The BRITISH UNIVERSITY IN EGYPT

Faculty of Informatics and Computer Science

Artificial Intelligence

AGE Classification

(Age_**G**ender_**E**thnicity)

By: Bavly Benyamin 206637

Supervised By

Prof. Nahla Barakat

June 2023

Abstract

Human profiling has become a significant field of research interest because of its numerous practical uses. Demographic characteristics including age, gender, and ethnicity are estimated for their multiple uses in security, marketing, and health. These characteristics can be combined with other data to increase the matching precision of a primary biometric system, such as the age and gender, or they can be used to provide descriptive summaries of a person (e.g., old white male). This latter method is especially helpful in closing the semantic gap between human and machine explanations of the biometric data. Hence, in this project, the state-of-the-art age classification models are investigated and how age detection can be affected by other soft biometrics such as gender and ethnicity. After extensive research, it was found that there is a gap in optimizing a model that works on age classification on a high-variance dataset such as UTKFace, and with realistic age bins. As a result, this project aims to fill in that gap by first, curating a literature review of previous work. Second, developing a base model. Then, empirically deciding on realistic age bins. After that, trying several model architectures and learning methods (i.e. transfer learning, multitasking, and hierarchical learning). Finally, developing an optimized classification model, that classifies a person's **AGE** (**A**ge, **G**ender, and **E**thnicity). The results are pleasing. The optimal model, AGECNN, was able to achieve for age, gender, and ethnicity F1 scores of 73.27%, 80.25%, and 93.41% respectively. While the model is mostly optimized for age, its gender classification is competitive to the state-of-the-art. Ethnicity, however, can be optimized more, especially with data balancing and classifying the 'other' ethnicity label. Most importantly, the age classification results exceeds the state-of-the-art models using the same age categories.

Turnitin Report

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Bavly 206637
Assignment title:	Graduation project report-AI Part 1 (Moodle TT)
Submission title:	bavly_206637_dissertation
File name:	371_Bavly_206637_bavly_206637_dissertation_161976_86283...
File size:	5.84M
Page count:	108
Word count:	21,429
Character count:	138,832
Submission date:	11-Jun-2023 02:20AM (UTC+0200)
Submission ID:	2113316247

The BRITISH UNIVERSITY IN EGYPT

Faculty of Informatics and Computer Science

Artificial Intelligence

AGE Classification

(Age_Gender_Ethnicity)

By: Bavly Benyamin 206637

Supervised By

Prof. Nahla Barakat

June 2023

Acknowledgement

First and foremost, I am indebted to my supervisor, Prof. Nahla Barakat, whose expertise, guidance, and patience have been very important in shaping this research. Her insightful feedback, constructive criticism, and support have hugely contributed to the refinement of my ideas and the overall quality of this work. I am truly grateful for her mentorship and constant availability in assisting me.

I would also like to extend my thanks to the academic staff of the ICS faculty, whose commitment to passing on their knowledge has been invaluable. Their lectures and labs have greatly helped with my understanding of the field of AI and deep learning.

Furthermore, I would like to thank my family for their love, encouragement, and support throughout my academic journey. Their belief in my abilities and constant motivation have been a driving force behind my accomplishments. I am truly grateful for their sacrifices, understanding, and unwavering faith in me.

Lastly, I would like to express my sincere appreciation to my friends who have been a constant source of support and encouragement. When things got tough, and I felt like giving up, they kept pushing me forward, and rooting for my success.

Table of Contents

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

Abbreviation	Definition
AGE	Age, Gender, and Ethnicity
CNN	Convolution Neural Network
VGG	Visual Geometry Group
MLP	Multi-Layer Perceptron
MAE	Mean Absolute Error
ANN	Artificial Neural Network
DSSAE	Deep Supervised Sparse Autoencoder
DAG	Directed Acyclic Graph
CMT	Conditional Multi-Tasking
MTCNN	Multi-Task Convolution Neural Network
BEFA	Bias Estimation in Face Analytics
ECCV	European Conference of Computer Vision
CIFAR	Canadian Institute for Advanced Research.
ILSVRC	The ImageNet Large Scale Visual Recognition Challenge
ResNet	Residual Neural Network
MBConv Block	Inverted Residual Block
SGD	Stochastic Gradient Descent
RMSProb	Root Mean Squared Propagation
Adam	Adaptive Moment Estimation
NAdam	Nesterov-accelerated Adaptive Moment Estimation
ReLU	Rectified Linear Unit
Tanh	Hyperbolic tangent function
SWA	Stochastic Weight Averaging
LR	Learning Rate
CDN	Content Delivery Network
GUI	Graphical User Interface
IDE	Integrated Development Environment
GPU	Graphical Processing Unit
API	Application Programming Interface
VSCode	Visual Studio Code
TMUX	Terminal Multiplexer

1 Introduction

1.1 Overview

Human profiling in this document refers to the process of gathering a group of traits that reveal some information about a person but cannot be used to identify them on their own, usually because they are not distinctive or lasting such as a person's gender, age, and ethnicity [1]. The ageing process of the face often follows a few typical ageing modes. The shape of a child's skull changes significantly during the growing period. This is due to the expansion of the skull. Adult ageing is mostly manifested by changes in the texture of face skin, such as the emergence and deepening of wrinkles, loose skin, and a rise in spots. But because ageing is a long process and face characteristics are complex, ageing severity is affected by several factors in addition to age, including gender, ethnicity, heredity, lifestyle choices, and health [2]. Despite the above hurdles, face age estimation technology has numerous potential applications in surveillance, information systems, smart human-computer interaction, social entertainment, and other domains.

Age classification is frequently used to enforce legal requirements and regulations to ensure compliance with the law. Age limitations may be imposed by law on certain goods, services, or activities to safeguard people—particularly children—from potential harm. For instance, only people over a specific age may purchase alcohol, cigarette items, or adult content. Protecting children's safety and well-being is made easier by using age groups. It enables the detection and prevention of children being exposed to unsuitable content, such as violent, pornographic, or risky internet content. Platforms can put in place the necessary precautions to protect minors by validating age. Advertisers frequently utilise age categorization to target populations in their advertising. Marketers can make their commercials more relevant and enticing by knowing the age of their audience. This can boost return on investment and the efficacy of advertising initiatives. Content Rating and Filtering: Age classification is used to label and filter content across various media forms, such as movies, video games, and websites. This aids with the decision-making process for people, parents, and guardians regarding the appropriateness of the content for various age groups. Systems for grading content offer direction and encourage media use responsibly.

To summarize, Age classification is essential for demographic studies, sociological research, and the collection of statistical data. Understanding a population's age distribution aids in trend analysis, policy formulation, and meeting the unique requirements of various age groups in areas like social services, healthcare, and education.

1.2 Problem Statement

Given a facial image of a person, generate a profile that includes the person's **AGE** (Age, Gender, and Ethnicity).

1.3 Scope and Objectives

Improve the performance of age classification while secondarily predicting gender and ethnicity and investigate how they affect age classification.

1.4 Report Organization (Structure)

Chapter 1 gave an overview for the topic and the motivation being it. Chapter 2 goes through the latest related work, starting with a background about the problem and how it is generally solved, then going into detail about different implemented solutions. Chapter 3 comes the step of designing the solution, this includes gathering and choosing a suitable dataset, discussing different metrics to measure the model's performance, pre-processing steps, several deep learning architectures, hyperparameters, and other techniques. Once solution design is done, chapter 4 is implementing the solution by first preparing the environment, including the training platform, framework and ensuring a reproducible environment. Then explain the code highlights most relevant to the results. Chapter 5 are listed, each with their new hyperparameters and performance. In the chapter 6, the results are analysed and discussed. Finally, chapter 7 concludes the dissertation with a summary of contributions and what was learned throughout the project.

1.5 Work Methodology

After comparing the latest work, the strengths and weaknesses of each solution will be investigated, then critically attempt to come up with an innovative solution that outperforms the previous ones. After that, the design phase where all the framework and learning techniques research happens. Then the model will be implemented, tested, validated, and improved multiple times until a higher performance is achieved. Finally, the results will be discussed, and the report will be concluded.

1.6 Work Plan (Gantt chart)

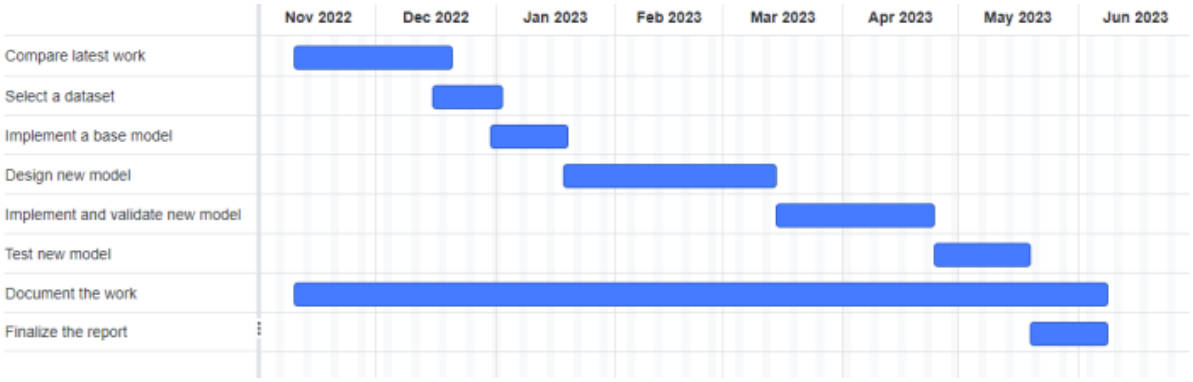

Figure 1 Gantt chart of how long each work item will take to complete.

2 Related Work (State-of-The-Art)

2.1 Background

In computer vision, age estimation is an essential and difficult endeavour. It is the process of determining a person's actual, apparent, or age range from a facial picture [3]. In recent years, numerous age estimation algorithms have been developed, and numerous age estimation methodologies have been published. In addition, researchers have been working to develop models that can detect a person's age or age group based on a variety of biometric characteristics, as these variables can influence the training of an age-detection model [4]. Age estimation of faces involves image pre-processing, feature extraction, and age estimation. Face recognition, face correction, and image cropping are all examples of image pre-processing [5]. Before training a neural network, image augmentation techniques can be used to enrich the dataset and prevent network overfitting [6]. Image enhancement techniques include sharpening, histogram and filtering optimisation, rotation, scale and flipping modification. Standard methods in the step of feature extraction rely heavily on explicit feature extraction to acquire age-based characteristics. Due to the limitations of hand-designed features, extracted age characteristics are not always ideal. [7]

The current feature extraction method based on CNNs can capture feature information that is related to faces, and features information from an image and is highly robust to image noise, resulting in a more accurate final age estimation step [8]. When age is considered a distinct designation, estimating age becomes a classification issue. In addition, because the age of the human visage can be expressed numerically, age estimation can also be viewed as a regression problem. This initiative treats age as a classification problem involving various bins (0-2, 3-9, etc.). Next, we will discuss several earlier works.

2.2 Literature Survey

2.2.1 Gender-specific Facial Age Group Classification Using Deep Learning

Raman et al. [9] states that while there have been research investigating the relationship between gender and age, few have looked into the concept of a gender-based system consisting of using separate models , each one trained on a specific gender. In order to close this disparity, their research introduces a two-component model for estimating age. The first component is a gender classifier that is specifically designed to distinguish precisely between females and males. The second component consists of two models for estimating age: Model A, trained exclusively on male images, and Model B, trained exclusively on images of females. The system receives an image as input, determines the gender from the face using a classifier, and then sends the image to the corresponding model depending on the

predicted gender classification. To accommodate the character of their problem, they have modified the VGG16 networks depicted in architecture for their age estimation models. Individually, these models accomplish greater than 85% accuracy, while the overall system achieves 80% accuracy. It is important to note, however, that they have only used four age groups, one of which is 20-59.

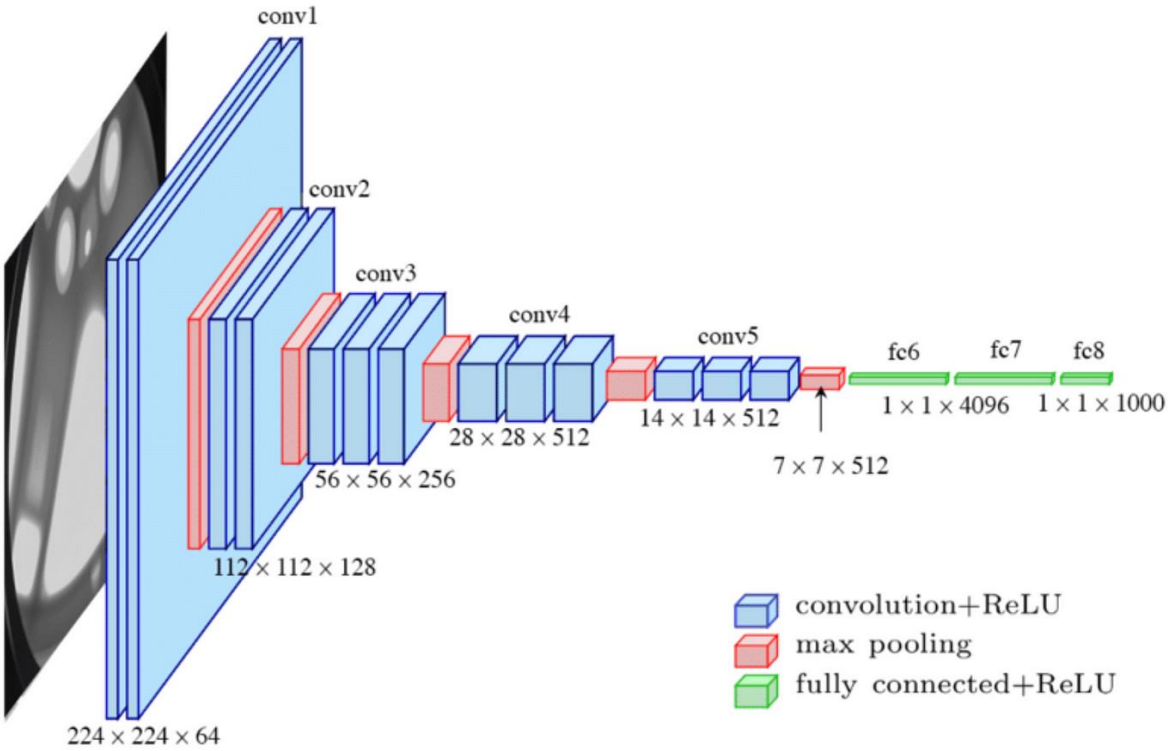

Figure 2 VGG16 architecture [9]

2.2.2 Age and gender classification in the wild using deep attention

Rodriguez et al. [10] proposed a novel model based on CNNs for identifying gender and age in a random environment with obstacles, disorder, obstruction, and distortions, i.e. in the wild. CNN is extremely sensitive to changes in facial images, so its accuracy is still not optimal. The authors identified the most discriminative low-resolution regions using a feedforward strategy and then converted them to high resolution. In addition, this method is resistant to noise and distractions because it prioritises the least distorted and least obscured areas of the image. The model performs well not only on unconstrained face image databases like Adience as well as Images of Group (IOG), but also on controlled face image databases like Morph-II. Three principal modules are used to construct the model: MLPs, CNN with attention, CNN patching. All training images are initially sent to the CNN with attention, which predicts the optimal attention grid. Patch CNN is then fed high-resolution patches determined by the optimal attention grid. As depicted in Figure 3, the classifier is then supplied with selected features from the attention CNN and patch CNN for MLP classification. TensorFlow is used to implement the VGG16 CNN architecture for all experiments. Mean absolute error (MAE) and accuracy are used to evaluate performance. The authors also discussed the effect of several design choices,

including mode of attention, sharing of weights, merging modes together, and the attention grid, and patch depth, on the proposed study endeavour. Without attention, the accuracy on the Adience dataset is 57.8% and MAE 4.9 and with attention, it is 61.8% and MAE 2.1. On the IoG dataset, the attention mode accuracy is 60% and the mean absolute error is 2.56.

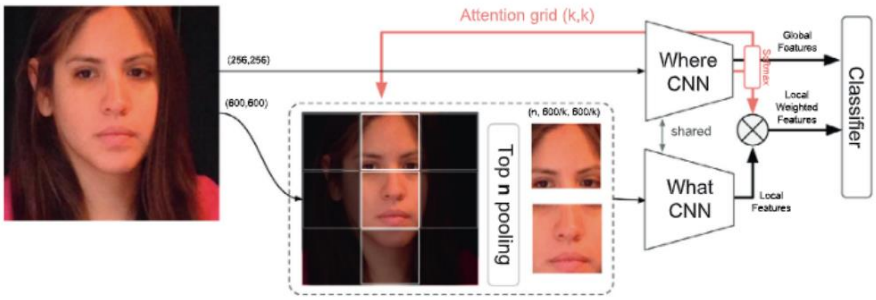

Figure 3 Attention model by Rodriguez et al [10]

2.2.3 Age estimation using autoencoders

Zaghbani et al. [11] investigated age estimates using Autoencoders. Autoencoders are a form of ANN used for unsupervised system learning. It has an input layer, one or more hidden layers, and an output layer as part of its architecture. The authors suggested a Deep Supervised Sparse Autoencoder for estimating human age (DSSAE) as shown in Figure 4. The proposed work is detailed in two stages. The Adaboost method is utilised to extract and crop pictures of faces, and a tan inverse formula is employed to rotate these extracted faces in-plane. The first phase reports on data preparation, face identification, and in-plane head rotation. The second stage investigates the classification challenge using (DSSAE). These autoencoders regard hidden neurons as having the same activation probability as input and output neurons. To achieve this condition, the data is first compressed. The data is then examined for correlation and categorised based on this association. MAE was 3.34 on the MORPH database.

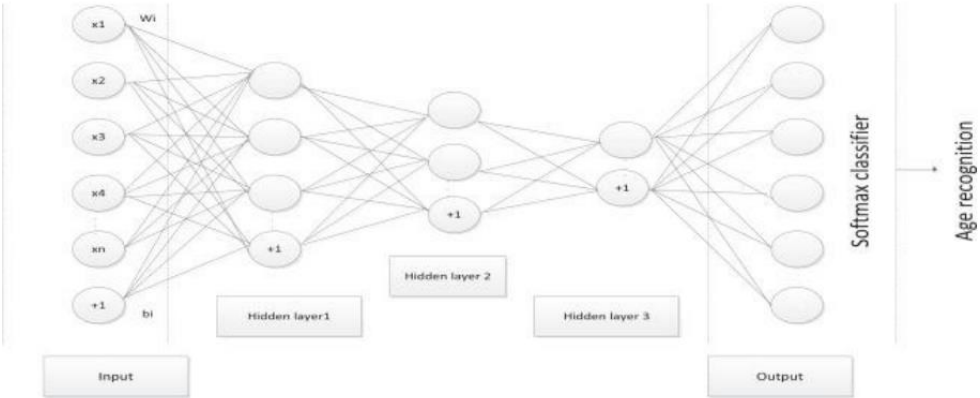

Figure 4 DSSAE Model [11]

2.2.4 Transfer learning for gender and age detection

Although VGG19 was not originally trained to detect faces, transfer learning techniques can nevertheless produce good results for gender recognition and age estimation. Input standardisation, data augmentation, and label distribution age encoding are all contrasted as training strategies by Smith at al. [12]. Finally, a deep CNN hierarchy is developed, which first classifies participants by gender and then predicts age using distinct models based on the classified gender as shown in Figure 5. Transfer learning using a more relevant to the task pretrained model, such as VGGFace, can provide results that outperform other algorithms, and can even outperform human performance. Changes in network designs and training approaches - 8 - can be examined without the need to train models from the start for weeks. This research has demonstrated the benefits of specific model designs, training approaches, and pretrained weights. It has been revealed that AI model hierarchies hold potential and are recommended to be considered while developing a classification system. The model reached a classification accuracy for gender of 98.7% and an MAE for age estimation of 4.1.

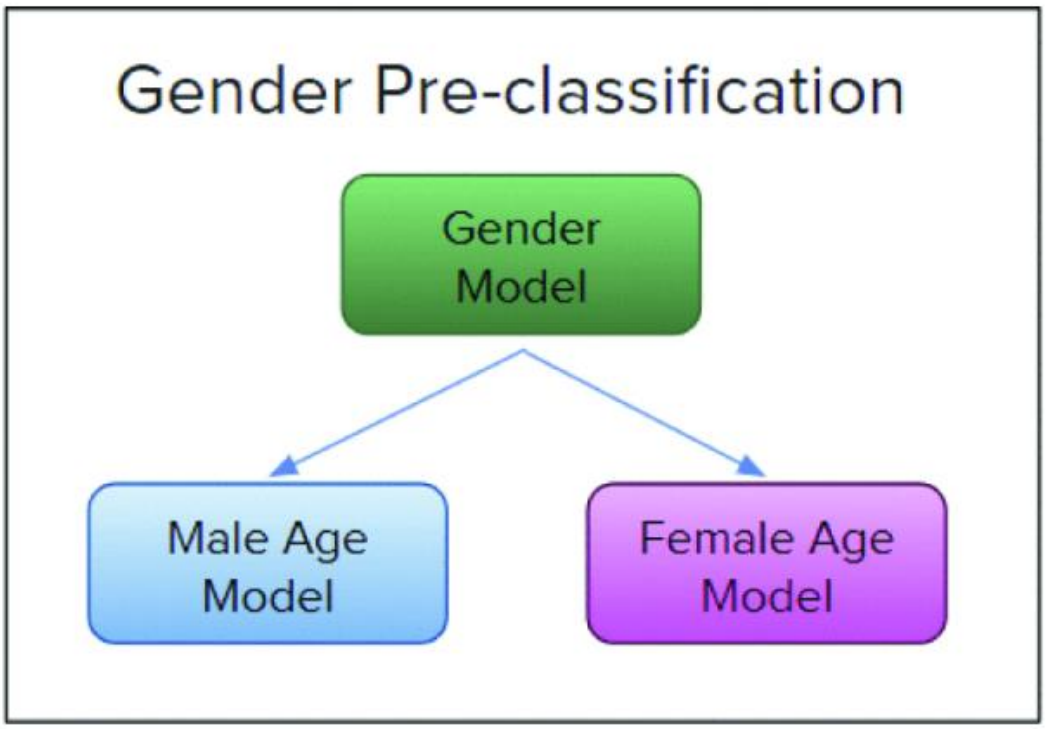

Figure 5 Different age detection model based on the classified gender [12]

2.2.5 Age estimation using DAG-CNN

Taheri and Toygar [13] utilised multi-stage characteristics from numerous layers of GoogleNet CNN [14] and VGG-16 CNN [15]. Directed Acyclic Graph Convolutional Neural Networks (DAG-CNN) is a novel architecture of CNNs for estimating human age that autonomously learns the distinguishing features of a the models and fuses these features using fusion Score-level. The proposed architecture is designed in two variants. DAG-GoogLeNet employs GoogLeNet CNN as the backbone structure

depicted in Figure 6, whereas DAG-VGG16 employs VGG-16 as the structure backbone. The FG-NET and MORPH-II databases are used for all experiments. All branches arising from the intermediate levels of the backbone structure are independently routed to layers of average pooling, normalisation, and fully conntected MLP layers. The total number of neurons across all final layers is identical to the number of age identifiers. Using Score-level fusion, the age is approximated in the decision layer after all score vectors have been combined. The DAG-GoogleNet model's experiment analysis reveals an MAE of 2.87 on the MORPH-II database and 3.05 on the FG-NET database. The DAG-VGG16 model's experiment analysis reveals an MAE of 2.81 on MORPH-II and 3.0 on the FG-NET database.

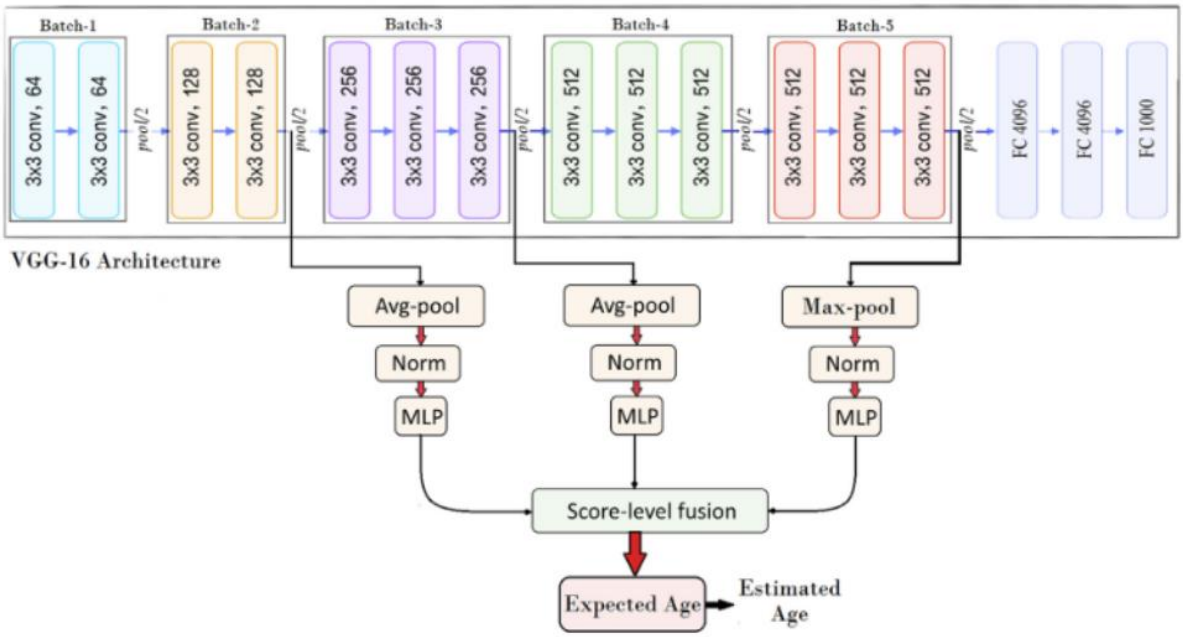

Figure 6 DAG-GoogLeNet proposed architecture [13]

2.2.6 Ranking CNN for age estimation

Shixing Chen et al. [16] offer a system that creates binary output for sub networks, which are then combined to generate labels for classification of age by using face photos as input. Independently, characteristics were discovered when developing the entire age range. As a result, distinct age class trends were observed, leading to an approximated evaluation. Labelled data was utilised to avoid overfitting, and every age group got trained separately. The authors have presented a stricter error restriction for age ranking, which is an average analysis of all mistakes evaluated by classifiers. This research article proved that one way to reduce binary error is to eliminate the classifier's final mistake and that for ranking CNN, a unique upper bound specifies precise error. The model was trained on the MORPH dataset with an MAE score of 2.96.

2.2.7 Age and gender estimation using conditional multitask learning with weak label expansion.

Conditional multitask (CMT) is a deep learning-based unified architecture that predicts a person's real age depending on gender. In June 2018, ByungIn Yoo et al. [17] created such a model, which incorporated weak label expansion as shown in Figure 7. Weak label expansion refers to the conversion of categorical age groups into discrete values. The authors also created a mean picture that is utilised as a test input to the CMT. All tests are carried out using the databases MORPH-II and FG-NET. The technique obtains an MAE of 3.46 on the FG-NET database and an MAE of 2.91 on the Morph-II database.

