NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

ANALYSIS OF THE PROFICIENCY OF FULLY-CONNECTED NEURAL NETWORKS IN THE PROCESS OF CLASSIFYING DIGITAL IMAGES

Benchmark of Different Classification Algorithms on High-Level Image Features from Convolutional Layers

Jonathan Gabriel Martin Janke

Dissertation presented as partial requirement for obtaining the

Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

2018

Title: Analysis of the Proficiency of Fully-Connected Neural Networks in the Process of Classifying Digital Images

Subtitle: Benchmark of Different Classification Algorithms on High-Level Image Features from Convolutional Layers

Jonathan Janke

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

ANALYSIS OF THE PROFICIENCY OF FULLY-CONNECTED NEURAL NETWORKS IN THE PROCESS OF CLASSIFYING DIGITAL IMAGES

by

Jonathan Gabriel Martin Janke

Dissertation presented as partial requirement for obtaining the

Master's degree in Advanced Analytics

Advisor: Mauro Castelli

November 2018

Analysis of the Proficiency of Fully-Connected Neural Networks in the Process of Classifying Digital Images

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I appreciate the various discussions I had with different people from my faculty from the ideation phase to the execution and the interpretation of the final results. I would like to thank everybody that took their time to listen to my ideas, help me fine-tune the thesis and shape it to what it is now. Furthermore, I am grateful for everybody that read my thesis and provided me with feedback. This also goes for the people that helped me write an abstract in Portuguese.

I also want to thank my supervisor Mauro Castelli for giving me the necessary support and feedback during the thesis. I am thankful for the freedom I had when choosing the topic that I want to do research on and the timeline to realise the thesis.

Further appreciation goes to Thaís Góes for the support and all the help during the thesis. Thanks to everybody else who accompanied me on my journey. It feels good to have people around you that support you and that you can rely on.

Lastly, I would like to thank my family, in particular my parents, Stephanie and Martin Janke. Thank you for the trust that you have put into me and my personal development and the liberty that you have given me during my studies. I appreciate all the support I have received during my Bachelor's and Master's degree that enabled me to fully focus on those.

Abstract

Over the course of research on Convolutional Neural Network (CNN) architectures, little modifications have been done to the fully-connected layers at the end of the networks. In image classification, these neural network layers are responsible for creating the final classification results based on the output of the last layer of high-level image filters. Before the breakthrough of CNNs, these image filters were handcrafted, and any classification algorithm was applied to their output. Because neural networks use gradients to learn their weights subject to the classification error, fully-connected neural networks are a natural choice for CNNs. But the question arises: Are fully-connected layers in a CNN superior to other classification algorithms? After the network is trained, the approach is to benchmark different classification algorithms on CNNs by removing the existing fully-connected classifier. Thus, the flattened output from the last convolutional layer is used as the input for multiple benchmark classification algorithms. To ensure the generalisability of the findings, numerous CNNs are trained on CIFAR-10, CIFAR-100 and a subset of ILSVRC-2012 with 100 classes. The experimental results reveal that multiple classification algorithms are capable of outperforming fully-connected neural networks in different situations, namely Logistic Regression, Support Vector Machines, eXtreme Gradient Boosting, Random Forests and K-Nearest Neighbours. Furthermore, the superiority of individual classification algorithms depends on the underlying CNN structure and the nature of the classification problem. For classification problems with multiple classes or for CNNs that produce many high-level image features, other classification algorithms are likely to perform better than fully-connected neural networks. It follows that it is advisable to benchmark multiple classification algorithms on high-level image features produced from the CNN layers to improve performance or model size.

Keywords: Convolutional Neural Networks, CNN, Neural Networks, Computer Vision, Image Classification, Image Features, CIFAR-10, CIFAR-100, ILSVRC-2012

Resumo

Desde a criação da arquitetura da Rede Neural Convolucional (CNN), poucas modificações foram feitas nas camadas totalmente conectadas (fully-connected) no final das redes. Na classificação de imagens, estas mesmas camadas são responsáveis por criar os resultados finais da classificação, com base no output da última camada de filtros de imagem de alto nível (high-level image filters). Antes do avanço das CNNs, estes filtros de imagem eram feitos à mão e qualquer algoritmo de classificação era aplicado ao seu output. Como as redes neuronais aprendem os seus pesos gradualmente com gradientes sujeitos a um erro, as redes neuronais totalmente conectadas são uma escolha natural para as CNNs. No entanto, a superioridade das camadas totalmente conectadas numa CNN em relação a outros algoritmos de classificação é contestada. Depois da rede neural ser treinada, a abordagem passa por comparar diferentes algoritmos de classificação em CNNs, removendo o atual classificador totalmente conectado. Deste modo, o output achatado (flattened output) da última camada convolucional é usado como input para vários algoritmos de benchmarking. Para assegurar a generalização dos resultados, várias CNNs são treinadas no CIFAR-10, no CIFAR-100 e num subconjunto do ILSVRC-2012. Os resultados experimentais revelam que múltiplos algoritmos são capazes de superar redes neuronais totalmente conectadas, nomeadamente Regressão Logística, Support Vector Machines, XG Boosting, Random Forests e K-Nearest Neighbours. Além disso, a superioridade de cada algoritmo depende da estrutura subjacente da CNN e da natureza do problema de classificação. Para problemas com várias classes de output ou para CNNs que produzem muitos recursos de imagem de alto nível, outros algoritmos de classificação provavelmente terão um desempenho melhor do que redes neuronais totalmente conectadas. Segue-se que é aconselhável comparar vários algoritmos em recursos de imagem de alto nível produzidos a partir das camadas da CNN, para melhorar o desempenho ou o tamanho do modelo.

Palavras-chave: Rede Neural Convolucional, Rede Neural, Visão Computacional

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

HDF5	Hierarchical Data Format 5 (HDF5) is a data storage format. It is supported by many programming languages and freely usable.
JFT-300M	JFT-300M is a dataset of more than 375 million images with noisy labels. It was developed by Google. The dataset is not publicly accesible.
LeNet-5	LeNet-5 is one of the first ever designed convolutional neural network. It was created by LeCun et al. in 1998 to automate the process of handwriting detection of digits in checks.
MNIST database	The Modified National Institute of Standards and Technology (MNIST) database is a database of handwritten digits. It found its way into multiple research and practical applications, mainly in the field of computer vision.
Python Pickle	Python Pickle is a Python-specific data format that is used to serialise data. It can be used to serialise any kind of Python object. Pickle is a binary serialisation format and, therefore, only readable by machines.
WordNet	WordNet is a hierarchical lexical database that structures words into synsets. A synset is a set of synonyms for a given concept. WordNet is designed for the English language.

Acronyms

ADB	Adaptive Boosting (AdaBoost) Classifier.
ANN	Artificial Neural Network.
AUC	Area under the curve.
CART	Classification and Regression Trees.
CNN	Convolutional Neural Network.
DT	Decision Tree.
ELU	Exponential Linear Unit.
FN	False Negatives.
FP	False Positives.
FPR	False Positives Rate.
GAN	Generative Adversarial Networks.
GBC	Gradient Boosting Classifier.
GRU	Gated Recurrent Unit.
ID3	Iterative Dichotomiser 3.
ILSVRC-YYYY	ImageNet Large Scale Visual Recognition Competition YYYY.
KNN	K-Nearest Neighbours.
LR	Logistic Regression.
LSTM	Long Short Term Memory Networks.
MECE	Mutually Exclusive Collectively Exhaustive.
MLP	Multi Layer Perceptron.

OCR	Optical Character Recognition.
ReLU	Rectified Linear Unit.
ResNet	Residual Network.
RFE	Random Forests Classifier/Estimator.
RGB	Red Green Blue.
RNN	Recurrent Neural Networks.
ROC	Receiver Operating Characteristic.
SVM	Support Vector Machine.
TN	True Negatives.
TP	True Positives.
TPR	True Positives Rate.
Xception	eXtreme Inception.
XGB	eXtreme Gradient Boosting (XGBoost) Classifier.

Chapter 1

Introduction

1.1 Background Information

Computer vision is a sub-field of artificial intelligence and computer science that enables computers to develop a visual perception of real-world entities. Computer vision is about the automatic extraction, analysis, and understanding of information from imagery data. The field of computer vision can be divided into multiple sub-areas that each focus on specific information of the image data, summarised in figure 11:

1. **Classification**: Image classification is about the assignment of a label from a predefined fixed set of categories to an input image. It can be seen as the core of most computer vision tasks because all of the subsequently introduced sub-fields rely on image classification.

2. **Localisation**: Object localisation enhances the scope of image classification to adding information about the position of the identified object. Thus, additionally to identifying the object in question, the algorithm needs to find the object in the picture and determine its position. This is often done with bounding boxes, where the object is surrounded by a rectangular box indicating its position in the image. Object localisation consists of a classification and a regression problem.

3. **Detection**: Object detection enlarges the scope of object localisation in the sense that multiple objects from different classes can be identified and localised in an image. Given a fixed set of categories and an input image, the task of an object detection algorithm is to return the position and the type of every identified occurrence of one of the classes. Object Detection itself does not only consist

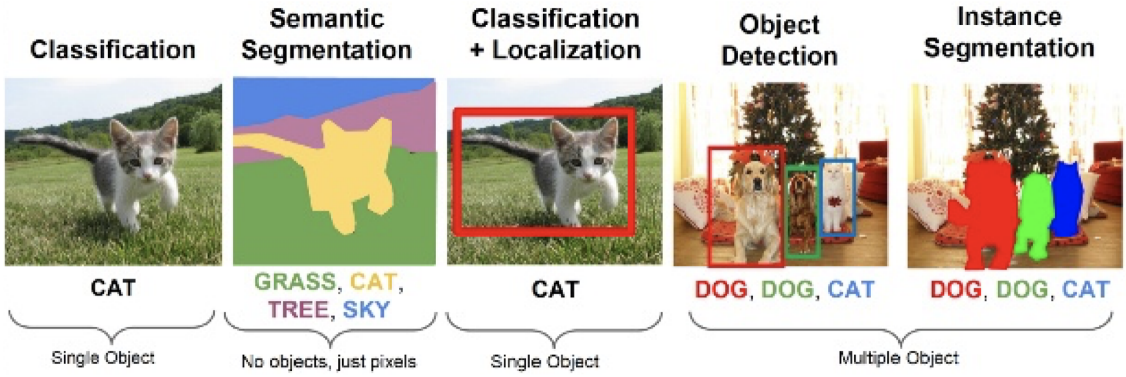


Figure 11: Summary of Different Computer Vision Tasks from [91]

of a regression a classification problem because the number of outputs of the algorithm is variable and not known beforehand.[1]

4. **Semantic Segmentation**: Semantic segmentation is a special case of image classification, where each pixel of an image is assigned to a specific category from a predefined set of categories. If an image contains multiple instances of the same base class, they are associated with the same category in semantic segmentation because the algorithm does not distinguish between instances of the same class.

5. **Instance Segmentation**: Instance segmentation is a special case of semantic segmentation where multiple instances of the same class are classified into different categories. Given a picture of two dogs in front of a random background and the input classes ”dog” and ”background”, the previously described semantic segmentation would classify each pixel of the dogs as category ”dog” and each pixel from the background as ”background”. Instance segmentation, on the other hand, would classify the two objects from the class dog as two different categories.

The concepts introduced in this thesis are applied to the task of image classification, but they can be integrated into the classification part of the other computer vision sub-areas.

Computer vision has gained traction since the breakthrough of artificial neural networks (ANN), which was fostered by the increases in available computational power and training data. But it was only in 2012 when a research team consisting of Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton from the University of Toronto constructed AlexNet [51] for the ImageNet Large Scale Visual Recognition Challenge

[1] more about object detection and how to resolve the problem of object detection can be read in [72], [71], [70] and [22]

(ILSVRC) in 2012[2], that a convolutional neural network (CNN) architecture outperformed traditional approaches in the classification and localisation tasks. This initiated the hype in deep learning for computer vision in the subsequent years [75] [53][3]. After that, CNN architectures were improved in many ways, found their way into multiple fields of research and were successfully applied in the industry. Nonetheless, the same Geoffrey Hinton that helped CNNs to their success is one of their greatest critics. He came up with the idea of capsule networks with a team of researchers from Google Brain [27], which take a different approach and give promising results, especially for the case of little training data and strong variations within training and test data [79]. Although these networks are conceptually fundamentally different to CNNs, they still make use of convolutional layers and fully-connected layers[4].

1.2 Structure of the Thesis

Although a general introduction to multi-layer perceptrons (MLPs) is given in section 3.6, the thesis assumes a general knowledge of important concepts related to ANNs, including the concepts of batch learning, gradient descent, activation functions and the notions of overfitting and generalisation. Further reading on these topics can be done in [26], [32]. Chapter 2 explains convolutional neural networks, the most relevant innovations, and state-of-the-art architectures. Chapter 3 introduces different classification algorithms that are commonly used in machine learning. Furthermore, different evaluation methods for model performance on classification tasks are presented. Afterwards, chapter 4 outlines the experimental setup of the thesis used to test the research question proposed above, including the datasets, the network architectures, the chosen classification algorithms, the software architecture to test the research question and the evaluation metrics applied. Lastly, chapter 5 provides an evaluation of the experimental results. Along with presenting the intermediate datasets created from the high-level image features, the chapter describes the final results, makes a comparison and a final evaluation.

1.3 Purpose of the Thesis

Image Recognition is arguably one of the most active parts of research within machine learning. Before the dominance of neural networks in computer vision research, "in the traditional model of pattern recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. A trainable classifier then categorises the resulting feature vectors into classes."([105], p.5) In this traditional model, any classification algorithm is used to distinguish the classes

[2] referred to as ILSVRC-2012

[3] they used the team name "SuperVision"

[4] given the loss function is enhanced with decoder neural network regularisation, see section 2.3.9

based on the hand-designed image features. The emergence of convolutional neural networks in computer vision meant a shift from hand-designed feature extractors to automatically generated feature extractors trained with backpropagation.

As chapter 2 shows, there have been many developments of CNN architectures recently that substantially change the model architectures. But chapter 2 also shows that all of these advances have been made in the earlier layers of the neural network, meaning everything up to the fully-connected layers. Little research has been devoted to the proficiency of the fully-connected layers. Fully-connected multi-layer perceptrons (MLP) seem to be the design choice for all network architectures with little research or modifications done to these layers. But as the no-free-lunch theorem from chapter 3 shows, no classifier can be said to be superior to all other classifiers. Therefore, following the idea of the traditional model of pattern recognition where any classification algorithm was used, the evolved research question for this thesis is:

Research Question "Can classification model performance in computer vision be improved by using different classification algorithms on high-level image features?"

Both convolutional and fully-connected layers of a neural network learn their weights during training with the weights being backpropagated from the output through the fully-connected-layers back to the convolutional layers. Due to this nature of the learning process, multi-layer perceptrons are a natural choice for convolutional neural networks. Convolutional neural network features can only be learned through a backpropagated error that usually comes from a multi-layer perceptron or directly from the output. Therefore, the focus of this thesis is not directly on replacing the fully-connected layers in the learning process, but rather enhancing the learning process to a two-step procedure:

1. **Regular CNN Training Process:** Train a given CNN architecture, including fully-connected layers to make convolutional layers learn image features from image inputs
2. **Enhancement of the Training Process:** Replace the fully-connected layers with a different classification algorithm and train the algorithm based on the high-level image features produced from the convolutional layers in the first step

This research question is tested in multiple settings on multiple datasets to investigate further under which conditions certain classification algorithms potentially perform better. The datasets benchmarked in this thesis are CIFAR-10, CIFAR-100 and a subset of ILSVRC-2012.

Chapter 2

Artificial Neural Networks for Computer Vision

In the late 19th century, the Spanish neuroscientist Santiago Ramon y Cajal discovered that the human nervous system consists of many independent nerve cells that are connected through nerve synapses that transfer nerve impulses between the cells. His works contributed significantly to the understanding of the human brain and the human thought process, which is the foundation for human intelligence. Neuroscientists and computer scientists, such as McCulloch and Pitts (1943) [62], Rosenblatt (1958) [73], Paul Werbos (1974) [104], Rumelhart, Hinton, Williams (1986) [74] and others, used this idea of how the human brain works to develop, train and improve artificial neural networks. Following a period of disrepute, data became more easily accessible through the internet and computational means increased following Moore's Law[1], ANNs gained popularity again and found their way into the industry. The breakthrough of ANNs led to an increase in research in the area and the development of advanced network architectures for specific domains, such as recurrent neural networks (RNNs) and their variations long short-term memory networks (LSTM) [37] and gated recurrent unit networks (GRU) [13], generative adversarial neural networks (GAN) [25] and others.

Although the visual perception of objects is an easy task for most human beings, it was traditionally not a task that computers were particularly good at. The 1980s brought some theoretical advances on custom tailored problem domains, but they were not widely applicable in practice. In the early 20th century, the idea of developing features

[1] as inspired by Gordon Moore's paper [65]

that are invariant to changes emerged. The idea is to identify critical features (building blocks) of an object and match these features to similar objects rather than pattern matching the entire object. This idea of leveraging object features along with the increasing popularity of ANNs gave rise to the breakthrough of convolutional neural networks.

2.1 How are Digital Images Represented

Before the introduction of convolutional neural networks, a common understanding of digital images is necessary.

An image is a two-dimensional function $f(x,y)$ that describes the color f at the spatial coordinates (x,y). A digital image is one where x, y, and f are finite, meaning the spatial coordinates are covered by a predefined number of elements, called pixels. The computational representation of a digital image is a three-dimensional array with the dimensions width w, height h and depth d: $I \in R^{w \times h \times d}$. Whereas width w and height h determine the shape of the pixel grid of an image, the depth determines the colour of a given pixel and is described by an array of length d. Common representations of images are grey-scale ($d = 1$) and RGB ($d = 3$). The depth is also referred to as the number of channels ([99], p. 3) but the author of this thesis chose to use the notion of depth to describe input and output channels.

Features of an image are any patterns of pixels that have unique shapes. Low-level features are simple shapes, e.g., lines and edges of different intensities. Higher-level features are combinations of multiple layers of lower-level features. Higher-level level features are more complex patterns and a combination of multiple lower-level features, e.g., a nose made up of multiple lower-level edges. The detection of features within an image is done with image filters. Each filter is responsible for detecting the existence of one feature. For more, see section 2.3.1.

2.2 Challenges in Computer Vision

Perturbation of imagery data can occur in any of the three dimensions. An object in question can be shifted, scaled or distorted in the grid and change its colour values while still belonging to the same category of images. Further variations regard the object itself and within its context, such as

- **Deformation**: The real-world object can take on different shapes and poses and be deformed in many ways.

- **Occlusion**: The real-world object could be occluded, e.g., the real world entity hides behind another object with just a part of it visible in the image.
- **Viewpoint Variation**: The image data can capture a real-world entity from different perspectives.
- **Scale Variation**: Apart from the scale variation within an image representation, real-world entities can also vary in size while still belonging to the same category.
- **Background Clutter**: The object and its background are composed of similar colours or structures and thus make it hard to differentiate between them.
- **Intra-class Variation**: Chosen classes usually represent concept from the real world. These concepts can be rather broad with many intra-class variations. For example, the concept of a dog can be represented by different breeds while still belonging to the class dog.

For a visual understanding of these challenges, see Appendix A.1.1.

An ideal computer vision algorithm is robust to all these intra-category variations within the images, while perceiving inter-category differences.

2.3 Convolutional Neural Networks

The neurobiologists David Hubel and Torsten Wiesel analysed the visual processing mechanisms in mammals in 1959, using cats as a reference to understand the human visual system. They attached electrodes to the back of the brains of cats to examine which visual stimuli makes the neurons in the primary visual cortex of a cat brain respond. They found out in their research that visual processing begins with simple, low-level structures of the world, such as edges and their orientations, which are then aggregated to more complex, high-level concepts in the visual processing pathway. Transferring this idea of the visual system to humans, one can conclude that the human brain builds up the perception of an object by aggregating multiple low-level image features and successively combining these building blocks to a larger high-level representation.

Convolutional neural networks adapt this principle from nature in that they aim to automatically detect high- and low-level features within the network during the training process. CNNs combine the architectural ideas of local receptive fields from section 2.3.1, shared weights (weight replication) from section 2.3.1, and spatial subsampling from section 2.3.3. ([105], p.6)

> *The ability of multi-layer networks trained with gradient descent to learn complex, high-dimensional, non-linear mappings from large collections of examples makes them obvious candidates for image recognition tasks.* LeCun et al. (1998) ([105], p.5)

Not only does this notion of learning more accurately imitate the human brain because CNNs automatically detect 2-dimensional local structures within images without prior manual definition. CNN structures are also computationally more efficient in comparison to their fully-connected counterparts because the convolutional network layers require fewer weights. Thus, the model has to learn fewer parameters during the training process.

The first published CNN, LeNet-5, was developed by LeCun et al. (1998) and was applied in alphanumeric character recognition to read "several million checks per day" ([105], p.1) automatically in 1998.

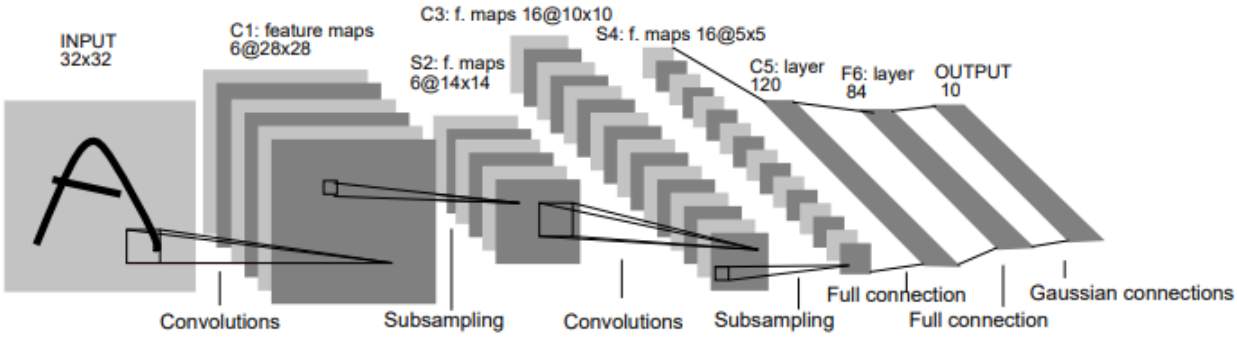


Figure 21: LeNet-5 Architecture from LeCun et al. (1998)

After their introduction, CNNs remained at the sidelines of research until AlexNet won the ILSVRC-2012 challenge in the classification and localisation task by a great margin compared to the other approaches. ILSVRC is the most famous computer vision challenge based on the ImageNet database, as introduced in section 4.3.3. For the classification task, AlexNet reached a top-5 error of 15.32 %, whereas the next best model architecture reached 26.17 %. AlexNet [51] is an 8-layer neural network that strongly resembles LeNet-5 from figure 21 in its architecture. After this success, CNNs became state-of-the-art in computer vision with more and more innovative additions to the networks outperforming earlier models. The next decisive changes to the network architecture were introduced in ILSVRC-2014, with GoogLeNet from Szegedy et al. (2014) and VGG from Simonyan and Zisserman (2015) reaching a top-5 error of 6.67 % and 7.32 % respectively ([95], p. 9) [54]. The main contribution of VGG was to use relatively small convolutional filters combined with deeper neural networks (16-19 layers) [86]. GoogLeNet is a 22-layer network without fully-connected layers at its end. Its main contribution to subsequent model architectures is the inception module (see section 2.3.7). Apart from computational limitations, a roadblock to even deeper

neural network architectures was the fact that deeper networks empirically produced worse results on training and test data [31]. They should, in theory, be at least as good as their shallower counterparts. The ResNet-152 architecture from He et al. (2015) won the ILSVRC-2015 challenge with a top-5 error of 3.57 %. With a depth of 152 layers, it is the first model to beat human performance. The solution to training deeper networks are residual blocks (see section 2.3.8), giving the name to ResNet. [31] [54]

The following subsections will explain the introduced concepts in more detail. The following sections show the key areas of research and improvements to CNNs from the past few years.

2.3.1 Convolutional Layers

A convolutional layer creates an n-dimensional output representation (depth n) of given input data (e.g., digital image data, see section 2.1), where each dimension is created with respect to one image filter. A convolutional layer consists of at least one, though usually multiple, different image filters ([105], p. 6), thus $n \in \mathbb{N}_1$.

Figure 22 visualises the concept of one convolutional layer. The convolutional layer consists of n filters. These n filters are applied on the input image. In this case, the input image consists of three feature maps, as would be the case for an RGB image. Through the application of these n filters, an output image with n feature maps is created. This output image will then be the input for the next convolutional layer.

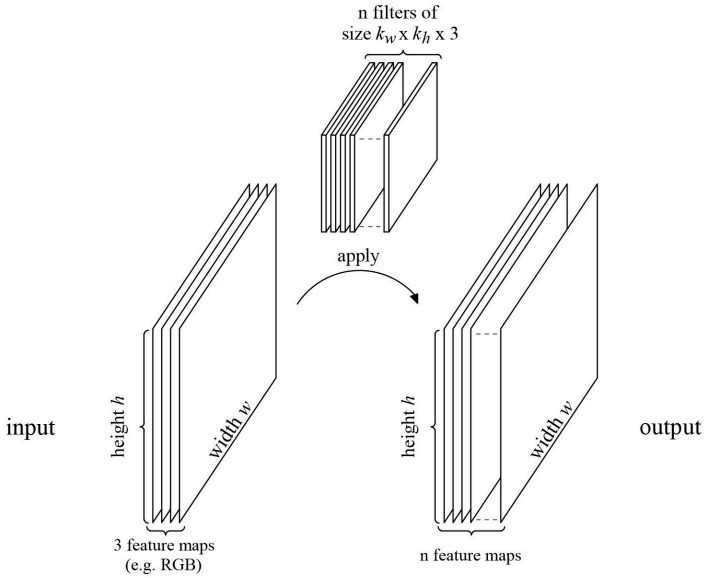


Figure 22: Application of convolutional layer with n image filters of size $k_w \times k_h \times 3$ with stride $s = 1$ on input data with size width$w$ × height h and an input depth of 3 [99]

The following subsections introduce the concept of image filters, how they can be learned and common model choices.

Image Filters

The following chapter introduces the concept of image filters for two-dimensional images. These two-dimensional images can still have three-dimensional representations in digital formats, e.g., if the color is represented in RGB format. Nonetheless, the concepts introduced can easily be enhanced to inputs with more spatial dimensions. This thesis only handles two-dimensional images.

An image filter (also called filter kernel) is a tensor $F \in \mathbb{R}^{k_w \times k_h \times I_d}$, where k_w, $k_h \in \mathbb{N}_1$ determine the filter's width and height respectively, and I_d corresponds to the depth of the input image to the convolutional layer. The output image I' is produced through a convolution of the input image $I \in \mathbb{R}^{w \times h \times d}$ with the image filter. Basically, the Hadamard product is applied to the image filter and a local input field, called receptive field, of size $k_w \times k_h \times I_d$ with center (x, y). Each entry, e.g., pixel, in the grid of the output image $I'(x, y)$ is thus calculated as a sum of these point-wise multiplied tensors. ([99], p. 3)

The formula for a given output entry can be found in equation 2.1 ([99], p. 3).

$$I'(x,y) = b + \sum_{i_x=1-\frac{k_w}{2}}^{\frac{k_w}{2}} \sum_{i_y=1-\frac{k_h}{2}}^{\frac{k_h}{2}} \sum_{i_c=1}^{I_d} I(x+i_x, y+i_y, i_c) \cdot F(i_x, i_y, i_c) \tag{2.1}$$

$$\text{for } x \in \{1, \cdots, I_w\} \text{ and } y \in \{1, \cdots, I_h\}, \text{ where } b \in \mathbb{R} \text{ denotes the bias}$$

Due to the aggregation from the summing operations, the depth of the output from a single image filter is always 1.

Figure 23 visualises this procedure.

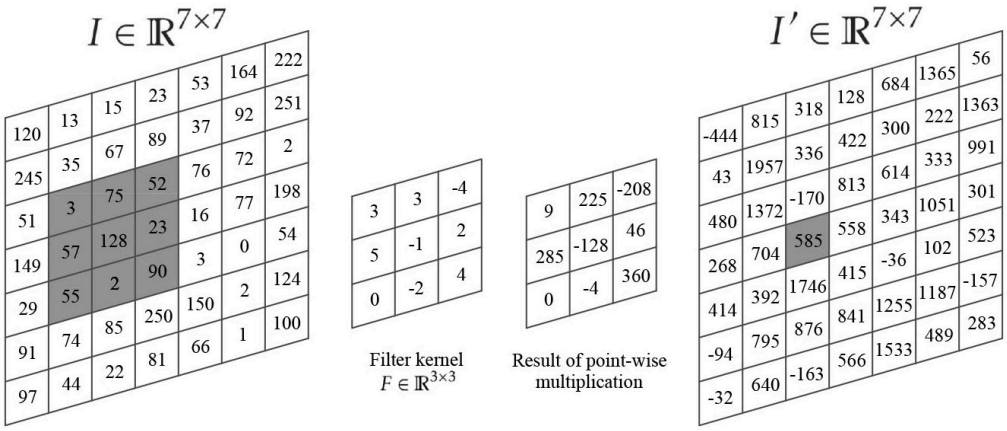


Figure 23: Visualisation of application of a linear $k \times k \times 1$ image filter on receptive field in input [99]

Formula 2.1 states the calculation of the output pixel at $I'(x,y)$ for a given receptive field with center $I(x,y)$. This operation is applied for all pixels $x \in \{1,\cdots,I_w\}$, $y \in \{1,\cdots,I_h\}$ and for all n image filters to create the full output image $I \in \mathbb{R}^{w'\times h'\times d'}$. In this case, the image filter operation can be thought of as sliding over the input image from the top left to the bottom right corner, shifting the center of the receptive field by 1 unit and performing the convolutional operation to produce the output data. Every output position can be regarded as a score of how well the given receptive field responded to the image filter. The higher the score of the output position, the more likely it is, that the receptive field in the input data was representing a structure formalised by the image filter. In the end, the output data will have a score at each position, indicating the likelihood that the image filter representing a given feature is present at that position in the input data. This also makes CNNs more robust to distortions and shifts of the object within an input image, because a distortion or shift would eventually lead to a distorted or shifted value in the output image. It follows that the feature would still be detected while its exact position is irrelevant.

But the question remains of how a receptive field greater than 1 can be produced at the image boundaries. In other words: What is the receptive field when the centre lies at the border of the image, and the receptive field would thus partially fall out of the boundaries of the input data? The two most common approaches to border handling are ([28], p. 3):

- Eliminate the border-dependent pixels by cropping the output image to perform convolution on the inner pixels only where possible or

- Pad the input image to account for the missing input pixels.

The first approach implies a diminishing output size of $k_w - 1$ and $k_h - 1$ for the width and height respectively of the convolutional layer. The second approach requires a padding technique. Common techniques are "replicate", "reflect", "wrap" and "constant" (e.g., zero-padding) (for more, see [28]).

As described above, performing this convolution leads to an overlap between the receptive fields of contiguous input positions. One hyper-parameter of the convolutional layer is the stride $s \in \mathbb{N}_1$. The stride determines by how many contiguous input positions the receptive field should be moved when creating the image output. The case described above uses a stride of 1, meaning no input position is skipped. If the stride is chosen to be bigger than 1, the effective output image gets reduced in size. Apart from the stride, the output image size is only determined by the border handling procedure for uneven filter sizes. Usually, the stride and border handling are chosen such as to preserve the image size. The filter width and height are commonly set to be uneven.

Each image filter performs $k_w \times k_h \times I_d$ multiplications at each position of the input. Hence, over the full image, each filter performs $I'_w \cdot I'_h \cdot k_w \cdot k_h \cdot I_d$ multiplications. For k_n image filters, each convolutional layer then performs the following number of multiplications[2]:

$$k_n \cdot I'_w \cdot I'_h \cdot k_w \cdot k_h \cdot I_d \tag{2.2}$$

Convolutional Layer Training

During training, the convolutional layers learn to construct the image filters. The weights of the image filters are the parameters of the learning process that are adapted to the training data ([99], p. 5). Convolutional layers leverage the concept of shared weights for the learning process, introduced by Waibel et al. (1989). Since the identical image filter is repetitively applied over the positions in the input image, this is like connecting a given position from the image filter to each input position with the same weight. The sizes of the receptive fields determines the number of necessary weights. Hence, $k_w \cdot k_h + 1$ (bias) parameters need to be learned per image filter.

Because of the concept of shared weights, every convolutional layer can be formulated as an MLP, with the drawbacks of then having more parameters to train. Nonetheless, this shows that a convolutional layer can be trained using gradient descent [60]. But for the case of shared weights as in CNNs, the training algorithm is changed to account

[2] Assuming that the initial image dimensions are preserved, e.g., a stride of 1 and padding for border handling, this simplifies to $k_n \cdot I_w \cdot I_h \cdot k_w \cdot k_h \cdot I_d$

for "the average of all corresponding [...] weight changes" ([100], p. 331) with respect to the training data.

Convolutional Layer Hyper-Parameters

The hyper-parameters to tune are ([99], p.5)

- the number of image filters $n \in \mathbb{N}_1$
- the image filter width and height $k_w, k_h \in \mathbb{N}_1$
- the activation function ϕ after the convolution
- the stride $s \in \mathbb{N}_1$
- the border handling function

Typical choices for n are values of an exponential function to the base 2 (numbers from the binary series), e.g., $n \in \{2^5, 2^6, 2^7\}$, up to 2^{11}. The kernel width k_w and height k_h are usually chosen to be equal and uneven. thus $k_w = k_h = k \in \{1, 3, 5, 7, 11\}$. The activation function is usually a rectified linear unit (ReLU) activation or other derived versions, such as ELU. The stride is usually set to $s = 1$ ([99], p. 5) and border handling is usually done through constant padding with 0s.

As outlined before, the inventors of VGG, Simonyan and Zisserman (2015), discovered that shallower but deeper networks are advantageous compared to their flatter but wider counterparts. The intuition behind using smaller filters combined with deeper networks is that the deeper networks can have the same effective receptive field as the wider network, but also increase the non-linearity and reduce the model parameters. The winners of ILSVRC-2013, Zeiler and Fergus, use 96 different 7×7 convolutional filters in the first hidden layer of their network ([111]: p. 6). Assuming a stride of 1[3], using three layers of 3×3 convolutional filters has the same effective receptive field as a 7×7 filter. Apart from the increased non-linearity through a deeper network, this separation reduces the model parameters in the convolutional layers. In this example, the parameters per channel reduce from 7^2 for one layer with a receptive field of 7×7 to $3 \cdot 3^2$ for three layers with receptive fields of 3×3. [86]

2.3.2 Depthwise Separable Convolutions

Convolutional layers are computationally expensive because they perform many multiplications, as shown in equation 2.2. This number of multiplications can be a bottleneck when training large convolutional neural networks.

[3] this assumption is for simplification; Zeiler and Fergus actually use a stride of 2 in their network

Depthwise separable convolution, as introduced by Chollet (2016) [12], breaks down the procedure of applying image filters from section 2.3.1 into two steps:

1. Depthwise Convolution: While regular convolution applies the image filter F on all channels in the input depth I_d, depth-wise convolution applies I_d image filters with a depth of 1: $k_w \times k_h \times 1$. These image filters of depth 1 are applied on all input channels, thus I_d times. The resulting cubicle is of output size $I'_w \times I'_h \times I_d$[4]. This produces $I'_w \cdot I'_h \cdot k_w \cdot k_h \cdot I_d$ multiplications.

2. Pointwise Convolution: The input to the pointwise convolution is the output from the depth-wise convolution and thus of size $I'_w \times I'_h \times I_d$. On this input, 1×1 convolutions are performed with a filter depth of I_d according to the number of input layers. These 1×1 convolutions inevitably preserve the image width and height. According to the number of desired output filters, these 1×1 convolutions are performed k_n times. Therefore the output image I' will be of size $I'_w \times I'_h \times k_n$. In total, that leads to $I_d \cdot I'_w \cdot I'_h \cdot k_n$ multiplications per convolutional layer.

The total number of multiplications thus amounts to the following number of calculations:

$$I'_w \cdot I'_h \cdot k_w \cdot k_h \cdot I_d + I_d \cdot I'_w \cdot I'_h \cdot k_n = I'_w \cdot I'_h \cdot I_d \cdot (k_w \cdot k_h + k_n) \tag{2.3}$$

The ratio between the original convolution from equation 2.2 and the depth-wise separable convolutions from equation 2.3 for the identical target image dimensions I' is then

$$\frac{I'_w \cdot I'_h \cdot I_d \cdot (k_w \cdot k_h + k_n)}{k_n \cdot I'_w \cdot I'_h \cdot k_w \cdot k_h \cdot I_d} = \frac{k_w \cdot k_h + k_n}{k_w \cdot k_h \cdot k_n} = \frac{1}{k_w \cdot k_h} + \frac{1}{k_n} \tag{2.4}$$

This ratio is smaller than or equal to 1 for all cases where $k_n \geq 2 \wedge k_w \cdot k_h \geq 2$ and thus depth-wise convolutions are usually more efficient than regular convolutional layers.

The ratio for the number of parameters between the regular convolution and the depth-wise separable convolution is equal to the ratio of the multiplications in equation 2.4. Thus, this is also usually reduced.

Apart from the eXtreme Inception (Xception) neural network architecture from Chollet (2016)[12], the idea of depth-wise separable convolutions is used in many other model architectures, such as MobileNet [39], a neural network designed to run on

[4] assuming that initial dimensions are preserved during the convolution, the output size is $I_w \times I_h \times I_d$

the limited hardware capabilities of end-user smartphones, in a proposed neural machine translation architecture [45] and a multi-purpose model designed by Kaiser et al. (2017), used for translation, image classification, speech recognition and parsing [46].

2.3.3 Pooling Layers

Pooling Layers

- make networks robust to translational variance,
- make networks robust to minor local changes and
- reduce the size of the data.

A pooling operation makes a summary of a receptive field $p_w \times p_h$, $p_w, p_h \in \mathbb{N}_1$, of the input data. It is responsible for the routing of information through the network by deciding which information to pass on to the next layer. Equivalently to the image filter in convolutional layers, the pooling operation is performed over the positions of the input image with a predefined stride $s \in \mathbb{N}_1$. Usually, the stride is chosen to be larger than 1, such that the pooling layer reduces the size of the input data by a factor of s^2. The pooling window defined by p_w and p_h can be any subset of the input image. If p_w and p_h are equal, the pooling window preserves the ratio between width and height from the input image. Thus, they are typically chosen to be equal and between 2 and 5, depending on the input image size and other factors (see also [51], [86], [95], [31]).

As the pooling operation only summarises the input data, which is usually passed through an activation function before, it is not common to apply an activation function after this layer. If padding is applied at the borders, Max-Pooling uses $-\infty$ values at the overlapping areas such that these entries cannot possibly be the maximum values for the max pooling operation.

Typical pooling functions are Max-Pooling, Average/Mean-Pooling, L2-Pooling, Stochastic-Pooling, Spatial Pyramid Pooling and Generalising Pooling Functions (for more, see [110], [7], [8], [57]).

The pooling function is often criticised due to its inevitable information loss. By sub-sampling the data and making it invariant to small perturbations, it loses the precise spatial relation between higher-level features (see also section 2.3.9).

2.3.4 Fully-Connected Layers

Fully-connected layers are layers with neurons that have full connections to all outputs from the previous layer. While the convolutional and pooling layers perform the detection and extraction of features from an image, the fully-connected layer(s) of a convolutional neural network create the final output of the network. Fully-connected layers of a convolutional neural network "are required to learn non-linear combinations of learned features" ([3], p. 2) from the previous layers. Fully-connected layers are simply a multi-layer perceptron classifier, as introduced in 3.6.

As described in sections 2.3.1 and 2.3.3, the output from the previous layer is a tensor $I \in \mathbb{R}^{w \times h \times n}$. Since fully-connected layers have no notion of the location of the input, they work with 1-dimensional input data. Therefore, the data is flattened to be a 1-dimensional feature vector. As introduced in section 2.3.1, convolutional layers are simply a specific case of regular fully-connected layers. Thus, what is referred to as fully-connected layers in neural networks are technically 1×1-dimensional convolutions with a full connection table, meaning there are no shared weights.

The numeric restrictions of the output depend on the problem domain. The output is either continuous for regression problems or a likelihood score for a classification problem with each output value indicating the probability for a given class. The likelihood is usually expressed between 0 and 1, e.g., through the application of a softmax function. The number of output neurons for the final fully-connected layer (output layer) is equivalent to the number of desired output classes. For a classification problem with n classes, the output is usually of length n and bounded between 0 and 1, such that every output neuron reflects the probability of a given class. Alternatively, each output neuron can reflect a binary value that encodes the classification. For this case, $log_2 n$ output neurons are necessary.

2.3.5 Dropout Layer

Dropout is used to foster generalisation of the learning process and prevent overfitting. Hinton et al. [34] introduced the idea of dropout to neural networks. Dropout means randomly deactivating certain neurons with a probability p during each run of the learning process. Neurons are deactivated by setting their output to 0. Dropout forces the network to more robustly learn features with multiple neuron combinations without the ability to ground certain assumptions on specific neurons (for more see [89]).

A dropout layer is simply a tensor $D \in \mathbb{R}^{w \times h \times d}$ of the shape of the input data that contains a 0 or a 1 for every position, depending on whether the neuron is activated (1) or not (0). Thus, dropout is calculated as the Hadamard product between this dropout

tensor and the input data: $D \circ in$. Every element d_i in D is independently sampled from a Bernoulli distribution with probability p. Since dropout reduces the number of outputs created from the network, the final output is multiplied with $\frac{1}{1-p}$ when dropout is applied to account for the missing neurons.

The dropout probability p is a hyper-parameter of the layer, typically set to 0.5. Furthermore, layers closer to the output tend to have higher dropout probabilities than layers closer to the input.

Dropout is only done during the training phase and not at inference time.

2.3.6 Batch Normalisation Layers

Batch normalisation is a means for more efficient deep network training by reducing internal covariate shift. Internal covariate shift refers to the phenomenon that the outputs of the nodes of different layers of a neural network follow different distributions ([85], p. 228). Internal covariate shift is mainly caused by layers closer to the output learning better with respect to the error than layers closer to the input.

Batch normalisation enables a more stable gradient propagation in a neural network, especially for deep neural networks. It can also lead to speed improvements in the learning process ([15], p.1), the possibility to use "higher learning rates without the risk of divergence and other ill side effects" ([80], p.59). It further reduces the risks associated with saturating nonlinearities (e.g., softmax) ([43], p.2) and can improve parameter initialisation ([43], p.1).

The method from Ioffe and Szegedy (2015) performs a point-wise normalisation over the mini-batches with input $x = (X^{(1)}, \cdots, X^{(d)})$. The formula is as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \bar{x}^{(k)}}{\sqrt{s'\left(x^{(k)}\right)^2 + \varepsilon}} \tag{2.5}$$

with sample mean $\bar{x}^{(k)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)}$ and sample variance $s'\left(x^{(k)}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x_i^{(k)} - \bar{x}^{(k)}\right)$ and mini-batch sample size $m \in \mathbb{N}_1$ and constant $\varepsilon > 0 \in \mathbb{R}$.

This is an adjusted calculation of z-score normalisation over the mini batch by subtracting the sample mean and dividing by an adjusted sample standard deviation. The standard deviation is adjusted by adding a small constant ε to account for the case of no variance in the data (division by zero).

Additional parameters for feature scaling and shifting are introduced which are learned

in the training process.

$$y^{(k)} = \gamma^{(k)} \cdot \hat{x}^{(k)} + \beta^{(k)} \tag{2.6}$$

These parameters allow for the model to adjust the normalisation that was done before. For example, they enable the normalisation layer to become an identity layer by reverting the standardisation by the sample variance for γ and sample mean for β, thus simply passing the input data through.

Adjusted forms of normalisation exist (for more, see [102]), though batch normalisation is the most common one.

2.3.7 Inception Modules

An inception module[5], introduced by Szegedy et al. (2014) [95] follows the idea of a network within a network, initially introduced by [58]. An inception block is a summary of multiple contained layers. The idea is to design a good local network topology that can then be combined to a larger network. These sub-network operations are performed in parallel and concatenated depth-wise at the end. Each sub-network consists of convolutional layers or pooling layers.

As in the case for GoogLeNet, the inception module contains convolutional layers with receptive field sizes of 1×1, 3×3, and 5×5 and a 3×3 max pooling operation, see figure 24.

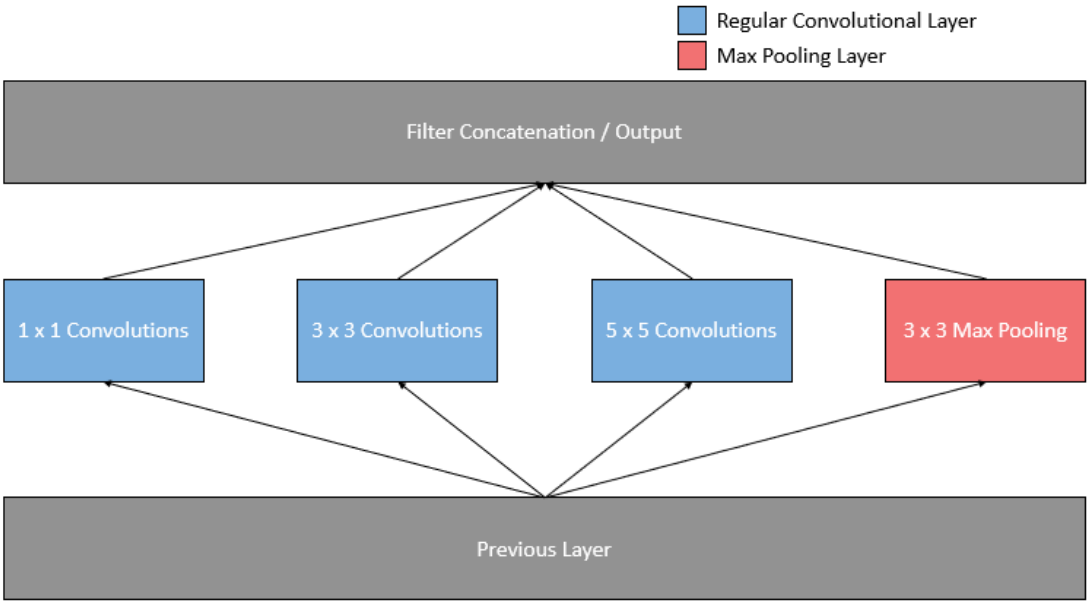


Figure 24: Simple Inception Module, modified from Szegedy et al. (2014) [95]

[5]also called inception module, concatenation block, aggregation block

The abovedescribed model architecture adds multiple convolutional layers to the network and thus requires additional parameters to be learned.

In the following, the input image to the inception module is denoted as I^{inp}, with the dimensions I_w^{inp}, I_h^{inp} and I_d^{inp}. The input to a given image filter k is denoted as I^k and is equal to I^{inp} when k is performed directly on the input. The set of initial nodes in the inception block are referred to as inception nodes K^{inc}. The set of added 1×1 convolutions are referred to as reduction nodes K^{red}. The set of all convolutional layers is represented by $K \setminus K^1$, with $k_w \geq 2$ and $k_h \geq 2$. The set of 1×1 convolutions, where $k_w = k_h = 1$, is referred to as K^1. All complexity reduction convolutional nets that occur after a pooling layer are referred to as K^{after_pool}. Furthermore, a stride of 1 and padding at the borders for both convolutional and pooling layers are assumed in this subsection to preserve the image dimension and facilitate the computation. The idea of an inception module works independently of the stride and border handling function chosen as long as the dimensions are identical for the depth-wise concatenation.

Given the setup introduced above, each convolutional layer k in the set of all convolutional layers K^{inc} performs $I_w^{inp} \cdot I_h^{inp} \cdot I_d^{inp} \cdot k_w \cdot k_h \cdot k_n$ operations. For the computationally complex inception module from figure 24, the number of computations is shown in equation 2.7.

$$Operations_{complex} = I_w^{inp} \cdot I_h^{inp} \cdot \sum_{k \in K^{inc}} I_d^{inp} \cdot k_n \cdot k_w \cdot k_h \tag{2.7}$$

For the network structure in figure 24, this means $I_w^{inp} \cdot I_h^{inp} \cdot I_d^{inp} \cdot (k_{n1} + 9k_{n2} + 25k_{n3})$ convolutional operations. It can be easily seen how this number gets large quickly and hinders network performance. This problem is amplified by the fact that the layers are concatenated depth-wise. The final depth is $I'_d = I_d^{inp} + k_{n1} + k_{n2} + k_{n3}$. Since the pooling layer only preserves the depth of the input image I_d^{inp}, concatenation with the convolutional layers can only increase the depth of the output image I'. This effect increases network size and complexity, especially over multiple inception modules.

The abovedescribed problems can be solved with 1×1 convolutions, which serve "as dimension reduction modules to remove computational bottlenecks" ([95], p. 2). A 1×1 convolution preserves the spatial dimensions of an image but can reduce its depth, if $k_n < I_d^k$. The 1×1 convolution is applied before the original set of convolutions, except if it is a 1×1 convolution itself, thus only on $K^{inc} \setminus K^1$. It is also applied after the pooling layer, to reduce the depth of the output, see figure 25.

