connected layer. For a visual representation of these checkpoints of interest, see Figure 10. The reason for studying these specific checkpoints of interest is because before each checkpoint a reasonable amount of similar operations have been used. Hence, it is possible to locate where in the model real improvements in classification occur (good class discrimination).

In order to study these checkpoints of interest, kNN graphs are constructed as described in theory section 2.4.2.1. The resulting choice for the number of neighbours, k, and choice of the constant a in the t distribution similarity is shown in Table 1 in the appendix. For a quick reminder, the graphs are constructed to find the graph Laplacian which is used to calculate the commute time distances in these spaces. These distances are used since they aim to capture the underlying non-linear shape of the manifold the data resides in. With these distances at the different checkpoints of interest the silhouette coefficients are calculated. Since the true class for each data point in these spaces (vertex in the graph) is known, the true cluster memberships are also known. This is because the true cluster membership for each point is the corresponding class. This means that the silhouette coefficient measures how well ResNet-18 groups data points of different classes at the different checkpoints. The silhouette coefficients and kNN graphs in the coming sections are calculated using the metrics and neighbour modules implemented in the Python library Scikit-learn, see Pedregosa (2011). The commute time distances are calculated using Python code from the author of this thesis.

8.1 Locating the point of improvement

With the commute time distances found for the test data at all the checkpoints of interest, the class-specific silhouette coefficient is calculated at each checkpoint. In Figure 16 the results can be seen at each checkpoint of interest. From the plot the most interesting result is the major increase for all classes in the silhouette coefficient between checkpoints $256\times8\times8$ and $512\times4\times4$. At the earlier checkpoints of interest, the silhouette coefficients show no improvement for any class. Also, at these checkpoints the silhouette coefficient stays negative or at around zero. A negative silhouette coefficient means that ResNet-18 is clustering the test data in these vector spaces worse than a random guessing model (which corresponds to a score of around zero).

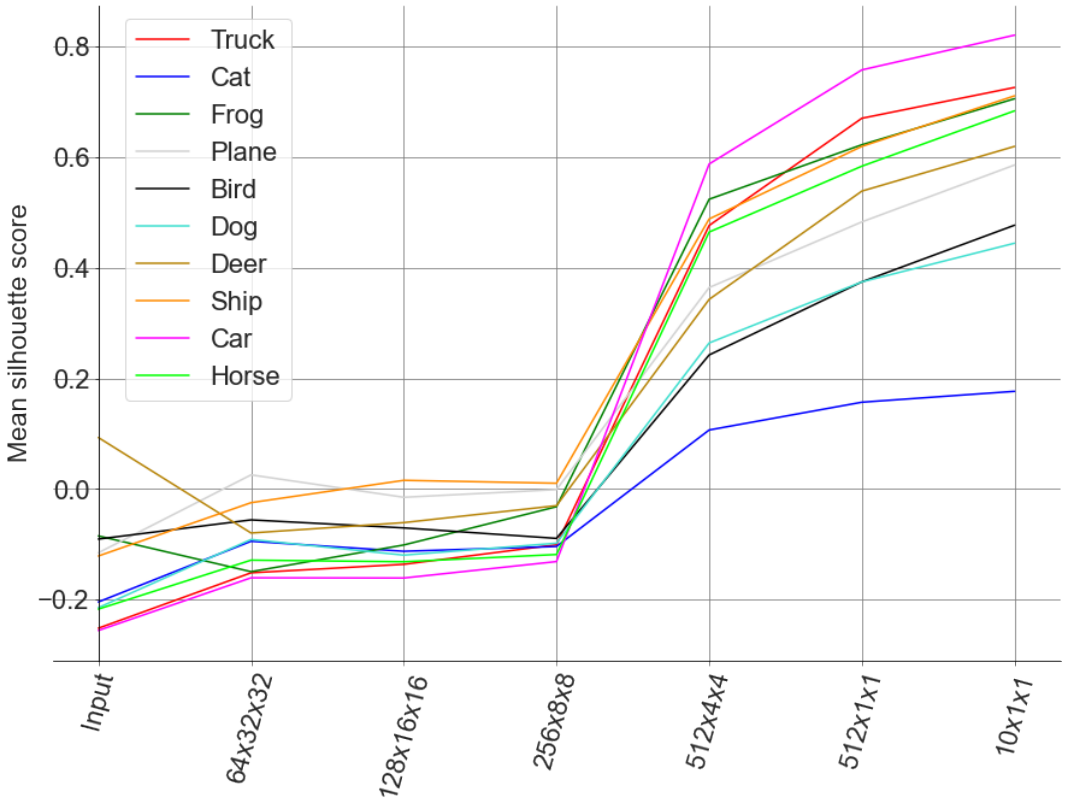


Figure 16: The mean silhouette coefficient for each class at the different checkpoints of interests in ResNet-18.

The meaning of this is that somehow the first three blocks in the ResNet-18 model actually make no significant improvement in discriminating between classes. Nevertheless, they most likely contribute to capturing other important features that are not class-specific. An example of an important feature that is not class-specific is edge detection, an aspect that is important in a lot of CNN classification models. That is, all classes have distinct edges separating the object from the background in the image. However, this does not help in discriminating whether the object in the picture is of a certain class.

Before studying the largest increase, it is interesting to note that the average pooling and the single fully connected layer display an increase in silhouette coefficients. The reason the average pooling causes an increase is because the operation makes ResNet-18 more robust to small changes in images. This causes more compact clusters (smaller cohesion) since small changes in the input to the average pooling layer will yield approximately the same coordinates in the $512 \times 1 \times 1$ output space of the pooling. The small increase from the fully connected layer, which is a linear transformation, suggests that the true underlying manifold in the $512 \times 1 \times 1$ dimension is more likely a smaller $10 \times 1 \times 1$ dimension. The increase can thus be explained by the additional 502 dimensions containing class confusing information, i.e., noise.

The largest increase in silhouette coefficients which is between checkpoints $256 \times 8 \times 8$ and $512 \times 4 \times 4$ will be the focus of the next section.

8.2 Locating the root cause for class discrimination in Block 4

The difference between checkpoints $256 \times 8 \times 8$ and $512 \times 4 \times 4$ (in which the largest increase is) are the operations contained within Block 4. These operations are convolutions, skip connections and the ReLU activation function. To locate which operations inside of Block 4 that are the cause of the large increase in silhouette coefficients, more checkpoints of interest inside of Block 4 are added. These additional checkpoints

of interest are presented in Figure 11. They are placed after each major convolution as well as between the last convolution and the ReLU activation function applied to the second to last convolution.

