Comparing Vision Transformers to Traditional Deep Learning Object Classification Architectures for Automotive Make and Model Recognition

Master's Thesis

Name of the study program

Business Informatics

Faculty 4

submitted by

Max Grundmann

Date of submission:

Berlin, 03.04.2023

First Supervisor: Prof. Dr. Martin Spott
Second Supervisor: Konstantin Müller

Abstract

The Transformer architecture has recently taken over the domain of natural language processing and is now quickly gaining popularity in computer vision, achieving state-of-the-art results on image classification benchmarks like ImageNet. Vision Transformers (ViT) work by dividing an image into fixed-size patches that can be processed as a sequence of tokens. Self-attention is used to compute the importance of these tokens in relation to each other, allowing the model to attend to different regions of the image and model long-range dependencies. This architecture has proven to work well on large datasets and shows better scaling performance than comparable convolutional neural networks (CNN). However, its performance on medium-sized datasets remains unclear.

This thesis set out to study the performance potential and limitations of the Vision Transformer architecture compared to the dominant convolutional architecture on a medium-sized dataset for vehicle model classification. For reference, an Inception ResNet v2 model provides a baseline for accuracy on this task. After a review of the basic concepts and building blocks of both architectures, a selection of promising, improved architectures is assembled, with each tackling one of the most prominent weaknesses of the Transformer: DeiT solves the reliance of ViT on large datasets for pertaining by introducing a distillation strategy that can utilise pre-trained CNNs to inject some inductive biases that Transformers are naturally lacking. Next, Swin introduces an approach using shifted windows and a hierarchical design, inspired by convolutional networks, solving the issue of quadratic scaling with input resolution. Finally, FlexiViT shows how fixed-sized patch embeddings can be made dynamic, making ViTs easy to adapt to specific computational budgets by allowing dynamic patch sizes as inputs. A similar selection was made for top-performing convolutional networks as well as hybrid networks, which have been evaluated on a dataset of 42.000 Mercedes-Benz vehicle images.

The evaluation results show very competitive results for nearly all tested architectures, with the Swin-T/8 v2 network achieving the highest accuracy with 96,3%, closely followed by ConvNeXt-Base, a convolutional model, with an accuracy of 95,6%. Both models are able to outperform the previous Inception ResNet v2 baseline, with the largest improvements seen in the representation of minority classes in this imbalanced dataset, manifesting in an 8,6% improvement in the F1-score over the baseline model (81,6% to 90,2%). In terms of inference speed, convolutional models are faster (48,1ms for Swin-T/8 v2 versus 29,1ms for ConvNeXt-Base), but trade-offs can be made to bring inference times down, resulting in only slightly reduced accuracy. Training times are also generally slower but more sample-efficient.

In conclusion, Transformers have demonstrated competitive performance by offering slightly superior accuracy at the cost of slower image throughput and slower training times. Both the best-performing CNN and Transformer easily surpass the previous baseline model, especially for minority classes. Since the real-world performance of these models has proven to be near-identical, a final decision would slightly favour the convolutional model for its faster inference speed.

Abstract German

Die Transformer-Architektur hat in den letzten Jahren das Gebiet des Natural Language Processing erobert und gewinnt nun auch im Bereich des Maschinellen Sehens schnell an Bedeutung, indem sie state-of-the-art Ergebnisse in Bildklassifikations-Benchmarks wie ImageNet erzielt. Vision Transformers (ViT) teilen Bilder in Patches mit fester Größe auf, die als Sequenz von Tokens verarbeitet werden können. Mithilfe von Self-Attention kann das Modell lernen, sich auf bestimmte Bildbereiche zu fokussieren und globale Abhängigkeiten zu modellieren. Die Transformer-Architektur konnte sich bereits für große Datensätzen bewähren und zeigt bessere Skalierungseigenschaften als vergleichbare Convolutional Neural Networks (CNNs). Die Performance auf mittelgroßen Datensätzen ist jedoch unklar.

Diese Thesis zielt darauf ab, das Leistungspotenzial und die Grenzen der Vision Transformer-Architektur im Vergleich zur dominierenden Convolutional-Architektur auf einem mittelgroßen Datensatz für die Fahrzeugmodellklassifikation zu untersuchen. Als Referenz dient ein Inception ResNet v2-Model. Nach einer Einführung der grundlegenden Konzepte und Bausteine beider Architekturen wird eine Auswahl vielversprechender, verbesserter Netzwerke vorgestellt, wobei jedes eine der prominentesten Schwächen des Transformers aufgreift: DeiT löst die Abhängigkeit von ViT von großen Datensätzen für die Generalisierung durch Einführung eines Distillationsverfahrens, das ein vortrainiertes CNN verwendet, um induktive Neigungen (inductive biases) beizufügen, die dem Transformer traditionell fehlen. Als Nächstes führt Swin einen Ansatz ein, der verschobene Fenster und eine hierarchische, von ConvNets inspirierte, Architektur verwendet, um das Problem der quadratischen Skalierung mit der Bildauflösung zu beheben. Zum Schluss zeigt FlexiViT, wie feste Patch-Embeddings dynamisch gestaltet werden können, wodurch ViTs einfacher auf bestimmte Rechenbudgets angepasst werden können. Eine ähnliche Auswahl wurde auch für die besten Convolutional Neural Networks sowie Hybrid-Netzwerke getroffen, die anschließend auf einem Datensatz von 42.000 Mercedes-Benz-Fahrzeugbildern ausgewertet wurden.

Die Ergebnisse zeigen, dass fast alle getesteten Architekturen sehr konkurrenzfähige Ergebnisse erzielen können. Swin-T/8 v2 erreicht die höchste Genauigkeit mit 96,3%, gefolgt von ConvNeXt-Base, einem CNN, mit einer Genauigkeit von 95,6%. Beide Modelle sind in der Lage, das frühere Inception ResNet v2-Baseline-Modell zu übertreffen, wobei die größten Verbesserungen in der Abbildung von Minderheitsklassen zu sehen sind, was sich in einer Verbesserung des F1-Scores gegenüber dem Basismodell um 8,6% (von 81,6% auf 90,2%) manifestiert. In Bezug auf die Inferenzgeschwindigkeit zeigen sich CNNs schneller (48,1ms für Swin-T/8 v2 gegenüber 29,1ms für ConvNeXt-Base), wobei es möglich ist, Kompromisse in der Genauigkeit einzugehen, um schnellere Inferenzzeiten zu ermöglichen. Die Trainingszeiten sind ebenfalls im Allgemeinen etwas langsamer, aber dateneffizienter.

Zusammenfassend konnte gezeigt werden, dass Transformer eine konkurrenzfähige Leistung gegenüber Convolutional Neural Networks erreichen können, wobei sie eine geringfügig höhere Genauigkeit bei langsamerem Datendurchsatz und langsameren Trainingszeiten bieten. Sowohl das am besten abschneidende CNN als auch beste Transformer Modell übertreffen das frühere Baseline-Modell in nahezu allen Belangen, insbesondere in der Genauigkeit der Vorhersage von Minderheitsklassen. Da die Genauigkeit beider Modelle unter realen Bedingungen praktisch identisch ist, würde dank der etwas schnelleren Inferenzzeit die finale Entscheidung auf das Convolutional Neural Network fallen.

Contents

List of Figures

List of Tables

List of Equations

1 Introduction

1.1 Motivation

In recent years, the Transformer architecture has emerged as the state-of-the-art architecture in the field of natural language processing (NLP), even gaining mainstream attention through its use in large language models like GPT3/4 and its now famous offshoot chatGPT. At the same time, I started to notice that the Transformer architecture was also gaining popularity for computer vision applications. Some researchers even hypothesise that its general-purpose computing capabilities could make this the one uniting architecture for most if not all domains of deep learning [2].

> *The ongoing consolidation in AI is fascinating. For a long time we had different architectures for different areas (e.g. CNNs for vision, RNNs/LSTMs/GRUs for NLP/audio). But as of approx. last two years, even the neural net architectures across all areas are starting to look identical - a Transformer (definable in 200 lines of PyTorch), with very minor differences. Either as a strong baseline or (often) state of the art.*
>
> — Andrej Karpathy [2]

However, despite early claims of state-of-the-art results on popular image classification benchmarks like ImageNet, the Transformer is still not considered generally superior to the best-performing convolutional networks [20]. Especially early results required enormous amounts of data for the network to even match the performance of traditional architectures, let alone exceed them. But this might be about to change, as an explosion in research has led to a number of proposals aiming to make the Vision Transformer more data efficient, faster, more robust, and critically, more accurate than traditional convolutional neural networks.

This sparked my initial interest in the topic and during my internship at Mercedes-Benz Mobility, I was fortunate enough to find the perfect use case to study this new architecture on: The model recognition service uses a deep learning model to predict which Mercedes-Benz model is pictured in an image. This presents a challenging task as far as image classification goes due to a large number of classes and the oftentimes minimal visual differences between models. Furthermore, this use case allows for a direct comparison of currently deployed, traditional models with the Transformer architecture and provides a comprehensive internal dataset, making this

a suitable use case to study the Transformer architecture under real-world conditions. Additionally, the model recognition use case has started to receive increased attention within the company and its growing user base makes the inference performance and overall cost of training and deployment more of a consideration.

In summary, I am not primarily motivated by the assumption that the Vision Transformer will necessarily yield superior results to state-of-the-art convolutional networks. Instead, I see this thesis as a challenge: study the relevancy of a cutting-edge architecture on a demanding dataset under conditions that are known to be sub-optimal for the Transformer architecture.

1.2 Geofence and Selfaudit

The motivation and background for using machine learning for model recognition at Mercedes-Benz Mobility can be better understood by looking at how a typical Mercedes-Benz dealership will acquire vehicles. Instead of buying them outright, which would tie up a lot of capital, a dealership will typically finance its inventory stock, like demonstrator vehicles. This is often done through the Mercedes-Benz Bank, which will remain the legal owner of the vehicle until it is sold to a customer. Once the dealership has received the payment it can then proceed to pay off the loan at Mercedes-Benz Bank. However, this process is susceptible to fraud. For example, the dealer could attempt to use a single car as collateral to take out multiple loans from multiple banks at once or withhold the payment even after selling the vehicle to a customer. Therefore, the bank has an interest in making sure that any financed vehicles are managed properly. In addition, banks are also required by law to monitor and audit their assets by financial regulators like the BaFin (Bundesanstalt für Finanzdienstleistungsaufsicht) in Germany. Hence, this creates the demand for Mercedes-Benz Bank to conduct regular audits of its wholesale inventory. [1]

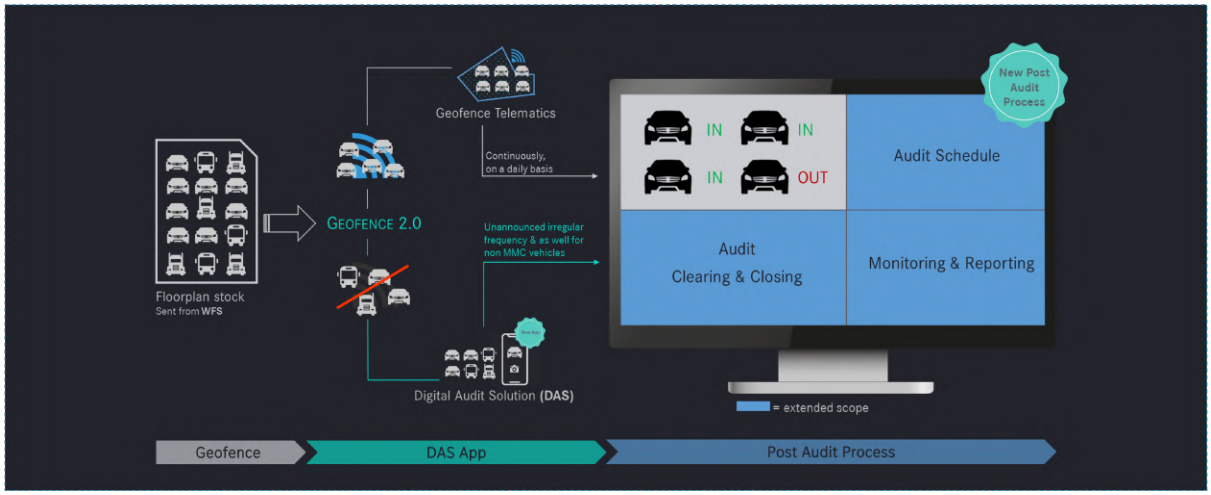


Figure 1.1: Geofence and Selfaudit Process Flow [1]

In the past, audits were performed by external auditors, racking up costs while also being time-consuming for the dealerships. To alleviate these pain points, the Geofence portal and Digital Selfaudit app (DAS) were launched in 2019. The solution aims to reduce costs and workload for all parties by removing the need for external auditors and fully digitising the auditing process. The tool centralises the entire inventory stock within a market, allows the automated scheduling of audits and enables better collaboration between departments by adding the ability to easily share audits and add notes while also improving transparency throughout the process. [1]

For any given vehicle in an audit, there are two possible process flows, as illustrated in figure 1.1:

Geofence

Geofence attempts to reduce the total number of vehicles that need to be audited by utilising the built-in telematics equipment and data of modern Mercedes-Benz vehicles, like GPS modules and cellular connectivity. As the name Geofence suggests, a virtual geographic fence can be constructed around the perimeter of the dealership. If the telematics data confirms that the vehicle is indeed inside the fence, then the vehicle status within Geofence is set to "in" and thus does not require any further auditing. However, in the scenario that a vehicle leaves the dealership, say for a test drive, the telematics tracking has to be deactivated due to privacy reasons as location data is considered personal identifiable information (PII). These vehicles receive the status "out" and will need to be audited manually. [1]

Digital Auditing Solution

In case a vehicle can not be automatically confirmed to be within the fence, a manual audit is required. However, this audit is now handled by the dealership itself using the Digital Auditing Solution app. The app will present a list of vehicles that have to be audited within a restricted period of time, ranging from one to five days, depending on the specifics of the market and the number of vehicles. For each vehicle, the dealer will be prompted to take three images: The first image is a front-on shot of the car, the second is an image of the vehicle identification number (VIN) sticker (usually in the door seal) and the third image needs to show the current mileage of the vehicle, thus it is usually an image of the dashboard screen. These images are used to verify that the dealership is indeed still in possession of the car. [1]

As checking these images for every audited vehicle manually would be very time-consuming, a machine learning service, called Vehicle Recognize, is used to automatically validate these images. This is where the model recognition and the use case for this thesis come into play, as machine learning is used to verify that, among other checks, the vehicle depicted in the exterior photo is actually the model the system is expecting it to be. Similar checks are performed to validate that the VIN sticker shows the correct VIN, which is arguably the most important

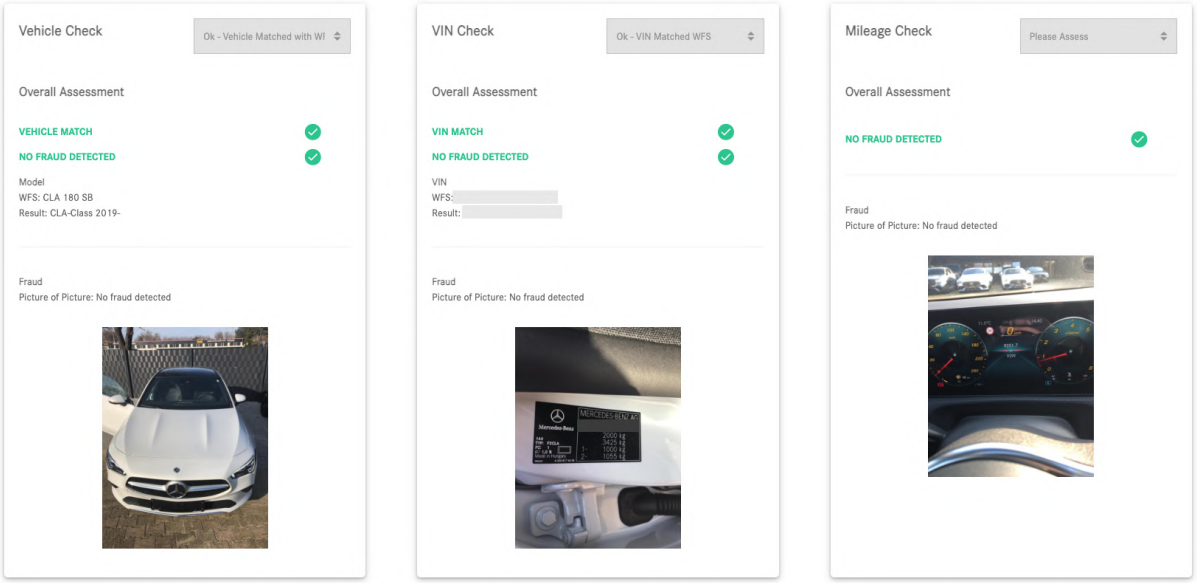


Figure 1.2: Findings for an audit in the Geofence portal

predictor of a matching vehicle and that the mileage image is valid. Upon completion of these checks, the images and the corresponding results are returned back to the Geofence portal (see figure 1.2) where they can also be re-checked manually if needed. [1]

Business Impact

As of February 2023, a total of 850 individual dealerships in nine markets across the world are using Geofence. This results in approximately 64.000 vehicles being tracked by the system, 1/3 of which are automatically validated using the telematics data. While the exact number of audits per dealership can vary, it typically falls in the range of one to four audits per year, possibly more. [1]

The omission of external auditors has been the biggest factor in cost savings. However, the exact savings depend heavily on the markets themselves as well as the size of the dealership network. For the German market, the cost advantage compared to previous audits is estimated as approximately 250.000€ per year, without taking any additional time-savings post-audit as a result of more streamlined digital processes into account. Globally, bigger markets like the US are expected to offer cost-savings in excess of two million euros per year. [1]

1.3 Mercedes-Benz Model Recognition

The model recognition was first integrated into Geofence and Self-Audit in 2019 and has since been an integral part of the auditing process as part of a solution called Vehicle Recognize. The

service exposes a REST-API that takes as arguments the image that should be classified as well as the vehicle identification number (VIN) of the vehicle that is being audited. The VIN encodes the model code that can be used to determine the true model of the vehicle so that it can be checked against the predicted class. The endpoint will respond with the detected model class and whether or not the prediction matches the given VIN. The results of this evaluation are displayed in the Geofence portal. Incorrect audits can be checked manually by an employee while automatically validated audits do not require any further attention. [1]

The model recognition employs a deep learning image classification model to predict the exact model that is pictured in a given image. This model is retrained regularly to support newly released Mercedes Benz models and to improve the performance of existing classes by increasing the number of training images. Currently, a total of 64 distinct models are supported and recognised.

When deployed to production, the detection may be limited to fewer classes if their detection accuracy is deemed too low. In 2022, the hosting of all Vehicle Recognize endpoints was migrated onto a higher-performance, GPU-enabled hosting platform on Microsoft Azure. A Kubernetes cluster, controlled by Ray Serve as the underlying service, is responsible for managing, serving, and if required, scaling each machine learning model as a separate Docker container deployment. [1]

Additionally, Recognize has recently been established as a standalone solution, enabling other teams within Mercedes Benz Mobility to use and integrate the endpoints into their own applications. In addition to the model recognition endpoint, Recognize also provides other services like optical character recognition to scan VIN sticker images or models that are trained to detect potentially fraudulent audits and inputs, like images taken from a computer monitor or fraudulent vehicle registration documents. [1]

1.4 Scope of the Study

The primary goal of this thesis is to provide valuable insights into the potential and limitations of the Vision Transformer architecture by comparing it to established convolutional neural networks. Among other aspects, this analysis will compare their training efficiency, inference speed and model accuracy. This will specifically include exploring how well Vision Transformers work on a small to medium-sized dataset. Additionally, this study seeks to examine how the architecture handles the domain-specific challenges of this use case and if it offers any benefits in terms of interpretability compared to CNNs. The research will also include possible hybrid architectures and additional optimisation techniques.

Additionally, as part of the practical implementation of this thesis, the existing training pipeline will be migrating from the deprecated tf-slim framework to TensorFlow's new top-level API

Keras. Furthermore, MLFlow will be introduced as a framework for tracking experiments that will allow data scientists to easily track model metrics, compare different experiments and enable reproducibility. However, this will not be referenced further.

1.5 Approach

First, a literature review will establish a basic understanding of the basic concepts and building blocks of each architecture and their respective strengths and weaknesses. It will also include covering the concepts of transfer learning and a number of evaluation metrics. Extending this first literature review, chapter 3 will present a selection of improved network variants based on convolutional, Transformer or hybrid architectures and optimisation techniques.

Chapter 4 will outline the approach and experimental methodology used for implementing, training and evaluating the previously introduced networks on the model recognition images. This will include an exploratory data analysis, highlighting some of the domain-specific challenges of this use case.

The results of the conducted experiments will be evaluated in chapter 5, comparing different network types in terms of accuracy, inference speed, training efficiency and interpretability and showcasing points of failure with examples for misclassification.

Chapter 6 will discuss the findings in the context of the model recognition service and revisit the trade-off between accuracy and inference speed. It will also present an outlook on further research on this topic., before concluding with a final summary of the results in chapter 7.

1.6 Limitations

This thesis does not intend to survey every convolutional or transformer architecture that has been proposed in literature. Instead, it will focus on the most widely used and influential ones, based on their citation counts, performance benchmarks, and availability of pre-trained weights and open-source implementation for Keras / TensorFlow. This choice is motivated by the fact that it is next to impossible to cover all the existing models in a single thesis, given the rapid pace of research and development in this field. Moreover, some models are not publicly available or well-documented, making it difficult to reproduce or compare them with others. Therefore, this thesis should not be expected to be a comprehensive study of every possible model architecture or optimisation technique available, but rather a selective and critical analysis of some of the most prominent and promising architectures and their application to the image classification task at hand.

2 Literature Review

Image classification can be defined as "the task of assigning a class label to an image based on its content" [47]. It is one of the fundamental problems in computer vision. Early approaches to solving this task included designing hand-crafted features and rules to recognise simple shapes and objects. However, these methods were limited by their complexity and required extensive domain knowledge. Even still, they could not scale to more realistic and diverse images. The breakthrough came in the late 2000s, when a deep learning method, namely the convolutional neural network (CNN), emerged as a powerful and flexible model for learning high-level features from raw pixel data. [64]

One of the key contributions that enabled the success of CNNs was the availability of large-scale annotated image datasets, such as ImageNet [64]. ImageNet is a project that aims to create a comprehensive visual database for visual object recognition [12]. It contains more than 14 million images of more than 20,000 categories derived from WordNet [12]. ImageNet also hosts an annual challenge, called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where participants compete to achieve the best performance on various image classification tasks using a subset of 1.3 million images [12]. The ILSVRC has been widely regarded as a benchmark for measuring the progress of image classification methods. [64]

However, as of recently, CNNs are not the only models that can perform well on image classification tasks. Another type of model that originated from natural language processing (NLP), called Transformer, has been adapted to handle images, also achieving state-of-the-art results on the ImageNet benchmark. [64, 13]

This chapter will review the literature on both CNNs and Transformers for image classification by covering the basic concepts and components of each and highlighting some of their strengths and weaknesses. Finally, this chapter will conclude with a brief overview of the concept of transfer learning and a summary of important metrics used to evaluate multi-class image classification models.

2.1 Convolutional Neural Networks

When it comes to solving the image classification task, one might try to approach the problem using a multi-layer perception (MLP). But while fully connected layers can be used for both

learning features and classifying images, they require a very high number of learnable parameters. In practice, this makes them very inefficient to train and store. Furthermore, a reliable classifier should also be able to detect relevant objects wherever they are in the image and no matter their orientation. This characteristic is called *translation invariance* or *translation equivariance*. [64] This need motivated the creation of the convolutional neural network (CNN). The architecture was inspired by the visual perception of the human brain, with a so-called convolutional kernel being able to learn to perceive and detect various features and eventually objects [32].

While the concept of convolutions can be traced back to the first iterations of the artificial neural network in 1943, LeCun et al. were the first to use this term in their original version of LeNet in 1989. Their model was used for the recognition of handwritten zip codes on checks and by the end of the 1990s the system using this model handled over 10% of all checks in the United States. [32, 18] The commercial interest in convolutional networks sparked again when Krizhevsky et al. set a new benchmark at the ImageNet object classification benchmark in 2012, finally surpassing previous approaches that relied heavily on hand-crafted features [30, 18].

Compared to fully connected or dense networks, convolutional networks tend to be sample and computationally efficient as they require fewer parameters and because modern graphical processing units (GPUs) are able to efficiently process convolutions in parallel [64]. This has led to the CNN becoming the de-facto default architecture for vision tasks [64, 18, 13]. Apart from images, convolutional networks have also been used to process any data in a grid-like structure, like time-series data, audio or text [18], [64].

The basic architecture of a CNN is inspired by the following thought process:

1. The first few layers of the network should be *translation invariant*, outputting the same response regardless of the location of a specific feature within the image.

2. These early layers should also tend to local regions first (*locality*) and disregard more distant parts of the image. With the increasing depth of the network, local features can be aggregated into more global ones.

3. Similar to the biology of higher-level vision in humans, later layers of the network should be able to extract long-range dependencies. [64]

The following chapters will outline how these characteristics can be achieved and highlight some of the most noteworthy improvements over early iterations of the convolutional architecture.

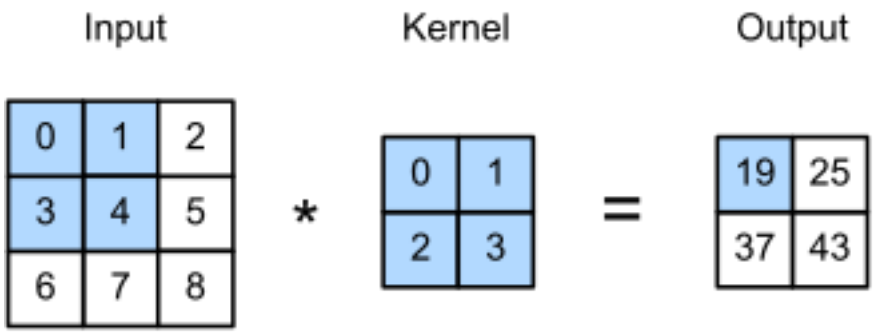


Figure 2.1: Convolution on two-dimensional data with a 2×2 kernel [64]

2.1.1 Convolutional Layer

To achieve the desirable characteristics mentioned above, convolutional networks rely on the eponymous mathematical operation of convoluting a kernel with a given input. More specifically, to establish translation invariance for a two-dimensional image, meaning a shift in input $\mathbf{X}$ should merely result in a shift in the hidden representation $\mathbf{H}$, a filter $[\mathbf{V}]_{a,b}$ is used to weigh pixels at $(i + a, j + b)$ in the neighbourhood of location (i, j): [64]

$$[\mathbf{H}]_{i,j} = u + \sum_{a} \sum_{b} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b} \tag{2.1}$$

a and b act as positive and negative offset indices covering the whole image, u is a constant bias term. Compared to a fully connected layer in a multi-layer perception (MLP), this reduces the number of parameters from 10^{12} down to $4 \cdot 10^6$. [64]

Next, the scope of a convolutional operation should be locally limited to fulfil the second principle: locality. This is achieved by setting $[\mathbf{V}]_{a,b} = 0$ if values fall outside the range $\mid a \mid > \Delta$ or $\mid b \mid > \Delta$: [64]

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b} \tag{2.2}$$

Parameter counts are further reduced to $4\Delta^2$. 2.2 is known as a convolutional layer, with V being known as the kernel or filter that holds the layer's learnable parameters. Strictly speaking, the equation described in 2.2 and implemented in most deep learning frameworks is called cross-correlation and not convolution. Strict convolution would require the operation to be performed with the kernel flipped both horizontally as well as vertically. However, when applied in practice, the learned kernel does not actually change and thus the output remains the same as if only cross-correlation was applied. Nonetheless, it is still commonly referred to as convolution in literature and the industry. [64]

Figure 2.1 visualises the convolutional operation: the kernel of size 2×2 creates a convolutional

window (highlighted in blue) on the input tensor of shape 3×3, starting in the top-left corner. For each window position, the elements of the input are multiplied element-wise with the corresponding value in the kernel and are summed at the end to a scalar value [21]. The window is moved across the input from left to right and top to bottom. The result is a tensor of shape 2×2, as the cross-correlation can only be computed where the whole kernel fits into the input tensor. The output size of a convolution for an input $n_h \times n_w$ and the kernel $k_h \times k_w$ can be calculated as [64]

$$(n_h - k_h + 1) \times (n_w - k_w + 1) \tag{2.3}$$

Expanding the concept to multiple layers, the area that affects the calculation of an element through all previous layers in the future layer $Y\prime$ is called the receptive field of that element. It is usually larger than the size of its input and allows the network to detect features over a larger area. [64]

As a final modification, convolutional networks should be able to handle images with multiple colour channels instead of only black-and-white images. This colour channel can be interpreted as a multidimensional representation of each pixel location, extending $\mathbf{X}$ to $[\mathbf{X}]_{i,j,k}$ and $[\mathbf{V}]_{a,b,c}$. In addition, the hidden layer is also extended to hold a third dimension d, containing a vector of hidden representations per pixel location. This can be visualised as a grid of stacked two-dimensional grids, also referred to as channels or feature maps. The intuition behind this design is to allow different channels to learn different representations to, for example, recognise textures or edges. At last, this means adding this dimension also to the kernel $[\mathbf{V}]_{a,b,c,d}$. A multi-channel convolutional layer is therefore defined as [64]

$$[\mathbf{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathbf{V}]_{a,b,c,d} [\mathbf{X}]_{i+a,j+b,c} \tag{2.4}$$

Figure 2.2 demonstrates this by showcasing one kernel per input channel, two in this example, and how they are first applied individually to each channel and then summed to produce the final output. [64]

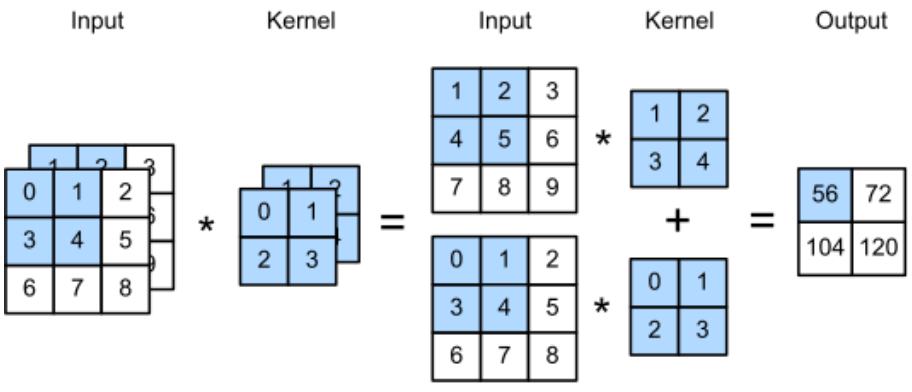


Figure 2.2: Convolution applied to input with two channels [64]

Finally, as mentioned in 2.4, more than one output channel c_o might be desirable to enable the model to learn more representations. Thus the convolutional kernel takes the form $c_o \times c_i \times k_h \times k_w$, where c_i denotes the channel dimension of the input and the output for each output channel is sampled from all input channels. [64]

Padding

As demonstrated by the example in 2.1, the convolution reduces the resolution of its input considerably. This might not be desirable, as doing this repeatedly over multiple layers may reduce the resolution to the point where important details may be lost. A technique to adjust this behaviour is called padding. Especially pixels on the outer edges of the image can get lost, as demonstrated in figure 2.3 for the kernel sizes 1×1, 2×2 and 3×3. [64]

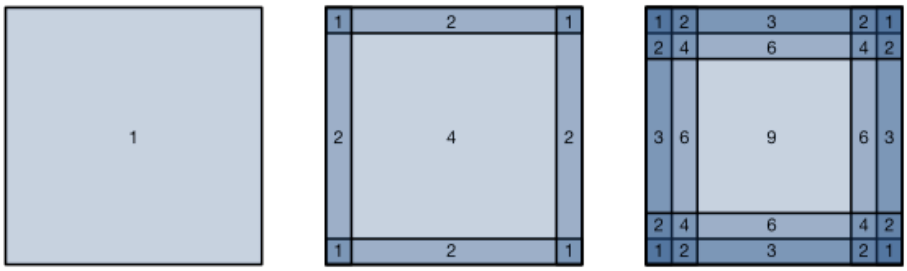


Figure 2.3: Visualisation of how often pixels are sampled without padding for convolutions of size 1×1, 2×2 and 3×3 [64]

The value within each area refers to the number of times that pixel is being sampled from. This effect can stack up with multiple layers so padding combats this by extending the image with extra pixels, with their values usually set to zero [21]. This increases the effective image size, resulting in a 4×4 output for a 2×2 kernel applied on a 3×3. image (2.4). [64]

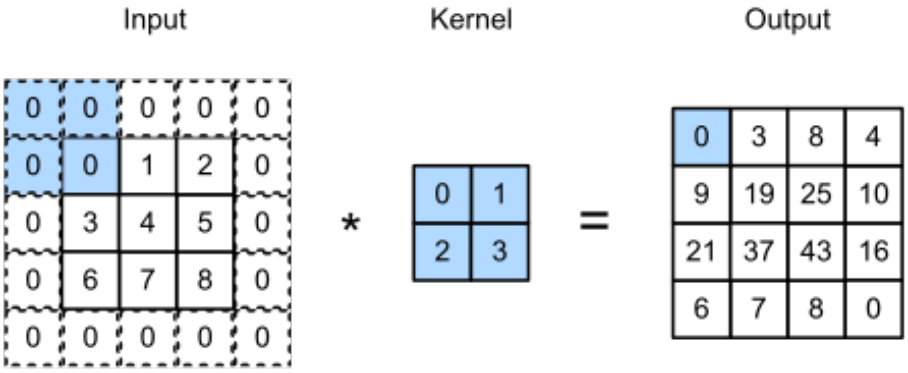


Figure 2.4: 2×2 convolution on a 3×3 image with padding [64]

Specifically, adding p_h rows and p_w columns of padding distributed evenly on the left and right

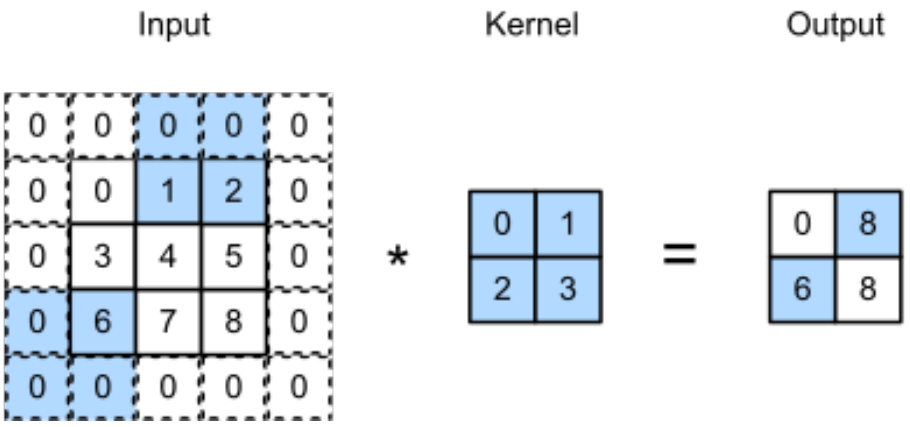


Figure 2.5: Convolution with a 2×2 kernel on a 3×3 image with equal padding and stride $= 1$ [64]

and top and bottom, the output size of the convolution changes by p_h and p_w: [64]

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1) \tag{2.5}$$

p_h and p_w are commonly set to $k_h - 1$ and $k_w - 1$ respectively, to maintain the size of the input. For odd values of k_h, the input will be padded by $p_h/2$ rows on either side. For odd values, either $[p_h/2]$ rows are added on the top and bottom or both sides are padded equally. Common values for kernel sizes include $1, 3, 5$ and 7, allowing equal padding on all sides to preserve the input size. [64]

Stride

The stride of a convolution describes a parameter that controls how far the convolutional window is moved for each step. It can be helpful to increase the computational efficiency or allow for downsampling of the input. Previous examples have used a stride of 1 for both vertical and horizontal movements. Figure 2.5 depicts a convolution with a vertical stride of 2 and a horizontal stride of 3. If the window does not completely fit into the remaining space either vertically or horizontally, its output is skipped. [64]

The output shape for a convolutional layer with vertical stride s_h and horizontal stride s_w is defined as:

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor \tag{2.6}$$

If input width and height are multiples of the stride in both dimensions and $p_h = k_h - 1$ and $p_w = k_w - 1$, the calculation can be simplified to $(n_h/s_h) \times (n_w/s_w)$. [64]

The computational cost of processing an input image of size $(h \times w)$ with a $k \times k$ convolution is $\mathcal{O}(h \cdot w \cdot k^2)$ or $\mathcal{O}(h \cdot w \cdot k^2 \cdot c_i \cdot c_o)$ for input channels c_i and output channels c_o [65]. Newer convolutional architectures propose various methods to reduce the number of computational operations required [65], [36], [57], [62].

