Computer Science and Artificial Intelligence
COC257
F027980

ESTIMATING AGE AND GENDER FROM FACIAL IMAGES USING CONVOLUTIONAL NETWORKS

by Jack Paine

Supervisor: Dr. Sara Saravi

Department of Computer Science
Loughborough University

Submitted May 2023

Abstract

This project builds upon the efforts made in recent years to train convolutional neural networks (CNNs) for facial feature analysis, specifically for the estimation of age and gender[1] from facial images.

Children as young as three are intrinsically capable of recognising these traits by glancing at a person's face [2], yet encoding this ability into a mathematical model or computer program has remained a difficult challenge for many decades. However, recent advances in machine learning and the advent of convolutional neural networks have allowed researchers to replicate similar perceptive skills on computers to a remarkable degree.

This report provides an in-depth review of the best neural networks trained for facial image classification, and explores the process of creating, training, and testing such networks from scratch using publicly available tools and resources.

[1] There has been a growing discussion in recent years on the differences between psychological 'gender' and biological 'sex' [1]. For readability and to maintain consistency with past research in the field, gender will be used throughout this report to refer to either, however the differences between the terms is respected.

Acknowledgements

I would like to give a huge thanks to my supervisor, Dr. Sara Saravi, for the wealth of knowledge and guidance she has provided me through the creation of this project.

I would also like to thank my parents for their continued support through my degree, and for always encouraging me to pursue my passion in computing.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Attempts at computational facial image analysis saw limited success prior to the advent of deep neural networks (DNNs) and convolutional neural networks (CNNs). Literature on the topic dates back to 1999, when Kwon and Lobo [12] proposed the first geometry-based approach for age estimation; using wrinkle patterns and other geometric identifiers, the model was only capable of differentiating between distinct groups of infants, adults, and elders. However, when Yan et al. attempted the same task in 2014 using a deep learning approach [13], a model was produced capable of classifying subjects into one of 13 age categories with remarkable accuracy.

Deep networks and modern machine learning techniques have repeatedly been shown to produce state-of-the-art image classification models with accuracy rates comparable or even surpassing that of humans [14]. This report investigates the methodologies behind the leading facial analysis tools, and explores the process of creating such models with publicly available resources.

1.1 Motivation

Automated age and gender categorisation could be hugely beneficial to the futures of various industries. AI-based age estimation models are already being adopted for age verification in mobile applications [15], and it is not difficult to envision a future where marketing firms use such technologies to rapidly tailor advertisements for a given demographic. Security and surveillance organisations could also benefit from a tool that automatically categorises footage by age and gender, reducing the time spent analysing irrelevant footage.

Although beyond the scope of this project, the same methodologies have also been utilised for facial mood perception. One might consider the field of healthcare, where detecting mood changes could help carers provide timely support to patients in need, or help psychiatrists assess patients with potential mood disorders; for criminal psychologists and interrogators, AI-assisted mood detection could be particularly beneficial during interrogations or other scenarios where emotions are difficult to discern. Facial mood analysis is also arguably paramount for enabling natural human-machine interactions in the growing fields of robotics and AI assistants [**?**].

1.2 Project Aims and Objectives

This section details the specific aims of the project, and the objectives established to achieve these aims.

1.2.1 Project Aims

The primary aim of this project is to design and develop an age and gender categorisation system using CNNs for facial feature analysis. A secondary aim is to document the process (via this report), such that the project may be easily understood and replicated.

1.2.2 Project Objectives

1. **Literature Review:** Review the available academic literature on CNN-based facial image categorisation. Through this an understanding can be formed of the appropriate methods, tools, and available datasets for such tasks.
2. **Dataset Collection:** Source and prepare a large dataset of appropriately labelled facial images to train the neural networks.
3. **Model Implementation:** Design and train the model(s) for estimating each category. This may involve experimentation with various network architectures, training methodologies, and model optimisation techniques.
4. **Model Evaluation:** Evaluate and compare the robustness and accuracy of the model against the current state-of-the-art systems.
5. **GUI Implementation:** Design and develop a user interface to graphically visualise the model(s) from various user inputs, including any still images or video inputs.

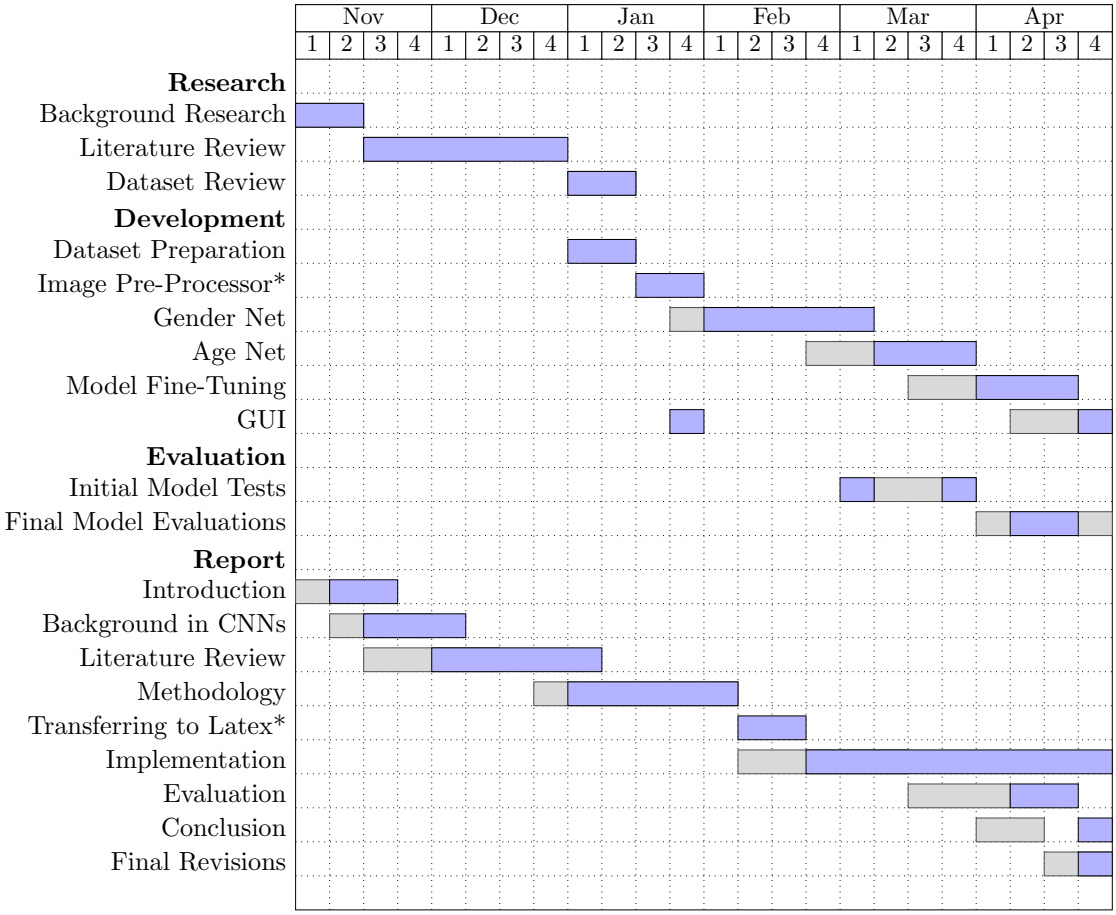
1.3 Project Deliverables

The following deliverables will be available at `https://github.com/jckpn/age-gender-cnn`:

- This report,
- Full source code for the creation and training of the neural networks,
- The final trained network models,
- A visualiser for testing and using the networks.

1.4 Project Timeframe

A Gantt diagram depicting the development timeline can be seen in *Figure 1.1* below. The original dates are marked in grey, and the actual dates (as logged during development) are shown in blue.



**Added after initial plan produced*

Figure 1.1: *Gantt chart depicting project development timeline*

1.5 Ethics

1.5.1 Ethical Concerns of Development Process

The development process that follows is likely to utilise one or more large datasets of facial images. The individuals in these images may not be aware or may disapprove of their images being used for this purpose.

Whilst every precaution will be taken to ensure that data is used responsibly, it is not plausible to gather consent from thousands of individuals for the use of their facial images in this project, and it is assumed that those compiling and publishing such datasets have ensured the images are sourced appropriately.

1.5.2 Ethical Implications of Tools Produced

While potentially effective for surveillance and crime reduction, automated facial analysis tools could pose a risk to the privacy of civilians if utilised for mass surveillance.

Automated mood estimation poses a similar risk to those unaware or non-consenting of its use. In 2014, Facebook controversially used a mood detection tool to secretly conduct psychological research on its users, leading to concerns about the potential for targeted marketing with more advanced AI systems [16, 17].

It is paramount that these tools are developed and utilised responsibly by those with access to them.

Chapter 2: Background in Neural Networks

This project focuses on using artificial neural networks (ANNs) to solve a long-standing problem in computational image analysis. By exploring the advent and evolution of ANNs, we can better understand their strengths and weaknesses to apply them in a meaningful way.

2.1 Advent and Rise

The concept for ANNs originally emerged in the 1980s as a potential method for imitating human cognition and decision making with computers [18]. Although initially unpopular, they gained a surge of popularity in the 2010s thanks to an increase in computing power and the financial motivation for large-scale data processing, such as for social media services [19].

2.2 The Perceptron: Building Block of Neural Networks

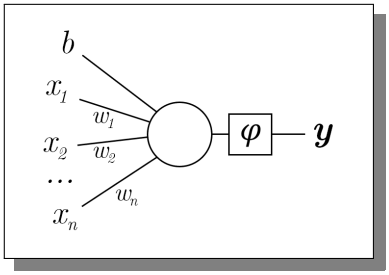
The perceptron is a type of artificial neuron, and the building block for neural networks used today. It is a simple mathematical function which takes a series of inputs and multiplies each input by a corresponding 'weight' value, indicative of the input's significance in the overall calculation.

An additional value, commonly called the *bias*, is added to provide an additional degree of adjustment independent of the input values. The outcomes of these multiplications are summed to produce the perceptron's output. The perceptron also crucially includes an 'activation function' to map its output to a specified range, such as $0 \to 1$ or $0 \to \infty$; the non-linearity introduced by the activation function is what makes neural networks so much more powerful than traditional linear regression models [20].

$$y = \phi(b + w_1x_1 + w_2x_2 + \cdots + w_nx_n) = \phi(b + \sum_{i=1}^{n} w_ix_i)$$

Where y is the output, ϕ is the activation function, b is the bias value, $x_1, x_2, \cdots x_n$ are the inputs, and $w_1, \cdots w_n$ are the corresponding weights.

(a) *Mathematical equation*



(b) *Visual representation*

Figure 2.1: *Different representations of the perceptron*

The perceptron can be represented mathematically by the equation shown in *Figure 2.1a*,

however it is often more intuitive to represent it visually, like in *Figure 2.1b*, to better demonstrate its structure and function.

2.3 Network Layers and Deep Networks

Neural networks are established by connecting multiple perceptrons, now referred to as the network's neurons; this is done by using each perceptron's output as the input for other perceptrons within the network.

The neurons are typically organised into layers, with the first layer taking the specified input values, and the final layer representing the network's outputs. Deep neural networks (DNNs) are a special class of neural networks with at least two 'hidden layers' positioned between the input and output layers (see *Figure* **??**, allowing for more complex computations to be performed.

This particular network contains 2 hidden layers, and could be purposed for a variety of problems – for example, it could be used to predict the best selling price of a used car by using the car's age and mileage as inputs (*Figure 2.2*).

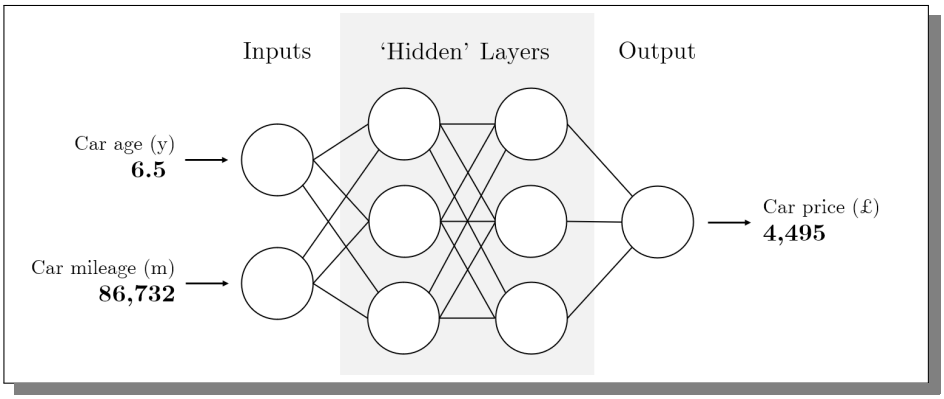


Figure 2.2: *Purposing the network for car pricing predictions*

The hidden layers aid in making sophisticated predictions that a classic 'car price equation' could not consider; for instance, it may deduce through training that a car with exceptionally high mileage for a lower age is indicative of poor care, and subsequently predict a lower price.

The above is an example of a *regression* problem, which yields a single numerical output. In contrast, classification problems (such as gender predictions) typically employ multiple output nodes, with each node corresponding to a specific class. For example, the previously illustrated deep network could be adapted to differentiate between flower species based on their petal dimensions, as in *Figure 2.3*.

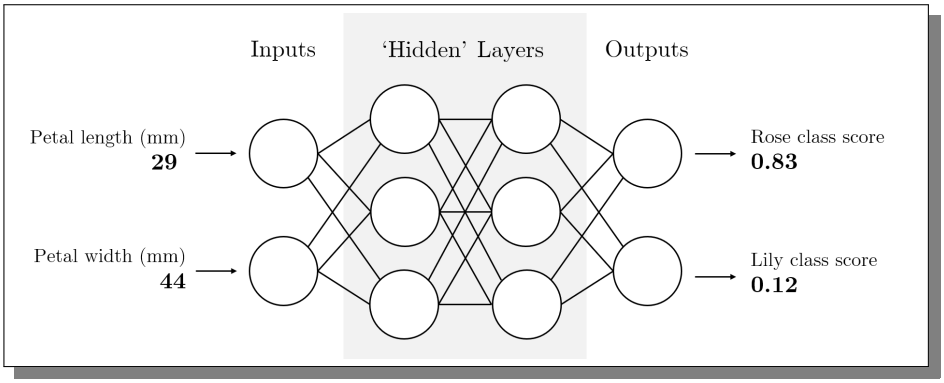


Figure 2.3: *Adapting the network structure for flower classification*

By incorporating two separate outputs nodes, the network can generate an individual score for each class, roughly indicative of its 'confidence' for that class's presence. For instance, if an input does not correspond to either a rose or a lily, the network could yield a low value for both output nodes; conversely, if the input image has characteristics of both roses and lilies, the network may assign high values for both classes.

The optimal architecture for a network largely varies by task, with countless possible combinations of inputs, outputs, and hidden layers. Generally, networks with more neurons and layers can produce more nuanced and accurate results, but using overly complex networks for a given task can lead to issues like overfitting, further detailed in **2.5.1**. [**?**].

2.4 Network Training

The accuracy of a network is fully dependent on the chosen weight and bias values, collectively known as the network's 'parameters', which are typically randomly generated when initialising a network. As such, a new, untrained network will produce random and seemingly arbitrary results. However, with repeated iterations of a suitable training process, the parameters for the network can converge to values that consistently output accurate results.

Various training processes exist, and the specific methodology employed for this project will be discussed in greater detail in the Methodology chapter. However, the fundamental concept can be summarised as follows [21]:

1. Initialise the network with random weights and biases.
2. Select an entry from an extensive pool of labelled training data.
3. Run the network layer-by-layer using the inputs from this entry.
4. Compare the network's output to the label provided with the data entry.
5. Adjust the weights and biases of the network, using the difference between its output and the entry label to guide the adjustment.

6. Repeat steps 2-5, iterating through all training samples until the network's accuracy ceases to improve.

Depending on the task and size of the dataset, it can take anywhere from 5 to 100s of epochs – full iterations over the dataset – to acquire the desired model weights. The success of the training process is heavily dependent on both the quality and quantity of training data, especially for complex tasks like image recognition [10].

This process of continuously adjusting a neural network's connections and 'training' it with data is analogous to the concept of neuroplasticity in animal brains – the mechanism by which brains modify their connections to promote or inhibit specific behaviors through experience, especially in young children [**?**].

2.5 Convolutions and Convolutional Networks

Despite their vast capabilities, traditional DNNs are inherently limited for image recognition due to the one-dimensional nature of the input layer. To combat this, convolutional layers can be integrated at the start of the network, providing it with a sense of two-dimensional spatial awareness [22].

In the context of digital images, a convolution is essentially a filter applied over an image to emphasise certain patterns, such as in *Figure 2.4*. During the network's training, these convolution functions become increasingly adept at extracting the necessary image features to solve the given task.

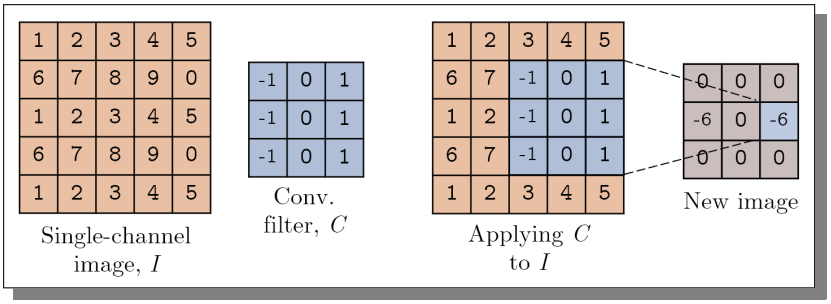


Figure 2.4: *Applying a convolution filter*

The outputs from the filters (referred to as 'feature maps') are then passed through fully connected linear layers, like those previously illustrated in *Figure* **??**, to produce the final model's output(s). The inclusion of convolution functions provides CNNs with a significant performance advantage in image recognition tasks when compared to traditional neural networks.

Although the example depicted in *Figure 2.4* is for a single-channel (i.e. grayscale) image, the method can be applied to standard 3-channel RGB images by running a separate convolution function for each channel.

2.5.1 Overfitting

Overfitting occurs when a network becomes exceptionally adept with its training data, yet fails to generalise well to unseen test data. A network is generally at higher risk of overfitting if the architecture is overly complex for the given task or has been provided with insufficient training data.

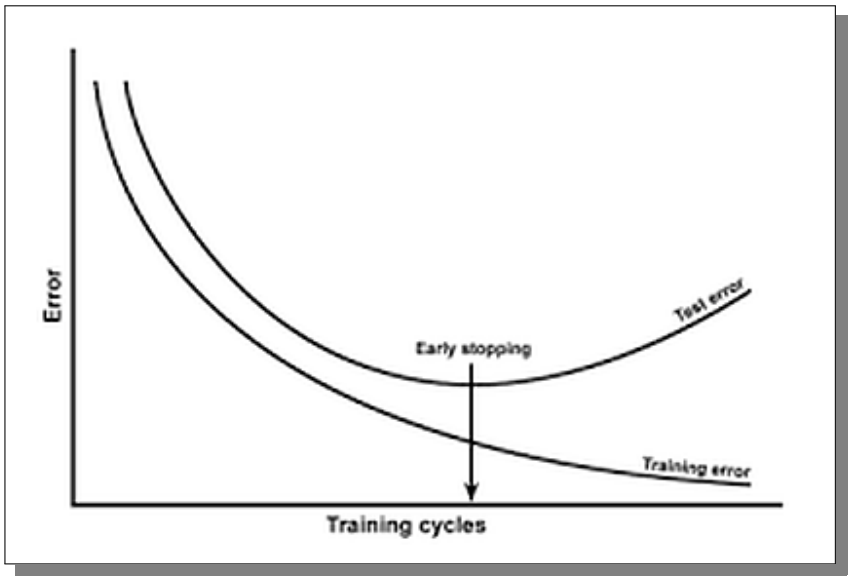


Figure 2.5: *Example of network overfitting – the training loss decreases while the validation loss increases. Source: [3]*

To detect and prevent overfitting, it is typical to run secondary tests during training with a separate, unseen set of 'validation data' – declining performance on the validation set is a good indicator that the network is overfitting, and can be used as an indication to halt the training process.

Chapter 3: Literature Review

Academic literature on CNN-based facial image analysis dates back to 2014, with a successful model from Yan et al.[23], and continuous advancements seen up to the present day. While by no means an exhaustive list, here are findings from some of the most cited papers on the topic[1], including research from: Yan et al., 2014 [23]; Huerta et al., 2015 [10]; Liew et al., 2016 [24]; Rodriguez et al., 2017 [25]; Rothe, Timofte and Gool, 2016 [26]; Mehendale, 2020 [27]; and Adile, 2021 [**?**]. A summary of the literature review is provided at the end of this chapter.

3.1 The Key Challenges of Facial Image Analysis

Yan et al. and Huerta et al. both cited similar challenges for the task, such as the unpredictable nature of aging, and camouflaging of facial features from occlusions like facial hair and makeup. They both emphasized the importance of large datasets to overcome these difficulties, with Huerta et al. specifically stating that over 10,000 training samples would be necessary for adequate results.

Rothe et al. identified the key challenges to be face detection errors, poor lighting conditions as well as the previously mentioned occlusions from make-up, glasses and facial hair.

3.2 First Success with CNN-Based Age Estimation

Yan et al. presented the first[2] attempt at CNN-based age and gender estimation at the Pacific Rim Conference on Multimedia in 2014 [23]. They demonstrated their deep learning approach using the 2009 WebFace dataset [**?**], citing several potential challenges such as gender differences, ethnicity, and lighting conditions.

They trained separate models for male and female age estimation, using a prior convolutional network to determine the individual's gender, allowing for more accurate age predictions than if they were trained together due to the cited gender ageing differences. Each age network was trained to classify facial images into one of 13 age groups between 0 and 80 years old.

The final age estimation model achieved varying accuracy scores between 96.4% and 67.8% for each group, with higher accuracy achieved for the younger age groups, perhaps indicative of a bias towards younger individuals in the dataset.

[1] Due to time constraints, only papers released before December 2022 were reviewed.

[2] To my knowledge, this is the first demonstrated attempt to use convolutional networks for age estimation, but it is possible that attempts were made privately prior to this.

3.3 Introduction of an Image Pre-Processor

The success of Yan et al.'s model was later improved on by Huerta et al. [10] and Liew et al., [24], who included image pre-processors in their process flow prior to the network's input. The image pre-processors cropped, aligned, and resized images, with Huerta et al. opting for a slightly larger size of 50×50 compared to the 28×28 size chosen by Liew et al. This alignment process enabled the networks to focus on important details for the task while ignoring background distractions.

