Degree Project in Information and Network Engineering
Second cycle, 30 credits

3D Gaze Estimation on Near Infrared Images Using Vision Transformers

EMIL EMIR VARDAR
3D Gaze Estimation on Near Infrared Images Using Vision Transformers

EMIL EMIR VARDAR

Master’s Programme, Information and Network Engineering, 120 credits
Date: June 4, 2023

Supervisors: Péter Almási, Johanna Frost, Yusen Wang
Examiner: Markus Flierl
School of Electrical Engineering and Computer Science
Host company: Tobii AB
Swedish title: 3D Ögonblicksuppskattning på Nära Infraröda Bilder med Vision Transformers
Abstract

Gaze estimation is the process of determining where a person is looking, which has recently become a popular research area due to its broad range of applications. For example, tools that estimate gaze are used for research, medical diagnosis, virtual and augmented reality, driver assistance system, and many more. Therefore, better products are sought by many. Gaze estimation methods typically use images of only the eyes or the whole face to estimate the gaze since these methods are the most practical and convenient options. Recently, Convolutional Neural Networks (CNNs) have been appealing candidates for estimating the gaze. Nevertheless, the recent success of Vision Transformers (ViTs) in image classification tasks has introduced a new potential alternative. Hence, this work investigates the potential of using ViTs to estimate the gaze on Near-Infrared (NIR) images. This is done in terms of average error and computational complexity. Furthermore, this work examines not only pure ViTs but other models, such as hybrid ViTs and CNN-Formers, which combine CNNs and ViTs. The empirical results showed that hybrid ViTs are the only models that can outperform state-of-the-art CNNs such as MobileNetV2 and ResNet-18 while maintaining similar computational complexity to ResNet-18. The results on hybrid ViTs indicate that the convolutional stem is the most crucial part of them. Improved convolutional stems lead to better outcomes. Moreover, in this work, we defined a new training algorithm for hybrid ViTs, the hybrid Data-Efficient Image Transformer (DeiT) procedure, which has shown remarkable results. It is 3.5% better than the pretrained ResNet-18 while having the same time complexity.

Keywords
Gaze estimation, Eye tracking, Vision Transformers (ViTs), Hybrid ViTs, Deep learning, Near-infrared (NIR) images
ii | Abstract
Sammanfattning


Nykkelord

Ögonblicksuppskattning, Blickspårning, Vision Transformers (ViTs), Hybrida ViTs, Djupinlärning, Nära-infraröda (NIR) bilder
Acknowledgments

I would like to express my heartfelt gratitude to my parents, Eleonor and Erol Vardar, for their unwavering emotional and financial support throughout my educational journey. Their love, encouragement, and sacrifices have been invaluable to me.

Furthermore, I am grateful to my host company, Tobii, for making this thesis possible and supplementing me with the resources and equipment necessary to complete my thesis. My main supervisors, Péter Almási and Johanna Frost at Tobii, guided me throughout my thesis with their expertise and support. Hence, I am thankful to them for their valuable contributions. I also wish to express my gratitude to Adam Ly and Viktor Åberg for their valuable discussions during my thesis. Furthermore, I want to thank my thesis colleague Jing Li for our extensive discussions during our projects.

Finally, I want to thank Yusen Wang and Robert Andersson, who gave valuable feedback on my draft report and during my presentation. Furthermore, Markus Flierl gave a couple of valuable ideas during my presentation, which I have added to the final version of this thesis. Hence I also want to thank him.

Stockholm, June 2023
Emil Emir Vardar
Acknowledgments
# Contents

1 **Introduction**  
1.1 Background ........................................ 2  
1.2 Purpose ............................................. 3  
1.3 Goals ............................................... 4  
1.4 Research Methodology ............................... 5  
1.5 Structure of the thesis ............................. 6  

2 **Background** ........................................ 7  
2.1 Convolutional Layers ................................. 7  
2.1.1 Depthwise Separable Convolutions ................. 9  
2.2 Pooling Layers ...................................... 10  
2.3 Fully Connected Layers .............................. 11  
2.3.1 Multi Layer Perceptron ........................... 12  
2.4 Activation Functions ................................. 12  
2.4.1 Rectified Linear Unit .............................. 13  
2.4.2 ReLU6 ............................................. 13  
2.4.3 Gaussian Error Linear Units ..................... 13  
2.4.4 Softmax .......................................... 14  
2.5 Dropout .............................................. 15  
2.6 Normalization ....................................... 15  
2.6.1 Batch Normalization ............................. 15  
2.6.2 Layer Normalization .............................. 17  
2.7 Loss Function and Training .......................... 18  
2.8 Pretraining and Finetuning ......................... 19  

3 **Related Work** ........................................ 20  
3.1 ResNet .............................................. 20  
3.2 MobileNetV2 ........................................ 22  
3.3 Transformers ........................................ 24
3.3.1 Encoder Layer ........................................... 27
3.3.2 Decoder Layer ........................................... 27
3.3.3 Multi-Head Attention ................................. 28
3.3.4 Pre-processing and Post-processing ............. 29
3.4 Vision Transformer ......................................... 30
3.5 Differences Between CNNs and ViTs ............... 32
3.6 Hybrid Vision Transformer .............................. 32
3.7 Advanced Methods in Vision Transformers ......... 33
   3.7.1 Locality Self-Attention Mechanism .............. 33
   3.7.2 Data-Efficient Image Transformers .............. 34
   3.7.3 CNN-Former ......................................... 35
3.8 Summary .................................................. 37

4 Gaze Estimation ............................................. 38
   4.1 Data Pre-Processing .................................. 38
   4.2 Data Post-Processing ................................ 41

5 Methods ...................................................... 43
   5.1 Model .................................................. 43
   5.2 Backbone ............................................. 44
      5.2.1 Convolutional Neural Networks .............. 44
      5.2.2 Pure Vision Transformers ...................... 45
      5.2.3 Hybrid Vision Transformers .................. 46
      5.2.4 Advanced Vision Transformers .............. 47
   5.3 Datasets ................................................ 49
      5.3.1 Internal Near-Infrared Dataset .............. 49
      5.3.2 ETH-XGaze Dataset ............................. 50
   5.4 Implementation Details ................................ 50
      5.4.1 Pretraining & Finetuning ..................... 52
   5.5 Evaluation Metrics .................................... 53
      5.5.1 Performance Metrics ............................ 53
      5.5.2 Computational Metrics ....................... 55
   5.6 Used Tools ............................................ 55
      5.6.1 Work Supplied Before This Thesis .......... 56

6 Results and Analysis ....................................... 57
   6.1 Best Learning Rate Schedule ......................... 57
   6.2 Pure ViTs Without Pretraining ....................... 58
   6.3 Hybrid ViTs Without Pretraining ..................... 60
   6.4 Advanced Methods on ViTs Without Pretraining .... 62
x | Contents
## List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAE</td>
<td>Average Angular Error</td>
</tr>
<tr>
<td>CCS</td>
<td>Camera Coordinate System</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DeiT</td>
<td>Data-Efficient Image Transformer</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>FC</td>
<td>Fully Connected</td>
</tr>
<tr>
<td>G2L</td>
<td>Global to Local</td>
</tr>
<tr>
<td>GELU</td>
<td>Gaussian Error Linear Units</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>HCS</td>
<td>Head Coordinate System</td>
</tr>
<tr>
<td>INIR</td>
<td>Internal Near-Infrared</td>
</tr>
<tr>
<td>L2G</td>
<td>Local to Global</td>
</tr>
<tr>
<td>LSA</td>
<td>Locality Self-Attention</td>
</tr>
<tr>
<td>MLP</td>
<td>Multilayer Perceptron</td>
</tr>
<tr>
<td>NCCS</td>
<td>Normalized Camera Coordinate System</td>
</tr>
<tr>
<td>NIR</td>
<td>Near-Infrared</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>RCCS</td>
<td>Rotated Camera Coordinate System</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RGB</td>
<td>Red-Green-Blue</td>
</tr>
<tr>
<td>RMSAE</td>
<td>Root Mean Square of Angular Error</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>SPT</td>
<td>Shifted Patch Tokenization</td>
</tr>
</tbody>
</table>
ViT  
Vision Transformer
List of Symbols Used

The following symbols will be later used within the body of the thesis.

- $\mathbb{R}^{m \times n}$ Shows that a matrix is of shape $m \times n$.
- $A$ Bold uppercase letters represent matrices.
- $q$ Bold lowercase letters represent vectors.
- $x, X$ Uppercase or lowercase non-bold italic letters represent scalars.
Chapter 1

Introduction

Gaze estimation is the process of estimating where a person is currently looking. Even though eye tracking is mainly used as a synonym to gaze estimation, it is a broader concept where the eye position and movements are also estimated. Eye trackers are devices for tracking the eyes and evaluating the gaze direction. There is numerous ongoing research on eye trackers since gaze is one of the most important non-verbal cues for human communication and intentions [1]. Eyes, more detailed gaze direction, pupil dilation, blink rate, etc., can indicate a person’s intentions, desires, and feelings to a certain degree [2, 3]. Therefore, eye trackers are used in psychology, ophthalmology, and neurology to study human behavior [2] and cognition [3].

Besides research and medical diagnosis mentioned above, eye tracking tools are also used in various applications, e.g., appropriate marketing, human-computer interaction, virtual and augmented reality, entertainment, driver assistance system, robotics, and many more [4]. Therefore, eye gaze tracking tools with high accuracy and precision are strived by many. This has led to many different approaches for estimating a person’s gaze in the past. One of the first experiments on gaze estimation in reading was done by Edmund B. Huey [5]. In his experiments, he placed contact lenses on a person’s eyes and observed how the eyes moved when reading a text.

These kinds of intrusive methods are impractical for many applications mentioned above. Therefore, research on non-intrusive gaze estimation tools has lately been the trend [1, 6, 7, 8, 9, 10]. These methods build on using a person’s face or eyes as input, captured by off-the-shelf cameras, and estimating the gaze from these inputs. The gaze direction from an image can be estimated by the appearance of the eyes and other cues around the eyes and in the face. This makes the process less painful and expands the practical usage
area of eye trackers. The cameras used for capturing images of a face or eyes are today so advanced that they can be implemented into wearable glasses [11] or small hardware that can be placed on top of the screen of a computer or in a vehicle [12].

In this chapter, we first briefly introduce the state-of-the-art methods used in gaze estimation. These methods are later detailed in chapter 3. Furthermore, in this chapter, we state the problem, purpose, and goals of this thesis and end this chapter by shortly describing the outline of this thesis.

1.1 Background

As mentioned above, gaze estimation is a topic that has been researched for a long time because of its vast application areas. The first method for estimating the gaze was in 1908, where a lens was attached to a person’s eye [5]. Later, methods, where electrodes were placed on the face were used [13]. In these methods, the difference in electrical potential was used to detect eye movements [1]. However, these kinds of intrusive methods are unsuitable for many tasks today. Therefore, they have lately been replaced by non-intrusive appearance-based methods.

In appearance-based methods, the gaze is estimated solely from an image a camera has captured. The most commonly used camera models for this purpose are the Red-Green-Blue (RGB) cameras [1]. However, this work focuses on Near-Infrared (NIR) images. Because Tobii’s Internal Near-Infrared (INIR) dataset consists of NIR images. The advantage of using NIR images is that gaze can be estimated even in dim and no light conditions.

There have been many approaches to appearance-based gaze estimation. For example, sparse, semi-supervised Gaussian process (S3GP) [14], adaptive linear regression [15], k-Nearest Neighbors, and random forest [16] have been used to estimate gaze direction. However, none of these conventional gaze estimation methods have managed to cope with gaze estimation methods based on Deep Neural Networks (DNNs) [1, 6].

In almost all computer vision applications, Convolutional Neural Networks (CNNs) [17] have long been the cutting-edge approach. This trend is also visible in gaze estimation. For example, [8], [18], [19], and [20] all use CNNs for estimating the gaze. However, a new DNN architecture, Vision Transformer (ViT), has managed to surpass CNNs on classification tasks when trained with huge datasets [21]. This recent success in classification tasks has also led to testing ViTs on regression tasks such as gaze estimation. Cheng et al. [6] showed that ViTs could not surpass CNNs when estimating the gaze.
direction. However, they showed that hybrid ViTs could achieve state-of-the-art results on gaze estimation. A hybrid ViT is a DNN architecture combining a CNN with a ViT. These concepts are introduced and explained more in chapter 2 and 3 and are the fundamental building blocks for this thesis.

1.2 Purpose

This thesis mainly aims to research whether there are better DNNs for appearance-based gaze estimation than current state-of-the-art CNNs. We investigate whether the contemporary architecture ViT can surpass current CNN-based approaches on NIR images and expand our knowledge of ViTs to regression tasks.

Furthermore, we investigate how the performance of pure and hybrid ViTs can be improved on limited NIR images. The original paper [21] underlines that ViTs surpass the performance of CNNs only when pretrained on huge datasets such as JFT-300M [22]. Therefore, it is interesting to see how pure and hybrid ViTs perform when trained on a smaller dataset (INIR dataset) with and without pretraining. [23] and [24] have already looked into improving the performance on limited datasets, to mention a few. However, these are all based on classification tasks which differ from our case. Moreover, for example, how the distillation idea [23] can be adapted to regression tasks and hybrid ViTs is not straightforward and has not been tested previous to this work according to our best knowledge. Hence, another purpose of this thesis is to investigate if methods used in classification tasks can improve the performance of regression tasks.

 Nonetheless, another purpose of this thesis is to investigate if the results published in [6] also are valid for NIR images. Here Cheng et al. show that hybrid ViTs can surpass state-of-the-art CNNs on gaze estimation when pretrained on the ETH-XGaze dataset [25]. However, a big difference between [6] and this work is that Cheng et al. use RGB images as their input both for pretraining and finetuning. This is convenient when an extensive dataset such as ETH-XGaze [25] exists in RGB. On the other hand, in this thesis, we use NIR images that do not have an extensive dataset for gaze estimation that can be used as pretraining. This brings us again to the issue explained in the previous paragraph.
1.3 Goals

The main goal of this thesis is to compare pure CNN-based models on gaze estimation on the INIR dataset by the new DNN architecture: ViT. Both pure and hybrid ViT models are tried to come to a more informed conclusion. Furthermore, we test the recent CNN-Former architecture, which consists of a parallel approach of Transformers and CNNs [26]. The goal is to compare these models’ performance and computational complexity and decide on a model that is both sufficiently fast and accurate; see the next subsection for details. To achieve this goal, we divided the process into the following subgoals in order to be more concrete.

1. Use MobileNetV2 [27] and ResNet-18 [28] as baselines (see chapter 3 for more on these models). ResNet-18 performs better than MobileNetV2. However, it is much slower than MobileNetV2. Therefore, the main goal is to surpass the performance of MobileNetV2 while keeping the computational complexity as close as possible to MobileNetV2.

2. Investigate if pure and/or hybrid ViTs without pretraining can outperform MobileNetV2.

3. Test whether CNN-Formers, introduced in [26], without pretraining, have the capability to surpass the performance of the CNN baselines.

4. Investigate if the performance of the best pure and hybrid ViTs and/or CNN-Formers obtained above can be improved when using pretraining on the ETH-XGaze dataset. MobileNetV2 is also pretrained on the same dataset to make the comparison fairer.

5. Adjust the distillation presented in [23] and Locality Self-Attention (LSA) presented in [24] to the best hybrid ViT obtained from the previous steps. Here we only try these methods on hybrid ViTs because the results showed that hybrid ViTs perform much better than pure ViTs.

6. Finally, investigate if more extended training when using cosine learning rate on the best models obtained above can improve the performance with and without pretraining on the ETH-XGaze dataset.

The results showed that hybrid ViTs are the ones that perform best among the DNNs investigated in this thesis. Although hybrid ViTs perform well, they are slower than MobileNetV2. Hence, they are not suitable for mobile
applications. It is worth noting that hybrid ViTs as fast as ResNet-18 could be constructed in this work. Therefore, in scenarios where ResNet-18 is being considered, hybrid ViTs should be considered strong candidates. Pure ViTs, CNN-Formers, and the LSA method could not surpass the performance of pure CNNs. However, the modified distillation method, hybrid Data-Efficient Image Transformer (DeiT) as we name it, gave remarkable results on hybrid ViTs. Finally, longer training has shown to be necessary to improve the performance on ViTs.

1.4 Research Methodology

In order to properly evaluate the models, the INIR dataset is divided into two sub-datasets: training and evaluation datasets, following the standard research methodology in machine learning. All the models used in this work are trained on the training dataset and evaluated on the evaluation dataset. We ensure that the training and evaluation dataset contains different images and subjects to make the results more convincing. Doing so can avoid the misleading effect of overfitting. This work will mainly compare the models based on their performance on the evaluation dataset. If not something else is mentioned, the results presented in chapter 6 are on the evaluation dataset. However, due to the privacy of the INIR dataset, we will refrain from using absolute measures when presenting the results on performance. The results will be given relatively, such as Model A performed x% better/worse than Model B.

The models are going to be compared based on two aspects listed below.

1. **The performance.** The following metrics will quantitatively measure the performance.
   - Average Angular Error (AAE).
   - Root Mean Square of Angular Error (RMSAE).
   - Inlier ratio.

2. **The computational complexity.** This will quantitatively be measured with the following two metrics.
   - Throughput.
   - Total number of learnable parameters.

These evaluation matrices and their calculations are detailed in section 5.5.
1.5 Structure of the thesis

Chapter 2 presents the fundamentals of CNNs and ViTs. Chapter 3 looks into more recent research in which the fundamental concepts have been used beautifully to obtain state-of-the-art results. We present MobileNetV2 [27], and ResNet-18 [28] together with advanced techniques in ViTs in this chapter as well. Chapter 4 explains the state-of-the-art data pre-processing and post-processing techniques used in gaze estimation. Chapter 5 explains our method and evaluation metrics used to compare the models. The results are presented and analyzed in chapter 6. The findings are discussed and concluded in chapters 7 and 8. Finally, in chapter 9, we give ideas for future research.

For readers with a strong background in deep learning, CNNs and ViTs, we recommend skipping the second and third chapters to save time. However, for readers who want to freshen their knowledge of these concepts, we recommend reading or at least skimming through the second and third chapters in order to understand the essential parts which are later used in chapter 4 and onwards. Nevertheless, we believe each reader is aware of their strengths and weaknesses. Therefore, to make the most out of this thesis, we recommend looking through the Contents of this paper and focusing on areas that are more relevant for each reader.
Chapter 2
Background

CNNs [17] have been the state-of-the-art deep learning models in computer vision in the last decade. After the breakthrough of AlexNet [29], many other CNN-based architectures have emerged in different fields of computer vision, e.g., image classification, object detection, and semantic segmentation. Some of the well-known CNN architectures are VGG-16 [30], Inception and GoogLeNet [31], ResNet [28] and MobileNetV2 [27]. This work primarily focuses on ResNet and MobileNetV2 (because of their appealing performance and low computational complexity) and compares these two with each other and the newly constructed ViT-based models.

