University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Convolutional Approaches to Facial Age Estimation: Generalisability and Bias Analysis of Machine Learning and Deep Learning Methods

Sam Beattie

February 3, 2025

Abstract

Accurate age detection is crucial for various applications, yet traditional methods like self-declaration or human estimation suffer from limitations such as intrusiveness, privacy concerns, and inconsistency. This dissertation investigates convolutional approaches to facial age estimation, comparing a machine learning approach using Support Vector Machines (SVMs) with a deep learning approach using Deep Residual Neural Networks (ResNet). The study focuses on analysing the generalisability and bias of these methods across two distinct datasets: APPA-REAL and UTKFace. Results demonstrate the effectiveness of ResNet for age estimation, achieving strong performance in binary classification, multi-class classification, and regression tasks. However, both approaches exhibit biases related to gender and race, highlighting the importance of addressing fairness in automated age detection systems.

Education Use Consent

Contents

1 Introduction

1.1 Motivation

Accurate age detection is crucial for a wide range of applications, from ensuring compliance with legal age restrictions in retail to personalising content in online platforms. In many cases, computer systems or humans need to verify the age of a person before proceeding. Traditional methods of age verification, such as self-declaration, require users to manually submit documents or personal information. While straightforward, YOTI (2023) suggests this has been shown to negatively impact user retention in digital applications, as individuals often find such processes intrusive and time-consuming. Furthermore, reliance on user documents introduces privacy concerns, which can lead to a reluctance in providing such sensitive data.

Human age estimation is another widely used approach, particularly in the retail industry, where workers may have to comply with the 'Challenge 25 Act', in which they must request identification before selling alcohol to those they determine to be under 25 years old. However, Ganel et al. (2022) suggests that humans perform poorly in age estimation, with estimates often influenced by bias, gender, and race. Such inconsistencies can introduce potential legal violations.

Machine learning and deep learning methods offer a promising solution to automate and improve age detection tasks. By leveraging large amounts of data, it is possible to train a machine learning system to accurately determine the age of a person using facial images. Such a system could improve user convenience and increase retention, while also removing inconsistencies caused by erroneous human age estimation.

Given the challenges with traditional verification methods and human estimation, there is a clear need for automated, accurate, and unbiased age detection systems. While computational approaches offer the potential for seamless user experience, their successful implementation requires careful consideration of both performance and practical constraints.

1.2 Objectives

Despite their potential benefits, automated age detection systems face several challenges. Model performance can vary under different image conditions, such as low illumination, noise, or facial occlusions. Additionally, models can often demonstrate bias, with performance varying across race or gender.

With these factors in mind, this dissertation will focus on the following key objectives:

- Investigate convolutional approaches to human age estimation.
- Produce successful age estimation models and evaluate their performance and generalisability.
- Analyse model bias in terms of gender and race.

1.3 Dissertation Structure

This dissertation is structured into the following sections:

- **Background** - Reviews existing work done in human age estimation, including commercial products and academic research, and introduces the technical methods used.
- **Methodology** - Describes the approach to achieve the objectives, including the entire classification pipeline.
- **Implementation** - Details the technical aspects of the methodology, covering dataset, model implementation, and optimisation.
- **Evaluation** - Present the results, including performance analysis, comparisons between models, and discussion of key findings.
- **Conclusion** - Summarises the work done, discusses challenges faced and limitations, and outlines potential future directions.

2 Background

2.1 Commercial Products

There exists a variety of commercial software products for age estimation, many of which are targeted to businesses for the regulation or personalisation of user content. This section will provide a review of these products and highlight any key considerations.

2.1.1 YOTI

YOTI Ltd. is a digital identity company offering age verification software, they are widely trusted for age verification tasks - working with government bodies, banks, and other large entities. In their recently published white paper YOTI claims a mean absolute error of less than 2.6 years for age estimation of ages 6-70 (YOTI 2024). In that same paper, YOTI highlights the importance of understanding model bias in relation to gender and race - providing separate accuracy metrics for male, female and across varying skin tones.

Their work demonstrates several key findings. Firstly, accuracy varies across different age ranges, with best performance achieved in the 6-12 year range where mean absolute error drops to 1.3 years, while the 25-70 age range achieved a mean absolute error of 3.0 years. This age-dependent accuracy pattern is particularly important given that many applications focus on age verification around legal thresholds like 18 or 21 years, as such it may be desirable to tune an age detection model for optimal performance in application specific age ranges.

The system's performance also varies under different image conditions. YOTI reports decreased accuracy when dealing with poor lighting, motion blur, or partial facial occlusion. These practical limitations highlight the importance of image quality in real-world age estimation applications. Their results suggest that facial features become less distinct under these conditions, making age-related characteristics harder to detect.

While some use cases allow for ideal image conditions, many real-world applications must account for variation in illumination, subject orientation, or occlusions. As such, developing an age estimation system which is robust to such conditions is crucial. Additionally, understanding model bias and failure cases is essential to tune a model for the desired application.

Although YOTI does not disclose the exact implementation details of its system, it describes the approach as utilizing a neural network. Given the reported performance and common practices in facial analysis, it is likely that their model employs a deep convolutional neural network (CNN) or a similar architecture for age estimation.

2.1.2 Amazon Rekognition

Amazon provide an age estimation system as a part of Amazon Rekognition (AWS 2023). Rekognition is a cloud-based computer vision platform that offers a variety of facial analysis features, including face detection, emotion recognition, and demographic estimation. Among these, its age estimation capability predicts an age range rather than a precise value, reflecting the inherent uncertainty in age estimation from facial images.

Unlike some dedicated age verification systems, Amazon Rekognition is designed for general-purpose image and video analysis, making it applicable across various industries, from content moderation to customer analytics. The model provides a confidence score for its predictions; this allows users to assess the reliability. However, the use of age ranges limits it's usability for legal age verification situations, where ranges may produce uncertain outcomes.

Amazon state that the model performance varies depending on the resolution of the input image as well as on the image conditions such as illumination, sharpness, rotation, and facial occlusion. As such it is again made clear that model performance varies largely depending on the quality of its input.

Amazon does not publicly disclose the specifics of it's architecture or training data. However, AWS (2023) states that it uses deep learning based image analysis, as such it is likely based on a deep CNN trained on a large dataset - as is the trend with many deep learning image classification tasks.

2.1.3 Summary of Commercial Products

The review of commercial age estimation products highlights several key trends and challenges in the field. Both YOTI and Amazon Rekognition utilize similar approaches, likely leveraging CNNs for facial analysis. However, their applications differ - YOTI is specifically designed for age verification, while Amazon Rekognition provides general-purpose facial analysis, including age estimation as one of many features.

A common theme across both systems is performance variability due to image quality factors such as lighting, resolution, and occlusion. While these models achieve high accuracy under ideal conditions, real-world use cases often introduce challenges that degrade performance. Additionally, bias in age estimation is a recognized concern, particularly in relation to gender and ethnicity, as reported by YOTI.

The insights gained from commercial products emphasize the importance of robustness and fairness in age estimation models. They also highlight different approaches to age estimation using either exact age estimation, or age range estimation with confidence scores. Additionally, in the case of YOTI it is clear that prioritising accuracy in some age ranges over others may be beneficial - particularly in legal applications.

2.2 Machine Learning

There has been much work done in the field of computer vision for facial analysis and age detection. Traditionally, these tasks have relied on machine learning approaches. However, deep learning has recently taken over as the preferred approach to such problems. Here, I will provide an overview of the machine learning methods used for computer vision.

Machine learning has gained immense popularity in recent years due to many new applications, particularly concerning deep learning; however, machine learning as a field of study is far from new. The first machine learning program was created in the 1950's by Arthur Samuel, who later coined the term 'machine learning' in 1959. Furthermore, the field originates from even earlier efforts to understand the human cognitive process.

Machine learning is a field of artificial intelligence which aims to develop statistical algorithms capable of learning from data and generalising to unseen inputs. This allows such systems to be applied to a variety of tasks for which they have not been explicitly programmed.

Machine learning is generally separated into three categories:

- **Supervised learning** involves training with labelled data, with the goal of learning an underlying mapping from unseen inputs to their outputs. This category encompasses the majority of computer vision tasks.
- **Unsupervised learning** involves training with unlabelled data, where the model must learn the structure of the input data. This is often involves finding groupings or patterns within the data.
- **Reinforcement learning** involves the program interacting with an environment to perform a certain objective, receiving feedback in the form a reward which it must maximise.

2.2.1 Support Vector Machines

There are many machine learning algorithms, with the choice of which to use depending on the task at hand; one such algorithm is the Support Vector Machine (SVM). SVMs were first introduced by Vapnik et al. (1992), they are a type of supervised machine learning algorithm that classifies data by finding an optimal hyperplane that maximally separates classes within the data. They use kernel functions which allow them to map non-linearly separable data into higher dimensions, allowing classification even for complex data. The decision boundary is defined by support vectors, which are the data points closest to the hyperplane.

One common application of machine learning is computer vision, which involves using machine learning algorithms to classify image data. Within computer vision, SVMs have been widely used for human age estimation due to their effectiveness in handling high-dimensional feature spaces, such as handcrafted facial features extracted using Local Binary Patterns (LBP) and Histogram of Oriented Gradients (HoG). As such, there are many examples of SVMs used for human age estimation (Madhavi et al. 2021; Ghosh and Bandyopadhyay 2015; Hiremath et al. 2022).

2.3 Deep Learning

While traditional machine learning approaches to computer vision are powerful, they have notable limitations. One such limitation is their inability to process raw data - instead having to rely on feature extraction to produce a feature vector (LeCun et al. 2015). The choice of which features to extract from an image is largely up to the implementer of the algorithm, often requiring significant domain knowledge. This results in many potential features to choose from, each of which comes with its own set of hyper-parameters. These can have a large impact on the performance of the algorithm, resulting in a considerable amount of manual testing and tuning of different features and hyperparameter values.

These challenges highlight the limitations of traditional machine learning approaches in computer vision. As a result, researchers have turned to deep learning, which can alleviate many of these issues, while offering significant advantages (O'Mahony et al. 2020). Deep learning is a subfield of machine learning which started in the 1980's and has had many recent advancements, largely due to improvements in computing power and increased access to large amounts of data - both of which are crucial for deep learning applications.

2.3.1 Neural Networks

Deep learning can leverage large amounts of data and computing power to learn abstract representations of the data. Deep learning focuses on the neural network, which is a machine learning model that imitates (a simplified understanding of) the human brain.

Artificial neurons are the building blocks of a neural network, these neurons are made up of three components: the weights, bias, and activation function. When a neuron receives an input (in the form of a floating point number or set of numbers) it performs a sequence of operations:

it first multiplies each input by the corresponding weight, then sums these products and adds the bias term, and finally passes the weighted sum through the activation function to ensure non-linearity, shown in Figure 2.1.

An artificial neural network (ANN) is a network of these artificial neurons, consisting of layers of neurons where the output from each neuron in one layer is passed as the input to one or more neuron(s) in the next layer. By stacking or layering multiple of these artificial neural networks, we obtain a deep neural network - this is the basis of deep learning.

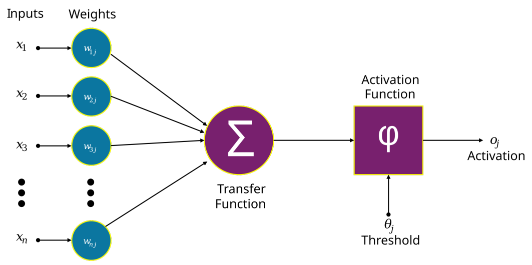


Figure 2.1: *A diagram of the structure of an artificial neuron, showing the inputs and weights being added in a weighted sum, then being passed through an activation function. The bias is committed for conciseness.*

Source: Wikipedia, CC0 1.0 Universal Public Domain Dedication, *`https://commons.wikimedia.org/wiki/File:Artificial_neuron_structure.svg`*

Any neural network, deep or otherwise, maps an input to an output by approximating the underlying function that relates them, it does this by learning patterns in the data through its weights and biases. An untrained network will be a poor approximation, with the predicted output far from the ground truth. We can quantify this error through the use of a loss function; there are many loss functions to choose from, with the choice of which to use being task-specific, however, they all produce some loss value which can be used to determine the error in the network's predictions.

Improving a networks performance is analogous to minimising its loss - to do this we use a method called backpropagation. The output of a neural network is a function of its input and parameters (weights and biases), backpropagation works by finding the partial derivative of the loss function with respect to each weight and bias in the network using the chain rule. This involves propagating gradients backwards through the network, layer by layer, starting from the output. These gradients indicate how each parameter should be adjusted to reduce the error. Using an optimisation algorithm like gradient descent, the parameters are updated in the direction that decreases the loss. This process is typically repeated multiple times until the network converges to a set of parameters that minimise the loss. (Nielsen 2015)

2.3.2 Convolutional Neural Networks

ANNs are extremely powerful, however, as explained by O'Shea and Nash (2015), they struggle with the added computational complexity of image data. For example, in the case of a low resolution 28 x 28 greyscale image, an ANN can work quite well, as the neurons in the first layer would have only 784 (28 x 28) weights - one for each pixel. However, if we wanted to use an ANN for higher quality image data, say a 64x64 image with 3 colour channels, then the number of weights of a single neuron in the first layer would increase to 12,288 (64 x 64 x 3). While it is possible to train a network with many parameters, it introduces additional computational demands, as well as the risk of overfitting.

Convolutional neural networks (CNNs) are a type of neural network primarily used in the field of pattern recognition within images. They are analogous to traditional ANNs in that they are built upon layers of neurons which are optimised through training. The notable difference is that they are set up in a way which is suited best for dealing with image data. A CNN consists of 5 key parts:

- The **input layer** is the same as in any ANN, and contains the pixel values of the image
- The **convolutional layers** consists of a set of learnable kernels which are convolved across the width and height of the input, they compute the dot product between the kernel weights and a region of the input and produce a feature map. This means that the network can learn mappings which activate upon detecting a specific type of feature in a region of the input.
- A **non-linear function** is applied to the feature map. Generally, the rectified linear unit (ReLU) function is used for this.
- The **pooling layers** will perform downsampling to reduce the number of parameters in the feature maps.
- The **fully connected layer** will then produce class scores from the feature maps.

(LeCun et al. 2015; O'Shea and Nash 2015).

A typical CNN will contain an input layer, followed by several stages of convolution, non-linearity, and pooling, finally followed by a fully connected layer, shown in Figure 2.2

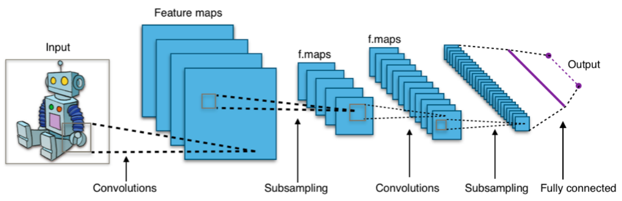


Figure 2.2: *A diagram of the structure of a typical CNN, showing two stages of convolutions and pooling, followed by a fully connected layer. Note that the nonlinearity is non explicitly shown and is performed during the convolution and pooling is referred to as 'subsampling'.*

Source: Wikipedia, Creative Commons Attribution-Share Alike 4.0 International, `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`

2.3.3 ResNet

Deep CNNs have been highly influential in image classification, with one of the key benefits being that increased network depth leads to improved performance, as the network can approximate more complex functions (Zhong et al. 2019). This, alongside recent advancements in computing power have allowed for excelent results in image classification.

However, some issues arise as networks become deeper. One issue is that in deep networks, backpropagation can lead to gradients approaching zero - this is known as the vanishing gradient problem. This issue was largely mitigated by techniques such as normalised initialisation and the ReLU activation function. Another key challenge was the degradation problem, this was the rather counter-intuitive problem that in deep networks, the accuracy saturates, then degrades rapidly. Importantly, this degradation is not caused by overfitting. (He et al. 2016; Goodfellow et al. 2016)

Deep residual networks (ResNet) were introduced by He et al. (2016) and provide a solution to the degradation problem. They work by introducing a 'skip connection' and by having the stacked layers approximate the residual function in place of the standard underlying function; this means that rather than learning the full transformation directly, ResNets instead learn the difference (or 'residual') between the input and the desired output, which helps with optimization and gradient flow. The skip connections allow the input to bypass one or more layers, this ensures that important information is retained, helping to mitigate both the degradation and vanishing gradient problems. These skip connections ensure that deep layers retain access to the original input. We can think of this as helping the network not to 'forget' the input.