Rothe et al., DATE HERE [26] used an even more sophisticated image pre-processing system that iteratively flipped and rotated each image until the best-scoring variant was obtained. The coordinates from the optimal variant were used to crop the image with a 40% margin around the face and resize it to 256x256 pixels.

Mehendale, 2020 [27] proposed a preliminary background-removal network to remove redundant data from the inputs using skin-tone-based background removal.

Pre-designed face alignment tools already exist for these purposes, with Adil, 2021 [**?**] utilising the FaceAligner Python module by PyImageSearch [28] to standardise images for his model.

3.3.1 Attention Mechanisms

Furthering on the concept of input pre-processing, Rodriguez et al., 2017 [26] introduced a novel 'attention mechanism' to improve facial image analysis, drawing inspiration from biological eye fixations. The mechanism used a preliminary CNN model that selected the most relevant parts of an image (for example, 2 squares from a 4x4 grid) for evaluation by the final model. The group used the MORPH II [**?**], Adience [29], and 'Images of Groups' (IoG) [**?**] datasets for training and testing their models. The model was comprised of two main parts: the attention network as described above, and a 'patch' network for evaluating the chosen image parts for age and gender distinctions; a final linear classification layer combined the results for the final classification. This approach to image processing addressed the camouflage issues mentioned by Huerta et al. [10], achieving a 4% improvement in age estimations over Rothe et al.'s age estimation performance [**?**] when evaluated using the same methodology.

3.4 Architecture Advancements

Yan et al. utilised a simple architecture of one convolutional layer, a pooling layer to compress the feature maps, and a support vector machine (SVM) model for the final classification; a somewhat different approach to the fully-connected linear classification models typically used today.

Huerta et al. specified their architecture as using a 32-channel convolutional layer, followed by another 50-channel convolutional layer, both utilizing pooling layers before output to a final linear layer for classification classification layer. Huerta et al. used a 'neuron

dropout' mechanism to prevent network overfitting by removing neurons which appeared underactive during training. Liew et al. utilised a somewhat similar 'partial-connection scheme' between the convolutional layers to ensure each kernel was trained for different features, increasing the efficiency of the network.

Rothe et al. employed a sophisticated 16-layered network architecture proposed by Simonyan and Zisserman (known as VGG-16) [**?**], opting to use 101 separate output nodes to classify age to the nearest whole year.

Mehendale experimented with architectures of various complexities before deciding to use LeNet-4 [30], a traditional four-layered CNN used to classify the enhanced images as normal.

3.5 Occlusion Experiments

Rothe et al. discovered that the eyes play a significant role in age classification, as performance declined the most when they were occluded compared to other facial regions.

3.6 Real-Time Application

Yan et al. found that their model worked similarly well with a live video (webcam) input as it did with still images, due to the relatively low processing times of convolutional network.

Mehendale implented a similar system for his emotion recognition model. To address the issue of camera blur during real-time use of the model, he developed a novel frame extraction algorithm, designed to capture frames using a frame difference algorithm.

3.7 Model Interface and Visualiser

In 2021, David Cochard of ax Inc. [4] demonstrated their real-time age and gender prediction system with a modern-looking visualiser to make results easy to interpret. The interface takes an image or video as input and draws a coloured box around each subject's face, along with the corresponding model predictions (*Figure 3.1*).

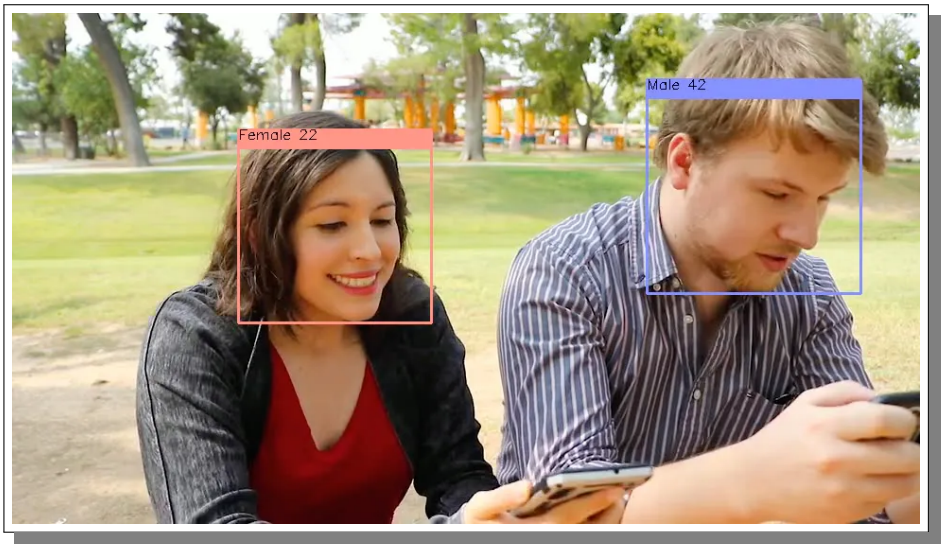


Figure 3.1: *Visualiser of ax Inc.'s model. Image source: [4]*

3.8 Datasets

Huerta et al. trained the network on two large datasets, FRGC v2.0 [**?**] and MORPH-II [**?**].

3.8.1 Introduction of Large-Scale Datasets

Until 2016, the datasets available for this task were all under 50,000 images; Rothe et al. hypothesized that an even larger data pool would enable for a more robust model. As such, they formulated an automated process to produce IMDB-WIKI [**?**], a large-scale dataset of over 500,000 entries with automated gender and age labels taken from IMDB and Wikipedia. This dataset was specifically created for their project, and introduced for the first time in this paper. Through initial experimentation with training datasets, they concluded that the quantity of training data was more crucial than the quality or accuracy. Specifically, they found that the 500,000+ entries of IMDB-WIKI yielded superior training results despite inaccurate labels, when compared to a smaller, cleaner dataset of 7,600 human-verified entries.

Their final evaluation resulted in an impressive MAE of just 2.68 years. The final model is known as the 'DEX' algorithm and is widely cited as one of the best and most robust age estimation models to date [**?**, **?**].

3.8.2 Dataset Equalisation

In his 2021 paper, Adil [**?**] ephasises the importance of reducing data bias to achieve accurate and consistent results for facial recognition networks; he points out that many existing models in this field are improperly trained due to the use of imbalanced datasets,

with a disproportionate numbers of photos per emotional category leading to bias in the final network's predictive process.

The success of Rothe et al.'s model could be partly attributed to their use of dataset equalisation, in which the classes of the dataset were equalised – in other words, data samples for underrepresented groups were augmented to allow for fairer network training, mitigating the risk of model bias from unequal representation of groups in the training data.

3.8.3 Data Augmentation

Huerta et al. utilised horizontal flipping to augment extra data for training, effectively doubling the dataset size for the same storage requirements. Adil, 2021 [**?**] also practiced this, augmenting additional training data by using random image cropping.

3.9 Extending to Mood Estimation

The same techniques employed for facial age and gender estimation have also recently been used for surface-level facial emotion detection: Mehendale, 2020 [27] and Adil, 2021 [**?**] both demonstrated success with CNN-based emotion recognition. Mehendale's 'FERC' (Facial Emotion Recognition with CNNs) could correctly classify facial expressions as happy, sad, fearful, surprised, angry or neutral in 96% of tests conducted, and Adil's model could classify into similar categories 80% of the time.

3.10 Table Summary

It would be unfair to directly compare the results of each paper as each research group employed a different testing methodology with completely different testing data. Regardless, an overview of each paper with their self-evaluated results is provided in *Table 6.1* to show the rough trend in image classification techniques and advancements.

Reference	Category	Model Pipeline?	Novelty	Accuracy
Yan et al., 2014 [23]	Gender	Preprocessor → CNN → SVM	First CNN-based gender estimation	98.1% (2 classes)
Liew et al., 2016 [24]	Gender	Cropping → CNN; Cross-correlation instead of convolution layers	Image pre-processor with eye alignment	99.4% (2 classes)
Rodriguez et al., 2017 [25]	Gender	'Attention' CNN → 'Patch analysis' CNN	Attention mechanism	93.0% (2 classes)
Yan et al., 2014 [23]	Age	Cropping → Conv layers → SVM classifier → 13 outputs; Separate models for male and female	Separate age estimation models for male and female	76.6% (13 classes)
Huerta et al., 2015 [10]	Age	Cropping + eye alignment → CNN with dropout → 1 output	n/a	3.6% MAE
Rothe et al., 2016 [26]	Age	VGG-16 CNN → 1 output	Real vs. apparent age estimation	2.7 years MAE
Rodriguez et al., 2017 [25]	Age	'Attention' CNN → Patch analysis CNN	Attention mechanism (i.e. CNN pre-processor)	61.8% (8 classes) or 2.6 years MAE
Mehendale, 2020 [27]	Mood	Background removal → 4-layer CNN	Still frame video extraction	96.0% accuracy (5 classes)
Adile, 2021 [?]	Mood	4-layer CNN	Data augmentation (flipping, rotating)	80.0% (5 classes)

Table 3.1: *Summary of reviewed literature*

Chapter 4: Methodology

4.1 Software

4.1.1 Programming Language

Few of the reviewed papers disclosed their choice of programming language, perhaps alluding to its triviality in the field of network development – it follows that the choice of language is down to the developer's preference and experience. Python [31] has recently emerged as a popular choice for machine learning [32] owing to its easy learning curve and widely supported frameworks for ANN development (see below).

Python 3.10, as available from the official website, will be used for the development of this project [33].

4.1.2 Network Development Framework

A network development framework provides the modules to build, train and test a network, significantly reducing the code requirements for this project. These frameworks are also often highly optimised, allowing for faster training and potentially better network performance.

There are many neural network development libraries available for Python; PyTorch [34] by Facebook was chosen for this project, due to prior experience and the abundance of online support for image recognition tasks [35]. As well as streamlining the process of network creation and training, PyTorch 1.13 includes several well-established network architectures for visual processing tasks, as well as numerous relevant tools for image pre-processing and standardisation.

4.1.3 Operating System and IDE

Microsoft Visual Studio Code [36] (VS Code) will be used for this project along with Windows 11 due to familiarity and compatibility with required software.

4.1.4 Environment Manager

Conda [37] is a virtual environment manager and package handler for Python, which aids development by creating distinct virtual environments for each project on the user's machine. It ensures that projects are self-contained, preventing conflicts with packages from other projects.

Conda also allows for easy replication of the development environment, enabling others to use (or develop) the code without needing to install each dependency individually.

4.1.5 Version Control

GitHub [38] is a popular code repository host allowing others to view and run the project, as well as enabling development across several machines. It also provides robust version control, allowing changes to be reviewed and reverted if necessary during development. VS Code also features extensive integration with GitHub.

4.1.6 Python Libraries

One of the main advantages of Python is the abundance of packages available for a wide range of tasks, reducing the amount of code needed to be written from scratch. At minimum, the following packages will be used for this project:

Numerical Operations: NumPy

NumPy [39] is a widely used package for numerical operations, with useful tools for handling arrays and matrices among other functions. NumPy can be installed and integrated into the project using either Pip [40] or Conda.

Image Processing: OpenCV

OpenCV [41] is another widely used Python package for computer vision, with a wide range of tools for image manipulation, colour normalisation, and object detection. It will be used to preprocess the input images and display images as necessary.

OpenCV also enables drawing on images, which should prove useful for creating the model's visualiser and graphical interface (GUI). As with NumPy, it can be included via Pip or Conda.

Other Packages

Any other packages used will be specified and cited in the relevant Implementation section.

4.2 Hardware

4.2.1 Local Hardware

The majority of processing development will be performed on a Huawei Matebook with the following specifications [42]:

- **CPU:** AMD Ryzen 5 Mobile 3500U @ 2.10 GHz
- **RAM:** 8 GB DDR4

4.2.2 Google Colab

If training is found to be too slow on the Huawei Matebook, more powerful hardware can be accessed through Google Colab [43], as suggested in [35]. Google Colab is a cloud-based

Python environment, providing users with access to high-end machine learning hardware at a relatively low cost.

4.3 Image Pre-Processor

The literature review highlighted the importance of pre-processing input images for robust model performance. Optimising and standardising input data is crucial for responsive network training, so a pre-processor should be included if the proposed model is to compete with the accuracy of existing models.

4.3.1 The Necessity of an Image Pre-Processor

There are several seemingly minor variations a facial image can have which could hinder training performance: lighting variations, face angle variations, camera quality, image noise, contrast differences, background details, and foreground occlusions were all cited as potential 'distractions' for a network being trained for facial image classification [23, 10, 27].

An image pre-processor should regulate these variances such that the inputs appear more standardised for the network. An effective pre-processor must also maintain, or even highlight the features relevant to the classification process. As an example, we might expect the pre-processed version of UTKFace to look like below (*Figure 4.1*), which would make for more suitable inputs to our networks.

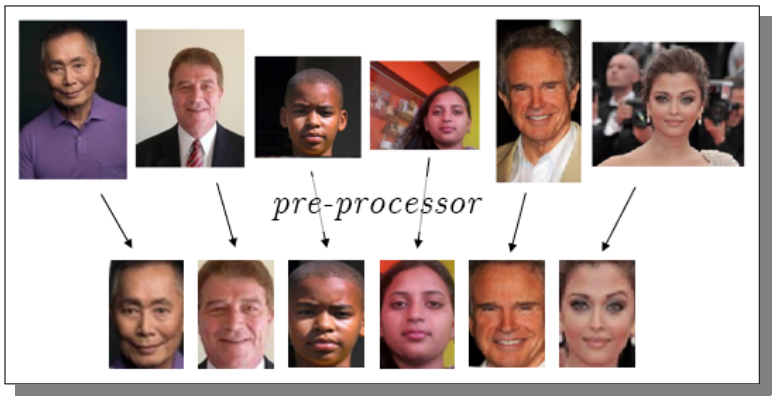


Figure 4.1: *Example of images before and after pre-processing. Source: UTKFace dataset [5]*

If utilised correctly, implementing a suitable pre-processor will enable:

- increased accuracy and robustness of the model,
- multi-face detection, allowing several subjects to be analysed from the same image,

- robustness to diverse image sources, such as wide-angle CCTV shots, group images, and close-up portraits.

4.3.2 Pre-Processor Specifications

Although the exact methodology varied between research groups, all pre-processors were designed to eliminate background details while highlighting and standardising the relevant facial features. Specifically, eye-alignment and cropping techniques were utilised by the majority of researchers to achieve this [10, 26, 10, 24], while Rodriguez et al. [25] took the concept further by incorporating an attention model to extract relevant image patches – essentially using a preliminary CNN for their image pre-processor.

Colour Normalisation

It is standard to apply some form of contrast and brightness normalisation to images when using neural networks as a way of standardising the image data contents. This functionality is built into OpenCV, which will be used for this project.

Face Detection

Regardless of the specifics of the alignment process, a face detector is necessary to locate the relevant parts of the image – most likely to be the individual eye positions and face borders. For the purposes of this project, the facial detection system must:

- Be relatively fast (i.e. less than 1 second) for proficient real-time use of the model,
- Not require dedicated hardware such as a GPU, as this would limit the amount of devices that can use the tools,
- Work for a wide range of face sizes,
- Work for both grayscale and colour images,
- Detect face edge co-ordinates for cropping, and
- Detect eye co-ordinates for rotational alignment.

In [44], Gupta (of LearnOpenCV) compared four popular models for facial detection in [**?**] and concluded that the OpenCV `FaceDetectorYN` module [**?**] performs the best in terms of accuracy, robustness, and speed.

Image Resizing

Resizing each image is necessary to keep input data consistent for the convolutional networks. A larger image size could lead to better performance due to the higher amount of detail present, but as with any increase in network complexity, it may reduce end performance by hindering training. Liew et al. [24] used a relatively small image size of 32x32, meanwhile [26] used a far larger size of 256x256, both to good effect.

Face Cropping and Alignment

The image cropping stage provides two purposes: it removes unnecessary background details, allowing the models to focus on important facial features; it also allows a more adaptable and robust model, enabling the CNN models to work on more diverse input sources like close-up portraits as well wide-view inputs.

Rothe et al. found through their occlusion experiments that the eyes, of all facial features, were the most indicative of a person's age [26] – as such, aligning facial images with the eyes is likely to enhance performance when compared to simple face cropping alone. The majority of pre-processors seen used a form of rotational alignment to align the eyes horizontally; for example, Huerta et al.'s pre-processor aligned each eye to 25% and 50% x and y co-ordinates respectively [10].

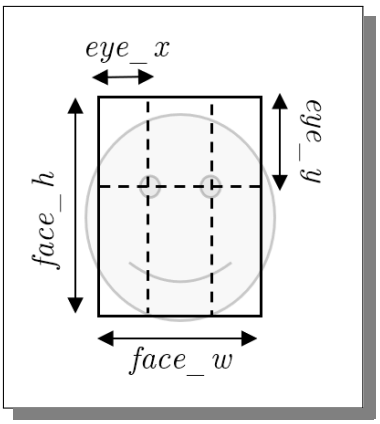


Figure 4.2: *Provisional face alignment grid*

A customisable alignment function can be implemented using the image manipulation tools of OpenCV, following the alignment grid as depicted in *Figure 4.2* as a basis. By altering the function's parameters, various alignment strategies can be tested to see what leads to the best end results.

The parameters of this alignment can, of course, be experimented with; we can initially use dimensions of 224×224 for *face_w* and *face_h* respectively, using the recommended input shape and dimensions for AlexNet and VGG16; the images may be resized after processing with PyTorch transforms, i.e. if we want to scale down images for LeNet.

eye_x and *eye_y* will initially be set to 0.25 and 0.5 respectively, following from the values used by Huerte et al.

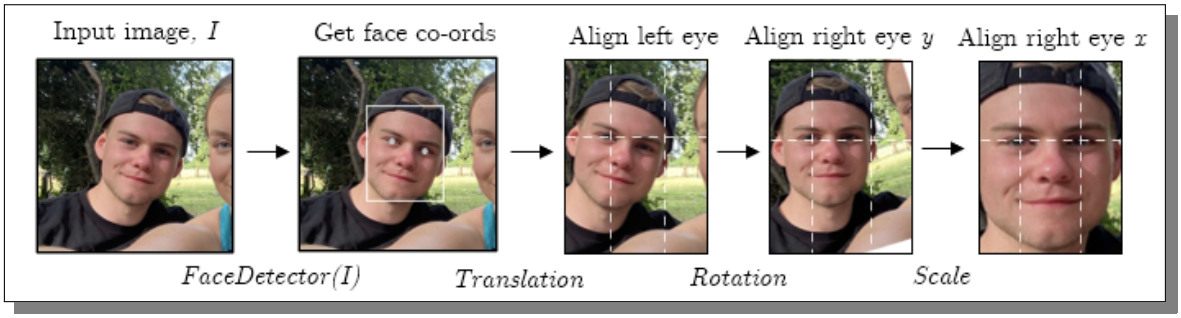


Figure 4.3: *Provisional alignment process*

Following alignment, image resizing and colour normalisation can be applied, before passing the final image to the relevant classification model(s).

Background removal

Mehendale employed skin-based background removal techniques in his model pipeline for mood estimation [27]; however, comparitive results were not provided. It is worth considering that background details such as clothing could influence perceived age perception, as people may subconsciously associate certain styles with specific age groups, so background removal would help mitigate the risk of our model learning these biases.

Given sufficient time during development, a background removal process may be experimented with and compared with 'traditional' cropping and alignment techniques as utilised by the other researchers.

Live Video Processing

The OpenCV module supports live video input from webcams. A technique akin to the one proposed in [27] could be employed to extract the most stable frames, with the rest of the analysis process remaining the same.

4.4 Network Architecture

4.4.1 Considerations

Depth and Parameter Count

As briefly mentioned in **Section 2.3**, the depth of a neural network may correlate to a higher degree of accuracy; conversely, an overly complex network can result in reduced performance due to overfitting, training difficulties and/or latency issues [45]. This is corroborated by the reviewed literature, as the chosen network depth was highly correlated with the size of the image input – for instance, [24] used a 4-layer network to good success with 32×32 sized images, while the 224×224 images used in [25] necessitated a 16-layer network.

Although the majority of early literature indicated an optimal network depth of 4-5 layers, the most recent attempts proved great success when utilising the 16-layer VGG-16 architecture [26, 25]. Notable downsides might be slower processing speeds, a far lengthier training process, and potentially slower end-user processing, all of which could cause issues for real-time processing.

Mehendale specifically compared the performance between several network sizes on emotion recognition and concluded that a network depth of around 4 layers provided the best results for his 5-class mood recognition task [27], with worsened results when attempting to utilise a network with 16 layers. This information can be taken into consideration when designing the various network architectures for this project, also taking into account the correlation between network depth and input image size.

It is not possible to determine the ideal architecture solely based on the reviewed literature, and the best performance may depend on the specific task – age and gender classification may exhibit different levels of complexity, and necessitate entirely different architectures for optimal results. Fortunately, PyTorch enables easy adaptation and retraining of multiple networks with varying architectures and parameters. Moreover, the pre-processor module discussed earlier allows for fully customisable alignment, which will enable easy comparison of their performances.

Number of Output Classes

For gender, all three datasets utilise a binary gender label, so this network will need to use two classes. It would be possible to use a single output node for this model, but using two is advantageous for the reasons described in **2.3**.

For age, a regression model is proposed due to easier testing and higher practicality in real-world use than distinct age groups.

Rothe et al. [26] specifically experimented with various output class numbers and concluded that using 100 output neurons – one for each age year – provided the most accurate results. However, their success with such a large-scale output classifier could be difficult to replicate given our comparatively limited processing power.

Convolutional Filters

Convolutional filters (or 'kernels'), are responsible for the feature detection within the neural network. A common choice for kernel sizes in popular convolutional neural networks (CNNs) such as LeNet and AlexNet is 5×5; with sufficient time for experimentation, the effects of kernel size could be further explored but initially the kernel sizes as originally specified for each corresponding architecture will be used (detailed later).

Pooling Layers

Pooling layers are crucial for making convolutional networks more efficient without sacrificing performance or classification accuracy. Two popular types of pooling are 'max pooling'

and 'average pooling' – max pooling selects the maximum value from a defined region in the feature map, whereas average pooling computes the mean value of the region.

The specifications for each pooling layer will be used as initially proposed for each architecture for the same reasons as described above.

Activation Functions

Activation functions introduce non-linearity into the network, allowing it to learn complex patterns within the input data. Without it, a neural network cannot perform any better than a traditional linear regression model; the activation function is what allows a network to learn and utilise advanced classification techniques.

Popular activation functions include ReLU, Sigmoid and Tanh; ReLU (*Figure 4.4* has emerged in recent years as a highly capable activation function, with faster training times than Sigmoid or Tanh due to its simplistic nature [46].