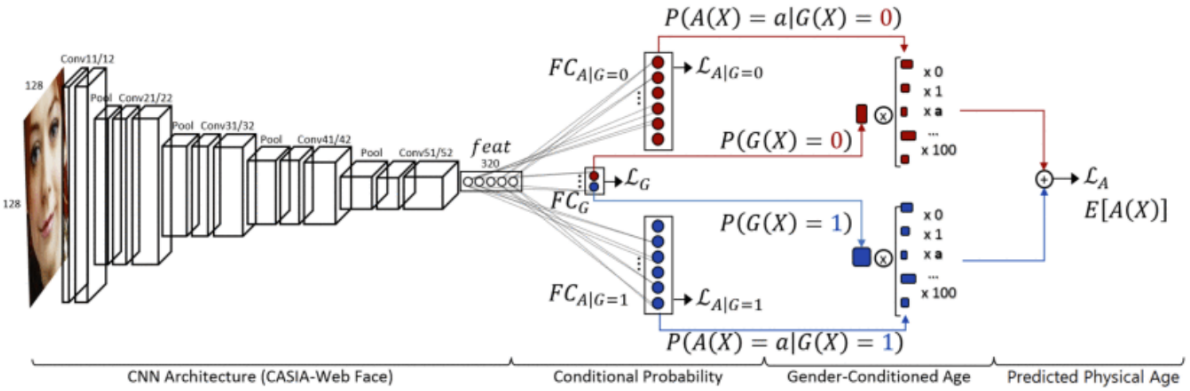

Figure 7 The MLT + weak label expansion model [17] [18]

2.2.8 CNN for fusion based multi-stage age estimation

The CNN model developed by Shahram Taheri and Onsen Toygar [18] includes handcrafted features and Multistage learned features of facial images from the FG-NET and MORPH-II databases, respectively. This model employs two techniques. At the score level, the first method combines feature vectors learned from multiple CNN layers. The second method incorporates multiple locally generated skin characteristics, wrinkles, and other biologically inspired aspects (BIFs). Before implementing feature level fusion, all of these manually crafted feature descriptors are standardised using Z-score, as depicted in Figure 8. On the MORPH-II database, the gender detection accuracy is 99.9%, while the

MAE for age is 3.04 on the MORPH-II database and 3.29 on the FG-NET database.

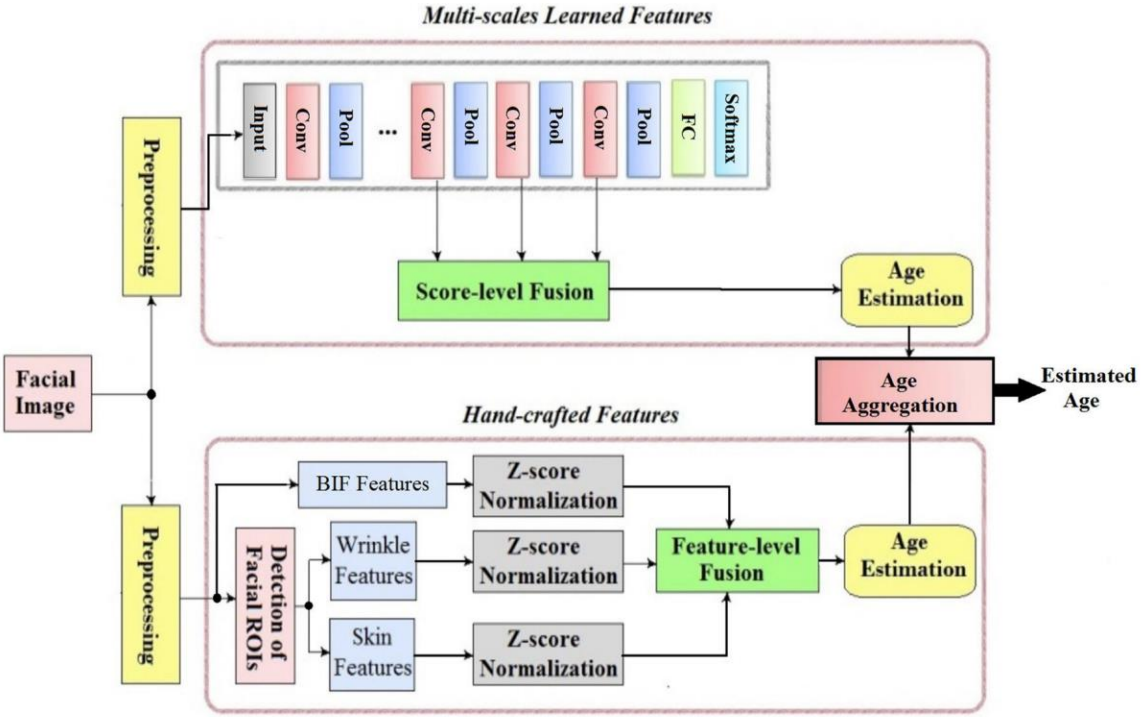

Figure 8 Fusion based model [18]

2.2.9 Age and gender detection using a multi-agent system.

González-Briones et al. [19] believed that there are common tasks between detecting a person's age and their gender. As such, they used a multi-agent system. A multi-agent system (MAS) is a system made up of several intelligent agents that coordinate, communicate, interact, and collaborate with one another. Multi-agent systems can tackle issues that would be difficult or impossible to handle with a single agent or a monolithic system [20]. They also attempted to measure the amount to which the - 12 - pre-processing step affects the effectiveness of classification systems (various face crops) and experimented with various filters to see whether they improve picture categorization accuracy. Another unique approach they took was designing and implementing the model for unconstrained images - images that do not have lighting or similar preconditions. Pre-processing consisted of taking the image, identifying the position of the eyes, rotating the image accordingly, and cropping the faces as shown in Figure 9. Then, both Bilateral and Gabor filters were used to account for different image environments e.g. sharpness and brightness

Figure 9 preprocessing the image by identifying the eyes and mouth then rotating [19]

Finally, the images were ready for classification. For gender classification, two techniques were used. The first one is fisherfaces [21] which eliminates effects that are due to different image lighting. The second technique was a multilayer perceptron where each image is inputted as a 64x64 array with pixel values normalised between 1 and -1. The network as such had 4096 inputs, a hidden layer with 4096 neurons, and 2 outputs - whether male or female. After that, for age classification, the classifier will be a Haar feature-based cascade classifier [22] that has been trained to distinguish human pupils in photos. The agents in the pre-processing layer will then decide whether the image needs to be aligned and compared to the Fisherfaces training phase, in which each of the acquired images will be compared, so that the image is classified based on how it adjusts to the model of the various training segments. This age range categorization technique is the same as the gender classification process above. The model achieves an average gender classification accuracy of 89.9% and age classification accuracy of 93.6%

2.2.10 Mitigating Bias in Gender, Age and Ethnicity Classification

Finally, Das et al. [23] delved into the simultaneous classification of gender, age, and race, aiming to address the challenge of soft biometrics classification. To tackle this, they proposed the utilization of a Multi-Task Convolution Neural Network (MTCNN) shown in Figure 10 that incorporates dynamic loss weight adjustment. This approach allows them to effectively classify named soft biometrics while also mitigating biases associated with these soft biometrics. The algorithm they introduced demonstrates promising performance on both the UTKFace and Bias Estimation in Face Analytics (BEFA) datasets. Notably, their proposed method secured the first rank in the BEFA Challenge, which took place during the European Conference of Computer Vision (ECCV) in 2018. They were able to get a 70.1% accuracy working with 7 age bins.

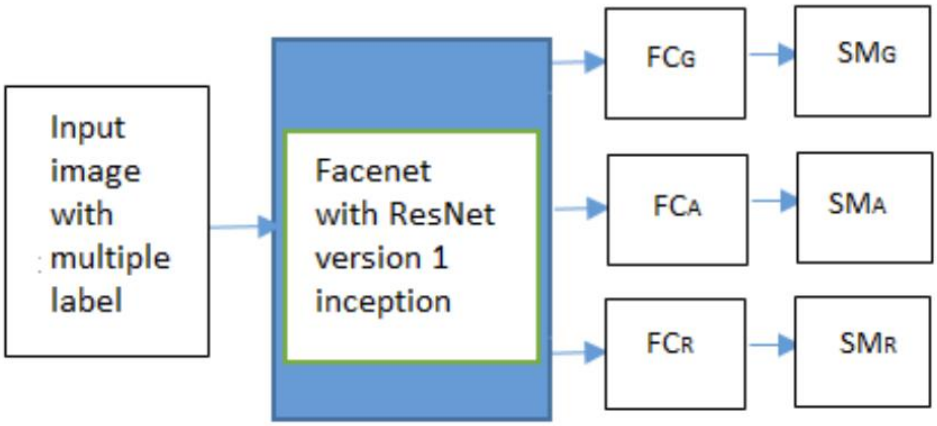

Figure 10 MTCNN architecture [23]

2.3 Analysis of the Related Work

Model	Metric	Value
Hierarchy VGG16 [8]	Accuracy	80.76%
Deep attention [10]	MAE	2.56
Deep supervised sparse autoencoder [11]	MAE	3.34
Transfer learning using VGG19 [12]	MAE	4.1
DAG-GoogleNet [13]	MAE	3.05
DAG-VGG16 [13]	MAE	2.81
Ranking CNN [16]	MAE	2.96
CMT with weak label expansion [17]	MAE	3.04
MTCNN [23]	Accuracy	70.1%

Table 1 Summarizing different models, their metrics, and their performances.

From Table 1 above, it can be concluded that most models are working towards age regression with MAE. Moreover, there is room for improvement in age classification, especially with harder datasets with a lot of variance such as UTKFace dataset [24], where some papers such as [8] and [24] attempted solving, though [8] had good results because they used much less realistic bins, as one of them is 20-59. As a result, there is a gap in optimizing a model that works on age classification on a high-variance dataset such as UTKFace, and with more realistic age bins. That, exactly, will be the purpose of this project.

3 Methodology and design

3.1 Dataset

While there are several datasets for age, very few of them contain age, gender, and ethnicity. As a result, UTKFace dataset was chosen for having these characteristics and having high variance as well, resulting in a harder challenge, but a more generalizing model. UTKFace is a large-scale face dataset with a wide age range (0 to 116 years old). The dataset contains over 20,000 200x200px face images with age, gender, and ethnicity labels.

The labels of each face image is embedded in the file name, formatted
like [age]_[gender]_[race]_[date&time].jpg

- [age] is an integer from 0 to 116, indicating the age
- [gender] is either 0 (male) or 1 (female)
- [race] is an integer from 0 to 4, denoting White, Black, Asian, Indian, and Others

For example, 28_0_4_20170117202327093.jpg.chip.jpg refers to an Indian 28 year old male.

The images offer a wide variety of in-the-wild poses, facial expressions, lighting, occlusion, and quality. This dataset may be utilised for a variety of applications, including face detection, age estimation, age progression/regression, landmark localization, etc. Several examples of images are shown in Figure 11

Figure 11 Samples from UTKFace dataset [24]

As mentioned before, the dataset contains a huge age range, starting from 0 to 116 years. As for the age distribution. The data is not completely balanced. As can be imagined, most images are of teens and middle-aged people, with decreasing number of images as age increases. This can be noticed in Figure 12

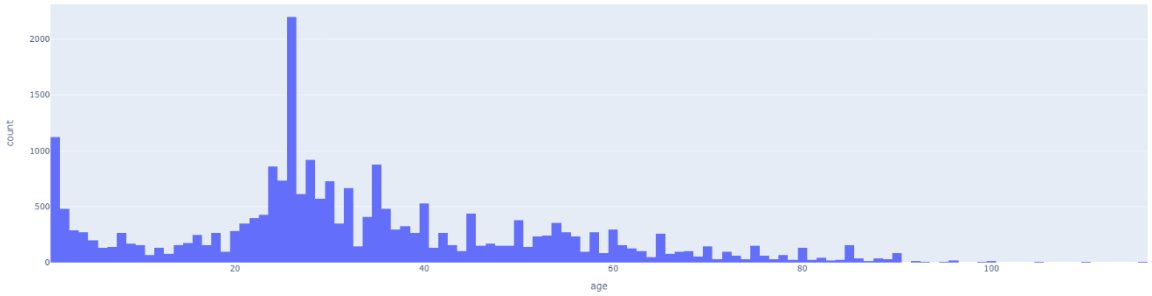

Figure 12 UTK age distribution

As for the gender, the dataset consists of almost-balanced images of both genders as shown in Figure 13

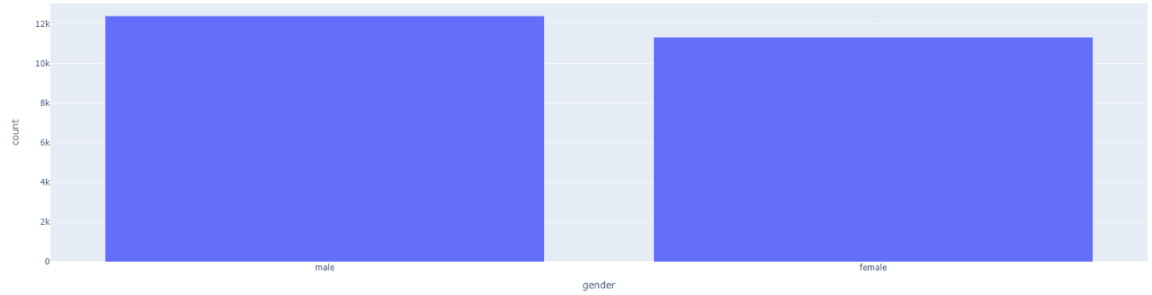

Figure 13 UTK gender distribution

Finally, the dataset is ethnicity rich. It has the following ethnicities: white, black, Asian, Indian, and other, which includes Hispanic, Latino, Middle Eastern, etc… The ethnicity distribution is seen in Figure 14. It is apparent that the dataset is not balanced since it contains mostly white ethnicity.

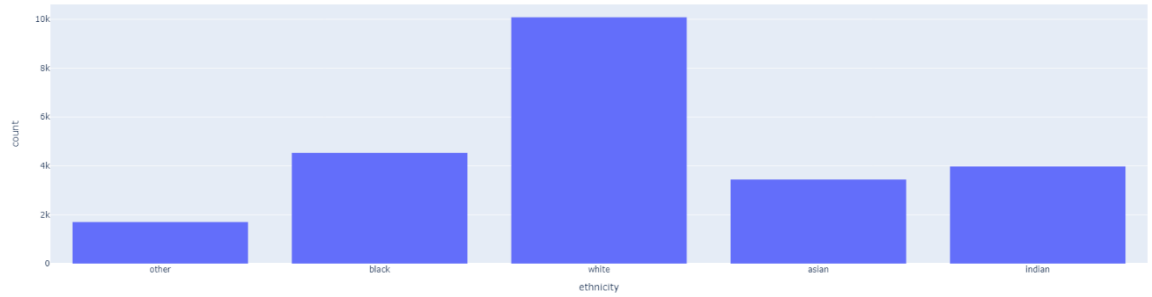

Figure 14 UTK ethnicity distribution

3.2 Pre-processing

There is no doubt that the previous datasets had its imbalances. Mainly in the age groups, since that is the primary focus of this project. As a result, a few steps are taken to counter that, and other problems, during the pre-processing of the data. The data was partitioned into 80/20 for training/testing respectively. The split was stratified on age to ensure both sets have the same age distribution.

3.2.1 Image size

The images are originally 200x200. Smaller images result in much faster training, since there are a lot less parameters to train. On the other hand, bigger images can be clearer for the model. So, finding that balance was critical While that worked well (neither too big, nor too small), many other variants were tried, including 32x32 (CIFAR dataset size), 128 (smaller than original, but not too small), 224(size for most pretrained models), and 400(bigger size images). It was concluded that 224 is the best size since it is the most used with pretrained models, and transfer learning was one of the techniques used in this project, as discussed later.

3.2.2 Data augmentations

Data augmentation is a method for augmenting (altering) the training set through the creation of modified duplicates of an existing dataset using existing data. It includes making slight adjustments to the dataset or generating new data points [26]. When the dataset is rich and exhaustive, models perform better and are more accurate. Data augmentation can help enhance the performance and results of deep learning models by adding new and diverse instances to training datasets.

There are several data augmentation techniques for images including:

- Translation
- Scaling up or down
- Rotating
- Random crop
- Gaussian blur
- Random brightness and contrast
- Greyscale (100% probability)

All the above augmentations were used in training the models, each with 0.5 probability of being applied which results in images never seen by the model before every single epoch. Shown in Figure 15 an example of data augmentation in the project's code.

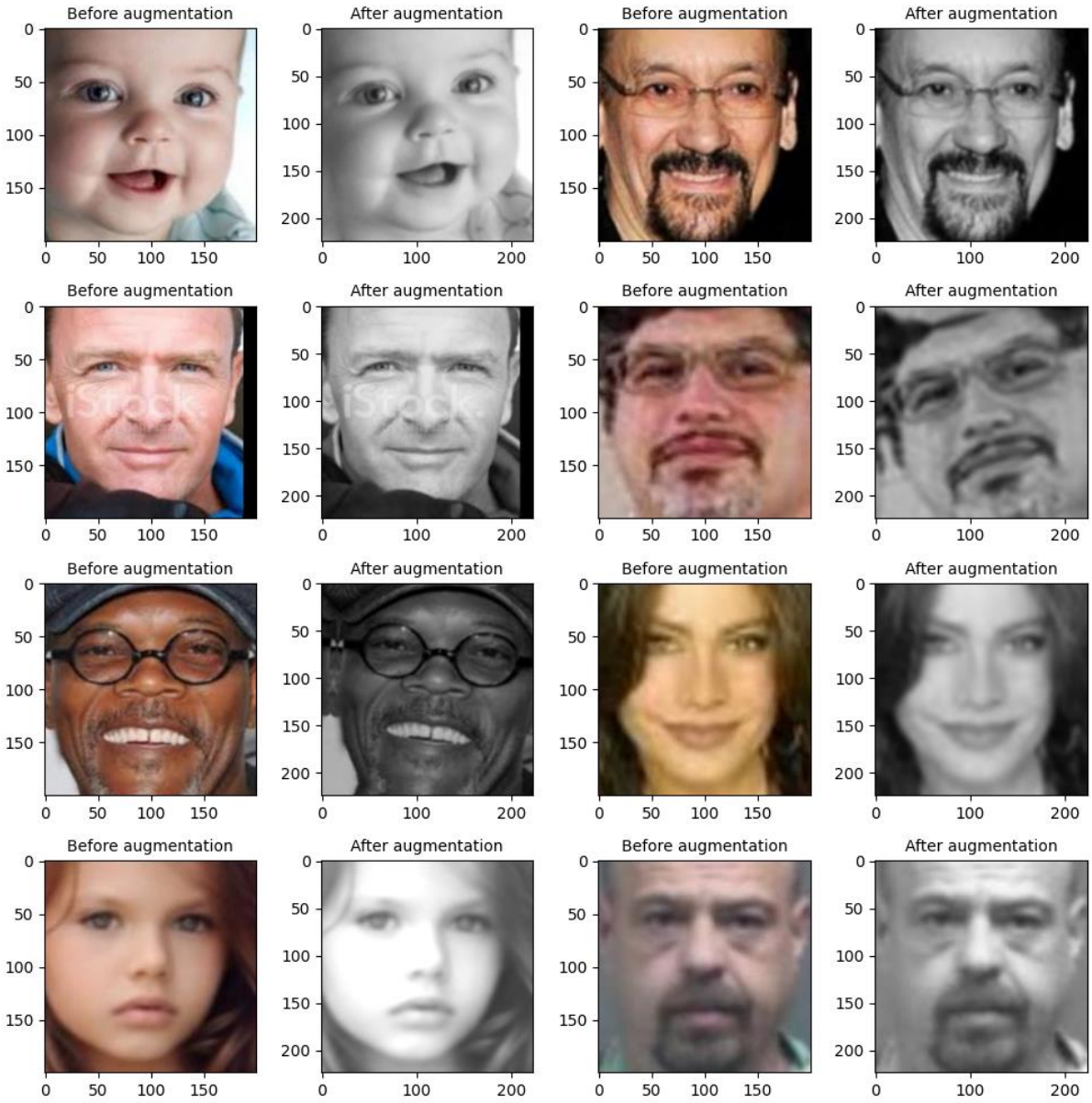

Figure 15 Example of augmentations in the code

3.2.3 Normalization

Normalisation and centering of image pixels are pre-processing techniques that intend to standardise an image's pixel values to make it more suited for further analysis or processing. That leads to adjusting the pixel values to a much narrower range, typically between 0 and 1 or -1 and 1. This procedure ensures that the pixel values are uniform across multiple images and reduces the impact of variations in illumination conditions or overall brightness [27]. By normalising an image, the relative relationships between pixel values are preserved, allowing algorithms to concentrate on more significant characteristics rather than absolute intensity values. When working with datasets that may have variations in overall luminance - such as UTKFaces - or when comparing images for similarity analysis, centering is particularly useful.

3.2.4 RGB vs Greyscale

In image classification, the choice between RGB and grayscale images depends on the specific task and dataset characteristics. While RGB images capture colour information and are useful when colour plays a crucial role, grayscale images are inherently resistant to colour variations and illumination inconsistencies [28]. They are appropriate when colour is unimportant or when they introduce commotion. After experimenting with both, it was concluded that greyscale is better for the purpose of this project.

3.2.5 Age categories/bins

This is one of the most important pre-processing steps in the project, which is grouping ages in specific bins. As mentioned before, age is treated as a classification problem in this project. In order to do that, ages will be grouped in different bins. It is important to balance between having enough bins so that age classification is still useful, but also not have too many that results in a poor performance. For example, both [8] and [23] worked on UTK, however [8] only used 4 bins, unlike [23] who used 7, which contributed to [8] performing better. After extensive testing, the chosen age categories are ['0-2', '3-6', '7-14', '15-30', '31-40', '41-65', '66-116']. Distribution of these categories is shown in Figure 16

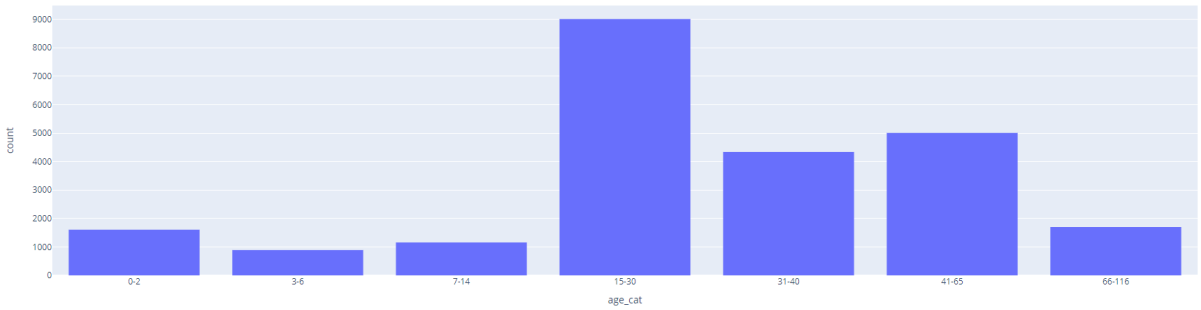

Figure 16 UTK age categories distribution

3.3 Used deep learning architectures.

3.3.1 Basic CNN

CNNs [29] are a form of deep learning algorithm ideally suited for image recognition and processing tasks. It consists of convolutional, pooling, and MLP. In the convolution layers of a CNN, filters are used to extract features such as edges, textures, and outlines from the input. The output of these layers is then fed into pooling layers, which are utilised to minimise the spatial dimensions of the map features while maintaining the most important information. As depicted in Figure 17, the output of the pooling layers is sent to one or more fully connected layers, which are used to generate an image prediction or

classification. The project involved three variants: Three-, four-, and five-layered CNNs

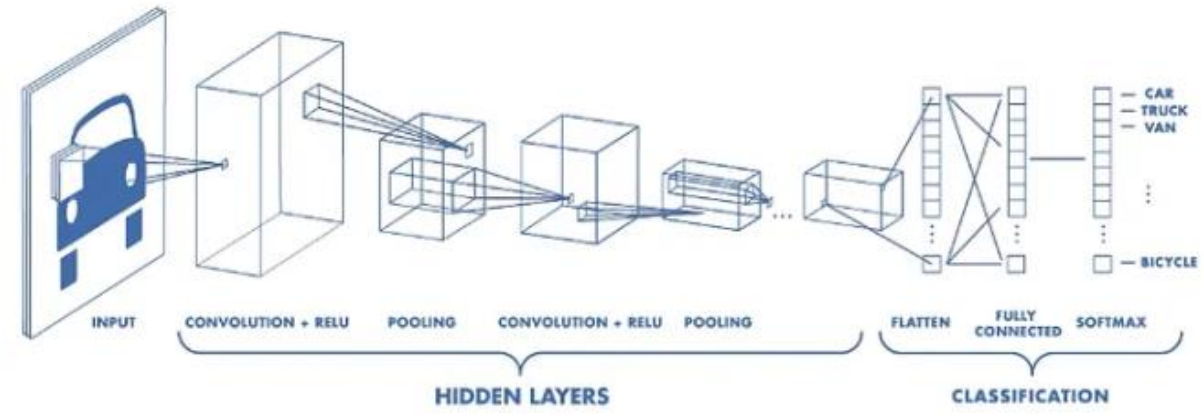

Figure 17 CNN Architecture [30]

3.3.2 VGG16

VGG16 [30] is a CNN proposed in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by K. Simonyan and A. Zisserman of the University of Oxford, a famous model submitted to the 2014 ILSVRC. It outperforms AlexNet by substituting large kernel-size filters (11 and 5 in the first and second convolutional layers, respectively) with multiple 3x3 kernel-size filters in succession as shown in Figure 18.

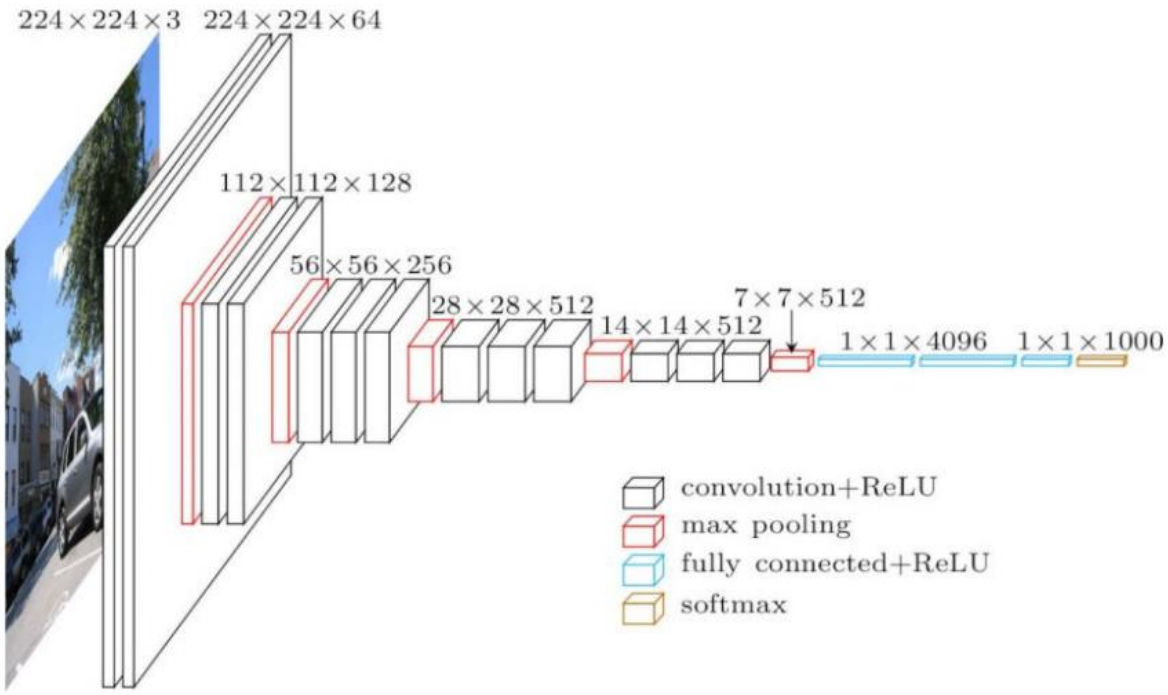

Figure 18 VGG16 architecture [31]

3.3.3 Resnet

CNNs needed to be deeper but there was no way to do that until 2015, when ResNet was created. The research describes a novel design for deep (CNNs) that tries to solve the problem of disappearing gradients, which arises when gradients are backpropagated across several layers of a network. According to the scientists, this limitation restricts the depth of CNNs, and hence their capacity to learn complicated features and attain state-of-the-art performance on image identification tasks. To address this issue, the authors [31] propose residual connections, which enable the network to learn residual functions rather than immediately translating inputs to outputs. Kaiming In their design ResNet, He et al. from Microsoft Research proposed the concept of "residual blocks," which are linked to one another by identity (skip) links. The following skip/shortcut link in fig. 5 is implemented to add the input x to the output after a few weight layers in order to address the issue of vanishing/exploding gradients. The result is H(x)=F(x) + x. The purpose of the weight layers is to discover a form of residual mapping, F(x)=H(x)-x. Two variants were used in this project: resnet50 and resnet152.