ResNet has since been widely adopted for many computer vision tasks due to its success with deep networks (Mandal et al. 2021; Gruber et al. 2017). ResNet architectures come in various depths, including ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, where the number indicates the total number of layers in the network. Shallower models like ResNet-18 and ResNet-34 consist of basic blocks with two layers each, since there are fewer parameters, they are more computationally efficient and work well with smaller datasets. Deeper models such as ResNet-50, ResNet-101, and ResNet-152 utilize bottleneck blocks comprising three layers each, enabling them to capture more complex feature (Koonce and Koonce 2021).

2.3.4 Transfer Learning

One of the primary issues with deep learning is the amount of computing resources required to train large models. For example, training ResNet-18 (the smallest of the ResNet architectures) would require updating ~11.7 million parameters per training epoch. This becomes computationally demanding for large datasets, and the problem is further exacerbated in deeper architectures.

A solution to this problem is transfer learning. Transfer learning is described by Weiss et al. (2016) as the process of improving predictive abilities in one task using related information from another domain and task. For example, in computer vision, transfer learning could involve using a model trained on ImageNet for image classification and applying it to a face or age estimation task.

Transfer learning can alleviate the computational cost of training by using a model pre-trained on one dataset and applying it to another. As Zhuang et al. (2020) explain, "if we have a neural network for the source task, we can freeze (or say, share) most of its layers and only finetune the last few layers to produce a target network." Typically, the earlier convolutional layers are frozen since they capture general features (e.g., edges and textures), while only the later fully connected layers are fine-tuned to adapt to the new task. This strategy significantly reduces the number of parameters that need updating while retaining the benefits of deep architectures. Additionally, since transfer learning allows for a significant reduction in learnable parameters, a model can effectively train on smaller datasets without risk of overfitting.

3 Methodology

To achieve the outlined objectives, two distinct approaches were used for age estimation via facial images:

- An initial machine learning approach using Support Vector Machines (SVMs).
- A subsequent deep learning approach using residual convolutional neural networks (ResNet).

The machine learning approach was constrained to binary age classification, whereas the deep learning approach was applied to binary classification, multi-class classification, and regression. The primary focus was not on directly comparing these methods but rather on investigating generalisability and bias in these different approaches. To this end, two separate datasets were used, and cross-dataset performance was evaluated.

3.1 Dataset Pre-processing

Often, the first step in starting a supervised learning task is to understand the data. Before training, I investigated the dataset to learn important information, such as variations in image resolution and brightness, class distribution, train/validation/test split sizes, and how well subjects were captured within the images (e.g., cropping).

The most useful information in the data comes from the face, as such pre-processing was performed to improve consistency across images. Variations in resolution, brightness, and framing can introduce confusion, as the model may learn unwanted patterns in the data, and not the relevant facial features. To address this, an automated face detection and cropping method was applied to ensure that the face remained the focal point of the image. Additionally, images without detectable faces were removed.

The dataset was then partitioned into training, validation, and test splits using a 60/20/20 ratio. A stratified sampling approach was used to maintain an even class distribution across all partitions, reducing the risk of bias in model evaluation. Two distinct datasets were used and the pre-processing steps for each are described in Chapter 4.

3.2 Age Classification using Support Vector Machines

This approach involved extracting features from the image data to construct feature vectors used for fitting the SVM. The process is visualised in Figure 3.1

3.2.1 Data Loading

Before features could be extracted, the image data was loaded and preprocessed by:

- **Reshaping** to uniform dimensions. Reshaping ensured that the feature vectors extracted were of the same size and could be compared. This process involved either upsampling or downsampling the original image.

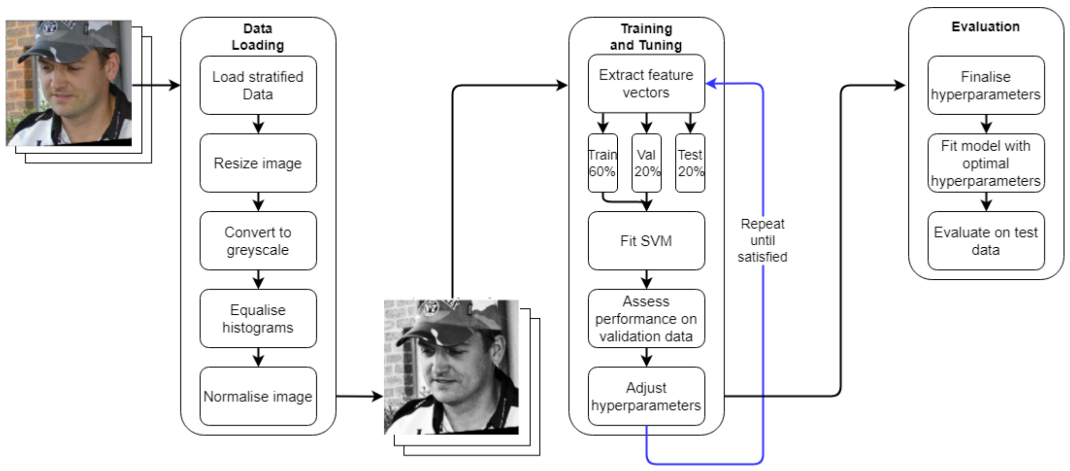


Figure 3.1: A diagram showing the process of training an SVM for age estimation. The input image undergoes preprocessing (resizing, greyscale conversion, histogram equalisation, and normalisation) before feature extraction. The extracted features are then used for training and hyperparameter tuning, with performance assessed iteratively until an optimal model is finalised

- **Converting to greyscale.** This step was performed to support the choice of feature vector and typically involved averaging the colour channels into a single channel.
- **Equalising histograms.** Histogram equalisation is a contrast enhancement technique which was used to reduce the illumination variability across different images by equalising the pixel intensities (Pizer et al. 1987). This was done to improve the feature extraction in images with poor illumination by enhancing image details.
- **Normalising.** This involved changing the range of pixel values to a new range, typically between 0 and 1 or 0 and 255. This helped maintain consistency across different features and improved stability.

This pre-processing step was essential in ensuring that images were fairly comparable in terms of resolution, illumination, and pixel intensity. Each image in the dataset was loaded following these pre-processing steps, after which they were used for training.

3.2.2 Training and Tuning

Since SVMs cannot process raw image data directly, feature extraction was performed to obtain meaningful representations of the images. These features captured important characteristics such as edges, textures, and gradient variations, which could then be used to classify the image. Feature vectors provide a more efficient representation of the image data; for example, a 150 x 150 image contains 22,500 pixel values, whereas its corresponding feature vector may be significantly smaller, often in the range of a few hundred to a few thousand values, depending on the chosen features and hyperparameters (Guyon and Elisseeff 2006). I chose to use Local Binary Pattern (LBP) and Histogram of Oriented Gradients (HoG) as image features due to their ability to extract texture and edge information respectively.

Once the feature vectors were obtained I used them to fit the SVM. A SVM is fit to data by finding an optimal hyperplane which maximises the distance between the support vectors of each class in a higher dimensional-space, were the support vectors are the vectors (data points) from each class that lie on the margin boundaries. A kernel function is used to perform calculations on the data in a higher-dimensional space, which allows for linear separation of data which may not normally be linearly separable – this is known as the 'kernel trick' .

Once the SVM was fit to the training data the performance was evaluated on the validation data. The hyperparameters were then adjusted to observe their effects on validation performance, this included both the hyperparameters of the SVM and the feature vector. The hyperparameters of the SVM include the choice of kernel, typically linear, polynomial, or Radial Basis Function (RBF), the regularisation parameter which controls the trade-off between maximising the margin and minimising the error, and gamma which determines the sensitivity a single point has on the decision boundary.

Hyperparameter tuning continued until satisfactory validation performance was achieved.

3.2.3 Evaluation

Once the optimal hyperparameters were found and finalised the SVM performance was evaluated on the validation data, which was completely separate from the training process. The final hyperparameters would be used to fit the model once more using the training data, and the model was evaluated on the test data to achieve the final performance of the model.

3.3 Age estimation using ResNet

This approach involved processing the pre-processed image data directly using ResNet and transfer learning. The process is visualised in Figure 3.2

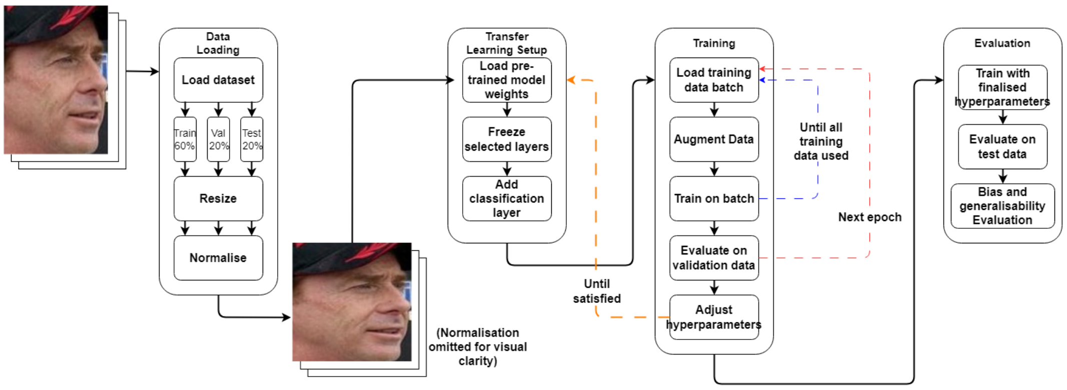


Figure 3.2: A diagram showing the process of training using ResNet with transfer learning for age estimation. The loaded data is preprocessed (resizing, normalisation). The pre-trained model weights are then loaded and select layers are frozen, a classification layers is added and then training begins. Training involves loading batches of training data, augmenting the data then training on the batch, this loops until all training data has been used. After one full epoch the model is evaluated on the validation data, then the loop begins again. Iterative hyperparameter tuning is performed until sufficient performance is obtained, then the model is evaluated on the test data as well as the second dataset to evaluate performance metrics as well as bias and generalisability.

3.3.1 Data Loading

Deep learning approaches allow for direct processing of image data, therefore, there is less need for extensive image pre-processing since we do not need to extract feature vectors. As such, the only pre-processing steps involved resizing images to standardised image dimensions, and normalisation. The normalisation used was tailored specifically for use with ResNet models pre-trained on ImageNet, and will be expanded upon in the Chapter 4.

Two distinct datasets were used in the training process. Identical models with the same hyperparameters were trained separately on these datasets, enabling comparison of performance across different data sources.

3.3.2 Transfer Learning Setup

Using the ResNet architecture there are many existing pre-trained models, the most popular of which was trained on ImageNet. By loading the weights of a pre-trained model I significantly reduced the number of trainable parameters, this allowed for faster training with limited resources and reduced the risk of overfitting on a smaller dataset.

The weights of the selected model (ResNet-18 or ResNet-50) were loaded and then frozen. Freezing the weights means that they are not updated during the backpropagation step of training. Some layers are then unfrozen, generally the later layers are unfrozen, since these extract high-level features, while the earlier layers, which often identify general image features such as edges and textures, remain frozen (Alzubaidi et al. 2021).

In order to adjust the network to perform age estimation, the final fully connected layer of the network was modified to transform the output into a predicted age label. The exact structure of this layer was task dependant:

- **Binary classification** required a single output of either 0 or 1. Therefore, the final layer was a fully connected (dense) layer mapping from the number of output features to a single output. This output was a raw logit, and applying sigmoid returned either 0 or 1.
- **Multi-class classification** required as many outputs as labels. Therefore, the final layer was a dense layer mapping from the number of output features to the number of labels. The outputs were raw logits (one for each label), and applying softmax returned the probability score of each class.
- **Regression** required a single floating point output. Therefore the final layer was a dense layer mapping from the number of output features to a single output. This time the output was the predicted age.

Once the final layer was configured, two approaches to unfreezing weights were used. The first was to unfreeze only the final dense layer, this resulted in fewer learnable parameters. The second approach was to unfreeze the dense layer and the fourth ResNet layer, this resulted in significantly more learnable parameters, while potentially allowing for more task specific features to be learnt.

3.3.3 Training

Training the model involved loading the training data in batches, these batches of training data were then augmented. Data augmentation was used for several reasons, as discussed by Shorten and Khoshgoftaar (2019), data augmentation can be used to tackle overfitting by artificially inflating the dataset, this is done through techniques such as cropping, flipping, noise injection, and rotation. Another benefit highlighted is that data augmentation can help overcome bias in datasets caused by factors such as lighting, occlusion, and scale. However, data augmentation cannot overcome all bias, such as under-represented classes within the data.

After augmentation, the batch was used to train, this training is done through backpropagation as described in 2.3.1 to update the learnable parameters. The choice of loss function depended on whether the task was binary classification, multi-class classification, or regression. This process is repeated, using batches of training data until all training data has been used to train, after which one training epoch has been completed.

At the end of each training epoch performance was evaluated on the validation data, this ensured that training was progressing steadily, and allowed for easy identification of potential issues such as overfitting. Training would repeat for a pre-determined number of epochs.

After a model had been trained the final validation metrics would be used to determine the performance, these metrics include accuracy, precision, recall, f1-score, and AUC-ROC (Area Under Curve - Receiver Operating Characteristic). The hyperparameters could then be adjusted to determine to determine their impact on model performance. The primary hyperparameters were batch size and learning rate, however, testing was also done to determine which layers to freeze as well as which ResNet architecture to use (ResNet-18 or ResNet-50). After updating hyperparameters a new model was trained using the new hyperparameters. This process repeated until satisfactory validation performance was achieved.

3.3.4 Evaluation

Once a model was obtained using optimal hyperparameters, the model performance was evaluated using the test data of both the dataset it was trained on (though this data was not involved in the training process) as well as the test data of another separate dataset. This allowed for an evaluation of bias and generalisability.

3.4 Evaluation Metrics

It is important to consider how a model is evaluated. Intuitively accuracy makes sense for classification, however, in the event of an unbalanced distribution of classes in the validation data it can be misleading. As such, a combination of various metrics were used:

- **Precision** is the proportion of correctly predicted positive cases out of all cases predicted as positive. A high precision indicates that most predicted positives were actually positive.
- **Recall** is the proportion of correctly predicted positive cases out of all actual positive cases. A high recall indicates that it is unlikely for a positive case to be predicted as negative.
- **F1-score** is the harmonic mean of precision and recall. A high f1-score indicates a good balance between precision and recall, and is generally the preferred metric for classification tasks.
- **Macro Precision/Recall/F1-score** is the average of precision/recall/f1-score for multiple classes and is used for multi-class classification tasks.
- **Area Under Curve - Receiver Operating Characteristic (AUC-ROC)** is the area under the curve of the true positive rate against false positive rate and measures the models ability to distinguish between classes. An AUC-ROC of greater than 0.5 indicates that the model is better than guessing.
- **Accuracy** is the proportion of correctly predicted cases out of the total number of cases. A high accuracy means that the model is performing well, but only if the dataset is well-balanced.
  (Hossin and Sulaiman 2015)

Throughout this project, all of these metrics are considered. While the F1-score is often the primary metric for assessing model performance, as it provides a convenient combination of precision and recall, there are instances where prioritising either precision or recall is important. In such cases, these individual metrics are used directly.

For regression tasks the Mean Squared Error (MSE) was the primary metric. This is the square of the distance between the expected output and predicted output. From this the Root Mean Squared Error (RMSE) can tell us the average distance from a predicted output to the expected output, when using RMSE the bias and standard deviation will also be presented and considered. Additionally, accuracy was used with a range of ± 5 years.

4 Implementation

This chapter will discuss the implementation details of the process as described in the previous chapter. The implementation was largely divided into two distinct approaches: the SVM approach and the ResNet deep learning approach. The dataset details are consistent across both approaches.

4.1 Dataset

For the purposes of evaluating bias and generalisability, two distinct datasets were used.

4.1.1 APPA-REAL

The APPA-REAL dataset, created by Agustsson et al. (2017), was designed to study the relationship between apparent age and biological age in face images - apparent age is the average age as estimated by other humans. I chose to use this dataset because the images are captured in less controlled environments, which makes it ideal for training a model that is robust to varying noise conditions and real-world variability. Additionally, many of the images contain glasses, hats, etc. further allowing for robustness to real-world conditions. See Figures 4.1, 4.2, 4.3 for example images.

