Degree Project in Machine Learning
Second cycle, 30 credits

3D Gaze Estimation on RGB Images using Vision Transformers

JING LI
3D Gaze Estimation on RGB Images using Vision Transformers

JING LI
Abstract

Gaze estimation, a vital component in numerous applications such as human-computer interaction, virtual reality, and driver monitoring systems, is the process of predicting the direction of an individual’s gaze. The predominant methods for gaze estimation can be broadly classified into intrusive and non-intrusive approaches. Intrusive methods necessitate the use of specialized hardware, such as eye trackers, while non-intrusive methods leverage images or recordings obtained from cameras to make gaze predictions.

This thesis concentrates on appearance-based gaze estimation, specifically within the non-intrusive domain, employing various deep learning models. The primary focus of this study is to compare the efficacy of Vision Transformers (ViTs), a recently introduced architecture, with Convolutional Neural Networks (CNNs) for gaze estimation on RGB images. Performance evaluations of the models are conducted based on metrics such as the angular gaze error, stimulus distance error, and model size. Within the realm of ViTs, two variants are explored: pure ViTs and hybrid ViTs, which combine both CNN and ViT architectures. Throughout the project, both variants are examined in different sizes.

Experimental results demonstrate that all pure ViTs underperform in comparison to the baseline ResNet-18 model. However, the hybrid ViT consistently emerges as the best-performing model across all evaluation datasets. Nonetheless, the discussion regarding whether to deploy the hybrid ViT or stick with the baseline model remains unresolved. This uncertainty arises because utilizing an exceedingly large and slow model, albeit highly accurate, may not be the optimal solution. Hence, the selection of an appropriate model may vary depending on the specific use case.

Keywords

3D Gaze Estimation, Vision Transformers (ViTs), Convolutional Neural Networks (CNNs), Multi-Head Attention, Red-Green-Blue (RGB) Images
Sammanfattning

Ögonblicksbedömning, en avgörande komponent inom flera tillämpningar såsom människa-datorinteraktion, virtuell verklighet och övervakningssystem för förare, är processen att förutsäga riktningen för en individs blick. De dominerande metoderna för ögonblicksbedömning kan i stort sett indelas i påträngande och icke-påträngande tillvägagångsätt. Påträngande metoder kräver användning av specialiserad hårdvara, såsom ögonspårare, medan icke-påträngande metoder utnyttjar bilder eller inspelningar som erhållits från kameror för att göra bedömningar av blicken.


Nyckelord

3D Blickriktningsestimering, Vision Transformers (ViTs), Konvolutionsneurala Nätverk (CNNs), Multi-Head Attention, Röd-Grön-Blå (RGB) Bilder
Acknowledgments

I would like to extend my sincere appreciation to Markus Flierl, my examiner, for not only approving my thesis topic but also providing invaluable guidance and advice prior to commencing my master’s thesis.

Additionally, I would like to express my gratitude to Sofia Rosenblad, my manager at Tobii, for granting me the opportunity to undertake my thesis work within the organization. I am also super grateful to Adam Ly and Viktor Åberg, my supervisors at Tobii, for their meticulous assistance and unwavering support throughout my thesis work at Tobii.

Furthermore, I would like to express my thanks to Yusen Wang, my supervisor at KTH, for dedicating time to thoroughly review my thesis report and offering valuable suggestions for improvement.

Last but not least, I would like to acknowledge Emil Emir Vardar for the unforgettable discussions we had during our projects.

Stockholm, August 2023
Jing Li
iv | Acknowledgments
# Contents

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

2 Background .................................................. 6
   2.1 Gaze Estimation ......................................... 6
      2.1.1 Model-based Gaze Estimation ....................... 7
      2.1.2 Appearance-based Gaze Estimation .................. 8
   2.2 Convolutional Neural Networks (CNNs) .................. 8
      2.2.1 Convolutional Layers ............................... 9
      2.2.2 Pooling Layers .................................. 10
      2.2.3 Fully Connected Layers ........................... 11
      2.2.4 Activation Functions ............................. 14
      2.2.5 Regularization Techniques ......................... 16
   2.3 ResNet ................................................. 20
      2.3.1 ResNet-18 ....................................... 21
      2.3.2 ResNet-50 ....................................... 22
   2.4 Transformers ........................................... 24
      2.4.1 Preliminaries .................................... 24
      2.4.2 Vision Transformer (ViT) ......................... 29
      2.4.3 Hybrid Vision Transformer ......................... 31
   2.5 Learning Rate Scheduler ............................... 31
      2.5.1 Cosine Learning Rate Scheduler .................... 32
      2.5.2 Multi-step Learning Rate Scheduler ................ 32
   2.6 Related Work .......................................... 32
List of Figures

2.1 Visualization of the eye structure. This diagram is inspired by [1]. ........................................ 8
2.2 A numerical example for the convolutional layer. ........................................ 10
2.3 A numerical example for the pooling layer. ........................................ 11
2.4 An example for the fully connected layer. ........................................ 12
2.5 Fully connected layer architecture. ........................................ 12
2.6 The two-layer MLP architecture. ........................................ 14
2.7 The graph for the softmax, ReLU and GeLU activation functions. ........................................ 16
2.8 The neural network having 2 hidden layers without (left) and with (right) dropout. [2] ........................................ 17
2.9 Residual mapping. This diagram is inspired by [3]. ........................................ 20
2.10 ResNet-18 architecture. This diagram is inspired by [4]. ........................................ 22
2.11 A bottleneck building block for ResNet-50. This diagram is inspired by [3]. ........................................ 23
2.12 ResNet-50 architecture. This diagram is inspired by [3]. ........................................ 23
2.13 The encoder and decoder block of the Transformer architecture. ........................................ 25
2.14 Detailed diagram explaining the architecture of the Transformer encoder layers (blue blocks) and the decoder layer (purple block). ........................................ 27
2.15 Scaled dot-product attention (left) and Multi-Head Attention (right) [5]. ........................................ 28
2.16 The Vision Transformer (ViT) encoder block. ........................................ 30
2.17 ViT model overview. The input is split into smaller patches and projected to a sequence of tokens. Then a class token is added to the beginning of the sequence to provide global information about the image, while the patch tokens provide the information for each individual patch. The positional embedding is added to each patch token to give spatial location information for each patch. The sequence is then fed to the Transformer encoder. This figure is inspired by [6].

2.18 The architecture of the hybrid ViT. It first applies a Convolutional Neural Network (CNN) to extract local feature maps and then feeds in the feature maps to the Transformer encoder. This figure is inspired by [7].

3.1 Sample images from the MPIIGaze dataset.

3.2 One sample image (left), the corresponding warped image (middle) and the original image with gaze direction annotations (right).

3.3 Representation of pitch and yaw.

4.1 The scatter plot of the number of trainable parameters vs. the relative MAGE on ETH-XGaze.

4.2 The scatter plot of the number of trainable parameters vs. the relative MSDE on ETH-XGaze.

4.3 The scatter plot of the number of trainable parameters vs. the relative MAGE on Tobii dataset.

4.4 The scatter plot of the number of trainable parameters vs. the relative MSDE on Tobii dataset.

4.5 The scatter plot of the number of trainable parameters vs. the relative MAGE on MPII-FaceGaze.
List of Tables

3.1 Summary of number of trainable parameters of the CNN models. .................................................. 41
3.2 Summary of training parameters of the CNN models. Note that the learning rate scheduler is slightly different for ResNet-18 and the external ResNet-50. For ResNet-18 the learning rate is 1e-4 for 20 epochs and thereafter 1e-5, while for the external ResNet-50 the learning rates starts at 1e-4 and decays by 0.1 every 10 epochs. .................................................. 41
3.3 Summary of number of trainable parameters of the pure ViT and hybrid ViT models. .............................. 42
3.4 Summary of training parameters of the ViT and hybrid ViT models. .................................................. 43
4.1 Evaluation results on ETH-XGaze. ................................. 50
4.2 Evaluation results on internal Tobii dataset. ...................... 53
4.3 Evaluation results on MPII-FaceGaze. .............................. 55
List of acronyms and abbreviations

CCS  Camera Coordinate System
CNN  Convolutional Neural Network
DETR  DEtection TRansformer
DNN  Deep Neural Network
ELU  Exponential Linear Units
EOG  Electrooculography
FC  Fully Connected
FLOPS  floating point operations per second
GeLU  Gaussian Error Linear Units
HCS  Head Coordinate System
HVT  Hierarchical Visual Transformer
MAGE  Mean Angular Gaze Error
MLP  Multi-layer Perceptron
MSDE  Mean Stimulus Distance Error
NLP  Natural Language Processing
PCCR  Pupil Center Cornea Reflection
ReLU  Rectified Linear Units
ResNet  Residual Network
RGB  Red-Green-Blue
RNN  Recurrent Neural Network
SCS  Screen Coordinate System
ViT  Vision Transformer
Chapter 1

Introduction

Gaze estimation is the process of predicting the direction of a subject’s gaze by analyzing the eye movements and head orientation. The gaze estimation technology plays an important role in various applications, like virtual reality, human-computer interaction, and driver monitoring systems. There are mainly two categories of gaze estimation methods: intrusive and non-intrusive techniques. Intrusive technique requires the subject to wear specific hardware so that the eye movements can be directly captured, while non-intrusive technique applies cameras to capture images or recordings of the subject’s face and perform the gaze estimation based on them. One typical example of intrusive techniques is known as the intrusive eye tracker built by Edmund Huey, which is a contact lens connected with an aluminium pointer [8].

Eye tracking, also referred to as gaze estimation in this report, involves different purposes for capturing and analysing the eye movements and can be performed by utilizing different technologies. One eye tracking strategy is known as image-based eye tracking, which makes use of the captured eye/face images and extracts features like the position of the pupil and eye corners. Then different algorithms and/or models are employed to estimate the gaze direction based on these extracted features [9].

Another notable strategy for eye tracking is known as Electrooculography (EOG), which measures the electrical potential difference between electrodes placed around eyes. The potential difference changes during the occurrence of the eye movements. The gaze can be estimated by analysing the obtained signals [10].

This chapter describes the specific problem that this thesis addresses, the context of the problem, the goals of this thesis project and outlines the structure of the thesis.
1.1 Background

The category of non-intrusive gaze estimation methods, introduced in the previous section, can be further divided into two sub-categories: model-based methods and appearance-based methods. For model-based methods, relevant features, like the geometry and mechanics of the eye movements, are first extracted from the images and then passed into the *pre-defined* model to represent the relationship between the extracted features and the gaze direction to make predictions. Appearance-based methods analyze the visual appearance of the eyes or faces, which are presented by images or recordings, to predict the gaze direction. These methods are commonly used to learn the mapping between the appearance features and the gaze directions by applying machine learning techniques like CNNs, which is introduced in detail in Section 2.2. CNNs have been widely employed for gaze estimation due to their ability of effectively extracting spatial features from the input images [11] [12], followed by some fully connected layers or Multi-layer Perceptron (MLP) to transform the output to useful information, i.e. the gaze direction in this case.

However, a new deep architecture, ViT has gained significant attention in recent years for their ability of modeling long-range dependencies in images using self-attention mechanisms. ViTs are discovered to have better accuracy and give more promising results than CNNs in classification related tasks. In [6], the authors propose the ViT architecture which applies the Transformer model directly to image data and demonstrate that ViTs can achieve satisfactory performance on different image classification benchmarks. One condition for ViT to outperform the existing Deep Neural Networks (DNNs), e.g. CNNs, is that the model is trained on a large amount of data. This paper initiates the research on ViTs to various computer vision tasks, including but not limited to image classification tasks.

Besides the pure ViTs, the authors of [7] design a hybrid architecture combining ViTs and CNNs. The CNN is used to extract the local features from the input image and after convolution, each local region is represented by certain feature extracted by the CNN. The generated feature matrix is then fed into a Transformer to capture global relations. They have shown that the hybrid Transformer achieves state-of-the-art performance for gaze estimation tasks. The detailed structure of CNNs, ViTs and hybrid ViTs is explained later in this report.
1.2 Purpose

The main purpose of this thesis is to understand the architecture of different types of ViTs, implement both the CNN models and ViT models, and compare the performance of these models using certain evaluation metrics.