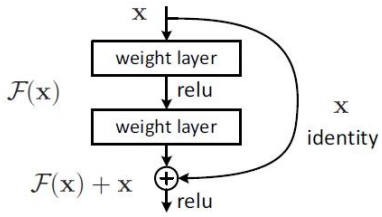

Figure 19 Residual block [32]

3.3.4 InceptionResnetV1

InceptionResNetV1[32] is a CNN architecture that incorporates the Inception module and Residual connections. As an extension of the original Inception architecture, it seeks to improve both the representational power and training process of deep neural networks. In the original Inception architecture, the Inception module was designed to capture multi-scale features by concatenating the outputs of different-sized filters (1x1, 3x3, 5x5) in parallel. This enables the network to extract features at various granularity levels. Inspired by the ResNet architecture, InceptionResNetV1 expands on this concept by incorporating residual connections as shown in Figure 20. Residual connections facilitate gradient flow and mitigate the problem of vanishing gradients, making it simpler to train deeper networks. Using residual connections, the InceptionResNetV1 architecture stacks multiple Inception modules to construct a deep network. The residual connections guarantee that information from earlier layers can be directly propagated to later layers, enabling the network to effectively learn both low-level and high-level representations. In addition, the use of 1x1 convolutions prior to broader convolutions reduces computational costs and increases efficiency.

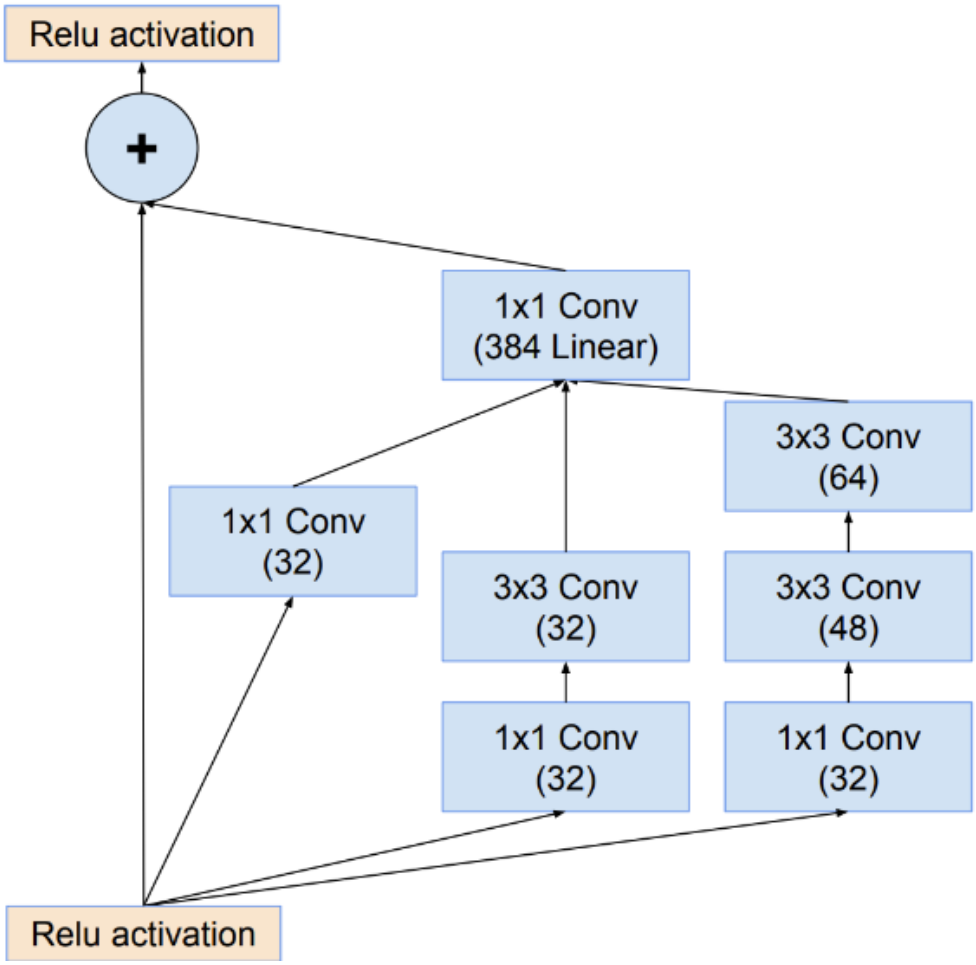

Figure 20 InceptionResNet block [33]

3.3.5 DenseNet

In 2016, another group of researchers introduced DenseNet [33], a novel architecture that maximises the benefits of shortcut connections. It connects all levels in an uncomplicated fashion. In this inventive design, each layer's input consists of the feature maps of all preceding layers, and its output is provided to each subsequent layer, as shown in Figure 21. To aggregate feature maps, depth-concatenation is employed. In addition to addressing the issue of vanishing gradients, the authors of "Aggregated Residual Transformations for Deep Neural Networks" propose that this architecture encourages the reuse of features, thereby making the network extremely parameter efficient. The output of the identity mapping was introduced to the subsequent section of the study on deep residual learning for image recognition and identity mappings in deep ResNets, which could limit the flow of information if the distributions of the feature maps of two layers are exceedingly dissimilar. Concatenating feature maps may therefore preserve all of them while increasing the outputs' diversity, thereby fostering feature

reuse. This project employs the DenseNet121 and DenseNet201 variants.

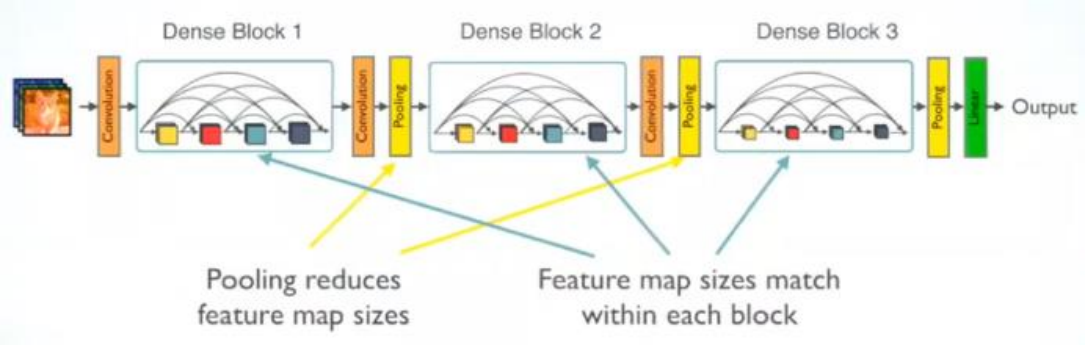

Figure 21 DenseNet Architecture [34]

3.3.6 EfficientNet

EfficientNet [34] is a family of (CNN) architectures designed to be highly efficient in terms of computational resources and model size, while attaining state-of-the-art performance on a variety of computer vision tasks. The EfficientNet models were introduced by Tan et al. in 2019 and have garnered a great deal of traction within the deep learning community. Compound scaling is the fundamental concept behind EfficientNet, which aims to achieve a balance between model depth, width, and resolution. This procedure scales the network's depth, width, and resolution uniformly and systematically. Models are scaled up using a compound coefficient, denoted "phi," which regulates the scaling of various dimensions. The compound scaling method enables EfficientNet models to attain greater precision while keeping computational demands and model size relatively low as shown in. By simultaneously scaling depth, width, and resolution, the models can effectively encompass both high-level and low-level features. In addition to mobile inverted bottleneck (MBConv) blocks, which consist of depthwise separable convolutions and squeeze-and-excitation blocks, the EfficientNet architecture incorporates additional techniques to improve performance and efficiency. These techniques reduce further the number of parameters and computations while maintaining or even enhancing the network's representational capacity. Variants used are EfficientNetS-L-M.

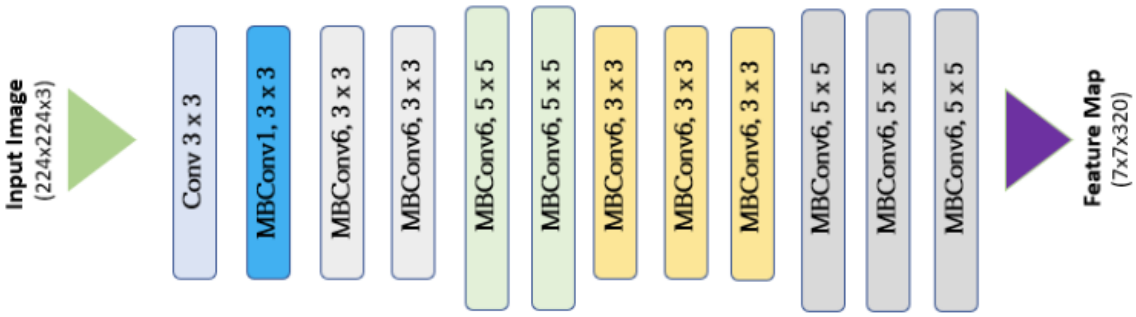

Figure 22 EfficientNet architecture [35]

3.4 Hyperparameters

3.4.1 Optimizer

Optimizers are algorithms employed in deep learning to update the model's parameters during training so as to minimise the loss function. The direction and magnitude of parameter adjustments are determined based on the gradients computed during backpropagation. Optimizers play a crucial role in deep learning because they aid in discovering the optimal set of parameters that minimises loss and improves the performance of the model.

3.4.1.1 Stochastic Gradient Descent (SGD)

is a foundational deep learning optimisation algorithm. It modifies the model's parameters by taking minor steps in the direction of the loss function's negative gradient. SGD [35] performs updates on random subsets of training data, making it computationally efficient for large datasets. It modifies the parameters iteratively to minimise the loss and enhance the model's performance. In the presence of sparse or noisy gradients, SGD can suffer from delayed convergence despite its simplicity and ease of implementation as shown Figure 23.

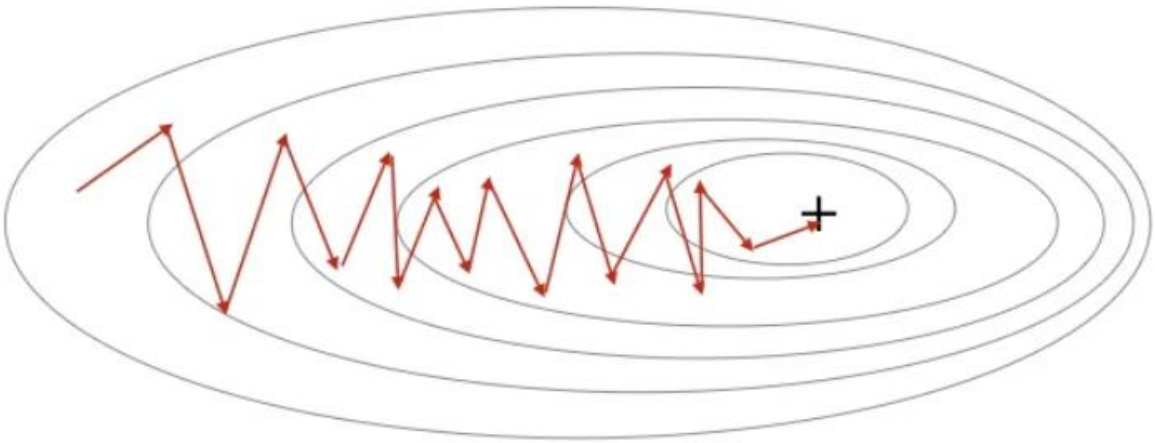

Figure 23 SGD navigating a loss function [35]

3.4.1.2 SGD with momentum

SGD with momentum is a commonly used optimisation algorithm in deep learning that improves upon Stochastic Gradient Descent (SGD) by adding a momentum parameter. Momentum aids in accelerating the convergence of the model and overcoming the limitations of SGD, particularly in situations with high curvature, minor or noisy gradients, or saddle points [36]. The momentum term adds a fraction of the previous parameter update to the current parameter update, enabling the optimisation process to accrue momentum in directions with consistent gradients over time as shown in Figure 24. This enables the model to make larger and more consistent updates, thereby accelerating convergence and enhancing

training efficiency. By incorporating momentum, SGD with momentum can more effectively navigate the loss landscape, avoid local minima, and converge on better solutions for deep learning tasks.

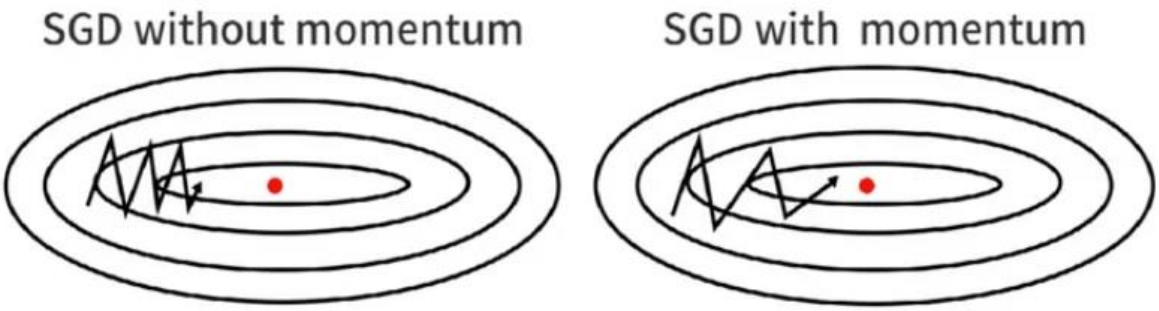

Figure 24 SGD vs SGD with momentum [36]

3.4.1.3 RMSprop

RMSprob [37] overcomes the limitations of SGD by adjusting the learning rate for every parameter. RMSProp divides the learning rate by the average of past squared gradients that decays exponentially as shown in Equation 1. By doing so, the learning rate is effectively scaled down for parameters with large gradients and scaled up for parameters with minor gradients. This aids in accelerating convergence and adjusting to various features and gradients. Adjusting the learning rate dynamically based on the history of squared gradients, RMSProp enables more stable and efficient training of deep learning models in situations with sparse or noisy gradients.

$$s \leftarrow \beta s - (1-\beta)\nabla_\theta J(\theta)^2$$

$$\theta_{nextstep} \leftarrow \theta + \frac{\eta \nabla_\theta J(\theta)}{\sqrt{s+\epsilon}}$$

β: decay rate, typically set at 0.9

s: exponential average square of past gradients

Equation 1 RMSProb updates [37]

3.4.1.4 Adam (Adaptive Moment Estimation)

Adam combines the advantages of RMSProp and momentum techniques. Adam maintains averages of prior gradients and gradients squared that decay exponentially [38]. Using these moving averages, it modifies the learning rate for each parameter. The algorithm includes both the momentum term, which helps to accelerate convergence, and the adaptive learning rate, which allows for the automatic adjustment of the learning rate based on the gradient magnitudes as shown in Equation 2. Adam's adaptable nature enables him to perform well on a broad variety of deep learning tasks, delivering rapid convergence and robust optimisation. It can better deal with sparse gradients, noisy data, and varying learning rate requirements for various parameters

$$m \leftarrow \beta_1 m - (1 - \beta_1)\nabla_\theta J(\theta)$$
$$s \leftarrow \beta_2 s - (1 - \beta_2)\nabla_\theta J(\theta)$$
$$\hat{m} \leftarrow \frac{m}{1-\beta_1^T} ; \hat{s} \leftarrow \frac{s}{1-\beta_2^T}$$
$$\theta_{nextstep} \leftarrow \theta + \frac{\eta \hat{m}}{\sqrt{\hat{s}+\epsilon}}$$

η: learning rate
s: exponential average square of gradients
m: momentum vector
β1: momentum decay, typically set at 0.9
β2: scaling decay, typically set at 0.999
ϵ: smoothing term

Equation 2 Adam updates [38]

3.4.1.5 NAdam (Nesterov-accelerated Adam)

NAdam incorporates the concept of Nesterov momentum. It includes the advantages of Adam's adaptive learning rates with the enhanced convergence characteristics of Nesterov momentum [39] shown in Equation 3. The momentum update is modified by Nadam to include the gradient lookahead technique. When updating the parameters, this enables the optimizer to look ahead in the direction of the momentum-adjusted gradient. By utilising this lookahead, Nadam seeks to improve convergence near the minima and provide faster and more stable optimisation than Adam. It is particularly useful in situations where the optimisation landscape is complex or contains flat regions, as it improves the model's ability to discover superior solutions and achieve enhanced performance.

$$m \leftarrow \beta m - \eta * \nabla_\theta J(\theta + \beta m)$$
$$\theta \leftarrow \theta + m$$

Equation 3 Momentum using Nesetrov [39]

3.4.2 Learning rate

In deep learning, the learning rate is a crucial hyperparameter that determines the step size of the model's parameter updates during training. It is essential for achieving convergence, training, and generalisation performance. A well-selected learning rate strikes a balance between slow convergence and overshooting as shown in Figure 25, thereby facilitating the optimisation process and allowing the model to come up with the optimal solution. It affects both the speed and quality of training, and improper calibration can result in sluggish convergence, unstable training, or suboptimal outcomes.

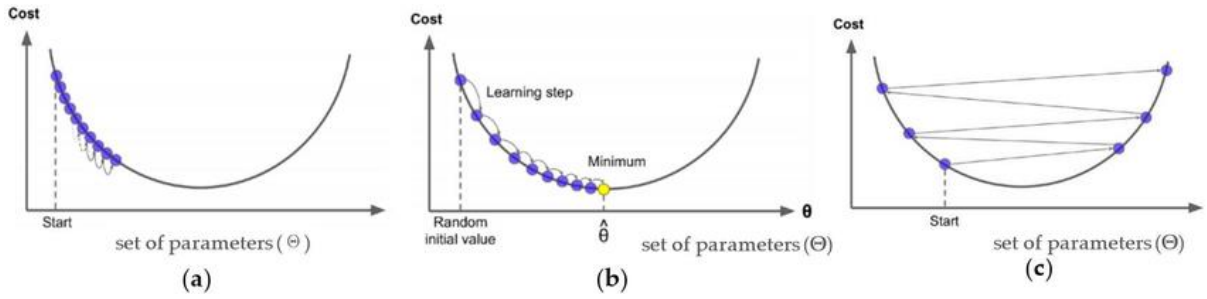

Figure 25 Importance of a just-right learning rate [40]

3.4.3 Activation functions

Activation functions are crucial components of deep learning that introduce non-linearity and allow neural networks to learn complex data patterns and relationships as shown in . They are applied to the outputs of individual neurons, transforming them into representations with greater expressiveness. Sigmoid, tanh, ReLU [41], Leaky ReLU, and softmax [42] are typical activation functions. Sigmoid and tanh functions provide smooth nonlinear transformations, whereas ReLU and its derivatives solve the problem of vanishing gradients and enhance gradient propagation. Softmax is used for classification assignments with multiple classes. The selection of activation function is dependent on the nature of the problem and the data.

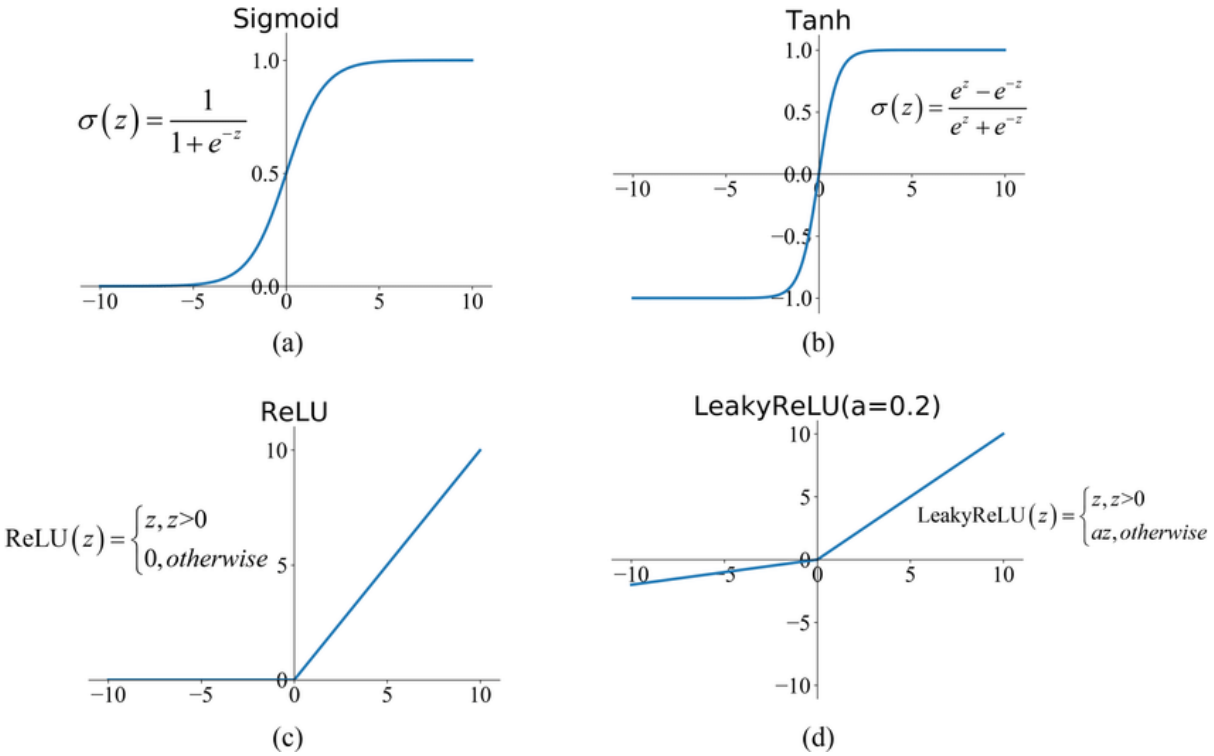

Figure 26 Activation functions [42]

3.4.4 Learning rate scheduler

Learning rate schedulers are techniques used in deep learning to modify the learning rate during training in a dynamic manner. Adapting the learning rate based on predefined rules or heuristics, in an effort to increase training efficiency and model performance. Step-based, exponential, polynomial, and cyclical schedulers are popular methods shown in Figure 27. Learning rate schedulers enable deep neural network training to surmount obstacles and achieve a balance between convergence speed and fine-tuning [44]. In conjunction with other regularisation techniques, they are essential for optimising model performance and attaining better convergence. Learning rate schedulers offer a flexible and dynamic mechanism for optimising and enhancing the overall performance of deep learning models.

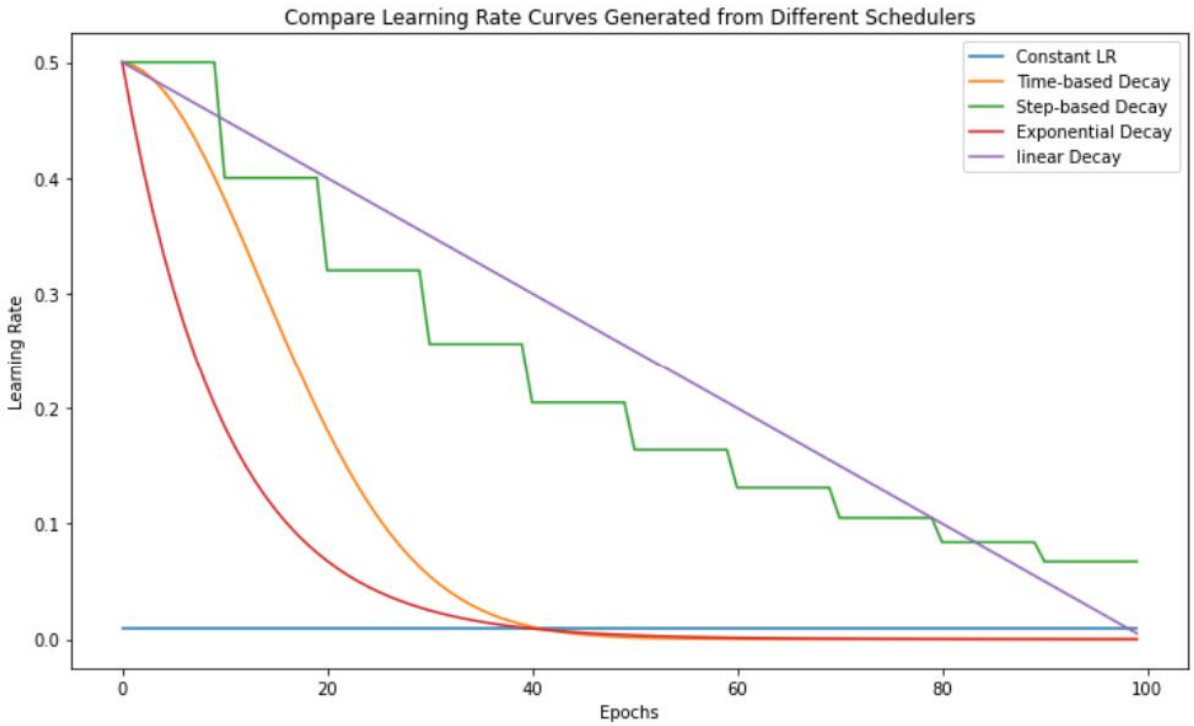

Figure 27 Different LR Schedulers [42]

3.4.4.1 ReduceLROnPlateau

ReduceLROnPlateau is a learning rate scheduling technique used in deep learning to adjust the learning rate based on the behaviour of a monitored metric, such as validation loss. It automatically reduces the learning rate when the monitored metric stops improving or plateaus over a certain number of epochs. By fine-tuning the learning rate in response to the model's performance, ReduceLROnPlateau helps prevent overfitting and aids in finding an optimal learning rate for the task at hand.

3.4.4.2 OneCycleLR

The OneCycle scheduling policy involves modifying the learning rate dynamically and cyclically during training [45]. The policy comprises of two phases: an increasing learning rate phase and a decreasing learning rate phase. During the increasing phase, the learning rate steadily increases to its

maximum value, in aim of faster convergence and loss landscape exploration as shown in Figure 28. Then, in the decreasing phase, the learning rate is progressively decreased, allowing for more precise adjustments and model fine-tuning. This cyclical behaviour prevents overfitting, enhances generalisation, and accelerates training by balancing exploration and exploitation of the learning rate.