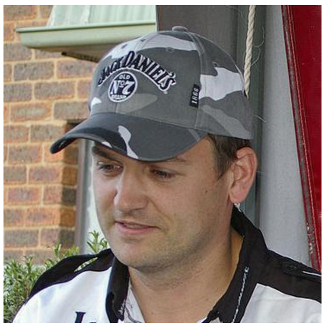


Figure 4.1: *Example image 1 from the uncropped APPA-REAL dataset*

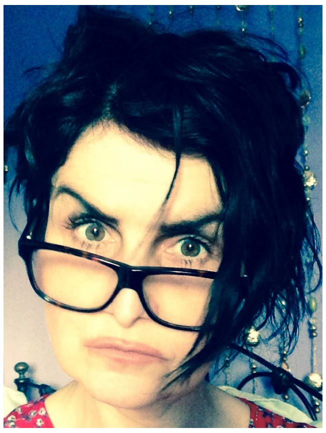


Figure 4.2: *Example image 2 from the uncropped APPA-REAL dataset*

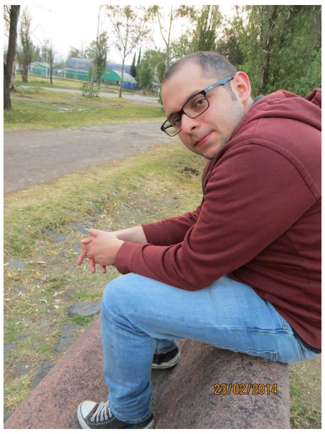


Figure 4.3: *Example image 3 from the uncropped APPA-REAL dataset*

The APPA-REAL dataset contains 7,591 images with associated real and apparent age labels. It is split into 4,113 train images, 1,500 validation images, and 1,978 test images. The dataset provides both face-cropped images and uncropped images. To minimise noise in the training data, the models were trained on the face-cropped images. This approach ensured that the models were exposed to challenging images while maintaining better performance by reducing background noise.

While APPA-REAL provides cropped images, I performed further cropping to improve the quality of the dataset. I used YOLOv8 to achieve a tighter bounding box focused purely on faces. See Figures 4.4 and 4.5 for a comparison of the APPA-REAL cropped image and my re-cropped version.

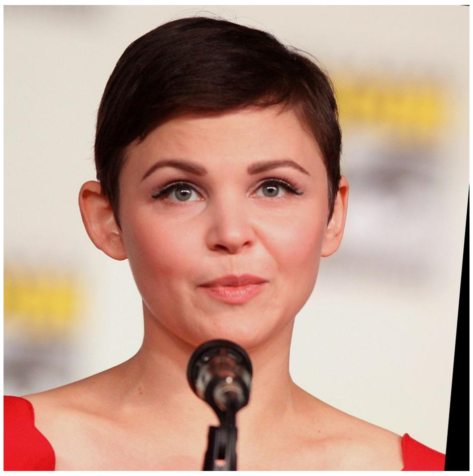


Figure 4.4: *Example image from APPA-REAL cropped*

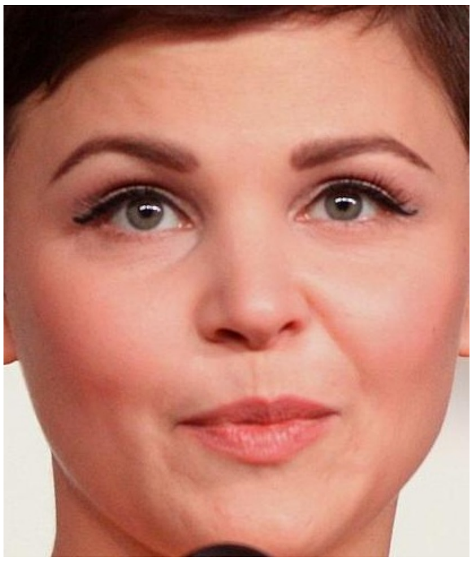


Figure 4.5: *Example image from APPA-REAL re-cropped using YOLOv8*

After cropping images and removing images with no detectable face the dataset had 3,920 train images, 1,438 validation images, and 1,828 test images. From this point on, the cropped subset of APPA-REAL was used.

Another important consideration is the distribution of ages in the dataset. The distribution is visualised using a histogram in Figure 4.6.

We can see a concentration around 25, with significantly fewer samples over the age of 60, and overall the dataset is slightly right-skewed, indicating a higher proportion of younger individuals. An inspection of the training set reveals 1,483 samples from individuals under 25 and 2,437 samples from those aged 25 and above, creating a natural class imbalance. Therefore, in an attempt to minimise bias I used a stratified approach to data loading by ensuring that there was an equal number of samples of each age group in the data, when training.

APPA-REAL is valuable for developing a age estimation system which is robust to real-world conditions. However, it lacks metadata relating to useful bias information such as gender and race, making it difficult to accurately directly asses models on fairness without first using some other system to determine the race or gender of an image - which introduces it's own error. To address this limitation, I incorporated a second dataset which not only enables bias evaluation but also provides an opportunity to assess the generalisability of models across different datasets.

4.1.2 UTKFace

UTKFace is a large face dataset created by Zhang and Qi (2017) designed for many computer vision tasks such as age estimation, gender classification, and face recognition. It contains real age labels, as well as useful race and gender metadata. The metadata was obtained using DEX

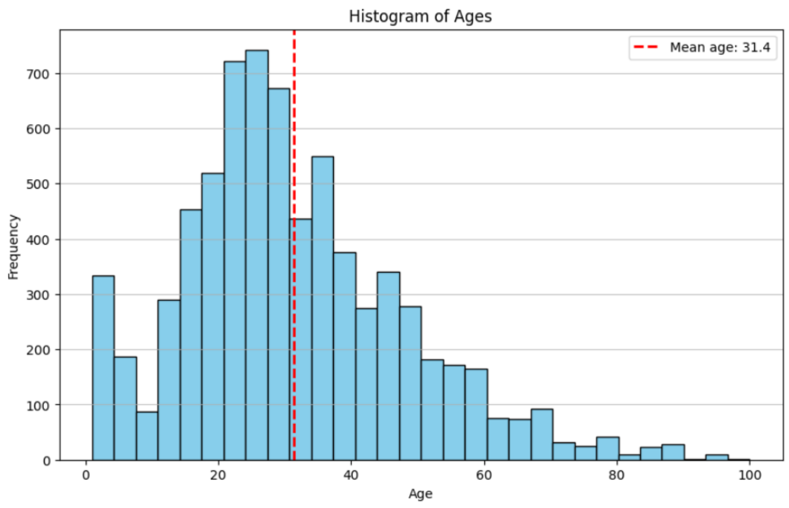


Figure 4.6: *A histogram visualising the distribution of real ages in the cropped APPA-REAL dataset. The mean age is marked as a red dashed line at 31.4. The histogram is right-skewed.*

algorithm and was double checked by a human annotator. Many images in UTKFace are captured in a more controlled environment compared to APPA-REAL, with subjects often facing the camera and under more consistent lighting conditions. This makes it valuable for evaluating model generalizability, as it allows performance comparisons between ideal and real-world conditions.

See Figures 4.7, 4.8, 4.9 for example images from the cropped UTKFace dataset

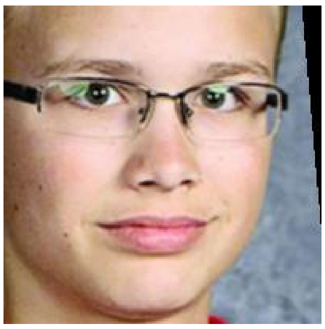


Figure 4.7: *Example image 1 from the cropped UTKFace dataset*

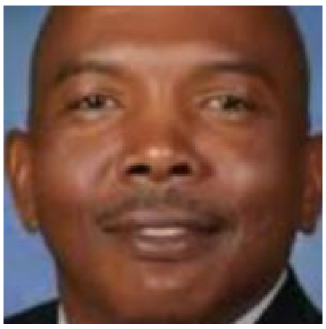


Figure 4.8: *Example image 2 from the cropped UTKFace dataset*

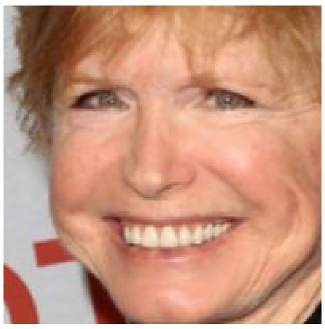


Figure 4.9: *Example image 3 from the cropped UTKFace dataset*

Again, investigating the distribution of ages in the dataset is important. The original distribution of ages in the UTKFace dataset is visualised in Figure 4.10

We can see a disproportionate number of ages less than 4, training on this data may lead to a model which is biased towards predicting younger ages. To address this, when creating the train, validation, and test splits I undersampled data points of age less than 4 in order to achieve a more uniform distribution.

Additionally, the pre-cropped images already use a strict bounding box, as such I chose not to re-crop as I did with APPA-REAL. I then created train, validation, and test splits using the cropped images and undersampling data points less than 4. This resulted in in 4972 train images,

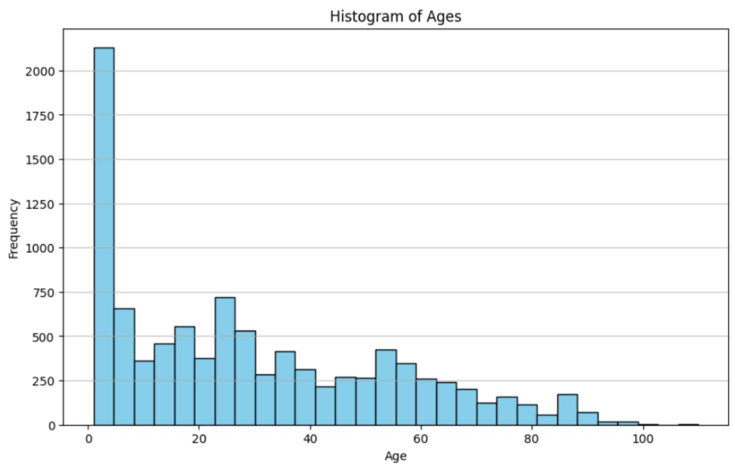


Figure 4.10: *A histogram visualising the distribution of real ages in the UTKFace dataset. There is a disproportionate number of data points of age less than 4. The histogram is right-skewed*

1658 validation images, and 1659 test images, which follow a more uniform distribution of ages. An inspection of the training split reveals 2,000 individuals under 25 and 2,972 individuals 25 and above. The the new distribution of ages in the UTKFace dataset is visualised in Figure 4.11.

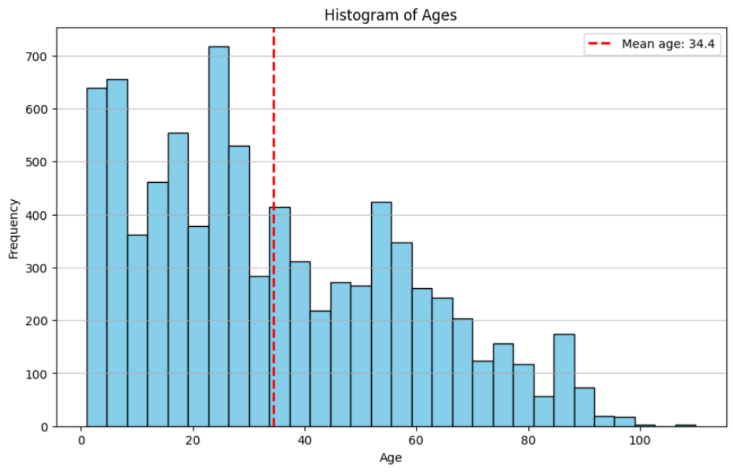


Figure 4.11: *A histogram visualising the distribution of real ages in the UTKFace dataset after under-sampling. The mean age is marked as a red dashed line at 34.4. The histogram is right-skewed*

After fixing the distribution, the dataset is still slightly right-skewed, this again indicates a higher proportion of younger individuals. In order to minimise bias I used a stratified approach to data loading, the same as with APPA-REAL.

The race and gender metadata in UTKFace is critical, as it enables easy evaluation of model bias in terms of race and gender. This works not only for models trained on UTKFace, but also allows those trained on APPA-REAL to be evaluate on UTKFace and bias to be investigated. However, it is important to understand also the distribution of races and genders within UTKFace. The distribution of races is visualised in Figure 4.12.

We can see that there is a clear bias towards white individuals in the training data, this will be a key consideration when evaluating the model performance. Keeping the distributions of race

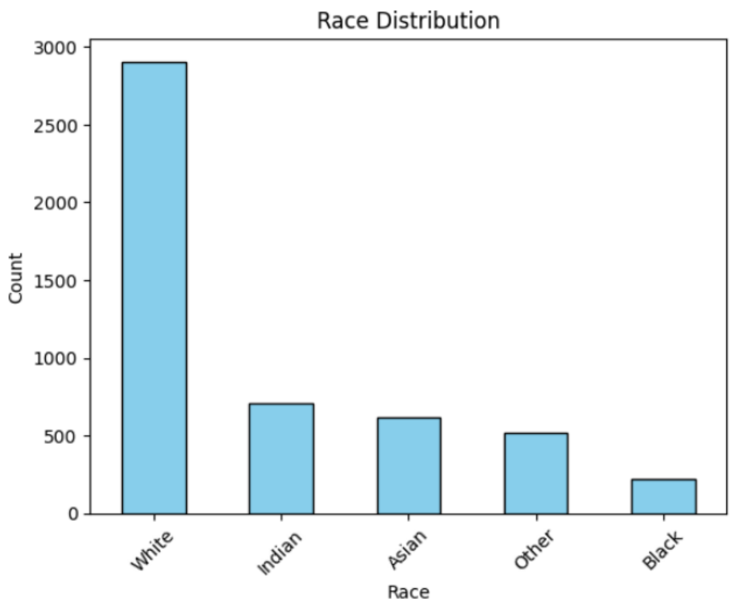


Figure 4.12: *A histogram visualising the distribution of races in the UTKFace dataset. The most frequent is White, then significantly fewer Indian, Asian, Other, and Black in order of decreasing frequency.*

and gender in mind, the additional metadata allows for a more through analysis of model bias during evaluation.

4.1.3 Summary

Two distinct datasets were used for the training and evaluation of models. APPA-REAL provides difficult images which are more representative of real-world conditions, this is ideal for training a robust model. While UTKFace provides more ideal conditions as well as useful metadata, enabling evaluation of model bias in terms of gender and race.

Additionally, the use of two datasets allows me to evaluate the generalisability of models, and compare their performance across datasets. To enable comprehensive bias analysis, I thoroughly investigated the distributions of age in both datasets, as well as gender and race distributions in UTKFace, revealing important imbalances that were considered during both training and evaluation.

4.2 Computing Environment and Libraries

This project was implemented entirely in Python 3.10.6. Primarily using the following libraries:

- **PyTorch** for deep learning implementations.
- **Scikit-Learn** for machine learning implementations.
- **NumPy** for general matrix manipulation.
- **Matplotlib** for visualisation.
- **OpenCV** for computer vision algorithms.

The hardware details are as follows

- **CPU**: 3.7GHz 12-core processor.
- **GPU**: Nvidia RTX 3070 Ti 16GB (CUDA-enabled).

- **Memory**: 16GB 3600MHz RAM.

Machine learning algorithms were primarily run on the CPU, while deep learning training was accelerated using the GPU.

4.3 SVM Approach

This approach involved using feature extraction and SVMs to perform binary age classification on face images. The data points were given a label of 0 or 1, for under 25 or 25 and over, respectively.

4.3.1 Stratified Sampling

To minimize bias, I used stratified sampling to achieve an equal distribution of class 0 and class 1 data points in the training set. All training data was loaded in one batch of size 2,800, I chose this batch size since it supports an equal sampling of class 0 and class 1 data points in both the APPA-REAL and UTKFace training splits. The APPA-REAL training split has the fewest samples from class 0 at 1,483 and was therefore the deciding factor on the batch size, for consistency I used the same batch size for UTKFace.

This was achieved by creating a custom stratified batch generator function. The generator takes as input:

- **The batch size**: 2,800 for the train split, 1000 for the validation and test splits.
- **A Pandas dataframe** containing the file names and labels of images in the desired split (train, val, or test).
- **The image directory** where the images are stored on the disk.