2.1.2 Pooling Layer

As explained in the previous section, increasing the number of filters in a given convolutional layer also increases its output dimensionality, indicating more parameters [50], [64]. Additionally, the network could still be sensitive to slight input distortions or translations, leading to poor generalisation performance [31]. To combat this, a pooling operation can be used to reduce the spatial size of the data and thus lower the number of parameters and the amount of memory and computation required and also reduce the risk of overfitting [49]. This increased spatial invariance can be interpreted as trading the exact location of features for a rough relative position of it to other features which is potentially more useful for the network. This operation is also known as *down-sampling*, with the most common variant being *max pooling*. [49] Formally, this entails selecting the maximum value within every receptive field so that

$$Y_{k,i,j} = \max_{p,q} x_{kpq} \tag{2.7}$$

A common set of parameters for pooling is applying a "MAX" operation using a 2×2 kernel with a stride of 2, reducing the input to 25% of its original size [42]. Figure 2.6 shows how a moving window of, in this case, size 2×2 is being slid over the input, the maximum value within that neighbourhood is kept and then the window is moved by two pixels (according to the stride). In contrast to convolutional layers, pooling layers have no learnable parameters and are deterministic [64]. Each input channel into the pooling layer is processed separately and thus the number of output channels remains the same [64].

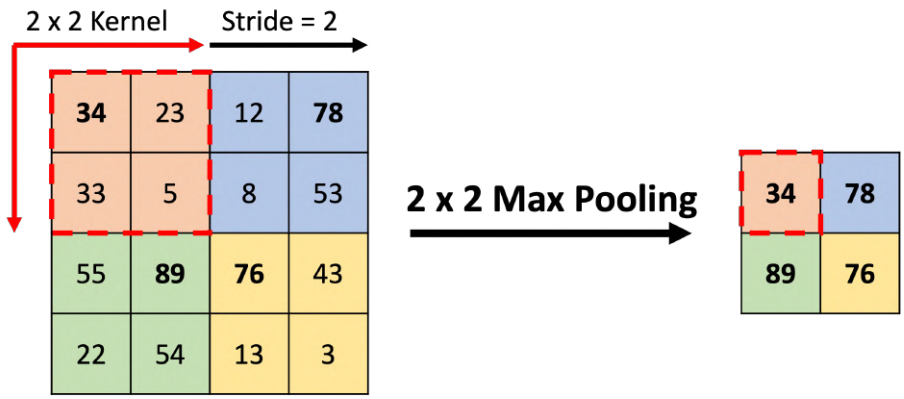


Figure 2.6: 2×2 Max Pooling operation with stride 2

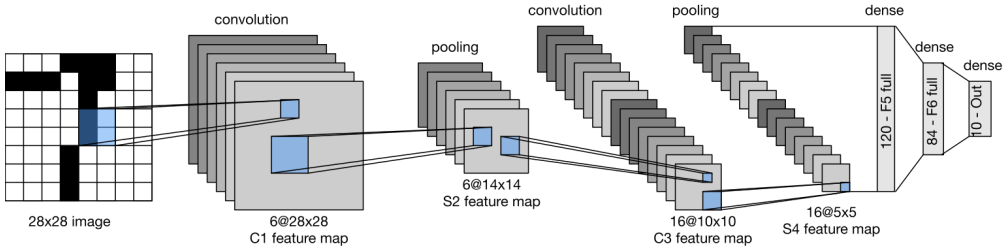


Figure 2.7: Dataflow in the LeNet convolutional network with two convolutional layers [64]

As pooling operations are inherently destructive, kernel sizes over 3 can lead to decreased model performance with the most popular choice being 2×2 max pooling, reducing the spatial resolution to a quarter of its original size [42], [64]. Average pooling is the second most common pooling operation and can help improve the signal-to-noise ratio, yet in most scenarios max pooling should be the default choice [64]. Other pooling variants include overlapping windows, subsampling or L1/L2-normalisation [42], [49].

2.1.3 Basic Architecture

To demonstrate how convolutions and pooling layers can be combined into a full object classification architecture this section will introduce LeNet by Yann LeCun et al.. Developed in 1989 at AT&T Bell Labs, it was the first convolutional network to be fully trained using backpropagation to classify handwritten digits in images. LeNet was able to achieve error rates of under 1% per digit thus matching existing state-of-the-art algorithms like support vector machines. [64] The architecture consists of two convolutional blocks, made up of a convolutional layer with a 5×5 kernel, a sigmoid activation function and a 2×2 average pooling layer. The first block is set to output 6 channels, while the second convolutional block generates 16 output channels. Before feeding the data into the fully connected layers for the final classification, inputs are flattened into a one-dimensional vector. The fully connected layers of size $[120, 84, 10]$ are then used to output the final classification probabilities for each digit. [64]

2.1.4 Advanced Techniques for Improving Convolutional Neural Networks

Since the inception of LeNet, countless improvements have been proposed and studied in literature, like replacing the sigmoid activation with a rectified linear unit (ReLU) or choosing max pooling over average pooling. Moreover, increasing the depth of a convolutional network has led to significant advances while also creating new challenges. [64]

The following sections will introduce a selection of the most significant modifications that have led to significant uplifts in performance and are now widely adapted throughout most modern convolutional architectures and beyond. Chapter 3 will ultimately introduce three convolutional architectures in detail that harness some or all of these techniques.

Other noteworthy concepts that have helped the advancement of convolutional networks include improved parameter initialization heuristics [17], modifications of the stochastic gradient descent algorithm (Adam optimizer, [29]), alternative activation functions (Introduction of ReLU, [40]) and the addition of more advanced regularization techniques (Dropout, [52]) [64].

VGG Blocks

A basic building block in LeNet or other early architectures like AlexNet would typically consist be a convolutional layer, a non-linear activation function followed by a pooling layer to down-sample the input. However, this setup hits a natural limit at $log_2 d$ blocks as the spatial resolution shrinks rapidly. For example, a resolution of 256×256 would allow for a maximum of 8 consecutive layers before its spatial resolution reaches 1×1. [64]

Simonyan and Zisserman proposed using a block of multiple convolutional layers in succession before applying a pooling operation. They studied whether increasing the width or the depth of the network would yield better performance and found that two stacked 3×3 convolutions would outperform one 5×5 convolutional layer using almost the same number of parameters ($3 \cdot 9 \cdot c^2$ vs. $25 \cdot c^2$). [51] This approach has been adopted broadly, also aided by the fact that GPUs became capable of specifically speeding up the computation of smaller convolutions. Dubbed the VGG block (used in the corresponding VGGNet, [51]), it is made up of a sequence of 3×3 convolutions with padding of 1 and a 2×2 max-pooling operation with a stride of 2.[64]

Network-in-Network Blocks

The network-in-network (NiN) block addresses the issue of having to use multiple fully connected layers at the end of the network that spawn a lot of parameters and consume a lot of memory. This used to be necessitated by the fact that adding linear layers earlier in the network would corrupt the spatial structure created by the convolutional layers. The NiN circumvents this by enabling non-linearity through 1×1 convolutions and the use of global average pooling.[33] This effectively creates a fully connected layer that is applied independently to each pixel. Finally, a global average pooling operation alleviates the need for a fully connected layer for classification as it averages every feature map into a scalar, creating a vector equal to the depth of the input. However, succeeding architectures often kept at least one final fully connected layer for classification. Still, this allowed for a drastic reduction in the number of parameters while also contributing to more spatial invariance. [64]

Batch Normalisation

The introduction of batch normalisation by Ioffe and Szegedy was motivated by the issue of internal covariate shift. This describes the phenomenon of weights in intermediate layers reaching values of vastly different magnitudes over time as the values in previous layers change. This distribution drift could slow down the convergence of the network and require lower learning rates. [28]

Batch normalisation works by taking the input, subtracting the mean and dividing by the standard deviation. The statistics are calculated across the current mini-batch. Finally, a scale coefficient is applied together with an offset in order to regain lost degrees of freedom [64]. Mini-batches of size one are therefore ineffective as the resulting value would be 0. To sum up, batch normalisation is defined as [28]

$$BN(\mathbf{x}) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma}_B} + \beta \tag{2.8}$$

where B is the mini-batch with input $x \in B$ as the input of the batch-normalisation function BN with sample mean $\hat{\mu}_B$ and sample standard deviation $\hat{\sigma}_B$. γ denotes the scale parameter and β the shift parameter, both equal in shape to x and learnable during training.[54]

$\hat{\mu}_B$ and $\hat{\sigma}_B$ are calculated using a small constant $\epsilon > 0$ to prevent division by zero:

$$\hat{\mu}_B = \frac{1}{\mid B \mid} \sum_{x \in B} x \tag{2.9}$$

and

$$\hat{\sigma}_B = \frac{1}{\mid B \mid} \sum_{x \in B} (x - \hat{\mu}_B)^2 + \epsilon \tag{2.10}$$

The noise introduced by the estimates of mean and variance has proven to aid generalisation performance and allow for more aggressive learning rates. [64]

For fully connected layers, batch-normalisation is often applied after the activation function contrary to the original paper. For convolutional layers, batch-normalisation is applied channel-wise and values by updating each spatial location with the operation also being performed either before or after the activation function. After training, the values for $\hat{\mu}_B$ and $\hat{\sigma}_B$ can be calculated based on the entire dataset and are subsequently fixed for inference. [64]

Layer Normalisation

Layer normalisation offers another similar approach to stabilise and speed up the training of deep networks. It can be applied to a single n-dimensional input x using both offset and the

scaling factor as scalars: [3]

$$x \to LN(x) = \frac{x - \hat{\mu}}{\hat{\sigma}} \tag{2.11}$$

with scaling factor and offset given by

$$\hat{\mu} =^{def} \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.12}$$

and

$$\hat{\sigma}^2 =^{def} \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2 + \epsilon \tag{2.13}$$

For every $\alpha \neq 0$ the output of layer normalisation is scale-independent, $LN(x) \approx LN(\alpha x)$. Applying this technique can help prevent divergence and compared to batch normalisation it is batch-size independent and behaves the same during training and inference. [64]

Residual Connections

When training very deep networks, an issue can arise during the backpropagation of gradients where very small or very large values can result in exponential growth or shrinking of gradients, commonly known as the exploding or vanishing gradient problem [23], [64]. Symptoms include slow convergence, unstable training, or getting trapped in local loss minima or NaN values [64, 23].

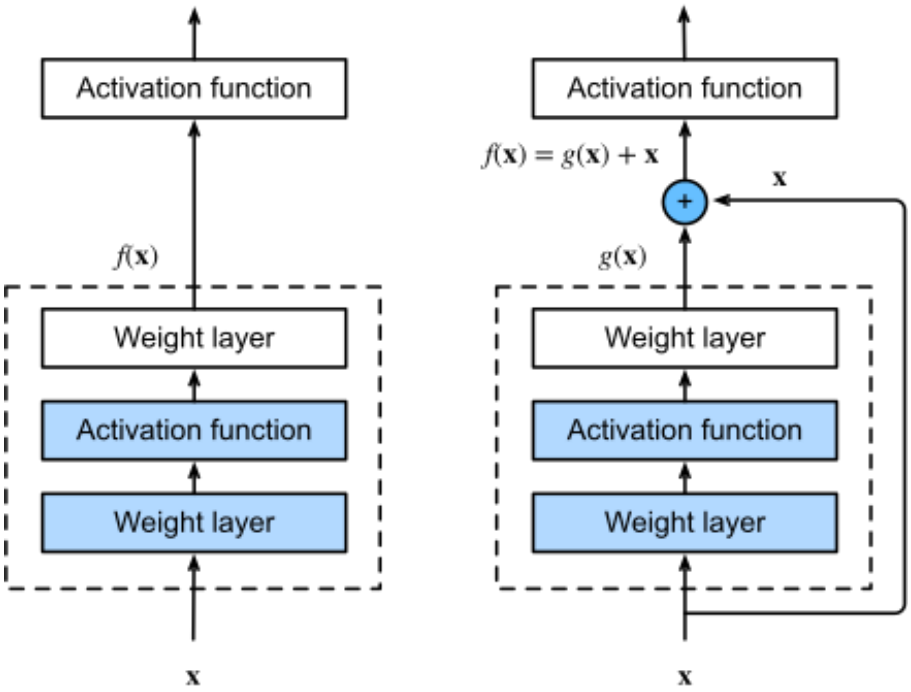


Figure 2.8: Regular convolution block (left) and a block using a residual connection [64]

As illustrated in figure 2.8, the residual connection (or shortcut or skip connection) allows the network to bypass a certain number of layers. This allows the network to learn a residual mapping $g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$, where $\mathbf{x}$ is the input, going as far as enabling $f(\mathbf{x}) = \mathbf{x}$ by mapping $g(\mathbf{x}) = 0$ and thereby skipping this set of layers. That way inputs can propagate faster through the network and gradients are less encouraged to converge to extreme values. [64]

This change has enabled the ResNet architecture to expand up to 152 layers deep, compared to VGGNet's 10 or LeNet's 22 layers [23], [50]. The use of residual connections quickly spread to other architectures like recurrent neural networks and has also been adopted in transformer-based architecture (see 2.2.5) [64].

2.1.5 Visualising Learned Representations

While deep learning models are often said to be "black boxes", convolutional networks actually allow for a number of ways to visualise what they learn. This section will briefly introduce two concepts for visualising learned representation.[7]

Visualising Convolutional Filters

Going back to the idea of convolutions, different kernels are supposed to learn different functions that might attempt to capture horizontal or vertical edges or recognise certain shapes. By applying gradient ascent in the input space, the response of a given filter can be maximised. This is achieved through a loss function that, starting from a blank input, uses gradient descent on a specific filter in a given convolutional layer to maximise its activation. Figure 2.9 shows an example of the first 64 filters of the first layer in the first and fourth convolutional block in a VGG network. [7]

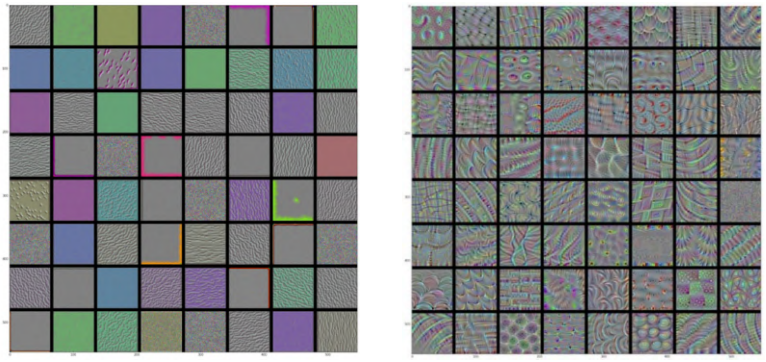


Figure 2.9: Filter activations of the first 64 filters in the first layer of the first (left) and fourth (right) convolution block in a VGG16 network [7]

As expected, it looks like earlier filters encode colours and directional edges while deeper layers start to encode textures and more complex patterns. [7]

Visualising class activations as heatmaps

To understand which parts of an image actually contribute to its classification, one can visualise the class activation as a heatmap, also known as a class activation map (CAM). For every location in the image, a score is computed that expresses how indicating this location is of a given class. The steps can be outlined as follows: for a given image, the output of a convolutional layer is weighted against the gradients of a given class and finally averaged over the channel dimension. This can be interpreted as weighing how much the image activates specific channels by the channel's importance in regard to the class. [7] The class activation map can help answer

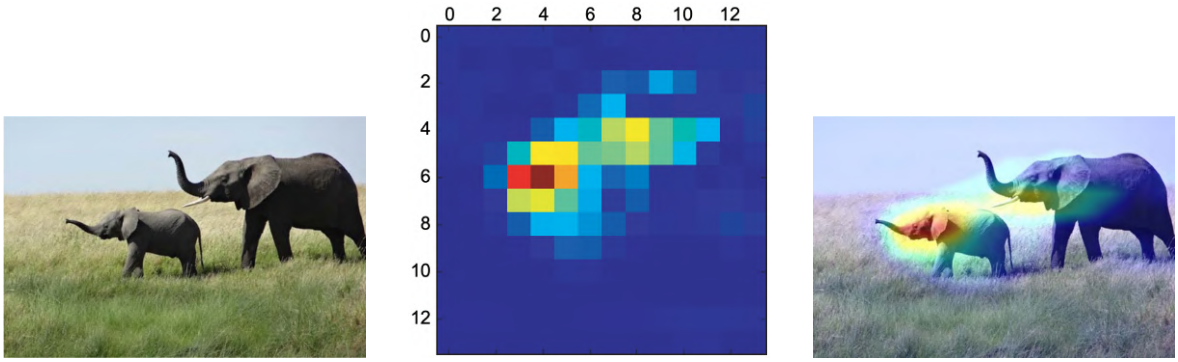


Figure 2.10: Class activation map (middle) for an input image (left) and overlayed as a heatmap (right) from a VGG16 network [7]

the question of which parts of the image led the model to conclude that it was of a certain class and where those features are located in the image. For example in figure 2.10, special attention is paid to the ear of the elephant, which is an important indicator that the image contains the class African elephant instead of an Indian elephant. [7]

2.2 Transformer Architecture

Initially introduced in the 2017 paper "Attention Is All You Need", the Transformer Architecture was developed for sequence-to-sequence learning in machine translation in the natural language processing (NLP) domain [61], [64]. The architecture has since emerged as the state-of-the-art model in a variety of NLP tasks. Big models pretrained on large amounts of data, called foundation models, are fine-tuned as they can serve as the backbone for a variety of downstream tasks. Naturally, this also piqued the interest of researchers to study whether the Transformer could be applied to other domains like computer vision, where CNNs have remained the dominant architecture [64]. Especially their trait to perform well at modelling long-range dependencies offered a unique advantage over CNNs and their inherent locality. [64]

Indispensable to the success of the Transformer architecture is a new type of layer called attention. It models the concept of attention in humans, enabling the model to learn to focus on specific parts of the input. In 2019, Ramachandran et al. show that convolutions can indeed

be replaced by a special type of attention, but the network remained hard to scale up on existing hardware. In 2020, Cordonnier et al. prove that attention is in fact able to replicate the behaviour of convolutional layers [8]. However, their use of 2×2 patches as inputs limit the model to low-resolution images [64].

While these approaches implemented parts of the Transformer architecture, they still use them in conjunction with convolutional networks or replace only some of their components while remaining true to their overall structure [13]. The paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by Dosovitskiy et al. changed this by proposing a pure Transformer architecture that processes a sequence of image patches. Dubbed "Vision Transformer" (ViT), this network was able to achieve state-of-the-art results on the ImageNet benchmark when trained on large datasets while claiming to be computationally less expensive to train than comparable convolutional networks. [13]

Before introducing the Vision Transformer architecture itself, the following chapters will first outline the fundamental building blocks of the generic Transformer architecture and the motivation behind some of the most influential design decisions.

2.2.1 Attention and Self-Attention

The concept of attention was first introduced by Bahdanau et al. in "Neural machine translation by jointly learning to align and translate," [4]. Its inception was driven by the restriction of recurrent neural networks (RNNs) to compute training inputs sequentially, making it hard to increase sequence lengths. Attention was meant to help capture long-distance relationships while also efficiently utilising modern hardware by being parallelizable [46].

Attention tries to model the intuition that given a sequence like text as the input, the model should be able to focus on specific parts of this sequence at different steps in the network. In practice, this would entail calculating a representation equal in length to the input that is derived from the weighted sum of said input, thereby modelling the ability to attend to specific parts of the input by influencing the weights accordingly. This feature was especially useful in machine translation, where attention was helpful to model cross-lingual synonyms when used as part of encoder-decoder network architecture. [64] Formally, denoting a set of key and value pairs as $D =^{def} \{(\mathbf{k}_1, \mathbf{v}_1), ...(\mathbf{k}_m, \mathbf{v}_m)\}$ with m tuples of keys and values and a search query $\mathbf{q}$, then attention over D, also known as attention pooling, is defined as

$$Attention(\mathbf{q}, D) =^{def} \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \tag{2.14}$$

with $\alpha(\mathbf{q}, \mathbf{k}_i) \in R(i = 1, ..., m)$ as the non-negative scalar attention weights. In this scenario, all the weights of $\alpha(\mathbf{q}, \mathbf{k}_i)$ are 0 except for one weight with value 1, taking inspiration from a typical database query. [64]

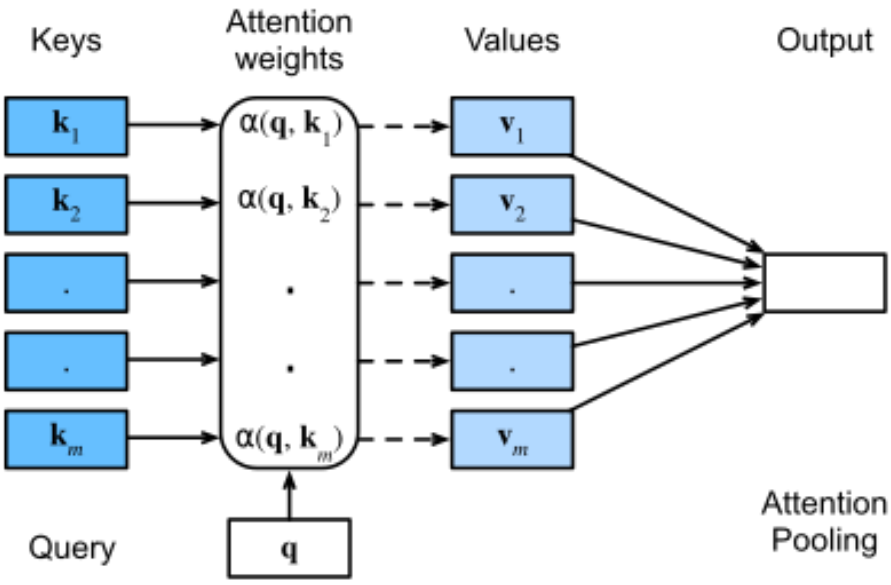


Figure 2.11: Attention Pooling [64]

To take this concept further, the weights can be normalised so they sum up to 1 and exponentiation and the softmax function can be used to enforce non-negative weights, allowing the weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ to model any function $a(\mathbf{q}, \mathbf{k})$. Importantly, this mechanism is differentiable and thus suited to be used and optimised using the backpropagation algorithm in neural networks. [64]

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_j \alpha(\mathbf{q}, \mathbf{k}_j))} \tag{2.15}$$

Scaled Dot Product Attention

Different scoring functions $\alpha(\mathbf{q}, \mathbf{k}_i)$ can be used to calculate the similarity between query and keys which ultimately determines what parts of the input should receive the most attention. The function employed by the original Transformer paper is called scaled dot product attention: [61]

$$a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^T \mathbf{k}_i / \sqrt{d} \tag{2.16}$$

The use of an additional scaling factor is motivated by the risk of unstable gradients during training, as large values of d_k may increase attention weights to the point where the softmax function reduces them to very small gradients [22, 61]. Additive attention is another attention function, utilizing a feed-forward network with one hidden layer for computation which makes it less space-efficient and slower to compute compared to dot product attention that can take advantage of fast matrix multiplication implementations [61].

Self-Attention

While attention by itself is already useful in an encoder-decoder architecture, where blocks in the decoder can learn how to attend to steps in the encoder to model language, it does not model dependencies within the sequence itself. Self-attention (or intra-attention), in contrast, assigns every input a query, key and value, allowing the mechanism to model a representation of the relationship between itself and all other inputs. [64]

For a sequence of inputs three vectors $\mathbf{q}, \mathbf{k}$ and $\mathbf{v}$ are constructed from every input with dimensions $d_q = d_k = d_v = d_{model}$ (with $d_{model} = 512$ in the original paper). These vectors are then combined into three corresponding matrices $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ to improve the computational efficiency. Next, the attention scores can be calculated using scaled dot product attention $S = \mathbf{Q} \cdot \mathbf{K}^T$ and then normalising the scores as shown previously by $S_n = S/\sqrt{d_k}$. After applying the softmax function $\mathbf{P} = softmax(\mathbf{S}_n)$, a weighted matrix is calculated $\mathbf{Z} = \mathbf{V} \cdot \mathbf{P}$. This method can be simplified into a single function with the computational complexity $\mathcal{O}(n^2 d)$: [22]

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{d_k}}) \cdot \mathbf{V} \tag{2.17}$$

2.2.2 Multi-Head Self-Attention

To enable a variety of different representations to be captured in the same attention mechanism, multi-head attention introduces the idea of first linearly projecting all queries, keys and values h times using learned parameter matrices $W_i^Q \in R^{d_{model} \times d_k}$, $W_i^K \in R^{d_{model} \times d_k}$, $W_i^V \in R^{d_{model} \times d_k}$. Next, these projections are processed by h attention pooling mechanisms, also called heads, in parallel whose output is then concatenated and linearly transformed once more with $W_i^O \in R^{hd_v \times d_{model}}$ to reach the desired output dimensionality. This process can be expressed as follows: [61]

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(head_1, ..., head_h)W^O \tag{2.18}$$

$$\text{where } head_i = Attention(\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V) \tag{2.19}$$

As indicated by the parallel placement of the attention heads in figure 2.12, this structure has the added benefit of enabling more computationally efficient execution. This is further improved by the reduced dimensionality of the matrices that are fed into each attention head, to the point where the computational cost is equal to a single attention operation using the full dimensionality. For the original Transformer paper, 8 heads were used with $d_k = d_v = d_{model}/h = 64$ and $d_{model} = 512$.[61]

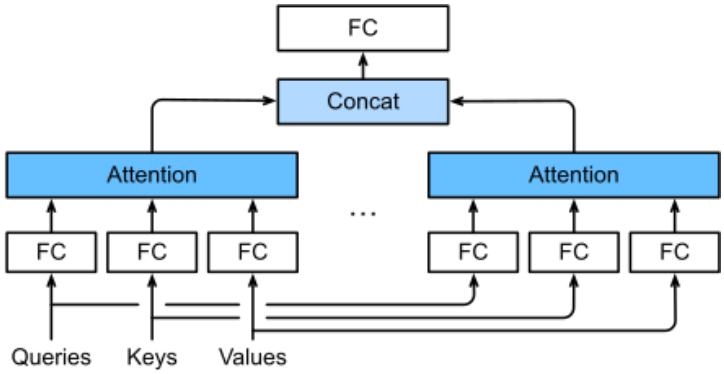


Figure 2.12: Multi-head attention [64]

2.2.3 Positional Encoding

One drawback of the self-attention mechanism is its complete disregard of any spatial information or sequence order. However, if this information is relevant, a positional encoding of the same dimension as the input can be injected by simple addition [64]. Encodings may either be learned or fixed in place. For example, Vaswani et al. use a fixed encoding based on the sine and cosine functions to inject absolute positional information into the input: [61]

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}} \tag{2.20}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}} \tag{2.21}$$

With *pos* being the input position and i being the dimension, a linear function can be learned by the model to derive PE_{pos+k} from PE_{pos}. Figure 2.13 visualises the application of this encoding scheme with rows representing the actual positional information of the sequence and columns referencing the dimensionality of the encoding [64]. While Vaswani et al. found performance to be similar to a learned encoding scheme they hypothesise that this fixed encoding may allow the model to extrapolate encodings beyond the sequence lengths it has seen during training. [61]

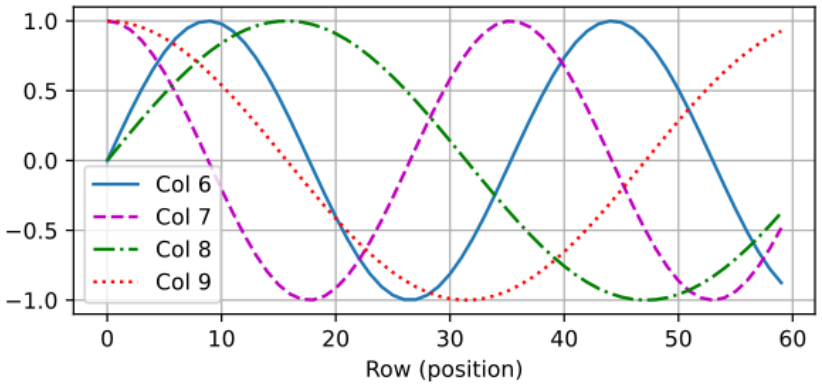


Figure 2.13: Generated sine and cosine waves as positional encodings [64]

2.2.4 Transformer Block

To form the complete attention block as proposed by Vaswani et al., two more elements are added to the multi-head self-attention (MHSA). First, a residual connection around the MHSA layer is added, feeding into a layer normalisation. Second, a position-wise feed-forward (FFN) network is added, defined as [61]

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.22}$$

This is equivalent to applying two convolutions with a 1×1 kernel and a ReLU activation. The name is derived from the fact that the same linear transformation is applied to all positions of the output of the multi-head self-attention layer. In the "Attention is all you need" paper, the hidden dimension was chosen to be of size $d_{ff} = 2048$ with $d_{model} = 512$ as the input and output dimension. [61]

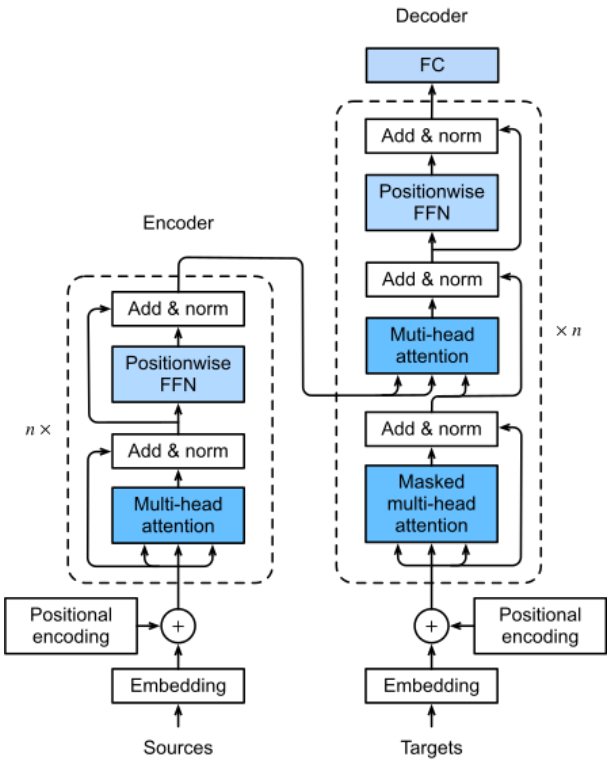


Figure 2.14: Transformer Architecture as proposed in "Attention is all you need" [64]

Lastly, another residual connection is added around the FFN, feeding into another layer normalisation. This Transformer block is then stacked n times, maintaining the dimensionality of its input and outputting a d-dimensional vector for each element in the input sequence. [61]

To complete this section, the original Transformer architecture also employs a slightly modified variant of this Transformer block that is used in the decoder of the network. Known as encoder-

decoder attention, it receives its queries from the previous decoder layer but its keys and values are extracted from the encoder output. Additionally, to cater for the task of machine translation, the inputs to the MHSA in the decoder may be masked to restrict the model to only attend to tokens that have already been generated. This is necessitated by the fact while for language modelling the target output tokens are known during training, they are not available during prediction where the model can and should only rely on tokens it has previously generated. [64]

2.2.5 Vision Transformer (ViT)

The development of the Vision Transformer explicitly set out to apply the "standard Transformer directly to images, with the fewest possible modifications" [13]. In the 2021 paper, "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale", the authors introduce the first pure Transformer architecture for image classification, achieving state-of-the-art results in the ImageNet benchmark. Since then, follow-up research has studied how and why they work so well, finding more and more areas in which this architecture exhibits superior performance over traditional convolutional networks. [13]

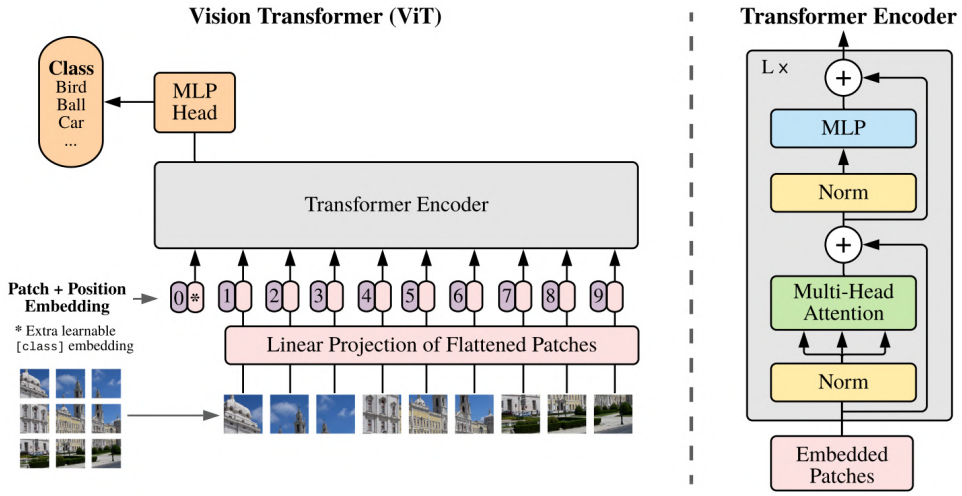


Figure 2.15: Vision Transformer Architecture [13]

The architecture of the Vision Transformer is depicted in figure 2.15. In contrast to the original architecture for translation, a separation into encoder and decoder is no longer required to perform simple classification. Notably, however, almost no modifications were made to the Transformer block itself. It still functions as described in the original paper, with the exception of placing the layer normalisation before the MHSA and bypassing it for the residual connection, but still being stacked n times. Instead, the only modifications needed have, in fact, remained minimal, and only concern the way images are fed into the model and how the final classification is achieved. [64]

Patch Embedding

The primary modification that had to be made is how data, specifically images with a spatial structure, are fed into the Transformer. Dosovitskiy et al. solve this by splitting the image into multiple patches that can be treated as a sequence that attention is able to model dependencies on. Given an input image of height h, width w with c channels and a patch size p for both height and width, $m = hw/p^2$ patches are created. These patches are then flattened into a vector of length cp^2, forming a sequence that the attention mechanism can process. Finally, the patches are projected into d dimensions using a learnable linear projection. [13]

Alternative approaches to this embedding strategy, specifically using features extracted by a CNN, will be discussed in chapter 3.