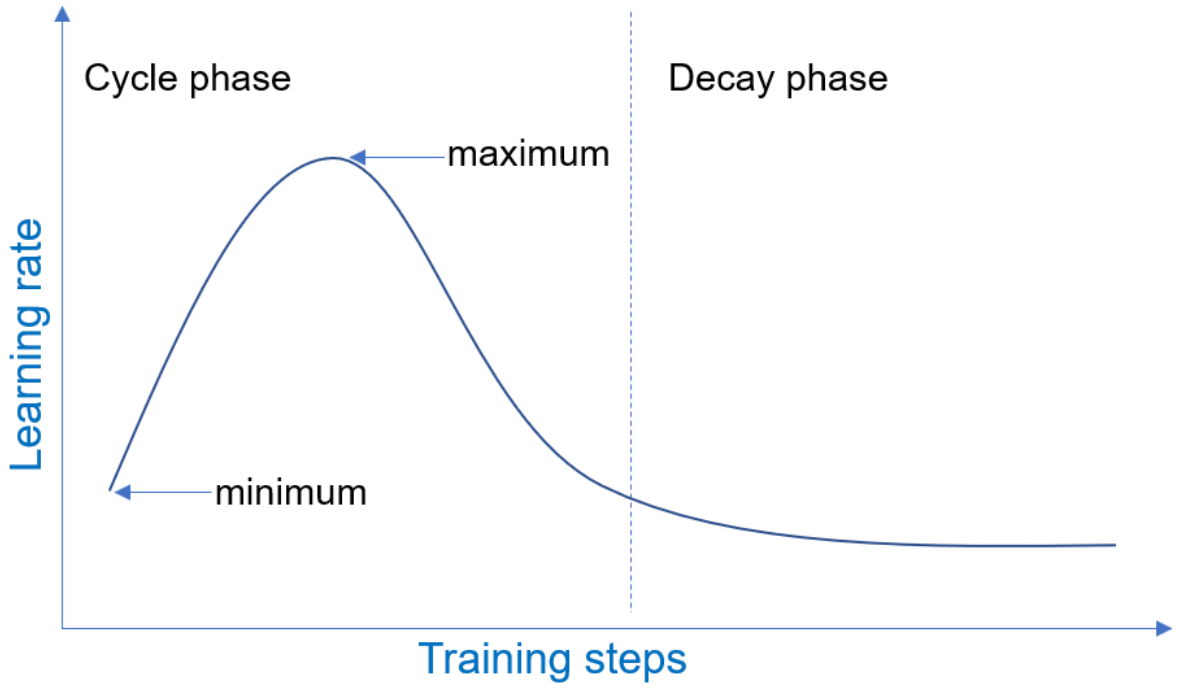

Figure 28 OneCycle policy [43]

3.4.5 Batch size

Batch size is an essential deep learning hyperparameter that influences the neural network's efficiency, convergence, and generalisation. The batch size determines the number of training examples evaluated in each iteration prior to parameter updating. The batch size influences the precision and consistency of the learning process; therefore, the learning rate must be adjusted accordingly. Due to a larger sample size, gradient estimates become more accurate when using larger batch sizes, enabling smoother updates in the parameter space. In such situations, a higher learning rate can be used because the gradients have less variance, allowing for quicker convergence. Due to the limited number of samples, gradient estimates become noisier with lower batch sizes, which can result in more erratic updates. Consequently, a slower learning rate is frequently advised for achieving stable convergence. In addition, batch size influences generalisation, as larger batches tend to provide steadier updates, whereas smaller batches may provide more diverse and representative samples. Determining the optimal batch size requires taking into account the available computational resources, the characteristics of the dataset, and striking a balance between computation efficiency and model performance.

3.4.6 Number of hidden layers

The depth or number of sequential layers between input and output layers is determined by the number of hidden layers. Each hidden layer applies a series of nonlinear transformations to the input data, enabling the network to acquire increasingly abstract and high-level representations. Including additional concealed layers enables the model to represent complex data relationships and hierarchical structures shown in Figure 29. Nevertheless, an excessively deep network may result in overfitting, an increase in computational complexity, and gradient vanishing or explosive issues. It is essential to strike the proper equilibrium between depth and generalisation, as too few layers can lead to underfitting and limited representational power.

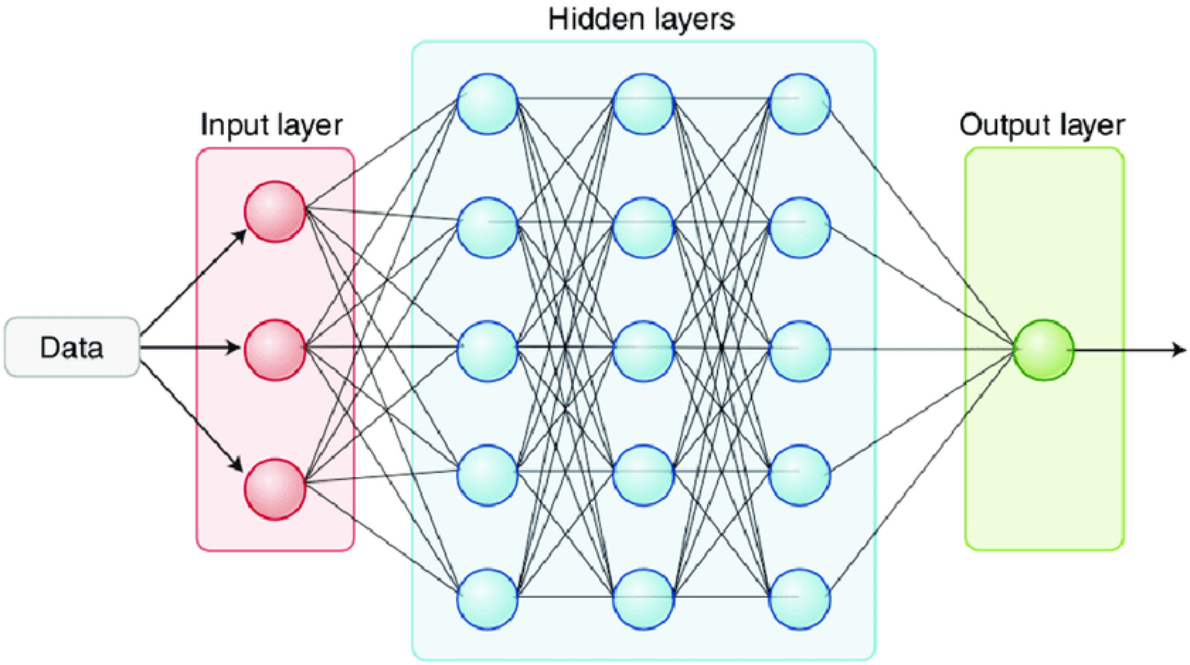

Figure 29 Hidden layers and neurons [46]

3.4.7 Number of hidden neurons

In each layer, the number of hidden neurons corresponds to the number of learnable parameters or activation units. Increasing the number of hidden neurons increases the model's capacity to represent more complex and nuanced characteristics. A greater number of neurons enables the network to capture more complex relationships and to manifest itself with greater efficiency. However, using an excessive number of hidden neurons may result in overfitting, as the model may memorise the training data rather than generalising patterns. In addition, a greater number of neurons increases the network's computational cost and memory requirements.

3.4.8 Kernel size and number of filters

Convolutional Neural Networks (CNN) behaviour and capabilities are significantly influenced by the kernel size and number of filters. These hyperparameters determine the receptive field and the depth of feature extraction, influencing the model's capacity to extract relevant details from input data. The kernel size determines the spatial extent of the receptive field and indicates the portion of input data that each filter processes at once. Smaller kernel sizes, such as 3x3 or 5x5, capture local features such as edges and textures effectively. They excel at extracting intricate data features and fine-grained patterns. In contrast, larger kernel sizes, such as 7x7 or 9x9, have larger receptive fields, allowing them to detect more global or high-level features. They are more adept at recognising broader structures or objects in the input. The choice of kernel size depends on the complexity and scale of the to-be-detected features, achieving a balance between capturing local and global information [47]. The depth or number of output channels generated by a convolutional layer is determined by the number of filters in that layer. Each filter in a convolutional layer is responsible for learning from the input data particular feature representations as shown in Figure 30. By increasing the number of filters, the network is able to acquire more complex and diverse characteristics. This increased capability enables the model to capture a wider variety of data patterns and variations. It is essential to observe, however, that a greater number of filters increases computational complexity and memory requirements. Consequently, it is essential to strike the proper equilibrium to avoid unnecessary overhead while maintaining the model's expressiveness.

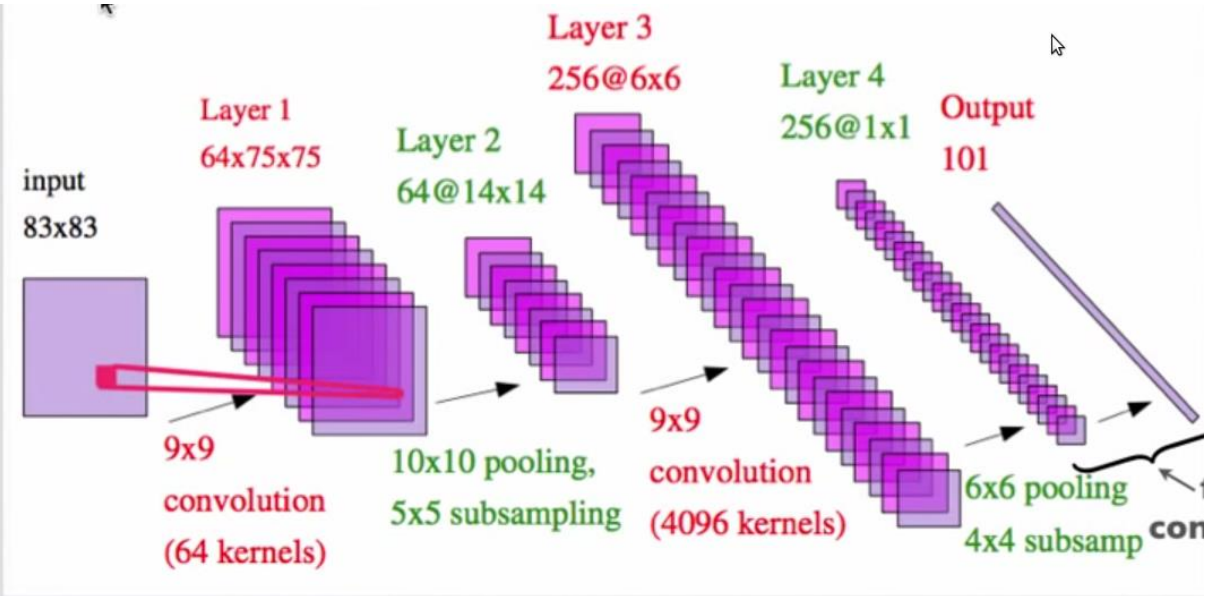

Figure 30 Kernel size and number of layers [47]

3.5 Other techniques

3.5.1 Weighted random sampler.

Weighted sampler is a sampling technique used during the training phase, especially when the dataset is imbalanced. It assigns various weights to the samples within a dataset based on the class distribution of those samples. The weights are proportional to the inverse of the class frequencies, so that classes that are underrepresented receive greater weights and classes that are overrepresented receive smaller weights. This enables the sampler to modify the sampling probabilities so that rare or minority class samples are included in training more frequently as shown in Figure 31. During model training, the weighted sampler is utilised in conjunction with the training data loader, which is responsible for loading quantities of data. The sampler modifies the sampling procedure to select samples based on their respective weights, resulting in a more equitable distribution of classes within each batch. Using a weighted sampler, models can learn to give more weight to the minority class and eliminate majority-class bias. This improves the overall performance of the model, particularly when working with unbalanced data sets.

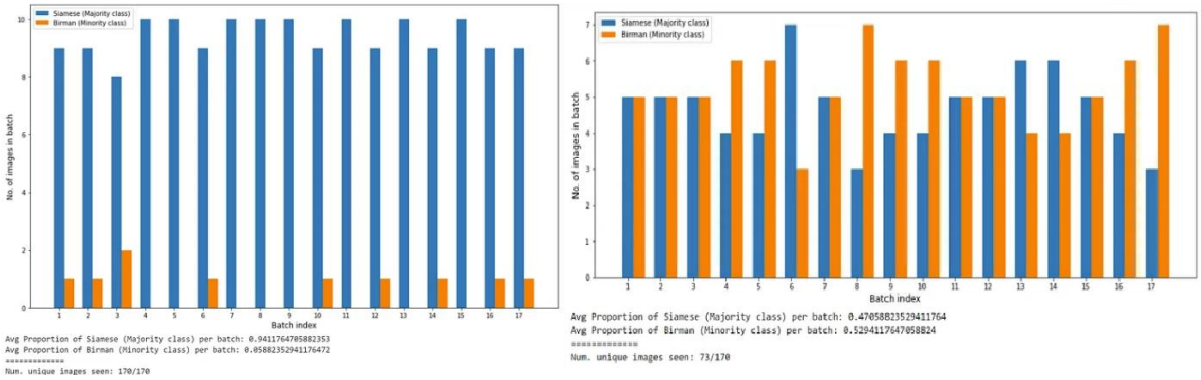

Figure 31 Weighted Random Sampler effect. [48]

3.5.2 Weighted loss

This is very similar to the weighted sampler. However, instead of affecting the sampler, i.e. how often the model sees a certain class, it affects how much penalty the model gets for classifying classes wrong, giving minority classes a larger penalty [49]. The equation for class weights is represented in Equation 4 Weighted loss class weights [49]Equation 4

$$class\,weight = 1 - \left(\frac{number\,of\,samples\,of\,the\,class}{total\,number\,of\,samples} \right)$$

Equation 4 Weighted loss class weights [49]

3.5.3 Regularization

Regularisation in deep learning is a collection of techniques used to prevent overfitting, increase generalisation, and boost the performance of deep neural networks. When a model performs well on the training data but fails to generalise to new, unobserved data, this is known as overfitting. By imposing additional constraints or penalties on the model during training, regularisation techniques help mitigate overfitting as shown in Figure 32. Regularization's primary objective is to establish a balance between model complexity and generalisation to unseen data [50]. Typically, deep learning models have so many parameters, allowing them to capture intricate patterns in the training data. This can, however, result in overfitting, where the model learns noise or specific examples from the training set rather than generalising well to new data.

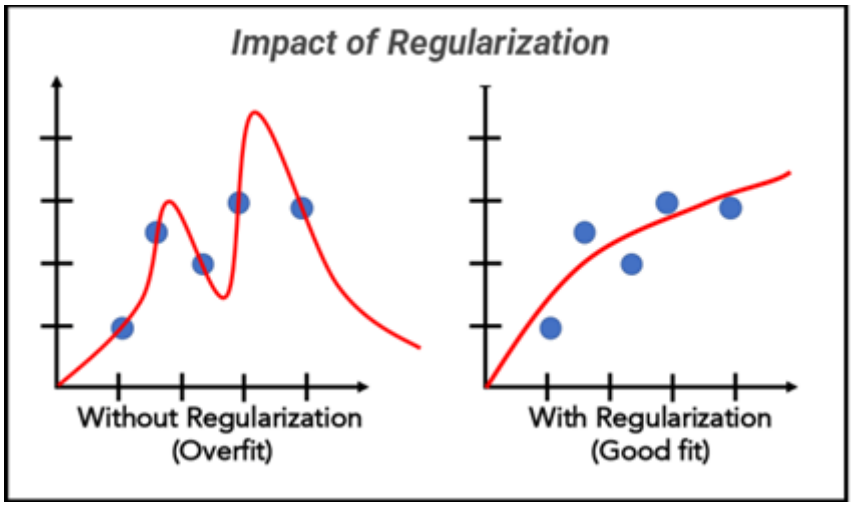

Figure 32 Regularization impact [50]

3.5.3.1 L2 (Weight decay)

L2 Regularisation (Weight Decay) is a regularisation technique that helps prevent overfitting in deep learning models. During the training procedure, a regularisation term is added to the loss function [51]. The regularisation term is proportional to the square sum of the model's weights. The objective of L2 regularisation is to encourage the model to learn reduced weight values, which results in a decision boundary that is smoother and less complex. By penalising large weight values, L2 regularisation discourages the model from relying excessively on individual features and encourages it to evaluate all features equally. This can enhance generalisation and decrease the propensity to overfit the training data.

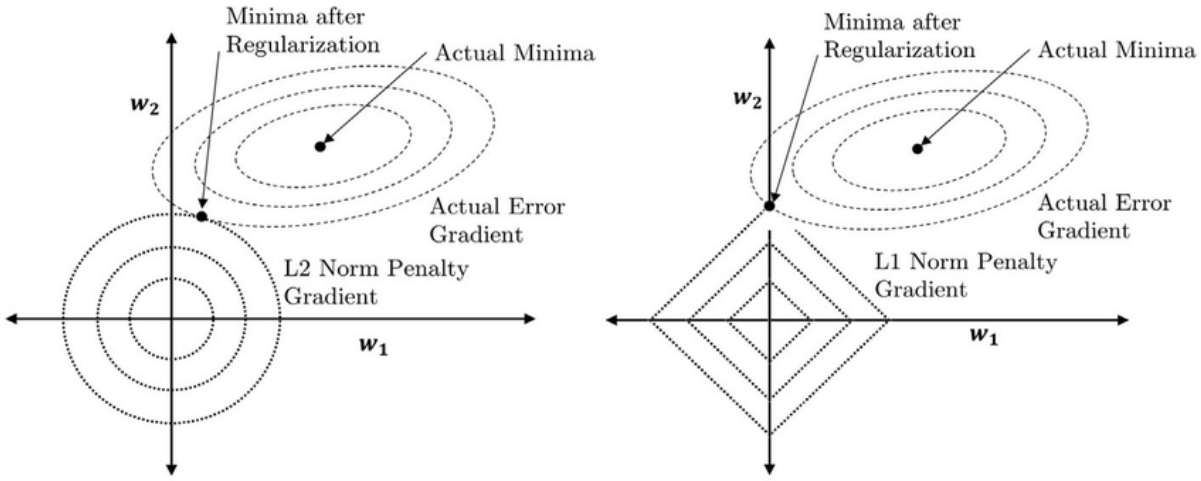

Figure 33 L2 regularization [51]

3.5.3.2 Dropout

Dropout is a regularisation technique utilised frequently in deep neural networks. During training, dropout sets a random proportion of neuron activations in a specific layer to zero at each iteration. This means that these neurons are momentarily "dropped out" or ignored as shown in Figure 34. The percentage of neurons to be eliminated is a hyperparameter that is typically between 0.2 and 0.5. Dropout makes the network more robust and less dependent on individual neurons. It forces the network to learn redundant representations and prevents it from over-reliance on specific activations [52]. Dropout functions as a form of ensemble learning within a single model, as distinct subsets of neurons are eliminated during training, resulting in greater model diversity and enhanced generalisation. During inference or prediction, dropout is typically disabled, and the entire network consisting of all neurons is utilised for making predictions.

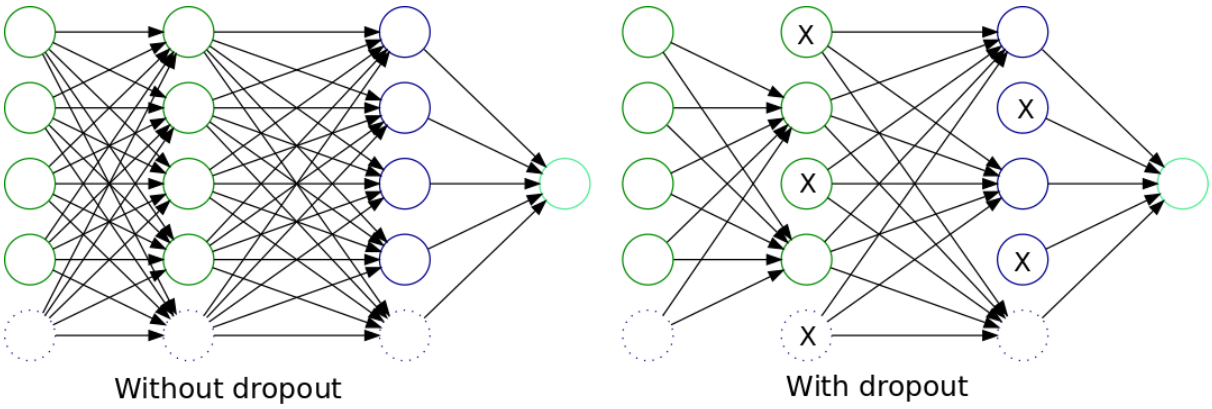

Figure 34 Dropout [52]

3.5.3.3 Early stopping

Early stopping is a basic yet effective regularisation technique that prevents overfitting by monitoring the model's performance during training and terminating the training process before the model begins to overfit the data as shown in Figure 35. It entails monitoring a validation metric (such as validation loss or accuracy) and halting training when the performance on the validation set ceases to improve or deteriorates [53]. The rationale behind early stopping is that, during training, the model initially learns the data's underlying patterns, resulting in enhanced validation performance. However, after a certain threshold, the model begins to overfit, meaning that it becomes overly tuned to the training data and performs inadequately on unseen data. Early stopping enables us to end training when the model has achieved good generalisation, thereby preventing it from memorising noise or specific examples from the training set.

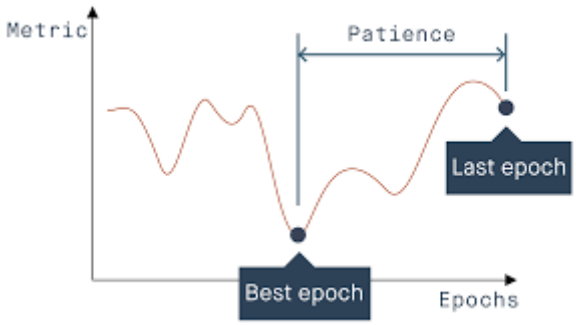

Figure 35 Early stopping [53]

3.5.3.4 Label smoothing

Label smoothing is a regularisation technique utilised frequently in deep learning, especially for classification tasks. It entails adding a small amount of noise or uncertainty to the labels of the training data's target labels. In conventional classification tasks, the target labels are typically encoded with a value of 1 for the correct class and 0 for all other classes for each instance of input data. Label smoothing introduces a small amount of uncertainty by assigning a value barely below 1 to the correct class and distributing the remainder among the other classes. For instance, in a classification problem with 5 classes, label smoothing might assign a smoothed label such as [0.9, 0.025, 0.025, 0.025] instead of the target label [1, 0, 0, 0, 0] for a particular instance as shown in Figure 36. The sum of the smoothed label values remains 1, but a minor portion of the probability mass is distributed to the wrong classes [54]. Label normalisation encourages the model to be more confident and generalise more effectively. It prevents the model from becoming overconfident in its predictions and reduces the possibility of overfitting to chaotic or incorrect labels in the training data. By introducing a small quantity of uncertainty, label smoothing encourages the model to discover more robust and less overconfident decision limits.

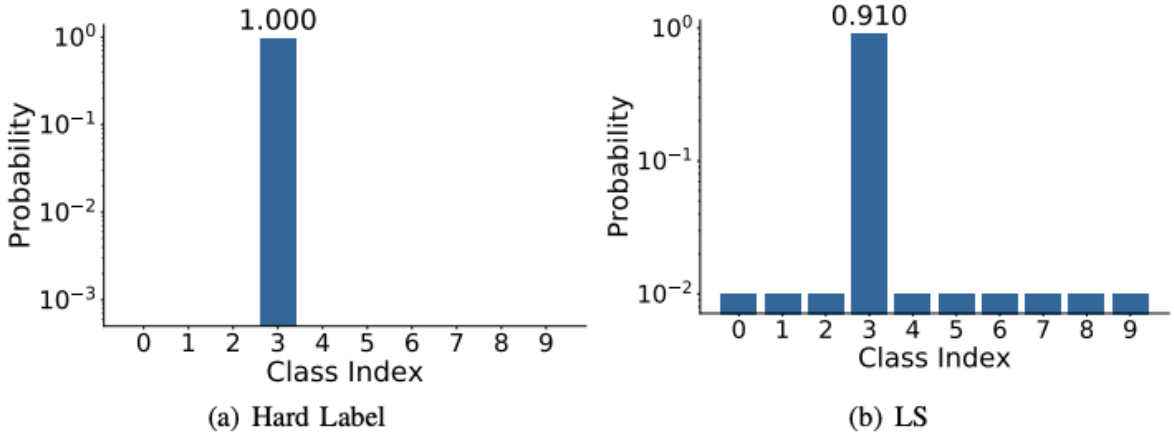

Figure 36 Label smoothing [54]

3.5.4 Batch normalization

Batch Normalisation is a deep learning technique that normalises the activations of each layer of a neural network by calculating the mean and standard deviation of a mini-batch of training examples as shown in Figure 37. By addressing the problem of internal covariate shift, Batch Normalisation enhances the network's training stability and performance [55]. It accelerates training by decreasing the reliance on subsequent layer gradients, allowing for increased learning rates and quicker convergence. It reduces overfitting by stabilising activations and making the network less sensitive to initial parameters. In addition, Batch Normalisation offers robustness during inference by adapting to various input distributions.

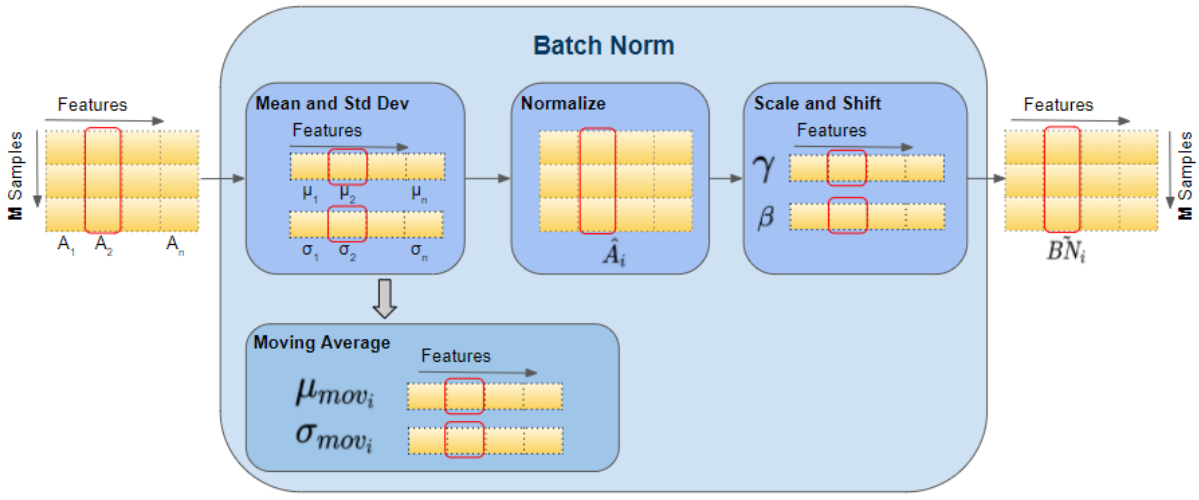

Figure 37 Batch normalization [55]

3.5.5 Transfer learning

Transfer learning is a deep learning technique in which a large dataset-trained model is used as a starting point for a new task or dataset. Transfer learning accelerates and enhances the learning process on a new, possibly smaller dataset by utilising the knowledge and representations learned by a previously trained model. Pre-training the model on a source task, such as image classification, in order to acquire meaningful representations is followed by fine-tuning the model on the target task using the new dataset. Transfer learning reduces training time, improves performance by utilising learned features, and increases robustness by utilising knowledge from diverse data sources [56]. The models were pre-trained on one of two datasets:

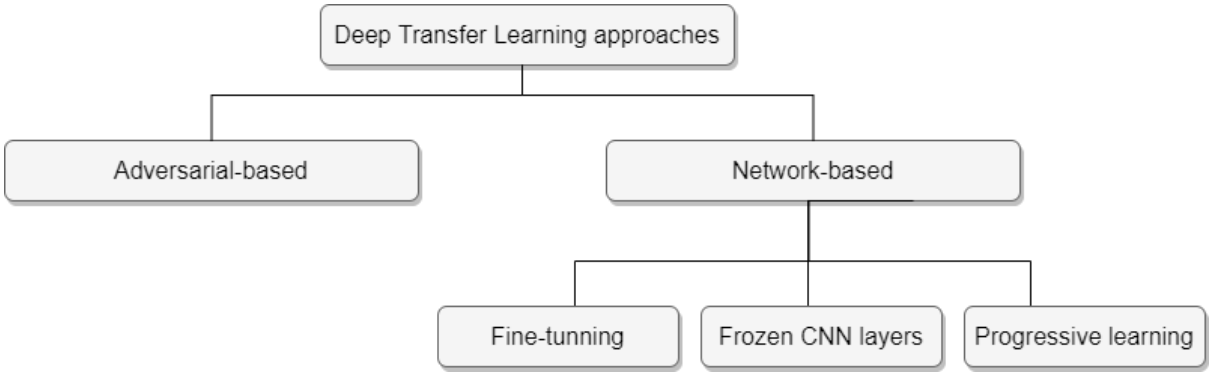

Figure 38 Transfer learning [56]

3.5.5.1 ImageNet

ImageNet [57] is one of the most popular large-scale image datasets for training deep learning models. It includes millions of labelled images spanning tens of thousands of categories. ImageNet includes a variety of object classes, such as animals, vehicles, and domestic items. It has been utilised extensively for image classification and object detection. ImageNet is a large and diverse collection of images that enables models to learn extensive and generalised representations of objects and visual features.

