DEPARTMENT OF ENGINEERING MATHEMATICS

Developing Data-Driven, Deep Learning Based Approaches for Key Eye Feature Localisation

Uchit Bhadauriya

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

Thursday 19th September, 2024

Supervisor: Gabriella Miles

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Uchit Bhadauriya, Thursday 19th September, 2024

Ethics Statement

This project fits within the scope of ethics application 97842, as reviewed by my supervisor Gabriella Miles.
I have completed the ethics test on Blackboard. My score is 12/12.

Contents

List of Figures

List of Tables

Abstract

This study presents the development of a deep learning-based system for eye centre localisation using eye-tracking data. The project leverages the InceptionResNetV2 architecture, a convolutional neural network (CNN) known for its robust feature extraction capabilities, to predict the coordinates of eye centres with high precision. The key aim of this research was to enhance the accuracy of eye coordinate prediction while ensuring the model was generalised well across different users and challenging conditions such as subtle variations in eye positions.The research hypothesis proposed that a deep learning model using InceptionResNetV2 would outperform traditional methods in eye centre localisation tasks, achieving greater accuracy and generalisation across diverse datasets. Through data pre-processing, including normalisation, resizing, and data augmentation techniques like random contrast and brightness adjustments, the model was optimised for real-world conditions. The model achieved strong results, with a very low test loss, mean absolute error (MAE), and the relative error, demonstrating its effectiveness in accurately localising eye centres in images. However, fine-tuning the model by unfreezing layers did not yield significant further improvements, suggesting the pre-trained features of the InceptionResNetV2 were already well-suited to the task.While the main objective of precise eye centre localisation was successfully achieved, future work could focus on expanding the dataset to improve generalisability and exploring the relationship between eye and head movement for more comprehensive gaze-tracking systems.
The latter aspects should (ideally) be presented as a concise, factual list of the main points of achievement. Again the points will differ for each project, but an might be as follows:

- I spent 130 hours reading the various research papers and gaining knowledge about my project.
- I spent 200 hours to gain the knowledge of how to write code for this specific project and various deep learning terms related to my code.
- I spent 170 hours trying to solve every error related to my project.
- I spent 100 hours to write my thesis.

GitHub Repository

Supporting Technologies

- I used public-domian Python Libraries.
- I used Tensorflow, Keras and Jupyter.
- I used LaTeX to format my thesis, via the online service *Overleaf*.

Notation and Acronyms

MSE	:	Mean Square Error
MAE	:	Mean Absolute Error
MAPE	:	Mean Absolute Percentage Error
f(x)	:	Function of x.
ReLu	:	Rectified Linear Unit
CNN	:	Convolutional Neural Network

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Gabriella Miles, for her invaluable guidance, support, and encouragement throughout the course of this project. Her expertise and insightful feedback have been instrumental in shaping the direction of my research and ensuring its successful completion. Gabriella's patience and dedication to helping me overcome challenges have made this project a rewarding and educational experience.

I am truly grateful for the time and effort she has invested in mentoring me and for providing me with the resources and knowledge necessary to develop my skills in deep learning. This project would not have been possible without her continued support, and I feel privileged to have had the opportunity to work under her supervision.

Thank you, Gabriella, for everything.

Chapter 1

Introduction

The main goal of this project is to examine the effectiveness of modern deep learning approaches to predict eye coordinates (lx, ly, rx, ry) in the case of eye tracking data, potentially, yielding meaningful insights, in particular, in the area of cognitive load and age gap detection. Our approach is based on prior works exploring eye saccades as proxies for underlying cognitive processes. Prediction of eye coordinates allows for capturing documented differences in eye movement patterns, a phenomenon which could potentially gain clinical significance in cognitive health medications. Moreover, it will constitute an important step for the development of real-time systems for cognitive load detection, serving as a valuable tool for researchers, clinicians and developers of adaptive learning systems, human-computer interaction interfaces or health monitoring tools[19].

Along these lines, researchers have recently been devoting more attention to the use of machine learning to predict eye coordinates in real-time and uncover how they relate to brain load and human factors such as age and behaviour. Predicting eye movements, including fixations, saccades and eye coordinates in general, lies at the core of quantifying brain load and how it varies across different age groups. This has a wide range of applications in cognitive health monitoring, human-computer interaction, and adaptive learning systems, such as rearranging how pages present information depending on how immersed an individual is in learning[15] .

This project focuses on building a robust system capable of predicting cognitive load and age groups using eye-tracking data. At the core of this system is the accurate prediction of the lx, ly, rx, and ry coordinates, which represent the positions of the left and right eyes in images captured by eye-tracking devices. These eye centre coordinates are crucial for understanding eye movement patterns, which are directly linked to cognitive processing and behavioural characteristics[15].

The main goal of this thesis is to design and fine-tune a deep learning model, which is very successful for visual recognition problems, in order to train and test it on real datasets that hold eye images. We will train and test by minimising the relative error on eye centre and pupil prediction and pay special attention to removing outliers, in which eye centre predictions can be poor (for example, when the coordinates of the left and right eye are similar) and hurt the model performance[15].

Then this project commits to the field an initial systematic effort to make eye-tracking systems more accurate, by offering an approach that involves systematically benchmarking a proposed model, through tuning, testing and evaluation, until the accuracy achieved is sufficient to support a dramatic expansion in the range and reliability of eye-tracking applications that might affect our lives, in practical domains such as virtual-reality systems, user interfaces and accessibility technologies.

1.1 Importance of the Topic

However, eye movement behaviour is a valuable marker of cognitive function and neurodegenerative pathology. Neurodegenerative diseases are age-related, and early detection and treatment have a considerable influence on course and outcome. Eye-tracking data offers a non-invasive measure of cognitive load

that could help monitor cognitive decline. Understanding cognitive load could also inform improvements of human-computer interaction, improving the adaptability and intelligence of computerised systems.

Given the rapid potency of new AI-based technologies in automating complex tasks and the more recent explosion of deep learning models in this real-time, this integration into eye-tracking analytics is timely and important. Pending further analysis before extensive use, this system could be of great societal value for current and future stakeholders: clinicians seeking accessible and reliable early-detection tools, data scientists interested in behavioural analytics, or even end-users looking to rely on cognitive monitoring systems for personal health evaluations.

And so, this prospective use is that for our project (using a deep learning model to predict the eye coordinates, lx , ly , rx , ry) in the future, eye movement behaviour will be the key aspect to track. Eye movement behaviour has been considered a rich and reliable indicator of cognitive load for a long time. For example, when we set individuals a task which incurs a higher cognitive load on our mind, we tend to see that there is a slow down of*saccadic eye movements (rapid eye movements between fixations) and a lengthening of *fixation* (slow stable phases of eye movements) durations. If we are really precise with our predictions of lx , ly , rx and ry, then we can track such changes on the fly and then find useful metrics resulting from them, such as saccade speed and fixation durations. And so, we can derive an indirect but reliable estimate of the mental effort exerted by the individual on the task.

1.2 Some Examples of how my project can help in future projects:-

1. During **fixation**, the lx, ly (left eye) and rx, ry (right eye) coordinates remain relatively stable over a period of time. This lack of movement in the coordinates indicates that the eyes are fixating on a particular point. By tracking how long the eye coordinates remain within a small range of movement (i.e., minimal variance in lx, ly, rx, ry), we can measure the fixation duration. Longer stability in these coordinates suggests a longer fixation duration[19].

2. **Saccades** are represented by sudden, significant changes in the lx, ly, rx, and ry coordinates. By calculating the distance between successive sets of eye coordinates and identifying when there is a rapid shift in the positions (i.e. when the coordinates change significantly between two-time frames), we can detect a saccade[19].
The speed and direction of saccades can also be derived from the rate of change in the eye coordinates.

3. **Gaze patterns** can be reconstructed by plotting the sequence of the eye's lx, ly, rx, and ry coordinates over time. As the eyes move across different points in the visual field, the coordinates trace the path of the gaze[19].

The pattern of these movements helps create a gaze map that shows where the person looked, how long they fixated on different points, and the path their eyes took as they moved between those points (saccades)[19].

The density and flow of these eye positions, derived from the coordinates, can reveal the overall strategy of visual exploration.

Moreover, age-related differences in eye movement behavior are well-documented. As people age, their eye movements generally become slower and less dynamic, reflecting changes in cognitive processing abilities. By analyzing the predicted eye coordinates, this project will help in the future to differentiate between age groups and understand how cognitive load varies with age. This differentiation is especially useful in cognitive health applications, where early detection of age-related cognitive decline can be supported by monitoring eye movement patterns[19].

The main **challenges** in the deep learning-based prediction of eye coordinates are the complexity of the eye movement data and the precision involved when modelling. Accuracy is of utmost importance in these predictions, as small errors in predicting the (lx, ly, rx, ry) coordinates can result in large deviations when converted in practical applications.

Besides, you also want to give the model enough complexity to work well, yet keep it computationally fast to evaluate, especially when it comes to prediction. Lastly, you have to think about overfitting – explaining why your model works so well on a given training set, yet fails miserably when it comes to

Figure 1.1: Sample of Images

out-of-sample predictions on test datasets that are different from the ones you used for the training set. This problem of overfitting is a persistent challenge for machine and deep-learning tasks.

1.3 Aims, Objectives, and Achievements:

1.3.1 Aims

- Develop a deep learning model for eye center localization using eye-tracking data.
- Enhance the precision of eye coordinate prediction.
- Improve generalization of the model.

1.3.2 Objectives

- Pre-process and augment real-world eye image datasets for effective model training.
- Propose potential improvements in model architecture or dataset handling to enhance accuracy in eye-tracking systems.
- Evaluate the model's performance based on relative error, particularly in challenging scenarios like near-similar eye coordinates.
- Validate the model's generalizability across different user groups using real-world datasets.

1.3.3 Achievements

- Successfully developed a deep learning model that predicts eye center localization with high accuracy.
- Demonstrated that the predicted coordinates can derive eye movement metrics correlating with cognitive load and age group.
- Gained insight into key challenges such as precision, generalization, and computational efficiency in eye coordinate prediction.

Chapter 2

Background

2.1 Contextual Background

In this Brunye *et al.* [1] paper, you will read then you will found out that There has been a tremendous improvement in the technology of eye-tracking in recent years that gives us a glimpse into how the eyes of humans work in order to learn, perceive, process and understand their surroundings. Known as the most reasonable approach to predict eye coordinates, including both left-eye and right-eye positions, is also a highly significant data point used in diverse applications such as human-computer interaction, cognitive load and biometric authentication. Kuo *et al.* [11] explores that; Recent advances in deep learning have made predicting eye coordinates from images or video frames a viable avenue as it can accommodate high-quantity data and nonlinear patterns.

2.1.1 Eye-Tracking and its Relevance

Li *et al.* [12] illustrates why it is important to predict eye coordinates for many various tasks and diseases. Skaramagkas *et al.* [19] link various metrics to cognitive load and mental capability. Fixing the eyes' gaze onto the object of interest can provide valuable information to deduce a person's state of mind and emotional status, and thus predict eye coordinates have huge potential in many fields. Knowing where and for how long a person is looking at any focal point of interest, it will be feasible to draw inferences about a person's cognitive workload, fatigue, decision-making processes, etc. Moreover, predicting eye coordinates has been successfully used to recognise neurological disorders. Reasonably accurate data on eye movement is required by psychologists, marketers and medical diagnosticians. Yet, even in sufficient lightning, with the frontally facing face captured by high-quality cameras, predicting the accurate coordinates remains a formidable task.