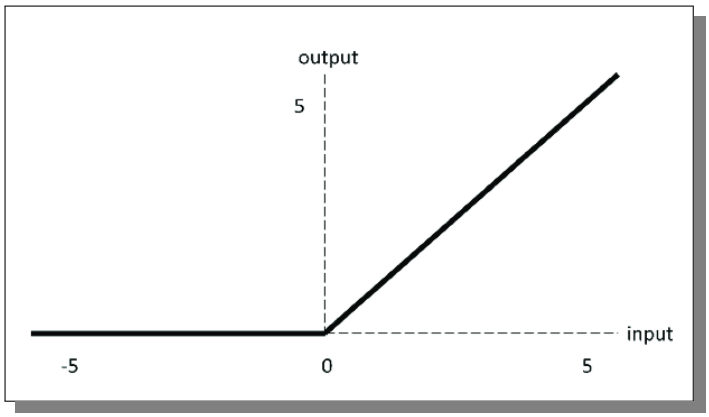


Figure 4.4: *ReLU activation function*

4.4.2 Proposed Architectures

As is often done in machine learning papers to convey network structure, the following architectures in this report may be informally represented with the following notation:

- I_n = input with n channels ($1 \rightarrow$ grayscale, $3 \rightarrow$ colour RGB)
- C_n = convolutional layer with n output channels (feature maps)
- P = pooling layer
- L = fully-connected linear layer

A diagram for the networks is also presented for further clarification of the proposed structures. The notation and diagrams are presented with two classes, as if purposed for gender classification, however the output layer will be adapted to have a single regression output as determined earlier.

2-layer ConvNet

Architecture: $I_1 \rightarrow C_6 \rightarrow P \rightarrow L_2$

This network contains a single convolution layer for feature extraction followed by a single linear layer for classification. Being significantly shallower than the networks seen in the reviewed literature, I expect this network to perform poorly, particularly with age classification. Nonetheless, it will be included for initial comparison, and it could have moderate results with gender classification.

As it is such a shallow network (with 'only' ∼10,000 parameters), it will be limited to handling single-channel images at 32×32 size.

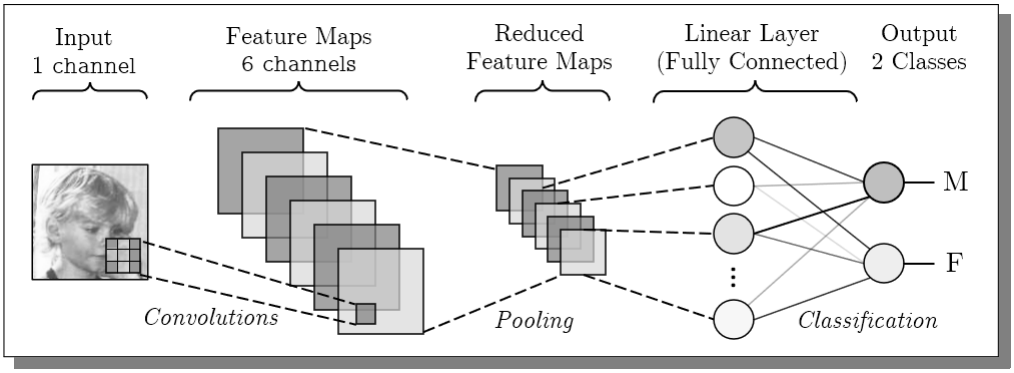


Figure 4.5: *Basic 2-layer CNN architecture diagram. Source: [6]*

LeNet

Architecture: $I_1 \rightarrow C_6 \rightarrow P \rightarrow C_16 \rightarrow P \rightarrow L_{120} \rightarrow L_{84} \rightarrow L_2$

LeNet-5, more commonly known as just LeNet, is a well-regarded architecture for image classification originally proposed over 20 years ago by LeCun et al., earning him a Turing Award for his contribution to deep learning [30, 47]. There are many variations of LeNet, including LeNet-1, LeNet-4 and LeNet-6 with various levels of depth and complexity. An explanation and comparison of some common LeNet variations is detailed by Sik-Ho Tsang in [48].

It is worth noting that LeNet was originally purposed for handwriting recognition and designed to take a grayscale input of 32×32, so it is possible that the architecture will fail with larger image sizes.

LeNet has a parameter count of ∼60,000.

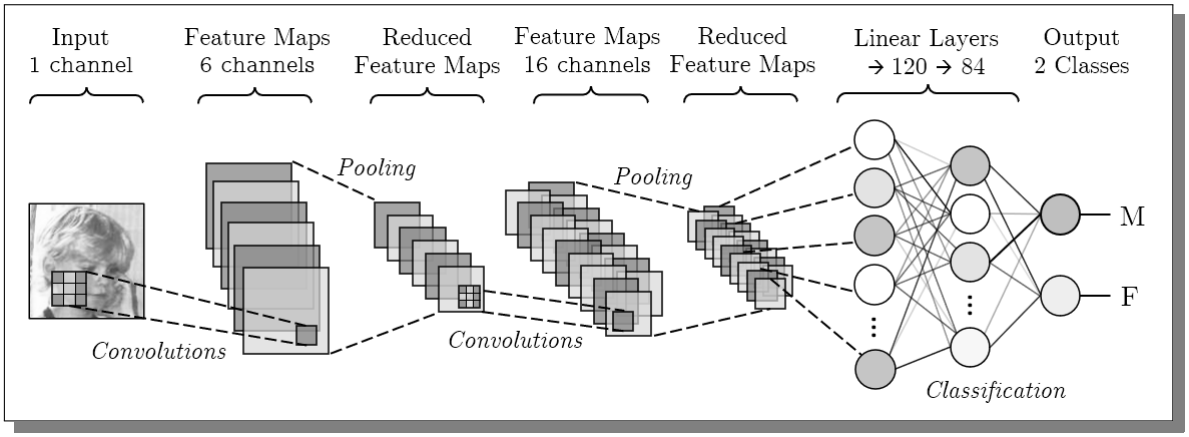


Figure 4.6: *LeNet-5 architecture diagram. Source: [6]*

AlexNet

Architecture: $I_3 \rightarrow C_{96} \rightarrow P \rightarrow C_{256} \rightarrow P \rightarrow C_{384} \rightarrow C_{384} \rightarrow C_{256} \rightarrow P \rightarrow L_{4096} \rightarrow L_{4096} \rightarrow L_2$

More sophisticated than LeNet, AlexNet [49] consists of 8 layers – five convolutional and three linear – with ∼60 million parameters, making it potentially better for more complicated image recognition tasks. It was a breakthrough network at the time, achieving state-of-the-art results in the 2012 ImageNet Large Scale Visual Recognition Challenge [50].

The PyTorch library provides the AlexNet model structure with customisable parameters, along with optional pre-trained initialisation weights, trained on the ImageNet database [51].

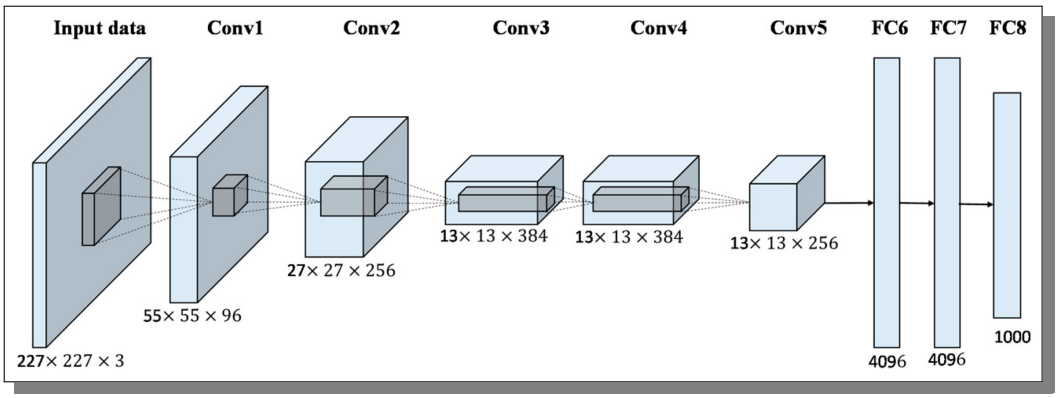


Figure 4.7: *AlexNet architecture diagram. Source: NeuroHive [7]*

VGG-16

Architecture: $I_3 \rightarrow C_{64} \rightarrow C_{64} \rightarrow P \rightarrow C_{128} \rightarrow C_{128} \rightarrow P \rightarrow C_{256} \rightarrow C_{256} \rightarrow C_{256} \rightarrow P \rightarrow C_{512} \rightarrow C_{512} \rightarrow C_{512} \rightarrow P \rightarrow C_{512} \rightarrow C_{512} \rightarrow C_{512} \rightarrow P \rightarrow L_{4096} \rightarrow L_{4096} \rightarrow L_2$

Even more complex than AlexNet, VGG-16 [**?**] consists of 16 layers in total, including 13 convolutional layers and three linear layers. Like with AlexNet, the PyTorch library provides a pre-trained VGG-16 model with customisable parameters [52].

It was seen utilised by Rothe et al. [26] in the reviewed literature to achieve state-of-the-art performance, using high-resolution inputs of 224x224. It has also been shown to provide accurate object detection for a range of image-analysis tasks [53].

It is worth noting that training with such an architecture may be difficult due to the incredibly high parameter count of ∼140 million, as well as the required high-resolution image input.

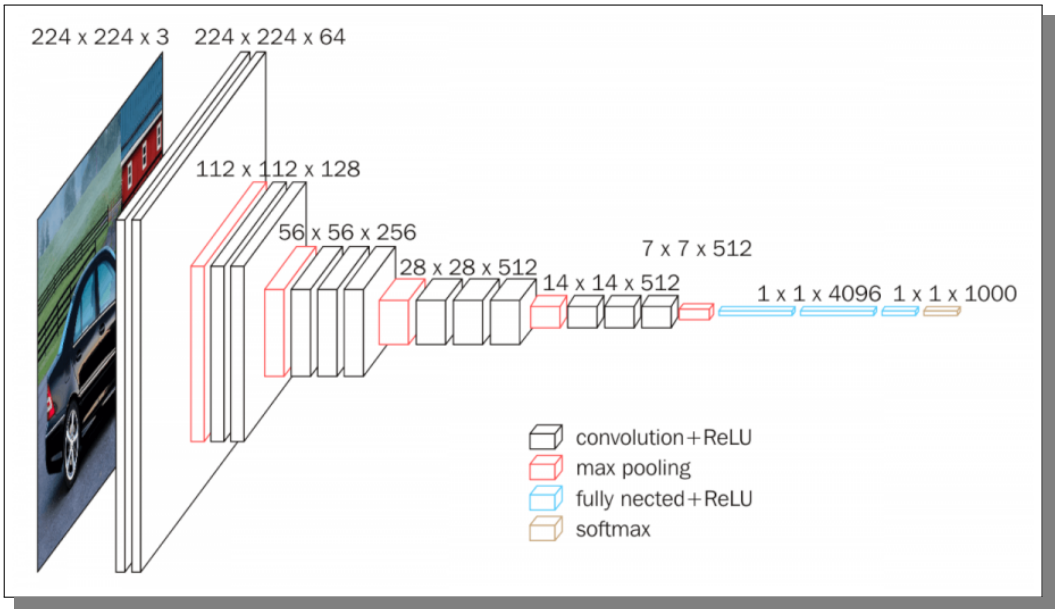


Figure 4.8: *AlexNet architecture diagram. Source: Han et al., 2017 [8]*

Table Summary of Proposed Architectures

Name	Input Size	No. of Layers	Architecture	No. of Parameters
BasicCNN	$32 \times 32 \times 1$	2	1 conv, 1 linear	∼10,000
LeNet [30]	$32 \times 32 \times 1$	7	2 conv, 3 linear	∼60,000
AlexNet [49]	$224 \times 224 \times 3$	8	5 conv, 3 linear	∼60 million
VGG-16 [54]	$224 \times 224 \times 3$	16	13 conv, 3 linear	∼140 million

Table 4.1: *Overview of Proposed Architectures*

Further Architecture Experimentation

Given sufficient time following the initial tests, a new architecture may be proposed to optimise results for the given tasks of this project. For example, if optimal results for age classification are obtaimed between LeNet and AlexNet, a network strecture could be utilised with 6 or 7 layers to find the middle ground and optimise training results.

4.5 Network Training

4.5.1 Weight Initialisation

Typically, networks are initialised with purely random weights and biases by the network creation framework. The benefit of using well-established architectures such as LeNet, AlexNet and VGG-16 is that they all have pre-trained models available online; in the case of the latter two, they are available within PyTorch itself [51, 52]. These weights are taken from models that have been successfully trained to a high accuracy on general-purpose image datasets.

While not essential, starting the training process with pre-established model weights can be highly beneficial as many of the trained feature extractors can transfer for another given task, even if seemingly unrelated. This can not only save training time and computational expense, but can also ensure the network has a solid basis for training, thus making it at less risk of overfitting.

4.5.2 Training Process

Once the weights are initialised, either using pre-trained weights as discussed or randomly by the framework, it's time to train the network. As detailed earlier in **2.4**, the typical training procedure involves 'running' the network layer-by-layer – the *forward pass* – and comparing its output with the expected output, as specified by the corresponding training entry label. The difference between these two values is used to update and optimise the network weights through a process known as *backpropagation*.

The exact procedure is more apparent in pseudo-code form:

Algorithm 1 Training Loop

Require: *TrainData*, *EpochLimit*

1: **for** *EpochCount* in 0 to *EpochLimit* **do**
▷ Training pass
2: *TrainLoss* ← 0
3: **for all** *Entry* in *TrainData* **do** ▷ Iterate through training data samples
4: Get *Image*, *Label* from *Entry*
5: *Output* ← ModelOutput(*Image*) ▷ Forward pass
6: *Loss* ← LossFunction(*Output*, *Label*) ▷ Calculate loss
7: UpdateWeights(*Loss*) ▷ Use loss to update weights
8: *TrainLoss* ← *TrainLoss* + *Loss*
9: **end for**
10: Print *TrainLoss*
11: **end for**

This gets iterated until the epoch limit is reached or over-fitting is detected. To minimise training time, networks are typically trained using 'batches' of data entries – typically 32, 64 or 128 entries at a time.

4.5.3 Training Hyperparameters

Hyperparameters refer to the values used in network training to specify the training process.

Optimiser and Learning Rate

The optimiser is the specific function which determines how a network's weights are updated after each pass. A comparison of the most commonly used network optimisers and learning rates is available at [55], in which they determined the 'Adam' optimiser with a learning rate of 0.0005 performed best for most cases; however, they advocate for experimentation for any given task, as the optimal function and learning rate is highly dependent on the task, network architecture, and training data.

The Adam optimiser function is included in PyTorch with a customisable learning rate parameter.

Loss Function

Loss functions are also provided with PyTorch. For classification issues, it is convention to use the 'cross-entropy loss' function, and for regression tasks it is typical to use either mean absolute error (MAE) or mean square error (MSE). MSE places more emphasis on larger network errors and is thus preferred for tasks like age prediction, where being inaccurate by a few years is far less important than being off by several decades.

As such, the cross-entropy loss and MSE loss functions will be utilised during for the project's training.

4.5.4 Overfitting Prevention

As previously described, overfitting occurs when a network's performance reduces on newly introduced data despite seemingly improved performance on the training dataset. It can be particularly problematic as it wastes time and resources while actually reducing the network's performance.

Validation Testing

To prevent overfitting, a segment of the training data – the 'validation' set – can be separated and reserved for separate tests during training.

By testing against the validation set after each epoch and comparing the network's performance, it can be ensured that training stops before any significant overfitting occurs. Testing against the validation data is done in the same way as the training data cycle; however, no backpropagation is performed this time, as the data entries are for testing only, and using them to update the network would 'reveal' them to the network, defeating the purpose of the validation tests.

The updated training cycle is as follows:

Algorithm 2 Training Loop

Require: *TrainData*, *ValidationData*, *EpochLimit*
1: **for** *EpochCount* in 0 to *EpochLimit* **do**
▷ Training pass
2: *TrainLoss* ← 0
3: **for all** *Entry* in *TrainData* **do** ▷ Iterate through training data samples
4: Get *Image*, *Label* from *Entry*
5: *Output* ← ModelOutput(*Image*) ▷ Forward pass
6: *Loss* ← LossFunction(*Output*, *Label*) ▷ Calculate loss
7: UpdateWeights(*Loss*) ▷ Use loss to update weights
8: *TrainLoss* ← *TrainLoss* + *Loss*
9: **end for**
▷ Validation test
10: *ValidationLoss* ← 0
11: **for all** *Entry* in *ValidationData* **do**
12: Get *Image*, *Label* from *Entry*
13: *Output* ← ModelOutput(*Image*)
14: *Loss* ← LossFunction(*Output*, *Label*) ▷ Evaluate only, don't change weights
15: *ValidationLoss* ← *ValidationLoss* + *Loss*
16: **end for**
17: Print *TrainLoss*, *ValidationLoss*
18: **end for**

Patience and Early Stopping

The loss values can be monitored during training to determine when to stop training, or alternatively, a 'patience' value can be used to automatically halt training after a certain number of epochs without improvement. This technique is typically referred to as 'early stopping'.

4.6 Datasets

The literature review highlighted the importance of data quality and quantity for neural network training. When selecting a dataset, it is important to consider the image quality, the label accuracy, sourcing methodology, and any potential biases which may be present in the dataset. It is also important to verify the capability of the dataset – for example, the outstanding performance demonstrated in [26] lends credence to the efficacy of the IMDB-WIKI dataset.

4.6.1 Adience (2014)

The Adience dataset [29], utilised by [25], consists of 26,580 images with 2,284 unique subjects. Each image is annotated with a gender label and one of eight age groups. The dataset is comprised of various real-world images, sourced from Flickr albums with considerable variation in lighting, pose, and image quality [**?**], making it a valuable addition to my training data. Although some of the images are group photos – which may be unsuitable for training – the dataset remains a valuable resource due to its emphasis on real-world, 'in the wild' images.

The age labels are provided as one of eight categories with the following ranges: 0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, and 60+.

As of writing this in January 2023, the Adience dataset is available for download at their website [56].

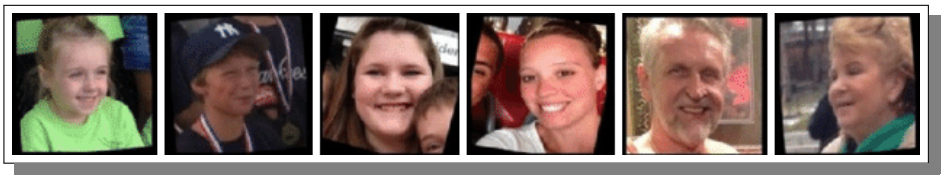


Figure 4.9: *Samples from the Adience dataset*

4.6.2 IMDB-WIKI (2016)

The IMDB-WIKI dataset [57] was introduced by Rothe et al. in 2016 [26], with the prospect of creating the largest and most diverse dataset for facial age and gender classification. The set contains automatically labelled images of over 20,000 subjects from

Wikipedia and IMDB for a total of over 520,000 images. The dataset, as well as their model obtained with it, are both freely available online for research purposes [**?**].

Since the dataset primarily consists of images of celebrities and actors, it is subject to certain biases. The subjects of these images may have heavy make-up or other cosmetic procedures with the specific intention of appearing younger, which could skew the predicted age of models trained with this. Lin et al. critiqued the dataset in their 2021 paper on age estimation [58], citing a high number of mislabelled entries, leading to worsened performance on networks. They proposed a 'cleaned' version of the dataset, aptly named IMDB-Clean [59], with newly provided annotations for the IMDB part of the set [60].

IMDB-WIKI was and still is the largest collection of faces labelled with true age and gender, making it a highly valuable resource for this project.

As of writing this in March 2023, the IMDB-WIKI dataset is available for download, for free given proper citation, at their website [56]. The cleaned IMDB metadata (IMDB-Clean) provided by [58] is available under the same terms.

Figure 4.10: *Samples from the IMDB-WIKI dataset*

4.6.3 UTKFace (2017)

The UTKFace [**?**] dataset is another large-scale face dataset with an extensive age span, ranging from 0 to 116 years old. The dataset contains just under 24,000 facial images with vast variations in pose, facial expression, illumination, occlusion, and resolution. Each image contains only one face, and is labelled by age, gender, and ethnicity. Like Adience, the images used in UTKFace are sourced from online with a focus on 'wild', non-standardised images.

The labels were generated automatically using the DEX estimation model from Rothe et al.[26], and then verified by a human. As such, the age labels are 'held back' by the accuracy of the DEX model.

The UTKFace dataset is available for download as of March 2023 at their website [56].

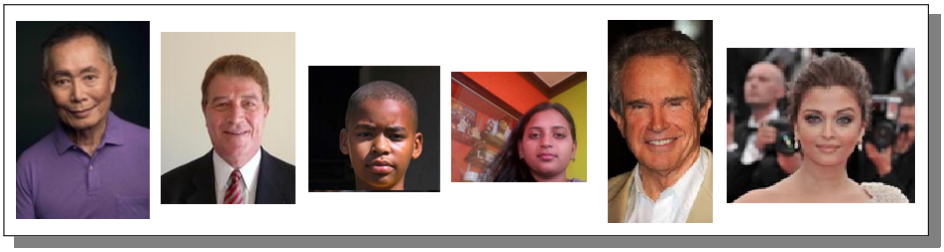


Figure 4.11: *Samples from the UTKFace dataset*

4.6.4 Ommitted Datasets

Some of the datasets cited in the reviewed papers were not suitable for use in this project, either due to redundancy or restricted access. These datasets include:

- The 2008 'MORPH-II' dataset is held behind a paywall, and thus is not included here.
- The 2009 'WebFace' dataset utilised by Yan et al. in [23] could not be found for download.
- The 2010 'FRGC v2.0' dataset utilised in [10] could not be found for download.
- The 2002 'FG-NET' dataset used by [26] could not be found for download.
- The 2009 'Images of Groups' dataset used by [25] is made redundant by the 2014 'Adience' dataset with the same sourcing methodology.
- The 2014 'CACD' dataset cited by [26] is made redundant by the 2016 'IMDB-WIKI' dataset with the same sourcing methodology.
- The 2021 'VMAGE' dataset [61] has been taken down following legal risks from the automatically web-crawled images. This dataset promised to be the largest dataset available for gender and age classification with over 3.3 million labelled images.