Current CNNs such as the ones mentioned above generally consist of three essential layers: convolutional, max pooling, and fully connected layers. These are linear operators and are explained in sections 2.1-2.3. However, activation functions need to be used to break the linearity of the DNNs. The most important activation functions we use in this work are explained in section 2.4. Furthermore, two regularization techniques, dropout, and normalization, widely used in deep networks, are presented in section 2.5 and section 2.6, respectively. This section ends with a brief explanation of how a DNN can be trained with backpropagation and what the terms pretraining and finetuning mean in sections 2.7 and 2.8, respectively.

2.1 Convolutional Layers

A convolutional layer takes as input a feature map of shape \((H_i, W_i, C_i)\) where \(H_i\) is the height, \(W_i\) is the width, and \(C_i\) is the channel (depth) of the input feature map. Then it applies a kernel (filter) of shape \((k, k, C_i)\) to this feature map, as illustrated in Figure 2.1 and Figure 2.2a [27]. The same kernel
Figure 2.1: A convolutional layer example, with an input feature map of shape (3, 3, 2) and a kernel of shape (2, 2, 2). The values in the kernel are multiplied with the corresponding indices in the input feature map and then added. Observe that similar colored squares correspond to the same output index. Furthermore, the red-colored numbers indicate that the values correspond to the first channel and the green-colored numbers to the second channel in the input feature map. Observe that each channel has a different kernel assigned to it, but the kernel weights for each channel are shared.

traverses the input feature map in $x$ and $y$ dimensions to yield an output of shape $(H_o, W_o, 1)$. This traversing gives a local inductive bias to CNNs, a vital feature for learning in visual inputs [24].

When traversing, the kernel values are multiplied with the input features and then added (see the example in Figure 2.1). Applying a kernel of shape $(k, k, C_i)$ results in an output with only one channel. In other words, the kernel is applied to all the channels in the input simultaneously. Hence, all the channel values collapse to one channel shown with the light blue layer in Figure 2.2a. Therefore, to have multiple channels in the output feature map, one must have $C_o$ kernels with the same shape but with different parameters. Thus, applying a kernel of shape $(k, k; C_i, C_o)$ yields an output of shape $(H_o, W_o, C_o)$. This results in a computational cost of $H_o \times W_o \times k \times k \times C_i \times C_o$ for a convolutional layer [27].

The $H_o$ and $W_o$ can be calculated by the following formula.

$$\text{Output shape} = \left\lfloor \frac{\text{Input shape} - k + 2p}{s} \right\rfloor + 1,$$

where ‘Input shape’ is $H_i$ or $W_i$ and ‘Output shape’ is $H_o$ or $W_o$. Furthermore, $p$ is the padding, and $s$ is the stride. Padding is the action of adding an appropriate number of rows and columns on each side of the input feature map before applying the kernel [32]. It is mainly used to preserve spatial information.
dimensions, i.e., it gives the flexibility to have \( H_o = H_i \) and \( W_o = W_i \). In Figure 2.1, \( p = 0 \) since we have not added any new columns or rows to the input feature map. Usually, zero-padding is used in practice, meaning that the input feature is padded with zeros (0). Furthermore, the stride is the step size when sliding the filter over the input feature map [32]. In Figure 2.1, the stride is \( s = 1 \) both in \( x \) (step size between the orange and blue box) and \( y \) (step size between the orange and purple box) direction.

Usually, after a kernel has been applied to the input feature map, a learnable bias term is added to the resulting feature map. The biases are of shape \( b \in \mathbb{R}^{1 \times C_o} \), i.e., the same bias is added to all the entries in the same channel. In the example given in Figure 2.1, the bias would be of shape \( b \in \mathbb{R}^{1 \times 1} \) and added with all the output values resulting in an output feature map with values: 4+b, 10+b, 1+b, and 3+b. Lastly, a non-linear activation function is applied to this, which is discussed in section 2.4. Hence, the total number of learnable parameters for a convolutional layer with kernel size \((k, k, C_i, C_o)\) is \((k \times k \times C_i + 1) \times C_o\).

### 2.1.1 Depthwise Separable Convolutions

Depthwise separable convolutions are the critical building blocks for networks that not only seek high accuracy but also strive for fast inference, such as MobileNetV2 [27], ShuffleNet [33], and Xception [34]. The idea of depthwise separable convolutions is illustrated in Figure 2.2b. Here, we see that the depthwise separable convolutions consist of two steps. The first step is the so-called ‘Depthwise’ convolution, and the second is the ‘Pointwise’ convolution [27].

Depthwise convolution consists of \( C_i \) kernels each of shape \((k, k, 1)\),
i.e., a single convolutional filter per input channel [27]. Hence, this step has no information exchange between the different channels [27]. Therefore, the computational complexity of this step is $H_o \times W_o \times k \times k \times C_i$.

The pointwise convolution consists of $C_o$ kernels, each of shape $(1, 1, C_i)$. This step is responsible for finding linear combinations between the different channels [27]. Therefore, this layer’s computational complexity is given by $H_o \times W_o \times C_i \times 1 \times 1 \times C_o$. This yields to a total complexity of $H_o \times W_o \times C_i \times (k^2 + C_o)$ [27]. Compared to the normal convolutional layers explained above, this reduces the computational cost by

$$\text{Reduction in complexity} = \frac{H_o \times W_o \times k \times k \times C_i \times C_o}{H_o \times W_o \times C_i \times (k^2 + C_o)} = \frac{k^2 \times C_o}{k^2 + C_o} = \{\text{assuming } C_o \gg k\}$$

(2.2)

So on a kernel with size $3 \times 3$, the depthwise convolution is approximately 9 times more efficient than regular convolutions, and the performance is hardly affected by this, according to [27].

### 2.2 Pooling Layers

A pooling layer is used to aggressively downsample the input feature map’s spatial size (i.e., width and height) [32]. While this is done, important features should be retained. A pooling layer is essential in CNNs since it allows the

![Figure 2.3: Illustration of average and max pooling operations. A window of size $2 \times 2$ with stride 2 for the pooling operation is used in this illustration. In max pooling, the biggest value in the window is outputted. In average pooling, the average in each window is outputted.](image)
convolutional layers to look at increasingly large windows while keeping the number of trainable parameters low [32]. There are two well-known pooling operations: max pooling and average pooling. In max pooling, the input features are divided into windows, and the biggest value in each is outputted. In average pooling, the mean value in each of these windows is outputted. This idea is illustrated in Figure 2.3 with an example. Observe that a pooling operation does not have any trainable/learnable parameters.

### 2.3 Fully Connected Layers

A Fully Connected (FC) layer, which is also called the Dense layer, is a DNN layer, where each input unit is connected to every output unit [35]. This is illustrated with the black arrows between the colored neurons in Figure 2.4. This is contrary to convolutional layers, where output values only depend on some specific spatial coordinates in the input feature map. Hence, obtaining global attention with only convolutional layers can be challenging and computationally heavy. Therefore, using FC layers after a couple of convolutional and pooling layers is a well-followed practice when designing CNNs [29, 30, 28]. The objective of convolutional and FC layers can be summarized as extracting visual features from the image and making a prediction/classification based on the outputs from the previous convolutional layers, respectively [35].

As mentioned in section 2.1, the output of a couple of convolutional layers will be of shape \((H_o, W_o, C_o)\) (ignoring the batch size). Hence, the tensor must be flattened before an FC layer can be applied to this, which can be done

![Figure 2.4: A 2-layer Multilayer Perceptron (MLP) with input size 3, hidden size 4, and output size 2.](image-url)
differently. ResNet, for example, uses average pooling to obtain a flattened vector of shape \((1, C_0)\) at the last layer [28], and some other networks directly flatten the input tensor to obtain a vector of shape \((1, H_o \times W_o \times C_o)\).

After the flattened vector \(x_i \in \mathbb{R}^{1 \times n_{in}}\) has been obtained, a weight matrix, \(W \in \mathbb{R}^{n_{in} \times n_{out}}\), is applied to this vector. The application of this weight matrix is the essence of an FC layer. Here \(n_{in}\) and \(n_{out}\) are the input and output dimensions, respectively. Mathematically, this has been shown in Equation 2.3 [35].

\[
x_o = x_i W + b,
\]

where \(b \in \mathbb{R}^{1 \times n_{out}}\) is the bias and \(x_o \in \mathbb{R}^{1 \times n_{out}}\) is the output vector. After each FC layer, a non-linear activation function is applied, as explained in section 2.4.

Since the shape of the weight matrix in an FC layer is \(n_{in} \times n_{out}\), the number of learnable parameters is \((n_{in} + 1) \times n_{out}\) per FC layer. This is usually much bigger than a convolutional layer, and therefore, it is preferable to use many convolutional and max pooling layers before the input features are flattened. This reduces the computational complexity of the model.

### 2.3.1 Multi Layer Perceptron

MLP is a DNN architecture consisting of multiple FC layers stacked each other, as illustrated in Figure 2.4 [35]. Here the first (blue) layer is called the input layer. Furthermore, the middle (red) layer is called the hidden layer. Finally, the last (green) layer is called the output layer. If there are many layers between the input and output layers, all these are called hidden layers. An MLP like in Figure 2.4 is called a 2-layer MLP because of the usage of two weight matrices, \(W_1 \in \mathbb{R}^{n_{in} \times h}\) and \(W_2 \in \mathbb{R}^{h \times n_{out}}\). The mathematical formulation for an MLP has been given in Equation 2.4 [35].

\[
x_o = \sigma_2(\sigma_1(x_i W_1 + b_1)W_2 + b_2),
\]

where \(\sigma\) represents a non-linear activation function.

### 2.4 Activation Functions

An activation function can be seen as a non-linear switch in the network. It triggers neuron outputs in order to break the linearity of the convolutional and FC layers. Activation functions used in this work are presented below.
2.4.1 Rectified Linear Unit

Rectified Linear Unit (ReLU) is today one of the most commonly used activation functions in deep learning. This is because ReLU converges faster and better than previously used activation functions such as tanh and sigmoid [36, 37]. The mathematical formulation for ReLU is given in Equation 2.5 [36, 37].

$$\text{ReLU}(z) = \max\{0, z\}.$$  \hspace{1cm} (2.5)

That is, if the input is bigger than 0, the input is returned as it is, while if the input is less than 0, the output will be 0 (i.e., no response). This is shown in Figure 2.5.

Observe that the max operation in Equation 2.5 is element-wise. That is, if $z$ had been a matrix, ReLU would apply to each index separately and independently. This breaks the linearity and allows the system to learn complicated functions which are not linear.

2.4.2 ReLU6

ReLU6 is the capped version of ReLU at unit 6, as shown in Equation 2.6 [38].

$$\text{ReLU6}(z) = \min\{\max\{0, z\}, 6\}.$$  \hspace{1cm} (2.6)

ReLU6 encourages the neural network to learn sparse features earlier [38] and is more robust to low-precision calculations [27].

2.4.3 Gaussian Error Linear Units

Gaussian Error Linear Units (GELU) was first introduced in [39]. In this paper, Hendrycks et al. showed that GELU outperformed ReLU and ELU [40] on different tasks consistently, making it a strong candidate for usage as an activation function. The formula for GELU is given by Equation 2.7 and the plot for it is shown in Figure 2.5 [39].

$$\text{GELU}(z) = z\Phi(z) = z \cdot \frac{1}{2}[1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right)],$$  \hspace{1cm} (2.7)

where $\Phi(z)$ is the standard Gaussian cumulative distribution function, i.e., $\mu = 0$ and $\sigma = 1$. Hendrycks et al. explain the success of GELU by being non-convex, non-monotonic, and non-linear in the positive domain.
2.4.4 Softmax

The Softmax activation function is used to represent an output of $n$ discrete variables as a probability distribution [41]. The formula for it is given in Equation 2.8, assuming $z \in \mathbb{R}^n$.

$$\text{Softmax}(z) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \text{ for } i = 1, \ldots, n.$$ (2.8)

Observe that this guarantees

$$\sum \text{Softmax}(z) = 1 \text{ and } 0 \leq \text{Softmax}(z)_i \leq 1,$$

which is a necessary and sufficient condition for being a probability distribution.

The Softmax activation function is mainly used in classification tasks as the last layer in a DNN [41]. However, as we will see in the case of Transformer [42] and ViT [21], it can also be used as an internal component, see e.g. Equation 3.5.
2.5 Dropout

A dropout is a form of a regularization technique that prevents overfitting in DNN [43]. Overfitting means the model performs much better on the training than the test set. Hence, the importance of dropout increases when using a small dataset. In dropout in each training case, some of the hidden units in a layer are randomly omitted with a predefined probability $p_d$ [43]. This prevents the hidden units in the next layer from entirely relying on a specific hidden unit in the previous layer [43]. This provides the network with a regularization effect and hence, better test set results. According to Hinton et al. [43], dropout can be thought of as training many different networks simultaneously.

In the prediction phase, a ‘mean network’ is used [43]. This means that all the neurons are active, but a factor of $p_d$ scales the learned weights in the model to compensate for the training time where the neurons were deactivated [43].

2.6 Normalization

In the study of DNNs, many different normalization techniques have been suggested. This work uses batch normalization [44] and layer normalization [45], which are the two most common ones. These normalization techniques aim to reduce the internal covariate shift and, by that, speed up the training process [44]. These two normalization techniques are explained in detail below.

2.6.1 Batch Normalization

One crucial property of DNNs is that the input to layer $l$ depends on all the previous layers $1, ..., l – 1$. Thus, when a small change occurs in one of the network parameters (due to backpropagation), this effect is carried on to the later stages of the network [44]. Sometimes even a small change in one layer can be amplified in the later layers. This is the undesirable property called ‘covariate shift’ [45]. Covariate shift causes the model to learn slower since the later layers must continuously adapt to big accumulated changes caused by earlier layers in the model [44]. To neutralize this issue and speed up the learning process, Ioffe et al. [44] proposed batch normalization.

Assume the output of an FC layer is a $d$-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ and the batch size is $m$, i.e., the output is a matrix of shape $\mathbf{X} \in \mathbb{R}^{m \times d}$. Then this
**X** is batch normalized as given in Equation 2.9.

\[
\hat{X}_{ij} = \frac{X_{ij} - \mu_j^B}{\sqrt{(\sigma_j^B)^2 + \epsilon}},
\]

(2.9)

where \( j \in [1,d], i \in [1,m], \) and \( \epsilon \) is given for numerical stability [44]. Furthermore, \( \mu_j^B \in \mathbb{R}^d \) is the mini-batch mean, and \( (\sigma_j^B)^2 \in \mathbb{R}^d \) is the mini-batch variance. These values are given by

\[
\mu_j^B = \frac{1}{m} \sum_{i=1}^{m} X_{ij} \quad \text{and} \quad (\sigma_j^B)^2 = \frac{1}{m} \sum_{i=1}^{m} (X_{ij} - \mu_j^B)^2.
\]

(2.10)

However, this normalization in some situations can be problematic. For example, using a sigmoid activation function causes the normalized values to always be in the linear regime. To counteract this problem, Ioffe et al. suggest adding two new learnable parameters to the batch normalization layer. Thus the resulting output from a batch normalization layer can be represented as follows:

\[
Y_{ij} = \gamma_j \hat{X}_{ij} + \beta_j.
\]

(2.11)

where \( \gamma \) and \( \beta \) are learnable parameters per batch normalization layer.

Batch normalization is used in ResNet and MobileNetV2, which are introduced later. However, both ResNet and MobileNetV2 are based on convolutional layers. Hence, batch normalization is done slightly differently. Assume that one wants to batch normalize a feature map \( X \) of shape \((N, C, H, W)\) where \( N \) is the batch size, \( C \) is the channel, \( H \) is the height, and \( W \) is the width. In convolutional layers, batch normalization is done among the pixels corresponding to the same channel according to Equation 2.12 [46].

\[
\mu_c^B = \frac{1}{HWN} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{n=1}^{N} X_{nchw} \quad \text{and} \quad (\sigma_c^B)^2 = \frac{1}{HWN} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{n=1}^{N} (X_{nchw} - \mu_c^B)^2.
\]

(2.12)

In words, we find a mean and variance for each channel in a feature map by adding all the pixels in \( N, H, \) and \( W \) directions. Finally, the normalized feature map values are calculated similarly to Equation 2.9. Figure 2.6 shows the batch normalization principle for a convolutional layer.
2.6.2 Layer Normalization

Another normalization technique is layer normalization, commonly used in Recurrent Neural Networks (RNNs) and Transformers [42]. In sequential inputs, which RNNs and Transformers are designed for, the input length can vary, which makes batch normalization insufficient and is the main reason why layer normalization in the first place was introduced.

To illustrate the weakness of batch normalization on different sequential lengths, let us look at an example. Assume we use a Transformer to translate an English sentence to French. Furthermore, assume that one of the English sentences in the batch is longer than the rest. Then when we try to normalize over the batch, no other values will correspond to the last features in this long English sentence. This is not ideal since it does not yield any normalization. Therefore, Ba et al. [45] proposed layer normalization.

In layer normalization, each sample is averaged over itself [45]. That is contrary to batch normalization; there is no normalization over the different samples in layer normalization. Equation 2.13 gives the mathematical formula for layer normalization.

\[
\mu^L_i = \frac{1}{d} \sum_{j=1}^{d} X_{ij} \quad \text{and} \quad (\sigma^L_i)^2 = \frac{1}{d} \sum_{i=1}^{d} (X_{ij} - \mu^L_i)^2.
\]  

(2.13)

Here, we again assume that \( X \in \mathbb{R}^{m \times d} \) where \( m \) is the batch size and \( d \) is the hidden size of the vector. Hence, this time we get \( \mu^L_i \in \mathbb{R}^m \) and \( (\sigma^L_i)^2 \in \mathbb{R}^m \).

After the mean and variance have been obtained, the normalized value is again obtained with Equation 2.9 by replacing \( \mu^B_j \) with \( \mu^L_i \) and \( (\sigma^B_j)^2 \)
with \((\sigma^L)^2\). Observe that the subscript \(j\) is replaced by \(i\) making it layer normalization. Furthermore, as in batch normalization, learnable parameters can be added to make the layer normalization more flexible (see Equation 2.11).

Observe again that above, we looked at a simple FC layer case. In inputs with bigger dimensions, once again, more complex averaging is obtained, as shown in Figure 2.6. Hence, the mathematical formulation for layer normalization on four-dimensional inputs expands to the following.

\[
\mu^L_n = \frac{1}{HWC} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} x_{nchw}
\]

\[
(\sigma^L_n)^2 = \frac{1}{HWC} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{n=1}^{C} (x_{nchw} - \mu^L_n)^2.
\]

Even if the more complex cases have been explained in terms of convolutional layers, the idea is the same for Transformer layers.

### 2.7 Loss Function and Training

The weights in a neural network are the core of learning. These weights need to be learned accurately and robustly to ensure good performance when different inputs are sent into the network. These weights are learned by a method called backpropagation [47]. In order to learn these parameters, they need to be initiated before the training starts. The initialization is done randomly. Different initialization techniques, such as Xavier initialization [48], He initialization [49], etc., are used to improve the initialization.