I will demonstrate the process for the training split. First, the dataframe is partitioned into images of class 0 and class 1. Then, a random sample of

$$\frac{\text{batch\_size}}{2} = \frac{2,800}{2} = 1,400$$

images is drawn from each class using the Pandas ‘sample’ method with the ‘replace’ parameter set to false to disallow repeated sampling of the same image. Now we have two partitions of size 1,400 from each of the two classes, which we concatenate together to obtain one batch of size 2,800.

After concatenation, the batch is initially ordered with all class 0 samples first, followed by all class 1 samples. This ordering may introduce bias in training, as one class always appears before the other. To mitigate this, the batch is shuffled before being returned.

This process was used to obtain the train, validation, and test splits, of size 2,800, 1,000, and 1,000 respectively. A roughly 60/20/20 split.

4.3.2 Image Pre-processing

As the splits were being created, the images were pre-processed before being added to the batch. This was implemented using OpenCV, a large computer vision library for Python.

After loading an image using OpenCV, the first step was to resize the image to be 150 x 150 pixels. I used the OpenCV resize function and chose the interpolation method depending on if upsampling or downsampling was required, this was primarily so I could experiment with different interpolation methods. The resizing function first calculates the resolution of the input image and compares it to that of the target size. If the input image is lower resolution than the target then upsampling was performed, otherwise downsampling was used.

For upsampling, the 'INTER_CUBIC' interpolation method was used, this is a bicubic interpolation method which uses a weighted average of the 16 nearest pixels in to determine new pixels values, this is often used for upsampling as it produces smooth results with better edge preservation. For downsampling, the 'INTER_AREA' interpolation method was used, this determines pixel values by averaging pixel regions in the image, this is often used for downsampling as it preserves detail in the image.

These interpolation methods were determined through experimentation with model performance using different methods from OpenCV, while keeping other hyperparameters consistent.

The next step was converting the image to greyscale, for this I used the OpenCV 'cvtColour' function with the 'COLOR_BGR2GRAY' parameter. This works using the following equation to convert the individual colour channels into a single greyscale intensity value:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B, \tag{4.1}$$

where Y is the new pixel intensity value. Note that BGR to grey was used as OpenCV uses the blue/green/red colour channel ordering when loading an image.

Then histogram equalisation was performed to increase the global contrast in the image by redistributing the intensity values of the pixels. This was done using the OpenCV 'equalizeHist' function. This function works by first computing the histogram of the image, which represents the distribution of pixel intensities. It then derives the Cumulative Distribution Function (CDF) from the histogram, capturing the cumulative sum of intensity frequencies. The CDF values are then mapped to a new, linear CDF, redistributing the intensity values more evenly across the image and enhancing contrast. The image histogram before and after equalisation for an image from APPA-REAL is visualised in Figures 4.13, 4.14

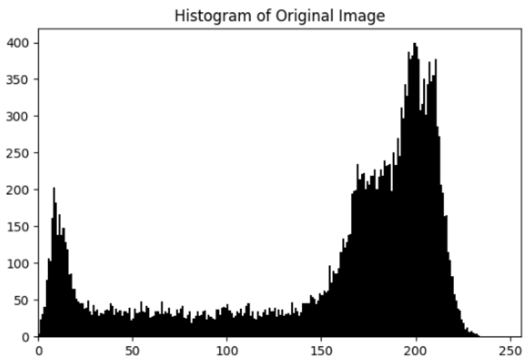


Figure 4.13: *Histogram showing the pixel intensities of an image before histogram equalisation*

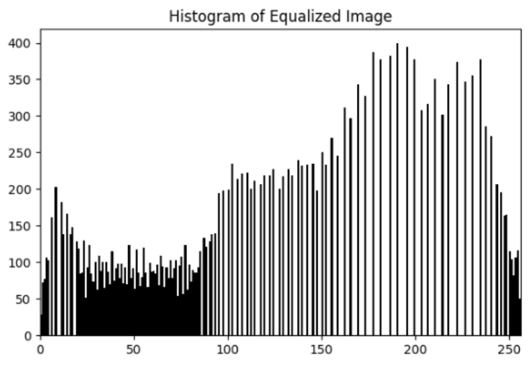


Figure 4.14: *Histogram showing the pixel intensities of an image after histogram equalisation*

Finally, the image was normalised between 0 and 255 using the OpenCV 'normalize' function, the 'min-max' normalisation method was used. The Min-Max normalisation formula is given by:

$$I' = \frac{I - I_{\min}}{I_{\max} - I_{\min}} \times (new\_max - new\_min) + new\_min$$

where:

- I is the original pixel intensity,
- $I_{\min}$ is the minimum intensity in the original range,
- $I_{\max}$ is the maximum intensity in the original range,

- *new_min* is the desired new minimum intensity,
- *new_max* is the desired new maximum intensity.

Since we are mapping from $[0, 1]$ to $[0, 255]$, the formula simplifies to:

$$I' = I \times 255$$

4.3.3 Feature Extraction

After the image was pre-processed feature vectors could then be extracted. For each image in the train, validation, and test batches a feature vector was extracted. I experimented with two different choices of feature vector, each focusing on a different characteristic of the image. Local Binary Pattern (LBP) extracts texture information from the image, while Histogram of Oriented Gradients (HOG) extracts edge information by computing gradient orientation distributions (Pietikäinen 2010; Dalal and Triggs 2005)

Local Binary Pattern: To construct the LBP feature vector, rather than computing a single LBP representation for the entire image, I divided the image into 50 × 50 blocks (a hyperparameter selected through tuning). LBP was calculated separately for each block, and the results were stored in a histogram. Since the original image was 150 x 150 and was divided into 50 x 50 blocks, the LBP was calculated for 3 x 3 = 9 sections of the image.

The hyperparameters for the LBP function are the:

- **Radius** which determines the size of the circular region around the central pixel where neighbouring pixels are sampled. For this I used 2.
- **Number of points** which defines the number of sampling points used to compare pixel intensities around the central point. For this I used 8 x radius = 16.
- **Method** determines how the LBP is encoded, this can introduce rotation invariance, reduce dimensionality, and more. For this I used 'uniform' which groups the LBP patterns into fewer bins based on uniformity, this reduces the dimensionality.
- **Block size** as described above, determines how many sections of the image the LBP is calculated for. For this I used 50.

These values of hyperparameters were obtained via hyperparameter tuning.

The LBP was then computed for each block in the image sequentially. When LBP is calculated we compare a pixel to its neighbouring pixels, if a neighbouring value is greater than or equal to the central pixel then it is assigned 1; otherwise it is assigned 0 - this creates a sequence of 0s and 1s for each pixel (a binary pattern).

Uniform patterns are those with at most 2 transitions from 0-to-1 or 1-to-0, these are important since they represent important texture information like edges, corners, flat areas, etc. Non-uniform patterns are any patterns which are not uniform, these are often noise or irregular patterns in the image and are less common.

Each histogram had 2 + 'number of points' = 2 + 16 = 18 bins. The first 16 bins were for uniform binary patterns, the 17th bin was for non-uniform patterns, and the 18th bin was to prevent edge-issues. After the histogram for a block was calculated using the Scikit-Learn 'local_binary_pattern' function, it was normalised to create a probability distribution, this represents the proportion of different texture patterns in that block.

Finally, all 9 LBP histograms were concatenated into a final LBP feature vector. Since there are just 18 bins per histogram, 1 histogram per block, and 9 blocks, the size of the feature vector is 18 x 9 = 162. Therefore, the LBP feature extraction effectively captures the information of a 150 x 150 image (22,500 pixel values) in a vector of size 162.

Histogram of Oriented Gradients: Constructing the HOG feature vector was comparatively more straightforward, and comprised of computing the HOG for the entire image at once using the Scikit-Learn 'hog' function.

The hyperparameters for the HOG function are the:

- **Orientations** which specifies the number of orientation bins to describe the gradient directions. For this I chose 8, which meant that each bin covered $\frac{180}{8} = 22.5$ degrees.
- **Pixels per cell** which determines the size of the cells which the image is divided into. For this I chose 18 x 18, which meant that the 150 x 150 image was divided into cells of size 18 x 18.
- **Cells per block** which determines the size of each block in terms of cells, each block is used to normalise the histogram within it. For this I chose 2 x 2, which meant that each block was comprised of 4 cells

These values of hyperparameters were obtained via hyperparameter tuning.

The HOG was then computed by passing the image along with the hyperparameters into the Scikit-Learn 'hog' function. The function works by computing the gradient magnitude and orientation of each pixel in the image. Then, the image is divided into cells of size 16 x 16 and for each cell a histogram of the gradient orientations is created, the magnitude of the gradient is used to weigh a pixels contribution to the histogram bin for its orientation. These cells are then grouped into blocks and the histograms within each block are normalised. Finally, the normalised histograms from all the blocks are concatenated into a feature vector.

The feature vector from HOG is generally much larger than that of LBP. Using the specified hyperparameters the HOG feature vector was of size 1,568, calculated as:

$$\begin{aligned}
\text{Size} &= \text{Number of Blocks} \times \text{Features per Block} \\
&= \left(\left\lfloor \frac{\text{Image Width}}{\text{Cell Width}} \right\rfloor - 1 \right) \times \left(\left\lfloor \frac{\text{Image Height}}{\text{Cell Height}} \right\rfloor - 1 \right) \times \text{Cells per Block} \times \text{Orientations} \\
&= \left(\left\lfloor \frac{150}{18} \right\rfloor - 1 \right) \times \left(\left\lfloor \frac{150}{18} \right\rfloor - 1 \right) \times 2 \times 2 \times 8 \\
&= (8-1) \times (8-1) \times 2 \times 2 \times 8 \\
&= 7 \times 7 \times 2 \times 2 \times 8 \\
&= 49 \times 32 \\
&= 1,568
\end{aligned}$$

While this is much smaller than the number of pixels in the original image, it is still considerable, especially when working with large batches of images.

Each of the two feature extraction methods would be used to fit the SVM individually, and the performance of each was evaluated to determine the best feature for the task.

4.3.4 Training and Tuning

After extracting the feature vectors from the images using either LBP or HOG, the next step was to train a SVM to make classifications based on these features.

I used the Support Vector Classifier (SVC) from Scikit-Learn. The Radial Basis Function (RBF) was chosen as the kernel function for the SVM. RBF is a common choice for non-linear classification tasks and works by enabling a mapping of the feature vectors into a higher-dimensional feature space. Although, the data is never actually transformed into a higher dimension, we can calculate the relationship between data points as if they were in a higher dimensions, and from this we can determine a hyperplane which separates the data - this is called the kernel trick.

The remaining hyperparameters of the SVM are C and γ. C is the regularisation parameter which controls the trade-off between maximising the margin and minimising the error, while γ determines the sensitivity a single point has on the decision boundary. To find optimal hyperparameter values for the task, I performed hyperparameter tuning using randomised search cross-validation. Randomised search was chosen as it allows for fast searching of parameters over a large parameter space.

To perform a randomised search of the parameter space, I defined a parameter distribution for C and γ:

- C was searched over a logarithmic distribution of 100 values from 10^{-3} to 10^{3}.
- γ was searched over a logarithmic distribution of 100 values from 10^{-4} to 10

These distributions were achieved using the NumPy 'logspace' function. This allowed the randomised search to explore a wide range of potential values for each hyperparameter.

Stratified k-fold cross-validation with k=5 was used to fit the model. This splits the training data into 5 subsets while ensuring that each fold maintains an even split between class 0 and class 1 data points. This helps to ensure that each fold was representative of the entire training dataset.

The hyperparameter tuning was the performed using the Scikit-Learn 'RandomisedSearchCV' function for 25 iterations. This samples 25 random combinations from the parameter distributions and fits the model using the chosen parameters to 4 folds of the training data, while the remaining fold was used to evaluate performance. Since we used 5-fold cross validation, we fit 5 folds for each of the 25 candidate parameters resulting in a total of 125 fits to find the optimal parameters. The f1-score was used to determine which parameters produced the best fit. Once the optimal hyperparameters were found the model would be fit to the entire training dataset.

While SVM hyperparameter tuning was efficiently automated using parameter grids and cross-validation, optimising feature extraction hyperparameters proved more laborious. This involved a manual, iterative process of adjusting each parameter individually while keeping others constant, and observing the resulting impact on model performance. Given that each feature extraction method (LBP and HOG) had three tunable parameters, this process, though less exhaustive, allowed for the identification of effective hyperparameter combinations.

4.4 ResNet Approach

This approach involved using transfer learning with ResNet to perform binary and multi-class classification as well as age estimation using regression.

4.4.1 Data Pre-processing and Augmentation

In order to load data for training, a custom dataset was created which loads the image data efficiently on to the GPU and handles the labelling of the data.

Custom Dataset: This was done by creating a 'CustomDataset' object which extends the PyTorch 'Dataset' class, it takes as input a CSV file containing image file names and corresponding ages, the directory where the images are stored on the disk, and the transformations to be applied to images as they are loaded. This allowed for the same dataset object to be used for both APPA-REAL and UTKFace.

When the dataset is initialised the labels are determined according to the task:

- For binary classification data points were labelled 0 for ages less than 25 and 1 for ages 25 and above.
- For multi-class classification data points were labelled 0-3 according to the age ranges 1-18, 19-28, 29-40, and 40+, respectively. These bins were decided as they allow for an even distribution of data points across labels.

- For regression the real age data was used.

Data retrieval was performed through the 'item' method to access individual samples. This was implemented by taking as input the index to be retrieved, then the file path and label were found by searching for the item in the dataset at the given index and loading it.

Image loading was handled by the TorchVision 'decode_jpg' and 'read_file' functions. This is significantly more efficient than loading images using another method such as PIL or NumPy since it allows for GPU acceleration and produces a PyTorch tensor as output, which eliminates the overhead of converting to a tensor. The 'read_file' function returns a tensor containing the byte contents of a file, this coupled with 'decode_image' allows for a memory efficient loading of an image directly as a tensor.

The loaded image tensor is then transferred to the GPU using '.to('cuda')'.This creates a copy of the tensor in GPU memory, ensuring all future operations on this variable are performed on the GPU. This is essential in order to do efficient operations on tensors.

The image is decoded into a tensor of shape (3, height, width) with an unsigned 8-bit integer data type, where pixel values range from 0 to 255. To normalize the image to the range [0,1], each pixel value is divided by the highest pixel value in the image. For example, if the maximum pixel value is 255, a pixel with a value of 51 would be normalized as 51.0/255.0 = 0.2, while a pixel with a value of 255 would become 255.0/255.0 = 1.0, This method ensures that the normalized image always falls within the range [0,1], even if the original image lacks pixels with a value of 255.

The final step involves performing data augmentation, which helps minimize overfitting and enables training on smaller datasets. This was implemented during data retrieval by applying the following transformations, selected based on the work of Shorten and Khoshgoftaar (2019):

- **RandomHorizontalFlip()**: Randomly flips an image across the y-axis with probability 0.5.
- **RandomRotation()**: Randomly rotates the image between -20°and 20°. The angle parameter was set to 20.
- **Gaussian Blur**: Performs a Gaussian blur with a kernel size of 3 and standard deviation, σ, [0.1, 1.0].
- **RandomResizedCrop()**: Randomly crops a portion between 80% and 100% of the image, this was determined using the scale parameter set to (0.8, 1.0). The random crop is then resized to 224 x 224 by setting the size parameter accordingly.
- **Normalize()**: The image is normalised using mean = [0.485, 0.456, 0.406], and std = [0.229, 0.224, 0.225]. This specifies the mean and standard deviation for each colour channel respectively. The value of a channel is calculated as:

$$\text{output[channel]} = (\text{input[channel]} - \text{mean[channel]})/\text{std[channel]}$$

The choice of resizing to 224 x 224 and normalising using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] was not arbitrary. Since transfer learning with an ImageNet trained ResNet model was used, these image pre-processing steps allow for enhanced performance by ensuring the training data matches the original ImageNet training data. The ImageNet training data used the above normalisation, and the first convolutional layer of ResNet uses a 7 x 7 kernel, stride of 2, and padding of 3, which is optimised for images of shape 224 x 224.

This can be shown by calculating how the first convolutional layer produces features from an image of shape 224 x 224:

$$\text{Output Size} = \left\lfloor \frac{\text{Input Size} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} \right\rfloor + 1$$

$$= \lfloor \frac{224 - 7 + 2(3)}{2} \rfloor + 1$$
$$= 112$$

Thus, the output tensor shape after applying the convolution is:

$$(64, 112, 112)$$

Where 64 is the number of output channels (filters) and 112×112 is the new spatial resolution after downsampling.