Class Token

To enable classification, a learnable "< cls >" (class) token is added to the patch embeddings to create $m + 1$ patches. For the final classification, only this class token is extracted and will serve as the final image representation used in the classification head. This head will typically be an MLP, consisting of one hidden and one output layer or just a single linear layer when performing finetuning. [13]

Positional Embedding

Dosovitskiy et al. experimented with a variety of positional embedding strategies, including adding no positional embedding at all, 1-dimensional embeddings, 2-dimensional embeddings that map patches onto a grid and embedding their X and Y coordinates and a relative position embedding that retains information about the relative spatial distance between patches. While adding no spatial context to the model significantly hampered performance, they found only small differences between the other embedding methods. One possible explanation for this is found in the patch-level inputs making the model already less sensitive compared to pixel-level inputs. Therefore, the proposed positional encoding strategies may all find it equally easy to represent this information. As a result, learnable 1-dimensional embeddings were used for the final model. [13]

Formally, the input to the Transformer block can be expressed as such

$$z_0 = [x_{class}; x_p^1\mathbf{E}; x_p^2\mathbf{E}; ...; \mathbf{x_p^N E}] + E_{pos}, \qquad \mathbf{E} \in R^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in R^{(N+1) \times D} \tag{2.23}$$

to produce the output

$$y = LN(\mathbf{z}_L^0) \tag{2.24}$$

, with $l = 1...L$ being the number of Transformer blocks and z_L being the output of the last block in the network. [13]

Network Variants

Dosovitskiy et al. present a total of three network variants of varying depth and complexity (see table 2.1). Additionally, models are categorised according to the patch size they use, with 16×16 and 32×32 being popular choices. Recall that the self-attention layers scale inversely proportional to the sequence length, making larger patch sizes less computationally expensive but usually also less accurate. To sum up, the notation commonly used to classify ViT models is a combination of their network size (base - B, large - L, huge - H) and their patch size (8, 16, 32), like ViT-B/16 or ViT-L/32. [13]

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 2.1: Network sizes for the Vision Transformer proposed by Dosovitskiy et. al [13]

Limitations

While the vanilla Vision Transformer did report state-of-the-art results in the ImageNet benchmark, these results were only achieved by prior pretraining on Google's internal JFT-300M dataset. With over 300 million images, it contains more than an order of magnitude more images than the ImageNet and ImageNet-21k datasets with 1.3 million and 14 million images respectively. Not only is pretraining at this scale out of the scope of most researchers and companies, but it also makes any comparison to existing networks more difficult. When only pre-trained on ImageNet, ViT is in fact not able to compete with similarly sized convolutional networks, only surpassing them when trained on over 100 million samples (see figure 2.16). [13] This observation can be explained by the lack of any inductive bias in the architecture. Where convolutional networks are inherently designed to have a focus on locality and to be translation invariant, only learning to model longer-range dependencies in later layers, Vision Transformers are not intrinsically built to do so. Instead, these patterns have to be learned first. However, this may also enable the Transformer to model long-range dependencies better than competing CNNs. [13]

This aligns with another characteristic documented by Dosovitskiy et al.: the Transformer's ability to scale to very large datasets while being more compute efficiently when compared to

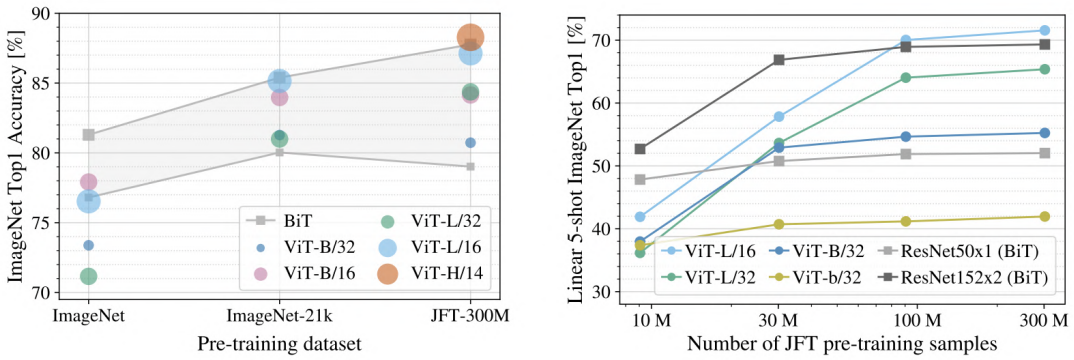


Figure 2.16: ImageNet accuracy based on different pre-training strategies for Vit and BiT [13]

other large-scale architectures like Big-Transfer (BiT) as demonstrated in figure 2.17. This scaling behaviour seems to indicate the point at which the network has learned to overcome its lack of inductive biases. Additionally, higher memory efficiency was observed compared to ResNets, allowing larger batch sizes which lend themselves better to large datasets.[13]

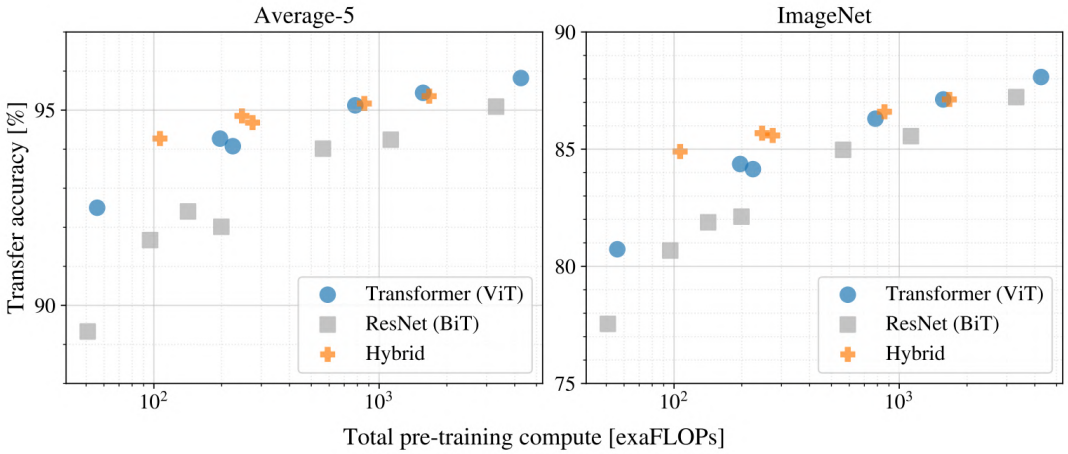


Figure 2.17: Required compute for pre-training for different architectures [13]

In practice, this behaviour presents a number of challenges: while the practice of using pre-trained models for fine-tuning on a smaller, task-specific dataset is prevalent in the industry and academia, large-scale datasets such as the one used here are often not publicly available. Additionally, the compute required to train them places them out of reach for most researchers and practitioners anyways. In general, vanilla Vision Transformers thus struggle to compete on small to medium-sized datasets. [64]

Finally, the self-attention operation results in quadratically scaling complexity that scales with the input image size n, with d being the dimension of the key and value vectors: [13]

$$\mathcal{O}(Transformerblock) = 12nd^2 + 2n^2d \tag{2.25}$$

High-resolution inputs, therefore, present another challenge to the network. The inverse applies

to the patch size p, which scales the number of sequences that will be created. Here, smaller patch sizes are more computationally expensive as every increase in the number of positions in the sequence is affected by quadratic scaling in complexity due to the multi-head self-attention operation needing to create a $p \times p$ matrix to calculate attention scores. Generally, however, the Transformer does not impose any restrictions on the maximum length of the sequence (apart from memory constraints). Larger inputs could consequently be adopted if the patch embedding was interpolated accordingly.[13]

In spite of these issues, Vision Transformers have been found to perform better than CNN's in a variety of other areas. To only name a few, ViTs seem to perform better on out-of-distribution generalisation compared to CNN's [44] while Ghiasi et al. found ViTs to more effectively utilise background information while relying less on high-frequency, textural information [15, 16].

Chapter 3 will introduce three recently proposed architectures that, among other objectives, aim to improve upon the vanilla Vision Transformer in these regards and further build upon its strengths.

2.2.6 Visualising Attention

Similar to the visualisation of kernels and class activations in a convolutional network, the output of Vision Transformers can be viewed by displaying the feature maps generated by the self-attention layers. These feature maps can show how different patches of an input image are related to each other and what features the network learns to attend to. The attention weights that are computed by the scaled dot product attention mechanism can also be visualised, which indicates how much each patch contributes to the output of each layer. These visualisations can help understand how vision transformers process image data and what they learn from it. [13]

One can visualise components throughout the network, starting at the linear projection layer that is used to flatten the input patches. Figure 2.19 (left) shows how principal component analysis (PCA) can be used to visualise these filters. They bear a striking resemblance to the low-level representations learned in early convolutional layers as shown in 2.9. [13] Moving up the network, the central figure in 2.19 plots the activations in the positional encoding layer, reaffirming that spatially related patches are learning similar embeddings. Even a grid-like row-column structure can be observed. This also supports the hypothesis that 2-dimensional positional or relative positional embeddings are not needed as they can be learned even from 1-dimensional representations. [13]

One of the benefits of the Transformer architecture is its ability to learn long-distance dependencies even in early layers, where convolutional networks need to build up to deeper layers to increase the receptive field of a given kernel. However, ViTs still generally appear to follow a similar strategy, as indicated by the plot of averaged attention distances at varying depths

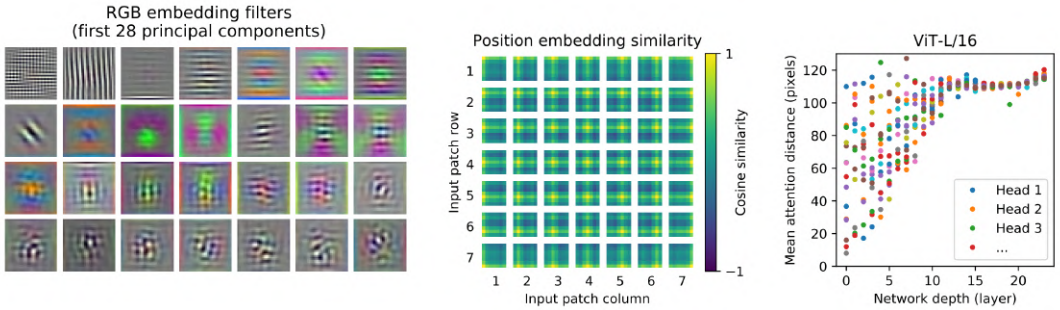


Figure 2.18: **Left**: Visualised filters of the linear embedding layer in a ViT-L/32 **Center**: Visualisation of Positional Embeddings as the cosine similarity **Right**: Average attention distance at increasing network depths [13]

of the network (figure 2.19 right). Attention distance is calculated as the geometric distance between a query and every other token multiplied by the attention weights. Its interpretation bears similarity to the concept of a receptive field in CNNs, expressing how far a given attention module attends back in space. Dosovitskiy et al. find that some heads indeed do attend to lower layers while others remain fairly local. They also observe that this effect is less pronounced for hybrid variants using convolutional features as inputs (see 3.3.1). [13]

Lastly, visualisation may also be performed by mapping "the output token to the input space", creating an attention map [13]. Similar again to class activation maps in convolutional networks, this can be used to visualise which regions in a layer are semantically relevant for the classification. [13]

Figure 2.19: Attention map for a ViT-B/16 [39]

2.3 Transfer Learning

In supervised machine learning, models rely on previously collected data that has been labelled and that follows the underlying assumption that it has been drawn from the same feature space and distribution. However, in practice this step is often one of the hardest parts of the development of a machine learning classifier, as collecting and labelling enough data is time-consuming and costly. [43]

One method of overcoming a lack of data is transfer learning. Defined as "the ability of a system to recognise and apply knowledge and skills learned in previous tasks to novel tasks" by the US Defense Advanced Research Projects Agency (DARPA), it allows machine learning models to apply knowledge learned on out-of-distribution tasks and apply it on a new target task[43]. The research in the field of transfer learning is extensive and remains very active. As such, this section aims to only introduce the basic concept and motivation behind transfer learning needed to understand how pre-trained models are leveraged in the practical research of this thesis.

Depending on how the relationship of the source knowledge to the target task and domain, one can separate two types of transfer learning: transductive transfer learning expects the source and target tasks to be the same while the data comes from a different domain. Inductive transfer learning implies the opposite, with data gathered from the same domain but applied to a different target task. Given a model trained on a large, diverse dataset of common objects, a second model could be trained with transfer learning, using the same set of weights of the first model and thus inheriting useful learned features that detect basic shapes and objects, even if the target class is more specific. Gradient changes needed to fit the new target are reduced due to already helpfully initialised weights. This process is known as finetuning. Other variations, such as multi-task learning and self-supervised learning also exist, but will not be covered in detail. In conclusion, transfer learning can help speed up the training process, inject helpful biases and improve accuracy above what would otherwise be possible. [56]

As a result, transfer learning has become indispensable for modern deep learning. As models and dataset sizes grow, finetuning models usually results in better and faster results than training from scratch. Additional techniques include freezing certain layers to reduce the number of parameters that will be updated during each backward pass to speed up training and replacing the final classification layer to change the number of outputs for classification. For image classification problems it is common to choose a model pre-trained on the ImageNet dataset. Other tasks that use pre-trained models for feature extraction, sometimes referred to as a backbone model, include image segmentation or object detection. [64]

2.4 Evaluation Metrics

Evaluating a multi-class classification problem is not a trivial task. It is hard to summarise the performance of a classifier into a single number that captures all aspects of its quality and trade-offs as different evaluation metrics may emphasise different aspects of the classifier's behaviour. This section will briefly introduce some of the most common evaluation metrics for multi-class classification without going into too much depth about each metric's derivation. Later chapters will then discuss how these metrics can be used to interpret the classification performance on the model recognition dataset.[19]

Accuracy

Accuracy expresses how often the model's predictions are correct across the entire dataset.

$$Accuracy = \frac{\sum_{i=1}^{n} I(y_i = \hat{y}_i)}{n} \tag{2.26}$$

Both interpretation and calculation are the same as for binary classification. While accuracy is one of the most popular metrics, for example being the primary metric reported for the ImageNet benchmark, it can be misleading for unbalanced datasets. As classes with a high number of samples will have a higher weight in the calculation, a high accuracy value may hide poor detection performance for underrepresented classes. [19]

Precision and Recall

Precision is defined as the number of correctly classified positives (true positives) divided by the number of all positive predictions (sum of true positives and false positives). In other words, precision answers the question of how many predicted positives are real positives. [19]

$$Precision = \frac{TP}{TP + FP} \tag{2.27}$$

Recall instead divides the number of true positives by the number of all positively predicted elements (sum of true positives and false negatives). Thus, recall answers the question of how many real positives are correctly predicted as such. [19]

$$Recall = \frac{TP}{TP + FN} \tag{2.28}$$

However, both metrics are defined in the scope of binary classification. There are two common ways to adapt them to multi-class classification: Macro-average computes precision and recall

for each class separately and then calculates the arithmetic mean over all classes. Conversely, the micro-average of precision and recall calculate the values for TP, FP and FN across all elements, essentially applying a weighted average instead.[19]

$$Recall_{macro} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FN_c} \tag{2.29}$$

$$Precision_{macro} = \frac{1}{C} \sum_{c=1}^{C} \frac{TP_c}{TP_c + FP_c} \tag{2.30}$$

As micro-average precision is equivalent to accuracy for multi-class classification, it is not commonly used. Macro-average values, however, are able to express the performance across all classes, regardless of the number of samples per class. This can be useful to judge how well the classifier generalises on underrepresented classes. Note that when calculating accuracy for specific classes, the calculation is equivalent to the precision for that class. [19]

F1-Score

As precision and recall can take very large or very small values independently from each other, the F1-Score is a way of expressing a trade-off between them in a single value. It is calculated as the harmonic mean of precision and recall and aims to punish models with diverging values for precision and recall while rewarding models with similar scores. Generally, more weight is given to lower values. The adaptation to multi-class classification follows the separation into macro and micro averages as described for precision and recall, with the formula consequently unaffected. However, this adaptation means that the micro f1-score is again identical to the accuracy, and therefore not used.[19]

$$F1 - Score = (\frac{2}{precision^{-1} + recall^{-1}}) = 2 \cdot (\frac{precision \cdot recall}{precision + recall}) \tag{2.31}$$

Mattheus Correlation Coefficient

Envisioned by Brian Mattheus in 1975, the mattheus correlation coefficient (MCC) is another balanced measure in the range of $[-1, 1]$. Values close to 1 indicate a good classification, whereas values close to 0 show no correlation between variables and negative values showing an inverse correlation. Negative values are usually caused by errors in the implementation. MCC is based on the Phi-Coefficient and is defined for binary classification as

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FN)(TN + FP)}} \tag{2.32}$$

For multi-class classification, the coefficient is modified as follows

$$MCC = \frac{c \times s - \sum_k^K p_k \times t_k}{\sqrt{(s^2 - \sum_k^K p - k^2)(s^2 - \sum_k^K t_k^2)}} \tag{2.33}$$

with

- $c = \sum_k^K C_{kk}$ as the total number of units predicted correctly
- $s = \sum_i^K \sum_j^K C_{ij}$ as the total number of units
- $p_k = \sum_i^K C_{ki}$ as the number of predictions of class k
- $t_k = \sum_i^K C_{ik}$ as the number of actual occurrences of k

MCC = 0	PREDICTED		
Classes	Positive (1)	Negative (0)	Total
ACTUAL Positive (1)	TP = 40	FN = 0	40
Negative (0)	FP = 10	TN = 0	10
Total	50	0	**50**

Figure 2.20: Mattheus Correlation Coefficient for a completely unbalanced binary classification [19]

The MCC can be a good indicator on unbalanced datasets, as showcased in the example in figure 2.20. However, results may be unstable, especially between 0 and -1, with wide fluctuations during training. [19]

3 Network Variants and Optimisation Techniques

This chapter will introduce the selection of network architectures and optimisation techniques that will the evaluated as part of this thesis. These include CNNs, Transformers and hybrid architectures. To evaluate a wider variety of models and also investigate the effects of different models on this thesis-specific problem, the selection does not strictly follow the current leaderboard of the ImageNet benchmark but instead is based on the most prominent architectures of each type.

Furthermore, practical aspects, such as the availability of public implementations of new architectures or optimisations, also had to be considered, as implementing and testing them from scratch would not be feasible. Additionally, due to limited computational resources training models from scratch is not an option, ruling out models for which no pre-trained weights were publicly available.

The order of presentation for each section follows the time of the release of each architecture, as many papers reuse or improve upon previously introduced concepts. Any reported benchmark numbers, therefore, are relative and correct only at the time of publishing of each paper. Also, note that the vanilla Vision Transformer is not covered here as it was already introduced in 2 but it will, of course, be included in the final evaluation.

3.1 Convolutional Networks

3.1.1 Inception

The Inception architecture was first introduced as GoogLeNet by Szegedy et al. in 2014, achieving state-of-the-art results in the ImageNet Large-Scale Visual Recognition Challenge [54]. Their goal was to tackle some of the most prominent issues of deep convolutional networks while also not increasing the computational budget:

- CNNs easily overfit
- inference and training get very computationally expensive

- capturing information on different scales (global, local) requires different kernel sizes [37]

The main building block of the Inception architecture is the naive Inception module. By calculating convolutions with different kernel sizes in parallel instead of sequentially, they allow for a wider instead of a deeper network [37]. Figure 3.1 shows a naive Inception module with 1×1, 3×3, 5×5 convolutions and one 3×3 max pooling operation. This allows the network to look at both local information (1x1 convolution block) as well as more global features with the 5×5 block [37].

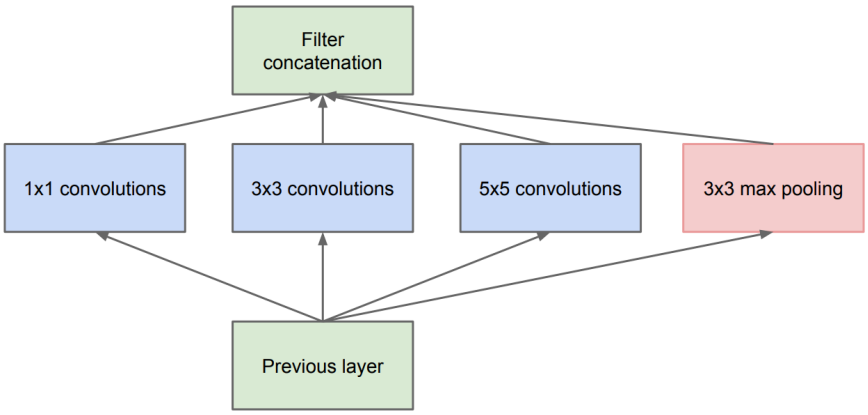


Figure 3.1: Naive Inception module [54]

When comparing the number of learnable parameters of one such inception block to three conventionally stacked convolutional layers with 1×1, 3×3 and 5×5 convolutions (all using 32 filters each, only one colour channel), the Inception block requires almost 30 times fewer parameters (1.216 vs. 34.944 parameters) [37].

To take this one step further, Szegedy et al. also proposed to use a 1x1 convolutional layer to reduce the input dimensionality going into the 3×3, 5×5 and max pooling layers [54]. To better demonstrate the effects of this, consider the following two layers: (example by [37])

- Input tensors shape: $1 \times 28 \times 28$
- Convolutional Layer 1 with 128 kernels, each 5x5: output shape $128 \times 24 \times 24$
- Convolutional Layer 2 with 16 kernels, each 1x1: output shape $16 \times 24 \times 24$

The 1x1 convolutional layer can reduce the number of channels of its input while maintaining the x and y dimensions [37]. This allows for an even greater reduction in learnable parameters, as shown in table 3.1.

Another technique adopted by Szegedy et al, which would later become a popular choice for other networks, is grouped convolutions. Originally motivated by the lack of GPU memory, grouped convolution divides the number of channels into multiple groups. At each layer, each

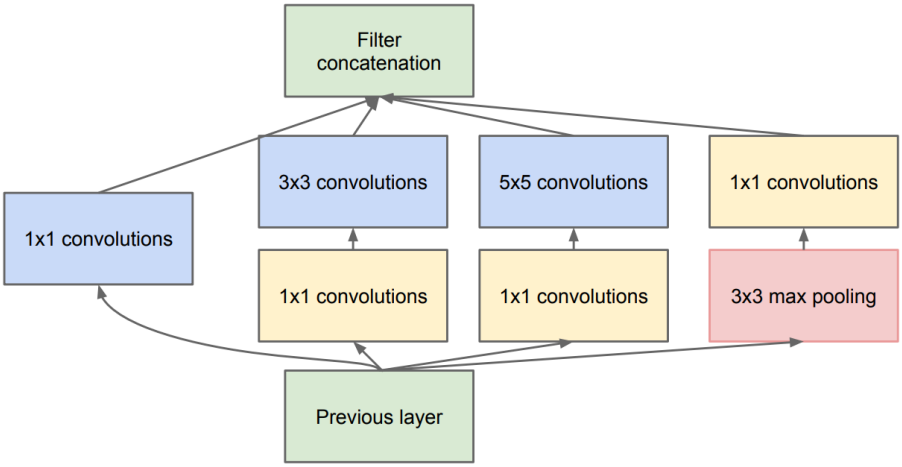


Figure 3.2: Inception module with dimension reduction [54]

Input shape: 256, 28, 28

	naive Inception block	with dimension reduction
1x1 convolutions with 8 kernels	2056	2056
3x3 convolutions with 8 kernels	18.440	2640
5x5 convolutions with 8 kernels	51.208	3664
3x3 max pooling with 1x1 convolution		2056
Total parameters	**71.704**	**10.416**

Table 3.1: Number of learnable parameters for a 256, 28, 28 input for a naive Inception block and a block with dimension reduction [37]

convolutional filter would then only process this group of channels. This was originally used to split the network into two GPUs but follow-up research has also shown it to also generally improve accuracy. [54, 64]

The full architecture of GoogleLeNet is depicted in figure 3.3. An interesting design choice involves the inclusion of multiple loss functions, as the authors discover that gradients in the middle layers fall to zero. Therefore, each part is trained as its own separate classifier, with the final loss function being computed as the weighted sum of the auxiliary losses. [37] A number of improved variants have been published over the years, adopting the name of their main building block to form the Inception family of networks. These improvements include batch normalisation, label smoothing and the introduction of the RMSProp optimiser. Later versions also include the Inception-ResNet model v1 and v2, which, despite their names, were actually released alongside Inception v4, with the v1 version matching Inception v3 in compute while v2 matches Inception v4. They introduce residual connections to the network, further helping with the vanishing gradients problem. Inception-ResNet v2 will serve as the baseline model for further comparison, as it is the one currently deployed for the model recognition service.[55]

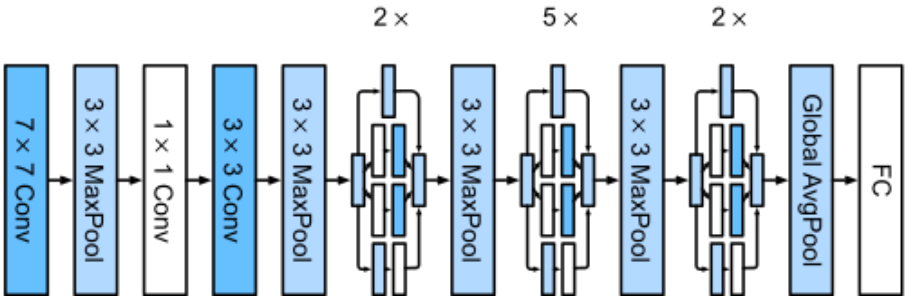


Figure 3.3: GoogleLeNet architecture [64]

3.1.2 EfficientNet

Tan et al. (2019) propose a new method for scaling convolutional neural networks based on balancing network width, depth, and resolution. The authors aim to improve efficiency and accuracy by systematically studying how to scale these three dimensions of network architecture. They hypothesise that there is a compound relationship between them that can be used to scale up a baseline network for different resource constraints. [57]

The paper examines the three dimensions that convolutional networks can be scaled by: one can increase the depth by increasing the number of layers in the model, scale the network width by increasing the number of channels in each layer or provide higher resolution inputs. Tan et al. prove that these dimensions are in fact linked and need to be balanced accordingly to achieve optimal performance. For example, when increasing the input resolution, the depth of the network needs to increase in order to expand the receptive field of the later layers while more channels are needed to capture more fine-grained information. [57]

For any desired increase in available compute N, three scaling factors α^N, β^N, γ^N can be derived for scaling depth, width and image size respectively. By using a multi-objective neural architecture search, optimising for accuracy and FLOPS (floating point operations), they develop a new mobile-size baseline network which is then scaled up to create the EfficientNet family of networks. The baseline network EfficientNet-B0 targets 400 million FLOPS and mainly implements the mobile inverted bottleneck layer (MBConv layer) introduced in the MobileNet v2 architecture [57, 48].

The inverted bottleneck layer first expands the input channels through a 1×1 convolution for a given expansion ratio. Next, a 3×3 depth-wise convolution is applied separately to each channel, reducing the number of parameters and computation compared to regular convolutions. Finally, the data is projected back to a lower dimension by another 1×1 convolution. A residual connection is added between the lower-dimensional input and the output. Tan et al. also apply a squeeze-and-excitation optimisation to these blocks as proposed by Hu et al. [57, 26]. This operation first applies global average pooling to each channel and then calculates a set of scaling factors by applying two dense layers with sigmoid activation to each channel. When multiplied

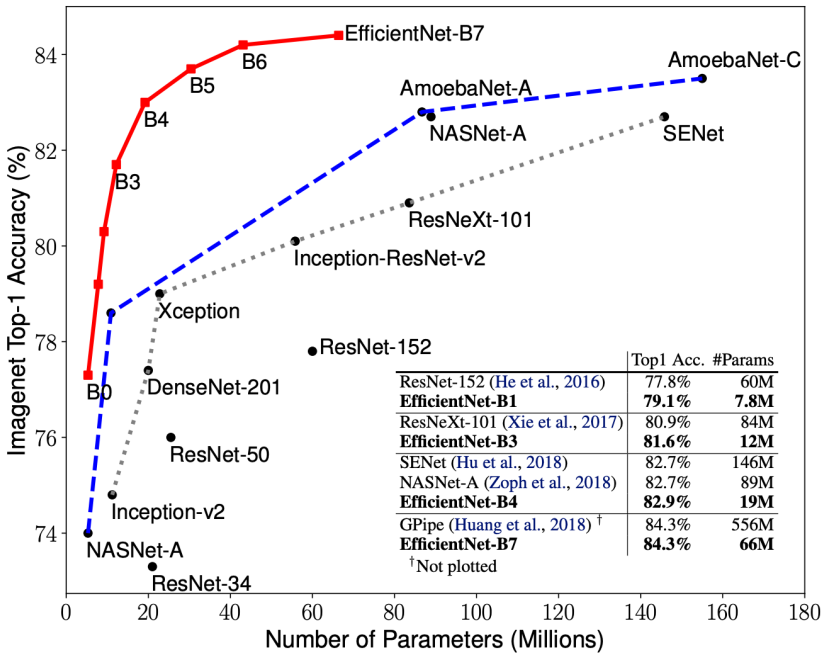
with the input feature map, this operation is used to *excite* or suppress different channels in relation to their importance. [57]

The full architecture of EfficientNet-B0 is denoted in table 3.2 for each stage i containing $\hat{L}_i$ layers, input resolution $[\hat{H}_i, \hat{W}_i]$ and channels $\hat{C}_i$ for a total of 5,3 million parameters. After applying their compound scaling method the scaling factors $\alpha = 1.2, \beta = 1.1$ and $\gamma = 1.15$ were derived and used to scale the network up to EfficientNet-B7, with scaling on ImageNet shown in figure 3.3. [57] Compared to competing architectures at the time of publishing, the

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 3.2: Architecture of the EfficientNet-B0 model [57]

authors find that their compound scaling method improves the accuracy of existing models when scaling using this method while maintaining model efficiency. They also show that their EfficientNets outperform other state-of-the-art CNNs at the time on ImageNet with up to 7,6x fewer parameters than ResNet-152 and 5,7x faster inference speed.[57]



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Table 3.3: EfficientNet Performance for variants B0 to B7 on ImageNet [57]

3.1.3 ConvNeXt

The 2022 ConvNeXt paper by Liu et al. takes notice of the uprising of Vision Transformers in the computer vision space and tries to incorporate typical Transformer characteristics like scalability and expressiveness into a convolutional architecture. [36]

ConvNeXt incorporates a large variety of changes from previous architectures as well as from the improved training schemes used by Transformer architectures. Namely, using extensive data augmentation like Mixup, Cutmix and RandAugment, which will be introduced later. Figure 3.4 lists the adaptations made in relation to a standard ResNet-50 model and the resulting accuracy improvements. This section will highlight a few of these modifications that led to the largest improvements or that took direct inspiration from Transformers. [36] Starting off with

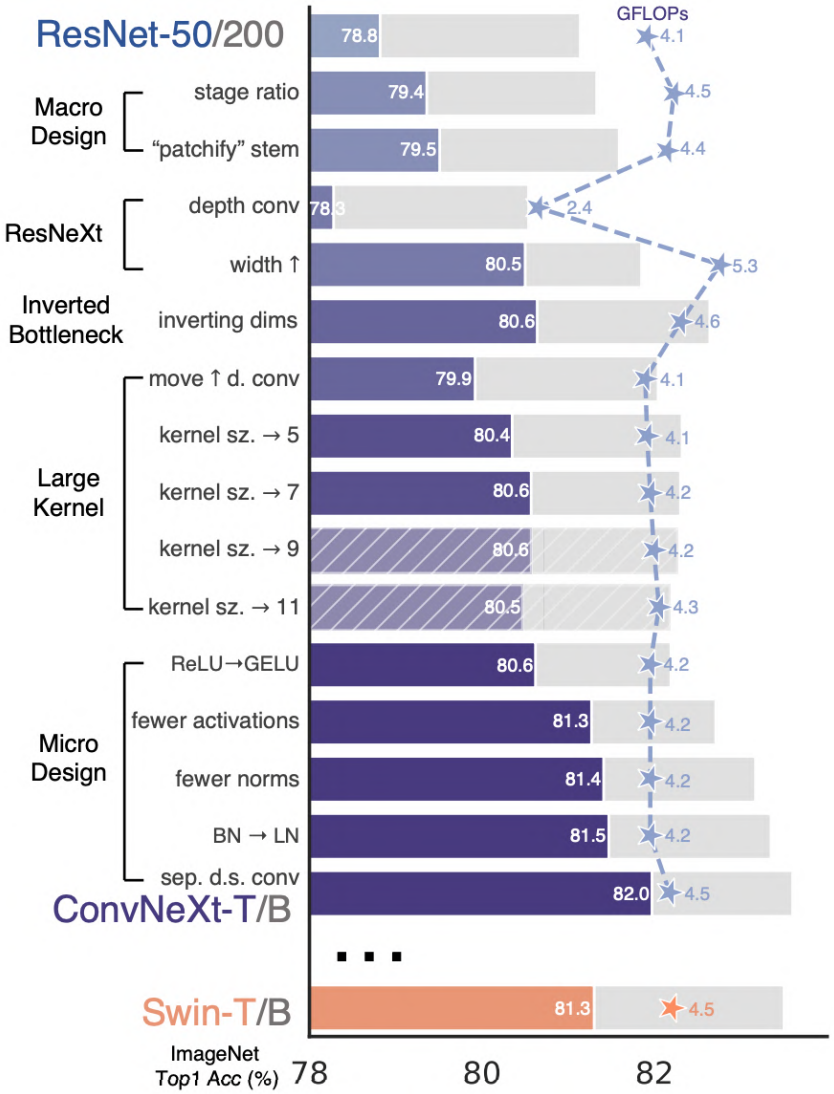


Figure 3.4: Adaptations to a ResNet-50 model and the resulting performance gains to arrive at the ConvNeXt architecture compared to a Swin Vision Transformer[36]

the model's stem, it is often designed to aggressively downsample the input image using large

convolutional kernels like 7×7 and Max Pooling in ResNet, to reduce the spatial dimensionality by 4x. Patch encoding in Transformers follows a similar concept, using even larger "kernels" with patch sizes of 16 or larger. For ConvNeXt, the authors took inspiration from the shifted window transformer, (Swin) which employs a hierarchical approach as it uses a multi-stage design, similar to some convolutional networks. For that reason, it uses a patch size of 4, which ConvNeXt adapts for its input, being equivalent to a 4×4 convolution with stride 4. [36]

Another modification is inspired by the ResNeXt architecture, which relies heavily on grouped convolutions which were introduced in AlexNet. Specifically, the special case of depth-wise convolution is used where the number of groups is matched to the number of channels. This technique was also used in the EfficientNet architecture. The authors note the similarity to taking the weighted sum with attention weights in a Transformer. Furthermore, combining depth-wise convolution with 1×1 convolutions creates a separation of spatial and channel information, another characteristic shared with the Transformer, that never mixes information across spatial and channel dimensions at the same time. [36]