In backpropagation, the parameters in the network are repeatedly adjusted so that a predefined loss function is minimized [47]. A loss function is a function that compares the predicted outputs from the network with the ground truth values. The goal is to minimize this loss function so that the predicted values become closer to the actual values. A loss function can be shown as

\[
\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}_i(\hat{y}_i, y_i),
\]

where \(\theta\) is the learnable parameters in the network, and \(m\) is the number of training samples (usually the batch size). Furthermore, \(\hat{y}_i\) are by the network predicted values corresponding \(x_i\) (i.e., \(\hat{y}_i = \text{DNN}(x_i, \theta)\)) and \(y_i\) are the
ground truth values.

The network parameters can be updated, according to the loss function, using different optimizers. Adam [50] and Stochastic Gradient Descent (SGD) are two well-known optimizers. In this work, we will use AdamW [51] which is a generalized version of Adam where weight decay has been added for better optimization. For details on AdamW we refer to [51].

An important hyperparameter in the optimization algorithm is the learning rate. The learning rate determines the step size at each iteration the optimizer takes in the direction of the gradient. Lower learning rates will result in smaller steps, and higher learning rates will result in bigger steps. A too-low learning rate will hence result in slow convergence, and a too-high learning rate will result in jumping over the local minima and diverging. Therefore, for better convergence, the learning rate of the optimizer can be changed during the training. This change is called the learning rate schedule. There are different learning rate schedules. In this work, we will use two of them: cosine learning rate [52] and multi-step learning rate. These learning rate schedules are explained more in section 5.4.

2.8 Pretraining and Finetuning

Deep learning methods are data-hungry. Usually, when the same model is trained on a big and small dataset in the same way, the model trained on the bigger dataset will outperform the model trained on a smaller dataset. However, data collection is an expensive procedure and is usually not the first method followed in practice. A method called pretraining has been proposed to alleviate this problem.

Pretraining is the phase where the DNN captures some relevant knowledge necessary for the target task from one or more source tasks [53]. These source tasks do not necessarily need to be the same as the target task, but having them closer to the target task helps.

The pretraining is followed by a phase called finetuning [53]. In the finetuning phase, the knowledge from the pretraining phase is used as a basis and further adapted to the target task [53]. In this work, gaze estimation in NIR images is the target task. However, before the model is trained on the target task, it might be trained on a source task such as gaze estimation in RGB images, which is the case in this work. The model in pretraining might learn some extra knowledge that it could not learn solely by training on NIR images and hence, boost the model’s performance when finetuned on the NIR dataset.
Chapter 3
Related Work

This chapter goes through all the relevant work done previous to this thesis using the concepts explained in the previous chapter. Sections 3.1 and 3.2 explain ResNet and MobileNetV2, which will be used as baselines in this work and as part of the hybrid ViTs. After that, in section 3.3, the Transformer is introduced, as it is used in Natural Language Processing (NLP) applications. Later in section 3.4, how the Transformer is used in visual inputs is explained, i.e., the ViTs are introduced. After that, how a CNN and a ViT can be combined to form a hybrid ViT is explained in section 3.6. This chapter ends with explaining advanced methods that have been tested on pure and hybrid ViTs. Overall this chapter gives the relevant background for the essential DNN architectures used in this thesis.

3.1 ResNet

ResNet is one of the best performing CNNs commonly used in computer vision tasks today and has inspired many other works. It was introduced by He et al. in 2015 [28]. ResNet became so popular because it successfully prevented a phenomenon called vanishing/exploding gradients [48]. Vanishing/Exploding gradients can cause a model to converge very slowly or, even in some cases, prevent a convergence. However, different methods have been suggested to prevent this phenomenon over the years. Some of them are normalized initializations [28], batch normalization [44], and using ReLU activation function as non-linearity in the network [44]. In ResNet, the vanishing/exploding gradients are prevented with the so-called residual connections.

Assume a DNN $\mathcal{F}$ (with learnable parameters $\Theta$), which takes a vector $x$
as input and produces \( y \) as output, i.e., \( y = F(x, \Theta) \). Theoretically, if we stack new layers on top of this neural network (let us denote these newly added layers with \( \mathcal{K}(\cdot) \)) to obtain a deeper neural network, \( F'(x, \Theta) = \mathcal{K}(y) \), the resulting network should not perform any worse than its shallower version \( F \). In the edge case scenario where the shallower network has managed to obtain the best possible solution to the problem at hand, the newly added layers should converge to behave as an identity function, i.e., \( \mathcal{K}(y) = y \). However, as mentioned by He et al., this is not what is observed in practice. This phenomenon is known as the degradation problem [28]. He et al. claim current optimizers cannot derive the identity function in very deep DNN.

Therefore, He et al. defined a residual mapping to emphasize the identity mapping, as shown in Figure 3.1. This led to state-of-the-art performance and made it possible to use more profound neural networks without decreasing the performance [28]. Formally the residual block is presented in Equation 3.1.

\[
y = H(x) = \text{ReLU}\{R(x) + x\}.
\]  

(3.1)

Here \( x \) is the input to the residual block, \( y \) is the output of the residual block, and \( R \) is the residual block itself.

Even though in Figure 3.1 we denoted layers inside the residual as ‘Weight layer,’ the idea is also applicable for convolutional layers [28]. The input \( X \) (now using uppercase because we assume the input is a matrix) can go through a residual part consisting of convolutional and pooling layers and then be summed with itself. However, observe that Equation 3.1 is only valid if the shape of \( R(X) \) is the same as \( X \). If this is not the case, the identity mapping must be slightly tweaked. In [28], the authors use so-called projection

![Figure 3.1: Residual connection building block [28.](image)]
Figure 3.2: Architecture of ResNet-18. ’18’ comes from having 18 layers. The solid lines denote identity mapping, and the dotted lines are projection shortcuts. 3 × 3 conv, 64, /2 denotes a convolutional layer with filter size 3 × 3, number of filters 64, and stride 2. If ’/2’ is not given, the stride is 1. The ReLU activation function is applied after each block [28].

shortcuts when the dimensions between \( R(X) \) and \( X \) differ.

In ResNet, the shape mismatch is usually caused when the first weight layer, see Figure 3.1, is a convolutional layer with stride 2 and a number of kernels twice as the input. Hence, the height and width of \( R(X) \) are half of \( X \), and the channel number is twice of \( X \) when a shape mismatching happens in ResNet. Then the projection shortcuts are done by convolutions with kernel size 1 × 1 with stride 2 and kernel numbers twice the channel size of \( X \).

There are different sizes of ResNet, and each denoted according to ResNet-x notation, where x is the number of layers used in the model. In this work, we focus on ResNet-18, and its architecture is given in Figure 3.2. The reason for focusing on ResNet-18 is its adequate performance and being computationally light compared to its bigger versions. If the size of ResNet is not specified in this report, it refers to ResNet-18. Observe in Figure 3.2 that the dotted lines represent a shape mismatch. Therefore, a projection shortcut is applied. The solid lines do not have shape mismatching, so identity mapping is applied. Identity mapping does not introduce new parameters, i.e., it does not increase the complexity of the model.

### 3.2 MobileNetV2

Most state-of-the-art CNNs are computationally heavy architectures unsuitable for many real-time applications. For example, in self-driving cars, gaze
estimation in virtual reality and driver tracking, mobile applications, robotics, etc., the inference time should not exceed some predetermined thresholds. Therefore, there is a need for CNNs that are fast and memory efficient while giving good results. This was the exact reason why MobileNet, in the first case, was introduced [54]. MobileNet is an efficient CNN adapted for mobile and embedded vision applications [54]. However, shortly after the first version of MobileNet was published, a second version of it was published in [27]. This was called MobileNetV2 and surpassed MobileNetV1 both in terms of accuracy and time complexity [27]. Hence, making MobileNetV2 one of the superior CNN for mobile applications. In continuation, if the version of MobileNet is not given, it refers to the second version, i.e., MobileNet = MobileNetV2.

The architecture of MobileNetV2 is given in Table 3.1. Here we see that the network consists of serialized blocks called ‘bottleneck’ beyond the usual convolutional and average pooling operations explained in chapter 2. These bottlenecks consist of three operations:

- A $1 \times 1$ convolutional layer followed by a ReLU6 activation function. This mainly increases the number of channels of the feature map.

- A depthwise convolution (see section 2.1.1) with stride $s$ followed by a ReLU6 activation function.

Table 3.1: MobileNetV2 architecture. $n$ denotes how many times each block is repeated, $t$ denotes the expansion ratio inside the bottleneck layers (see Table 3.2), $c'$ denotes the number of output channels of each block, and $s$ denotes the stride used in the depthwise separable convolutional layer in the first sequence [27].

<table>
<thead>
<tr>
<th>Operator</th>
<th>$t$</th>
<th>$c'$</th>
<th>$n$</th>
<th>$s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>-</td>
<td>32</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>bottleneck</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>24</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>32</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>64</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>96</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>160</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>bottleneck</td>
<td>6</td>
<td>320</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>conv $1 \times 1$</td>
<td>-</td>
<td>1280</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>avgpool $7 \times 7$</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>conv $1 \times 1$</td>
<td>-</td>
<td>k</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 3.2: Bottleneck block used in MobileNetV2. It transforms the input activation layer of shape $h \times w \times c$ to $h \times w \times c'$. Here $t$ is the expansion ratio and $s$ is the stride used in depthwise separable convolution [27].

<table>
<thead>
<tr>
<th>Input dimension</th>
<th>Operator</th>
<th>Output dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h \times w \times c$</td>
<td>$1 \times 1$ conv, ReLU6</td>
<td>$h \times w \times (tc)$</td>
</tr>
<tr>
<td>$h \times w \times (tc)$</td>
<td>$3 \times 3$ dwise conv $s=s$, ReLU6</td>
<td>$h/s \times w/s \times (tc)$</td>
</tr>
<tr>
<td>$h/s \times w/s \times (tc)$</td>
<td>linear $1 \times 1$ conv</td>
<td>$h/s \times w/s \times c'$</td>
</tr>
</tbody>
</table>

- A $1 \times 1$ convolutional layer without any activation function [27]. This is the pointwise convolution introduced in section 2.1.1.

See Table 3.2 for more detail.

These bottlenecks are called 'Inverse residual blocks' and, at the same time, 'Linear bottlenecks' [27]. They are called 'Linear bottlenecks' because the last convolutional layer has no activation function. The reason for not having an activation function, as explained by Sandler et al. [27], is that when going from an activation layer with a bigger channel size to a smaller one ($tc > c'$) with $1 \times 1$ convolutional layer followed by a ReLU activation function (or ReLU6) information gets lost. This makes the ReLU non-invertible, making it difficult for the model to learn. Therefore, the last convolutional layer in the bottleneck does not have a non-linearity, i.e., the last layer in the bottleneck is linear. See Appendix A for a more detailed explanation.

The bottlenecks are called 'Inverse residual blocks' because they do the opposite of what deeper ResNets do (see section 3.1). In other words, it expands the input with a factor of $t$ (see Table 3.1 and 3.2) and applies depthwise separable convolution on the expanded activation layers. Furthermore, as opposed to deeper ResNets, the residual connection is between the 'shallower' layers. All of these explain why the bottleneck used in MobileNetV2 is called the 'Inverse residual block.' These Inverse residual blocks make the model more memory efficient and faster [27].

3.3 Transformers

The Transformer is a relatively new deep learning architecture, mainly designed for NLP applications by Vaswani et al. [42]. Current state-of-the-art NLP approaches such as BERT (Bidirectional Encoder Representations from Transformers) [55] and GPT (Generative Pre-training Transformer) [56] are based on large Transformers. This section explains the architecture of a
Figure 3.3: The overall architecture of a Transformer consists of an encoder (left block) and a decoder (right block). In this example, the encoder and decoder consist of \( N_e = 3 \) and \( N_d = 3 \) identically stacked layers, respectively.

Transformer as it is used in NLP applications. The following section discusses how Transformers can be applied to visual inputs.

The Transformer differs from previous DNN architectures because it is exclusively based on attention layers, most of which are self-attention layers [42]. Self-attention layers allow an input sequence (no matter how long) to interact and exchange information with all of its parts simultaneously. For example, assume the sentence 'I love my dog.' Then, in a self-attention layer, the word 'I' interact with all the other words (including itself) simultaneously, i.e., with 'I,' 'love,' 'my,' and 'dog.' In this way, the Transformer differs from all previous state-of-the-art NLP approaches, such as Long Short-Term Memory (LSTM) [57] and Gated Recurrent Unit (GRU) [58].

A Transformer consists of two parts: an encoder and a decoder, as shown in Figure 3.3. The encoder and the decoder consist of \( N_e \) and \( N_d \) identically stacked layers, respectively. Even though this does not need to be the case in Figure 3.3 \( N_e = N_d \). Each stacked layer is called an 'encoder layer' if it is in the encoder and a 'decoder layer' if it is in the decoder. Inside the encoder/decoder block, the output of one encoder/decoder layer \((E_i/D_i, \text{ where } i \text{ is the layer number})\) is fed as input to the next encoder/decoder layer.

All of these intermediate layers’ outputs are of the same size as the input, i.e., \( \{X, E_i, Z, D_i\} \in \mathbb{R}^{n \times d_{\text{model}}} \) (ignoring the batch size). Here \( n \) is the input
sequence length (four in the example ‘I love my dog’), and $d_{\text{model}}$ is the number of features in the encoder/decoder inputs. In other words, $d_{\text{model}}$ is the length of the numerical vector obtained via a look-up table for each word in the input and is called the hidden dimension of the model. The output of the encoder, $Z$, is used as input to all the decoder layers; see Figure 3.3. This allows every position in the decoder to pay attention to every position of the input sequence [42]. The inside of an encoder and a decoder layer have been depicted in Figure 3.4a and 3.4b, respectively, and explained next.

![Figure 3.4: Detailed architecture of an encoder layer (left) and a decoder layer (right).](image-url)

(a) There are two sub-layers inside an encoder layer: a Multi-Head Attention mechanism and a 2-layer MLP followed by layer normalization. Each of these sub-layers uses residual connections (the black arrows).

(b) There are three sub-layers inside a decoder layer. The first and last ones are the same as the encoder’s sub-layers (blocks 1 and 2). The third sub-layer is a Multi-Head Attention layer, where the keys and values come from the encoder, and the queries come from the previous decoder layer.
3.3.1 Encoder Layer

An encoder layer takes an input $E_i \in \mathbb{R}^{n \times d_{model}}$ and obtains query $Q \in \mathbb{R}^{n \times d_k}$, key $K \in \mathbb{R}^{n \times d_k}$, and value $V \in \mathbb{R}^{n \times d_v}$ matrices from it, respectively. Here $d_k$ is the dimension of query and key values, and $d_v$ is the dimension of value values. The $Q$ matrix contains the query values for each token in the sequence. That is, the $i$:th row of $Q$ contains the query corresponding to the $i$:th token in the sequence, $q_i$. Hence, the row number is $n$. The same logic is also valid for $K$ and $V$ matrices.

To obtain $Q$, $K$, and $V$ matrices, projection matrices $W_Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_V \in \mathbb{R}^{d_{model} \times d_v}$ are applied to the input $E_i$. In other words

$$Q = E_iW_Q, \quad K = E_iW_K, \quad V = E_iW_V.$$  \hfill (3.2)

Then these query, key, and value matrices are fed through a Multi-Head Attention layer, which is explained in detail in section 3.3.3. However, the output of this layer is of shape $n \times d_{model}$. This allows the residual network to easily add the output with the input. Then we use a layer normalization as explained in section 2.6.

The second block in the encoder layer consists of a 2-layer MLP such as

$$\text{MLP}(P_e) = \text{ReLU}(P_eW_1 + b_1)W_2 + b_2.$$  \hfill (3.3)

Here $P_e$ is the output of block 1 in the encoder layer and is of shape $n \times d_{model}$. Furthermore, $W_1 \in \mathbb{R}^{d_{model} \times h_{mlp}}$ and $W_2 \in \mathbb{R}^{h_{mlp} \times d_{model}}$ where $h_{mlp}$ is the hidden MLP dimension. Not using any activation function in the second step is intentional and can be explained with the Lemma given in Appendix A (also see section 3.2). MLP is applied to each position separately and identically [42]. This means that each token in the sequence (each row in $P_e$) is exposed to the same weights $W_1$ and $W_2$ in the same layer. However, these weight matrices differ between different layers in the encoder/decoder. Finally, the 2-layer MLP output is layer normalized after being added to itself. This creates an output $E_{i+1} \in \mathbb{R}^{n \times d_{model}}$.

3.3.2 Decoder Layer

A decoder layer is built similarly to the encoder layer but with a new block added between blocks 1 and 2; see Figure 3.4b. This new block in the decoder is a so-called ’Encoder-Decoder Multi-Head Attention layer.’ Because it obtains the keys $K$ and values $V$ from the last encoder layer, and the queries
Q come from the previous decoder block. Here again, a residual connection together with layer normalization is used.

In the Encoder-Decoder Multi-Head Attention layer, observe that Equation 3.2 is tweaked such that $E_i$ is replaced with the output from the first block in the decoder layer (let us call it $M_d$) corresponding $Q$ and $E_i$ is replaced by $Z$ (see Figure 3.3) corresponding $K$ and $V$. That is, we have

$$Q = M_d W_Q, \quad K = Z W_K, \quad V = Z W_V. \quad (3.4)$$

Furthermore, all $E_i$'s are replaced by $D_i$'s in Equation 3.2 in the first block in the decoder layer.

### 3.3.3 Multi-Head Attention

The Multi-Head Attention layer is depicted in Figure 3.5. Observe here that the $Q_i, K_i, \text{ and } V_i$ all have been obtained via Equation 3.2 (or Equation 3.4), where different $W^i_Q, W^i_K, \text{ and } W^i_V$ matrices have been applied to the input. The main part of a Multi-Head Attention layer is the Scaled Dot-Product Attention layer. The following equation mathematically explains the Scaled Dot-Product Attention layer.

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V. \quad (3.5)$$

![Figure 3.5: The Multi-Head Attention layer [42].](image-url)
Here $d_k$ is the scaling factor mentioned above. This scaling factor is sometimes referred to as the *temperature* [59]. This temperature prevents the Softmax from having a small gradient in backpropagation [24].

The output dimension of the Scaled Dot-Product Attention layer is $\text{Attention}(Q, K, V) \in \mathbb{R}^{n \times d_v}$ following the dimension notations given in section 3.3.1. However, Vaswani et al. [42] used $h$ of these attention heads in parallel, as shown in Figure 3.5. The reason for having multiple parallel heads is that when only using one attention layer, the current word in focus can put too much weight on itself [42]. However, when there are other parallel Scaled Dot-Product Attentions, the current word in focus can try to find a connection/link with other words. Take, for example, the sentence, ‘I love my dog, he is smart.’ Here, ‘he’ can be associated with ‘I’ and the ‘dog.’ If we only had one Scaled Dot-Product Attention, the word ‘he’ would put too much importance on itself since this word is the word in focus. However, with parallel branches, the word ‘he’ can put some importance to the word ‘dog,’ to which it actually refers. Hence, these Scaled Dot-Product Attention layers are combined to form the Multi-Head Attention layer, which has been mathematical formulated in Equation 3.6.

$$\text{Multi-Head}(Q, K, V) = \text{Concatenate}(\text{head}_1, \ldots, \text{head}_h) W_O$$

where $\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$ according to Equation 3.5. \hspace{1cm} (3.6)

Here the concatenation is done such that the shape of the concatenation is $n \times h d_v$. Then by having a matrix $W_O \in \mathbb{R}^{h d_v \times d_{model}}$, we can obtain an output of the Multi-Head Attention layer that is the same shape as the input. This makes the residual connection easy.