Then a ReLU activation function is applied, followed by max pooling with kernel size 3 x 3, stride of 2, and padding of 1. Further reducing to (64, 56, 56) (He et al. 2016). Therefore, we can see that this structure is optimised to extract filters from images of size 224 x 224, as such I opted to resize the dataset to the same dimensions.

After data augmentation the normalised and augmented image tensor is returned along with its label and can be used for training.

Batch Loading and Stratification: A PyTorch 'Dataloader' was used to obtain batches of data from the custom dataset above. A batch size of 128 was used, this was determined through hyperparameter tuning.

The training batches used stratified sampling via the PyTorch 'WeightedRandomSampler' with sample weights calculated as $\frac{1}{\text{class count}}$. For UTKFace the training split contains 2,000 class 0 images and 2,971 class 1 images. Resulting in a weight of 0.0005 and 0.000337 for classes 0 and 1 respectively - this means that the rarer class has a higher weight, and therefore is more likely to be sampled. These weights are then passed into the 'WeightedRandomSampler' and the training batch loader is created using batch_size=128, the training dataset, and the sampler. The batch loaders for validation and testing use a batch size of 128, and are not stratified.

This stratified sampling approach was used for binary and multi-class classification. While regression does not require stratified sampling as there are no classes to balance.

4.4.2 Transfer Learning

PyTorch provides implementations of the ResNet architecture from Deep Residual Learning for Image Recognition (He et al. 2016), this includes ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. In addition to the architecture, PyTorch provides pre-trained weights which follow closely the results of the original paper, trained on the ImageNet dataset.

Although ImageNet is not a face dataset, transfer learning enables the weights trained on ImageNet to be applied to face images to perform facial age estimation. Two different ResNet architectures were used in this project:

- ResNet-18: The shallowest ResNet architecture with 18 layers and ~11.1 million learnable parameters
- ResNet-50: A deeper ResNet architecture with 50 layers and ~23.5 million parameters.

This was to enable comparison between model architectures and assess the benefit a deeper network has in this task and with these datasets, while balancing computational costs.

The chosen architecture and corresponding weights were loaded, then the final fully connected layer was modified and some weights would be unfrozen. The final layer of ResNet-18 is a fully connected layer mapping from 512 input features to 1,000 output neurons, while the final layer of ResNet-50 is a fully connected layer mapping from 2048 features to 1,000 outputs. This fully connected layer would be modified depending on the task.

Two approaches of transfer learning were experimented with to investigate performance. The first was to freeze all model weights aside from the weights in the final fully connected layer. With this approach ResNet-18 had 513 learnable parameters, 512 weights (one per input feature) and 1 bias term. Similarly, ResNet-50 had 2049 learnable parameters.

The second approach involved training on the parameters of the fully connected layer and the fourth layer. For ResNet-18 this results in $513 + 8,393,728 = 8,394,241$ learnable parameters, while for ResNet-50 this gives $2049 + 14,964,736 = 14,966,785$ learnable parameters.

4.4.3 Binary Age Classification

The first task which ResNet was applied to was binary classification. As described prior, this involves assigning images a label of 0 or 1, for under 25 or 25 and older respectively. The task was then to take an unseen image and determine which class it belongs to. In order to adjust the network to train for this objective the final fully connected layer was modified as described in 4.4.2, by mapping from either 512 or 2048 inputs to 1 output.

Activation Function: By default, the outputs of the network are raw logits, these are simply the scores produced by the final layer of the network. They carry useful information but cannot be directly interpreted in terms of likelihood. As stated by Goodfellow et al. (2016), the network needs only to predict $P(y = 1|\, \mathbf{x})$, that is the probability of a positive class given $\mathbf{x}$. For this to be a valid probability, it must lie in the range [0, 1]. An output logit of the network is calculated as a linear combination of the weight and bias:

$$z = w^T x + b \tag{4.2}$$

A logit defines a distribution over the binary values, and represents the log-odds of the positive class:

$$z = \text{logit}(p) = \ln(\frac{p}{1-p}) \tag{4.3}$$

This is a distribution which maps probability values, p, in the range (0, 1) into real numbers in the range $(-\infty, \infty)$. As such, the raw output logits of the network are in the range $(-\infty, \infty)$.

For binary classification, we want to express the networks raw output as a confidence score. To do this we can apply an activation function which maps the logits into a range [0, 1]. For binary classification the sigmoid activation function is often used. The sigmoid activation function maps values from the range $(-\infty, \infty)$ on to the range (0, 1) asymptotically. The mapping is obtained using the following equation:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.4}$$

Which is the inverse of the logit function. Where z is the output logit. Larger values of z cause the function to approach 1, while smaller values cause it to approach zero.

Using this allows us to express the output of the network as the probability that the sample belongs to the positive class. For example, a raw logit value $z >> 0$ results in a final output (after sigmoid activation) closer to 1, while a logit value $z << 0$ corresponds to an output value closer to 0. Generally, the decision boundary would occur at $z = 0$ which maps to 0.5 after sigmoid activation - which is ideal for binary classification between 0 and 1.

Loss Function: Rather than explicitly add a sigmoid layer after the fully connected layer, I opted to use it in conjunction with the loss function. The PyTorch 'BCEWithLogitLoss' combines the sigmoid activation function and the Binary Cross Entropy (BCE) loss function into one.

Hyperparameter Version	v1	v2	v3
Batch Size	64	32	128
Learning Rate	0.001	0.0001	0.0001
Weight Decay	N/A	0.01	0.01

Table 4.1: *Hyperparameter configurations used for model training.*

This is more numerically stable than using both in sequence as stated by the PyTorch (2024) documentation.

This meant that the output of the network was still raw logits, however the loss would be calculated by first applying the sigmoid activation function, then computing the Binary Cross Entropy on the probability/confidence scores. The BCE loss is computed by comparing the ground truth label $y \in \{0, 1\}$ to the predicted value $\hat{y}$, where $0 \leq \hat{y} \leq 1$. Using these, the loss is computed as:

$$\text{BCE} = L(\hat{y}, y) = -\left(y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})\right) \tag{4.5}$$

Since, BCEWithLogitLoss was used in place of sequential sigmoid activation and loss calculation, the full loss function would be calculated as:

$$\text{BCEWithLogitLoss} = L(z, y) = -\left(y \ln \sigma(z) + (1 - y) \ln(1 - \sigma(z))\right) \tag{4.6}$$

Where σ represents the sigmoid function, and is applied to z - the logit outputs, to obtain probabilities, $\hat{y}$.

During training, the loss is calculated for an entire batch of 128 images, and is averaged using the mean to determine a single loss value for each batch. For example, training with 4,971 images (as in UTKFace) with a batch size of 128 results in $\lceil \frac{4971}{128} \rceil = 39$ loss values, each of which is the mean of all 128 BCE loss calculations.

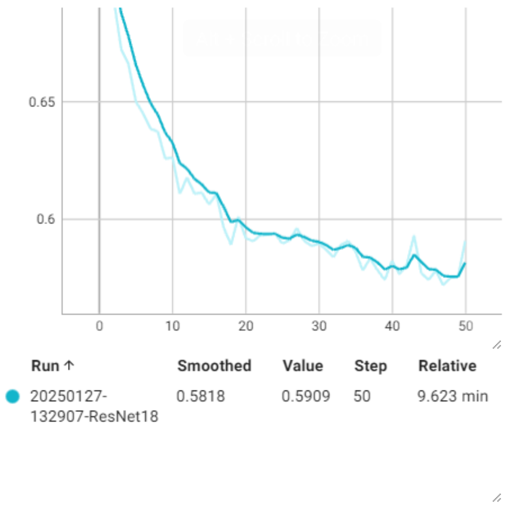
Training: Now that the network was configured for the task, training could begin. Multiple models were trained for this task. Two ResNet-18 models, one with only the final fully connected layer weights unfrozen, and one with the fully connected layer and the fourth ResNet layer unfrozen. Similarly, two ResNet-50 models were trained, one with the final fully connected layer weights unfrozen, and one with the fully connected layer and the fourth ResNet layer unfrozen.

The Adam optimiser was used during training to update the weights, different combinations of hyperparameters were used to train the models and are shown in Table 4.1.

Models were trained using these different versions. The training and validation loss was monitored during training to observe progress and identify any issues. For example, the training log of a ResNet-18 model trained on the fully connected layer using hyperparameter version 2 is shown in Figure 4.15 which shows the training loss over the epochs, Figure 4.16 shows the validation loss and accuracy for the same model.

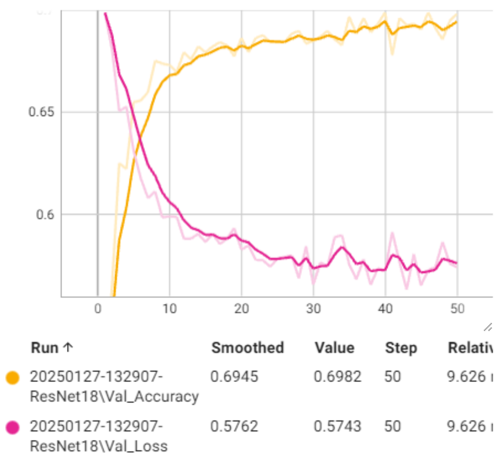
The general shape of these graphs is similar across all models trained only on the fully connected layer parameters, for both ResNet-18 and ResNet-50. This is an ideal graph of the training process, with training and validation loss both steadily decreasing over time, while the validation accuracy increases.

When training on the fully connected layer and the fourth ResNet layer using hyperparameter version 2, a less ideal pattern is observed. The training log of a ResNet-50 model trained on the fully connected layer and the fourth ResNet layer is shown in Figure 4.17 which shows the training loss over the epochs, Figure 4.17 shows the validation loss and accuracy for the same model.



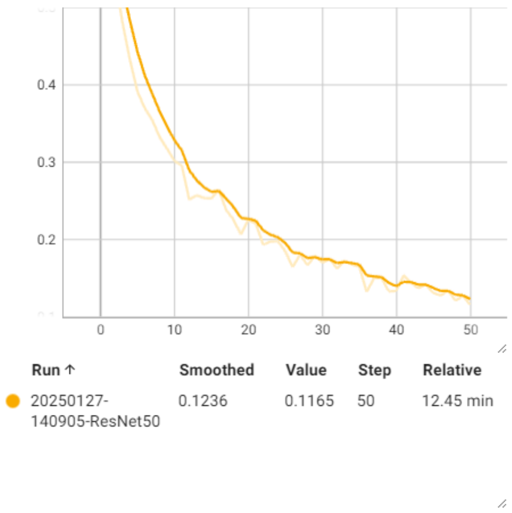
Run ↑	Smoothed	Value	Step	Relative
● 20250127-132907-ResNet18	0.5818	0.5909	50	9.623 min

Figure 4.15: A graph showing the training loss on the y-axis against the training epoch on the x-axis.



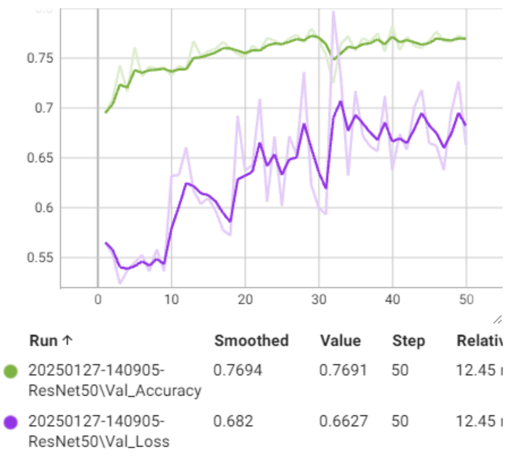
Run ↑	Smoothed	Value	Step	Relativ
● 20250127-132907-ResNet18\Val_Accuracy	0.6945	0.6982	50	9.626
● 20250127-132907-ResNet18\Val_Loss	0.5762	0.5743	50	9.626

Figure 4.16: A graph showing the validation loss (in pink) and accuracy (in yellow) improving over the epochs. The validation loss reaches 0.57 and the validation accuracy reaches 69%.



Run ↑	Smoothed	Value	Step	Relative
● 20250127-140905-ResNet50	0.1236	0.1165	50	12.45 min

Figure 4.17: A graph showing the training loss on the y-axis against the training epoch on the x-axis.



Run ↑	Smoothed	Value	Step	Relativ
● 20250127-140905-ResNet50\Val_Accuracy	0.7694	0.7691	50	12.45
● 20250127-140905-ResNet50\Val_Loss	0.682	0.6627	50	12.45

Figure 4.18: A graph showing the validation loss (in purple) and accuracy (in green) over the epochs. The validation accuracy climbs to around 76% and then plateaus. The validation loss climbs from around 0.55 to 0.62 while fluctuating greatly.

We can see that the validation accuracy climbs slightly then plateaus, while the validation loss climbs erratically. This pattern continues for all models with a higher number of parameters. Intuitively, I would expect this to be caused by overfitting. However, after investigation I do not believe that to be the case since the validation accuracy does not decrease, and in fact continues to increase slightly. I also propose that this is not caused by a class imbalance, since other metrics such as f1-score remain strong for these models, and steps such as stratified loading have been taken to reduce class imbalance.

After inspection of the model outputs, I suspect that this is caused by overly confident model predictions. In order to confirm this, I logged the model output (after sigmoid activation) on the validation data for each epoch. The early epochs show the average output for both classes is close to 0.5, causing a generally high loss. As training progresses, correctly classified samples tend towards extreme values (0 or 1), while misclassified samples tend towards the incorrect extreme, increasing loss due to the logarithmic nature of BCE.

We can demonstrate how this causes an increase in loss using Equation 4.5:

Early Training Epochs

For early epochs, let the predicted outputs be:

$$\begin{aligned} \hat{y} &\approx 0.6 \quad \text{for} \quad y = 1, \\ \hat{y} &\approx 0.4 \quad \text{for} \quad y = 0. \end{aligned}$$

Suppose we evaluate loss on 10 samples consisting of 6 true positives (TP), 1 false negative (FN), 1 true negative (TN), and 2 false positives (FP):

$$\begin{aligned} L_{TP} &= -\ln(0.6) \approx 0.511, \\ L_{FN} &= -\ln(0.4) \approx 0.916, \\ L_{TN} &= -\ln(0.6) \approx 0.511, \\ L_{FP} &= -\ln(0.4) \approx 0.916. \end{aligned}$$

The total loss is:

$$6(0.511) + 0.916 + 0.511 + 2(0.916) = 6.326, \quad \text{with an average loss of } 0.632.$$

Later Training Epochs

As the model becomes more confident, let:

$$\begin{aligned} \hat{y} &\approx 0.95 \quad \text{for} \quad y = 1, \\ \hat{y} &\approx 0.05 \quad \text{for} \quad y = 0. \end{aligned}$$

Using the same class distribution:

$$\begin{aligned} L_{TP} &= -\ln(0.95) \approx 0.051, \\ L_{FN} &= -\ln(0.05) \approx 2.996, \\ L_{TN} &= -\ln(0.95) \approx 0.051, \\ L_{FP} &= -\ln(0.05) \approx 2.996. \end{aligned}$$

The total loss is:

$$6(0.051) + 2.996 + 0.051 + 2(2.996) = 9.345, \quad \text{with an average loss of } 0.934.$$

Thus, as the model becomes increasingly confident, misclassified samples (false positives and false negatives) incur much higher penalties, leading to an overall increase in loss, despite a steady or increasing accuracy / f1-score.

Although the validation loss increases during training, this does not indicate overfitting or performance degradation. Instead, it reflects the model's growing confidence in its predictions. Because BCE loss is logarithmic, incorrect but highly confident predictions receive a much higher penalty, leading to a rise in overall loss. However, since accuracy and F1-score remain stable or improve, this does not negatively impact performance.

4.4.4 Multi-class Age Classification

The second task which ResNet was applied to was multi-class classification. This involved assigning a label of 0, 1, 2, or 3 following the age ranges 1-18, 19-28, 29-40, and 40+. The task was then to take an unseen image and assign it to one of these classes. We first start by modifying the final fully connected layer to map from either 512 or 2048 inputs to 4 outputs (one for each class), depending on architecture.