Another similarity to EfficientNet includes the use of inverted bottleneck blocks. Additionally, the authors experiment with using larger kernel sizes and find a saturation point at 7×7 convolutions, again being inspired by the non-local attention in Transformers which they try to simulate by increasing the receptive field through larger convolutions. On a micro level, more implementation details were copied over from Transformers, like replacing batch normalisation layers with layer normalisation and reducing the overall number of normalisation layers. [36]

In conclusion, Liu et al. highlight how almost all of the design choices have been inspired and adapted from Vision Transformers and while some have existed in convolutional networks for a while, they have never been combined into one architecture. The resulting architecture is able to compete and slightly surpass the Swin Vision Transformer on both the ImageNet-1k and 22k benchmarks. [36]

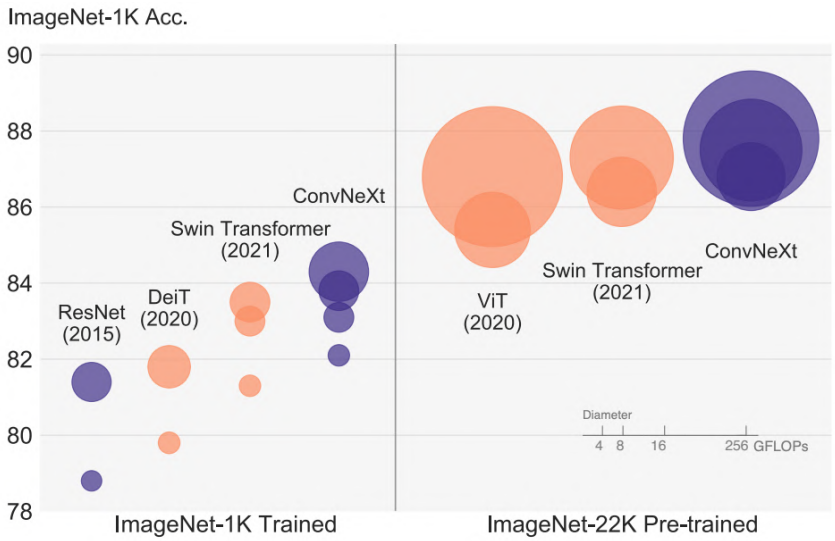


Figure 3.5: ConvNeXt Performance with bubble size being proportional to FLOPs on ImageNet-1k and ImageNet-22k compared to Vision Transformer architectures and ResNet[36]

3.2 Transformer Networks

3.2.1 Data-Efficient Image Transformer (DeiT)

The data-efficient image transformer (DeiT) by Touvron et al. aims to address the issue of subpar generalization performance of the vanilla Vision Transformer when trained on "insufficient amounts of data" by using extensive data augmentation techniques, previously introduced by Steiner et al. [13, 53, 60]. Additionally, they introduce a distillation strategy that uses a teacher network to boost performance further. [60]

Knowledge distillation, initially proposed by Hinton et al in 2015, describes the process of a strong teacher model injecting "soft" labels, specifically its softmax output, into a student model [24]. This can improve the student model's performance by acting like label smoothing and in the context of Vision Transformers, this can be used to transfer inductive biases from a convolutional teacher network. As an alternative to soft labels, hard-label distillation considers only the top predicted label, similar to training on ground-truth labels. Touvron et al. find this to perform better while being conceptually simpler. By using label smoothing, a conversion into soft labels is also still possible. [60] The key innovation of DeiT is to use a distillation token in the student model. A distillation token is a special input token that is added to the image patches before feeding them to the transformer blocks (see figure 3.7). The distillation token acts as a query for the student model to attend to the teacher's outputs. This way, the student model can learn from the teacher's attention patterns and representations. The token is output after the last layer in addition to the class token and is incorporated into the combined loss function. [60] The distillation token also helps to regularise the student model and prevent

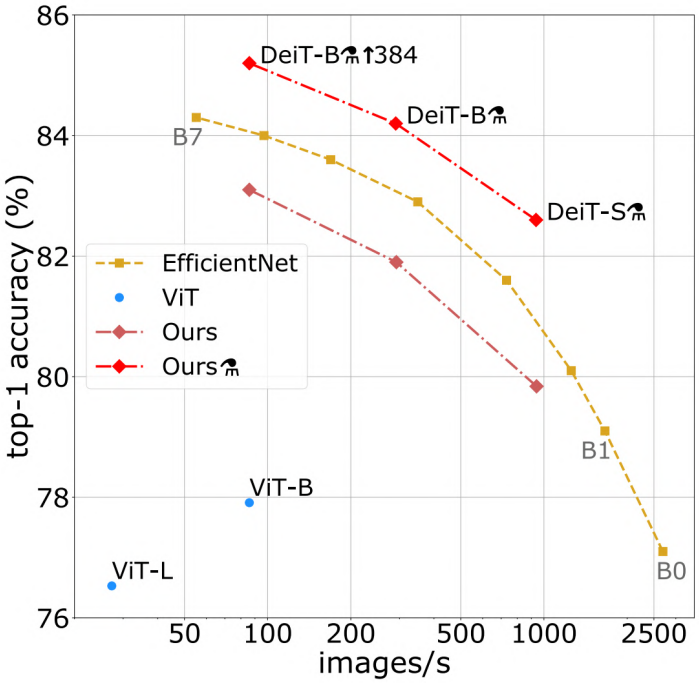


Figure 3.6: Inference and throughput performance of DeiT variants on the ImageNet-1k benchmark [60]

overfitting. By attending to the teacher's outputs, the student model is encouraged to focus on the most relevant parts of the image and ignore the noisy or irrelevant ones. Moreover, the distillation token can be randomly masked during training, which forces the student model to rely on its own inputs and not depend too much on the teacher. At inference time, both the class token and the distillation embedding can be used for prediction, with the authors recommending fusing them together, Other than distillation, no other changes are made to the architecture over the vanilla ViT. [60]

As DeiT relies on strong, pre-trained models for training, this raises the question of much it can truly learn from the teacher model and how much that helps it to generalise. Touvron et al. show that a distilled model can actually outperform its teacher, with the teacher being a RegNetY-16GF with 84 million parameters. Interestingly, the distillation process also works better with CNN teachers compared to Transformers. This would support the theory that they are able to distil their inductive biases into the Transformer.[60]

3.2.2 Shifted Window Transformer (Swin)

When adaption Transformers from NLP to vision, two challenges arise: images require the understanding of scale, as visual elements can vary drastically in size whereas word tokens in NLP remain at a fixed scale. Second, the high resolution of images is problematic due to the issue of quadratic self-attention scaling, where many downstream tasks like image segmentation rely on pixel-level predictions. Liu et al. propose the shifted windows transformer (Swin) that

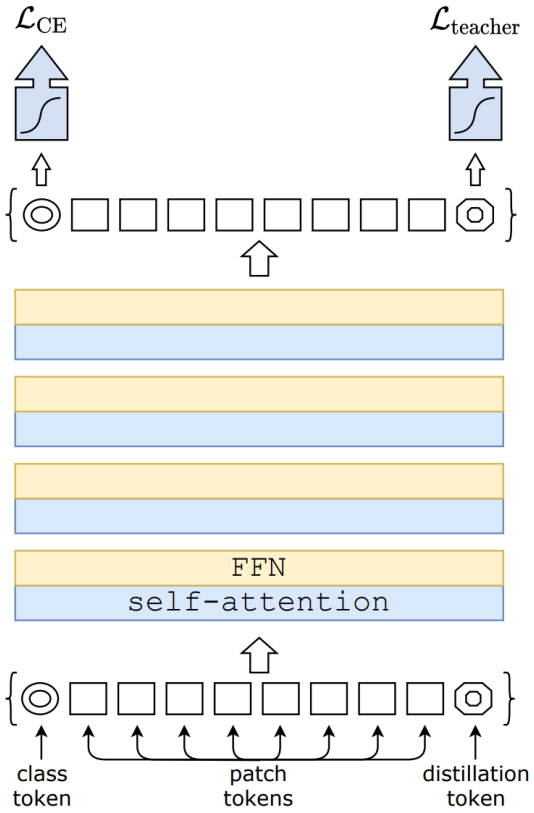


Figure 3.7: Integration of the distillation token into the Vision Transformer architecture for DeiT [60]

creates hierarchical feature representations and uses shifted windows to restrict the complexity of the self-attention operation, solving the quadratic scaling problem. [35]

Starting at the patch creation layer, Swin chooses a much smaller patch size of 4. In later stages, multiple patches will then be concatenated to form larger patches, thus reducing sequence lengths deeper into the network. Figure 3.9 shows the architecture for Swin-T, with each stage progressively reducing the effective spatial dimension by combining multiple smaller patches. This effectively allows the network to operate on different patch sizes and resolves the need for a single static patch size that needs to balance accuracy and throughput. [35] The next

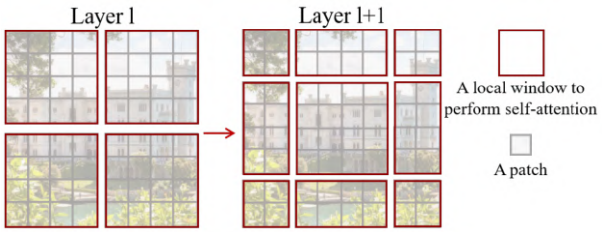


Figure 3.8: Visualization of the shifted window approach for Swin, where the initial patches are smaller than in ViT, but self-attention is restricted to local, non-overlapping windows [35]

modification involves the way self-attention is computed. While traditionally, the transformer blocks calculate global attention across all sequences, Swin instead performs non-overlapping windowed multi-head self-attention (W-MSA). This involves selecting a window with $M \times M$ patches on an image $h \times w$, reducing the computational scaling from

$$\mathcal{O}(MSA) = 4hwC^2 + 2(hw)^2C \tag{3.1}$$

to

$$\mathcal{O}(\text{W-MSA}) = 4hwC^2 + 2M^2hwC \tag{3.2}$$

M is usually set to 7. While this windowed approach successfully reduced the scaling to linear for a fixed M, this also results in a loss of global information, as the model is not able to attend to overlapping regions. For this reason, each alternating transformer block uses shifted window partitioning, where the windows are shifted to allow overlap over the previous arrangement, as visualised in figure 3.8. [35] Liu et al. introduce one final change by replacing the absolute

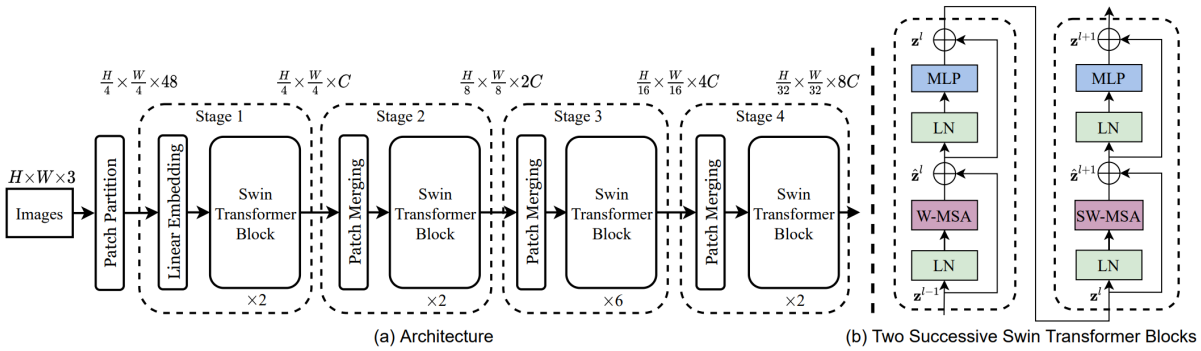


Figure 3.9: The architecture of Swin-T (left) and Swin-Transformer blocks using windowed and shifted window MSA (right) [35]

positional embedding layer from the vanilla ViT with a relative position bias added to each attention head:

$$Attention(Q, K, V) = Softmax(QK^T/\sqrt{d} + B)V \tag{3.3}$$

Elements in B are retrieved from the bias matrix initialised as $\hat{B} \in R^{(2M-1)\times(2M-1)}$ according to the axis values of M in the range $[-M+1, M-1]$. This fully replaces the positional embedding usually added at the start of the network, as the authors find this to actually decrease performance. This learned bias may also be used for higher resolutions for finetuning through bi-cubic interpolation. [35]

In 2022, Liu et al. introduce an improved variant of Swin, addressing two issues: the original model exhibits unfavourable characteristics when the model capacity is scaled up, resulting in

unstable training. Second, performance can suffer when transferring models across different window resolutions. The proposed solutions include using residual post-normalisation with cosine attention and a new log-spaced continuous position bias. [34]

3.2.3 Flexible Vision Transformer (FlexiViT)

Beyer et al. take issue with the reliance of ViTs on the patch size parameter. As has become evident, patch size is an important lever in controlling the computational complexity and predictive performance of the Transformer. However, changing the patch size after training results in large performance drops, even though the actual parameter initialisation is not that different between the same model being trained on different patch sizes. FlexiViT solves this limitation and matches or outperforms traditional ViTs trained on a fixed patch size without any added computational overhead. [5] The main strategy employed to achieve this robustness

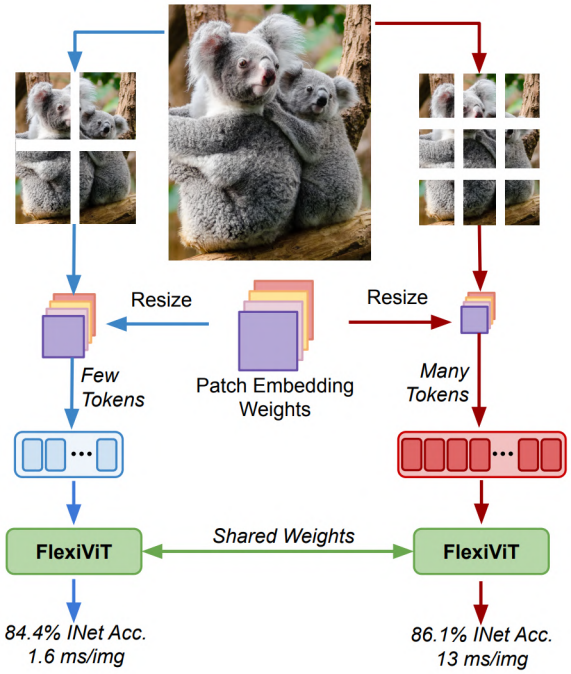


Figure 3.10: A high-level overview of the flexible patch size architecture of FlexiViT [5]

is to randomise the patch size during training. This entails dynamically resizing the patch and positional embedding layers. However, this is where the originally proposed bilinear interpolation causes issues: when applied to both the patch and its embedding, the resulting tokens can take on massively varying magnitudes, thus throwing off the model further downstream and resulting in a loss of predictive performance. This unintentionally learned inductive bias can be countered by normalisation after the embedding. But given that this would require changes to the underlying architecture, Beyer et al. instead settle on what they call the pseudo-inverse resize transformation (PI-resize). Figure 3.11 captures the effect of different rescaling operations, with PI-resizing performing significantly better. [5] Interestingly, the specific shape of either

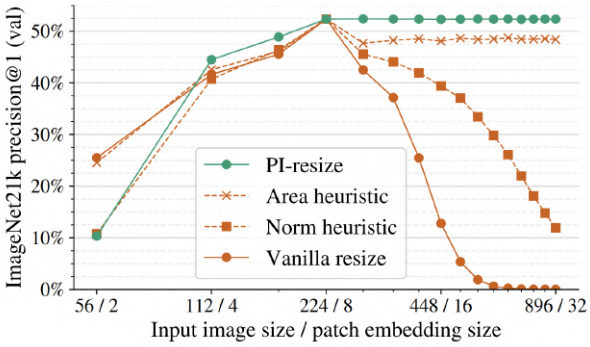


Figure 3.11: Effect of different resizing strategies applied to a pre-trained ViT-B/8 model [5]

embedding does not seem to contribute much to performance, the authors find. They choose 32×32 and 7×7 as the underlying size for the patch and position embedding respectively, while during training patch sizes are chosen at random between 48 and 8. [5]

The change made to the embedding interpolation also allows the adaption of standard, pre-trained ViT models to be adapted to flexible patch sizes after the fact. Beyer et al. use this for knowledge distillation inspired by DeiT to use a large, pre-trained ViT-B/8 model as the teacher. [5]

The proposed changes make FlexiViT unique in its ability to trade off accuracy for more efficient compute and its implementation requires no further changes to existing architectures, possibly expanding the usefulness of large, pre-trained models for smaller compute budgets. [5]

3.3 Hybrid Networks

3.3.1 ViT-ResNet

The authors of the original Vision Transformer paper also included a proposal for a hybrid model, using a convolutional stem as a feature extractor instead of partitioning the image into slices. Instead, the patch embedding is directly applied to the feature maps extracted from the CNN. These patches may also have a spatial size of 1×1, as a result of flattening the spatial dimension of the convolutional output and transforming it into the Transformer dimension. The classification token and position embedding are kept unchanged. [13] The paper suggests bundling ViT with a ResNet-26 or ResNet-50 in both ViT-B and ViT-L variants, with the last part of the notation R50+ViT-B/32 not referring to the patch size but the final downsampling ratio of the ResNet output. Dosovitskiy et al. show the hybrid architecture to surpass pure ViTs at lower computational budgets while these advantages disappear when scaled up higher. The effects of the CNN-induced biases can also be observed in the attention maps, where hybrid models show less localised attention compared to pure Transformers, as demonstrated by the

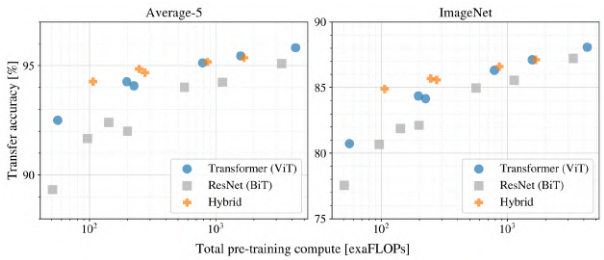


Figure 3.12: Compute required to reach a given accuracy for the vanilla Vision Transformer, ResNet and Hybrid ViT-ResNet [13]

last entry in figure 3.13. [13]

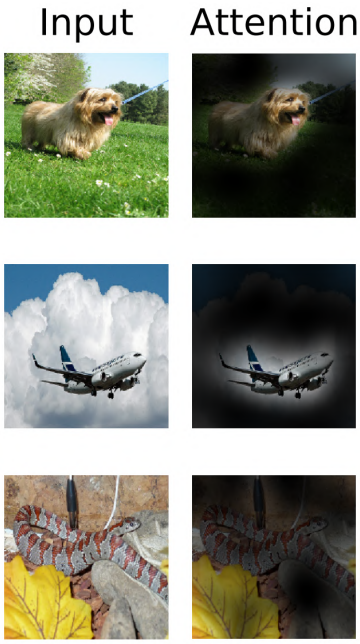


Figure 3.13: Attention maps for ViT and ViT-ResNet hybrid model (bottom image) [13]

3.3.2 CMT Hybrid Vision Transformer

Motivated by the deficiencies of Transformers at capturing low-resolution and multi-scale features as well as their quadratically scaling memory consumption for higher resolution images, Guo et al. propose another hybrid architecture. CMT (CNNs meet transformers) uses an adapted transformer block, which is enriched by depth-wise convolution and a convolutional stem for feature extraction. Four convolutional layers with stride 2 are used for resolution down-sampling, following a stage-wise architecture adopted from CNNs. This is meant to help with extracting multi-scale features without the need for higher-resolution inputs. As a final modification, the class token traditionally used for classification is replaced by an average pooling operation. [20]

Figure 3.14 shows the CMT architecture next to a ResNet and DeiT., highlighting the stage-wise architecture adapted from leading convolutional networks. Before each stage, the input is downsampled by 2x through convolution and layer normalisation. Global average pooling and a projection layer then feed into the dense classification head to form the final layers of the model. [20] The CMT block, illustrated in figure 3.14, is made up of three modules: local

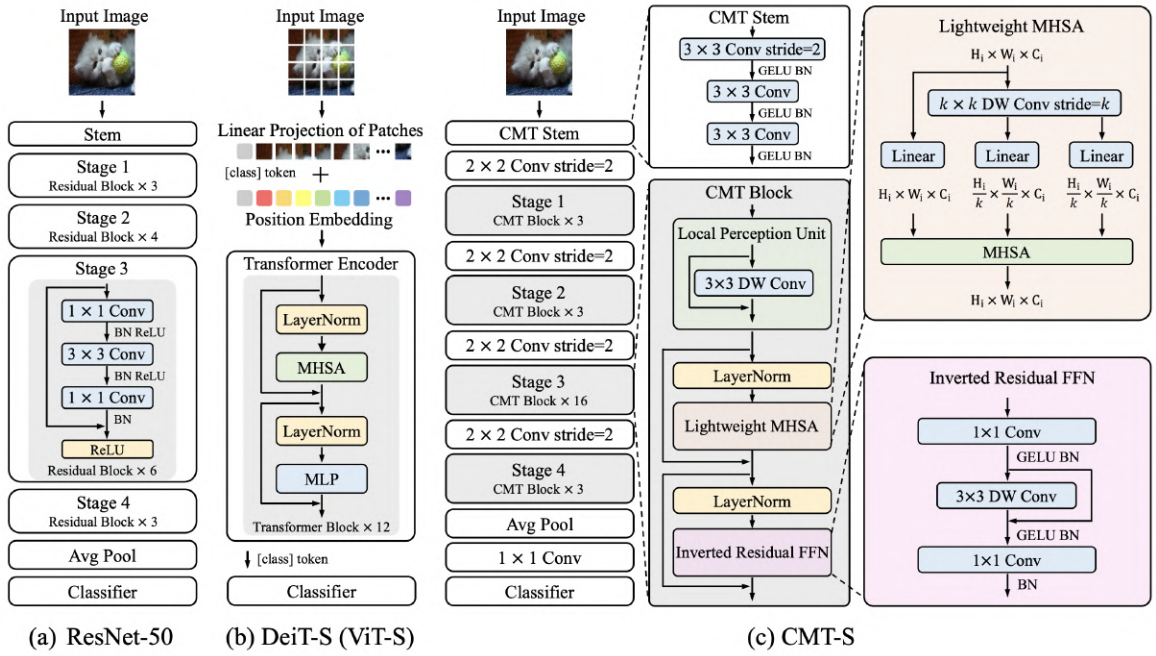


Figure 3.14: Network architectures of ResNet-50, DeiT-S and CMT-S[20]

perception unit (LPU), lightweight multi-head self-attention (LMHSA) and an inverted residual feed-forward network (IRFFN) [20].

Local Perception Unit

The positional encoding used in the original Vision Transformer conflicts with the requirement of translation invariance. To combat this, Guo et al. propose the local perception unit (LPU) to extract local features using depth-wise convolution $DWConv()$. For an input $\mathbf{X} \in R^{H \times W \times d}$, with $H \times W$ being the resolution of the input with d feature dimensions, LPU is defined as: [20]

$$LPU(\mathbf{X}) = DWConv(\mathbf{X}) + \mathbf{X} \tag{3.4}$$

Lightweight Multi-head Self-attention

The self-attention operation is modified by applying $k \times k$ depth-wise convolution on the key $\mathbf{K}' = DWConv(\mathbf{K}) \in R^{\frac{n}{k^2} \times d_k}$ and value $\mathbf{V}' = DWConv(\mathbf{V}) \in R^{\frac{n}{k^2} \times d_v}$ vectors before the self-attention calculation. Additionally, a learnable relative position bias $\mathbf{B}$ is added to each module using random initialisation. To adapt this to other input sizes, bicubic interpolation can be used. [20]

$$LightAttn(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Softmax(\frac{\mathbf{Q}\mathbf{K}'^T}{\sqrt{d_k}} + \mathbf{B})V' \tag{3.5}$$

Inverted Residual Feed-forward Network

Taking inspiration from inverted residual blocks, a depth-wise convolution follows an expansion layer which feeds into a projection layer. Furthermore, the shortcut connection now excludes the activation layer. According to the report, this allows the network to extract additional local information without much added computational cost. [20]

The computational complexity of the CMT is defined as:

$$\mathcal{O}(CMTblock) = 10nd^2(1 + 0.2/k^2) + 2n^2d/k^2 + 45nd \tag{3.6}$$

with $k \geq 1$ being the reduction ratio in the LMHSA, [20]

3.4 Other Optimisations

This section presents three approaches that have proven to enhance the performance of Vision Transformers without making any significant changes to their architecture. While some of these techniques can also be applied to convolutional networks, they have shown to be most beneficial when used in conjunction with Vision Transformers.

3.4.1 Sharpness-Aware Minimisation

Foret et al. conducted research investigating the effect of the loss landscape on the generalisation performance of a model. They found that only optimising for training loss can result in sub-optimal model quality. Instead, they propose a process named sharpness-aware minimisation (SAM), which aims to minimise both the loss value as well as the loss sharpness. [14]

By looking for parameters within neighbourhoods of uniformly low losses instead of only individual parameters with low loss values, they claim that gradient descent can be performed more efficiently, thus improving model generalisation. Figure 3.15 shows a comparison of the loss landscape of a standard ResNet trained with stochastic gradient descent (SGD) against the same model trained using SAM, resulting in a much wider minimum. [14]

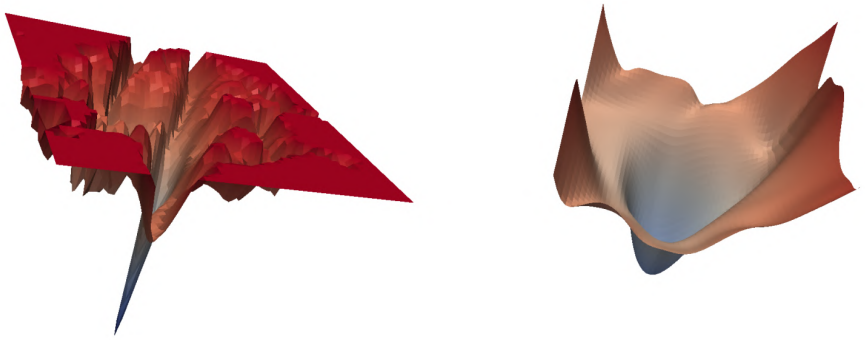


Figure 3.15: Sharp loss landscape of a ResNet trained with SGD (left) compared to the same model trained using sharpness-aware minimization (right) [14]

The paper applies SAM to both training models from scratch as well as for fine-tuning from standard pre-trained models, showing improvements for both types in a variety of common benchmarks like CIFAR-10, CIFAR-100 and ImageNet. Table 3.4 shows the results for fine-tuning, as they are more relevant to the problem at hand. [14]

Dataset	EffNet-b7 + SAM	EffNet-b7	Prev. SOTA (ImageNet only)
FGVC_Aircraft	$6.80_{\pm 0.06}$	$8.15_{\pm 0.08}$	**5.3** (TBMSL-Net)
Flowers	$\mathbf{0.63}_{\pm 0.02}$	$1.16_{\pm 0.05}$	0.7 (BiT-M)
Oxford_IIIT_Pets	$\mathbf{3.97}_{\pm 0.04}$	$4.24_{\pm 0.09}$	4.1 (Gpipe)
Stanford_Cars	$5.18_{\pm 0.02}$	$5.94_{\pm 0.06}$	**5.0** (TBMSL-Net)
CIFAR-10	$\mathbf{0.88}_{\pm 0.02}$	$0.95_{\pm 0.03}$	1 (Gpipe)
CIFAR-100	$\mathbf{7.44}_{\pm 0.06}$	$7.68_{\pm 0.06}$	7.83 (BiT-M)
Birdsnap	$\mathbf{13.64}_{\pm 0.15}$	$14.30_{\pm 0.18}$	15.7 (EffNet)
Food101	$7.02_{\pm 0.02}$	$7.17_{\pm 0.03}$	7.0 (Gpipe)
ImageNet	$15.14_{\pm 0.03}$	15.3	**14.2** (KDforAA)

Table 3.4: Top-1 error rates for finetuning EfficientNet-B7 [14]

While the models used in the original paper mostly include ResNets of different sizes this thesis will investigate the effectiveness of SAM specifically on Vision Transformers. Chen et al. found that applying SAM to Vision Transformers can lead to performance that outperforms ResNets of similar sizes and matches Transformers without large-scale pretraining or strong data augmentations [6]. They report an improvement of 5,3% (74,6% to 79,9% with SAM) for ImageNet for a ViT-B/16 model trained with and without SAM [6]. The authors also note that using SAM allows the model to be trained for more epochs before overfitting [14]. This could

turn out to be very beneficial for vision transformers with a high number of parameters which might tend to overfit easily on small to medium-sized datasets.

As the computational budget for this thesis is limited to a single, fairly modest GPU, full-scale pretraining using SAM is not an option. Experiments will therefore be limited to fine-tuning standard pre-trained weights.

3.4.2 Regularisation and Data Augmentation

Steiner et al. attempt to tackle the problem of the enormous data requirements of the original Vision Transformer that is needed to compete with convolutional networks. They find that added regularisation and augmentation helps the transformer overcome its weaker inductive bias and match the results of a ViT trained on ten times as much data. In contrast, the original ViT paper used inception-style preprocessing, which entails only light augmentations like flipping the image, applying random brightness and saturation and random cropping. [53]

Specifically, the authors use stochastic depth regularisation "with linearly increasing probability of dropping layers" [53], which was proposed by Huang et al.and Mixup and RandAug for data augmentation.[27]

Stochastic Depth Regularisation

Stochastic depth regularisation works by randomly dropping a subset of layers for each mini-batch by bypassing them using the identity function during training. Huang et al. report reduced training time as well as a drop in the test error on the CIFAR-10 dataset. [27]

MixUp

Mixup describes a data-agnostic augmentation strategy that creates virtual training images by using the knowledge that linearly interpolated feature vectors should result in linearly interpolated targets:[65]

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j, \qquad \text{where } x_i, x_j \text{ are raw input vectors} \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j, \qquad \text{where } y_i, y_j \text{ are one-hot label encodings} \end{aligned} \tag{3.7}$$

(x_i, y_i) and (x_j, y_j) are two images and their labels which were randomly selected from the training data with $\lambda \in [0, 1]$, sampled from a Beta distribution. The simplicity of this implementation allows for minimal computational overhead while also improving the robustness

of the model against corrupt labels or adversarial examples in the training data. Adversarial examples describe images that have been altered in a way that is imperceptible to the human eye but affect the performance of machine learning model. [65]

RandAugment

The authors of the RandAugment paper propose an automated data augmentation approach with a significantly reduced hyperparameter search space compared to previous strategies. As large search spaces add training complexity, they also make it challenging to find the ideal regularisation strength for any given model and dataset size. Thus RandAugment only exposes two hyperparameters - N for the number of sequentially applied transformations per image and M, as the magnitude for each transformation. This reduces the search space to 10^2 options compared to competing augmentation strategies with 10^{32} (AutoAugment in [9]) up to 10^{64} (Population Based Augmentation in [25]) possible combinations while matching or exceeding their performance. [10]

RandAugment includes the following transformations [10]:

- identity
- auto contrast
- equalize
- rotate
- solarize
- colour
- posterize
- contrast
- brightness
- sharpness
- shear X
- shear Y
- translate X
- translate Y

While the paper adapting both of these augmentation techniques to Transformers claims to achieve the same performance compared to a dataset roughly 10x its size, the authors also highlight that this approach still requires similar computing time to get the same results [53]. They report that for small datasets (tested for the Pet37 dataset with around 3000 images) a ViT trained from scratch cannot match the results achieved by transfer learning. For a medium-sized dataset (Resisc45), which is comparable in size to the model recognition dataset (30.000 images), results may come very close without quite matching transfer results but require up to "two orders of magnitude more compute and performing a heavy search" [53].

4 Methodology

This chapter will outline the methodology used to select, train and evaluate the networks for further examination. Also, the model recognition dataset will be introduced and examined using exploratory data analysis, showcasing some of the domain-specific challenges of this use case. Finally, this chapter discusses the baseline performance achieved by the recognition model that is currently deployed in production.

4.1 Approach

This thesis set out to compare the effectiveness of the Vision Transformer architecture for medium-sized datasets. To achieve this goal, the first step includes selecting the most promising model architecture of each type. A thorough search was conducted to choose the most promising Vision Transformer, convolutional neural network and transformer-convolutional hybrid architectures. Factors considered included their impact on the ImageNet benchmark and overall popularity based on citation counts, follow-up research and wide stream adaption on computer vision competitions and overall presence and prominence in the computer vision community. The final selection was already introduced in chapter 3. In addition, a ResNet-50 model was also included to serve as an additional baseline by highlighting the advances made by more modern convolutional architectures over the years.

A practical concern for the final selection was also the availability of pre-trained weights for the Keras / TensorFlow framework, as research and early experiment suggest that training a model from scratch is both inefficient and therefore unreasonably time-consuming, cost-prohibitive and will likely still yield worse accuracy. Therefore, all models evaluated in this thesis are initialised with weights pre-trained on the ImageNet-1k dataset.

Tests were also conducted on models pre-trained on the much larger ImageNet-21k dataset with subsequent finetuning on the ImageNet-1k subset. However, early results did not show significant performance benefits and since these weights were also not available for every model implementation, it was ultimately decided to stick to ImageNet-1k weights.

Next, given the large number of models to evaluate and the restrictions on computing resources, a first round of model selection will compare models "out-of-the-box", with hyperparameter tuning limited to finding the largest possible batch size and choosing an initial learning rate that

would not stagnate the training process. These results will form the foundation for identifying first patterns and choosing the most promising architectures of each type for further finetuning.

Once these models have been finetuned and ideally reached their full potential, a more thorough evaluation can take place to establish a final ranking between architectures and model variants. Final questions regarding the inference speed, interpretability and specific mistakes made by Vision Transformers compared to convolutional architectures can then be investigated by looking at the top-performing model of each type.

The overall experimental setup is constrained by the restrictions on computing power. With expected training times in excess of over 2 hours per epoch or more, some compromises had to be made regarding the experimentation setup, as the use of k-fold cross-validation was deemed too computationally expensive. Instead, the traditional train-validation-test split was employed (see details below). The test set would only be used for the final evaluation of a model and was prohibited from being used for any finetuning as this would disrupt its statistical significance in confidently estimating the test error for the total population.

4.2 Dataset Introduction

The model recognition data set was initially assembled from two sources: images provided by Atlon, a subsidiary focused on leasing Mercedes-Benz vehicles to businesses and consumers, that took and collected images whenever leased cars were returned to document their condition. As a complementary source, the publicly available collection of Mercedes-Benz marketing images was used to supplement classes with only a few samples. Over time, as the Geofence and Selfaudit process was introduced in more markets, the data coming in would be used to update the existing dataset. Conveniently, as the image collection process already associates images with their vehicle identification number, no manual labelling is usually required as the vehicle class can be extracted from said VIN. Today, the dataset is updated roughly four times per year, sometimes more often to reflect changes in newly released models. [1]

4.2.1 Exploratory Data Analysis

The model recognition dataset contains a total of 42.474 images across 64 vehicle classes. The number of images per class varies drastically, with the most frequent class being the 2006-2018 Mercedes Sprinter with almost 3.500 images. Meanwhile, the GLE SUV Coupe 2015-2019 only includes 25 training images, which is the cut-off threshold set for including new classes. The whole extent of the class imbalance is visualised in figure 4.1.