In this paper [7], the authors mention that the hybrid Transformer outperform both the pure ViTs and the CNNs with pre-training. Since pre-training is not a part of this thesis, it is worth comparing the results of our experiment with the results they provide to see if our finding accords with their conclusion when pre-training is not introduced. One common factor of their study and this thesis is that the ETH-XGaze dataset is used as training dataset in this thesis and as pre-training dataset in their work.

Another paper that is worth mentioning is [13], in which the authors employ cross-dataset evaluations. Cross-dataset evaluation means that a model is trained on one dataset and tested on a completely different dataset. This method is the same as the method applied in this thesis and the same datasets (i.e. ETH-XGaze and MPII-FaceGaze) are used both in their work and this project. Therefore, it is interesting to see if similar results are obtained and if any of our models is able to outperform the public model they trained and published.

1.3 Goals

The main goal of this thesis is to investigate the performance of different DNNs, including CNNs, pure ViTs and hybrid ViTs, in 3D gaze estimation using Red-Green-Blue (RGB) images. The term "performance" will be discussed in different dimensions, like the size of the model, represented as the number of trainable parameters, the angular gaze error, referred to as the angle between the ground truth gaze vector and the estimated gaze vector, and the stimulus distance error, indicating the distance between the ground truth gaze point and the predicted gaze point on the screen. In order to plan the project in a more manageable way, the goal is broken down into the following sub-goals:

1. Implement and train the ResNet-18 architecture and get the external open-source ResNet-50 model [13] so that all CNN models used in this project are gathered.

2. Implement and train the pure and hybrid ViTs with different sizes, i.e. different number of trainable parameters.
3. Implement reasonable and efficient evaluation metrics that can be deployed by all models.

4. Investigate whether ViTs can outperform CNNs and whether pure ViTs or hybrid ViTs perform better.

5. Test the possibility to construct pure/hybrid ViTs with smaller size and better performance than the CNNs.

1.4 Research Question

*How do different architectures of deep neural networks (DNNs) perform in terms of 3D gaze estimation for RGB images, more specifically, pure CNNs, pure ViTs and hybrid ViTs?*

1.5 Research Methodology

To ensure the reliability of the results obtained in this thesis, the same training dataset is used for all models and three different datasets, including the ETH-XGaze dataset, the internal Tobii dataset and an external open-source MPII-FaceGaze dataset are used for evaluation. Note that the evaluation for ETH-XGaze dataset happens at the same time as the training process.

To compare the size of different models, the number of trainable parameters are measured. To compare the performance of different models, the mean angular gaze error, defined as the mean angle between the ground truth gaze vector and the predicted gaze vector over all evaluation images, is calculated. In addition to the gaze error, the mean stimulus distance error, referred to as the mean distance between the ground truth gaze point and the estimated gaze point over all evaluation images within a specific dataset is computed for the ETH-XGaze dataset and the internal Tobii dataset. Note that relative comparison is done through this thesis due to the privacy of the internal Tobii dataset, which indicates all results in Section 4 are given in the form of percentages, i.e. how much better or worse the given model is than another model.

In addition, augmentation is introduced during training in order to increase the diversity of the training data. In this project, each training image is mirrored so that the training dataset is doubled in size, overfitting is mitigated and the model generalization is enhanced. Moreover, different hyper-parameters and learning rate schedulers are tested in order to obtain the
optimal model that gives the best performance in terms of both the number of trainable parameters and the gaze error.

1.6 Structure of the thesis

Chapter 2 introduces the background of gaze estimation, CNNs, ViTs as well as some related work together with their findings. Chapter 3 dives into the datasets this thesis deploys, the components of the gaze estimation model used in this work, i.e. the GazeNet, the hyper-parameters of the trained models, and finally the evaluation metrics designed for all evaluation datasets. In chapter 4 the experimental results are presented and analysed. Chapter 5 concludes the findings of this thesis and discusses potential future work that can be conducted.
Chapter 2

Background

This chapter will review the history and applications of gaze estimation, describe the deep architectures that are commonly applied in gaze estimation, explain the types and architecture of CNNs and ViTs and finally describe work that is related to this project.

2.1 Gaze Estimation

The research of gaze estimation has a long history and has achieved rapid development in recent years. In early studies, eye tracking and gaze estimation is performed by observing eye movements like saccadic, pursuit and compensatory [14]. The major technique used there is to place skin electrodes around the eyes and measure the potential difference in order to detect the position of the eye.

With the development of computer vision, remote eye trackers and head-mounted eye trackers that rely on cameras and image processing algorithms are introduced. The remote eye trackers are placed at a certain distance from the user and are using pupil detection together with corneal glints to determine the gaze direction. The head-mounted eye trackers usually have two cameras embedded in a frame of glasses, with one camera detecting the pupil center and the other one looking outwards capturing the user’s view [15].

Image-based gaze estimation methods can be further divided into two categories: model-based gaze estimation methods and appearance-based gaze estimation methods. For each method, the gaze can be estimated as either a 2D or 3D vector. In 2D gaze estimation, the gaze angle, represented as pitch and yaw, is collected from the image and no additional information about the surrounding geometry is provided. In contrast, assumptions about the
geometry are required in 3D gaze estimation in order to correctly position the eye relative to the eye tracker in space.

### 2.1.1 Model-based Gaze Estimation

Model-based gaze estimation builds a geometric eye model to estimate the human gaze and one example of such method is called Pupil Center Cornea Reflection (PCCR). The main idea of PCCR is to calculate the angle of the visual axis by tracking the relative position of the pupil center and the corneal reflection, which is usually visualized as a glint. During data collection, infrared light is directed to the eye and causes a reflection on the cornea surface. The main reason near infrared light is used is because it is invisible to human and therefore will not bother the users. A generalized model of a human eye is presented in Figure 2.1, in which different eye parameters are marked. The visual axis is the line joining the fovea and center of corneal curvature. The retina is known as the light-sensitive layer of tissue at the back of the eyeball and the fovea is the center of the retina with the highest visual acuity. The line that joins the pupil center and the center of curvature of cornea is called the optical axis. The gaze direction is determined by the visual axis and **deviates** from the optical axis. The angle between these two axes is called the kappa angle, which differs from user to user. The kappa angle are unique for all individuals and therefore, the visual axis and kappa angle for each user need to be determined through personal calibration.

However, there exist some drawbacks of the PCCR techniques: first, large angles between the gaze vector and the camera might let the glints fall off the cornea; second, varying light condition will affect the performance of PCCR, which makes it non-applicable to some scenarios, e.g. estimating the gaze direction of a car driver; third, PCCR method requires specific hardware devices in all circumstances.
2.1.2 Appearance-based Gaze Estimation

Appearance-based gaze estimation methods directly make use of eye images and learn the mapping from images to gaze vectors. Appearance-based methods use the entire eye image or face image as a high-dimensional input feature and map the feature to low-dimensional gaze prediction, instead of extracting small-scaled features [16]. Different regression models have been used in previous studies, for example, the CNN [11] and random forest regression [17].

2.2 Convolutional Neural Networks (CNNs)

CNN is a type of deep learning model that consists of several layers of neurons and is commonly used in image recognition and computer vision tasks. The layers include convolutional layers, pooling layers and fully connected layers. These layers will be explained in detail in Section 2.2.1-2.4. In CNNs, activation functions are mathematical functions that are applied to the output of each neuron in a convolutional layer in order to introduce non-linearity into the network, allowing it to learn and represent complex patterns in the input data. Different activation functions are introduced in Section 2.2.4. In addition, in order for CNNs to generalize well without over-fitting, regularization techniques are usually included. Those are covered in Section 2.2.5.
2.2.1 Convolutional Layers

A convolutional layer is a key component of a CNN. The convolutional layer is designed to learn local patterns or features in an image by convolving a set of learnable filters or kernels with the image. The filters are in most cases smaller than the input image and each filter convolves with the input image to generate an activation map. To perform the convolution operation, the flipped filter (rotated 180 degrees) slides across the image, and the dot product between the elements of the flipped filter and those of the input image is calculated at each spatial position.

Given that the input image is of size \( (H_{in}, W_{in}, C_{in}) \), where \( H_{in} \) is the height of the input image, \( W_{in} \) is the width of the input image and \( C_{in} \) is the number of channels of the input image, and a kernel that applied to this image is of size \( (K, K, C_{in}) \), where \( K \) is the height and width of the kernel. It is worth mentioning that a single kernel will result in a single channel output, which means multiple kernels with same shape should be applied to the same input image in order to yield multi-channel outputs. The number of multiplications can be found by applying the following formula:

\[
\text{Number of multiplications} = N_k \times H_{out} \times W_{out} \times K \times K \times C_{in} \tag{2.1}
\]

where \( N_k \) is the total number of kernels, \( H_{out} \) is the height of the output map, and \( W_{out} \) is the width of the output map. \( H_{out} \) and \( W_{out} \) corresponds to the output shape and can be calculated using the following formula:

\[
\text{Output size} = \frac{\text{Input size} + 2P - K}{S} + 1 \tag{2.2}
\]

where \( P \) is the padding size and \( S \) is the stride. Zero padding is one typical type of padding technique that is used to increase the size of the input image by adding zeros around the borders of it. The main use of zero padding is to preserve the spatial resolution of the input image. Stride is the number of pixels by which the filter slides horizontally or vertically at each step during the convolution operation. The main use of stride in CNN is to control the spatial resolution of the output volume. By increasing the stride, the spatial resolution of the output volume can be decreased, which can help to reduce the computational cost and memory usage of the network, and vice versa.

Regarding the number of trainable parameters in a convolutional layer, after obtaining the output by applying the kernels to the input map, an activation function is applied to it to break its linearity. It is worth mentioning that a bias term will be added to the activated output in order to enable the
model to capture certain patterns in the data. The bias term allows the neural network to learn a better representation by allowing the activation function to shift horizontally, instead of always passing through the origin. This will be further explained in Section 2.2.4. With $N_k$ kernels, the biases are of shape $(1 \times N_k)$. Therefore, the total number of trainable parameters can be calculated using the following formula:

$$\text{Total number of trainable parameters} = (K \times K \times C_{in} + 1) \times N_k \quad (2.3)$$

According to the example shown in Figure 2.2, the zero padding $P = 0$ since no zero is padded to the input and the stride $S = 1$ so that the filter moves by 1 step both horizontally and vertically. In this example, the input is of size $(4 \times 4 \times 2)$ and the kernel is of size $(3 \times 3 \times 2)$. The boxes of different colors distinguish different multiplications between the input and the kernel. Each boxed input matrix and its corresponding kernel matrix are multiplied item-wise and all numbers in the resultant matrix are added and put in the output matrix. The values with different colors represent values coming from different channels of the input map.

Figure 2.2: A numerical example for the convolutional layer.

### 2.2.2 Pooling Layers

The pooling layer is a commonly used layer in CNN that is applied to reduce the spatial size of the input representation, while at the same time, retaining the most important features. The pooling layer downsamples the input feature maps to reduce the computational complexity of subsequent layers.

The pooling operation includes several steps: it first partitions the input
map into a set of non-overlapping regions, then a single value is computed for each region. The value summarizes the information of that region. There are different pooling operations, the most commonly used ones are **Max Pooling** and **Average Pooling**. Max pooling takes the maximum value in each region and Average pooling takes the average value in that region. One example for each of these two pooling operations is visualized in Figure 2.3. In this example, a filter of size \((2 \times 2)\) and a stride of value 2 is applied. Note that there are no trainable parameters in pooling layers.