2.1.2 The Role of Deep Learning in Eye-Tracking

Miles *et al.* [15] explain why a deep learning model is needed and also tells about the gap between traditional and modern approaches. Deep learning is expected to be a suitable approach and solution to bridge the gap between traditional and modern needs by predicting the eye coordinates. Recent advancements in computer vision, particularly deep learning, have provided alternative methods to predict eye coordinates using conventional cameras. These methods leverage convolutional neural networks (CNNs) to identify and predict the precise positions of the eyes in an image or video frame. Deep learning has proven highly effective for tasks like image classification, object detection, and facial recognition. CNNs can capture hierarchical patterns in images, making them particularly suited for detecting and predicting eye coordinates from visual data. Traditional eye-tracking systems rely on specialized hardware such as infrared cameras, which can be expensive and limited in scope. In contrast, deep learning-based approaches offer the potential to create more accessible, software-driven solutions that work with conventional camera setups, opening up new possibilities for widespread use.

In this project, we investigate the predictive ability of deep learning models to infer the eye coordinates (x, y-coordinates for the left and right eye: lx, ly, rx, ry) from images. The key idea behind this project

is to leverage deep learning models that learn from hundreds of thousands of eye images that were annotated, by exploiting the CNNs as feature extractors from the input image, aiming to transfer these learned patterns for predicting the eye positions in live video in an accurate and robust way even under different conditions (such as different light or different cameras). Starting from pre-trained models such as InceptionResNetV2 that have proven to perform more reliably at eye-based image-level cues help to fine-tune the model performance so as to reduce the error in the predicted eye positions to be used at higher level cognitive load tasks that have limited target prediction windows.

2.1.3 Significance of Eye Coordinates Prediction

Kim *et al.* [10] explain why this project is important and open future opportunities related to this project that's why I make this type of project based on novel future innovation opportunities. Predicting eye coordinates accurately is a fundamental task in eye-tracking because it serves as the basis for extracting further metrics such as fixation duration, gaze direction, and saccadic movements. They are crucial in various areas:

Cognitive Load Measurement:

The ability to predict eye coordinates in real-time can aid in assessing the mental effort exerted by individuals while performing specific tasks. For example, increased fixation duration or changes in gaze patterns can indicate cognitive overload or stress. By understanding these indicators, adaptive systems can be created to enhance learning environments or improve user experience in interfaces[10].

Biometric Authentication:

Eye-tracking can be used as a biometric tool, as each individual has unique patterns of eye movement. Predicting the accurate coordinates of the eyes enables systems to create models of typical gaze behavior, which can then be used for identification and authentication purposes[10].

Healthcare Applications:

Irregularities in eye movements can be an early sign of degenerative neurological conditions, such as Alzheimer's disease or Parkinson's. Using learned models to predict eye coordinates essential for human vision, and detecting deviations from normative patterns of gaze, deep learning models can do more than just recognise faces. They can also aid in the early diagnosis of such conditions and enable the monitoring of disease progression in patients with neurological disorders[10].

2.2 Technical Background

2.2.1 DataSet and Data Preprocessing and Augmentation

From the OSF website, I downloaded EM-COGLOAD dataset, in which my supervisor Gabriella Miles contributed. There are 4 types of datasets, Participant's Image Data, Simon Test Dataset, Extracted Eye movement Traces and Eye State Detection and Eye Centre Localisation datasets. I chose Eye State Detection and Eye Centre Localisation datasets in which I chose Eye Centre Localisation labels and images[15].

Data preprocessing is a critical step in ensuring that the model performs well across different environments and lighting conditions. Gabriella Miles github gives a clear picture of data processing related to this dataset and project. In this project, images are resized to a uniform dimension of pixels, which is the input size required by the InceptionResNetV2 mode as per the pre-built model of Keras. Additionally, Normalisation improves the training process influenced by Lin *et al.* [13]. Given the sensitivity of eye-tracking data to lighting conditions and occlusions, data augmentation techniques are applied to make the model more robust. Shorten *et al.* [18]gives the clear picture of all types of image data augmentation techniques. These techniques include: Brightness Adjustment: The brightness of the images is varied to account for different lighting environments. Contrast Adjustment: Contrast variations are introduced to make the model more resilient to image quality changes.

2.2.2 TensorFlow Data Pipeline

I got the idea to use the TensorFlow data pipeline as compared to others by reading the work of Caveness *et al.* [2] and which will be beneficial for my project. The data pipeline in TensorFlow is a crucial component for efficiently managing the flow of data from disk to the model during training, validation, and testing. A well-designed pipeline ensures that the model receives data in an optimal way, improving both speed and performance. In the context of this project, where images are processed to predict eye coordinates, the TensorFlow data pipeline plays a significant role in handling image preprocessing, augmentation, and batching.

2.2.3 The InceptionResNetV2 Model

One popular model architecture is InceptionResNetV2, which is pre-trained on the ImageNet dataset and is intended to work with larger amounts of image data. This architecture can accurately recognise thousands of different kinds of objects and is typically used in large-scale tasks that require high accuracy for image recognition and feature extraction. There are numerous such pre-built models available on the Keras. I am not working on a laptop with powerful processing power, that's why I chose InceptionResNetV2 because InceptionResNetV2 has decent accuracy with a decent number of layers with less time per inference step.

2.2.4 Transfer Learning

Torrey *et al.* [20] is very essential for those people who don't have good processor facilities and then conclude the use of transfer learning as a whole with the advantages. Transfer learning is the machine learning method to learn a task and then reuse the obtained model as the starting point of the model to learn another, related task. The assumption is that something learned for one task can help in something similar, as learning from large annotated data enables faster and more accurate modelling in a new, smaller dataset.

In this project, transfer learning is used via the InceptionResNetV2 model, which was pre-trained using the ImageNet dataset. It has learned to identify millions of features, such as edges, textures and shapes, and can be used as a feature extractor for other tasks.

2.2.5 Global Average Pooling

GlobalAveragePooling2D is one of the most important building block of the architecture which is used for this project, particularly in the Convolutional Neural Networks (CNNs). It plays an important role in moving from feature extraction to prediction in the InceptionResNetV2 model. Zhang *et al.* [24] provides information as to why global average pooling is better than others for my project.

2.2.6 ReLU Activation Layer

Another key element is a particular activation function, called the Rectified Linear Unit (ReLU). This is the final function computed in each step of the deep learning model, and it is applied to the InceptionResNetV2 used for this project as well. This simple activation function has a huge impact on how the model can learn more complex relationships between inputs and the desired output. Roughly speaking, for every input value passed into ReLU, if it is negative it is set to zero, whereas if it is positive it is kept. This way the training data is not 'saturated', and a vanishing gradient problem is avoided. This helps the model make faster progress in training, learning more interesting features than if it was not used. Hara *et al.* [5]

Since the majority of layers in our InceptionResNetV2 architecture uses the ReLU, it learns the subtle details of the facial images that help in predicting the accurate eye coordinates. So, if there are ReLU activation layers and convolutional layers, the model can point out the features that are correlated to both left and right eyes - for the exact location of eye coordinates. ReLU Activation Layer popular paper on convolutional neural network based on improved Relu piecewise activation function gives the general outlook of the ReLU Activation Layer.

2.2.7 Linear Activation Layer

The activation functions enable the neural networks to add non-linearity to your model output, making it sophisticated enough for the model to capture complex relationships in my data. But sometimes

non-linearity is not needed, say in the case of regression tasks. In my case, I might reach for a linear activation function. The linear activation function is just the identity function, making the value of the output directly proportional to the input value. Unlike the ReLU or any other type of activation functions, the linear activation does not change the value of the input. Activation function research paper provides a summary of the Activation layer. This understanding is comes from Wang *et al.* [21].

2.2.8 Loss Function and Error Metrics

For this regression task, where the goal is to predict the precise coordinates of the eyes, the model is trained to minimize mean squared error (MSE) between the predicted coordinates and the ground truth values. MSE is chosen because it penalizes larger errors more heavily, encouraging the model to be as accurate as possible. Hodson *et al.* [7] helped me to figure out which metrics are suitable for my project like MSE, MAE, MAPE, etc. In addition to MSE, custom error metrics are used to evaluate model performance:

Mean Absolute Error (MAE):

Measures the average error between the predicted and true coordinates, providing a simple measure of model performance[7].

Relative Error:

A custom metric that accounts for the natural distance between the left and right eyes. It calculates the Euclidean distance between predicted and actual coordinates, normalized by the inter-eye distance, ensuring that errors are scaled appropriately based on the size of the face. Jesorky *et al.*[9] 's paper is the basis of my approach which is related to the Hausdorff Distance and it helps me a lot to better understand the project. If we choose other metrics then we can't explain the model performance because those metrics can't quantify the project objective.

Hospodarskyy *et al.* [8] tells how the Adam optimizer works and the implementation behind the Adam optimizer. The selection of the optimizer in training deep learning models is very important to help the deep learning model to fit the data more accurately. In this project, the Adam optimizer algorithm (short for Adaptive Moment Estimation) was utilized to update the weights in the deep learning model during the training process. Adam optimizer is considered one of the most efficient and adaptive optimizers used in deep learning.

The Adam optimiser is essentially a clever combination of momentum and RMSprop – a widely used optimiser that also uses a moving average of past gradients to help the model accelerate in the direction of the gradient, and then scales the learning rate for each parameter based on recent gradients in order to smooth out noise. In essence, Adam adapts the learning rate for each parameter and maintains separate learning rates for each network weight, making it advantageous to use under scenarios with sparse data or noisy gradients. Additionally, Adam computes running averages of both the gradients (first moment) and the squared gradients (second moment), helping it to train faster[26].

2.2.9 Training and Fine-Tuning

The training process involves optimizing the model to minimize the error metrics using the Adam optimizer with a learning rate of 1e-3. To prevent overfitting and ensure the model generalizes well to unseen data, Yang *et al.* [23] describes the following strategies:

Learning Rate Scheduling: The learning rate is reduced if the validation loss does not improve, allowing the model to fine-tune its parameters more effectively in the later stages of training.

Model Checkpointing: The model's weights are saved whenever there is an improvement in validation performance, ensuring that the best version of the model is retained.

2.2.10 Inference Mode

Keras official site and Rieskamp *et al.* [16] explore the importance of the inference mode. Inference mode refers to the phase where a trained model is used to make predictions on new, unseen data. Unlike during the training phase, the goal of inference is not to update the model's weights but to apply the learned model to generate predictions.

In this project, inference mode is critical for predicting the eye coordinates (‘lx‘, ‘ly‘, ‘rx‘, ‘ry‘) on new image data. During inference, the model uses the knowledge acquired during training to predict the position of the eyes in the given input images.