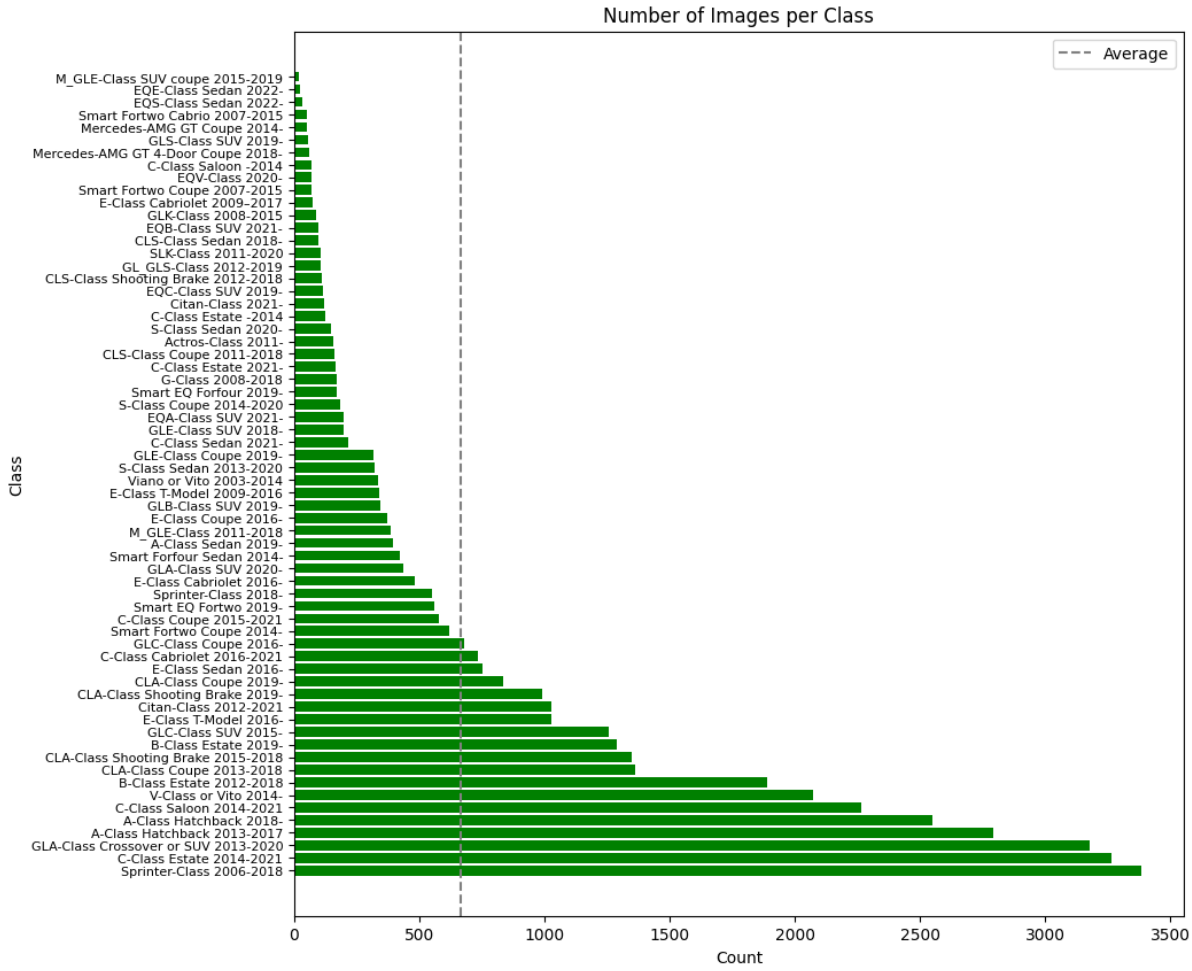


Figure 4.1: Number of images per Vehicle class

On average, each vehicle model provides 663 images for training. 25% of classes have to manage with less than 110 images, with half of all classes containing up to 329 images and only 25% of classes holding more than 771 training images.

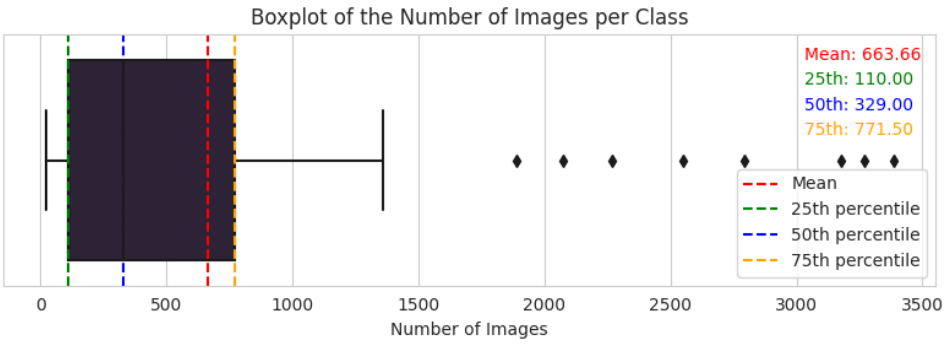


Figure 4.2: Boxplot for the distribution of training images per class

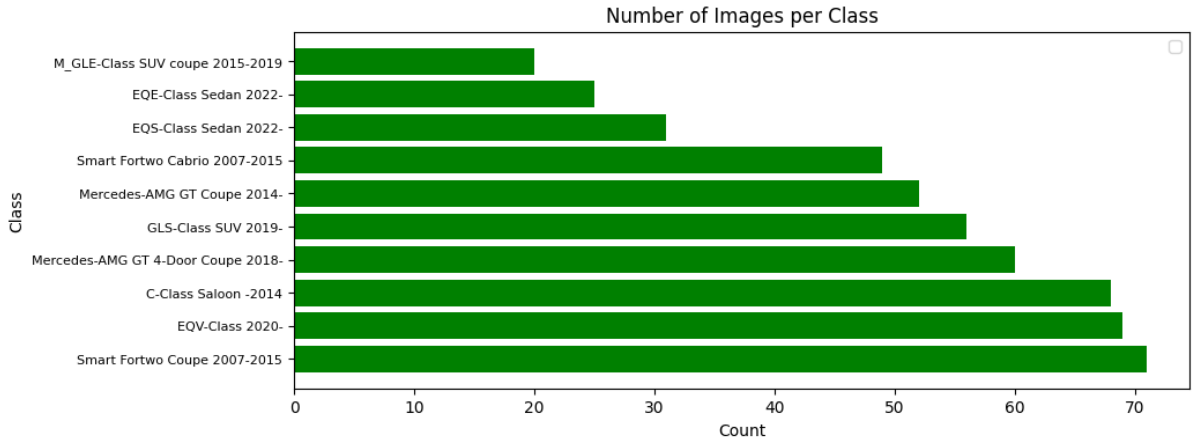


Figure 4.3: Number of images per Vehicle class for the 10 classes with the fewest images per class

For further analysis, one can group vehicle classes into their main model groups, disregarding information about body style and generation. Note that this is done exclusively for the purpose of this analysis and is generally not referenced anywhere else in the training pipeline. This shows that the C- and A-Class are the most frequent images in the dataset, which makes sense, given their popularity in vehicle sales. Notably, the Sprinter class is also highly represented, as there are additional data sources available for this class, namely Mercedes-Benz Van Rental. Other observations include the small number of samples for the only recently released electric model ranges like the EQE, EQS or EQV. One final outlier here is the Actros class, as it is not a Mercedes-Benz passenger vehicle but a full-sized truck.

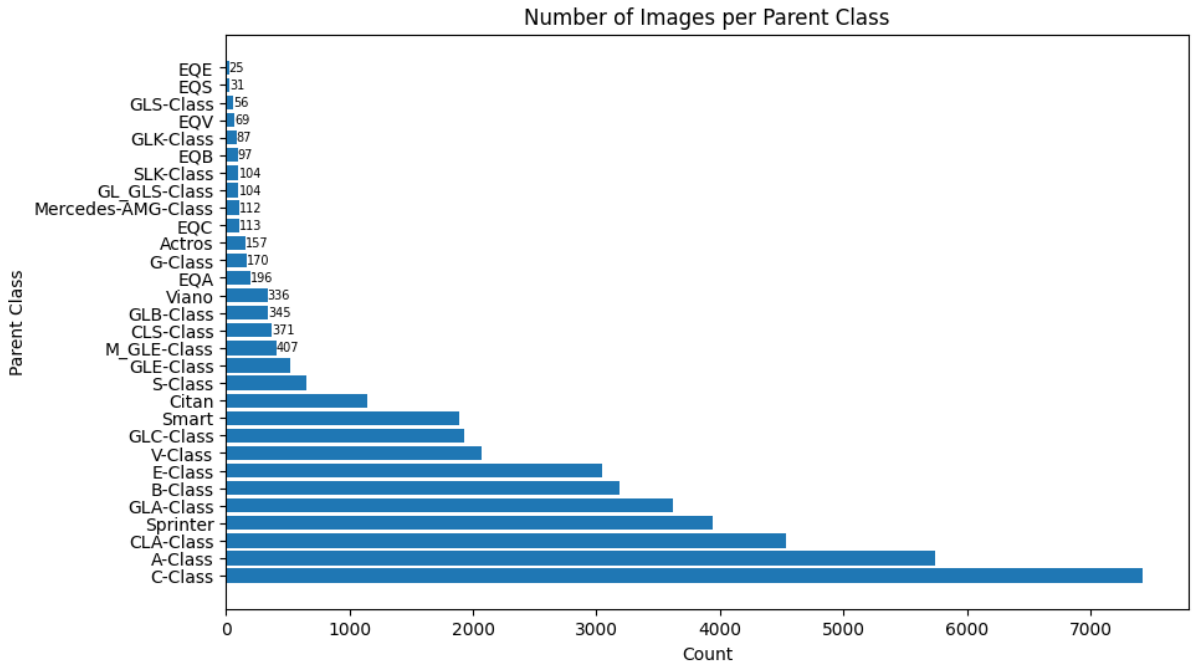


Figure 4.4: Number of images grouped by parent class

Looking at individual box plots for each parent model showcases how different vehicle models have more individual subclasses, specifically body style variants or multiple generations, as the spread of the boxplot indicates. Parent classes with only a single subclass are visualised by a single line. Interestingly, despite the C-Class having the subclass with the most training images, the distribution of the boxplot indicates a higher number of subclasses with fewer individual training samples than compared to the A-Class. In fact, the A-Class parent group maintains the highest mean number of samples across its subclasses.

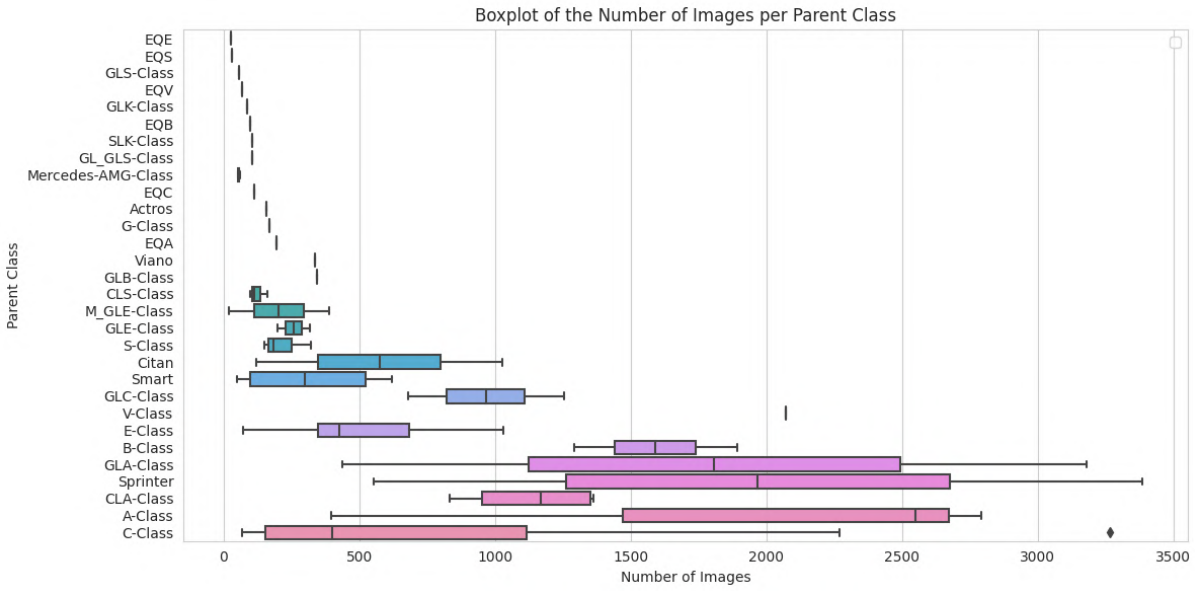


Figure 4.5: Boxplot across all parent vehicle groups

Figure 4.6 shows a random selection of training images. Across all images, the average resolution is 460 pixels in width and 706 pixels in height, with the highest resolution image being 1426 by 724 pixels. The lowest resolution throughout the dataset is 253 pixels in height and 415 pixels in width. Occasionally duplicate images were found in the dataset. However, as they were still within the correct vehicle class, this has not been a big issue so far. During the implementation of this thesis, one example has also been found that does not show any car at all, which will be discussed further in chapter 5.

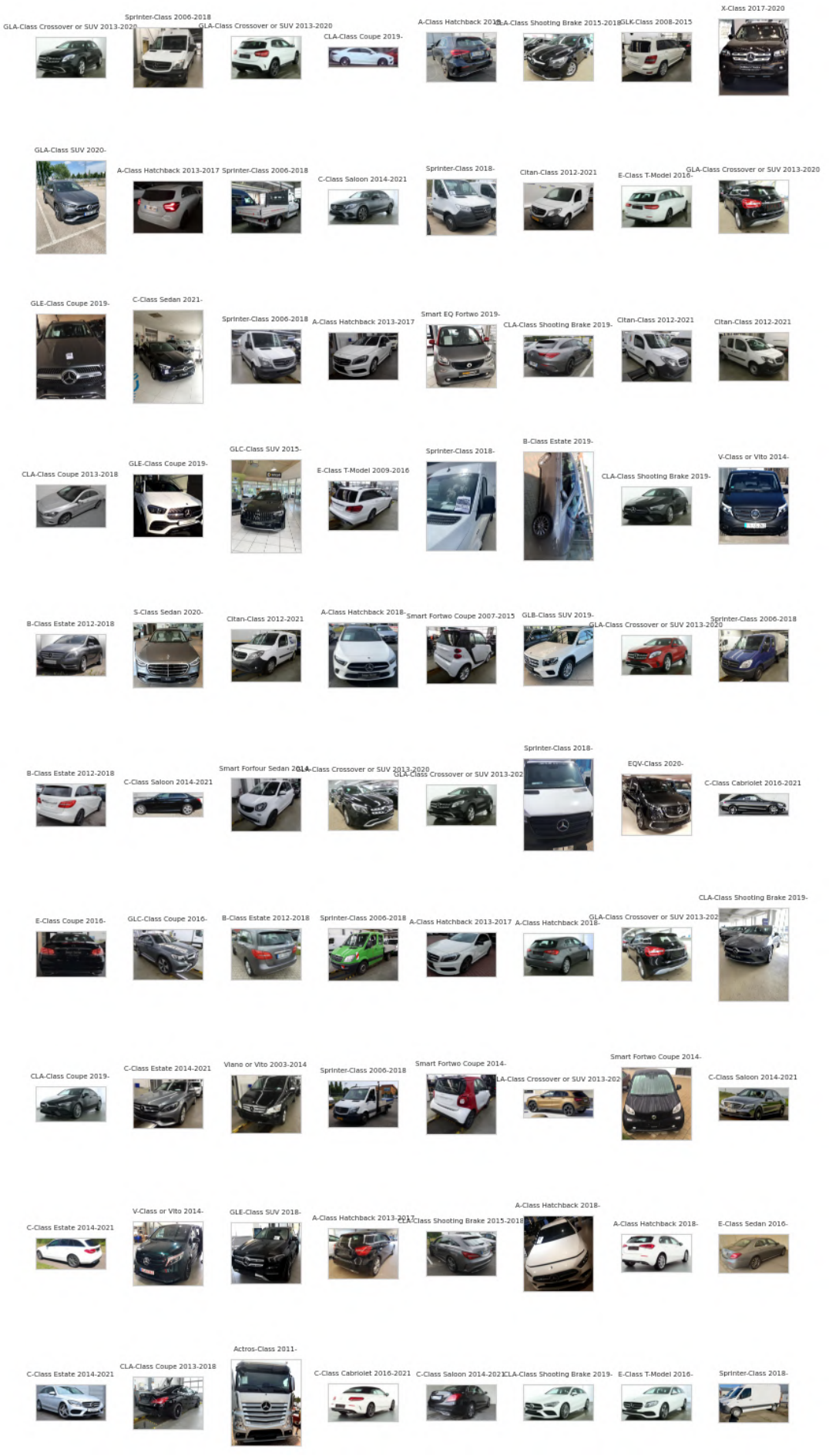


Figure 4.6: 80 randomly sampled training images

4.2.2 Domain-specific Challenges

While deep learning models for image classification have come a long way in the last few years the task of automotive model recognition remains a particularly challenging one. Not only do models have to deal with typical problems for computer vision models like image noise, low-resolution photos and changes in lighting. But they also have to be able to classify images based on very small visual differences that can be very hard to spot, even for humans. Additionally, the model can not, or only in a very limited capacity, rely on other visual cues like colour, as these features are only weakly correlated to different models.

Furthermore, the setting in which these images will be taken by an employee of a dealership will likely not result in the most optimal outcome in terms of consistency or quality due to factors such as time pressure or simply lack of space to physically take photos from. Tightly parked cars in showrooms or parking lots may result in skewed or partially obstructed images. Backgrounds can also be noisy, including other vehicles, furniture, buildings, humans or unwanted elements like fingers or hands in the image.

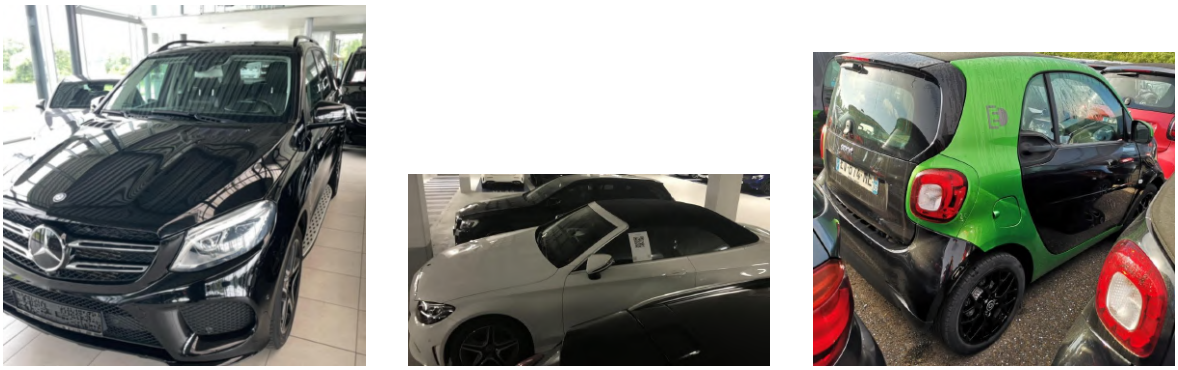


Figure 4.7: Example images of partially obstructed cars taken in tight spaces

Lighting and weather conditions can also not be expected to be consistent, as images may be taken indoors or outdoors and in varying weather conditions, like snow or rain, in artificial, natural or mixed lighting. This is further complicated by the reflective nature of car body panels and windows, often resulting in distracting reflections.

Figure 4.8: Example images of weather influences and distortion introduced by wide-angle lenses

The smartphone cameras used for taking these images will likely also struggle with dynamic range, leading to noticeably under or over-exposed photos or blurry images due to long shutter speeds or noisy images if the camera can not capture enough light. Naturally, images will also be shot at vastly different focal lengths, ranging from ultra-wide to slight telephoto (though most smartphone cameras shoot images around the 24 - 28mm focal length full frame equivalent, which is rather wide), possibly using digital zoom and will be taken from various heights and angles. Images shot with ultra-wide focal lengths can be especially challenging as a lot of distortion is introduced that can significantly alter the proportions of a depicted car. Conversely, the training dataset also includes professionally shot images sometimes taken in studios or photo-realistic rendered images that are used for advertising. While ideally training images taken in realistic dealership environments are strongly preferred due to a lack of real-world images provided by the markets this is often unavoidable, especially for newly introduced models.

Figure 4.9: Example for training images taken outside the expected dealership environment

Figure 4.10: Examples of distorted or blurred images and prominent reflections

Lastly, even ignoring the technical aspects, some model variants simply look very similar. Especially models within the same general vehicle category and the same generation can often be hard to distinguish (4.11). This can get even harder when it comes to differentiating between two different body styles within the same model (limousine, coupe or estate, 4.12). This can even be impossible if images are taken completely head-on. For this reason, we do not separate facelift variants of any given model, as changes between them can be almost impossible to detect or non-existent on the exterior. On the other hand, some vehicle classes do show significant visual differences in their bumper design or their exhaust arrangement across different trim levels (like AMG-line) even within the same model (4.13). As further separating each model by trim levels would balloon the total number of classes by up to five times or more, while also requiring even more training data to achieve acceptable accuracy, images are grouped instead only by model and year.

Figure 4.11: Example images of similar-looking models (C-Class, E-Class, S-Class)

Figure 4.12: Examples of different body styles of the same model (C-Class coupe, estate and limousine)

To remedy some of these challenges, in production, models are grouped based on their parent class (i.e. C-Class limousine, estate, cabriolet and coupe have the parent class C-Class) when comparing incoming requests. This is done for a number of reasons: a) not all images can "physically" be classified confidently, as their perspective may not allow any judgment about the body style. b) grouping introduces an additional level of "fuzziness", essentially acting like a top-k-classification, as even if the model predicts the wrong model, it is likely that the result is still of the same parent class. c) requirements may change in the future. More accurate predictions may be needed for advanced fraud detection or other unrelated use cases within Mercedes-Benz.

Figure 4.13: Example of the visual differences between two trim levels

4.3 Experimental Settings

The training is conducted on a virtual machine with half an NVIDIA K80 GPU with 12GB of video memory and a 150W TDP, 56GB of system memory and a 6-core Intel Xeon E5-2690 v3 CPU (NC6 instance on Azure). Before training, images are converted into a sequence of binary records (tf.records data format), lowering storage requirements and improving loading times when using the TensorFlow framework [58].

TensorFlow and Keras were used in version 2.11.0 with a global seed for weight initialization and data augmentation (see 4.3) across all experiments.

Train-Test Split

The train-test-split was generated with 85% of images going towards the training set and the remaining 15% being split evenly across the validation and test set. Stratified sampling was used to make sure each set contains a representative distribution of the overall dataset. While the conscious decision was made to also train classes with very few training images to observe their performance, as they could still be excluded from the list of supported models further down the deployment pipeline, a minimum number of images still has to be supplied. Otherwise even with stratified sampling one could not be sure that the validation and test set actually captured these minority classes let alone provide meaningful evaluation metrics about them. This threshold was chosen at 25 images.

Preprocessing and Data Augmentation

Preprocessing followed the unique requirements of each model, ranging from simple scaling of input values to a range between 0 and 1, to more elaborate ways of normalisation and preprocessing. Experiments used one or more out of three data augmentation strategies. Basic augmentation implements common basic augmentations like random rotation, cropping, brightness and contrast adjustments among others. This is commonly called "inception-style" preprocessing, referring to the steps proposed by the original Inception model authors.

Images are dynamically resized at loading time according to the specifications of the training config file. Given prior testing and results shown in literature, the aspect ratio is not preserved. Instead, both edges are scaled to the same square resolution. As the GPU video memory is limited to 12 gigabytes, batch sizes usually cap out at 64 for smaller networks, and sometimes have to be reduced to as few as 12 images per batch when training larger models with more parameters or computationally heavy optimisation steps like sharpness-aware optimisation.

Pre-training

All evaluated models are finetuned from a pre-trained ImageNet-1k checkpoint. The implementations for each model are sourced from a variety of different packages and repositories and are usually not supplied by the authors of the original papers. Small details in the specifics of the implementation and the process used to migrate or train the pre-initialised weights can therefore differ from the originally reported numbers.

Mixed-precision Training

Micikevicius et al. propose Mixed Precision Training, using only half-precision floating point numbers (float16 instead of float32) to reduce the memory consumption of deep networks [38].

By using a number of tricks to make up for the loss of the reduced numerical range, like storing a copy of full-precision weights and loss scaling, this technique can improve training performance without sacrificing accuracy, especially for newer GPUs with optimised hardware for float16 computation, enabling up to 8 times faster matrix multiplications [41]. Unfortunately, the NVIDIA K80 available for training is not equipped with the tensor cores required to benefit from this speed increase [11]. In the tests conducted, mixed precision training did not noticeably speed up training or allow for significantly larger batch sizes. Additionally, during experimentation, some pre-trained and compiled models have proven challenging to convert to and then train with mixed precision. As mixed precision generally has next to no effect on the final accuracy of a model, any experiments mentioned have been trained without mixed precision unless otherwise stated.

Future development of the training pipeline should revisit this technique as the potential benefits are promising and any technical hurdles along the way are potentially easier to overcome for a single model than trying to adapt multiple models from different packages and origins.

Hyperparameter Tuning

Hyperparameter tuning was performed for the most promising models of each architecture type. A hyperband optimiser was used to efficiently traverse the search space. Parameters considered for tuning included learning rate, weight decay, optimiser, and augmentation strategies, while other influences like image sizes were tested manually. The optimised grid search was performed on a stratified subset of around 20% of the full training data to speed up the search. Batch size was not included in this search as the restrictions on GPU memory did not allow a large range of values to be tested. As such, the batch size remains unexplored as an additional form of regularisation.

Training and Evaluation

During training, validation metrics are calculated after every epoch and logged to MLFlow. "Early stopping" and "reduce learning rate on plateau" callbacks are used to prevent the model from overfitting and help it converge in the later stages of training. The concept of learning rate warm-up was also explored but did not offer any additional benefits. Model selection relied on the best-achieved validation metrics while the findings in chapter 5 report the metrics generated on the test set.

4.4 Baseline Performance

The original model recognition training pipeline was implemented using the TensorFlow Slim (tf.slim) framework at the end of 2018. However, tf.slim has since been deprecated. While this has no immediate performance implications as the pipeline can still be used, this does still confine the future development of the pipeline, as no new features are being added to the high-level API and publicly available implementations for new models are often only available for other frameworks. Furthermore, the previously used pipeline only included rudimentary logging of training and evaluation metrics, with results not being reliably tracked for reproducibility and evaluation. For example, validation loss or accuracy was not monitored during training and the only evaluation metrics used were accuracy and weighted top-5 recall.

	Inception ResNet v2 (tf.slim)	**Inception ResNet v2 (Keras)**
Accuracy	0,942	0,948
Top-5 Recall // Top-3 Accuracy	0,997	0,995
F1-Score	-	0,816
Macro Avg. Precison	-	0,850
Macro Avg. Recall	-	0,808
MCC	-	0,901

Table 4.1: Evaluation metrics for the implementation of Inception ResNet v2 in TensorFlow Slim and Keras

This encouraged the decision to migrate the training to a new high-level API for TensorFlow, called Keras, which is now an official part of the TensorFlow library. Together with the introduction of MLFlow as a framework to automatically track experiments by logging relevant hyperparameters, metrics and plots, this resulted in an overall modernisation of the training pipeline with better reproducibility and the option to easily control run settings like image sizes or data augmentation strategies through a config file.

However, this transition also meant that the legacy model would not be able to be compared easily to new models trained with Keras. For this reason, the training setup for the baseline model had to be replicated as closely as possible in Keras by matching all high-level hyperparameters like batch size, optimiser, learning rate and weight decay. This retrained model then serves as the baseline for further evaluation.

Solely based on the comparison of accuracy and recall, the retrained model matches and even slightly exceeds the original TensorFlow Slim variant. Ultimately it remains hard to asses how accurately this retrained model gets to the original network, as the implementation details in the underlying frameworks can differ and factors like dissenting sources of pre-trained weights may result in overall slightly different results.

5 Findings

This chapter presents the results of the experiments conducted to evaluate the performance of Transformer, convolutional and hybrid image classification models. It describes the initial model selection, a detailed evaluation of the best-performing model of each architecture type and the effect of additional optimisation techniques. Finally, it attempts to capture the trade-offs the respective architectures make, by looking at inference speed and interpretability.

5.1 Qualitative Evaluation

5.1.1 Model Selection

For the first round of model evaluation, a selection of models was trained as a baseline. Specifically, hyperparameter tuning was limited to finding the largest possible batch size and choosing a non-stagnating learning rate and optimiser. All models were trained using basic inception-style preprocessing and augmentation. Figure 5.1 shows the best accuracy per model. Overall, the spread between models is relatively small and without any obvious outliers. Also, no clear pattern has formed yet indicating any clear performance benefit for either Transformers or CNNs. However, "out of the box", two Transformers, one hybrid model and two more recent CNNs are able to outperform the baseline Inception ResNet v2 model. However, the same does not apply to the vanilla variants of the Vision Transformer. As expected, the ResNet-50, included for reference, ranks at the bottom end of the scale, being the oldest and least complex architecture here. For further evaluation, a selection of the highest-performing baseline models was chosen for further hyperparameter tuning. Figure 5.2 reports the improved accuracy on fine-tuned models. While accuracy is an intuitive metric to judge models on, it does not tell the whole story, as the underlying dataset is fairly unbalanced. Figure 5.13 attempts to showcase how the relative rank between models changes when evaluated on different metrics. The motivation behind this can be showcased by the performance of the Inception ResNet v2 model. While it achieves a strong accuracy of 0.948, its F1-Score and macro average metrics are sub-par. This indicates that the network has mainly learned to classify high-frequency classes and struggles to correctly predict minority classes. Generally, models achieve slightly higher macro average precision than recall, with the only exception being EfficientNet.

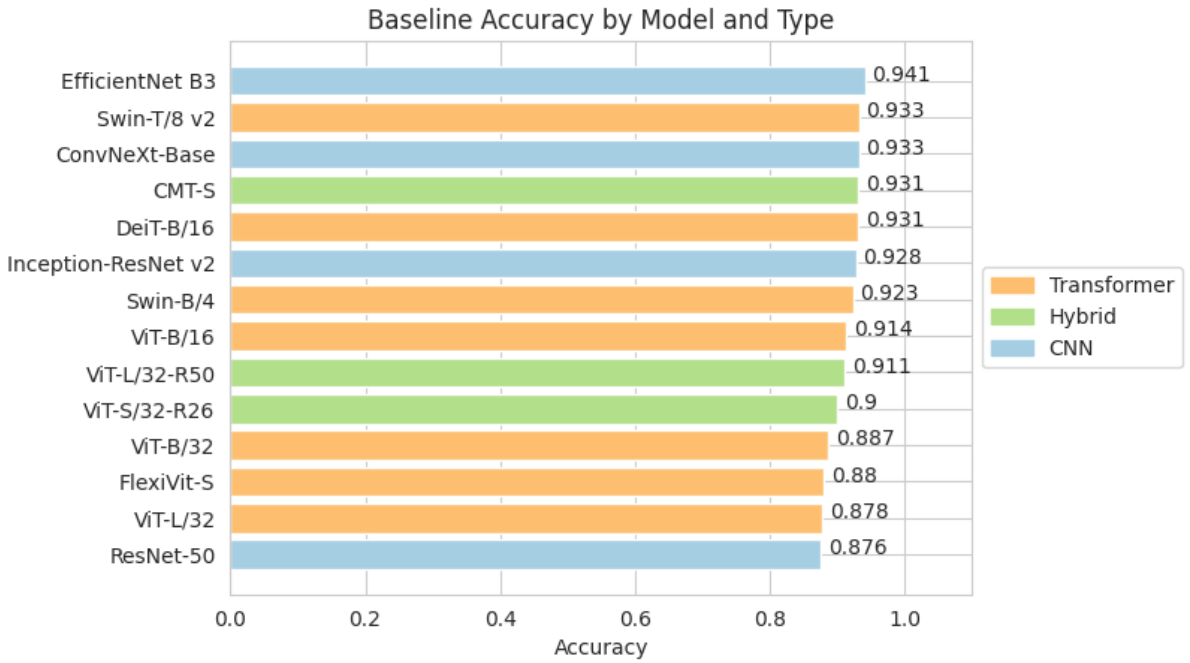


Figure 5.1: Baseline Accuracy by Model and Network Type

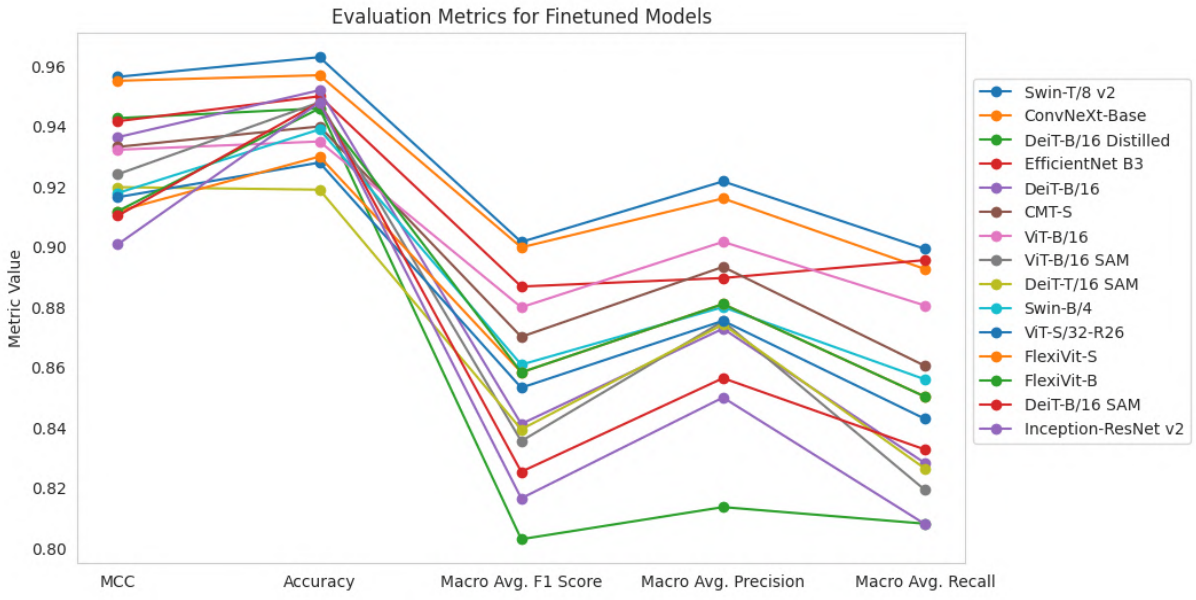


Figure 5.3: Model Performance for MCC, Accuracy, F1-Score, Macro Average Precision and Recall

Again though, the achieved scores remain very close, with the Swin-T/8 v2 taking the top spot, achieving an accuracy of 0,963 and an F1-score of 0,902. Compared to the Inception ResNet v2 baseline, accuracy is up 1,5% and the F1-score is up by 8,6%. Section 5.2 will further investigate

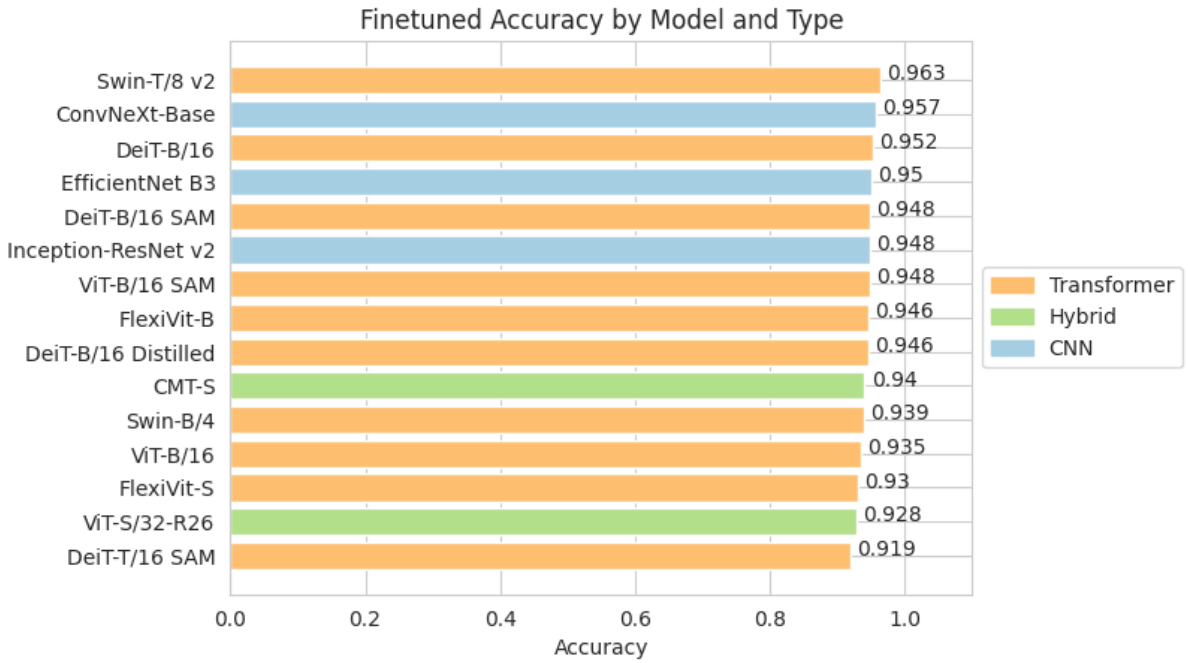


Figure 5.2: Baseline Accuracy by Model and Network Type

how these numbers manifest themselves in actual predictions. As the ImageNet benchmark has traditionally been used to judge the progress of newly released architectures, figure 5.4 tries to investigate how the reported top-1 accuracies in every model's original paper correspond to the accuracy obtained on this custom dataset. Note, however, that the results displayed here should be taken with a grain of salt, as the models submitted to the benchmark can be expected to have been tuned and optimised beyond what would be possible for every model in this evaluation. As such, the specific performance ranking may shift slightly, as values are already very close. It should therefore just serve as a demonstration that, generally speaking, just choosing the model based on the highest ImageNet score is not a guarantee that it will be the best-performing model for a given task.