### 3.3.4 Pre-processing and Post-processing

In Figure 3.3, observe that we have stated the inputs to the encoder and decoder as *Embedded*. This is because the words in NLP applications need to be pre-processed before they can be fed into a Transformer. This is done by a look-up table. With this look-up table, each word is mapped to a *numerical* vector of size $d_{model}$.

After embedding Vaswani et al. [42] also suggest adding a positional encoding in order to let the model know where in the sequence the token actually is located. Vaswani et al. show that this increases the performance of the model. In [42], they use sine and cosine functions of different frequencies for the positional encoding.
Post-processing is the last process done by the model. Post-processing is located after the decoder. In other words, the input to the post-processing box is the output of the decoder $D_{N_d}$. The post-processing part is adapted to the problem at hand, while the rest of the Transformer is more or less similar for different applications.

### 3.4 Vision Transformer

Transformers’ success in NLP applications has led to this idea being tested on visual inputs. However, Transformers, as they are, were unsuitable for computer vision tasks. Therefore, Dosovitskiy et al. [21] proposed modifying the Transformer architecture to accept visual inputs. Below we explain the differences between the original Transformer, explained above, and the ViT as introduced by Dosovitskiy et al.

Contrary to Transformers, ViTs only use the encoder block. That is, the decoder shown in Figure 3.3 is not used in the ViT architecture. Another difference in ViTs compared to the Transformers is that there are no look-up tables as it is for words. The *Embedding* vectors must be obtained in another way.

In [21], the authors proposed to parse the input image to patches of shape $P \times P \times C$ as shown in Figure 3.6. Here $P$ is the patch size, and $C$ is the channel size of the input image. Then each patch is flattened into vectors of shape $P^2 \cdot C$.

![Figure 3.6: The overall architecture of a ViT [21].](image)
Observe that we will obtain \( n = \frac{H}{P} \cdot \frac{W}{P} \) of these vectors. That is from an input image that is of shape \( \mathbf{X}_{\text{img}} \in \mathbb{R}^{H \times W \times C} \), we will obtain a flattened 2D matrix of shape \( \mathbf{X}_{\text{flat}} \in \mathbb{R}^{n \times P^2 \cdot C} \). Then this matrix is fed through a linear projection layer, shown in Equation 3.7, which is responsible for obtaining tokens that the Encoder can accept, as explained in the previous section.

\[
\text{Embedded input to the Encoder} = \mathbf{X} = \mathbf{X}_{\text{flat}} \mathbf{W}_E \tag{3.7}
\]

where \( \mathbf{W}_E \in \mathbb{R}^{P^2 \cdot C \times d_{\text{model}}} \).

After the patches have been flattened and linearly projected, a class token is added in front of every input example (see the blue vector in Figure 3.6). Even though this was not used in the Transformer as introduced by [42], it was used in later Transformer architectures such as BERT [55]. This added class token showed improved performance on NLP tasks and therefore was also used in ViT. Moreover, this class token is a learnable parameter and is concatenated to the input sequence making the input of shape \( \mathbf{X} \in \mathbb{R}^{(n+1) \times d_{\text{model}}} \). Before the visual tokens are sent into the Transformer Encoder, we add positional embeddings as in the original Transformer. Dosovitskiy et al. [21] use simple 1D position embeddings since they show that 2D positional embeddings do not improve performance.

After the sequence has fed through the encoder block, we obtain the output \( \mathbf{Z} \in \mathbb{R}^{(n+1) \times d_{\text{model}}} \). However, usually, we want to predict a class or a value for the problem in hand (e.g., gaze direction). Therefore, the vector corresponding to the class token is taken (leaving us with a vector of shape \( d_{\text{model}} \times 1 \)) and fed through an MLP layer to obtain the outputs (see Figure 3.6).

In ViT also, the inside of an encoder layer is slightly changed. As shown in Figure 3.4a, the layer normalization in the original Transformer is applied after the Multi-Head Attention layer and MLP layer. However, in ViTs, the order is flipped. That is, layer normalization is applied before the Multi-Head Attention and MLP layers. Furthermore, the layer normalization in ViTs is inside the residual connection.

Another detailed change in ViTs is that the GELU activation function is used instead of the ReLU activation function in the 2-layer MLP, shown in Equation 3.3. Furthermore, Dosovitskiy et al. use dropout after each Multi-head Self-Attention layer and MLP layer in the encoder layer. These dropouts are inside the residual connections. This is to regularize the model and prevent overfitting, as explained in section 2.5.
3.5 Differences Between CNNs and ViTs

Although there have been explanations about the differences between CNNs and ViTs, this section will provide a more straightforward outline of the distinctions between these architectures. The main distinction between the two is that CNNs process the input locally, while ViTs process the input globally. In other words, different patches in a ViT are consistently interacting with one another, whereas in a CNN, a kernel moves over the input, and the output is only affected by the pixels in its neighboring area. Due to this global processing, ViTs require more data than CNNs.

One key difference between CNNs and ViTs is that CNNs rely on convolutional kernels, while ViTs use attentions (as explained in section 3.3.3). Multi-Head Attention is the heart of a ViT’s global processing, while the convolutional kernels are the heart of a CNN’s local processing. Additionally, CNNs are generally less affected by choice of optimizer, hyperparameters, and training schedules than ViTs, making them more robust [60]. However, despite this, ViTs have shown to outperform CNNs, but typically only when pretrained on large datasets. Since we do not have access to that much data in this project, we will also explore hybrid ViTs, which is explained next.

3.6 Hybrid Vision Transformer

A hybrid ViT is an architecture where CNNs are combined with ViTs. As discussed in the original paper [21], the ViTs outperform CNNs only when pretrained on enormous datasets. Dosovitskiy et al. show that ViTs only surpass the performance of state-of-the-art ResNets when pretrained on JFT-300, which contains over 300 million images [22]. On the other hand, when ViTs are pretrained on smaller datasets such as ImageNet [61], ResNet outperforms ViTs [21]. This has been explained by the local processing inherent in CNNs being replaced by ViTs’ global processing [60]. A global processing network requires more data since it does not have local inductive biases. Higher local inductive biases make the model perform better on smaller datasets [60].

Therefore, to get competitive results with state-of-the-art CNNs, inductive biases in ViTs must be increased. Much research has done this by placing convolutional layers before a Transformer, as shown in Figure 3.7 [6, 60, 62]. The feature maps used as input to the ViT are now obtained by convolutional layers instead of hard-coded patches obtained from image parsing. By this,
Figure 3.7: The architecture of a hybrid ViT. The features to ViT are obtained via early convolutions.

the sharp border between the different patches disappears. In other words, the early convolutions are responsible for obtaining local information and sending this local information in the right way to the feature maps inputted to the ViT. On the other hand, the ViT is responsible for finding global relations between different parts of the image. In this sense, the ViT replaces the late FC layers (see section 2.3) in CNNs.

It has been shown that using convolutions before the ViTs improves the performance in general tasks. Furthermore, Cheng et al. [6] showed that hybrid ViTs outperform state-of-the-art CNNs on gaze estimation tasks, while pure ViTs cannot achieve competitive results. [60] showed that using hybrid ViTs can make the neural network more stable and easier to optimize.

3.7 Advanced Methods in Vision Transformers

Since the first paper on ViTs, many ideas have been proposed to improve the performance, decrease the inference time and make ViTs effective on small datasets. Some of them are LeViT [63], DeiT [23], LSA and Shifted Patch Tokenization (SPT) ideas [24], T2T ViT [64], and Mobile-Former [26]. Some of these are explained in the following to understand this work better.

3.7.1 Locality Self-Attention Mechanism

In [24], Lee et al. proposed two new methods; SPT and LSA. These methods have been shown to improve the performance of pure ViTs. However, as shown in chapter 6, hybrid ViTs consistently outperform pure ViTs. Therefore, we mainly looked into ideas to improve hybrid ViTs. The SPT is a method that is highly adapted for pure ViTs [24]. Therefore, we do not go through this method here.
On the other hand, we hypothesized that LSA could positively impact hybrid ViTs because LSA increases inter-token attention and prevents intra-token attention \cite{24}. This is done by masking the diagonal after the matrix multiplication $QK^T$ before Softmax is applied in Equation 3.5 (see \cite{24} for the details). We state the hypothesis above because, in a hybrid ViT, the convolutional stem should ideally extract the local information. Hence, obstructing the intra-token attention in the ViT head could improve the performance by focusing more on inter-token attention.

Another detail in LSA is to use learnable temperature scaling. The idea here is to learn the temperature in Equation 3.5 instead of predetermining it as $\sqrt{d_k}$. According to Lee et al., the learnable temperature scaling sharpens the attention scores. Therefore it should improve the performance. In other words, Equation 3.5 becomes

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{\text{LSA-filter}\{QK^T\}}{\tau}\right) V, \quad (3.8)$$

where $\tau$ is a learnable parameter.

### 3.7.2 Data-Efficient Image Transformers

Another interesting idea presented for pure ViTs is to add a distillation token, which in some sense plays the same role as the class token \cite{23}. The distillation token is added in the same way as the class token (see section 3.4). This increases the input shape from $n+1$ to $n+2$, i.e., $X \in \mathbb{R}^{(n+2) \times d_{\text{model}}}$. The duty of this distillation token is to reproduce the label estimated by a teacher network as shown in Figure 3.8. The interaction of the distillation token in each encoder layer is exactly the same as the class token. Hence, the distillation token also interacts with all the input tokens and should successfully be able to estimate correct values.

The student network is the network in focus, and the goal is to improve this network. In \cite{23}, Touvron et al. use a pure ViT as the student network. Furthermore, they show that using a CNN-based teacher network improves the performance of the student network (when it is a pure ViT) more than using a Transformer-based teacher network. This is because the inductive bias present in the CNN-based teacher network is baked into the ViT-based student network by having a distillation token. Hence, the shortcomings of a student network are completed by a teacher network. In this work, we do not use the distillation token. However, we originate from this idea. The details for the inspiration are explained in section 5.2.4.
3.7.3 CNN-Former

CNN-Former is a hybrid ViT, i.e., it consists of both CNNs and Transformers (hence the name 'Former'). However, unlike the hybrid ViT explained in section 3.6, the CNN and ViT are not placed consecutive but rather parallel in the CNN-Former [26]. Several two-way bridges connect the CNN part and the Transformer part. These bridges transfer local information from the CNN-blocks to the Transformer-blocks (see the Local to Global (L2G)-block) and vice versa (see the Global to Local (G2L)-block) as shown in Figure 3.9 [26].

The CNN-block takes as input a feature map and applies convolutional layers as a regular CNN, as explained in chapter 2. Furthermore, the Transformer-block is an encoder layer (see section 3.3) and takes a couple of tokens as input and applies Multi-Head Attention together with a Feed-Forward Network. Hence, the Transformer is responsible for global attention. Interestingly in a CNN-Former, the tokens are not obtained by parsing the input image. The tokens are learnable parameters, and Chen et al. showed that the number of tokens can be as small as 1 but gives better results when set to 3 or 6. This keeps the computational complexity down and makes the CNN-Former a strong candidate for the problem at hand.

The L2G-block is responsible for fusing the local features $X$ into the global tokens $Z$ using a lightweight Multi-Head Attention layer [26]. It is lightweight
because the weight matrices corresponding to the local features are removed. The residual connection is between the input tokens and the resulting output. Similarly, the L2G-block is also a lightweight Multi-Head Attention layer. However, the residual connection is between the from the CNN-block received feature maps and the resulting output. We refer to [26] for a more detailed explanation.

Observe the red arrow in Figure 3.9. If a Dynamic ReLU is used in the CNN-block, the output from the Transformer-block, defines its parameters.

Figure 3.9: Illustration of CNN-Former block.

Figure 3.10: A complete CNN-Former.
However, if a non-Dynamic ReLU is used in the CNN-block, this connection can be removed, and the process can be parallelized more effectively.

As shown in Figure 3.10, the CNN-Former blocks are stacked on top of each other to obtain the complete CNN-Former. Observe that the local and global information is exchanged several times during the inference. As a last step, a Classifier is defined. This classifier combines the output from the CNN and Transformer parts to obtain the estimates.

3.8 Summary

This chapter showed how convolutional, pooling, and FC layers (introduced in chapter 2) are used in ResNet and MobileNet. Furthermore, we have explained the importance of residual connections in ResNet and seen that these kinds of residual connections recurrently appear in other architectures, such as Transformers.

After that, in section 3.3, we explained the different modules in a Transformer and how a Transformer, as it is used in NLP, differs from a ViT. We underlined that local inductive biases in Transformers are replaced by global attention, leading them to need more data than CNNs. This is the most significant difference between CNNs and ViTs. Therefore, we introduced the hybrid ViT, a mix of a CNN and a ViT.

The chapter ends with state-of-the-art methods presented for pure and hybrid ViTs. These methods, in one sentence each, have been summarized in Table 3.3.

Table 3.3: The different methods introduced for improving the performance of pure and hybrid ViTs that are of importance to this work.

<table>
<thead>
<tr>
<th>Method</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid ViT</td>
<td>Combines CNNs with ViTs in a consecutive manner.</td>
</tr>
<tr>
<td>LSA method</td>
<td>Removes the intra-token attention in the Transformer.</td>
</tr>
<tr>
<td>DeiT</td>
<td>Introduces a distillation token and teacher/student networks.</td>
</tr>
<tr>
<td>CNN-Former</td>
<td>A hybrid ViT where the CNN and Transformer are placed in parallel.</td>
</tr>
</tbody>
</table>
Gaze estimation is the art of estimating where a person is looking. In order to estimate the gaze as accurately as possible, some pre-processing and post-processing techniques have been suggested over the years. This chapter explains these ideas and focuses mainly on the pre-processing and post-processing techniques used in this work.

4.1 Data Pre-Processing

All the information needed to estimate the gaze can be found in a person’s face. The surrounding area, on the other hand, contains minimal information about the gaze. For example, the gaze direction is correlated with a person’s eyes appearance, head position, eyelids’ shape, etc. Hence, by cropping the background, the amount of data needed to obtain a good model can be reduced, and having smaller training images can improve the model’s efficiency [9]. Hence, data normalization has shown to be a good technique for making the training more efficient and getting better results with smaller datasets [8, 1, 16, 19, 18, 10, 20]. In some works, only the eyes are cropped [16, 19, 18] and sent to the neural network. In other works, the whole face is sent into the neural network [18, 10, 20]. This thesis uses the whole face as input since it provides much more information than only using the eyes [1].

We follow the data normalization process presented in [9] and [65]. This data normalization is depicted in Figure 4.1. To give an overview of this process, we define four coordinate systems; Camera Coordinate System (CCS), Head Coordinate System (HCS), Normalized Camera Coordinate System (NCCS), and Rotated Camera Coordinate System (RCCS) (see Figure 4.1 and 4.2a).
(a) Rotate the CCS so that the \( z \)-axis of it points to the center of the HCS, the red coordinate system. The dashed-gray coordinate system represents the CCS.

(b) Rotate again so that the \( x \)-axis of the HCS lies inside the \( xz \)-plane of the CCS. The red coordinate system represents the RCCS.

(c) Scale the RCCS to obtain the NCCS. This is done by the scaling matrix \( S \). The red coordinate system represents the NCCS.

Figure 4.1: Data normalization. This process transforms an image into a normalized image. When looked through the normalized camera, the person’s face appears in the middle of the image and covers the whole image as shown in Figure 4.2b.

HCS is defined such that the origin is the eyes’ middle point, shown with \( O_{HCS} \) in Figure 4.2a. Moreover, the \( x \)-axis points out from the left ear, the \( y \)-axis points up to the forehead, and the \( z \)-axis points forward (i.e., in the gaze direction), as shown in Figure 4.2a [65].

CCS takes the camera as a reference. The camera is located at the origin of this coordinate system. In CCS, the \( x \)-axis points to the camera’s right, the \( y \)-axis points downwards, and the \( z \)-axis points forward (see Figure 4.1a). The idea is to obtain a NCCS (see Figure 4.1c). When looking through this normalized camera, the face of the person should be centered, and most of the background should be cut off [9, 65]. This process is shown in Figure 4.2b.

To obtain this normalized image, we first need to rotate the original camera space with a matrix \( R \). How the matrix \( R \) is obtained can be found in [9]. This matrix \( R \) ensures that the RCCS’s \( z \)-axis points to the origin of the HCS (see \( z'' \) in Figure 4.1b) and the \( x \)-axis of the HCS lies inside the \( xz \)-plane of RCCS (see \( x'' \) in Figure 4.1b) [9]. Furthermore, to go from the RCCS to the NCCS, a scaling matrix \( S \) needs to be defined. Again, we refer to [9] to see how this scaling matrix is obtained. Then the total translation matrix from the CCS to the NCCS is given by \( M = SR \). This means a point in the original CCS (\( p_{CCS} \))
Equation 4.1 transforms a point in CCS to a point in the NCCS. However, by default, the input image is not in the CCS. Therefore, it must first be transformed from image coordinates to CCS coordinates. This can be done by applying the inverse of the intrinsic camera matrix to the image [66]. An intrinsic camera matrix describes the mapping from 3D points in the original camera space to 2D points in an image for a pinhole camera [66]. Let us denote the intrinsic camera matrix for the original camera with $\mathbf{C}_o$ and the normalized camera with $\mathbf{C}_n$. Then an image is transformed into CCS by

$$
\mathbf{p}_{CCS} = \mathbf{C}_o^{-1}\mathbf{p}_{img}.
$$

Now, the image can be transformed into NCCS by the matrix $\mathbf{M}$ we derived above. Finally, the 3D points in the NCCS must be transferred to a 2D normalized image. This is done by $\mathbf{C}_n$. In other words, an arbitrary image captured by the original camera can be transformed into a normalized image (seen as captured by the normalized camera) via

$$
\mathbf{p}_{normalized_{img}} = \mathbf{C}_n\mathbf{M}\mathbf{C}_o^{-1}\mathbf{p}_{img} = \mathbf{Wp}_{img}.
$$
where \( W = C_n M C_o^{-1} \) and is called the transformation matrix. Hence, an arbitrary image can be normalized such that we obtain images shown in Figure 4.2b by a simple transformation matrix \( W \). The normalized images are then sent into the DNN.

### 4.2 Data Post-Processing

This work only sends the normalized image to the neural network. Therefore, the network cannot estimate the gaze direction in the CCS where the ground truth gaze directions (\( g_{gt} \)) are defined \([9, 1]\). Therefore, we interpret the raw output from the backbone as the gaze direction in the NCCS. In order to be able to compare the ground truth and the estimated gaze, we need to transform the estimated gaze vector from NCCS to CCS (another alternative would be to transform the ground truth gaze directions from CCS to NCCS). This is done similarly to the process explained in the previous section. However, Zhang et al. \([9]\) showed that including the scaling factor (\( S \)) when transforming the gaze direction harms the performance. Therefore, they suggested only rotating the estimated gaze direction (\( \hat{g}_n \)) given in NCCS into the CCS by

\[
\hat{g}_o = R^{-1} \hat{g}_n. \tag{4.4}
\]

Now the ground truth gaze directions \( g_{gt} \) can be compared with the estimated gaze directions \( \hat{g}_o \) in a proper way such that the network can learn valuable information.