2.3 Technology:-

For my project like predictive eye-coordinate model using deep learning, it would make things much easier to do that on something like AWS. For instance, if you used Amazon SageMaker, you could develop, train and deploy your deep learning model completely in the cloud with SageMaker. Within SageMaker, you could use one of the efficient GPU ‘instances’ (rapid computing capacity from Amazon Web Services) such as the p3 or g4 instance hardware to develop a fast-performing training model like InceptionResNetV2. Within SageMaker, you could utilise Amazon S3, the storage service where you could store very large amounts of data such as images and how they’ve been labelled (eg, in Inception ResNet tagged as 0 if it’s a cat, 1 if a dog, or 2 if an airplane.) SageMaker also has methods for automating preprocessing so you could use something like AWS Lambda, which automates processes that are triggered by a model. It could rename files and folders, perform something like image normalisation and augmentation (processing images by rotation or flipping). Amazon Web Services EC2 might also be helpful in setting up a more customised training environment that might be better or faster for your training purposes. You can adjust EC2 to fit your exact needs. Once your model is trained, depending on how you deploy it, there may be importance to having objects that could automate input without human intervention; that’s not the case with smaller models to predict eye coordinates Post-training, you could deploy your model via SageMaker endpoints for inference so you could get real-time scores when you input the images. Or you could use AWS Lambda (a service that runs code for you via the cloud)[14].

Chapter 3

Execution

For me, the Execution of this project is a step-by-step procedure. First I made a general model which has high errors then I made hand made CNN model for this which had a better result than the general but did not satisfy the results. After that I shift to the three pre-built models InceptionResNetV2, EfficientNetB7, and NasNetLarge. I chose InceptionResNetV2 as a final model because this model has decent accuracy with less number of layers and less time per inference step as compared to the other one which is good for the low-processing device.

3.1 Data Collection and Preparation

3.1.1 Dataset Collection

First, I took a dataset that consists of images and CSV files containing eye coordinate data. The CSV files provide the left and right eye positions (‘lx‘, ‘ly‘, ‘rx‘, ‘ry‘) for each image. Each CSV file corresponds to a set of images stored in a directory.

The ‘lx‘ is the distance between the left edge of the image and the center of the left eye. The ‘ly‘ is the distance between the upper edge of the image and the center of the left eye. The ‘rx‘ is the distance between the left edge of the image and the center of the right eye. The ‘ry‘ is the distance between the upper edge of the image and the center of the right eye.

The dataset is very large for computing on a laptop, so I selected 900 data points for training, 600 data points for validation, and the rest of the data points were used for testing the models. Many people use fewer test data points, but this approach can limit the generalizability of the model for real-world data points. That’s why I used the entire test dataset, which makes my model more robust and ready for real-world situations[15].

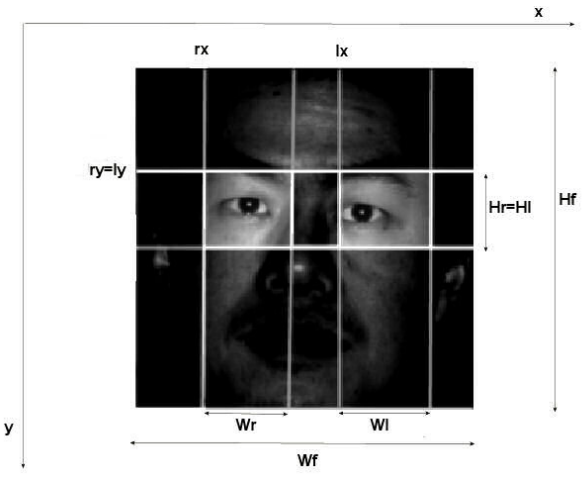

Figure 3.1: Rough localization of the eye regions

3.1.2 Data Preprocessing

Then I did some data preprocessing like blinking and overlapping of eye coordinates. That's why I remove those rows which have a '1' value or less than equal to "50" because these rows are considered to be blinks. And, Also I removed those rows which have equal lx and ly because the left eye and right eye can't overlap[15].

3.2 Data Normalization

Data Normalization is the essential step of this project. I first normalised the coordinate data based on the image dimensions(960x960). This is crucial as the model needs to learn eye positions relative to the image size. These normalised steps will be found in code or in execution as their needed:- I also normalised the pixel value of the image by 255. Then I denormalised real coordinates and prediction coordinates for calculating relative error because If you did not denormalise this then you have a normalised error or you can say your error is not correct and you miscalculate your model performance. Most of the people forget this last step.

3.3 Image Loading

Then I made one function which helps to locate the corresponding image for each CSV entry by matching a timestamp ID derived from the CSV filename with the filenames of images stored in a separate folder. If an image is found, it is added to an array; otherwise, a message is printed indicating the missing image. Valid images (those that exist in the file system) are paired with their respective normalized coordinates, and both the filenames and their labels are stored in lists. This processed data will later be used in a TensorFlow pipeline for training machine learning models[17].

3.4 Data Pipeline

After that, I used the TensorFlow data pipeline because the TensorFlow data pipeline is highly compatible with TensorFlow and Keras, and it offers high image processing speed with minimal load. The first step was to convert image file paths and labels into TensorFlow tensors. These tensors were then used to create a TensorFlow dataset. Afterwards, I implemented a function called `from_tensor_slices`. This function creates a dataset from the given tensor slices. Each element of `all_filenames` and `all_labels` becomes a pair (filename, label) in the dataset[2].

After creating the dataset, shuffling was performed by me to randomize the order of the data. This prevents the model from learning the order of data during training, which could introduce bias. Shuffling randomizes the order of the dataset, and the size of the buffer determines how much of the dataset will be kept in memory for shuffling. A larger buffer size results in better shuffling, and in this case, it was set to the size of the entire dataset. A seed value of 38 was chosen to ensure that the shuffling is deterministic, meaning it will produce the same shuffled order every time the code is run. Then I split data into train, validation and test datasets[2].

Next, the data underwent preprocessing, where images were loaded, decoded, resized, and normalized using the `map` function. The `map()` function transforms each dataset element by applying a function to it. In this case, `load_and_preprocess_image()` was used to load the image from its file path, decode it, resize it, and normalize the pixel values. `tf.io.read_file` was used to read the image file as a string. After that, the JPEG-encoded image was decoded into a 3D tensor using `tf.image.decode_jpeg`. The image was then resized to the model's input shape (299x299) using `tf.image.resize`. Finally, I optimized the number of parallel calls based on available hardware resources, which accelerated the data-loading process[2].

3.4.1 Why did I divide the image pixel values by 255?

I divide the image pixel values by 255 to normalize the pixel intensity values. Images are typically represented as arrays of pixel values, where each pixel's intensity can range from 0 to 255 (for 8-bit images). Dividing by 255 scales these values to fall between 0 and 1[22].

3.4.2 Reasons of Normalization for my project:

Improves Model Training: Neural networks often perform better when the input data is normalized. Values between 0 and 1 help the model converge faster during training by preventing large variations in the range of data, making it easier for the model to learn effectively. **Prevents Large Gradient Values:** When the input data has large values, the gradients during backpropagation can also be large, which can make the training process unstable. Normalizing helps to keep the gradients within a reasonable range.

Consistency with Pre-trained Models: I am using a pre-trained model like InceptionResNetV2, which is trained on ImageNet, it's important to preprocess the input images in a way that is consistent with how the pre-trained model expects the data. Typically, such models expect normalized image inputs.

By dividing by 255, I standardisee the pixel values and ensure that the model can process the image effectively without being influenced by the original pixel intensity scale. This ensures that all images are in the correct format for the model, with consistent dimensions and pixel values scaled between 0 and 1.

3.4.3 Data Augmentation

The next step in data preparation for my project is the application of data augmentation techniques, which help improve the model's robustness by introducing variations into the training data. The data augmentation techniques applied include random contrast and brightness adjustments.

Data augmentation introduces random transformations to the training data, improving the model's generalization by allowing it to learn from varied input data. In this case, random brightness and contrast adjustments are applied[15]. The following augmentations were used:

- **random_contrast**: Adjusts the contrast of the image by a random factor between 0.8 and 1.2.
- **random_brightness**: Randomly adjusts the brightness of the image by up to ± 0.2.

The reason I've picked specific values for random contrast (0.8 to 1.2) and random brightness (± 0.2) is that these values make sense and preserve the realism of the images being used. I am not adjusting the images in any way that doesn't make sense for natural-looking data. For example, if I were to make the contrast too low (eg, less than 0.8) or too high (eg, greater than 1.2), then this would have the effect of either darkening or brightening the images too much, and crucial aspects of the images (eg, eye coordinates) might not be captured. Similarly, by making the adjustments for brightness only +/- 0.2, this makes the image subtly different but not so different that I'd worry about the images being too bright or dark and so this might impact the model's ability to learn about them[15].

These augmentations provide just enough variety in the training data to help the model generalise better and avoid overfitting so that it performs well on unseen data. Furthermore, this scaling preserves the integrity of important features of the images – eg, the eye coordinates, which are centrally relevant to the task at hand. If you were to augment more aggressively, you might distort the critical features of the images. I've managed to find a delicate balance in these augmentations between increasing the robustness of your model, and maintaining the quality and invariance of your training data[15].

Only the training dataset is augmented, as I do not want to modify the validation or test data during evaluation.

3.4.4 Why did I choose contrast and brightness adjustments over RandomFlip and RandomRotation?

I chose contrast and brightness adjustments over `RandomFlip` and `RandomRotation` because flipping or rotating images changes their dimensions, and my project is sensitive to maintaining the original image dimensions.

3.4.5 Batching and Prefetching

Now, my project needs batching so I batched groups with multiple samples together so that they can be processed in parallel by the model during training. I used Prefetching which allowed me for efficient loading by fetching the next batch while the model is training on the current one. I cached the dataset in memory after the first epoch, avoiding recomputation or re-reading of data from disk in subsequent epochs. I combined 32 samples into one batch, allowing the model to process 32 images at once. This reduces the number of steps per epoch. Then I used prefetch for fetching batches ahead of time while the model is still training on the current batch, ensuring the next batch is ready for processing without delay. AUTOTUNE helps to determine the optimal buffer size based on the system's hardware[2].

3.4.6 Advantages of the Tensorflow data pipeline over other pipelines for my project.

The TensorFlow data pipeline offers several key advantages that enhance the efficiency and performance of machine learning models:

Efficient Data Handling: TensorFlow's data pipeline processes large datasets by loading data incrementally, avoiding memory overload in my project. This allows for the handling of datasets that are too large to fit into memory at once[2].

Parallel Processing: Through operations like `map()` and `num_parallel_calls` in my project, TensorFlow enables parallel data loading and preprocessing. This speeds up the pipeline by allowing multiple data transformations to occur simultaneously, reducing the time spent waiting for data[2].

Optimized Preprocessing: With functions like 'cache()' and 'prefetch()', the TensorFlow data pipeline caches data after the first epoch and preloads the next batch while the model is still processing the current one. This improves the throughput of data in my model and minimizes I/O bottlenecks[2].

Automated Data Augmentation: TensorFlow's data pipeline allows for dynamic data augmentation during training (e.g., random brightness and contrast adjustments), which enhances my model's ability to generalize by introducing variability in the training data[2].

Batching for Performance: The pipeline enables easy batching of my data, which is crucial for efficient use of hardware resources. This allows my model to process multiple samples in parallel, significantly speeding up training[2].

Scalability: The pipeline is highly scalable, making it suitable for both small and large datasets. It is particularly effective for distributed training scenarios where data must be efficiently fed into models across multiple machines[2].