![Figure 2.3: A numerical example for the pooling layer.](image)

### 2.2.3 Fully Connected Layers

The **Fully Connected (FC)** layer is a type of neural network layer commonly placed at the end of the network, followed by the convolutional and pooling layers. The FC layer contains neurons that are connected to the neurons in the adjacent layers, without being connected to any layers within them. The neuron in the FC layer applies a linear transformation to the input vector using a weight matrix and bias term, followed by the application of a non-linear transformation through an activation function. Assume that there are \(N_{in}\) input nodes and \(N_{out}\) output nodes, this process can be presented using Equation 2.4, where \(x \in \mathbb{R}^{1 \times N_{in}}\) is the input vector, \(y \in \mathbb{R}^{1 \times N_{out}}\) is the output vector, \(b \in \mathbb{R}^{1 \times N_{out}}\) is the bias vector, \(W \in \mathbb{R}^{N_{in} \times N_{out}}\) is the weight matrix, and \(f\) is a non-linear activation function. Assume that the output of the convolutional layers is of shape \((H_{out}, W_{out}, C_{out})\) and is going to be sent to the FC layer as input. It needs to be flattened and a common method is to apply \texttt{tensor.flatten()} to obtain an input vector of shape \((1, H_{out} \times W_{out} \times C_{out})\). This input vector is denoted as \(x\) in Equation 2.4.
\[ y = f(x \cdot W + b) \]  \hspace{2cm} (2.4)

Figure 2.4: An example for the fully connected layer.

Figure 2.4 shows an example operation in an FC layer. It takes in a flattened input size of \((1 \times 6)\) with a weight matrix of shape \((6 \times 3)\) and outputs a vector of shape \((1 \times 3)\). This example ignores the bias term because it does not affect the output shape. The weight matrix is trainable, which means its values will be updated and optimized during training. The FC layer can also be visualized using the diagram shown in Figure 2.5.

Figure 2.5: Fully connected layer architecture.

Figure 2.5 shows the connection between each pair of nodes, which indicates that every value in the input vector affects every value in the output
vector. However, it is worth mentioning that not all values in the weight matrix will affect all output values. In this example, the weights of the first neuron \( (x_1 \text{ on the diagram}) \) will only affect the first output \( y_1 \) but not \( y_2 \) and \( y_3 \).

The FC layer has trainable parameters and this category of layers usually contains more parameters because each neuron from the previous layer (i.e. the input node) is connected to every neuron in the current layer (i.e. the output node). The number of trainable parameters can be calculated using the following formula:

\[
\text{Total number of trainable parameters} = (N_i + 1) \times N_o \quad (2.5)
\]

where \( N_{in} \) is the number of input nodes and \( N_{out} \) is the number of output nodes. \( N_{in} \) and \( N_{out} \) are multiplied because each input node is linked to every output node. In addition, the FC layer has a bias for each output node, therefore an extra \( N_{out} \) is added to the result. The output size of a fully connected layer is independent of the input shape and is only dependent on the number of neurons the layer contains.

**MLP** is a type of neural network that consists of one or more fully connected layers. An MLP is composed of an input layer, hidden layer(s) and an output layer. The 2-layer MLP structure is visualized in Figure 2.6. The layer on the left is called the input layer, which contains input nodes that are fully connected to the hidden nodes in the hidden layer, i.e. the layer in the middle. The output of the hidden layer is sent to the output layer, which is the right layer in the figure and contains the output nodes. Assume that there are \( N_{in} \) input nodes, \( N_{out} \) output nodes and \( N_h \) hidden nodes, the output can be computed using Equation 2.6, where \( x \in \mathbb{R}^{1 \times N_{in}} \) is the input vector, \( y \in \mathbb{R}^{1 \times N_{out}} \) is the output vector, \( b_1 \in \mathbb{R}^{1 \times N_h} \) is the bias vector for the first FC layer, \( b_2 \in \mathbb{R}^{1 \times N_{out}} \) is the bias vector for the second FC layer, \( W_1 \in \mathbb{R}^{N_{in} \times N_h} \) is the weight matrix for the first FC layer, \( W_2 \in \mathbb{R}^{N_h \times N_{out}} \) is the weight matrix for the second FC layer, \( f_1 \) is a non-linear activation function for the first FC layer and \( f_2 \) is a non-linear activation function for the second FC layer.

The purpose of using multiple layers in an MLP is to allow the network to learn complex representations of the input data. Each layer can learn to represent the input data at a different level of abstraction, with higher layers representing more abstract features.

\[
y = f_2(f_1(x \cdot W_1 + b_1) \cdot W_2 + b_2) \quad (2.6)
\]
2.2.4 Activation Functions

Activation functions are used in CNNs in order to introduce non-linearity into the network and to control the output of each neuron. Without activation functions, a CNN would be limited to perform only linear transformations on its inputs. However, most real-world problems involve complex, non-linear relationships between input variables, therefore non-linearity is crucial to be involved for accurate prediction. Activation functions also control the output of each neuron by applying a squashing function to the neuron’s input. This helps ensure that the output of each neuron is within a certain range, which can stabilize the training of the network.

Typical activation functions like sigmoid and argmax are used quite often in artificial neural networks such as MLP and Boltzmann Machine [18]. However, there are some limitations with these activation functions. The sigmoid function can be written as the Equation 2.7 and it squishes the inputs onto the range between $[0, 1]$. This property makes sigmoid only suitable for binary classification and a threshold can be set to predict which class the input belongs to.

$$ f(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{2.7} $$

The argmax activation function takes in an input array and returns the index of the maximum value. One notable limitation with using both the sigmoid function and the argmax function is that the gradients at large values are almost zero, which makes the updates by the stochastic gradient descent quite small [18]. This problem is known as the "vanishing gradient" problem and this will lead to slow learning because the parameters are barely adjusted. To alleviate
the vanishing gradient problem, activation functions like softmax, ReLU and GeLU are designed. These activation functions will be described in detail in Section 2.2.4.1-2.2.4.3.

2.2.4.1 Softmax

The Softmax activation function is a function that converts a vector of $N$ real numbers into a probability distribution of $N$ possible outcomes [19]. It is commonly used as the last activation function of a neural network for multi-class classification problems. It normalizes the outputs to a probability distribution, i.e. $f(z)_i$ is between 0 and 1 and the sum $\sum f(z) = 1$. The Softmax function is defined as in Equation 2.8.

$$ f(z) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \text{ for } i = 1, ..., N \text{ and } z \in \mathbb{R}^N $$

2.2.4.2 Rectified Linear Units (ReLU)

The Rectified Linear Units (ReLU) activation function was introduced in order to improve the performance of the restricted Boltzmann Machine [20]. The ReLU function is defined as Equation 2.9 and the graph for the ReLU function is shown in Figure 2.7 as the blue curve. ReLU is applied element-wise to the output of each neuron in a neural network. The output is the same as the input when the input is positive (i.e. larger than zero), otherwise, the output is returned as zero.

$$ ReLU(z) = \max(0, z) $$

2.2.4.3 Gaussian Error Linear Units (GeLU)

The Gaussian Error Linear Units (GeLU) activation function was first proposed in [21] as a high-performing neural network activation function, which weights the inputs by their values rather than processing the inputs by their signs. The authors show that GeLU outperforms both ReLU and Exponential Linear Units (ELU) for all tasks with different purposes. The GeLU function is defined as Equation 2.10 and the graph for the GeLU function is shown as the orange curve in Figure 2.7. In Equation 2.10, $\Phi(z) = P(Z \leq z), Z \sim \mathcal{N}(0, 1)$ is the cumulative distribution function of the standard Gaussian distribution. This distribution is chosen because the inputs tend to follow a normal distribution, especially with the Batch Normalization.
\[ GeLU(z) = z \cdot P(Z \leq z) = z \cdot \Phi(z) = z \cdot \frac{1}{2} \left[ 1 + erf\left( \frac{z}{\sqrt{2}} \right) \right] \] (2.10)

\[ \Phi(z) = \frac{1}{2} \left[ 1 + erf\left( \frac{z}{\sqrt{2}} \right) \right] \]

Figure 2.7: The graph for the softmax, ReLU and GeLU activation functions.

2.2.5 Regularization Techniques

In the training of DNNs, regularization techniques are used to prevent overfitting, which is a common problem where the model performs well on the training data but poorly on the unseen data, i.e. the test data. Regularization techniques introduce additional constraints or penalties into the learning process, which can help the model generalize better to the test data. Two types of regularization techniques Dropout and Normalization are introduced in Section 2.2.5.1 - 2.2.5.2.

2.2.5.1 Dropout

Dropout is a common regularization technique in machine learning in order to prevent the DNN models from over-fitting. Dropout was first introduced in [2] and it can both prevents over-fitting as well as providing a way of approximately combining exponentially many neural network architectures efficiently.

The key idea of Dropout is to randomly dropout both hidden and visible units from the neural network during training to prevent the units from co-adapting too much. According to Figure 2.8, the crossed units in the sub-figure 2.8(b) are the dropped units.
2.2.5.2 Normalization

Normalization is a technique used in training DNNs to improve the performance of neural networks by addressing the issue of vanishing gradients. Batch normalization [22] and layer normalization [23] are two types of normalization that will be introduced in this chapter. Both batch normalization and layer normalization have been proven to improve the performance of the DNNs, particularly in cases where the data distribution varies widely or the network architecture is complex. The main objective of both normalization techniques is to reduce the interval covariance shift and improve the conditioning of the optimization problem, which helps to speed up and stabilize the training process.

2.2.5.2.1 Batch Normalization

Batch normalization normalizes the outputs of each layer by computing the mean and the variance of the activations over a mini-batch of examples. Such a strategy helps to reduce the internal covariance shift and leads to faster training and better generalization performance.

Batch normalization is done through normalizing each scalar feature independently by setting the mean to zero and the variance to one. Assume that a layer has the d-dimensional input $x = (x^{(1)}, ..., x^{(d)})$, then each dimension of the input will be normalized as:

$$
\hat{x}^{(i)} = \frac{x^{(i)} - [x^{(i)}]}{\sqrt{Var(x^{(i)})}}
$$

(2.11)

It has been shown in [24] that such normalization can speed up the convergence of training the neural networks and it even works with
decorrelated features. Instead of simply normalizing the input of a layer by following Equation 2.11, we also need to ensure that the transformation inserted in the network can represent the identity transform. This can be done by introducing a pair of trainable parameters $\gamma^{(i)}$ and $\beta^{(i)}$, which scale and shift the normalized input value:

$$\hat{y}^{(i)} = \gamma^{(i)} \hat{x}^{(i)} + \beta^{(i)} \quad (2.12)$$

These two parameters will be trained together with other model parameters in order to restore the representation power of the network [22]. Under the mini-batch setting in stochastic gradient training of the neural network, each mini-batch produces estimates of the mean and the variance of each activation. Consider that we have a mini-batch $B$ of size $n$, i.e. $B = x_{1...n}$, then the normalization is applied to all $n$ activations independently. The comprehensive process of first normalizing and then linearly transforming the mini-batch $B$ can be written as:

$$x_{1...n} \rightarrow \hat{x}_{1...n} \rightarrow y_{1...n} \quad (2.13)$$

Such transformation can also be written in the following steps:

- Mini batch mean: $\mu_B \leftarrow \frac{1}{n} \sum_{i=1}^{n} x_i \quad (2.14)$
- Mini batch variance: $\sigma_B^2 \leftarrow \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_B)^2 \quad (2.15)$
- Normalization: $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2.16)$
- Scaling and shifting: $y_i \leftarrow \gamma \hat{x}_i + \beta \quad (2.17)$

Batch normalization can be applied to different activations in the network and assume a linear transformation followed by a non-linear element-wise activation:

$$y = f(Ws + b) \quad (2.18)$$

where $W$ and $b$ are trainable parameters of the model and $f(\cdot)$ is the non-linear activation function like ReLU and softmax as introduced in Section 2.2.4. The batch normalization is therefore applied to $Ws + b$ before it is sent to the activation function. It is worth noting that the bias term $b$ can be ignored
when normalizing $W_s + b$ since it can be canceled out by the subsequent mean subtraction. Therefore Equation 2.18 can be rewritten as:

$$ y = f(BN(W_s)) $$

(2.19)

where $BN$ is the batch normalization transform applied independently to each dimension of $x = W_s$.

Batch normalization enables higher learning rates because it addresses the problem of vanishing gradients or getting stuck at local minima [22]. Batch normalization also contributes to the generalization of the network, while Dropout is usually used to eliminate model overfitting [25].

### 2.2.5.2.2 Layer Normalization

According to Section 2.2.5.2.1, batch normalization uses the distribution of the summed input to a neuron over a mini-batch of training cases to calculate the mean and variance. These values are then used to normalize the summed input to that neuron on each training case. However, two limitations of batch normalization are 1) it depends on the batch size and 2) it is not obvious how to apply batch normalization to recurrent neural networks. In order to solve these two problems, a relevant normalization technique, layer normalization is designed.