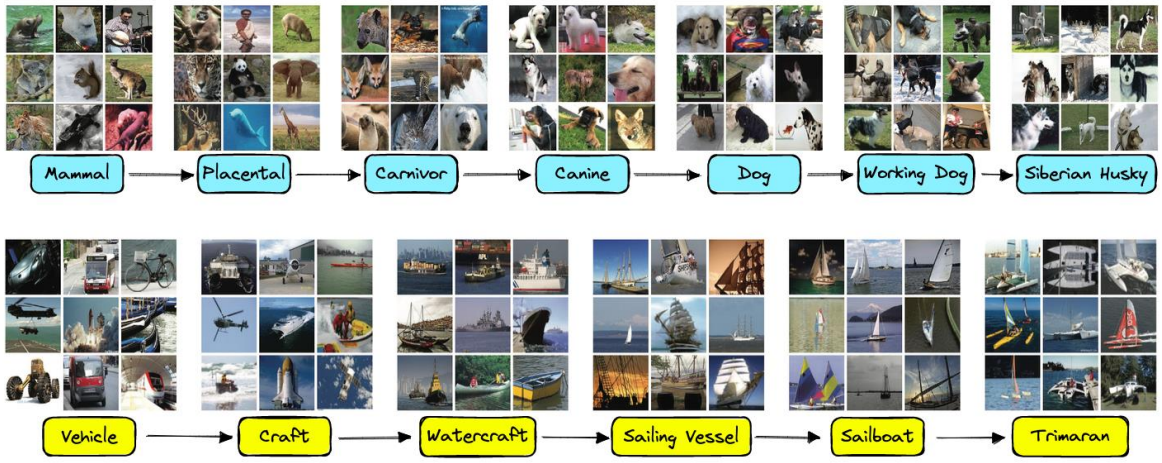

Figure 39 ImageNet sample [57]

3.5.5.2 CASIA-WebFace

The CASIA-WebFace [58] data set is a large-scale face recognition data set used in training MTCCN and FaceNet [23]. It is comprised of more than 500,000 images of faces collected from the Internet, representing approximately 10,000 people. The dataset contains a wide variety of poses, lighting conditions, facial expressions, and other variables, making it complex. CASIA-WebFace is a valuable resource for training and evaluating face recognition models, enabling researchers to develop robust and accurate algorithms for identifying and authenticating individuals based on facial features. It was hypothesized that using a model that was pre-trained on faces would for recognition would help in age classification.

Figure 40 CASIA-WebFace sample [58]

3.5.6 Residual attention

Residual attention networks incorporate residual connections and attention mechanisms to improve the representation learning capabilities of deep neural networks. By incorporating attention modules within residual blocks, these networks selectively amplify or suppress features based on their significance, enabling the model to concentrate on relevant portions of the input [59]. This combination enables the network to recognise long-distance dependencies, pay attention to distinguishing characteristics, and allocate computational resources accordingly.

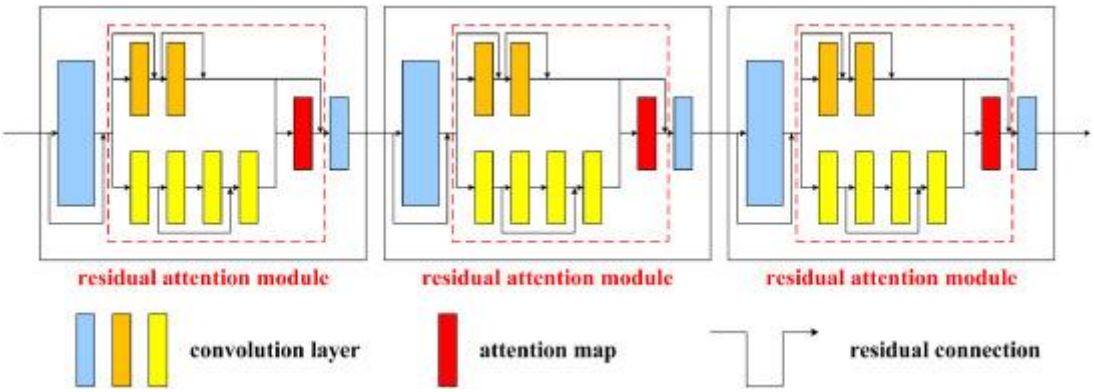

Figure 41 Residual Attention [59]

3.5.7 Hierarchical models

Hierarchical models are deep learning models designed to analyse hierarchically organised data, where data elements are grouped or nested within higher-level units. These models account for dependencies and variations within and between groups, enabling parameter estimation at various levels of the hierarchy. By incorporating both individual-level and group-level effects, hierarchical models provide a more thorough comprehension of the data and improve estimation efficiency by utilising information from multiple levels [60]. They are especially useful for analysing nested or clustered data and provide insight into the data's structure and relationships as shown in Figure 42. For example, in this project, the relationship between gender and age, and ethnicity and age, were analysed using hierarchical models, where two separate age classification models were trained based on whether the person is male or female. Similarly, 5 age classification models were trained for the 5 different ethnicity groups in the dataset. The results are shown and discussed in the following sections.

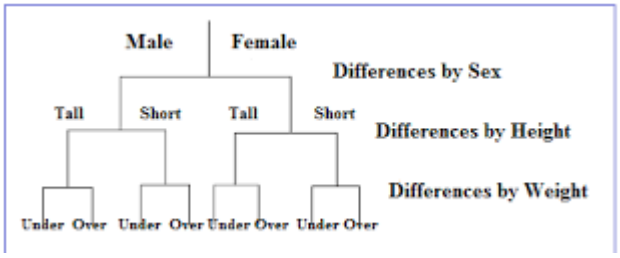

Figure 42 Hierarchical model example [60]

3.5.8 Multitasking

Multitasking in deep learning is the capacity of a model to perform multiple tasks simultaneously or sequentially. This can be accomplished by designing a neural network architecture with multiple output branches, each of which is devoted to a particular mission. During training, the model receives input data and simultaneously predicts outputs for all tasks. The training loss function is a combination of losses from all tasks, which encourages the model to acquire representations that are advantageous for all tasks [61]. By jointly optimising the model on multiple tasks, multitasking in deep learning enables the extraction of shared features and may contribute to enhanced generalisation and efficiency. It allows the model to utilise shared knowledge across tasks, capture complex relationships, and execute a variety of tasks within a single framework as shown in Figure 43. In this case, since the age, gender, and ethnicity features all come from faces, it is logical to assume that there are common features between the three of them. The multitasking model's job is to find those common features.

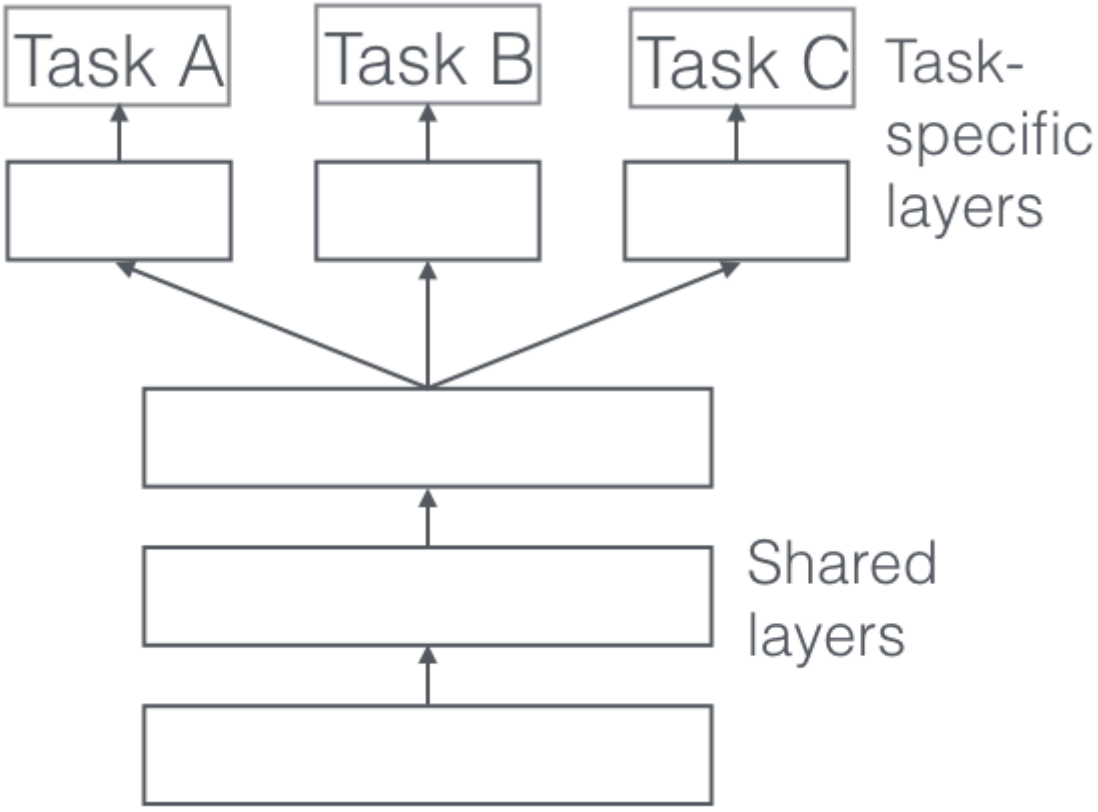

Figure 43 Multitasking models [61]

3.5.9 Stochastic Weight Averaging (SWA)

Stochastic Weight Averaging (SWA) is a deep learning technique used to enhance model generalisation and robustness. Instead of utilising the final learnt parameters, SWA keeps a running average of the model's weights throughout training. This process of weight averaging helps the model converge to a more generalised solution by eliminating noise introduced by stochastic updates. By integrating

solutions discovered in various regions of the loss landscape, SWA captures a more stable and level region, resulting in improved generalisation performance [62]. SWA is a straightforward and efficient technique that reduces overfitting, improves model robustness, and can be readily integrated into the training procedure.

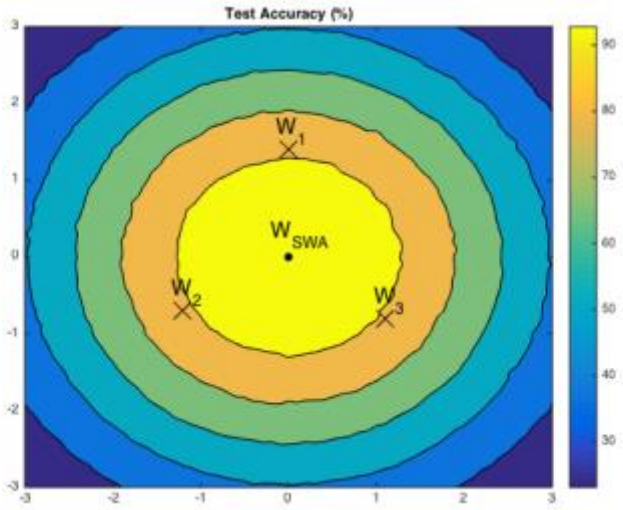

Figure 44 SWA produces better results since it averages the three different model weight instances [62]

4 Implementation and code

4.1 Environment

Vast.ai was used for remote training as it contains machines with high-grade GPUs. For this project, an approximate of **1000 hours** were spent on training models using RTX3090 to train more than **400 different models** on Vast.ai. Where some machines had 1-8 GPUs at once, and multiple machines were rented at the same time for multiple parallel training of models. Tmux is what was used to connect to Vast.ai remote machines, and VSCode was the main IDE used for writing python scripts to train the models. FileZilla, is the GUI interface utilized to transfer trained models and logs from the Vast.ai machine to the local machine once training was completed. PyTorch lightning is the framework of choice for training and validating deep learning models and determinism was employed to ensure the reproducibility of the code and avoid fluctuations of random operations in model training. The above tools are briefly explained below.

4.1.1 Vast.ai (Dockerization)

Vast.ai is a cloud-based platform that provides machine learning and deep learning tasks with a distributed computational infrastructure. It enables parallelization and acceleration of computationally intensive workloads by permitting users to rent GPU resources from a pool of available machines as shown in Figure 45. One of the most important aspects of Vast.ai is its utilisation of Dockerization. Docker is a containerization platform that enables the construction and deployment of containers, which are lightweight, isolated environments. Dockerization is utilised within the context of Vast.ai to provide a consistent and reproducible environment for executing machine learning models. Users can construct a Docker container that contains their entire machine learning pipeline, including all dependencies, libraries, and code. The container can then be uploaded to Vast.ai, which will provision the necessary GPU resources and execute the containerized application. Dockerization provides a number of advantages for Vast.ai. It ensures that the execution environment is uniform across multiple platforms, eliminating compatibility and versioning issues. It also facilitates the deployment and management of machine learning workflows by encapsulating the entire setup within a container. This facilitates collaboration and reproducibility by allowing users to share and disseminate their models and configurations with ease.

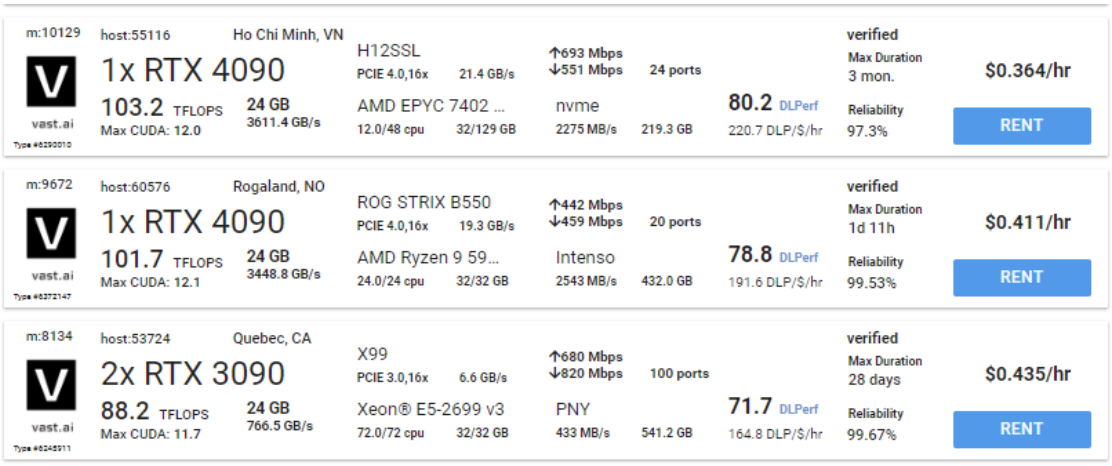

Figure 45 Vast.ai interface

4.1.2 Tmux

tmux is a terminal multiplexer that enables the creation and management of multiple terminal sessions within a single window. It is especially useful when working with remote servers or tasks that necessitate running multiple processes concurrently. Users can divide the terminal window into panes, each of which displays a distinct shell session, using tmux. These panes can be arranged horizontally or vertically, enabling effective multitasking and process monitoring. Users can toggle between panes, resize them, and even withdraw and reattach to a tmux session without losing the state of their terminal sessions. tmux supports windows, which provide an additional level of organisation in addition to panes. Within a tmux session, users can create multiple windows, each with one or more panels as shown in Figure 46. Similar to having multiple tabs in a web browser, this enables effortless toggling between distinct terminal sessions.

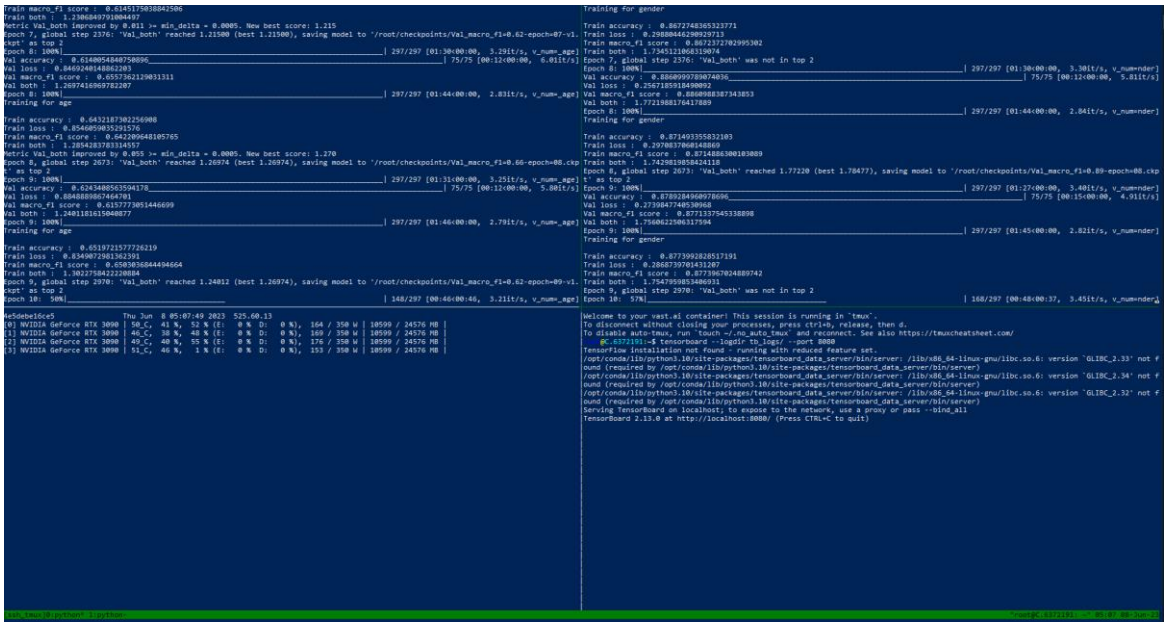

Figure 46 tmux running 4 models, 2 windows, and tracking gpu and tensorboard

4.1.3 Backblaze

Backblaze is a cloud storage system where different data can be stored and accessed remotely. It was used to store the dataset, and any pretrained models/weights that were constantly needed when training models on a fresh remote instance. The reason backblaze was used instead of local machine is because their network connection speeds are much faster than the local environment, saving hours of time waiting for data to be uploaded and downloaded. Moreover, all data was routed through a Content Delivery Network (CDN) that goes through a personal domain to ensure the performance and availability of data.

4.1.4 Filezilla

Filezilla is an open-source FTP tool that allows users to transfer files between a local machine and a remote server using SSH. It offers high transfer speeds and allows the comparison of local and remote storage contents as shown in Figure 47.

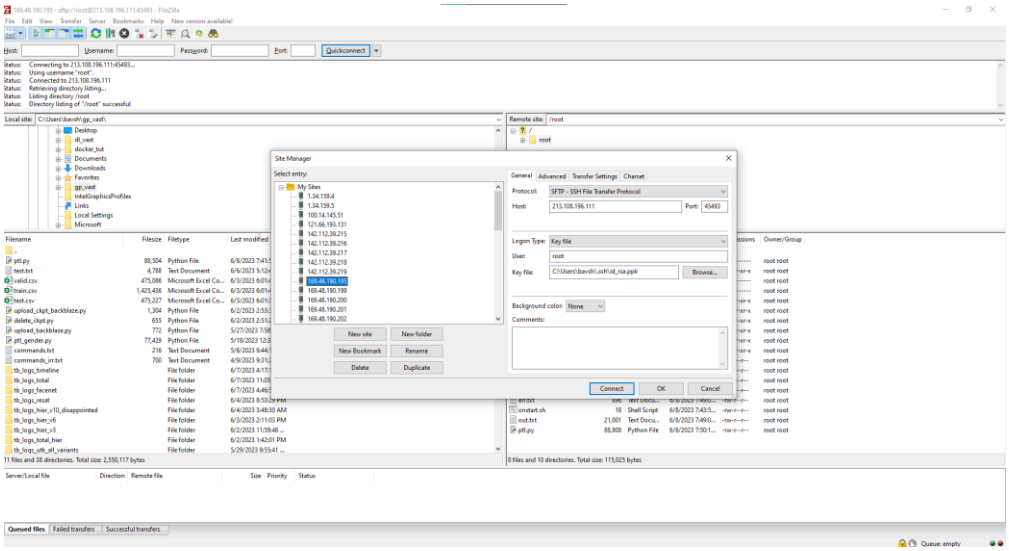

Figure 47 Filezilla interface

4.1.5 Visual Studio Code (VSCode) and Google Colab

VSCode is a powerful source code editor that is cross-platform and offers a rich environment and extensions for almost everything. VSCode was used as the choice of code editor to write and execute python script with. One very important feature of VSCode for this project is Remote host development, where all extensions and settings of the local VSCode can be used in a remote machine. This saved a lot of time when editing code on the remote machines was necessary. When the interactivity of notebooks was needed, Colab was a good choice. However, because of its GPU limitations, along with limitations of interactive python, it was never used for training. That is where Vast.ai shined.

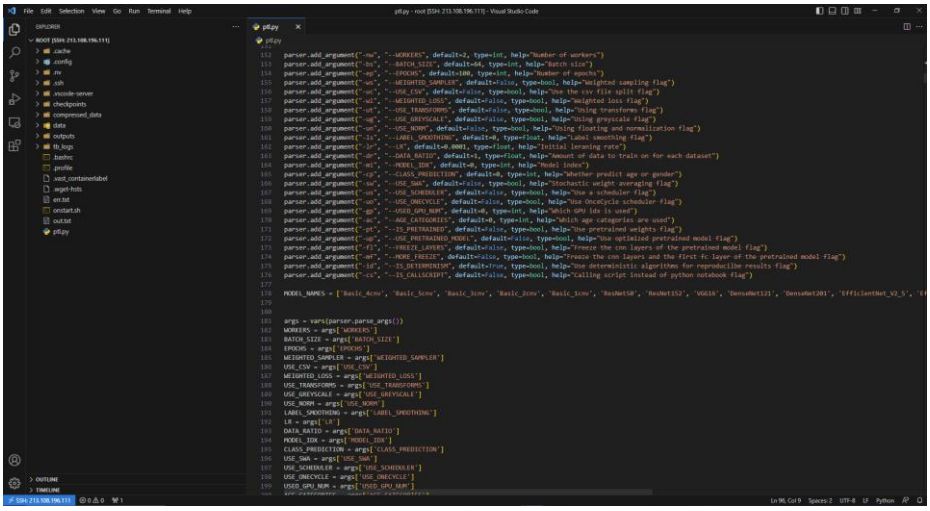

Figure 48 VSCode connected to a remote host and editing from local machine

4.1.6 PyTorch Lightning

PyTorch Lightning is a powerful and lightweight Python library that facilitates the training and organisation of PyTorch models. It offers a high-level interface and a collection of abstractions that make the training cycle modular, readable, and reproducible. With PyTorch Lightning, developers can concentrate more on the design of their models and experiments and less on boilerplate code for training loops and other repetitive duties. PyTorch Lightning divides model architecture, data loading, and training logic into distinct modules called LightningModules. This modular structure improves the organisation and reusability of code. It also provides convenient hooks for common training tasks such as checkpointing, logging, and distributed training, which makes scaling models for distributed computing simpler. In addition, PyTorch Lightning is fully compatible with PyTorch's APIs and integrates seamlessly with PyTorch. This enables users to leverage the flexibility and power of PyTorch while taking advantage of the added simplicity and structure offered by PyTorch Lightning. PyTorch in general, and Lightning in particular, were chosen over TensorFlow and Keras for better documentation, community, and backward compatibility.

4.1.7 Determinism and reproducibility

Determinism and reproducibility in deep learning refer to the capacity to acquire consistent and repeatable results when executing the same code or model on the same input data. Determinism is the property of producing the same output given identical input and parameters. In other words, a deterministic deep learning model will always produce the same results, including the same predictions or loss values, if the code and data remain unaltered. In contrast, reproducibility expands the concept of determinism to include not only the model's behaviour but also the entire experimental apparatus. It indicates that the entire training procedure, including the random initialization of weights, random shuffling of data, and any other sources of randomization, can be replicated to achieve the same outcomes. Due to numerous factors, achieving determinism and reproducibility in deep learning can be

difficult. Frequently, deep learning models include nondeterministic elements such as random weight initialization, dropout, and data augmentation techniques. Moreover, hardware and software differences between platforms or environments can contribute to variability.

To ensure determinism and reproducibility, a number of factors must be considered. These include initialising model parameters and seeding random number generators with random data. In addition, utilising fixed validation or test sets and meticulously documenting experimental configurations can facilitate the reproduction and comparison of results. Researchers and practitioners can have confidence in the consistency and dependability of their deep learning experiments by attaining determinism and reproducibility. It facilitates the comparison of various models or techniques, aids in troubleshooting and refining, and encourages the dissemination and replication of research findings. To achieve this, all models are trained using RTX3090 GPU and everything is seeded, more details in the code section.

4.2 Webapp

After the model is trained and optimized, the model was deployed on a webapp. The webapp runs with flask backend and its frontend is done using Webflow. The way it works is the frontend sends an image to the flask server. The flask server sends the image to the models, and after the tasks are classified, the server sends the results back to the frontend to be displayed for the user as shown in Figure 49. Right now, the images are pre-set, but adding more flexibility is a plan in the future.

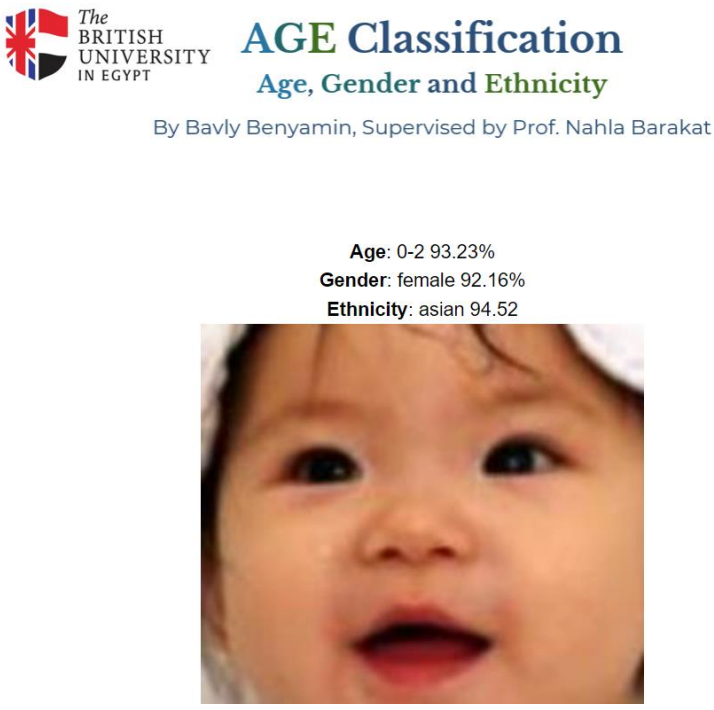

Figure 49 Webapp frontend

4.3 Code

The complete code can be found at https://github.com/bavShehata/AGE_Classification

A lot of the effort went into the code, with utility functions written from scratch and a deep understanding of PyTorch and Lightning frameworks, which resulted in dynamic, well-commented code. As a result, I will mention the most important parts in the following section.