In Figure 17 the silhouette coefficients for the additional checkpoints of interest inside of Block 4 show that the largest increase is between the ReLU (applied to the second to last convolution) and the last convolution. This shows that the major cause for the increase in silhouette coefficients from Block 3 to Block 4 is caused by the very last convolution of ResNet-18. In other words, the final convolution is the main reason why ResNet-18 is able to discriminate well between classes. What the final convolution actually does to the images in this part of the network is impossible to understand without further modelling. A suggestion for studying the effect of the convolution is presented in the discussion section of the thesis. Despite the hardship of really digging into this operation, it is possible to explain the improvement in silhouette coefficients in terms of cohesion and separation. That is, it is possible to achieve an improvement in silhouette coefficients by lower cohesion or higher separation of classes. By studying this it is possible to understand whether the final convolution improves within-class similarities (lower cohesion) or between-class dissimilarities (higher separation). This is the main result of the thesis and is discussed in the next and final section of the result part. Before moving on to the last section a few more minor details in Figure 17 are interesting to study.

In Figure 17 the silhouette coefficients for the classes actually decrease or is almost the same after the final convolution and after the output of Block 4. The only two operations separating these two vector spaces is a skip connection and a ReLU activation. The skip connection essentially adds the output from the first skip connection in Block 4 with ReLU activation to the output of the last convolution. This addition is the input to the ReLU activation which in turn is the output of Block 4. Why this procedure degrades the silhouette coefficients for some classes can be explained by the ReLU activation. This is because the ReLU activation puts negative inputs equal to zero which means that some information from the input is lost and not presented to the rest of ResNet-18. One can think of the ReLU function as a person that only tells you about good news and is otherwise silent. Now, this means that some of the classes, those with a degradation in silhouette coefficients, had some relevant information erased by the ReLU activation in some instances.

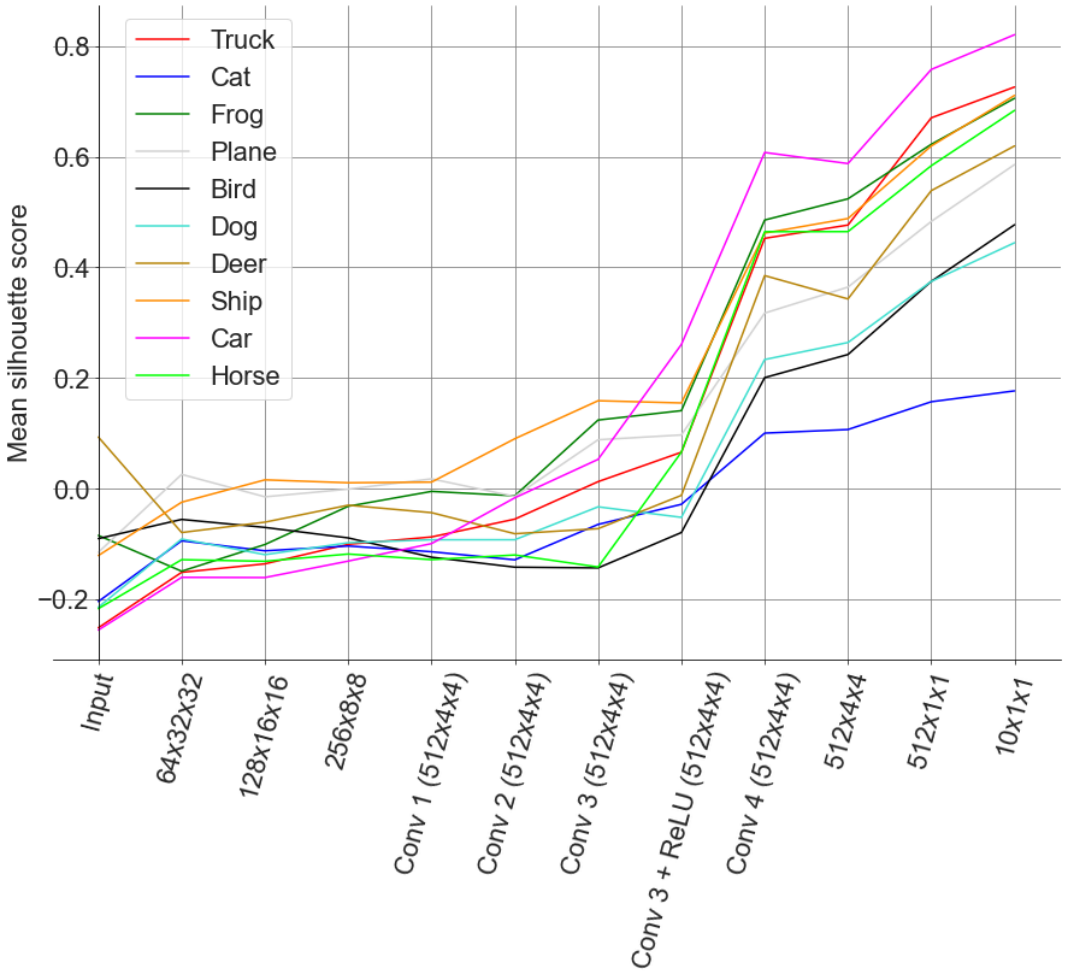


Figure 17: The mean silhouette coefficient for each class at the different checkpoints of interests in ResNet-18.

8.3 Deeper study of the final convolution - Separation or Cohesion?

This is the final part of the result section for this thesis. In this section the difference in cohesion and separation is studied before and after the final convolution. It is enlightening to think of cohesion as a measure of within-class similarity while separation measures between-class dissimilarities. In Figure 18 the histograms of the cohesion and separation before and after the last convolution are shown for the classes with the best (car) and worse (cat) increase in silhouette coefficient. In the figure a scatterplot between cohesion and separation for images within the class is shown after the final convolution. Similar plots for all the other classes can be found in Figure 1 in the appendix. These plots show similar results as the two most extreme classes shown in this section.