Instead of one single neuron, layer normalization is normalizing the summed inputs to all neurons in a layer on a single training case [23]. The mean and variance used are obtained as follows:

Mean: $\mu^l = \frac{1}{N} \sum_{i=1}^{N} a^l_i$  

(2.20)

Variance: $\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (a^l_i - \mu^l)^2$  

(2.21)

where $N$ denotes the number of neurons in a layer and $a^l_i$ denotes the summed inputs to the $i$-th neuron in the $l$-th layer. In layer normalization, all neurons in a layer share the same mean and variance and there is no constraint on the size of a mini-batch.
2.3 ResNet

DNNs have been revolutionized the computer vision field, especially enable remarkable progress in image classification related tasks. However, DNNs will encounter the challenge of vanishing gradients as well as performance degradation when the model gets deeper, which means that a better model is not equivalent to a model with more layers. The problem of "vanishing gradients" has been mentioned in previous sections and the potential solutions to it are normalized initialization [26], intermediate normalization layers like batch normalization [22], using activation functions like ReLU and GeLU to break the linearity of the network, etc. The degradation problem arises when the deep networks start converging, which means the classification accuracy gets saturated when the depth of the network increases. It is surprising that the degradation is found to be not caused by over-fitting. In 2015, He et al. [3] introduced Residual Network (ResNet), which is an architecture that addresses both the vanishing gradient problem and the degradation problem by introducing a strategy named residual learning.

Let $x$ be the input vector of the deep network, $y$ be the output vector, $\{W_i\}$ be the set of learnable parameters and $F$ be the shallower version of the deep network, then the output of the network can be represented as $F(x, \{W_i\})$. However, when adding more layers to $F$, a new network, denoted as $G$, is formed and should not perform worse than the shallower network $F$. In practice, the aforementioned degradation problem arises, i.e. the deeper model $G$ performs even worse than the shallower model $F$.

![Residual mapping](image)

Figure 2.9: Residual mapping. This diagram is inspired by [3].

In order to address the degradation problem, the authors in [3] introduces a residual learning framework, which let the layers fit a residual mapping instead
of hope them to fit an expected underlying mapping. This solution is visualized in Figure 2.9, in which the residual building block can be defined as:

$$y = \mathcal{F}(x, \{W_i\}) + x \quad (2.22)$$

where \(\mathcal{F}(x, \{W_i\})\) is the residual mapping that needs to be learned. In Figure 2.9, there are two weight layers, meaning that \(\mathcal{F}(x)\) in the figure can be written as:

$$\mathcal{F}(x) = W_2 f(W_1 x) \quad (2.23)$$

where \(f(\cdot)\) represents the activation function ReLU while the biases are ignored for simplification. The element-wise addition operation \(\mathcal{F}(x) + x\) is done through a shortcut, presented as the curved arrow in the figure. Another ReLU activation is performed on the output in order to further break the linearity of the network. It is worth noting that the dimensions of the input vector and the residual \(\mathcal{F}(x)\) have to be identical, otherwise a linear projection mapping will be introduced by the shortcut connections to match the dimensions.

### 2.3.1 ResNet-18

The architecture of ResNet-18, containing in total 18 layers, is shown in Figure 2.10. The first 17 layers are convolutional layers using \(3 \times 3\) filters and the last layer consists of an average pooling layer, an FC layer and a softmax activation layer [4].

The amount of filters in each convolutional layer is determined through the size of the output feature map. If the spatial size of the output feature map remains the same, then the layers in the network will have the same number of filters. However, if the output feature map is halved in size, the number of filters will get doubled in the subsequent layers. This kind of doubling allows the network to capture more complex and diverse features due to the fact that the spatial resolution decreases. The down-sampling of the network is achieved by applying a stride of 2, which means each kernel will move two pixels at a time and therefore the spatial size of the feature map get reduced by \(1/2\).

The residual shortcut connections are introduced in ResNet-18. There are two types of such connections. When the input and output of the layer have the same dimensions, the identity mapping are involved and the connections denoted by the solid lines are in use. In contrast, the connections denoted by the dotted lines are used when the input and output have different dimensions,
i.e. the projection mapping will take place to increase dimensions.

Figure 2.10: ResNet-18 architecture. This diagram is inspired by [4].

2.3.2 ResNet-50

ResNet-50 is also an architecture involving similar layers and connections as ResNet-18. However, one important difference is that ResNet-50 uses a bottleneck design for the residual block, which deploys $1 \times 1$ convolutions. Such feature would reduce the number of parameters of the model as well as the number of matrix multiplications in order to obtain a faster training. A building block for ResNet-50 is shown in Figure 2.11. For each residual function $F$, a stack of 3 layers is used instead of 2 layers, which are used for ResNet-18. The $1 \times 1$ layers are used for first reducing and then increasing the dimensions, which enables the middle $3 \times 3$ layer to have a smaller input and output dimension. Although the shortcut shown in Figure 2.11 is marked as identity, it can also be replaced with a projection shortcut. It has been shown that the projection shortcut will lead to a worse time complexity and larger model size than the identity shortcut, therefore, identity shortcut is preferred for the bottleneck building blocks [3].
Figure 2.11: A bottleneck building block for ResNet-50. This diagram is inspired by [3].

Figure 2.12: ResNet-50 architecture. This diagram is inspired by [3].

The architecture of ResNet-50 is shown in Figure 2.12 and includes in total 50 layers. It starts with a convolutional layer with $7 \times 7$ kernel size, followed by a max pooling layer. Thereafter 3 convolutional layers with $1 \times 1$, 64 kernels, $3 \times 3$, 64 kernels and $1 \times 1$, 256 kernels respectively, are introduced and repeated 3 times, i.e. in total 9 convolutional layers. Then another 3 convolutional layers with $1 \times 1$, 128 kernels, $3 \times 3$, 128 kernels and $1 \times 1$, 512 kernels respectively, are iterated 4 times, i.e. in total 12 convolutional layers. Similarly, another 18 convolutional layers are added as 6 iterations of the following layers: $1 \times 1$, 256 kernels, $3 \times 3$, 256 kernels and $1 \times 1$, 1024 kernels. Finally, 9 more layers are added as $1 \times 1$, 512 kernels, $3 \times 3$, 512 kernels and $1 \times 1$, 2048 kernels, repeating 3 times. The number of layers are added up to 50 up to this point, however, an average pooling layer is added in the end, followed by an FC layer using the softmax activation function.
2.4 Transformers

2.4.1 Preliminaries

Transformers are deep learning models that have revolutionized mainly Natural Language Processing (NLP) tasks, including machine translation [27], text generation [28], sentiment analysis [29], question answering [30] etc.

Transformers were first introduced in [5]. The CNNs and traditional Recurrent Neural Networks (RNNs) suffer from certain limitations in some NLP tasks. For RNNs, more specifically, they have sequential computation constraints and cannot handle long-range dependencies. While for CNNs, are more suitable for extracting local feature maps (local patterns) but struggle with maintaining positional information. Transformers are designed to address these mentioned problems with the help of the self-attention mechanism.

The fundamental building block of a Transformer model is the self-attention mechanism. The attention mechanism makes it possible for the model to weigh the importance of different words in a sequence when processing each word, i.e. every word in a sentence will get the chance to interact with all words in this sentence. It calculates the attention scores by taking the dot product between a query vector and the keys and values of other words in the sequence. The self-attention mechanism allows the Transformer to process the entire sequence simultaneously, i.e. parallel computation are enabled and long-term dependencies are facilitated.

Transformers contain an encoder block and a decoder block, where both of them consist of certain number of identical stacked layers. Each single one of such layer is called an encoder layer if it belongs to the encoder block or a decoder layer if that belongs to the decoder block. The encoder block takes in an input sequence of representations \( x = (x_1, \ldots, x_n) \) and outputs a sequence of continuous representations \( z = (z_1, \ldots, z_n) \). Then, \( z \) is taken in as the input by all decoder layers and an output sequence is generated. This logic flow is visualized in Figure 2.13.
2.4.1.1 Input embedding and positional encoding

Before passing the input sequence into the Transformer, the pre-processing needs to be done. The input sequence is first embedded into a continuous vector space, indicating that each token in the sequence is projected to a vector with fixed dimension. The output of this projection is with dimension $\mathbb{R}^{n \times d_{\text{model}}}$, where $n$ is the sequence length and $d_{\text{model}}$, also called hidden dimension, is the constant latent vector size through all layers.

Once the input embeddings are obtained, the positional encodings, which are of the same dimension as the input embeddings, are added to the input embeddings in order to incorporate the positional information of the tokens into the model. The main use of these encodings is to capture the position of the tokens in the sequence.

2.4.1.2 Encoder Layer

The encoder block of the Transformers is widely used in computer vision tasks [31]. Each encoder layer is mainly composed of three components: the layer normalization, the Multi-Head Attention and the MLP. With the embedded input $X \in \mathbb{R}^{n \times d_{\text{model}}}$, the feature is projected into queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$ and values $V \in \mathbb{R}^{n \times d_v}$ with MLP, where $d_k$ is the dimension of...
the queries and keys, and $d_v$ is the dimension of the values. This projection is done with the help of a projection matrix for each feature and the $Q$, $K$ and $V$ are obtained using the following formulas:

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

where $W_Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_V \in \mathbb{R}^{d_{model} \times d_v}$ are the projection matrices of the query, key and value matrices, respectively.

According to the architecture of the encoder layer shown in Figure 2.14, for each encoder layer, the query, key and value matrices are passed into a Multi-Head Attention layer, which is the main mechanism used in Transformers and will be explained in detail in Section 2.4.1.4. Then an addition operation is performed to add the output of the Multi-Head Attention layer to the input of the layer, followed by layer normalization, which is introduced in Section 2.2.5.2.2. Then the MLP layer is applied between Multi-Head Attention layers for non-linearity. Last but not least, the normalized output of the MLP is added to the sum of the output of the Multi-Head Attention layer and the input of the entire encoder layer, followed by layer normalization. These operations can be represented by the following mathematical equations:

$$X' = LN(MHA(X) + X)$$
$$X = LN(MLP(X') + X')$$

where $MHA$ represents Multi-Head Attention and $LN$ represents layer normalization.
2.4.1.3 Decoder Layer

The decoder layers are designed in a similar way to the encoder layers. However, in addition to the two sub-layers in any encoder layer, i.e. the MLP layer and the Multi-Head Attention layer, the decoder block inserts a third sub-layer named encoder-decoder attention layer to all decoder layers to perform Multi-Head Attention over the output of the encoder layer. The keys $K$ and values $V$ are obtained from the last encoder layer, while the queries $Q$ are obtained as the previous normalized output of the Multi-Head Attention from the same decoder layer. An image overview of the decoder layer is shown in Figure 2.14, in which the abbreviated sub-layer 1 and sub-layer 2 are the same as those in the encoder layers, and sub-layer 3 is the newly added encoder-decoder attention layer. Note that the decoder block is not used for ViTs and therefore is not the main focus of this thesis.

2.4.1.4 Multi-Head Attention

Instead of jumping into the introduction of Multi-Head Attention directly, we start with a particular attention mechanism called "Scaled Dot-Product Attention". The input for this consists of queries $Q$ and keys $K$ of dimension $d_k$ and values $V$ of dimension $d_v$. The attention score is calculated using the
following attention function [5]:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$$  \hspace{1cm} (2.29)

This attention algorithm is identical to dot-product attention, except for the scaling factor $\frac{1}{\sqrt{d_k}}$. This scaling factor is added by the authors in [5] due to the suspicion that for large values of $d_k$, the fast growth of dot products might push the softmax function into regions that correspond to extremely small gradients.

Multi-Head Attention refers to that the queries, keys and values are linearly projected $h$ times with different learned linear projections to $d_k$, $d_k$ and $d_v$, respectively [5]. According to Figure 2.15, on each of the $h$ projected versions of queries, keys and values, the scaled dot-product attention is proceeded in parallel. The output values are concatenated and linearly projected. This Multi-Head Attention can be represented by the following formula:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$  \hspace{1cm} (2.30)

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and the projection matrices are of dimension $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

![Scaled Dot-Product Attention](image1)

![Multi-Head Attention](image2)

**Figure 2.15:** Scaled dot-product attention (left) and Multi-Head Attention (right) [5].
2.4.2 Vision Transformer (ViT)

The ViT is a deep learning model that applies the original Transformer architecture, which is initially designed for NLP, to images related tasks like image classification. As mentioned before, ViT only contains the encoder layers but not the decoder layers, which is one of the main differences between the Transformer and the ViT.