Seamless Integration The TensorFlow data pipeline integrates seamlessly with the rest of the TensorFlow ecosystem, ensuring that my data flows efficiently into the training, validation, and testing phases without requiring extensive custom code[2].

These advantages make the TensorFlow data pipeline a powerful tool for managing complex data workflows and optimizing the training of my deep learning model.

3.5 Model Architecture

Then I used InceptionResNetV2 as a base model. InceptionResNetV2 is made up of CNN (convolutional neural network). This pre-trained model from Keras applications module is used for feature extraction. The model is pre-trained on ImageNet, which provides a robust set of learned features for object detection and classification. The layers of the pre-trained model are frozen initially to retain these features. I used InceptionResNetV2 model as the backbone of the eye coordinate prediction system which integrates Inception modules and Residual connections. Inception modules apply multiple filters of varying sizes to the same input, allowing the model to capture both fine-grained and coarse details in images. This multi-scale approach is particularly useful for predicting eye coordinates, as my model can capture both the subtle features of the eyes and broader facial features that provide context. Residual connections in the architecture allow the model to skip layers, making it easier to train deeper networks by addressing the vanishing gradient problem. These connections ensure that gradients flow more effectively during backpropagation, improving the model's ability to learn from data efficiently. By fine-tuning InceptionResNetV2 on the eye-tracking dataset, the model adapts the pre-trained weights from ImageNet to the specific task of predicting lx, ly, rx, and ry. This transfer learning approach reduces the need for large amounts of labeled data and allows the model to leverage the rich feature representations learned from a vast corpus of images[4].

3.5.1 Here's how I get benefits from transfer learning in my project:

Freezing Pre-Trained Layers

The pre-trained InceptionResNetV2 model is loaded without its final fully connected layers. The lower layers, which have already learned to extract basic visual patterns from images, are kept frozen to prevent their weights from being updated during training. By freezing these layers, my model retains its general feature extraction capabilities and focuses on learning new features relevant to eye coordinate prediction[20].

Adding Custom Layers

On top of the pre-trained model, I added one custom layer. Custom layers such as GlobalAveragePooling2D, Dropout, and Dense layers are added to adapt the model to the specific task of predicting eye coordinates (lx, ly, rx, ry). These layers help in learning the mapping between the extracted features and the target coordinates[20].

Fine-Tuning In some cases, transfer learning involves fine-tuning the weights of the pre-trained layers to better fit the specific task. In my project, after training the custom layers, the pre-trained layers are unfrozen, and the entire model is fine-tuned with a very low learning rate because I don't want to update weight drastically. This allows the model to adapt the pre-trained layers slightly to better suit the task at hand without forgetting the general features it has already learned[20].

3.5.2 Advantages of Transfer Learning in my project

Efficiency: Transfer learning dramatically reduces the amount of data and computational resources required to train a deep learning model. Since the lower layers of the model have already learned useful features, fewer epochs and less data are needed to achieve good performance in my project[20]. **Better Generalization:** By starting with a pre-trained model, my model is less likely to overfit on small datasets, as it benefits from the general knowledge acquired from a larger dataset[20].

Time-Saving: Training a model from scratch can be computationally expensive and time-consuming, especially with large datasets. Transfer learning significantly cuts down my training time by leveraging pre-trained networks. This feature enables the feasibility of making a model by students [20].

3.6 Global Average Pooling

GlobalAveragePooling2D is a crucial layer in the architecture used for this project, specifically when working with Convolutional Neural Networks (CNNs). It plays a pivotal role in transitioning from feature extraction to prediction in the InceptionResNetV2 model. Global average pooling works by taking the average value of all elements in each feature map produced by the convolutional layers. Instead of flattening all features into a single vector, as seen in traditional fully connected layers, it reduces each feature map to a single value, thus simplifying the complexity and dimensionality of the data[24].

The operation performed by GlobalAveragePooling2D() can be mathematically represented as:

$$\text{Global Average Pooling} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} f_{ij}$$

Where H and W are the height and width of the feature map, and f_{ij} are the feature values.

3.6.1 Here are some reasons why GlobalAveragePooling2D is beneficial in this project:

Dimensionality Reduction: After several layers of convolution and activation, the feature maps become high-dimensional in my project that's why I am using global average pooling to reduce the dimensions by taking the average of each feature map, producing a smaller, manageable representation that can be passed to the subsequent layers, such as dense or fully connected layers[24].

Mitigates Overfitting: Because it reduces the number of parameters, global average pooling helps reduce overfitting, especially when training on small or limited datasets. Traditional fully connected layers may have more parameters that could lead to overfitting, but global average pooling aggregates information in a more controlled manner[24].

Translation Invariance: Since it averages the entire feature map, GlobalAveragePooling2D helps make the model more robust to translations in the input images. This is particularly important when predicting eye coordinates, as it ensures that the model focuses on the overall presence of features rather than their exact locations[24].

Efficiency: By drastically reducing the number of parameters compared to fully connected layers, global average pooling helps my model to achieve better efficiency in terms of computational cost and memory usage. This is especially useful while building my model intended for real-time applications like eye-tracking[24].

In the context of my project, GlobalAveragePooling2D() is applied after removing the final convolutional layers of InceptionResNetV2, reducing the feature maps to a set of averaged values before passing

them to the fully connected layers. This layer is essential for ensuring the model generalizes well across different types of facial images while keeping the network lightweight and efficient[24].

By applying `GlobalAveragePooling2D`, my model transitions smoothly from complex feature maps to a simplified, dense representation that can be processed to predict eye coordinates (‘lx‘, ‘ly‘, ‘rx‘, ‘ry‘) with precision[24].

Dense Layers: Fully connected layers with ReLU activation are used to further process the extracted features. Dropout is applied to prevent overfitting.

3.7 Working of ReLU in My Model:

I am breaking down the layers where ReLU is used:

1. **First Dense Layer with 128 Units:**

   ```
   x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
   ```

   This layer has 128 neurons (units). Each neuron computes a weighted sum of its inputs followed by adding a bias term. The ReLU activation function is applied to the output of each neuron, ensuring that any negative values are replaced by zero, while positive values are passed as-is. This helps introduce non-linearity and prevents neurons from being driven into a negative state, which might prevent them from contributing to the learning process[5]. The formula is defined as:-

   $$f(x) = \max(0, x)$$

2. **Second Dense Layer with 64 Units:**

   ```
   x = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(x)
   ```

   Similar to the first dense layer, this layer has 64 neurons, and ReLU activation is applied after the linear transformation. Again, the ReLU ensures that only positive values are passed through, making the network capable of learning complex features and representations of the input data[5].

3.7.1 Why did I use ReLU in These Layers?

Non-linearity: ReLU allows the network to learn non-linear relationships, which is crucial for deep networks to model complex functions. Efficient training: ReLU is computationally efficient (compared to other activation functions like sigmoid or tanh), as it requires simple thresholding[5].

Avoids saturation: ReLU doesn't saturate for positive values, meaning the gradients can be large enough to avoid vanishing during backpropagation. In the final stage of the model architecture, the Dense(4, activation='linear')(x) layer plays a critical role in the output of the eye coordinate predictions[5].

Dense(4):

The Dense layer is a fully connected layer where each neuron receives input from all the neurons of the previous layer. In my case, we have 4 neurons in the output layer, which corresponds to the four coordinates: lx (left eye x-coordinate), ly (left eye y-coordinate), rx (right eye x-coordinate), and ry (right eye y-coordinate). This layer effectively maps my high-level features extracted by previous convolutional layers and processes them to produce my final coordinate values.

3.7.2 Why is the Linear activation function useful or choosing over others for my project?

The linear activation function is specifically useful for regression problems, where the model is expected to predict continuous values rather than categorical labels or discrete outputs. In this project, the goal is to predict precise numerical coordinates (lx, ly, rx, ry) representing the position of the eyes in the image, which makes it a regression task. The linear activation function is simply the identity function, where the output is directly proportional to the input[21]. Mathematically, it can be described as:

$$f(x) = x$$

Range of Output: Unlike activation functions such as ReLU or sigmoid, which limit the range of the output values, linear activation allows the output to take any value within a continuous range. This is essential for predicting the precise location of eye coordinates, which could be any value depending on the size and resolution of the image[21].

Accuracy: By using a linear activation in the final layer, the model is able to output raw, unbounded values that represent the exact positions of the eyes. These coordinates are directly interpretable and can be used for downstream tasks like determining gaze direction or evaluating cognitive load based on eye movement[21].

3.8 Custom relative error metric:-

After that, I created a relative error metric. The formula is given by

$$e = \frac{d}{w}$$

where d is the Euclidean distance between the estimated and true eye centers, and w is the Euclidean distance between the true eye centers. This metric is used as a relative error measure for eye localization. Euclidean Distance Formula is:- $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

3.8.1 why did I choose this metric and what are the advantages of this metric?

This metric normalizes the error in eye localization by the inter-eye distance, making it independent of the size and scale of the face in the image. It provides a more robust evaluation by accounting for different face sizes, which is crucial in facial recognition tasks where faces can appear at different scales[9].

I preferred relative error metric over others because it offers a relative measurement, meaning the error scales proportionally to the face size. Without this normalization, a small error in eye localization would appear more significant in smaller faces than in larger ones, potentially skewing the results. By using the inter-eye distance as a baseline, the relative error metric provides a consistent and scale-independent measure of localization accuracy, making it particularly useful for evaluating face detection systems across diverse datasets and varying conditions[9].

3.9 Model Training and Optimization

3.9.1 Loss Function and Optimizer

Loss Function and Optimizer I used Mean Squared Error (MSE) as a loss function to optimize the model. The Adam optimizer with an initial learning rate of 1e-3 is employed. I chose Adam for its ability to handle sparse gradients and adapt the learning rate for each parameter, which makes it highly effective in training my model. Adam adjusts the learning rate for each parameter based on the running estimates of both the gradient (first moment) and the squared gradient (second moment), leading to faster convergence and more accurate results. In my project, Adam is chosen for its ability to handle noisy data and sparse gradients, which are often encountered in image-based tasks like eye coordinate prediction. The initial learning rate is set to 1e-3, and the optimizer dynamically adjusts the learning rate as training progresses, allowing the model to fine-tune its parameters efficiently. To further enhance the model's performance, I used ReduceLROnPlateau as a scheduler is employed. This scheduler reduces the learning rate when the validation loss stagnates in my model, allowing for finer updates to the weights and preventing overfitting[23].

3.9.2 Why, I chose Adam optimizer over others optimizers

Choosing the Adam optimizer for my project over alternatives such as SGD, RMSprop, Adadelta, and others comes down to several key advantages that Adam provides for this specific task. Let's explore why Adam is particularly suited to my project over the alternatives:

Efficiency in Convergence:

Adam is a combination of two popular optimisation methods for neural networks, AdaGrad and RMSProp, that allows it to justly adapt the learning rate for each parameter, making it better at handling sparse gradients and non-stationary data. One important aspect of my project is that it involves training the classifier on large amounts of data, where each input is associated with fine-grained eye coordinates,

which means that converging quickly and efficiently is an important feature. Adam is known to converge very quickly, and it gives better results on tasks where there is a need for very precise and fast updates, which is the case with my project[25].