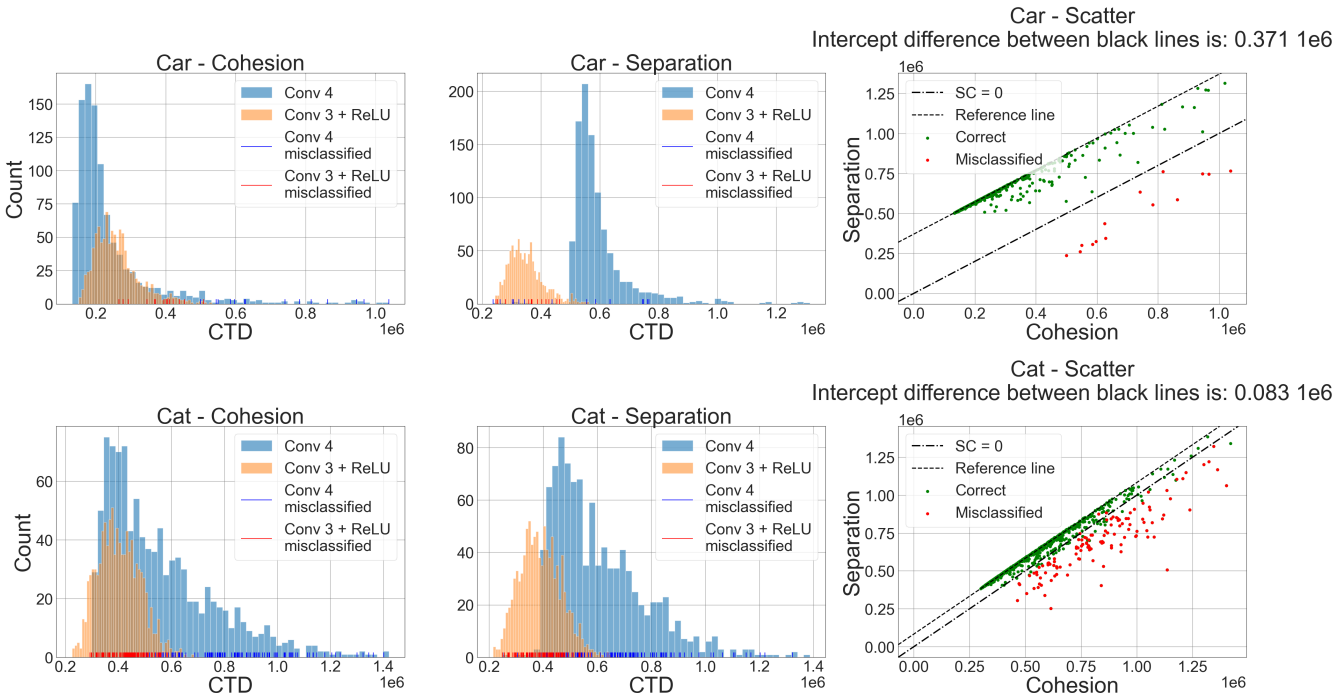


Figure 18: Histograms and scatterplots of cohesion and separation in Conv 3 + ReLU and Conv 4 outputs for classes car and cat. The scatterplot shows the cohesion and separation at Conv 4.

In the cohesion histogram for the classes car and cat, it can be seen that the cohesion for the misclassified points (marked by ticks on the x-axes) are far out in the tail before and after the final convolution. Being far out in the tail means that they have a high cohesion (more separated from their correct cluster). This is expected since misclassified points should be dissimilar from the correct cluster. In other words they should be far away from similar points (images) of the same class.

For car, the cohesion histogram has a longer tail after the convolution while the peak is sharper and more to the left. This indicates a small improvement in cohesion. For cat the cohesion histogram has a fatter and longer tail, while the mass of it is shifted to the right after the final convolution. The meaning of this is that some images are even farther away from the rest of the correct class, which is also the general pattern for the cat class. This indicates that the final convolution fails to find within-class similarities for the cat class, while it makes some small improvements in finding class similarities for the car class. For the other classes the histograms shown in Figure 1 in the appendix exhibit similar patterns like the one for car. This is true except for the classes dog, plane and bird which are more like the histograms for cat. This means that there is a minor overall improvement in cohesion for most of the classes, but it is not large. The reason is that the final convolution does not focus too much on finding within-class similarities.

To delve deeper into the mechanisms of the final convolution it is interesting to study the corresponding histograms for separation of the classes. As before, only car and cat are present in the histograms in Figure 18, the other classes are presented in Figure 1 in the appendix. Interestingly, for both the classes car and cat the histograms for separation after the convolution have fatter tails while also having their centres of mass shifted to the right. The shift to the right is even more prominent in the case of the car class. Now it is important to note that the separation is always shown for the closest class (see equation (52b)), meaning that the separation can be much larger to some other class. For the other classes the separation looks similar to the case of the car class except for the classes dog and bird which are similar to the cat case. However, for all classes it is the case that there is a clear improvement in separation after the final convolution. The interpretation of this is that this final convolution focuses a lot more on finding between-class dissimilarities while focusing less on within-class similarities.

It is also worth noting that the shape of the histograms in cohesion and separation are similar when studied at each checkpoint of interest (before and after the final convolution). This means that the distribution in cohesion and separation tend to be the same while the class centres are quite different.

The conclusion from the cohesion and separation histogram is that the shift in separation after the final convolution is the cause of the increase in silhouette coefficients. This is indicated by the largely un-shifted cohesion but shifted separation. In other words, the improvement in silhouette coefficients is most likely due to the final convolution increasing the separation which corresponds to finding between-class dissimilarities to discriminate between classes. For the classes with worse cohesion, the silhouette coefficient is still increased due to the large improvement in separation.

Furthermore, it is interesting that the scatterplots between the cohesion and separation indicate a positive correlation between the two for all classes after the final convolution. This can be seen in both the figures presented in this section and the appendix. The meaning of this is that images that are far away from other classes also tend to be more unique for their true class. In the scatterplots it is also visible that the misclassified points tend to have a higher level of cohesion. The interpretation of this is that they often are on the edge of their true class which in turn makes them hard to correctly classify.

In all the scatterplots (in the appendix and in this section) two black lines are shown. The middle line (with slope equal to one) is the set of points with silhouette coefficient equal to zero. The other shifted line has the same slope, but with another intercept. Everything above the middle line has a silhouette coefficient larger than zero. The shifted line captures the linear relationship between cohesion and separation pretty well for the correctly classified images.

For all the points in the scatterplots, a line with the slope of one seems to accurately describe the relationship between cohesion and separation. That is, when comparing two points with a difference of ten in cohesion then they also have a difference of ten in separation. The meaning of this result is that the distribution in cohesion and separation are of similar shapes. For example, a large slope would imply that the distribution in cohesion is very sharp compared to the distribution for separation. This is because the dispersion in cohesion would be much smaller compared to the one in separation. This observation can also be found when independently examining the histograms for cohesion and separation after the final convolution.

Furthermore, since the slope is the same for the two black lines in the scatterplots there must be another mechanism that causes the, in general, high and positive silhouette coefficients. Now, the difference between the lines are the intercepts. This difference in intercepts (while the slopes are one) is caused by the distribution of separation being positioned to the right of the distribution for cohesion. Since the shift is what keeps the line above the line representing silhouette coefficients equal to zero, the reason for the improvement after the final convolution must be the general increase in separation in relation to the change in cohesion. In summary, these scatter plots show the same results as the histograms for cohesion and separation independently.

Interestingly it is possible to find similar results found in the cohesion and separation plots by studying the mean commute time distance between classes. That is, for images in two different classes C_i and C_j the mean commute time distance is given by equation (53).

$$\text{Mean CTD} = \frac{\sum_{i=1}^{|C_i|}\sum_{j=1}^{|C_j|} c_{i,j}}{1000^2} \tag{53}$$

In Figure 19 the difference in mean commute time distance before and after the final convolution is shown in the bar plots to the left. The bar plots for all classes are positive, meaning that all classes have an increase in mean commute time distance after the final convolution. For animal classes the difference is the largest between itself and inanimate classes and vice versa for inanimate classes. Interestingly, the classes with the highest number of misclassification (bird, cat and dog) are the ones with the lowest mean difference before and after the convolution, which is consistent with Figure 12. Furthermore, the class plane exhibits the interesting trait of having a very low range between the maximum and minimum difference before and after the convolution. The interpretation is that it tends to separate equally to pretty much all classes. That is, the plane class is most likely not extremely different from any particular class before and after the convolution.