The standard Transformer takes in 1D sequence of token embeddings, while ViTs take in 2D images as inputs. Assume that the input image is of shape \((H, W, C)\), where \(H\) is the number of pixels along the height dimension, \(W\) is the number of pixels along the width dimension and \(C\) is the number of channels. More specifically, \(C = 3\) for RGB images. To handle 2D images, the images are reshaped into a sequence of patches of shape \((P, P, C)\), where \(P\) is the patch size and \(C\) is again the number of image channels. After the image is broken down into patches, each patch is flattened into a vector of shape \(P^2 \cdot C\). Hence, there are in total \(N = HW/P^2\) patches, which also indicates the length of the input sequence for the ViT. Similar to the standard Transformer, the flattened 2D input matrix \(X_{\text{in}} \in \mathbb{R}^{N \times P^2 \cdot C}\) experiences a linear project with the help of a weight matrix and the projected matrix gets passed to the encoder block as input:

\[
X = X_{\text{proj}} = X_{\text{in}} W
\]

where \(w \in \mathbb{R}^{P^2 \cdot C \times d_{\text{model}}}\) and therefore \(X \in \mathbb{R}^{N \times d_{\text{model}}}\).

The next step is to add a learnable class embedding in the beginning of each input sequence (i.e. one class token per input image). This idea is similar to the BERT’s [class] token [32], and adding such token with update the dimension of the input to \(X \in \mathbb{R}^{(N+1) \times d_{\text{model}}}\). In addition to this, positional embeddings are added to the patch embeddings as well to give location information for each patch. 1D positional embeddings are used instead of the 2D ones due to the findings in [6].

When the resulting sequence of embedding vectors are served as input to the encoder, the output \(Z \in \mathbb{R}^{(N+1) \times d_{\text{model}}}\) is obtained. Depending on the purpose of the task, post-processing of the output needs to be performed. For example, in image classification tasks, a class needs to be predicted. Then the class token is extracted from the output of the encoder and passed to an extra MLP layer appended in the end as input to obtain the desired output, i.e. a predicted class. In this project, the output of the network is expected to be a vector of a certain dimension, for example \((16, 1)\), where the first three dimensions represent the 3D gaze direction etc. Then the MLP parameters should be changed accordingly.
The ViT encoder block is visualized in Figure 2.16, where the position of the layer normalization layer is switched when being compared to the standard Transformers.

Figure 2.17: ViT model overview. The input is split into smaller patches and projected to a sequence of tokens. Then a class token is added to the beginning of the sequence to provide global information about the image, while the patch tokens provide the information for each individual patch. The positional embedding is added to each patch token to give spatial location information for each patch. The sequence is then fed to the Transformer encoder. This figure is inspired by [6].
2.4.3 Hybrid Vision Transformer

Hybrid ViTs are a variant of the pure ViTs that combine the CNN with the Transformers. The initial purpose of designing such model is to leverage the local pattern capturing ability of CNNs and the global contextual modeling capability of Transformers.

Figure 2.18 gives an overview of how the hybrid ViT is built. A CNN is placed in the beginning to extract the local feature maps from the input images. Then these extracted feature maps are sent to the Transformer encoder instead of the original patches introduced in Section 2.4.2. According to the authors of [7], gaze estimation is a regression task and it is hard to predict the gaze with a patch containing only half of the eye. Hence, using CNN is beneficial since the feature maps generated by convolution contain the information of a local region instead of a small patch.

After the feature maps are generated, they are reshaped to a 2D patch, the class token and positional embeddings are added and the resulting patch embedding is passed to the Transformer encoder for prediction making.

Figure 2.18: The architecture of the hybrid ViT. It first applies a CNN to extract local feature maps and then feeds in the feature maps to the Transformer encoder. This figure is inspired by [7].

2.5 Learning Rate Scheduler

Learning rate scheduler is an important component during the training of machine learning and deep learning models. The scheduler determines the rate at which the weights of the network are updated and plays a crucial role in achieving faster convergence and better generalization performance. Both schedulers are widely used in deep learning practices but the effectiveness of them varies depending on the specific task, model structure and the dataset. In
this project, these two schedulers are applied to different models, which will be further explained in Section 3.

2.5.1 Cosine Learning Rate Scheduler

The cosine learning rate scheduler adjusts the learning rate based on half a cosine function. It starts with a linear warm up, i.e. rapidly increase the learning rate within a small amount of epochs, and then the learning rate decreases gradually over time according to the cosine curve.

The cosine learning rate scheduler is inspired by the idea of cyclic learning rates, which aim to improve the optimization process by escaping from sharp minima [33].

2.5.2 Multi-step Learning Rate Scheduler

The multi-step learning rate scheduler adjusts the learning rate at specific epochs during the training process. It starts with an initial learning rate and then it gets reduced by a certain factor at the predefined epochs. This process repeats until the end of the training.

The multi-step learning rate scheduler is extremely useful when the learning rate needs to be adjusted at certain points during training.

2.6 Related Work

2.6.1 Gaze Estimation

As mentioned before, gaze estimation has attracted significant attention due to its applications in various domains, like human-computer interaction and automotive systems. Early gaze estimation approaches rely on hand-crafted features and some classic machine learning algorithms. In [34], the authors present the description of a pupil-corneal reflection technique together with some recent techniques for remote eye tracking. The solutions they find manage to eliminate the calibration procedure and free head motion is allowed. Zhang et al. present an improved PCCR technique, which solves several problems the existing PCCR system has. The paper proposes a head position compensation of head motion effects on pupil-glint vectors, and presents one-point calibration. This method minimize the hardware requirements without affecting the estimation accuracy, which makes the solution effective for human-computer interaction gaze estimation.
Although traditional methods have made tremendous contributions to the gaze estimation field, they often suffer in handling complex gaze behaviour. With the development of deep learning, CNNs have revolutionized the field of computer vision and have been identified excellent performance in gaze estimation tasks. One of the pioneering works in CNN-based gaze estimation is the work presented by Zhang et al. [11], in which they propose a CNN architecture that performs appearance-based gaze estimation based on eye images collected from participants during natural everyday laptop use. The authors show that their CNNs significantly outperform state-of-the-art methods in the hardest cross-dataset evaluation.

In [35], the authors investigate the commercial applications of gaze estimation by building the eye tracking software on commodity hardware like mobile phones and tablets, without employing any additional sensors etc. In their work they train a CNN named iTracker and achieve a significant reduction in error over existing methods. Cheng et al. also use eye images as input and build an asymmetric regression-evaluation network to perform 3D gaze estimation [36]. They explore the asymmetry of the two eyes and investigate asymmetric regression. Wang et al. identify the major challenges for getting good generalization performance for CNN-based gaze estimation are: appearance variation, head pose variation and over-fitting [37]. They try to incorporate adversarial learning and Bayesian inference to address the identified issues and obtain a better generalization performance.

2.6.2 Transformers and Vision Transformers

Transformer is a deep network architecture proposed by Vaswani et al. in [5]. The Transformer is solely based on attention mechanism without any convolutional operations and has been proved to be powerful for NLP related tasks. When being compared to recurrent networks, Transformers are designed in a way that can handle longer sequences due to the global computations and descent memory of the self-attention layers [38].

Due to the great performance of Transformers in NLP field, people are inspired for applying the attention mechanism in computer vision related tasks. Carion et al. apply the Transformer encoder-decoder architecture for object detection [39]. The DEtection TRansformer (DETR), i.e. the Transformer-based object detector, reasons about the object and the global image context to output the predictions in parallel and is shown to have similar accuracy and run-time performance with the well-optimized Faster R-CNN baseline model. In [40], the authors present an attention-based decoder module to
bridge other representations into a well-performing object detector and obtain significant improvement in performance. Zhu et al. identify the problem of slow convergence and limited feature spatial resolution for DETR due to the limitation of Transformer’s attention module in feature map processing [41]. Therefore, the deformable DETR is proposed so that the attention modules only attend to a small set of key sampling points around a reference instead of all key sampling points.

The Vision Transformer architecture is first proposed by Dosovitskiy et al. in [6]. They have shown that pure Transformers can perform really well when image is divided into patches and fed into the Transformer decoder as sequences. The authors pre-trained on a large amount of data and found that ViT performs better than the state-of-the-are convolutional networks and at the same time requires fewer computational resources during training.

In [42], the authors identify that it might be problematic if the Transformers can only perform well when being pre-trained on a large amount of data. Hence, they propose a competitive ViT trained on ImageNet using a single computer in less than 3 days. They apply a teacher-student strategy during training and the result Transformers give quite competitive classification accuracy.

Pan et al. point out the redundancy and lacking of hierarchical representation during the training of ViT because it takes in the full-length patch sequences as input [43]. As a solution, they come up with a novel Hierarchical Visual Transformer (HVT) architecture, which intends to shorten the input sequence and therefore reduce the required computational resources. This idea is found beneficial because the dimensions of depth, width, patch size and resolution can be scaled to increase the model capacity, without adding any extra computational complexity to the model. The results of the conducted experiments show that the HVTs outperform the baseline networks when they are comparable in floating point operations per second (FLOPS).

Cheng et al. consider two forms of vision transformers, which are the pure ViTs and the hybrid ViTs. The hybrid ViTs employ convolutional layers from the CNNs to generate feature maps and make predictions based on the feature map. The authors compare the performance of pure ViTs and hybrid ViTs in terms of the angular gaze error. Experiments show that the hybrid ViTs performs the best in all benchmarks with pre-training.

Cai et al. presents an ensemble learning method for gaze estimation using ETH-XGaze dataset. In the work they adopt four network architecture and ensemble their individual predictions through linear combination. This method is proved to outperform other existing methods in terms of the angular
gaze error.
Chapter 3

Method

3.1 Datasets

3.1.1 ETH-XGaze

ETH-XGaze [13] is used as both the training and the evaluation dataset for this project. Note that the evaluation on ETH-XGaze happens at the same time as training. ETH-XGaze is a new large-scale dataset for gaze estimation in unconstrained environments. This dataset is composed of 600 thousand training frames collected from 64 subjects and roughly 100 thousand evaluation frames collected from 16 subjects, with varying head poses, eye movements, and facial expressions, and was collected using a head-mounted eye tracker in real-world scenarios.

The dataset samples large variations in head poses, up to the limit of where both eyes are still visible (maximum $\pm 70^\circ$ from directly facing the camera) as well as comprehensive gaze directions (maximum $\pm 50^\circ$ in the head coordinate system). It also includes a diverse set of people with different ages, ethnicities, and eye colors, making it suitable for evaluating the performance of gaze estimation algorithms in a wide range of demographics.

To ensure fair comparisons between different methods that leverage the ETH-XGaze dataset, the authors proposed a standardized evaluation protocol, which refers to a website that is open to the public to submit, evaluate and compare gaze estimation methods based on ETH-XGaze.

To evaluate the performance of the dataset and compare that with the other existing gaze datasets, the authors provide a baseline gaze estimation method using a ResNet-50 network. All datasets are trained using the same network and the cross-dataset evaluation is performed. Cross-dataset evaluation is
selected since it indicates the generalization capabilities of a gaze estimation method. The model is first trained on ETH-XGaze and tested on other datasets, as well as trained on other datasets and tested on ETH-XGaze. The results show that larger gaze estimation errors when training on existing dataset and testing on ETH-XGaze, indicating that there is a big domain gap between ETH-XGaze and previous datasets. This also proves that ETH-XGaze exhibits much larger variation in head pose and gaze direction compared to other datasets.

The ETH-XGaze includes the ground truth of the 3D-gaze targets as well as the annotations for 2D facial landmarks. With the help of the 2D landmarks, the 3D gaze origin, which is defined as the central point between the eyes and the mouth in this thesis, can be estimated using a warped mesh, which makes it possible for us to estimate the gaze ray from the 3D gaze origin to the 3D gaze target.