4.3.1 CLI (Command Line Interface) Training

As mentioned before, I was able to train more than 400 hundred different models. Each with slightly different hyperparameters. Going inside the code every single time to change a number, for example changing the learning rate from 1e-3 to 1e-2, would have taken a lot of time and definitely would have been error prone. To fix this problem, all training was done through the CLI (Command line interface) where different hyperparameters can be input using arguments while calling the script. For example,

<python AGE.py -bs 64 -ws 1 -ut 1 -lr 0.0001 -cp 0 -mi 0 -gp 0> would run the script with a batch size of 64, use a weighted sampler, a learning rate of 0.0001, 4-layered cnn, to classify the age on the first GPU (in case of multiple GPUs in a machine). This was possible using ArgumentParser as shown in Figure 50

```
# Using arguments to run multiple python scripts with different parameters through the CLI
# For example: python AGE.py -cs 1 -bs 64 -ws 1 -ut 1 -ug 1 -lr 0.0001 -cp 18 -uc 1 -mi 0 -us 1 -gp 0

parser = ArgumentParser(formatter_class=ArgumentDefaultsHelpFormatter)

parser.add_argument("-nw", "--WORKERS", default=2, type=int, help="Number of workers")
parser.add_argument("-bs", "--BATCH_SIZE", default=64, type=int, help="Batch size")
parser.add_argument("-ep", "--EPOCHS", default=100, type=int, help="Number of epochs")
parser.add_argument("-ws", "--WEIGHTED_SAMPLER", default=False, type=bool, help="Weighted sampling flag")
parser.add_argument("-uc", "--USE_CSV", default=False, type=bool, help="Use the csv file split flag")
parser.add_argument("-wl", "--WEIGHTED_LOSS", default=False, type=bool, help="Weighted loss flag")
parser.add_argument("-ut", "--USE_TRANSFORMS", default=False, type=bool, help="Using transforms flag")
parser.add_argument("-ug", "--USE_GREYSCALE", default=False, type=bool, help="Using greyscale flag")
parser.add_argument("-un", "--USE_NORM", default=False, type=bool, help="Using floating and normalization flag")
parser.add_argument("-ls", "--LABEL_SMOOTHING", default=0, type=float, help="Label smoothing flag")
parser.add_argument("-lr", "--LR", default=0.0001, type=float, help="Initial leraning rate")
parser.add_argument("-dr", "--DATA_RATIO", default=1, type=float, help="Amount of data to train on for each dataset")
parser.add_argument("-mi", "--MODEL_IDX", default=0, type=int, help="Model index")
parser.add_argument("-cp", "--CLASS_PREDICTION", default=0, type=int, help="Whether predict age or gender")
parser.add_argument("-sw", "--USE_SWA", default=False, type=bool, help="Stochastic weight averaging flag")
parser.add_argument("-us", "--USE_SCHEDULER", default=False, type=bool, help="Use a scheduler flag")
parser.add_argument("-uo", "--USE_ONECYCLE", default=False, type=bool, help="Use OnceCycle scheduler flag")
parser.add_argument("-gp", "--USED_GPU_NUM", default=0, type=int, help="Which GPU idx is used")
parser.add_argument("-pt", "--IS_PRETRAINED", default=False, type=bool, help="Use pretrained weights flag")
parser.add_argument("-up", "--USE_PRETRAINED_MODEL", default=False, type=bool, help="Use optimized pretrained model flag")
parser.add_argument("-fl", "--FREEZE_LAYERS", default=False, type=bool, help="Freeze the cnn layers of the pretrained model flag")
parser.add_argument("-mf", "--MORE_FREEZE", default=False, type=bool, help="Freeze the cnn layers and the first fc layer of the pretrained model flag")
parser.add_argument("-ip", "--IS_PREDICT", default=False, type=bool, help="Predicting for new inputs flag")
parser.add_argument("-id", "--IS_DETERMINISM", default=True, type=bool, help="Use deterministic algorithms for reproducilbe results flag")
parser.add_argument("-cs", "--IS_CALLSCRIPT", default=False, type=bool, help="Calling script instead of python notebook flag")
```

Figure 50 ArgumentParser

4.3.2 Reproducibility

One important characteristic of this code is that it is reproducible, which means that running the code with the same hyperparameters will always yield the exact same results. As a result, any slight change in results in response to one of the hyperparameters is certainly only due to the change of the hyperparameter, and not any external random operations (weight initialization for example). This

contributes to the confidence of how effective a hyperparameter change is. That was achieved by seeding everything and making sure that all algorithms are deterministic as shown in Figure 51

```
#reproducability
random_seed = 42
deterministic = False
if DETERMINISM:
  torch.manual_seed(random_seed)
  torch.cuda.manual_seed(random_seed)
  torch.backends.cudnn.benchmark = False
  np.random.seed(random_seed)
  pl.seed_everything(random_seed, workers=True)
  torch.backends.cudnn.deterministic = True
  deterministic=True
  torch.use_deterministic_algorithms(True)
```

Figure 51 Ensures code reproducibility.

4.3.3 Logging and keeping track of hyperparameters.

With the huge number of trained models, meticulous logging of hyperparameters is crucial as to not get lost in what hyperparameters has changed. The way this is reflected in the code, is by grouping all hyperparameters in a dictionary, then save the dictionary to TensorBoard as shown in Figure 52

```
# add all hyperparameters to tensorboard
if self.logger:
  self.logger.experiment.add_text("PARAMS", str(PARAMS))
```

Figure 52 Logging hyperparameters into tensorboard

4.3.4 Multiprocessing

Multiprocessing was utilized in the project due to the amount of data augmentations. Since data augmentations on images can be relatively slow, it was noted that data augmentations were a bottleneck in training the models as the GPUs were idle for a long time. To fix this problem, multiple CPU cores were used at the same time to augment the images, which resulted in more than 1.5x increase in speed. The tricky part was to ensure that only the data augmentation was distributed, and not every single line of code. Otherwise, each process will call multiple processes resulting in an infinite recursive call. This was achieved by running code only if it was called by the __main__ process as shown in Figure 53

```
PARAMS = {
'IMG_SIZE': IMG_SIZE,
'DATA_RATIO': DATA_RATIO,
'BATCH_SIZE': BATCH_SIZE,
'EPOCHS': EPOCHS,
'WEIGHTED_SAMPLER': WEIGHTED_SAMPLER,
'USE_CSV': USE_CSV,
'WEIGHTED_LOSS': WEIGHTED_LOSS,
'USE_TRANSFORMS': USE_TRANSFORMS,
'USE_GREYSCALE': USE_GREYSCALE,
'USE_NORM': USE_NORM,
'LABEL_SMOOTHING': LABEL_SMOOTHING,
'LR': LR,
'MODEL_NAME':MODEL_NAME,
'DATA_DIRS':DATA_DIRS,
'DETERMINISM':DETERMINISM,
'USE_SWA':USE_SWA,
'USE_SCHEDULER':USE_SCHEDULER,
'SCHEDULER_FACTOR':SCHEDULER_FACTOR,
'SCHEDULER_PATIENCE':SCHEDULER_PATIENCE,
'SCHEDULER_THRESHOLD':SCHEDULER_THRESHOLD,
'USE_ONECYCLE':USE_ONECYCLE,
'CYCLE_DIV_FACTOR':CYCLE_DIV_FACTOR,
'CYCLE_FINAL_DIV_FACTOR':CYCLE_FINAL_DIV_FACTOR,
'SWA_LRS': SWA_LRS,
'SWA_EPOCH_START': SWA_EPOCH_START,
'ANNEALING_EPOCHS': ANNEALING_EPOCHS,
'IS_PRETRAINED' : IS_PRETRAINED,
'USE_PRETRAINED_MODEL':USE_PRETRAINED_MODEL,
'FREEZE_LAYERS': FREEZE_LAYERS,
'MORE_FREEZE': MORE_FREEZE,
'IS_PREDICT': IS_PREDICT,
'GPU_NAME' : GPU_NAME,
'METRIC_TO_MONITOR': METRIC_TO_MONITOR,
'OPTIM':OPTIMS,
'base_features':base_features,
'hidden_neurons':hidden_neurons,
}
# add all hyperparameters to tensorboard
if self.logger:
  self.logger.experiment.add_text("PARAMS", str(PARAMS))
```

Figure 53 Multiprocessing

4.3.5 Multiple Optimizers

One more hyperparameter was the optimizer to be used. Since there are many optimizers including SGD, RMSProb, and Adam, the code attempted to allow as many different trials as possible. This was achieved as seen in Figure 54

```
# using different optimizers
params = self.parameters()
lr = self.learning_rate if self.learning_rate else LR
if optimi=='Adam':
  optimizer = torch.optim.Adam(params, lr, betas=betas, eps=eps, weight_decay=weight_decay, amsgrad=amsgrad)
elif optimi=='AdamW':
  optimizer = torch.optim.AdamW(params, lr, betas=betas, eps=eps, weight_decay=weight_decay, amsgrad=amsgrad)
elif optimi=='NAdam':
  optimizer = torch.optim.NAdam(params, lr, betas=betas, eps=eps, weight_decay=weight_decay, momentum_decay=momentum_decay)
elif optimi=='SGD':
  optimizer = torch.optim.SGD(params, lr, momentum=momentum, dampening=dampening, weight_decay=weight_decay, nesterov=nesetrov)
elif optimi=='RMSprop':
  optimizer = torch.optim.RMSprop(params, lr, alpha=alpha, eps=eps, weight_decay=weight_decay, momentum=momentum, centered=centered)
elif optimi=='Adadelta':
  optimizer = torch.optim.Adadelta(params, lr, rho=rho, eps=eps, weight_decay=weight_decay)
elif optimi=='Adamax':
  optimizer = torch.optim.Adamax(params, lr, betas=betas, eps=eps, weight_decay=weight_decay)
```

Figure 54 Multiple optimizers

4.3.6 Backblaze and data download

While renting multiple machines, transferring the data from the local machine became a bottleneck since the upload internet speed was relatively slow. As a result, it was decided that all data is to be downloaded through a 3rd-party storage system that offers high speed. In this case, Backblaze. Every time the code is run, the data is downloaded again and unzipped, along with any pretrained models as shown in Figure 55

```
# Get B2 resource and define helper functions
ENDPOINT='https://s3.us-east-005.backblazeb2.com'
KEY_ID_RO='005b80aefd1b9d50000000002'
APPLICATION_KEY_RO='K0055+OLfky70Z21NQSahzMTq8uUU3I'
b2 = get_b2_resource(ENDPOINT, KEY_ID_RO, APPLICATION_KEY_RO)

BUCKET_NAME = 'testing-612'
CDN_ENDPOINT = 'https://files.bavlifyweb.com/file'

# Download and unzip the data
if __name__ == '__main__':
  run_command("mkdir data")
  print("Downloading data")
  # 1 - DOWNLOAD FILE
  # Call function to return list of object 'keys' formatted into friendly urls
  browsable_urls = list_objects_browsable_url(BUCKET_NAME, ENDPOINT, b2)
  for url in browsable_urls:
    for download_filename in DOWNLOAD_FILENAMES:
      if url.split('/')[-1] == download_filename or download_filename == None:
        if not os.path.exists(f'compressed_data/{download_filename}'):
          download_url(url, directory='./compressed_data/',chunk_size=1024)
          print(f"Downloading {url.split('/')[-1]}")
        if(url.split('/')[-1].split('.')[-1] == 'rar'):
          if not os.path.exists(f"data/{download_filename.split('.')[0]}"):
            print("Unzipping file")
            patoolib.extract_archive(f"compressed_data/{url.split('/')[-1]}", outdir="data")
```

Figure 55 Backblaze and data download

4.3.7 Data preparation

As indicated before, the true labels of every UTK image are inside the image's name/index. As a result, the following function in Figure 56 was written to extract those labels and group them in a dictionary to prepare them to csv.

```
# extract age, gender, and ethnicity from the image name
def get_data_utk_face(DATA_DIR, img):
  img_idx = DATA_DIR+img
  age_label = None
  age_cat = None
  age = int(img.split('_')[0])
  gender_label = int(img.split('_')[1])
  ethnicity_label = int(img.split('_')[2])
  gender = LABEL_TO_GENDER[gender_label]
  ethnicity = LABEL_TO_ETHNICITY[ethnicity_label]
  data_file = DATA_DIR[:-1].split('/')[1]
  img_dict = {'img_idx': img_idx,
              'age_label': age_label,
              'age_cat': age_cat,
              'gender_label': gender_label,
              'ethnicity_label': ethnicity_label,
              'gender': gender,
              'ethnicity': ethnicity,
              'age': age,
              'data_file': data_file
              }
  return img_dict
```

Figure 56 Data preparation

4.3.8 Data augmentation

Data augmentation is vital to a highly generalizing model. Especially due to the data imbalance, data augmentation along with a weighted sampler results in the model seeing multiple new images from minority classes. The data augmentations used are shown in Figure 57

```
train_transform = A.Compose(
  [
    A.ToFloat(max_value=255, always_apply=True),
    A.ToGray(p=1),
    A.SmallestMaxSize(max_size=IMG_SIZE),
    A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.05, rotate_limit=15, p=0.5),
    A.RandomCrop(height=IMG_SIZE, width=IMG_SIZE),
    A.GaussianBlur(sigma_limit=9, p=0.5),
    A.RandomBrightnessContrast(p=0.5),
  ]
)

val_transform = A.Compose(
  [
    A.ToFloat(max_value=255, always_apply=True),
    A.ToGray(p=1),
    A.SmallestMaxSize(max_size=IMG_SIZE),
    A.RandomCrop(height=IMG_SIZE, width=IMG_SIZE),
  ]
)
```

Figure 57 Data augmentations

4.3.9 Weighted sampler

A weighted sampler is what makes the model 'see' more images from a minority class in an imbalanced dataset. It does that by assigning a weight to each sample, inverse to its class's majority. The higher the weight, the more probably the image will be picked. This is implemented by the piece of code in Figure 58

```
# using weighted sampler
if self.weighted_sampler:
  class_weights = list(1/pd.Series(y_train_temp).value_counts().sort_index())
  weights = [class_weights[label] for label in y_train_temp]
  self.sampler = WeightedRandomSampler(weights, len(y_train_temp))
```

Figure 58 Weighted sampler

4.3.10 Metrics tracking

An essential part of any model training is tracking the metrics. Since this is a classification problem, the relevant metrics are accuracy, F! Score, etc…And given that the data is imbalanced, the choice of F1 score was decided as the main metric. Metrics were extracted using a classification report and a confusion matrix as shown in Figure 59

```
loss = merged_dict['loss']
total_loss = statistics.mean(loss)
if CLASS_AGE:
  labels = merged_dict['age_labels']
elif CLASS_GENDER:
  labels = merged_dict['gender_labels']
elif CLASS_ETHNICITY:
  labels = merged_dict['ethnicity_labels']


label_vs_pred = (labels, merged_dict['pred_idxs'])

cl_rep = classification_report(*label_vs_pred, target_names=CLASS_RANGES, output_dict=True, zero_division=0)
cf_matrix = confusion_matrix(*label_vs_pred)
df_cm = pd.DataFrame(cf_matrix / np.sum(cf_matrix, axis=1)[:, None], index=[i for i in CLASS_RANGES],
                     columns=[i for i in CLASS_RANGES])
for idx, (key, val) in enumerate(cl_rep.items()):
  precision.append(val['precision'])
  recall.append(val['recall'])
  f1_score.append(val['f1-score'])
  if idx == len(CLASS_RANGES)-1:
    break
accuracy = [i for i in (cf_matrix.diagonal()/(cf_matrix.sum(axis=0)+1e-5))]
total_accuracy = cl_rep['accuracy']
macro_f1 = cl_rep['macro avg']['f1-score']
plt.figure(figsize=(12, 7))
sn_fig = sn.heatmap(df_cm, annot=True).get_figure()
```

Figure 59 Metrics tracking

4.3.11 Inference and prediction

Finally, once the model is trained, it is ready for predicting new images. The way to do that is first, the model is frozen and in evaluation mode so that the weights stay the same. Next, the images are supplied into the model, and finally, each task is classified on its own using its respective trained model and the results are concatenated and plotted as shown in Figure 60

```
FacesModel = get_model(MODEL_IDX)().load_from_checkpoint(FINAL_CKPT, map_location=torch.device(device))
print(f"\nPredicting {pred_indicator} based on model")
FacesModel.eval()
FacesModel.freeze()
outputs = trainer.predict(FacesModel, faces_pred)
for batch in outputs:
  for output in batch[0]:
    _, pred_idx = torch.max(output, 0)
    pred = CLASS_RANGES[pred_idx]
    # convert output probabilities to predicted class
    _, preds_tensor = torch.max(output, 0)
    pred_idx = preds_tensor.cpu()
    pred_prob = F.softmax(output, dim=0)[pred_idx]*100
    pred = CLASS_RANGES[pred_idx]
    pred_label = f"{pred_indicator}: {pred} {pred_prob:.1f}%"
    print(f"Predicted output: {pred_label}")
    if CLASS_AGE:
      pred_labels_a.append(pred_label)
    elif CLASS_GENDER:
      pred_labels_g.append(pred_label)
    elif CLASS_ETHNICITY:
      pred_labels_e.append(pred_label)
pred_labels = [f"{pred_labels_a[i]}\n{pred_labels_g[i]}\n{pred_labels_e[i]}" for i in range(len(pred_labels_a))]
plotted_fig = plot_classes_preds(FacesModel, torch.clone(outputs[0][1]).detach(), imgs_to_predict[:predict_viz_image_limit],
plotted_fig.savefig(f'outputs/predictions-{get_time()}.png', facecolor="white", edgecolor='none')
```

Figure 60 Model inference

4.3.12 Models and learning techniques.

4.3.12.1 Basic CNNs

A basic CNN consisted of 1-5 layers. Each layer is a cnv -> relu -> batchnorm -> maxpool and then flattening the features and passing them through two fully connected layers. It is important to note that the number of features and hidden neurons were hyperparameters as well, and the code was written in a way that keeps that flexibility in mind as shown in Figure 61

```
def __init__(self, num_of_layers):
  super().__init__()
  self.bn1 = nn.BatchNorm2d(base_features)
  self.bn2 = nn.BatchNorm2d(base_features*2)
  self.bn3 = nn.BatchNorm2d(base_features*4)
  self.bn4 = nn.BatchNorm2d(base_features*8)
  self.bn5 = nn.BatchNorm2d(base_features*16)
  self.doi = nn.Dropout(0.1)
  self.do1 = nn.Dropout(0.1)
  self.do2 = nn.Dropout(0.1)
  self.do3 = nn.Dropout(0.1)
  self.do4 = nn.Dropout(0.1)
  self.do5 = nn.Dropout(0.1)
  self.doo = nn.Dropout(0.6)
  self.cnv1 = nn.Conv2d(3, base_features, kernel_size = 3, padding = 1)
  self.cnv2 = nn.Conv2d(base_features, base_features*2, kernel_size = 3, padding = 1)
  self.cnv3 = nn.Conv2d(base_features*2, base_features*4, kernel_size = 3, padding = 1)
  self.cnv4 = nn.Conv2d(base_features*4, base_features*8, kernel_size = 3, padding = 1)
  self.cnv5 = nn.Conv2d(base_features*8, base_features*16, kernel_size = 3, padding = 1)
  self.rel1 = nn.ReLU()
  self.rel2 = nn.ReLU()
  self.rel3 = nn.ReLU()
  self.rel4 = nn.ReLU()
  self.rel5 = nn.ReLU()
  self.relo = nn.ReLU()
  self.max1 = nn.MaxPool2d(2, 2)
  self.max2 = nn.MaxPool2d(2, 2)
  self.max3 = nn.MaxPool2d(2, 2)
  self.max4 = nn.MaxPool2d(2, 2)
  self.max5 = nn.MaxPool2d(2, 2)
  self.flat = nn.Flatten()
  num_of_features = int(IMG_SIZE/(2**num_of_layers))
  num_of_features *= num_of_features
  num_of_features *= base_features * (2**(num_of_layers-1))
  self.fc1 = nn.Linear(num_of_features, hidden_neurons)
  self.fc2 = nn.Linear(hidden_neurons, CLASSES)
```

Figure 61 Basic CNN

4.3.12.2 Transfer learning

Transfer learning is a very promising technique where models pretrained on a different task transfer their knowledge to a new task. This is done by first downloading the weights, freezing the layers, and replacing the classifier with a custom one for the current task. This is done in Figure 62 with an example of ResNet50.

```
# bases of all transfer learning
class BaseTransfer(BaseModel):
  def __init__(self, model, pretrained):
    super().__init__()
    self.pretrained = pretrained
    # whether to use pretrained weight anf reeze, or train from scratch
    if pretrained:
      print("Using pretrained weights")
      weights="DEFAULT"
    else:
      print("Training from scratch")
      weights=None
    self.backbone = model(weights=weights)
    layers = list(self.backbone.children())[:-1]
    self.feature_extractor = nn.Sequential(*layers)
    if pretrained:
      # layers are frozen by using eval()
      self.feature_extractor.eval()
      # freeze params
      for param in self.feature_extractor.parameters():
        param.requires_grad = False


  def forward(self, x):
    representations = self.feature_extractor(x).flatten(1)
    x = self.classifier(representations)
    return x

class ResNet50(BaseTransfer):
  def __init__(self, model=resnet50, pretrained=IS_PRETRAINED):
    super().__init__(model, pretrained)
    num_filters = self.backbone.fc.in_features
    self.classifier = nn.Linear(num_filters, CLASSES)

  def forward(self, x):
    return super().forward(x)
```

Figure 62 Transfer learning

4.3.12.3 Residual attention

Not only was transfer learning used, but attention as well. The purpose is to have the model focus more on more prominent facial features, such as around the eyes and the top of the head an example network is shown in Figure 63.

```
class ResidualAttentionModel(BaseModel):
    # for input size 224
    def __init__(self):
        super(ResidualAttentionModel, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias = False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)
        )
        self.mpool1 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.residual_block1 = ResidualBlock(64, 256)
        ....
        self.mpool2 = nn.Sequential(
            nn.BatchNorm2d(2048),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=7, stride=1)
        )
        self.fc = nn.Linear(2048,10)

    def forward(self, x):
        out = self.conv1(x)
        out = self.mpool1(out)
        # print(out.data)
        out = self.residual_block1(out)
        out = self.attention_module1(out)
        out = self.residual_block2(out)
        ...
        out = self.residual_block6(out)
        out = self.mpool2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)

        return out
```

Figure 63 Residual attention

4.3.12.4 Multitasking

Since the AGE tasks are very similar, as they all use the face for classification, there are probably multiple shared features between them. To make use of this, a multitasking model was created that learns features for all the tasks at the same time, then use 3 fully connected layers at the end, one for each task as shown in Figure 64

```
    out = self.flat(out)
    out = self.doo(self.relo(self.fc1(out)))
    if Multitasking:
      out_a = self.fc_a(out)
      out_g = self.fc_g(out)
      out_e = self.fc_e(out)
      return {'a': out_a, 'g': out_g, 'e': out_e}
    else:
      out = self.fc2(out)
      return out
```

Figure 64 Multitasking

4.3.12.5 Hierarchical models

It was hypothesized that there is less variance in age classifications if all samples were of the same gender and/or ethnicity. To test that further, a hierarchical model was created that classifies the gender first, then based on the classified gender, it uses a specialized age classification mode for that gender, as shown in Figure 65

```
# Classifying age based on predicted gender
elif classifying_option == 'apga':
  # Classifying gender
  CLASS_RANGES = LABEL_TO_GENDER
  CLASSES = len(CLASS_RANGES)
  CLASS_GENDER = True

  FINAL_CKPT = f'compressed_data/{GENDER_CKPT}'
  FacesModel = get_model(MODEL_IDX)()
  FacesModel = FacesModel.load_from_checkpoint(FINAL_CKPT, map_location=torch.device(device))
  kwargs_trainer = {
    "max_epochs": EPOCHS,
    "deterministic": deterministic,
  }
  if not device=='cpu':
    kwargs_trainer['accelerator']="gpu"
    kwargs_trainer['devices']=[USED_GPU_NUM]

  main_model_train(**get_kwargs_main_model("gender", None), predict_labels = True)

  # Classifying age
  CLASS_GENDER = False
  CLASS_AGE = True
  FINAL_CKPT = f'compressed_data/{AGE_CKPT}'

  df_copy_male = df_copy[df_copy['predicted_gender_label']==0]
  df_copy_female = df_copy[df_copy['predicted_gender_label']==1]

  # Classifying age on predicted male
  main_model_train(**get_kwargs_main_model("male_age", df_copy_male))

  # Classifying age on predicted female
  main_model_train(**get_kwargs_main_model("female_age", df_copy_female))
```

Figure 65 Hierarchical models

5 Trials and Results

After explaining different hyperparameters, and showing in code how these hyperparameters are treated, this section will show different trials and their results. As mentioned before, more than 400 models were trained during this project. Since there is no way to mention all of them, I will list some of the key models that helped make various decisions. Each trial will include the hyperparameters used and a visualization of the highest F1 score (using tensorboard) for age classification, since that is the main goal of the project. Each trial is built on, and includes the previous hyperparameters, unless stated otherwise. A summary of the most important trials is shown in Table 2

Model variant	F1 score
Base Model	53.38%
UTKFace	55.96%
Age categories - 3 bins	84.31%
Age categories - 7 bins	65.89%
Hierarchical model	72.45%
FaceNet	69.04%
Residual attention	69.95%
Multitasking	70.71%
Optimal model AGECNN	**73.27%**

Table 2 Results of main trials

These results will be discussed and analysed in the next section. First, the metrics and loss functions used are mentioned below.

5.0 Metrics and loss functions

Different metrics are used to evaluate the efficiency of models in deep learning, while loss functions are used to optimise the model during training. A loss function, also known as an objective function or cost function, is a metric used in machine learning and optimisation algorithms to quantify the difference between predicted and actual outputs or labels. The objective of a loss function is to provide a quantifiable measure of a model's performance and to direct the learning process towards minimising this error. In tasks involving supervised learning, in which the model learns from labelled examples, the loss function compares the model's predictions to the ground truth labels. It computes a singular scalar

value that represents the error or loss incurred by the model on a particular sample or group of samples. The selection of a loss function is contingent upon the nature of the problem being addressed and the type of output produced by the model (e.g., regression or classification). Listed below are the accuracy metrics used in this project:

5.0.1 Accuracy

Accuracy measures the proportion of instances that are correctly classified. It is frequently employed for classification tasks. For multi-class classification, the corresponding loss function is typically categorical cross-entropy and for binary classification, binary cross-entropy [25].

5.0.2 Precision and Recall

Recall is the ratio of true positives to the sum of true positives and false positives, while precision is the ratio of true positives to the sum of true positives and false positives. These metrics are frequently employed in binary classification, particularly when working with unbalanced data sets. Precision and recall share the same loss function as accuracy, namely categorical cross-entropy or binary cross-entropy. A comparison between these metrics can be shown in Figure 66

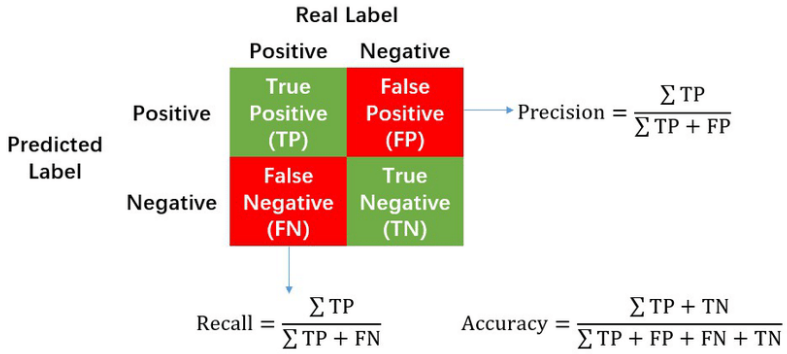

Figure 66 Accuracy vs precision vs recall

5.0.3 F1-Score:

The F1-score incorporates accuracy and recall into a single, balanced metric. It is commonly used in assignments involving imbalanced classification. Due to the fact that F1-score is derived from precision and recall, the corresponding loss function remains categorical or binary cross-entropy. As a result, F1 will be the used metric to measure the performance of the following models.