Activation Function: Again, we need to interpret the raw logit outputs of the network in a way which is useful for the task. Unlike with binary classification, where we want the network to predict the probability of a single positive class given x, we now want the network to produce a vector $\hat{\boldsymbol{y}}$ with $y_i = P(y = i|x)$, where $0 \leq y_i \leq 1$ and $\sum_{i=1}^{n} y_i = 1$, for n classes and for all i. That is, we want to produce a valid probability distribution across the classes (Goodfellow et al. 2016).

The output of the network is comprised of 4 raw logit values (one for each class). We can apply the softmax function, shown in equation 4.7 to transform the raw logits into a valid probability distribution for each class.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{4.7}$$

Where z_i is the logit representing class i, and $\sum_{j=1}^{n} e^{z_j}$ is the sum of each exponentiated logit.

After applying softmax we have $\hat{\boldsymbol{y}} = (y_0, y_1, y_2, y_3)$, where y_i is the probability of class i for $i \in \{0, 1, 2, 3\}$.

Loss Function: Just like with sigmoid and BCE, PyTorch offers an optimised combination of softmax with Cross-Entropy Loss using the 'CrossEntropyLoss' class. However, instead of explicitly applying softmax, it uses the log-softmax function internally to improve numerical stability.

The cross-entropy loss for a single datapoint is given by:

$$\text{CrossEntropyLoss} = L(z, y) = -\ln P(y|x) = -\ln\left(\frac{e^{z_y}}{\sum_{j=1}^{n} e^{z_j}}\right) \tag{4.8}$$

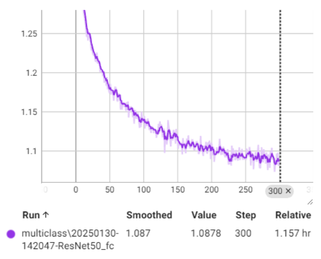
Where:

- $z = (z_1, z_2, ..., z_n)$ is the vector of raw logits.
- y is the true class label.
- z_y is the logit corresponding to the true class, y.
- n is the total number of classes, which is 4.

Training: Now that the network was configured for the task, training could begin. Using knowledge from training for binary classification I used the following hyperparameters with the Adam optimiser:

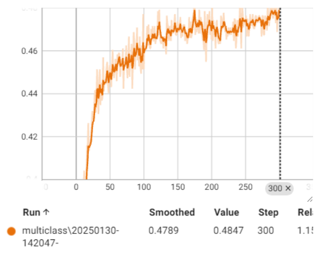
- Batch Size: 128
- Learning Rate: 0.0001
- Weight Decay: 0.01

Training this network showed a similar trend to the binary classifier, lower parameter models produced a standard learning curve, while higher parameter models demonstrated an increasing validation loss. Figures 4.19, 4.20, and 4.21 show the training loss as well as the validation accuracy and loss for a ResNet-50 model trained only on the fully connected layer.



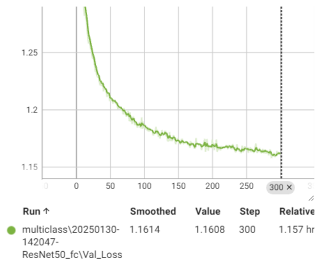
Run ↑	Smoothed	Value	Step	Relative
● multiclass\20250130-142047-ResNet50_fc	1.087	1.0878	300	1.157 hr

Figure 4.19: A graph showing a decreasing training loss over epochs



Run ↑	Smoothed	Value	Step	Rel
● multiclass\20250130-142047-	0.4789	0.4847	300	1.1!

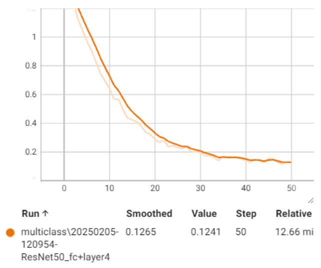
Figure 4.20: A graph showing an increasing validation accuracy over epochs



Run ↑	Smoothed	Value	Step	Relative
● multiclass\20250130-142047-ResNet50_fc\Val_Loss	1.1614	1.1608	300	1.157 hr

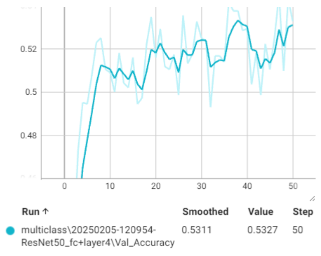
Figure 4.21: A graph showing a decreasing validation loss over epochs

Similar to the binary classification model, Figures 4.22, 4.23, and 4.24 show the training of a ResNet-50 model trained on the fully connected layer and the fourth ResNet layer, this shows an increasing validation loss while accuracy continues to improve.



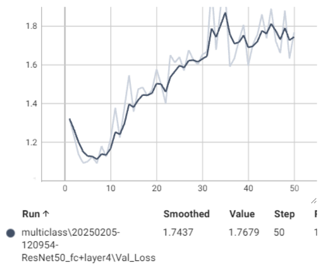
Run ↑	Smoothed	Value	Step	Relative
● multiclass\20250205-120954-ResNet50_fc+layer4	0.1265	0.1241	50	12.66 mi

Figure 4.22: A graph showing a decreasing training loss over epochs



Run ↑	Smoothed	Value	Step
● multiclass\20250205-120954-ResNet50_fc+layer4\Val_Accuracy	0.5311	0.5327	50

Figure 4.23: A graph showing an increasing validation accuracy over epochs



Run ↑	Smoothed	Value	Step	I
● multiclass\20250205-120954-ResNet50_fc+layer4\Val_Loss	1.7437	1.7679	50	1

Figure 4.24: A graph showing an initial drop in validation loss followed by a steady climb

I propose that this is again caused by overconfident predictions being penalised harshly just like with 'BCEWithLogitLoss'. Since 'CrossEntropyLoss' also uses the negative log likelihood, a confidently incorrect prediction would incur a large penalty, e.g. if the ground truth class is 0, but class 2 was predicted with high confidence.

4.4.5 Regression Age Estimation

The final task which ResNet was applied to was regression. This involved predicting the exact age rather than an age range. The task was then to take an unseen image and predict the age. The process of this is slightly different to that of binary or multi-class classification, since there is no need to interpret the model output as probability scores. Instead, we can interpret the model output directly as the estimated age, and train to minimise the distance between the predicted and actual age.

We first modify the final fully connected layer to map from either 512 or 2048 inputs to 1 output (the estimated age), depending on architecture. Since the model outputs are already real numbers in the range $(-\infty, \infty)$ there is no need to apply an activation function as this would only constrain the model outputs. Note that the network permits negative outputs, this is not inherently an issue, as the training process will update weights in the direction away from such values, as all ages are positive values. From inspection of the training process, negative outputs disappear after the first training epoch.

Loss Function: The output of a network for a batch of N inputs is $\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_N)$, where each $\hat{y}_i$ is the predicted output for input i. The expected output for the same batch is represented by $\boldsymbol{y} = (y_1, y_2, ..., y_N)$, where each y_i is the expected output for input i.

A common function for computing the loss between the predicted and expected outputs is the Mean Squared Error (MSE) function. The MSE function is defined as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.9}$$

Which simply computes the squared distance between predicted and expected output pairs, then sums these and divides by N, the number of samples, to compute the mean.

MSE computes the squared difference between predicted and actual values, which disproportionately penalizes larger errors compared to smaller ones. This means that predictions far from the true value contribute significantly more to the overall loss than those closer to it. In contrast, Mean Absolute Error (MAE) measures the absolute difference, treating all errors linearly. While MAE provides a more balanced penalty across all errors, MSE is often preferred for its smooth gradient properties, making it more suitable for gradient-based optimization.

Training: Training for regression follows closely the process of training for binary and multi-class classification, with the key difference is how accuracy is defined. Since the output is a continuous (floating point) value, it is highly unlikely for it to match an expected output exactly. Given the inherent difficulty of regression tasks, a traditional accuracy metric would not be appropriate. Instead, I opted to use tolerance-based accuracy, considering a prediction correct if it within ±5 years of the expected value, i.e., if $|y - \hat{y}| \leq 5$.

Training used the same hyperparameters as outlined prior, and showed an ideal trend for both the higher-parameter and lower-parameter models. As seen in Figures 4.25, 4.26, and 4.27, which show the training loss, validation accuracy, and validation loss respectively for a ResNet-50 model trained on the fully connected layer and ResNet layer 4.

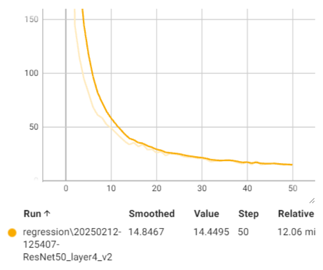


Figure 4.25: A graph showing a decreasing training loss over epochs

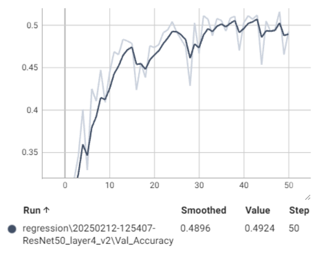


Figure 4.26: A graph showing an increasing validation accuracy over epochs

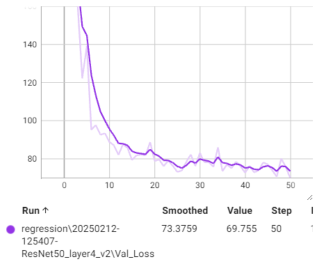


Figure 4.27: A graph showing a decreasing validation loss over epochs

These graphs indicate steady training over the epochs, and similar trend is observed for all regression models trained.

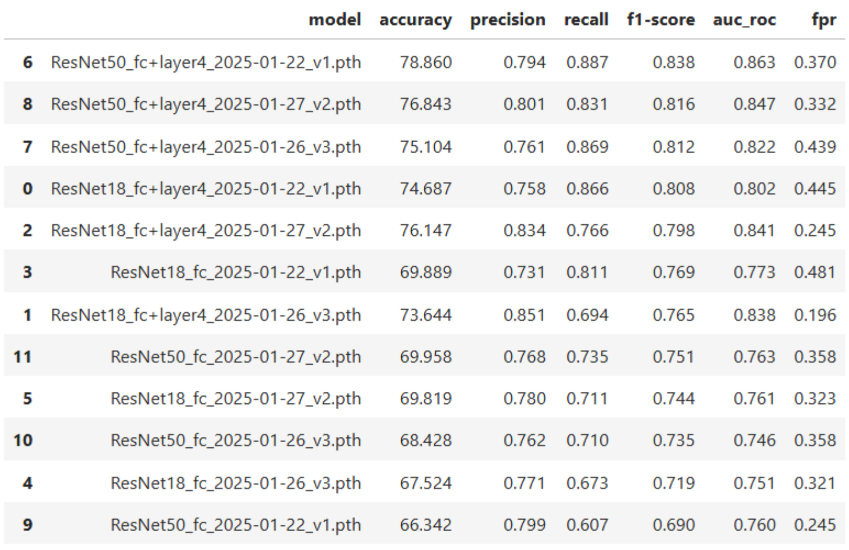
5 | Evaluation

Here I present the final performance for the best models from each of the approaches outlined prior. The results will then be discussed and compared in terms of bias and generalisability.

5.0.1 Model Selection

For each task, one model per dataset is selected for evaluation based on its validation performance. For example, two final models are chosen for SVM classification—one trained on APPA-REAL and the other on UTKFace. Likewise, two models are selected for ResNet binary classification, and so on. This resulted in a total of 8 models to be compared and evaluated.

To demonstrate the selection process, the APPA-REAL validation performance for APPA-REAL trained ResNet binary classification models are shown in Figure 5.1.



	model	accuracy	precision	recall	f1-score	auc_roc	fpr
6	ResNet50_fc+layer4_2025-01-22_v1.pth	78.860	0.794	0.887	0.838	0.863	0.370
8	ResNet50_fc+layer4_2025-01-27_v2.pth	76.843	0.801	0.831	0.816	0.847	0.332
7	ResNet50_fc+layer4_2025-01-26_v3.pth	75.104	0.761	0.869	0.812	0.822	0.439
0	ResNet18_fc+layer4_2025-01-22_v1.pth	74.687	0.758	0.866	0.808	0.802	0.445
2	ResNet18_fc+layer4_2025-01-27_v2.pth	76.147	0.834	0.766	0.798	0.841	0.245
3	ResNet18_fc_2025-01-22_v1.pth	69.889	0.731	0.811	0.769	0.773	0.481
1	ResNet18_fc+layer4_2025-01-26_v3.pth	73.644	0.851	0.694	0.765	0.838	0.196
11	ResNet50_fc_2025-01-27_v2.pth	69.958	0.768	0.735	0.751	0.763	0.358
5	ResNet18_fc_2025-01-27_v2.pth	69.819	0.780	0.711	0.744	0.761	0.323
10	ResNet50_fc_2025-01-26_v3.pth	68.428	0.762	0.710	0.735	0.746	0.358
4	ResNet18_fc_2025-01-26_v3.pth	67.524	0.771	0.673	0.719	0.751	0.321
9	ResNet50_fc_2025-01-22_v1.pth	66.342	0.799	0.607	0.690	0.760	0.245

Figure 5.1: *A table of APPA-REAL validation metrics for 12 ResNet binary classification models, sorted by f1-score. The best performing model has an f1-score of 0.838, the worst has an f1-score of 0.690.*

We see a general trend of models with more learnable parameters achieving better performance. This is to be expected and reinforces the choice of ResNet as an ideal model for this task.

The chosen model was the one with the highest f1-score. Given the task of age estimation, it may be that precision is more important in order to minimise people under 25 being identified as 25 and above. However, for consistency the model with the highest f1-score was selected for all tasks. As such, the best APPA-REAL trained model for binary classification was identified as 'ResNet50_fc+layer4_v1' which was a ResNet-50 model trained on the fully connected layer and ResNet layer 4 using hyperparameter versions 1 (as described in implementation).

This selection process was repeated for each of the tasks, to produce a total of 8 final models. Note that for regression models the model with the lowest MSE was selected. Results for multi-class and regression models are presented in Appendix A.1

The following eight models were selected:

- **SVM Binary Classification:**
  - LBP feature SVM trained on APPA-REAL dataset.
  - LBP feature SVM trained on UTKFace dataset.
- **ResNet Binary Classification:**
  - `ResNet50_fc+layer4_v1` trained on APPA-REAL dataset.
  - `ResNet50_fc+layer4_v3` trained on UTKFace dataset.
- **ResNet Multi-class Classification:**
  - `ResNet50_fc+layer4_v1` trained on APPA-REAL dataset.
  - `ResNet50_fc+layer4_v1` trained on UTKFace dataset.
- **ResNet Regression:**
  - `ResNet18_layer4_v2` trained on APPA-REAL dataset.
  - `ResNet50_fc+layer4_v2` trained on UTKFace dataset.

5.0.2 Test Performance and Generalisability

Each of these model can be compared across datasets to understand generalisability.

Table 5.1: *Performance of SVM and ResNet Binary Classification and Multi-class Models Across Datasets. The models shown are those listed above, and are evaluated on the test split of the 'Test Dataset'.*

Model Type	Training Dataset	Test Dataset	Accuracy (%)	F1-score
SVM Classification	APPA-REAL	APPA-REAL	61.5	0.62
	APPA-REAL	UTKFace	58.6	0.41
	UTKFace	APPA-REAL	51.9	0.58
	UTKFace	UTKFace	75.9	0.75
ResNet Binary Classification	APPA-REAL	APPA-REAL	81.0	0.87
	APPA-REAL	UTKFace	83.2	0.85
	UTKFace	APPA-REAL	72.9	0.81
	UTKFace	UTKFace	88.1	0.90
ResNet Multi-Class Classification	APPA-REAL	APPA-REAL	52.8	0.54
	APPA-REAL	UTKFace	60.9	0.59
	UTKFace	APPA-REAL	41.4	0.33
	UTKFace	UTKFace	72.5	0.65

Table 5.2: *Performance of ResNet Regression Models Across Datasets*

Model Type	Training Dataset	Test Dataset	MSE	RMSE	Mean	Std.
ResNet Regression	APPA-REAL	APPA-REAL	172.0	13.1	3.9	12.7
	APPA-REAL	UTKFace	148.5	12.2	-2.6	11.9
	UTKFace	APPA-REAL	197.2	14.0	1.5	14.1
	UTKFace	UTKFace	63.0	7.9	-0.3	7.9

We can see that both the APPA-REAL and UTKFace trained SVM models show low generalisability outside of their native dataset. Potentially indicating that the models did not learn general features, but instead more dataset specific features. Note however, that worsened performance here may also be due to lack of colour information due to feature extraction.