### 3.1.2 Internal Tobii dataset

In order to accurately evaluate the performance of the model, Tobii’s internal RGB dataset is used as the second evaluation dataset. The Tobii dataset has the 3D gaze target annotation, which is the main information used during evaluation.

### 3.1.3 MPII-FaceGaze

In order to evaluate the model on different datasets, MPII-FaceGaze is selected to be the external evaluation dataset, which will also be referred to as the MPII dataset later in this report for simplicity. MPII contains 37667 frames collected from 15 subjects covering multiple months of their daily life [44]. The sample frames from the MPII dataset is visualized in Figure 3.1. The dataset contains a large variety of illumination conditions, head poses, gaze directions and personal appearance. The images are taken outside of controlled laboratory conditions using laptop cameras of the subjects. Laptops are used mainly because of two reasons: 1) they are suitable for long-term daily data collection and 2) they are important platform for eye tracking applications [45]. MPII-FaceGaze includes the annotated 3D face center and 3D gaze target.
3.2 Gaze Estimation Model (GazeNet)

3.2.1 Pre-processing and Data Normalization

One typical assumption made in appearance-based gaze estimation methods is that frontal head pose is used, however, the head rotation would also need to be considered in real-world settings. Moreover, the subject position, i.e. the distance between the face and the camera can also affect the eye size/appearance shown in the image. Obviously, a closer distance results in a larger resolution in the captured image. According to [46], the images that correspond to different distances between the subject and the camera will have different appearances even if they are cropped to be of the same size. To solve the problem of appearance difference, we can physically scale the 3D space by changing the focal length of the camera. These factors need to be considered during the training processes, however, infinite variations of head poses and image resolutions will bring a difficulty to practical work. Therefore, certain image normalization technique should be designed to align the images so that they have a fixed range of variations.

The data normalization method applied in this project refers to [46], and the main idea is to scale and rotate the camera so that the Camera Coordinate System (CCS) and the Head Coordinate System (HCS) are standardized. The x-axis, denoted as $x_r$, of HCS is a line going from right eye to left eye, the y-axis, denoted as $y_r$, is perpendicular to the x-axis and pointing upwards through the head and the z-axis, denoted as $z_r$, is pointing forwards from the face. The origin of HCS is the center point between the eyes and the mouth. The translation and rotation of the CCS to the HCS are denoted as $e_r$ and $R_r$, respectively.

The normalization process starts by rotating CCS using the rotation matrix
so that the x-axes of HCS and CCS are on the same plane, the normalized camera is pointing towards the origin of HCS and the origin of the HCS is at the center of the normalized image. After such rotation the z-axis of CCS, denoted as $z_c$ should be our predefined $e_r$ and therefore, the rotated y-axis of CCS can be calculated as $y_c = z_c \times x_r$. Once $y_c$ and $z_c$ are calculated, the x-axis of CCS, denoted as $x_c$, can be determined as $x_c = y_c \times z_c$. Given all this information, the rotation matrix can be defined as $R = \begin{bmatrix} \frac{x_c}{\|x_c\|}; & \frac{y_c}{\|y_c\|}; & \frac{z_c}{\|z_c\|} \end{bmatrix}$.

In order to have the normalized camera located at a fixed distance from the eye center and to always have the eye in the same size in the normalized image, the scaling matrix $S$ can be defined as a diagonal matrix: $S = diag(1, 1, \frac{d_n}{\|e_r\|})$, where $d_n$ is the distance between the normalized camera and the eye center. In conclusion, the overall transformation for CCS should be written as $M = SR$.

In order to normalize the input image, landmarks of the image is first obtained by using a pre-implemented face detector and head pose is estimated from the detected landmarks. The face model can be rotated and translated using the estimated head pose. Then a transformation matrix $W = C_nMC_r^{-1}$ is obtained from the transformed face model and applied to the input image to obtain normalized/warped image, which will be feed into the backbone network for training. $C_n$ is the intrinsic matrix for the normalized camera and $C_r$ is the original camera intrinsic matrix obtained from the camera calibration. In Figure 3.2, the example of an unwarped image, the corresponding warped image and the image with annotated ground truth (green arrow) and predicted (pink arrow) gaze direction from the MPII dataset are visualized. The process of obtaining the predicted gaze direction will be explained in detail later in this Section.

Figure 3.2: One sample image (left), the corresponding warped image (middle) and the original image with gaze direction annotations (right).
3.2.2 Backbone Deep Neural Network

After the input image is pre-processed, the warped image is passed to a backbone DNN in order to make predictions. According to the previous section, the pre-processing stage is used to standardize the input images and decrease the number of input pixels so that less computational resources are required by the model. Considering the pre-processing as a auxiliary step, the quality of the prediction mainly relies on the design and training of the backbone DNN. In this work, the performance of ResNet-18, one of the state-of-art CNN models, is going to be compared with that of an external ResNet-50 model, the pure ViTs and the hybrid ViTs using self-determined evaluation metrics. In this chapter, the details of all types of model covered in this project are provided, including model hyper-parameters and training parameters.

3.2.2.1 ResNet-18 and External ResNet-50

Different CNN models have been involved and tested on gaze estimation related tasks. Two typical CNN models are ResNet and MobileNet. ResNet has a relatively deeper architecture compared to MobileNet. It usually consists of more number of layers, including residual connections, which help alleviate the vanishing gradient problem during training. On the other hand, MobileNet has a more light-weight architecture with depth-wise separable convolutions, which significantly reduces the number of parameters and computational complexity. These two types of models are usually used in different scenarios depending on the purpose of the tasks.

In addition to the self-implemented ResNet-18, an external ResNet-50 trained by the authors in [13] is also involved in this project, mainly for comparing the performance of the existing models and our models. The external ResNet-50 is designed and trained as a baseline network that takes the full-face patch covering $224 \times 224$ pixels as input and outputs the horizontal and vertical gaze angles (i.e. pitch and yaw). The ADAM [47] optimizer is utilized with an initial learning rate of 0.0001, and the batch size is set to be 50. The ResNet-50 is trained for 25 epochs and the learning rate gets decayed by a factor of 0.1 every 10 epochs. The number of trainable parameters for both ResNet-18 and the external ResNet-50 are shown in Table 3.1, and the detailed training hyper-parameters of the two models are shown in Table 3.2.
Table 3.1: Summary of number of trainable parameters of the CNN models.

<table>
<thead>
<tr>
<th>Model</th>
<th># trainable parameters ($\times 10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>11.2</td>
</tr>
<tr>
<td>External ResNet-50</td>
<td>25.3</td>
</tr>
</tbody>
</table>

Table 3.2: Summary of training parameters of the CNN models. Note that the learning rate scheduler is slightly different for ResNet-18 and the external ResNet-50. For ResNet-18 the learning rate is 1e-4 for 20 epochs and thereafter 1e-5, while for the external ResNet-50 the learning rates starts at 1e-4 and decays by 0.1 every 10 epochs.

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch size</th>
<th># epochs</th>
<th>Image size</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>128</td>
<td>25</td>
<td>(256, 256)</td>
<td>1e-4 $\rightarrow$ 1e-5</td>
</tr>
<tr>
<td>External ResNet-50</td>
<td>50</td>
<td>25</td>
<td>(224, 224)</td>
<td>1e-4 $\rightarrow$ 1e-6</td>
</tr>
</tbody>
</table>

3.2.2.2 Pure Vision Transformers

In this work, three pure ViTs with different sizes, i.e. small, medium and large, are implemented and compared. The size of a model is represented by the number of hyper-parameters in order to keep track of the size-performance trade-off. It is apparent to get better performance if the size of a model gets increased within a certain range, however, it is not the goal to build a well-performing model that is extremely resource consuming. The initial selection of hyper-parameters is based on [7] and the capacity of the available GPUs.

For each type of pure ViT, including small, medium and large pure ViT, different hyper-parameters are tested and one model that performs the best for each type is reported and compared to the other models covered in this work.

3.2.2.3 Hybrid Vision Transformers

Similar to the pure ViTs, three hybrid ViTs with different sizes, i.e. small, medium and large, are implemented and compared. For each type of pure ViT, including small, medium and large pure ViT, different hyper-parameters are tested and one model that performs the best for each type is reported and compared to the other models covered in this work. The number of trainable parameters for all pure ViTs and hybrid ViTs are shown in Table 3.3, and the detailed training hyper-parameters for the models are recorded in Table 3.4.
For large ViTs, regardless of whether it is a pure or hybrid model, the number of trainable parameters is around twice of that of ResNet-18. For small ViTs, the number of trainable parameters is approximately half of that of ResNet-18 and for medium ViTs, the number of parameters is almost at the similar level to ResNet-18. Note that for all pure ViTs and hybrid ViTs, the following parameters are consistent:

- Batch size = 64
- Number of epochs = 41 for pure ViTs
- Number of epochs = 25 for hybrid ViTs
- Image size = (256, 256)
- Patch size = 16
- Number of heads = 16
- Cosine learning rate scheduler for pure ViTs
- Multi-step learning rate for hybrid ViTs
- Initial learning rate = 1e-4 for pure ViTs
- Initial learning rate = 1e-3 for hybrid ViTs
- Dropout = 0.1
- Embedding dropout = 0.1

Table 3.3: Summary of number of trainable parameters of the pure ViT and hybrid ViT models.

<table>
<thead>
<tr>
<th>Model</th>
<th># trainable parameters ($\times 10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>small pure ViT</td>
<td>4.81</td>
</tr>
<tr>
<td>medium pure ViT</td>
<td>12.7</td>
</tr>
<tr>
<td>large pure ViT</td>
<td>33.8</td>
</tr>
<tr>
<td>small hybrid ViT</td>
<td>4.37</td>
</tr>
<tr>
<td>medium hybrid ViT</td>
<td>12.2</td>
</tr>
<tr>
<td>large hybrid ViT</td>
<td>17.8</td>
</tr>
</tbody>
</table>
Table 3.4: Summary of training parameters of the ViT and hybrid ViT models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Hidden dimension</th>
<th>Depth</th>
<th>MLP dimension</th>
<th>Head dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>small pure ViT</td>
<td>256</td>
<td>12</td>
<td>256</td>
<td>16</td>
</tr>
<tr>
<td>medium pure ViT</td>
<td>256</td>
<td>16</td>
<td>1024</td>
<td>16</td>
</tr>
<tr>
<td>large pure ViT</td>
<td>512</td>
<td>16</td>
<td>1024</td>
<td>32</td>
</tr>
<tr>
<td>small hybrid ViT</td>
<td>256</td>
<td>6</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>medium hybrid ViT</td>
<td>512</td>
<td>6</td>
<td>128</td>
<td>16</td>
</tr>
<tr>
<td>large hybrid ViT</td>
<td>512</td>
<td>16</td>
<td>1024</td>
<td>32</td>
</tr>
</tbody>
</table>

3.2.3 Projection and Post-processing

The raw output of the backbone is given as a 2D vector which contains the pitch $\theta$ and the yaw $\phi$. In order to evaluate the performance of the network, the gaze error, represented as the angular difference between the ground truth gaze vector and the predicted gaze vector, is calculated. This indicates that a 3D predicted gaze vector should be transformed from the 2D raw prediction. According to Figure 3.3, the pitch is defined as the angle between the xz-plane and the 3D gaze vector, and the yaw is defined as the angle between the z-axis and the projection of the 3D gaze vector on the xz-plane. The estimated 3D gaze vector $\hat{\mathbf{g}}_n = (x, y, z)$ can be transformed from the pitch and yaw by using the following formulas:

\[
x = \sin(\phi) \cos(\theta) \tag{3.1}
\]
\[
y = \sin(\theta) \tag{3.2}
\]
\[
z = \cos(\phi) \cos(\theta) \tag{3.3}
\]
Method

It is intuitive that the gaze is estimated from the normalized image and therefore should be transformed back to the un-normalized camera coordinate system, in which the ground truth values also lies. As mentioned in Section 3.2.1, 2D image normalization is achieved by warping the image using the matrix \( W = C_nMC_r^{-1} \). This operation makes an assumption that the eye is a planar object, which means the scaling operation will not change the gaze direction. Using only the rotation matrix, the normalized gaze direction vector can be obtained as \( g_n = Rg_r \). This formulation corresponds to the interpretation that warping an image does not affect the physical space in terms of scaling and therefore, the gaze direction vector is only affected by the rotation matrix \( R \). Based on all this discussion, the estimated gaze direction vector can be projected back to the original camera coordinate system using the formula \( \hat{g}_r = R^{-1}\hat{g}_n \), where \( \hat{g}_n \) is the estimated gaze direction from the normalized image and \( \hat{g}_r \) is the estimation in the original CCS.