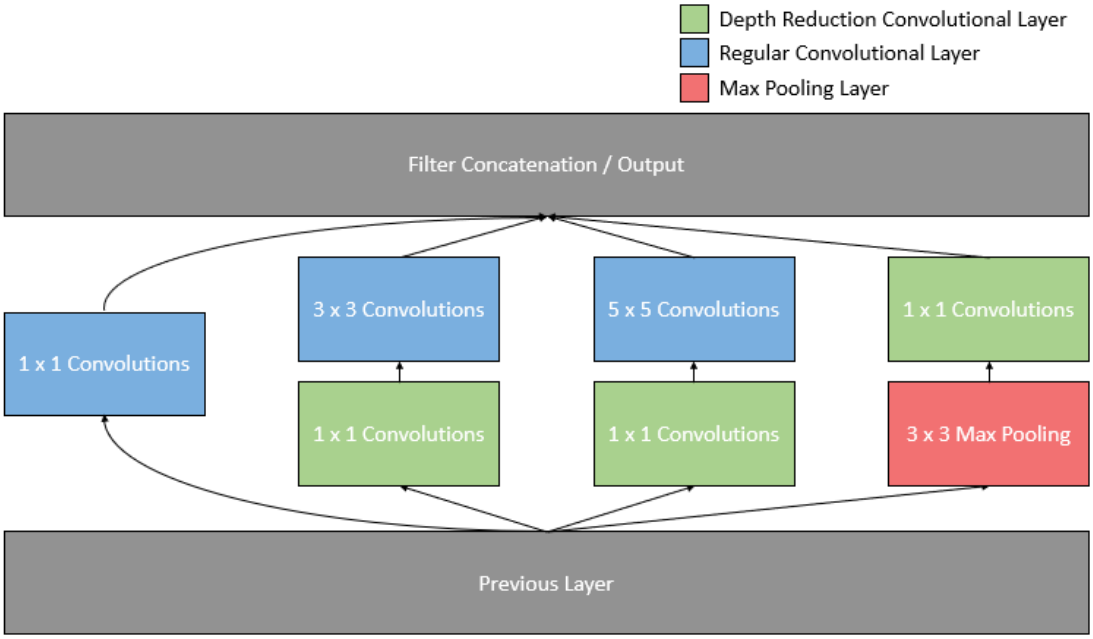


Figure 25: Advanced Inception Module designed to reduce the overall complexity, modified from Szegedy et al. (2014) [95]

The first advantage of this network structure is that the depth of the concatenation node is not necessarily larger than the input depth to the inception module because the pooling from the output layer can be down-sampled to length n. Thus the final depth is:

$$I'_d = \sum_{k \in K^{inc}} k_n + \sum_{k \in K^{after\_pool}} k_n \tag{2.8}$$

Despite additional complexity of $\sum_{k \in K^{red}} I_w^{inp} \cdot I_h^{inp} \cdot I_d^{inp} \cdot k_n$ multiplications caused by the additional convolutional layers, the goal is to reduce the overall complexity by reducing the depth parameter in the subsequent layers from I_d^{inp} to k_n in the formula above.

The total number of operations is thus

$$Operations_{reduced} = I_w^{inp} \cdot I_h^{inp} \cdot \left(\sum_{k \in K^{red}} I_d^{inp} \cdot k_n + \sum_{k \in K^{inc}} I_d^k \cdot k_n \cdot k_w \cdot k_h \right) \tag{2.9}$$

$Operations_{complex}$ should be bigger than $Operations_{reduced}$. In the following, it is assumed that this condition holds and a simplified formula is derived to verify the

correct choice of parameters with regards to the width k_w, height k_h and depth k_n of all convolutional layers within the inception module $k \in K$.

$$Operations_{complex} > Operations_{reduced} \tag{2.10}$$

$$\Leftrightarrow I_w^{inp} \cdot I_h^{inp} \cdot \sum_{k \in K^{inc}} I_d^{inp} \cdot k_n \cdot k_w \cdot k_h > I_w^{inp} \cdot I_h^{inp} \cdot \left(\sum_{k \in K^{red}} I_d^{inp} \cdot k_n + \sum_{k \in K^{inc}} I_d^k \cdot k_n \cdot k_w \cdot k_h \right) \tag{2.11}$$

$$\Leftrightarrow \sum_{k \in K^{inc}} I_d^{inp} \cdot k_n \cdot k_w \cdot k_h > \sum_{k \in K^{red}} I_d^{inp} \cdot k_n + \sum_{k \in K^{inc}} I_d^k \cdot k_n \cdot k_w \cdot k_h \tag{2.12}$$

$$\Leftrightarrow \sum_{k \in K^{inc}} k_n \cdot k_w \cdot k_h > \sum_{k \in K^{red}} k_n + \sum_{k \in K^{inc}} \frac{I_d^k}{I_d^{inp}} \cdot k_n \cdot k_w \cdot k_h \tag{2.13}$$

$$\Leftrightarrow \sum_{k \in K^{inc}} k_n \cdot k_w \cdot k_h - \sum_{k \in K^{inc}} \frac{I_d^k}{I_d^{inp}} \cdot k_n \cdot k_w \cdot k_h > \sum_{k \in K^{red}} k_n \tag{2.14}$$

For every $k \in K^{inc}$, k_n is the depth of the input to the inception module if k is a 1×1 convolution ($I_d^k = I_d^{inp}$). Else, k_n is the depth of the output from the dimensionality reduction layer. Thus the term can be simplified as follows, where $k \in K \setminus K^1$ is the set of all layers that are not 1×1 convolutions:

$$\Leftrightarrow \sum_{k \in K \setminus K^1} \left(1 - \frac{I_d^k}{I_d^{inp}} \right) \cdot k_n \cdot k_w \cdot k_h > \sum_{k \in K^{red}} k_n \tag{2.15}$$

The formulation in (2.15) thus gives an easy-to-use formula to control for the depth of the additional convolutional layers $k_n \forall k \in K^{red}$ to reduce the overall complexity.

The equation needs to hold for the reduced size convolutional neural net having less operations than the original one. It is tailored for the general case that for every convolutional layer that has a window greater than 1, $k_w, k_h \geq 2$, a convolutional 1×1 layer of depth n is inserted. It assumes that all dimensionality reductions are done with 1×1 convolutions as is the usual case. Nonetheless, the restriction of using 1×1 convolution is not mandatory. The general inception idea would also work with other window sizes as long as the depth is reduced, although larger window sizes lead to more computations than 1×1 convolutions.

2.3.8 Residual Blocks

Residual blocks, as introduced by He et al. (2015), lay the groundwork for the triumph of very deep neural networks. They enable a transition from a 22-layer neural network as used in GoogLeNet to a 152-layer network as used in ResNet-152 at ILSVRC-2015 [55]. For a more profound analysis of the advantages of residual network connections, see [29].

The key idea to residual blocks is the usage of residual connections that skip layers and pass on the values which are added to the output after the skipped layers. The intuition behind that is that instead of having the network learn the full transformation of the data, the skipped blocks now only need to learn the delta of how to adjust the previous output values. In the original paper, the authors skipped two layers with the residual connections, but this is a design choice, see figure 26.

In mathematical terms, instead of learning the output $H(x)$ of a given input x direct, the network learns $H(x)$ as a function $H(x) = F(x) + x$. Thus, the network only needs to fit the residual $F(x) = H(x) - x$, which is smaller and easier to learn than the whole representation of $H(x)$. This change facilitates the gradient flow between layers and thus enables learning in deep networks. A similar concept is used in LSTM networks [36] and other variations of recurrent neural networks [50] [108] [68] [103].

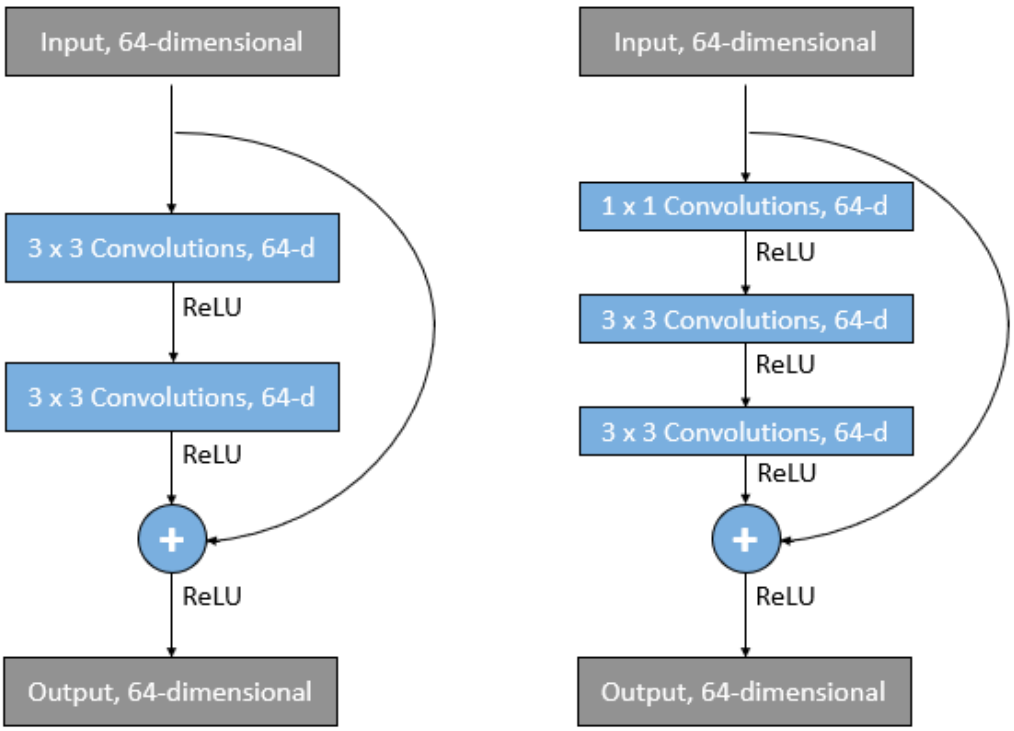


Figure 26: Two example residual block architectures, modified from He et al. (2015) [31]

Because the learned residual is added to the original output values, the data needs to be of the same shape.

The idea and the insights from residual blocks also led to the idea of densely connected convolutional networks, where the concept of residual connections is abstracted to every layer being densely connected to all other layers. For more on densely connected convolutional networks, see the original paper [41].

2.3.9 Capsules

Conventional convolutional neural networks have difficulties in generalising to novel viewpoints different from the data they have been trained on. This implies that convolutional neural network either lose their predictive power in unrestrained, real-world environments or they need to be trained on large amounts of data. This limitation adds exponential complexity to the training of neural networks to make them robust to these challenges [33]. Furthermore, Su et al. (2018) showed that the output of deep neural networks for image recognition can be easily altered through adversarial attacks, even by only adding small perturbations to the input image (e.g., one-pixel modification) that would not be recognised by humans [93]. Apart from that, convolutional neural networks cannot develop a notion of the relation between different concepts within an object. Although CNNs are quite good at detecting features, aggregating them and making a prediction based on their presence or absence, they are not able to infer whether these features have the correct spatial relationship. For instance, a CNN will identify a face in an image, even if the mouth, the eyes and the nose are misplaced, see figure 27.

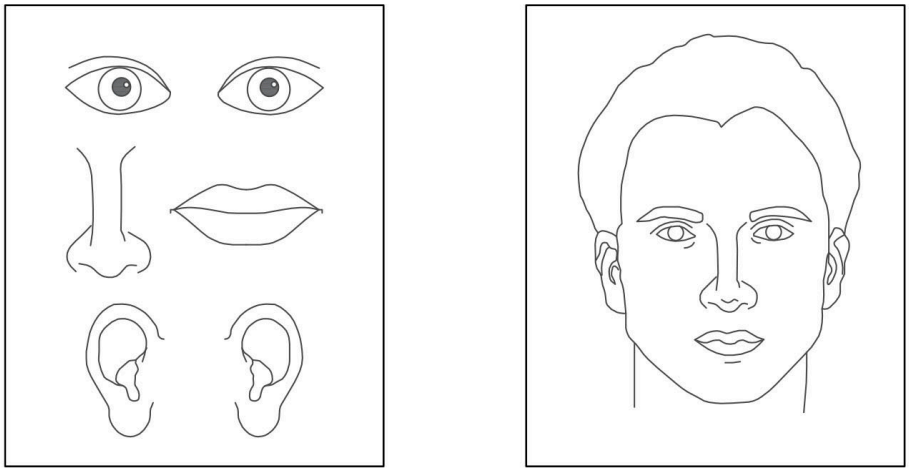


Figure 27: A convolutional neural network does not effectively capture the spatial relationship among concepts but simply detects the existance of certain elements

Based on the idea of fixation points that help the human vision identify shapes and objects, Sabour et al. (2017) propose a new network architecture that enables a different representation of the underlying object features: Capsules [79]. Although their proposed model change still uses the fundamental concepts of convolutional neural networks, convolutional layers, it is usually abstracted and referred to as capsule networks, defining a new category of networks for computer vision. Although capsule networks are still at its beginning, they are estimated to have a significant impact on the future of neural networks, especially for computer vision, because "there are fundamental representational reasons for believing it is a better approach" ([79], p.9).

The idea of capsule networks is similar to that of factor analysis - an image is built hierarchically from existing features with each feature having an associated probability (weight) determining its importance to the underlying concept. To exemplify this, a face could be built up by the latent features "eye", "nose" and "beard". While the features "eye" and "nose" are expected to have a strong contribution towards the concept of a face, especially when they are arranged in a certain relationship to each other, a beard or the absence of such is not always such a strong indicator of a face.

The basis for that is that an entity (e.g., a feature) in an image is represented by a capsule. A capsule can be seen as a layer within a layer because a capsule is a vector of neurons of length n. Each capsule can be seen as a part of the description of the picture, encoding specific properties, as each neuron within a capsule encodes a feature of the entity, e.g., its width, pose, thickness, scale or locally specific properties. The probability that a given entity is encoded by a capsule is represented by the length of the capsule vector. The longer the vector, the more likely the presence of the encoded entity. The creation of capsule networks can be split into two sub-steps

1. **Creation of Primary Capsules:** Capsule networks start with one or more convolutional layers that create a tensor consisting of feature maps as described in section 2.3.1. These feature maps can then be reshaped to get multiple vectors - the capsules. The reshaping is arbitrary, as long as all the information is retained. For instance, if the output depth n of the convolutional layer is 16, this could be reshaped to 2 vectors of length 8 or 4 vectors of length 4, etc. Furthermore, since the vectors' length encode the likelihood of a given feature, the vectors are squashed to be of a length not bigger than 1. These created capsules are called the primary capsules. This process is exemplified in figure 28, where two primary capsules of length 4 are created at each input position from a convolutional layer of depth 8.

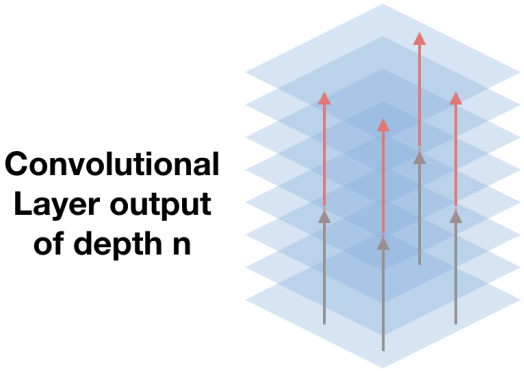


Figure 28: Exemplified vector creation from convolutional layers

2. **Dynamic Routing to Higher-Level Capsules:** Dynamic Routing between capsules determines the parse tree - which lower-level capsules are related to which higher-level capsules. In other words, routing determines which concepts are related to each other within an image. To do so, every capsule u_i of a given layer tries to predict the output of all the capsules from the next layer. This procedure is applied in every layer from input to output.

   For every capsule u_i in layer C_i and every capsule u_j in layer C_{i+1}, the prediction is made as $\hat{u}_{j,i} = W_{i,j} \cdot u_i$, where W is a weight matrix that is adapted to the training data. If there exists strong agreement among the predicted capsules $\hat{u}_{j,i}$ from multiple capsules, these capsules weights will be strong. Hence these lower level capsules' outputs contribute strongly to the higher-level capsules, whereas the output of the capsules that made different predictions only weakly influence that particular higher-level capsule. This process is called routing by agreement. Agreement is achieved through a clustering-like approach, where the contribution of each prediction vector $\hat{u}_{j,i}$ is determined by the similarity of that prediction vector to the (weighted) mean vector of all predictions. The distance metric used by Sabour et al. (2017) for similarity is the scalar product of the vectors, although other metrics could be used. The detailed procedure for routing between two layers l and $l+1$ from Sabour et al. (2017) can be found in Algorithm 1.

   Another positive side effect of this routing process is that it can deal with ambiguity within a picture. As Sabour et al. (2017) show with the example of the overlaying MNIST example, a capsule network makes predictions that are mutually exclusive and collectively exhaustive (MECE). Thus, it chooses the classification that is most likely taking into consideration all other objects and their classification.

Algorithm 1 Routing Algorithm from [79]

procedure ROUTING($\hat{u}_{j,i}, r, l$)
  for every capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$
  for r `iterations` **do**
    for `all capsule` i `in layer` l **do**
      $c_i \leftarrow softmax(b_i)$
    end for
    for `all capsule` j `in layer` $(l+1)$ **do**
      $s_j \leftarrow \sum_i c_{ij} \hat{u}_{j|i}$
    end for
    for `all capsule` j `in layer` $(l+1)$ **do**
      $v_j \leftarrow squash(s_j)$ ▷ $squash(s_j) = \frac{\|s_j\|^2}{1+\|s_j\|^2} \frac{s_j}{\|s_j\|^2}$
    end for
    for `all capsule i in layer l and capsule j in layer` $(l+1)$ **do**
      $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} v_j$
    end for
  end for
  return v_j
end procedure

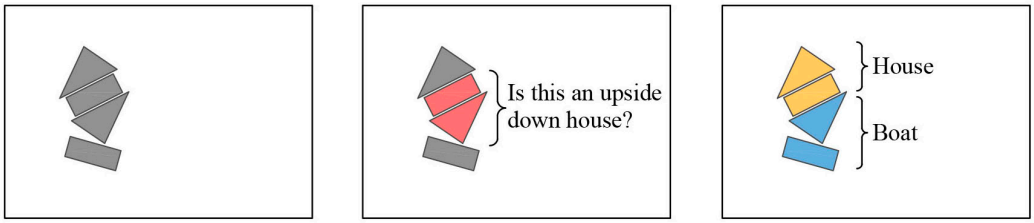


Figure 29: Example of a MECE solution in a multi-label classification task from [10]

The last layer of a capsule network consists of a layer of capsules, where each capsule represents one target class and encodes the probability of the given class represented by its length.

Sabour et al. (2017) train the capsule network with ℓ_2 loss between the length of the capsules from the last layer and the actual objects in a picture. Despite that, a different loss function could be chosen. A different loss function called margin loss is used to detect multiple objects within an image, such as for the overlaying MNIST case. Further loss adaption can be done by adding a regularisation parameter that is determined by a decoder network. This decoder network attempts to recreate the original image from the last layer capsule representation by using fully-connected

layers. The difference between the decoded picture and the actual picture can then be used as a regularisation parameter for the loss function. The output from the decoder is usually scaled down by a parameter α (e.g., 0.00005) to reduce its influence. For more on the loss functions proposed, see the original paper [79].

2.3.10 Transfer Learning

The idea of **transfer of practice**, introduced by Edward Thorndike and Robert Woodworth (1901) (see also [76], [77] and [78]), is an idea from the field of psychology that applied knowledge can be transferred from one context to another context. The more related the contexts are, the easier the transfer learning task.

Analogously in machine learning, transfer learning is the idea of using pre-trained artificial neural network models and retraining them on different but related datasets. The general notion is that training speed can be increased because many features will have already been learned and the neural network only needs to adapt to the new task. Its fundamentals rely on the psychological concept of transfer of practice [19].

The research area of transfer learning in computer science evolved in the late 19^{th} century with the works of Lorien Pratt (1995) [69] and others. Traditionally, models have been designed and tuned to fit specific purposes without direct applicability to other use cases. Nonetheless, transfer learning became widely applied in neural network training, especially after the publication of competition-winning models and their pre-trained weights, like Inception, ResNet and others that could be retrained on other image recognition tasks. Through transfer learning, these large neural networks can be fine-tuned with less computational means, since fewer iterations are required. Some transfer learning tasks are even possible on commodity hardware.

Transfer learning can be formally defined as: "Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T , transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$." ([66], p. 3)

The superiority of transfer learning over traditional learning, e.g., random initialisations, are [52]

- Initial performance on target task without training
- Time to have the model learn a representation of a problem
- Earlier convergence towards final model performance

Nonetheless, learning tasks T_S and T_T need to be related to produce a positive learning impact, else there is a risk of inverting the effect and producing a negative learning impact [52].

In computer vision, it was observed that convolutional neural networks learn similar patterns in their first layers independently of the problem at hand. The first layer filters of neural networks usually resemble Gabor filters or colour blobs [107]. This implies that these models develop a general notion of differentiating between the classes through the creation of these image patterns independently from the actual pictures. These first layer filters are thus considered *general*. This observation leads to the conclusion that a CNN does not need to be separately trained on creating these filters in the convolutional layers but could already be initialised with these. The closer to the output, the layers lose generality and become more specific [107].

Modern ANN packages, like Keras, offer various model architectures with pre-trained weights. This simplifies the programmer's work of loading models into the right format for different ANN packages. The concept of multi-task learning, creating a single ML algorithm for multiple tasks, is different to transfer learning, although the ideas share some common concepts.

Transfer learning can be applied in many different forms, e.g., inductive transfer, bayesian transfer, hierarchical transfer or others (for more, see [52]). Transfer learning can be particularly powerful if the target domain has few training examples and thus would be difficult to train from scratch. [42]

Chapter 3

Classification in Supervised Environments

"There is no single classifier superior over the rest" ([87], p. 74). As the no-free-lunch theorem proves, the superiority of an algorithm depends on the context of its usage. While a specific algorithm may work well for a given problem, it is not guaranteed to work well on a different problem. Therefore, it is the machine learning practitioner's job to try out different algorithms, architectures, and hyper-parameters before choosing the most appropriate algorithm for any given problem.

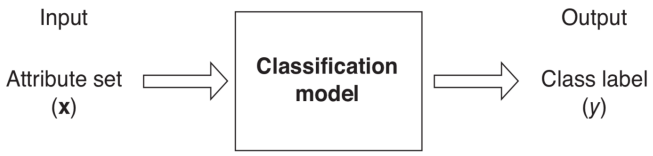


Figure 31: General Classification Procedure from [98]: p. 146

Classification is the "task of learning a target function $\hat{F}(x)$ that maps each attribute set x to one of the predefined class labels y"([98], p. 146). $\hat{F}(x)$ is an approximation of the actual underlying target function $F(x)$. A classification model can represent the target function. The input data consists of records of tuples of the shape (x_i, y_i). Each set of n attributes is denoted as x_i, where $x_i \in X \subset \mathbb{R}^n$. The target attribute (class label) is denoted as $y_i \in Y = \{-1, +1\}^l$

$$\{(x_1, y_1), \ldots, (x_l, y_l)\} \subseteq X \times Y, i = 1, 2, \cdots, l \tag{3.1}$$

A learning algorithm is applied to learn the mapping function $\hat{F}(x)$ from the input samples X to the target y and generalise the mapping function to unseen examples,

minimising the expected value of a predefined loss function $L(F(x), \hat{F}(x))$ or $L(y, \hat{F}(x))$. The applied loss function is a hyper-parameter of the machine learning model.

In the following chapter, several machine learning algorithms are introduced. The chapter starts by introducing Decision Trees in section 3.1. After that, several ensemble model techniques are presented in section 3.2. Section 3.3 introduces the logistic regression algorithm, followed by introductions of support vector machines in section 3.4, K-nearest neighbours in section 3.5 and multi-layer perceptron in section 3.6. The final section 3.7outlines evaluation methods to compare the performance of multiple machine learning models.

Although all of them can be used for real-valued targets as well, this chapter illustrates their application on categorical predictions.

3.1 Decision Trees

Decision Trees (DTs) are one of the most popular data mining techniques. Decision Trees are computational trees that can classify information through an application of a chain of rules with the principle of divide and conquer. A DT consists of nodes that are connected through edges. Every internal node makes a decision that determines which of the connected nodes to pass the information flow to based on an attribute of the input data. By doing so, a series of rules can be applied until a leaf node is reached. A leaf node is a node that does not have any outgoing edges into other nodes and is thus an endpoint of the graph. A leaf node represents a state or decision of the Decision Tree, e.g., a categorical prediction. Oftentimes, Decision Trees are binary trees that only have two outgoing edges, but there is no theoretical limitation to the number of outgoing edges at each node. The tree structure is determined in a learning process, where the tree is presented with training data to learn on, and its internal structure is built. Various algorithms have been developed, with Ross Quinlan laying the groundwork with the Iterative Dichotomiser 3 (ID3) algorithm and later the C4.5 algorithm, that are both based on Hunt's algorithm. Another learning strategy based on Hunt's algorithm is Classification and Regression Trees (CART), introduced by Breiman (1984). Figure 32 shows how an exemplary Decision Tree is built in four steps by subsequently adding splitting points (decision nodes) to the tree. Algorithm 3 describes the steps to create a Decision Tree following the ID3 algorithm. This algorithm is recursively applied until some stopping criterion is reached. This can be related to the size of the tree, or in its simplest form, until all training examples can be correctly classified.

Hunt's algorithm is a top-down, greedy algorithm because it makes a decision at every node that is based on the highest information gain. Information Gain is a metric that

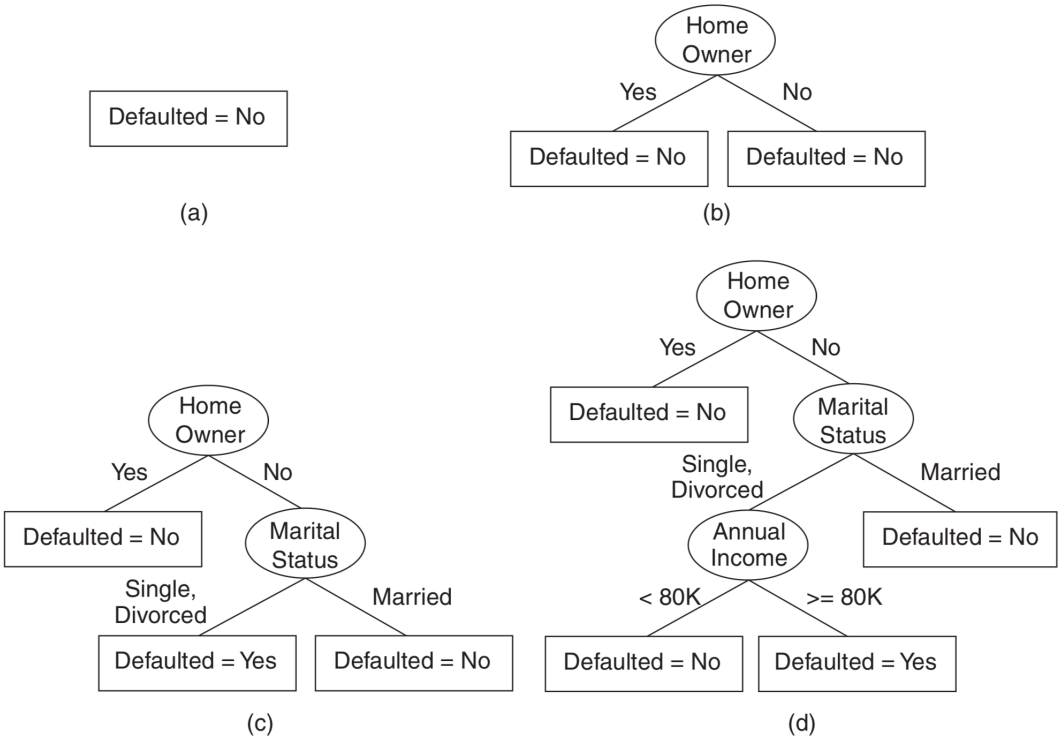


Figure 32: Illustration of a Decision Tree creation process using Hunt's algorithm from [98], p. 154

measures how much information has been gained through a given choice at a given node. It is defined as the influence of the knowledge of a feature at a given node. The influence is determined through a measure of disorder, such as Gini or Entropy. These measures are higher the more unordered the data is.

Algorithm 2 describes the steps to select the attribute A with the highest information gain ratio. The introduced attribute selection Algorithm 2 will illustratively look at the more common case of using Entropy, but the usage of Gini[1] is analogous. Firstly, the expected information needed $Info(D)$ to classify a tuple in D optimally is calculated using the probability p_c that a tuple in D belongs to class c. The probability p_c is calculated as $\frac{|D_c|}{|D|}$, where D_c is the set of entries of a given class c. In the second step, the expected information required to classify a tuple from D based partitioning by attribute A is calculated in $Info_A(D)$, where $D_a \forall a \in values(A)$ are the entries in D where the attribute A has a specific value a. Afterwards, the information gain $Gain(A)$ on splitting at attribute A can be determined as a difference of the previous two steps. Information Gain can already be a good measure to determine the best split. But information gain only seeks purity among the features. Nonetheless, if a given feature has very many entries, the probability of this feature being a good predictor for the

[1] $Gini(D) = 1 - \sum_{i=1}^{m} p_1^2$

final class is higher due to its nature of holding many possible splits. This is prone to overfitting. To avoid this problem, the fourth and fifth steps introduce a gain ratio measure. This problem only occurs for non-binary Decision Trees with a flexible number of outgoing edges. For binary Decision Trees, information gain is sufficient to determine the best split attribute.

Algorithm 2 Splitting Attribute Selection

procedure SPLIT_ATTRIBUTE_SELECTION(D)
  $Info(D) = Entropy(D) = \sum_{c \in C} p_c \log_2(p_c) = \sum_{c \in C} \frac{|D_c|}{|D|} \log_2\left(\frac{|D_c|}{|D|}\right)$
  for each attribute A **do**
    $Info_A(D) = \sum_{a \in values(A)} \frac{|D_a|}{|D|} \cdot Info(D_a)$
    $Gain(A) = Info(D) - Info_A(D)$
    $SplitInfo_A(D) = -\sum_{a \in values(A)} \frac{|D_a|}{|D|} \cdot \log(\frac{|D_a|}{|D|})$
    $GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$
  end for
  return Attribute A with highest $GainRatio(A)$ or highest $Gain(A)$
end procedure

Algorithm 3 ID3 algorithm from [61], p. 18

procedure ITERATIVE DICHOTOMISER 3(D, A)
  Create a new tree T with a single root node
  if no more split(s) OR other stopping criterion reached **then**
    Mark T as a leaf with the most common value of c a label
  else
    Find $a \in A$ maximising split_attribute_selection(D)
    Label T with f
    for each value v_j of a **do**
      Set $Subtree_j = ID3(D_{a=v_j}, A - \{a\})$
      Connect node T to $Subtree_j$ with edge labeled v_j
    end for
  end if
  return Decision Tree T
end procedure

A significant advantage of Decision Trees is that they have understandable knowledge structures that are easily traceable. Furthermore, the computational cost at runtime is not very high. The construction process is also relatively fast in comparison to other classification methods [84]. Decision Trees can handle both categorical and real values. On the other hand, Decision Trees are unstable on small training sets. This caveat can be solved with ensemble methods (see section 3.2). Nonetheless, Hunt's algorithm is a greedy algorithm and thus likely to get stuck in local optima[87]. Finding a globally optimal Decision Tree is currently computationally infeasible for most problems because there are oftentimes very many combinations between the attributes. Although DTs are a widely used machine learning method, other machine learning models, e.g., kernel methods and ensemble techniques oftentimes outperform DTs in practice [59].

To handle the complexity and avoid overfitting, several strategies have been developed:

- Pre-pruning (Early Stopping Rule): A measure is put in place that indicates overfitting during training and therefore stops the training, e.g., "gain in impurity measure or improvement in the estimated generalisation error" ([98], p. 184).
- Post-pruning: The initial tree is grown to its maximum size (until all training utterances are classified correctly) and the fully-grown tree is then trimmed in a bottom-up fashion by
    1. Replacing a subtree with a leaf node determined from the majority class of the records
    2. Replacing a subtree with the most frequently used branch of the subtree

Post-pruning usually leads to better results than pre-pruning, because the decisions are based on the full tree. This comes at the cost of computational efficiency.

One particularly popular variant of Decision Trees in the context of ensemble learning from section 3.2 is Decision Tree stumps. Decision Tree stumps are Decision Trees with only one decision node. They are rather simple trees that draw linear decision boundaries based on one feature. A dataset with n observations in d dimensions can potentially hold $n \cdot d$ Decision Tree stumps, although only the pareto-optimal ones are usually relevant.

3.2 Ensemble Methods

Ensemble methods combine various models and generate an output prediction from a combination of the models' outputs. Usually, an ensemble model combines multiple weak learners. Various ways of combining the individual models' outputs exist. One can generally differentiate between Bagging and Boosting.

Bagging

In bagging, multiple classifiers are created in parallel and combined into the final output. Usually, each classifier is built on a bootstrapped version of the original dataset. Bootstrapping in this context means that the features (columns) are randomly subsampled. Thus, each classifier does not learn on all features but only on a subsample of all features. For bagging, one common method is majority vote: Every model makes a prediction, and the most often predicted class is chosen. Another conventional approach is averaging the likelihood score over all predictions and selecting the class label with the highest likelihood:

$$\hat{F}'(x)_i = \phi(x_i) = \frac{\sum_{k=1}^{K} f_k(x_i)}{K}, f_k \in \hat{F}, \tag{3.2}$$

where $\hat{F}$ is the set of all Decision Trees in the ensemble.

Since the number of estimators k is a fixed number, the division is not necessary, and the formula can be changed to be the sum of the predictions. Thus, it would be an additive maximum vote that behaves equivalently to the average voter.

Boosting

For boosting, multiple classifiers are created sequentially. Each sequentially created classifier is generated depending on the result of the prior classifier. The classifiers are not trained on a randomly bootstrapped dataset but a weighted dataset depending on the previous model prediction. Boosting is usually done with Decision Trees, especially Decision Tree stumps, as "weak learner" or "base learner". Nonetheless, it can generally be done with any classification algorithm that outperforms a random guess. Schwenk & Bengio (2000) demonstrate this on the example of boosting multiple neural networks using AdaBoost, see section 3.2.2 [81].

The general idea of boosting is to reduce the classifier's error subsequently by iteratively adding models. In doing so, the algorithm focuses on the parts of the training data that have been misclassified before by giving more importance to those. In analogy, this can be seen as a voting process of weighted experts in certain areas. Multiple expertise classifiers' predictions are combined and weighted to produce the final prediction.

The algorithm works by fitting new models to the residuals (errors) of the previous models, assuming that this is the part that the algorithm needs to focus on to improve the overall prediction. This focus is done by introducing a weight $w_t(x_i)$ that determines the impact of the entry x_i at time step t on the classification algorithm. Correctly classified instances get fewer weights in the next generation, while misclassified examples get higher weights in the next training iteration. This enables the algorithm to focus its efforts on the specifics of the dataset that are hard to learn. This sequential model fitting on the residual per weak learner is done until a predefined stopping criterion[2] is reached.

The overall idea of boosting can be simplified as shown in Algorithm 4, where ϵ describes the remainder (error) that was not captured by the weak classifier $\hat{F}_m$.

[2] e.g., number of iterations, remaining error, model size, model generalisation ability, model score

Algorithm 4 General Boosting Procedure

```
1: procedure BOOST_LEARNERS(X, y)
2:     Approximate y with weak learner F̂(X)
3:     ε ← L(y, F̂(X))
4:     if Stopping criterion not reached then
5:         Boost importance of misclassified samples in X based on ε
6:         F̂(X) ← F̂(X)+boost_learners(X, y)
7:     end if
8:     return F̂(x)
9: end procedure
```

The combination of multiple models' predictions can become complex, especially for larger models. This constraint can be resolved by combining the ensemble model's classifiers in a single classifier [35].

In the following subsections, common tree learning ensemble methods are introduced.

3.2.1 Random Forest Classifier

Developed by Leo Breiman (2001) [9], Random Forests Estimators (RFE) are a "group of un-pruned classification or regression trees made from the random selection of samples of the training data" ([4], p. 274). Random Forests are one of the mostly used Bagging techniques in ensemble learning. Multiple base regression trees $r_n(x, \theta_m, W_t), m \geq 1$ are combined to form an aggregated estimate. The symbol θ denotes an i.i.d. randomising variable ([9], p.2; [6]).

During induction, the features are selected at random. The final prediction is made through a majority vote (prediction through aggregation). The process for a random forest classifier is as follows:

1. Random Column Sub-sampling: Sample n items randomly from all columns N (with replacement).

2. Select $m \ll M$ features at each node and choose the best split via information gain, e.g., using Gini or Entropy.

3. Grow each tree as large as possible without pruning it.

The aggregated regression estimate is then: $\bar{r}_n(X, W_t) = E_\theta[r_n(X, \theta, W_t)]$ where E_θ denotes the expectation with respect to the random parameter θ, conditional on the input data X and the data weights W_t ([6], p. 1064).

This procedure makes Random Forests robust to noise in the data and enables it to generalise better. Oftentimes, Random Forests outperform DTs in practice, especially for large training datasets. They are also often used for datasets with many input variables since each tree only selects a sample of input variables to make a prediction on. They often perform comparably to AdaBoost [9] from section 3.2.2, but the process can be less complex than training boosting approaches. Contrary to the intuition, adding trees to a random forest model does not lead to overfitting, since the result is a majority vote between all trees. On the other hand, adding trees oftentimes improves generalisation because it leads to a more democratic decision process that is less reliant on potentially overfitted trees.

Random Forests also enable weighting of feature importance.

3.2.2 Adaptive Boost Classifier

Adaptive Boosting (AdaBoost, later also referred to as ADB) is a tree boosting algorithm developed by Yoav Freund and Robert Shapire (1995). Analogously to random forests, the idea is to create a strong classifier from a combination of weak classifiers. Multiple weak classifiers are used as features for a linear regression model that then makes the final classification:

$$\hat{F}(x_i) = \alpha_1 h_1(x_i) + \alpha_2 h_2(x_i) + \cdots + \alpha_l h_l(x_i) \tag{3.3}$$

where $l \in \mathbb{N}$ denotes the number of weak learners and α_k is equivalent to the weights of each weak classifier. The weights α_k are learned in cycles over the data. The exact AdaBoost algorithm can be found in Algorithm 5.

Adaptive Boosting can also be used for feature selection, see [101], p. 802 ff.

3.2.3 Gradient Boosting Classifier

Gradient Boosting (GBC), introduced by Friedman (1999) [20], is an enhancement of the AdaBoost algorithm that allows for more general loss functions and presents a new way of handling the residuals ϵ.

From a mathematical perspective, each weak learner indexed $t = 1, 2, 3, \ldots, T$ makes a prediction $F_t(x)$ of t on the input data x as described in equation 3.5 with the weak learner a_t from equation 3.4.

$$(\beta_t, a_t) = \underset{\beta, a}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \hat{F}_{t-1}(x_i) + \beta h(x_i; a)) \tag{3.4}$$

$$\hat{F}_t(x) = \hat{F}_{t-1}(x) + p_t h(x; a_t) \tag{3.5}$$

Algorithm 5 AdaBoost algorithm from [101], p. 801 f.

procedure ADAPTIVE BOOSTING(X, Y, T)
  Initialise $W \leftarrow \frac{1}{\|X\|}$
  Initialise $\hat{F}(x)$ with an initial weak learner
  for $t \in T$ **do**
    Train weak learner using distribution $W(t)$
    Get weak hypothesis $h_t(x) : X \rightarrow \{-1, +1\}$ with error $\epsilon_t = L(h_t(x), y)$
    $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    for $i \in N$ **do**
      if $h_t(x_i) == y_i$ **then**
        $W_{t+1}(x_i) \leftarrow \frac{W_t(x_i)}{Z_t} \cdot e^{-\alpha_t}$ ▷ Z_t is a norm. factor, see formula 3.11
      else
        $W_{t+1}(x_i) \leftarrow \frac{W_t(x_i)}{Z_t} \cdot e^{\alpha_t}$ ▷ Z_t is a norm. factor, see formula 3.11
      end if
    end for
    $\hat{F}(x) \leftarrow \hat{F}(x) + \alpha_t h_t(x)$
  end for
  return $\hat{F}(x)$
end procedure

The loss function $L(y, \hat{F}(x))$ is usually chosen to be either exponential or the negative binomial log-likelihood, although there is no theoretical limitation. The target $y \in \{-1, 1\}$ is binary, but the algorithm can be enhanced to multi-class problems (see also [20]). The weak learner $h(x; a_t)$ can be of any classification algorithm. Equation 3.6 shows the overall formula for gradient boosting, where T weak learners $h(x; a_t)$ are combined with individual coefficients p_t.

$$\hat{F}(x) = \sum_{t=1}^{T} p_t h(x; a_t) + \epsilon \tag{3.6}$$

The formula 3.6 cannot be optimised with traditional optimisation methods in the Euclidean space. Therefore, an approximation with respect to minimising the loss has to be made. The steepest-descent step for decreasing the loss function is the negative gradient with respect to $\hat{F}(x_i)$, as in 3.7.

$$-g_t(x_i) = -\left[\frac{\delta L(y_i, \hat{F}(x_i))}{\delta \hat{F}(x_i)}\right] \tag{3.7}$$

The weak learner most correlated with $-g_t(x_i)$ is chosen to get a good, generalisable estimate. It is the most suitable approximation as it is the weak learner that is most parallel to the steepest-descent function. Equation 3.8 is used to find most suitable

weak classifier.

$$a_t = \underset{a,\beta}{\texttt{argmin}} \sum_{i=1}^{N} \Big[g_t(x_i) + \beta h(x_i; a) \Big]^2 \tag{3.8}$$

The chosen steepest descent approximation is $h(x; a_t)$ instead of the unconstraint negative gradient $-g_t(x)$ for minimising the loss function.

Thus, the final model approximation can be found in equation 3.9 with the coefficient p_t from equation 3.10.

$$\hat{F}_t(x) = \hat{F}_{t-1}(x) + p_t h(x; a_t) \tag{3.9}$$

$$p_t = \underset{p}{\texttt{argmin}} \sum_{i=1}^{N} L(y_i, \hat{F}_{t-1}(x_i) + p h(x_i; a_t)) \tag{3.10}$$

To summarise this procedure, the original function minimisation problem from formula 3.6 is approximated by a least-squares function minimisation as in formula 3.8. Instead of least-squares, another fitting criterion can be applied.

Algorithm 6 reproduces the algorithmic steps to perform gradient boosting, for the common case of an exponential loss function.

Algorithm 6 Gradient Boosting algorithm from [20], p. 5

procedure GRADIENT BOOSTING(x, y, T)
  Initialise $W \leftarrow \frac{1}{\|X\|}$
  Initialise the classifier with a weak classifier $\hat{F}_0 = \texttt{argmin}_f \sum_{i=1}^{N} L(y_i, f(x))$
  $\epsilon_0 = L(y, \hat{F}_0)$
  for $t = 1$ to T **do**
    for $i = 1$ to N **do**
      $\tilde{y}_i = -\left[\frac{\delta L(y_i, \hat{F}_{t-1}(x_i))}{\delta \hat{F}_{t-1}(x_i)} \right]$
    end for
    Get candidate classifiers $H(W, X, Y)$ from input X, target Y and weights W
    $a_t = \texttt{argmin}_{a,\beta} \sum_{i=1}^{N} [\tilde{y} i - \beta h(x_i, a)]^2$ for all $h(x_i, a) \in H$
    $p_t = \texttt{argmin}_p \sum_{i=1}^{N} L(y_i, \hat{F}_{t-1}(x_i) + p h(x_i, a_t))$
    for $i = 1$ to N **do**
      Get the new weights $W_{t+1}(x_i) = \frac{W_t(x_i)}{Z_t} e^{-p_t h(x_i, a_t) y(x_i)}$ ▷ Z_t from 3.11
    end for
    $\hat{F}_t(x) = \hat{F}_{t-1}(x) + p_t h(x, a_t)$
  end for
  return $\hat{F}(x)$
end procedure

The regularisation parameter Z at iteration t is merely the sum of all the weight values, as described in equation 3.11. In this formula, the set of correctly classified examples is denoted as $Y(X) \cap \hat{F}_t(X)$, whereas the misclassified samples are in the set $Y(X) \setminus \hat{F}_t(X)$.

$$
\begin{aligned}
Z_t &= \sum_{x \in Y(X) \cap \hat{F}_t(X)} w_t(x) \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} + \sum_{x \in Y(X) \setminus \hat{F}_t(X)} w_t(x)) \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} \\
&= \sum_{x \in X} w_t(x) \sqrt{\left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{Y(x) \cdot \hat{F}_t(x)}} = 2 \cdot \sqrt{\epsilon_t \cdot (1-\epsilon_t)}
\end{aligned}
\tag{3.11}
$$

For an exponential loss function, p_t is bound by $\frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$. Although the error itself does not necessarily constantly go down from iteration to iteration, it is constraint by an exponentially decaying function and thus eventually approaches 0 for $t \to \infty$.

Simplifying the formula for w_t from Algorithm 6, it follows equation 3.12 for the correct classifications and equation 3.13 for the incorrect ones.

$$
w_{t+1}(x) = \frac{w_t(x)}{Z_t} \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \frac{w_t(x)}{2 \cdot \sqrt{\epsilon_t \cdot (1-\epsilon_t)}} \cdot \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \frac{w_t(x)}{2} \cdot \frac{1}{\epsilon_t} \tag{3.12}
$$

$$
w_{t+1}(x) = \frac{w_t(x)}{Z_t} \cdot \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} = \frac{w_t(x)}{2 \cdot \sqrt{\epsilon_t \cdot (1-\epsilon_t)}} \cdot \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} = \frac{w_t(x)}{2} \cdot \frac{1}{1-\epsilon_t} \tag{3.13}
$$

Formulas 3.12 and 3.13 can be combined to 3.14, where $Y(x)$ is the correct value and $\hat{F}_t(x)$ is the estimation of the correct value of the target of x in iteration t. Since both $Y(x)$ and $\hat{F}_t(x) \in \{-1, +1\}$, it follows for their product: $Y(x) \cdot \hat{F}_t(x) \in \{-1, +1\}$.

$$
w_{t+1}(x) = \frac{w_t(x)}{2} \cdot \frac{1}{\frac{Y(x)\hat{F}_t(x)-1}{-2} + Y(x)\hat{F}_t(x) \cdot \epsilon_t} = \frac{w_t(x)}{-Y(x)\hat{F}_t(x)(1-\epsilon_t) + 1} \tag{3.14}
$$

The gradient boosting algorithm can also be applied to multi-class problems. For more on this, see [20].

Gradient Tree Boosting is a very applicable technique that provides promising results in practice. "Tree boosting has been shown to give state-of-the-art results on many standard classification benchmarks" ([11], p. 1).

In practical observations, Gradient Boosting does not seem to overfit, although this has not been theoretically explained.

3.2.4 eXtreme Gradient Boosting Classifier

Gradient Boosting performed on Decision Trees only is called Gradient Boosted Trees [59].

eXtreme Gradient Boosting, XGBoost (later also abbreviated with XGB), introduced by Chen & Guestrin (2016), is a specialisation of Gradient Boosted Trees that introduces new concepts to enable faster convergence, fewer parameters and more robustness [11]. Due to these advantages, its impact on machine learning competitions is striking: Of the 29 published Kaggle winning teams in 2015, about 59 % incorporated XGBoost into their solution. This number is higher than the one of neural networks (38 %). Other competitions showed similar figures [11]. Apart from its theoretical advantages, the XGBoost package is particularly successful because of its speed and executability on commodity devices. It is a scalable machine learning system for tree boosting, available as an open source software package.

Generally spoken, XGBoost enhances Gradient Boosting with four components:

1. Approximate branch splitting
2. Regularisation
3. Shrinkage
4. Column Subsampling

Firstly, an approximate algorithm for finding the optimal split point for each tree in the Gradient Boosted Trees makes the algorithm faster[3], run better in parallel and run on larger datasets without RAM limitations. In the greedy, traditional approach, the entire search space of potential splitting points is searched and the best splitting point chosen. This is especially computationally expensive for continuous input values. Therefore, a more efficient, distributed algorithm to approximate a global optimum is developed. This approach creates a set of candidate splitting points according to a specified criterion (see chapter 3.3 in [11]) at percentiles of the feature distribution. Continuous features are then binned into buckets that are split by the set of candidate splitting points. Lastly, the best splitting point is chosen through aggregated statistics.

Secondly, XGBoost enhances the loss function of a regular Gradient Boosting model with a regularisation parameter that penalises model complexity to incentivise simpler, less sophisticated models. In a tree-based model, the goal of the classifier is to find the best splitting criterion for the right tree and the left tree of a given node. In detail, this is about which attribute and value to define as a splitting criterion. Chen et al. introduce a function to use for evaluating every possible split that takes a regularisation of the model's complexity into consideration [11]. Apart from the regularisation proposed in their paper, different regularisation techniques could be used.

[3] assuming a sufficiently large dataset that applying the statistical calculations takes up less time than exhaustively searching all possibilities

Third of all, the Gradient Boosting algorithm is enhanced with a shrinkage parameter that reduces the influence of each added tree, similar to the concept of a learning rate in gradient descent optimisation. This shrinkage parameter scales the added weights by a learning rate n, thus reducing the influence of added trees and enabling future trees to contribute to the model output [21]. The idea behind that is to create a more robust final classifier that is less dependent on individual trees.

Lastly, column subsampling, as usually done in bagging, is applied. This enables a predictor at iteration t to focus on learning from certain features in a dataset.

3.3 Logistic Regression

Linear regression models predict a value as a linear combination of its attributes with some previously determined coefficients:

$$\hat{F}(x) = \sum_{a \in A} \beta_a \cdot x_a \tag{3.15}$$

where $\hat{y}(x)$ is an approximation of $y(x)$ and x_a for $a \in A$ is the attribute a of dataset entry x. The coefficient associated with a given attribute a, including a constant attribute of $x_a = 1$, is denoted by β_a. The coefficients β are learned through Ordinary Least Squares (OLS) estimation, which minimises the total error from the predictions subject to the training data. These linear regression models make predictions between $-\infty$ and $+\infty$. This is suitable for real-valued predictions, but for classifications, the goal is to make categorical predictions. Thus, the real output values of the predictor should be squashed between 0 and 1. A logistic regression (LR) model, introduced around 1970 [67], does that through the application of a non-linear transformation: logits [67]. The logistic regression model is stated as

$$P(x) = \hat{F}(x) = \frac{e^{\sum_{a \in A} \beta_a \cdot x_a}}{1 + e^{\sum_{a \in A} \beta_a \cdot x_a}} \tag{3.16}$$

which can be rewritten as follows, where the logit is the natural log of an odds ratio:

$$\sum_{a \in A} \beta_a \cdot x_a = \ln \frac{P(X)}{1 - P(x)} = \ln(odds) = logit(Y) \tag{3.17}$$

While it is possible to directly interpret the influence of β on the prediction in the linear case, in logistic regression models, β can only be interpreted as the direction between X and the logit of Y without any linear assumptions possible. This makes it harder to explain the feature importance using logistic regression. Due to the convex nature of the logistic cost function, logistic regressions have the advantage of reaching a global maximum with regards to the cost function.