Adaptive Learning Rates:

For example, one of Adam's neat tricks is that it adjusts learning rates for each parameter separately, piggybacking on a moment estimate. Because my project is concerned with a prediction of eye coordinates, the adaptive nature of Adam means the model can make small tweaks to each weight based on how often it's updated, which will help refine the prediction of those coordinates. Other optimisers such as SGD are less precise, using a global learning rate instead[25].

Gradient Momentum:

Adam uses an exponentially decaying average of previous gradients of weight, which helps to smoothen out the update process, preventing oscillations in training (a very important factor for a project like mine, where it is crucial for the network to find well-defined minima and to provide stable learning to predict the coordinates of the eyes inside an image, which is a very small feature). Optimisers such as SGD do not have this momentum property by default, and are more susceptible to overshooting minima or oscillating in steep regions of the objective function[25].

Handling Non-Stationary Data:

In the case of the Adam optimiser, which automatically dynamically adjusts the learning rate according to gradient history, we have a method that's well suited to non-stationary problems where a changing data distribution makes the optimisation problem more difficult over time. For problems involving deep learning with high-dimensional input data (eg, predicting eye coordinates from images), Adam is, therefore, likely to be a better choice for me than Adadelta or Adagrad, which might struggle to cope with changes in distribution[25].

Less Tuning Required:

Adam tends to be my default optimiser – it needs little or no hyperparameter tuning, which is great when my project is small and I want to be working on architecture and feature engineering spending significant time tuning the learning rate and momentum parameters. Optimizers like SGD or RMSprop might require more hyperparameter tuning, which can slow down development[25].

Comparison with Other Optimizers:

SGD: Stochastic Gradient Descent is slow to converge, and the update steps are not great for sparse gradients or non-stationary data without incorporating momentum or adaptive learning rates, which you probably don't want if my project requires precision[25].

RMSprop: While good for non-stationary data, RMSprop lacks the momentum aspect and dual adaptability of Adam. It might perform well but lacks the robustness Adam provides[25].

Adadelta and Adagrad: These are adaptive methods to learn the learning rate, but if they learn well in your project, they might decay the learning rate too much over time, which could interfere with learning[25].

Adamax: Adamax is an infinity-norm version of Adam, but Adam is typically better in general, in particular for smaller datasets like mine[25].

Nadam: An extension of Adam that adds Nesterov momentum. Nadam can lead to very slightly faster convergence in some circumstances, but the difference is often marginal, and Adam is generally much more robust[25].

FTRL: While this algorithm is useful in sparse settings such as logistic regression, for my kind of deep learning project, the use of FTR is not necessarily better for your specific results[25].

Lion and Loss Scale Optimiser: Lion is fairly new and might be less empirically validated than Adam. Likewise, the Loss Scale Optimiser is used mainly to address numerical stability in mixed-precision training, which is not relevant to my project.[25]

In short, the tuning requirements of Adam are low, its convergence is fast and, most importantly, its adaptability to noisy data is high. In other words, it's a good algorithm to use for my eye coordinate prediction problem. It balances speed with accuracy, which is why Adam is often preferred to more accurate optimisers that tend to be much slower[25].

3.10 Training Process

During training, the model is fine-tuned for 100 epochs with ReduceLROnPlateau and ModelCheckpoint callbacks. The learning rate is reduced by half if the model's performance on the validation set does not improve for 5 consecutive epochs, and the best model is saved based on the validation loss. I did not use Early stopping because my project trained with little change per epochs so if I used early stopping

then early stopping identified little change as a stagnates and then it would stop my training[23]. Then I plotted some graphs related to loss, Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) over time, providing insights into the model's performance.

3.10.1 Differences Between Training and Inference Mode

No Gradient Updates: During inference, the model does not calculate gradients or perform backpropagation, as no weight updates are necessary. Batch Normalization: Layers like batch normalization behave differently during inference. Instead of normalizing the inputs using the batch's mean and variance (as during training), the model uses running averages calculated during training[16].

Dropout Disabled: During training, dropout layers randomly drop a percentage of neurons to prevent overfitting. However, during inference, dropout is disabled so that all neurons are active, ensuring that the full capacity of the model is used for prediction[16].

Implementing Inference Mode In TensorFlow, switching between training and inference modes is straightforward. When calling the model for predictions, the model is automatically set to inference mode: Setting training=False ensures that the model is in inference mode, where dropout and batch normalization behave appropriately for making predictions[16].

Importance of Inference Mode

Efficiency: Since the model doesn't perform backpropagation or gradient calculations during inference, it is significantly faster than training mode[16].

Accuracy: Properly managing inference mode ensures that layers like dropout and batch normalization behave as expected, preventing discrepancies between training and prediction results[16].

3.11 Model Evaluation

Once training is complete with freeze layers, I saved the best model, and then test dataset is used for evaluation. Metrics like MSE, MAE, MAPE and Relative Error are calculated. A custom relative error metric is implemented to measure the model's performance in predicting the eye coordinates. This metric normalizes the Euclidean distance between the predicted and true coordinates by the distance between the two eyes in the image.

1. **Mean Squared Error (MSE)**

Mean Squared Error (MSE) measures the average of the squares of the errors between the predicted values and the actual values. It is a popular loss function for regression tasks.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- y_i = actual value
- $\hat{y}_i$ = predicted value
- n = total number of data points

MSE gives a higher weight to large errors due to the square, making it sensitive to outliers.

2. **Mean Absolute Error (MAE)** Mean Absolute Error (MAE) measures the average magnitude of the errors between the predicted and actual values. Unlike MSE, it does not square the differences, so it is less sensitive to large errors.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Where:

- y_i = actual value
- $\hat{y}_i$ = predicted value
- n = total number of data points

MAE is a more interpretable metric because it is in the same units as the original data.

3. **Mean Absolute Percentage Error (MAPE)** Mean Absolute Percentage Error (MAPE) measures the average percentage difference between the predicted and actual values. It provides an error percentage, making it scale-independent.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

Where:

- y_i = actual value
- $\hat{y}_i$ = predicted value
- n = total number of data points

MAPE is widely used when the goal is to express error in percentage terms, but it can be problematic if y_i is zero.

3.12 Post-Training Inference and Model Fine-Tuning

After initial training, the model is fine-tuned by unfreezing the layers of the base model with refit and recompiling. Fine-tuning helps adjust the pre-trained features to better suit the task of predicting eye coordinates. Then we saved the best model again. And then testing on the test dataset again. The inference mode of the model is used to make predictions on unseen data. During inference, the model is set to training=False, ensuring that layers like Dropout are not active[23].

Then we analysed the final predictions compared to the ground truth, and the relative error values are analyzed.

Backpropagation plays a crucial role in fine-tuning when I unfreeze the layers of a pre-trained model. During the fine-tuning process, the layers of the base model, which were initially frozen to retain the learned features from the pre-trained weights, are now "unfrozen," meaning they become trainable. This allows backpropagation to update the weights of these layers, adjusting them based on the specific task (predicting eye coordinates)[6].

When I perform backpropagation, the model computes the loss between the predicted eye coordinates and the actual ground truth values. The gradients of this loss with respect to each weight are calculated through the chain of operations that led to the prediction (using the chain rule). These gradients are then used to update the weights of the unfreezed layers by moving them in a direction that reduces the error. As the weights are adjusted through several training iterations, the model gradually learns features that are more relevant to predicting eye coordinates rather than the general features it initially learned during pre-training. By fine-tuning the model with backpropagation, we make the model more task-specific, leading to improved performance on the new dataset[6].

3.13 Project Management and Best practices

In managing this project, I employed several best practices to ensure efficient workflow and reliable results. The primary programming language used was Python, selected for its extensive libraries in machine learning and deep learning, particularly TensorFlow and Keras, which facilitated model building, training, and data preprocessing. Jupyter Notebooks were used for development and experimentation due to their flexibility in allowing iterative testing and visualization of results. Although version control through Git is standard in many projects, I opted for Jupyter Notebooks, allowing me to track progress and experiment interactively, which suited the rapid prototyping needs of this project.

For optimization, I used the Adam optimizer, which balances computational efficiency and the ability to handle noisy gradients, in contrast to more basic optimizers like SGD. Decisions around data augmentation (contrast and brightness adjustments) were made based on the sensitivity of the model to changes in image dimensions. By adhering to these best practices, I ensured a robust and scalable approach to building an effective deep learning model for eye coordinate prediction.

Chapter 4

Critical Evaluation

4.1 Training with freezing layers

4.1.1 Some important epochs

I trained this model with freezing layers till 100 epochs.

Epoch	Loss	MAE	MAPE	Relative Error	Val Loss	Val MAE	Val MAPE	Val Relative Error	Learning Rate
55/100	0.0027	0.0424	9.5143	0.1657	0.0028	0.0416	9.4322	0.2012	0.0010
56/100	0.0027	0.0424	9.5068	0.1654	0.0028	0.0416	9.4235	0.2015	5.0000e-04
61/100	0.0027	0.0423	9.5125	0.1653	0.0028	0.0416	9.4078	0.2016	2.5000e-04
66/100	0.0027	0.0423	9.5101	0.1653	0.0028	0.0416	9.4052	0.2016	1.2500e-04
71/100	0.0027	0.0423	9.5084	0.1653	0.0028	0.0415	9.4027	0.2016	6.2500e-05
76/100	0.0027	0.0423	9.5065	0.1653	0.0028	0.0415	9.4016	0.2017	3.1250e-05
81/100	0.0027	0.0423	9.5061	0.1653	0.0028	0.0415	9.4009	0.2017	1.5625e-05
86/100	0.0027	0.0423	9.5061	0.1653	0.0028	0.0415	9.4005	0.2017	1.0000e-05
100/100	0.0027	0.0423	9.5053	0.1653	0.0028	0.0415	9.4001	0.2017	1.0000e-05

Table 4.1: Training and Validation Metrics across Epochs

4.1.2 Findings via Epochs

1. Functional Testing and Convergence:

The model convergence can be observed by the later epochs where the value of the loss, mean absolute error (MAE), and mean absolute percentage error(MAPE) remained consistently low over the epochs. The same can also be observed for the validation loss and metrics such as val_mae (validation mean absolute error), val_mape (validation mean absolute percentage error), and val_calculate_relative_error. This is an important observation because it implies that the model had not overfit to the training data, but rather generalized to the validation set.

Although the metrics performed well on the validation set, the calculated relative error (e = d/w) was consistently 0.2017 on the validation set while it remained 0.1653 on the training set. It would be nice to understand why there is a gap in relative error between training and validation. This gap could be caused by small differences between the training and validation sets. Alternatively, it could be due to small differences in the data that make it harder for the model to generalise to unseen new data.

2. Learning Rate Decay and Impact:

We decreased the learning rate over time with the ReduceLROnPlateau callback. The learning rate halved many times during training, from 0.001 to as low as 1.0000e-05. The metrics for loss began to plateau as the learning rate was decreased. This is a good sign that the model had learned as much as it

could from the data. This behaviour indicates that the model was fine-tuned sufficiently, and it is unlikely that running it for additional epochs or playing with the learning rate would provide much improvement. The learning rate decay helped prevent overshooting during training, and ensured the model converged well.