Other than evaluating the gaze direction vector, it is also important to look at the screen-based 2D gaze target (a.k.a point of gaze) in this project, i.e. the point on the screen where the subject is looking at. To calculate the coordinates of the target, we need to get access to three pieces of information: 1) the gaze origin, 2) the gaze direction, and 3) the equation of the screen plane.

After gathering all information, we assume that the screen has four corner points in the original CCS, which are denoted as \( p_1, p_2, p_3, \) and \( p_4 \). Then, two non-parallel vectors on the screen plane can be calculated and the normal vector of the plane, \( \hat{n} = (n_1, n_2, n_3) \) is obtained by taking the cross product of the two vectors lying on the plane. As the result, the Cartesian form of

---

Figure 3.3: Representation of pitch and yaw.
one equation of the plane is given as \( n_1x + n_2y + n_3z = n_1p_{1x} + n_2p_{1y} + n_3p_{1z} \), where \( p_{1x}, p_{1y}, p_{1z} \) are the \( x, y \) and \( z \) coordinates of the corner point \( p_1 \). Recall that the gaze ray has \( \vec{g}_r = (g_1, g_2, g_3) \) as the direction and \( \vec{o}_r = (o_1, o_2, o_3) \) as a point lying on the it, the parametric equation of the ray is given by:

\[
\begin{align*}
  x &= g_1t + o_1 \\
  y &= g_2t + o_2 \\
  z &= g_3t + o_3
\end{align*}
\]

where \( t \in \mathbb{R} \). To find the intersection point of the gaze ray and the screen plane, the parametric equation of the line is substituted into the equation of the plane to first determine the value of \( t \):

\[
\begin{align*}
  n_1 \cdot (g_1t + o_1) + n_2 \cdot (g_2t + o_2) + n_3 \cdot (g_3t + o_3) &= n_1p_{1x} + n_2p_{1y} + n_3p_{1z} \\
  t &= \frac{n_1p_{1x} + n_2p_{1y} + n_3p_{1z} - n_1o_1 - n_2o_2 - n_3o_3}{n_1g_1 + n_2g_2 + n_3g_3}
\end{align*}
\]

Once we find the \( t \) value, it can be substituted back into Equations 3.4 - 3.6 to find the coordinates of the intersection point. However, the ground truth target point coordinates are all normalized to \([0, 1]\), which indicates that a transformation between the original CCS and the Screen Coordinate System (SCS) should be defined. This can be done using a transformation matrix \( A \), which links the coordinates of the points from both coordinate systems according to the following formula:

\[
A \cdot \begin{bmatrix} x_{s1} & x_{s2} & x_{s3} \\ y_{s1} & y_{s2} & y_{s3} \\ 1 & 1 & 1 \end{bmatrix}_{SCS} = \begin{bmatrix} x_{c1} & x_{c2} & x_{c3} \\ y_{c1} & y_{c2} & y_{c3} \\ z_{c1} & z_{c2} & z_{c3} \end{bmatrix}_{CCS}
\]

where each column of the CCS matrix represents a point in the original CCS and the corresponding column of the SCS matrix represents the stimulus point in homogeneous coordinates. From equation 3.9, the transformation matrix \( A \)
can be determined by using the formula:

\[
A = \begin{bmatrix}
  x_{c1} & x_{c2} & x_{c3} \\
  y_{c1} & y_{c2} & y_{c3} \\
  z_{c1} & z_{c2} & z_{c3}
\end{bmatrix}_{CCS} \cdot \begin{bmatrix}
  x_{s1} & x_{s2} & x_{s3} \\
  y_{s1} & y_{s2} & y_{s3} \\
  1 & 1 & 1
\end{bmatrix}_{SCS}^{-1}
\] (3.10)

With the help of matrix \(A\) and given a point \(\hat{p}_c\) in the original CCS, its corresponding point in SCS can be found as \(\hat{p}_s = A^{-1} \cdot \hat{p}_c\). Please note that the coordinates of \(\hat{p}_s\) are given in the homogeneous coordinates.

### 3.3 Evaluation Metrics

To evaluate a trained model, three pieces of quantitative information will be taken into consideration: 1) the angular gaze error for all evaluation datasets (i.e. the ETH-XGaze dataset, the MPII-FaceGaze dataset and the internal Tobii dataset) and 2) the stimulus distance error for the ETH-XGaze dataset and the internal Tobii dataset and 3) the number of trainable parameters indicating the size of the models. To evaluate a specific evaluation image, the image will first need to be normalized and sent into the network. Note that there is no ground truth gaze origin collected due to the difficulty and inaccuracy of such collection. In addition, the model does not predict the correct gaze origin. Therefore, the gaze origin is set to be the face center and the position of the face is obtained through an external head pose estimator as mentioned in Section 3.2.1. To achieve a valid evaluation and calculate both the angular gaze error and the stimulus error, the following steps are followed:

1. The 3D gaze origin \(g_o\), i.e. the face center is obtained through the head pose estimator.

2. The predicted 3D gaze direction \(v_{pred}\) is output by the network.

3. The ground truth gaze point on the screen, denoted as \(g_t\) is collected from the metadata of the input image.

4. Then the ground truth gaze vector can be determined by doing: \(v_{gt} = g_t - g_o\).

5. **Angular gaze error** = angle between \(v_{gt}\) and \(v_{pred}\). The formula for
obtaining the angular gaze error is known as:

\[
\text{Angular gaze error} = \cos^{-1}\left(\frac{\mathbf{V}_{gt} \cdot \mathbf{V}_{pred}}{|\mathbf{V}_{gt}| \cdot |\mathbf{V}_{pred}|}\right) \tag{3.11}
\]

6. To find the stimulus distance error, we need to find the position of the predicted gaze point, denoted by \( g_i' \) on the screen. That can be determined by finding the intersection of \( \mathbf{v}_{pred} \) ray and the screen plane. The detailed steps are explained in Section 3.2.3.

7. **Stimulus distance error** = distance between \( g_i \) and \( g_i' \). These two points are given as 3D format and therefore the formula for finding the stimulus distance error can be written as:

\[
\text{Stimulus distance error} = \sqrt{(g_{ix} - g_{ix}')^2 + (g_{iy} - g_{iy}')^2 + (g_{iz} - g_{iz}')^2} \tag{3.12}
\]

where \( g_{ix}, g_{iy} \) and \( g_{iz} \) are the \( x, y \) and \( z \) coordinates of the ground truth gaze point and \( g_{ix}', g_{iy}', g_{iz}' \) are the \( x, y \) and \( z \) coordinates of the estimated gaze point.

Note that the stimulus distance error will only be calculated for the Tobii dataset because the screen information is not provided in the MPII-FaceGaze dataset.

To better present the experimental results, the **mean errors** are calculated over all evaluated image samples, mainly because some referenced work also use the mean error metric and it would be easier to check the validity of our results by using the same metrics. Both the angular gaze error and the stimulus distance error follow the same error calculation metric. The mean error can be calculated using the following equation:

\[
\text{Mean error} = ME = \frac{\sum_{k=1}^{N} e_k}{N} \tag{3.13}
\]

where \( N \) is the total number of samples and \( e_k \) is the error (either angular error or distance error) for the current sample.

In addition, due to the privacy of the internal Tobii dataset, all results will be given in a relative form. To explain this in a better way, we set the baseline model to be the self-trained ResNet-18 and its error is denoted as \( ME_{\text{resnet}} \). Imagine that we have another model \( x \) and want to compare the performance of \( x \) relative to the baseline ResNet-18. Then the relative percentage is given
as:
\[
\text{Relative percentage} = \frac{ME_x - ME_{\text{resnet}}}{ME_{\text{resnet}}} \times 100\%
\] (3.14)

where \( ME_x \) is the mean error of the model \( x \). It is worth mentioning that when the relative percentage is a positive value, it indicates that model \( x \) has a larger mean error than the baseline model and that further proves that the model \( x \) performs worse than the baseline model. In contrast, if the relative percentage is a negative value, then the model \( x \) outperforms the baseline model since it is getting a smaller mean error value.

Last but not least, the number of trainable parameters is computed to evaluate the computational performance of the trained models. These trainable parameters are introduced in different layers (e.g. the Multi-Head Attention layers in the ViTs, the convolutional layers in the CNNs etc.) and added up for each individual model.
Chapter 4

Results and Analysis

In this chapter, the results for all experiments conducted in this project are presented and discussed. Firstly, the evaluation results of all models on all evaluation datasets are presented separately, followed by a section summarizing the findings and giving additional information. There are in total eight models being evaluated: an external ResNet-50, a ResNet-18, three pure ViTs of different sizes and three hybrid ViTs of different sizes, where the ResNet-18 is selected as the baseline model.

4.1 Evaluation Results on ETH-XGaze

The evaluation results on the ETH-XGaze dataset is presented in Table 4.1. For ETH-XGaze, it is slightly different from the other two evaluation datasets because it is both the training dataset and the evaluation dataset, and the evaluation is done together with the training process. As explained in Section 3.3, the relative Mean Angular Gaze Error (MAGE) and the Mean Stimulus Distance Error (MSDE) are calculated for all models. In Table 4.1, both the MAGE and the MSDE value are missing for the external ResNet-50 because the model is pre-trained but not a model trained by us, therefore, the evaluation during training can not be performed for external models. In addition, both the MAGE and the MSDE are zero for ResNet-18 because it is selected as the baseline CNN model, which the rest of the models are compared to.

According to the results shown in Table 4.1, both the MAGE and the MSDE are showing a decreasing trend for pure ViTs when the size gets larger, which means a more accurate prediction with smaller error is made for the larger pure ViTs. However, none of the trained pure ViTs manages to perform better than the baseline model. In addition, it is essential that further increasing
the size of the ViT will introduce an extra significant number of trainable parameters. Therefore, a trade-off between the size of the pure ViTs and the prediction accuracy should be taken into consideration. According to the current results, the size of the pure ViT needs to be increased a lot if better performance than the baseline is expected.

When it comes to the hybrid ViTs, they perform much better than the pure ViTs in general in terms of both smaller MAGE and smaller MSDE. Similar to the pure ViTs, the hybrid ViTs perform better when the size of the model increases. It is notable that both the medium and large hybrid ViTs outperform the baseline model by certain extent due to the fact that the relative errors are presented as negative values.

In summary, according to the scatter plots shown in Figure 4.1 and Figure 4.2, among all trained models on the ETH-XGaze dataset, the large hybrid ViT demonstrates the best performance in terms of the smallest MAGE and MSDE. However, the number of trainable parameters for the large hybrid ViT is approximately twice as much as that of ResNet-18. When taking the size of the model into account, the medium hybrid ViT also performs better than the baseline model but has a much smaller size than the large hybrid ViT. The trade-off between size of model and performance should be considered when choosing a model between the medium hybrid ViT and large hybrid ViT.

Table 4.1: Evaluation results on ETH-XGaze.

<table>
<thead>
<tr>
<th>Model</th>
<th>MAGE (%)</th>
<th>MSDE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>External ResNet-50</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>0 (baseline)</td>
<td>0 (baseline)</td>
</tr>
<tr>
<td>small pure ViT</td>
<td>27.5</td>
<td>21.0</td>
</tr>
<tr>
<td>medium pure ViT</td>
<td>14.1</td>
<td>11.5</td>
</tr>
<tr>
<td>large pure ViT</td>
<td>8.87</td>
<td>8.80</td>
</tr>
<tr>
<td>small hybrid ViT</td>
<td>1.52</td>
<td>1.13</td>
</tr>
<tr>
<td>medium hybrid ViT</td>
<td>-2.16</td>
<td>-1.35</td>
</tr>
<tr>
<td>large hybrid ViT</td>
<td>-3.03</td>
<td>-1.81</td>
</tr>
</tbody>
</table>
Figure 4.1: The scatter plot of the number of trainable parameters vs. the relative MAGE on ETH-XGaze.