As stated above, a logistic regression predicts a real value bounded between 0 and 1. Therefore, one logistic regression model can only make a binary classification prediction[4]. Nonetheless, through the application of multiple logistic regression models, the prediction can be enhanced to multiclass.

3.4 Support Vector Machines

Support Vector Machines (SVM) is a non-probabilistic binary classifier that separates a dataset into two classes. Its general idea was developed by Vladimir Vapnik in the early 1960s but only published in the 1990s after performing well on optical character recognition (OCR). An SVM establishes a so-called boundary line between the classes that distinguishes which class a given entry is associated with. The boundary between the classes is determined to have the largest possible distance to both classes while separating the classes and therefore creates the highest amount of disambiguity (see figure 33). The boundary between two classes does not have to be linear. A procedure called kernel-transformation is applied to make predictions on non-linear datasets. It transforms the data and maps it to a higher-dimensional space such that a linear classifier can divide the data. The optimisation space of SVMs is convex. Therefore, it inevitably converges towards a global optimum.

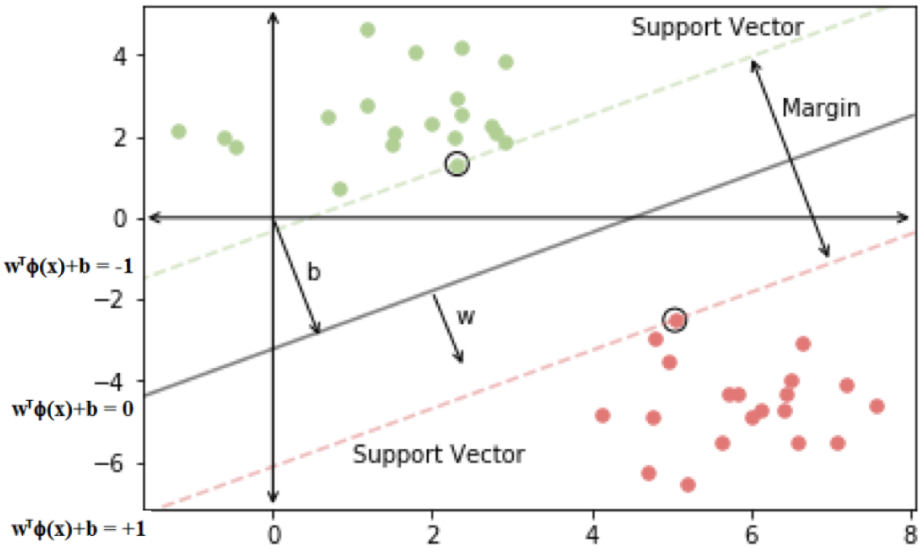


Figure 33: Explanation of SVM

Figure 33 visualises an SVM classifier of linearly separable data points. A vector $\vec{w}$ is defined to be perpendicular to the boundary line. Another vector $\vec{u}$ is randomly picked pointing in any direction. A decision rule can be established depending on $\vec{w}$ and any random vector $\vec{u}$ as $\vec{w} \times \vec{u} \geq c$, where c is a constant. This can be rewritten to

[4] for multiclass: see [17], ch. 5

formulate the decision rule as in equation 3.18. In this chapter, × denotes the cross product between vectors.

$$\vec{w} \times \vec{u} + b \geq 0, \; b = -c \tag{3.18}$$

The latter transformation is defined as the decision rule for whether an entry belongs to a given class or not. Many solutions fulfil the two conditions above. Therefore, the problem needs to be more constraint to solve it.

For the training process, it is defined thus that:

1. For all positive input samples x_+: $\vec{w} \times \vec{x_+} + b \geq 1$
2. For all negative input samples x_-: $\vec{w} \times \vec{x_-} + b \leq 1$

It follows that $y_i(\vec{x_i} \times \vec{w_i} + b) \geq 1$ and therefore $y_i(\vec{x_i} \times \vec{w_i} + b) - 1 \geq 0$, where $y_i \in \{-1, 1\}$. For the elements on the boundary line, it can then be said that

$$y_i(\vec{x_i} \times \vec{w_i} + b) - 1 = 0 \tag{3.19}$$

The street is referred to as the area between the positive and the negative boundary line. The width of the street is defined to be as large as possible while still separating all points. This goes for the simple case introduced in figure 33 but it can be enhanced to account for noisy datasets as visualised in figure 34. Using two points $\vec{x_+}$ and $\vec{x_-}$ that lie on the boundary line and the vector $\vec{w}$, the width of the boundary zone can be defined as:

$$width = (\vec{x_+} - \vec{x_-}) \times \frac{\vec{w}}{||\vec{w}||} \tag{3.20}$$

Combining 3.19 and 3.20 leads to

$$width = (1 - b - (-1 - b)) \cdot \frac{1}{||\vec{w}||} = \frac{2}{||\vec{w}||} \tag{3.21}$$

This expression needs to be maximised to make the boundary between the two classes as large as possible. Maximising this expression is the same as minimising $||\vec{w}||$, which is equal to minimising[5]

$$width = \frac{1}{2}||\vec{w}||^2 \tag{3.22}$$

Equation 3.22 is constraint by equation 3.19. Applying Lagrangian $\mathscr{L}$ with α being the Lagrange multiplier, we can derive

$$\mathscr{L} = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{l} \alpha_i \Big[y_i(\vec{w} \times \vec{x_i} + b) - 1 \Big] \tag{3.23}$$

[5]The factor $\frac{1}{2}$ is just a scalar and the squaring operation is strictly increasing for the case that $||\vec{w}|| \geq 0$ which it is by the definition of the length of a vector.

The derivative with respect to vector $\vec{w}$ is determined to find extrema of $\mathcal{L}$. The derivative can be found in equation 3.24.

$$\frac{\delta\mathcal{L}}{\delta\vec{w}} = \vec{w} - \sum_{i=1}^{l} \alpha_i \cdot y_i \cdot \vec{x_i} \stackrel{!}{=} 0 \tag{3.24}$$

The derivative with respect to b is defined in equation 3.25.

$$\frac{\delta\mathcal{L}}{\delta b} = -\sum_{i=1}^{l} \alpha_i \cdot y_i \stackrel{!}{=} 0 \tag{3.25}$$

From equation 3.24, one can derive the value of $\vec{w}$ that maximises $\mathcal{L}$

$$\vec{w} = \sum_{i=1}^{l} \alpha_i \cdot y_i \cdot \vec{x_i} \tag{3.26}$$

Formula 3.23 can then be resolved in its dual form as

$$\alpha^* = \underset{\alpha}{\operatorname{argmax}} \frac{1}{2} \cdot \sum_{j=1}^{l} \sum_{k=1}^{l} \alpha_j \cdot \alpha_k \cdot y_j \cdot y_k \cdot \vec{x_j} \times \vec{x_k} - \sum_{i=1}^{l} \alpha_i \tag{3.27}$$

From equation 3.18, an entry is classified positive if $\sum_{i=1}^{l} \alpha_i \cdot y_i \cdot \vec{x_i} \cdot \vec{u} + b \geq 0$

As indicated above, this formula works well for linearly separable examples. If a problem is not linearly separable it has to be transformed by a kernel $K\left\langle x_i, x_j \right\rangle$ to project it in a different space where the problem is linearly separable. The most common kernels are

- Linear kernel: $K\left\langle x_i, x_j \right\rangle = (\vec{x_i} \times \vec{x_j})^n$
- Radial Basis kernel: $K\left\langle x_i, x_j \right\rangle = e^{\frac{-\gamma\|x_i - x_j\|^2}{2\sigma^2}}$, where $\gamma > 0 \in \mathbb{R}$

Another common issue in real-world datasets is that data points are not perfectly separable, simply because they contain some random deviations (noise) that are not captured by any of the existing features, as indicated in figure 34. In the binary case, these are negative examples that fall into an area expected to be positive or vice versa. Transforming these examples with kernels to classify them would lead to overfitting the model. Therefore, the slack variable ξ and the parameter c are introduced to make the model robust to deviations.

These additions lead to a modified formula 3.22:

$$width = \frac{1}{2}\|\vec{w}\|^2 + c\sum_{i=1}^{l} \xi_i \tag{3.28}$$

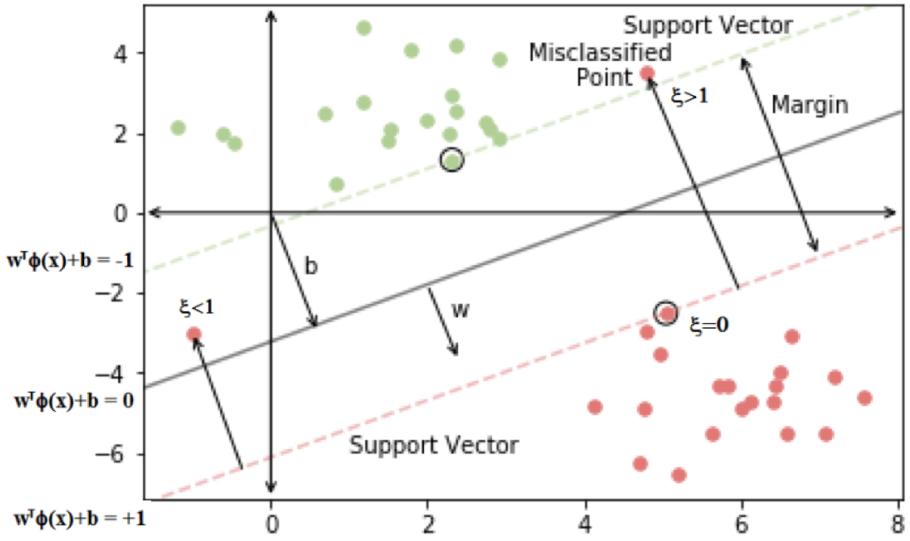


Figure 34: Explanation of SVM

subject to

$$y_i\left(K\left\langle \vec{w}, \vec{x_i} \right\rangle + b\right) \geq 1 - \xi_i \tag{3.29}$$

And then the problem from 3.27 enhances to

$$\alpha^* = \underset{\alpha}{\mathrm{argmax}} \frac{1}{2} \sum_{j=1}^{l} \sum_{k=1}^{l} \alpha_j \alpha_k y_j y_k K\left\langle x_j, x_k \right\rangle + \sum_{i=1}^{l} \alpha_i \tag{3.30}$$

The RBF kernel is also referred to as Gaussian kernel. A well-tuned RBF kernel cannot be worse than a linear kernel since the linear kernel is a degenerate version of an RBF kernel [47]. On the other hand, linear kernels are usually faster to train [18].

3.5 K-Nearest Neighbours

K-Nearest Neighbours (KNN) algorithm is an algorithm that is used in many areas of machine learning, such as classification and regression but also reinforcement learning for example.

A Voronoi tessellation divides the input data space into several regions depending on which is the nearest data point in that particular region. At the boundaries of these regions, at least two different training examples are equidistant. Therefore, the decision boundary between to classes follows the Voronoi cell borders between their points closest to the border and can be of any shape.

For input data $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{R}^d, y_i \in \{0, \ldots, c\}$, KNN algorithm makes use of Voronoi tessellations by assigning a new input x_{new} to the class that most of its k nearest neighbours belong to. KNN is a non-parametric algorithm, meaning it does not make an assumption on the input data distribution. Its classifiers have little constraints on the decision boundary and are therefore prone to overfitting. This effect can be regularised with the parameter k as this gives the model more robustness. Apart from k, another hyper-parameter of KNN is the distance metric to be used, e.g., Euclidean distance.

The KNN algorithm does not require an extensive training phase since the association to a given class is decided at runtime. Nonetheless, some computational optimisations can be done since only the boundary areas between classes are relevant. The size of the final KNN model depends on the separability of the dataset, depending on the size of the border of the Voronoi cells.

3.6 Multi-Layer Perceptrons

A Multi-Layer Perceptron (MLP) is the most common type of feed-forward artificial neural network that evolved from the idea of the perceptron. The perceptron was invented by Frank Rosenblatt. An MLP consists of multiple layers: an input layer, an output layer and at least one intermediate layer that are stacked on top of each other and control the information flow from input to output. In this thesis, a network is also referred to as MLP if it only consists of an input and an output layer without intermediate layers. Each layer consists of one or multiple nodes that are referred to as neurons in analogy to the human brain. In an MLP, each neuron is connected to all neurons of the previous layer and all neurons of the subsequent layer, therefore referred to as fully-connected. Each connection is unidirectional from the layer closer to the input towards the subsequent layer closer to the output. Each connection holds a weight value, indicating the strength of the connection between two nodes. Each node receives inputs from the nodes in the previous layer, depending on the strength of the weight, performs an activation on the sum of these inputs and distributes its output to its subsequent layer. This creates a directed information flow graph from the input layer through the intermediate layers to the output layer, where the final output is made. Each input to the network from the input layer corresponds to one feature in the dataset. Each output of the algorithm corresponds to one classification to be made. For the case of categorical predictions, the number of output classes can either be equal to the total number of classes to predict, thus each output neuron represents a class and outputs a likelihood value for that class. Alternatively, each neuron can make a binary prediction of {-1,1}, representing two classes. Thus n classes can be presented through $\log_2(n)$ (rounded up) output neurons as in a binary representation.

The learning procedure of an MLP uses the backpropagation algorithm to learn the weights between the nodes through training examples that are presented to it. Starting randomly, the model learns to reduce the error between its output predictions and the actual outputs by modifying its weights. The error is passed back from the neural network output to the input layer using the chain rule. Every weight connection then updates the respective weight for the subsequent iteration.

The output of an MLP can be continuous for real value predictions or categorical for classifications. A single MLP is also able to combine categorical predictions with continuous predictions, as in the case of predicting a class (categorical) and a position (continuous) of an object in an image (object localisation).

For more a mathematical description of MLPs and further deep learning concepts, see Higham et al. (2018) [32] and other authors [26].

3.7 Classification Evaluation Methods

The quality of any given algorithm for a specific problem is manifold and not deterministic. Many evaluation metrics for different algorithms exist. Nonetheless, there is no single best evaluation metric, and it often depends on the problem at hand to choose which metric to use.

Generally spoken, evaluation metrics are based on an evaluation of misclassified entries over correctly classified entries, but they differ in the way they evaluate each. Some general concepts are:

- True Positives (TP_c): Number of entries predicted to be of a given class c that actually belong to that given class c
- True Negatives (TN_c): Number of entries predicted not to be of a given class c that actually do not belong to that given class c
- False Positives (FP_c): Number of entries predicted to be of a given class c that actually do not belong to that given class c
- False Negatives (FN_c): Number of entries predicted not to be of a given class c that actually belong to that given class c

3.7.1 Accuracy

Accuracy is arguably one of the most intuitive classification metrics. It is made up of the relation between all correctly classified items over all items. Thus, it is a relative

value between 0 and 1. The accuracy can be calculated for a single class:

$$Accuracy_c = \frac{correct\ predictions\ of\ class\ c}{total\ predictions\ of\ class\ c} = \frac{TP_c + TN_c}{TP_c + TN_c + FP_c + FN_c} \tag{3.31}$$

Or over all classes from the dataset:

$$Accuracy = \frac{correct\ predictions}{total\ predictions} = \frac{\sum_{c \in classes} TP_c + TN_c}{\sum_{c \in classes} TP_c + TN_c + FP_c + FN_c} \tag{3.32}$$

The inversion of accuracy is called error rate. Whereas accuracy predicts the relative amount of correctly predicted labels, error rate calculates the relative amount of falsely predicted labels.

$$Error\ Rate = \frac{wrong\ predictions}{total\ predictions} = \frac{FP + FN}{FP + TP + FN + TN} \tag{3.33}$$

Thus either a high accuracy or a low error rate is desirable.

Accuracy is a good measurement if the goal is to find the exact class and penalise everything that was falsely predicted. It is sometimes criticised because it does not weigh errors between classes. For example, if an image of a cat were predicted as a dog, the accuracy would be 0, although predicting the cat as a dog might still be better than predicting it as being a truck. So there can be a notion of closeness between labels that accuracy is not able to capture. Moreover, a given entry's correct label may be ambiguous, e.g., if there are multiple objects in a picture that could change the label of the image, see figure 35.

Additionally, accuracy is a measure that can be applied for supervised learning problems with balanced datasets. Nonetheless, it usually gives biased results for imbalanced datasets and should not be used for such problems. Given an imbalanced dataset with 99 % of the samples belonging to class A and 1 % belonging to class B, as it is oftentimes the case in fraud detection cases. A predictor that always predicts class A would have an accuracy of 99 %. Nonetheless, this classifier does not fulfil the purpose of predicting class B. Therefore, accuracy is not appropriate as an evaluation metric here.

3.7.2 Top-n Accuracy

The top-n accuracy allows a model to have n chances of making the correct prediction. The model makes n predictions, and if the correct label is within these n labels, the model predicted it correctly. Doing this gives the model more robustness to the problem of closely missing the target with one prediction as described in chapter 3.7.1 as it is more likely to have the right label within n predictions if the other predictions

Figure 35: Examples of ambiguous images from [5] and [14]

are closer to the real label ([75], p. 15). To illustrate this with the same example from section 3.7.1: The predictor of the cat could have predicted dog as the most likely label, but a top-2 accuracy could have predicted the labels (dog, cat), where the correct label is within the set of predictions. The other exemplified predictor could have predicted (truck, elephant), which clearly misses the target. Another advantage of the top-n accuracy is that it is more robust to ambiguities or errors within the correct labels of an image. For example, training images in ILSVRC-2012 are human-annotated, and they can have a bias or even errors. An example of potentially ambiguous pictures can be found in figure 35. This is the case when there are multiple objects in a picture, e.g., a dog and a cup. An annotator could have focused on, detected and labeled a particular part of an image, say a ”dog”, while another annotator or a predictive algorithm would have labelled a ”cup” in the picture. If both classes appear in the image, a good image classification algorithm is likely to put both labels within the top-n-labels and thus reduces the impact of subjectivity.

A suitable choice of the hyper-parameter n depends on the number of targets. The most commonly used top-n accuracy is top-5 accuracy, but this choice depends on the total number of classes and the ambiguity between the classes. The above-introduced

measure of accuracy from section 3.7.1, is merely the special case of top-1 accuracy. The opposite of top-n accuracy is the top-n error. The top-n error was used in the ImageNet competition, with top-n accuracy = 1− top-n error. Benchmarks on ImageNet data usually report top-5 errors ([75], p.5).

3.7.3 Confusion Matrices

A confusion matrix is used for categorical predictions. It shows the confusion a predictor has among predicting samples into given classes. For every pair of classes (A, B), the confusion matrix shows the number of examples predicted to be in A that are actually labelled as B for all $A, B \in CLASSES$. Visually spoken, the x-axis denotes the predicted class, and the y-axis denotes the actual class, see figure 36.

This way, a matrix is created that shows how many examples were mislabelled between which classes. The diagonal of the matrix has the pair (A, B) with $A = B$ and thus shows the number of correct predictions.

Figure 36 shows an example confusion matrix for a binary classification problem. While the green cells represent correctly classified samples, the red cells represent misclassified examples. From the table, one can quickly identify the primary cause of confusion and include this knowledge into the next step's model creation. In this example, most confusion is among cases that are classified to be positive but are actually negative (False Positives).

n = 165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	TN = 5	TP = 100	105
	55	110	

Figure 36: Example of a binary confusion matrix

A confusion matrix is useful to improve a model's prediction because the modeller learns which part of the prediction still requires better training. It can be helpful as a qualitative evaluation measure, but it is not objectively quantifiable.

3.7.4 Precision, Recall and F_n Measure

Precision and Recall are quantitative metrics for binary classification tasks. They enable a modeller to focus on a specific property of a system. They can be used in combination to evaluate the overall quality of a model or individually to emphasise specific characteristics.

$$Precision = \frac{TP}{TP + FP} \tag{3.34}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.35}$$

The recall is also called sensitivity or True-Positive-Rate (TPR) as it measures the relative amount of correctly predicted positive examples. The False-Positive-Rate (FPR), also called fallout or probability of false alarm, is the opposite of the TPR, measuring the relative amount of falsely positive predicted examples over all negative examples in the sample:

$$FPR = \frac{FP}{FP + TN} \tag{3.36}$$

The lower the FPR, the better, while the TPR behaves contrarily.

Precision and recall are both measures that can individually give information about the quality of a binary classifier. But there is no general rule which measure to focus on. Therefore, the F_n measure was developed. The F_n measure is a combination of precision and recall that is meant to make model evaluation more objective. It is calculated as follows:

$$F_n = (1 + n^2) \cdot \frac{Precision \cdot Recall}{(n^2 \cdot Precision) + Recall} = \frac{(1 + n^2) \cdot TP}{(1 + n^2) \cdot TP + n^2 \cdot FN + FP} \tag{3.37}$$

The most common cases of the F_n measure are F_1 and F_2.

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{3.38}$$

Different to top-n accuracy measure, precision and recall have been developed for binary predictions. Their application on multi-class problems is limited. The only possibility of applying precision and recall to multi-class problems is by making them binary. This way, a precision and recall value can be calculated for every class in the form of one label versus all other labels. This can be used for evaluating a single class' performance but is not expandable to all classes. Due to this, the F_n measure is also limited to binary classifications.

3.7.5 ROC AUC Score

The Receiver Operating Characteristic (ROC) curve is a graphical visualisation of the True-Positive-Rate against the False-Positive-Rate. These two measures are calculated

at various threshold settings to form a curve that starts at (0,0) and ends at (1,1). Various TPR and FPR values are calculated as a function of a classifier's parameters, and the results form a curve that can be plotted on the ROC grid. A perfect classification would be at (0,1)[6]. Everything below the diagonal line of $TPR = FPR$ is considered a "bad" classifier while everything above the diagonal outperforms a random estimation. Nonetheless, every "bad" classifier can be inverted to be above the diagonal line by always choosing the opposite.

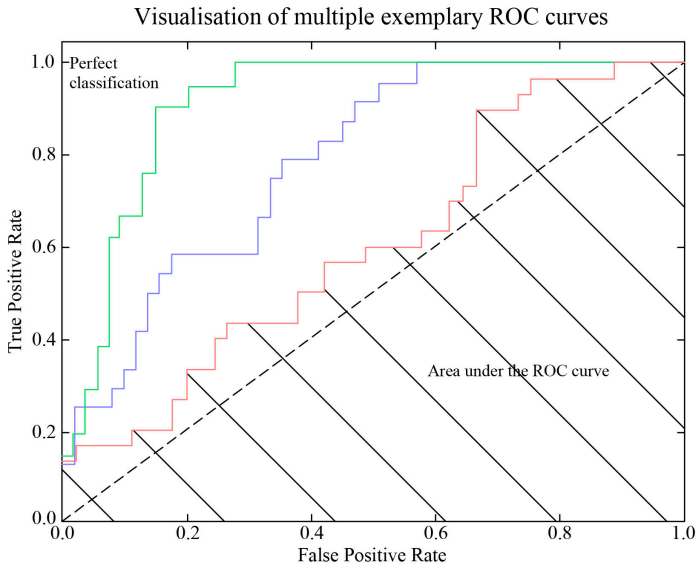


Figure 37: Example of a ROC curve

The ROC area under the curve (ROC AUC) score measures the area under the ROC curve. The larger this measure, the better a classifier performs because ROC AUC measures how much better the TPR rate is in comparison to the FPR. As precision and recall from section 3.7.4, the ROC AUC score has been developed for binary classification problems. Although there are approaches to convert this to multi-class problems [47], this ends up being pairwise ROC AUC scores of a binary form of one class versus all other classes.

3.7.6 Model Speed

If speed were not an issue in scientific research, any exhaustive search algorithm would outperform every other algorithm eventually. Therefore, every algorithm comes with a tradeoff between its effectiveness and efficiency, where the efficiency is related to speed and memory limitations. A theoretical new approach is valuable in itself, but it is only practically applicable if it is designed within the constraints of computational means.

[6] assuming the general case of FPR on the x axis

Chapter 4

Experimental Setup

The following chapter covers the experimental setup chosen for this thesis. The experimental setup was developed to test the research question.

Research Question "Can classification model performance in computer vision be improved by using different classification algorithms on high-level image features?"

Along displaying previous work in this area, this chapter outlines the procedure to test the research question in section 4.2 and introduces the choices of benchmark datasets in section 4.3, applied convolutional neural network model architectures in section 4.4 and the selected classification algorithms and evaluation metrics in sections 4.6 and 4.7. The computation was mainly done on commodity hardware[1] and remote computational cloud resources[2], which limits the size of the datasets, the benchmark algorithms and the hyper-parameters used for the comparison to a reasonable degree. Nonetheless, the algorithms and ideas developed are scalable and applicable on larger datasets subject to computational means.

The approach of this thesis assumes that time is not a constraint in the training process, but it focuses on improving the accuracy of the final model. The process introduced in sections 1.3 and 4.2 incorporates the normal process of training a convolutional neural network and thus will inevitably be longer than just the normal process of training

[1] Windows 10, Intel Core i7-4510U CPU @ 2.00 GHz 2.60 GHz, 8.00 GB RAM, GPU not used; MacBook Pro 2017, Intel Core i7 @ 2.80 GHz, 16 GB RAM, GPU not used

[2] Google Colaboratory (https://colab.research.google.com/): up to 12 GB GPU, subject to availability and usage, mostly unstable

a CNN. Therefore, the proposed modification from this thesis comes at the cost of potentially longer training time. It was not further investigated whether the training time of the modified training process could be improved. Apart from the training time, the final models can be stored more efficiently if the chosen classification algorithm takes up less space than the fully-connected neural networks. Hoffer et al. (2018) demonstrated that the majority of a CNN's complexity lies within the fully-connected layers [94]. Thus, replacing these layers with a more efficient classifier could make these networks less complex at runtime. Other than many developments in computer vision, this modification does not have a biological origin. Nonetheless, this idea goes back to the origins of computer visions when classification of high-level image filters was not limited to neural networks.

4.1 Work References

The field of image recognition is an actively researched field. Ever since the breakthrough of AlexNet in 2012 [51], most of its advances have been focussed on neural network architectures, especially around convolutional neural networks, as described in section 2.3. Some authors removed the fully-connected layers, e.g., GoogLeNet [95]. This is also called a fully-convolutional network. Further research has been done by Hoffer et al. (2018), showing that the computationally expensive last fully-connected layer can be set to constant values to reduce model complexity with little or no performance loss [38]. To illustrate their example, the researchers show that about 60 % of the 36 million model parameters of a ResNet-50 model from He et al. (2016) reside in the last fully-connected layer for the JFT-300M dataset[3] ([38], p.8).

Another interesting approach to reduce the complexity of the model architectures has been developed by Hasanpour et al. (2016) [30], where the authors show that they can perform on par to very large and deep neural networks with much fewer parameters and smaller model sizes. They achieve these results by applying a set of defined design principles. They prove their networks to perform similarly to state-of-the-art[4] CNN architectures on various well-known datasets, e.g., MNIST [64], CIFAR-10 (section 4.3.1), CIFAR-100 (section 4.3.2) and ILSVRC-2012 (section 4.3.3).

4.2 Scientific Procedure

From a technical perspective, the procedure is similar to the two-step procedure from section 1.3. The first part of the algorithm is a script that generates *intermediate* output features. These output features are produced on several image datasets as described

[3] JFT-300M is an internal dataset from Google with over 18k different classes in 300 million images, see also [94]

[4] as of August 2016

in section 4.3: CIFAR-10, CIFAR-100, ILSVRC-2012. Preprocessing is applied to the images, as described in section 4.5. The high-level output features are generated by training different neural network architectures, as described in section 4.4. The trained CNNs are then cut such that the fully-connected layers are removed, and only the flattened output from the convolutional layers remains. A high-level image feature in this context describes the output from the image filters from the last convolutional layer. The intermediate dataset is then generated as the flattened output of the trained convolutional layers along with the correct classification of the input image. The networks were created with the intention to produce good image filters, but the hyper-parameters were only fine-tuned to a reasonable degree, as this is not the focus of this thesis. Although a high initial benchmark score in the first step is desirable to assure that the produced image filters are useful to the neural network, the overall performance in the first step is not too relevant because every classification algorithm in the second step gets the same input from the first step. Therefore, it is still a fair competition between the models' performances in the second step. A visualisation of the first step as derived from figure A7 can be found in figure 41.

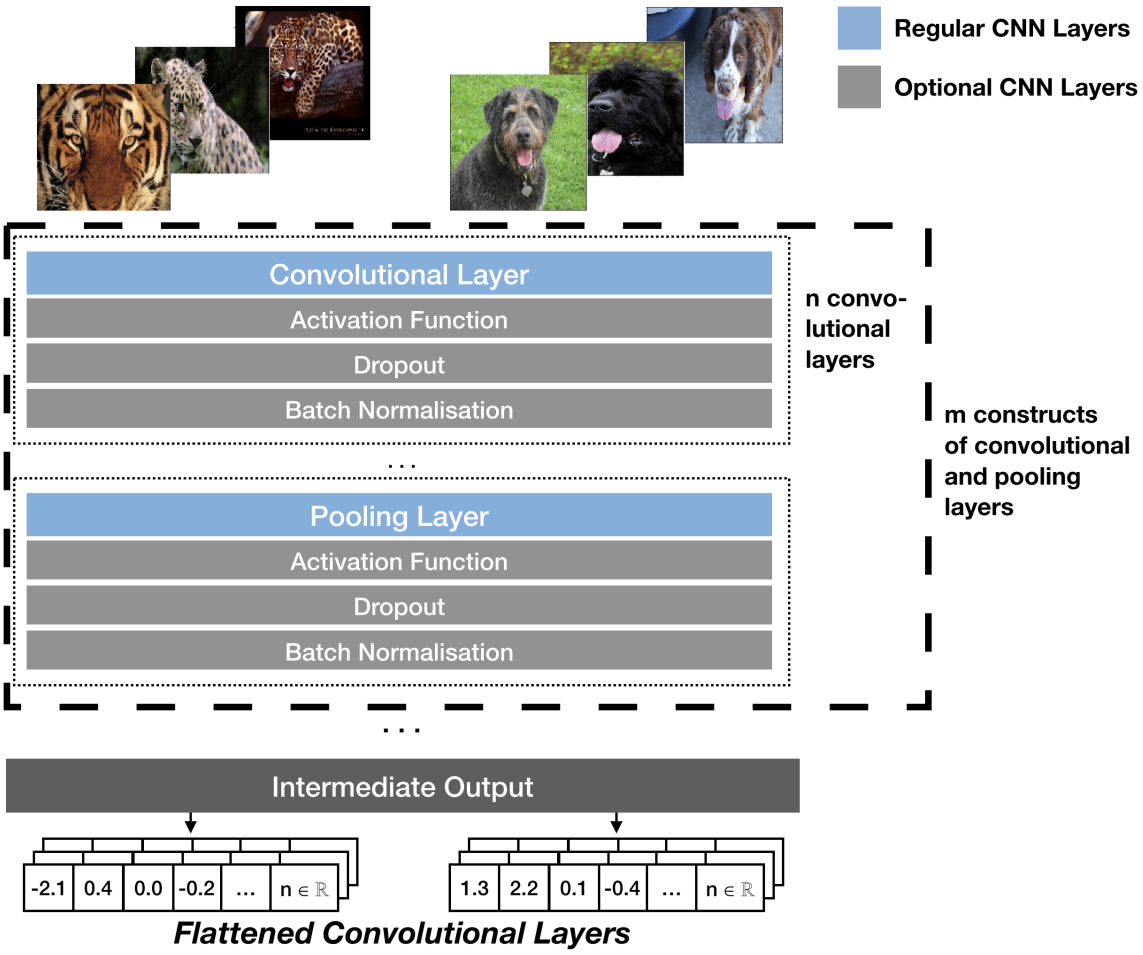


Figure 41: Visualisation of Intermediate Data Creation

The second script trains different classification algorithms on the intermediate datasets to test if they can behave better than the original convolutional neural network, as described in section 4.6. The performance of conventional MLP architectures is benchmarked through the inclusion of multiple MLP architectures in the second step. Furthermore, the performance of the initially trained CNNs is disclosed for reference (where available). A further benchmark of the original convolutional network is determined by including MLP architectures as part of the classification algorithms. The goodness of each classification algorithm is evaluated along several classification metrics, as introduced in section 3.7 and selected in section 4.7. A visualisation of the second step as derived from the example in figure A7 can be found in figure 42.

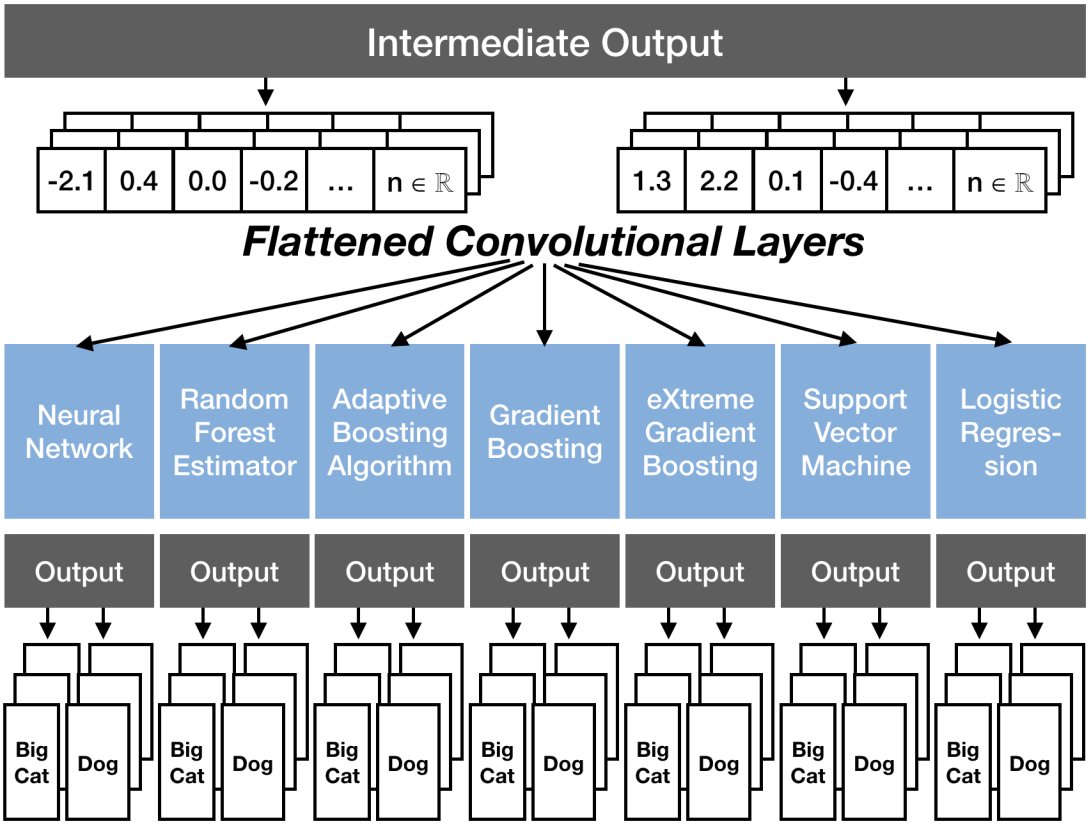


Figure 42: Visualisation of Intermediate Data Classification

To get statistically significant, convincing results, several measures are put into place. The approach is tested on different datasets, as introduced in section 4.3. This will evaluate whether there are patterns or specific characteristics in the datasets that make certain algorithms perform better than others. These datasets have different sizes and different numbers of classes. Furthermore, the conventional MLPs' performance is compared to multiple other algorithms. All algorithms' hyper-parameters are explored through a grid search with previously chosen options as outlined in section 4.6. The

training set is split from the test set, with a 5:1 split for CIFAR-10 and CIFAR-100[5] and a 3:1 split for ILSVRC-2012 (75 % training, 25 % test). This is done even before training the original CNN in the first step. The whole CNN then learns on the data from the training set. The intermediate data for both the training and the test set is produced from that trained CNN model.

In step 2, when training the different classification algorithms on the intermediate data, the hyper-parameters are chosen from, and the final model is trained on the training split. Hereby, the training data is cross-validated with k-fold cross-validation, with a value of $k = 10$, thus 10 % validation data per cross-validation run. The validation data in this second step is, therefore, part of the training data from the first step. The results of the cross-validations are saved and the p-values for statistical significance are determined. The final algorithms' performances are validated on the initially split test data that the algorithm was never trained on. This procedure enables comparability and statistical significance of the results. For the pre-trained models, the author cannot verify whether the external models were already trained on the test data. For ILSVRC-2012, the data was taken from the validation set which is usually not used for training. For CIFAR-10 and CIFAR-100, there is a high chance that the network was partially trained on the test data. This implication should not make a difference to determine the classifiers' performance in the second step because all algorithms in the second step are trained on the same input data.

Although this idea is not transfer learning in its standard application, as introduced in section 2.3.10. But the availability of pre-trained models for transfer learning enables the reuse to enhance the learning process. Thus, this is more of a learning enhancement than a transfer learning approach because the dataset to train on is the same[6].

4.3 Dataset Benchmarks

Subject to the limitations of computational resources, the author chose three well-known datasets to benchmark the research question of this thesis.

4.3.1 CIFAR-10

CIFAR-10 is a dataset of 60000 32×32 pixel colour images that belong to 10 classes, therefore $I^{inp} = 32 \times 32 \times 3$. The dataset is balanced. Hence, every class holds 6000 images. The dataset is already pre-split into 50000 training images and 10000 test images.

[5] as per default options of the dataset

[6] technically, the dataset to further train the classifier is a subset of the original dataset, as described in section 4.3

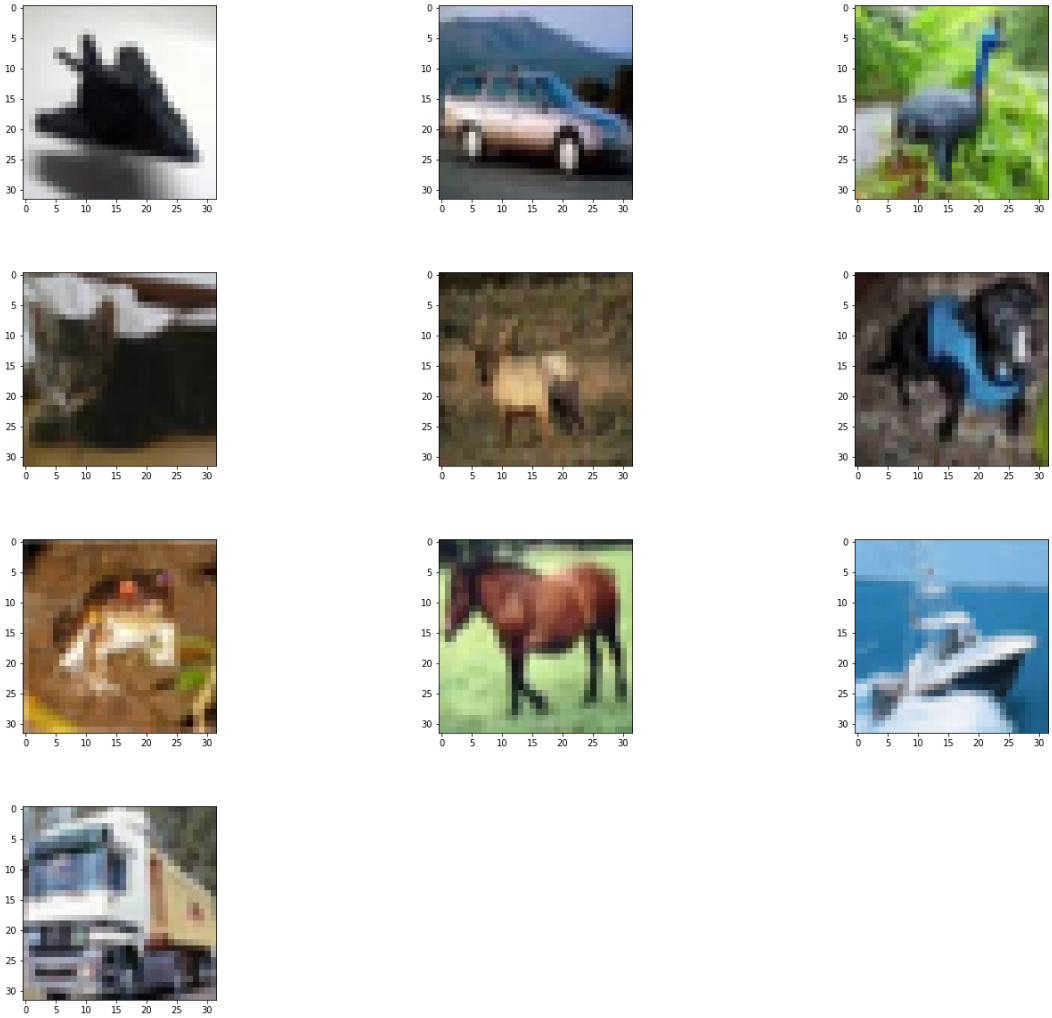


Figure 43: Samples from CIFAR-10 (from left to right, top to bottom): airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck

An overview of the classes with sample images that are contained in the CIFAR-10 dataset can be found in figure 43. A list of the normalised versions of these photos that were used for training can be found in Appendix A.2.3. The classes are mutually exclusive. The sample images in figure 43 show that the classification task can even be challenging for a human observer. Due to the low resolution of 32×32 pixels, the objects in the images are difficult to identify.

The CIFAR-10 dataset has a size of 163 MB when loaded into Python[7].

A list of successful CIFAR-10 architectures can be found in table 41.

[7] This number varies slightly in other programming environments

Model	Source	Param	Accuracy
SimpleNet	Hasanpour et al. (2018) [30]	5.48M	95.32 %
SD-110L	Huang et al. (2016) [40]	1.7M	94.77 %
VGG-19 (local benchmark)	Simonyan et al. (2014) [86]	15M	93.59 %
WRN	Zagaruyko & Komodakis (2016) [109]	600K	93.15 %
ALLCNN	Springenberg et al. (2014) [88]	1.3M	92.75 %
DSN	Lee et al. (2015) [56]	1M	92.03 %
FitNet	Romero et al. (2014) [90]	1M	91.61 %
ResNet-32 (depth of 32)	He et al. (2015) [31] (tested by [30])	475K	91.60 %
NiN	Lin et al. (2013) [58]	1M	91.19 %
dasNet	Stollenga et al. (2014) [92]	6M	90.78 %
Maxout (k=2)	Goodfellow et al. (2013) [24]	6M	90.62 %
SimpleNet (local benchmark)	Hasanpour et al. (2018) [30]	5.48M	88.52 %

Table 41: CIFAR-10 Benchmarks

Model	Source	Param	Accuracy
SD-110L	Huang et al. (2016) [40]	1.7m	75.42 %
SimpleNet	Hasanpour et al. (2018) [30]	5.48M	73.42 %
VGG-19 (local benchmark)	Simonyan et al. (2014) [86]	15M	70.48 %
WRN	Zagaruyko & Komodakis (2016) [109]	600K	69.11 %
ResNet-32 (depth of 32)	He et al. (2015) [31] (tested by [30])	475K	67.37 %
ALLCNN	Springenberg et al. (2014) [88]	1.3M	66.29 %
dasNet	Stollenga et al. (2014) [92]	6M	66.22 %
Maxout (k=2)	Goodfellow et al. (2013) [24]	6M	65.46 %
DSN	Lee et al. (2015) [56]	1M	65.43 %
FitNet	Romero et al. (2014) [90]	1M	64.96 %
NiN	Lin et al. (2013) [58]	1M	64.32 %
SimpleNet (local benchmark)	Hasanpour et al. (2018) [30]	5.48M	60.99 %

Table 42: CIFAR-100 Benchmarks

4.3.2 CIFAR-100

The CIFAR-100 dataset consists of 60000 images divided into 100 classes of 600 images each. The 100 classes are further group into 20 superclasses. The images are colored and 32×32 pixels in size, therefore $I^{inp} = 32 \times 32 \times 3$. The default train-test-split is in a relation of 5:1, meaning 50000 training and 10000 test images. The image classes are mutually exclusive. A sample of images from 10 classes can be found in figure 44. An exhaustive list of all 100 image classes, as well as their normalised version, can be found in Appendices A.2.2 and A.2.3 respectively. CIFAR-100 takes up a total of 161 MB space in Python[8].

A list of successful CIFAR-100 architectures can be found in table 42.

[8] This number varies slightly in other programming environments

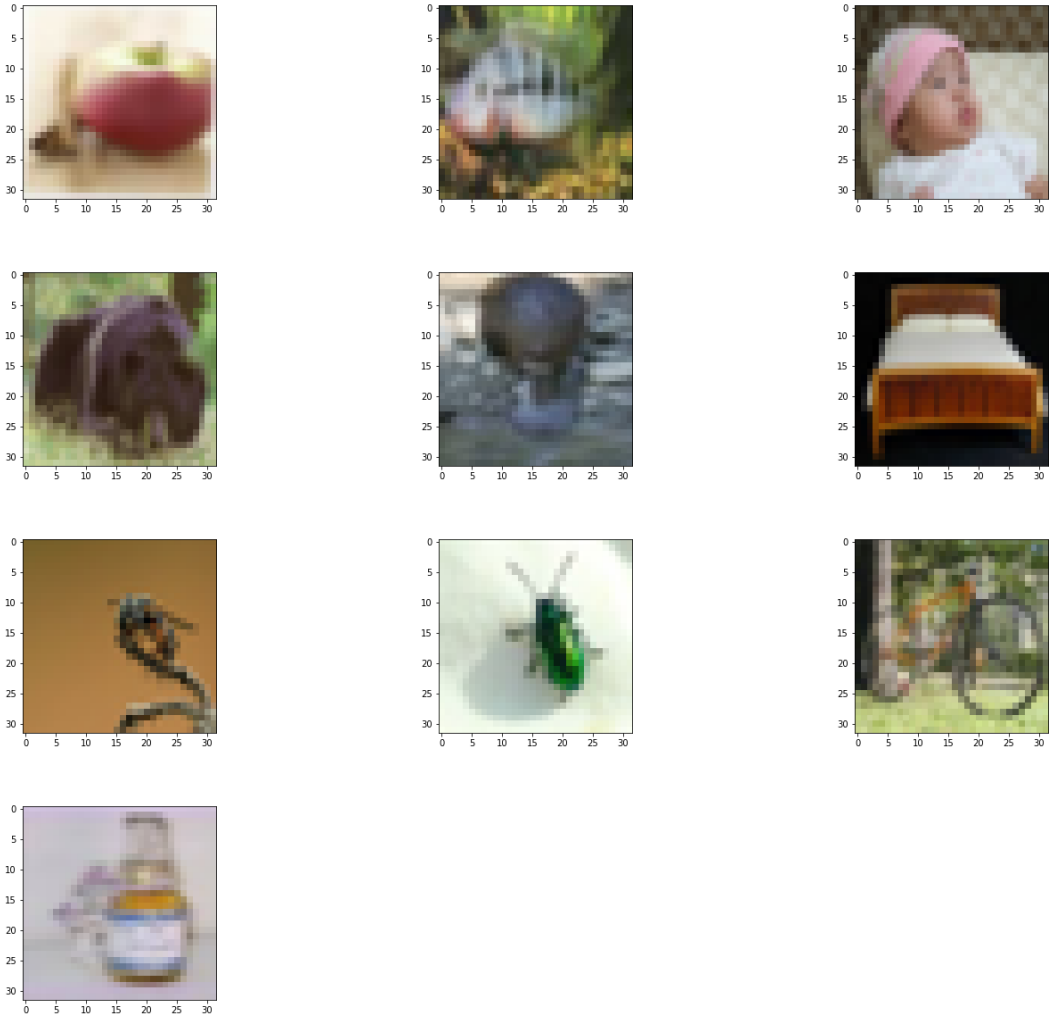


Figure 44: 10 classes from CIFAR-100, like apples, aquarium fish, baby, bed, snake, beetle, bicycle, bottles

4.3.3 ImageNet Large Scale Visual Recognition Challenge 2012

One of the most famous benchmarks for computer vision tasks is the immense ImageNet database that is built on the hierarchical WordNet [63] structure and aims to provide "an average of 500-1,000 clean and full resolution images" ([16], p. 1) per synset[9], with currently over 14,000,000 images in almost 22,000 synsets indexed ([16], p.1; [75]). The evolution of most famous model architectures and adjustments were revealed at the yearly ImageNet competition that measures model performance on various tasks, such as the detection of 1,000 categories ([75], p.15). The error measure used for the classification task in the competition is top-5 error, as described in section 3.7.2. A benchmark with an expert human annotator reached a top-5 error of 5.1 % on 1,500 test images.

[9] a synset is a set of synonyms on WordNet

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2012) dataset is the ImageNet dataset used for the 2012 ImageNet competition. It consists of 1.2 million images from 1000 categories with another 150000 images used for testing and validation. The dataset is very large, exceeding the limitations of most commodity hardware, especially when running algorithms on them that perform additional computations. Therefore, the author chose to subsample the dataset. From ILSVRC-2012, the author used the validation dataset with a size of 6.6 GB. From this validation data, the number of classes was reduced from 1000 classes to 100 classes to run the benchmark on, thus reducing the dataset size to 660 MB[10]. In chapter 5, this subset from the ILSVRC-2012 validation dataset is usually referred to as ILSVRC-2012 data for simplicity. Figure 45 shows 10 randomly chosen sample classes out of the 100 selected classes for this benchmark. From these sample images, the complexity of the classification task can be seen. The dataset contains all computer vision challenges from section 2.2. As an example, different breeds of the class concept belong to different classes. Thus, the model needs to learn unique properties from each breed. In fact, the pictures used to exemplify the computer vision challenges in Appendix A.1.1 all stem from the ILSVRC-2012 dataset used in this thesis.

The history of successful ILSVRC benchmarks was introduced in section 2.3. Nonetheless, this list is not comparable to the results created in this thesis because these benchmarks were made with 1000 target classes and with much more training data to learn on. A comparison of successful architectures on ILSVRC-2012 with 1000 classes can be found in table 43. One can see that the development of these architectures is chronological from the worst performing model AlexNet from 2012 to the best performing model Inception ResNet V2 in 2016. In this span of four years, the top-5 accuracy was improved from 79.8 % in AlexNet to 95.1 % in Inception ResNet V2, beating human performance. Another observation is that deeper models generally perform better than shallower ones. Merely comparing the ResNet architectures benchmarked (Resnet-50, ResNet-101, ResNet-152), the top-5 accuracy is higher the deeper the model.