In summary, these graphs illustrate the same results as the histograms for separation in terms of increased cluster separations. However, they also show that some classes tend to separate more compared to others. The explanation is that the final convolution discriminates more between classes by finding between-class dissimilarities, in particular dissimilarities between animal and inanimate classes.

In summary, the final convolution improves separation rather than cohesion, which means that it captures further between-class dissimilarities, resulting in the increase in silhouette coefficients after the final convolution.

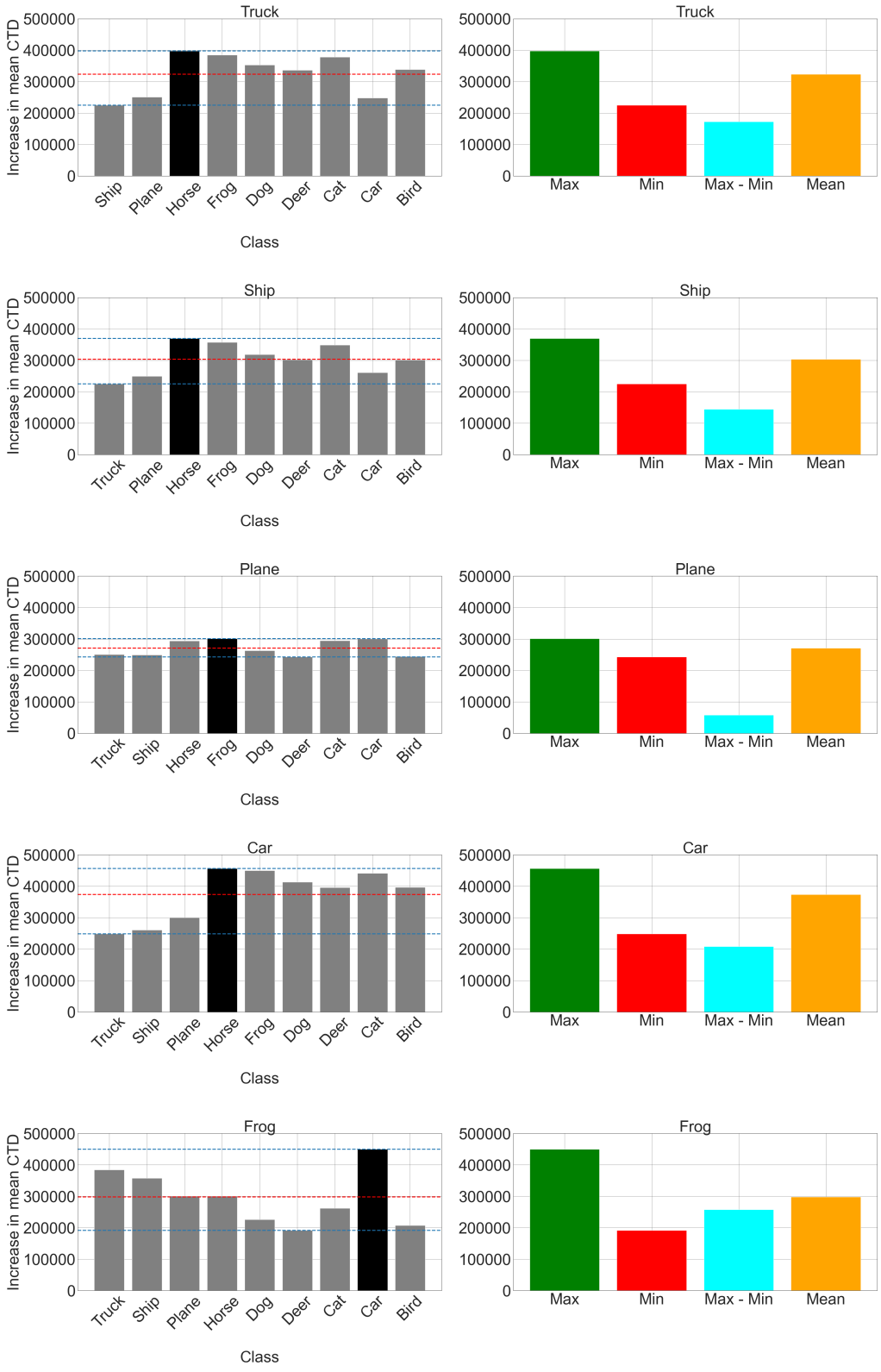


Figure 19: Differences in mean CTD between classes in the two different checkpoints of interest Conv 3 + ReLU and Conv 4. The difference is from Conv 3 + ReLU to Conv 4 and is positive. The red dotted line is the mean of all the bars in the subplot. The blue dotted lines mark the height of the tallest and smallest bars.