The ResNet binary classification task showed strong generalisability from the APPA-REAL trained model to the UTKFace dataset - improving accuracy by 2.2% and maintaining a similar f1-score. However, while the UTKFace trained model showed very strong performance on UTKFace test data, when tested on the APPA-REAL dataset accuracy dropped by 15.2% and f1-score dropped by 0.09 - Indicating poor generalisability.

Looking at the multi-class performance, the accuracies as a whole are considerably lower, this is not surprising as with 4 classes the baseline to beat is 25% accuracy rather than the 50% of binary classification. In terms of generalisability we again see the APPA-REAL trained model perform well across both datasets, with slightly better performance on UTKFace. Alternatively, while the UTKFace trained model has very strong performance on UTKFace it generalised poorly to APPA-REAL, with an accuracy decrease of 31.1% and f1-score decrease of 0.32. Note that macro f1-score was used for multi-class evaluation.

The same trend is observed again with the regression models. The APPA-REAL trained model sees a slight improvement when tested on UTKFace, while the UTKFace trained model shows strong UKTFace test performance, and significantly worse APPA-REAL test performance.

The APPA-REAL trained models consistently demonstrate better cross-dataset performance, suggesting that this dataset may capture more robust and transferable features. This could be due to factors such as more diverse samples and more challenging image characteristics. Additionally, we can see that ResNet performs considerably stronger in binary classification than SVM, as is to be expected.

5.0.3 Model Bias

Another important consideration when evaluating model performance is bias. This is especially critical when performing facial analysis, as underlying bias such as an under-representation of races or genders, may result in a model which sees worse performance for select groups of people. As mentioned before, APPA-REAL has no race or gender metadata to evaluate on, to alleviate this each model was evaluated on UTKFace again, but with a focus on per-race and per-gender performance.

Table 5.3: *Performance of Binary Classification ResNet Models by Gender and Race Evaluated on UTKFace Test Split.*

Model	Group	Accuracy (%)	F1-Score	FPR
UTKFace Trained	Overall	88.1	0.90	0.16
APPA-REAL Trained	Overall	83.1	0.85	0.14
UTKFace Trained	Female	84.5	0.87	0.19
	Male	92.9	0.94	0.10
APPA-REAL Trained	Female	76.5	0.78	0.17
	Male	91.9	0.94	0.11
UTKFace Trained	Asian	81.8	0.78	0.10
	Black	91.2	0.93	0.17
	Indian	80.8	0.84	0.29
	Other	82.4	0.79	0.20
	White	92.3	0.94	0.11
APPA-REAL Trained	Asian	76.6	0.68	0.07
	Black	82.4	0.86	0.17
	Indian	73.1	0.76	0.26
	Other	80.4	0.75	0.18
	White	87.8	0.91	0.12

Table 5.3 shows the performance of both ResNet binary classification models by gender and race. Since the UTKFace trained model has better overall performance on UTKFace we cannot compare these values directly between models. We can however, directly compare the values per-model. For example, UTKFace trained model shows improved f1-score for males over females - indicating a potential bias towards males. Similarly, the APPA-REAL trained model shows significantly better performance for males over females. Indicating that both models are biased towards males.

Looking at the False Positive Rate (FPR) for each race, we can begin to investigate racial bias in the model predictions. Ideally, each race would have a FPR similar to the overall FPR, indicating that age class predictions are homogenous across races, however this is not the case. We can see for UTKFace that Indian and Other individuals show a FPR considerably higher than 0.16 - indicating that the model often predicts them to be older than they are, while Asian individuals show a lower FPR - indicating that they are predicted to be younger on average.

Another useful analysis is to investigate the per-race deviation of f1-score from the overall f1-score. To do this we can simply compute the difference between the overall f1-score and the f1-score of each race for both models. Plotting this as a heat-map allows for a clearer interpretation of the results, as seen in Figure 5.2

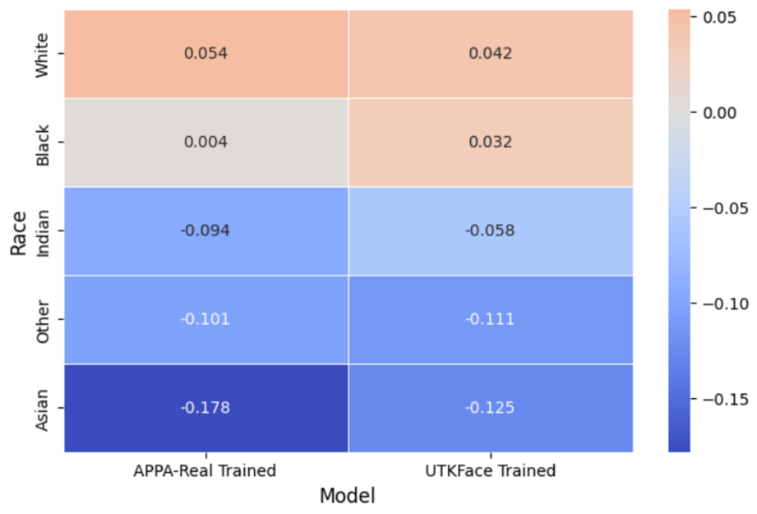


Figure 5.2: *A heat map of the UTKFace test metrics for both binary classification models showing the per-race f1-score deviation from the overall f1-score.*

The APPA-REAL model exhibits the highest positive deviation from its overall f1-score for White individuals, with performance gradually decreasing across Black, Indian, and Other racial groups, reaching the largest negative deviation for Asian individuals. UTKFace shows a similar trend. From this we can infer that the APPA-REAL model shows a bias towards white individuals - showing above average performance, while performing considerably bellow average for Asian individuals.

Since this shows the deviation from the overall f1-score of the model these values can be compared between models to see that the APPA-REAL trained model and the UTKFace trained model show a similar bias towards White individuals. Additionally, the APPA-REAL model shows a stronger bias against Asian individuals than the UTKFace trained model. Overall, both binary classification models show the same trend in bias however the impact is amplified noticeably in the APPA-REAL trained models.

When observing the bias in multi-class classification we observe a slightly different pattern in

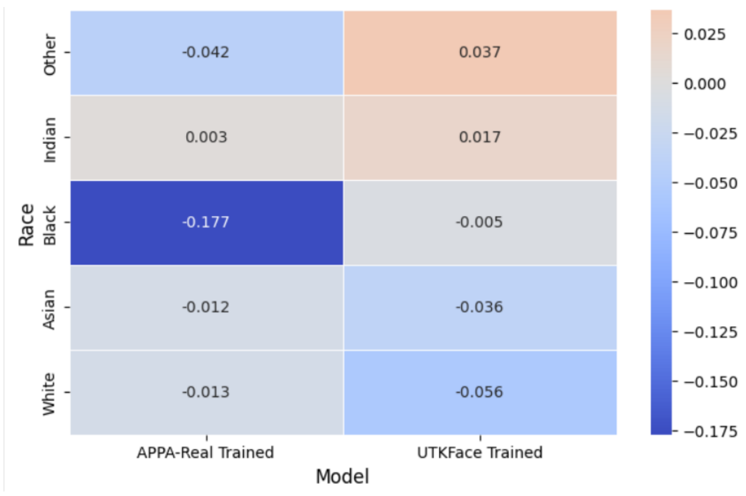


Figure 5.3: *UTKFace test metrics for both multi-class classification models showing the per-race macro f1-score deviation from the overall macro f1-score.*

racial bias as seen in Figure 5.3. Seeing significantly worse performance for Black individuals with the APPA-REAL trained model, while gender bias shows the same trend as for binary classification.

A similar pattern can be observed in both regression models (using RMSE in place of f1-score). This may indicate that while there is sufficient representation of Black individuals for binary classification, there may not be enough samples across age ranges to facilitate more precise age estimation tasks such as multi-class classification and regression. The complete results for these tasks can found in Appendix A.2.

Overall, analysis suggests that APPA-REAL trained models present more racial bias across tasks when compared to UTKFace trained models, with the exact bias observed being task specific. Additionally, both models across all tasks show a similar trend of gender bias towards male individuals. I believe this to be a result of under-representation of some racial groups in general and across certain age ranges.

5.0.4 Visual Results

Finally, observing actual model predictions for input images allows us to better understand the results.

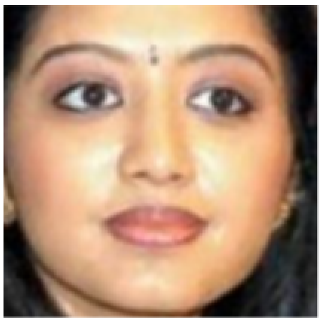


Figure 5.4: *UTKFace test image. Predicted class 0, was class 1*

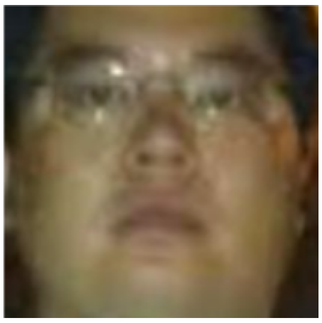


Figure 5.5: *UTKFace test image. Predicted class 0, was class 1*

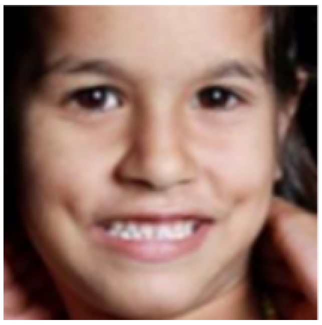


Figure 5.6: *UTKFace test image. Predicted class 1, was class 0*

Observing incorrect predictions from the APPA-REAL trained binary classification model, we can see Figures 5.4 and 5.6 show failed classifications for high-quality input images, this indicates a failure in being able to map these facial features to the correct class - potentially caused by bias in the training data. Alternatively, Figure 5.5 shows failure for a lower-quality input image, potentially meaning that failure here was caused by inability to extract useful information from the image, rather than inherent bias.

For a complete evaluation we can observe successful classifications from the same model

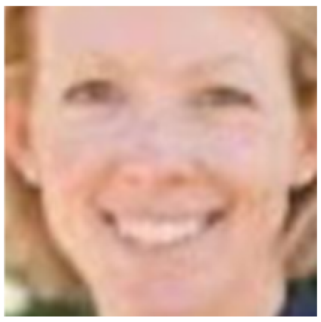


Figure 5.7: *UTKFace test image. Predicted class 1, was class 1*

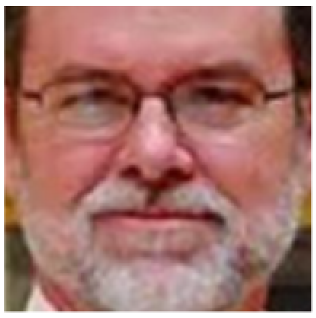


Figure 5.8: *UTKFace test image. Predicted class 1, was class 1*

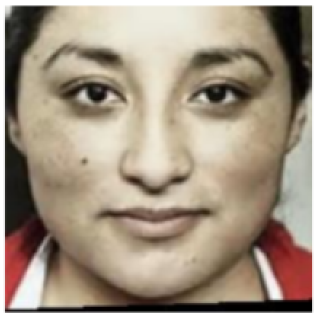


Figure 5.9: *UTKFace test image. Predicted class 0, was class 0*

Figure 5.9 shows a successful classification of a high-quality input image, while Figures 5.8 and 5.7 show successful classification from lower-quality input images. This shows that the model can classify successfully even for low-quality input images. Note however, that the successfully classified low-quality images are of White individuals, which is the race which this model presented the most noticeable bias towards.

Overall, inspection of the outputs of this model generally supports the bias analysis, with better performance for White individuals. However, not all failure cases are caused by potential bias, as seen in Figure 5.5, failure may be the result of low-quality input data.

Inspection of outputs for multi-class classification and regression show a similar trends in image-quality relating to accuracy, and supports the findings of bias analysis.

5.0.5 Summary

Overall, ResNet proved successful for convolutional facial age estimation, achieving classification accuracies of 88.1% in binary classification, 72.5% in multi-class classification, and achieving a RMSE of 7.9 years for regression on the UTKFace dataset and only slightly worse performance on the more challenging APPA-REAL dataset. SVM showed decent performance in binary classification, particularly for UTKFace where it achieved 75.9% accuracy, though it saw worsened performance on APPA-REAL.

Inspection of the models reveals that APPA-REAL trained models show better generalisability when compared to UTKFace trained models - likely due to the more diverse and challenging images within the APPA-REAL dataset. Evaluation of model bias reveals that models trained on each dataset present similar trends in both gender and racial bias per task - with improved performance for male and White individuals, and worsened performance for Asian individuals. However, the impact is amplified across all tasks for the APPA-REAL trained models - indicating potential under-representation of races in the training data.

Visual inspection of the results supports these conclusions, showing failure cases for certain racial groups, even with high-quality images, while finding success for White individuals with varying image quality.

6 Conclusion

6.1 Summary

Throughout this dissertation I have investigated convolutional approaches to computer vision for age estimation using facial images using two primary approaches:

- A Support Vector Machine which was applied to binary age classification.
- A Deep Residual Neural Network which was applied to binary age classification, multi-class age classification, and regression age estimation.

The APPA-REAL and UTKFace datasets were used throughout this project in order to train and evaluate models. This allowed for an understanding of how well these approaches and models generalised and enabled evaluation of model bias. Extensive inspection of each dataset was conducted to understand the data as well as class distributions and, where possible, additional gender and race metadata.

The SVM approach involved feature extraction, for this Local Binary Pattern and Histogram of Oriented Gradients was experimented with to see which fit the task best. Once a feature vectors were obtained, they would be used to fit the SVM and determine the optimal hyperplane which separates the classes. This approach saw moderate success in binary classification, achieving high accuracy on the UTKFace test data, and worsened performance on the more challenging APPA-REAL test data.

The ResNet approach involved three different tasks: binary classification, multi-class classification, and regression. Two different depths of architecture were experimented with: ResNet-18 and ResNet-50. Generally, deeper architectures saw improved performance across tasks. The primary difference between these tasks is how the model output is interpreted, either as the probability of a positive class prediction for binary classification, a collection of 4 probabilities for each of the multi-class classifications, or a continuous age value for regression. This approach saw good success across tasks and across datasets, proving to be an effective convolutional method for facial age estimation.

For each task, the best model trained on each dataset was selected to be evaluated in terms of generalisability and bias. APPA-REAL trained models showed better generalisability across datasets when compared to UTKFace trained models, likely due to the more challenging images in the training data. Similar trends in bias were observed per-task across models trained on either dataset, however, the impact of bias was greater with APPA-REAL trained models than with UTKFace trained models - indicating a potential under-representation of certain racial groups in the training.

6.2 Reflection

Overall, I believe I have successfully investigated convolutional methods for facial age estimation, and have provided sufficient analysis of generalisability and bias. I learned much about machine learning and deep learning throughout this project, not just in relating to computer vision but also more generally applicable knowledge.

Upon reflection, a more comprehensive analysis of methods could have been performed by comparing multiple different deep learning architectures, rather than one machine learning approach and one deep learning architecture. A key challenge during this project was the inability to train on large amounts of data, or with deeper models. I am fortunate to have had a GPU to train on, however deeper networks (such as training a ResNet model from scratch) were too computationally expensive still. This was somewhat alleviated through techniques such as transfer learning and data augmentation. Additionally, narrowing the scope to focus on more datasets with fewer approaches may have enabled better evaluation in terms of generalisability and bias.

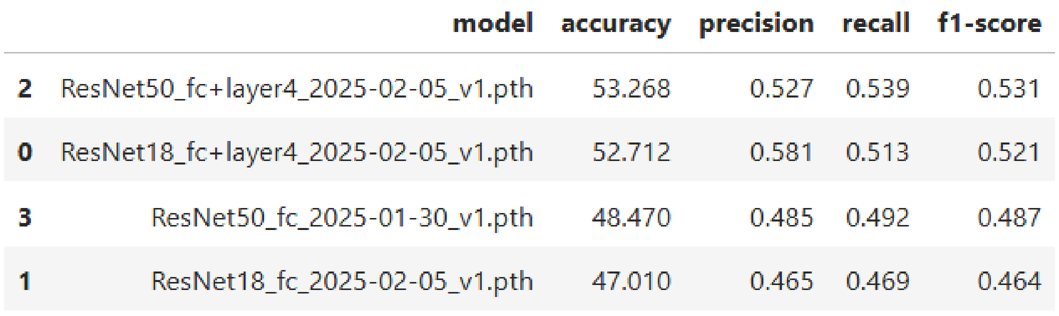
6.3 Future work

Expanding on this work could involve inclusion of different deep learning architectures. This could enable not just comparison of bias and generalisability between datasets, but also an evaluation of how different architectures perform across tasks with the same datasets. Allowing an investigation of how the network architecture itself might have an impact on bias and generalisability.