4.4 Model Architecture Benchmarks

Different convolutional neural network architectures are used for the individual datasets. While the same network architectures can be applied on CIFAR-10 and CIFAR-100, many architectures that were designed for ILSVRC-2012 are not suitable for the much smaller images in CIFAR-10 and CIFAR-100. Through the procedure of the pooling operation from section 2.3.3, the images are downsampled in the network. If not enough pixels exist in the input layer, larger networks that use many pooling operations would downsample the image more than possible. Therefore, these network architectures

[10]This choice was also made after various memory errors causing the system to get stuck or fail

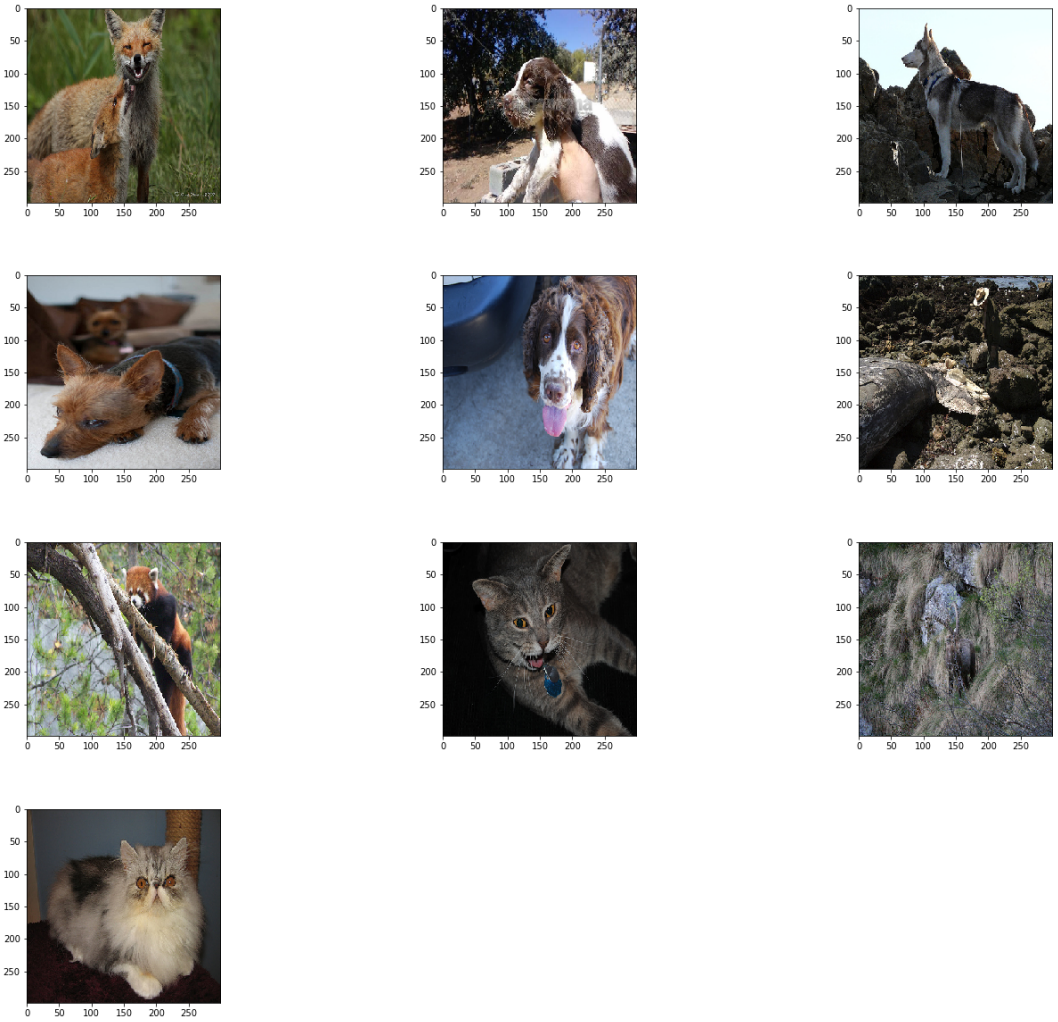


Figure 45: 10 sample classes from ILSVRC-2012

Model	Source	Top-5 accuracy
Inception ResNet V2	Szegedy et al. (2016) [97]	95.1 %
Xception	Chollet(2016) [12]	94.5 %
Inception V3	Szegedy et al. (2015) [96]	94.4 %
ResNet-152	He et al. (2015) [31]	92.9 %
ResNet-101	He et al. (2015) [31]	92.6 %
ResNet-50	He et al. (2015) [31]	92.0 %
VGG-16	Simonyan et al. (2014) [86]	89.9 %
GoogLeNet	Szegedy et al. (2014) [95]	89.1 %
Network in Network	Lin et al. (2013) [58]	81.2 %
CaffeNet	Jia et al. (2014) [44]	79.9 %
AlexNet	Krizhevsky et al. (2012) [51]	79.8 %

Table 43: ILSVRC 2012 Benchmarks

would need to be modified in order to apply them to the CIFAR datasets. One example of such can be found in tables 41 and 42, where a ResNet model with a reduced depth of 32 is used. Thus, images need to be of a minimum size for some networks to work on them.

4.4.1 ILSVRC-2012 Architecture Benchmarks

For the ILSVRC-2012 dataset, preexisting models designed for the ImageNet dataset are leveraged to create the intermediate data. Firstly, these networks are already pre-trained and thus do not require computational efforts of further training. Moreover, these models have been developed explicitly as benchmarks for the ImageNet dataset and should, therefore, perform well on these datasets. Three models are chosen because they achieve state-of-the-art results on the ImageNet dataset, all work best with an identical image input size of $299 \times 299 \times 3$, require the same preprocessing steps, and are available with pre-trained weights in software suites[11]. The following models are used:

- **Inception V3**: InceptionV3 is based on a paper from Szegedy, et al. from 2016 [96]. It is based on the idea of inception modules from section 2.3.7 but is more of a combination of multiple ideas from researchers over the past few years. "The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully-connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax." [2] InceptionV3 manages to reach an accuracy of 78.8 % and a top-5 accuracy of 94.4 % on the ILSVRC-2012 validation set (without blacklisted examples)[12]. Inception V3 requires a minimum input size of $139 \times 139 \times 3$ pixels.

- **Xception**: The eXtreme Inception (Xception) network is also based on the overall architecture of the inception network, but the inception modules are enhanced with depthwise separable convolutions, as described in section 2.3.2. Xception reaches an accuracy of 79.0 % and a top-5 accuracy of 94.5 % on the ImageNet dataset. The minimum input size for the Xception network is $71 \times 71 \times 3$ pixels.

- **Inception ResNet V2**: Inception ResNet is a combination of inception modules from section 2.3.7 with loophole connections from ResNet models as described in section 2.3.8. This modification mainly decreases training time but proved to lead to slight improvements in classification accuracy[13]. InceptionResNet

[11] e.g., see [48] for model architectures in Keras
[12] see [96] for more details
[13] see [97] for more details

reaches an accuracy of 80.1 % and a top-5 accuracy of 95.1 % on the ILSVRC-2012 validation set (without blacklisted examples)[14]. The minimal input size for Inception ResNet V2 is $139 \times 139 \times 3$ pixels. For simplicity, Inception ResNet V2 is also simply referred to as Inception ResNet in this thesis.

4.4.2 CIFAR Architecture Benchmarks

For the CIFAR-10 and CIFAR-100 datasets, identical model architectures are used, because the two datasets have the same input shape of $32 \times 32 \times 3$. Only the last fully-connected output layer differs since CIFAR-10 expects 10 output neurons and CIFAR-100 has 100 output classes. All custom MLPs are designed with constant padding at the borders. For padding, the overlapping entries are filled with 0s for the convolutional layers and with $-\infty$ for the Max-pooling layers.

The following MLP architectures were chosen for both CIFAR-10 and CIFAR-100:

- **CNN-1:** The first CNN can be found in figure 46. Its architecture is a 10-layer convolutional neural network[15] with a final fully-connected output layer. The network has 6 convolutional layers which all use convolutional windows of $k_w = k_h = 3$ and a horizontal and vertical stride of 1. The first two convolutional layers have depths of 128, the following two convolutional layers have depths of 256 and the last two convolutional layers have depths of 512. After each convolutional layer, an exponential linear unit (ELU) activation is applied. After the activation of every second convolutional layer, a pooling layer with a pooling window of 2×2 using Max-Pooling is applied. The horizontal and vertical strides are also 2, such that the image width and height are approximately halved[16] after each pooling operation. The pooling operations do not use padding at the borders[17]. After every pooling operation, a dropout layer is added. As proposed in section 2.3.5, the dropout probability increases from 0.1 after the first pooling layer to 0.25 after the second one and 0.5 after the third pooling layer. After this, the output is flattened, and two dense layers follow. The first dense layer has 1024 neurons and is followed by an ELU activation and a dropout layer with probability 0.5. The second dense layer (output layer) makes the mapping to the number of output classes, 10 or 100. This final output is squashed between 0 and 1 through the application of a softmax function. This network produces 2048 high-level intermediate features.

[14] see [97] for more details

[15] only counting hidden convolutional, dense and pooling layers

[16] approximately halved depending on whether the width and height are even or uneven numbers

[17] since the input images for both CIFAR datasets have 32 pixels in both dimensions, they can be perfectly divided by 2 for 5 times. Since the width and height stay even during these division, border handling is not necessary.

- **CNN-2:** The second convolutional neural network (CNN-2) can be found in figure 47. It is constructed to be smaller than the first network and only has 7 hidden layers. It follows the same design principle as the CNN-1 of creating two convolutional layers followed by one Max-Pooling layer. The activation functions after the convolutional layers are changed to rectified linear units (ReLU). This construct is only used twice instead of three times as in the first convolutional neural network. This means only four convolutional and two pooling layers are used in total. The parameters for the pooling layers are left the same. The depth of the convolutional layers is reduced to 32 for the first block (first two convolutional layers) and 64 for the latter block (last two convolutional layers). Dropout after these blocks is kept constant at 0.25. The flattened output is then passed to a dense layer with 512 neurons, followed by a ReLU activation and a dropout layer with probability 0.5. The output layer is identical in structure to the one of CNN-1, making the squashed prediction of 10 or 100 classes with a fully-connected layer. This network structure produces 2304 high-level intermediate features as input to the fully-connected layers.

- **SimpleNet:** SimpleNet is visualised in figure 48. It has 18 hidden layers of which 13 are convolutional layers, and five are Max-Pooling layers. The output layer making the final prediction is a regular fully-connected layer. The convolutional layers use 3×3 convolutional windows for the first ten and the 13th layers, while the layers 11 and 12 use 1×1 convolutions. The depth of the convolutional layers increases from 64 in the first layer to 128 for the layers 2 till 6, 256 for the layers 7 to 9, 512 for layer 10 and 2048 for the 11^{th} convolutional layer. The convolutional layers 12 and 13 have a depth of $k_d = 256$. All convolutional layers have horizontal and vertical strides of 1 and apply padding at the borders such that the image dimensions are not altered after the convolutional operation. Batch normalisation and ReLU activation are applied after the convolutional layers 1 to 6 and 8 to 10. In layer 7, batch normalisation and the activation function are only applied after the pooling operation. Convolutional layers 11 to 13 only use a ReLU activation after the convolutional layer, without batch normalisation. Dropout is consistently applied after either batch normalisation, the activation or max pooling (where applicable) for the convolutional layers 1 to 8 as well as 10 and 11 with a probability of 20 %. In the 9^{th} layer, the dropout function was applied before the pooling operation. The five max pooling operations were applied after the convolutional layers 4, 7, 9, 12, 13. The pooling operations all use pooling windows of 2×2 with horizontal and vertical strides of 2 and no padding at the borders. All convolutional layers are initialised with the Glorot normal initialiser [23]. SimpleNet produces 256 high-level image features.

- **VGG-19:** VGG-Net won the localisation task on ImageNet in 2014 and got second

in the classification task after GoogLeNet. The general idea is to build deeper networks with smaller convolutional filters that essentially cover the same receptive fields but enable more transformations and non-linearities, as explained in section 2.3.1. The VGG network of this thesis is a version adapted to the CIFAR datasets with 19 hidden layers (VGG-19). The network is visualised in figure 49. Of the 19 hidden layers, 13 layers are convolutional, 5 layers are Max-Pooling layers, and there is one dense layer at the end before the fully-connected output layer. As previously mentioned, the convolutional layers all use 3×3 image filters with padding at the borders and with horizontal and vertical strides of 1 such that the image dimensions are maintained. As the network gets deeper, the depth of the convolutional layers increases as well. The first two convolutional layers have a depth of 64, the proceeding two layers have a depth of 128, the subsequent three layers have a depth of 256, and the following 6 convolutional layers have a depth of 512. The final dense hidden layer also has 512 neurons. After each convolutional layer, a ReLU activation function is applied. The same goes for the final hidden fully-connected layer. After the activation, batch normalisation is applied. The five Max-Pooling layers are layers 3, 6, 10, 14 and 18. They all use a pooling window of size 2×2 with horizontal and vertical strides of 2. Over the whole network, dropout is only applied in some layers. Nonetheless, the principal of increasing dropout probabilities is adhered to. Throughout the network, dropout is applied after the layers 1, 4, 7, 8, 11, 12, 15, 16, 18 and 19. Layer 1 has a dropout probability of 0.3 and layers 18 and 19 have a dropout probability of 0.5. The layers between layers 2 and 18 use a dropout probability of 0.4. VGG-19 produces 512 high-level image features as input to the subsequent classifier.

4.5 Image Preprocessing

Image preprocessing describes any transformation that is done to an input image before passing it to a classifier. Apart from changing the dimensions, this includes the colours, illumination or the orientation. Furthermore, filters can be applied to the image, shapes or edges may be modified and much more [106]. Image Preprocessing is principally used to facilitate overcoming the computer vision challenges introduced in section 2.2.

For ILSVRC-2012, the images were compressed to an image size of 299×299 pixels, thus $I^{inp} = 299 \times 299 \times 3$, with the depth representing the RGB colour channels.

Regarding the architectures used to benchmark the ILSVRC-2012 model, the models each have an identical preprocessing procedure, where the pixel values are scaled down to be between −1 and 1:

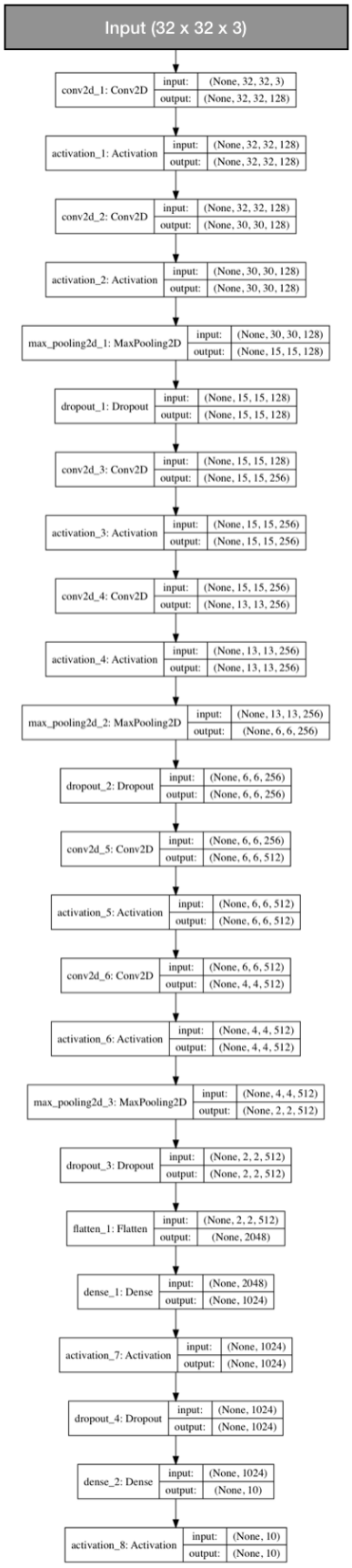


Figure 46: Visualisation of First Neural Network Architecture (CNN-1)

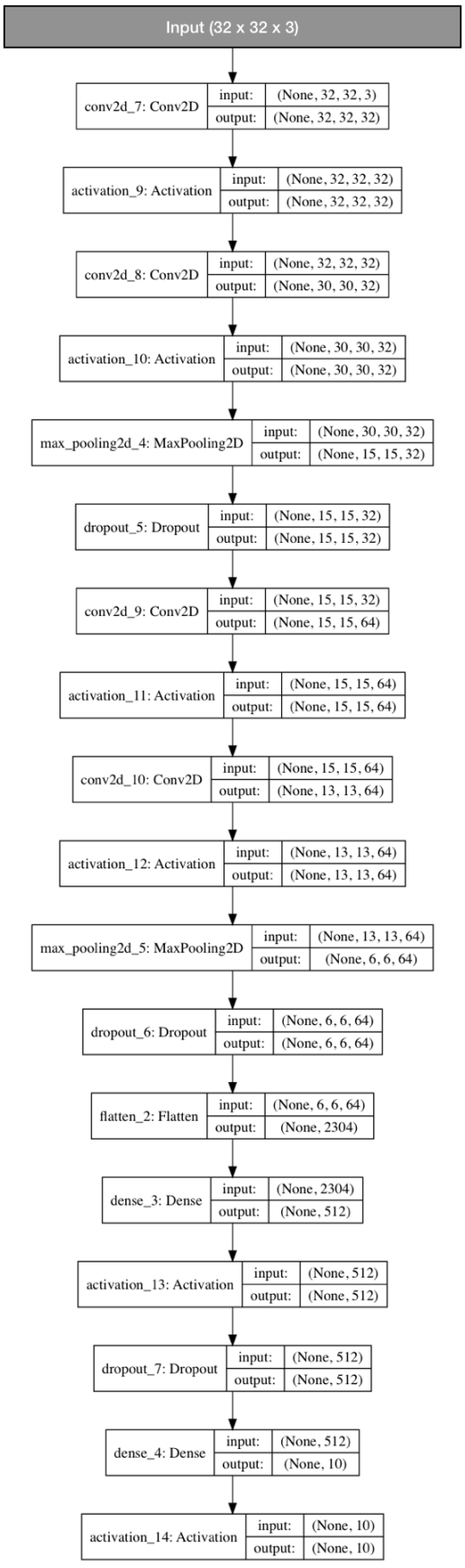


Figure 47: Visualisation of Second Neural Network Architecture (CNN-2)

Figure 48: Visualisation of SimpleNet Architecture

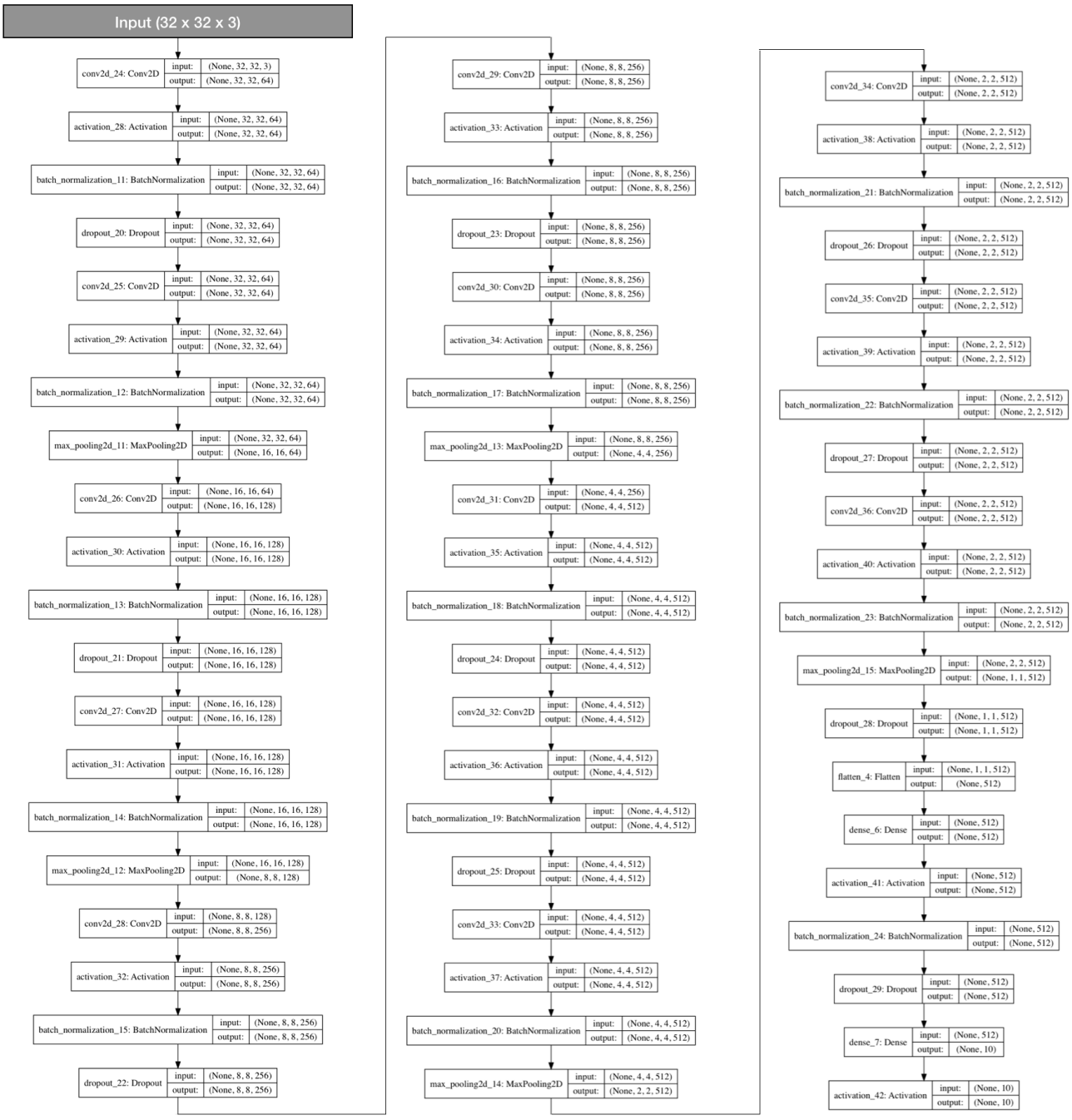


Figure 49: Visualisation of VGG-19 Architecture

1. **Divide all pixels by 255:** Scaling the pixels between 0 and 1

2. **Subtract 0.5 from every pixel value:** Moving the range to -0.5 to 0.5

3. **Multiply every pixel by 2:** Modifying the range to be between -1 and 1

For CIFAR-10 and CIFAR-100, the images are normalised using z-scaling. For CIFAR-10, the mean is 120.707 with a standard deviation of 64.15, while for CIFAR-100, the average is 121.936 with a standard deviation of 68.389. The standardised images can be found in Appendices A.2.1 and A.2.3. To exemplify this procedure, an example from the original class car can be found in figure 410, and its normalised version can be seen in figure 411.

Further image preprocessing techniques could be applied but are not used, mainly due to computational restrictions. It is common to enhance the input data with modified versions, e.g., add a rotated version of the input image, to make the learned features more robust. Another technique is to add images whose pixels were partially modified (pixel attacks) to let the algorithm be robust to these attacks. Apart from these modifications, excessive image preprocessing using filters or changing shapes or edges is not typical for CNNs.

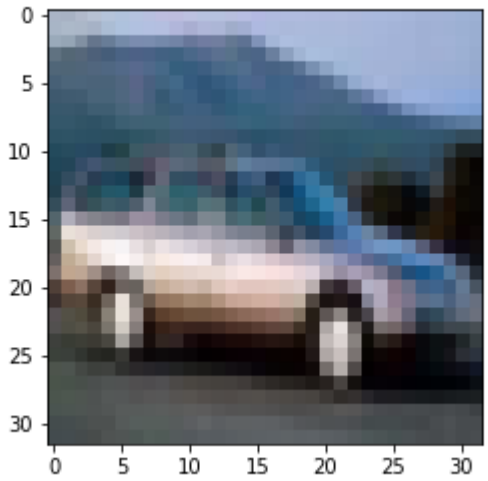


Figure 410: Raw example Picture of Class Car

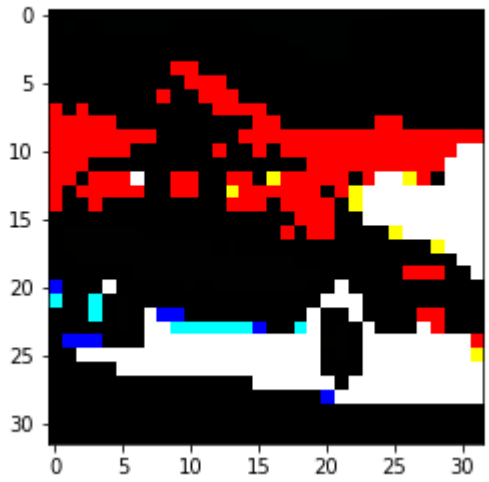


Figure 411: Normalised Example Picture of Class Car

4.6 Classification Algorithms Used

After creating the intermediate dataset, several classification algorithms, introduced in chapter 3, are applied to the intermediate dataset. The following classification algorithms are used with the specified parameters exhaustively tested in a grid search:

- **Support Vector Machine (SVM)**: 12 different SVM configurations are tested in the grid search. The first four SVMs use a linear kernel with c values of 1, 10, 100 and 1000. The other eight SVM configurations use RBF as the kernel with C[18] values of 1,10,100 and 1000 and gamma (γ)[19] values of 1×10^{-3} and 1×10^{-4}.
- **Logistic Regression (LR)**: 12 LR configurations are tested. For the loss function, each model either uses a $\ell 1$ or $\ell 2$ loss as the penalty of the model, while the value for c is set in the range of $1 \times 10^{-3}, 1 \times 10^{-2}, \ldots, 1 \times 10^{2}$.
- **K-Nearest Neighbours (KNN)**: 6 KNN configurations are tested. KNN can get computationally very expensive, especially for large values of k. The benchmark was done with $k \in 1, 5, 10$ and the leaf size either being 1 or 5 as well. All configurations use Euclidean distance as the distance metric.
- **Random Forests Estimator (RFE)**: 6 different RFE configurations are tested. The number of estimators is set to either 10, 100 or 1000 combined with Gini or Entropy diversion measure. The estimators are decision trees.

[18] c is the parameter associated to the slack variables from ξ
[19] γ is the free parameter of the RBF kernel function

- **AdaBoost Classifier (ADB)**: AdaBoost is benchmarked on 9 configurations. Models with either 10, 100 or 1000 estimators are combined with learning rates of 1.0, 0.5, or 0.1. The estimators are decision trees.

- **Gradient Boosting Classifier (GBC)**: Gradient Boosting's benchmark parameter combinations total to 6 models. The models have 10 or 100 estimators with learning rates of 0.05, 0.1 or 0.5. The estimators are decision trees.

- **XGBoosting Classifier (XGB)**: XGBoosting is benchmarked on 16 configurations. The number of estimators is also either 10 or 100. The learning rate is set to be 0.1 or 0.01. The estimators are decision trees. The maximum depth of each weak tree learner is limited to 1, 3, 5 or 10, where weak learners with depth 1 are decision tree stumps from section 3.1.

Each of these classifiers has many more hyper-parameters that could be potentially tested, but that would slow down computation a lot. It is important to notice that each classification algorithm is tested with various hyper-parameters that enable an objective comparison. Some algorithms have a greater search space of hyper-parameters. Also, for the case of XGBoosting, it is designed to perform better than regular Gradient Boosting and AdaBoost. Thus, more configurations are tested. These classification algorithms are compared against various Multi-Layer Perceptron architectures. All MLP architectures are trained with categorical cross-entropy loss, Adam optimiser with learning rates of 0.0001, 0.001 and 0.01 and accuracy as the optimisation metric. The models are trained for 10,20,30,50 and 100 epochs[20]. The final layer is encoded with one neuron per output class, thus each neuron hold a continuous value between 0 and 1 indicating the likelihood of that given class. Regarding topology, the author chose the three architectures that are inspired by the fully-connected layers of existing convolutional neural network architectures. These architectures are fine-tuned evolutionally through trial-and-error.

- **MLP-1**: MLP-1 is a neural network with one fully-connected hidden layer and one output layer. Its hidden layer is a dense layer with 1024 neurons, followed by an ELU activation function and a dropout layer with probability 0.5. The output layer with as many neurons as final classes is followed by a softmax activation to squash the values between 0 and 1. Its architecture is visualised in figure 412.

- **MLP-2**: MLP-2 has one fully-connected hidden and one output layer. Its hidden dense layer has 512 neurons, followed by a ReLU activation function and a

[20] Although this could be done more efficiently, the models are created from scratch every time to leverage the differences of random initialisation. Else, it would be sufficient to create a model for the maximum number of iteration and save the model after each indicated number of iteration

dropout layer with probability 0.5. The dense output layer with as many neurons as final classes is squashed between 0 and 1 with a softmax activation function. Its architecture is visualised in figure 413.

- **MLP-3**: MLP-3 only has one fully-connected output layer. Hence, it directly performs the mapping from the flattened image features to the output. The only layer has as many neurons as final classes, followed by a softmax activation function. Its architecture is visualised in figure 414. Technically, this is not an MLP since it consists only of an input and an output layer.

Since these three architectures are combined with 3 possible learning rates and 5 different number of epochs, the total number of created models is 45. During training, all model architectures converged. Thus, the hyper-parameters are assumed to be good.

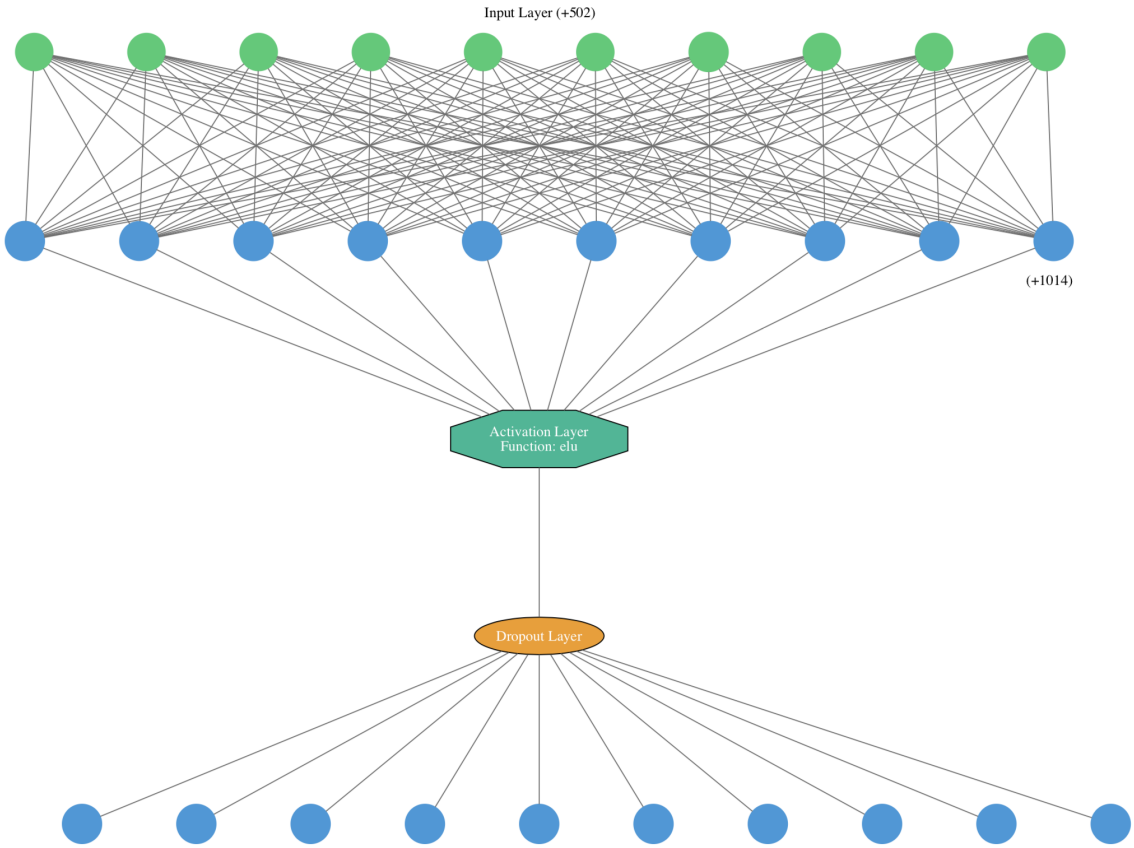


Figure 412: MLP-1 Visualisation for CIFAR-10

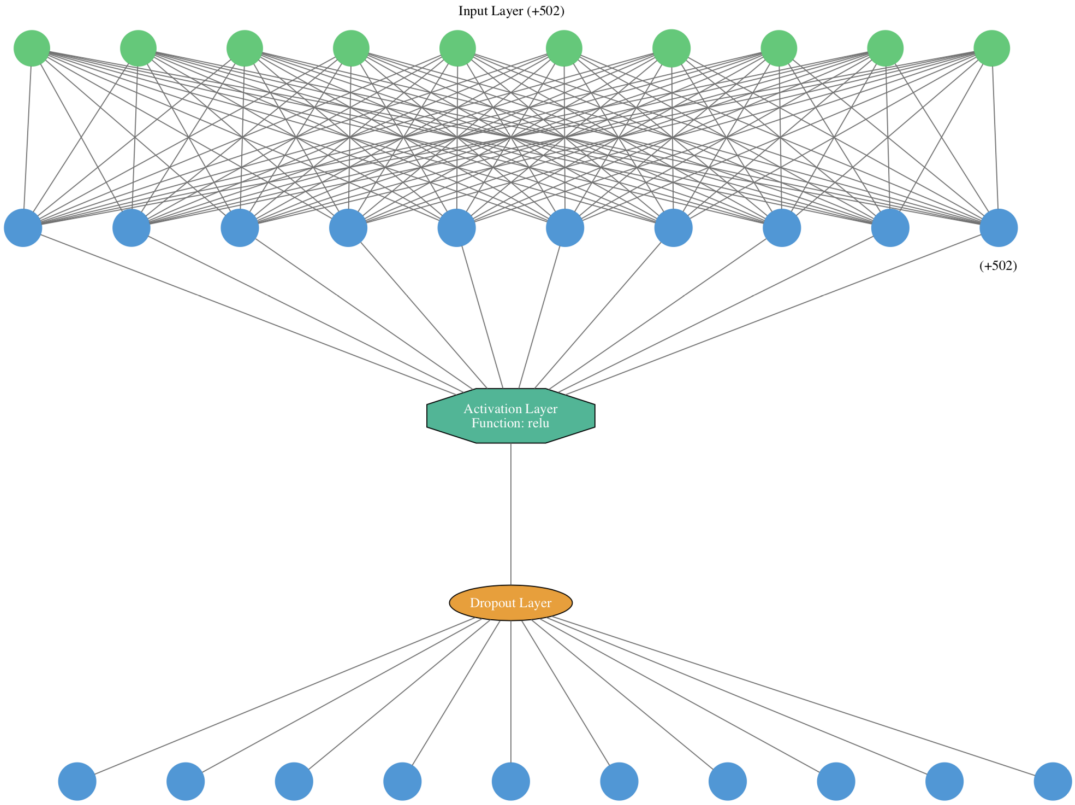


Figure 413: MLP-2 Visualisation for CIFAR-10

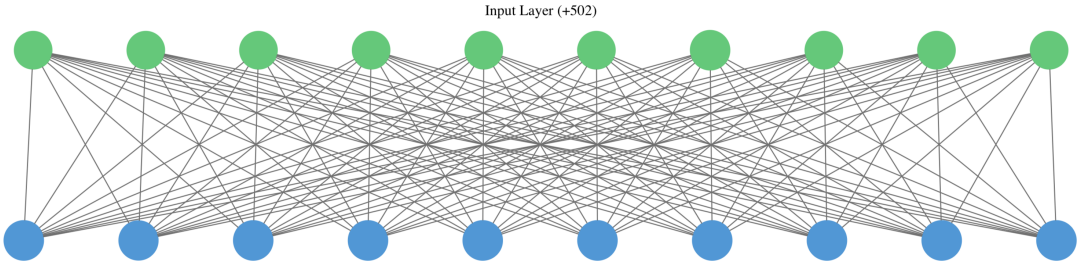


Figure 414: MLP-3 Visualisation for CIFAR-10

4.7 Selection of Evaluation Metrics

Since the thesis is dealing with a multi-class problem, many of the metrics from section 3.7 are not directly applicable. Therefore, only the top-n-accuracy score (or top-n-error), including "normal" accuracy (top-1-accuracy) are used as evaluation metrics. For CIFAR-10, only top-1, top-2, and top-5 accuracy are reported, where top-5 accuracy simplifies the problem to a binary problem since CIFAR-10 only has 10 classes. For CIFAR-100 and the reduced ILSVRC dataset, top-1, top-2, top-5, top-10, and top-20 accuracy are reported. Top-1 and top-5 test accuracy turn out to be strongly correlated on most datasets. Therefore, mostly top-1 accuracy is used in the discussions

in chapter 5, unless top-5 accuracy or other accuracy measures provide any relevant insights or new perspectives.

Apart from that, the complexity of the models is measured in terms of the size of the produced models. For the MLP models, the size is determined as the size of the model structure in JSON format and the trained weights serialised in HDF5 format. The other models are serialised in Python Pickle format.

The time complexity of training the individual models is not measured due to different training environments and conditions. The overall approach from this thesis is inevitably longer than just training a convolutional neural network. If n time units were available to train different CNNs of average complexity c_{CNN} time units, one could create the following number of CNN networks: $p_{CNN} = \frac{n}{c_{CNN}}$. The additional complexity of training k additional classification algorithms with an average complexity of c_{class} time units reduces this to $p_{thesis} = \frac{n}{k \cdot c_{class} + c_{CNN}}$. It can easily be seen how $p_{thesis} < p_{CNN}$ for any $k, c_{class} > 0$, hence for any additional classification algorithm benchmarked.

4.8 Software Architecture of the Final Solution

The code base used for this thesis can be found on GitHub[21]. The program is entirely written in Python but makes use of various external libraries that use other programming languages. The main external packages that are used are Keras [49] with TensorFlow [1] in the backend, sklearn [82] using various classification algorithms, cross-validation methods, and evaluation metrics and XGBoost [83] for the XGBoosting algorithm implementation.

The program is split over two iPython notebooks that recreate the process described in section 4.2. The first notebook does the data-producing step of loading in the input images, training the neural networks (where applicable) and returning the flattened output from the last convolutional layer. This notebook also saves the created models for later reuse, if necessary.

The second notebook performs the image feature analysis. It loads the data from the first step, applies different classifiers on the intermediate data and produces the final model. Furthermore, this notebook saves the created models such that they can be reused in the future, aggregates the performance metrics from the different models to save them in a CSV file and performs a significance test on the produced results.

[21] https://github.com/novajon/classy-conv-features

Chapter 5

Experimental Evaluation

This chapter summarises and interprets the results that were achieved using the experimental setup from chapter 4. The experimental setup consists of an initial step of creating candidate intermediate datasets through different CNNs, followed by a benchmark of different classifiers on these intermediate datasets. Firstly, the results and observations from the intermediate data creation step are displayed in section 5.1. After that, the results from the second classification step on these intermediate datasets are presented and described in sections 5.2 and 5.3. The achieved results are afterwards analysed and interpreted in section 5.4. Finally, section 5.5 summarises the previous results and draws general conclusions.

Throughout sections 5.1 and 5.3, the discussion is separated between the three initial image datasets used: CIFAR-10, CIFAR-100, and a subset of ILSVRC-2012. The concluding sections 5.4 and 5.5 abstract these individual observations to develop general patterns.

In the following, CIFAR-10, CIFAR-100, and the used subset of ILSVRC-2012 are referred to as initial or original datasets. The CNNs trained or obtained on these initial datasets, e.g., CNN-1, are called original, trained or benchmark CNNs. The produced datasets from the CNNs are referred to as intermediate datasets, usually denoted with the corresponding CNN and original dataset, e.g., CNN-1 intermediate dataset trained on CIFAR-10. This information is omitted if the context allows. The benchmarked classification algorithms are named benchmark classifiers or generally classification algorithms. When top-n accuracy is mentioned, the author refers to the top-n accuracy on test data, unless explicitly stated differently.

5.1 Output from Convolutional Neural Network Structures

The following section describes the observations that were made during the creation of the intermediate datasets through the training of CNNs on the initial datasets.

The training progress of CNN-1, CNN-2 and SimpleNet with the development of both loss and accuracy over time for the train and test data can be found in Appendices A.3.1.1 for CIFAR-10 and A.3.2.1 for CIFAR-100. The networks were trained for 80 iterations, and they all seem to have converged for both the training and the test dataset, apart from CNN-2 on CIFAR-100 (see figure A32). As described in section 4.4.2, the algorithms' architectures were selected after trying out different architectures and hyper-parameters, which is why the set of CNN models does not contain any entirely erroneously trained models.

5.1.1 CIFAR-10 Intermediate Observations

For CIFAR-10, the training of the three convolutional neural networks took 101:25 h[1]. The results can be found in table 51. VGG-19 is not trained separately because it is available as a pre-trained model for CIFAR-10. The overall accuracy of VGG-19 on CIFAR-10 is 0.9359, as reported for test and train accuracy in table 51. VGG-19 produces 512 high-level image features in the intermediate dataset.

Architecture	Top-1 Test Accuracy	Top-1 Train Accuracy	Model Size [MB]	Train Data Size [MB]	Test Data Size [MB]	Intermediate Features	Param
CNN-1	0.8504	0.9846	26.5	636.8	127.7	2,048	6.7M
CNN-2	0.8113	0.9155	5	520.2	104.4	2,304	1.3M
SimpleNet	0.8852	0.9869	22.1	69.6	14.3	256	5.5M
VGG-19	0.9359	0.9359	60.1	143.2	28.9	512	15M

Table 51: CIFAR-10 Intermediate Dataset Results, trained on MacBook Pro

The learning process of CNN-1 on CIFAR-10 for 80 iterations can be found in figure A24. CNN-1 has a very steep learning curve with both train and test data reaching an accuracy around 0.8 after 10 iterations already. After that, the test curve flattens out while the training accuracy improves until around iteration 35, where the accuracy is around 0.95. The final training accuracy is at 0.9846 while the test accuracy is only at 0.8504. This gap is slightly bigger than the one for the other training algorithms and indicates that this network architecture seemingly observes patterns in the training image data that are not generalisable. The loss of CNN-1 architecture on test data behaves a bit unsteady after around 20 iterations. The loss rises slightly again but in

[1] MacBook Pro 2017, Intel Core i7 @ 2.80 GHz, 16 GB RAM, GPU not used

combination with the development of the accuracy on test data, this is not a significantly negative behaviour. CNN-1 produces 2,048 high-level image features as input to its intermediate dataset.

The training progress of CNN-2 on CIFAR-10 for 80 iterations is plotted in figure A26. CNN-2 makes a similar but less steep development in comparison to CNN-1. Until around iteration 20, the learning progress on train and test data is quite similar, with the algorithm reaching an accuracy of around 0.75 on both datasets. After these initial 20 iterations, the CNN-2 architecture starts to learn some features that are not generalisable on the test set and thus overfits a bit on the training data. The final accuracy on train data is 0.9155. Nonetheless, the test dataset still improves slightly, reaching a final score of 0.8113, with a final difference of 10 percentage points between the train and test data. CNN-2 produces 2,304 high-level image features in its intermediate dataset, which is the most among all CNNs trained on CIFAR-10.

The learning process of SimpleNet on CIFAR-10 over 80 iterations is visualised in figure A28. SimpleNet has the steepest learning curve with most of its learning happening in the first 20 iterations, where it already reaches a training accuracy of more than 0.9. Remarkably, the difference between test and training accuracy does not increase much after 20 iterations, and the two measures are generally close. This indicates that the network can create relevant image features from the input data without overfitting too much on training data. SimpleNet produces 256 high-level image features in its intermediate dataset. The final accuracy for SimpleNet is 0.8852 on test and 0.9869 on train data.

To sum up, it is notable that SimpleNet converges the fastest on both training and test data and also reaches the best final accuracy. Nonetheless, the CNN-1 architecture achieves only marginally worse results with SimpleNet outperforming the test accuracy of CNN-1 by 3.5 percentage points. CNN-2 performs worse than the aforementioned two architectures, which could be caused by the reduced complexity of the model, leading to it being unable to observe more complex patterns. Overall, despite some indications of overfitting of the algorithms on the training data compared to the test data, the detected features that are not applicable on the test data do not worsen the test data's accuracy score. Since the test accuracy did not go down but stayed steady or even improved slightly, the networks are considered to be well suited for further analyses. Compared to the results from the pretrained VGG-19 net, the three benchmark models performed slightly worse (between 5 to 12 percentage points) on

unseen data[2]. By the size of the VGG-19 network, it can be seen that it is more complex than the other networks which is likely why it can observe more complex patterns in the image data. Apart from that, more time could have been invested in image preprocessing before passing the data to the CNNs to improve their final accuracy, as described in section 4.5. As mentioned before, this is not primarily within the scope of the thesis because the benchmark from this initial training step is not too relevant for the subsequent step.

5.1.2 CIFAR-100 Intermediate Observations

For CIFAR-100, the training of the three convolutional neural networks took 412:01 h[3]. The results can be found in table 52. As for CIFAR-10, a pre-trained VGG-19 model with an accuracy of 0.7048 on CIFAR-100 is used. VGG-19 produces 512 high-level image features in the intermediate dataset.

Architecture	Top-1 Test Accuracy	Top-1 Train Accuracy	Model Size [MB]	Train Data Size [MB]	Test Data Size [MB]	Intermediate Features	Param
CNN-1	0.5834	0.9297	26.5	602.5	120.5	2048	6.7M
CNN-2	0.4750	0.6308	5	501.9	100.4	2304	1.3M
SimpleNet	0.6099	0.9303	22.1	72.0	14.4	256	5.5M
VGG-19	0.7048	0.7048	60.3	141.3	28.3	512	15M

Table 52: CIFAR-100 intermediate dataset results

For CNN-1, most of the learning on the training data seems to take place within the first 60 iterations. At the end of the graph, the network converges towards an accuracy of 0.9297 after 80 iterations. The chart from figure A31 does not indicate that more training would have much additional effect on the network. Anyways, the accuracy curve already flattens out on test data after 20 iterations, reaching an accuracy of 0.5834. The loss curve even indicates that the loss on the test data increases after 20 iterations, suggesting that the algorithm starts to lose its generalisability and overfits on the train data. This observation does not reflect the development of the accuracy score. Furthermore, CNN-1 produces 2,048 intermediate high-level image features.

CNN-2 ends up with a much lower accuracy score, both on train and test than the other two network structures trained. This overall lower score seems to be explainable by CNN-2 having less complexity within its internal structures and thus not being able to detect more complex patterns in the input images. Nonetheless, the performance

[2]VGG-19 was not trained by the author thus it is not possible to determine which data points were used as train and test data. A comparison between the final model's performance with the performance of the other models on unseen data is therefore not wholly accurate.

[3]Windows 10, Intel Core i7-4510U CPU @ 2.00 GHz 2.60 GHz, 8.00 GB RAM, GPU not used

on test data seems to converge during the training process after around 40 epochs, see figure A32. The training curve indicates that it does not reach its maximum performance after 80 epochs with an accuracy of 0.6308. This does not imply that the results cannot be used for the benchmark, because the network converges on test data. Thus, any additional training would not be generalisable to unseen data. The final accuracy on test data is 0.4750. All additionally observed patterns in the input image would probably not be generalisable. CNN-2 produces 2,304 high-level features in its intermediate dataset, which is the most among all benchmarked CNN architectures.

As for CIFAR-10, SimpleNet converged the quickest towards its maximum accuracy of 0.9303 on train and 0.6099 on test data from CIFAR-100. The training curve from SimpleNet makes a similar but faster development than the one of CNN-1 on CIFAR-100, as can be seen in figure A30 for CNN-1 and figure A34 for SimpleNet. Although the graph from figure A35 indicates that the training loss could have reduced more if the CNN was trained for more epochs, the test accuracy did not improve much after 30 epochs into the training process, indicating that most further training is only applicable on the training data. This would lead to overfitting on training data. Since the test metrics for loss and accuracy stayed steady, there is no sign of malicious overfitting or less generalisability. The steepest learning process is within the first 10 epochs of model training. SimpleNet produces the fewest high-level image features with 256 features.

To summarise these findings, CNN-1 and SimpleNet reach a similar accuracy on their training data around 0.93. Nonetheless, SimpleNet can learn more quickly and abstracts better to unseen data, as its test data accuracy is about 2 percentage points higher. As on CIFAR-10, CNN-2 performs worse than the other architectures, probably because it can capture less non-linear relationships. The results from the custom-trained neural networks are worse than the pre-trained and more complex one for VGG-19. All other networks perform more than 9 percentage points worse. VGG-19 seems to detect more patterns in the train data that generalise to unseen data[4], reaching an ultimate accuracy of 0.7048, which is almost 10 percentage points higher than the second best from SimpleNet. Apart from the increased complexity of VGG-19, another reason for its superiority could lie within the data preprocessing stage. Section 4.5 indicates some more preprocessing techniques that could have been applied but were not used on the models within this thesis due to practical implications.

[4]VGG-19 was not trained by the author thus it is not possible to determine which data points were used as train and test data. A comparison between the final model's performance with the performance of the other models on unseen data is therefore not wholly accurate.

5.1.3 ILSVRC-2012 Subset Intermediate Observations

An overview of the model specifications created from the benchmarked CNN architectures on the subset of ILSVRC-2012 with 100 classes can be found in table 53. The top-5 accuracy measures slightly differ from the values reported in table 43 and described in section 4.3.3. The reason for that is that the numbers referenced below are the numbers reported from the Keras implementation [48], while section 4.3.3 reports the numbers from the authors' papers. From the numbers of features in table 53, one can already see that the intermediate datasets from the ILSVRC-2012 CNN networks contain many more high-level image features.