The estimated gaze direction is usually a 2D polar angle vector, i.e., \( \hat{g}_n \in \mathbb{R}^2 \) \([65]\). That is, \( \hat{g}_n \) contains the pitch (\( \theta \)) and yaw (\( \phi \)) angles of the 3D gaze vector. Here, we define the pitch as the angle between the \( xz \)-plane and the gaze direction; see Figure 4.3. Yaw is the angle between the \( z \)-axis and the

![Figure 4.3: Illustration of pitch and yaw angles in gaze estimation.](image)

\[ y-axis \]
\[ x-axis \]
\[ z-axis \]
\[ Gaze \]
\[ direction \]
\[ \theta \]
\[ \phi \]
projection of gaze on the $xz$-plane. The 3D gaze vector can be obtained by the following equations given the pitch and yaw angles.

\[
x = \cos(\theta) \sin(\phi) \\
y = \sin(\theta) \\
z = \cos(\theta) \cos(\phi)
\] (4.5)
Chapter 5

Methods

5.1 Model

The model takes a raw image as input, shown to the left in Figure 4.2b. Then the model normalizes it according to section 4.1 to obtain a centralized face image, as shown to the right in Figure 4.2b. The obtained normalized image is of shape $256 \times 192$ and is fed through a DNN-based backbone. The backbone is the main focus of this work, and, therefore, it is explained in more detail in section 5.2. Since this is the case, the pre- and post-processing are not changed throughout this thesis.

We replace the CNN-based backbone with pure and hybrid ViTs in order to see which DNN architectures are more suitable for gaze estimation tasks. Identical inputs are used in all these DNN architectures. That is, the data augmentation techniques, pre-processing and post-processing techniques, input dimension, etc., are the same for all the experiments. However, we do not use a constant seed; therefore, the samples are trained in different orders, and the data augmentation for each sample can vary, which can affect the results. Nevertheless, these effects should be minimal and have been considered in chapter 6 when presenting the results. Moreover, in all the experiments, the output format from the backbone is the same.

The backbone takes only the normalized image of shape $256 \times 192$ as input and returns raw estimates for the gaze direction (among other things). Here raw emphasizes that, for example, the gaze direction is not calculated directly in the CCS as a 3D vector but a 2D vector (containing the pitch and yaw angles) in NCCS, as discussed in section 4.2. Then these raw estimates are transferred to actual predictions in CCS so that the predictions can be compared to the ground truth values.
The comparison is made through a loss function such that the weights are updated correctly, as explained in section 2.7. We used a loss function that minimizes the angular error between the predicted and ground truth gaze directions in CCS (among other things). The angular error is defined in section 5.5.

5.2 Backbone

Even though pre-processing and post-processing steps are essential, correct estimation of gaze is highly dependent on the success of the backbone. Therefore, this work focuses on the backbone. Specifically, we build different backbones based on pure and hybrid ViTs and CNN-Formers and compare these with the state-of-the-art CNNs such as ResNet and MobileNet. Hence, this section explains how ResNet and MobileNet have been modified to accept NIR images. Furthermore, this section explains the different pure and hybrid ViT architectures and advanced models and training algorithms used in this work. Each model’s performance is later presented and discussed in chapter 6.

5.2.1 Convolutional Neural Networks

This work uses two different CNNs. The first one is ResNet (referring to ResNet-18), and the second one is MobileNet (referring to MobileNetV2). However, both ResNet and MobileNet were constructed for ImageNet [61], which consists of RGB images. On the other hand, in this work, we use NIR images which consist of only one channel. Therefore, ResNet and MobileNet were both modified to accept NIR images as input and be used for a regression task instead of a classification task.

This was achieved by replacing the first convolutional layer, which accepted a tensor map of three channels, $C_i = 3$, with a convolutional layer that accepts a tensor map of one channel, $C_i = 1$. The kernel size, padding, stride, and the number of output channels were the same as in the original versions in this first convolutional layer. Furthermore, since we are not dealing with a classification task, the Softmax activation function placed at the end was removed. No other activation function was used at the last layer. These were the modifications made on both ResNet and MobileNet. In other words, the rest of the ResNet used in this work is identical to as introduced in section 3.1.

However, another modification on MobileNet compared to as explained in section 3.2 is that we replaced the ReLU6 activation function with the normal ReLU. The rest of the MobileNet was kept the same. After the modifications,
the number of parameters in ResNet and MobileNet is given in Table 5.1 when applied on an input of shape $256 \times 192$. Considering the changes made, these values match the ones given in [28, 67].

### 5.2.2 Pure Vision Transformers

This work defines five pure ViTs with different sizes and hyperparameters. The idea of having different sizes is to investigate the trade-offs between the size and the performance of a ViT. In this work, we refrain from an ablation study since it has already been done in [6] by Cheng et al. and considering the computational resources available. Therefore, here we only define three different sizes of ViTs and name them big, medium, and small, respectively. The size of a ViT is indicated in its name with the letters B, M, and S, as shown in Table 5.2. The subscript $P$ stands for ’Pure’ ViT. Later we use the subscript $H$ to identify a ’Hybrid’ ViT.

However, we define three different small ViTs. The reason for this is to see which of the hyperparameters are more important while keeping the number of parameters in a certain range. Observe that the number of parameters of the small ViTs are closer to MobileNet than ResNet. The reason for not having even smaller ViTs is justified by the empirical results in chapter 6.

All of the pure ViTs use the same patch size $P = 16$. Hence, using inputs of size $256 \times 192$ results in $n = 192$ feature vectors. However, other hyperparameters differ and are given in Table 5.2. $d_{model}$ is the feature vector

### Table 5.1: Number of parameters for ResNet and MobileNet.

<table>
<thead>
<tr>
<th>Name</th>
<th>parameters (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet-18</td>
<td>11.2</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

### Table 5.2: Pure ViTs used in this work and their hyperparameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>$d_{model}$</th>
<th>$h_{mlp}$</th>
<th>$N$</th>
<th>$h$</th>
<th>$d_k$</th>
<th>parameters (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-B$_P$</td>
<td>768</td>
<td>2048</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>38.0</td>
</tr>
<tr>
<td>ViT-M$_P$</td>
<td>384</td>
<td>2048</td>
<td>8</td>
<td>6</td>
<td>64</td>
<td>17.4</td>
</tr>
<tr>
<td>ViT-S1$_P$</td>
<td>192</td>
<td>1024</td>
<td>8</td>
<td>3</td>
<td>64</td>
<td>4.4</td>
</tr>
<tr>
<td>ViT-S2$_P$</td>
<td>256</td>
<td>1024</td>
<td>4</td>
<td>8</td>
<td>64</td>
<td>4.3</td>
</tr>
<tr>
<td>ViT-S3$_P$</td>
<td>256</td>
<td>1024</td>
<td>3</td>
<td>12</td>
<td>92</td>
<td>5.0</td>
</tr>
</tbody>
</table>
length, $h_{\text{mlp}}$ is the MLP dimension in the second block of the encoder layer, and $N$ is the depth of the encoder. This was shown with $N_e$ in section 3.3. However, since we do not have a decoder in a ViT, we remove the subscript $e$. Furthermore, $h$ is the number of heads used in the Multi-Head Attention layer, and $d_k = d_v$ is the dimensions of the $Q$, $K$, and $V$ matrices. This table also includes the number of parameters in terms of million. The number of parameters differs significantly between the big and small ViTs, while the difference is relatively small amongst the small ViTs. The hyperparameters were chosen based on Graphics Processing Unit (GPU) limitations and experimental results presented in [6].

5.2.3 Hybrid Vision Transformers

Five different hybrid ViTs are defined in this work. Three use MobileNet, and two use ResNet as the convolutional stem. The hyperparameters for the ViTs placed on top of the convolutional stems and their names are given in Table 5.3. As mentioned above, the subscript $H$ indicates that a model is a hybrid ViT. Furthermore, the subscripts $M$ and $R$ indicate that the convolutional stem is based on MobileNet and ResNet, respectively.

Hybrid ViTs with MobileNet as the convolutional stem use the MobileNet layers (with the modifications mentioned in section 5.2.1) until the last bottleneck layer (see Table 3.1). The feature map does not go through the last bottleneck layer in MobileNet, and an output of shape $8 \times 6 \times 160$ is obtained. Then a convolutional layer with a kernel of shape $(k, k, C_i, C_o) = (1, 1, 160, d_{\text{model}})$ is applied to this feature map in order to get the feature vector length as desired. This results in $n = 8 \times 6 = 48$ feature vectors, four times less than pure ViTs. After that, the patches are fed to a ViT, as specified in Table 5.3.

Hybrid ViTs with ResNet as convolutional stem use the ResNet layers (with
the modifications mentioned in section 5.2.1) until Layer 4; see Figure 3.2. The feature map outputted from Layer 3 is of shape $16 \times 12 \times 256$. Then a max pooling layer is applied to decrease the height and width of the feature map to $8 \times 6$. The reason for this is to have the same number of feature vectors ($n = 48$) as those with MobileNet convolutional stem. Finally, a convolutional layer with a kernel of shape $(k, k, C_i, C_o) = (1, 1, 256, d_{\text{model}})$ is applied for the same reason as above. Then the obtained patches are fed through a ViT.

The reason for having an input of shape $X \in \mathbb{R}^{(48+1) \times d_{\text{model}}}$ instead of $X \in \mathbb{R}^{(192+1) \times d_{\text{model}}}$ as in the pure ViT case was to decrease the computational complexity and believing the CNN stem would support the ViT. Furthermore, observe that the definition of big, medium, and small in pure and hybrid ViT differs.

5.2.4 Advanced Vision Transformers

This work uses four methods that we classify as advanced methods. These are using LSA, replacing layer normalization in Transformer with batch normalization, hybrid DeiT, and using CNN-Former.

The LSA method is applied to the model as explained in section 3.7.1, which is the exact same way as presented in [24]. In this work, we do not add anything new to this method but use it as a black box to see whether it can improve the performance of hybrid ViTs on gaze estimation tasks.

Furthermore, batch normalization in the Transformer replaces all the layer normalization. As discussed previously, layer normalization is superior to batch normalization when having inputs of different sizes. Nevertheless, using visual inputs of the same size makes batch normalization to a strong candidate to improve performance. The other two methods are explained below.

5.2.4.1 Hybrid Data-Efficient Image Transformers

As explained in section 3.7.2, in DeiT, we have a teacher and a student network where using different architectures have shown to improve the performance the most. In this work, as shown in chapter 6, hybrid ViTs consistently outperform pure ViTs. Therefore, we focus on improving the hybrid ViTs. However, hybrid ViTs already contain two different network architectures; therefore, defining a good teacher network is challenging. Hence, instead of having a teacher-student relation as presented in [23], we train a teacher network that is a pure CNN. Then we remove the last layers and add a ViT on top of it; see Figure 5.1. The resulting network can be seen as the student network. Thus, the convolutional stem has pretrained weights from when it was trained as a pure
CNN. Then, in the finetuning phase, it adapts to be a hybrid ViT. We call this architecture and training method ‘hybrid DeiT.’ We hypothesized that hybrid DeiT would learn both the architectures and bake the pure CNN ideology into the final hybrid ViT. In this sense, this idea imitates the idea presented in [23] without using a distillation token. The loss function is the same for pretraining and finetuning. Nevertheless, it could be changed.

5.2.4.2 CNN-Former

This work defines three different CNN-Formers. The first two are the MobileFormer-26M and Mobile-Former-52M, defined in [26]. This work names these two as Mobile-FormerV3-1 and Mobile-FormerV3-2 since they build on MobileNetV3. The notations ‘26M’ and ‘52M’ represent the number of multiply-adds in each model. We hesitate to use these notations because we believe these numbers are presented wrong in [26] (see section 5.6 for an explanation). These models use blocks such as Squeeze-and-Excite, dynamic ReLU, h-swish, etc. which are not used in MobileNetV2. Therefore, we refer to [26] for a more detailed description. The number of parameters for each of these models is given in Table 5.4.

The third model we use is the one that uses MobileNetV2 as the CNN part, as presented in section 3.2. Each bottleneck block in Table 3.1 corresponds to a CNN-block in Figure 3.9. Furthermore, the first convolutional layer in Table 3.1 corresponds to the ‘Stem’ block shown in Figure 3.10. The last three
Table 5.4: Number of parameters for the different CNN-Formers used in this work.

<table>
<thead>
<tr>
<th>Name</th>
<th>parameters (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile-FormerV3-1</td>
<td>3.2</td>
</tr>
<tr>
<td>Mobile-FormerV3-2</td>
<td>3.5</td>
</tr>
<tr>
<td>Mobile-FormerV2</td>
<td>8.3</td>
</tr>
</tbody>
</table>

operators in Table 3.1 correspond to the Classifier in Figure 3.10. We name this model Mobile-FormerV2 since it builds on MobileNetV2—the number of parameters for it is given in Table 5.4.

The Transformer part of Mobile-FormerV2 has 6 tokens, each with dimension $d_{\text{model}} = 192$. Furthermore, the number of heads in the L2G and G2L is 2, while the number of heads in the Transformer-block is 4. The MLP dimension is $h_{\text{mlp}} = 384$. The depth for the Transformer is $N = 17$ since there are 17 bottleneck blocks in MobileNetV2 (see Table 3.1). Contrary to the Mobile-FormerV3s explained above, Mobile-FormerV2 does not use the newly added blocks in [26], such as dynamic ReLU, Squeeze-and-Excite, etc., and the output of the Transformer is not used in the classification stage. In MobileNetV3s, the outputs from CNN and Transformer are used together in the classification stage.

5.3 Datasets

The main goal of this work is to compare the performance of different models based on CNNs and Transformers on the INIR dataset. This dataset is explained in section 5.3.1. However, based on previous research, large datasets are needed for ViTs to outperform CNNs. Therefore, we also use a pretraining dataset. This work uses the publicly available ETH-XGaze dataset, explained in section 5.3.2. The reason for using this dataset is that it is the biggest publicly available dataset for gaze estimation.

5.3.1 Internal Near-Infrared Dataset

The INIR dataset consists of images captured by NIR cameras and provided by Tobii, the host company. Hence, the images in this dataset are NIR images consisting of only one channel. NIR images are advantageous compared to RGB images in no-light, dim, and too-bright conditions. For example, a
person’s gaze can be estimated easier from NIR images at night, when it can be hard to see the pupil or even the face of the person from an RGB camera. Nevertheless, finding a proper pretraining dataset for NIR images is challenging. Therefore, this work tries pretraining on an RGB dataset, as explained in the next section.

The INIR dataset used in this work consists of 305,067 images collected from 895 participants. We partition the dataset into 175,317 training samples collected from 518 participants and 129,750 evaluation samples collected from 377 subjects. When training, these images are data augmented with different state-of-the-art techniques to increase the dataset size. Observe that none of the participants in the training dataset appears in the evaluation dataset. This is a robust way to partition the data and prevent misleading inferences caused by overfitting. After being pre-processed according to section 4.1, all the images in this dataset are of shape $256 \times 192$ and contain a person’s whole face.

5.3.2 ETH-XGaze Dataset

The ETH-XGaze dataset comprises a total of 1,083,492 images collected from 110 participants [25]. In this work, we use ETH-XGaze’s training dataset, which comprises 756,540 images collected from 80 participants, similar to [6]. Observe that the images in ETH-XGaze are 3-channel RGB images. Since the goal is to achieve state-of-the-art results on a NIR dataset consisting of only one channel, the ETH-XGaze images are converted to gray-scale images. By doing so, we do not encounter a mismatch in the first layer when transferring the pretrained model to finetuning. Nevertheless, we emphasize that the distribution of NIR and gray-scaled images is still quite different.

5.4 Implementation Details

In this work, we use multi-step and cosine learning rate schedules. The initial learning rate was set to $10^{-4}$ in the multi-step learning rate. Then, the learning rate was decreased to $10^{-5}$ and $10^{-6}$ after 85% and 95% of the epochs, respectively. This has been illustrated in Figure 5.2a.

We use a cosine learning rate with a linear warm-up of five epochs. In the linear phase, the learning rate increases from $10^{-6}$ to $1.25 \cdot 10^{-4}$, except for the CNN-Former experiments where the learning rate increases from $10^{-6}$ to $8 \cdot 10^{-4}$ following [26]. After the first five epochs, the learning rate starts decreasing following half a period of a cosine curve. The peak of the cosine
Figure 5.2: Two different learning rate schedules used in this work.

curve is $1.25 \cdot 10^{-4}$ ($8 \cdot 10^{-4}$ for CNN-Formers) and is located at epoch 5. The valley of the cosine curve is $10^{-6}$ and is located at the last epoch. This has been shown in Figure 5.2b for a training with 100 epochs.

This work uses AdamW [51] as the optimizer in all the experiments. Weight decay ($wd$) is applied to all the layers except the bias and normalization layers. However, the weight decay varies between the different learning rate schedules. When using the multi-step learning rate, the $wd = 0.1$, and when using the cosine learning rate, the $wd = 0.05$ ($wd = 0.08$ when using CNN-Former). Furthermore, the initial decay rates for AdamW are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

All the models have been trained with a batch size of 64. Usually, larger batch sizes are recommended for pure and hybrid ViTs [21, 6, 23]. However, this was unfortunately not possible with the equipment in hand. All the GPUs available had a memory size of 8 GB and could not accept bigger batch sizes than 64.

This thesis is structured so that first, the models introduced in sections 5.2.2-5.2.4 are trained in the absence of any pretraining for 100 epochs on the INIR dataset. These results are presented in sections 6.1-6.4. Then the better ones are pretrained on the ETH-XGaze dataset for 20 epochs. How the pretraining and finetuning are made is explained below. Furthermore, how a fair comparison between pretrained and non-pretrained models is made is also explained below.

Later in section 6.8, we also look if longer training sequences can help improve the performance of the best models. Here the models are trained for a total of 400 epochs. If the model is pretrained, it is finetuned for 314 epochs; if it is not pretrained, then it is trained for 400 epochs on the INIR dataset; see
the following section for the reason for the difference of 86 epochs.

5.4.1 Pretraining & Finetuning

As explained in section 5.3.2, this work uses the gray-scaled ETH-XGaze dataset as the pretraining dataset. The same pre-processing and post-processing techniques as for the INIR dataset are used on this dataset. Because letting the model learn under the same circumstances maximizes the number of relatable weights that can be transferred. Furthermore, the same optimizer and weight decay values, as stated above, are used in pretraining.