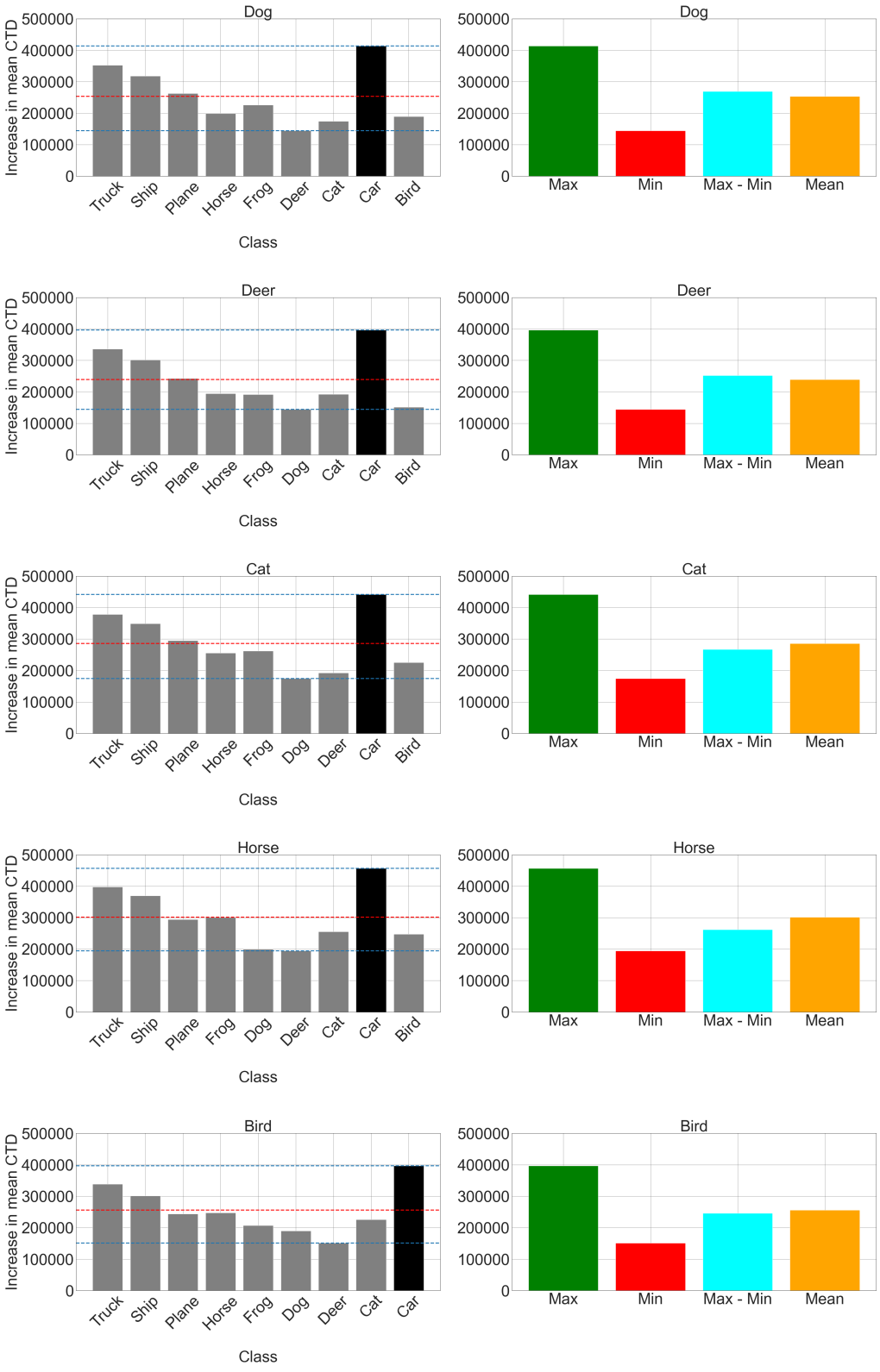


Figure 19: Differences in mean CTD between classes in the two different checkpoints of interest Conv 3 + ReLU and Conv 4. The difference is from Conv 3 + ReLU to Conv 4 and is positive. The red dotted line is the mean of all the bars in the subplot. The blue dotted lines mark the height of the tallest and smallest bars.

9 Discussion

The aim of this section is to shortly summarise the thesis and the findings in the study of ResNet-18. Then a subsection of future studies and improvements is presented.

9.1 Summary and conclusion

This report starts with a motivation and a short background which inspired this thesis. Next, a theoretical overview and some repetition of the methods and models used are presented to provide a deeper understanding of the results. Then the data and modelling procedure is presented followed by the result section. The result section starts with a small preliminary study of the main model, ResNet-18, and finishes with locating the most important operation in the model along with an explanation. The thesis is finalised with an important discussion section which provides the next step to really understand ResNet-18.

The Python code for training of ResNet-18 on the dataset CIFAR-10 was provided by Liu (2021). The Python code created by Liu (2021) for fitting of ResNet-18 is based on the commonly used package torch and library torchvision as a part of the PyTorch library, see PyTorch (2022b) and PyTorch (2022c). For this thesis the code for training is only changed to use 300 epochs instead of the original 200 epochs. That is, the code is used as it was provided to ensure stable and correct training of ResNet-18. For extracting data at checkpoints and processing the test set of CIFAR-10 in the trained ResNet-18 as well as calculating commute time distances, Python code from me, the author of this thesis, is used. That is, the use of the trained ResNet-18 is done using my own Python code. The calculations for t-SNE, kNN-graphs, silhouette coefficients and classical metric MDS are all made using the modules implemented in the library Scikit-learn as well as my own code, see Pedregosa (2011).

The main findings of this thesis are that the early CNN specific operations in ResNet-18 do not seem to do much for class discrimination. However, in the last part of the CNN section of ResNet-18 there is a major improvement in silhouette coefficients (a measure of how well the model groups data points of different classes). The improvement is caused by the very last convolutional operation in ResNet-18. When studying the cause for improvement it is mainly due to an increase in class separation. That is, the final convolution is improving between-class dissimilarities rather than within-class similarities. Furthermore, the final convolution is also separating animal type classes from inanimate type classes from each other. Hence, the final convolution is a key component to separating classes based on their differences, especially differences between animals and inanimate objects. This is the driving mechanism as to why the ResNet-18 under study has a high accuracy on the test set.

In conclusion, the final convolution is really important for ResNet-18 to be able to make good discrimination between the classes. The earlier operations seem to be more focused on extracting non-class-specific features such as edges. The average pooling and fully connected layers also provide some improvements in class discrimination. Furthermore, the improvement from the fully connected layer reveals that there might be a lot of redundant information in the input to the layer. Hence, all parts of ResNet-18 seem to play a role in the decision making, but the key mechanism is the final convolution. To truly understand the decisions that the model is making, further studies are presented in the next section.

9.2 Future studies and improvements

The main result of this study is that the final convolution in ResNet-18 is a key mechanism in discriminating between classes by separating them based on between-class dissimilarities. However, with the results produced for this thesis it is not possible to understand what the convolution is actually doing to its input. Hence, the most logical step forward is to further study this final convolution in detail. A proposed method of doing this is as follows:

1) In the vector space before the final convolution and the vector space right after the convolution, implement a dimensionality reduction to two dimensions. A suggestion would be to use t-SNE in a way that preserves global structures in the lower dimension.

2) In the lower dimension of the space before the final convolution, one can generate new images in the gaps between classes that separate well. This can for instance be done with deep generative models, such as variational autoencoders (VAE), for details on these models see Kingma and Welling (2019).

3) Study what happens to these newly generated images when the final convolution is applied to them and projected into a lower-dimensional space.

This procedure can help locate what specific features the final convolution picks up in order to discriminate between classes.

Another aspect to include in a future study would be to study ResNet-18 in training. This thesis only studies the already trained model on the test data. From the study of the training of the CNN, it might be possible to see if there are sudden changes in silhouette coefficients after a number of epochs. Furthermore, it might also be possible to see if some operations contribute more or less during certain parts of the training. With these insights it is possible to tell more about the mechanisms of ResNet-18.

The major improvements that can be made in this thesis is to implement generation of new images and a study of the training as discussed above. Furthermore, it would also be interesting to train ResNet-18 and similar models on CIFAR-10 and much more complex data sets such as ImageNet-1k. This should be done in order to see if there are similar mechanisms within the networks. This would provide insight into whether some operations in CNN:s seem to be more important than others and if some are even necessary to implement. However, the challenge with this is that the study would be expensive in terms of computation time and power when done on a laptop. It would most likely require computers purely dedicated to being run for days or weeks without interruption.

Lastly it would be interesting to study what the operations in the other parts of ResNet-18 do. By understanding all of the mechanisms of the model it would be possible to make a flow chart of how the CNN processes images. This can hopefully be used to understand why the model makes correct and erroneous classifications.