4.6.5 Table Summary of Datasets

Dataset	Labels	Images	Subjects	Image Sources
Adience (2014) [29]	Gender (2 classes), Real age (8 classes)	26,580	2,284	Random user-uploaded images from Flickr
IMDB-WIKI (2016) [57]	Gender (2 classes), Real age as given by IMDB or Wikipedia	523,051	20,284	Various celebrity photos from IMDB/Wikipedia
UTKFace (2017) [5]	Gender (2 classes), Age estimated by DEX algorithm	23,700	?	Web-crawled facial images

Table 4.2: *Summary of available datasets*

4.6.6 Dataset Choice

Training Data

The gender model will initially be trained using a mix of Adience, IMDB-WIKI, and UTKFace as these all provide accurate labels for gender. The age model will be trained without the Adience dataset as the age group labels are not suitable for regression model training.

Different combinations of the datasets may be experimented with during the implementation to assess their individual and combined effectiveness in training.

Validation Data

A subsection of the training datasets will be automatically removed and utilised for validation testing in an attempt to prevent over-fitting, as detailed in **Section 4.3.3**.

Testing Data

For testing purposes, 500 images from each dataset will be randomly selected and removed from each of the above training sets. These images can be used to evaluate the final models on data which is completely unseen to the network's training process.

4.6.7 Dataset Storage

The datasets will be separated into distinct `training` and `testing` folders, with the entries of each dataset contained in its own folder. Each image file will have its filename in the form `gender_age_rand.jpg`, where:

- `gender` is the gender as either 'M' or 'F';

- `age` is the age to the nearest whole year, or as supplied by the dataset;
- `rand` is a unique (but arbitrary) number used to avoid filename conflicts.

This will allow for easy label reading, as well as allowing each dataset to be used for both gender and age model training if desired.

4.6.8 Data Bias and Augmentation

Data Equalisation

Reducing data bias is crucial for developing a robust and accurate prediction model, as emphasized by [10]. For instance, IMDB-WIKI contains 310,277 male and 201,669 female images, which may result in a model with bias towards men – by 'seeing' more male subjects than female, the network is essentially incentivised to be biased towards male predictions, as this would have favoured results during training. In addition, age distribution was largely skewed to those aged 30-40 years, as shown in the graph displayed on their website (*Figure 4.12*), which poses a similar risk of prediction bias.

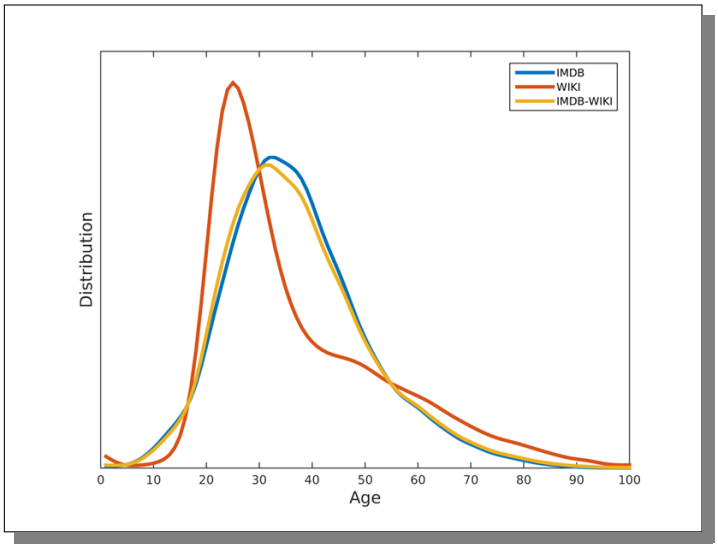


Figure 4.12: *IMDB-WIKI age distribution graph. Image source: [9]*

This issue can be mitigated by implementing a dataset equaliser to assign a consistent number of data entries to each class for training, ensuring that the network is guided by the details of the entries themselves, as opposed to the classes' levels of representation.

More advanced bias reduction, such as equalising datasets by ethnicity, could further improve model robustness and guarantee fair predictions for different ethnic groups. However, this approach depends on having accurate ethnicity labels, which only one of our chosen datasets (UTKFace) provides.

Data Augmentation

Data augmentation is a technique used to expand and diversify training datasets by applying various transformations – such as rotations, translations and flips – to the original images, increasing variability of training data. It can also aid data equalisation by reducing the number of duplicate images in the dataset when ensuring consistently high entries per class, potentially leading to a more accurate model.

However, entries which are augmented poorly or unnecessarily can reduce the dataset quality, leading to worse end performance. Given the substantial combined size of the chosen datasets (with 300,000+ entries), it is unlikely to be necessary for our purposes, but may be incorporated later if deemed beneficial for training performance.

4.7 Experimental Techniques for Model Performance

There are a few techniques which could be utilised to potentially improve performance after getting the initial network results.

4.7.1 Multi-Stage Processing

Multi-stage processing involves using the outputs of one network to aid in the classification of another network [**?**]; for example, a two-stage process was implemented in [23], as the results of the gender classification network determined the use of either a male or female network for the final age estimation.

Attention Mechanism

The attention mechanism utilised in [25] would be another example of multi-stage processing, as the results of their attention ('patch decider') network was used as input to the following patch analysis network.

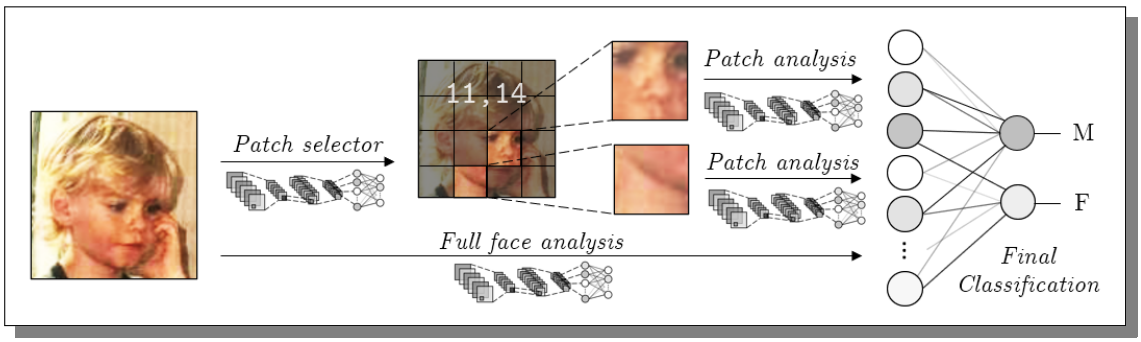


Figure 4.13: *Diagram of grid attention model, originally proposed by Huerta et al. [10]. Image source: [6]*

Following from the idea of Rodriguez et al.'s grid-based attention mechanism, an additional

'Variable Attention' mechanism is proposed in this project to allow patches of any position and size to be analysed, as opposed to being restricted by a square grid. This additional degree of freedom could potentially enable more nuanced estimations, leading to improved classification results. The adjustable patch size is loosely analogous to the way human eyes change depth focus during eye fixations – rather than being limited by a consistent depth of focus (patch size) as their model imposes.

However, it is possible that the network will not learn how to utilise the added layer of freedom, leading to poor training performance.

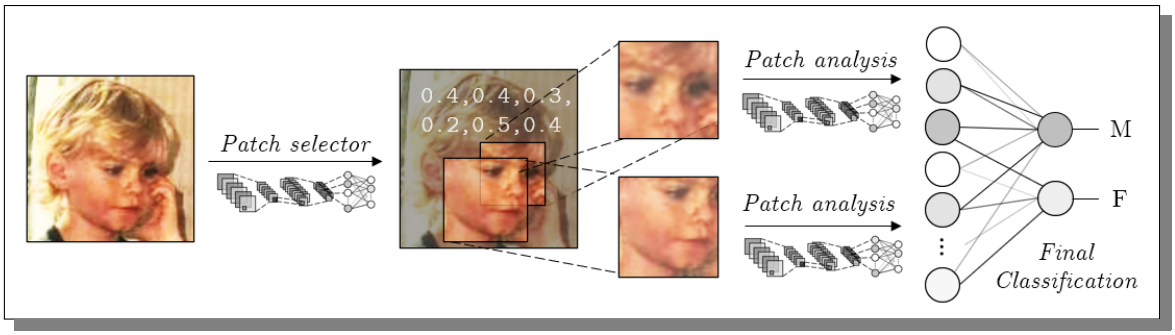


Figure 4.14: *Variable patch model diagram*

Both of the above methods may be experimented with as potential ways to boost network performance.

4.7.2 Model Stacking

Model stacking is the use of multiple networks in conjunction, perhaps trained on the same or different datasets, to form a final classification [62]. This typically involves feeding all selected networks the same input and combining their respective outputs using some form of output averaging function, or an additional classification layer as shown in *Figure 4.15*.

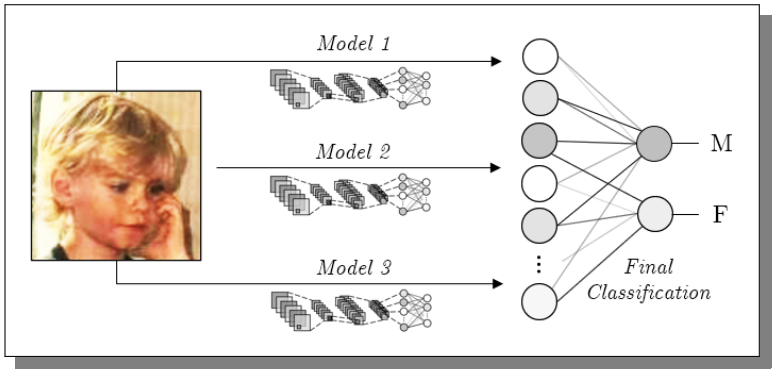


Figure 4.15: *Model stacking diagram, featuring three LeNets networks 'stacked' to form an overall classification*

The effectiveness of this technique depends on the task at hand and the reliability of the models. It is possible that combining models in this fashion could yield worsened results if the predictions of one/several models significantly interfere with those of others; conversely, it could allow for improved classification results as the models can collaborate and fill the gaps in each other's predictive capabilities.

Model stacking has been shown to be beneficial in image recognition tasks before [48], and thus is worth experimenting with if there is sufficient time to do so..

4.7.3 Image Quality Enhancements

In recent years, breakthrough image enhancement models such as GFPGan [63] have demonstrated amazing enhancement capabilities for the recovery of poor-quality images. These enhancement models can increase image resolution, reduce noise, and provide a more detailed input for the prediction network to analyse.

However, there are potential drawbacks to using such models, such as the potential for inaccurate details to be added. Additionally, implementing these techniques may reduce processing speed and computational efficiency, leading to longer training times for the network.

As with the previous experimental techniques, image quality enhancements may be explored if time permits, allowing for an evaluation of the trade-offs between artificial image quality improvements and their impact on the end-model performance.

4.8 GUI

The proposed GUI (see *Figure 4.16*) is somewhat minimalist, indicating the predicted age and gender of an individual with a simplistic box indicative of their detected face position in the image. This will also help differentiate between individuals in input where multiple faces are present, with separate predictions required for each subject.

4.9 Methodology Overview

A visual summary of the methodology is depicted below (*Figure 4.16*).

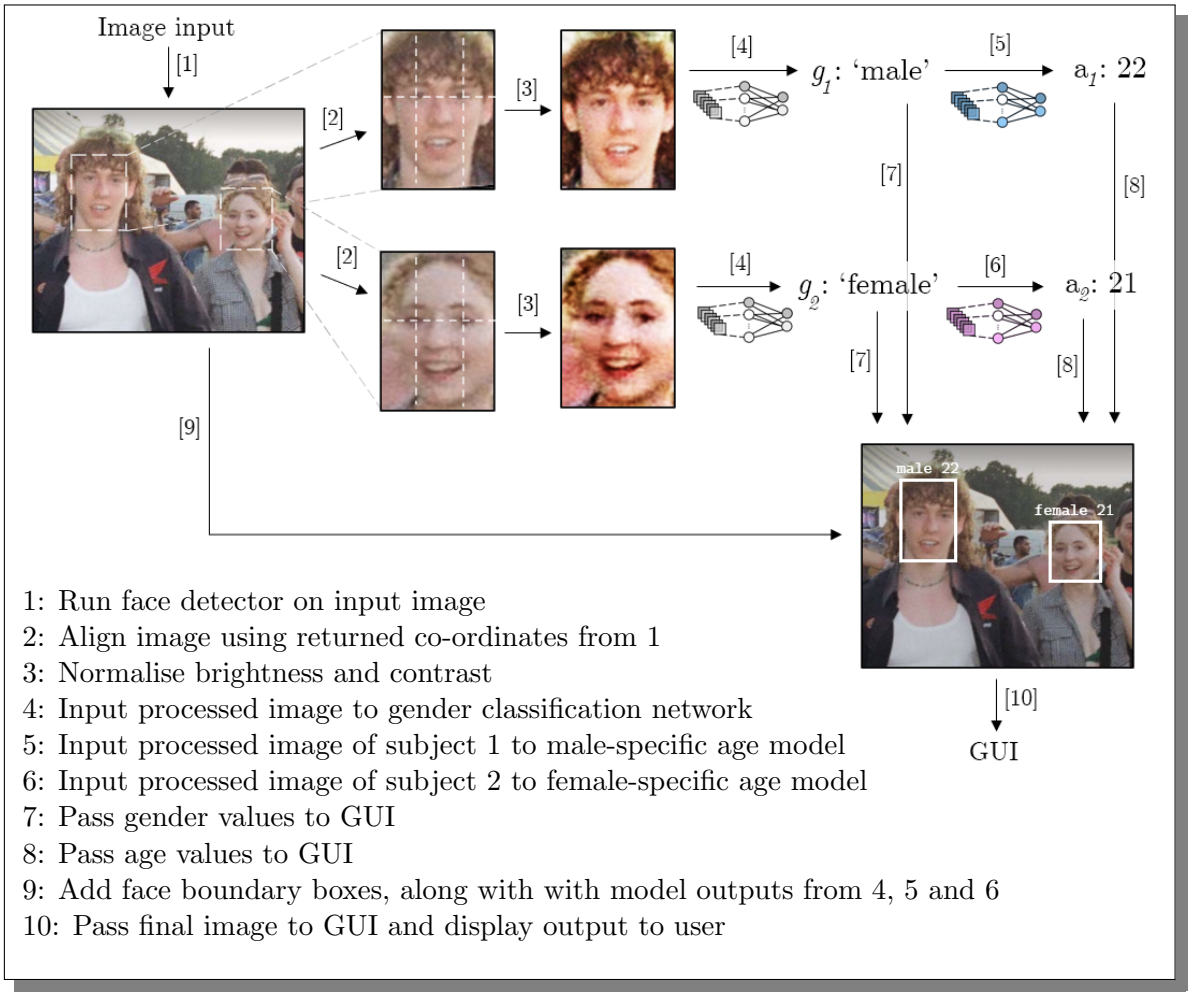


Figure 4.16: *Model overview using own image (subjects agreed)*

4.10 Testing Methodology

As mentioned, separate datasets will be produced for final testing of the network against human-verified subject photos. A completely separate set of images, including group images, portraits and other images to mimick real-world use will be utilised for further insight into the final model's efficacy.

These tests differ from the validation tests performed during training, as they are the final indication of the network's performance, and thus are standardised to a far higher degree than the randomly-chosen validation data during training.

4.10.1 Gender Classification Test

Gender Classification Accuracy Algorithm

For testing gender classification, a simple binary classification test like below is suitable as below.

$$\text{Accuracy} = \frac{\text{No. of correct predictions}}{\text{No. of tests conducted}}$$

The testing process to achieve this value looks like the following:

Algorithm 3 Classification Test Algorithm

Require: *TestData*
Tests ← 0
TestsCorrect ← 0
for all *Entry* in *TestData* **do**
    Get *Image*, *Label* from *Entry*
    Prediction ← ModelPrediction(*Image*)
    Tests ← *Tests* +1
    if *Prediction* = *Label* **then**
        TestsCorrect ← *TestsCorrect* +1
    end if
end for
Accuracy ← *TestsCorrect*/*Tests*
Print *Accuracy* ∗ *100* ▷ Get accuracy as percentage

This would generate a percentage accuracy score of the model's performance; for instance, a score of 95% would indicate the model was correct in 95% of all gender prediction tests, whereas a score of 50% would indicate a model with similar predictive capabilities to a coin-toss.

Confusion Matrix

A confusion matrix sorts cases from the model test into categories, enabling the accuracy for each class to be more easily analysed. An example of this for gender classification is below.

		Model Prediction	
		Male	Female
Label	Male	a	b
	Female	c	d

a = No. of correct male predictions
b = No. of male predictions when label was female
c = No. of female predictions when label was male
d = No. of correct female predictions

Figure 4.17: *Confusion matrix for gender model evaluation*

4.10.2 Age Prediction Error Algorithm

Mean Absolute Error

For regression tasks or classification tasks with a continuous range like age estimation, it makes more sense to use MAE testing, as seen in the evaluations of many of the literature reviews. It is perhaps more intuitive to understand the error of a regression network from its MAE value as opposed to using a category confusion matrix, however both are possible methods for evaluating a regression netowkr.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Algorithm 4 Mean Absolute Error Test Algorithm

Require: *TestData*
Tests $\leftarrow 0$
TotalError $\leftarrow 0$
for all *Entry* in *TestData* **do**
    Get *Image*, *Label* from *Entry*
    Prediction $\leftarrow$ ModelPrediction(*Image*)
    Tests$+= 1$
    TotalError$+=$ *Absolute*(*Label* $-$ *Prediction*)
    if *Prediction* $=$ *Label* **then**
        TestsCorrect$+= 1$
    end if
end for
MAE $\leftarrow$ *TotalError* / *Tests*
Print *MAE*

Confusion Matrix

By segmenting the different predictions and labels into age groups, a confusion matrix similar to that to above can be implemented to evaluate age predictions:

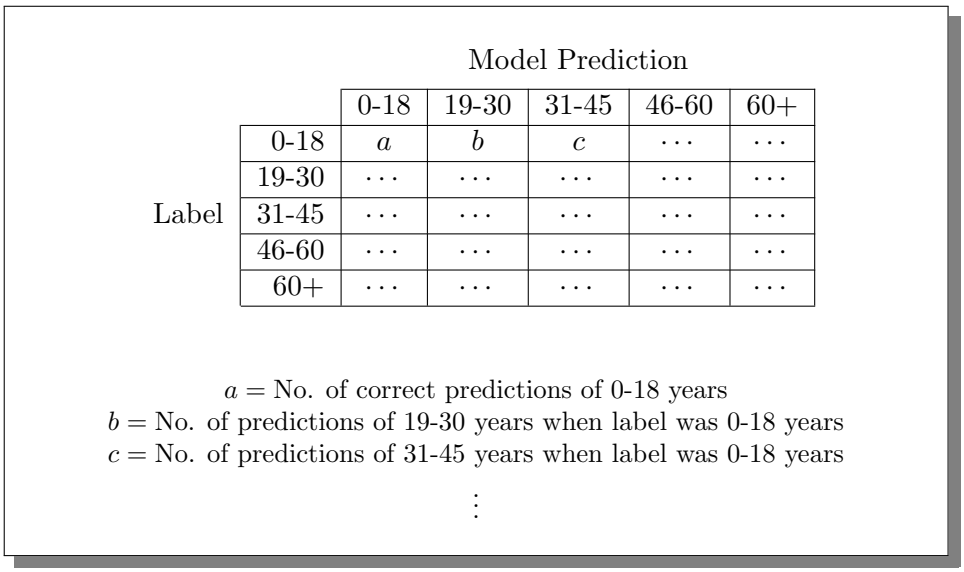


Figure 4.18: *Confusion matrix structure for age model evaluation*

4.11 Limitations

It is important to consider the expected limitations of the model produced from this project.

4.11.1 Methodological Limitations

The methods used in this project may present certain limitations that inherently limit the tool's capacity and usefulness. Some key issues include:

- As with all image classification models, the input quality (i.e. camera resolution) will largely impact the accuracy of the model;
- As the detection is entirely based on subject's facial appearance, any facial occlusions or facial irregularities (e.g. acne, hyper-pigmentation) can reduce model performance;
- Training the network using datasets found online poses the risk for certain biases to be learned by the network;
- By being purely based on facial images, non-related visual cues such as height and stature are ignored despite being potentially good predictors.

4.11.2 Hardware Limitations

Given the processing power constraints, the models produced in this project are unlikely to reach the level of performance shown in the reviewed literature.

Experimentation with large network architectures like AlexNet (8 layers) and VGG-16 (16 layers) may be too computationally demanding or require substationally lengthy training times. Using the full proposed dataset might prove unfeasible on our development machine, and may even be too computationally expensive to perform with Google Colab.

Chapter 5: Implementation

This chapter covers the implementation and development of the proposed system using the methodologies from the previous chapter.

5.1 Prerequisites

This section covers the brief software prerequisites for the project's development.

5.1.1 VS Code and GitHub

VS Code and GitHub were already installed on the machine prior to development of this project. A repository was created via the GitHub website and cloned on the development machine such that all changes would remain syncronised.

5.1.2 Conda, Python, and Relevant Packages

Conda was downloaded and installed from the official website [**?**].

A new virtual environment was created including Python 3.10 and the relevant packages from – namely PyTorch, Torchvision, NumPy and OpenCV – via the Command Prompt. The environment was then selected within VS Code (see *Figure 5.1*) such that all of the project's development could progress within the contained environment.

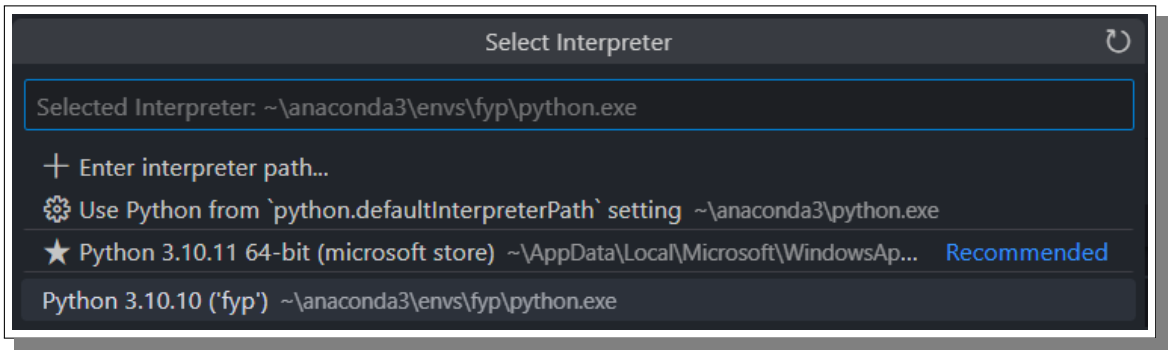


Figure 5.1: *Selecting the new Anaconda environment in VS Code*

5.2 Image Pre-Processor Implementation

While not strictly necessary for network training, the pre-processor is one of the most important components of the model, bridging the image entries of the datasets to the networks. As specified in the Methodology chapter, this module extracts faces from images in a standardised form such that any image input – from the dataset, or from real-world use-cases – appears similar enough to the network to provide a robust prediction, while highlighting the relevant differences between images such that an accurate classification can be made.