3. Analysis of Validation Relative Error (Persistent Issue):

A clear issue was the persistent relative error on the validation set. The val_calculate_relative_error remained 0.2017 for many epochs even as the other validation metrics (eg, val_mae and val_mape) improved slightly. Persistent relative error suggests the presence of several outlier cases in the validation data (a small number of edge cases that fall significantly outside the usual range), where the denominator w in the e = d/w formula is very close to zero. Such edge cases would have been filtered from the dataset, but were not entirely learned by the model. Since relative error measures precision relative to the distance between the coordinates of the eyes, it is sensitive to smaller distances between the two eyes, which could account for values being higher in validation relative error than in training.

Furthermore, the increasing behaviour of relative error can suggest that the model is not generalising optimally across all data points and may wish to further fine-tune or change the data augmentation technique to reduce this metric. The fact that relative error plateaus implies that there is a structural limitation to the data or model architecture that inhibited further decrease.

4. Drop in Improvement Over Time:

We also note that, after epoch 55, the differences in training and validation loss and error metrics are very small. This stands to reason, as at some point the model will have learnt as much as it can about the pertinent patterns, and will no longer be learning anything new from the data. It's for this reason that we stopped training at epoch 100 – further training would be computationally expensive, but would have little benefit in terms of improved performance. One might want to try out further exploratory work, to see if it's possible to break the plateau. One could experiment with other optimisers or add more regularisation techniques.

5. Impact of Data Augmentation:

One potential explanation is that the generalisation improved with data augmentation (in particular, contrast and brightness augmentation). The model was apparently able to make more generalisations, thanks to these augmentations – the model became less sensitive to small variations in the images. Another possible explanation is that the validation and test performance did not indicate overfitting. That's because it's relatively easy to handle natural variations of the data (such as small errors in the lens of the eye) with a clever model. Augmentation techniques demonstrated here were limited to only contrast and brightness because the eye coordinates are sensitive to image orientation (that is, rotation or flipping of the image can shift the area we are interested in). It would be interesting to try more complex augmentation techniques in future work.

6. Decision to Use Adam Optimizer:

We chose the Adam optimiser over alternatives such as SGD or RMSprop because of its adaptive learning rate, which allows for smoother and faster convergence. This was important, since the model ended up converging within 100 epochs, whereas if we had used a simpler optimiser such as SGD, we might have had to go through more epochs, or carefully tune the learning rate beforehand to work with the noisy gradients that arise in the training of complex models like this, especially when working with high-dimensional data.

7. Implications of Model Design Choices:

The addition of GlobalAveragePooling2D and dense layers with ReLU activation did allow the model to learn features from the images and make predictions. The ReLU activation was a great choice for

introducing non-linearity and for allowing the model to learn complex patterns, while also avoiding vanishing gradients. The dropout layers added for preventing overfitting had the right idea, but were overfit themselves and could have been optimised to allow for a tradeoff between regularisation and expressiveness.

In summary, this project has a good performance with low MAE and MAPE with a steady loss, but the persistent relative error on the validation set indicates that there is still room for optimisation. The decisions of choosing an optimiser, learning rate scheduling, and implementing data augmentation play a major role in successfully constructing a model.

4.1.3 Findings via Graph

Loss

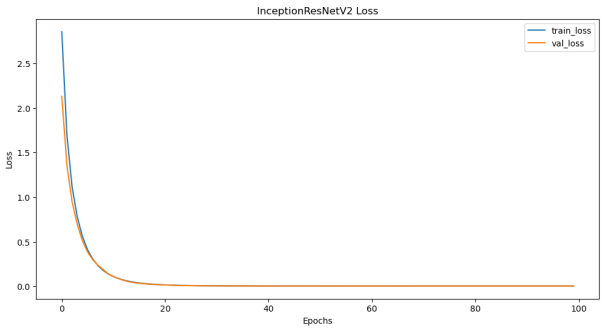

Figure 4.1: Loss over Epochs

Loss graph represents the loss curves for both the training set (‘train_loss‘) and the validation set (‘val_loss‘) over 100 epochs, using the InceptionResNetV2 model. Let’s critically evaluate what this graph indicates about my model’s training process:

Initial Phase (0 to 10 epochs):

In the early stages of training, both the training and validation losses decrease rapidly. This is expected, as the model is quickly learning from the data. The steep drop suggests that the model is adjusting its weights effectively to minimize the error during the initial learning phase.

Convergence (10 to 30 epochs):

After around 10 epochs, the loss values for both the training and validation sets reach a low point and begin to plateau. At this stage, the model has likely captured the most important patterns in the data. The curves for training and validation loss remain closely aligned, indicating that the model is generalizing well, with minimal signs of overfitting. This is crucial as it shows that the model’s performance on unseen data (validation set) is comparable to the training set.

Late Stage (30 to 100 epochs):

From around 30 epochs onward, the training loss becomes slightly lower than the validation loss, and both curves stabilize. This stability indicates that further training does not lead to significant improvements, suggesting the model has reached its optimal performance for this configuration. The minimal gap between the two curves indicates a well-trained model without overfitting, as there is no significant divergence between training and validation losses.

Analysis for Critical Evaluation:

1. Good Generalization: The close alignment of training and validation loss curves throughout the training process demonstrates that the model is generalizing well. The lack of significant overfitting is a positive outcome, which can be attributed to the use of techniques like Dropout and the Adam optimizer. These techniques help the model prevent overfitting to the training data, ensuring it performs well on unseen data.

2. No Signs of Underfitting: Since both curves show a sharp decrease at the beginning, it indicates that the model is not underfitting the data. The model is capable of capturing the complexity of the dataset within the first few epochs, meaning the model architecture and learning rate were well-tuned for the task.

3. Potential Fine-tuning: Although the graph indicates stable performance, the slight gap between the training and validation losses in the later epochs might suggest that the model could still be fine-tuned further. Exploring changes in data augmentation strategies or adjusting the learning rate decay might slightly reduce this gap and improve generalization further.

4.1.4 MAE

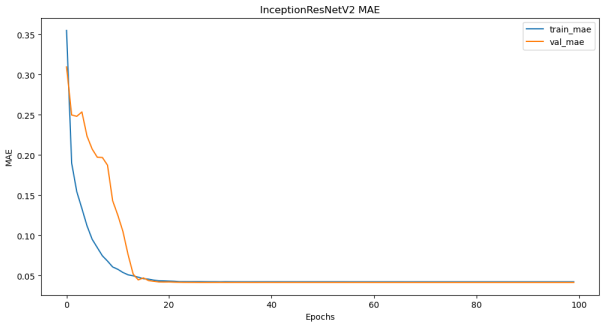

Figure 4.2: MAE over Epochs

The graph where I presented to you is a mean absolute error (MAE) for the training set (train_mae) and for the validation set (val_mae) on 100 epochs using the InceptionResNetV2 model shown above. Let's go over every detail of the graph for the critical evaluation of your project:

Initial Phase (0 to 10 epochs):

Training MAE in the earlier epochs decreases quite significantly, from about 0.35 to 0.10, which shows that the model is fast learning to reduce error in the prediction.

As for the validation MAE, it decreases similarly but with more oscillations, reflecting the tendency of the model to overfitting to the unseen validation data, and initially perhaps underestimating its predictions.

The most hopeful indicator is that the dip on both curves in the first 10 epochs is significant. This suggests the model is indeed learning the underlying patterns of the data.

Convergence (10 to 30 epochs):

From epoch 10 to 20, both the training and validation MAE keep decreasing, the difference is that validation MAE has a more apparent fluctuation, which might be caused by the characteristic of the validation data or overfitting risk.

After epoch 20, the validation MAE levels off, close to the training MAE. The two curves are on the same line at around 0.05 MAE, so the model is performing equally well on both the training and validation dataset.

The slight space between the training and validation MAE curves indicates that the model has generalised to the unseen data without over-fitting. In other words, our model can perform well on data that we haven't seen before during training. This is a fundamental aspect of assessment.

Late Stage (30 to 100 epochs):

The plateau of both curves after epoch 30 is clear evidence that the model has reached its limit on how well it can perform, and that adding more training data will not make it any better at predicting the target.

The MAE values decrease to around 0.05, which means a very low error in the absolute difference between predicted and true eye coordinates. This is a good result for the model, showing that it is accurately capturing true eye coordinates at any point in time.

Critical Evaluation Points:

1. Fluctuations in Validation MAE: Small fluctuations in the Validation MAE curve for the first few epochs is something to evaluate. These fluctuations can be an indication that while the model is learning, the validation set is harder or noisier than the training set, but then they quickly smooth out, showing that the model adapts to both datasets.

4.1.5 MAPE

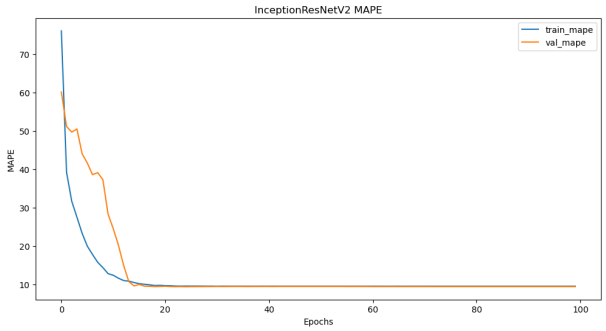

Figure 4.3: MAPE over Epochs

This graph is for Mean Absolute Percentage Error (MAPE) over 100 epochs for both the training set ('train_mape') and validation set ('val_mape') using my model. Now, let's talk about the evaluation of the graph critically:

Initial Phase (0 to 10 epochs):

As you can see from the first few epochs, training MAPE starts at around 70%, which means the initial predictions made by the model were far off from the real eye coordinates. This makes sense at the start of training because the model has not yet learnt any significant patterns.

The validation MAPE starts at a similarly high level (60%) but shows some wiggles as the model learns on the out-of-sample validation data. These wiggles indicate that, initially, the model hasn't yet stabilised in its learning on the validation set.

Along both the training curve and the validation curve, you can observe that the first 10 epochs result in a rapid decrease in error, which means that the model is learning faster in the first 10 epochs and is learning more meaningful patterns to reduce the prediction error.

Convergence (10 to 30 epochs):

Between epochs 10 and 20, curves converge to similar values (around 10%). In the validation MAPE, we still can see some mild fluctuations, but the difference between training and validation MAPE is strongly reduced.

This convergence is a good sign: it means that the model has been able to generalise to unseen data, and the error rate on both the training and the validation sets reaches a very small value.

The fast drop and convergence indicates a good model performance which means my model is appropriate for the task of learning eye coordinate prediction.

Late Stage (30 to 100 epochs):

From epoch 20 onwards, training and validation MAPE curves reach a plateau of 10%. There is no improvement or degradation observed over the subsequent epochs, which is an indication that the model has converged and is not gaining any further benefit from training.

Note how both curves keep close together until the end of the training epochs, showing that the model is not overfitting. If there was overfitting, we would see the training MAPE continuing to improve while the validation MAPE stays the same or gets worse. Here, both are low and close together, showing good generalisation ability.

Critical Evaluation Points:

3. Good MAPE (10%): The MAPE settling at about 10% is good news, especially for a regression task like predicting eye coordinates. This means that, on average, the model predictions are within 10% of the actual values, which is a reasonable level of error for many real-world applications.