The best performing model on both top-1 and top-5 is Inception ResNet V2 with a top-1 accuracy of 0.8030 and a top-5 accuracy of 0.9530. From the model's size, it is also the biggest and most complex model. Apart from the structural advantages that the Inception ResNet architecture brings, this additional model complexity enables the model to learn more complex non-linear relationships within the data. Contrarily to the model's size, the produced train and test data size for the subsequent step is the smallest with 1,620 MB for train and 539 MB for test. This is an indicator that the network's complexity does not substantially lie in its last fully-connected layers. The number of produced features from the flattened convolutional layer is also the lowest among the compared models. Inception ResNet V2 produces 98,304 features for the intermediate dataset.

Inception V3 and Xception seem to be similar in overall model complexity with Inception V3 having a model size of 92 MB while Xception only has a size of 88 MB. Nonetheless, Xception can outperform Inception both on top-1 and on top-5 test accuracy. Xception reaches a top-1 test accuracy of 0.7900 while Inception V3 only reaches a top-1 test accuracy of 0.7790. The top-5 test accuracy of Xception is 0.9450 while the top-5 test accuracy of Inception is 0.9370. Regarding the complexity of the produced intermediate datasets, Xception generates the intermediate dataset with the most features: 204,800. Inception V3's intermediate dataset has 131,072 features. The train and test intermediate data sizes for Inception V3 are 2,200 MB and 732 MB respectively. For Xception, the train intermediate dataset size is 3,240 MB, and its test set has a size of 1,080 MB.

Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size [MB]	Train Data Size [MB]	Test Data Size [MB]	Intermediate Features	Param
Inception V3	0.7790	0.9370	92	2,200	732	131,072	21.8M
Xception	0.7900	0.9450	88	3,240	1,080	204,800	20.8M
Inception ResNet	0.8030	0.9530	215	1,620	539	98,304	54.3M

Table 53: ILSVRC-2012 Intermediate Dataset Results for 100 Classes

5.2 MLP Architecture Comparison on Intermediate Datasets

The following chapter describes the observations made when retraining MLP architectures on the intermediate datasets. This benchmark mainly shows how well a separated learning process can imitate the original learning process of a full convolutional neural network. Since the intermediate data is created by removing the fully-connected layers from the trained convolutional neural networks, training a fully-connected neural network on this intermediate dataset should be able to achieve at least comparable results to the initially trained models, apart from random influences. This is, given that the network structure used in the second step is similar to the one that is used for the final classification in the first step. The observations in this chapter are based on the intermediate datasets created from CIFAR-100. Nonetheless, the findings generalise to other datasets since the observations on CIFAR-100 were similar to the ones on the other benchmarked datasets.

As described in section 4.6, the MLP architectures are trained on the intermediate output data for 10, 20 and 50 epochs combined with learning rates of 0.001, and 0.0001. As this is done with 10-fold cross-validation, this leads to 60 neural networks being created per architecture with a total of 1,600 epochs of training 45,000 entities per epoch. Nonetheless, it can be observed that some of these hyper-parameters do not work well on the intermediate dataset. To exemplify this, the results on the first intermediate dataset, created through CNN-1, can be found in Appendix A.3.2.3. Most networks already converge after 10 epochs, and further training does not lead to any improvements, neither on the train nor on the validation set (during cross-validation). Figure 51 shows the behaviour of MLP-1 being trained on CNN-1 intermediate data after 10 epochs, whereas figure 52 shows the same training procedure after 50 epochs. It can be seen that the networks seemingly cannot detect any new patterns after 10 iterations. As for the learning rate, it turns out that a lower learning rate works better than a higher learning rate. Both on the training and the validation set, a higher learning rate led to less smooth learning curves with the networks making rather big jumps without converging to an optimal value. This can be seen by comparing figure 52 with MLP-1 being trained on CNN-1 intermediate data for 50 epochs with a learning rate of 0.0001 to figure 53 with the same network being trained with a higher learning rate of 0.001. The same behaviour is observable for the other MLP architectures. Experimentation with an even lower learning rate than 0.0001 did not lead to any significant improvements.

For the three MLP architectures, the results in tables 54, 55, 56 and 57 were achieved on the intermediate datasets from CNN-1, CNN-2, SimpleNet and VGG-19 respectively. Tables A17, A18, A19 and A20 in Appendix A.3.2.4 show the p-value of a two-sided student's t-test for the respective validation results. From the p-values, the statistical

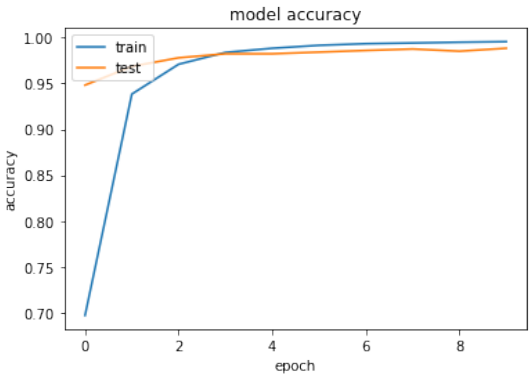


Figure 51: Learning Curve of MLP-0 on CNN-1 Intermediate Data for 10 Iterations with Learning Rate 0.0001

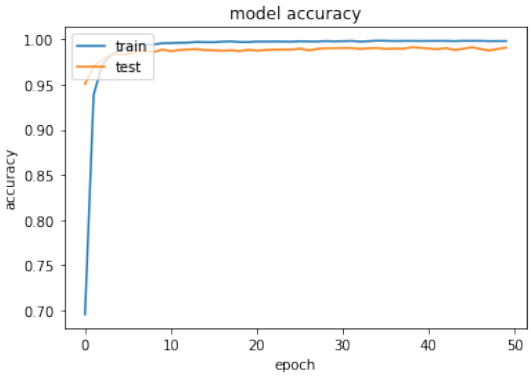


Figure 52: Learning Curve of MLP-0 on CNN-1 Intermediate Data for 50 Iterations with Learning Rate 0.0001

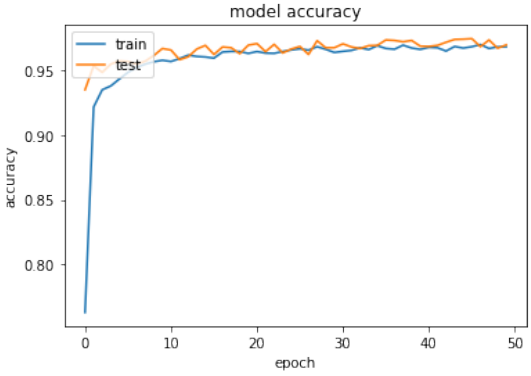


Figure 53: Learning Curve of MLP-0 on CNN-1 Intermediate Data for 50 Iterations with Learning Rate 0.001

significance levels of the results can be derived.

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP-1 LR-0.0001	0.5585	0.6770	0.8117	0.8790	0.9349	0.9896	0.9997	8,611 KB
MLP-2 LR-0.0001	0.5630	0.6860	0.8148	0.8874	0.9422	0.9884	0.9997	4,313 KB
MLP-3 LR-0.0001	0.5592	0.6753	0.8002	0.8717	0.9310	0.9888	0.9998	811 KB
MLP-1 LR-0.001	0.5415	0.5976	0.6277	0.6521	0.6922	0.9680	0.9929	8,161 KB
MLP-2 LR-0.001	0.5500	0.6381	0.7006	0.7305	0.7645	0.9738	0.9979	4,313 KB
MLP-3 LR-0.001	0.5293	0.6466	0.7803	0.8593	0.9288	0.9681	0.9979	811 KB

Table 54: CIFAR-100 MLP Classification Results on CNN-1 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP-1 LR-0.0001	0.4498	0.5773	0.7272	0.8279	0.9060	0.4494	0.9069	9,635 KB
MLP-2 LR-0.0001	0.4402	0.5692	0.7311	0.8235	0.9058	0.4402	0.8191	4,825 KB
MLP-3 LR-0.0001	0.4223	0.5439	0.7051	0.8144	0.9055	0.4235	0.7330	911 KB
MLP-1 LR-0.001	0.4017	0.5247	0.6844	0.7843	0.8807	0.4102	0.8852	9,635 KB
MLP-2 LR-0.001	0.3741	0.4974	0.6616	0.7748	0.8740	0.3762	0.6694	4,825 KB
MLP-3 LR-0.001	0.3687	0.4911	0.6520	0.7653	0.8717	0.3713	0.8906	911 KB

Table 55: CIFAR-100 MLP Classification Results on CNN-2 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP-1 LR-0.0001	0.6326	0.7549	0.8744	0.9320	0.9683	0.9838	0.9970	1,443 KB
MLP-2 LR-0.0001	0.6265	0.7531	0.8777	0.9354	0.9715	0.9589	0.9825	729 KB
MLP-3 LR-0.0001	0.6067	0.7394	0.8642	0.9276	0.9699	0.9417	0.9694	111 KB
MLP-1 LR-0.001	0.6066	0.7277	0.8549	0.9172	0.9638	0.9430	0.9711	1,443 KB
MLP-2 LR-0.001	0.6027	0.7316	0.8601	0.9220	0.9653	0.9275	0.9643	729 KB
MLP-3 LR-0.001	0.6137	0.7410	0.8602	0.9207	0.9593	0.9557	0.9707	111 KB

Table 56: CIFAR-100 MLP Classification Results on SimpleNet Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP-1 LR-0.0001	0.7048	0.7960	0.8646	0.9069	0.9423	0.9962	0.9980	2,467 KB
MLP-2 LR-0.0001	0.7080	0.8051	0.8817	0.9193	0.9533	0.9968	0.9984	1,241 KB
MLP-3 LR-0.0001	0.7052	0.8034	0.8782	0.9183	0.9525	0.9969	0.9982	211 KB
MLP-1 LR-0.001	0.6927	0.7574	0.8059	0.8312	0.8583	0.9940	0.9957	2,467 KB
MLP-2 LR-0.001	0.6994	0.7767	0.8387	0.8798	0.9150	0.9955	0.9966	1,241 KB
MLP-3 LR-0.001	0.6973	0.7829	0.8498	0.8938	0.9317	0.9948	0.9978	211 KB

Table 57: CIFAR-100 MLP Classification Results on VGG-19 Intermediate Data

Comparing the models from tables 54, 55, 56 and 57, one can observe that some intermediate datasets' test accuracy scores are consistently lower than others. For example, the accuracy scores on CNN-2 intermediate data in table 55 are between 0.3687 (MLP-3, LR-0.001) and 0.4498 (MLP-1, LR-0.0001) on the test dataset, while the accuracy scores on VGG-19 intermediate data in table 57 are between 0.6927 (MLP-1, LR-0.001) and 0.7080 (MLP-2, LR-0.0001) on the test dataset. The VGG-19 network already performed much better in the previous step of creating the intermediate dataset (see table 52). This emphasises the importance of the previously learned image features in

this subsequent step.

Concerning architectures, it is hard to pick a configuration that works best over all datasets. Of the six compared configurations (3 MLP architectures, 2 learning rates), MLP-1 with a learning rate of 0.0001 performs the best on CNN-2 and SimpleNet intermediate data regarding top-1 test accuracy. For the other two intermediate datasets, MLP-2 reaches higher accuracy scores.

- On **CNN-1 Intermediate Data**, the difference between MLP-1, MLP-2 and MLP-3 on validation data accuracy, all trained with learning rates of 0.0001, is not statistically significant at the 5 % significance level (see table A17). Nonetheless, MLP-2 scores the highest top-1 test accuracy score of 0.5630 on CNN-1 intermediate data. Compared to the initially reached top-1 accuracy of the CNN-1 network of 0.5834 from table 52, this network architectures learns almost as well as the initially trained model.

- On **CNN-2 Intermediate Data**, the best scoring model, MLP-1 with a learning rate of 0.0001, scores an accuracy of around 0.4498 on the test set. Comparing MLP-1 to MLP-2 with a learning rate of 0.0001 that reaches an accuracy of 0.4402, the average validation accuracy of MLP-1 is significantly higher at the 5 % significance level (see table A18). The accuracy score from MLP-1 is slightly lower than the accuracy of 0.4750 that CNN-2 reached during the initial training process from table 52. Interestingly for all configurations on CNN-2 intermediate data, the model's validation accuracy is similar to the accuracy on the test set.

- On **SimpleNet Intermediate Data**, MLP-1 with a learning rate of 0.0001 reaches a top-1 accuracy of 0.6326 on the test data. MLP-1 significantly outperforms all other architectures on the average validation accuracy (see table A19). The top-1 test accuracy of 0.6326 is better than the top-1 test accuracy of 0.6099 that SimpleNet reached during the initial training process on CIFAR-100 (see table 52).

- On **VGG-19 Intermediate Data**, the best performing model is MLP-2, trained with a learning rate of 0.0001. This model achieves a top-1 accuracy of 0.7080 on the test data. This score is slightly higher than the initial VGG-19 model's score on CIFAR-100 of 0.7048 from table 52.

Overall, apart from the MLP architectures trained on CNN-2 intermediate data, all models perform much better on both the train and the validation dataset than on the test dataset. The average validation accuracy for CNN-1, SimpleNet, and VGG

intermediate data is in the range of 0.9275[5] up to 0.9969[6], whereas the above-described test scores are lower. Although this indicates that the models overfit on the validation data, this outperformance is not very surprising. The validation data is generated from the data that the respective initial CNN architecture was trained on, which already performed better than the test data in this first step, see section 5.1.2. As can be seen in table 52, because the CNN models trained better on the training data and were not able to generalise all the learned features to the test data, the validation data created from the training data from the first step will inevitably perform better than the test dataset. For CNN-2 intermediate data, a potential explanation for the lack of outperformance on the validation data could be that the initial CNN-2 model already did not learn enough features to reach much higher scores properly. This was not further investigated since it could be clearly seen that the benchmarks on these datasets perform inevitably worse.

To sum up, this section shows that the separation of the learning process to create image features and train fully-connected neural networks on those image features performs comparable to the initial training process of an entire convolutional neural network. While the final top-1 test accuracy was slightly lower on CNN-1 and CNN-2 intermediate data than on the initially trained CNN-1 and CNN-2 models, the contrary was the case for the VGG-19 and SimpleNet intermediate data. Therefore, it can be concluded that a detached final classifier is in general still able to learn as well as the full network in practice. These observations were also confirmed during the other training processes on CIFAR-10 and the ILSVRC-2012 subset.

5.3 Comparing Different Classification Algorithms on Image Features

The following section lists the accuracy values that were achieved in the second step on the intermediate datasets produced on CIFAR-10, CIFAR-100, and ILSVRC-2012. The tables are split by the intermediate datasets. Each table holds a comparison of different classifiers trained on the respective intermediate datasets. Since four CNNs were used on the CIFAR networks to create the intermediate datasets and three CNNs were used on ILSVRC-2012, the number of intermediate datasets benchmarked in this section totals to 11. Each section contains a textual description of the findings, pointing out the key points from the tables to focus on and comparing the performance of the different classifiers. Apart from the top-n test accuracies, the tables list the average validation and average train accuracy achieved over the 10-fold cross-validation. Moreover, the model size of the best performing parameter selection that achieved these

[5] MLP-2 with LR of 0.001 on SimpleNet intermediate data
[6] e.g., for MLP-3 with LR of 0.0001 on VGG-19 intermediate data

results is included each row. The best performing model parameters can be found in Appendices A.3.1.2, A.3.3 and A.3.2.3.

5.3.1 CIFAR-10 Results

The performance results of the classification algorithms on the intermediate datasets produced from CNN-1, CNN-2, SimpleNet and VGG-19 on CIFAR-10 can be found in tables 58, 59, 510 and 511 respectively. The corresponding model configurations for the best performing models of each architecture can be found in tables A1, A2, A3 and A4 in Appendix A.3.1.2. The significance matrices on the validation data benchmarks are presented in tables A9, A10, A11 and A12.

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.8490	0.9367	0.9880	0.9987	0.9994	4,200 KB
SVM	0.8454	0.9321	0.9850	0.9819	1.0000	497,200 KB
LR	0.8469	0.9339	0.9868	0.9998	0.8130	191 KB
KNN	0.7369	0.8503	0.9385	0.8245	0.8691	1,300,000 KB
RFE	0.8120	0.9202	0.9893	0.9254	1.0000	1,990,000 KB
ADB	0.7967	0.9086	0.9833	0.9119	0.9614	843 KB
GBC	0.8175	0.9204	0.9874	0.9722	1.0000	1,200 KB
XGB	0.8182	0.9198	0.9877	0.9527	0.9985	2,400 KB

Table 58: CIFAR-10 Final Classification Results on CNN-1 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.7686	0.8974	0.9851	0.7908	0.8893	4,800 KB
SVM	0.7952	0.9140	0.9869	0.7851	1.0000	724,600 KB
LR	0.7622	0.8933	0.9810	0.7750	0.8130	191 KB
KNN	0.6170	0.6577	0.8050	0.5789	1.0000	1,460,000 KB
RFE	0.6994	0.8496	0.9736	0.6962	1.0000	206,300 KB
ADB	0.6446	0.8306	0.9699	0.6456	0.6616	844 KB
GBC	0.7022	0.8539	0.9735	0.7015	0.7818	1,300 KB
XGB	0.7353	0.8788	0.9790	0.7331	1.0000	22,100 KB

Table 59: CIFAR-10 Final Classification Results on CNN-2 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.8941	0.9591	0.9929	0.9994	0.9997	1,100 KB
SVM	0.8945	0.9605	0.9921	0.9995	1.0000	2,800 KB
LR	0.8806	0.9572	0.9945	1.0000	1.0000	27 KB
KNN	0.8960	0.9366	0.9682	0.9982	0.9987	162,900 KB
RFE	0.8933	0.9632	0.9958	0.9986	1.0000	221,600 KB
ADB	0.8766	0.9472	0.9916	0.9909	0.9950	843 KB
GBC	0.8907	0.9594	0.9937	0.9982	1.0000	1,300 KB
XGB	0.8881	0.9586	0.9940	0.9971	1.0000	1,500 KB

Table 510: CIFAR-10 Final Classification Results on SimpleNet Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.9352	0.9696	0.9835	0.9999	0.9999	1,100 KB
SVM	0.9349	0.9797	0.9952	0.9999	0.9999	3,500 KB
LR	0.9346	0.9772	0.9932	0.9999	0.9999	46 KB
KNN	0.9350	0.9541	0.9757	0.9999	0.9999	324,800 KB
RFE	0.9350	0.9661	0.9864	0.9999	1.0000	868 KB
ADB	0.9330	0.9659	0.9927	0.9998	1.0000	844 KB
GBC	0.9318	0.9523	0.9803	0.9996	1.0000	1,100 KB
XGB	0.9327	0.9683	0.9890	0.9998	1.0000	273 KB

Table 511: CIFAR-10 Final Classification Results on VGG-19 Intermediate Data

The results on CNN-1 intermediate data can be found in table 58. Firstly, the separately trained fully-connected neural networks perform comparably to the initially trained CNN-1 from table 51, reaching a final accuracy around 0.85 on test data. Moreover, other network structures can learn similarly well on the data. SVM and LR achieve a top-1 test accuracy of 0.8454 and 0.8469 respectively. From the tree-based methods, both GBC and XGB reach a top-1 test accuracy around 0.82, while RFE reaches a top-1 accuracy of 0.8120 and ADB reaches a top-1 accuracy of 0.7967 on test data. KNN's top-1 test accuracy on the CNN-1 intermediate data is only at 0.7369. For the best performing models, the average validation accuracy is very close to 1.00 while the gap to the top-1 test accuracy is rather big with up to 15 percentage points, e.g., for MLP and LR. The top-2 and top-5 accuracies in table 58 are highly correlated with the top-1 accuracy on test data with a Pearson correlation coefficient ρ of 0.9781 and 0.8634 respectively. The produced model sizes vary largely between the classification models. LR produces the smallest model with 191 KB and RFE produces the largest model of almost 2 GB. MLP has the fifth largest model after LR and the three boosting

approaches.

For the CNN-2 intermediate data, the benchmark top-1 accuracy of 0.8113 on test data from table 51 could not be reached by any of the models from table 59. Nonetheless, an SVM configuration ($C = 10$, $\gamma = 0.001$, RBF kernel) reaches the highest top-1 test accuracy of 0.7952. The benchmark MLP has a test accuracy of only 0.7686. The other models' accuracy scores on test are mostly between 0.6994 for RFE up to 0.7622 for the linear regression model. The XGB algorithm performs better than the other boosting algorithms but still only manages to reach a test accuracy of 0.7353. The worst performing model is KNN, which reaches a top-1 test accuracy of 0.6170. Interestingly, none of the models seem to overfit much on the training data. For all models, the gap between the validation and the test accuracies is quite small. Regarding top-2 and top-5 test accuracies in table 59, the values follow the same order as the top-1 accuracy. E.g., the best performing model, SVM, also has the highest top-2 test accuracy of 0.9140 and the highest top-5 test accuracy of 0.9869. Regarding model size, LR produces the smallest model again, followed by the three boosting approaches and MLP in fifth place. The largest model comes from the KNN model with 1,460,000 KB.

The benchmarked models on SimpleNet intermediate data in table 510 all perform comparable or outperform the initial benchmark of 0.8852 on CIFAR-10 from table 51. While the MLP benchmark architecture performs well with a top-1 accuracy of 0.8941 on test data, KNN even reaches a slightly higher top-1 accuracy of 0.8960 on the test set. On the validation set, LR is the best performer, but its top-1 test accuracy only reaches 0.8806. Apart from MLP, the models SVM, RFE, GBC, and XGB all perform comparably to the MLP benchmark with accuracies around 0.89. ADB performs worse than the rest with a test accuracy of 0.8766. Apart from minor variations in the validation performance, this indicates that all these models perform similarly to an MLP. Some models even show slightly better performances than the benchmarked MLP model. Regarding top-2 and top-5 test accuracy, RFE produces the highest values with 0.9632 and 0.9958 respectively. All top-5 test accuracy values lie very closely, apart from the value of KNN. On top-2 test accuracy, RFE is followed by SVM with a score of 0.9605. The MLP (0.9591) model only follows in fourth place after GBC (0.9594). The produced model sizes are rather small in comparison to the other intermediate datasets from CIFAR-10. The LR model only takes up 27 KB. MLP is the third smallest model with 1,100 KB. The largest model, RFE, has 221,600 KB.

VGG-19 originally scored a top-1 test accuracy of 0.9359 on CIFAR-10 (see table 51)[7]. The best performing models from table 511 perform similarly to this benchmark result regarding top-1 accuracy, including MLP (0.9325), RFE (0.9350), KNN (0.9350), SVM

[7]The test accuracy is identical to the train accuracy because an external model was used for the benchmark

(0.9349) and LR (0.9346). But even the remaining three models score top-1 accuracies above 0.93, meaning the differences between the models' performances are very small. This indicates that the VGG network's performance is less reliant on the final classification layer and more focused on the CNN's internal structure. Interestingly, on the top-2 and top-5 test accuracy values, SVM reaches the best results with 0.9797 and 0.9952 respectively. These accuracy values are more than one percentage point higher than the respective MLP values. In fact, the MLP model only has the third highest top-2 test accuracy after SVM and LR (0.9772) and even has the third lowest top-5 test accuracy, only outperforming KNN and GBC. The LR model has a size of 46 KB. Interestingly, the RFE model is small compared to the other intermediate datasets with only 868 KB. The largest model is produced by KNN with 324,800 KB.

To summarise the observations from the benchmark on the intermediate datasets trained on CIFAR-10, most intermediate datasets' models were not able to consistently produce better values than the initially benchmarked CNN models. Only for SimpleNet intermediate data, six of the eight benchmarked models score better than the SimpleNet benchmark score. For most intermediate datasets, the best top-1 accuracy values are similar to the benchmarked values. Apart from the retrained MLP models, the models from SVM and LR frequently appear among the best performing models. LR has the advantage of petite model sizes. Although SVM outperforms MLP at times, the model sizes of SVMs have a high variance. E.g., the best performing model on SimpleNet intermediate data is an SVM model of 2,800 KB size, while the best performing model on CNN-2 intermediate data is an SVM model of size 724,600 KB. Considering model size and performance, MLP and LR perform best. Regarding the tree-based models, ADB performs the worst among these. Overall, XGB manages to produce slightly better results than GBC, particularly on CNN-2 intermediate data. Nonetheless, GBC is remarkably constant in model size between 1,100 and 1,300 KB, whereas XGB fluctuates more. Between RFE and XGB it is hard to choose of whether the bagging or the boosting approach performed better overall. While RFE reaches slightly higher top-1 test accuracies on VGG-19, SimpleNet, and CNN-1 intermediate data, XGB performs much better on CNN-2 intermediate data. Nonetheless, all models produced from boosting algorithms are much smaller in size than RFE. The KNN model has some interesting behaviour: While it performs comparably to the best model on VGG-19 and outperforms all other models on SimpleNet, its performance on CNN-1 and CNN-2 is much lower than the other benchmarked models. While the CNN-2 intermediate data uses a model with only 1 neighbour, all other intermediate datasets use models with 10 neighbours. Apart from these observations, KNN generally produces huge models, sometimes even over 1 GB in size, e.g., on CNN-1 and CNN-2.

5.3.2 CIFAR-100 Results

The results of the classification algorithms on the intermediate datasets from CNN-1, CNN-2, SimpleNet and VGG-19 on CIFAR-100 can be found in tables 512, 513, 514 and 515 respectively. The corresponding model configurations for the best performing models of each architecture can be found in tables A13, A14, A15 and A16 in Appendix A.3.2.3. The significance matrices on the validation data benchmarks are presented in tables A21, A22, A23 and A24.

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.5630	0.6860	0.8148	0.8874	0.9422	0.9884	0.9997	4,313 KB
SVM	0.6030	0.7250	0.8515	0.9146	0.9623	0.9820	0.9998	791,833 KB
LR	0.5706	0.6797	0.7994	0.8705	0.9269	0.9935	0.9996	1,605 KB
KNN	0.3284	0.3320	0.3547	0.3929	0.4694	0.3769	0.9998	1,266,158 KB
RFE	0.3192	0.4173	0.5485	0.6525	0.7514	0.3576	0.9998	3,223,027 KB
ADB	0.2042	0.3058	0.4851	0.6325	0.7876	0.2482	0.2758	335 KB
GBC	0.2547	0.3602	0.5422	0.6111	0.7493	0.4369	0.9994	12,150 KB
XGB	0.3326	0.4397	0.5692	0.6801	0.8115	0.4721	0.9996	4,100 KB

Table 512: CIFAR-100 Final Classification Results on CNN-1 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.4498	0.5733	0.7272	0.8279	0.9060	0.4494	0.9988	9635 KB
SVM	0.4741	0.6020	0.7554	0.8499	0.9242	0.4576	0.9770	890827 KB
LR	0.4518	0.5702	0.7266	0.8281	0.9089	0.4417	0.6336	1805 KB
KNN	0.2878	0.2912	0.3139	0.3538	0.4308	0.2666	0.9998	1424346 KB
RFE	0.2931	0.3822	0.5211	0.6243	0.7298	0.2793	0.9998	3518103 KB
ADB	0.1728	0.2682	0.4336	0.5793	0.7313	0.1732	0.1845	335 KB
GBC	0.2880	0.3402	0.5041	0.6034	0.7021	0.3044	0.9555	12150 KB
XGB	0.3009	0.3798	0.5386	0.6403	0.7567	0.3518	0.9993	3400 KB

Table 513: CIFAR-100 Final Classification Results on CNN-2 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.6326	0.7549	0.8744	0.9320	0.9683	0.9838	0.9970	1443 KB
SVM	0.6380	0.7615	0.8826	0.9419	0.9772	0.9801	0.9989	80543 KB
LR	0.6247	0.7439	0.8554	0.9141	0.9543	0.9937	0.9997	205 KB
KNN	0.6066	0.7286	0.8237	0.8465	0.8646	0.8380	0.8904	158842 KB
RFE	0.5955	0.7216	0.8436	0.9029	0.9460	0.8258	0.9998	2049021 KB
ADB	0.3850	0.5275	0.7259	0.8408	0.9261	0.5165	0.5377	335 KB
GBC	0.5532	0.6802	0.7947	0.8834	0.9339	0.8812	0.9995	12150 KB
XGB	0.5894	0.7107	0.8473	0.9132	0.9450	0.9683	0.9996	2200 KB

Table 514: CIFAR-100 Final Classification Results on SimpleNet Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.7080	0.8051	0.8817	0.9193	0.9533	0.9968	0.9984	1241 KB
SVM	0.7114	0.8113	0.8950	0.9336	0.9651	0.9972	0.9978	32203 KB
LR	0.7071	0.8056	0.8768	0.9133	0.9568	0.9969	0.9983	406 KB
KNN	0.7095	0.7644	0.7892	0.8016	0.8262	0.9970	0.9972	317030 KB
RFE	0.7095	0.8086	0.8886	0.9266	0.9568	0.9969	0.9999	2397625 KB
ADB	0.6629	0.7505	0.8453	0.8939	0.9403	0.9854	0.9897	335 KB
GBC	0.6747	0.7502	0.8147	0.8534	0.8939	0.9911	0.9999	12150 KB
XGB	0.6897	0.7707	0.8423	0.8851	0.9260	0.9948	0.9999	4787 KB

Table 515: CIFAR-100 Final Classification Results on VGG-19 Intermediate Data

A comparison of the models trained on the intermediate data from CNN-1 being trained on CIFAR-100 can be found in table 512. Two models perform better on test accuracy than the trained MLP model: SVM and LR. The best performing SVM model configuration uses an RBF kernel and parameters C and γ of 10 and 0.0001 respectively. It reaches a top-1 test accuracy of 0.6030 which is higher than the initially benchmarked accuracy of 0.5834 of CNN-1 on CIFAR-100 from table 52. The LR model reaches a top-1 test accuracy of 0.5706, and the retrained MLP model achieves a top-1 test accuracy of 0.5630. Both accuracy values are slightly worse than the one of the initially benchmarked CNN-1 from table 53. MLP, SVM, and LR follow the same order for top-2, top-5, top-10, and top-20 test accuracy. SVM reaches a top-5 test accuracy of 0.8515, LR reaches one of 0.7994 and MLP achieves a top-5 test accuracy of 0.8148. On the downside, SVM also produces the largest model with 791,833 KB. The MLP and LR models are much smaller with 4,313 KB and 1,605 KB respectively. MLP, SVM, and LR have a small gap between train and validation top-1 accuracy, which shows that the algorithms can reproduce the non-linear combinations from the first step.

Nonetheless, the numbers from the test accuracy described above are lower than the validation and train accuracy values. This is caused by the fact that the validation data was part of the train dataset during the creation of the intermediate dataset. KNN and all ensemble models perform much worse than the other algorithms. XGBoost reaches a top-1 test accuracy of 0.3326, which is 23 percentage points worse than MLP. The worst performing model, ADB, only achieves a top-1 test accuracy of 0.2042. GBC reaches a top-1 test accuracy of 0.2547 and KNN reaches one of 0.3284. Interestingly, converge on the training data, apart from ADB. The largest model in the comparison is produced by RFE. The final RFE model has a size of 3,223,027 KB.

The benchmarked performances on CNN-2 intermediate data from CIFAR-100 are displayed in table 513. As for CNN-1 intermediate data, SVM (0.4741) and LR (0.4518) perform better than the benchmarked MLP architecture (0.4498). SVM's performance of 0.4741 is also similar to the initially benchmarked performance of CNN-2 on CIFAR-100 of 0.4750 from table 52. As for CNN-1, the best performing SVM model on CNN-2 is configured with an RBF kernel, a parameter C of 10 and a γ value of 0.0001. The KNN and ensemble models reach top-1 accuracy values between 0.3009 for XGB and 0.1728 for ADB. GBC reaches a top-1 test accuracy of 0.2880 and RFE reaches a top-1 test accuracy of 0.2931. The top-n accuracies for higher values of n mainly follow the trend of the previously described models. SVM's top-5 accuracy is 0.7554, followed by MLP with a value of 0.7272 and LR with 0.7266. Only the top-n-accuracy of KNN seems to be worse for higher values of n than for the other models. Despite having the best accuracy values, the SVM model is much larger than LR and MLP. While the LR model only takes up 1,805 KB and MLP takes up 9,635 KB, the SVM model has a size of 890,827 KB. Among the other models, KNN is the largest with 1,424,346 KB. Most model's top-1 train accuracy reach values close to 1, meaning these models overfit on the training data. Only ADB reaches a much lower top-1 train accuracy of 0.5377 and top-1 validation accuracy of 0.5165. For MLP, SVM, and LR, the differences between train and validation top-1 accuracies are not that large, meaning these models did not overfit. Nonetheless, the difference to the test data is larger. This gap is caused by the fact that the validation data was part of the train data in the first step of creating the high-level image features for the intermediate dataset.

An overview of the performances of the classification models on SimpleNet intermediate data from CIFAR-100 is given in table 514. Contrarily to CNN-1 and CNN-2, the top-1 accuracy difference between most models is not that big. Apart from ADB, all models perform between a top-1 test accuracy of 0.5532 (GBC) and 0.6380 (SVM), meaning a total gap of around 8 percentage points. ADB on the other hand only reaches a top-1 test accuracy of 0.3850. SVM, MLP and LR all outperform the initial SimpleNet benchmark on CIFAR-100 of 0.6099 from table 514 with top-1 test accuracy values of 0.6380, 0.6326 and 0.6247 respectively. KNN performs similarly to the

benchmark with a top-1 test accuracy of 0.6066. From the ensemble models, RFE has the best top-1 test accuracy value of 0.5955 followed by XGB with a value of 0.5894. The GBC model reaches a slightly lower top-1 accuracy of 0.5532. The top-n accuracy scores for higher values of n behave similarly to the top-1 accuracies. SVM reaches a top-5 accuracy of 0.8826, followed by MLP with a value of 0.8744 and LR with a top-5 accuracy of 0.8554. As for CNN-1 and CNN-2 intermediate data, the KNN model's top-n accuracy for higher values of n gets worse in comparison to the other benchmarked models. E.g., on top-20 accuracy, KNN has the lowest value of all with 0.8646, which is about 6 percentage points lower than the second worst. The largest model on this intermediate data benchmark is RFE with a size of 2,049,021 KB. This is by far the largest model, followed by KNN with a size of 158,842 KB and SVM with a size of 80,543 KB. Despite its good performance, the SVM model is much larger than the MLP and LR models that are of sizes 1,443 KB and 205 KB respectively.

The performance of the different classification models on the intermediate dataset produced from VGG-19 on CIFAR-100 can be found in table 515. On VGG-19 intermediate data from CIFAR-100, five out of eight models can outperform the benchmarked VGG-19 score of 0.7048 on CIFAR-100 from table 52. Apart from the MLP model that scores a top-1 test accuracy of 0.7080, three models even reach higher top-1 accuracies. Among those, SVM achieves a top-1 test accuracy of 0.7114, while KNN and RFE both reach a value of 0.7095. The LR model only scores slightly worse than MLP with a top-1 test accuracy of 0.7071. All boosted models, ADB, GBC, and XGB, perform worse than the aforementioned models on top-1 test accuracy, ranging between 0.6629 (ADB) to 0.6897 (XGB). For all the models, the developments on top-2, top-5, top-10, and top-20 test accuracy mostly remain similar and in the same order as the one of the top-1 test accuracy. SVM has a top-5 accuracy of 0.9337, followed by RFE with a value of 0.9266, MLP with 0.9193 and LR with 0.9133. Only the KNN model seemingly performs much worse on these top-n accuracy measures in comparison to its top-1 accuracy, with its top-5, top-10 and top-20 test accuracies even being the lowest in the comparison of all models. The three best performing models are also the largest models in the benchmark: RFE has a size of 2,397,625 KB, KNN has a size of 2,397,625 KB and SVM is of size 32,203 KB. In comparison, the smallest models are LR and ADB with sizes of 406 KB and 335 KB respectively.

To sum up the findings, the best performing model on all intermediate datasets from CIFAR-100 is SVM. After that, mostly LR and MLP follow with LR being superior on CNN-1 and CNN-2 intermediate data and MLP performing better on SimpleNet and VGG-19 intermediate data. SVM also outperforms the top-1 accuracy of the CNN benchmarks from the first step (table 52) on CNN-1, SimpleNet, and VGG-19 intermediate data. On SimpleNet and VGG-19 intermediate data, the retrained MLP and the LR model are also superior to the initial benchmark in performance. Considering the

external benchmarks on CIFAR-100 from table 42, the achieved top-1 test accuracy scores on VGG-19 intermediate data from table 515 rank among the best available model benchmarks on CIFAR-100.

The bagging and boosting approaches perform similarly concerning top-1 accuracy, but their performance is generally worse than the other benchmarked algorithms. Similar performance to the other classification algorithms is only achieved by RFE on VGG-19 intermediate data. Among the boosting algorithms, XGB is the best algorithms on all intermediate datasets. ADB has the lowest top-1 test accuracy among all classifiers on every intermediate dataset. Regarding model size, RFE produces much larger models than all other models, with the largest model being of size 3.5 GB from CNN-2 intermediate data. The boosting algorithms generally produce rather small models. The smallest in each intermediate dataset benchmark is either ADB or LR. The latter additionally has good performance values despite being very small in size of up to 7 times smaller than MLP (e.g., see table 514). The top-5 accuracy scores generally follow the same trend as the top-1 accuracy scores. SVM has the highest top-5 accuracy on every intermediate dataset. Interestingly, MLP performs better than LR in top-5 accuracy on every dataset, although the same is not always the case for top-1 accuracy. For larger values of n, the top-n accuracy of KNN gets worse in comparison to other models.

5.3.3 ILSVRC-2012 Results

The results of the classification algorithms on the intermediate datasets from Inception ResNet V2, Inception V3 and Xception on the ILSVRC-2012 subset can be found in tables 516, 517 and 518 respectively. The corresponding model configurations for the best performing models of each architecture can be found in tables A25, A26 and A27 in Appendix A.3.3. The significance matrices on the validation data benchmarks are presented in tables A28, A29 and A30.

Since the original models for Inception V3, Xception and Inception ResNet were trained on 1000 classes, there is no benchmark for the CNN performance on the ILSVRC-2012 dataset with 100 classes as used in this thesis. Therefore, the other classification models can only be compared to the respective MLP models for the intermediate datasets.

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.7936	0.8984	0.9480	0.9672	0.9808	0.7760	0.8920	537,300 KB
SVM	0.3552	0.6184	0.6576	0.7000	0.7576	0.3168	0.9997	1,204,000 KB
LR	0.8512	0.9304	0.9768	0.9920	0.9944	0.8459	0.9997	104,900 KB
KNN	0.7608	0.7680	0.7792	0.7904	0.8064	0.7509	0.9997	4,580,000 KB
RFE	0.8256	0.9176	0.9656	0.9792	0.9880	0.8157	0.9997	1,780,000 KB
ADB	0.2024	0.2856	0.4360	0.5560	0.7048	0.1341	0.3835	3,800 KB
GBC	0.4152	0.5488	0.6912	0.7448	0.7816	0.4515	0.9997	6,100 KB
XGB	0.7888	0.8848	0.9336	0.9560	0.9752	0.7461	0.9973	4,700 KB

Table 516: ILSVRC-2012 Final Classification Results on Inception V3 Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.8208	0.9128	0.9488	0.9664	0.9800	0.8245	0.9136	419,700 KB
SVM	0.7608	0.8800	0.9216	0.9512	0.9696	0.7483	0.9997	1,530,000 KB
LR	0.8528	0.9336	0.9792	0.9936	0.9976	0.8528	0.9976	163,800 KB
KNN	0.8296	0.9152	0.9464	0.9552	0.9592	0.8184	0.8667	5,921,000 KB
RFE	0.8496	0.9336	0.9776	0.9920	0.9960	0.8429	0.9997	1,640,000 KB
ADB	0.3552	0.4912	0.6360	0.7304	0.8352	0.1874	0.5829	3,800 KB
GBC	0.3760	0.5008	0.6496	0.7000	0.7536	0.3885	0.9992	6,300 KB
XGB	0.7888	0.8912	0.9416	0.9568	0.9728	0.7744	0.9987	6,000 KB

Table 517: ILSVRC-2012 Final Classification Results on Xception Intermediate Data

Model	Top-1 Test Accuracy	Top-2 Test Accuracy	Top-5 Test Accuracy	Top-10 Test Accuracy	Top-20 Test Accuracy	Average Validation Accuracy	Average Train Accuracy	Model Size
MLP	0.8424	0.9248	0.9568	0.9704	0.9816	0.8459	0.9211	201,500 KB
SVM	0.7360	0.8184	0.8512	0.8736	0.9136	0.7334	0.9997	984,000 KB
LR	0.8712	0.9472	0.9768	0.9960	0.9992	0.8632	0.9957	70,100 KB
KNN	0.8504	0.9344	0.9528	0.9568	0.9600	0.8451	0.8816	78,600 KB
RFE	0.8536	0.9384	0.9792	0.9856	0.9896	0.8629	0.9997	1,530,000 KB
ADB	0.4296	0.5512	0.6720	0.7544	0.8256	0.2347	0.5965	3,800 KB
GBC	0.5056	0.5976	0.6776	0.7192	0.7776	0.4549	0.9997	6,200 KB
XGB	0.8216	0.9056	0.9456	0.9608	0.9728	0.8117	0.9984	4,200 KB

Table 518: ILSVRC-2012 Final Classification Results on Inception ResNet V2 Intermediate Data

The benchmarked performances of the different classification algorithms on the InceptionV3 intermediate dataset from ILSVRC-2012 is shown in table 516. The performance of the classification models varies between a top-1 accuracy of 0.2024 for ADB and 0.8512 for LR. Two models outperform the MLP top-1 test accuracy benchmark of 0.7936: LR reaches a top-1 test accuracy score of 0.8512 and RFE reaches a score of 0.8256. XGB performs almost as good as MLP with a top-1 test accuracy of 0.7888, followed by KNN with a top-1 accuracy of 0.7608. The top-1 accuracy of GBC is 0.4152 and SVM reaches a score of 0.3552. The smallest models are produced by the boosting algorithms ADB, GBC, and XGB with sizes of 3,800 KB, 6,100 KB and 4,700 KB respectively. The best performing model, LR, has a size of 104,900 KB, which is five times smaller than the MLP model with a size of 537,300 KB. The largest model is produced by KNN with 4,580,000 KB. The RFE model that performed the second best in the benchmark has a size of 1,780,000 KB. On top-5 test accuracy, LR and RFE still perform much better than the compared MLP model. The top-5 test accuracy of LR is 0.9768, and for RFE it is 0.9656. The MLP model's top-5 test accuracy is 0.9480.

The results for the benchmark on Xception intermediate data trained on the ILSVRC-2012 subset is displayed in table 517. Three models score better top-1 test accuracy values than the MLP model: LR with a top-1 test accuracy of 0.8528, RFE with a top-1 test accuracy of 0.8496 and KNN with a top-1 test accuracy of 0.8296. The LR and the RFE models' top-1 test accuracies are about 3 percentage points higher than the top-1 test accuracy from the MLP model. XGB managed to get a top-1 test accuracy of 0.7888 and SVM reaches one of 0.7608. The other boosting models, ADB and GBC, are much worse with values of 0.3552 and 0.3760 respectively. The delta between LR, RFE and MLP stays similar for top-5 test accuracy with LR reaching a score of 0.9792 and RFE reaching a score of 0.9776, while MLP reaches a score of 0.9488. The top-5 accuracy of XGB of 0.9416 is relatively close to the one from MLP. The largest model in the benchmark is the KNN model with a total size of 5,921,000 KB, followed by the RFE model with 1,640,000 KB and the SVM model with 1,530,000 KB. The remaining models are much smaller. The best performing LR has 163,800 KB. The MLP model is of size 419,700 KB. The smallest model is produced from ADB with a size of 3,800 KB. The GBC and XGB models are similar in size with 6,300 KB and 6,000 KB respectively.

The benchmark on Inception ResNet V2 trained on the ILSVRC-2012 subset is presented in table 518. LR, KNN, and RFE all perform better on top-1 and top-2 accuracy than MLP. LR and RFE also outperform MLP on all other top-n accuracy values. KNN performs worse for top-n accuracies with larger values of n. LR reaches the highest top-1 test accuracy of 0.8712. This is also the highest among all benchmarked intermediate datasets trained on the ILSVRC-2012 subset. RFE has a top-1 accuracy of 0.8536 and KNN has one of 0.8504. MLP follows with 0.8424. The next highest performing model regarding top-1 accuracy is XGB with a value of 0.8216. SVM reaches a top-1

test accuracy of 0.7360. The remaining boosting algorithms, ADB and GBC, perform much worse with top-1 accuracy values of 0.4296 and 0.5056 respectively. On top-5 test accuracy, the best performing model is RFE although its value of 0.9792 is only slightly higher than the one from LR of 0.9768. MLP reaches a top-5 test accuracy of 0.9568. The KNN model performs worse than MLP on top-5 accuracy with a score of 0.9528. XGB manages to reach a top-5 accuracy of 0.9456. The largest model is produced by RFE with a size of 1,530,000 KB, followed by SVM with 984,000 KB. The other models are much smaller. The next biggest model is MLP with a total size of 201,500 KB. KNN has a size of 78,600 KB and LR has a size of 70,100 KB. The boosting models are much smaller in size with ADB having 3,800 KB, GBC having 6,200 KB and XGB having 4,200 KB.

On ILSVRC-2012 intermediate datasets, LR constantly reaches the highest accuracy results among the benchmarked models. LR outperforms MLP by about 3 percentage points top-1 accuracy or more on all ILSVRC-2012 intermediate datasets. The same observation goes for the top-5 accuracy, although the minimal difference between MLP and LR is 2 percentage points on this measure in table 518. Moreover, the LR models are much smaller than the corresponding MLP models, thus enabling a reduction of size and better performance. Apart from LR, the RFE model also manages to outperform MLP but with much larger model sizes. E.g., RFE reaches an accuracy of 0.8536 on ResNet V2 intermediate data with a model of 1,530,000 KB size. The corresponding MLP model is more than seven times smaller at 201,500 KB. The performances of KNN models vary. On Inception ResNet V2 and Xception intermediate data, KNN is the third best model on top-1 test accuracy, outperforming MLP. Interestingly, the performance on other metrics such as top-5 test accuracy is not as good for KNN as for other models. KNN models do not improve that much between top-1 and higher values of n. To illustrate this, on Inception V3 in table 516, KNN performs almost 10 percentage points better than ADB regarding top-1 test accuracy with a score of 0.76608 for KNN and 0.6629 for ADB. On top-10 test accuracy, ADB performs about 10 percentage points better than KNN with a performance of 0.8938 versus 0.7904 respectively. The same pattern can be observed between other models and KNN and on other intermediate datasets. On every ILSVRC-2012 intermediate dataset, bagging manages to reach at least 3 percentage points higher accuracy values than any of the boosting algorithms. Among the boosting approaches, XGB is consistently proficient to ADB and GBC, outperforming the latter by more than 30 percentage points. GBC still consistently performs better than ADB on all datasets. The ADB model's top-1 test accuracy was less than half of the score of the best performing model (LR) on every dataset. On all datasets, the accuracy on validation data is similar to the test data. This shows that the image filters from the benchmarked CNNs are not overfitted on the validation dataset.

5.4 Interpretation of Results

The following section provides an interpretation of the previously observed and described results from section 5.3. This section concludes which classification algorithms perform the best on high-level image features. The chapter further analyses under which conditions specific classification algorithms outperform MLPs and can be considered superior to the traditional approach. Lastly, the chapter reveals which initial and intermediate datasets are more suitable for testing different classification algorithms. The goal of this section is to derive a guideline of when to test which classification algorithms on high-level image features, based on the observations from section 5.3. The evaluation is based on the top-n test accuracy values and the model sizes[8].

First of all, section 5.2 proves that separately training the fully-connected layers does not worsen the overall results. Neither did it lead to an improvement in classification accuracy. It follows that the performance during the separated learning process is only dependant on the chosen classifier and is not worse than a unified learning procedure. Among the benchmark classifiers, the findings in section 5.3 reveal a set of pareto-optimal algorithms over all datasets. The pareto-optimum is derived with respect to top-1 test accuracy and model size. Top-5 test accuracy is not used to build this set because some models randomly reach slightly higher top-5 accuracy values. E.g., GBC would only be pareto-optimal on CNN-1 intermediate data from CIFAR-10 because of a marginally higher top-5 accuracy than LR. This would make the set of pareto-optimal classifiers noisier.

On CIFAR-10 intermediate datasets, the set of pareto-optimal classifiers is MLP, LR, SVM, and RFE. On CIFAR-100 intermediate datasets, the set of classifiers is SVM, LR, MLP, and ADB and on the ILSVRC-2012 intermediate datasets, it is LR, ADB and XGB.

The following analysis of classification algorithms will focus on the union of these pareto-optimal sets: LR, SVM, MLP, XGB, RFE, and KNN. These algorithms have unique characteristics that can be reasons for applying them to high-level image features. ADB is discarded from the comparison because it has consistently very bad accuracy scores. ADB is only pareto-optimal on some intermediate datasets because of its small model sizes, but ADB consistently underperformed on top-n accuracy scores on all datasets. The following unique features can be identified among the models:

[8] The model size depends on the choice of parameters and is therefore only an indicator of the general complexity of a given classification algorithm. As can be seen in the observations from the previous section, for some classification algorithms, the model size has a high variance depending on the chosen parameters and the dataset.

- **Logistic Regression:** LR produces very small models that usually have very high and consistent top-n accuracy values
- **Support Vector Machines:** The overall top-n accuracy of SVM varies. On certain intermediate datasets, SVM is very good. I.e., SVM is the best classifier on all intermediate datasets from CIFAR-100. On the other hand, SVM seems to perform worse for datasets with more input features, such as the intermediate datasets from ILSVRC-2012. Additionally, the produced models are up to 100 times larger than the ones from MLP.
- **Multi-Layer Perceptron:** MLP consistently has solid performances among the top performing models with moderate model sizes. On some benchmarks, MLP even outperforms all other models on top-1 accuracy, e.g., see table 58. An LR model can, in theory, be reproduced by an MLP model. Nonetheless, LR models can be advantageous because they have fewer parameters to learn. Although this takes away complexity and limits the ability to detect more non-linear structures, this can be positive for fewer training samples.
- **eXtreme Gradient Boosting:** XGB produces reasonable top-n accuracies at small model sizes. In most cases, both top-n accuracy and model size are worse than LR, but XGB manages to perform better on some intermediate datasets. Anyhow, XGB can be proficient on very many input features and output classes, because the produced model is still relatively small. E.g., XGB keeps the model size very small on ILSVRC-2012 intermediate datasets while having reasonable accuracy scores.
- **Random Forests Estimator:** The benchmark performance of RFE varies. The only benchmark run where RFE is pareto-optimal is on the VGG-19 intermediate dataset from CIFAR-10, and this superiority is caused by the small model size. Although a small model size is an atypical characteristic of an RFE model, RFE was still included in this set of optimal classifiers, because it can be superior to MLP in many cases, e.g., on all ILSVRC-2012 intermediate benchmarks.
- **K-Nearest Neighbours:** KNN produces very good top-1 accuracies on some benchmark datasets. Nonetheless, the top-n-accuracies for higher values of n are much worse than the compared classification algorithms. This indicates that the KNN model has more trouble in detecting similarities among images because similar images should be more likely to be included in the top-n accuracy scores. KNN typically produces very large models.