In order to determine if pretraining on an RGB dataset can improve the performance of a model on a NIR dataset, we pretrain our models on the ETH-XGaze for 20 epochs. We used fewer epochs on the pretraining since the ETH-XGaze dataset is much bigger than INIR dataset. We use a cosine learning rate schedule for models including ViTs with the same values explained above. The only difference is that the learning rate derives to $10^{-6}$ in 20 epochs instead of 100, making the learning rate difference between consecutive epochs bigger. For the CNNs, we again use the multi-step learning rate schedule with the parameters mentioned above. Observe that the learning rate is decreased at epochs corresponding to 85% and 95% of 20 in this case.

When a model is pretrained on the ETH-XGaze dataset for 20 epochs, the number of samples it encounters is $756540 \times 20$. This is equivalent to the number of samples a model would encounter if the model had been trained on the INIR dataset for 86 epochs:

$$\frac{756540 \times 20}{175317} \approx 86.$$  

Following the pretraining, the model is then finetuned on the INIR dataset for 100 epochs. Therefore, it can be argued that the finetuned model has effectively been trained on a total of 186 epochs on the INIR dataset. This is the reason why the pretrained models in section 6.8 are finetuned for 314, and non-pretrained models are trained for 400 epochs.

To start with, the finetuned models are compared to their versions where they only have been trained for 100 epochs on the INIR dataset. Nevertheless, to ensure a fairer comparison between the pretrained and non-pretrained models, we also finetune these models on the INIR dataset for 86 more epochs. In other words, we can see the process such that the models first are pretrained for 100 epochs on the INIR dataset and then finetuned again on the INIR dataset for 86 epochs. To distinguish these models, we will refer to them as
Methods | 53

pretrained on ETH-XGaze and INIR dataset, respectively. In this way, we can
draw more meaningful conclusions about whether pretraining on the ETH-
XGaze dataset or the total number of samples a model has encountered is the
actual reason for better performance.

The ideal scenario would be to first train on the INIR dataset for 86 epochs
and then further train it for 100 epochs on the INIR dataset. Nevertheless,
because of the computational resources and the difference being small between
86 and 100, we use the already existing models trained for 100 epochs and
further finetune them for 86 epochs.

5.5 Evaluation Metrics

The performance of a model is quantitatively evaluated with three different
metrics, which are denoted as 'performance metrics.' These metrics are AAE,
RMSAE, and inlier ratio. Furthermore, the computational complexity of a
model is quantitatively evaluated by the number of parameters in a model
and the throughput of the model. These metrics are denoted as 'computational
metrics.'

5.5.1 Performance Metrics

Angular error is a measure showing the amount of deviation between two
vectors. We explain it here since all the performance metrics in this work are
based on the angular error. This work calculates the angular error according
to Equation 5.1 between the 3D estimated gaze direction, which has been
transferred to CCS ($\hat{g}$) and the 3D ground truth gaze direction ($g_{gt}$).

$$ e = \arccos\left(\frac{g_{gt} \cdot \hat{g}}{\|g_{gt}\| \|\hat{g}\|}\right). $$

(5.1)

As its name suggests, AAE averages the angular errors for all the samples
given in a set according to Equation 5.2.

$$ \text{AAE} = \frac{1}{N} \sum_{i=1}^{N} e_i, $$

(5.2)

where $N$ is the number of samples in the set, and $e$ is the angular error
according to Equation 5.1. The set can, e.g., be the training or evaluation
datasets. Furthermore, RMSAE is the square root of the mean square angular
errors for samples in a set. It is calculated by Equation 5.3.

\[
\text{RMSAE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (e_i)^2},
\]

(5.3)

where \(e_i\)'s are again calculated by Equation 5.1. In this work, observe that both AAE and RMSAE are given in degrees and in the CCS.

However, we will refrain from giving absolute values due to confidentiality reasons. Hence, this work does not directly state the values for AAE and RMSAE obtained from the equations above but states them relatively. The relative values are obtained from the following equation.

\[
\frac{A_{\text{AAE}} - B_{\text{AAE}}}{B_{\text{AAE}}} \cdot 100 = x\%.
\]

(5.4)

Here the model appearing in the denominator is the baseline, i.e., Model B is the baseline. This work also expresses this as ‘Model A is compared with respect to Model B.’ When comparing different models, we tried to be transparent with this in chapter 6 since the order matters. For example, in Table 6.1, it is clearly stated that the models have been compared with respect to MobileNet, making MobileNet the baseline. Moreover, if \(x > 0\), we say that Model A is \(x\%\) worse than Model B. Contrary if \(x < 0\), we say that Model A is \(x\%\) better than Model B.

Finally, the inlier ratio is the percentage of samples with an angular error less than a specified threshold. Take, for example, ten samples with angular errors as \([5, 11, 15, 9, 7, 2, 22, 23, 59, 6]\). Then the inlier ratio with a threshold \(t = 10\) is calculated by

\[
\text{inlier ratio} = \frac{\text{Values less than } t = 10}{\text{Number of samples}} = \frac{5}{10} \rightarrow 50\%.
\]

Using this, inlier graphs can be obtained where the \(y\)-axis is the threshold, and the \(x\)-axis is the inlier ratio in percentage. For example, one point in the example above would be \((x, y) = (50, 10)\). However, as explained above, this work only presents relative results on performance metrics. Therefore, the \(y\)-axis in the inlier plots is normalized with respect to the baseline model. See, for example, Figure 6.1. Here the baseline is the MobileNet; therefore, the normalized gaze error threshold for it is equal to 1 at the last index. The last index in this work is 95% to increase the readability of the plots. A model that performs better should ideally be below the rest of the models in an inlier
graph. For example, in Figure 6.1, ResNet is the best model.

Unless otherwise stated, the results presented in chapter 6 are calculated on the evaluation datasets.

### 5.5.2 Computational Metrics

One of our computational metrics is the number of parameters in a model. Here, the number of parameters refers to all the learnable parameters in the network. That is, the learnable parameters in convolutional layers, MLP layers, normalization layers, Multi-Head Attention layers, etc., all are counted towards the total number of parameters in the model.

Nevertheless, since the number of parameters does not always reflect the inference time (especially when comparing ViTs and CNNs), we also define a metric called throughput. The throughput is calculated by first converting the PyTorch model to an ONNX model [68]. Then 50 samples are passed through it, and the inference times are logged. Later, by averaging, a mean inference time is obtained. The throughput is then calculated by

\[
\text{Throughput} = \frac{1}{\text{Mean Inference Time}}
\]

Observe that the dimension of the throughput is images per second (img/s). Since the inference time depends on many different processes which we do not have direct control over, the throughput values have a variance of ±20 images per second even if we average over 50 images. We take this into account in chapter 6.

### 5.6 Used Tools

The deep learning framework PyTorch [69] on Python was used to train all the models. The number of parameters in the models was calculated using the Python package thop [70]. thop can also calculate the number of Multiply–Accumulates (MACs) in a model, a good metric for computational complexity used in many other works. Nevertheless, we believe this package calculates the number of MACs wrong for ViTs. We observed, for example, that the MACs for Equation 3.5 are ignored in the thop package. More generally, the MACs where neither of the multiplicands is learnable parameters are ignored in this package. Therefore, we refrain from using the number of MACs to compare the models in this work. This is why we renamed the CNN-Formers in section 5.2.4.2.
In this work, all the models were either trained on NVIDIA GeForce GTX 1080 GPU or NVIDIA GeForce RTX 2080 GPU. The GPU memory on both of these is 8 GB. Furthermore, the throughputs presented in chapter 6 are calculated when running the model on Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz. Finally, we used Grammarly to improve this text.

5.6.1 Work Supplied Before This Thesis

At the start of this thesis, a Python code that could efficiently handle pre-processing and post-processing (see chapter 4) was supplied. The supplied code had MobileNetV2 and ResNet-18 implemented as the DNN part. Nevertheless, these CNNs could only be trained with a multi-step learning rate. All the other models and techniques discussed in this thesis have been implemented as a part of this work. That is, the pure and hybrid ViTs, CNN-Formers, LSA and DeiT methods have all been implemented as part of this thesis. Inspiration for the implementation of the pure ViTs was taken from [71].
Chapter 6

Results and Analysis

This chapter presents all the experiments done in this work and analyses them. To start with, section 6.1 compares the performance of MobileNetV2 (shortened as MobileNet) and pure and hybrid ViTs when trained with multi-step and cosine learning rates. Later in sections 6.2-6.4, we present the results for pure, hybrid, and advanced ViTs, explained in chapter 5, when trained for 100 epochs on the INIR dataset.

To investigate the effect of pretraining, we choose the best models from above based on performance and computational complexity. These models are pretrained on the ETH-XGaze dataset for 20 epochs and then finetuned on the INIR dataset for 100 epochs. To make the comparison fairer, we compare these models both with their versions only trained for 100 epochs and with their versions that have been finetuned on INIR dataset after having been pretrained on the INIR dataset, see section 5.4.1 for the details. Finally, in section 6.8, we select the absolute best models and train them for a total (see section 5.4.1 why total is emphasized) of 400 epochs to see whether longer training helps the performance.

6.1 Best Learning Rate Schedule

Using the multi-step learning rate on MobileNet gave 5.97% better AAE and 4.66% better RMSAE compared to the cosine learning rate. Therefore, this work takes MobileNet with the multi-step learning rate as the baseline, and the goal is to surpass this model. Since ResNet is a CNN as MobileNet, the same scheduler as MobileNet was used for it.

In order to compare multi-step and cosine learning rates on pure and hybrid ViTs, we chose ViT-\(B_P\) and ViT-\(M_{HM}\) as representatives for each
architecture. The results showed that when using the cosine learning rate on ViT-B, the AAE decreased by 5.67% compared to the multi-step learning rate. Furthermore, the RMSAE decreased by 4.24%. Similar results could be observed in the training dataset. This shows that the cosine learning rate is superior to the multi-step learning rate for pure ViTs. Therefore, all the experiments using pure ViTs presented from this point and beyond, use a cosine learning rate.

On the other hand, the cosine learning rate did not considerably impact the performance on hybrid ViTs. In fact, on the evaluation dataset, the AAE for ViT-M when using the cosine learning rate was 2.22% worse than using the multi-step learning rate. Furthermore, the RMSAE was 1.55% worse. Nevertheless, we observed the opposite results on the training dataset, i.e., the cosine learning rate performed better on the training dataset than the multi-step learning rate. Therefore, we concluded that between multi-step and cosine learning rates, there is no favorable scheduler on hybrid ViTs. However, in the following, to be consistent with the pure ViT experiments and previous research [6], we will use cosine learning rate on all the hybrid ViT experiments.

To summarize, all the stated results from this point onward for pure, hybrid, and advanced ViTs use the cosine learning rate for training. Meanwhile, all the results given for CNNs use the multi-step learning rate for training. Hyperparameters for multi-step and cosine learning rates can be found in section 5.4.

6.2 Pure ViTs Without Pretraining

The relative performance of ResNet and pure ViTs, introduced in section 5.2.2, with respect to MobileNet is given in Table 6.1. In this section, all the models have been trained on only the INIR dataset for 100 epochs. To begin with, from this table, we see that ResNet outperforms MobileNet. However, as shown in Table 5.1, the number of parameters in ResNet is more than five times that of MobileNet. This also affects the throughput, as seen in Table 6.1. MobileNet is more than three times faster than ResNet. Hence, MobileNet is preferred in applications where speed is much more important than performance.

From Table 6.1, we see that none of the pure ViTs even manage to come close to the level of MobileNet in means of both performance and computational complexity. The closest one, which is also parameter-wise the biggest, differs with almost 41% AAE and 30% RMSAE. Furthermore, we see a correlation between the size and performance of the ViTs. Hence, defining bigger ViTs than ViT-B might result in better performance. Nevertheless, this
Table 6.1: The relative performance of pure ViTs and ResNet with respect to MobileNet. Negative values mean that the corresponding model is better than MobileNet, and positive values mean that the model is worse than MobileNet.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
<th>Throughput (img/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>0.0</td>
<td>0.0</td>
<td>315</td>
</tr>
<tr>
<td>ResNet</td>
<td>-4.13</td>
<td>-3.33</td>
<td>91</td>
</tr>
<tr>
<td>ViT-B_P</td>
<td>41.40</td>
<td>29.93</td>
<td>23</td>
</tr>
<tr>
<td>ViT-M_P</td>
<td>50.40</td>
<td>36.36</td>
<td>37</td>
</tr>
<tr>
<td>ViT-S1_P</td>
<td>94.58</td>
<td>67.35</td>
<td>103</td>
</tr>
<tr>
<td>ViT-S2_P</td>
<td>99.67</td>
<td>71.23</td>
<td>122</td>
</tr>
<tr>
<td>ViT-S3_P</td>
<td>122.00</td>
<td>87.09</td>
<td>101</td>
</tr>
</tbody>
</table>

would result in impractical inference times for mobile applications. Even ViT-B_P (which is not that big compared to ViTs defined in [21]) is much slower than MobileNet. ViT-S1_P, the best model among the small ViTs in terms of the number of parameters and performance, performs very poorly compared to MobileNet. However, even if it would have given good results, it is almost

Figure 6.1: The normalized inlier ratio graph for pure ViTs. The baseline is MobileNet.
three times slower than MobileNet.

Interestingly, even if ViT-S3P has more parameters than ViT-S1P, it performs 14.09% worse. This shows that ‘depth’ is one of the most important hyperparameters in a ViT. Furthermore, by comparing ViT-S3P with ViT-S2P, we can again see the significant impact of depth on the performance.

The normalized inlier ratio graph for the pure ViTs together with ResNet and MobileNet is given in Figure 6.1. From this figure and Table 6.1, we see that there is a big difference between the small ViTs and bigger ViTs, i.e., a pure ViT with more learnable parameters perform better than a ViT with fewer parameters. Nonetheless, even the best ViT’s inlier ratio is bad compared to the CNNs.

### 6.3 Hybrid ViTs Without Pretraining

The AAE and RMSAE for the hybrid ViTs, defined in section 5.2.3, with respect to MobileNet, are given in Table 6.2. This table shows that hybrid ViTs perform much better than pure ViTs, even with fewer learnable parameters and faster inference times. Furthermore, we see that the size of the ViT added on top of the CNN does not significantly affect the performance. For example, the difference between ViT-B_{HM} and ViT-S_{HM} is less than 1%, even though the number of parameters in ViT-B_{HM} is more than four times that of ViT-S_{HM}.

Moreover, hybrid ViTs, which have ResNet as the convolutional stem, perform much better than those with MobileNet as the convolutional stem. This indicates that the convolutional stem affects the performance more than the ViT added on top. However, as it was for the pure ViTs, the pure CNNs

Table 6.2: The relative performance of hybrid ViTs. MobileNet is the baseline. Negative values mean the corresponding model is better than MobileNet, and positive values mean the model is worse than MobileNet.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
<th>Throughput (img/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>0.0</td>
<td>0.0</td>
<td>315</td>
</tr>
<tr>
<td>ResNet</td>
<td>-4.13</td>
<td>-3.33</td>
<td>91</td>
</tr>
<tr>
<td>ViT-B_{HM}</td>
<td>7.82</td>
<td>5.81</td>
<td>81</td>
</tr>
<tr>
<td>ViT-M_{HM}</td>
<td>7.76</td>
<td>5.50</td>
<td>108</td>
</tr>
<tr>
<td>ViT-S_{HM}</td>
<td>8.48</td>
<td>6.01</td>
<td>140</td>
</tr>
<tr>
<td>ViT-M_{HR}</td>
<td>-0.50</td>
<td>-0.27</td>
<td>80</td>
</tr>
<tr>
<td>ViT-S_{HR}</td>
<td>-0.49</td>
<td>-0.34</td>
<td>89</td>
</tr>
</tbody>
</table>
Figure 6.2: The normalized inlier ratio graph for hybrid ViTs. The baseline is MobileNet.

surpass their hybrid ViT analogs. That is, even if hybrid ViTs with ResNet stem perform better than MobileNet, they are far behind pure ResNet. Nevertheless, the models with ResNet convolutional stem is slower than the ones with MobileNet stem. This can be explained by MobileNet being faster than ResNet. MobileNet is still by far the fastest model.

Figure 6.2 shows the normalized inlier ratio graph. Here we see a smaller variance between the models compared to Figure 6.1. ViTs with ResNet convolutional stem are very close to the MobileNet and hence hard to see in the figure. ViTs with MobileNet stem are still far away from MobileNet but are very close to each other. Nevertheless, this graph shows hybrid ViTs perform much better than pure ViTs on the current gaze estimation task.
6.4 Advanced Methods on ViTs Without Pre-training

6.4.1 Batch Normalization and Locality Self-Attention

As stated many times until now, ViTs have the potential to outperform CNNs under large datasets. Batch normalization normalizes the values over all the batch samples. Hence this could cause data augmentation. Therefore, our hypothesis was that using batch normalization instead of layer normalization (see section 2.6) in Block 1 and Block 2 in Figure 3.4a could improve the performance of hybrid ViTs. To investigate this hypothesis, this section uses ViT-S\textsubscript{HM} as the baseline. This is because hybrid ViTs perform much better than pure ViTs (as discussed above), and it makes more sense to try to improve them. Furthermore, the small ViT with MobileNet backbone is parameter and throughput-wise closest to MobileNet.

However, the results in Table 6.3 contradict our hypothesis. Batch normalization does not improve the performance of ViT-S\textsubscript{HM}, although it seems to be faster than using layer normalization. Nevertheless, this is misleading since the throughput experiments have a high variance (see section 5.5.2). Thus, batch normalization has no benefits over layer normalization in hybrid ViTs. Hence, in continuation, we use layer normalization in Block 1 and Block 2 as the original work [21].

Furthermore, as for the batch normalization case, the LSA method did not improve the performance nor the inference time on ViT-S\textsubscript{HM}, as seen in Table 6.3. Therefore, we do not use the LSA method in the following experiments. We did not repeat the batch normalization and LSA experiments on other hybrid ViTs since we believe that ViT-S\textsubscript{HM} represents all of them.

Table 6.3: The relative performance of ViT-S\textsubscript{HM} when using batch normalization (BN) and LSA method compared to the original ViT-S\textsubscript{HM}.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
<th>Throughput (img/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-S\textsubscript{HM}</td>
<td>0.0</td>
<td>0.0</td>
<td>140</td>
</tr>
<tr>
<td>ViT-S\textsubscript{HM} with BN</td>
<td>0.26</td>
<td>0.50</td>
<td>151</td>
</tr>
<tr>
<td>ViT-S\textsubscript{HM} with LSA</td>
<td>1.32</td>
<td>0.82</td>
<td>141</td>
</tr>
</tbody>
</table>
Table 6.4: The relative performance of CNN-Formers. The baseline is MobileNet. Positive values mean the model is worse than MobileNet.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
<th>Throughput (img/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>0.0</td>
<td>0.0</td>
<td>315</td>
</tr>
<tr>
<td>Mobile-FomerV3-1</td>
<td>40.59</td>
<td>29.42</td>
<td>108</td>
</tr>
<tr>
<td>Mobile-FomerV3-2</td>
<td>23.32</td>
<td>16.56</td>
<td>107</td>
</tr>
<tr>
<td>Mobile-FomerV2</td>
<td>1.24</td>
<td>0.47</td>
<td>77</td>
</tr>
</tbody>
</table>

### 6.4.2 CNN-Former

The performance for the CNN-Formers, defined in section 5.2.4.2, is presented in Table 6.4. Mobile-FormerV2, which encapsulates MobileNetV2, cannot surpass the performance of MobileNet. Even so, it contains a parallel Transformer architecture, and the number of parameters in this model is almost four times that of MobileNet, which affects the throughput as seen in Table 6.4. On the other hand, Mobile-FomerV3-1 and Mobile-FomerV3-2 are slightly faster than Mobile-FormerV2, albeit comparatively deficient, in contrast to that of MobileNet. This contradicts the results in [26]. However, in [26], the authors trained for longer; therefore, we also try training for longer. These results are later presented in section 6.8.