Type	Model	Accuracy	F1 Score	Macro Avg. Precision	Macro Avg. Recall	MCC
CNN	ResNet-50	0,876	0,724	0,715	0,693	0,793
	ConvNeXt-Base	**0,957**	**0,900**	**0,916**	0,893	**0,955**
	Inception-ResNet v2	0,948	0,816	0,850	0,808	0,901
	EfficientNet B3	0,95	0,887	0,890	**0,896**	0,942
Hybrid	ViT-S/32-R26	0,928	0,853	0,875	0,843	0,917
	CMT-S	**0,94**	**0,870**	**0,893**	**0,861**	**0,933**
Transformer	ViT-B/16	0,935	0,880	0,902	0,881	0,932
	ViT-L/32	0,878	0,711	0,742	0,712	0,839
	ViT-B/32	0,887	0,719	0,755	0,731	0,835
	ViT-B/16 SAM	0,948	0,836	0,875	0,819	0,924
	DeiT-B/16	0,952	0,841	0,873	0,828	0,936
	DeiT-B/16 SAM	0,948	0,825	0,856	0,833	0,910
	DeiT-B/16 Distilled	0,946	0,803	0,814	0,808	0,943
	DeiT-T/16 SAM	0,919	0,839	0,874	0,826	0,920
	Swin-B/4	0,939	0,861	0,880	0,856	0,918
	Swin-T/8 v2	**0,963**	**0,902**	**0,922**	**0,899**	**0,956**
	Swin-B/16 v2	0,946	0,891	0,909	0,888	0,946
	FlexiViT-S	0,93	0,858	0,881	0,850	0,912
	FlexiViT-B	0,946	0,840	0,843	0,821	0,932

Table 5.1: Best model scores for Accuracy, (macro) F1-Score, Precision, Recall and MCC (best score, both baseline and fine-tuned)

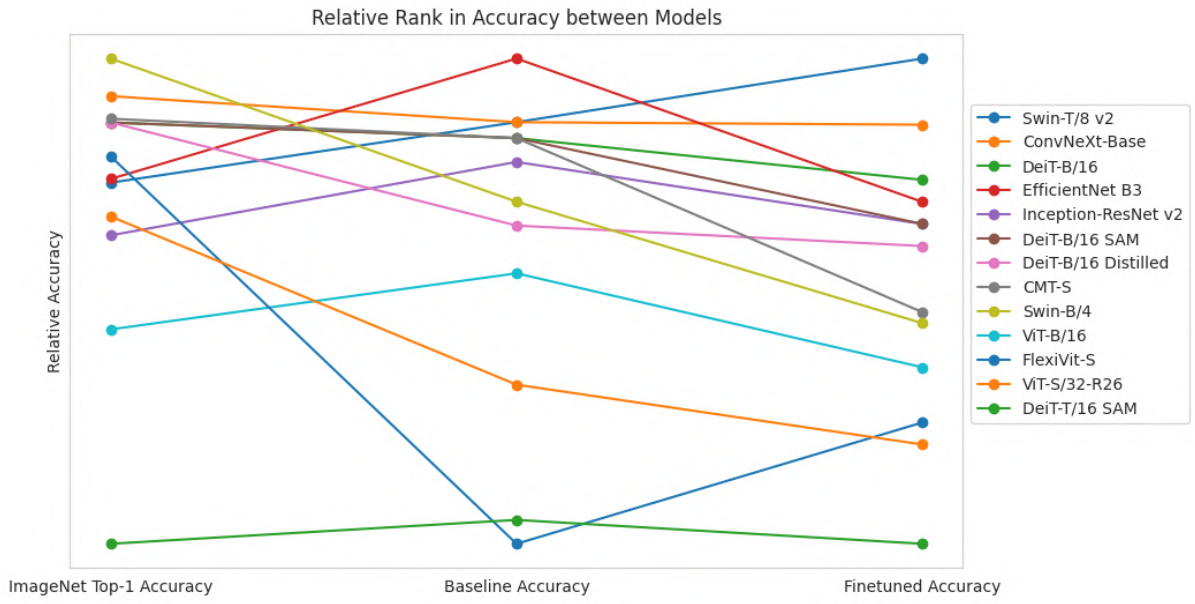


Figure 5.4: Normalised relative accuracy on the ImageNet benchmark reported in the model's original paper compared to the accuracy achieved on the real-world dataset before and after finetuning

Image Resolution

To study the effect that the input resolution of images has on model performance networks were trained in different resolutions ranging, including 224x224, 288x288, 300x300 (CNN only), 384x384, and 512x512. Overall, no significant performance benefit was observed for training at higher resolutions than 300x300. Looking at example images scaled to this resolution close up usually still reveal more than enough details for a human to identify the model correctly. Only for images where the vehicle is very small in the photo and takes up significantly fewer pixels does an input resolution of 224x224 become limiting. However, while in practice these differences remain small, they come at the cost of smaller batch sizes and longer training times. The effect on inference speed will be presented in a later section.

For the final models, a resolution of 300x300 (ConvNeXt) and 288x288 (Swin, requires size to be multiple of patch size 16) was chosen that also matches the resolution of the baseline Inception ResNet v2 model.

Model Complexity

The selection of models included a wide range of network depths and parameter counts. Looking at the best-performing models it appears that even smaller models (Swin-T/8 v2 with 28M parameters) are able to fit the training data well. In general, testing the same architecture in different sizes does not reveal any beneficial scaling for higher-parameter-count models for this particular dataset. While not included in figure 5.5, larger networks, specifically ViT-L variants, were also tested that present a substantial increase in parameters over ViT-B (86M versus 307M). Unfortunately, training models of this size start to reveal deficiencies in the available computing resources, as batch sizes are restricted to less than 8 images per batch before running out of memory and significantly slower training times. For this reason, models larger than 100M parameters were not considered for other architectures and are not pictures in the plot in figure 5.5.

However, results in this regard may be subjective to the specifics of the experiment setup and further hyperparameter tuning and training for longer than 20 epochs or 24 hours might reveal additional benefits. Given the already high performance even on small models, this direction was not further explored. Notably, the best-performing Vision Transformer (Swin-T/8 v2) was able to capture better performance with 3x fewer parameters compared to the best CNN (ConvNeXt Base).

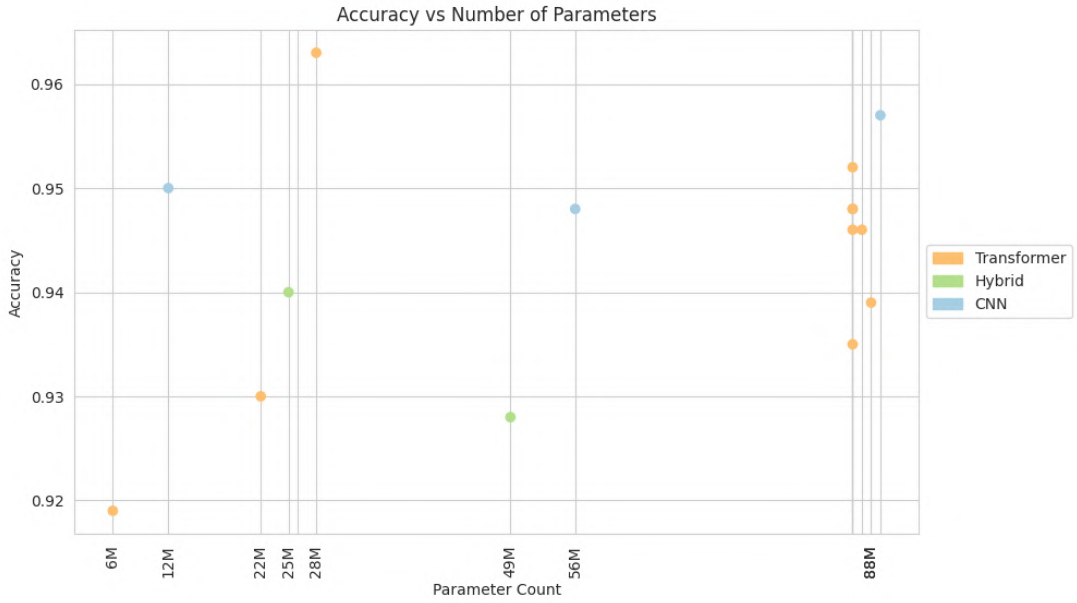


Figure 5.5: Accuracy versus the number of parameters for different models

Model	SAM	Accuracy	F1-Score	Macro Avg. Precision	Macro Avg. Recall	MCC
ViT-B/16	without SAM	0,935	0.836	0.875	0.819	0.924
	with SAM	**0.948**	**0.880**	**0.902**	**0.881**	**0.932**
Swin-B/4	without SAM	0.925	0.821	0.856	0.821	0.918
	with SAM	**0.954**	**0.890**	**0.922**	**0.895**	**0.941**

Table 5.2: Training Results achieved with and without using Sharpness-aware Optimisation for ViT-B/16 and Swin-B/4

5.1.2 Effect of Optimisation Techniques

Sharpness-aware Minimisation

One of the advantages of sharpness-aware minimisation is that it can be applied to any model. In practice, this makes it very easy to adapt to new architectures. Sadly, as the compute available during this thesis was severely limited, the experimentation concerning SAM was limited. Specifically, experiments included training a vanilla Vision Transformer with and without SAM and then applying it again to the best performing Transformer architecture.

The results achieved are promising, yielding decent uplifts in accuracy and overall generalisation performance. Less desirable is SAM's effect on training times, increasing the time per epoch by 1,85x (295min versus 160min per epoch). This is a result of needing two forward and two backward passes per batch as the second gradient is computed. However, models trained with SAM still achieve higher accuracy in a given period of time, even when trained for fewer epochs.

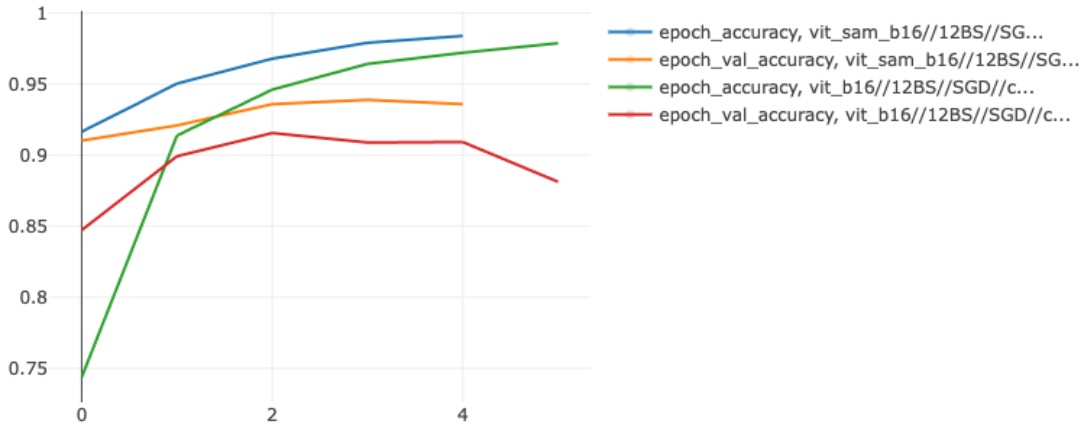


Figure 5.6: Epoch Accuracy and Validation Accuracy (left) and Epoch Loss and Validation Loss (right) for a ViT-B/16 trained with and without SAM

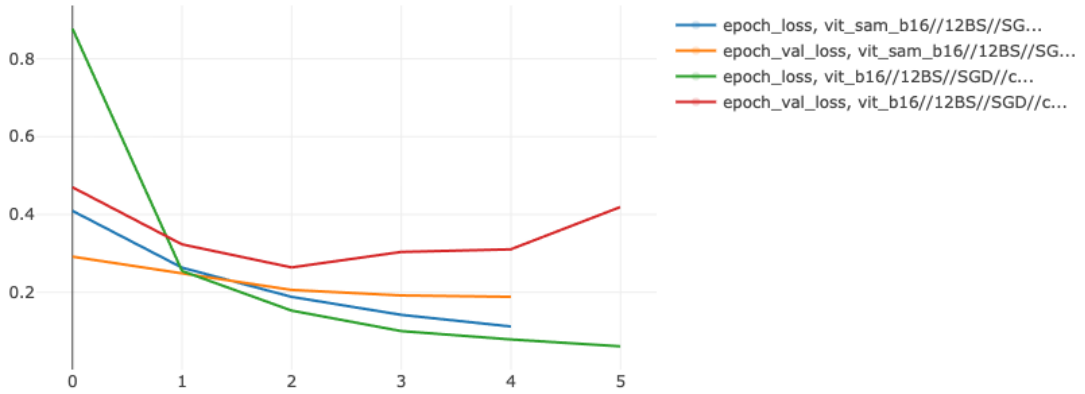


Figure 5.7: Epoch Accuracy and Validation Accuracy (left) and Epoch Loss and Validation Loss (right) for a ViT-B/16 trained with and without SAM

Data Augmentation

The authors of the "How to train your ViT?" paper have conducted comprehensive ablation studies on the best mix of augmentation strategies, the results of which served as the baseline for fine-tuning [53]. As such, the surveyed models have been trained using either basic augmentation ("inception-style" preprocessing) or full augmentation (RandAugment plus Mixup) to reduce the search space during hyperparameter tuning.

Figure 5.8: Examples of images augmented with Mixup

Combining this augmentation strategy together with the Swin v2 architecture led to the best-

performing model tested here. Comparing the effects on training versus validation performance, it becomes apparent that full augmentation successfully prevents the network from overfitting the training data. However, as the training accuracy seems to stagnate fairly early at around 85%, this raises the question if the chosen augmentation strategies may have introduced too much noise, prohibiting the model from fitting it any better. Interestingly, it seems to have the desired effect of boosting generalisation performance above any other tested model. This finding presents an interesting research question for the future, by studying how much noise can be introduced into the training data until the model stops improving in terms of generalisation and actually starts to lose overall model performance.

It should also be noted that the dataset in question might be uncommonly sensitive to certain types of augmentations. For example, cropping and rotating images can easily lead to critical parts of vehicles being cut off, as in many images the vehicle fills the frame completely (see the right-most image in 5.9). While this might not be an issue for other types of common objects where one would hope to retain classification performance even when an image is only partially visible. However, for the type of images expected in this use case, this may become next to impossible for certain images and crops, especially centre crops, that only leave parts of the windshield and bonnet exposed, which can prevent confident predictions. This issue would require further finetuning and maybe an adaptation of the prebuilt RandAugment implementation to better cater for the specific dataset.

Figure 5.9: Examples of images augmented with RandAugment (n=9, magnitude=0.7)

To summarise, data augmentation has proven to play a critical role in improving generalisation performance for Vision Transformers and CNNs alike. Performance is boosted across the board and especially underrepresented classes can benefit from more virtual training examples. Interestingly, performance is already up compared to no augmentation within one epoch, thus not necessarily requiring additional training time (although the ceiling is higher before overfitting the training data). The specific augmentation strategy used also has a negligible effect on the training speed, as augmented images are created by the CPU and therefore do not require any extra GPU memory. Lastly, early results also indicate a certain robustness against too much artificial training noise, still resulting in strong generalisation performance but meriting further research.

5.1.3 Comparing the best-performing Model of each Type

Comparing the best results achieved by each architecture type shows how close these models are to each other. Looking at the Transformer and CNN model specifically, it is hard to determine a clearly superior model, as these results are within the margin of error. Further hyperparameter tuning or simply a different random initialisation would most likely be able to account for any observed differences. In comparison, the hybrid CMT model falls slightly further behind, also being beaten by other Transformer and CNN variants. As a result, further sections will focus on Swin-T/8 v2 and ConvNeXt Base as the competing models.

Figure 5.10: Accuracy, F1-Score, Macro Average Recall and Precision for the best-performing models per Type (Transformer, Hybrid, CNN)

Improvements over the Baseline Model

As eluded to in 4.4, a strict comparison to the model currently deployed in production is not possible. As such, the retraining results may not accurately reflect the actual performance of the old model. While the retrained Inception ResNet v2 model reaches the same accuracy as last reported for the original pipeline, its overall performance is easily surpassed by both Swin-T/8 v2, ConvNeXt and CMT-Small. Metrics like the F1-Score and the MCC report improvements of up to 9%, indicating much better classification capabilities for minority classes. This jump comes as a bit of a surprise, as the solely reported accuracy seemed to suggest decent performance. As a result, both evaluated Transformer and CNN models present a significant uplift in performance for underrepresented classes.

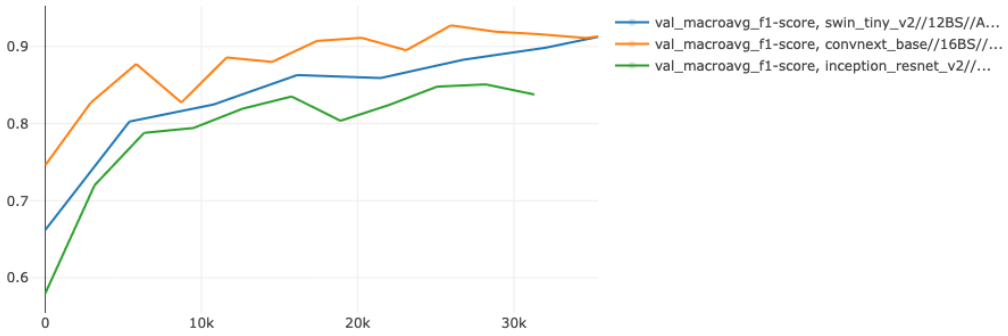


Figure 5.11: Validation F1-Score for Swin-T/8 v2, ConvNeXt and Inception ResNet v2

	Inception ResNet v2	**Swin-T/8 v2**	**ConvNeXt-Base**	**CMT-Small**
Accuracy	0.948	**0,963 (+0,02)**	0,957 (+0,01)	0,940 (-0,01)
F1-Score	0,816	**0,902 (+0,09)**	0,899 (+0,08)	0,870 (+0,05)
Macro Avg. Precision	0,850	**0,922 (+0,07)**	0,916 (+0,07)	0,893 (+0,04)
Macro Avg. Recall	0,808	**0,899 (+0,09)**	0,893 (+0,08)	0,861 (+0,05)
MCC	0,901	**0,956 (+0,06)**	0,955 (+0,05)	0,933 (+0,03)

Table 5.3: Baseline model Inception ResNet v2 in comparison to best Transformer, CNN and Hybrid model

5.2 Examples of Model Failures and Challenges

Starting off with a look at the top 10 worst-performing classes by precision for both models, it appears the Transformer model predicts more classes with less than 80% precision, going down as low as 57% for the AMG GT Coupe while the worst class sits at 72% for the convolutional model. Looking at the number of available training examples for the respective classes, it looks like the Transformer struggles slightly more with underrepresented classes compared to the CNN. However, these differences remain too small to draw any meaningful conclusions, as overall scores remain very similar. Extended visualisations comparing the F1-score for each class relative to the number of samples across all models reveal no significant trends or clear patterns proving the superiority of either model. However, as expected, a clear correlation between the number of samples and the training performance can be observed, with classes with over 1000 images never falling below 80%.

	Swin-T/8 v2			ConvNeXt-Base		
Rank	**Model**	**Precision**	**Training Samples**	**Model**	**Precision**	**Training Samples**
64.	Mercedes-AMG GT Coupe 2014-	0.571	43	GLE-Class SUV 2018-	0.722	170
63.	Smart Fortwo Cabrio 2007-2015	0.666	43	CLS-Class Shooting Brake 2012-2018	0.727	102
62.	C-Class Sedan 2021-	0.666	184	EQC-Class SUV 2019-	0.727	102
61.	EQS-Class Sedan 2022-	0.666	22	GLS-Class SUV 2019-	0.750	45
60.	C-Class Estate 2021-	0.700	140	Smart Fortwo Cabrio 2007-2015	0.750	45
59.	CLS-Class Shooting Brake 2012-2018	0.700	97	EQB-Class SUV 2021-	0.750	80
58.	Smart Fortwo Coupe 2007-2015	0.714	65	C-Class Sedan 2021-	0.777	193
57.	EQB-Class SUV 2021-	0.750	76	EQA-Class SUV 2021-	0.818	158
56.	CLS-Class Coupe 2011-2018	0.818	130	C-Class Estate -2014	0.818	102
55.	GLE-Class Coupe 2019-	0.821	259	Smart Fortwo Coupe 2007-2015	0.833	68

Table 5.4: Top-10 worst-performing classes by Precision for Swin-T/8 and ConvNeXt-Base

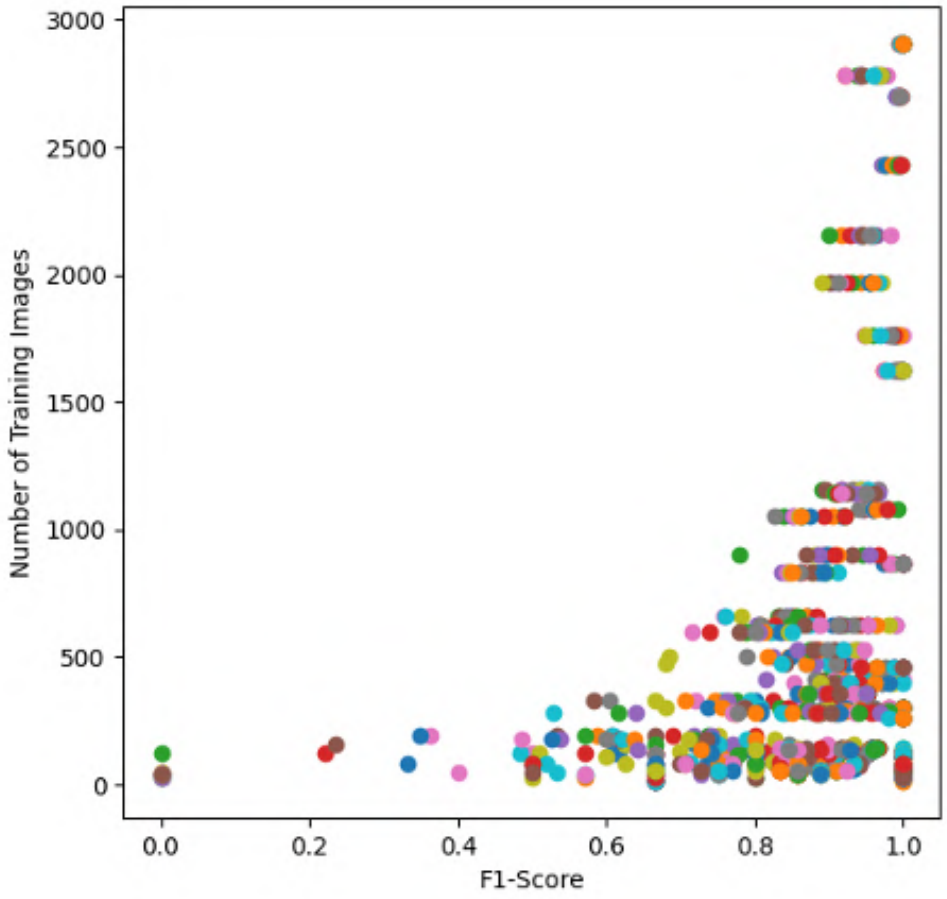


Figure 5.12: F1-Score per Vehicle Class relative to the number of samples of that class across all experiments and models

The Transformer model produced a total of 133 wrong predictions on the test set of 3186 images while the ConvNeXt network made 137 mistakes, thus resulting in practically identical accuracy. When examining the mistakes the models made it quickly becomes apparent that there is primarily one cause of mispredictions: poor-quality input images. As described previously in chapter 4, the quality of input photos can vary drastically, with some images being virtually impossible to classify correctly. On the contrary, given some examples, the performance seems

impressive, as the base model is correct in almost all cases. In fact, when investigating this further, it turns out that out of the 133 and 137 misclassified images, 75% and 70% respectively, are mistakes within the correct parent model group, only getting the body style or generation wrong. Given that the details of the implementation of the recognition model do not take the body style into account and instead only rely on the parent class of the vehicle, this discovery actually implies a significantly higher real-world accuracy than initially believed.

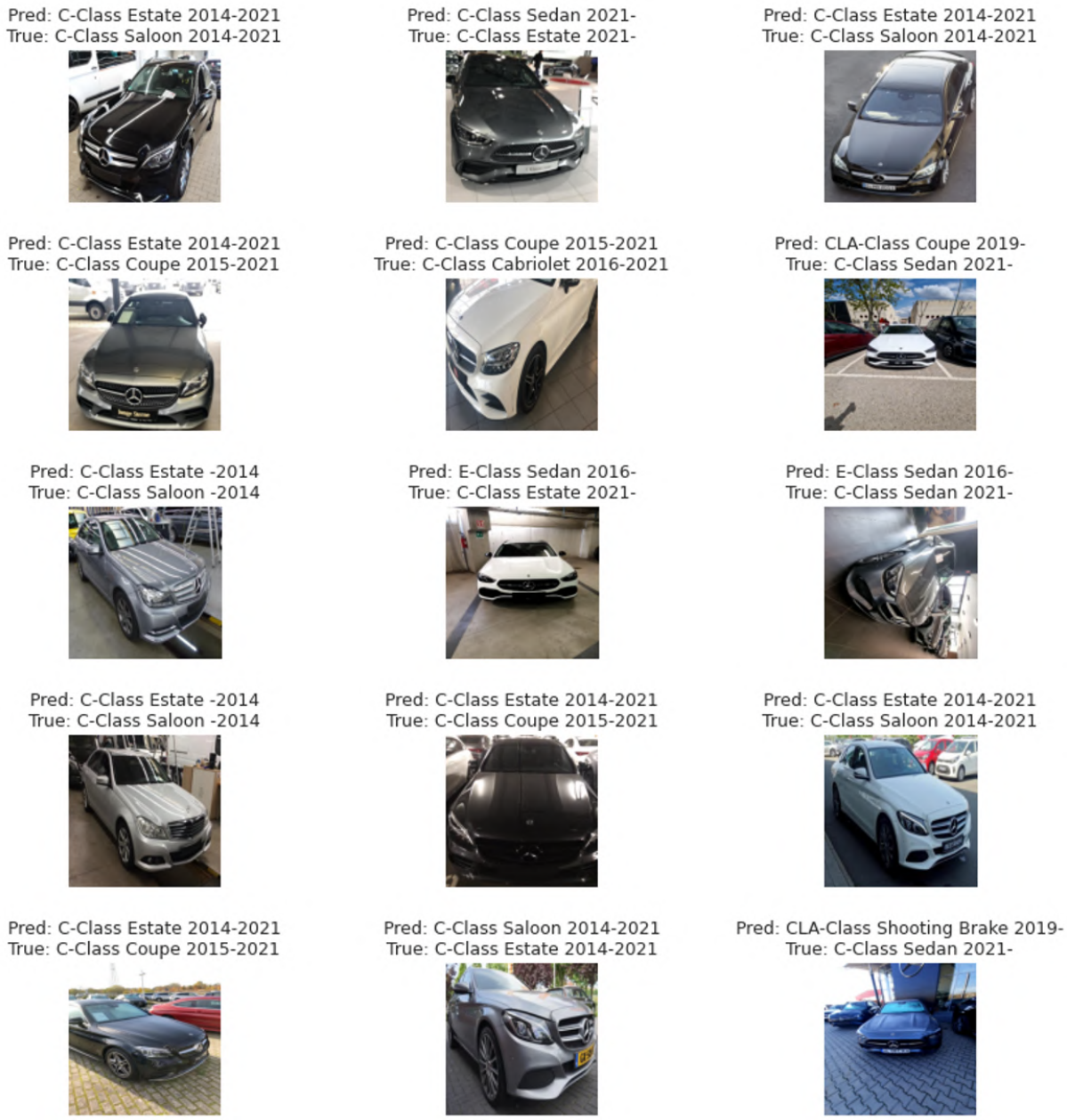


Figure 5.13: Examples of misclassifications predicted by Swin-T/8

This can be further emphasised by looking at a selected part of the confusion matrix, focusing on entries where either a true C- or E-Class was classified as something else or something else

was labelled incorrectly as a C- or E-Class. On closer inspection, most of these mistakes can be explained by the actual similarity of the models in question., with neither model exhibiting any especially noteworthy behaviour. This is additionally backed up when calculating the top-3 accuracy, with both models scoring above 99,5%.

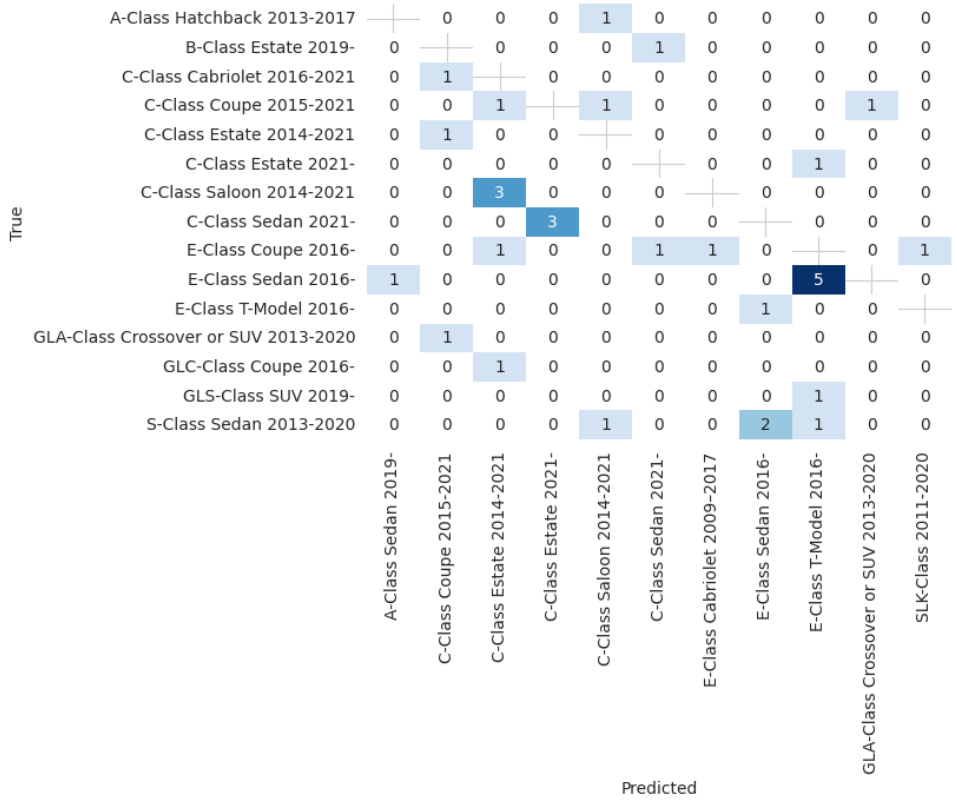


Figure 5.14: Confusion Matrix for C- and E-Classes (Swin-T/8)

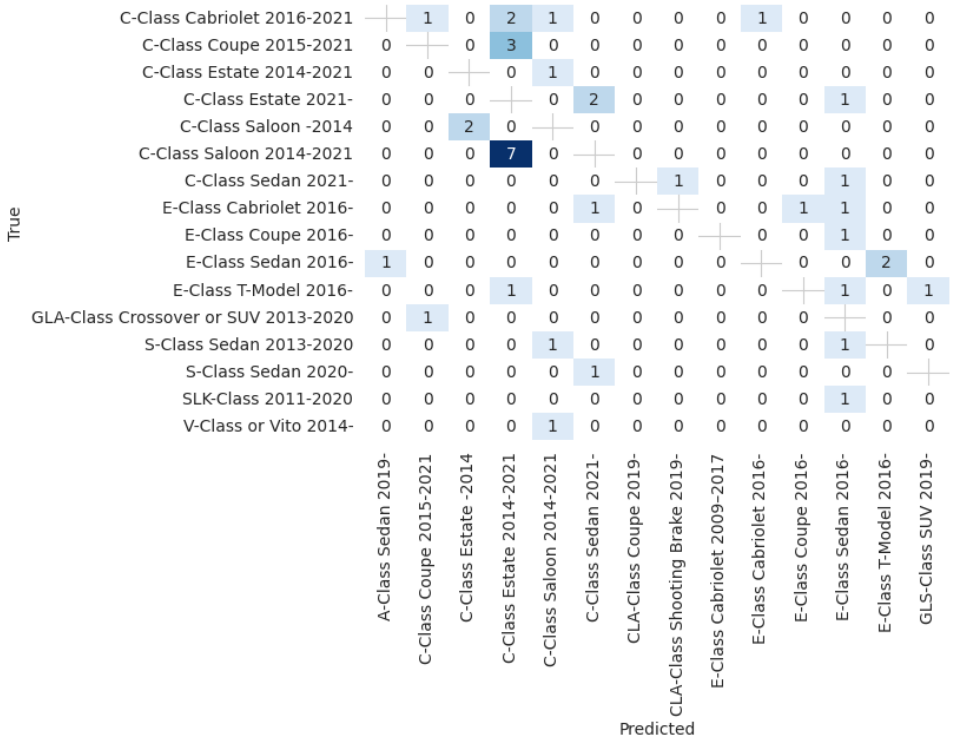


Figure 5.15: Confusion Matrix for C- and E-Classes (ConvNeXt-Base)

Lastly, one can look at the failed predictions where the class does not match at least the correct parent vehicle class. The corresponding images are presented in figure 5.16 (Swin) and figure 5.17(ConvNext) and include some more random errors. While this analysis also revealed an invalid image that does not actually contain a car at all, some of the other examples still seem like reasonable mistakes. However, while being quite an unusual image, both models fail to correctly identify the image of the G-wagon and struggle with the SLK model.