Regarding model properties, the 11 benchmarked models differentiated between their number of input features and output classes. While CIFAR-10 produces 10 output classes, CIFAR-100 and the sub-sample of ILSVRC-2012 predict among 100 possible

classes. For CIFAR-10 and CIFAR-100, the intermediate datasets had comparably few input features. SimpleNet intermediate data has the lowest number of input features with 256 features, followed by VGG-19 intermediate data with 512 features. CNN-1 intermediate data has 2,048 input features, and CNN-2 intermediate data has 2,304. In comparison, the intermediate datasets from ILSVRC-2012 have much more input features. Inception ResNet V2 intermediate data has 98,304 input features, Inception V3 intermediate dataset has 131,072 input features, and Xception intermediate dataset has 204,800 input features. This gives four intermediate datasets with relatively few input features and few output classes (CIFAR-10 intermediate datasets), four intermediate datasets with relatively few input features and many output classes (CIFAR-100 intermediate data) and three intermediate datasets with relatively many input features and many output classes (ILSVRC-2012 intermediate datasets).

- **CIFAR-10 Intermediate Dataset:** On CIFAR-10 intermediate datasets, the initial CNN model's top-1 test accuracy was only outperformed on the SimpleNet intermediate dataset. Nonetheless, the differences between the MLP and the outperforming models are rather small with MLP only performing 0.2 % worse than the best model (KNN). Apart from the SimpleNet intermediate dataset, the retrained MLP benchmark is only outperformed by SVM on CNN-2.
- **CIFAR-100 Intermediate Dataset:** On CIFAR-100 intermediate datasets, the CNN model's top-1 test accuracy is beaten on every intermediate dataset, apart from CNN-2. Many models outperform the CNN benchmark on SimpleNet and VGG-19 intermediate dataset, including MLP, SVM, LR, KNN, and RFE. The MLP model is outperformed on every intermediate dataset, although the differences on SimpleNet and VGG-19 intermediate datasets are comparably smaller. In general, the accuracy values on those intermediate datasets were closer together. SVM has the highest top-1 test accuracy on every intermediate dataset.
- **ILSVRC-2012 Intermediate Dataset:** On ILSVRC-2012 intermediate datasets, LR and RFE outperform the MLP benchmark on every dataset while LR still performs better than RFE. Additionally, KNN outperforms MLP on top-1 test accuracy on the Xception and Inception ResNet V2 intermediate datasets. SVM, on the other hand, performs much worse on all intermediate datasets in comparison to its previous performance on CIFAR-10 and CIFAR-100 intermediate datasets.

From the above described, one can derive a guideline to choose the best classification algorithms, depending on the intermediate dataset attributes. For intermediate datasets with few high-level image features as inputs and few output classes, like CIFAR-10 intermediate datasets, MLP seems to be proficient or equal to the other

benchmarked algorithms. LR is a good choice if the model size needs to be kept small. On intermediate datasets with few output classes and relatively many input features, like the CNN-2 and CNN-1 intermediate datasets, SVM is likely to perform better than MLP.

For datasets with few input features and many output classes, like CIFAR-100 intermediate datasets, it is advisable to use other classification algorithms than MLP. The best choice of model is SVM, although this comes at the cost of large model sizes. For the models with fewer input features (SimpleNet, VGG-19), the choice of the final classifier does not have such a big impact on top-n accuracy. On the other hand, the effect of the model choice on the final performance is larger for intermediate datasets with more input features (CNN-1, CNN-2). The superior performance of the benchmarked classification algorithms over MLP can also be caused by the fewer training samples of 600 samples per class compared to 6000 samples per class for CIFAR-10. E.g., SVMs generally learn well on small datasets.

For datasets with many input features and many output classes, like ILSVRC-2012 intermediate datasets, more proficient models than MLP are available. The LR and RFE models consistently outperform MLP in the benchmark and can thus be considered superior for these datasets. LR has the additional advantage of small model sizes whereas RFE should not be used if model complexity is an issue. But even some of the other benchmarked models, including KNN and XGB, should be considered to replace MLPs as they can be advantageous.

Overall, other classification algorithms are usually more proficient than MLP if the intermediate dataset has many input features or if the classification problem has many output classes. Another decisive factor can be the number of samples per class for the classifier to learn on. The number of output classes is determined by the problem at hand. The number of input features from the intermediate dataset can be controlled for in the configuration of the CNN that is used to create the intermediate dataset. Therefore, it is a design choice whether fewer high-level output features are created and a higher likelihood of an MLP being the best choice or more output features in combination with other classification algorithms is desirable.

Apart from the established guidelines, it is always good practice to benchmark as many different classification algorithms and setups as possible, subject to time and computational constraints.

5.5 Conclusion of Results

The choice of the best classifier for a given problem is not a deterministic decision. The overall CNN architecture, independently from the final classifier, already makes a big difference for the final classification. The final classifier's performance depends on the produced high-level image features from the convolutional layers. The performances of the initially trained CNN architectures from table 52 correlate strongly with the performances of the subsequent classification step on the respective intermediate datasets from tables 58, 59, 510 and 511. The better these high-level image features, the higher the range of the top-n accuracy scores of the following classifiers.

But apart from modifying the internal CNN network structures with all the innovative approaches outlined in chapter 2.3, one can choose different setups for the final classification step of the high-level image features. Traditionally, this final classification is done with fully-connected neural network layers (MLP). Their advantage is that these models can be integrated into the learning process of CNNs because both can use backpropagation as their training procedure. Nonetheless, sections 5.3 and 5.4 show that other network architectures can be even more suitable for the final classification step.

This thesis answers the research question from section 1.3 by showing that other classification algorithms, namely LR, SVM, XGB, RFE, and KNN, have unique characteristics that can lead to a better performance than MLP models, depending on the dataset at hand and the computation and time constraints. Particularly high-dimensional intermediate datasets, i.e., the ILSVRC-2012 intermediate datasets, seemed to give other models an advantage over the benchmarked MLP models. But also setups with many output classes, as is the case in the CIFAR-100 intermediate models, proved to have better classification accuracies with models other than MLP. Overall, MLP is usually a safe choice and among the best performing models, but specific classifiers can improve the accuracy or model size. For example, LR consistently returned comparable or better results to MLP over all intermediate datasets at much smaller model sizes. RFE models also give promising results at the cost of much larger models. For lower-dimensional intermediate datasets, SVM provides the best results. XGB produces very small model sizes in comparison to the other models, especially on high-dimensional intermediate datasets. KNN can perform well on top-1 accuracy for some datasets, but its top-n accuracy for higher values of n usually gets worse. This shows that KNN is not able to develop a notion of closeness between related pictures or classes.

To summarise, other models can outperform MLPs. It is advisable to consider the guidelines established above and benchmark different classification algorithms before making a final decision about the preferred final solution. Overall, models with fewer

high-level image features are less reliant on the choice of the final classifier. This observation enables the choice of smaller model sizes for these intermediate datasets without sacrificing performance.

The findings above could be amplified and overall training time reduced if the final classifier were directly integrated into the CNN learning process. For the current process, only the MLP architecture can be directly embedded into the CNN architecture as both use gradients to fine-tune their weights iteratively. The other architectures are therefore still dependent on a CNN being trained with fully-connected layers first. Nonetheless, if it were possible to combine these architectures, the produced image filters might even be more appropriate for the respective models because they would be established in conjunction. Since boosting approaches use gradients during the learning process, the training of XGB could be directly integrated into a convolutional network to replace the final classification layer. The other algorithms would have to be adapted to be trainable with gradients.

Chapter 6

Conclusion and Outlook

Before the introduction of neural networks, computer vision relied on manually crafted image filters. For classification problems, digital images were inputted to these image filters and their output was passed to a classification algorithm to predict the final class of a given image. Any classification algorithm could be used. With the breakthrough of deep convolutional neural networks, fostered by research, the availability of data and computational means, these manually crafted image filters were replaced with multiple layers of automatically generated image filters. The convolutional neural network layers form a hierarchy of image filters from low-level filters at its beginning to high-level filters towards the output. Image filters are learned with respect to the training data by iteratively reducing the overall error of the CNN. By design, CNNs use neural networks to make the final classification on the last layer of image filters. The approach of this thesis is to reiterate over this process by analysing the proficiency of fully-connected neural networks for this final classification step.

Section 6.1 summarises the approach from this thesis and presents the key findings. Section 6.2 gives an outlook and mentions aspects that are not covered in this thesis. These could be addressed in future research.

6.1 Conclusion

Most of the recent developments around CNNs affect their convolutional layers. Section 2.3 lists numerous developments that improve the networks effectiveness or efficiency. While inventions such as the Inception module from section 2.3.7 facilitate the learning process by breaking the complex non-linear relationships to be learned into

multiple sub-problems, batch normalisation and residual blocks from sections 2.3.6 and 2.3.8 improve the gradient flow and overall learning process. Dropout from section 2.3.5 increases generalisability and depthwise separable convolutions introduced in section 2.3.2 reduce the network's complexity. But none of these developments affect the final classification step. In the nature of a CNN, this step is performed by a fully-connected neural network.

Nonetheless, once the image filters produced by a CNN are available, the fully-connected neural network can be removed from the CNN and other classification algorithms can be used to make predictions on the output from the last layer of high-level image filters. As the no-free-lunch theorem states, no classification algorithm is superior to any other classification algorithm. Therefore, the research question is formulated as:

Research Question "Can classification model performance in computer vision be improved by using different classification algorithms on high-level image features?"

The answer to that question is found in a two-step approach. In the first step, different instances of convolutional neural network configurations are trained on several benchmark datasets. In the second step, the fully-connected neural network layers, that make the final prediction, are removed from the convolutional neural networks, such that the last remaining layer produces a flattened output from the last layer of image filters. This output is referred to as intermediate dataset. On this intermediate dataset, multiple classification algorithms are benchmarked. Chapter 3 introduces several classification algorithms that can potentially be used for this benchmark and evaluation metrics to make an objective comparison. The following classification algorithms are used for the benchmark: Support Vector Machines (SVM), Logistic Regression (LR), K-Nearest Neighbours (KNN), Random Forests (RFE), Adaptive Boosting (ADB), Gradient Boosting (GBC), eXtreme Gradient Boosting (XGB). Additionally, some Multi-Layer Perceptron (MLP) architectures were added to the benchmark to represent the original fully-connected neural network classifier. The reasoning for this and the respective hyper-parameter choices are explained in section 4.6. For the evaluation of the models, top-n accuracy is used as a performance measure, with $n \in \{1, 2, 5, 10, 20\}$.

In the description of the experimental setup in chapter 4, the choice of benchmark datasets and CNNs to produce the high-level image filters is introduced. As for the datasets, this thesis uses CIFAR-10, CIFAR-100 and a subset of ILSVRC-2012 to enable an objective analysis of the research question. As for CNN architectures to produce the high-level image filters, CIFAR-10 and CIFAR-100 use four different CNN architectures. Of these architectures, two CNN models have been designed for this

thesis and are manually trained, one is taken from previous research but also manually trained (SimpleNet) and one model is available as a pre-trained model (VGG-19). ILSVRC-2012 uses three pre-trained CNN architectures that are among the best available models on this dataset: Inception V3, Xception, Inception ResNet V2. The CNN architectures applied on ILSVRC-2012 produce intermediate datasets with relatively many input dimensions, while the CNN architectures trained on CIFAR-10 and CIFAR-100 produce fewer high-level image features. This diversity of number of input features and number of output classes between the datasets enables the author to draw conclusions about the proficiency of individual algorithms for different setups.

The finding of this thesis with regards to the research question is that model performance can be improved by using different classification algorithms on high-level image features. Depending on the dataset at hand and the CNN architecture to produce the high-level image filters, different recommendations are derived for the choice of model architecture. Generally spoken, LR, SVM, XGB, RFE and KNN have been found to be able to outperform the initial CNN benchmark or the respective MLP benchmark. Therefore, these are considered to have unique characteristics that can make them superior to fully-connected neural networks for classifications on high-level image features. The likelihood of a model outperforming a MLP model is higher for high-dimensional intermediate datasets or more output classes. While the classification algorithms largely perform on par with MLP and also the initial CNN benchmark on CIFAR-10, some models perform clearly better than those two benchmarks on CIFAR-100 or ILSVRC-2012. Particularly, LR and SVM show strong performances throughout the benchmark. While LR works well for high-dimensional datasets, e.g., the intermediate datasets from ILSVRC-2012, SVM works better for lower-dimensional datasets with many output classes, e.g., CIFAR-100. LR produces among the smallest models in the benchmarks while the SVM models are rather large. From the remaining classifiers with unique characteristics, RFE and KNN tend to produce very large models. KNN also gets worse on top-n accuracy for higher values of n. The used configurations of XGB have the advantage of producing small model sizes even for high-dimensional datasets, e.g., on the ILSVRC-2012 intermediate datasets.

For datasets with very few input features, e.g. SimpleNet and VGG-19 intermediate data from CIFAR-10 and CIFAR-100, the final classification performance is less reliant on the actual classification algorithm used in the end. This insight can be used to make these models more performant by choosing smaller models to produce their outputs.

This thesis proves that the principle of the no-free-lunch theorem is also applicable in computer vision. Although fully-connected neural networks are required for the training process of CNNs. If available training time allows, it is recommended to benchmark other classification algorithms on the produced image features.

6.2 Outlook

The following chapter reflects on the research from this thesis and suggests modifications or new aspects that could be interesting to investigate.

The insights from this thesis give various opportunities for further research. This thesis focuses on the classification task from computer vision and shows that the inclusion of other benchmark algorithms can clearly outperform fully-connected neural network layers. This insight alone is very valuable for further research and for industry applications. Nonetheless, the scope of these findings can even be expanded from image classification to the classification part of the other computer vision tasks introduced in chapter 1. Furthermore, the same idea can be expanded from classification to regression tasks. It would need to be investigated whether there is a performance increase when using other regression models instead of fully-connected neural networks to make continuous outputs.

The observations presented were conducted on a limited set of datasets, including only a partial version of ILSVRC-2012. Therefore, it could be interesting to apply these ideas on larger datasets with more space to learn for the models. Additionally, a more extensive model training and hyper-parameter search could yield more interesting insights into proficient model configurations. Apart from the hyper-parameters, a more thorough image preprocessing on these datasets can have an impact on the overall performance of the benchmarks. The thesis discovered some situations where MLP performance is inferior to other classification algorithms, but it would need further evaluation with more datasets to identify the root causes for these outperformances. E.g., the fact that other classification algorithms outperform MLP on CIFAR-100 could be caused by the fact that CIFAR-100 makes a prediction among 100 output classes and thus gives the models more space to outperform MLPs. But the outperformance could also be caused by the fact that CIFAR-100 contains less samples of each class to learn on in comparison to CIFAR-10.

The approach could also be tested in a transfer learning scenario, where the high-level image features are not modified and only the final classification algorithm is altered.

Another addition could be the unification of the two-step procedure from this thesis. The classification algorithm from the second step would have to be integrated into the overall learning process of the convolutional network. In a first step, this could be implemented for classifiers that use gradients to learn, e.g., boosting approaches. In a second step, this could be extended to other classification algorithms.

Currently, the final prediction is only done on the final high-level image filters. A

different approach could be to use the output from the image filters from multiple layers as features for the final predictor. That way the classifier can decide whether specific low-level features are proficient to make specific classifications.

Lastly, the set of benchmark classification algorithms can be enlarged to more classification algorithms. It could be interesting to introduce evolutionary algorithms or other innovative approaches into this procedure.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] *Advanced Guide to Inception v3 on Cloud TPU*. https://cloud.google.com/tpu/docs/inception-v3-advanced. Accessed: 2018-03-21.

[3] S. Akbar, M. Peikari, S. Salama, S. Nofech-Mozes, and A. L. Martel. "Transitioning between Convolutional and Fully Connected Layers in Neural Networks." In: *CoRR* abs/1707.05743 (2017). arXiv: 1707.05743. URL: http://arxiv.org/abs/1707.05743.

[4] J. Ali, R. Khan, N. Ahmad, and I. Maqsood. "Random Forests and Decision Trees." In: 2012.

[5] *Animal Big Blur Breed*. https://www.pexels.com/photo/animal-big-blur-breed-532310/. Accessed: 2018-01-27.

[6] G. Biau. "Analysis of a Random Forests Model." In: *J. Mach. Learn. Res.* 13.1 (Apr. 2012), pp. 1063–1095. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2343682.

[7] Y. Boureau, J. Ponce, and Y. Lecun. "A theoretical analysis of feature pooling in visual recognition." English (US). In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. 2010, pp. 111–118. ISBN: 9781605589077.

[8] Y. Boureau, N. Le Roux, F. Bach, J. Ponce, and Y. Lecun. "Ask the locals: Multi-way local pooling for image recognition." English (US). In: *2011 International Conference on Computer Vision, ICCV 2011*. 2011, pp. 2651–2658. ISBN: 9781457711015. DOI: 10.1109/ICCV.2011.6126555.

[9] L. Breiman. "Random Forests." In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[10] *Capsule Network Explanation*. https://www.oreilly.com/ideas/introducing-capsule-networks. Accessed: 2018-01-27.

[11] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[12] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions." In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[13] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. "Gated Feedback Recurrent Neural Networks." In: *CoRR* abs/1502.02367 (2015). arXiv: 1502.02367. URL: http://arxiv.org/abs/1502.02367.

[14] *Clack and Brown Short Haired Puppy in Cup*. https://www.pexels.com/photo/black-and-brown-short-haired-puppy-in-cup-39317/. Accessed: 2018-01-27.

[15] T. Cooijmans, N. Ballas, C. Laurent, and A. C. Courville. "Recurrent Batch Normalization." In: *CoRR* abs/1603.09025 (2016). arXiv: 1603.09025. URL: http://arxiv.org/abs/1603.09025.

[16] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[17] G. Enderlein. "McCullagh, P., J. A. Nelder: Generalized linear models. Chapman and Hall London – New York 1983, 261 S., £ 16,–." In: *Biometrical Journal* 29.2 (), pp. 206–206. DOI: 10.1002/bimj.4710290217. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bimj.4710290217. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.4710290217.

[18] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. "LIBLINEAR: a library for large linear classification." In: 9 (Aug. 2008), pp. 1871–1874.

[19] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. "PathNet: Evolution Channels Gradient Descent in Super Neural Networks." In: *CoRR* abs/1701.08734 (2017). arXiv: 1701.08734. URL: http://arxiv.org/abs/1701.08734.

[20] J. H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 00905364. URL: http://www.jstor.org/stable/2699986.

[21] J. H. Friedman. "Stochastic Gradient Boosting." In: *Comput. Stat. Data Anal.* 38.4 (Feb. 2002), pp. 367–378. ISSN: 0167-9473. DOI: 10.1016/S0167-9473(01)00065-2. URL: http://dx.doi.org/10.1016/S0167-9473(01)00065-2.

[22] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[23] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[24] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. "Maxout Networks." In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 1319–1327. URL: http://proceedings.mlr.press/v28/goodfellow13.html.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

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

[27] *Google Brain*. https://ai.google/research/teams/brain. Accessed: 2018-02-02.

[28] L. G. C. Hamey. "A Functional Approach to Border Handling in Image Processing." In: *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2015, pp. 1–8. DOI: 10.1109/DICTA.2015.7371214.

[29] M. Hardt and T. Ma. "Identity Matters in Deep Learning." In: *CoRR* abs/1611.04231 (2016). arXiv: 1611.04231. URL: http://arxiv.org/abs/1611.04231.

[30] S. H. HasanPour, M. Rouhani, M. Fayyaz, and M. Sabokrou. "Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures." In: *CoRR* abs/1608.06037 (2016). arXiv: 1608.06037. URL: http://arxiv.org/abs/1608.06037.

[31] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[32] C. F. Higham and D. J. Higham. "Deep Learning: An Introduction for Applied Mathematicians." In: *CoRR* abs/1801.05894 (2018).

[33] G. E. Hinton, A. Krizhevsky, and S. D. Wang. "Transforming Auto-encoders." In: *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*. ICANN'11. Espoo, Finland: Springer-Verlag, 2011, pp. 44–51. ISBN: 978-3-642-21734-0. URL: http://dl.acm.org/citation.cfm?id=2029556.2029562.

[34] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[35] G. E. Hinton, O. Vinyals, and J. Dean. "Distilling the Knowledge in a Neural Network." In: *CoRR* abs/1503.02531 (2015).

[36] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory." In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[37] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory." In: 9 (Dec. 1997), pp. 1735–80.

[38] E. Hoffer, I. Hubara, and D. Soudry. "Fix your classifier: the marginal value of training the last weight layer." In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=S1Dh8Tg0-.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[40] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. "Deep Networks with Stochastic Depth." In: *CoRR* abs/1603.09382 (2016). arXiv: 1603.09382. URL: http://arxiv.org/abs/1603.09382.

[41] G. Huang, Z. Liu, and K. Q. Weinberger. "Densely Connected Convolutional Networks." In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[42] Z. Huang, Z. Pan, and B. Lei. "Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data." In: *Remote Sensing* 9.9 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9090907. URL: http://www.mdpi.com/2072-4292/9/9/907.

[43] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[44] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding." In: *CoRR* abs/1408.5093 (2014). arXiv: 1408.5093. URL: http://arxiv.org/abs/1408.5093.

[45] L. Kaiser, A. N. Gomez, and F. Chollet. "Depthwise Separable Convolutions for Neural Machine Translation." In: *CoRR* abs/1706.03059 (2017). arXiv: 1706.03059. URL: http://arxiv.org/abs/1706.03059.

[46] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. "One Model To Learn Them All." In: *CoRR* abs/1706.05137 (2017). arXiv: 1706.05137. URL: http://arxiv.org/abs/1706.05137.

[47] S. S. Keerthi and C.-J. Lin. "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel." In: *Neural Comput.* 15.7 (July 2003), pp. 1667–1689. ISSN: 0899-7667. DOI: 10.1162/089976603321891855. URL: http://dx.doi.org/10.1162/089976603321891855.

[48] *Keras Applications - Pretrained Models*. https://keras.io/applications/. Accessed: 2018-01-12.

[49] *Keras: The Python Deep Learning library*. https://keras.io. Accessed: 2018-01-22.

[50] J. Kim, M. El-Khamy, and J. Lee. "Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition." In: *CoRR* abs/1701.03360 (2017). arXiv: 1701.03360. URL: http://arxiv.org/abs/1701.03360.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[52] J. S. L. Torrey. "Transfer Learning." In: *Handbook of Research on Machine Learning Applications* ().

[53] *Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) All Results*. http://www.image-net.org/challenges/LSVRC/2012/results.html. Accessed: 2018-05-01.

[54] *Large Scale Visual Recognition Challenge 2014 (ILSVRC2014) All Results*. http://image-net.org/challenges/LSVRC/2014/results. Accessed: 2018-05-01.

[55] *Large Scale Visual Recognition Challenge 2015 (ILSVRC2015) All Results*. http://image-net.org/challenges/LSVRC/2015/results. Accessed: 2018-05-01.

[56] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. "Deeply-Supervised Nets." In: *CoRR* abs/1409.5185 (2015).

[57] C.-Y. Lee, P. W. Gallagher, and Z. Tu. "Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree." In: (Sept. 2015).

[58] M. Lin, Q. Chen, and S. Yan. "Network In Network." In: *CoRR* abs/1312.4400 (2013). arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400.

[59] J. Luna, E. Eaton, L. Ungar, E. Diffenderfer, S. Jensen, E. Gennatas, M. Wirth, I. Charles B. Simone, T. D. Solberg, and G. Valdes. "Tree-Structured Boosting: Connections Between Gradient Boosted Stumps and Full Decision Trees." In: (Nov. 2017).

[60] W. Ma and J. Lu. "An Equivalence of Fully Connected Layer and Convolutional Layer." In: *CoRR* abs/1712.01252 (2017). arXiv: 1712.01252. URL: http://arxiv.org/abs/1712.01252.

[61] A. F. Mashat, M. M. Fouad, P. S. Yu, and T. F. Gharib. "A Decision Tree Classification Model for University Admission System." In: *International Journal of Advanced Computer Science and Applications* 3.10 (2012). DOI: 10.14569/IJACSA.2012.031003. URL: http://dx.doi.org/10.14569/IJACSA.2012.031003.

[62] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulsin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[63] G. A. Miller. "WordNet: A Lexical Database for English." In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: http://doi.acm.org/10.1145/219717.219748.

[64] *MNIST Dataset Introduction and Benchmarks*. http://yann.lecun.com/exdb/mnist/. Accessed: 2018-05-01.

[65] G. E. Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 33–35. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4785860.

[66] S. J. Pan and Q. Yang. "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191.

[67] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll. "An Introduction to Logistic Regression Analysis and Reporting." In: *The Journal of Educational Research* 96.1 (2002), pp. 3–14. DOI: 10.1080/00220670209598786. eprint: https://doi.org/10.1080/00220670209598786. URL: https://doi.org/10.1080/00220670209598786.

[68] S. Pradhan. "Exploring the Depths of Recurrent Neural Networks with Stochastic Residual Learning." In: 2016.

[69] L. Y. Pratt. "Discriminability-Based Transfer between Neural Networks." In: *Advances in Neural Information Processing Systems 5*. Ed. by S. J. Hanson, J. D. Cowan, and C. L. Giles. Morgan-Kaufmann, 1993, pp. 204–211. URL: http://papers.nips.cc/paper/641-discriminability-based-transfer-between-neural-networks.pdf.

[70] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger." In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[71] J. Redmon and A. Farhadi. "YOLOv3: An Incremental Improvement." In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[72] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[73] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain [J]." In: 65 (Dec. 1958), pp. 386 –408.

[74] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1." In: ed. by D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

[75] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[76] R S. Woodworth and E L. Thorndike. "The influence of improvement in one mental function upon the efficiency of other functions. Upon the efficiency of other functions (I)." In: 8 (May 1901), pp. 247–261.

[77] R S. Woodworth and E L. Thorndike. "The influence of improvement in one mental function upon the efficiency of other functions. Upon the efficiency of other functions (II) - The estimation of magnitudes." In: 8 (May 1901), pp. 384–395.

[78] R S. Woodworth and E L. Thorndike. "The influence of improvement in one mental function upon the efficiency of other functions. Upon the efficiency of other functions (III) - Functions involving attention, observation and discrimination." In: 8 (May 1901), pp. 553–564.

[79] S. Sabour, N. Frosst, and G. E. Hinton. "Dynamic Routing Between Capsules." In: *CoRR* abs/1710.09829 (2017). arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

[80] F. Schilling. "The Effect of Batch Normalization on Deep Convolutional Neural Networks." In: (2016). URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-191222.

[81] H. Schwenk and Y. Bengio. "Boosting Neural Networks." In: *Neural Comput.* 12.8 (Aug. 2000), pp. 1869–1887. ISSN: 0899-7667. DOI: 10.1162/089976600300015178. URL: http://dx.doi.org/10.1162/089976600300015178.

[82] *scikit-learn: Machine Learning in Python*. http://scikit-learn.org/stable/. Accessed: 2018-01-22.

[83] *scikit-learn: XGBoost library*. https://xgboost.readthedocs.io/en/latest/. Accessed: 2018-01-22.

[84] H. Sharma and S. Kumar. "A Survey on Decision Tree Algorithms of Classification in Data Mining." In: 5 (Apr. 2016).

[85] H. Shimodaira. *Improving predictive inference under covariate shift by weighting the log-likelihood function*. 2000.

[86] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[87] M. B. A. Snousy, H. M. El-Deeb, K. Badran, and I. A. A. Khlil. "Suite of decision tree-based classification algorithms on cancer gene expression data." In: *Egyptian Informatics Journal* 12.2 (2011), pp. 73 –82. ISSN: 1110-8665. DOI: https://doi.org/10.1016/j.eij.2011.04.003. URL: http://www.sciencedirect.com/science/article/pii/S1110866511000223.

[88] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. "Striving for Simplicity: The All Convolutional Net." In: *CoRR* abs/1412.6806 (2014). arXiv: 1412.6806. URL: http://arxiv.org/abs/1412.6806.

[89] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[90] R. K. Srivastava, K. Greff, and J. Schmidhuber. "Highway Networks." In: *CoRR* abs/1505.00387 (2015). arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

[91] *Stanford Computer Vision CS231n: Lecture 11 - Detection and Segmentation*. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf. Accessed: 2018-01-02.

[92] M. F. Stollenga, J. Masci, F. J. Gomez, and J. Schmidhuber. "Deep Networks with Internal Selective Attention through Feedback Connections." In: *CoRR* abs/1407.3068 (2014). arXiv: 1407.3068. URL: http://arxiv.org/abs/1407.3068.

[93] J. Su, D. V. Vargas, and K. Sakurai. "One pixel attack for fooling deep neural networks." In: *CoRR* abs/1710.08864 (2017). arXiv: 1710.08864. URL: http://arxiv.org/abs/1710.08864.

[94] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era." In: *CoRR* abs/1707.02968 (2017). arXiv: 1707.02968. URL: http://arxiv.org/abs/1707.02968.

[95] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions." In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[96] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision." In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[97] C. Szegedy, S. Ioffe, and V. Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning." In: *AAAI*. 2017.

[98] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, 2006.

[99] M. Thoma. "Analysis and Optimization of Convolutional Neural Network Architectures." Masters's Thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology, June 2017. URL: https://martin-thoma.com/msthesis/.

[100] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. "Phoneme recognition using time-delay neural networks." In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), pp. 328–339. ISSN: 0096-3518. DOI: 10.1109/29.21701.

[101] R. Wang. "AdaBoost for Feature Selection, Classification and Its Relation with SVM, A Review." In: *Physics Procedia* 25 (2012). International Conference on Solid State Devices and Materials Science, April 1-2, 2012, Macao, pp. 800 –807. ISSN: 1875-3892. DOI: https://doi.org/10.1016/j.phpro.2012.03.160. URL: http://www.sciencedirect.com/science/article/pii/S1875389212005767.

[102] X. Wang, L. Wang, and Y. Qiao. "A Comparative Study of Encoding, Pooling and Normalization Methods for Action Recognition." In: *Computer Vision – ACCV 2012*. Ed. by K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 572–585. ISBN: 978-3-642-37431-9.

[103] Y. Wang and F. Tian. "Recurrent Residual Learning for Sequence Classification." In: *EMNLP*. 2016.

[104] P. J.P. J. Werbos. *The roots of backpropagation : from ordered derivatives to neural networks and political forecasting*. English. Published simultaneously in Canada. New York : Wiley, 1994. ISBN: 0471598976 (cloth : alk. paper).

[105] Y., L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[106] J. Yim and K. Sohn. "Enhancing the Performance of Convolutional Neural Networks on Quality Degraded Datasets." In: *CoRR* abs/1710.06805 (2017). arXiv: 1710.06805. URL: http://arxiv.org/abs/1710.06805.

[107] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792.

[108] B. Yue, J. Fu, and J. Liang. "Residual Recurrent Neural Networks for Learning Sequential Representations." In: *Information* 9.3 (2018). ISSN: 2078-2489. DOI: 10.3390/info9030056. URL: http://www.mdpi.com/2078-2489/9/3/56.

[109] S. Zagoruyko and N. Komodakis. "Wide Residual Networks." In: *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146. URL: http://arxiv.org/abs/1605.07146.

[110] M. D. Zeiler and R. Fergus. "Stochastic Pooling for Regularization of Deep Convolutional Neural Networks." In: *CoRR* abs/1301.3557 (2013). arXiv: 1301.3557. URL: http://arxiv.org/abs/1301.3557.

[111] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks." In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

Appendix A

Appendix

A.1 Convolutional Neural Networks

A.1.1 Exemplification of Computer Vision challenges

Deformation

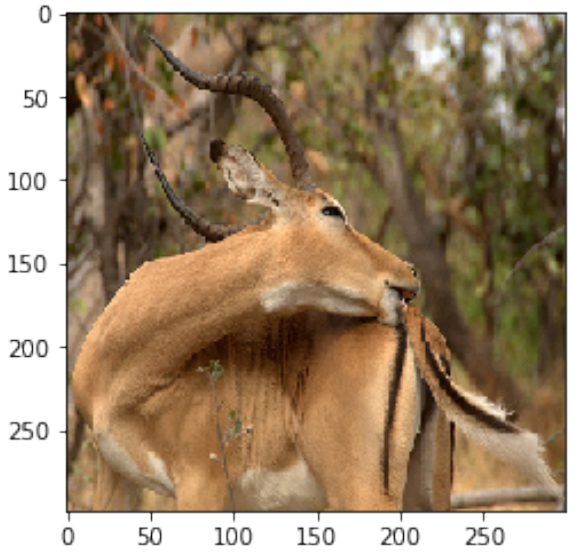


Figure A1: Example of deformation

Occlusion

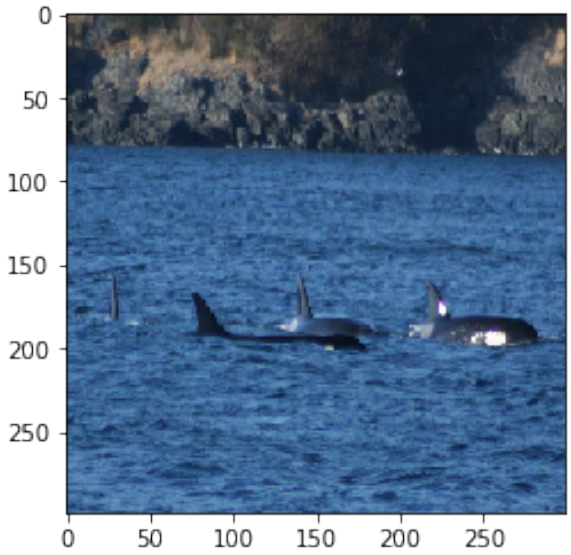


Figure A2: Example of occlusion

Viewpoint variation

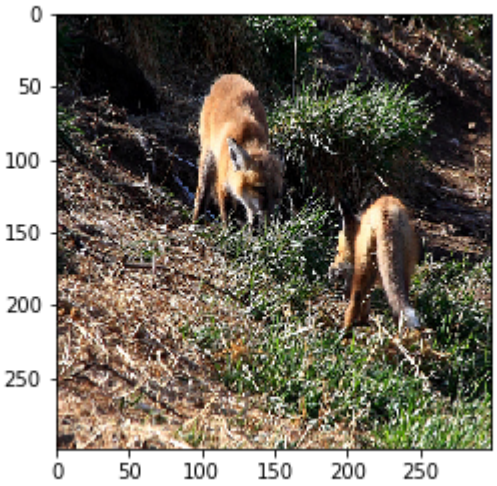


Figure A3: Example of viewpoint variation

Scale Variation

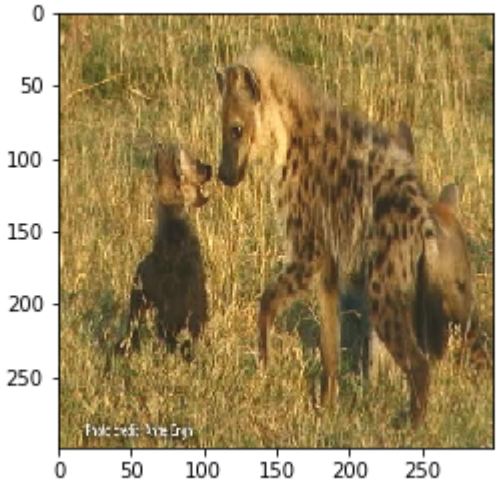


Figure A4: Example of scale variation

Background Clutter

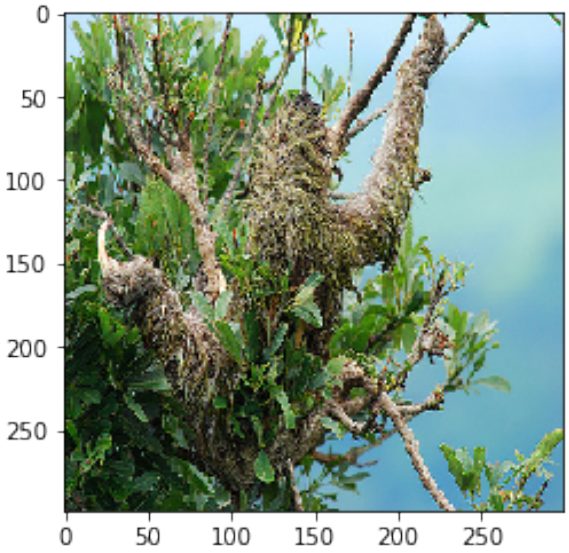


Figure A5: Example of background clutter

Intra-class variation

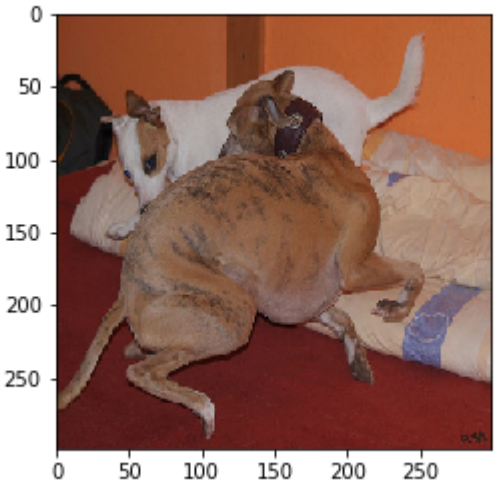


Figure A6: Example of intra class variation

A.1.2 Reformulation and simplification of Inception Module Formula

The depth (n) of $k \in K^{pred}$ on the right-side of the equation equals I_d^k of $k \in K \setminus K^1$ if $k \in K^{pred}$ is followed by a convolutional layer within the inception block. This is the case, if the reductional convolutional layer was not used after a pooling layer.

$$\sum_{k \in K \setminus K^1} \left(1 - \frac{I_d^k}{I_d^{inp}}\right) \cdot k_n \cdot k_w \cdot k_h > \sum_{k \in K \setminus K^1} I_d^k + \sum_{k \in K^{after\_pool}} k_n \tag{A.1}$$

$$\sum_{k \in K \setminus K^1} \left(\left(1 - \frac{I_d^k}{I_d^{inp}}\right) \cdot k_n \cdot k_w \cdot k_h - I_d^k\right) > \sum_{k \in K^{after\_pool}} k_n \tag{A.2}$$

There is no theoretical limit to the number of pooling layers within an Inception module, although Szegedy et al. (2014) only use one pooling layer within the inception module. In this case, the term on the right side of the equation A.2 becomes a constant of the depth of the reduction layer after that pooling layer. If no reduction were used after pooling, this term would be zero.

A.1.3 Visualisation of general architecture of CNN

Figure A7 is a visualisation of the general classification procedure of a convolutional neural network. This image is used during the thesis to explain the scientific approach, see chapter 4.

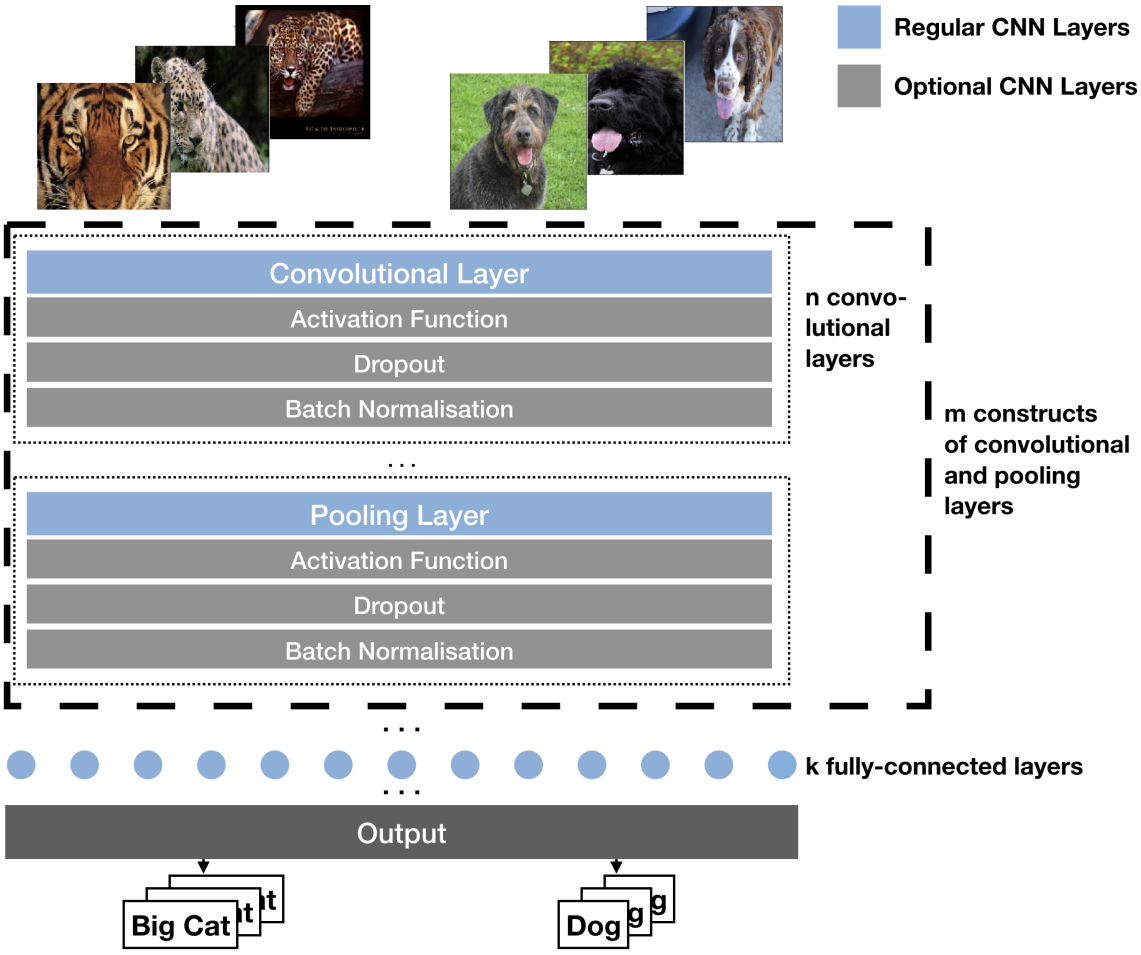
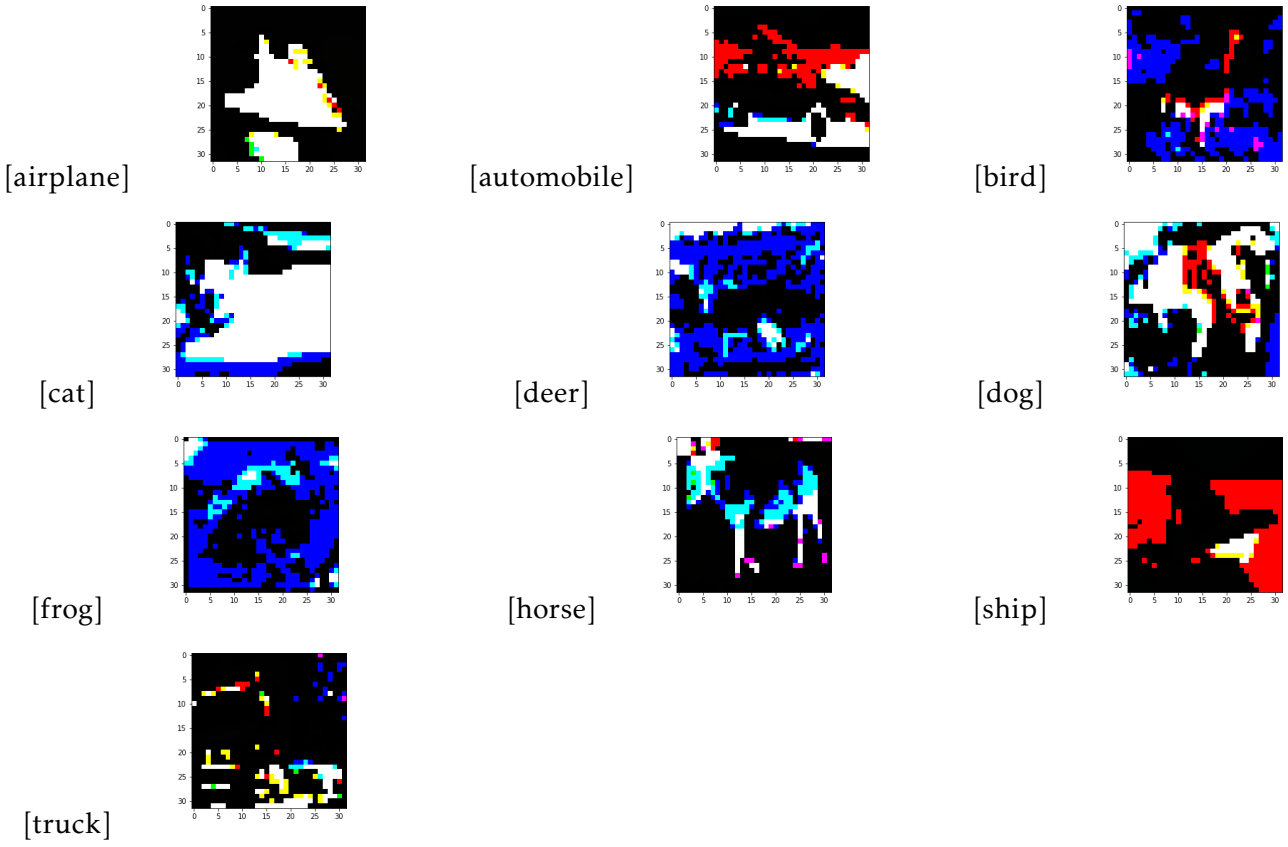