5.2.1 Reading images with OpenCV

OpenCV was used to read images into memory using the `cv.imread` function. The image can then be displayed with `cv.imshow`, which results in the window shown in *Figure 5.2*.

```
import cv2 as cvz

# Load and display image with OpenCV
image = cv.imread('me.jpg')
cv.imshow('CV imread test', image)

# Stop window closing itself
cv.waitKey(0)
```

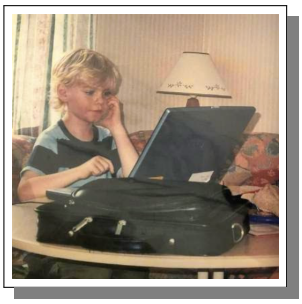


Figure 5.2: *OpenCV image reading test. Image source: [6]*

5.2.2 Brightness/Contrast Normalization

OpenCV also includes the `cv.equalizeHist` function [**?**] which performs histogram equalisation on given image channel, effectively normalising the brightness and contrast in one step. To equalise all 3 channels of an RGB image, the function gets applied to each channel individually before merging them back together (as suggested in [64]). An example of this process is shown in *Figure 5.3*.

```
                    ...
r, g, b = cv.split(image)
r_eq = cv.equalizeHist(r)
g_eq = cv.equalizeHist(g)
b_eq = cv.equalizeHist(b)
image_eq = cv.merge((r_eq, g_eq, b_eq))
cv.imshow('equalised', image_eq)
```

Figure 5.3: *OpenCV split-channel equalisation test: original on left, equalised on right*

5.2.3 Face Detection

Following brightness/contrast normalisation, the next step is to detect any faces in the image. The detector returns the coordinates necessary to align and crop the face, removing irrelevant background details while mitigating variance in face rotation.

To do this, the `FaceDetectorYN` module provided with OpenCV was used as determined in the previous chapter with a pre-trained detection model by Shiqi Yu [65]. Using a pre-trained detection model greatly reduced the amount of code and time necessary to implement the image pre-processor.

Using the relevant documentation from the OpenCV website, a basic implementation of the detector was implemented as follows:

```
# Init detector
detector = cv.detectorYN.create(···)

image_w, image_h = image.shape[1], image.shape[0]
detector.setInputSize((image_w, image_h))

# Get face coords
every_face_coords = detector.detect(image_eq)[1]
first_face_coords = every_face_coords[0]
print(first_face_coords)
```

Running this code on the same image as before, we get the following output:

Terminal

```
[ 90.02575    84.47276    64.25879    77.70696   120.27142
115.98464   142.83359   115.55218   136.3779   130.24591
120.36769   142.83272   139.977     142.6604   0.9998304 ]
```

For each detected face, an array is returned with 15 numbers: the first four are the coordinates of the face's bounding box (*Figure 5.4*), and the following 10 are coordinates of various facial landmarks including the eyes, nose and mouth.

The 15th and final number is the model's confidence score, roughly indicative of the likelihood that the given face is actually present in the image. In this example, a score of over 99.9% is returned, indicating an extremely high confidence that a face is present. The model omits faces with a confidence score lower than 90% by default, configurable by the `setScoreThreshold` parameter when initialising the detector.

The face coordinates can be represented on the image using `cv.rectangle` and `cv.circle` [**?**]; *Figure 5.4* shows the face detection model successfully identifying the face in the image along with the relevant landmarks.

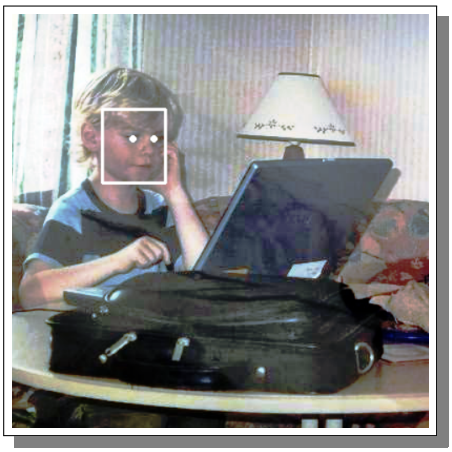


Figure 5.4: *OpenCV face detection test with single subject*

This was updated for multi-face detection with a `for` loop, iterating the OpenCV drawing functions over each face array. The example in *Figure 5.5* shows the detection model successfully identifying all 9 faces from a sample image of both male and female subjects.

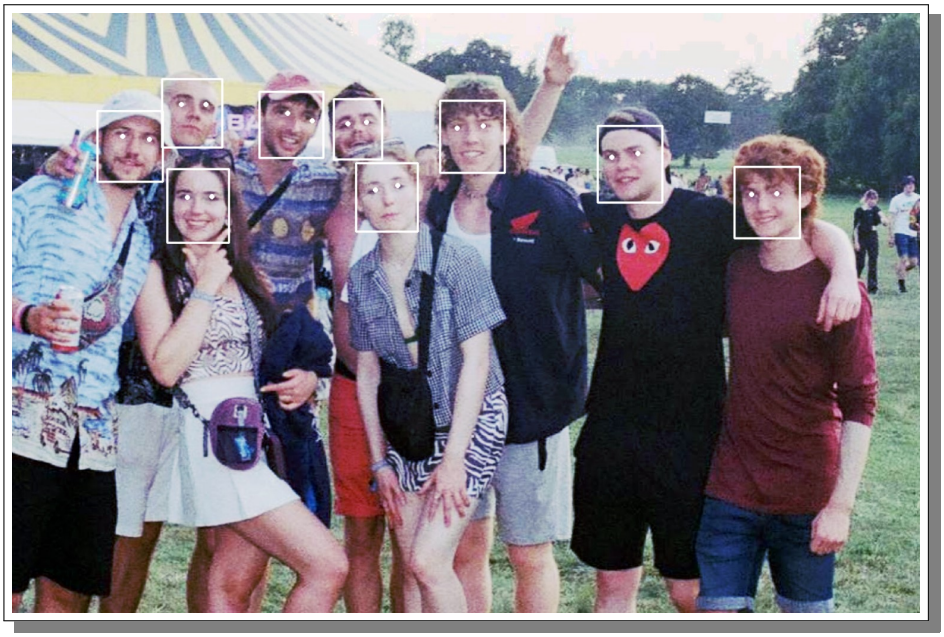


Figure 5.5: *OpenCV face detection test with large group photo. Image source: [6]*

5.2.4 Face Cropping and Alignment

Since the face detection system returns the co-ordinates for a facial bounding box, a basic pre-processor module could be made to crop to the given face boundaries using OpenCV array slicing [66]. However, it is more efficient to combine the face cropping and alignment into a single process, like proposed in **Section 4.3.4**; a series of linear algebra operations is used to achieve this, following the same process and alignment specifications as intended.

Implementation of the Eye-Based Alignment System

The alignment process was implemented by utilising a mix of trigonometry and matrix operations along with OpenCV's matrix-based image manipulation functions as seen in the code snippet below:

https://github.com/jckpn/age-gender-cnn/preprocessors/eyealign.py

```
                                        . . .
# Customisable alignment params
dest_w = 224              # Width of final image
dest_h = 224              # Height of final image
eye_y = 0.5*dest_h        # Y coord of both eyes in final image
eye_x = 0.25*dest_w       # X coord of left eye in final image
                          # (right eye gets calculated later)
                                        . . .
def eye_align(image, face_coords):
```

```
    # calculate transformation matrix from above parameters:
    # get translation
    left_eye_x = eye_x
    right_eye_x = dest_w-eye_x
    translate_x = left_eye_x - face_coords["left_eye_x"]
    translate_y = eye_y - face_coords["left_eye_y"]

    # use sohcahtoa to get angle of rotation for eye alignment
    # --> theta = tan-1 ( opp / adj )
    opp = face_coords["right_eye_y"] - face_coords["left_eye_y"]
    adj = face_coords["right_eye_x"] - face_coords["left_eye_x"]
    theta = math.atan(opp/adj)

    # get scale factor
    scale = (left_eye_x - right_eye_x) / (face_coords["left_eye_x"] -
                                          face_coords["right_eye_x"])

    # create matrix
    align_matrix = cv.getRotationMatrix2D(center=(int(face_coords["left_eye_x"]),
    ↪  int(
        face_coords["left_eye_y"])), angle=math.degrees(theta), scale=scale)

    # add transformation coords to rotation matrix
    align_matrix[0][2] += translate_x
    align_matrix[1][2] += translate_y

    # apply matrix transformation to image

    image = cv.warpAffine(image, align_matrix, (dest_w, dest_h))
    return image
                                        . . .
```

By implementing the alignment process in this manner, it allows for full customisability over the specifications of the crop, enabling us to later experiment with various crop margins. The `dest_w` and `dest_h` parameters can be changed to any desired size, and the `eye_x` and `eye_y` parameters can be changed to position the eyes within the image as desired. The co-ordinates of the right eye are calculated to be the same distance from the right edge of the image as the left eye is from the left edge to ensure symmetry.

Initial Tests and Adjustments to Alignment System

The `eye_y` value was initially set to 0.5, or half of the image height, as assumed to be a common position for the eyes to be in a facial image; likewise, the `eye_x` parameter was set to a quarter of the width of the image, as this was assumed to be a common position for the left eye to be in a facial image.

Running the new pre-processor function with the same input image as before returns the following facial images, all with consistent size and horizontally aligned eyes as planned:

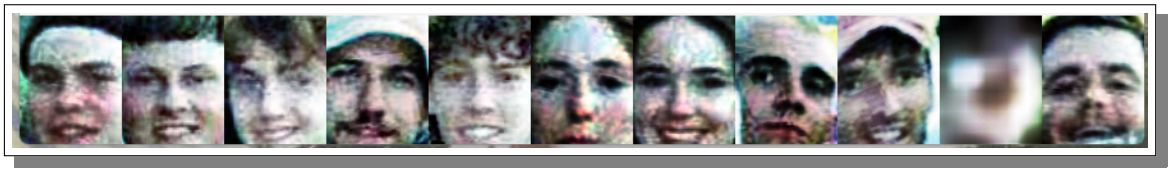
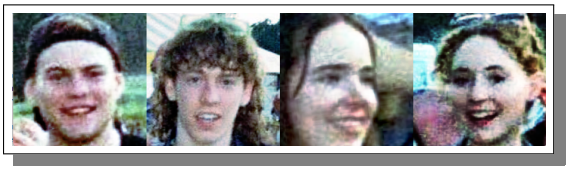


Figure 5.6: *Initial test of eye alignment implementation*

It was apparent from initial testing that the values for `eye_x` and `eye_y` resulted in a fairly zoomed crop of the subjects, ommitting much of the details like the chin and hair. Through further experimentation it was determined that respective values of 0.38 and 0.47 aligned images to the entire head with a suitably minimal margin (see *Figure 5.7b* below), and values of $0.30, 0.35$ provided a more 'focused' crop on the subject's face, occluding the hair and jawline (*Figure 5.7a*). Both crops were tested in the later experimentation section to determine which provided the best results.

(a) *Revised 'face' crop*
($eye\_x = 0.30, eye\_y = 0.35$)



(b) *New 'head' crop*
($eye\_x = 0.38, eye\_y = 0.47$)

Figure 5.7: *Comparison of pre-processor crop values*

It's also worth noting the flaws of the face detector became apparent from this testing, such as seen in *Figure 5.8*; we see that the face detector has incorrectly positioned the eyes, perhaps indicating a limit to the face detector's capabilities when a large number of subjects are involved. Such errors may interfere slightly with the alignment process, but should not be so significant as to affect the end results of the network.

Figure 5.8: *The eye alignment fails somewhat for subject 9*

Also note that no face cropping is explicitly applied during the alignment process – it instead occurs indirectly through the eye alignment. If the faces were cropped using the face boundaries, differently shaped heads may have the eyes in different relative positions; as [26] determined that eyes were the most important part of facial image classification, this would be undesirable. By aligning the eyes, we ensure that the eyes are always in the same position relative to the image, regardless of the shape of the head.

5.2.5 Transforms

Although incorporated separately to the pre-processor module, the image 'transforms' provide a further level of image standardisation for the network.

The PyTorch `torchvision` package provides various relevant functions for transforming images, such as normalisation and resizing. These transforms are applied to the images as they are loaded into the network, following the pre-processing module.

Due to the differing inputs required by the various networks, two transform processes were implemented: one for the single-channel networks (BasicCNN and LeNet), and another for the 3-channel networks (AlexNet and VGG-16).

Grayscale Transform – For BasicCNN and LeNet

The transform for use on the single-channel networks was implemented as follows:

1. Resize to given size (default 32×32)
2. Apply grayscale to convert from RGB $\rightarrow$ single channel
3. Convert to PyTorch tensor
4. Normalise: this converts the image contents from values of range $0 \rightarrow 255$ to the range $0 \rightarrow 1$, while standardising the pixel values such that their average is 0.5 with a standard deviation of 0.5.

A default image size of 28 was chosen as specified by the creator of LeNet-5 in the initial publication [30]; however, the size parameter was left customisable for easy experimentation during the grid search.

RGB Transform – For AlexNet and VGG-16

The transform for the other two architectures was implemented similarly, with the main difference the lack of a grayscale conversion step:

1. Resize to given size (224x224 as suggested in PyTorch docs [51])
2. Convert to PyTorch tensor
3. Normalise with suggested RGB values from PyTorch docs: `mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]`

The 'suggested values' are based on the values suggested in the PyTorch documentation, based on the values used when originally training the models. Note that these values will be used regardless of whether the pre-trained weights are chosen for fairer comparison between the weight initialisation processes.

5.2.6 Further Developments to Pre-Processor

The pre-processor module went through a few iterations; the main alterations were as follows:

- Faces which are too small (i.e. under 28px width) are discarded as they are unlikely to be suitable for training or contain enough detail for accurate predictions during testing.
- Faces which fail the alignment process (e.g. due to the face detector failing or overlapping eye landmarks) are also discarded.
- The algorithm was also updated to normalise the brightness and contrast of each facial image, as opposed to normalising the entire input image. This further ensures that the input images have similar colour profiles, presumably aiding the model more than having the entire input image normalised.

The full code for the pre-processor module can be found at
`https://github.com/jckpn/age-gender-cnn/preprocessor.py`.

5.3 Dataset Preperation

5.3.1 Adience

The official Adience website [56] provides connection details for a server containing the dataset, but during development this server was not available.

Fortunately, a copy of the dataset was able to be obtained via the *Deep Lake* Python library [67]. Once installed with Pip, the Adience dataset was downloaded with the following short script:

```
import deeplake
dataset = deeplake.load('hub://activeloop/adience')
```

The dataset entries can be loaded and viewed using the `cv.imshow` function from OpenCV, using the code provided by the Deep Lake documentation [68]. Doing so revealed that the labels were returned as 'm' or 'f' for the gender, and the corresponding age range (e.g. 8-12, 12-20, ...) was also provided.

Although not intended for this project, the images from this copy of Adience were pre-cropped and pre-aligned; however, as the crop is fairly wide, it should not severely interfere with the cropping of our own pre-processor later. Some examples of the images can be seen in *Figure 5.9*.

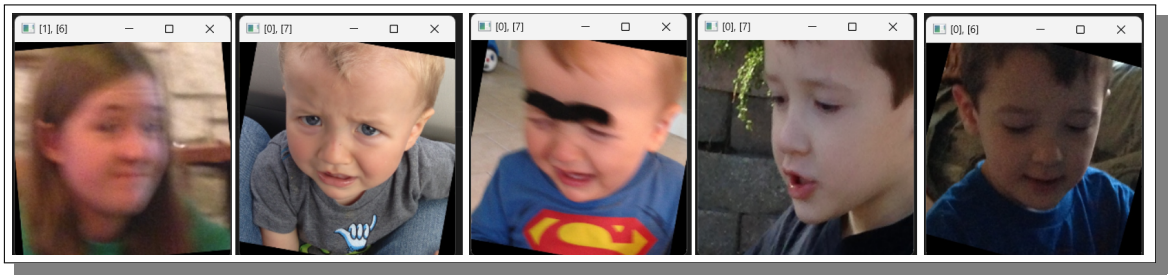


Figure 5.9: *Example images from the Adience dataset as downloaded from Deep Lake*

Although Deep Lake allows for network training directly from its cloud-stored datasets, Adience was downloaded locally for consistency with the other datasets. This was done using another short Python script, utilising `cv.imwrite` to download and save the entries while renaming them to the desired `gender_age_rand.jpg` format:

```
                                        . . .
for i in range(len(tensors['images'])):
```

```
    # convert image tensors to cv images and labels to str
    gender = tensors['genders'][i].data()['text'][0]
    gender = gender.upper()
    age = tensors['ages'][i].data()['text'][0]
    image = tensors['images'][i].numpy()
    image = cv.cvtColor(image, cv.COLOR_RGB2BGR) # convert from BGR to RGB
    age = age.replace('(', '').replace(')', '').replace(', ', '-')
    # `(8, 12)` --> `8-12`
    fname = f'{gender}_{age}_{i}.jpg'
    cv.imwrite(os.path.join(save_dir, fname), image)
```

Once downloaded, the full dataset occupied around 1.16 GB of disk storage.

5.3.2 IMDB-WIKI

The Wikipedia and IMDB sections of this dataset are offered individually at the publisher's website [9]. The option was given for the original uncropped images, totalling 272 GB, or pre-cropped images occupying significantly less space at 8 GB.

The latter option was chosen for this project as the hugely reduced dataset size allowed for faster processing and training, and the 40% cropping margin was sufficient to avoid interference with our own pre-processor. This also reduced the chance of images containing multiple faces, which could have presented issues later in the training process. Each portion was provided as a compressed `.tar` folder, which were extracted with 7zip [69].

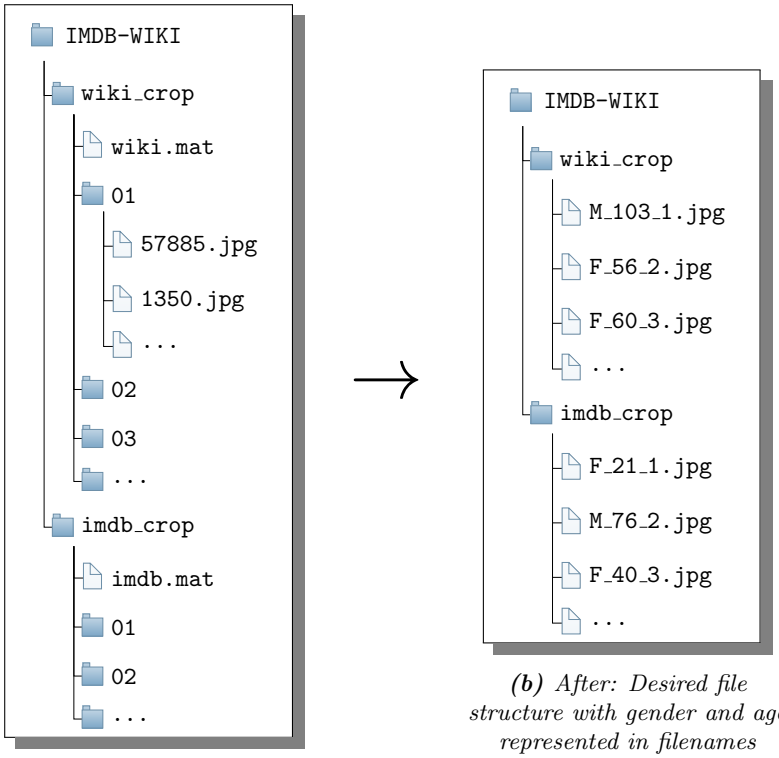
Restructuring `wiki_crop.tar`

Following extraction, it was noted that the filenames did not yet correlate to any useful age or gender labels; instead, a separate .mat metadata file was provided with all relevant information. Using MATLAB [70] to open the metadata file revealed the following details for each image:

- A binary gender label (`0` for female, `1` for male),
- The individual's date of birth (`dd-mm-yyyy`),
- The year the photo was taken (`yyyy`), and
- Additional information, such as face confidence scores, which were not relevant for this project.

By assuming the photo date to be mid-way through the given year, the yearly age of each subject could be calculated using a short MATLAB script supplied on the IMDB-WIKI download page [9].

Following extraction, the dataset had the file structure displayed in *Figure 5.11a*, with arbitrarily named image files divided into 100 folders – aother MATLAB script was written to obtain the desired file structure (*Figure 5.11b*), by iteratively reading each image's metadata, computing the subject's age, and renaming the file as required.



(a) *Before: Random filenames with arbitrary folder structure*

(b) *After: Desired file structure with gender and age represented in filenames*

Figure 5.10: *Renaming and restructuring the IMDB-WIKI dataset*

The Wikipedia section of the dataset was now ready for use, but the IMDB section required further restructuring using the updated annotations from IMDB-Clean.

Restructuring and Cleaning `imdb_crop.tar` with IMDB-Clean

Instead of using the provided `.mat` file for the IMDB portion of the dataset, the annotations provided by the IMDB-Clean repository [58] were used; these annotations promised superior accuracy to the ones originally provided, using the data cleaning methodology proposed by Lin et al., 2021 [**?**]. A Python script was written to iterate through each entry and rename the image file similarly to the previous sections.

This process reduced the dataset from over 460,000 entries to under 290,000, a significant 38% reduction in size – the presumption is that removing such entries improved the dataset's quality despite the major drop in size.

The IMDB-WIKI dataset was now fully prepared and split into two folders, `imdb_crop` and `wiki_crop`, with each image filename in the desired form (*Figure 5.11b*).

From hereon, IMDB-Clean refers to the IMDB portion of IMDB-WIKI as cleaned using the above process; IMDB-WIKI refers to both IMDB-Clean and the Wikipedia portion of IMDB-WIKI. The original annotations for the IMDB portion are not utilised in this project at any point.

5.3.3 UTKFace

As with the previous datasets, UTKFace was obtained from the creator's website [**?**]. Like with IMDB-WIKI, the choice was provided of either full-sized images totalling 1.3 GB or pre-cropped images at 107 MB. The first option was chosen this time, as the dataset size was already relatively small and doing so increased the diversity of the combined dataset, potentially promoting a more robust classification model.

The files came contained in three arbitrary folders with gender and age labels provided in the filename, similarly to the already-prepared data. The filenames were renamed using a Python script in the same fashion as the previous sets; no other alterations to the data were deemed necessary for UTKFace.