### 6.5 Pure ViTs With Pretraining

In some practical applications, the inference time and the number of parameters are critical factors. Therefore in the following experiments, we do not consider ViT-B<sub>P</sub>. Nevertheless, to keep this thesis more general and to see the impact of pretraining on different sizes of pure ViTs, we consider ViT-M<sub>P</sub>. Amongst the three small pure ViTs, this section only considers ViT-S<sub>1</sub><sup>P</sup>. Because ViT-S<sub>1</sub><sup>P</sup> gives the best performance amongst the small ViTs and is reasonably fast.

We pretrain ViT-M<sub>P</sub> and ViT-S<sub>1</sub><sup>P</sup> on the ETH-XGaze dataset for 20 epochs. After the pretraining, we finetune these models on the INIR dataset for 100 epochs. Similarly, to make the comparison with respect to MobileNet and ResNet fair, we also pretrain and then finetune these models. The pretraining and finetuning steps have been explained in detail in section 5.4.1. In Table 6.5, we compare all the models with respect to the pretrained MobileNet. The models pretrained on the ETH-XGaze dataset are indicated with the ‘†’ symbol.
in this table.

However, to see whether pretraining on the ETH-XGaze dataset or the total number of samples a model has encountered is the actual reason for better performance, we finetune the models in section 6.2 for 86 epochs on the INIR dataset. The reason for finetuning and the choice of ‘86’ has been explained in section 5.4.1. From this point onwards, we will refer to these models as ‘pretrained on the INIR’ dataset. These have been indicated with the ‘♦’ symbol in Table 6.5.

Table 6.5 shows that pretraining on the ETH-XGaze dataset improves the performance of pure ViTs despite being an RGB dataset compared to their versions in section 6.2. Suppose the models have not learned any valuable information from the pretraining. In that case, we expect to get the same AAE and RMSAE after finetuning on the INIR dataset as their versions had in section 6.2. Additionally, the loss function decreased faster during finetuning when the model was pretrained on the ETH-XGaze dataset compared to without pretraining. This indicates that pretraining on the ETH-XGaze dataset was beneficial for the model.

In order to quantize the improvement of the pretrained models with their non-pretrained versions, we use Equation 5.4. Non-pretrained versions are the baselines. Thus, for ViT-M\textsuperscript{P} and ViT-S1\textsuperscript{P}, the improvements are 13.43%

Table 6.5: Effect of pretraining on ETH-XGaze and INIR datasets on pure ViTs. The baseline is MobileNet\textsuperscript{†}.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>2.76</td>
<td>3.02</td>
</tr>
<tr>
<td>ResNet\textsuperscript{†}</td>
<td>3.42</td>
<td>2.94</td>
</tr>
<tr>
<td>MobileNet</td>
<td>1.43</td>
<td>1.07</td>
</tr>
<tr>
<td>MobileNet\textsuperscript{†}</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ViT-M\textsuperscript{P}</td>
<td>52.55</td>
<td>37.82</td>
</tr>
<tr>
<td>ViT-M\textsuperscript{†}</td>
<td>32.06</td>
<td>23.26</td>
</tr>
<tr>
<td>ViT-M\textsuperscript{P}</td>
<td>33.77</td>
<td>24.55</td>
</tr>
<tr>
<td>ViT-M\textsuperscript{♦}</td>
<td>30.66</td>
<td>22.65</td>
</tr>
<tr>
<td>ViT-S1\textsuperscript{P}</td>
<td>97.36</td>
<td>69.14</td>
</tr>
<tr>
<td>ViT-S1\textsuperscript{†}</td>
<td>62.21</td>
<td>44.10</td>
</tr>
<tr>
<td>ViT-S1\textsuperscript{♦}</td>
<td>59.86</td>
<td>42.55</td>
</tr>
</tbody>
</table>

\textsuperscript{†} indicates that the models have been pretrained on the ETH-XGaze dataset. \textsuperscript{‡} indicates that the MLP layer after the last Encoder layer has been randomly initialized when finetuning. \textsuperscript{♦} indicates that the models have been pretrained on the INIR dataset.
and 17.81%, respectively. For MobileNet and ResNet, the improvements are only 1.41% and 0.68%, respectively. Hence, pretraining helps to improve the performance of the pure ViTs much more than CNNs. Notice also that MobileNet still cannot surpass the performance of ResNet.

Nevertheless, the models trained on the INIR dataset for a total of 186 epochs exhibit superior performance when compared to those pretrained on the ETH-XGaze dataset. Specifically, ViT-M\textsuperscript{P} performs 1.06% better than ViT-M\textsuperscript{P}, and ViT-S1\textsuperscript{P} performs 1.45% better than ViT-S1\textsuperscript{P}. Observe that the magnitude of this difference is relatively small compared to the improvement of, e.g., ViT-M\textsuperscript{P} over ViT-M\textsuperscript{P}. This can clearly be seen in Figure 6.3, which illustrates the normalized inlier ratios for small and medium ViTs.

Therefore, we can conclude that despite ETH-XGaze being an RGB dataset, the models learn much from it under pretraining. Nevertheless, it is possible that a longer finetuning phase may be necessary to achieve optimal results both with and without pretraining. Therefore, to better understand the impact of pretraining, in section 6.8, we train hybrid ViTs for more epochs with and without pretraining to be sure that the models converge no matter what. These experiments yield valuable insights into the relationship between pretraining on the ETH-XGaze dataset and performance.

In ViT-M\textsuperscript{P}, ViT-S1\textsuperscript{P}, ViT-M\textsuperscript{P}, and ViT-S1\textsuperscript{P}, all the weights in the pretraining have been transferred to the finetuning stage. However, we also test by randomly initializing the weights in the last MLP layer the class token goes through. This model is indicated with the ‘‡’ symbol in Table 6.5. From
here, we see that randomly initializing the last layer compared to transferring the weights on the MLP layer worsens the performance. Therefore, in the rest of this thesis, we will transfer all the pretrained weights. Finally, observe that the pretrained (both on ETH-XGaze and INIR datasets) medium ViTs perform better than the pretrained small ViTs. This indicates that ViTs with more learnable parameters tend to perform better even under pretraining, as also stated in section 6.2.

### 6.6 Hybrid ViTs With Pretraining

For the same reasons stated in the previous section, here, we do not consider the ViT-B$_{HM}$. This model is too big (parameter-wise) and slow for real-time applications. All the other hybrid ViTs, even if some are too slow, are used to appeal to a larger audience. All these models are pretrained on the ETH-XGaze dataset for 20 epochs and then finetuned on the INIR dataset for 100 epochs, as explained in section 5.4.1. The models pretrained on the ETH-XGaze dataset are again indicated with the ‘†’ symbol in Table 6.6.

The models pretrained on the INIR dataset are indicated with ‘♦’ in Table 6.6:

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>-2.76</td>
<td>-2.30</td>
</tr>
<tr>
<td>ResNet†</td>
<td>-3.42</td>
<td>-2.94</td>
</tr>
<tr>
<td>MobileNet</td>
<td>1.43</td>
<td>1.07</td>
</tr>
<tr>
<td>MobileNet†</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ViT-M$_{HM}$</td>
<td>9.30</td>
<td>6.63</td>
</tr>
<tr>
<td>ViT-M$_{HM}$†</td>
<td>3.43</td>
<td>2.62</td>
</tr>
<tr>
<td>ViT-S$_{HM}$</td>
<td>10.03</td>
<td>7.14</td>
</tr>
<tr>
<td>ViT-S$_{HM}$†</td>
<td>3.59</td>
<td>2.37</td>
</tr>
<tr>
<td>ViT-S$_{HR}$</td>
<td>1.15</td>
<td>1.04</td>
</tr>
<tr>
<td>ViT-M$_{HR}$</td>
<td>0.92</td>
<td>0.79</td>
</tr>
<tr>
<td>ViT-M$_{HR}$†</td>
<td>-1.73</td>
<td>-1.30</td>
</tr>
<tr>
<td>ViT-S$_{HR}$</td>
<td>0.93</td>
<td>0.73</td>
</tr>
<tr>
<td>ViT-S$_{HR}$†</td>
<td>-1.30</td>
<td>-1.15</td>
</tr>
<tr>
<td>ViT-S$_{HR}$</td>
<td>-3.99</td>
<td>-3.05</td>
</tr>
</tbody>
</table>

† indicates that the models have been pretrained on the ETH-XGaze dataset. ♦ indicates that the models have been pretrained on the INIR dataset.

Table 6.6: Effect of pretraining on ETH-XGaze and INIR datasets on hybrid ViTs. The baseline is MobileNet†.
6.6. How these models are pretrained and finetuned and why are explained in sections 5.4.1 and 6.5. Only the smallest hybrid ViTs for each convolutional stem are pretrained on the INIR dataset. We hold back on pretraining the medium hybrid ViTs on the INIR dataset because of computational resources and their performance not being better enough compared to the small hybrid ViTs with the same convolutional stem.

As for the pure ViTs, the pretraining on the ETH-XGaze dataset helps improve hybrid ViTs’ performance. The improvement of ViT-M\(\dagger\)\_HM, ViT-S\(\dagger\)\_HM, ViT-M\(\dagger\)\_HR, and ViT-S\(\dagger\)\_HR compared to their non-pretrained versions are 5.33\%, 5.85\%, 2.62\%, and 2.21\%, respectively. Again, this shows that a model can acquire valuable information from a dataset with another distribution than the final task. As presented in the previous section, the improvements for MobileNet and ResNet are 1.41\% and 0.68\%, respectively. Hence, pretraining improves the performance of hybrid ViTs more than pure CNNs but less than pure ViTs. This makes sense because hybrid ViTs combine CNNs and ViTs. Furthermore, this supports the results presented in [6].

It is interesting to notice that the improvement on hybrid ViTs with ResNet convolutional stem is less than the ones with MobileNet stem. This is correlated with the improvement on ResNet and MobileNet when pretraining on the ETH-XGaze dataset. From this, we can conclude that the improvement in the convolutional stem significantly impacts the improvement of the hybrid ViTs when utilizing pretraining.

From Table 6.6, we see again that training on the INIR dataset for a total of 186 epochs exhibits superior performance than pretraining on the ETH-XGaze dataset.
XGaze dataset. In this case, we even get ViT-S\textsubscript{HR} to be better than ResNet\textsuperscript{†} while having a similar inference time, as shown in Table 6.2. The improvement of pretraining on the INIR dataset compared to pretraining on the ETH-XGaze dataset is for ViT-S\textsubscript{HM} and ViT-S\textsubscript{HR} 2.36% and 2.72%, respectively. These values are much higher than for pure ViTs, which is also reflected in Figure 6.4. Since longer training seems to be beneficial for better performance, in section 6.8, we train the small hybrid ViTs for a total of 400 epochs with and without pretraining.

6.7 Hybrid Data-Efficient Image Transformers

MobileNet\textsuperscript{†} and ResNet\textsuperscript{†} presented above are used as the teacher networks (see section 5.2.4.1) for the hybrid DeiTs in this section. These pure CNNs have been pretrained on the ETH-XGaze dataset and then finetuned on the INIR dataset using a multi-step learning rate. Then, the last couple of layers in MobileNet\textsuperscript{†} and ResNet\textsuperscript{†} are removed according to section 5.2.3. Next, the small ViT is incorporated onto the corresponding convolutional stem to create the student networks, namely, ViT-S\textsubscript{DeiT}\textsubscript{HM} and ViT-S\textsubscript{DeiT}\textsubscript{HR}. These student networks are further trained for 100 epochs on the INIR dataset. Thus, the newly added ViT layers can learn, and the convolutional stem can adapt to the modifications. Simply put, the convolutional stem must adjust itself to function as a hybrid ViT instead of a pure CNN. The results for hybrid DeiT models are shown in Table 6.7.

According to this table, the hybrid DeiTs perform better than all other hybrid ViTs with the same convolutional stem. Furthermore, it is worth noting that the hybrid DeiT with MobileNet convolutional stem (ViT-S\textsubscript{DeiT}\textsubscript{HM}) outperforms MobileNet\textsuperscript{†}. This is the first time a hybrid ViT with MobileNet stem outperforms MobileNet\textsuperscript{†}. Additionally, ViT-S\textsubscript{DeiT}\textsubscript{HR} performs better than ResNet\textsuperscript{†} by 3.07%, which is a significant improvement.

According to section 5.4.1, training for 20 epochs on the ETH-XGaze dataset sample-wise corresponds to training for approximately 86 epochs on the INIR dataset. Hence, sample-wise, ViT-S\textsubscript{DeiT}\textsubscript{HM} and ViT-S\textsubscript{DeiT}\textsubscript{HR} are trained for 286 epochs on the INIR dataset. In order to rule out the possibility that the improvement is caused by more extended training, in section 6.8, we train ViT-S\textsubscript{DeiT}\textsubscript{HM} for the same number of samples as the other models. Notice that we count the total number of epochs the CNN stem has encountered in a hybrid DeiT model, which might be unfair since the number of epochs the ViT head
Table 6.7: The performance of hybrid DeiTs together with the pretrained small hybrid ViTs. The baseline is MobileNet†. The hybrid DeiT models are indicated with the ‘DeiT’ superscript.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>−2.76</td>
<td>−2.30</td>
</tr>
<tr>
<td>ResNet†</td>
<td>−3.42</td>
<td>−2.94</td>
</tr>
<tr>
<td>MobileNet</td>
<td>1.43</td>
<td>1.07</td>
</tr>
<tr>
<td>MobileNet†</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ViT-S_HM</td>
<td>10.03</td>
<td>7.14</td>
</tr>
<tr>
<td>ViT-S†_H_M</td>
<td>3.59</td>
<td>2.37</td>
</tr>
<tr>
<td>ViT-S♦_H_M</td>
<td>1.15</td>
<td>1.04</td>
</tr>
<tr>
<td>ViT-S_DeiT_H_M</td>
<td>−1.58</td>
<td>−1.15</td>
</tr>
<tr>
<td>ViT-S_H_R</td>
<td>0.93</td>
<td>0.73</td>
</tr>
<tr>
<td>ViT-S†_H_R</td>
<td>−1.30</td>
<td>−1.15</td>
</tr>
<tr>
<td>ViT-S♦_H_R</td>
<td>−3.99</td>
<td>−3.05</td>
</tr>
<tr>
<td>ViT-S_DeiT_H_R</td>
<td>−6.39</td>
<td>−4.70</td>
</tr>
</tbody>
</table>

The meaning of the superscript † and ♦ are presented in Table 6.6.

encounters would be less than in other models.

6.8 Training ViTs for Longer

Previous sections showed that longer training enhances the performance of hybrid ViTs. Nevertheless, it is unclear whether pretraining on the ETH-XGaze dataset, the hybrid DeiT method, or simply longer training on the INIR dataset is the process that helps the hybrid ViTs the most. To address this question, in this section, we train the smallest hybrid ViTs using the different training procedures for 400·175317 samples, sample-wise equivalent to 400 epochs on the INIR dataset. To provide transparency, the methods tested in this section are listed below.

- Trained the smallest hybrid ViTs (with MobileNet and ResNet convolutional stem) for 400 epochs on the INIR datasets.

- Pretrained the smallest hybrid ViTs for 20 epochs on the ETH-XGaze dataset. As discussed many times, this sample-wise corresponds to 86 epochs on the INIR dataset. Then finetuned these for 314 epochs on the INIR dataset.
• Removed the last couple of layers of MobileNet† and ResNet† to experiment on the hybrid DeiT method introduced in section 5.2.4.1. MobileNet† and ResNet† were already pretrained on the ETH-XGaze dataset for 20 epochs and then finetuned on the INIR dataset for 100 epochs. Therefore, to reach 400 epochs, we trained ViT-S_DeiTHM and ViT-S_DeITHR on the INIR dataset for 214 epochs.

Training the models for the same number of samples allows us to compare the different training procedures fairer. Nevertheless, one can still argue that for the hybrid DeiT method, the hybrid ViT should be trained for longer. This is because even though the convolutional stem has been trained for 400 epochs, the model as a hybrid ViT has only been trained for 214 epochs. We will further discuss this below. In contrast to the results in sections 6.5 and 6.6, where the models were also trained for the same number of epochs, this time, we can be more confident that the models have converged. [60] mentions that the performance of models after 400 epochs on ImageNet [61] does not improve considerably. This caused us to choose the value 400. The results for the hybrid ViTs using MobileNet and ResNet as convolutional stems are given in Table 6.8 and 6.9, respectively. The models trained for 400 epochs are indicated with the ‘≫’ symbol next to their name.

Table 6.8 and 6.9 show that hybrid ViTs significantly improves with longer training, even when no other changes are made. For example, although the only difference between ViT-S_HM and ViT-S_HM≫ is the number of encountered training samples, the latter shows a performance increase of 10.96%. Similar results can be seen between the shorter and longer trained versions of hybrid DeiT and pretrained models as well. Nevertheless, the improvement for the

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-S_HM</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ViT-S_HM†</td>
<td>-5.85</td>
<td>-4.46</td>
</tr>
<tr>
<td>ViT-S_DeIT_HM</td>
<td>-10.55</td>
<td>-7.74</td>
</tr>
<tr>
<td>ViT-S_HM≫</td>
<td>-10.96</td>
<td>-7.95</td>
</tr>
<tr>
<td>ViT-S_HM†≫</td>
<td>-11.84</td>
<td>-9.10</td>
</tr>
<tr>
<td>ViT-S_DeIT_HM≫</td>
<td>-11.48</td>
<td>-8.43</td>
</tr>
</tbody>
</table>

† indicates that the models have been pretrained on the ETH-XGaze dataset, DeiT indicates the idea presented in section 5.2.4.1, and ‘≫’ is the models that have been trained for a total of 400 epochs (400 · 175317 samples).
Table 6.9: The performance of ViT-S_{HR} when trained with different methods. ViT-S_{HR} is the baseline.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-S_{HR}</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>ViT-S^†_{HR}</td>
<td>-2.21</td>
<td>-1.86</td>
</tr>
<tr>
<td>ViT-S^{DeiT}_{HR}</td>
<td>-7.25</td>
<td>-5.39</td>
</tr>
<tr>
<td>ViT-S_{HR}≫</td>
<td>-7.14</td>
<td>-5.31</td>
</tr>
<tr>
<td>ViT-S^†_{HR}≫</td>
<td>-6.90</td>
<td>-5.20</td>
</tr>
<tr>
<td>ViT-S^{DeiT}_{HR}≫</td>
<td>-7.66</td>
<td>-5.73</td>
</tr>
</tbody>
</table>

The meaning of the symbols †, DeiT, and ≫ are explained in Table 6.8.

hybrid ViTs with ResNet convolutional stem is less than the ones with MobileNet convolutional stem. This can be explained by the improvement in ResNet being less than MobileNet when trained for longer, see section 6.6. When carefully comparing Table 6.8 and 6.9, we see that the different training procedures affect the performance of hybrid ViTs with different convolutional stems differently. For hybrid ViTs with MobileNet stem, we see that pretraining on the ETH-XGaze dataset improves the performance more than only training for longer on the INIR dataset. On the other hand, this is not true for hybrid ViT with ResNet stem, for which longer training only on the INIR dataset helps more.