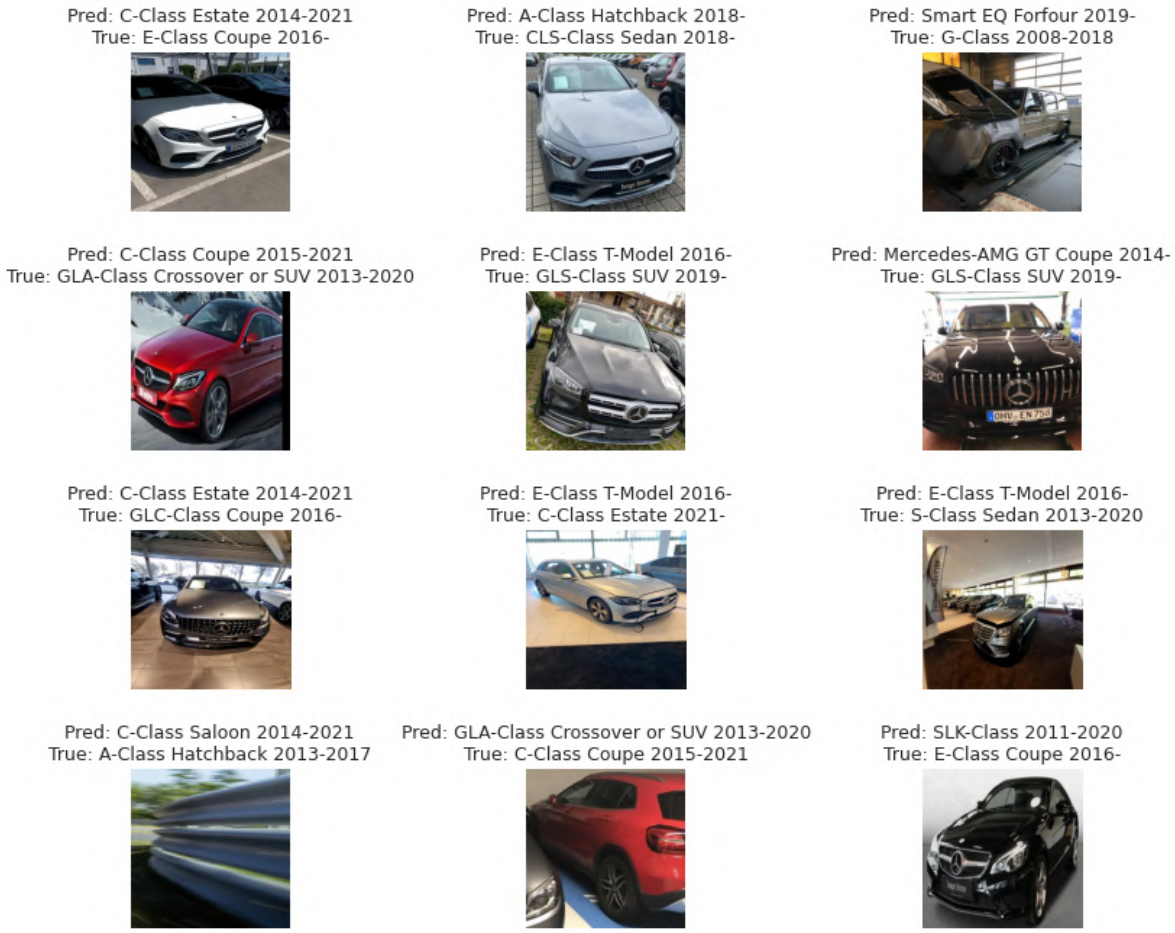


Figure 5.16: Examples of misclassifications predicted by Swin-T/8 v2

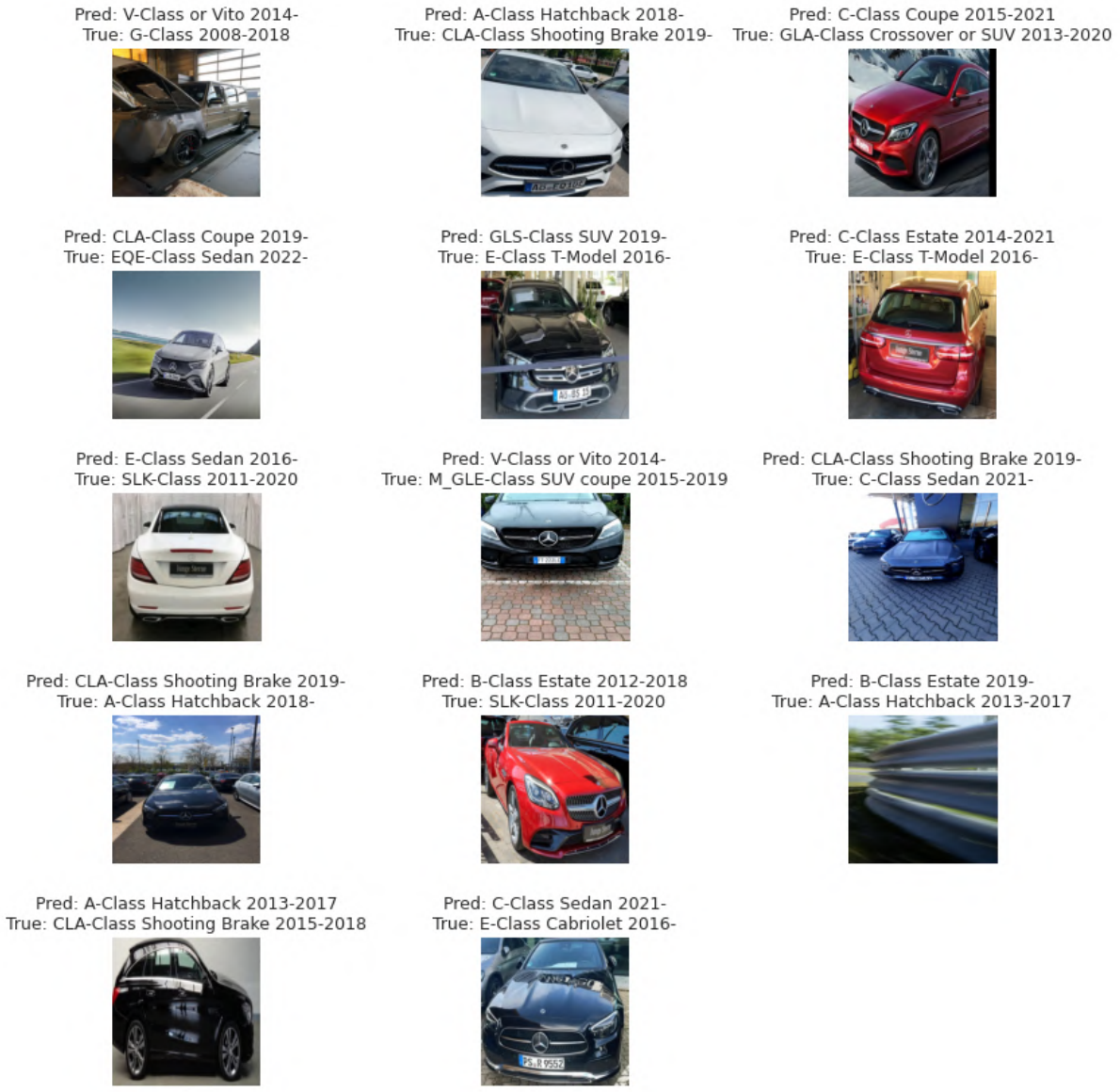


Figure 5.17: Examples of misclassifications predicted by ConvNeXt

In conclusion, close observation has revealed that a majority of errors can actually be attributed towards poor input images while models often still correctly classify the correct parent class, suggesting better real-world accuracy than originally assumed. Also, neither model could be shown to exhibit any concerning random behaviour as most mistakes are "human-interpretable". However, this also raises concerns about general noise in the dataset, meaning images that are virtually not able to be classified confidently. As this would be a dangerous way for the model to overfit the training data by learning to predict the body style even without actually being given enough information to do so, the importance of regularisation through data augmentation is highlighted. On the other hand, this might also put a ceiling on the accuracy that the model can realistically achieve on the dataset due to the noise introduced by these types of images.

	Image Size	Time per Image	Images per Second	% Time compared to Inception ResNet v2
FlexiViT-S	224	10,48	95,4	0,39
EfficientNet-B3	300	17,07	58,57	0,63
CMT-S	224	21,19	47,2	0,79
Swin-S/4	224	23,01	43,46	0,85
Inception-ResNet v2	299	26,91	37,15	1
ConvNeXt-Base	300	29,07	34,4	1,08
ViT-R26	224	41,29	24,22	1,53
Swin-B/8	224	41,85	23,9	1,55
Swin-T v2	288	48,1	20,79	1,79
ViT-B/16	304	48,61	20,57	1,81
Swin-S/8 v2	288	58,96	16,96	2,19
DeiT-B/16	384	88,48	11,3	3,29
Swin-B/8 v2	288	118,49	8,44	4,4
Swin-B/4	384	138,74	7,21	5,15

Table 5.5: Average inference time per image, number of predictions per second and relative time increase compared to fastest inference speed

5.3 Inference Performance

While model accuracy is often the primary point of attention, improvements there may also come at the expense of increased computing time per prediction. Since the Recognize service is expecting an increase in users over the coming years, this aspect should not be overlooked.

The results in table 5.5 show a general trend of CNNs being slightly faster than their Transformer counterparts when matched on resolution and parameter count. However, there are some notable exceptions: FlexiViT-S reached better accuracy than the baseline Inception ResNet v2 model with the inference being almost three times faster. While its performance does not quite match the Swin-T model, it remains a solid option if throughput were the primary concern. EfficientNet-B3 also shows strong results even at a significantly higher resolution. When tested at 224x224 resolution, results match FlexiVit-S.

Looking at Swin-T and ConvNeXt-Base, the difference in inference speed is larger than one might expect, given that Swin-T has just 1/3 as many parameters as ConvNeXt-Base. Comparing the two again at 224x224 resolution, the delta shrinks to only 33% slower performance down from 66%. It should also be noted that both networks are able to beat the baseline model in accuracy at a lower resolution and thus also undercutting it in inference speed. Additionally, performance differences have been observed for the Swin architecture for using different window sizes. This has the potential for further finetuning as these choices may be able to further help with image throughput with minimal sacrifices in accuracy.

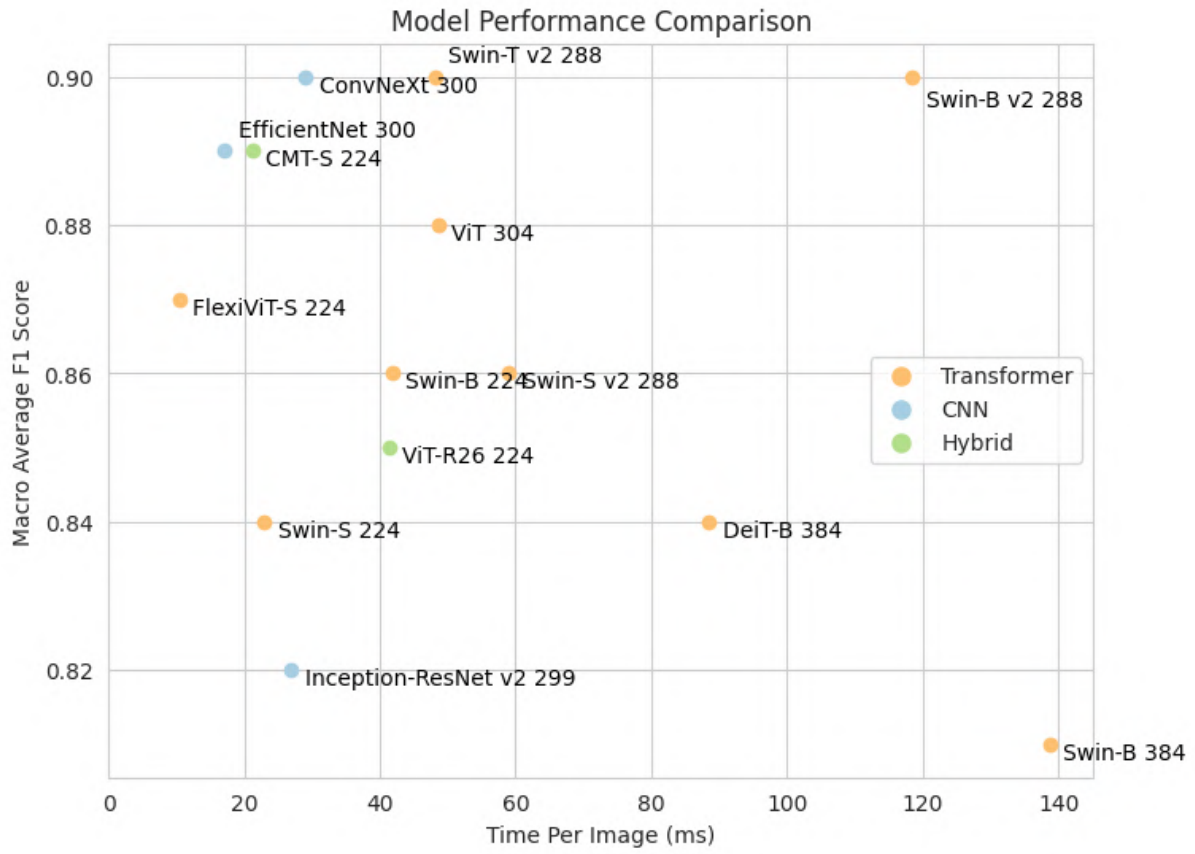


Figure 5.18: Model variants plotted by F1-Score and Inference time in milliseconds

Scaling with Model Size

To isolate the effects of increased model complexity on inference speed, multiple ViT and ConvNeXt variants were used to predict 224x224 images. For the vanilla Vision Transformer, the base and the large model were used with 86 million and 304 million parameters for the patch size 16 versions and an additional 2 million parameters for the patch size 32 variants. ConvNeXt variants include the 28 million parameters ConvNeXt-Tiny, 50 million parameter ConvNeXt-Small, 88 million for ConvNeXt-Base and ConvNeXt-Large with 198 million parameters.

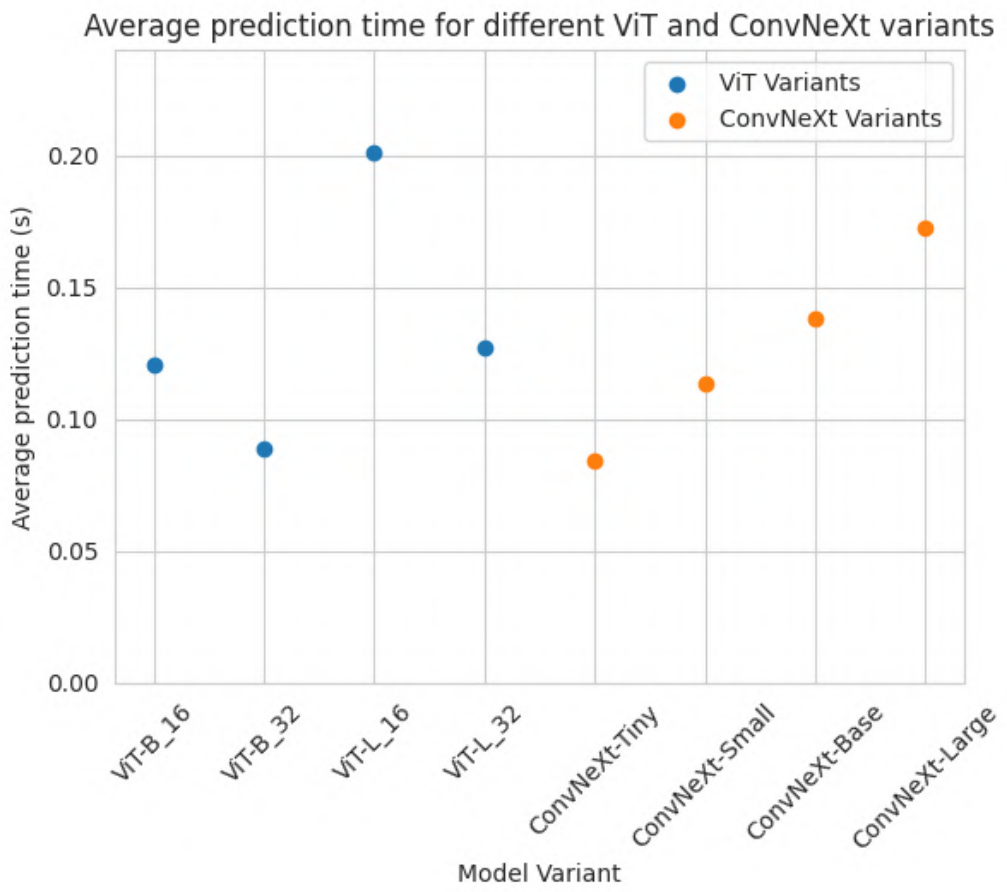


Figure 5.19: Inference Speed for different variants of ConvNeXt and ViT

Two conclusions can be drawn from this: first, the larger patch size leads to faster inference speeds, with the improvements being more dominant for the larger ViT-L/32 model. Second, both ViT and ConvNeXt do not scale quite linearly with increased model complexity. Interestingly, with higher parameter counts the inference speeds of both architectures seem to converge, with the ViT-L/16 model being only slightly slower than ConvNeXt-Large, despite having 1/3 more parameters. This would support the argument that Transformers are well suited for scaling to very deep architectures without an exponential hit to inference speeds.

Scaling with Input Resolution

Due to the quadratically scaling nature of the self-attention operation in Transformers, increasing the input resolution to this type of model is a concern. To find out how relevant this behaviour is for the present use case, the average inference time over 1000 predictions was taken for ViT-B/16, Swin-T/8 v2 and ConvNeXt Base for increasing resolutions between 128x128 and 2048x2048. Figure 5.20 plots the resulting inference speed at these resolutions for all models.

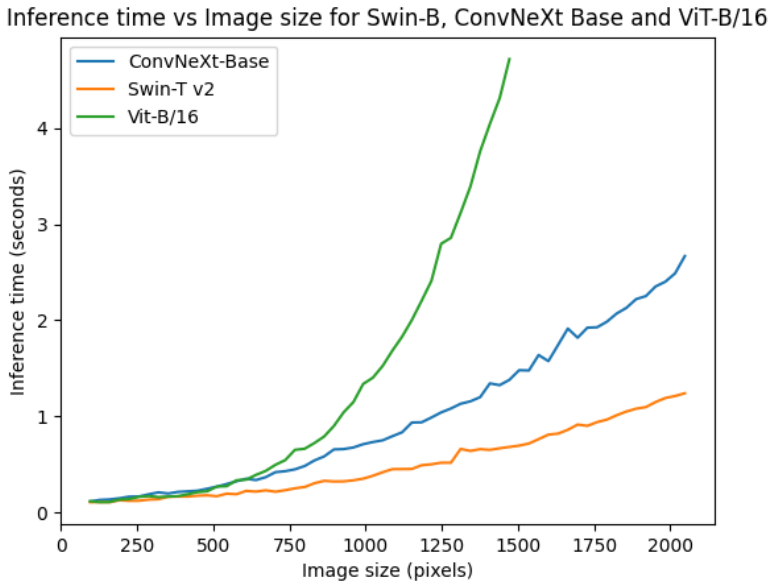


Figure 5.20: Inference Speed per Image for increasing input resolutions for Swin-B v2 and ConvNeXt Base

As expected, the vanilla Transformer exhibits considerably worse scaling characteristics whereas the CNN shows only minor signs of slowing down. Conversely, the shifted window approach from Swin proves to be effective, manifesting in much better-scaling characteristics than both ConvNeXt and ViT. Realistically, however, these results are not a major cause for concern for the model recognition dataset, as higher resolutions are not required for the models to perform well as shown in 5.1.1. Still, this behaviour should be kept in mind when scaling up image sizes in the future or on other tasks, especially when considering predicting images of higher resolution than the training images.

5.4 Training Efficiency

While previous sections have touched on the sample efficiency of different networks, this section seeks to study their compute efficiency during training. To recap the parameters for this evaluation: all finetuned models were capped at either 20 epochs, 24h of training or were stopped early on plateauing validation accuracy. As plots of all validation curves can get quite chaotic and hard to interpret, the most relevant results have been captured in table 5.6. Looking at the training time per epoch first, the patterns here mirror that of the inference speed results presented above. Generally, smaller networks compute faster and can use larger batch sizes, making the small FlexiViT model by far the fastest model to train on one full round of the dataset. As the use of mini-batches is also considered a form of generalisation, smaller batch sizes might provide less stable results. However, this concern did not materialise in the experiments conducted during this thesis. Still, going forward, techniques such as gradient accumulation could be ex-

	Validation Accuracy						
Model	**Epoch 1**	**Epoch 2**	**Epoch 3**	**...**	**final**	**Time per Epoch**	**Time to reach 80% Training Accuracy**
Swin-T/8 v2	0.837	0.897	0.916		**0.965**	0.75h	6,9h
Swin-B/8 v2 SAM	**0.929**	0.935	**0.943**		0.954	5h	2,2h
DeiT-B/16 SAM	0.92	**0.936**	0.937		0.937	2.60h	3,9h
ConvNeXt-B	0.876	0.915	0.933		0.944	0.86h	2,0h
EfficientNet-B3	0.837	0.900	0.903		0.943	0.62h	**0,6h**
CMT-Small	0.877	0.901	0.899		0.946	0.85h	0,8h
Inception ResNet v2	0.878	0.902	0.934		0.936	0.91h	1,4h
ViT-B/16	0.894	0.922	0.931		0.944	1.5h	1,6h
FlexiViT-B	0.521	0.735	0.793		0.930	**0.33h**	8,3h

Table 5.6: Comparison of Validation Accuracy after 1, 2 and 3 epochs, average time per epoch and time elapsed till training accuracy reached 80% accuracy.

plored to simulate larger batch sizes by accumulating gradients across multiple mini-batches before updating the weights. Nevertheless, smaller batches will still entail a larger overhead for copying data between the CPU and GPU.

Other interesting observations include the time it takes for a model to reach 80% training accuracy. This metric has been chosen as the validation accuracy is only logged once per epoch but most models would surpass 80% accuracy before that. Here, EfficientNet is by far the fastest model to achieve this milestone, after just over 30 minutes. Notably, all convolutional networks reach this point much earlier than any Transformer architecture, with hybrid models sitting in the middle. This could be attributed to the helpful inductive biases in convolutional networks, also helping the hybrid architecture to learn more quickly.

Another observation concerns the use of regularisation techniques like advanced data augmentation and sharpness-aware optimisation. As is evidenced by the reported times for Swin-T/8 v2 and DeiT-B/16 SAM, they delay the time to reach 80% accuracy by a significant margin. On the other hand, when looking at the highest accuracy achieved after only one epoch, these models, and the Transformer architectures in general, seem to perform better. This advantage holds for the first few epochs until eventually converging at similar peak accuracy values.

In summary, Transformer networks take longer to train as they need to learn to overcome their lack of inductive biases. At the same time, they are more sample-efficient, reaching higher accuracy after only a few epochs. Considering the full training run, however, this results in both architectures taking roughly the same time to converge. Generalisation techniques further exaggerate this finding, as they achieve better generalisation with fewer samples while also taking considerably longer to train.

5.5 Interpretability

As machine learning applications become more and more integrated and integral to day-to-day lives, end-users and regulators have an increasing interest in understanding the decision-making process of such models. Due to the stochastic nature of machine learning and especially deep learning, this poses an interesting challenge. In recent years researchers have proposed methods that aim to visualise what a computer vision model like a CNN sees. These insights can help with debugging models and trying to understand certain shortcomings.

While both model architectures allow the visualisation of low-level features, they are not inherently useful. Instead, this section will focus on calculating class attention maps for Swin-T/8 v2 and ConvNeXt Base. The example images chosen here are inspired by the misclassified images presented in section 5.2. For each example, the class activation map of the first and second predictions is plotted.

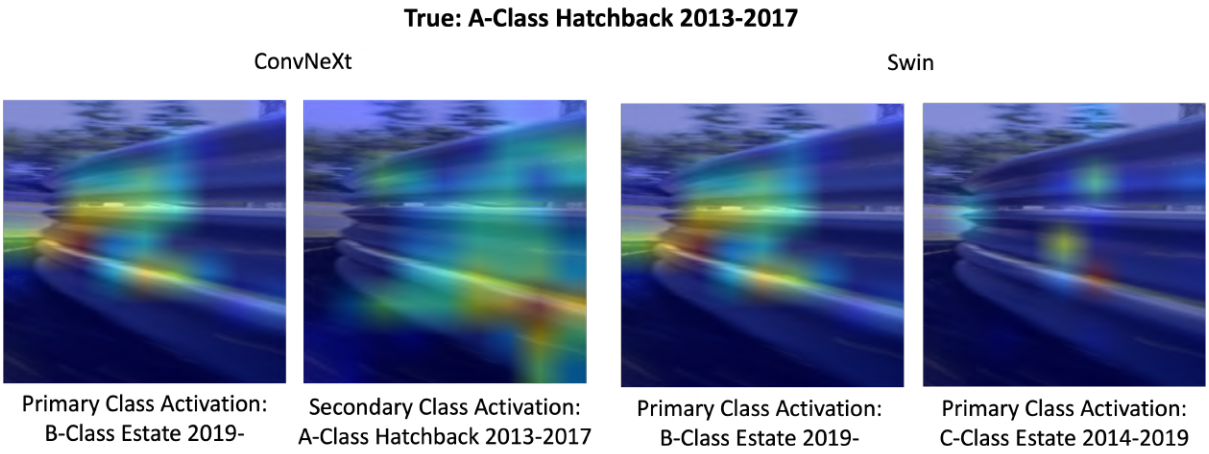


Figure 5.21: Class Activation maps for first and second highest predictions from Swin-T/8 v2 and ConvNeXt

The example in 5.21 shows how both networks struggle to find any identifiable information to classify a vehicle that does not exist. The guardrail seems to be the closest thing resembling a car for the model to latch onto. The second example shows a classical example of a slight confusion of similar-looking models. The attention map showcases how the model refers to different parts of the car. For ConvNeXt, the side profile seems to be of importance for the classification as an E-Class, whereas activations for the S-Class label tend to come from the rear lights. For Swin, seemingly no activations were triggered to push the model towards the correct classification.

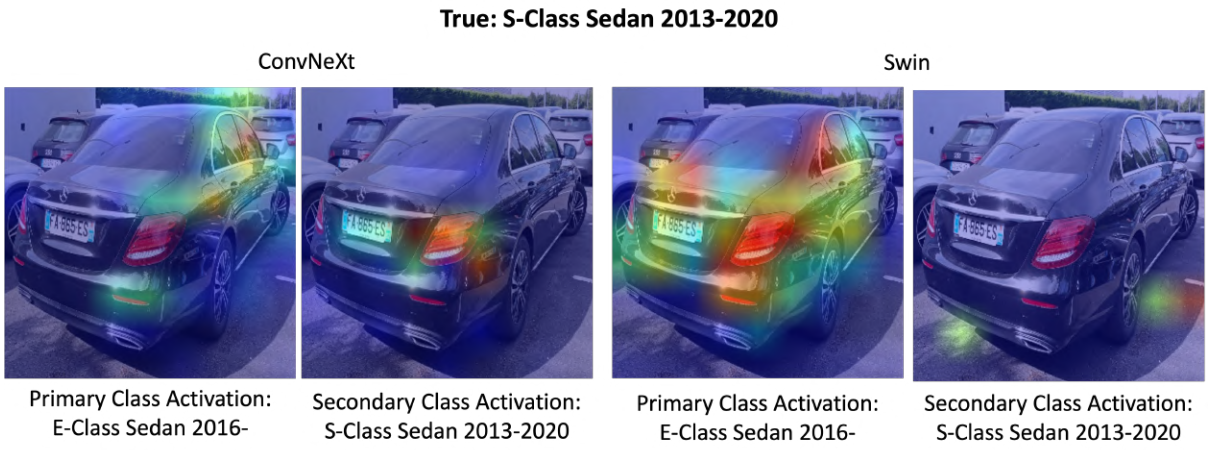


Figure 5.22: Class Activation maps for first and second highest predictions from Swin-T/8 v2 and ConvNeXt

Further examples show, how both models are able to pick up even small details like the roof railings that the model associates more with the E-Class. The transformer also appears to pay attention to the a-pillars as well as the front grill and bumper. The ConvNeXt model, however, seems to have been convinced by the form of the bonnet and a region around the headlight that the image shows a GLE SUV instead.

When looking at multiple class activation maps side by side, a subtle pattern seems to emerge: CNN activation maps appear to be slightly softer and more clustered. The transformer, meanwhile, often shows individual spots of higher intensity and are often further distributed across the frame. Some of this could be explained by the downsampling of features throughout the convolutional network whereas the self-attention blocks in the Transformer might form slightly sharper patterns. This observation is supported by similar findings by Dosovitskiy et al. regarding the effect of hybrid Vision Transformers and their use of convolutional feature maps on attention maps.

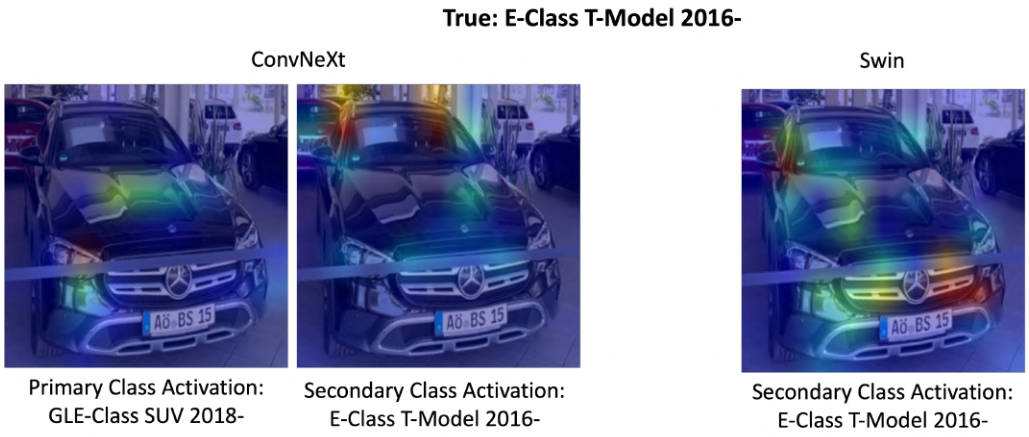


Figure 5.23: Class Activation maps for first and second highest predictions from Swin-T/8 v2 and ConvNeXt

Lastly, figure 5.24 shows a very challenging example. Not only do the models have to deal with the rotated image, but there is also very little information in the shadows as the contrast is very high. The Transformer seems to have been fooled by the a-pillar ad the wheel, predicting the similar-looking A-Class hatchback. Thankfully, example images like these are rare in reality but they test the model's ability to be resistant to translations in the input space. While inferring superior translation invariance by the CNN from this one example would be unwise, the question posed would warrant further experimentation to determine the robustness of either model against extreme image translations.

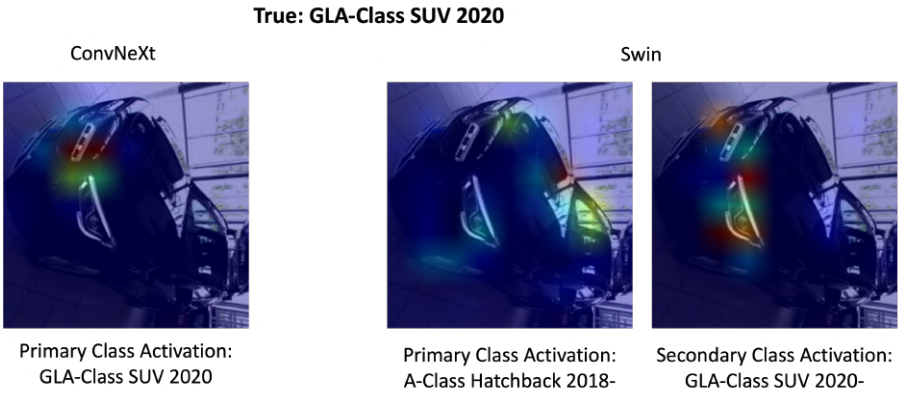


Figure 5.24: Class Activation maps for first and second highest predictions from Swin-T/8 v2 and ConvNeXt

6 Discussion

In this section, I discuss the implications of the findings from the previous chapter for the practical use case of model recognition. I also critically evaluate the results of my study in relation to the research questions and discuss the limitations of my research design and methodology, and the implications for future research in this field.

6.1 Vision Transformer versus Convolutional Neural Network

The main research interest of this thesis was to study if the Vision Transformer is a viable architecture for use on a challenging medium-sized dataset. This question specifically considered the lack of inductive biases in architecture and the lack of powerful computing resources as challenges compared to traditional architectures for image classification. Additionally, because the effects on inference speed are often not prominently featured when new state-of-the-art accuracies on the ImageNet benchmark are reported, this thesis aimed to capture the trade-offs that would have to be made to achieve competitive performance with CNNs.

As it turns out, Vision Transformers can indeed be trained effectively, competing with the best convolutional networks both in terms of accuracy and not falling far behind when it comes to inference speed. However, when evaluated in the context of this use case, Vision Transformers do not clearly come out as the superior architecture. Their tendency for slower training times and slower inference speeds and the limited size of this dataset does not allow this type of architecture to really shine. Moreover, adopting some of the lessons learned on Transformers back to convolutional networks has seemingly brought the two back closer together. Additionally, their quadratic attention scaling in architectures, not using the shifted window approach may still make them unsuitable for tasks requiring training on large image sizes. Results also show that the accuracy achieved in the ImageNet benchmark is not a perfect predictor for the performance in real-world datasets.

When placing both models on the spectrum between high variance and high bias, the Vision Transformer shows more rigid results, by being resistant to overfitting and slightly underfitting the training data, which is partially induced by the heavy augmentation strategy. On the other hand, the convolutional network can be placed more towards the other end of the spectrum, being at the risk of overfitting and thus being slightly more sensitive towards outliers and noise. Which of these behaviours is more desirable depends entirely on the dataset and specific business

problem at hand, with the model recognition seemingly lending itself to both approaches as the final evaluation shows them to be very similar. However, as will be discussed in later sections, this could also be a reflection of certain limitations of the model recognition dataset. In conclusion, the thesis does not provide enough evidence to confidently draw any conclusions concerning the clear superiority of either architecture on this real-world, medium-sized dataset.