References

Adadi, A., and Berrada, M. (2018), "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE access*, IEEE, 6, 52138–52160.

Aphex34 (2015), "Typical CNN architecture," Available at:https://commons.wikimedia.org/wiki/File:Typical_cnn.png.

Cover, T. M., and Thomas, J. A. (2006), *Elements of information theory*, Hoboken, N.J.: Wiley.

Goodfellow, I., Bengio, Y., and Courville, A. (2016), *Deep learning*, MIT Press.

Hastie, Trevor., Tibshirani, Robert., and Friedman, Jerome. (2009), *The elements of statistical learning [electronic resource] data mining, inference, and prediction*, New York, NY: Springer New York.

He, K., Zhang, X., Ren, S., and Sun, J. (2016), "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Kingma, D. P., and Welling, M. (2019), "An introduction to variational autoencoders," *arXiv preprint arXiv:1906.02691*.

Klein, D. J., and Randić, M. (1993), "Resistance distance," *Journal of mathematical chemistry*, Springer, 12, 81–95.

Krizhevsky, A., Hinton, G., and others (2009), "Learning multiple layers of features from tiny images," Citeseer.

Lee, J. A., and Verleysen, M. (2008), *Nonlinear dimensionality reduction [electronic resource]*, New York, NY: Springer Science+Business Media, LLC.

Liu, K. (2021), "Train CIFAR10 with PyTorch," *GitHub repository*, https://github.com/kuangliu/pytorch-cifar; GitHub.

Molnar, C. (2022), *Interpretable machine learning: A guide for making black box models explainable*.

Papoulis, A. (1962), *The fourier integral and its applications*, New York: McGraw-Hill, pp. 244–245.

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

PyTorch (2022a), "Transforming and augmenting images," https://pytorch.org/vision/stable/transforms.html.

PyTorch (2022b), "Torch," https://pytorch.org/docs/stable/torch.html.

PyTorch (2022c), "Torchvision," https://pytorch.org/vision/stable/#module-torchvision.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016), "" why should i trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144.

Tan, P. N., Steinbach, M., Karpatne, A., and Kumar, V. (2019), *Introduction to data mining*, What's new in computer science series, Pearson, pp. 581–582.

Van der Maaten, L., and Hinton, G. (2008), "Visualizing data using t-SNE." *Journal of machine learning research*, 9.

Von Luxburg, U. (2007), "A tutorial on spectral clustering," *Statistics and computing*, Springer, 17, 395–416.

Wängberg, T. (2020), "A survey of stochastic neighbor embeddingfor dimension reduction and data visualization," *Masterarbeten i matematisk statistik 2020*.

Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., and Zhu, J. (2019), "Explainable AI: A brief survey on history, research areas, approaches and challenges," in *CCF international conference on natural language processing and chinese computing*, Springer, pp. 563–574.

Appendix

9.3 PCA

The proofs in this section is inspired by Lee and Verleysen (2008).

9.3.1 Global variance

$$\sigma_{\hat{X}} = \sum_{i=1}^{d} \sigma_{\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_i} = \sum_{i=1}^{d} \frac{1}{N-1} \sum_{j=1}^{N} \hat{x}_{i,j}^2 = \\ \sum_{i=1}^{d} \frac{[\mathbf{v}_i^T \mathbf{X}][\mathbf{v}_i^T \mathbf{X}]^T}{N-1} = \sum_{i=1}^{d} \mathbf{v}_i^T C_X \mathbf{v}_i = \sum_{i=1}^{d} \lambda_i \tag{1}$$

9.3.2 Maxmimal variance

In this section the following maximization problem in equation (2) is solved. All notations are the same as in the theory section for PCA.

$$\max_{\mathbf{P}} \ \text{Trace}(C_{\hat{X}}) \ \text{ subject to } \ \mathbf{P}^T\mathbf{P} = \mathbf{I}_d \tag{2}$$

First rewrite the target function to equation (3) where $\mathbf{u}_i = \mathbf{V}^T\mathbf{p}_i$ and $u_{i,j} = \mathbf{v}_j^T\mathbf{p}_i$.

$$\begin{aligned} \text{Trace}(C_{\hat{X}}) &= \text{Trace}(\mathbf{P}^T[\mathbf{V}\Lambda\mathbf{V}^T]\mathbf{P}) \\ &= \sum_{i=1}^{d}\sum_{j=1}^{D}(\mathbf{v}_j^T\mathbf{p}_i)^2\lambda_i \\ &= \sum_{i=1}^{d}\sum_{j=1}^{D}(u_{i,j})^2\lambda_i \end{aligned} \tag{3}$$

Since all $\mathbf{p}_i$:s are constrained to have a norm of one and be orthogonal to all other $\mathbf{p}_k$:s, equation (4) must hold.

$$\begin{aligned} \|\mathbf{u}_i\| = \left\|\mathbf{V}^T\mathbf{p}_i\right\| = \|\mathbf{p}_i\| = 1 \\ \mathbf{u}_i^T\mathbf{u}_j = \mathbf{p}_i^T\mathbf{V}\mathbf{V}^T\mathbf{p}_i = \mathbf{0} \end{aligned} \tag{4}$$

This implies that solving the maximization problem in equation (2) is equivalent to maximizing the expression in equation (3) with the constraint that all $\|\mathbf{u}_i\| = 1$ and $\mathbf{u}_i^T\mathbf{u}_j = \mathbf{0}$. Since the eigenvalues are sorted according to size the maximal solution would be to set $\mathbf{u}_i$ equal to the vector with a one at position i and zero everywhere else. Surely, setting them all equal to the vector with 1 at position one would yield a maximal value for the target function. However, this would violate our constraint of $\mathbf{u}_i^T\mathbf{u}_j = \mathbf{0}$.

Now, it also holds that $\mathbf{u}_i = \mathbf{V}^T\mathbf{p}_i \Leftrightarrow \mathbf{V}\mathbf{u}_i = \mathbf{v}_i = \mathbf{p}_i$. This in turn implies that the optimal solution to the maximization problem in equation (2) is equal to $\mathbf{p}_i = \mathbf{v}_i$ which was to be shown.

9.4 Clustering

9.4.1 Properties of the graph Laplacian

The theory and proofs in this section are inspired by Von Luxburg (2007).

For this section assume that G is an undirected and weighted graph as in the clustering part of the theory section.