5.3.4 Final Dataset Structure

The completed dataset structure was now as seen in *Figure 5.11*. While it would be possible at this stage to split the datasets into training and validation sets, the step was instead incorporated in the training function (**5.4.1**), allowing for greater customisability over the dataset partition size.

Around 100 images from each dataset were removed and placed in separate folders for later testing (see **Evaluation**). Doing this ensured that the images would be entirely new to the network, and thus was better representative of true real-world testing.

5.3.5 Creating the PyTorch Dataset Class

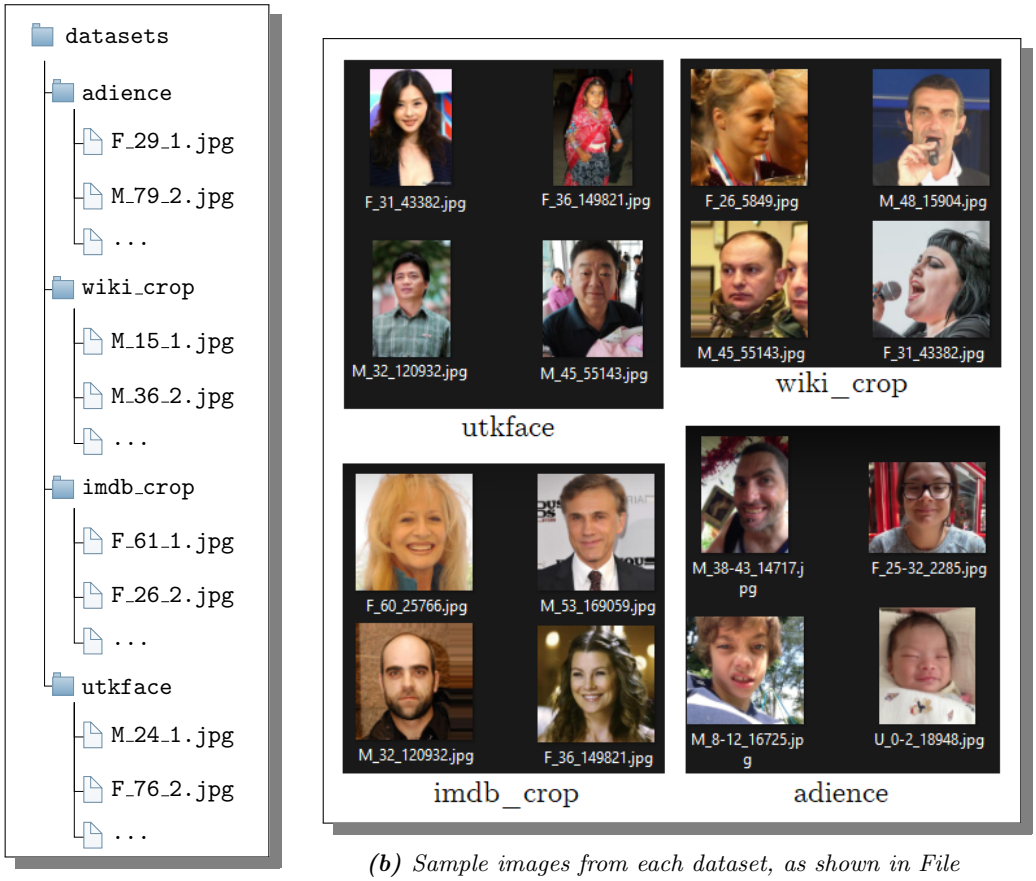
To use the obtained datasets with PyTorch, it was necessary to create a custom `Dataset` class, which enabled use of the PyTorch Dataloaders for training; this was done by inheriting the class properties of `torch.utils.data.Dataset` and defining custom functions for `__init__`, `__len__` and `__getitem__`, as specified in the official documentation [71].

https://github.com/jckpn/age-gender-cnn/face_dataset.py

```
from torch.utils.data import Dataset

class FaceDataset(Dataset):
    def __init__(self):
                                        . . .
    def __len__(self):
                                        . . .
    def __getitem__(self):
                                        . . .
```

A Pandas dataframe [72] is created when initialising the dataset, allowing data entries to be stored and retrieved from the system memory – this is significantly faster than reading from disk each time, however restricts the training session to the amount of RAM available.



(a) The dataset folder structure

(b) Sample images from each dataset, as shown in File Explorer

Figure 5.11: *Final dataset file structure*

The `os` module is used to obtain a full list of files from a given directory, using which the dataframe is iterative populated with entries from the specified dataset using `cv.imread`. Standard Python functions are used to extract the label from the filenames using their consistent filename format.

https://github.com/jckpn/age-gender-cnn/face_dataset.py

```
                                        . . .
def __init__(self, images_dir):
                                        . . .
    for filename in all_files:
        # Read entry image from file
        try:
            filepath = os.path.join(images_dir, filename)

            # Read image
            image = cv.imread(filepath)
```

```
                                    . . .
                # Get entry label from filename
                label = ...

                entry = {'image': image, 'label': label}
                self.dataframe.append(entry)
            except:
                continue    # Just ignore bad files
```

To test the custom dataset class, we can initiate a dataset with the Adience folder from earlier, and call `print(ds.__len__())` to confirm the final dataset length:

Terminal

```
25,473
```

Further Developments for the Dataset Class

Some further additions were made to improve the dataset class, including:

- The image pre-processor and transformations were implemented such that they get applied prior to adding each image to the dataframe.
- A separate `label_funcs.py` file was made containing the relevant label extraction functions, which could be passed to the dataset class as required for a given category.
- The loading process was wrapped in a `try/except` block [**?**], allowing for custom keyboard interruptions as well as enabling the process to continue past loading errors.
- The `tqdm` package was added to display a progress bar during the loading process, which allowed for stalls and other issues to be more easily monitored. Other useful debugging outputs were added, such as the number of files present in the specified folder, and the number of files that were successfully loaded into memory.

The final, full-length code for this file can be viewed at
`https://github.com/jckpn/age-gender-cnn/face_dataset.py`.

5.4 Network Implementation

As previously decided, the PyTorch framework was used to implement the network models for this project. An article by Analytics Vidhya [**?**] provided guidance for this stage of the development, as well as help from the official PyTorch documentation [**?**] for general guidance on CNN creation/training with PyTorch.

5.4.1 Network Architecture Implementation

PyTorch networks are defined as classes, allowing for easy repurposing and enabling architectures to be easily swapped by specifying a different network class. Using `nn.Sequential` as suggested in [**?**] makes it easy to define and interpret network architectures, as well as allowing for easy addition of new layers to the network if needed.

BasicCNN Implementation

The BasicCNN network class is severely limited by design, and was mainly implemented as a basis for the remaining architectures; the implementation for BasicCNN was written as follows, with a custom `num_classes` parameter to allow for easy repurposing of the network:

https://github.com/jckpn/age-gender-cnn/networks.py

```
from torch import nn
import torch

# BasicCNN - 1 conv layer, 1 linear layer
class BasicCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.num_classes = num_classes

        self.layers = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2),

            nn.Flatten(1), # Flatten conv output for linear layers

            nn.LazyLinear(num_classes) # LazyLinear calculates the input size,
            ↪  which is important as input size may vary by pre-processor
        )

    def forward(self, batch_in):
        # For processing batches during training
        batch_out = self.layers(batch_in)
        return batch_out

    def predict(self, single_in):
        # For computing predictions on single images, for testing or use
        out = self.forward(single_in)
        pred = torch.argmax(out) if self.num_classes > 1 else out[0][0]
        pred = pred.cpu().detach().numpy()
        return pred
```

LeNet Implementation

This was repeated for LeNet using the LeNet-5 architecture [30], with minor changes: it features ReLU activation functions instead of sigmoid, as well as using an average pooling layer instead of max pooling – two changes that are commonly made to LeNet to improve its performance [48].

AlexNet and VGG-16 Implementation

AlexNet and VGG-16 models both came pre-installed with PyTorch, so their implementations were far more abstract than for BasicCNN or LeNet. A custom `pretrained` parameter was added to each class so that the models could be loaded with or without pre-trained weights, as well as the same `num_classes` parameter to alter the model's final classifier as necessary.

https://github.com/jckpn/age-gender-cnn/networks.py

```
                                        . . .
self.layers = torch.hub.load('pytorch/vision:v0.10.0', 'alexnet', pretrained)
self.layers.classifier[-1] = nn.LazyLinear(num_classes)
                                        . . .
```

5.4.2 Network Training Function

The network training function was implemented according to the algorithm in **Section 4.5.4** with guidance from the PyTorch documentation [**?**] and an Analytics Vidhya article [35].

Implementing the Basic Training Loop

A default batch size of 32 was chosen as recommended in the PyTorch documentation [73].

https://github.com/jckpn/age-gender-cnn/train.py

```
                                        . . .
# TRAIN FUNCTION
def train_model(···):
                                        . . .
    # Init variables
    optimizer = optim_fn(model.parameters(), learning_rate)
    train_dataloader = DataLoader(train_set, batch_size, shuffle=True)
    val_dataloader = DataLoader(val_set, batch_size, shuffle=True)
                                        . . .
    # TRAINING LOOP
    for epoch_count in range(max_epochs):
        # Train model with training set
        model.train()  # Set model to training mode
```

```
        train_loss = 0
        for images, labels in tqdm(train_dataloader, ···) :  # iterate through
        ↪  batches
            optimizer.zero_grad()
            outputs = model(images)  # Forward pass
            loss = loss_fn(outputs, labels) # Compute loss
            loss.backward() # Backpropagation
            optimizer.step() # Update network weights
                                        ···
    # Return trained model
    return model
```

Validation Tests

Validation tests were implemented in a similar manner as previously described in the Methodology section. This required the network to be set to evaluation mode using `model.eval()`, and the validation loss to be calculated and returned as with the training loss. A history is stored of this value each epoch to aid with early stopping as well as for visualising the training process later on.

Implementing Patience and Model Saving

As described in **4.5.5**, patience is a technique used to prevent overfitting by stopping training if the validation loss does not improve for a certain number of epochs. This is implemented by saving the model's weights to a file each time the validation loss improves, and reloading the weights if the validation loss does not improve.

In this particular implementation, the learning rate is decreased by 50% each time the validation loss does not improve, to promote more nuanced changes to the weights once overfitting is detected. The training ceases after 3 full epochs of validation loss increasing, although the value can be modified to be more lenient when training the final models.

The basic patience implementation was added to the code as follows:

https://github.com/jckpn/age-gender-cnn/train.py

```
for epoch_count in range(1, max_epochs+1):
    # Train model with training set
                                        ···
    # Test model against separate validation set
                                        ···
    if best_val_loss is None or val_loss < best_val_loss:
        torch.save(model.state_dict(), model_save_path) # Save best model
        best_val_loss = val_loss
        patience_count = 0
    else:
        model.load_state_dict(torch.load(model_save_path)) # Reload best model
        patience_count += 1
```

This utilises PyTorch's `state_dict()`, which saves the model's weights to a file for later use. This is used instead of `save()`, which would save additional information about the model's architecture which is not needed for this purpose.

As an unintended side effect, this also saves the model's best weights to a file, allowing for later testing and use with the interface.

Further Developments to Training Function

The training function went through many changes. The most important additions to this file include:

- The status of the network is printed each epoch, including the current epoch count, the patience count, the current learning loss, the validation loss, and the elapsed time.
- The models are assigned a unique save path based on the architecture used as well as the current date and time, which is printed to the console for later reference.
- Some parameters of the network such as the chosen architecture, optimiser, learning rate, and loss function are also printed for clarity when training multiple models.
- The main loop was wrapped in a try/catch block similarly to the previous Dataset loading function, which allowed for custom keyboard interruptions to stop training early if desired.
- Tqdm progress bars were added to the training and validation loops to provide further visual indications of the training progress.

Initial Testing of the Training Function

To create and train a model using the function, it simply needs to be called with the desired parameters. While this could be done from another Python file or the command line, I opted to use a Jupyter notebook as it allowed the datasets to be kept in memory between training sessions.

A basic example of this is shown below:

https://github.com/jckpn/age-gender-cnn/run.py

```
from train import train_model
from networks import BasicCNN

train_set, val_set = ···

my_model = train_model(
    model=BasicCNN(num_classes=2),
    train_set=train_set,
    val_set=val_set)
```

Running the above code produces the following output:

Terminal

```
TRAINING MODEL BasicCNN-2_2004-2241.pt WITH PARAMS:
- Architecture: BasicCNN
- Learning rate: 0.0005
- Optimizer: Adam
- Loss function: CrossEntropyLoss()
- Other notes: None
```

For each epoch, a line is added to the output displaying its results with some other useful information for monitoring the training process.

Terminal

```
+----------+---------------+------------+----------+--------------+
|    EPOCH | EXAMPLES SEEN | TRAIN LOSS | VAL LOSS | ELAPSED TIME |
+----------+---------------+------------+----------+--------------+
|        1 |            78 |      0.726 |    0.693 |         0:10 |
|        2 |           156 |      0.697 |    0.691 |         0:18 |
|        3 |           234 |      0.693 |    0.687 |         0:25 |
|        4 |           312 |      0.679 |    0.682 |         0:33 |
|        5 |           390 |      0.609 |    0.738 |         1:02 |
|        6 |           468 |      0.722 |    0.678 |         1:09 |
|        7 |           546 |      0.622 |    0.676 |         1:16 |
+----------+---------------+------------+----------+--------------+

Halting training - 3 epochs without improvement

Best model from session saved to './BasicCNN-2_2004-2241.pt'
```

By outputting the training and validation losses per epoch, we can see that the BasicCNN model begins overfitting after the 4th epoch as the validation loss begins to increase while the training loss continues to decrease.

5.4.3 Network Testing Functions

A `tests.py` file is creatine to define the test functions proposed in **4.10**. Each function takes a given model and test dataset as input, and returns the relevant accuracy score.

Class Accuracy Test Function

The first function, `tests.class_accuracy()`, tests the given model on a binary true or false basis of whether it predicted the correct category or not. An accuracy score is provided based on the number of correct predictions divides by the total number of tests conducted.

MAE Test Function

The second function, `tests.mae()`, finds the average of the absolute difference between each prediction and its corresponding correct answer, returning the mean absolute error (MAE) of the model on the given test set.

Confusion Matrix Test Function

The third and final test function, `tests.confusion_matrix()`, creates and outputs a confusion matrix of the model's predictions on the given test set. This is done by creating an $n \times n$ matrix of zeros, where n is the number of classes in the dataset, and adding to the corresponding cell after each test. The matrix is then printed to the console along with an accuracy score for each class.

5.5 Network Training

This section details the initial training process for the prediction models, as well as the subsequent grid search to find the best hyperparameters for each model.

5.5.1 Transferring to Google Colab

It became apparent during early testing of the training function that the hardware of the Matebook was not sufficient to train the models in a reasonable time. As such, the training process was transferred to Google Colab, a cloud-based Jupyter notebook service which provides access to powerful machine learning hardware [**?**]; Colab Pro was used to gain access to high-RAM runtimes and premium GPUs, both of which significantly reduced the training time of the models and allowed for experimentation with larger datasets as well as the more demanding VGG-16 model.

To transfer the training process to Google Colab, the datasets were compiled into separate `.zip` files totalling just under 9 GB in size. These files were uploaded to a private Google Drive account, which allowed Colab to access and decompress the datasets in just a few minutes [**?**, **?**]. One downside to Colab was the lack of permanent storage, meaning the datasets had to be re-downloaded every time the notebook was restarted.

The previously written code was also copied into the notebook each time using `git clone` to access the repository created earlier. The original code required some modifications to work with Colab, such as changing the file paths to the datasets and adding the `.to_device()` function to move the model to the GPU. Aside from these caviets, the code worked as expected and the models were now able to be trained in several minutes as opposed to hours (or days!).

5.5.2 Initial Model Training and Results

Using the custom dataset class from before as well as the `train` and `test` functions from earlier, it was possible to quickly train and test several models for gender classification. To train the regression models for age estimation, some minor adjustments had to be made

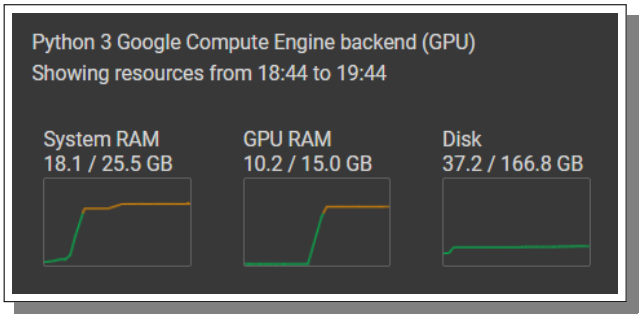


Figure 5.12: *Caption*

to the `train` function to enable the use of mean-square-error (MSE) loss, which requires the labels to be converted to floats [**?**].

To perform brief initial tests for each architecture, 20,000 entries were randomly sampled from the collective datasets (5,000 from each folder) and loaded into memory; this was done to reduce the training time and resource requirements of the models, as the full datasets would take several hours to train even using Colab's hardware. Adience was ommitted for age training, as the group labels provided did not match the yearly age labels of the other datasets. 6,700 entries were loaded from the remaining three datasets to compensate for the reduced data.

The models were trained with the following initial hyperparameters:

- **Loss function:** Cross-entropy loss
- **Optimiser:** Adam (suggested by [55])
- **Learning rate:** 0.0005 (suggested by [55])
- **Batch size:** 32
- **Validation split:** 0.2 (20% of the training set)
- **Epoch limit:** 30
- **Patience limit:** 3 (stop after 3 epochs of decreased results)

The pre-processor was set to crop the images to the face only , and the image size was set to the minimum size required by each model (32×32 for BasicCNN and LeNet, 224×224 for AlexNet and VGG-16).

It was also noted that even with this relatively small subsample of the datasets, loading the images into memory used over 16 GB of RAM as seen in *Figure 5.12*. A modified version of the dataset class was created to remedy this later on, but the tests performed at this stage were limited to the 20,000 entries. Making up only around 3% of the full combined datasets, the increased learning capabilties of the more sophisticated architectures (AlexNet and particularly VGG-16) were likely hindered. Due to time constraints, training on the full dataset was reserved for when the final parameters were determined.

Initial Results

Category	Model (Depth)	Input Size	Epochs	Training Time	Accuracy (%) or MAE (yrs)
Gender 50.2%	BasicCNN (2)	$32 \times 32 \times 1$	0		
Gender	BasicCNN (2)	$32 \times 32 \times 1$	12	11s	75.10%
Gender	LeNet [30] (5)	$32 \times 32 \times 1$	17	14s	76.62%
Gender	AlexNet [49] (8)	224x224x3	10	3m 42s	80.39%
Gender	VGG-16 [54] (16)	224x224x3	12	44m 28s	80.42%
Age (Mixed) 14.22 years	BasicCNN (2)	$32 \times 32 \times 1$	0		
Age (Mixed)	BasicCNN (2)	$32 \times 32 \times 1$	7	10s	13.82 years
Age (Mixed)	LeNet (5)	$32 \times 32 \times 1$	6	9s	13.57 years
Age (Mixed)	AlexNet (8)	224x224x3	4	1m 15s	13.21 years
Age (Mixed)	VGG-16 (16)	224x224x3	6	12m	13.24 years

Table 5.1: *Initial training results*

Observations of Initial Results

We see moderate results from the gender prediction models, with poor initial results from the age prediction networks, perhaps as a result of the reduced dataset size. This makes sense as binary gender classification would appear to be a far simpler task than age estimation, and the reduced dataset size would have a greater impact on age prediction with a much greater variety of labels.

A minor improvement in performance is seen to correlate with the depth of the network up to the 16-layered VGG-16 architecture; the improvements from VGG-16 over AlexNet are insignificant, perhaps alluding to the simplicity of the tasks in relation to the architecture's sophistication, or perhaps indicating insufficient training data in the initial tests. This is something that was investigated further by the grid search performed in the next section.

5.6 Grid Search for Hyperparameter Optimisation

In the field of machine learning, a grid search is a method of optimising the hyperparameters for a model. Hyperparameters are the training variables which are not directly learned by the model, but are set by the user to control the learning process.

5.6.1 Grid Search Methodology

As part of the grid search, the following values and hyperparameters were tested:

- Architecture (LeNet / AlexNet / VGG-16)
- Dataset selection (Adience / IMDB-WIKI / UTKFace / combined; Adience ommitted from age prediction due to labelling differences)
- Dataset division by gender (male / female / combined)
- Input image size (28×28 / 84×84 / 168×168 / 224×224)
- Pre-processor crop (face only / entire head)
- Optimiser (Adam / SGD)
- Learning rate (range 0.01-0.0001)
- Initial weights (randomised / pretrained from PyTorch)

To save time and allow more models to be tested, early stopping (as proposed in **4.5.5**) was implemented with a patience limit of 3. In other words, if the model failed to show progress for three consecutive epochs, the training was halted and the model reverted to its best state (as given by the lowest validation loss).

Note that BasicCNN was left out of the grid search as it was not expected to perform as well as the other architectures, and would only serve to increase the time taken to complete the process.

The training methodology remained similar to the initial tests in that a 20% validation split was used, and the training was limited to 30 epochs with a patience limit of 3. The same subsample was used of 20,000 random images pulled equally from each set.

To complete the grid search in full would require over 1,500 tests, which this was not feasible within the time constraints of the project; instead, a subset of the grid search was conducted by ommitting certain values once they were found to be ineffective. This process took several days of continuous training on Google Colab, with the 78 test results being saved to a spreadsheet. This spreadsheet can be found at `https://github.com/jckpn/age-gender-cnn/gri`

5.6.2 Architecture

While VGG-16 consistently performed the best for gender classification, AlexNet was found to achieve superior results for age prediction. This was unexpected as the VGG-16 architecture is significantly deeper than AlexNet, and age prediction is typically considered a more sophisticated task than gender categorisation. It is possible that the reduced dataset size contributed to its worsened performance, and that it would have performed better with a larger dataset.

LeNet consistently performed worse than the other architectures, which is unsurprising given its relative simplicity. As such, it was ommitted from the final model tests.

Unfortunately, despite the promising potential of VGG-16, it was found to be far too slow to train on the full dataset (taking around one hour per epoch), so AlexNet was chosen as the final architecture for both gender and age prediction.

Conclusion: AlexNet is chosen for the final models. VGG-16 would have been preferable if training time and computing power was not a limiting factor.

5.6.3 Dataset Selection

For gender classification, the IMDB-Clean dataset was found to be the most effective, providing an accuracy score even higher than that of the combined tests (75.65% vs 75.38%). The Wikipedia portion of IMDB-WIKI performed the worst, providing only a 66.00% accuracy score with all other parameters kept constant. Given the superior performance of IMDB-Clean as well as its size, it was chosen to be used as a single dataset for training the final gender models.