Another advantage of the Transformer architecture is described as its improved capability for modelling long-range dependencies. Again, this proved challenging to test, as convolutional networks had seemingly just as easy a time achieving high accuracies. However, one could argue that the model recognition dataset does not necessarily pose an unusually hard challenge in this regard. Given that a lot of the relevant information for classifying vehicles is typically located physically close together, the model may not benefit measurably from more efficiently captured long-range information. Overall, the dataset highlights that both models are able to handle noisy data and using a combination of techniques to improve regularisation, they perform very similarly, possibly indicating a soft ceiling for performance on this dataset.

Similarly, this thesis fails to capture another prominently mentioned characteristic that has led to the hype surrounding the Vision Transformer. Namely, the favourable scaling characteristics reported by Dosovitskiy et al. do not manifest themselves on this medium-sized dataset. Therefore the Vision Transformer is likely not yet able to showcase its full potential in form of its more generic computational design.

A critical remark must be made regarding the process of hyperparameter fine-tuning. As computational resources are limited, the search space for hyperparameters and other factors such as batch size, optimiser and data augmentations had to be limited to allow enough time for multiple experiments. As a result, some performance may have been left on the table, with the potential to slightly shake up the exact order of performance between models, as the results are so closely matched.

In conclusion, the results achieved here are competitive and the observed discrepancies are often so slim as to not make any practical difference. Given that convolutional networks have been around for over a decade and have benefited from multiple incremental improvements it is still remarkable that a new architecture is able to match, and under the right circumstances, even beat them almost from the get-go. As the literature review has also shown, both architectures still have plenty to learn from each other, as demonstrated particularly clearly by ConvNeXt and Swin. Given this background, it is fair to assume that with more iterative improvements to come, the Transformers will eventually be able to fully surpass ConvNets for most applications, given its more generic computing capabilities.

6.2 Hybrid Networks

As the results presented in the previous chapter have shown, both examined hybrid models do not stand out between the pure convolutional and Transformer-based models. Going into this comparison, one could have hoped for only the combination of strengths from both architectures without any of the drawbacks. And while the models have demonstrated slightly faster inference speed and faster training adaption than vanilla Transformers, they also do not manage to outperform either of them. This matches with the initial research on hybrid models by Dosovitskiy et al., who observe slight advantages on small computational budgets that vanish quickly with more training time [13].

Given the abundant research on Vision Transformers, hybrid architectures seem to have taken a back seat. with subjectively fewer architectures being released that claim state-of-the-art performance. While this thesis cannot present any final conclusions on this, there are two areas of potential concern: as proven by the results of this thesis, modern Vision Transformers seem to be able to learn any inductive biases given enough time and training data, at which point they no longer benefit from any artificially induced inductive biases. On the contrary, one could hypothesise that these convolutional "leftovers" might actually start to become a bottleneck for the network, limiting its scaling capabilities over pure Transformer variants.

Regardless, the dataset for this use case does not require scaling of this level and as such may not show the benefits of different architectures as clearly. Within the scope of this study then, hybrid networks did not manage to consistently outperform either CNNs or ViTs, with the results fitting somewhere in the middle. While this made them a less interesting subject to study in this context, this behaviour might fit other tasks better than the alternatives.

6.3 Speed and Accuracy Trade-off

One of the challenges of model recognition is to balance the speed and accuracy of the inference process. In this section, the performance of the approach in the practical use case is analysed with respect to these two factors.

While previous sections may have made the impression of naively classifying the best performing Transformer as the higher accuracy model while the CNN convinced on inference speed, they should not be seen as polar opposites. However, when summarising the results across all examined models this might hold some truth, as CNNs generally still outperform Transformers at inference speed.

With that being said, one also needs to consider that the raw inference time makes up only a small portion of the request handling time. Given the relatively low latency of less than 100ms and the asynchronous nature of the use case, even a 3x increase in inference time would be

tolerable. Nevertheless, it should also be noted that the hosting environment does differ from the virtual machine that was used for training (NVIDIA T4), meaning the specific measurements provided here will not directly transfer over. More testing should be done to better understand the differences in performance between the two environments.

This still leaves the question if the loss in inference performance is worth the small gain in accuracy. To answer this question, one must first define how qualitative performance should actually be measured for the given use case. Specifically, the choice is influenced by two factors:

First: as shown in chapter 4, the dataset for model recognition is fairly imbalanced. As such, the use of a balanced metric for evaluation, like F1-Score or MCC, would be desirable. That way one can make sure that minority classes are also captured by the model. While this is a worthy goal, one should also consider that there might be a strong correlation between the number of training samples available and the frequency of this class in the actual population. This assumption obviously does not hold for newly released vehicles where the markets have not yet provided enough training data and also does not account for any inconsistencies in the collection of new training images. However, overall the premise is valid that high-frequency classes should be weighted higher in the overall performance.

Second: the implementation details of the model recognition endpoint result in an increased "fuzziness" added to any prediction, as the body style is disregarded and only the parent model is considered for classification. As the findings show, both models are very confident in this regard, with most mistakes being made at incorrectly predicted body styles, often on images that do hardly contain this information.

Given this background and the negligible differences in accuracy between the Transformer and the CNN, provided two models perform identically in terms of accuracy, it would be more desirable to pick the network with higher inference performance. However, as the current uses cases do not require real-time performance and requests can be queued without huge repercussions, one might want to trade even small gains in accuracy against slightly slower inference compared to choosing the faster model, which in turn would allow for more headroom in the current implementation and potentially better scaling to more requests in the future.

Another factor to consider is that a single GPU on the hosting server will not only serve this endpoint but also others. Therefore, the compute resources are shared and limited. From a business standpoint, this endpoint also does not contribute to the operating income of the company. Costs should therefore be kept as low as possible, making any additional compute hard to justify. At the point where the load on the system becomes a real concern, other optimisations could also be employed to boost the performance, for example by batching multiple requests together.

Given the very close performance of both models, it is unlikely that either choice translates to any noticeable differences in practice. The same can however not be said about comparing them

to the original baseline model. While also here the improvements in accuracy alone may not be substantial, the uplift in the F1-Score and MCC show, that generalisation is much improved and both Swin-T/8 v2 and ConvNeXt present noticeable improvement over the baseline Inception ResNet v2 model.

6.4 Network Complexity and Hardware Limitations

Given the relatively modest hardware, training was limited to models with fewer than 100 million parameters and batch sizes smaller than 64. Some of these challenges can be overcome by using techniques such as gradient accumulation, where gradient updates are averaged over multiple mini-batches before updating the model. However, this generally limits the scope of how models can be trained and how much hyperparameter tuning can be performed. This widening gap can be observed across academia with networks requiring more and more computing resources to effectively train, putting them out of reach for individual researchers and small companies.

As such, it is also hard to benefit from proclaimed benefits of the Transformer architecture in regard to scaling on large datasets and models. However, results seem to indicate that for the task of model recognition, small to medium-sized networks are sufficient. As mentioned previously, the GPU memory on the hosting system is shared across multiple models. It would therefore be desirable to keep the model's GPU memory requirements as low as possible. Given that the Swin-T/8 v2 network is about 1/3 the size of ConvNeXt Base (28,8 million versus 88 million parameters), this would potentially leave more room for other models running on the same machine. However, because using the model for inference only does not require additional gradients and optimiser steps to be stored, the memory requirements are reduced compared to training.

For obvious reasons, the aspect of sustainability is also becoming an increasing consideration. As the training time accounts for less than 1% of the total up-time per year, assuming a re-training every quarter for 24h and a 24/7 deployment of the production system, efficiency gains should focus on the inference part of the pipeline. Smaller and more efficient models can help save energy, however. However, the actual impact is hard to quantify. As the GPU will still consume power at idle, the goal should be to choose the smallest possible hardware configuration to sufficiently handle all requests. Scaling up or down the GPU based on the current load would be estimated to be the most effective way of conserving power. Meanwhile for training, one could experiment with using a higher-powered GPU to speed up training, using more power over less time. As distributed training tends to scale very well and pretty much linearly, this Looking at the deployed models specifically, the difference is very likely to be too small to make any conceivable difference in regard to energy consumption. To iterate on the idea of saving computing time, some experimentation could be conducted on the feasibility of only finetuning certain layers of very deep networks, thus lowering training times drastically.

6.5 Interpretability

The spectrum of techniques to visualise the learned representations and features of both convolutional networks and Transformers is large and only the most basic ones have been explored in this thesis. Improving interpretability remains an important area of research for all kinds of machine learning models. However, when looking specifically at computer vision, the usefulness of these techniques for deployed models remains limited.

Class attention maps, as used in 5.5, can be a useful tool in detecting unwanted behaviour of a model or debugging the technical implementation. They can also be an interesting tool for comparison between different models or between model checkpoints. Analysing how they change might also help with improving certain aspects of the architecture. However, even with some forms of visualisations offering quite an intuitive interpretation, they are often not inherently useful as their interpretation can easily be misleading while it can also not be used to directly contribute towards model performance. In this regard, Transformers have unfortunately also not shown to be significantly more interpretable than convolutional networks.

The application of attention maps in communication with less technical stakeholders is also limited to the point where the question arises of *why* the network is or is not paying attention to certain parts of the image. Expectation management should not be overlooked here as these images might imply a level of control over the way the network learns which is unrealistic.

While this may sound very pessimistic, it is not at all meant to undermine the importance of this topic. On the contrary, this should be the motivation to find additional approaches that not only help researchers but also provide an intuitive and consistent way of communicating the inner workings of neural networks to all parties involved, be it developers, business departments or end users.

6.6 Dataset Considerations

The decision to include classes with very few training samples is motivated by the low cost of false negative predictions. Consider the use case of checking images for fraud, one might prefer to check images manually more often as opposed to incorrectly classifying a non-match as a match and thus not raising any flags. Therefore, high precision for a class is generally more useful than higher recall.

One issue that can arise with classes that bring only a few samples, is the lack of images for the validation and test set. Stratified sampling is used to create them but given the training split of 85%, this might leave as little as 4 images per set for evaluation. This can result in unstable results, as the misclassification of even a single image can lead to large changes in the performance metrics. This especially extends to metrics that do not take the number of samples

per class into account like the macro average precision and recall. Therefore they are subject to higher fluctuation during training, possibly also accounting for some of the differences measured on the evaluated models. Finally, the correlation between the number of training samples per class and the frequency of that class in the real world could be studied to determine the effectiveness of accuracy as the primary metric. Another issue concerns the noise exhibited by images in the training data, which due to their perspective or other corruption, do not allow the model to accurately predict the specific parent model and body style combination. Further investigation into the training data might be needed to establish a better understanding of the overall data quality. One idea to combat both of the aforementioned issues is to split the prediction into two parts. One classification head would predict the parent vehicle class while a second head would only be responsible for classifying the body style. This setup would allow tracking metrics for both classifications independently and potentially evaluate them more reliably. Potential downsides include increased training complexity and the risk of slower convergence when using two loss functions.

One critical aspect also had to suffer slightly as a result of the computational bottleneck: the train-test split. As the statistical confidence in a given test set to estimate the error of a population with a maximum error of x decreases with the number of models that are tested, another approach would be recommended. Namely, using k-fold cross-validation presents a way to more reliably estimate the performance across the whole dataset. However, because a single training run can take from 6 up to 24 hours, repeating this for k folds, even without accounting for extra runs for hyperparameter tuning, is impractical with the given compute budget. Instead, it was important to maintain proper "dataset hygiene" in order to not leak information from the test set into the training and validation set and thereby statistically invalidate any results. For future runs, it might be worth transitioning to adopting this for regular training runs that are not as time-sensitive to obtain more reliable evaluation metrics and potentially a model with better generalisation performance.

6.7 Further Work

This section will present proposals for further work in both the academic domain of image classification as well as for practical steps that can improve the performance and efficiency of the model recognition endpoint.

One direction for further work is to dive deeper into the effects of different data augmentation strategies for Vision Transformers. MixUp and RandAugment have proved very effective and even irreplaceable to achieve competitive performance, with additional techniques like CutMix or a fine-tuned custom augmentation scheme offering potentially even greater benefits [63]. This would also entail studying how much generalisation can be added to the model before gradient updates get too small. Furthermore, some methods may even introduce noise or occlusion that can interfere with the attention patterns learned by the model. Therefore, it would be interesting

to investigate which data augmentation methods are most suitable for Vision Transformers and how they can be combined or adapted to enhance overall performance.

A second direction for further work regarding the model recognition training pipeline is to evaluate the trade-offs between using small GPUs with long training times versus using large GPUs with shorter training times. Vision transformers are typically trained on large-scale datasets such as ImageNet-22k or JFT-300M, which can take days or weeks to complete on a single GPU. However, using multiple GPUs or cloud services can speed up the training process significantly, but at a higher monetary cost. Therefore, it would be useful to analyse the cost-benefit ratio of different training strategies for vision transformers and how they affect the final performance and quality of the models. On a related note, techniques such as pruning or quantisation can be surveyed which can help to reduce the computational cost and memory footprint of Vision Transformers. Pruning and quantisation are techniques that aim to reduce the number of parameters used by a model without sacrificing much accuracy. This might help further in bridging the gap in inference speed between Transformers and CNNs.

A third direction for further work is to evaluate the feasibility and effectiveness of reusing old weights as starting points for new training runs when adding new classes to the Vision Transformer model. Intuitively, one would hope that some of the learned features may still be relevant and useful. Therefore, it may be possible to reuse some of the old weights as initialisation. Moreover, the practice of freezing large parts of the network and only finetuning a few layers can be explored. This could save time and resources while preserving or improving the accuracy of the model.

A fourth direction for further work is to explore the potential of language supervision in vision transformers, as proposed by CLIP. CLIP is a vision transformer model that is trained on a large-scale dataset of image-text pairs, where the objective is to learn a joint embedding space that aligns images and texts semantically. This allows the model to perform zero-shot image classification on any natural language label without fine-tuning. The advantage of this approach is that it enables vision transformers to learn not only simple features corresponding to physical objects but also abstract concepts that are expressed in natural language. When fine-tuned on a target dataset, this could lead to more robust and generalisable models. This approach would potentially open up the domain of image classification to benefit from large pre-trained natural language datasets by inheriting useful features from text. [45]

A final direction for further work is to explore another emerging architecture, the MLP mixer. Foregoing any attention and convolutional layers, it instead relies entirely on multi-layer perceptrons. Tolstikhin et al. achieve competitive ImageNet performance when using large datasets or strong regularisation techniques. These characteristics bear a striking resemblance to early Vision Transformers and a closer analysis could reveal which model is more promising going forward. [59]

7 Conclusion and Outlook

This thesis set out to explore the performance and limitations of Vision Transformers in comparison to convolutional neural networks on a medium-sized dataset for vehicle model classification. First, a thorough literature review was conducted to establish a basic understanding of both architectures and their respective strengths and weaknesses. This revealed that Vision Transformers are able to match and even outperform convolutional networks given enough training data and compute in terms of their accuracy/speed trade-off. Here, the self-attention mechanism is crucial in allowing Transformers to attend to different parts of the image globally and thereby model longer-range dependencies better than convolutional networks while allowing for efficient computation and parallelization on modern hardware. This gives Transformers the unique advantage to scale to extremely large datasets and model sizes.

However, the absence of convolutions also means that ViTs lack the natural inductive biases to prioritize local context over global features compared to the induced locality in CNNs. This limitation made them challenging to use on small to medium-sized datasets and when only limited computing resources are available, as the network first needs time to learn these patterns. Moreover, the self-attention mechanism scales quadratically with the input resolution, thus limiting their use for applications requiring high-resolution inputs.

Given these initial limitations, chapter 3 assembled a selection of improved architectures that aim to enhance performance in aspects like training efficiency, inference speed and prediction performance. For Transformers, these networks specifically target the need for largescale pretraining (DeiT), quadratically scaling complexity (Swin) and reliance on a fixed patch size with implications on transferability between models (FlexiViT).

On the convolutional side of things, the Inception family of models was introduced, as it provides the baseline model that is currently used in production. Interestingly, it was discovered that both Transformers and CNNs take strong inspiration from each other, enabling ever-increasing performance on benchmarks like ImageNet. Additionally, a third category of models was introduced which combines the building blocks of both architectures to form a more robust model, harnessing the power of both Transformers and ConvNets.

Having reviewed these network variants, extensive experiments were conducted to establish the best-performing architecture for the model recognition use case. The results show good performance across all models, with Swin-T/8 v2 and ConvNeXt-Base taking the top spots with 0,963 and 0,957 test accuracy. Compared to the baseline Inception ResNet v2 model, this

is an increase of 1,6% and 0,9% respectively. However, since accuracy foregoes insights on the performance of minority classes, the F1-score shows a more drastic improvement of over 8,6% for Swin-T/8 v2. This indicates much better generalization abilities compared to the previous baseline. Overall, the experiments show very close performance between the best CNN and best Vision Transformer model.

The evaluations regarding inference speed show a slightly different picture, with CNNs generally outperforming Vision Transformers. However, as networks like FlexiViT demonstrate, ViTs allow for a variety of ways to trade off inference speed for improved accuracy, with the models tested focusing primarily on the latter. When comparing Swin and ConvNeXt again, the differences here are not dramatic but show the convolutional architecture ahead with 35 images per second versus 18 images per second. Other observations made included the Transformers showing slightly slower training time while being more sample efficient compared to ConvNets and the importance of data augmentation schemes for ViT training.

A close examination of misclassified images of both models revealed that most false predictions are actually related to poor input images that prohibit any judgment about the body style of the vehicle through obstruction or occlusion. In chapter 6, the concern was discussed to what extent this issue of noisy images influences the final accuracies of both models. In the end, it was concluded that the added fuzziness in the implementation layer of the API endpoint and the strong top-3 accuracy performance of both models likely resulted in higher-than-expected real-world performance. Compared to the baseline Inception ResNet v2 model, both architecture types provide more modern and superior alternatives that also allow for a larger range of adjustments regarding the required speed and accuracy trade-off.

Nevertheless, this thesis shows that Vision Transformers can be successfully applied to medium-sized real-world datasets, easily surpassing the legacy baseline model in almost all measurable areas. The new model shows much-improved generalization performance, as it is able to more reliably predict underrepresented classes with less error. It also uses fewer parameters, making it more compact and elegant.

Given the scale of the dataset for this use case and the limited computing resources available, this thesis was not able to demonstrate all of the proclaimed benefits of the Transformer architecture. Disappointingly, they, therefore, remain mostly of theoretical nature, being hard to capture on this real-world dataset. Furthermore, the model recognition dataset has not proven to be as big of a challenge as originally believed, as most models are easily able to achieve high accuracies, with there likely being a soft ceiling for any metric given the noisy nature of the dataset. Therefore, both architectures are not pushed to their limits, negating the most prominent advantages of the Transformer architecture. The final conclusion, given the results demonstrated in this thesis, still ranks the convolutional architecture as the overall slightly better choice for this use case. This is primarily explained by the faster image throughput and training times while achieving virtually the same predictive performance.

Generally, it was discovered that this domain is an extremely fast-paced area of development with regular innovation in both architectures. This has been showcased particularly well by the two best-performing models, with Swin taking inspiration from the hierarchical structure in CNNs and ConvNeXt implementing and harnessing a lot of the training schemes that have proven successful for Vision Transformers. This arms race is also documented by regularly changing rankings on public image classification benchmarks like ImageNet. While this pace in research is commendable, it also proved very challenging for this thesis to keep up with newly released papers and prioritise the most influential ideas and modifications for each architecture.

Key takeaways of this thesis for practitioners can be summarised as follows: state-of-the-art Transformer models can be successfully trained on medium-sized datasets even without strong data augmentations. They generally offer a slightly higher performance ceiling compared to convolutional networks if one is willing to trade in higher compute requirements and slightly slower inference speeds. Furthermore, this thesis has provided scientific insights that show Transformers to be robust against heavy and unrefined data augmentation strategies. Results also indicate that hybrid models do not benefit from the added inductive biases for long enough to provide a significant benefit over pure transformer-based architectures, even on only a medium-sized dataset.

When looking at the future potential of both architectures, it is hard to neglect the fact that convolutional networks have been around for over a decade, whereas vision transformers were able to match their performance almost from the get-go. Within less than two years, multiple new variants have been proposed, only a select few of which were covered in this thesis, that successfully address some of the most pressing issues, like quadratic scaling with the input size (Swin). Despite the intuition that hybrid models would be able to further help with that by combining the strengths of both architectures, this theory could not be substantiated by the experiments of this thesis. Looking at the trends in literature, this direction also seems to receive less attention.

Given the current rate of progress in terms of research and industry interest but also in data and compute availability, the trend of training ever-growing models on even faster-growing datasets is only likely to accelerate, an area where the Transformer has shown to be clearly superior to any other architecture. Thus this thesis concludes with the outlook that given the already competitive performance of Transformers today, the interest in them will continue and likely ultimately lead to the Transformer taking over as the go-to architecture not only for NLP but also for computer vision.

List of References

Literature References

[1] Mercedes-Benz Mobility AG. "Unpublished internal company document". 2023.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. arXiv:1607.06450. type: article. arXiv, July 2016. DOI: 10.48550/arXiv.1607.06450. arXiv: 1607.06450[cs,stat]. URL: http://arxiv.org/abs/1607.06450 (visited on 03/23/2023).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv:1409.0473. type: article. arXiv, May 2016. DOI: 10.48550/arXiv.1409.0473. arXiv: 1409.0473[cs,stat]. URL: http://arxiv.org/abs/1409.0473 (visited on 03/14/2023).

[5] Lucas Beyer et al. *FlexiViT: One Model for All Patch Sizes*. arXiv:2212.08013. type: article. arXiv, Dec. 2022. DOI: 10.48550/arXiv.2212.08013. arXiv: 2212.08013[cs]. URL: http://arxiv.org/abs/2212.08013 (visited on 03/21/2023).

[6] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. *When Vision Transformers Outperform ResNets without Pre-training or Strong Data Augmentations*. arXiv:2106.01548. type: article. arXiv, Mar. 2022. arXiv: 2106.01548[cs]. URL: http://arxiv.org/abs/2106.01548 (visited on 02/02/2023).

[7] François Chollet. *Deep Learning with Python, Second Edition*. Manning Publications Co. LLC, 2021, p. 400. ISBN: 9781617296864.

[8] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. *On the Relationship between Self-Attention and Convolutional Layers*. arXiv:1911.03584. type: article. arXiv, Jan. 2020. DOI: 10.48550/arXiv.1911.03584. arXiv: 1911.03584[cs,stat]. URL: http://arxiv.org/abs/1911.03584 (visited on 03/14/2023).

[9] Ekin D. Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. arXiv:1805.09501. type: article. arXiv, Apr. 2019. DOI: 10.48550/arXiv.1805.09501. arXiv: 1805.09501[cs,stat]. URL: http://arxiv.org/abs/1805.09501 (visited on 03/21/2023).

[10] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. arXiv:1909.13719. type: article. arXiv, Nov. 2019. DOI: 10.48550/arXiv.1909.13719. arXiv: 1909.13719[cs]. URL: http://arxiv.org/abs/1909.13719 (visited on 03/07/2023).

[12] Jia Deng et al. *ImageNet: a Large-Scale Hierarchical Image Database*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[13] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv:2010.11929. type: article. arXiv, June 2021. arXiv: 2010.11929[cs]. URL: http://arxiv.org/abs/2010.11929 (visited on 02/16/2023).

[14] Pierre Foret et al. *Sharpness-Aware Minimization for Efficiently Improving Generalization*. arXiv:2010.01412. type: article. arXiv, Apr. 2021. DOI: 10.48550/arXiv.2010.01412. arXiv: 2010.01412[cs,stat]. URL: http://arxiv.org/abs/2010.01412 (visited on 03/06/2023).

[15] Robert Geirhos et al. *Generalisation in humans and deep neural networks*. arXiv:1808.08750. type: article. arXiv, Oct. 2020. DOI: 10.48550/arXiv.1808.08750. arXiv: 1808.08750[cs,q-bio,stat]. URL: http://arxiv.org/abs/1808.08750 (visited on 03/14/2023).

[16] Amin Ghiasi et al. *What do Vision Transformers Learn? A Visual Exploration*. arXiv:2212.06727. type: article. arXiv, Dec. 2022. DOI: 10.48550/arXiv.2212.06727. arXiv: 2212.06727[cs]. URL: http://arxiv.org/abs/2212.06727 (visited on 12/14/2022).

[17] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, Mar. 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html (visited on 03/23/2023).

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. arXiv:2008.05756. type: article. arXiv, Aug. 2020. DOI: 10.48550/arXiv.2008.05756. arXiv: 2008.05756[cs,stat]. URL: http://arxiv.org/abs/2008.05756 (visited on 03/19/2023).

[20] Jianyuan Guo et al. *CMT: Convolutional Neural Networks Meet Vision Transformers*. arXiv:2107.06263. type: article. arXiv, June 2022. arXiv: 2107.06263[cs]. URL: http://arxiv.org/abs/2107.06263 (visited on 03/20/2023).

[21] Elnaz Jahani Heravi Hamed Habibi Aghdam. *Guide to Convolutional Neural Networks*. Springer-Verlag GmbH, May 2017. 282 pp. ISBN: 9783319575506. URL: https://www.ebook.de/de/product/29154077/hamed_habibi_aghdam_elnaz_jahani_heravi_guide_to_convolutional_neural_networks.html.

[22] Kai Han et al. "A Survey on Vision Transformer". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (Jan. 2023), pp. 87–110. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2022.3152247. arXiv: 2012.12556[cs]. URL: http://arxiv.org/abs/2012.12556 (visited on 03/24/2023).

[23] Kaiming He et al. *Deep Residual Learning for Image Recognition*. arXiv:1512.03385. type: article. arXiv, Dec. 2015. DOI: 10.48550/arXiv.1512.03385. arXiv: 1512.03385[cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 03/24/2023).

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531. type: article. arXiv, Mar. 2015. DOI: 10.48550/arXiv.1503.02531. arXiv: 1503.02531[cs,stat]. URL: http://arxiv.org/abs/1503.02531 (visited on 04/01/2023).

[25] Daniel Ho et al. "Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules". In: International Conference on Machine Learning. PMLR, May 2019, pp. 2731–2741. URL: https://proceedings.mlr.press/v97/ho19b.html (visited on 03/21/2023).

[26] Jie Hu et al. *Squeeze-and-Excitation Networks*. arXiv:1709.01507. type: article. arXiv, May 2019. DOI: 10.48550/arXiv.1709.01507. arXiv: 1709.01507[cs]. URL: http://arxiv.org/abs/1709.01507 (visited on 03/31/2023).

[27] Gao Huang et al. *Deep Networks with Stochastic Depth*. arXiv:1603.09382. type: article. arXiv, July 2016. DOI: 10.48550/arXiv.1603.09382. arXiv: 1603.09382[cs]. URL: http://arxiv.org/abs/1603.09382 (visited on 03/20/2023).

[28] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167. type: article. arXiv, Mar. 2015. DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167[cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 03/23/2023).

[29] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980. type: article. arXiv, Jan. 2017. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980[cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 03/23/2023).

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[31] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[32] Zewen Li et al. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. arXiv:2004.02806. type: article. arXiv, Apr. 2020. arXiv: 2004.02806[cs,eess]. URL: http://arxiv.org/abs/2004.02806 (visited on 02/28/2023).

[33] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. arXiv:1312.4400. type: article. arXiv, Mar. 2014. DOI: 10.48550/arXiv.1312.4400. arXiv: 1312.4400[cs]. URL: http://arxiv.org/abs/1312.4400 (visited on 03/23/2023).

[34] Ze Liu et al. *Swin Transformer V2: Scaling Up Capacity and Resolution*. arXiv:2111.09883. type: article. arXiv, Apr. 2022. DOI: 10.48550/arXiv.2111.09883. arXiv: 2111.09883[cs]. URL: http://arxiv.org/abs/2111.09883 (visited on 04/01/2023).

[35] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. arXiv:2103.14030. type: article. arXiv, Aug. 2021. DOI: 10.48550/arXiv.2103.14030. arXiv: 2103.14030[cs]. URL: http://arxiv.org/abs/2103.14030 (visited on 03/13/2023).

[36] Zhuang Liu et al. *A ConvNet for the 2020s*. arXiv:2201.03545. type: article. arXiv, Mar. 2022. arXiv: 2201.03545[cs]. URL: http://arxiv.org/abs/2201.03545 (visited on 03/21/2023).

[37] Umberto Michelucci. *Advanced Applied Deep Learning*. Apress, 2019. DOI: 10.1007/978-1-4842-4976-5.

[38] Paulius Micikevicius et al. *Mixed Precision Training*. 2017. DOI: 10.48550/ARXIV.1710.03740. URL: https://arxiv.org/abs/1710.03740.

[39] Fausto Morales. *vit-keras*. original-date: 2020-11-07T23:20:16Z. Mar. 2023. URL: https://github.com/faustomorales/vit-keras (visited on 03/27/2023).

[40] Vinod Nair and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, June 2010, pp. 807–814. ISBN: 9781605589077. (Visited on 03/23/2023).

[42] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. arXiv:1511.08458. type: article. arXiv, Dec. 2015. DOI: 10.48550/arXiv.1511.08458. arXiv: 1511.08458[cs]. URL: http://arxiv.org/abs/1511.08458 (visited on 11/22/2022).

[43] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), pp. 1345–1359. ISSN: 1558-2191. DOI: 10.1109/TKDE.2009.191.

[44] Sayak Paul and Pin-Yu Chen. *Vision Transformers are Robust Learners*. arXiv:2105.07581. type: article. arXiv, Dec. 2021. DOI: 10.48550/arXiv.2105.07581. arXiv: 2105.07581[cs]. URL: http://arxiv.org/abs/2105.07581 (visited on 03/14/2023).

[45] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. arXiv:2103.00020. type: article. arXiv, Feb. 2021. DOI: 10.48550/arXiv.2103.00020. arXiv: 2103.00020[cs]. URL: http://arxiv.org/abs/2103.00020 (visited on 03/30/2023).

[46] Prajit Ramachandran et al. *Stand-Alone Self-Attention in Vision Models*. arXiv:1906.05909. type: article. arXiv, June 2019. DOI: 10.48550/arXiv.1906.05909. arXiv: 1906.05909[cs]. URL: http://arxiv.org/abs/1906.05909 (visited on 11/11/2022).

[47] Paulo Rauber. "Visual Analytics Applied to Image Analysis: From Segmentation to Classification". Thesis fully internal (DIV). [Groningen], 2017. ISBN: 9789036792882.

[48] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. arXiv:1801.04381. type: article. arXiv, Mar. 2019. DOI: 10.48550/arXiv.1801.04381. arXiv: 1801.04381[cs]. URL: http://arxiv.org/abs/1801.04381 (visited on 03/31/2023).

[49] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". In: *Artificial Neural Networks – ICANN 2010*. Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 92–101. ISBN: 9783642158254. DOI: 10.1007/978-3-642-15825-4_10.

[50] Mohit Sewak, Md. Rezaul Karim, and Pradeep Pujari. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Ed. by Sunith Shetty. 1st ed. Published by Packt Publishing Ltd.Livery Place35 Livery StreetBirminghamB3 2PB, UK.: Packt Publishing - ebooks Account, 2018. ISBN: 9781788392303.

[51] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556. type: article. arXiv, Apr. 2015. DOI: 10.48550/arXiv.1409.1556. arXiv: 1409.1556[cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 03/23/2023).

[52] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 03/23/2023).

[53] Andreas Steiner et al. *How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers*. arXiv:2106.10270. type: article. arXiv, June 2022. arXiv: 2106.10270[cs]. URL: http://arxiv.org/abs/2106.10270 (visited on 02/02/2023).

[54] Christian Szegedy et al. *Going Deeper with Convolutions*. arXiv:1409.4842. type: article. arXiv, Sept. 2014. DOI: 10.48550/arXiv.1409.4842. arXiv: 1409.4842[cs]. URL: http://arxiv.org/abs/1409.4842 (visited on 11/18/2022).

[55] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. arXiv:1602.07261. type: article. arXiv, Aug. 2016. arXiv: 1602.07261[cs]. URL: http://arxiv.org/abs/1602.07261 (visited on 11/30/2022).

[56] Chuanqi Tan et al. *A Survey on Deep Transfer Learning*. arXiv:1808.01974. type: article. arXiv, Aug. 2018. DOI: 10.48550/arXiv.1808.01974. arXiv: 1808.01974[cs,stat]. URL: http://arxiv.org/abs/1808.01974 (visited on 03/25/2023).

[57] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv:1905.11946. type: article. arXiv, Sept. 2020. DOI: 10.48550/arXiv.1905.11946. arXiv: 1905.11946[cs,stat]. URL: http://arxiv.org/abs/1905.11946 (visited on 03/08/2023).

[59] Ilya Tolstikhin et al. *MLP-Mixer: An all-MLP Architecture for Vision*. arXiv:2105.01601. type: article. arXiv, June 2021. DOI: 10.48550/arXiv.2105.01601. arXiv: 2105.01601[cs]. URL: http://arxiv.org/abs/2105.01601 (visited on 03/30/2023).

[60] Hugo Touvron et al. *Training data-efficient image transformers & distillation through attention*. arXiv:2012.12877. type: article. arXiv, Jan. 2021. DOI: 10.48550/arXiv.2012.12877. arXiv: 2012.12877[cs]. URL: http://arxiv.org/abs/2012.12877 (visited on 03/07/2023).

[61] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762. type: article. arXiv, Dec. 2017. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 11/30/2022).

[62] Saining Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. arXiv:1611.05431. type: article. arXiv, Apr. 2017. DOI: 10.48550/arXiv.1611.05431. arXiv: 1611.05431[cs]. URL: http://arxiv.org/abs/1611.05431 (visited on 03/22/2023).

[63] Sangdoo Yun et al. *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features*. arXiv:1905.04899. type: article. arXiv, Aug. 2019. DOI: 10.48550/arXiv.1905.04899. arXiv: 1905.04899[cs]. URL: http://arxiv.org/abs/1905.04899 (visited on 03/30/2023).

[64] Aston Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[65] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. arXiv:1710.09412. type: article. arXiv, Apr. 2018. DOI: 10.48550/arXiv.1710.09412. arXiv: 1710.09412[cs, stat]. URL: http://arxiv.org/abs/1710.09412 (visited on 03/07/2023).

Web References

[2] Andrej Karpathy [@karpathy]. *This consolidation in architecture will in turn focus and concentrate software, hardware, and infrastructure, further speeding up progress across AI. Maybe this should have been a blog post. Anyway, exciting times.* Twitter. Dec. 2021. URL: https://twitter.com/karpathy/status/1468370613886599170 (visited on 03/12/2023).

[11] *CUDA GPUs - Compute Capability*. Jan. 2023. URL: https://developer.nvidia.com/cuda-gpus (visited on 03/15/2023).

[41] *NVIDIA Documentation: Train With Mixed Precision*. URL: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html (visited on 03/15/2023).

[58] *Tensorflow Documentation: TFRecord and tf.train.Example*. URL: https://www.tensorflow.org/tutorials/load_data/tfrecord (visited on 03/15/2023).

Declaration of Authenticity

I declare that I wrote this thesis on my own and did not use any unnamed sources or aid. Thus, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made by correct citation. This includes any thoughts taken over directly or indirectly from printed books and articles as well as all kinds of online material. It also includes my own translations from sources in a different language. The work contained in this thesis has not been previously submitted for examination. I also agree that the thesis may be tested for plagiarized content with the help of plagiarism software. I am aware that failure to comply with the rules of good scientific practice has grave consequences and may result in expulsion from the program.

Berlin, 03.04.2023

Max Grundmann