For $\mathbf{x} = (x_1, ..., x_N)^T \in \mathbb{R}^N$ we have

$$\begin{aligned}
&\mathbf{x}^T L\mathbf{x} = \mathbf{x}^T \left[\mathbf{D} - \mathbf{W}_G\right] \mathbf{x} = \sum_{i=1}^{N} d_i x_i^2 \; - \sum_{i,j=1}^{N} w_{i,j} x_i x_j = \\
=&\frac{1}{2} \sum_{i=1}^{N} d_i x_i^2 \; - \sum_{i,j=1}^{N} w_{i,j} x_i x_j + \frac{1}{2} \sum_{j=1}^{N} d_j x_j^2 = \\
=&\frac{1}{2} \left[\sum_{i=1}^{N} d_i x_i^2 \; - 2 \sum_{i,j=1}^{N} w_{i,j} x_i x_j + \sum_{j=1}^{N} d_j x_j^2 \right] = \frac{1}{2} \left[\sum_{i,j=1}^{N} w_{i,j} (x_i - x_j)^2 \right] \geq 0
\end{aligned} \tag{5}$$

Since $\mathbf{L}$ is real and symmetric we can use that the eigenvalues are real in combination with the above result. This leads to the following where $\mathbf{x}_i$ is the eigenvector with eigenvalue λ_i.

$$0 \leq \mathbf{x}_i^T \mathbf{L} \mathbf{x}_i = \mathbf{x}_i^T \lambda_i \tag{6}$$

From equation (6) above it must be that equality holds only when $\lambda_i = 0$ since eigenvectors by definition cannot be the zero vector. Hence, it must be that the eigenvalues of $\mathbf{L}$ must satisfy $0 \leq \lambda_1 \leq ... \leq \lambda_N$.

It is now time to prove that the number of connected components of G corresponds to the multiplicity m of the eigenvalue zero of $\mathbf{L}$.

First we assume that the multiplicity is equal to one. This corresponds to the case when the graph is fully connected. That is, it is possible to traverse the graph between any two vertices. Let $\mathbf{x}$ be an eigenvector with corresponding eigenvalue zero. Then it must be that equation (7) holds only when all $w_{i,j}(x_i - x_j)^2$ are equal to zero since no weights are negative. If $w_{i,j} = 0$ no constraints are put on x_i or x_j. However, if two vertices v_i and v_j are connected, then $w_{i,j} > 0$ and thus $x_i = x_j$ must hold for the expression to equal zero.

$$0 = \mathbf{x}^T \mathbf{L} \mathbf{x} = \sum_{i,j=1}^{N} w_{i,j} (x_i - x_j)^2 \tag{7}$$

Now since the graph is fully connected all weights are larger than zero and for all i, j, $x_i = x_j$ must hold. This means that $\mathbf{x}$ is the identity vector times a zero-separated scalar. Without loss of generality we can assume that $\mathbf{x}$ is equal to the identity vector, which corresponds to the indicator vector of the single connected component. That is, it is equal to the vector containing zeros and ones marking whether a vertex is a member of the connected component or not. Since all vertices in the graph are members of the single fully connected component $\mathbf{x}$ must only contain ones. Hence, the eigenvector corresponding to the eigenvalue zero is the indicator vector with only ones, which implies that there is only one connected component.

We assume that the multiplicity of the eigenvalue zero is equal to $m > 1$. Additionally, we assume that the vertices are ordered according to the connected components. That is, we get a block diagonal form for $\mathbf{L}$ since the weight matrix is block diagonal. The reason for this is because the weights are non-zero only for elements in the same connected component. In other words, one can arrange the rows and columns to form symmetric blocks of non-zero weights. All other elements that are off from the block diagonal are set to be zero.

$$\mathbf{L} = \begin{pmatrix} \left[\ \mathbf{L}_1\ \right] & & & \\ & \left[\ \mathbf{L}_2\ \right] & & \\ & & \ddots & \\ & & & \left[\ \mathbf{L}_m\ \right] \end{pmatrix} \tag{8}$$

Now note that each block $\mathbf{L}_i$ forms a valid graph Laplacian on its own, since it is independent of the rest of the components in G. It is now convenient to use the known result that the eigenvectors of a block diagonal matrix are the union of eigenvectors for the different block matrices. It is also known that the corresponding eigenvectors for a block diagonal matrix are the eigenvectors for the block matrices filled with zero at the positions of the other block matrices. Since each block matrix $\mathbf{L}_i$ is a graph Laplacian for a fully connected sub-graph of G, then it must have multiplicity one for its eigenvalue zero. This means that the corresponding eigenvector for $\mathbf{L}$ (the one paired with the zero eigenvalue in $\mathbf{L}_i$) is the eigenvector with ones only at the positions for the block matrix $\mathbf{L}_i$ and zeros elsewhere. This implies that the graph Laplacian has as many eigenvalues equal to zero as it has connected components. In other words, the multiplicity of the eigenvalue zero of the graph Laplacian corresponds to the number of connected components in the graph. Lastly, this also proves that the eigenspace of the eigenvalue zero is spanned by the indicator vectors of the fully connected components.