The results in Table 6.8 show that ViT-S^{DeiT}_{HM}≫ performs worse than ViT-S^†_{HM}≫. Nevertheless, the difference is small, which can have been caused by the stochastic behavior of backpropagation or that the model has been trained too short as a hybrid ViT. Indeed, the model as a whole has been trained for 400 epochs but only as a hybrid ViT for 214 epochs. Increasing the training time as a hybrid ViT may lead to better results.

On the other hand, even with a shorter training of 286 epochs, ViT-S^{DeiT}_{HR}≫ outperforms models trained for longer periods, as shown in Table 6.9. When using the DeiT method and training for a total of 400 epochs, it even improves this and becomes the best-performing model among all the tested. This indicates the enormous potential of the hybrid DeiT training method introduced in this thesis.

Figure 6.5 visualizes the results presented in Table 6.8 and 6.9 of the various training methods and make it possible to compare hybrid ViTs with MobileNet and ResNet convolutional stem to each other. Hybrid ViTs with ResNet convolutional stem perform much better on all the training procedures while being around 36% slower than those with MobileNet convolutional
This can be explained by ResNet being a better but slower model than MobileNet. Additionally, none of the training procedures for hybrid ViTs with MobileNet convolutional stem manage to surpass the performance of pure ResNet\textsuperscript{†}. On the other hand, all the models that have been trained for a total of 400 epochs outperform MobileNet\textsuperscript{†}. Furthermore, we see that most of the training procedures for hybrid ViTs with ResNet convolutional stem manage to surpass the performance of ResNet\textsuperscript{†} while having almost the same throughput, see Table 6.10.

We also trained Mobile-FomerV3-2 for longer to see whether the bad performance for CNN-Formers presented in section 6.4.2 is caused by the models not converging with only 100 epochs. Hence, Mobile-FomerV3-2 was pretrained on the ETH-XGaze dataset for 20 epochs and then finetuned on the INIR dataset for 314 epochs. The results show that this did not help the performance at all. This shows that CNN-Formers are less influential than previously indicated in [26]. This can be due to using NIR images in this work instead of RGB, as in [26].

Finally, in Table 6.10, the best training procedures for the different ViT stem.
Table 6.10: The best training methods for the different small hybrid ViTs presented in this section. The baseline is MobileNet†.

<table>
<thead>
<tr>
<th>Name</th>
<th>AAE (%)</th>
<th>RMSAE (%)</th>
<th>Throughput (img/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet†</td>
<td>−3.42</td>
<td>−2.94</td>
<td>91</td>
</tr>
<tr>
<td>MobileNet†</td>
<td>0.0</td>
<td>0.0</td>
<td>315</td>
</tr>
<tr>
<td>ViT-S\textsc{HM} \gg</td>
<td>−2.99</td>
<td>−2.61</td>
<td>140</td>
</tr>
<tr>
<td>ViT-S\textsc{DeiT} \gg</td>
<td>−6.80</td>
<td>−5.04</td>
<td>89</td>
</tr>
</tbody>
</table>

The meaning of the symbols †, DeiT, and \gg are explained in Table 6.8.

architectures are summarized and compared to MobileNet†. We also point out that ViT-S\textsc{DeiT} \gg—the best hybrid ViT with the same computational complexity as ResNet-18—performs 3.5% better than ResNet† which is remarkable since ResNets are state-of-the-art DNNs in computer vision tasks. Furthermore, the normalized inlier ratio graph corresponding to this table is given in Figure 6.6.

![Normalized gaze angle error for different models](image)

Figure 6.6: The normalized inlier ratio graph for the models given in Table 6.10. The baseline is MobileNet†.
Chapter 7

Discussion

7.1 Scientific Discussion

In sections 6.2 and 6.5, we can see that pure ViTs with more learnable parameters outperform pure ViTs with fewer parameters. For example, ViT-M performs significantly better than ViT-S1 with and without pretraining. This is understandable since a model that has more learnable parameters should ideally be able to learn more complex functions. Eventually, this should result in better performance at the cost of being slower and requiring more memory.

However, this trend is less apparent for hybrid ViTs. Sections 6.3 and 6.6 show that medium and small hybrid ViTs with the same convolutional stem are performance-wise very close to each other. This is interesting because the small hybrid ViTs have almost half the parameters of what the medium ViTs have. This also affects the inference time, i.e., small ViTs are faster than medium ViTs. On the other hand, changing the convolutional stem has a more significant impact on performance. This can be explained by the convolutional stem having a central role in achieving good performance in hybrid ViTs. The ViTs added on top has less significance.

In general, we have seen that pure CNNs and hybrid ViTs perform much better than pure ViTs. We believe there are three essential reasons for this. The first reason is that the parsing in pure ViTs is done in an unsophisticated definite manner which might cause one eye to be separated into two different patches. This can, for example, be seen in the example in Figure 3.6. This makes the task much harder and hence could explain the lower performance in pure ViTs compared to the models having local processing. The second reason is that this work focuses on NIR images. It is well known that NIR images are noisier
than, e.g., RGB images. Furthermore, convolutional layers have a smoothing effect that helps to reduce noise in images. This could explain the significant performance improvement in models having convolutional layers. When using pure ViTs, not including a convolutional layer may cause the models to struggle with noisy images, leading to lower performance. Nonetheless, we believe the most prominent reason for pure ViTs to perform worse than hybrid ViTs is that the INIR and ETH-XGaze datasets are too small so that a global attention-based model can outperform a local processing based model.

In [60], Xiao et al. have shown that CNNs converge faster than both pure and hybrid ViTs. However, they show that the gap between CNNs and hybrid ViTs is less than compared to the gap between CNNs and pure ViTs. Hence, it is not surprising that the effect of pretraining or longer training on pure ViTs is more significant than the effect of it in hybrid ViTs and CNNs. This also shows that if we had trained the pure ViTs for longer, we would possibly achieve better outcomes than the 30% and 60% presented for medium and small ViTs in section 6.5.

That the CNNs converge faster than the ViTs might also be why medium and small hybrid ViTs perform similarly. When trained only for 100 epochs, it can be so that only the convolutional stem can converge. Therefore, when not trained for enough epochs, the performance of a hybrid ViT depends on the convolutional stem’s performance. Hence the medium and small ViTs perform similarly. We can strengthen this hypothesis by looking at the difference between ViT-M\(_{HR}\) and ViT-S\(_{HR}\) in section 6.6. Here we see that when only trained for 100 epochs on the INIR dataset, the ViT-M\(_{HR}\) performs worse than ViT-S\(_{HR}\). Nevertheless, when these models are pretrained on the ETH-XGaze dataset, we see that ViT-M\(_{HR}\) performs better than the ViT-S\(_{HR}\). Hence, we believe that the bigger hybrid ViTs need more time to converge. Unfortunately, we did not have time for this in this thesis, but it is a possible future study.

In section 6.4, we showed that using batch normalization and layer normalization gave similar results. In other words, none of them was better than the other. This can be explained by that batch normalization, as opposed to our hypothesis, does not provide helpful data augmentation. Additionally, in the same section, we saw that the LSA method did not improve performance compared to not using it. This can be explained by that the ViT added on top does not only handle the global processing. Hence, obstructing it from handling the local processing worsens the performance. Consider the example of a dishwasher cleaning the dishes. Occasionally, the dishes may not come out as clean as desired from the dishwasher. In such cases, a person can clean the small leftovers by hand to improve the appearance of the dishes. Similarly,
in the case of hybrid ViTs, self-attention (local processing) in the ViT acts as the human that somewhat enhances the model’s performance when necessary.

When training the hybrid ViTs for longer, i.e., 400 epochs, we have observed that hybrid ViTs perform much better than the CNNs independent from the training procedure. This can be explained by hybrid ViTs and CNNs having a similar architecture up until the global processing. The early convolutions in both models handle local processing in the same way since, e.g., ResNet-18 and ViT-S_HR have the exact same convolutional layers up until Layer 4 (see section 5.2.3). However, in CNNs, the global processing is achieved by a couple of FC layers at the end. The ideology of FC layers is much more primitive compared to the Transformers’ ideology. In other words, the global processing in Transformers is much more sophisticated and logical, which explains why hybrid ViTs outperform state-of-the-art CNNs.

In sections 6.5 and 6.6, we saw that pretraining on the ETH-XGaze dataset, which is an RGB dataset, improves the performance of the models on the final task. As discussed previously, if the pretraining would not enhance the performance of the models, we would expect to get the same level of AAE which is not the case. However, we saw that training for longer only on the INIR dataset improved the performance more. This can be explained by the fact that the model pretrained on the ETH-XGaze dataset, needing more time to adapt to the INIR dataset.

Therefore, the models were trained for longer in section 6.8. Here we saw that the models pretrained on the ETH-XGaze dataset, and those who had not gave more or less the same results. The stochastic behavior of backpropagation can explain the slight differences. Although we have demonstrated in sections 6.5 and 6.6 that pretraining on a dataset with a different distribution from the final task’s dataset could enhance performance, in this particular example we believe that the ETH-XGaze dataset does not provide anything new. That is, the INIR dataset already contains all the possible scenarios that ETH-XGaze covers and even more. Therefore, when we let the models converge on the INIR dataset, there is no significant difference in the results obtained from models pretrained on the ETH-XGaze dataset and those not pretrained on it.

In sections 6.7 and 6.8, we saw that the hybrid DeiT training procedure is giving remarkable results. This can be explained by the fact that the pure CNN idea is baked into the final hybrid ViT. Combining two different models into one has also shown to be a good idea in [23], as presented by Touvron et al. To further clarify, let us use a real-life parallel example. Assume two people are applying for a manager position in a tech company. Both of the candidates have solid engineering skills. However, one of them has a business and management
background as well. In this case, the person with both engineering and business skills would be the best fit for this position, as they can effectively combine their knowledge from both areas to excel in this role. We believe this is also the case for DNN. A hybrid ViT that has only been trained as a hybrid ViT can only see the world from one perspective. On the other hand, a DNN model that has been trained both as a pure CNN and hybrid ViT will be able to see the world from two different perspectives and hence, perform better on the final task.

7.2 Sustainability and Ethics Discussion

Improving models is always beneficial economically because it puts a company ahead of its competitors. Obviously, performance is not the only factor for choosing a product over another, but it is an important criterion. Therefore, a work like this one is of high interest to many. Furthermore, the results of our work are significant as we have achieved better models than the current state-of-the-art CNNs. Hence, the findings in this work could benefit a company’s future economic growth. We do not restrict ourselves to companies that only produce or sell gaze estimation tools. Instead, we speak more generally that, e.g., hybrid ViTs, the hybrid DeiT training algorithm, etc., could also benefit many other companies mainly dealing with computer vision applications.

From a sustainability standpoint, it is widely recognized that DNNs have a significant carbon footprint [72]. However, in this work, we have seen that training the CNNs and the best hybrid ViTs (i.e., ViT-S\(_{HR}\) and ViT-S\(_{HM}\)) takes more or less the same amount of time. As a result, the carbon footprint remains the same. However, this work has not investigated the amount of power required to run the models, for example, for an hour. Therefore, we cannot determine how the models impact the carbon footprint when operated on a battery.

From an ethical point of view, we believe this work has no harmful side. Gaze estimation is a well-established area, and many gaze estimation tools are already in the market. This work only tried different DNN models in the specific application area. That is, the applications area is not changed. Hence, no new ethical concept arises.

Although there are no new ethical concerns arising, there are specific issues to consider when utilizing gaze estimation technology. These concerns should not be crossed by anyone producing or selling gaze estimation tools. Some of these ethical rules have been given below.
• It is inappropriate to monitor an individual’s actions without their knowledge or consent.

• Images used for gaze estimation should not be reused or stored for any other purpose unless there is an agreement.

• Gaze estimation technology should only be used for its intended purpose.

Hence, these rules and many more should be taken into account by companies that produce gaze estimation tools. Furthermore, in a situation where a person detects that these kinds of rules are broken or have been broken should immediately notify the necessary institutions. In other situations than gaze estimation, there will be other ethical concerns. In those situations, those ethical concerns should be taken into account.
Chapter 8

Conclusions

Comparing Tables 6.5 and 6.6, we can conclude that hybrid ViTs are superior to pure ViTs on the INIR dataset, even with pretraining. Additionally, these tables indicate that larger models perform better than smaller ones. However, larger models have the disadvantage that they are slower. Even the smallest hybrid ViTs defined in this work are much slower than MobileNetV2. Hence, for mobile applications making the models bigger has no advantage. Therefore, ViTs, as they are today, are not suitable for real-time mobile applications. Instead, CNNs such as MobileNetV2 should be considered in these situations.

However, in situations where, e.g., ResNet-18 is considered, using a hybrid ViT with ResNet convolutional stem is to be preferred. Because as shown in Figure 6.5, we have managed to construct hybrid ViTs that significantly outperform ResNet-18 while having a similar throughput and less number of parameters. Observe that the throughput was calculated on inputs of shape $256 \times 192$ and can vary if the input shape changes.

The experiments in section 6.6 showed that the convolutional stem in a hybrid ViT is the most important part of the model. In other words, the convolutional stem’s effect on the performance is much more than the ViT head on the top. Therefore, when constructing a hybrid ViT, one should use state-of-the-art CNNs as baselines for the convolutional stem.

In section 6.5 and 6.6, we see that pretraining on the ETH-XGaze dataset improves the performance. Therefore, pretraining on images from a different distribution than the target task is worth trying. Nevertheless, we did not see this improvement in this work when trained for longer because we think the ETH-XGaze dataset provides nothing new to the INIR dataset. Although more data could improve the performance, in order to surpass the performance of a state-of-the-art CNNs, there is no need for more data than, e.g., the INIR
dataset when using hybrid ViTs. However, longer training when using a cosine learning rate significantly improves performance. Therefore, the ViTs should be trained for extended periods before concluding.

To our best knowledge, this work tested a new training method which we named ‘hybrid DeiT.’ By looking at sections 6.7 and 6.8, the hybrid DeiT approach has a promising future for hybrid ViTs. Especially in Table 6.10, even the ViT-S$^{\text{DeiT}_{HR}}$, which has been trained for 114 epochs less than the other models, manages to outperform them. Using longer training for the DeiT method further improves this performance to make it angular error-wise the best model in this work. Hence, we recommend using the hybrid DeiT training algorithm when using hybrid ViTs. Additionally, the CNN-Former method presented in [26] did not give results as expected. We used the same Mobile-Formers presented in [26] and still got poor results on the INIR dataset.

Even though this thesis had a specific goal, such as improving gaze estimation on NIR images, we believe the findings apply in a broader context. Hence, we give the punch line of this thesis below.

In scenarios where the threshold on inference time is not too strict, hybrid ViTs, with state-of-the-art convolutional stem, should be preferred over pure ViTs, CNNs and Mobile-Formers. The hybrid ViTs should be trained with a cosine learning rate for a sufficient number of epochs to get the best results. Primarily, we suggest training the hybrid ViTs with the hybrid DeiT training algorithm presented in this work.
Chapter 9

Future Work

Due to constraints such as limited time and computational resources, several ideas that could have enhanced the performance of gaze estimation on NIR images could not be explored during this thesis. In order to pave the way for future research, we have listed some of them next. Perhaps, the easiest thing to do is to use a bigger batch size. Because of limited GPU memory, we could only use a batch of size 64, whereas previous researches have used bigger batch sizes for training on ViTs. For example, [21] and [6] use batch sizes 4096 and 512, respectively. These values are much bigger than 64 and can affect the training behavior.

This work aimed to improve the gaze estimation on NIR images. For this task, a pretraining dataset consisting of one channel was needed. Due to the lack of any other good NIR dataset for gaze estimation, the ETH-XGaze dataset consisting of 3-channel RGB images was chosen. Hence, to make the pretraining and finetuning compatible, the RGB images in the ETH-XGaze dataset were converted to gray-scale images. However, gray-scale and NIR images have distinct distributions. This might have led to not seeing an apparent noticeable effect (when finetuned for longer) of pretraining in this work. To address this, one could investigate the effects of using only the red channel of the ETH-XGaze dataset in the pretraining. Red light and NIR light have more similar energy levels compared to other colors. This could enhance the benefits of pretraining.

This work has shown that hybrid ViTs can outperform state-of-the-art CNNs such as ResNet-18 and MobileNetV2. However, for mobile applications, the model’s inference time is crucial, and none of the models presented in this thesis can match the speed of MobileNetV2. The closest one is more than twice as slow as MobileNetV2. Therefore, to implement pure or
hybrid ViTs in mobile applications, they must be optimized for faster inference while keeping the same performance levels. We assess this as one of the most important future studies.

This work utilized five hybrid ViTs with two distinct convolutional stems. Hybrid ViTs with the same convolutional stem were separated from each other solely by the ViT head. That is, the convolutional stems were identical in means of depth, number of convolutional layers, kernel sizes, strides, etc. Hence, this work has not, e.g., investigated how the depth of the convolutional stem affects the performance. Thus, a study on this could provide valuable insights. Furthermore, the used kernels in the CNN-part could be changed. This work has mainly used kernels of size $3 \times 3$. Nevertheless, using bigger kernels in the first layers so that both the eyes in the face could end up in the same kernel could yield better performance. Moreover, this could also improve the performance of the pure CNNs.

In section 6.8, hybrid ViTs trained for a total of 400 epochs were compared to each other, and the CNNs. Nevertheless, the CNNs in this section were only trained for a total of 186 epochs. Training them with the multi-step learning rate for longer would not be productive because the training curves showed that the models had converged before 186 epochs. However, training with the cosine learning rate for 400 epochs could yield different results and potentially improve the CNNs.
References


References | 85


Appendix A

Lemma on Invertibility of ReLU

Consider an operator \( \text{ReLU}(Bx) \), where \( B \) is an \( m \times n \) matrix and \( x \in \mathbb{R}^n \). Let \( y_0 = \text{ReLU}(Bx_0) \in \mathbb{R}^m \) for some \( x_0 \in \mathbb{R}^n \), then equation \( y_0 = \text{ReLU}(Bx_0) \) has a unique solution with respect to \( x \) if and only if \( y \) has at least \( n \) non-zero values and there are \( n \) linearly independent rows of \( B \) that correspond to non-zero coordinates of \( y_0 \).

In the case of using \( 1 \times 1 \) convolutional layer and the input channel size \( n \) is bigger than the output channel size \( m \), i.e. \( n > m \), there is (according to the lemma above) no way that \( x \) can have a unique solution from the obtained \( y \) values. This means that the information is lost when using ReLU activation function when the input channel size is bigger than the output channel size.