Figure A7: Visualisation of general architecture of CNN

A.2 Experimental datasets

A.2.1 Normalised images from CIFAR-10

Figure A8: Normalised images from CIFAR-10



A.2.2 Image classes from CIFAR-100

Figure A9: Sample images from CIFAR-100

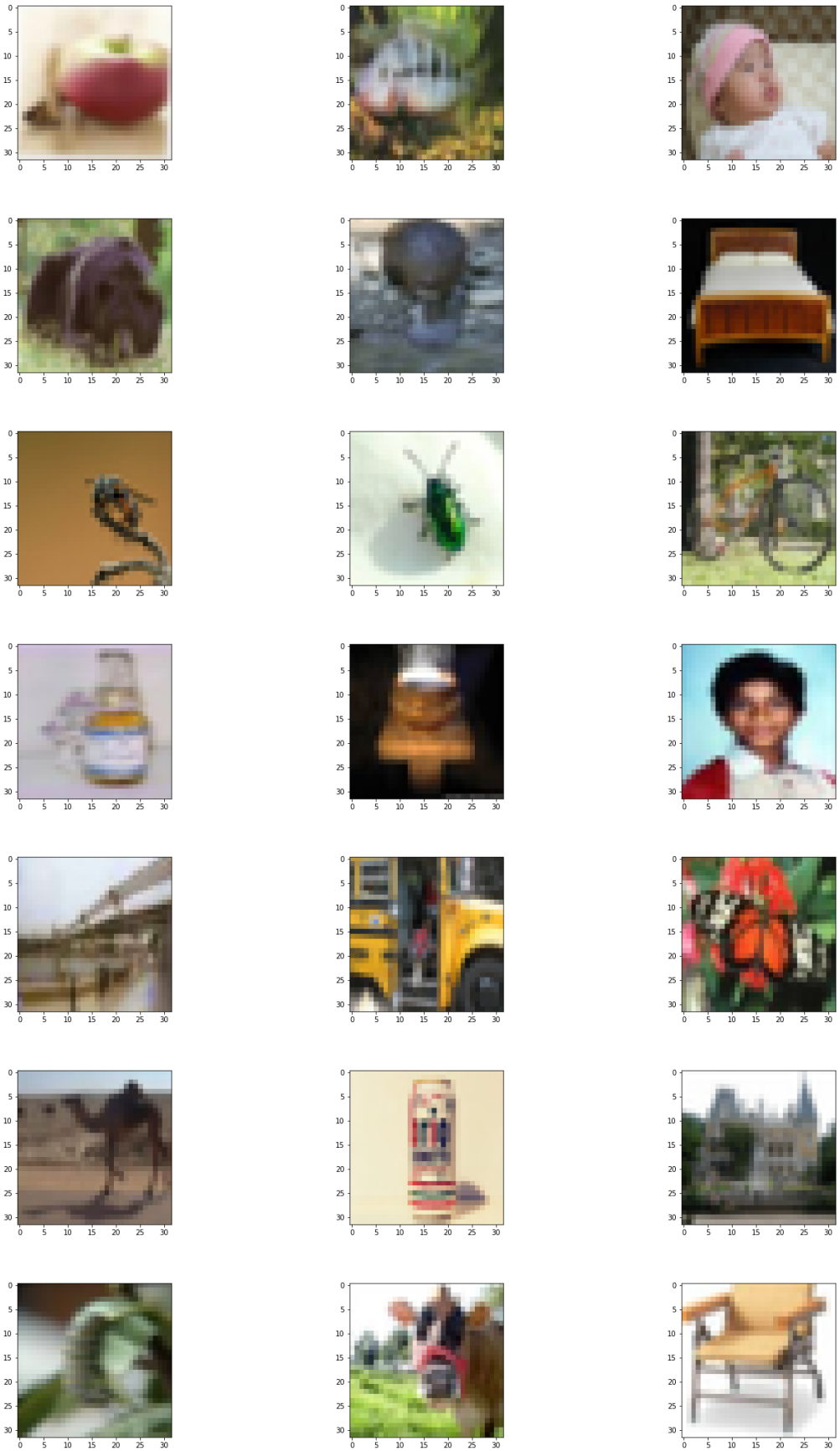


Figure A10: Sample images from CIFAR-100

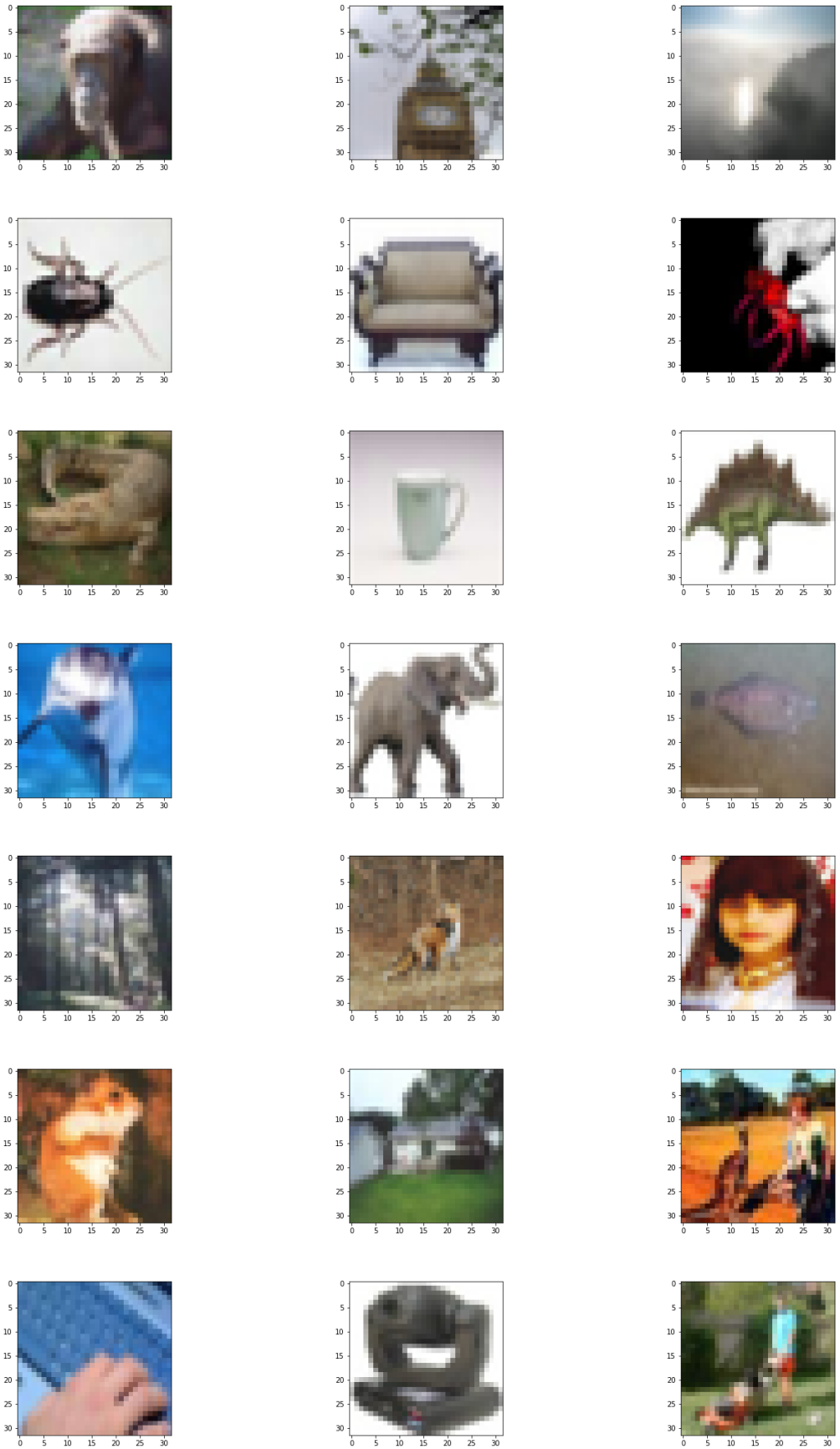


Figure A11: Sample images from CIFAR-100

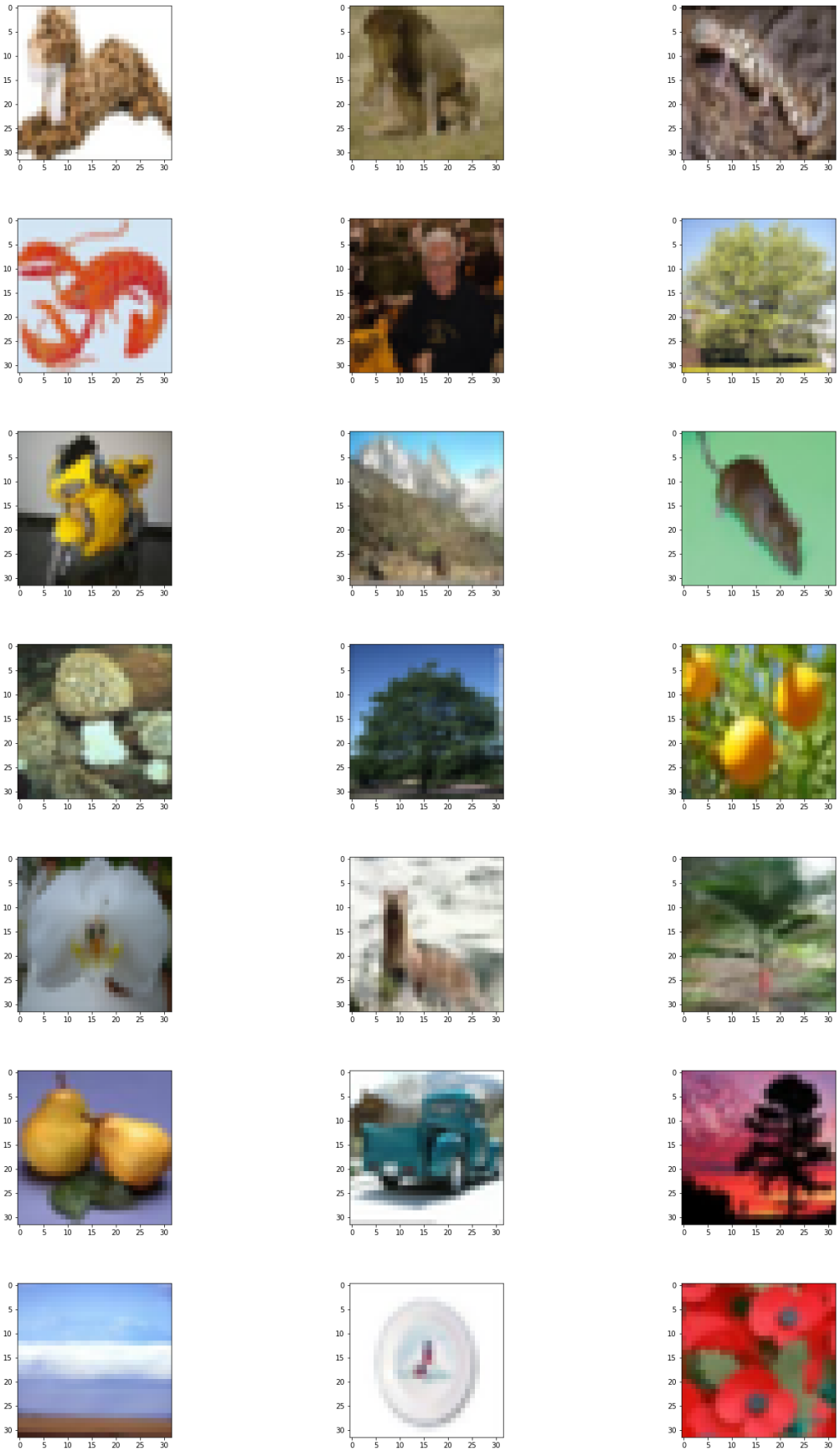


Figure A12: Sample images from CIFAR-100

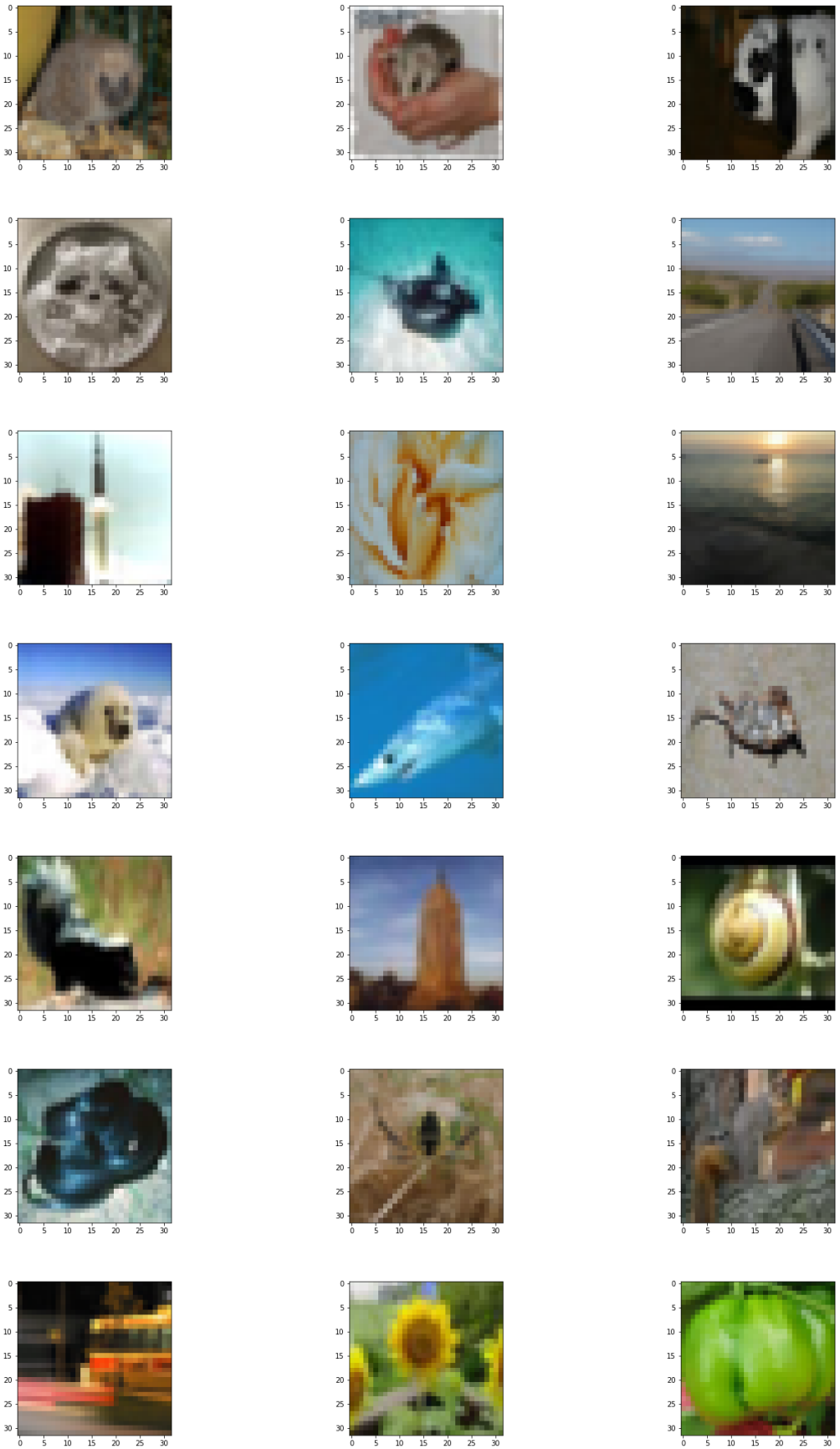
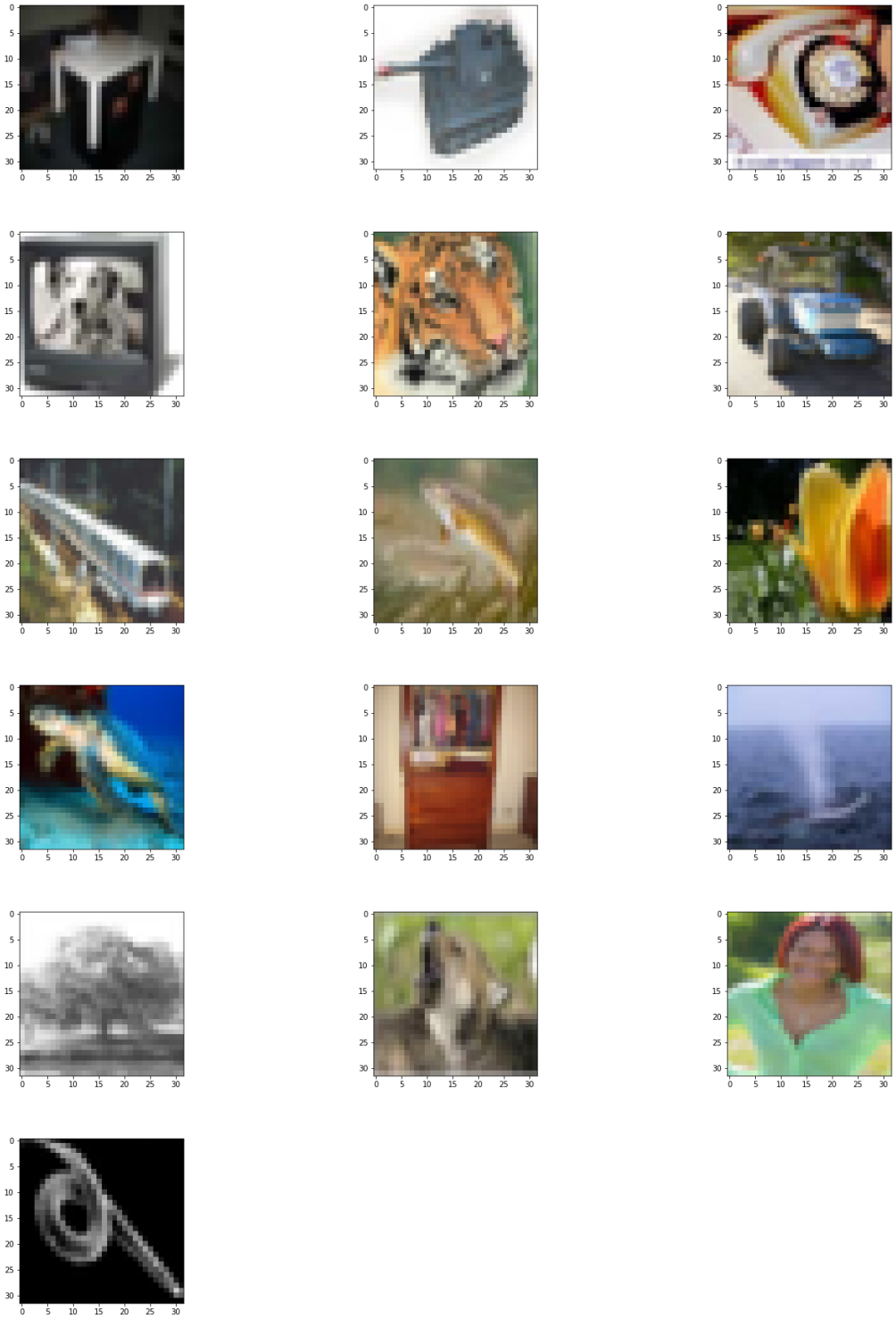


Figure A13: Sample images from CIFAR-100



A.2.3 Normalised images from CIFAR-100

Figure A14: Normalised images from CIFAR-100

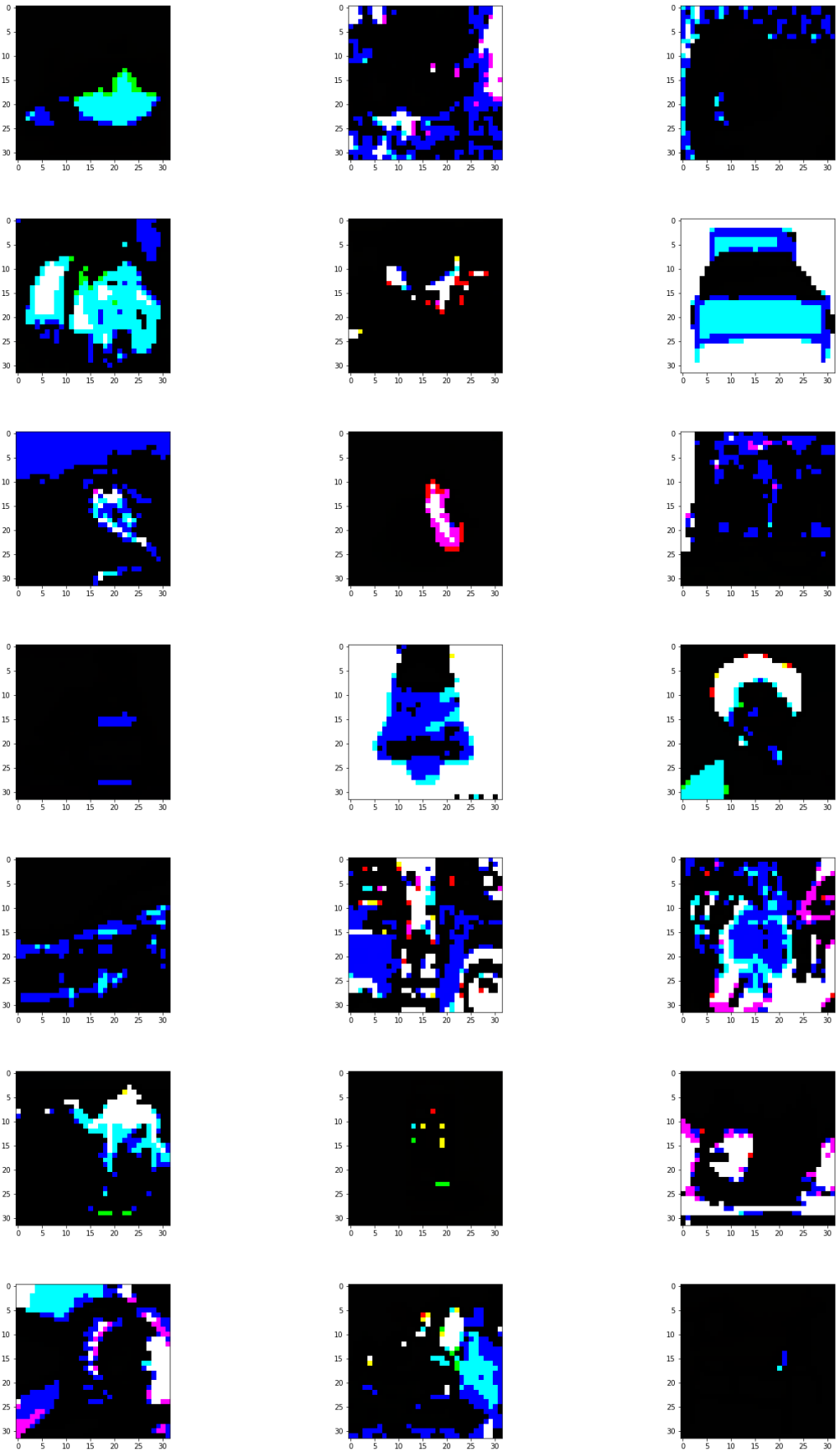


Figure A15: Normalised images from CIFAR-100

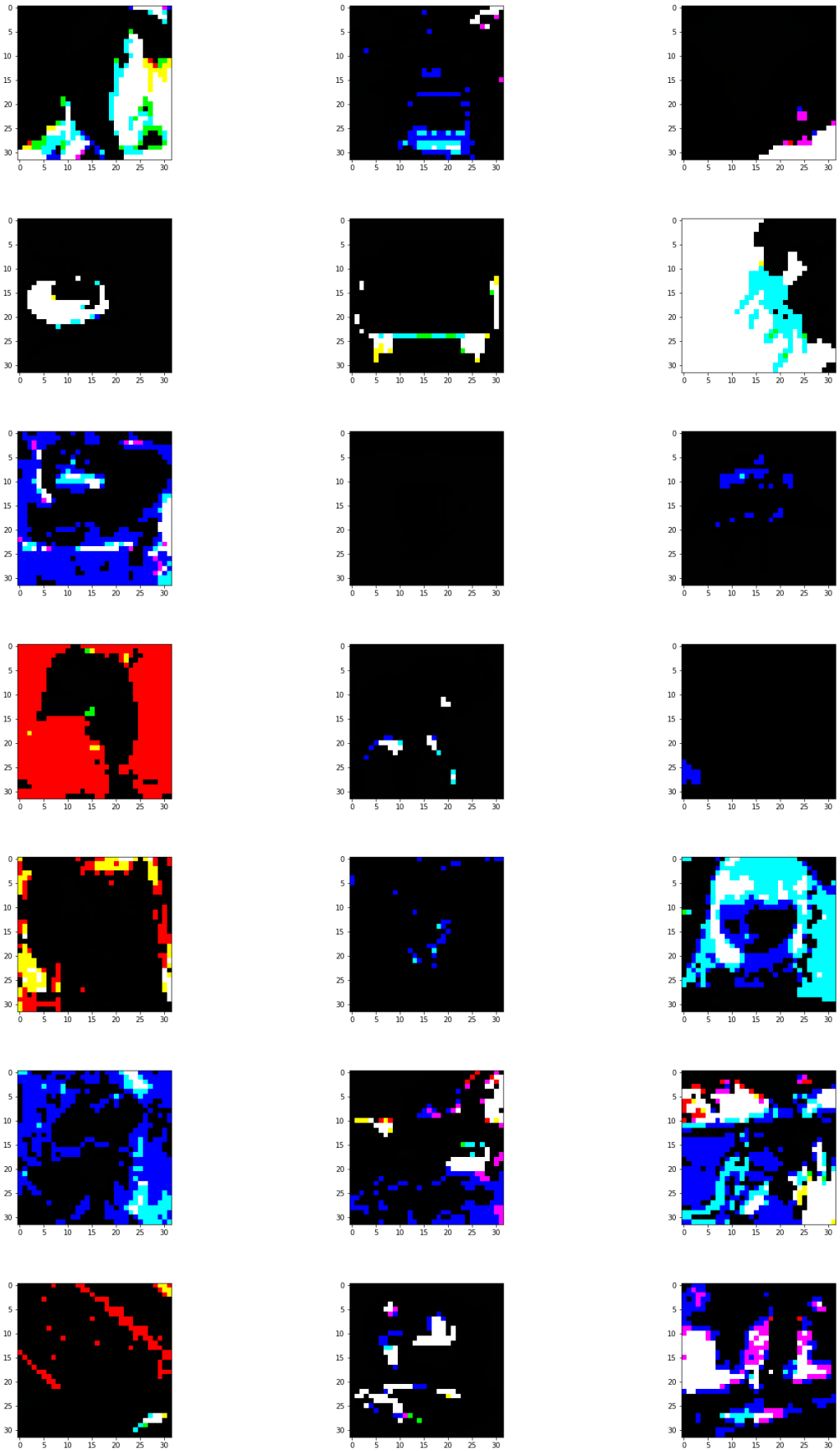


Figure A16: Normalised images from CIFAR-100

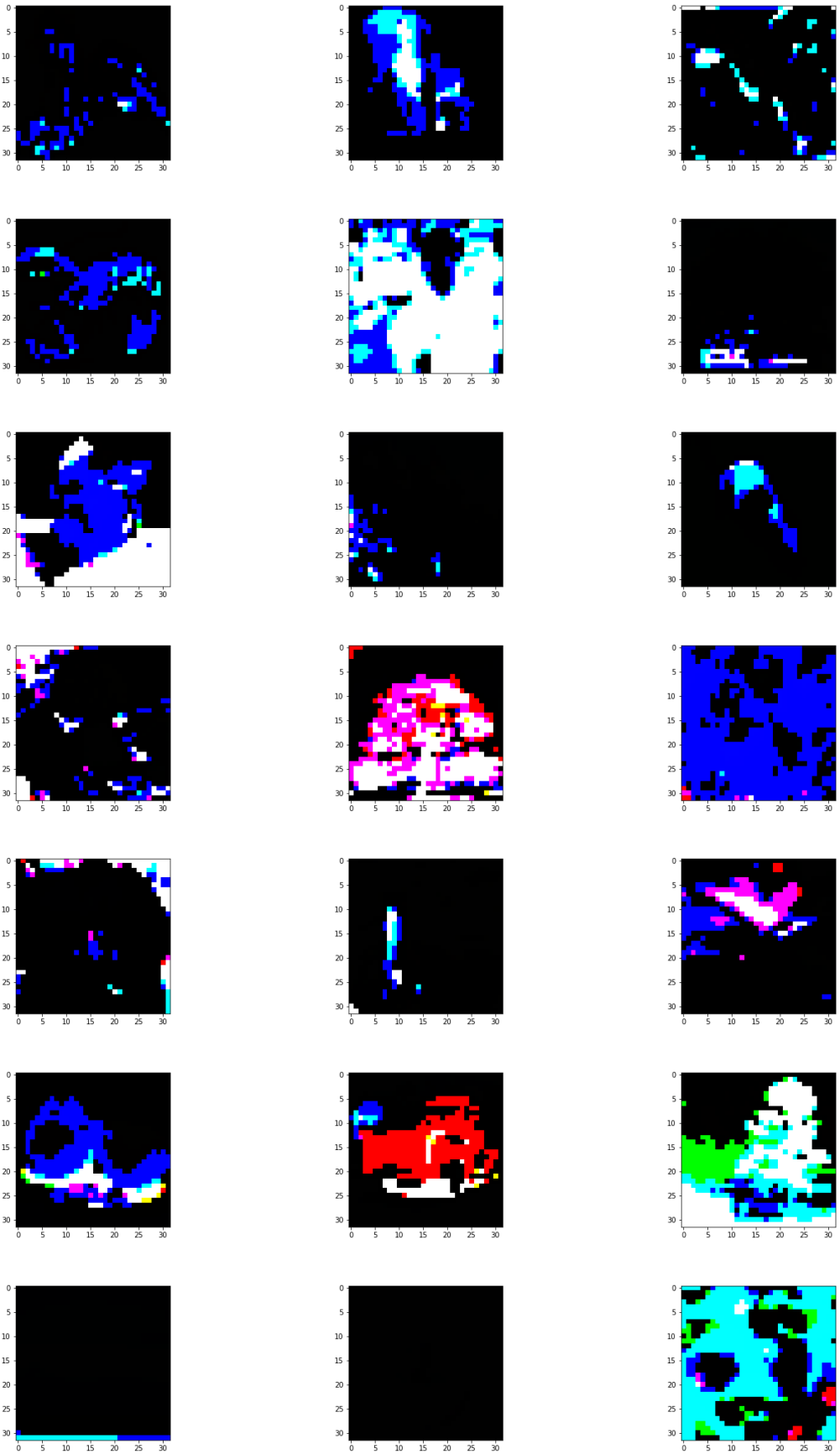


Figure A17: Normalised images from CIFAR-100

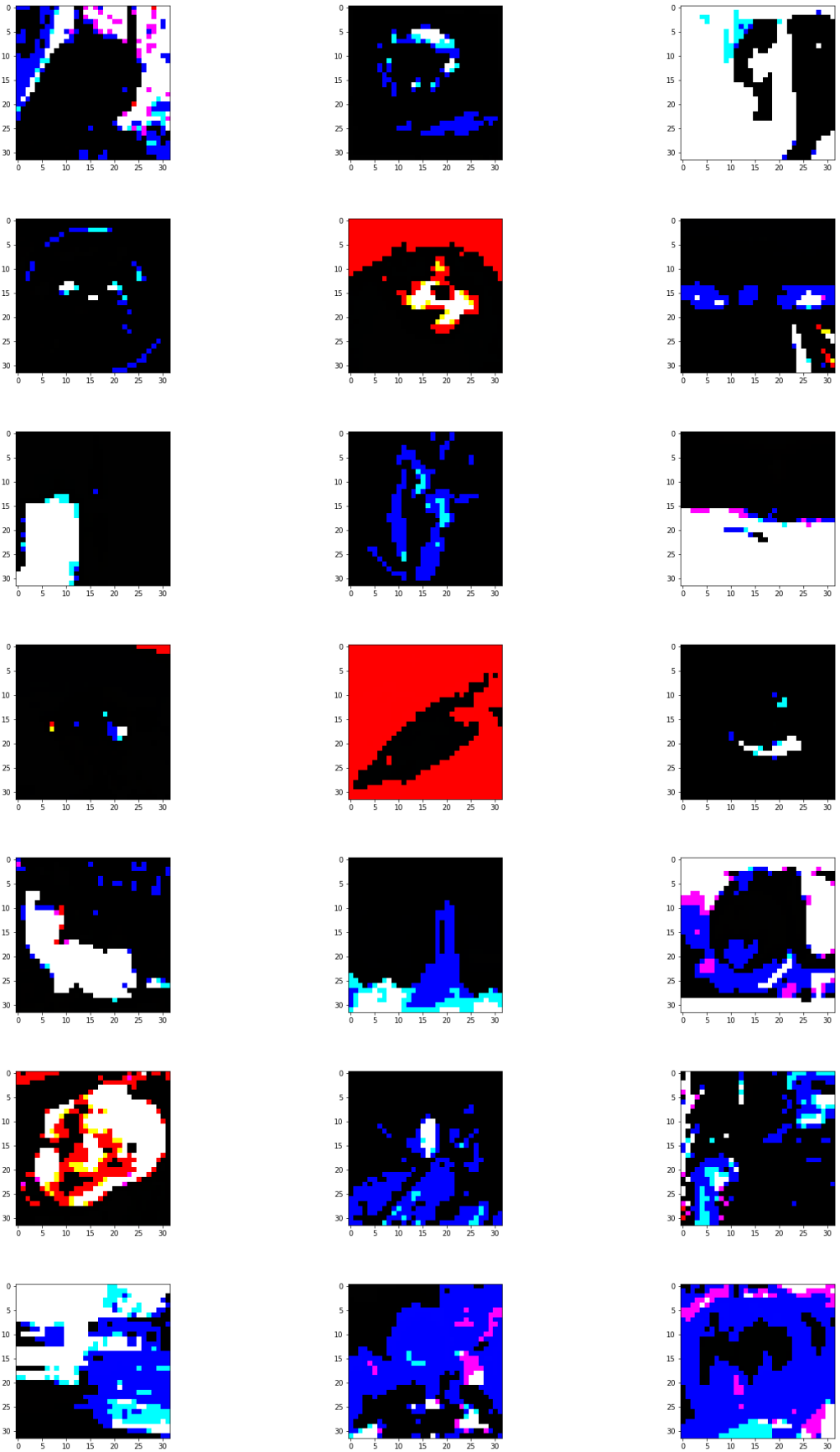
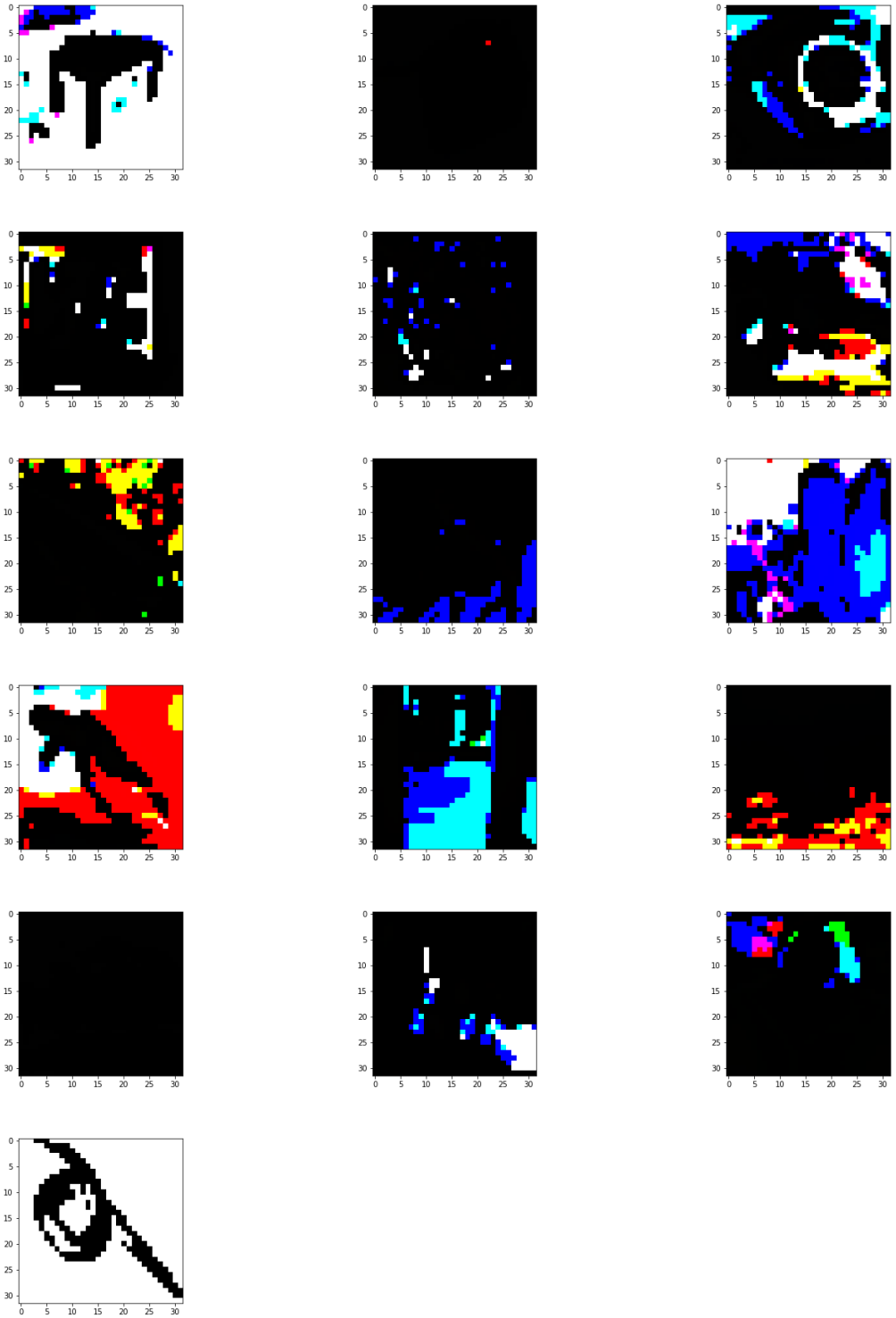


Figure A18: Normalised images from CIFAR-100



A.2.4 100 classes used from ILSVRC-2012

Figure A19: Sample images from ILSVRC-2012

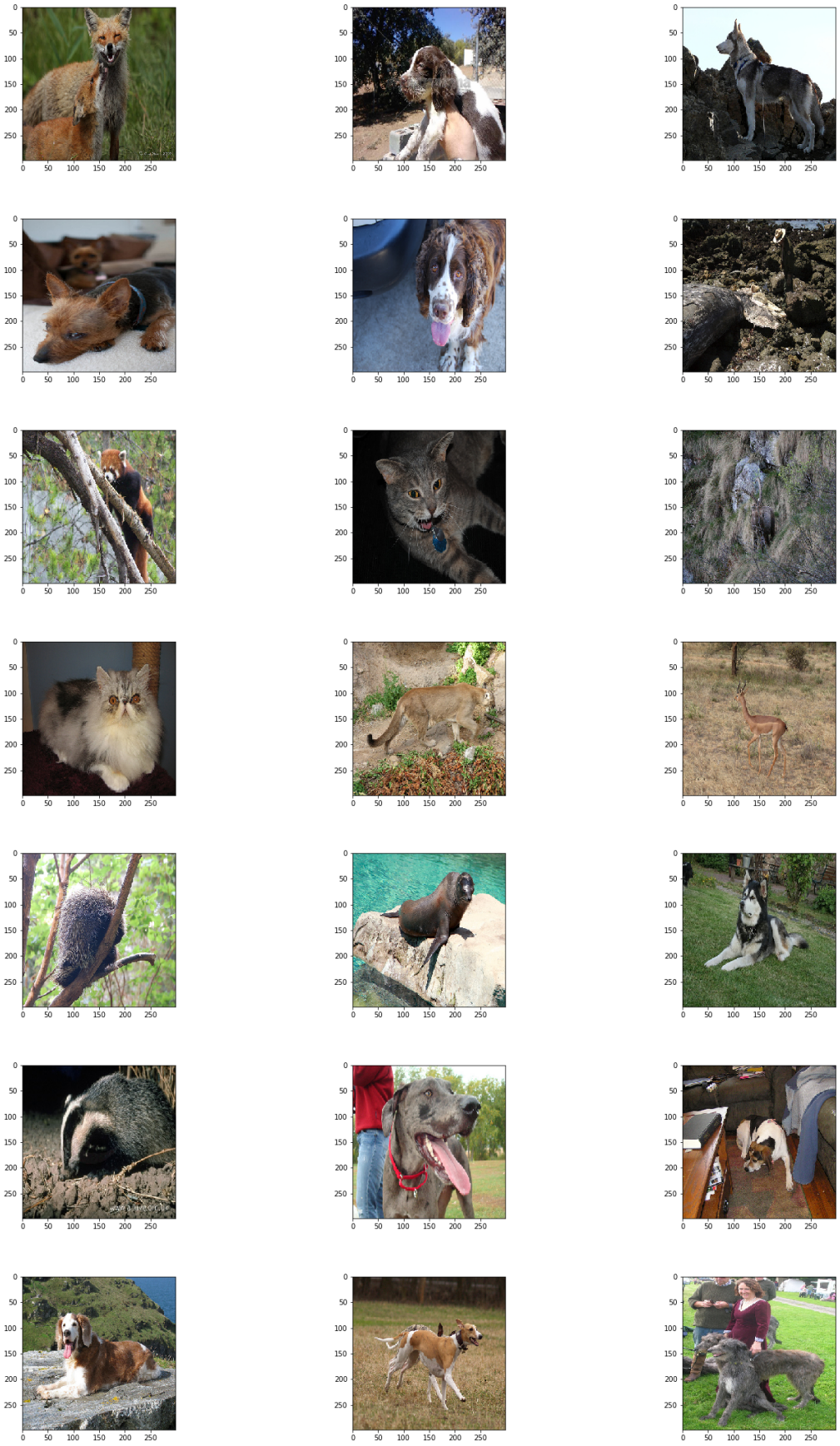


Figure A20: Sample images from ILSVRC-2012

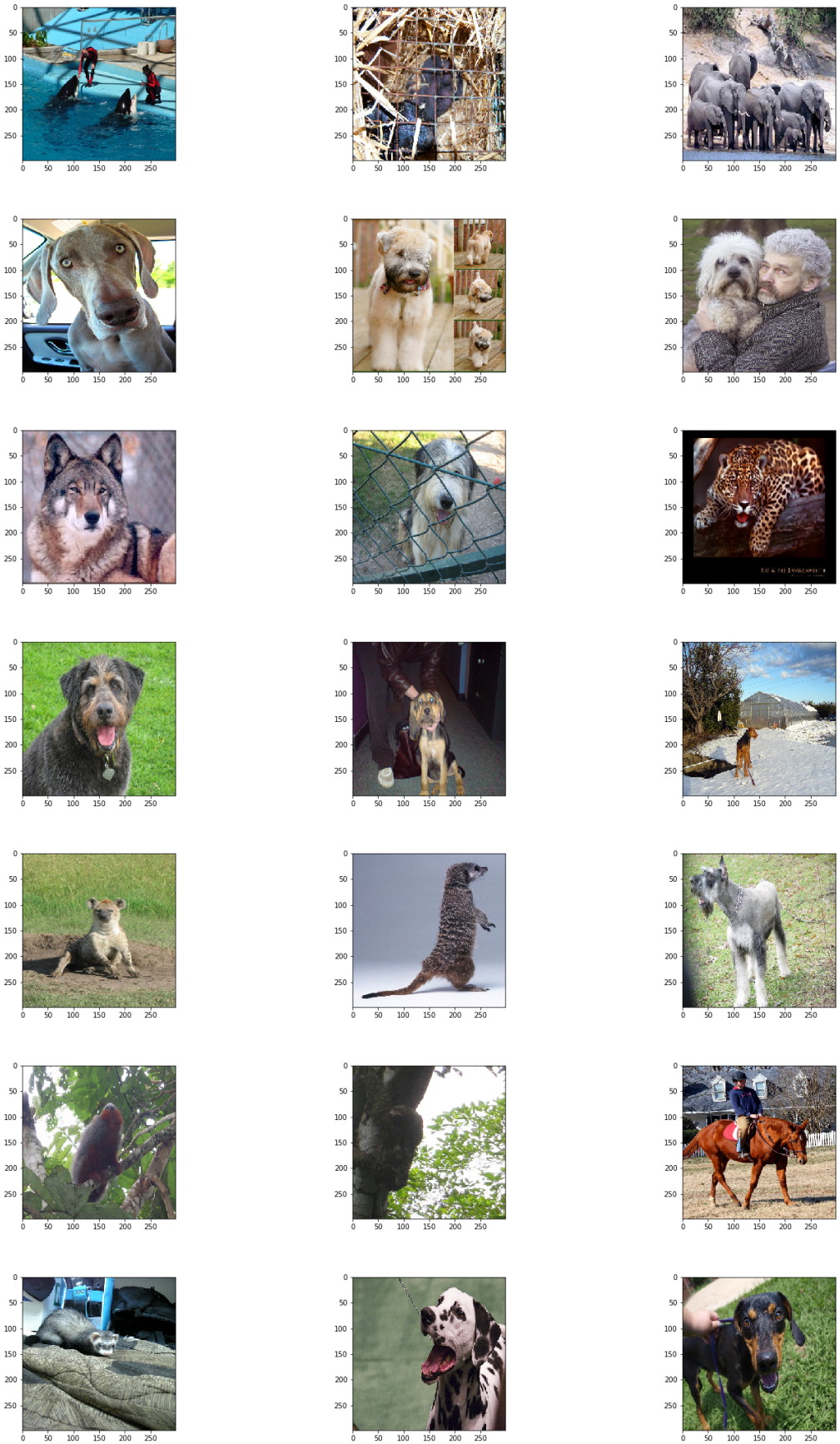


Figure A21: Sample images from ILSVRC-2012

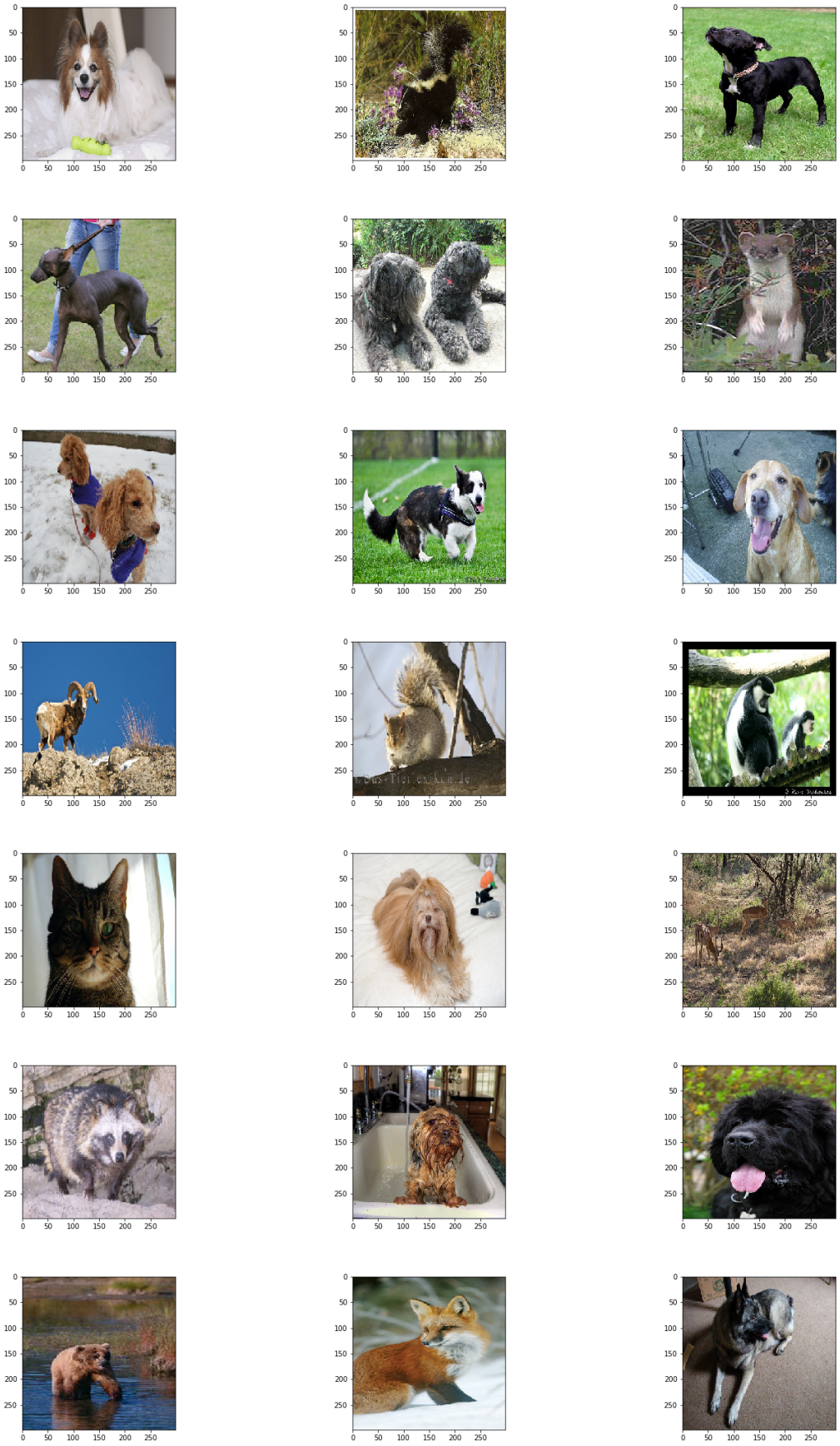


Figure A22: Sample images from ILSVRC-2012

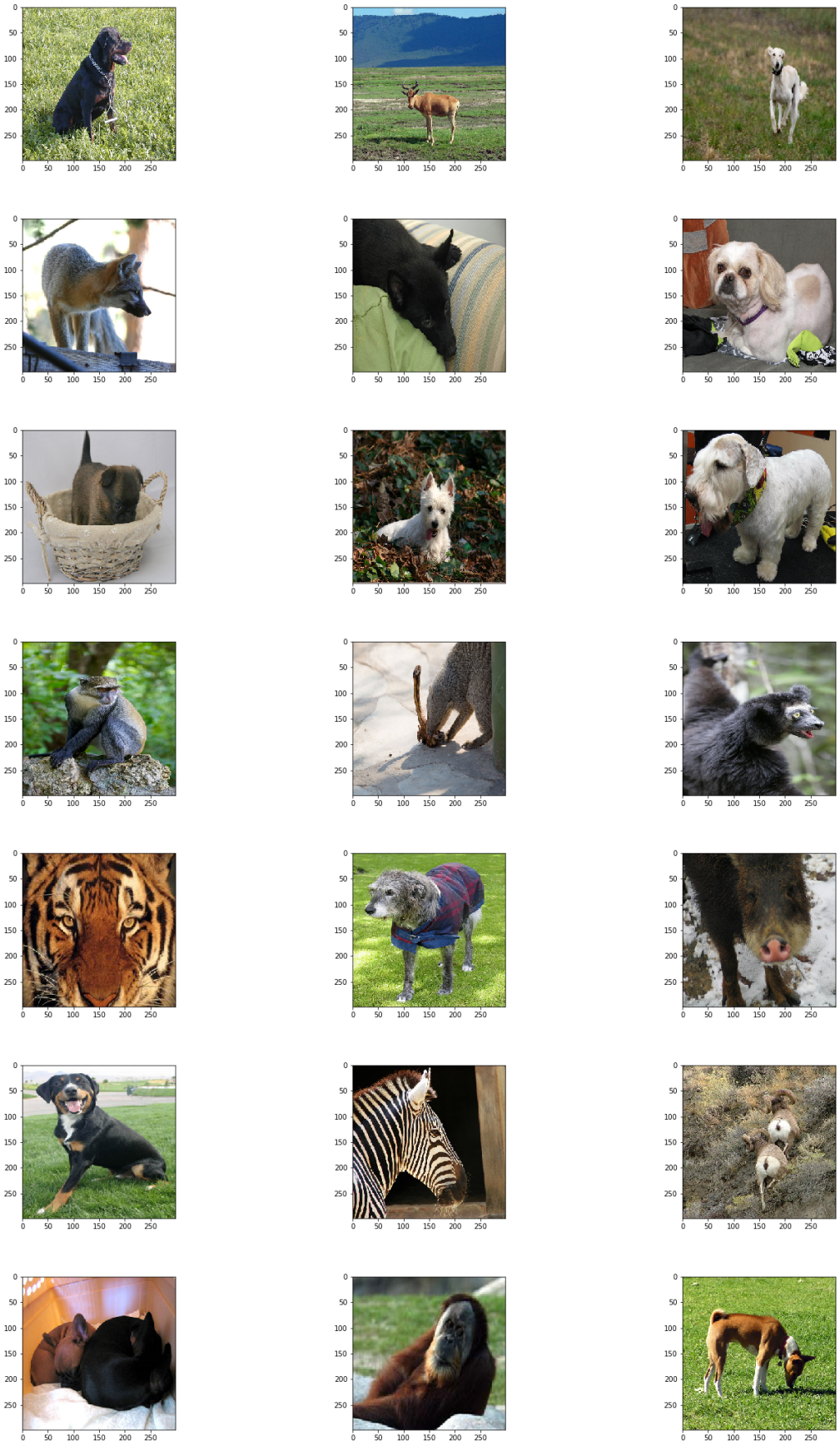
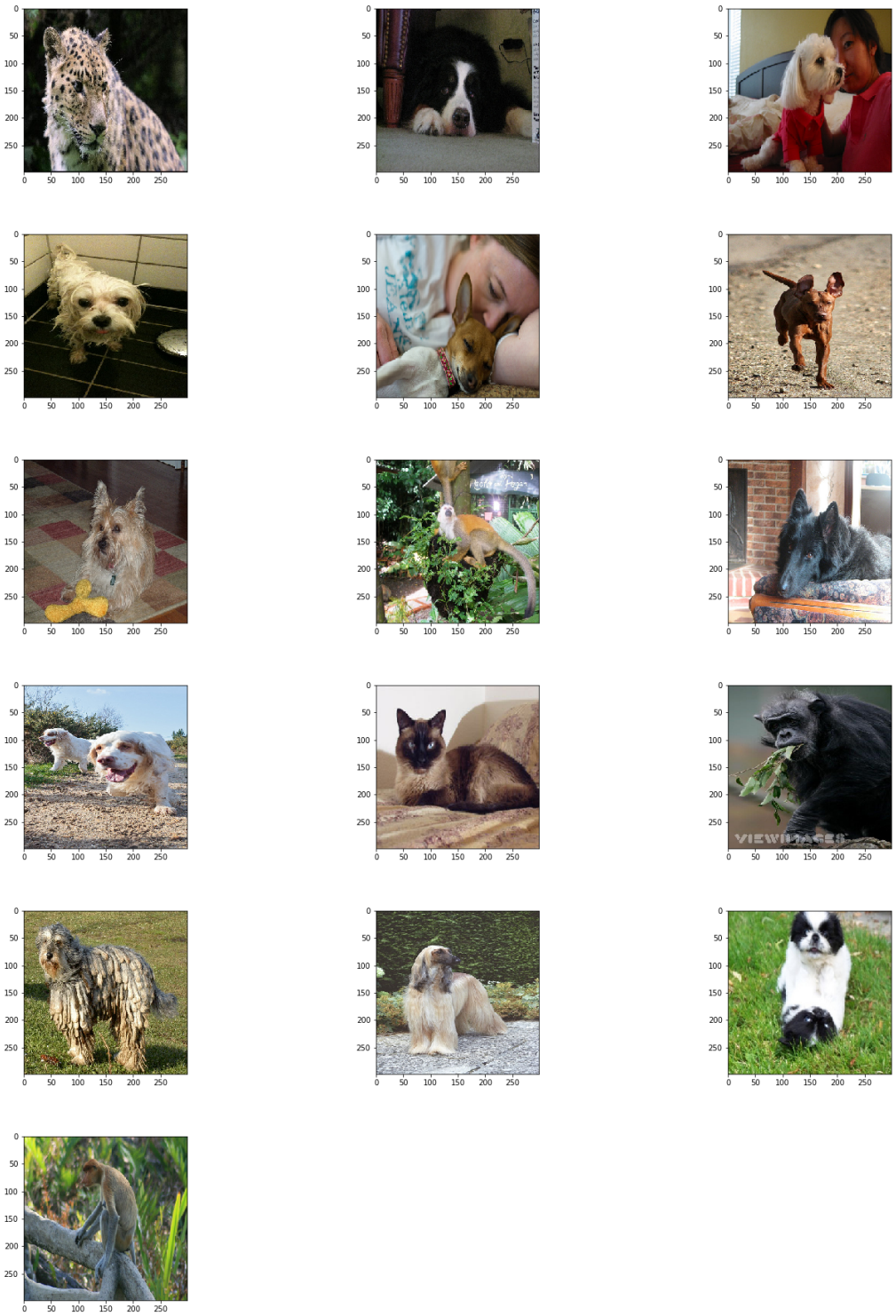