Another direction would be to expand on the datasets used. One example would be to increase the number of datasets to enable a comprehensive analysis of generalisability across various datasets. Another could be to curate a controlled dataset which can more easily identify bias in the network, for example by ensuring that image quality is consistent across all ages, races, and genders -eliminating this as a potential variable for misclassification, or by ensuring more fair representation of ages, races, and genders across the entire dataset. Additionally, including multiple datasets with useful metadata like UTKFace would help improve the evaluation of bias in the model.

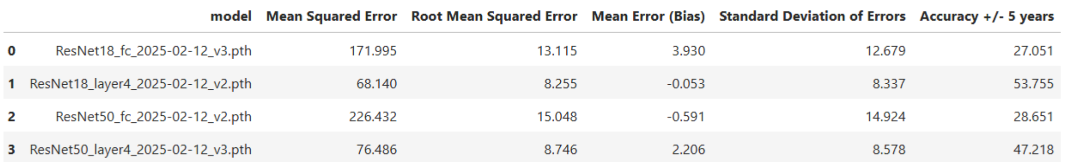
A Appendices

A.1 Evaluation: Model Selection Tables



	model	accuracy	precision	recall	f1-score
2	ResNet50_fc+layer4_2025-02-05_v1.pth	53.268	0.527	0.539	0.531
0	ResNet18_fc+layer4_2025-02-05_v1.pth	52.712	0.581	0.513	0.521
3	ResNet50_fc_2025-01-30_v1.pth	48.470	0.485	0.492	0.487
1	ResNet18_fc_2025-02-05_v1.pth	47.010	0.465	0.469	0.464

Figure A.1: *Validation performance for 4 APPA-REAL trained multi-class ResNet models. Sorted by macro f1-score.*



	model	Mean Squared Error	Root Mean Squared Error	Mean Error (Bias)	Standard Deviation of Errors	Accuracy +/- 5 years
0	ResNet18_fc_2025-02-12_v3.pth	171.995	13.115	3.930	12.679	27.051
1	ResNet18_layer4_2025-02-12_v2.pth	68.140	8.255	-0.053	8.337	53.755
2	ResNet50_fc_2025-02-12_v2.pth	226.432	15.048	-0.591	14.924	28.651
3	ResNet50_layer4_2025-02-12_v3.pth	76.486	8.746	2.206	8.578	47.218

Figure A.2: *Validation performance for 5 APPA-REAL trained regression models. Metrics are MSE Root MSE, including mean and standard deviation.*

A.2 Evaluation: Other Model Bias Metrics

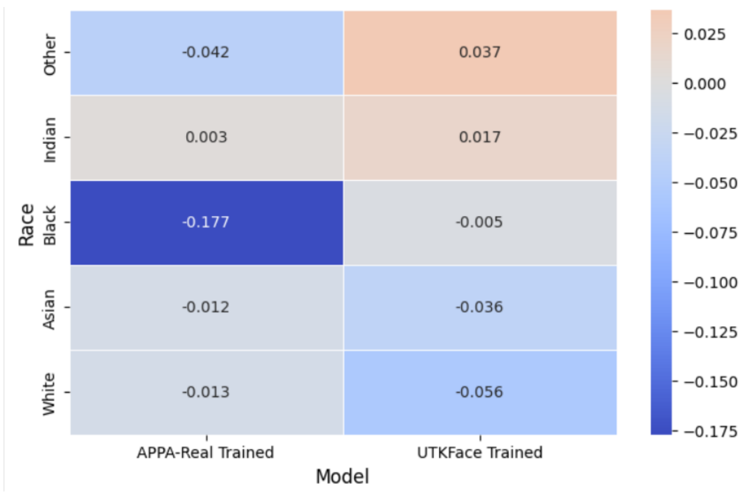
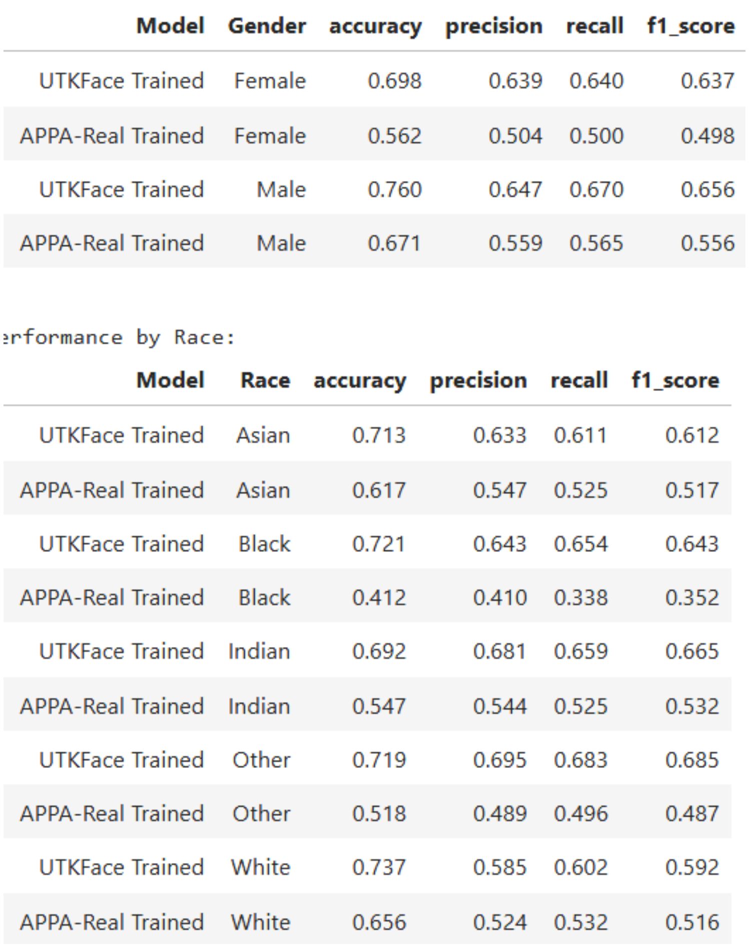


Figure A.3: *UTKFace test metrics showing the per-race macro f1-score deviation from the overall macro f1-score.*

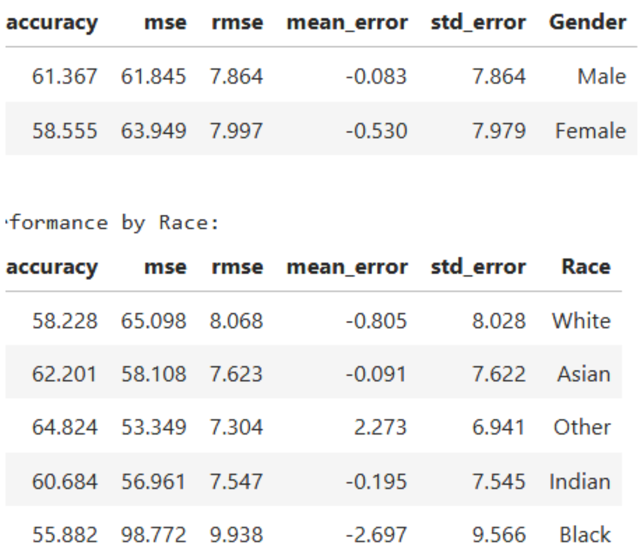


Model	Gender	accuracy	precision	recall	f1_score
UTKFace Trained	Female	0.698	0.639	0.640	0.637
APPA-Real Trained	Female	0.562	0.504	0.500	0.498
UTKFace Trained	Male	0.760	0.647	0.670	0.656
APPA-Real Trained	Male	0.671	0.559	0.565	0.556

```
erformance by Race:
```

Model	Race	accuracy	precision	recall	f1_score
UTKFace Trained	Asian	0.713	0.633	0.611	0.612
APPA-Real Trained	Asian	0.617	0.547	0.525	0.517
UTKFace Trained	Black	0.721	0.643	0.654	0.643
APPA-Real Trained	Black	0.412	0.410	0.338	0.352
UTKFace Trained	Indian	0.692	0.681	0.659	0.665
APPA-Real Trained	Indian	0.547	0.544	0.525	0.532
UTKFace Trained	Other	0.719	0.695	0.683	0.685
APPA-Real Trained	Other	0.518	0.489	0.496	0.487
UTKFace Trained	White	0.737	0.585	0.602	0.592
APPA-Real Trained	White	0.656	0.524	0.532	0.516

Figure A.4: *UTKFace test metrics for both multi-class classification models across gender and race.*

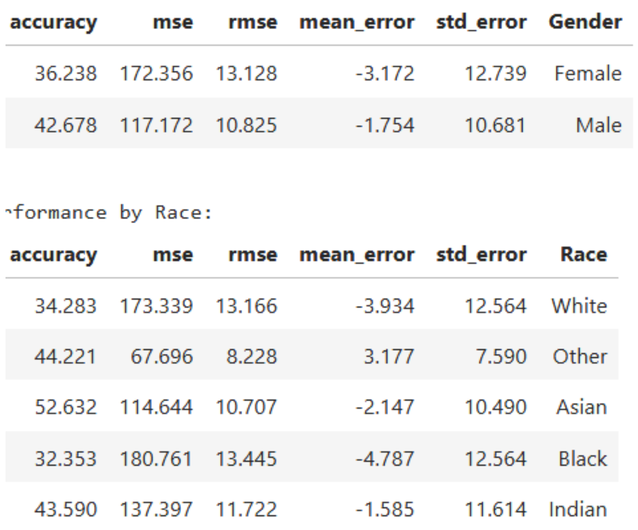


accuracy	mse	rmse	mean_error	std_error	Gender
61.367	61.845	7.864	-0.083	7.864	Male
58.555	63.949	7.997	-0.530	7.979	Female

```
formance by Race:
```

accuracy	mse	rmse	mean_error	std_error	Race
58.228	65.098	8.068	-0.805	8.028	White
62.201	58.108	7.623	-0.091	7.622	Asian
64.824	53.349	7.304	2.273	6.941	Other
60.684	56.961	7.547	-0.195	7.545	Indian
55.882	98.772	9.938	-2.697	9.566	Black

Figure A.5: *UTKFace test metrics for the UTKFace trained regression model across gender and race*



accuracy	mse	rmse	mean_error	std_error	Gender
36.238	172.356	13.128	-3.172	12.739	Female
42.678	117.172	10.825	-1.754	10.681	Male

```
formance by Race:
```

accuracy	mse	rmse	mean_error	std_error	Race
34.283	173.339	13.166	-3.934	12.564	White
44.221	67.696	8.228	3.177	7.590	Other
52.632	114.644	10.707	-2.147	10.490	Asian
32.353	180.761	13.445	-4.787	12.564	Black
43.590	137.397	11.722	-1.585	11.614	Indian

Figure A.6: *UTKFace test metrics for the APPA-REAL trained regression model across gender and race*

Bibliography

Agustsson, E., Timofte, R., Escalera, S., Baro, X., Guyon, I. and Rothe, R. (2017), Apparent and real age estimation in still images with deep residual regressors on appa-real database, *in* '2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)', pp. 87–94.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Fadhel, M. A., Al-Amidie, M. and Farhan, L. (2021), 'Review of deep learning: concepts, cnn architectures, challenges, applications, future directions', *Journal of big Data* **8**, 1–74.

AWS (2023), 'Amazon rekognition: Developer guide', `https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html`. Last Accessed: 16/02/2025.

Dalal, N. and Triggs, B. (2005), Histograms of oriented gradients for human detection, *in* '2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)', Vol. 1, pp. 886–893 vol. 1.

Ganel, T., Sofer, C. and Goodale, M. A. (2022), 'Biases in human perception of facial age are present and more exaggerated in current ai technology', *Scientific Reports* **12**(1), 22519.

Ghosh, S. and Bandyopadhyay, S. K. (2015), 'Gender classification and age detection based on human facial features using multi-class svm', *British Journal of Applied Science & Technology* **10**(4), 1–15.

Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y. (2016), *Deep learning*, Vol. 1, MIT press Cambridge.

Gruber, I., Hlaváč, M., Železnỳ, M. and Karpov, A. (2017), Facing face recognition with resnet: Round one, *in* 'Interactive Collaborative Robotics: Second International Conference, ICR 2017, Hatfield, UK, September 12-16, 2017, Proceedings 2', Springer, pp. 67–74.

Guyon, I. and Elisseeff, A. (2006), An introduction to feature extraction, *in* 'Feature extraction: foundations and applications', Springer, pp. 1–25.

He, K., Zhang, X., Ren, S. and Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Hiremath, J. S., Hiremath, S. S., Kumar, S., Hebbare, E., Patil, S. B. and Hiremath, M. S. (2022), Age detection based on facial features using support vector machine, *in* '2022 International Conference on Knowledge Engineering and Communication Systems (ICKES)', pp. 1–6.

Hossin, M. and Sulaiman, M. N. (2015), 'A review on evaluation metrics for data classification evaluations', *International journal of data mining & knowledge management process* **5**(2), 1.

Koonce, B. and Koonce, B. (2021), 'Resnet 50', *Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization* pp. 63–72.

LeCun, Y., Bengio, Y. and Hinton, G. (2015), 'Deep learning', *nature* **521**(7553), 436–444.

Madhavi, A., Bhuvana Sree, G., Shriya, V., Shanmukh, B. and Harshitha, T. (2021), Human age estimation using support vector machine, *in* 'Machine Learning Technologies and Applications: Proceedings of ICACECS 2020', Springer, pp. 273–286.

Mandal, B., Okeukwu, A. and Theis, Y. (2021), 'Masked face recognition using resnet-50', *arXiv preprint arXiv:2104.08997* .

Nielsen, M. A. (2015), *Neural networks and deep learning*, Vol. 25, Determination press San Francisco, CA, USA.

O'Shea, K. and Nash, R. (2015), 'An introduction to convolutional neural networks'.
URL: *https://arxiv.org/abs/1511.08458*

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D. and Walsh, J. (2020), Deep learning vs. traditional computer vision, *in* 'Advances in computer vision: proceedings of the 2019 computer vision conference (CVC), volume 1 1', Springer, pp. 128–144.

Pietikäinen, M. (2010), 'Local binary patterns', *Scholarpedia* **5**(3), 9775.

Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J. B. and Zuiderveld, K. (1987), 'Adaptive histogram equalization and its variations', *Computer vision, graphics, and image processing* **39**(3), 355–368.

PyTorch (2024), 'torch.nn.bcewithlogitsloss'. Accessed: 2025-03-23.
URL: *https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html*

Shorten, C. and Khoshgoftaar, T. M. (2019), 'A survey on image data augmentation for deep learning', *Journal of big data* **6**(1), 1–48.

Vapnik, V. N., Boser, B. E. and Guyon, I. M. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of the fifth annual workshop on Computational learning theory', pp. 144–152.

Weiss, K., Khoshgoftaar, T. M. and Wang, D. (2016), 'A survey of transfer learning', *Journal of Big data* **3**, 1–40.

YOTI (2023), 'What do customers really think about online age verification?', `https://www.techuk.org/resource/what-do-customers-really-think-about-online-age-verification.html`. Last accessed: 03/02/2025.

YOTI (2024), 'Yoti facial age estimation white paper', `https://www.yoti.com/wp-content/uploads/2024/11/Yoti-Age-Estimation-White-Paper-September-2024-PUBLIC.pdf`. Last Accessed: 16/02/2025.

Zhang, Zhifei, S. Y. and Qi, H. (2017), Age progression/regression by conditional adversarial autoencoder, *in* 'IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', IEEE.

Zhong, G., Ling, X. and Wang, L.-N. (2019), 'From shallow feature learning to deep learning: Benefits from the width and depth of deep architectures', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **9**(1), e1255.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H. and He, Q. (2020), 'A comprehensive survey on transfer learning', *Proceedings of the IEEE* **109**(1), 43–76.