5.1 Base model

This is the model where the first plateau was reached. At this point, the model was set to train on 35 epochs, using weighted loss and learning rate of 1e-4 Adam optimizer on a 4-layered cnn and on **multiple datasets** including UTK. You can see that it has early stopped. This model achieves a maximum of 53.38% F1 score, and early stops at epoch 14. The confusion matrix is presented in Figure 68

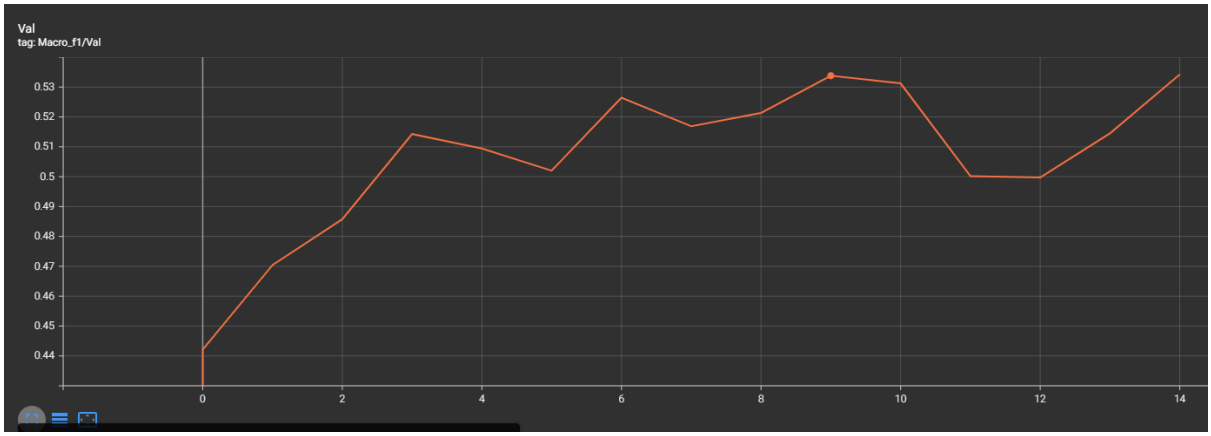

Figure 67 Base model, F1 53.38%

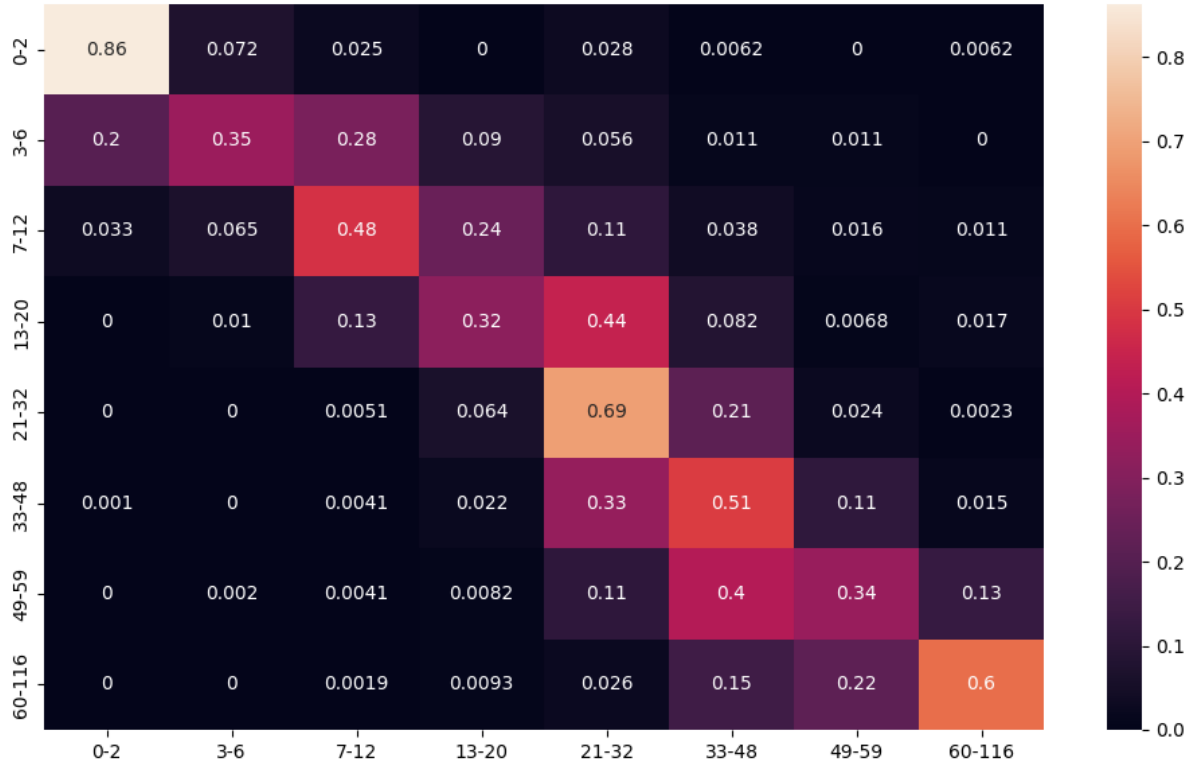

Figure 68 Base model confusion matrix

5.2 UTKFace model

It was pointed out that the use of multiple datasets most probably is adding too much variance to the data that ends up confusing the model. So, from here on, all the previous hyperparameters were the same, but the choice of dataset was only limited to UTKFace since it still carries enough variance with a lot of ethnicities and a balance of genders. That resulted in an increase of almost 2.5% performance, achieving an F1 score of 55.96%, and early stops at epoch 21. The confusion matrix is presented in Figure 70

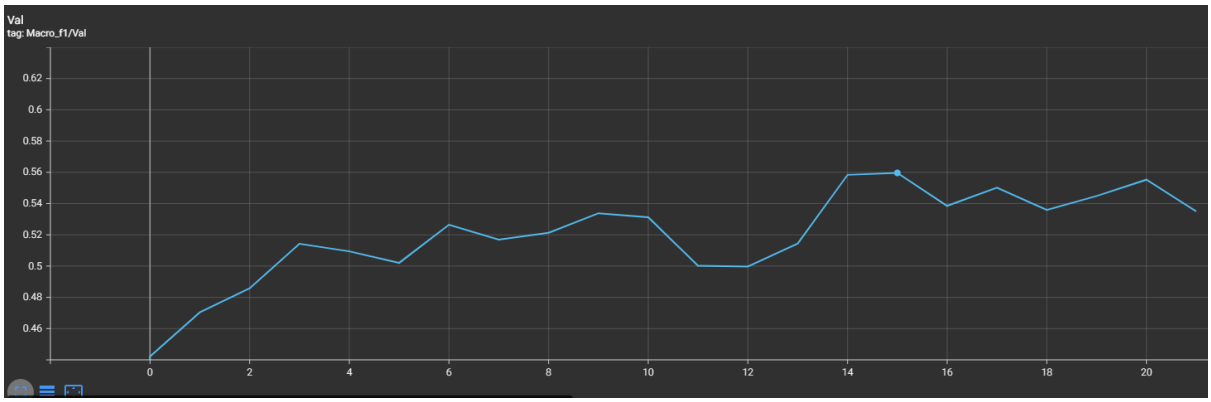

Figure 69 UTKFace model, F1 55.96%

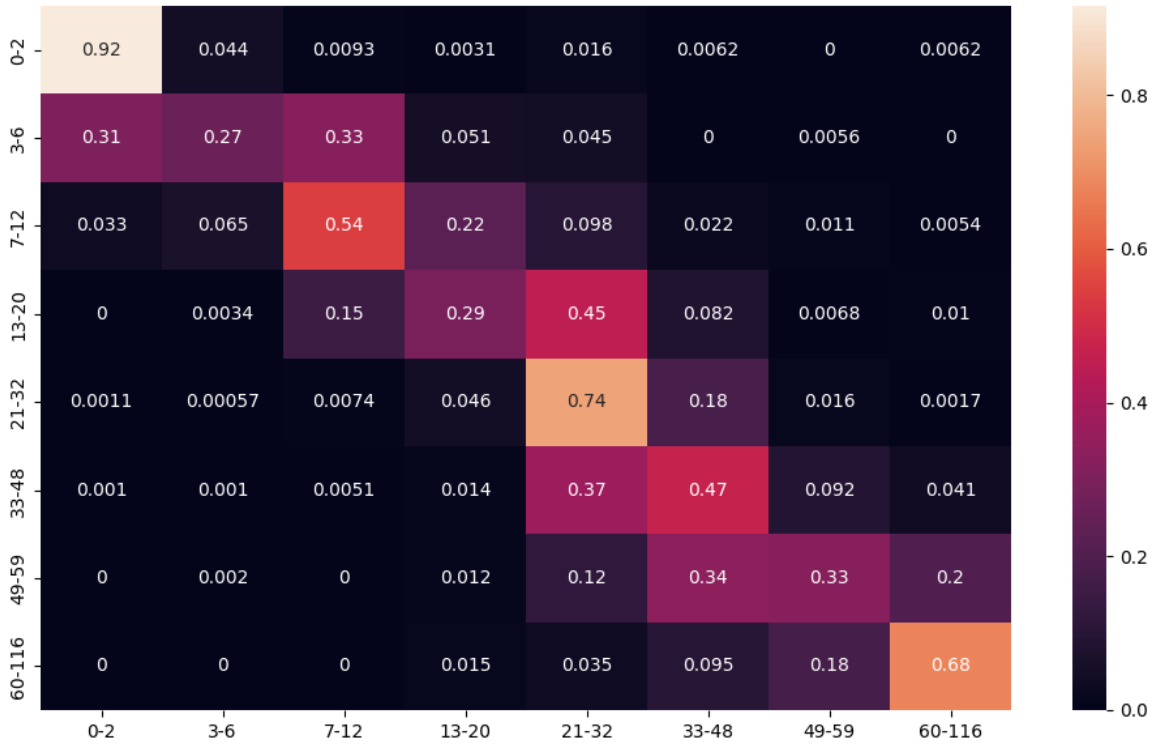

Figure 70 UTKFace model confusion matrix

5.3 DenseNet201

For this trial, a more complex model was chosen, to decide whether the traditional CNN is too simple for my model. With all other hyperparameters constant, a DenseNet201 was tried. Replacing the traditional cnn with a DenseNet model results in a maximum of 51.93% F1 score, and early stops at epoch 9. The confusion matrix is presented in Figure 72

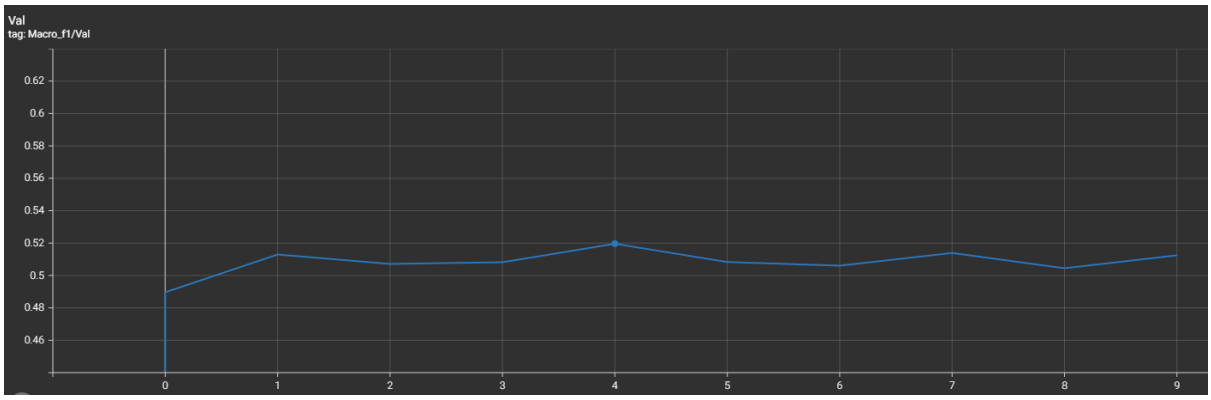

Figure 71 DenseNet201. F1 51.93%

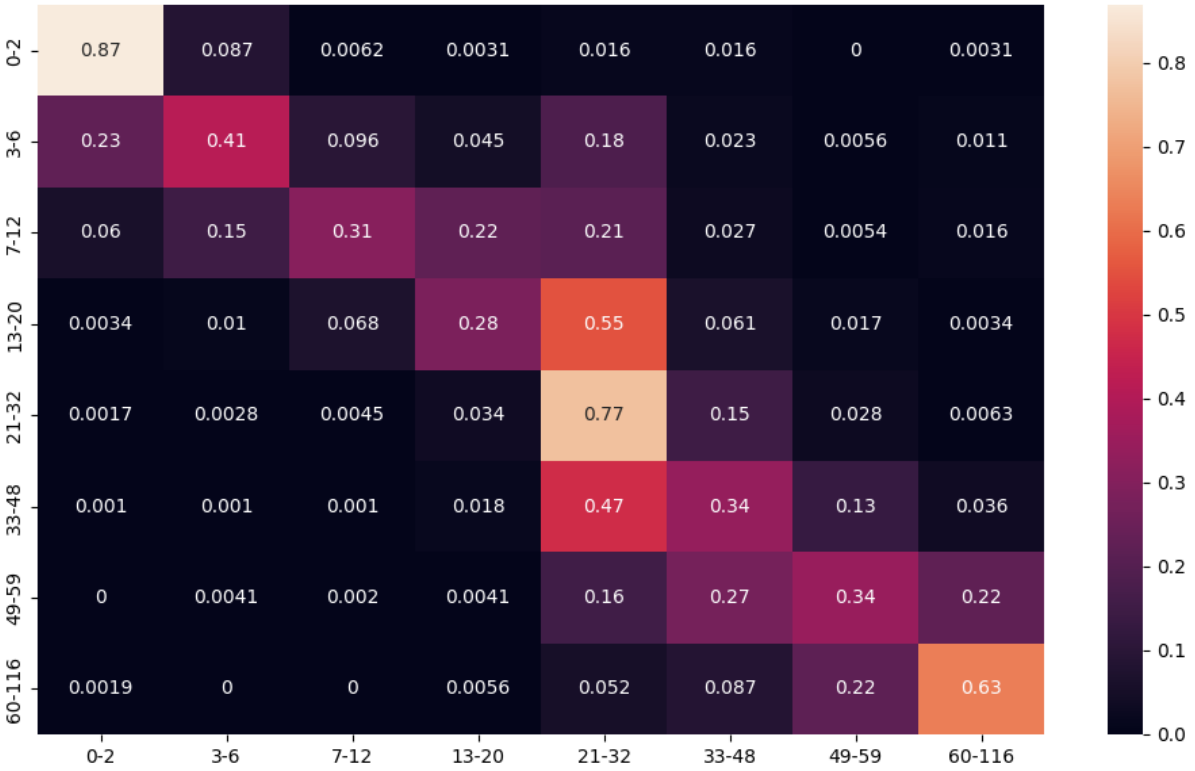

Figure 72 DenseNet201 confusion matrix

5.4 ResNet152

Similar to the previous trial, another more complex model was chosen for this one, and that is ResNet152. All other hyperparameters are constant beside the learning rate which was 0.001. Removing the densnet and adding a resnet instead achieves 47.56% F1 score, and early stops at epoch 15. The confusion matrix is presented in Figure 74

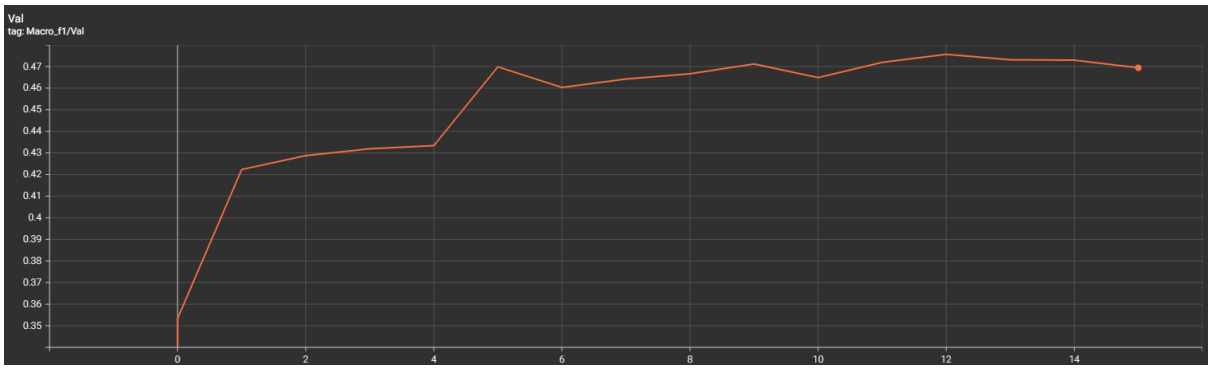

Figure 73 ResNet152. F1 47.56%

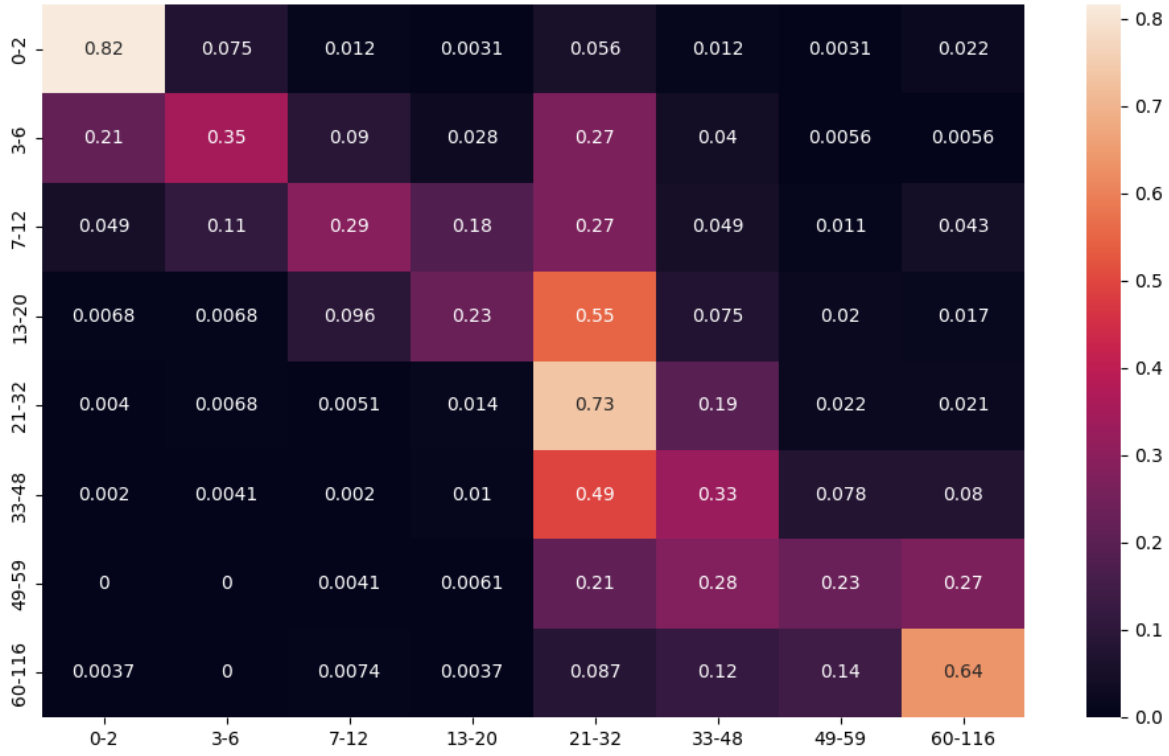

Figure 74 ResNet152 confusion matrix

5.5 Data augmentation

Now we are back to the 4-layered traditional CNN. Starting here, data augmentations were introduced. This was in preparation for adding a weighted sampler to the data. But first, the data had to be augmented so the model gets to see different images every time. Adding the data augmentations results in an F1 score of 53.38%, and early stops at epoch 23. The confusion matrix is presented in Figure 76

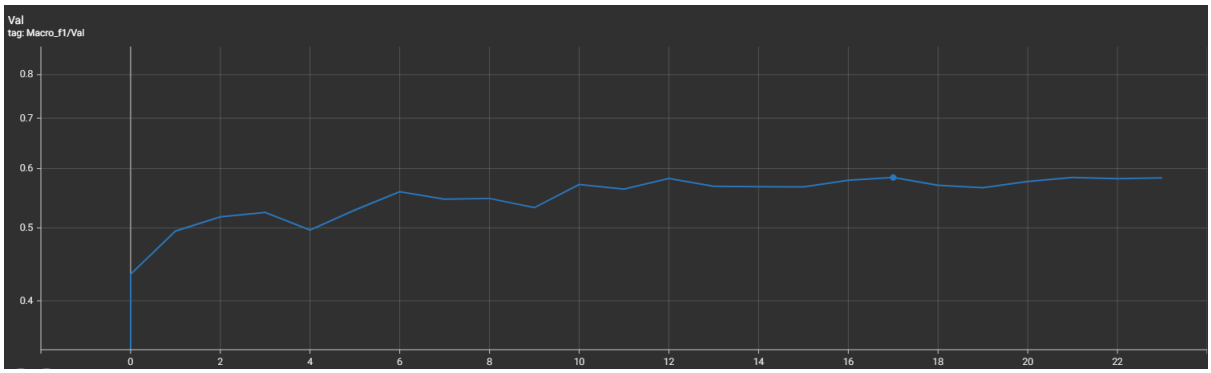

Figure 75 Data augmentation. F1 58.37%

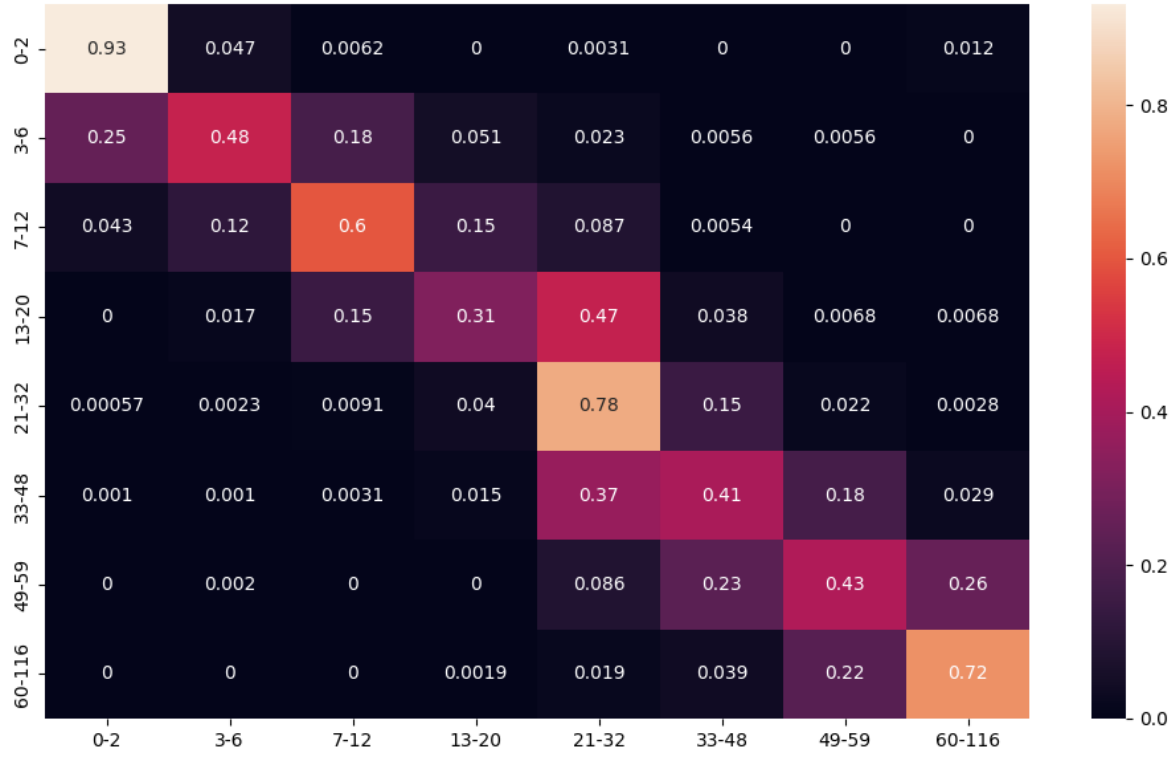

Figure 76 Data augmentaion confusion matrix

5.6 Age categories - 3 bins

During the literature review, I have noticed that different papers use different age categories in their classification. So, for this and the next few trials, I will be experimenting with different number of bins. Keeping in mind that so far there were 8 bins. For this trial, 3 age bins are used and a learning rate of 0.001. This change results in 84.31% F1 score, and early stops at epoch 16. The confusion matrix is presented in Figure 78

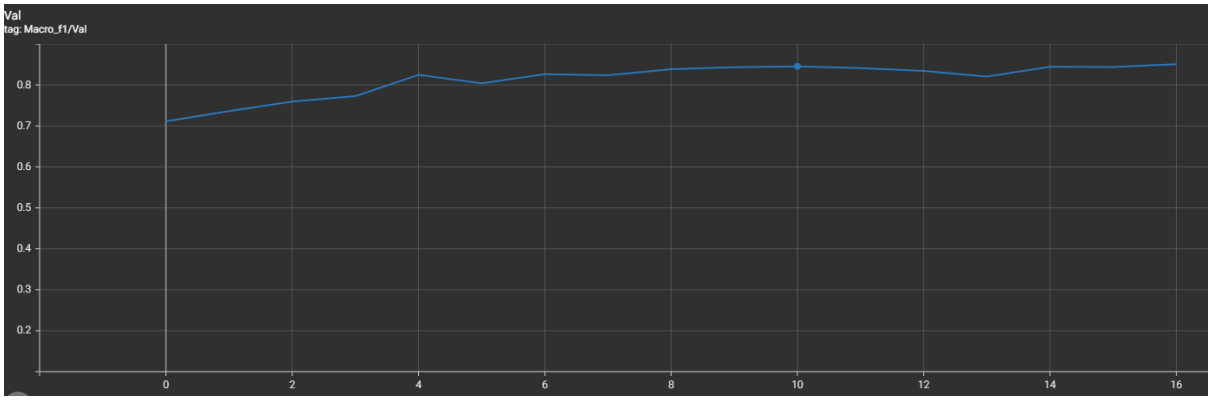

Figure 77 Age categories - 3 bins. F1 84.31%

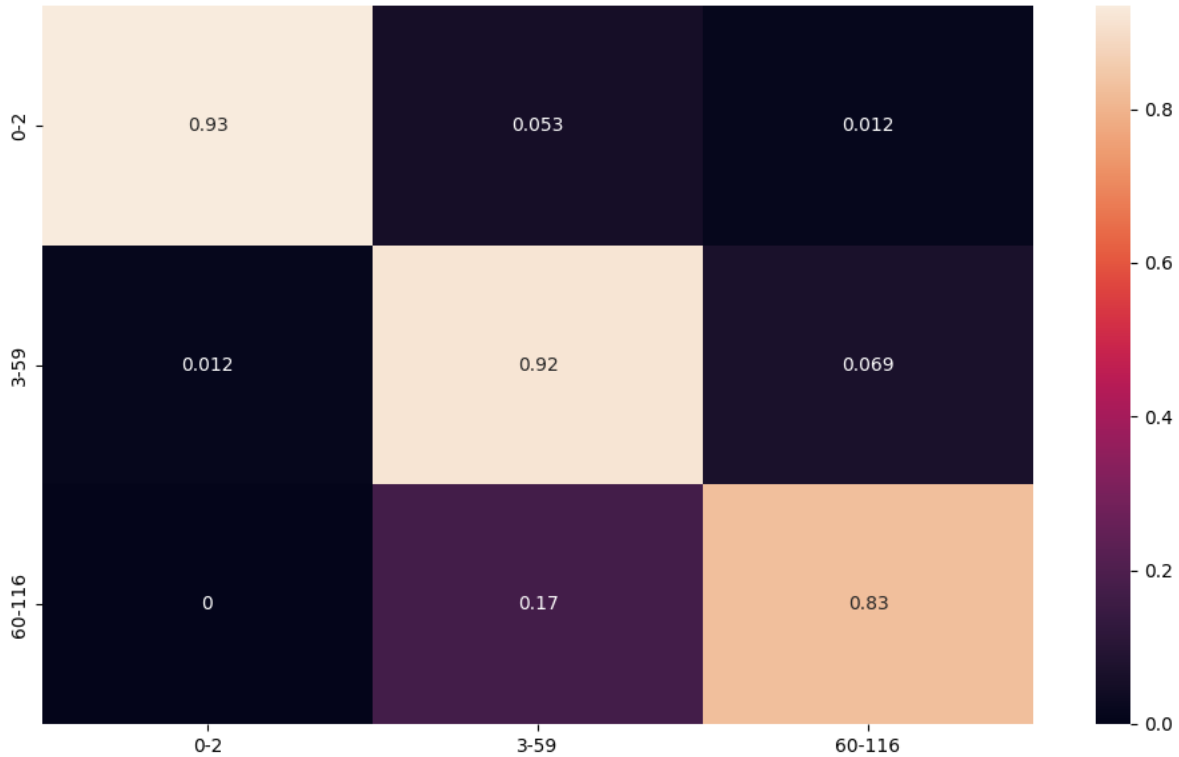

Figure 78 Age categories - 3 bins confusion matrix

5.7 Age categories - 4 bins

For this trial, the number of bins has been increased to four instead of three. Ages were categorised into these four groups: ['0-12', '13-19', '20-59', '60-116']. Adding an extra bin results in an F1 score of 75.10% and causes the model training to early stop at the 25th epoch. The confusion matrix for this model is shown in Figure 80