Conversely, for age estimation the Wikipedia portion actually performed best (7.51 years MAE) compared to IMDB-Clean or UTKFace alone (9.16 and 8.76 MAE respectively), however the combined datasets outperformed it further as expected.

Conclusion: IMDB-Clean is chosen to train the gender estimation models, while the combined datasets is chosen for training age prediction.

5.6.4 Dataset Division (Multi-Stage Processing)

Splitting the datasets by gender did not show any significant impact on the model's performance, with both male-only and female-only networks performing worse (albeit within 3% of the score acheived from the full dataset). As this method effectively reduces the data by half and no performance benefit was seen from this initial implementation, it was decided that it would be more beneficial to simply train the final models on the full dataset as provided.

Conclusion: Multi-stage processing in this fashion did not appear to improve performance. The full dataset is chosen for training the final models.

5.6.5 Input Image Size

N.B. 32×32 and 224×224 were chosen as they are the recommended image sizes for the architectures chosen; 84×84 and 168×168 are intermeniary sizes, chosen as the values $1/3$ and $2/3$ of the way between the previous two sizes.

As expected, the larger images showed superior training performance for both and and gender prediction. However, the benefits appeared to plateau after 168×168, perhaps due to the limited training size, or alluding to diminishing returns with the increased input size.

Smaller input sizes are easier to train, however larger image sizes can provide more detail for the network, which may be necessary to maximise prediction accuracy. This was largely reflected in these initial tests as 6/10 of the top performing gender models and 9/10 of the top performing age models utilised the largest tested image size of 224×224. 168×168 showed comparable performance for both categories in these initial tests, however the larger image size likely provides an even greater improvement when used with a larger

dataset. 84×84 did not perform significantly worse for gender classification, but it was found to be far worse for age prediction, likely due to the increased complexity of the task, and the fact that AlexNet and VGG-16 were both designed for use with the larger image size.

However, using 224×224 image files requires substantionally more resources and takes a significantly longer time to process. A 224×224 image requires over 2 times the storage of a 168×168 image, meaning only half of the data can be utilised for the same time and computation requirements, which may have a greater impact on the final model results than the image size itself.

One possible solution is to implement a resize function within the main training loop to resize images to the size required by the model, allowing smaller sized images to be loaded into the dataframe, while potentially mitigating the reduced performance of doing so.

As such, while 224×224 would likely perform better with sufficient training, 168×168 was chosen due to the lower computational requirements. The images will be resized to 224×224 during the training loop to match the intended architecture, allowing the final models to take use 224×224 inputs during user testing.

Conclusion: 168×168 is chosen for the final models' datasets. The training function now resizes images to 224×224 to match AlexNet's specification while keeping computational requirements low.

5.6.6 Pre-Processor Crop

The pre-processor crop did not initially appear to have a significant impact on predictive accuracy for either category, with marginally better age predictions when using the face crop (8.16 vs 8.44 years MAE, with AlexNet, all other params consistent).

However, analysing the confusion matrix for two gender models with differing crops revealed that it had a significant impact on classification accuracy per class despite virtually identical results for overall accuracy (77.50% vs 77.51%).

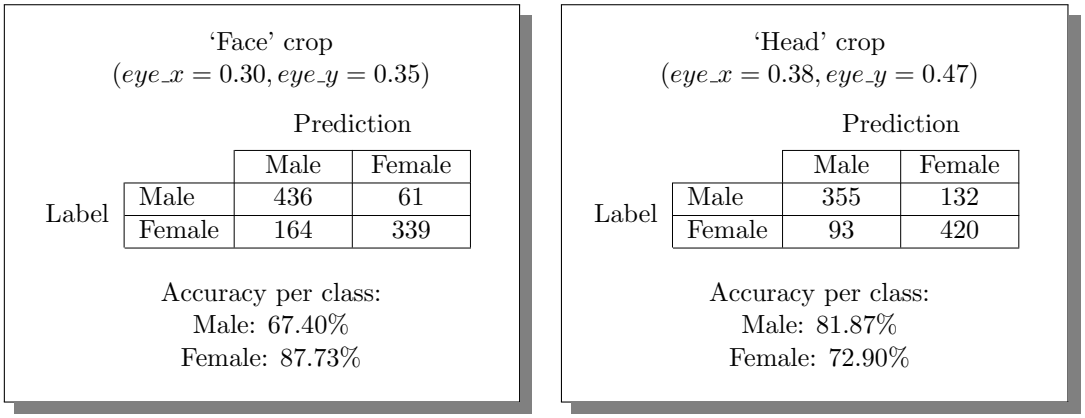


Figure 5.13: *Caption*

The smaller 'face' crop presented a significant bias towards female predictions, while the larger 'head' crop skewed the model towards male predictions. In other words, the smaller crop meant male subjects were more likely to be incorrectly predicted as female, and vice versa for the larger crop.

As such, new alignment values were chosen with the average of the previous values used. With new values of `eye_x = 0.34` and texttteye_y = 0.42, the pre-processor cropped to show slight details of hair and face shape while remaining focused on the face itself.

Conclusion: New alignment values are proposed, enabling a middleground between the previous cropping methods for the final model's pre-processor.

5.6.7 Pre-Trained Weights

The two best-performing gender estimation models were both pre-trained VGG-16 models, achieving 83.33% and 82.75% accuracy (trained with SGD 0.005 vs SGD 0.01 respectively). This is unsurprising given the performance capabilities proven with the VGG-16 architecture, however it is interesting to observe that the pre-trained AlexNet models performed consistently worse than the models initialised with random weights. This may be due to the pre-training method causing AlexNet to specialise for features not relevant to facial image analysis.

For age prediction, the pretrained models consistently performed worse with both AlexNet and VGG-16. Again, it is possible that the pretraining processed specialised the models for features not necessarily relevant for the task at hand, and that the models would have adapted better given enough time with the full dataset. However, given the results of these initial tests and provided that VGG-16 is too slow to train on the full dataset, it was decided that the final models would be initialised with random weights.

Conclusion: The final models are to be initialised with random weights.

5.6.8 Optimiser and Learning Rate

It was suggested by [55] that the Adam optimiser is typically most effective, and that a learning rate of 0.0005 is a good starting point. While this provided adequate results, better models were obtained using an even lower learning rate of 0.0001. In the case of the pre-trained architectures, the SGD optimiser combined with a learning rate of 0.005 was found to be more effective than Adam.

Conclusion: A learning rate of 0.0001 with the Adam optimiser will be used to train our final models.

5.6.9 Conclusion of Grid Search

The following values and hyperparameters were concluded as provisionally providing the best results for the final neural networks:

- **Architecture:** AlexNet
- **Dataset selection:** IMDB-Clean (gender), combined (age)
- **Dataset division by gender:** No (ineffective)
- **Input image size:** 168×168, upscaled to 224×224 during training
- **Pre-processor crop:** cropped with new values from **5.6.6**
- **Optimiser:** Adam
- **Learning rate:** 0.0001
- **Initial weights:** Randomly generated

5.7 Further Developments

5.7.1 Dataset Equalisation and Data Augmentation

As described in the Methodology section, imbalances in the dataset can steer models towards unnecessarily predicting one particular class. One way of mitigating this is to automatically equalise the datasets by A) duplicating images from under-represented classes and B) reducing the number of images from over-represented classes.

This was implemented by calculating the number of entries requires for each class (the 'class goal', as the number of classes divided by the total dataset size). Using this number, entries can either be rejected if the class goal is already fulfilled, or entries can be duplicated to meet the target number. The code for this implementation is as follows:

Implementing Dataset Equalisation

The custom dataset class was updated to equalise data like follows:

https://github.com/jckpn/age-gender-cnn/face_dataset.py

```
                                        . . .
class FastDataset(Dataset):
    def __init__(...):
                                        . . .
        # Set up equalisation 'requirements'
        class_goal = ds_size // classes
        eq_requirements = []
        if classes is not None:
            for i in range(classes):
                requirement = {'class': i, 'count': 0, 'aug_count': 0}
                eq_requirements.append(requirement)
                                        . . .
        while len(self.dataframe) < ds_size:
                                        . . .
            label = label_func(filename)
            if equalise and eq_requirements[label]['count'] > class_goal:
                continue # Skip if we have enough of this class
                                        . . .
            self.dataframe.append(entry)
            eq_requirements[label]['count'] += 1
                                        . . .
```

Now, when loading a dataset, the number of entries for each class is made equal.

Experimental Results of Dataset Equalisation

To test the effects of the updated dataset class, two AlexNet models were trained with the same parameters, with the only difference being the equalisation on the datasets for the second model. Both networks were tested for accuracy against the same equalised portion of IMDB-Clean (separated from either training dataset). The training methodology remained the same as the grid search.

While not necessarily an accurate representation of the final training process, data equalisaion boosted classification accuracy on this initial test from 78.0% to 80.3%. The benefit is even more apparent when viewing the test results for each class as below.

Without dataset equalisation

Label \ Prediction	Male	Female
Male	433	87
Female	133	347

Accuracy per class:
Male: 83.27%
Female: 72.29%

With dataset equalisation

Label \ Prediction	Male	Female
Male	418	102
Female	95	385

Accuracy per class:
Male: 80.38%
Female: 80.21%

Figure 5.14: *Caption*

The same test was conducted for age estimation, however this time the performance actually decreased dramatically – going from 10.32 MAE to 15.61 MAE. As such, for the final model training, dataset equalisation will only be applied to the gender model.

Implementing Data Augmentation

The issue with equalising a set in this fashion is that it causes the under-represented classes to be overly represented with duplicate images as opposed to 'genuine' entries, which increases the risk of the network overfitting to those particular classes; additionally, the over-represented classes are reduced to a small number of entries, reducing the potential effectiveness of the end model. To mitigate this, data augmentation can be applied to the initially under-represented classes to increase the number of 'genuine' entries.

Data augmentation also helps to reduce overfitting even when equalisation is not used, by providing the network with 'new' images even if the data entries have already been exhausted through training.

The PyTorch `transforms` library provides several operations for data augmentation [73], which were easy to incorporate within the dataset loading process.

The augmentations were incorporated such that, if enabled, they are applied to each batch of images before being processed by the training network. The process is ommitted from the validation test entries to provide a fairer and consistent test for the network every epoch.

When called, the augmentations are applied at random, containing any combination of the following:

- `RandomHorizontalFlip` – may flip the image horizontally
- `RandomAdjustSharpness` – randomly adjusts the sharpness of the image
- `GaussianBlur` – applies a gaussian blur to the image with a random radius
- `RandomRotation` – randomly rotates the image slightly (between -10 and 10 degrees)

- `RandomPerspective` – applies a random, subtle 3D perspective transformation to the image
- `RandomGrayscale` – may convert the image to grayscale

The implementation within Python was as follows, with `aug_transform` containing the above augmentations with random probabilities of being applied:

```
                                        . . .
class FaceDataset(Dataset):
    def __init__(···):
                                        . . .
                if augment is True and cycles > 0:
                    # Randomly apply various augmentations to bolster dataset
                    aug_transform = ···
                    image = aug_transform(image)
                    eq_requirements[label]['aug_count'] += 1
                                        . . .
```

By converting some of the augmented images back to OpenCV matrices and displaying them with `cv.imshow`, we can see the augmentation process successfully generates several 'new' images with subtle yet realistic variations on the original (*Figure 5.15*).

Figure 5.15: *Example images generated by the new augmentation process*

This should greatly improve the robustness and accuracy of the network, particularly to grayscale and/or low-resolution inputs.

Experimental Results of Data Augmentation

Testing data augmentation with the same methodology as above, we see a slight further improvement from 79.9% to 81.3% accuracy, with the augmentation model able to train for 7 more epochs (14 vs 21) before overfitting occurred.

We see a modest improvement of 3.7% by implementing data augmentation on this reduced dataset, allowing the network to train for 10 more epochs before overfitting.

5.7.2 Enabling Larger Datasets

A new dataset class was written, `SlowDataset`, to enable images to be read from the machine's storage as opposed to from the RAM. This allows for much larger datasets to

be used, however training speeds are likely to be significantly reduced as even modern SSDs are significantly slower than RAM.

A sufficiently powerful training machine would have allowed for the use of the full datasets in RAM, allowing for far greater experimentation and better end results. However, due to the inherently slow read speeds of disk storage compared to modern RAM modules, this module was not able to be utilised.

Given sufficient time (or sufficiently fast disk speeds), this modification dataset class may have enabled further improvements to the results

5.8 Training the Final Models

With the model values and hyperparameters determined from the grid search, it was time to train the final models using the full datasets.

5.8.1 Methodology of Final Model Training

Training Hyperparameters

The final models were trained using an increased epoch limit of 50 and a far more lenient patience limit of 10, using the values determined from the grid search.

The epoch limit and patience limit were increased to allow for more exploration during the training process, enabling the weights to become even more nuanced and accurate for the given tasks.

Dataset Selection

While it likely would have been beneficial to utilise the full datasets for the final training, time and resource constraints restricted us to using ∼125,000 dataset entries – using more than this risked crashing Google Colab due to the memory limits imposed on the tier used.

The entries were randomly sampled from the datasets; for gender classification, this was entirely from IMDB-Clean, and for the age regression model, the data were equally sub-sampled from UTKFace and IMDB-WIKI, as justified in **5.6.3**.

With a validation data split of 20%, this leaves 100,000 data entries for training and 25,000 data entries for validation tests during training (and to provide some initial accuracy tests following training).

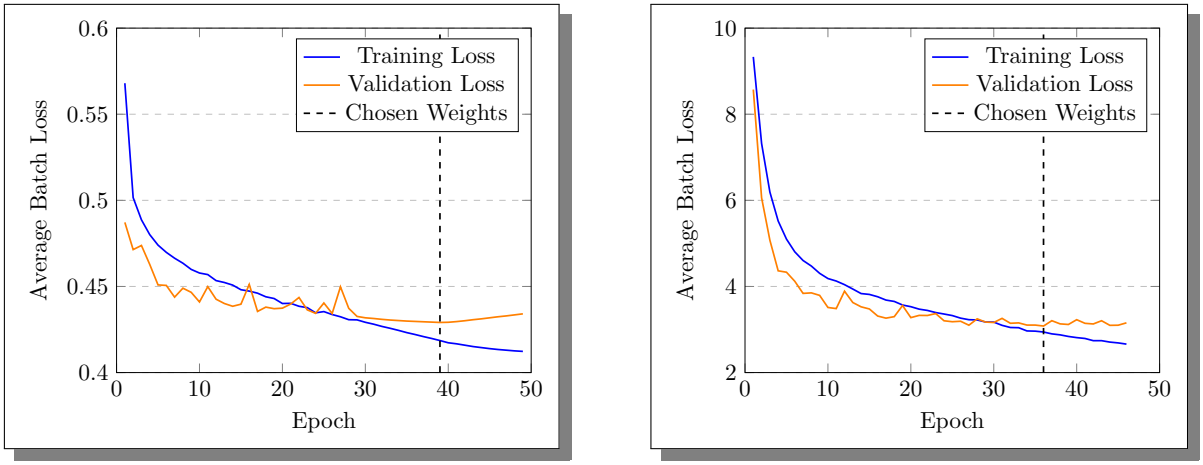
5.8.2 Final Model Training Results

The larger datasets proved to be beneficial to this task, producing models with 82.37% class accuracy and 7.20 MAE for gender and age predictions respectively.

Category	Epochs	Training Iterations	Training Time	Best Val. Loss	End Val. Loss	Best Val. Test Score
Gender	49	4,900,000	1:02:22	3.081	3.155	82.37%
Age	46	4,600,000	58:45	0.4291	0.4341	7.20 MAE

Table 5.2: *Final training results*

The following figure shows the training and validation loss from the gender and age model training sessions (*Figures 5.16a* and *5.16b* respectively).



(a) *Final gender prediction model training session*

(b) *Final age prediction model training session*

Figure 5.16: *Training and validation loss values during final training sessions*

We see the validation loss started to increase at epoch 39 for the gender model and 36 for the age model, indicating overfitting began to occur after these points. The final weights were taken from the epoch with the best (lowest) validation loss.

The models are evaluated more thoroughly in the following chapter.

5.9 GUI Implementation

A basic GUI was implemented to allow for visualisation of the models, as well as for improved testing. The GUI also allows the model to be used with custom image inputs as well as a camera if available, largely opening the potential use cases of the model.

5.9.1 Visualising on Image Input

Implementing the GUI was relatively straightforward, reusing much of the code from the pre-processor experiments:

https://github.com/jckpn/age-gender-cnn/visualise.py

```
                                        ...
def run_model(input_img, net, processor, transform):
                                        ...
    return face_images, coords, preds

def visualize_model(input_img, coords, g_preds, a_preds):
                                        ...
    for idx, e in enumerate(coords):
                                        ...
            # Add boxes and labels on detected faces
            this_gender = g_preds[idx]
            this_age = a_preds[idx]

            cv.rectangle(input_img,
                         (face_x, face_y-14),
                         (face_x+face_w, face_y),
                         (255, 255, 255),
                         -1) # fill rectangle
            cv.putText(
                input_img,
                text=('F' if this_gender==0 else 'M') + str(int(this_age)),
                org=(face_x+1, face_y-2),
                fontFace=0,
                fontScale=0.5,
                color=(0,0,0),
                thickness=1)
                                        ...
    return input_img

def visualise_image(...):
    in_image = cv.imread(image_path)
                                        ...
    out_image = visualize_model(in_image, coords, g_preds, a_preds)
                                        ...
    cv.imshow(image_path, out_image)
    cv.waitKey(0)
```

Python's `argparse` module was utilised to enable the file to be easily run from a command prompt. The user can specify an image to test the model with using the `--image-path` argument. The model runs on the image, and the predictions are iteratively visualised with `cv.rectangle` and `cv.putText` using the face bounds of `FaceDetectorYN` and the predictions from the final models.

Example of Model Visualisation on Image

Utilising `visualise.py` through the command prompt as follows:

Terminal

```
python3 visualise.py --image-path example_images/yum.jpg
```

Displays the following image:

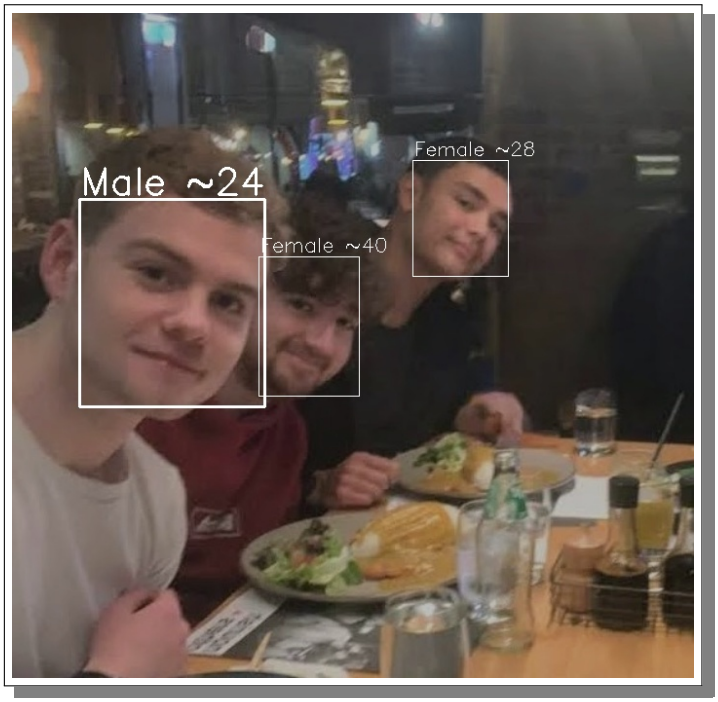


Figure 5.17: *Caption*

5.9.2 Visualising on Live Camera Input

If no image path is supplied by the user, a camera input is initialized with `cv.VideoCapture` [**?**], and the model is run recursively on the camera's input.

https://github.com/jckpn/age-gender-cnn/visualise.py

```
def visualise_cam(···):
    cam_input = cv.VideoCapture(cam_id)
                                        ···
    counter = 0
    while cv.waitKey(1) < 0:
        # Iteratively get camera input frame
```

```
        _, frame = cam_input.read()
                                        . . .
        out_image = visualize_model(frame, coords, g_preds, a_preds)

        if show_processed_faces:
                                        . . .
        cv.imshow(win_title, out_image)
        counter += 0
```

Still Frame Extraction

A still frame extraction mechanism was implemented similar to the one proposed by Mehendale in [27]. The mechanism by which this works is as follows.

- An update interval is specified, which is the number of frames between each update.
- A 'frame score' value is calculated each update using `np.sum(frame)/frame.size` – the sum of all pixel values divided by the size of the image.
- A `frame_diff` value is calculate by taking the difference between `this_frame_score` and last_frame_score. This value is indicative of the difference in content between frames – literally speaking, it is the average difference in pixel values across the entire image. A value of 255 would indicate the maximum possible content difference – a shift from all-black to all-white, for example – whereas a value of 0 would indicate no change between frames.
- Then, if the value of `frame_diff` is below the given threshhold, the frame is assumed to be still, and is used as input for the models.

The Python implementation for this appears as below:

https://github.com/jckpn/age-gender-cnn/visualise.py

```
this_frame_score = np.sum(frame)/frame.size
    frame_diff = abs(this_frame_score - last_frame_score)
    last_frame_score = this_frame_score
    if frame_diff < frame_diff_threshold:
        face_images, coords, g_preds = run_model(frame, g_net, g_processor,
        ↪  g_transform)
        _, _, a_preds = run_model(frame, a_net, a_processor, a_transform)
```

Example of Model Visualisation on Video

5.9.3 Visualising the Processed Images

The ability was added to specify to see the processed/transformed images alongside the input.

https://github.com/jckpn/age-gender-cnn/visualise.py

```
. . .
if show_processed_faces:
            # paste each image from face_images into frame
            for idx, face in enumerate(face_images):
                # transform expects PIL image
                face = Image.fromarray(face)
                face = g_transform(face)
                face = face.squeeze(0).numpy()
                if len(face.shape) == 2:
                    face = cv.cvtColor(face, cv.COLOR_GRAY2RGB)
                else:
                    face = np.transpose(face, (1,2,0))
                face_min, face_max = face.min(), face.max()
                face = (face - face_min) / (face_max - face_min) * 255
                face = cv.resize(face, (resize//10, resize//10))
                # convert to rgb if grayscale
                out_image[idx*face.shape[0]:(idx+1)*face.shape[0],
                ↪  0:face.shape[1]] = face[:,:]
. . .
```

The user can use this by adding the `--show-processed-faces` flag to the previous command line prompt – this is particularly useful for debugging.