Appendix - Figures

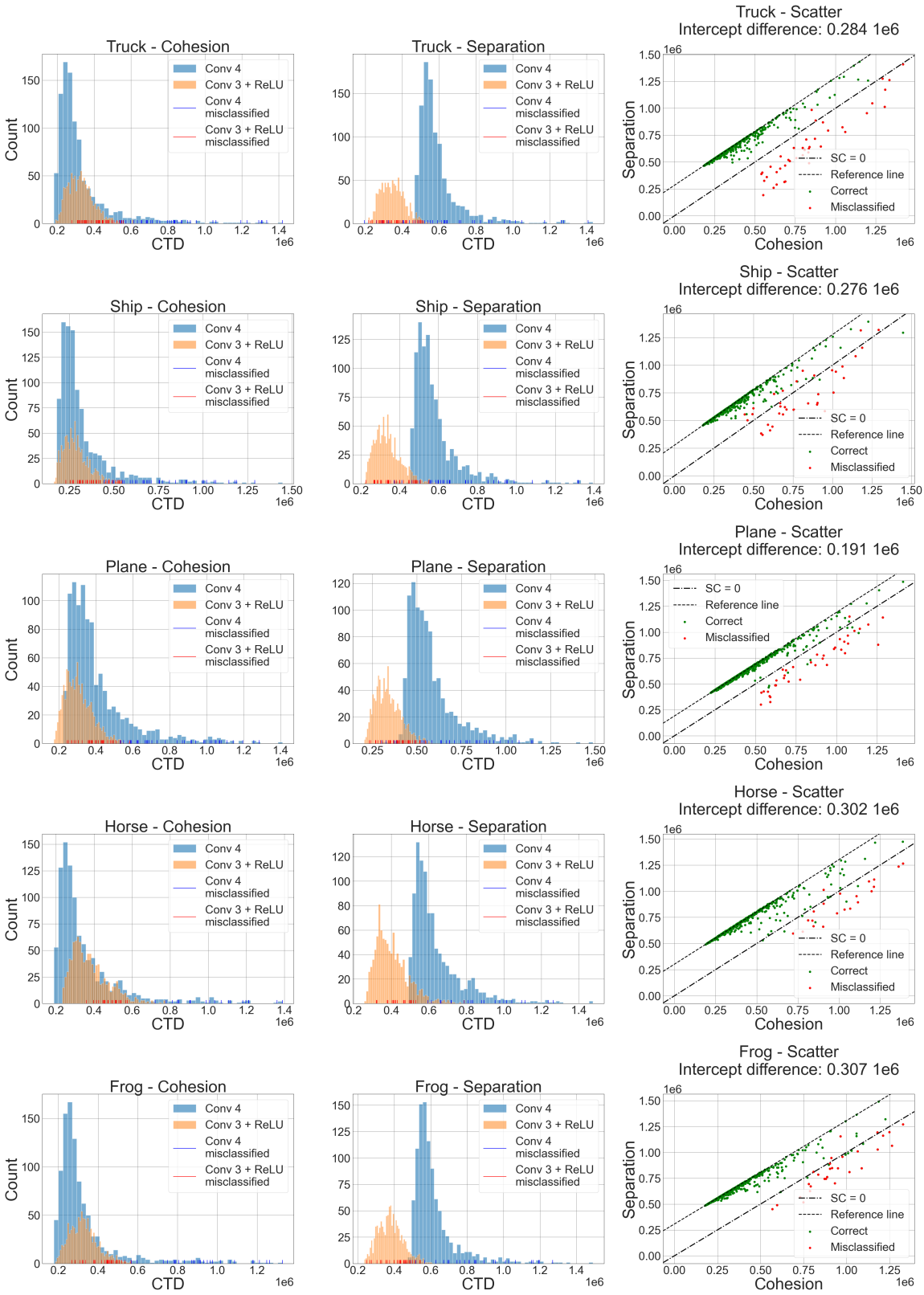


Figure 1: Histograms and scatterplots of cohesion and separation in Conv 3 + ReLU and Conv 4 outputs for each class. The scatterplot shows the cohesion and separation at Conv 4.

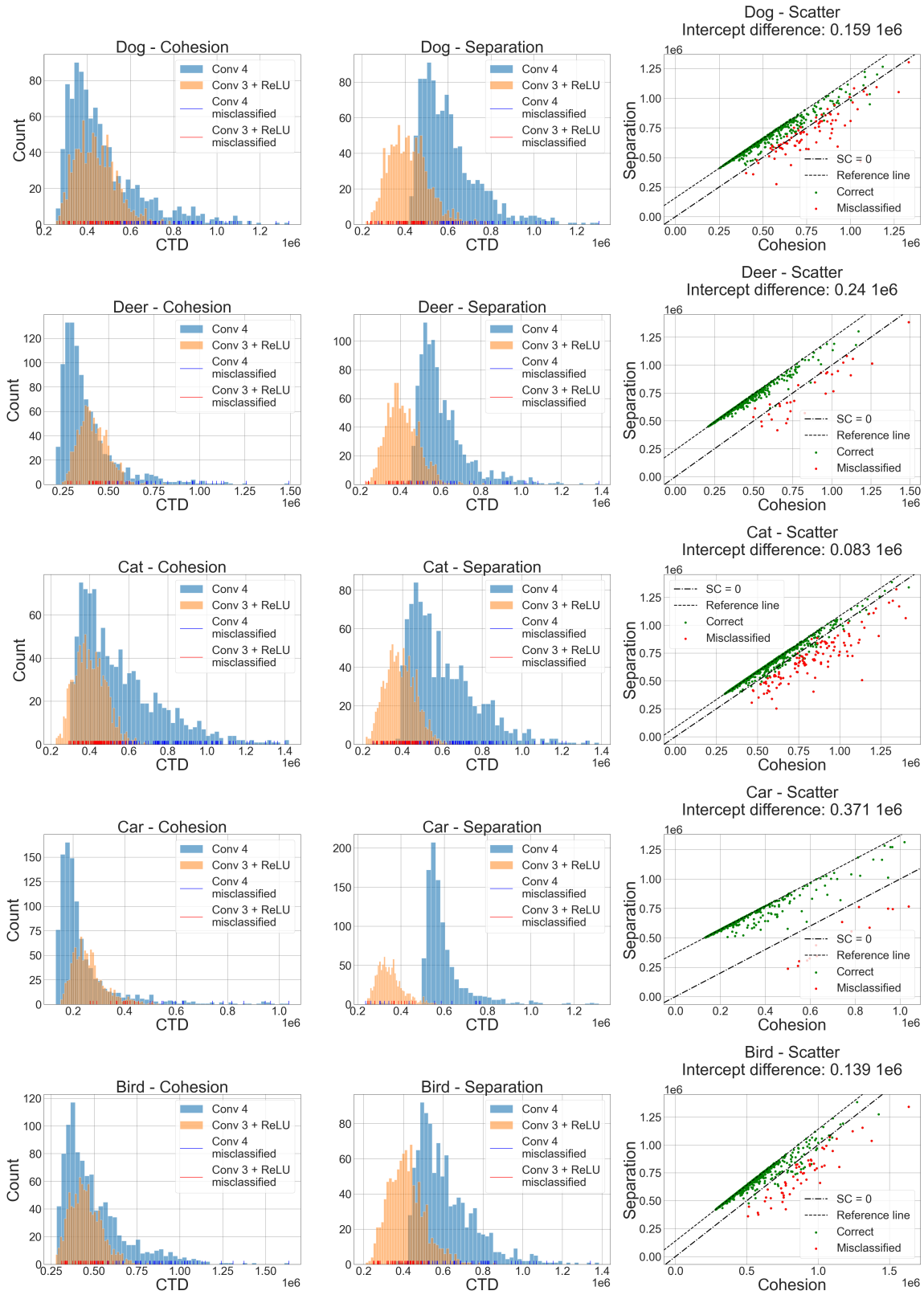


Figure 1: Histograms and scatterplots of cohesion and separation in Conv 3 + ReLU and Conv 4 outputs for each class. The scatterplot shows the cohesion and separation at Conv 4.

Appendix - Tables

Model Layer	**k**	**a**	**Dimensions**
After fully connected layer	3	0.210364	$10 \times 1 \times 1$
After average pooling layer	3	0.134224	$512 \times 1 \times 1$
After block 4	3	0.811621	$512 \times 4 \times 4$
After convolution 4 in block 4	3	0.042545	$512 \times 4 \times 4$
After convolution 3 and ReLU in block 4	3	0.173391	$512 \times 4 \times 4$
After convolution 3 in block 4	3	0.164879	$512 \times 4 \times 4$
After convolution 2 in block 4	3	0.185049	$512 \times 4 \times 4$
After convolution 1 in block 4	3	0.392566	$512 \times 4 \times 4$
After block 3	4	1.071954	$256 \times 8 \times 8$
After block 2	4	2.633042	$128 \times 16 \times 16$
After block 1	3	3.987486	$64 \times 32 \times 32$
Input (raw data)	4	6.920132	$3 \times 32 \times 32$

Table 1: The table shows the values for the constant a and the number of neighbours, k, in the graph construction and CTD for each layer