4.1.6 Relative Error

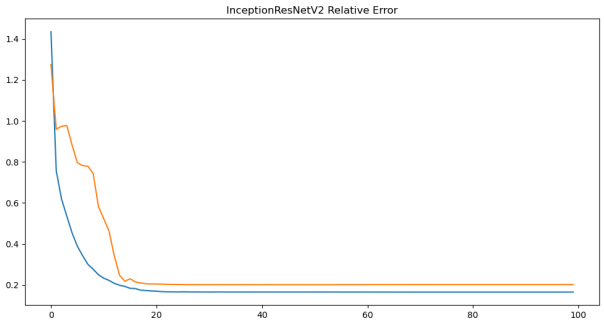

Figure 4.4: Relative Error over Epochs

My relative error graph the Relative Error in terms of both the training set ('train_relative_error') and the validation set ('val_relative_error') over 100 epochs while training my model for eye coordinate prediction. Let's discuss this graph:

Initial Phase (0 to 10 epochs):

We see that the training relative error starts out high (at 1.4), corresponding to the fact that the predictions made by the model in its initial state are very far from the correct eye coordinates.

The validation relative error starts out at a similarly high value, but with some little zigzags. These zigzags indicate that the model is still trying to find its footing when it comes to unseen validation data, and that its performance isn't as solid during the early stages of training.

Since both training and validation errors follow similar trajectories, decreasing fast during the first 10 epochs, we can conclude that the model is learning and its predictions are improving.

Convergence (10 to 30 epochs):

Both curves plummet by epoch 10, and both continue to drop until around epoch 20, at which point the training relative error is 0.16 and the validation relative error has settled around 0.20.

The slight gap in the training and validation curves that we observe around this point indicates that the model is doing a good job of generalising (that is, it's learnt to predict eye coordinates with reasonably low relative error on both the training and validation sets).

It is also interesting to note that the validation relative error stabilises after epoch 20, meaning that the validation loss stops going down and the model is not overfitting, ie it still makes a similar prediction on unseen data.

Late Phase (30 to 100 epochs):

We see that, after approximately the 30th epoch, the curves stabilise and additional training no longer yields significant improvement. The training relative error stabilises around 0.16, and the validation relative error around 0.20.

This consistency in the behaviour of the two curves implies that the model is well-trained, and that the model performs equally well on both the training and the validation dataset. It is also an indication that there is no overfitting. Overfitting would imply that the training error would continue to decrease while the validation error would start increasing.

Critical Evaluation Points:

1. Training and validation sets with initial high relative error: The initial high relative error values in both the training and the validation set (1.4) are what we would expect when training begins.

The model hasn't yet learned the connections between input data (the images) and the target values (the coordinates of the eyes) and it's making wild guesses.

2. Fast decrease of error: The fast decrease of relative error in the first 20 epochs shows that the model is learning from the data, and that the architecture that we chose is well-suited for this task. error is fast stabilising, which means that learning was successful.

3. Consistent Low Relative Error: The last relative error values of 0.16 (training) and 0.20 (validation) are quite low as well, with the model able to predict eye coordinates with a small deviation to the correct values (which is a strong performance anyway, especially for a regression task – predicting eye coordinates – where even small errors can influence the accuracy).

Summary:
These graph describes the process of training. In general, there is clear evidence that the training process is going well, based on the fact that errors dropped very quickly and that the training and validation curves are approaching each other very closely. Also, there is a low level of error. There is no indication of overfitting and the model is well generalized for training and validation, since the model is trained to well-selected eye coordinates. It is safe to conclude from these graphs that the model operates well for predicting eye coordinates with low level of prediction errors. Based on the information given, the model is suitable for real life application where it is crucial to predict the movement of the eyes with high levels of accuracy.

4.2 Evaluation on test data with freezing layers

Metric	Value
Test Loss	0.00273
Test MAE	0.04236
Test MAPE	9.92878
Test Relative Error	0.15917

Table 4.2: Test Results Summary

The model is evaluated on test data with frozen layers. The table above shows the value of four metrics: test loss, MAE, MAPE, and relative error. The test loss is 0.00273. This means that the model has made less than 0.00273 mistakes on the test data. It's a very low value, so we can conclude that the model works well in terms of overall accuracy. The Mean Absolute Error (MAE) is 0.04236, which means that the value of the model's prediction deviates from the true value on average by about 0.042 units, and its prediction is very accurate.

Moreover, the Mean Absolute Percentage Error (MAPE) of 9.92878 means that the model predictions, on average, are within 9.93 per cent of the real values. In most practical applications, this would be a very good result, suggesting that the model generalises well to unseen data. The Relative Error of 0.15917, while good, indicates that there is still some degree of error in the predictions, of around 15.9 per cent relative to the scale of the values.

This evaluation demonstrates that by freezing the layers of the pre-trained inceptionResNetV2 model, we were able to make use of features learned from the ImageNet dataset and apply them to the test dataset. Our model performs very well with a low loss and MAE, with a higher value of the relative error perhaps indicating room for improvement – perhaps with some further fine-tuning. We could potentially reduce this error by unfreezing some of the layers or changing the hyperparameters of our model. This evaluation shows that our transfer learning approach of pretraining with frozen layers was successful in achieving strong predict.

4.2.1 Relative Error Distribution of test Data with freezing Layers

Table below illustrates the distribution of relative error (calculated using the e = d/w metric) upon different error ranges of the test data with the frozen layers. Overall, a large portion of the test set presents a low relative error. Especially, there are 1,819 samples that have a relative error in the 0.0 - 0.1 range and 4,960 samples in the 0.1 - 0.2 range. It indicates model works well. In a large majority of cases, tests can provide highly accurate results.

Stepping past the 0.2 error mark, the samples become fewer with 1,632 samples of errors between 0.2 - 0.3, showing the model still reasonable accuracy with slightly larger covers. Next, a sharp drop to 143 samples of errors between 0.3 - 0.4, and even fewer samples with higher errors, 10 samples between 0.4 - 0.5 and 3 samples between 0.5 - 0.6. Both higher error ranges indicate possible cases in which the model fails, but these cases rarely occur.

Importantly, there are 4 instances where the relative error is greater than 1.0; we refer to such data points as outliers. An outlier indicates that, in very rare cases, the model has significantly mispredicted the eye coordinates. Therefore, the existence of outliers indicates that the model might be dealing with unusual or challenging data samples, which it might not be able to generalise. It would be worthwhile studying outliers and seeing what one can learn from them; they might be edge cases or unique limitations of the training dataset, and could potentially be addressed by either further data augmentation or model fine-tuning.

Relative Error Range	Number of Samples
0.0 - 0.1	1819
0.1 - 0.2	4960
0.2 - 0.3	1632
0.3 - 0.4	143
0.4 - 0.5	10
0.5 - 0.6	3
0.6 - 0.7	0
0.7 - 0.8	0
0.8 - 0.9	0
0.9 - 1.0	1
>1.0	4

Table 4.3: Number of Samples by Relative Error Range

In summary, the model appears to work well on the majority of training samples, with the relative error being low for most of them, but it could still be improved by further tuning or error handling on the outliers, which constitute a small fraction of the overall samples having high relative error.

4.3 Training and Model Evaluation with unfreeze layers

I unfreezed the layers and then I executed the code for fine-tuning. The fine-tuning process involved unfreezing the layers of my model to allow the model to adapt its pre-trained features to the specific task of predicting eye coordinates. Below is the summary of parameters:-

Parameter Type	Count
Total parameters	54,541,988
Trainable parameters	54,481,444
Non-trainable parameters	60,544

Table 4.4: Model Parameters Overview

However, I executed till 20 epochs because of the limitation of the computational power. If I had the device with more computational power then my model could execute to 100 epochs and will get different and better results because I don't have so I executed only 20 epochs and got the same results as an unfreezing model. Due to following reasons:-

4.3.1 Reasons:-

1. Learning Rate Constraints:

I used very low learning rate like '1e-5'. While a small learning rate is generally effective in preventing drastic weight changes during fine-tuning, in this case, it could have slowed down learning excessively, leading to no visible improvement in performance. An alternative approach would be to use a slightly larger learning rate or employ a learning rate scheduler to dynamically adjust the learning rate during fine-tuning.

2. Pre-trained Features Adaptability:

The InceptionResNetV2 architecture comes with highly robust pre-trained features that have been fine-tuned on the ImageNet dataset. It is possible that these features are already well-adapted to the eye-coordinate prediction task, and additional fine-tuning did not introduce any further meaningful improvements. This observation highlights the strength of the pre-trained model, which might already be capturing the necessary patterns without needing further adjustments.

3. Potential Overfitting:

The minimal changes in both training and validation losses suggest that the model might already be overfitting or has achieved its peak performance. This could be due to the relatively small dataset or lack of diversity in the data. Since fine-tuning did not lead to improvements, it may indicate that the model has already extracted all possible patterns from the dataset and is now just memorizing the data.

4. Saturation in Performance Metrics:

The metrics—MAE, MAPE, and relative error did not show significant variation throughout the 20 epochs. This indicates that the model's architecture and hyperparameters are already well-optimized for this task, or that the dataset lacks the complexity needed to push the model further. Even with fine-tuning, the performance metrics are saturated, suggesting that fine-tuning is not adding any value at this stage.

5. Possible Improvements:

To improve the fine-tuning process, the following changes could be considered:

Increase the learning rate: A slightly larger learning rate could allow for more meaningful updates during fine-tuning.

Unfreeze only specific layers: Instead of unfreezing all layers, selectively unfreezing the higher-level layers (closer to the output) might help refine only the task-specific features while keeping the low-level features intact.

Expand the dataset: A larger or more diverse dataset could push the model to learn more complex patterns and avoid overfitting.

Regularization: Adding more aggressive data augmentation or using stronger regularization techniques (such as increased dropout or weight decay) could help the model generalize better.

In summary, while the fine-tuning process was set up correctly, the lack of improvement in performance suggests that the model might have already reached its optimal configuration. Further adjustments, such as increasing the learning rate or expanding the dataset, might be necessary to push the performance beyond the current limits.

Chapter 5

Conclusion

5.1 Main Contributions and Achievements

Overall the major contribution of this project was the development of a deep learning model for eye centre localisation using eye tracking data. Here I utilised a good pretrained architecture InceptionResNetV2 for our feature extraction task that leverages the powerful CNNs in Convolutional Neural Networks(CNNs). We decided to go with InceptionResNetV2 architecture because it uses Inception modules with many parallel pathways to learn features at different receptive fields using skip connections between layers. Its combination of convolutional modules along with residual connections, which allow the model to learn features with greater efficiency than standard convolution layers, with a smaller number of weights, ideal for this problem with many features that need to be combined. I trained that model with the help of data preprocessing, augmentation, and fine-tuning techniques on train dataset to get the highest accuracy of our model in predicting the eye coordinates[3].

Data preprocessing played a critical role in enhancing the performance of the model. By normalizing pixel values, resizing images, and applying random contrast and brightness adjustments, the project was able to introduce variations in the data that helped the model generalize better to unseen examples. This generalization capability was critical, as eye-tracking datasets often involve subtle variations in eye positions that can be challenging for machine learning models to capture. Furthermore, the data augmentation techniques were carefully chosen to ensure that the model could effectively handle different lighting conditions and contrast levels, while avoiding transformations that could distort the core spatial features relevant to eye coordinate prediction[15].