Terminal

```
python3 visualise.py --image-path example_images/jack.jpg
↪  --show-processed-faces
```

5.10 Ommitted Developments

Several proposed developments were ommitted from the final implementation due to time and/or resource constraints:

- **Attention mechanisms** – although implemented in `networks.py` and available for use, the complexity of these models meant that they took an infeasible amount of time to be trained and thus could not be tested or included in the final report. Initial tests showed comparable results to the regular architectures, but it is possible that they would have outperformed the regular architectures given sufficient time and data.
- **Video file input** – the ability to run the model on a given video input was proposed, but not implemented. This would have been a relatively straightforward extension of the live camera input, but was not implemented due to time constraints.
- **Further experimentation of pre-processing methods** – one area which could have been improved on is the pre-processor. Due to time constraints, the crop

size was the only thing experimented with, but it is possible different alignment techniques could produce improved results.

Chapter 6: Evaluation and Comparison

6.1 Initial Results of Final Models

It was seen in the previous chapter that the gender and age models achieved 82.37% and 7.20 MAE accuracy scores respectively when tested against the training session's validation set.

As specified in the implementation, 500 images from each dataset were pulled and removed for testing the final models. Testing the models against these combined datasets we get a gender classification accuracy of 82.09%, with a mean absolute error of 7.22 for the age predictions.

6.1.1 Performance of Final Gender Model

The confusion matrix for gender classification can be seen in *Figure 6.1.* From this we see that gender predictions are still somewhat biased towards male subjects, however accuracy remains relatively high for both classes.

6.1.2 Performance of Final Age Model

By dividing age into distinct groups of 10 years, a confusion matrix can be produced to get a general idea of the network's accuracy for age classification. This reveals that the majority of predictions made are within 20 years of the labelled value:

		Prediction							
		0-9	10-19	20-29	30-39	40-49	50-59	60-69	70+
Label	0-9	112	10	5	2	0	1	0	0
	10-19	13	35	89	19	5	1	0	0
	20-29	2	29	507	254	42	7	1	0
	30-39	1	5	224	339	108	23	3	0
	40-49	0	1	47	161	150	45	8	1
	50-59	1	0	16	55	100	87	28	5
	60-69	0	0	6	16	31	55	50	10
	70+	0	0	1	1	2	3	8	14

Figure 6.1: *Confusion matrix for age prediction model with 1500 total tests*

Another (perhaps more intuitive) way of visualising the network's performance is by plotting each predicted age against the corresponding labelled age for each test, as in *Figure* ... (note that the dataset was equalised for this graph to produce more readable results).

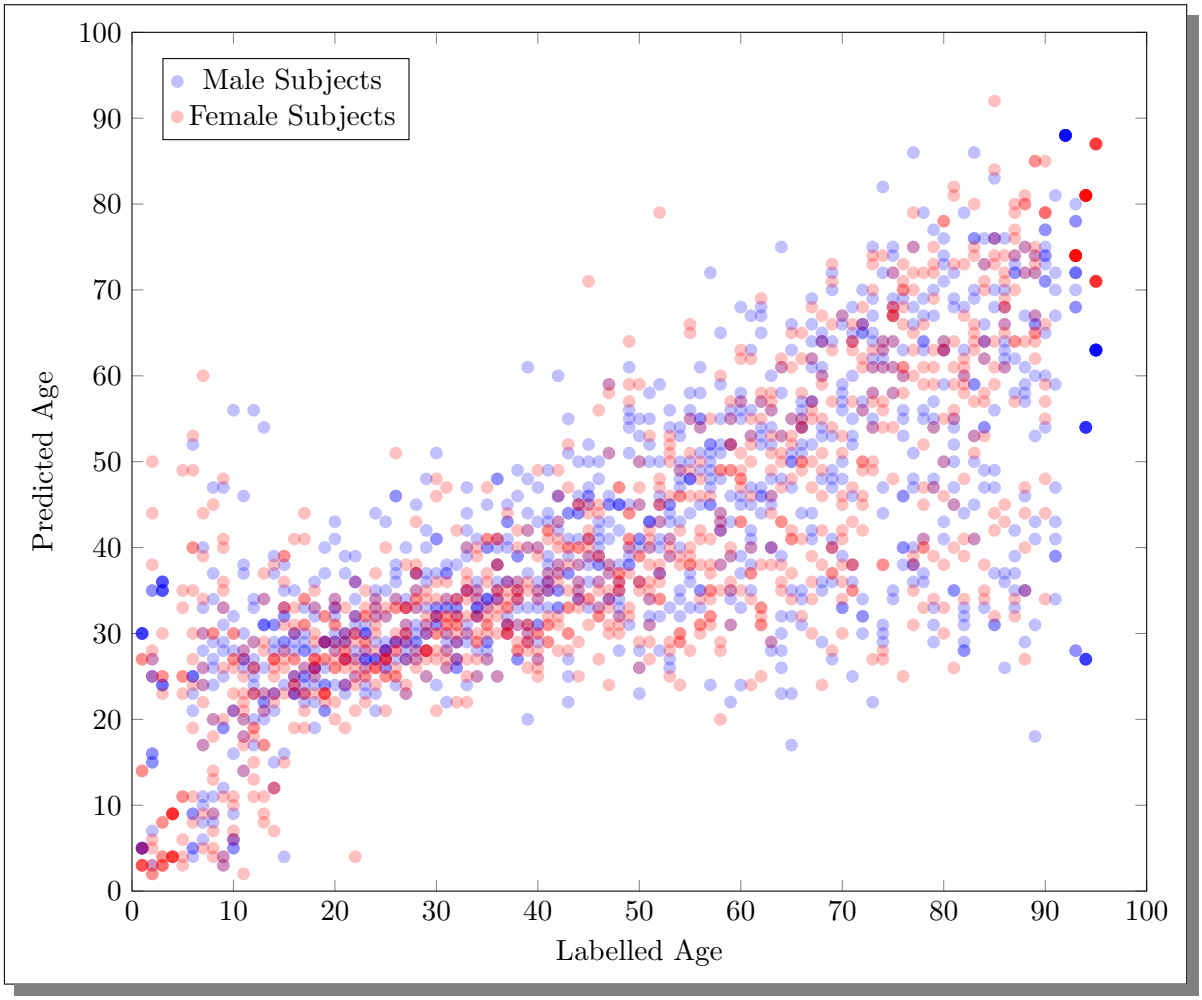


Figure 6.2: *Scatter graph plot of subject age vs predicted age of final age model*

It's particularly apparent from the scatter plot that the age predictions are most accurate in the 20-40 age range, getting particularly unreliable below 10 and above 50 years old. This unsurprisingly matches the distribution of the datasets used; since the age datasets for this model were not equalised, it would have been most exposed to images of 20-40 year olds during its training, and thus became most proficient at predicting for those in this age range.

6.2 Comparison to Other Prediction Models

The accuracy of the system produced in this paper does not exceed that of pre-existing models.

Reference	Category	Results
Yan et al., 2014 [23]	Gender	98.1% classification accuracy (2 classes)
Liew et al., 2016 [24]	Gender	99.4% classification accuracy (2 classes)
Rodríguez et al., 2017 [25]	Gender	93.0% classification accuracy (2 classes)
Our model	**Gender**	**82.4% classification accuracy (2 classes)**
Rodríguez et al., 2017 [25]	Age	61.8% (8 classes) or 2.6 years MAE
Rothe et al., 2016 [26]	Age	2.7 years MAE
Huerta et al., 2015 [10]	Age	3.6% MAE
Yan et al., 2014 [23]	Age	76.6% classification accuracy (13 classes)
Our model	**Gender**	**7.2 years MAE**

Table 6.1: *Summary of reviewed literature*

6.3 Comparison to Human Perception

The model produced in this paper performs significantly worse than humans at gender estimation but performs comparably with age estimation, according to the results of [**?**] (7.20 MAE vs 8.36 RMSE[1] (*Figure 6.3*) depicting similar inaccuracies to our computational model.

[1]While it is not fair to directly compare MAE and RMSE results, it can be assumed that the values would be similar, further bolstered by the plot provided in [74]

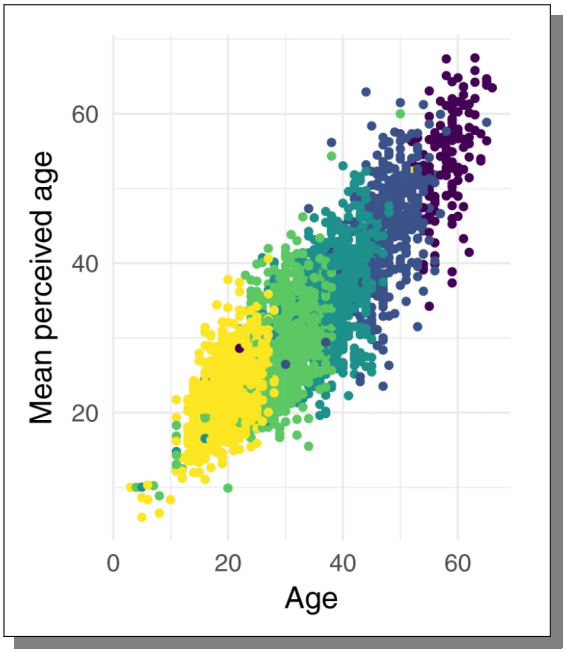


Figure 6.3: *Graph depicting the age estimation accuracy of humans*

6.4 Testing on Real-World Examples

Here are some examples of the model in some potential real-world use cases, using a mix of images obtained online (cited) and from my personal camera roll.

6.4.1 Portrait Images

Portrait images are arguably the easiest use cases for the model, as portrait images are typically frontal photographs where the subject's face makes up a large fraction of the image contents. It also makes up the large majority of the training dataset, particularly in the case of the gender model where training entries came exclusively from photos of IMDB actors.

Figure 6.4: *Testing the model on portrait photos. Source: Unseen partition of UTKFace dataset [5]*

We see the networks performing admerably here, with 3 out of 3 correct gender predictions and age estimations fairly close to the true values (from left: 27, 45, 23).

6.4.2 Group Photos

Group photos are more challenging and pose more emphasis on the pre-processor, testing its effectiveness to isolate and standardise faces from an image of several subjects.

Figure 6.5: *Testing the model on group photos. Sources: [11], [6]*

We see that while the pre-processor successfuly identifies all subjects, the gender predictions suffer on one of the subjects, perhaps due to the smaller image size and slight

occlusion from the neighbouring subject. The age predictions are within reasonable range of the true value for all subjects.

6.4.3 Surveillance Footage

One of the proposed use cases for this model was for security or surveillance footage. This is hard to test due to the lack of high-quality surveillance footage available online, however it is demonstrated as a proof of concept below.

Figure 6.6: *Testing the model on surveillance footage. Source: [**?**]*

Here the model accurately identifies and classifies both subjects as male despite the low image quality, however the age predictions are incorrect by a big margin (actual ages: 34, 72).

6.5 Areas for Future Improvement

This project demonstrated that it is possible to create a moderately accurate system for facial age and gender prediction, using only publically available tools, resources, and datasets.

However, the final results are far from perfect, and there are several areas the model could be improved further.

- **Deeper Exploration of Architecture** – due to hardware limitations, we were not able to experiment with larger architectures such as VGG-16, which likely would have provided superior results to AlexNet.

- **Further Experimentation with Hyperparamters** – given more time, it may have been possible to experiment further with different loss functions, optimiser momentum, etc.
- **Pre-Processor Experimentation and Types** – it is possible that different pre-processor methodologies could have provided superior results, but this was not tested due to time limitations.
- **Ability to Use Video Files as Inputs** – adding this capability would increase the use cases of the model.
- **Model Stacking** – model stacking (combining multiple models for a final classification) may have provided superior results, especially for the more complex age estimation task.

Chapter 7: Conclusion

The aims of this project as established at the start were to produce and document a facial analysis model for predicting age and gender. This has been achieved and, while the performance does not compete with the pre-existing models produced by high-end research groups, it still contributes to the growing number of successes from utilising deep learning for facial image analysis.

Through the literature review and thorough experimentation, it was found that these models are very responsive to large scale datasets, and that the quality of these datasets is paramount for producing a well-performing model, as shown through the comparitive tests on dataset equalisation and augmentation. Further, the grid search revealed the necessity of experimentation when producing such models, as the end performance is greatly impacted by several variables – including but not limited to the architecture, optimiser, learning rate and input size. It was also found that these models are particularly sensitive to variations in input, and high accuracy is typically achieved by employing a pre-processor to standardise inputs for the network. For any developers wishing to endeavour the same task, it's recommended to dedicate as much time as possible to experiment with these nuances, as such minor changes to the methodology can have a profound impact on the end performance of the model.

Through optimisation of these parameters, a model was eventually trained capable of predicting gender with an accuracy of over 82% and estimating age within 8 years of the true value on average, using only publicly available tools and resources. The system produced could be utilised for a variety of purposes, such as for automated video surveillance or enhanced marketing. The modules produced through development could easily be repurposed for other similar machine learning challenges, such as facial mood detection or object recognition.

This project also revealed that computational power can be a major limiting factor in the potential to produce a well-performing neural network. This limitation was partially overcome through use of rented GPUs via Google Colab; however, with the current state of deep network architectures and current hardware capabilities, it is bound to be a barrier for many wanting to experiment in the field of machine learning. The proposed 'variable attention' pipeline could have the potential to produce far better results than the eventual models did, however, the technique could not be sufficiently experimented with due to time and resource constraints.

To conclude, this project has shown that with enough time and experimentation, a fairly accurate facial analysis model can be produced using entirely free and publicly accessible resources.

References

[1] What is the difference between sex and gender? https://www.ons.gov.uk/economy/environmentalaccounts/miscs/whatisthedifferencebetweensexandgender/2019-02-21.

[2] Early emotion understanding: When do babies learn about emotions? https://www.psychologyinaction.org/psychology-in-action-1/2016/07/12/early-emotion-understanding-when-do-babies-learn-about-emotions.

[3] Sann overviews - network generalization. https://docs.tibco.com/pub/stat/14.0.0/doc/html/UsersGuide/GUID-297FE548-BC4D-4749-AFB0-A4A987E9BD74.html.

[4] ax Inc.

[5] Song et al. Zhang, Zhifei. Age progression/regression by conditional adversarial autoencoder, 2017.

[6] Jack Paine (Author).

[7] Neurohive. Alexnet – imagenet classification with deep convolutional neural networks, 2018. https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/.

[8] Neurohive. Vgg16 – convolutional network for classification and detection, 2018. https://neurohive.io/en/popular-networks/vgg16/.

[9] Luc Van Gool Rasmus Rothe, Radu Timofte. Imdb-wiki – 500k+ face images with age and gender labels, 2015. https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/.

[10] Ivan Huerta, Carles Fernández, Carlos Segura, Javier Hernando, and Andrea Prati. A deep analysis on age estimation, 06 2015.

[11] Monkey Business Images. Shutterstock, 2023. https://www.shutterstock.com/image-photo/three-generation-hispanic-family-standing-park-1284992623.

[12] Kwon et al. Age classification from facial images.

[13] Artificial neural network. https://link.springer.com/chapter/10.1007/978-3-319-13168-9_22.

[14]

[15] Yoti. Age verification tools for online customers and custom-built apps. https://www.yoti.com/business/age-verification/.

[16] Facebook's emotional contagion study and the ethical problem of co-opted identity in mediated environments where users lack control.

[17] On facebook's emotional ad targeting, the manipulation of younger users, and the concerns of big data. https://www.socialmediatoday.com/social-networks/facebooks-emotional-ad-targeting-manipulation-younger-users-and-concerns-big-data.

[18] Larry Hardesty. Mit news - explained: Neural networks, 2017. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[19] Andrey Kurenkov. Skynet today - a brief history of neural nets and deep learning, 2020. https://www.skynettoday.com/overviews/neural-net-history.

[20] Shraddha Goled. How do activation functions introduce non-linearity in neural networks?, 2021. https://analyticsindiamag.com/how-do-activation-functions-introduce-non-linearity-in-neural-networks/.

[21] Vitaly Bushaev. How do we 'train' neural networks?, 2017. https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73.

[22] Jason Brownlee. Crash course in convolutional neural networks for machine learning, 2016. https://machinelearningmastery.com/crash-course-convolutional-neural-networks/.

[23] Chenjing Yan, Congyan Lang, Tao Wang, Xuetao Du, and Chen Zhang. Age estimation based on convolutional neural network, 2014.

[24] Shan Sung Liew, Mohamed Khalil-Hani, Feeza Radzi, and Rabia Bakhteri. Gender classification: A convolutional neural network approach, 03 2016.

[25] Pau Rodriguez, Guillem Cucurull, Josep Gonfaus, Xavier Roca, and Jordi Gonzàlez. Age and gender recognition in the wild with deep attention, 07 2017.

[26] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks, 04 2018.

[27] Ninad Mehendale. Facial emotion recognition using convolutional neural networks (ferc), 03 2020.

[28] PyImageSearch. Github: Facealigner python module, 2017. hhttps://github.com/PyImageSearch/imutils/blob/master/imutils/face_utils/facealigner.py.

[29] Roee Enbar Eran Eidinger and Tal Hassner. Age and gender estimation of unfiltered faces, 2014. https://talhassner.github.io/home/projects/Adience/Adience-data.html.

[30] Yann LeCun, L'eon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition, 1998.

[31] Guido Van Rossum and Fred L Drake Jr. Python reference manual, 1995.

[32] Monomita Chakraborty. What are the best programming languages for artificial intelligence, 2021. https://www.analyticsinsight.net/what-are-the-best-programming-languages-for-artificial-intelligence/.

[33] Python Software Foundation. Python - active python releases, 2023. https://www.python.org/downloads/.

[34] Pytorch: An imperative style, high-performance deep learning library, 2022. https://pytorch.org/.

[35] Siddharth M. Build your first artificial neural networks using pytorch, 2021. https://www.analyticsvidhya.com/blog/2021/08/build-your-first-artificial-neural-networks-using-pytorch/.

[36] Microsoft. Visual studio code, 2022. https://code.visualstudio.com/.

[37] Anaconda software distribution. https://docs.anaconda.com/, journal=Anaconda Documentation, version=Vers. 2-2.4.0, publisher=Anaconda Inc., year=2020.

[38] github. Github, 2023. https://github.com/,.

[39] Stefan van der Walt et al. Charles Harris, Jarrod Millman. Array programming with numpy, 2020.

[40] Python package index - pypi. https://pypi.org/, urldate = 2021-03-28, publisher=Python Software Foundation.

[41] G. Bradski. The opencv library, 2000.

[42] Huawei. Huawei matebook 13 2021, 2021. https://consumer.huawei.com/en/laptops/matebook-13-2021/specs/.

[43] Google. Colaboratory - google, 2023. https://colab.research.google.com/.

[44] Vikas Gupta. Face detection – dlib, opencv, and deep learning (c++ / python), 2018. https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/.

[45] Pierre Ecarlat. Cnn — do we need to go deeper?, 2017. https://medium.com/finc-engineering/cnn-do-we-need-to-go-deeper-afe1041e263e.

[46] Ayush Thakur. Relu vs. sigmoid function in deep neural networks, 2022. https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI.

[47] ACM. Am turing award - yann lecun, 2018. https://amturing.acm.org/award_winners/lecun_6017366.cfm.

[48] Sik-Ho Tsang. Review: Lenet-1, lenet-4, lenet-5, boosted lenet-4 (image classification), 2018. https://sh-tsang.medium.com/paper-brief-review-of-lenet-1-lenet-4-lenet-5-boosted-lenet-4-image-classification-1

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks, 2012.

[50] Sik-Ho Tsang. Review: Alexnet, caffenet — winner of ilsvrc 2012 (image classification), 2018. https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b9359831

[51] PyTorch Team. Pytorch - alexnet, 2022. https://pytorch.org/hub/pytorch_vision_alexnet/.

[52] PyTorch. Pytorch - vgg16, 2023. https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html.

[53] Sik-Ho Tsang. Review: Vggnet — 1st runner-up (image classification), winner (localization) in ilsvrc 2014, 2018. https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a1

[54] Simonyan and Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. https://arxiv.org/abs/1409.1556.

[55] David Mack. How to pick the best learning rate for your machine learning project, 2018. https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2.

[56] The Open University of Israel. The oui-adience face image project, 2014. https://talhassner.github.io/home/projects/Adience/Adience-data.html.

[57] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks, 2018.

[58] Yiming Lin, Jie Shen, Yujiang Wang, and Maja Pantic. Fp-age: Leveraging face parsing attention for facial age estimation in the wild, 2021.

[59] Yiming Lin, Jie Shen, Yujiang Wang, and Maja Pantic. Fp-age: Leveraging face parsing attention for facial age estimation in the wild, 2021.

[60] Yiming Lin, Jie Shen, Yujiang Wang, and Maja Pantic. Imdb-clean: A novel benchmark for age estimation in the wild, 2023. https://github.com/yiminglin-ai/imdb-clean.

[61] Vento et al. Greco, Saggese. Effective training of convolutional neural networks for age estimation based on knowledge distillation, 2021. https://doi.org/10.1007/s00521-021-05981-0.

[62] Trever Pedersen. How to use "model stacking" to improve machine learning predictions, 2021. https://medium.com/geekculture/how-to-use-model-stacking-to-improve-machine-learning-predictions-d113278612d4.

[63] Xintao Wang, Yu Li, Honglun Zhang, and Ying Shan. Towards real-world blind face restoration with generative facial prior, 2021.

[64] Freedom VC. Color image histograms, 2021. https://www.freedomvc.com/index.php/2021/09/11/color-image-histograms/,

[65] Hanyang Peng and Shiqi Yu. Yunet. https://github.com/opencv/opencv_zoo/tree/master/models/face_detection_yunet.

[66] OpenCV. Basic operations on images, 2022. https://docs.opencv.org/3.4/d3/df2/tutorial_py_basic_ops.html.

[67] Activeloop. Deep lake - data lake for deep learning, 2022. https://www.deeplake.ai/.

[68] Deep Lake. Deep lake - adience dataset, 2022. https://datasets.activeloop.ai/docs/ml/datasets/adience-dataset/.

[69] 7-Zip. 7-zip, 2022. https://www.7-zip.org/.

[70] The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022. https://www.mathworks.com.

[71] Rahmad Sadli. How to create a custom dataset class in pytorch, 2023. https://machinelearningspace.com/creating-custom-dataset-class-in-pytorch/.

[72] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[73] PyTorch Team. Pytorch - transforming and augmenting images, 2023. https://pytorch.org/vision/stable/transforms.html.

[74] Nash et al. Jones, J.A.B. The ageguess database, an open online resource on chronological and perceived ages of people aged 5–100, 2019.