Figure 4.2: The scatter plot of the number of trainable parameters vs. the relative MSDE on ETH-XGaze.
4.2 Evaluation Results on internal Tobii dataset

The evaluation results for the internal Tobii dataset is presented in Table 4.2. The scatter plots of the number of trainable parameters vs. the relative errors for all models are visualized in Figure 4.3 and Figure 4.4. Although it is more meaningful to look into the MSDE values, the gaze errors are also calculated to make the format of the table be consistent with that of other datasets. Similar to ETH-XGaze, both errors are recorded as zero for the baseline model, a positive percentages indicates that the evaluated model performs worse than the baseline and a negative percentage indicates that the evaluated model outperforms the baseline model.

Firstly, the external ResNet-50 performs worse than ResNet-18 but outperforms all pure ViTs. It is interesting to see that the external ResNet-50 makes a better prediction for the gaze direction but a worse gaze point prediction than the small hybrid ViT.

To compare the performance of all pure ViTs, both MAGE and MSDE are again showing a decreasing trend when the size of the ViT increases. When comparing the results of pure ViTs from Table 4.1 and Table 4.2, the pure ViTs performs much worse on the internal Tobii dataset than on ETH-XGaze. This is reasonable because the ETH-XGaze dataset is also used as the training dataset, which means similar images have been processed by the model and that makes it easier for the model to make predictions on images belonging to the same dataset. Again, neither of the pure ViTs surpasses the baseline.

When looking into the hybrid ViTs, both medium and large hybrid ViTs perform better than the baseline. The MAGE and the MSDE value of the medium hybrid ViT are marked as red because that is the best-performing model for the internal Tobii dataset. This result differs from that in Table 4.1 and it is not the largest hybrid ViT that gives the best prediction accuracy. One possible reason is that ViTs are quite sensitive to hyper-parameters of training and the selected hyper-parameters for the large hybrid ViT may not be the optimal choices. Another possible reason is that there is always stochasticity introduced during the training process, therefore, the values might change when the same model gets trained multiple times. However, due to the limited time and resources, only a limited amount of models can be trained during the given period of time.

To summarize the findings, according to Table 4.2, Figure 4.3 and Figure 4.4, the medium hybrid ViT surpasses all other models in terms of a smaller
MAGE and MSDE. This can be regarded as a valuable finding because the medium hybrid ViT is almost of the same size as the baseline ResNet-18.

Table 4.2: Evaluation results on internal Tobii dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>MAGE (%)</th>
<th>MSDE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>External ResNet-50</td>
<td>5.29</td>
<td>2.82</td>
</tr>
<tr>
<td>ResNet-18 (baseline)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>small pure ViT</td>
<td>30.9</td>
<td>33.7</td>
</tr>
<tr>
<td>medium pure ViT</td>
<td>27.4</td>
<td>30.3</td>
</tr>
<tr>
<td>large pure ViT</td>
<td>24.5</td>
<td>25.1</td>
</tr>
<tr>
<td>small hybrid ViT</td>
<td>5.85</td>
<td>2.26</td>
</tr>
<tr>
<td>medium hybrid ViT</td>
<td>-7.52</td>
<td>-7.76</td>
</tr>
<tr>
<td>large hybrid ViT</td>
<td>-1.53</td>
<td>-1.41</td>
</tr>
</tbody>
</table>

Figure 4.3: The scatter plot of the number of trainable parameters vs. the relative MAGE on Tobii dataset.
4.3 Evaluation Results on MPII-FaceGaze

The evaluation results for the MPII-FaceGaze dataset is presented in Table 4.3 and the scatter plot of number of trainable parameters vs. MAGE is visualized in Figure 4.5. As mentioned before, only the gaze error is recorded for the MPII dataset because no screen information is provided and therefore, no distance error in mm can be obtained.

Similar to the previous two sections, the MAGE value is set to zero for the baseline model and the relative MAGE is given for other models in order to compare the performance. First of all, the external ResNet-50 performs worse than the baseline model by 3.02% but outperforms all pure ViTs of different sizes.

When it comes to the pure ViTs, the performance gets improved when the number of parameters increases. However, none of the pure ViTs surpasses either the baseline model or the external ResNet-50 model. Regarding the hybrid ViTs, the performance also gets enhanced when the size of the model increases. The only model that outperforms the baseline is found to be the large hybrid ViT but it does not surpass it by a significant amount but only 0.68%. Therefore, this is most likely not extremely solid evidence to show that the large hybrid ViT is actually the best-performing model because of the
stochasticity introduced during training.

When comparing the evaluation results for all three evaluation datasets, MPII-FaceGaze is determined to be the dataset, for which the performance difference between the baseline and the best-performing model is the smallest. This does not indicate that the MPII-FaceGaze is the hardest dataset because only relative results are presented. It can only be preliminarily concluded that neither the pure ViTs nor the hybrid ViTs should replace the CNNs for the MPII-FaceGaze dataset in gaze estimation tasks. The reasoning for not using pure ViTs on MPII-FaceGaze is that they all perform worse than the baseline CNN and the reason for not using hybrid ViTs on MPII-FaceGaze is that the large hybrid ViT does not surpass the baseline to a large extent but has more number of trainable parameters.

Table 4.3: Evaluation results on MPII-FaceGaze.

<table>
<thead>
<tr>
<th>Model</th>
<th>MAGE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>External ResNet-50</td>
<td>3.02</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>0 (baseline)</td>
</tr>
<tr>
<td>small pure ViT</td>
<td>16.8</td>
</tr>
<tr>
<td>medium pure ViT</td>
<td>10.3</td>
</tr>
<tr>
<td>large pure ViT</td>
<td>5.63</td>
</tr>
<tr>
<td>small hybrid ViT</td>
<td>9.34</td>
</tr>
<tr>
<td>medium hybrid ViT</td>
<td>3.98</td>
</tr>
<tr>
<td>large hybrid ViT</td>
<td>-0.68</td>
</tr>
</tbody>
</table>
Figure 4.5: The scatter plot of the number of trainable parameters vs. the relative MAGE on MPII-FaceGaze.

### 4.4 Summary and Additional Information

In summary, the evaluation results for all evaluation datasets are showing a similar trend. In general, the baseline model, i.e. the ResNet-18, performs quite well and can hardly be surpassed. The external ResNet-50 performs similar to the baseline but still gives less accurate predictions than ResNet-18 for both the internal Tobii dataset and the MPII dataset (the results for ResNet-50 on ETH-XGaze is not applicable because this is not a self-trained model and ETH-XGaze is evaluated together with the training process).

In addition, the pure ViTs all have trouble outperforming the baseline model, although the overall performance gets enhanced when the size of the model increases. When it comes to the hybrid ViT models, the large hybrid ViT is proved to be the best-performing model in terms of the smallest MAGE and MSDE value for the ETH-XGaze dataset and the MPII dataset, while the medium ViT is found to be the model that gives the most accurate prediction for the internal Tobii dataset.

When the medium hybrid ViT outperforms the baseline model, it is identified as the most suitable model for that specific dataset because it has the similar number of trainable parameters as the baseline. However, the discussion remains if the large hybrid ViT is the best performing model in
terms of the smallest errors. The main reason for this is that the large hybrid ViT has twice the number of parameters as the baseline model. In this case, it is important to examine the significance of the improvement gained by switching from the baseline model to the large hybrid ViT, i.e. to investigate whether the magnitude of such improvements is practically important or it is merely a marginal gain. This may differ depending on where the model is employed and whether the computational complexity or model size matters more than the prediction accuracy or the other way around.

In addition to the results presented in Section 4.1 - 4.3, some extra information can be briefly mentioned. It is mentioned in Section 3.2.2.3 that the cosine learning rate is used during the training of all pure ViTs and the multi-step learning rate is deployed for all CNN models and all hybrid ViTs. It is worth mentioning that both learning rate schedulers have been tested on all self-trained models in this project and the final choice of learning rate scheduler is determined based on the training and evaluation performance in terms of MAGE and MSDE.

Hybrid ViTs and the CNNs both perform better under the multi-step learning rate scheduler may due to their inherent architecture, which consists of multiple layers of abstraction. The multi-step learning rate scheduler allows the model to converge by adjusting the learning rate at appropriate stages, aligning with the hierarchical nature of the layers in these model. The step-wise reduction in learning rate helps to refine the representations captured by different layers and achieve better performance. In contrast, pure ViTs are solely based on transformer architecture, which is known for the attention mechanism. The cosine learning rate scheduler facilitates the convergence of transformers by conducting a more gradual adjustment of the learning rate, which is beneficial for the self-attention mechanism presented in the pure ViTs.
Chapter 5

Conclusions and Future work

5.1 Conclusions

To conclude the results obtained in this project, we compare the performance of all trained models in terms of the angular gaze error, the stimulus distance error and the size of the model, which is quantified by the number of trainable parameters. In general, the hybrid ViTs performs better than the pure ViTs to a large extent. In addition, no matter for the pure ViTs or the hybrid ViTs, the performance is improved when the size of the model increases in most cases, although there are special cases where the medium hybrid ViT surpasses the large hybrid ViT (more specifically, for the internal Tobii dataset). This phenomenon may due to the stochasticity introduced in the model and different results might be obtained when the same model is trained multiple times.

When comparing the performance of the ViTs with the CNNs, i.e. the baseline ResNet-18 and the external ResNet-50, the pure ViTs all perform worse than the CNNs regardless of the size of the model. However, there is always one hybrid ViT that gives more accurate predictions than the CNNs for all evaluation datasets. This proves that combining the local pattern capturing ability of CNNs and the global contextual modeling capability of Transformers gives a more satisfactory result in terms of smaller error. Nevertheless, the large models are slower than the medium and the small models, even though the large hybrid ViT is the model that gives the most accurate predictions in most cases.

Therefore, regarding the selection of model in real-life scenarios, the selection may vary depending on the specific use case. In some situations, where accuracy takes precedence over time and resource efficiency, opting for a highly accurate model, i.e. the best-performing hybrid ViT, even if it is
computationally expensive, is justifiable. However, in cases where speed is a more critical factor than accuracy due to real-time requirements, prioritizing models that can provide reasonably accurate predictions within strict time constraints becomes more important. Thus, the choice of model should align with the specific requirements and constraints of the intended application.

5.2 Future work

Due to temporal constraints, several untested ideas remain that merit exploration in future research endeavors. Primarily, empirical evidence suggests that Transformers exhibit enhanced performance when subjected to pre-training on substantial datasets [48]. When pre-trained on a large amount of data, Transformers can acquire a broad range of knowledge and representations, which can be beneficial for subsequent fine-tuning on specific tasks. The benefits of pre-training Transformers on a large amount of data can be attributed to different factors. First of all, the enormous number of data offers a diverse range of examples, enabling the model to learn a comprehensive set of features and patterns associated with gaze behavior. This enhances the generalization performance of the Transformer for the unseen data. Secondly, pre-training on general tasks, such as face recognition can be beneficial for gaze estimation. By leveraging the wealth of annotated data for general tasks, the model learns to extract useful features that are transferable to gaze estimation. This is known as transfer learning and can allow the model to improve the gaze estimation performance by using the knowledge acquired from the pre-training process.

Moreover, personal calibration presents a promising avenue for potentially improving the performance of gaze estimation models, despite the inherent challenges and time-intensive nature associated with its implementation. The process of personal calibration involves customizing the gaze estimation model to the unique characteristics and behaviors of each individual user, thereby tailoring the model to their specific gaze patterns. One of the key advantages of personal calibration is that it is able to account for inter-individual variations in eye geometry, appearance, etc. These variations help to improve the accuracy of gaze estimation, as they introduce complexity into the training process. Additionally, personal calibration facilitates the correction of systematic biases inherent in the gaze estimation process. The biases can be introduced because of the misalignment of the camera, differences in eye physiology for individuals, etc. Therefore, it is worth trying to introduce personal calibration to the training process in the future and see how much the gaze estimation
Conclusions and Future work

performance can be enhanced.
References