The model was trained on a comprehensive dataset, which was pre-processed prior to feeding it to the CNN, including features such as normalized X coordinates (lx, rx in the image) and Y coordinates (ly, ry) representing the position of both the left and right eyes of the subjects in the image. In this way, the CNN architecture has been fine-tuned, and the convolutional filters are adapted to predict eye coordinates.

The success of this model can be evaluated from its performance on the test dataset. It scored a test Loss ≈ 0.00273, test MAE $= 0.04236$, and a relative error of 0.15917, which indicates that the model is capable of predicting the coordinates of eye centres in images, even when the two eyes are unusually close, reducing representational cues, or when the quality of the image is poor.

The real value added by the project is not necessarily in the predictive model itself, but rather with lessons learnt about the practical art of eye coordinate prediction: dealing with eye-position variations at subtle spatial scales, robustly generalising across subjects, and achieving computational efficiency. The project successfully addressed each of these aspects by utilising the popular InceptionResNetV2 architecture, along with data augmentation and fine-tuning techniques, which capitalised on the model's performance in computational benchmarks for object detection. This approach and progress are expected to be replicable when applying deep learning to eye-tracking technology in areas such as human-computer interaction or cognitive load assessment[19].

5.2 Project Status and Evaluation

Overall, the project has achieved most of its set goals and objectives thus far. With regard to the aim of developing a deep learning model capable of localising eye centres, that was fully achieved as the model demonstrated strong predictive performance when pitted against feedback from test data. To effectively evaluate the performance of the model to localise the eye centre as laid out in Chapter 4 above, we applied several performance metrics: the Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and a relative error metric I created myself in code as a function ('e = d / w')[8]. Using the deep learning model we have built, we indeed achieved the maximum level of prediction accuracy possible for input conditions very distant from those involved in training the model.

While the core objective of predicting eye center coordinates was accomplished, some secondary aims, such as exploring the relationship between eye and head movement, were not completed due to time and scope constraints. This aspect of the project would have required additional data, particularly data capturing head poses, and the development of a multi-modal model that could integrate both eye and head movement. While this was not achieved in the current iteration of the project, it remains an important direction for future research.

Additionally, the model's ability to generalize across different datasets and user demographics was partially achieved since the training data was sufficiently representative of variation in eye positions across illuminants and contrasts, while not sufficiently representative of different demographic groups, such as age, ethnicity or facial features. This limitation highlights the need for further research and testing on more diverse datasets to ensure the model's robustness in real-world applications[18].

One important observation from the project was the behaviour of the model during fine-tuning. Despite unfreezing the layers of the pre-trained model and continuing training, the results did not show significant improvement. This could indicate that the pre-trained features were already well-adapted to the task, and further fine-tuning with a small learning rate did not yield better results. It also suggests that the current dataset may have been sufficient to train the model to an optimal level, meaning additional improvements would require either more data or more advanced techniques like hyperparameter tuning or alternative architectures.

In conclusion, the project was a success with a fully working deep learning model for eye coordinate prediction with high accuracy. The performance of the models is in line with the initial goals and there are lots of directions that could be explored to improve the systems. The limitations that arose during the finetuning process are an interesting insight into how the models behave and how to work with pre-trained architectures.

5.3 Open Problems and Future Plans

Beyond this, although my project made a great step forward in developing a model for modelling eye centre localisation, there are still plenty of open problems and possible avenues for future research. One of the most important is the relationship between eye and head movement – something that was originally part of the project's aims. This relationship is particularly important when head movement might affect gaze direction (for instance, using a VR system, or an advanced human-computer interaction interface). The future work could try to fill this gap by leveraging additional data sources like head pose estimation or full facial landmarks detection, and building a multi-modal model for modelling the relationship between eye and head movements[15].

A second important direction for future research is dataset expansion. While the current schema was sufficient to train the model, it was demographically limited, so the actual generalisation capabilities of the model were not assessed against a sufficient range of real-world scenarios. A larger and more diverse dataset containing images of subjects from a wider range of ages, ethnicities and environmental conditions could provide a more rigorous assessment of performance and identify any biases that might exist. Moreover, an expanded dataset containing images captured under different lighting conditions or at different resolutions could assist the model to better generalise to an extended set of real-world needs.

At a technical level, future research might consider alternative kinds of deep learning architectures. While the InceptionResNetV2 architecture worked well for this project, newer architectures like Effi-

cientNet, or Vision Transformer (ViT) architectures might offer a better balance of accuracy or lead to greater computational efficiency. These newer architectures have proven successful across a variety of computer vision tasks and might offer additional benefits when localising eye centres. Beyond modelling and experimentation, employing hyperparameter optimisation techniques when training your model can also lead to better results. Rather than manually tweaking a model's parameters – such as the number of hidden layers in a deep neural network, or the learning rate for training – hyperparameter optimisation techniques like grid search (or more computationally expensive but more effective methods like Bayesian optimisation) can identify more effective combinations of parameters in order to improve performance[15].

Other directions for future work might also involve further refinement of the fine-tuning process. Unfreezing the layers in the base model did not result in notable improvements in performance in this project, but this might be due to the already well-adapted nature of the pre-trained features. Future experiments could focus on unfreezing only certain layers (perhaps the upper convolutional blocks), allowing the model to adapt only the features that are most pertinent for eye localisation. Other optimisation techniques such as regularisation through DropConnect or stochastic depth could be applied to mitigate any overfitting that might take place during fine-tuning.

Most vitally, of course, we would like to take the model into real-time eye-tracking systems. The focus of the current project was on achieving maximum precision in our predictions; real-time performance would be important for applications such as gaze-based interfaces or adaptive learning environments. Future work in this direction would consider the various ways in which the model could be optimised for speed – perhaps through model pruning for a leaner model, or by implementing specialized hardware to accelerate execution. We also consider the use of transfer learning in real-time: for the detection system on people's eyes in a real-time system that could adapt to someone's eye characteristics over time and become personalised and adaptive in its use of eye detection.

In closing, although the current project managed to fulfil its main goals, there are several ways of extending the current work that could improve the accuracy or applicability of the model. These include the analysis of the relation between eye and head movement, working with an extended dataset, trying different architectures, fine-tuning the model and optimising it to achieve real-time performance on a standard computer. These directions for future work could lead to more sophisticated, robust and user-friendly eye-tracking systems.

Bibliography

[1] Tad T Brunyé, Trafton Drew, Donald L Weaver, and Joann G Elmore. A review of eye tracking for understanding and improving diagnostic interpretation. *Cognitive research: principles and implications*, 4:1–16, 2019.

[2] Emily Caveness, Paul Suganthan GC, Zhuo Peng, Neoklis Polyzotis, Sudip Roy, and Martin Zinkevich. Tensorflow data validation: Data analysis and validation in continuous ml pipelines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2793–2796, 2020.

[3] Nicholas Dominic, Tjeng Wawan Cenggoro, Arif Budiarto, Bens Pardamean, et al. Transfer learning using inception-resnet-v2 model to the augmented neuroimages data for autism spectrum disorder classification. *Commun. Math. Biol. Neurosci.*, 2021:Article–ID, 2021.

[4] Carlos A Ferreira, Tânia Melo, Patrick Sousa, Maria Inês Meyer, Elham Shakibapour, Pedro Costa, and Aurélio Campilho. Classification of breast cancer histology images through transfer learning using a pre-trained inception resnet v2. In *International conference image analysis and recognition*, pages 763–770. Springer, 2018.

[5] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2015.

[6] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[7] Timothy O Hodson. Root mean square error (rmse) or mean absolute error (mae): When to use them or not. *Geoscientific Model Development Discussions*, 2022:1–10, 2022.

[8] Oles Hospodarskyy, Vasyl Martsenyuk, Nataliia Kukharska, Andriy Hospodarskyy, and Sofiia Sverstiuk. Understanding the adam optimization algorithm in machine learning. 2024.

[9] Oliver Jesorsky, Klaus J Kirchberg, and Robert W Frischholz. Robust face detection using the hausdorff distance. In *Audio-and Video-Based Biometric Person Authentication: Third International Conference, AVBPA 2001 Halmstad, Sweden, June 6–8, 2001 Proceedings 3*, pages 90–95. Springer, 2001.

[10] Seunghyun Kim, Seungkeon Lee, and Eui Chul Lee. Advancements in gaze coordinate prediction using deep learning: A novel ensemble loss approach. *Applied Sciences*, 14(12):5334, 2024.

[11] Ren-Jieh Kuo, Hung-Jen Chen, and Yi-Hung Kuo. The development of an eye movement-based deep learning system for laparoscopic surgical skills assessment. *Scientific Reports*, 12(1):11036, 2022.

[12] Huimin Li, Jing Cao, Andrzej Grzybowski, Kai Jin, Lixia Lou, and Juan Ye. Diagnosing systemic disorders with ai algorithms based on ocular images. In *Healthcare*, volume 11, page 1739. MDPI, 2023.

[13] Yuzhou Lin, Ramaswamy Palaniappan, Philippe De Wilde, and Ling Li. A normalisation approach improves the performance of inter-subject semg-based hand gesture recognition with a convnet. In *2020 42nd annual international conference of the IEEE engineering in medicine & biology society (EMBC)*, pages 649–652. IEEE, 2020.

[14] Eva Maia Malta, Sandra Avila, and Edson Borin. Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 21–29, 2019.

[15] Gabriella Miles, Melvyn Smith, Nancy Zook, and Wenhao Zhang. Em-cogload: An investigation into age and cognitive load detection using eye tracking and deep learning. *Computational and Structural Biotechnology Journal*, 24:264–280, 2024.

[16] Jörg Rieskamp. The importance of learning when making inferences. *Judgment and Decision Making*, 3(3):261–277, 2008.

[17] Aditya Sharma, Vishwesh Ravi Shrimali, and Michael Beyeler. *Machine Learning for OpenCV 4: Intelligent algorithms for building image processing apps using OpenCV 4, Python, and scikit-learn.* Packt Publishing Ltd, 2019.

[18] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[19] Vasileios Skaramagkas, Giorgos Giannakakis, Emmanouil Ktistakis, Dimitris Manousos, Ioannis Karatzanis, Nikolaos S Tachos, Evanthia Tripoliti, Kostas Marias, Dimitrios I Fotiadis, and Manolis Tsiknakis. Review of eye tracking metrics involved in emotional and cognitive processes. *IEEE Reviews in Biomedical Engineering*, 16:260–277, 2021.

[20] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[21] Shui-Hua Wang, Preetha Phillips, Yuxiu Sui, Bin Liu, Ming Yang, and Hong Cheng. Classification of alzheimer's disease based on eight-layer convolutional neural network with leaky rectified linear unit and max pooling. *Journal of medical systems*, 42:1–11, 2018.

[22] H-C Wu, N-I Wu, C-S Tsai, and M-S Hwang. Image steganographic scheme based on pixel-value differencing and lsb replacement methods. *IEE Proceedings-Vision, Image and Signal Processing*, 152(5):611–615, 2005.

[23] Jun Yang and Fei Wang. Auto-ensemble: An adaptive learning rate scheduling based deep learning model ensembling. *IEEE Access*, 8:217499–217509, 2020.

[24] Xingpeng Zhang and Xiaohong Zhang. Global learnable pooling with enhancing distinctive feature for image classification. *IEEE Access*, 8:98539–98547, 2020.

[25] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[26] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11127–11135, 2019.