Figure A23: Sample images from ILSVRC-2012



A.3 Classification results

A.3.1 CIFAR-10

A.3.1.1 Plots of train and test metrics during training of CNN models on CIFAR-10

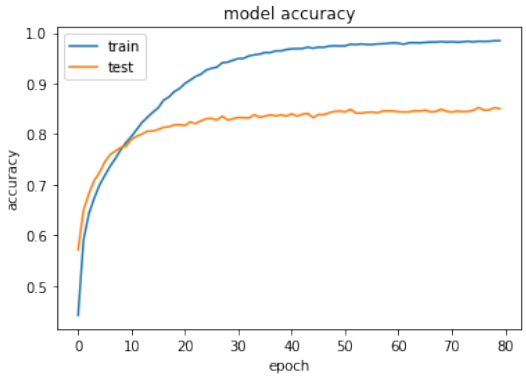


Figure A24: CNN1 Accuracy Graph on CIFAR-10

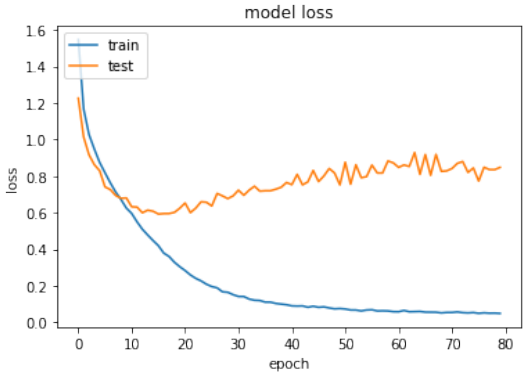


Figure A25: CNN1 Loss Graph on CIFAR-10

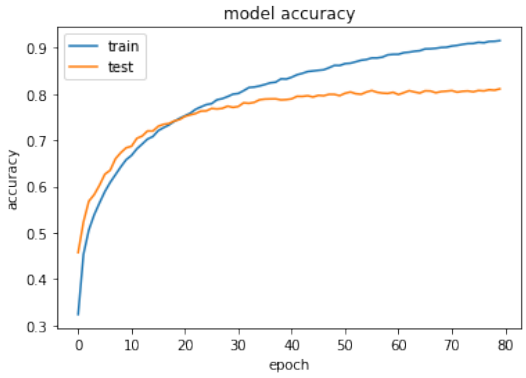


Figure A26: CNN2 Accuracy Graph on CIFAR-10

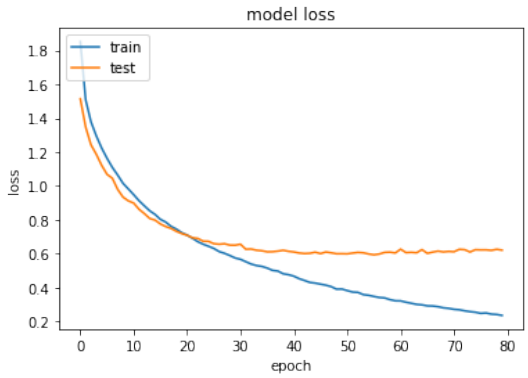


Figure A27: CNN2 Loss Graph on CIFAR-10

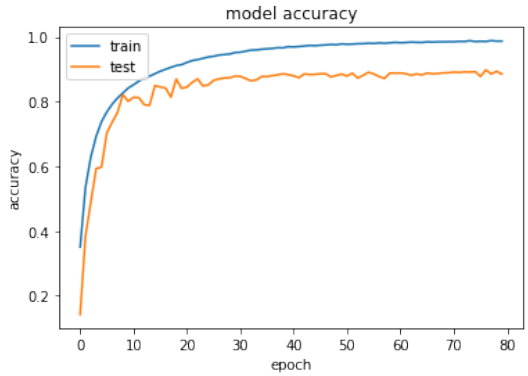


Figure A28: SimpleNet Accuracy Graph on CIFAR-10

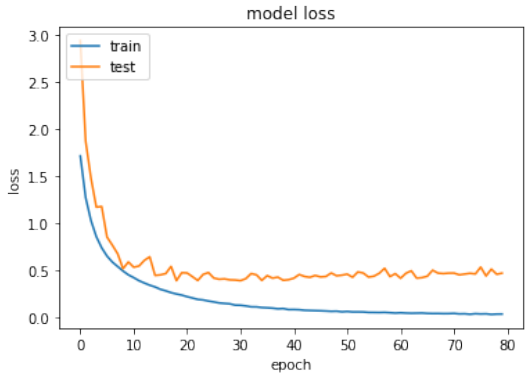


Figure A29: SimpleNet Loss Graph on CIFAR-10

A.3.1.2 Best parameters for classification algorithms on CIFAR-10 intermediate data

model	best model parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 10, γ: 0.001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 10
RFE	1000 estimators, gini criterion
ADB	1000 estimators, LR: 0.5
GBC	100 estimators, LR: 0.5
XGB	100 estimators, LR: 0.1, max. depth: 5

Table A1: CIFAR-10 final classification results on CNN-1 intermediate data

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 10, γ: 0.001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 1
RFE	1000 estimators, gini criterion
ADB	1000 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1, max. depth: 10

Table A2: CIFAR-10 final classification results on CNN-2 intermediate data

model	best parameters on test
MLP	MLP-1, LR: 0.0001
SVM	C: 10, γ: 0.001, kernel: rbf
LR	C: 0.1 , penalty: $\ell 2$
KNN	neighbours: 10
RFE	1000 estimators, entropy criterion
ADB	1000 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1,max. depth: 5

Table A3: CIFAR-10 final classification results on SimpleNet intermediate data

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 1, γ: 0.0001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 10
RFE	100 estimators, gini criterion
ADB	1000 estimators, LR: 0.5
GBC	100 estimators, LR: 0.05
XGB	100 estimators, LR: 0.1, max. depth: 1

Table A4: CIFAR-10 final classification results on VGG-19 intermediate data

A.3.1.3 Validation data performance from best parameters on CIFAR-10 with significance tests

In the diagonal lines of the tables, the mean accuracy of the validation data over 10 cross-validation folds is displayed. In the other cells, the p-value of the comparison of the two indicated networks is given. The p-value is calculated for the validation results during the 10-fold cross-validation. Since the comparison in tables A5, A6, A7, A8, A9, A10, A11 and A12 are symmetric, the p-value is only displayed for those values where the estimated population mean of the network in the column name is higher than the estimated population mean of the network in the row name that it is compared to. The other corresponding cell is left empty.

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9987**	1.000000				
MLP-2_LR-0.0001	1.000000	**0.9987**				
MLP-3_LR-0.0001	0.000486	0.000460	**0.9913**	0.007849	0.001729	0.000714
MLP-1_LR-0.001	0.000040	0.000017		**0.9948**	0.004341	0.000031
MLP-2_LR-0.001	0.165070	0.160896			**0.9974**	0.569094
MLP-3_LR-0.001	0.003684	0.000180				**0.9979**

Table A5: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on CNN-1 intermediate data from CIFAR-10 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.7781**	0.049141			0.856546	
MLP-2_LR-0.0001		**0.7908**				
MLP-3_LR-0.0001	0.017194	0.000465	**0.7638**	0.002109	0.001123	
MLP-1_LR-0.001	0.357700	0.002737		**0.7742**	0.031511	
MLP-2_LR-0.001		0.019511			**0.7789**	
MLP-3_LR-0.001	0.000077	0.000009	0.000068	0.000001	0.000003	**0.7364**

Table A6: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on CNN-2 intermediate data from CIFAR-10 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9993**					1.000000
MLP-2_LR-0.0001	0.492346	**0.9992**				0.601552
MLP-3_LR-0.0001	0.031542	0.149381	**0.9990**			0.114346
MLP-1_LR-0.001	0.088899	0.117000	0.193333	**0.9983**	0.362304	0.096405
MLP-2_LR-0.001	0.216891	0.313552	0.591050		**0.9988**	0.238096
MLP-3_LR-0.001	1.000000					**0.9993**

Table A7: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on SimpleNet intermediate data from CIFAR-10 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9999**	0.813477	1.000000	1.000000		1.000000
MLP-2_LR-0.0001		**0.9999**				
MLP-3_LR-0.0001	1.000000	0.813477	**0.9999**	1.000000		1.000000
MLP-1_LR-0.001	1.000000	0.632812	1.000000	**0.9999**		1.000000
MLP-2_LR-0.001	0.638186	0.546509	0.638186	0.627193	**0.9998**	0.627193
MLP-3_LR-0.001	1.000000	0.632812	1.000000	1.000000		**0.9999**

Table A8: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on VGG-19 intermediate data from CIFAR-10 with different learning rates (LR)

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9987**		0.000447					
SVM	0.000009	**0.9819**	0.000007					
LR			**0.9998**					
KNN	0.000003	0.000005	0.000003	**0.8245**	0.000030	0.000058	0.000007	0.000012
RFE	0.000000	0.000000	0.000000		**0.9254**		0.000002	0.000022
ADB	0.000001	0.000003	0.000001		0.001773	**0.9119**	0.000006	0.000033
GBC	0.000029	0.001752	0.000024				**0.9722**	
XGB	0.000009	0.000058	0.000008				0.000364	**0.9527**

Table A9: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on CNN-1 intermediate data from CIFAR-10

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.7908**							
SVM	0.220137	**0.7851**						
LR	0.079600	0.182234	**0.7750**					
KNN	0.000001	0.000001	0.000003	**0.5789**	0.000007	0.000070	0.000007	0.000002
RFE	0.000014	0.000005	0.000047		**0.6962**		0.133099	0.000113
ADB	0.000003	0.000001	0.000007		0.000052	**0.6456**	0.000042	0.000004
GBC	0.000024	0.000011	0.000078				**0.7015**	0.000379
XGB	0.000061	0.000016	0.000387					**0.7331**

Table A10: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on CNN-2 intermediate data from CIFAR-10

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9994**	0.309181	0.000622					
SVM		**0.9995**	0.000276					
LR			**1.0000**					
KNN	0.073066	0.060078	0.018841	**0.9982**	0.565505			
RFE	0.000664	0.000376	0.000046		**0.9986**			
ADB	0.000901	0.000868	0.000685	0.001720	0.001316	**0.9909**	0.002936	
GBC	0.000512	0.000356	0.000077	0.788268	0.035673		**0.9982**	
XGB	0.000011	0.000008	0.000003	0.010997	0.000069		0.000324	**0.9971**

Table A11: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on SimpleNet intermediate data from CIFAR-10

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9999**	1.000000	1.000000	1.000000	1.000000			
SVM	1.000000	**0.9999**	1.000000	1.000000	1.000000			
LR	1.000000	1.000000	**0.9999**	1.000000	1.000000			
KNN	1.000000	1.000000	1.000000	**0.9999**	1.000000			
RFE	1.000000	1.000000	1.000000	1.000000	**0.9999**			
ADB	0.360622	0.370111	0.365139	0.382835	0.382835	**0.9998**		1.000000
GBC	0.031259	0.033679	0.032393	0.037590	0.037152	0.130492	**0.9996**	0.110787
XGB	0.226325	0.242443	0.234023	0.266265	0.263719	1.000000		**0.9998**

Table A12: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on VGG-19 intermediate data from CIFAR-10

A.3.2 CIFAR-100

A.3.2.1 Plots of train and test metrics during training of CNN models on CIFAR-100

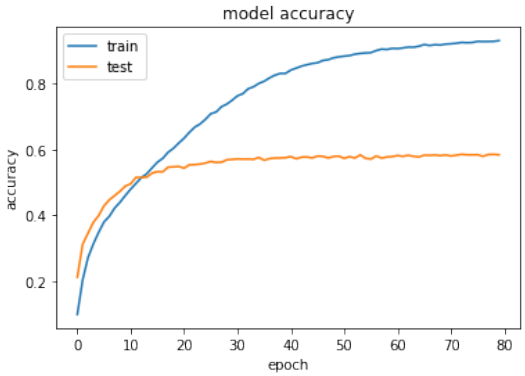


Figure A30: CNN1 Accuracy Graph on CIFAR-100

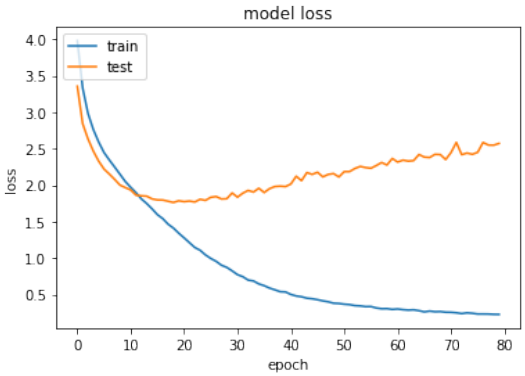


Figure A31: CNN1 Loss Graph on CIFAR-100

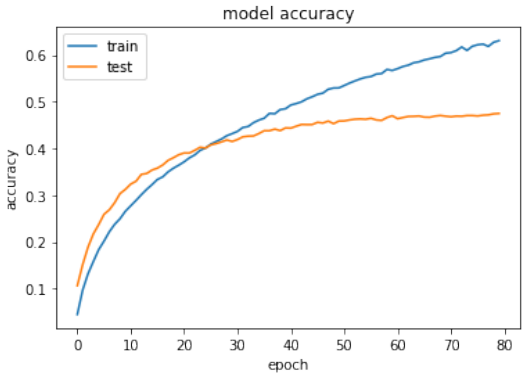


Figure A32: CNN2 Accuracy Graph on CIFAR-100

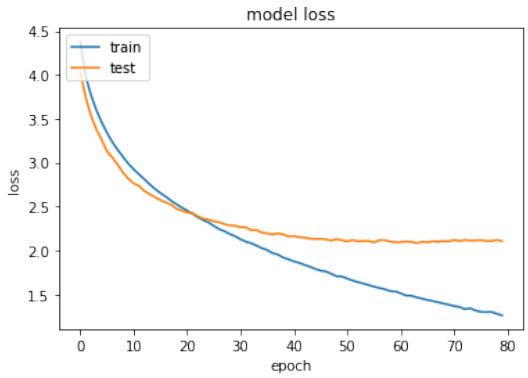


Figure A33: CNN2 Loss Graph on CIFAR-100

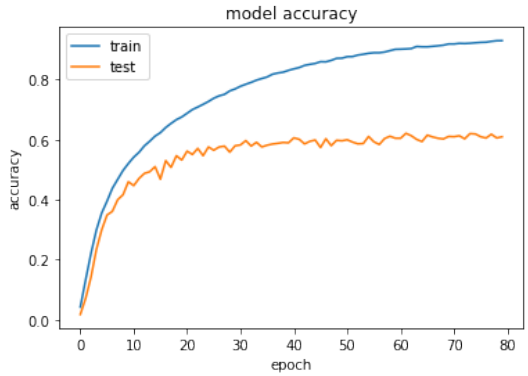


Figure A34: SimpleNet Accuracy Graph on CIFAR-100

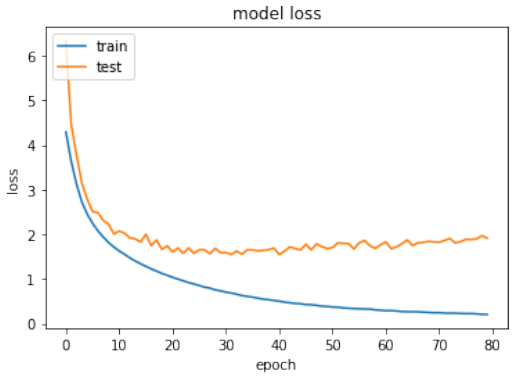


Figure A35: SimpleNet Loss Graph on CIFAR-100

A.3.2.2 Images of training graphs from MLPs on intermediate dataset from CIFAR-100

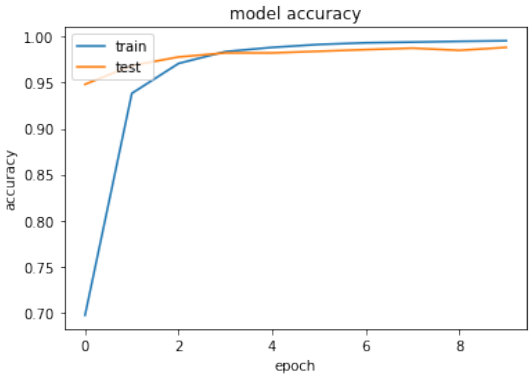


Figure A36: Accuracy Curve of MLP 0 on CNN1 for 10 epochs with a learning rate of 0.0001

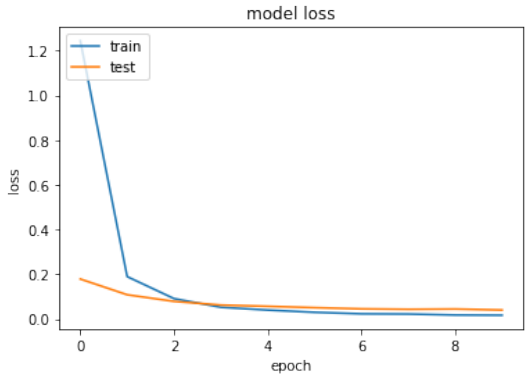


Figure A37: Loss Curve of MLP 0 on CNN1 for 10 epochs with a learning rate of 0.0001

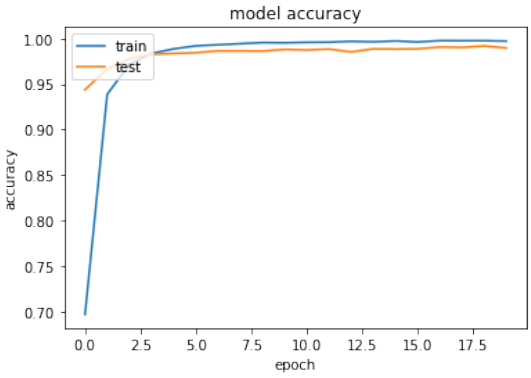


Figure A38: Accuracy Curve of MLP 0 on CNN1 for 20 epochs with a learning rate of 0.0001

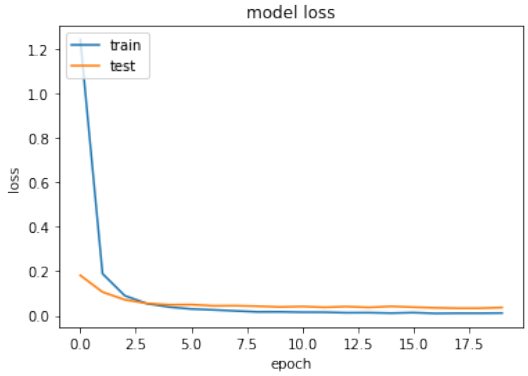


Figure A39: Loss Curve of MLP 0 on CNN1 for 20 epochs with a learning rate of 0.0001

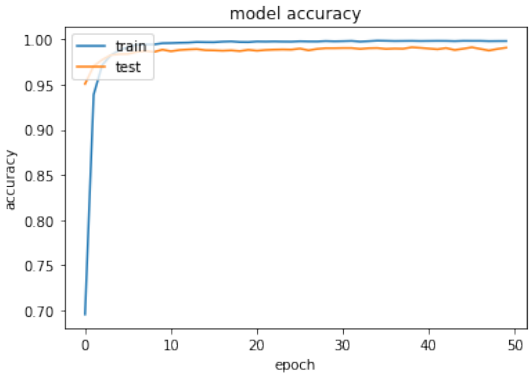


Figure A40: Accuracy Curve of MLP 0 on CNN1 for 50 epochs with a learning rate of 0.0001

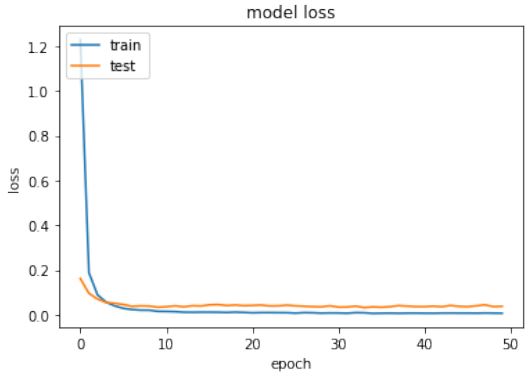


Figure A41: Loss Curve of MLP 0 on CNN1 for 50 epochs with a learning rate of 0.0001

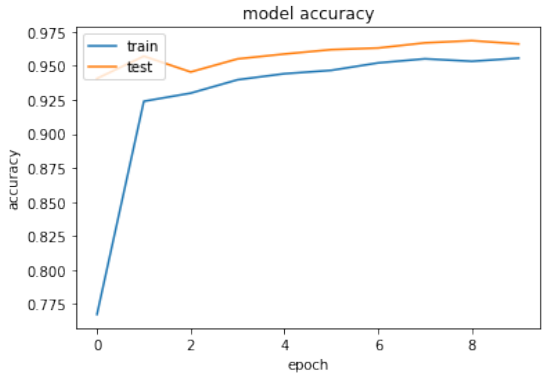


Figure A42: Accuracy Curve of MLP 0 on CNN1 for 10 epochs with a learning rate of 0.001

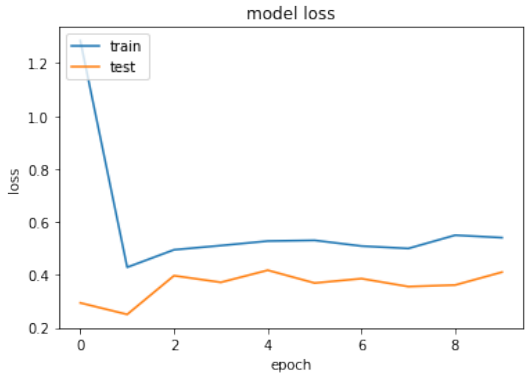


Figure A43: Loss Curve of MLP 0 on CNN1 for 10 epochs with a learning rate of 0.001

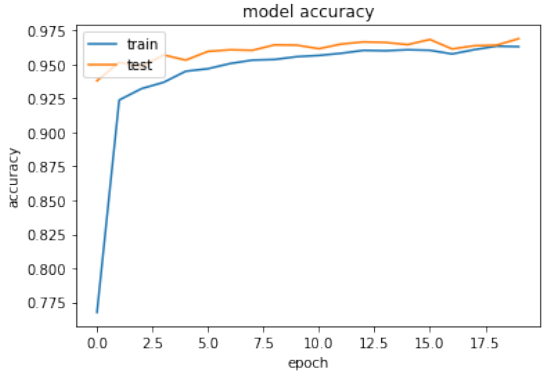


Figure A44: Accuracy Curve of MLP 0 on CNN1 for 20 epochs with a learning rate of 0.001

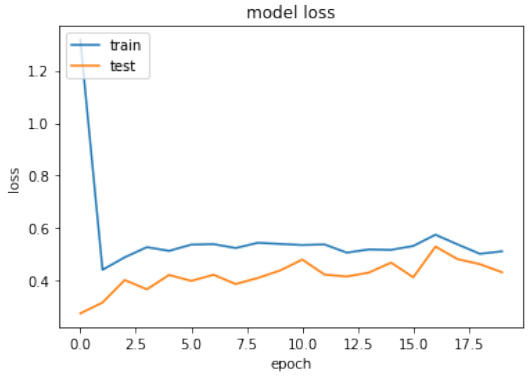


Figure A45: Loss Curve of MLP 0 on CNN1 for 20 epochs with a learning rate of 0.001

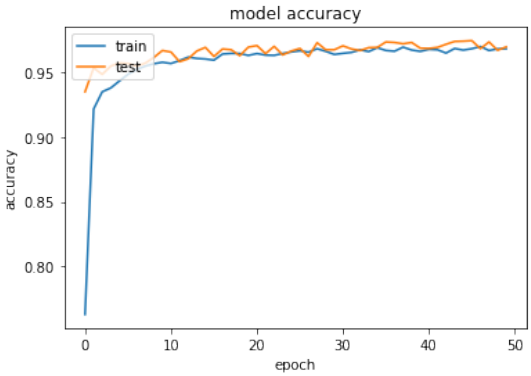


Figure A46: Accuracy Curve of MLP 0 on CNN1 for 50 epochs with a learning rate of 0.001

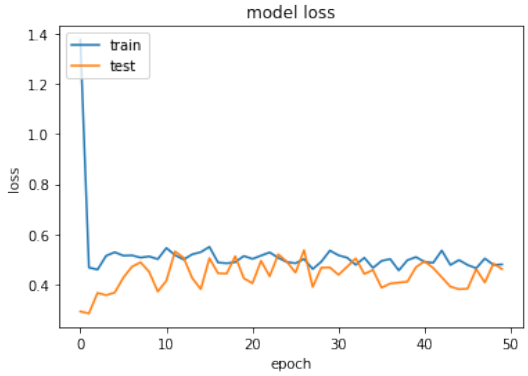


Figure A47: Loss Curve of MLP 0 on CNN1 for 50 epochs with a learning rate of 0.001

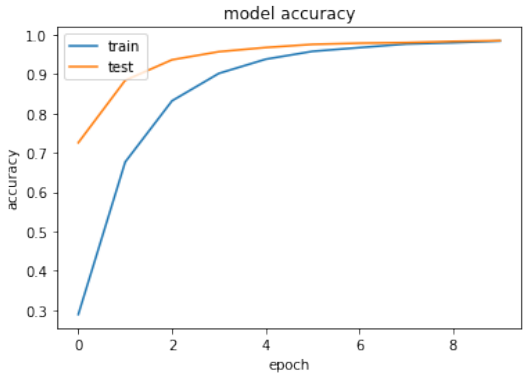


Figure A48: Accuracy Curve of MLP 1 on CNN1 for 10 epochs with a learning rate of 0.0001

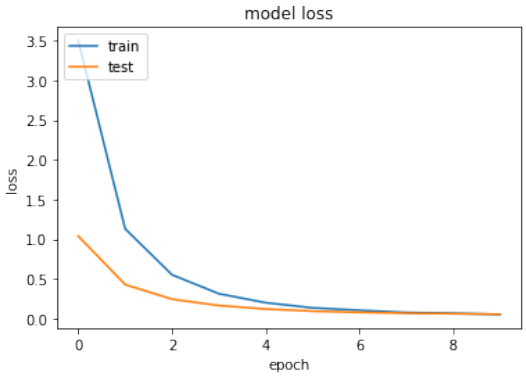


Figure A49: Loss Curve of MLP 1 on CNN1 for 10 epochs with a learning rate of 0.0001

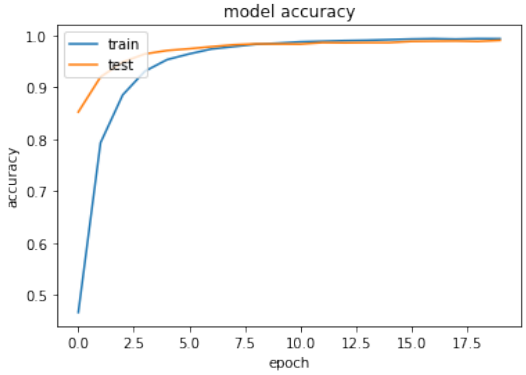


Figure A50: Accuracy Curve of MLP 1 on CNN1 for 20 epochs with a learning rate of 0.0001

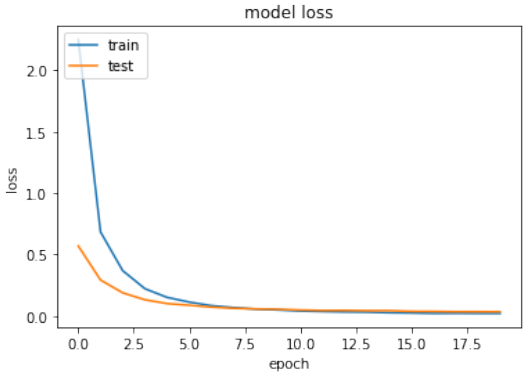


Figure A51: Loss Curve of MLP 1 on CNN1 for 20 epochs with a learning rate of 0.0001

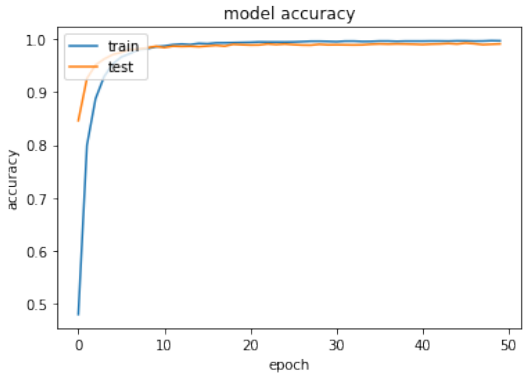


Figure A52: Accuracy Curve of MLP 1 on CNN1 for 50 epochs with a learning rate of 0.0001

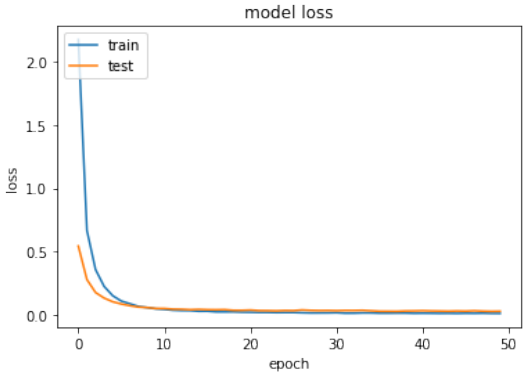


Figure A53: Loss Curve of MLP 1 on CNN1 for 50 epochs with a learning rate of 0.0001

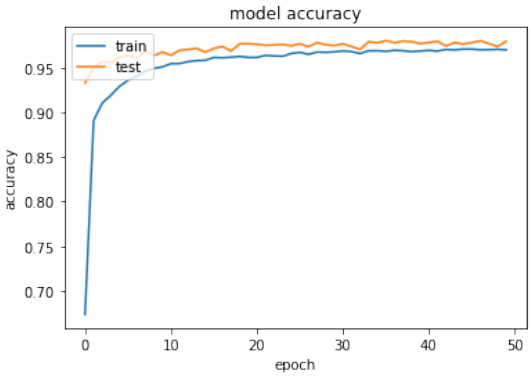


Figure A54: Accuracy Curve of MLP 1 on CNN1 for 50 epochs with a learning rate of 0.001

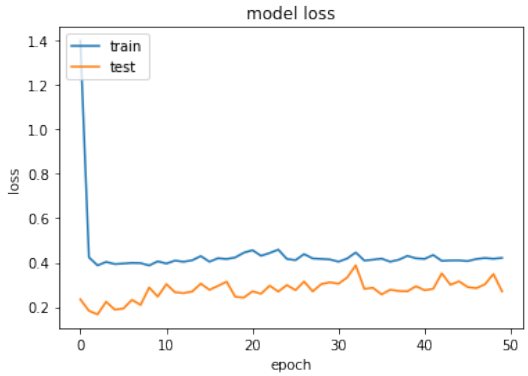


Figure A55: Loss Curve of MLP 1 on CNN1 for 50 epochs with a learning rate of 0.001

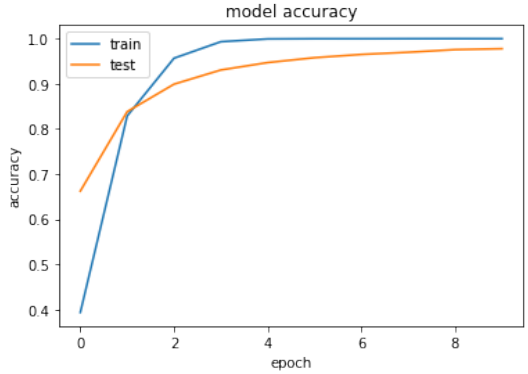


Figure A56: Accuracy Curve of MLP 2 on CNN1 for 10 epochs with a learning rate of 0.0001

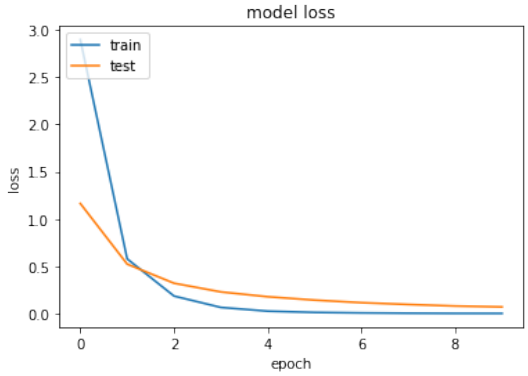


Figure A57: Loss Curve of MLP 2 on CNN1 for 10 epochs with a learning rate of 0.0001

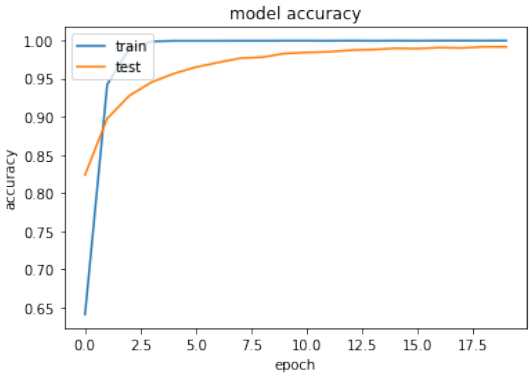


Figure A58: Accuracy Curve of MLP 2 on CNN1 for 20 epochs with a learning rate of 0.0001

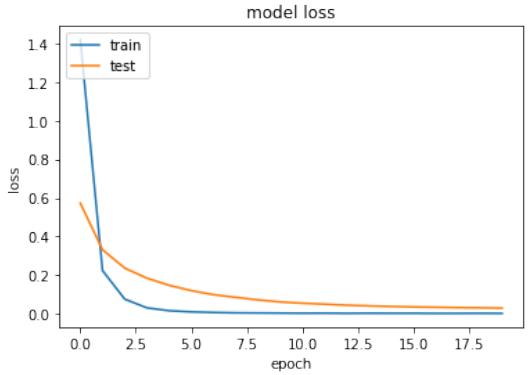


Figure A59: Loss Curve of MLP 2 on CNN1 for 20 epochs with a learning rate of 0.0001

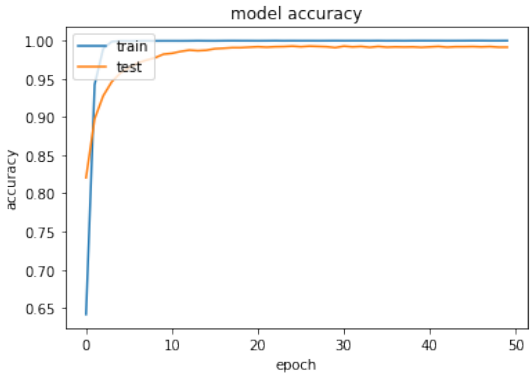


Figure A60: Accuracy Curve of MLP 2 on CNN1 for 50 epochs with a learning rate of 0.0001

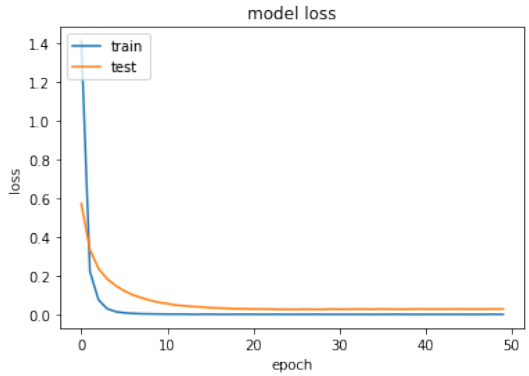


Figure A61: Loss Curve of MLP 2 on CNN1 for 50 epochs with a learning rate of 0.0001

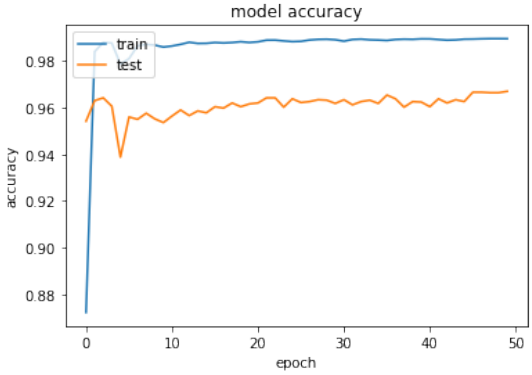


Figure A62: Accuracy Curve of MLP 2 on CNN1 for 50 epochs with a learning rate of 0.001

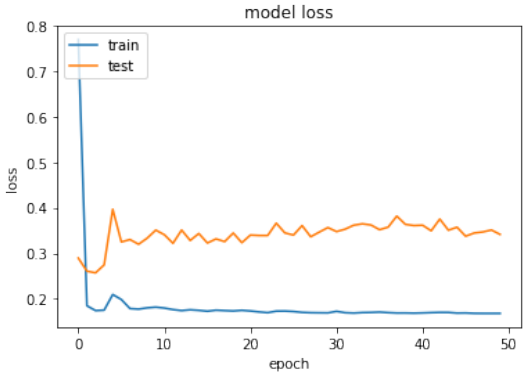


Figure A63: Loss Curve of MLP 2 on CNN1 for 50 epochs with a learning rate of 0.001

A.3.2.3 Results of classification algorithms on CIFAR-100 intermediate data

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.1, penalty: $\ell 2$
KNN	neighbours: 1
RFE	100 estimators, entropy criterion
ADB	100 estimators, LR: 0.1
GBC	100 estimators, LR: 0.05
XGB	100 estimators, LR: 0.1, max. depth: 5

Table A13: CIFAR-100 final classification results on CNN-1 intermediate data

model	best parameters on test
MLP	MLP-1, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 1
RFE	100 estimators, entropy criterion
ADB	100 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1, max. depth: 1

Table A14: CIFAR-100 final classification results on CNN-2 intermediate data

model	best parameters on test
MLP	MLP-1, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.1, penalty: $\ell 2$
KNN	neighbours: 10
RFE	100 estimators, entropy criterion
ADB	100 estimators, LR: 0.1
GBC	100 estimators, LR: 0.05
XGB	100 estimators, LR: 0.1, max. depth: 5

Table A15: CIFAR-100 final classification results on SimpleNet intermediate data

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 1, γ: 0.001, kernel: rbf
LR	C: 0.1, penalty: $\ell 2$
KNN	neighbours: 10
RFE	1000 estimators, gini criterion
ADB	100 estimators, LR: 0.1
GBC	100 estimators, LR: 0.05
XGB	100 estimators, LR: 0.1, max. depth: 5

Table A16: CIFAR-100 final classification results on VGG-19 intermediate data

A.3.2.4 Validation data performance from best parameters on CIFAR-100 with with significance tests

In the diagonal lines of the tables, the mean accuracy of the validation data over 10 cross-validation folds is displayed. In the other cells, the p-value of the comparison of the two indicated networks is given. The p-value is calculated for the validation results during the 10-fold cross-validation. Since the comparison in tables A17, A18, A19, A20, A21, A22, A23 and A24 are symmetric, the p-value is only displayed for those values where the estimated population mean of the network in the column name is higher than the estimated population mean of the network in the row name that it is compared to. The other corresponding cell is left empty.

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9896**					
MLP-2_LR-0.0001	0.495178	**0.9884**	0.911870			
MLP-3_LR-0.0001	0.768048		**0.9887**			
MLP-1_LR-0.001	0.000041	0.000170	0.000776	**0.9680**	0.117228	0.964277
MLP-2_LR-0.001	0.000948	0.002707	0.006929		**0.9738**	
MLP-3_LR-0.001	0.000022	0.000117	0.000652		0.111027	**0.9681**

Table A17: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on CNN-1 intermediate data from CIFAR-100 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.4494**					
MLP-2_LR-0.0001	0.047952	**0.4402**				
MLP-3_LR-0.0001	0.000395	0.009172	**0.4235**			
MLP-1_LR-0.001	0.000030	0.000419	0.019135	**0.4102**		
MLP-2_LR-0.001	0.000003	0.000013	0.000069	0.000391	**0.3762**	
MLP-3_LR-0.001	0.000000	0.000001	0.000004	0.000016	0.266753	**0.3713**

Table A18: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on CNN-2 intermediate data from CIFAR-100 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9838**					
MLP-2_LR-0.0001	0.000016	**0.9589**				
MLP-3_LR-0.0001	0.000001	0.000483	**0.9417**	0.785663		0.064169
MLP-1_LR-0.001	0.000064	0.010291		**0.9430**		0.123347
MLP-2_LR-0.001	0.000006	0.000237	0.014039	0.029564	**0.9275**	0.006670
MLP-3_LR-0.001	0.003429	0.622582				**0.9557**

Table A19: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on SimpleNet intermediate data from CIFAR-100 with different learning rates (LR)

	MLP-1_LR-0.0001	MLP-2_LR-0.0001	MLP-3_LR-0.0001	MLP-1_LR-0.001	MLP-2_LR-0.001	MLP-3_LR-0.001
MLP-1_LR-0.0001	**0.9962**	0.143564	0.046350			
MLP-2_LR-0.0001		**0.9968**	0.629954			
MLP-3_LR-0.0001			**0.9969**			
MLP-1_LR-0.001	0.000577	0.000032	0.000008	**0.9940**	0.007765	0.122792
MLP-2_LR-0.001	0.133959	0.004566	0.000736		**0.9955**	
MLP-3_LR-0.001	0.000872	0.000008	0.000000		0.036959	**0.9948**

Table A20: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for MLP-1, MLP-2, MLP-3 on VGG-19 from CIFAR-100 intermediate data with different learning rates (LR)

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9884**		0.017690					
SVM	0.011099	**0.9820**	0.000015					
LR			**0.9935**					
KNN	0.000000	0.000000	0.000000	**0.3769**			0.000891	0.000000
RFE	0.000000	0.000000	0.000000	0.002812	**0.3576**		0.000174	0.000000
ADB	0.000000	0.000000	0.000000	0.000000	0.000000	**0.2482**	0.000001	0.000000
GBC	0.000000	0.000000	0.000000				**0.4369**	0.009697
XGB	0.000000	0.000000	0.000000					**0.4721**

Table A21: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on CNN-1 intermediate data from CIFAR-100

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.4494**	0.007095						0.000000
SVM		**0.4576**						0.000000
LR	0.011338	0.000003	**0.4417**					0.000000
KNN	0.000000	0.000000	0.000000	**0.2666**	0.017161		0.000017	0.000000
RFE	0.000000	0.000000	0.000000		**0.2793**		0.000028	0.000000
ADB	0.000000	0.000000	0.000000	0.000001	0.000000	**0.1732**	0.000001	0.000000
GBC	0.000699	0.000317	0.001287				**0.3885**	0.000000
XGB								**0.7744**

Table A22: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on CNN-2 intermediate data from CIFAR-100

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9838**		0.000140					
SVM	0.031662	**0.9801**	0.000002					
LR			**0.9937**					
KNN	0.000000	0.000000	0.000000	**0.8380**				
RFE	0.000000	0.000000	0.000000	0.000173	**0.8258**			
ADB	0.000000	0.000000	0.000000	0.000000	0.000000	**0.5165**		0.000000
GBC	0.000000	0.000000	0.000000	0.000000	0.000000	0.000011	**0.3885**	0.000000
XGB	0.000000	0.000000	0.000000	0.000000	0.000000			**0.7744**

Table A23: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on SimpleNet intermediate data from CIFAR-100

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.9968**	0.538834	0.844880	0.780704	0.837241			
SVM		**0.9972**						
LR		0.591208	**0.9969**	0.916699	1.000000			
KNN		0.672895		**0.9970**				
RFE		0.435668	1.000000	0.889494	**0.9969**			
ADB	0.003409	0.002797	0.003162	0.003104	0.003087	**0.9854**	0.033907	0.006504
GBC	0.000145	0.000010	0.000059	0.000058	0.000022		**0.9911**	0.000098
XGB	0.009915	0.000921	0.004280	0.003928	0.002136			**0.9948**

Table A24: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on VGG-19 intermediate data from CIFAR-100

A.3.3 Subset of ILSVRC-2012

A.3.3.1 Best parameters of classifiers on high-level image features

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 10
RFE	1000 estimators, gini criterion
ADB	1000 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1, max. depth: 1

Table A25: ILSVRC-2012 final classification results on Inception ResNet V2 intermediate data

model	best parameters on test
MLP	MLP-1, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 1
RFE	1000 estimators, entropy criterion
ADB	1000 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1, max. depth: 1

Table A26: ILSVRC-2012 final classification results on Inception V3 intermediate data

model	best parameters on test
MLP	MLP-2, LR: 0.0001
SVM	C: 10, γ: 0.0001, kernel: rbf
LR	C: 0.001, penalty: $\ell 2$
KNN	neighbours: 10
RFE	1000 estimators, entropy criterion
ADB	1000 estimators, LR: 0.1
GBC	100 estimators, LR: 0.1
XGB	100 estimators, LR: 0.1, max. depth: 1

Table A27: ILSVRC-2012 final classification results on Xception intermediate data

A.3.3.2 Comparison of best MLP architectures with other classification algorithms

In the diagonal lines of the tables, the mean accuracy of the validation data over 10 cross-validation folds is displayed. In the other cells, the p-value of the comparison of the two indicated networks is given. The p-value is calculated for the validation results during the 10-fold cross-validation. Since the comparison in tables A28, A29 and A30 are symmetric, the p-value is only displayed for those values where the estimated population mean of the network in the column name is higher than the estimated population mean of the network in the row name that it is compared to. The other corresponding cell is left empty.

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.8459**		0.066856		0.0385599			
SVM	0.004757	**0.7334**	0.004751	0.004816	0.004160			0.011838
LR			**0.8632**					
KNN	0.640929		0.061322	**0.8451**	0.035040			
RFE			0.966227		**0.8629**			
ADB	0.000267	0.000635	0.000303	0.000267	0.000277	**0.2347**	0.008479	0.000340
GBC	0.002082	0.004832	0.002029	0.002090	0.001969		**0.4549**	0.002602
XGB	0.013122		0.013178	0.013648	0.009488			**0.8117**

Table A28: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on Inception ResNet V2 intermediate data from ILSVRC-2012 with 100 classes

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.7760**		0.013603		0.014036			
SVM	0.000677	**0.3168**	0.000295	0.000427	0.000286		0.005692	0.000598
LR			**0.8459**					
KNN	0.090350		0.000589	**0.7509**	0.000207			
RFE			0.004382		**0.8157**			
ADB	0.000732	0.009356	0.000476	0.000628	0.000473	**0.1341**	0.002601	0.000718
GBC	0.000846		0.000186	0.000298	0.000166		**0.4515**	0.000651
XGB	0.092342		0.003518	0.490825	0.003357			**0.7461**

Table A29: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on Inception V3 intermediate data from ILSVRC-2012 with 100 classes

	MLP	SVM	LR	KNN	RFE	ADB	GBC	XGB
MLP	**0.82453**		0.016872		0.029549			
SVM	0.014604	**0.74827**	0.007258	0.023661	0.008464			0.099931
LR			**0.85280**					
KNN	0.492943		0.037980	**0.81840**	0.066340			
RFE			0.037032		**0.84293**			
ADB	0.000200	0.000466	0.000168	0.000288	0.000166	**0.18467**	0.005867	0.000216
GBC	0.000970	0.001933	0.000825	0.001180	0.000844		**0.38853**	0.001192
XGB	0.005395		0.001140	0.023656	0.000777			**0.77440**

Table A30: Pairwise p-values of performance on validation set with performance on test and validation set displayed on the diagonal for different classification algorithms on Xception intermediate data from ILSVRC-2012 with 100 classes