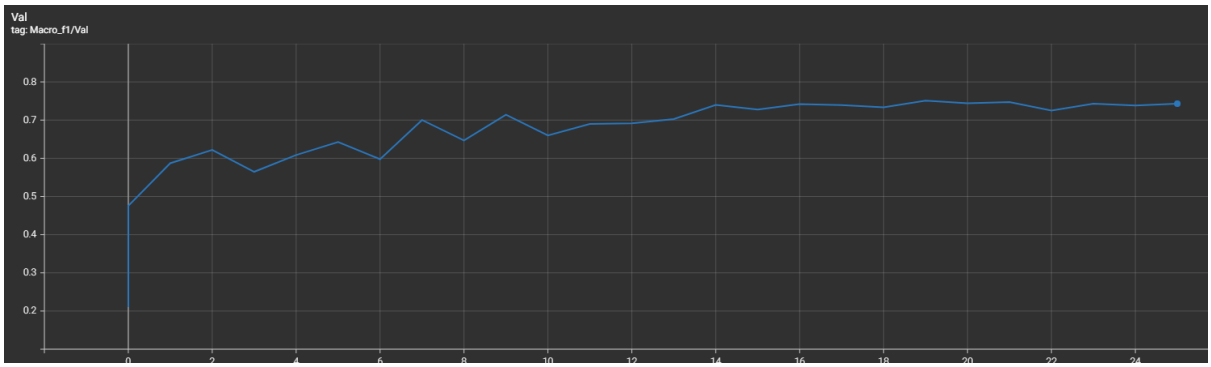

Figure 79 Age categories - 4 bins. F1 75.10%

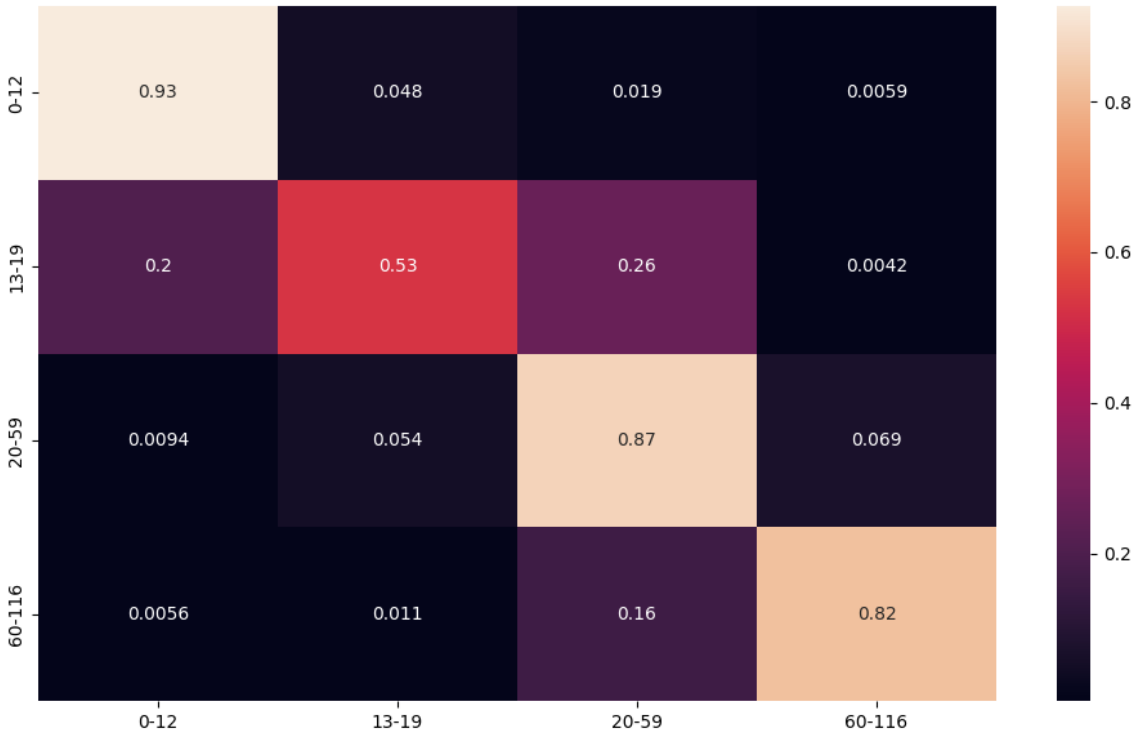

Figure 80 Age categories - 4 bins confusion matrix

5.8 Age categories - 7 bins

I have skipped listing the results for variants including five and six bins to avoid redundancy but showing the 7 bins trial is vital for the project. So, this variant uses the following age categories: ['0-2', '3-9', '10-16', '17-25', '26-45', '46-60', '61-116']. This model achieves a maximum of 63.89% F1 score, and early stops at epoch 24. The confusion matrix is presented in Figure 82

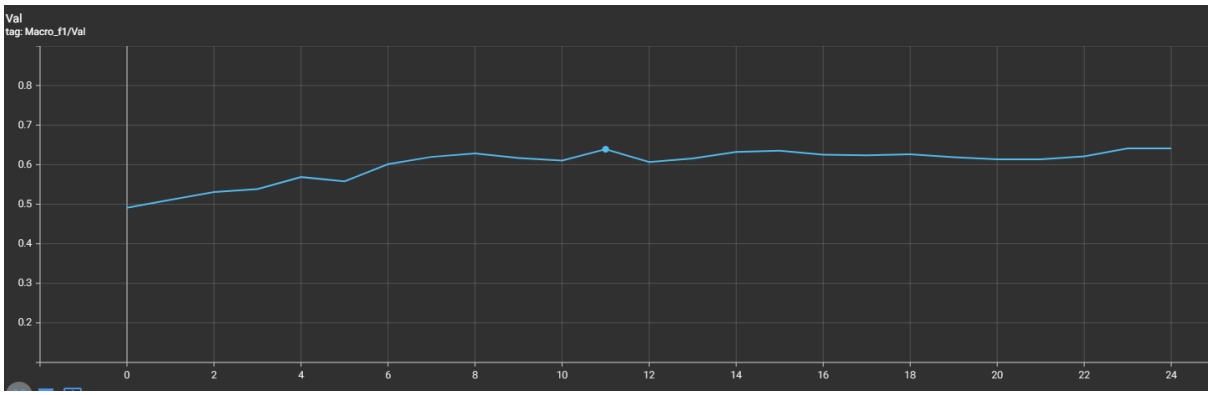

Figure 81 Age categories - 7 bins. F1 63.89%

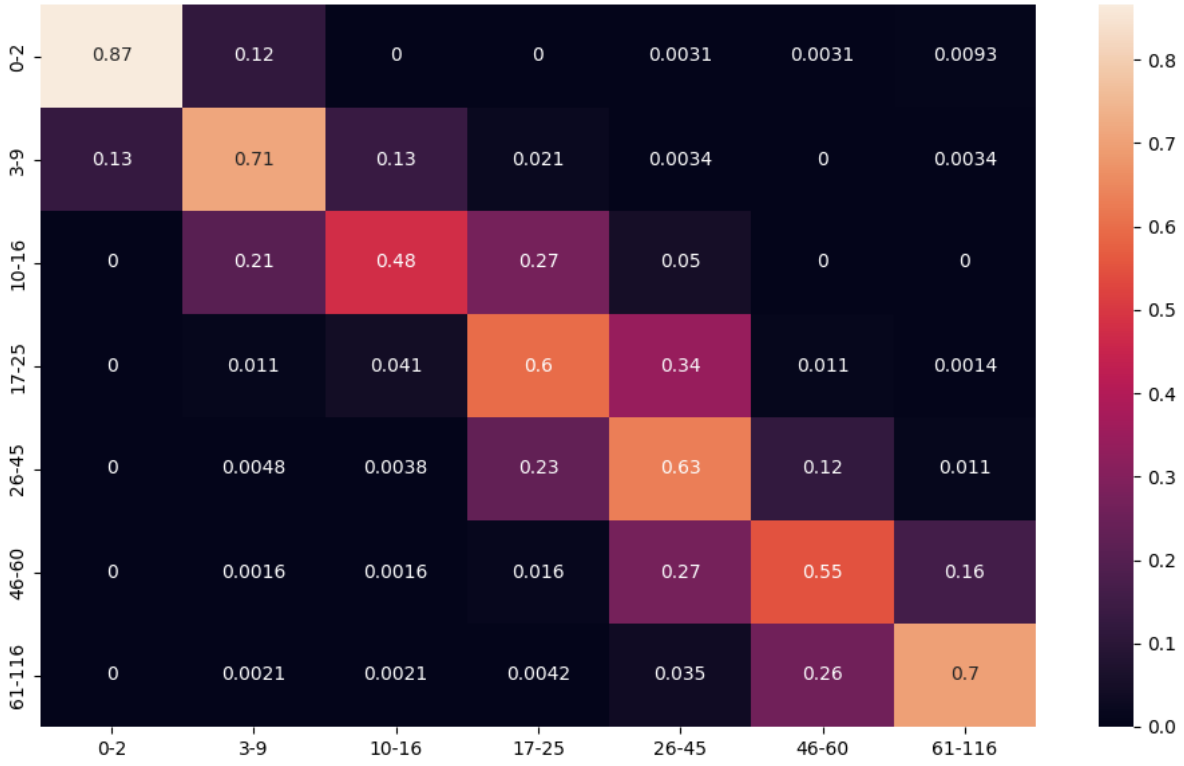

Figure 82 Age categories - 7 bins confusion matrix

5.9 Weighted sampler

Now that the data augmentations are ready and different age categories and models were tested. It is time to add a weighted sampler. This variant uses seven categories as before, a 4-layered CNN, and a weighted sampler along with data augmentations without weighted loss. This model was able to reach 67.29% F1 score, and early stopped at epoch 48. The confusion matrix is presented in Figure 84

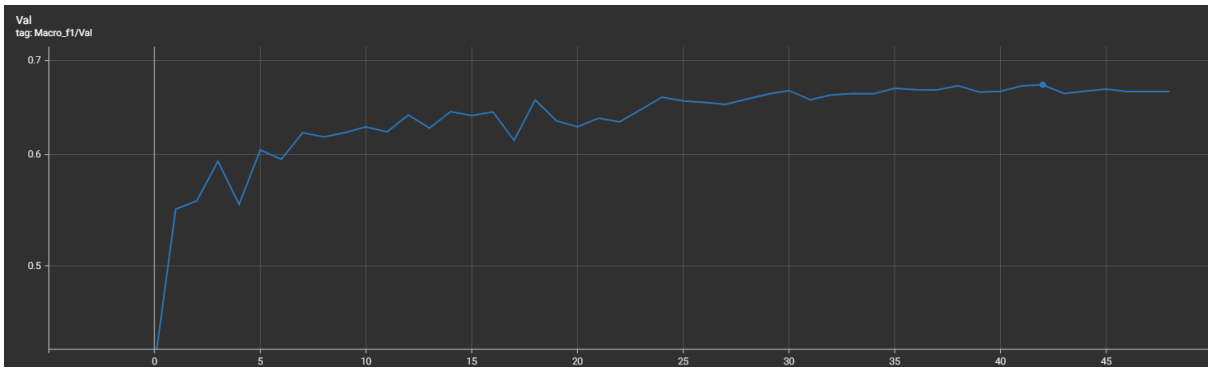

Figure 83 Weighted sampler. F1 67.29%

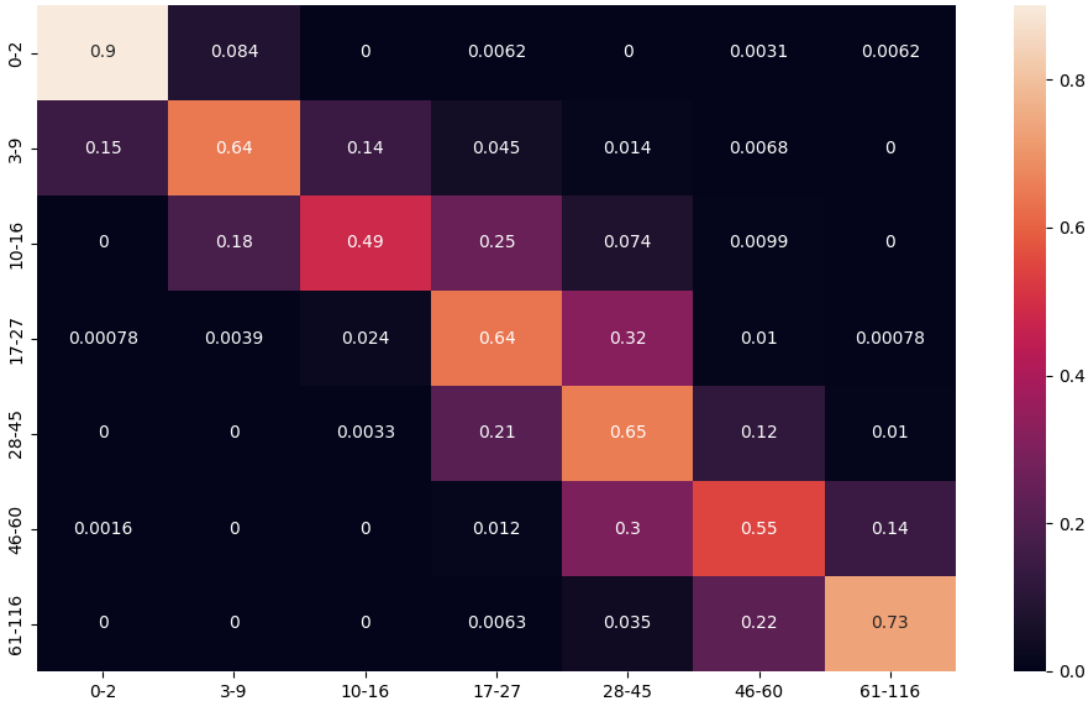

Figure 84 Weighted sampler confusion matrix

5.10 Greyscale

Now that performance increase has become harder, it is time to try lower impact parameters. One important hyperparameter portrayed in this trial is changing the image colour format from RGB to greyscale, with all other hyperparameters unchanged. This resulted in an F1 score of 68.82%, and early stops at epoch 43. The confusion matrix is presented in Figure 86

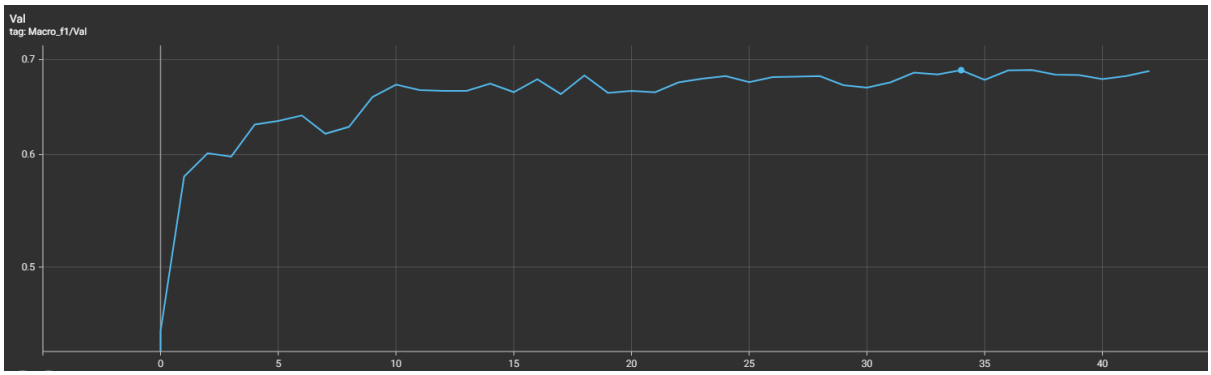

Figure 85 Greyscale. F1 68.82%

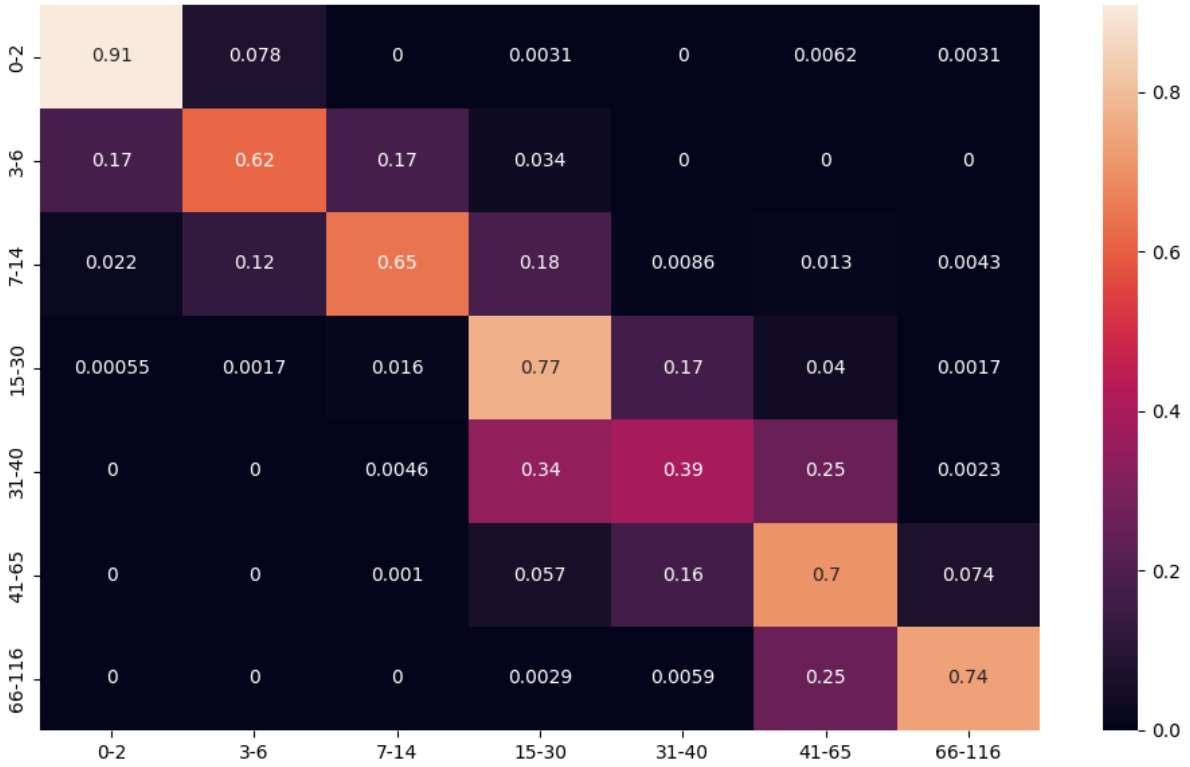

Figure 86 Greyscale confusion matrix

5.11 Dropout

In an effort to increase the model generalization, dropout was introduced. Several variants were tried including different dropout probabilities and placement (either on input, output, both, after activation function, before it, etc…) and listed is the best one, with a 0.6 probability on output and no dropout after the input or the hidden layers. This model achieves 69.85% F1 score, and early stops at epoch 34. The confusion matrix is presented in Figure 88

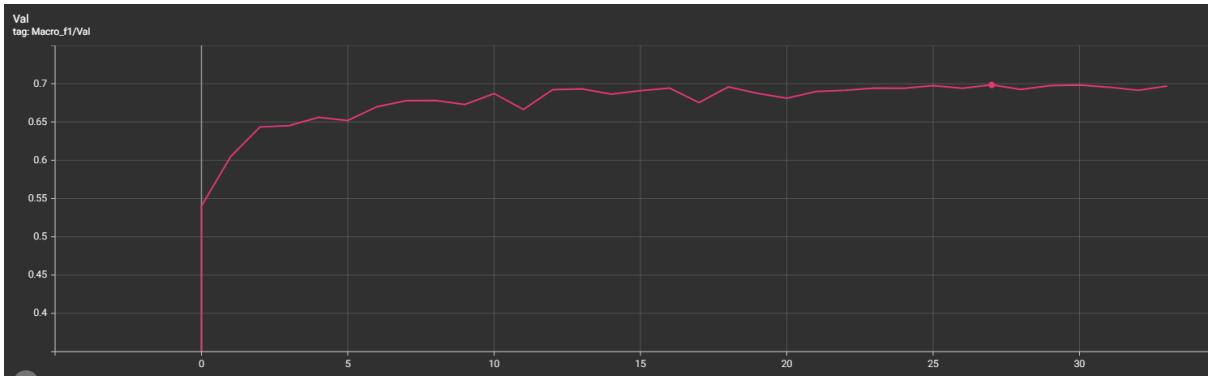

Figure 87 Dropout. F1 69.85%

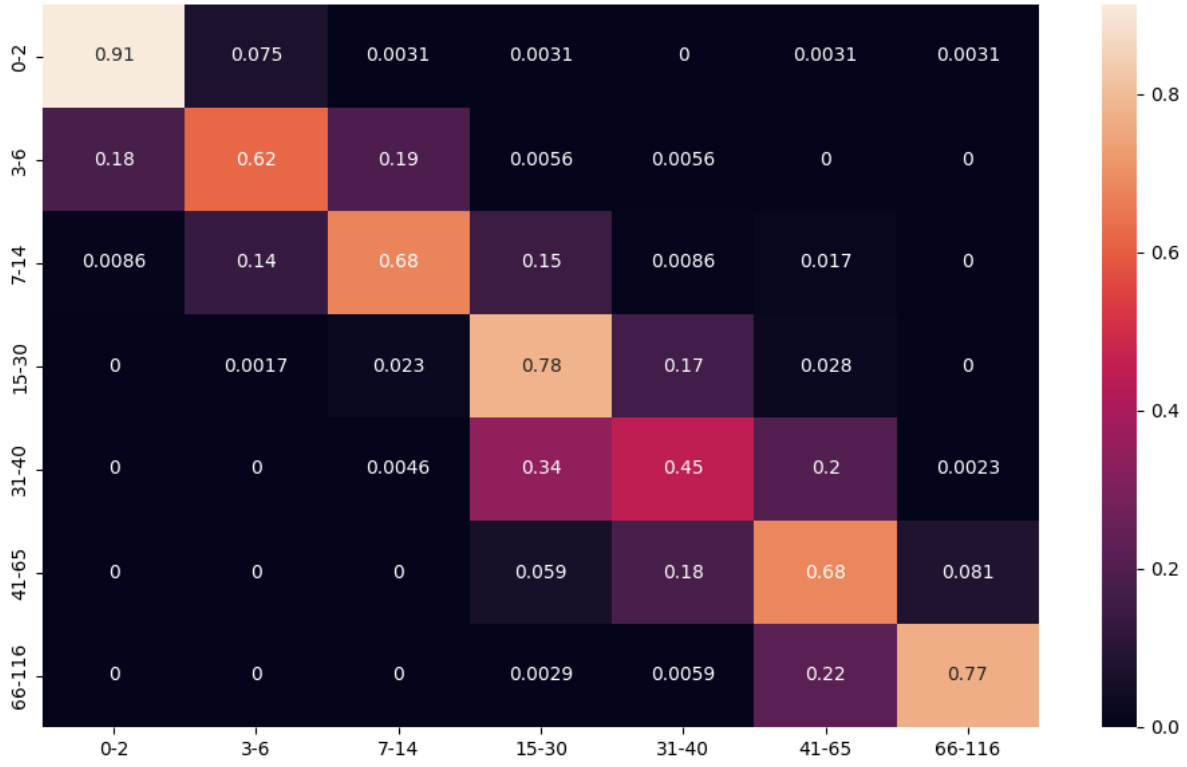

Figure 88 Dropout confusion matrix

5.12 RMSProb

So far, all models used Adam optimizer, so several optimizers were tested at this point and two of them will be listed below. The first one is RMSProb. After trying different alphas, momentum values, learning rates, and other hyperparameters, the best one is listed below with a learning rate of 1e-4 and 0.99 alpha with no momentum reaching an F1 score of 70.94%, and early stopped at epoch 34. The confusion matrix is presented in Figure 90

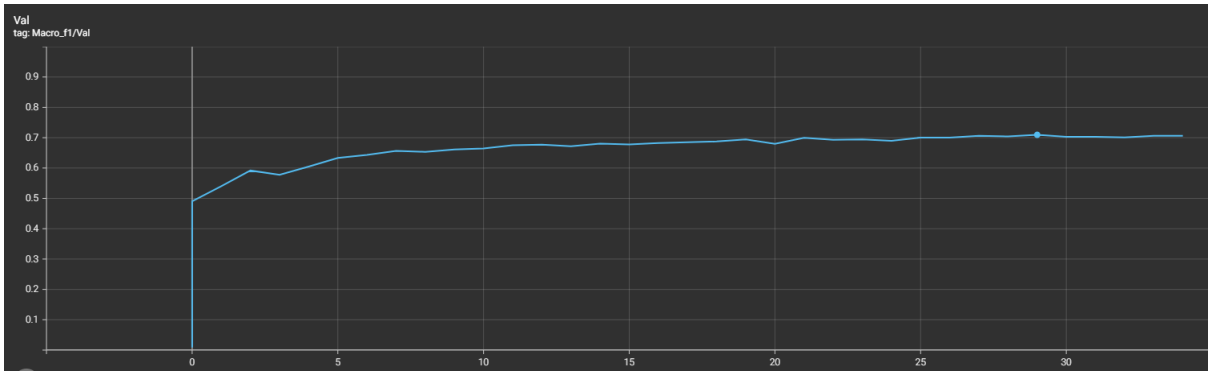

Figure 89 RMSProb. F1 70.94%

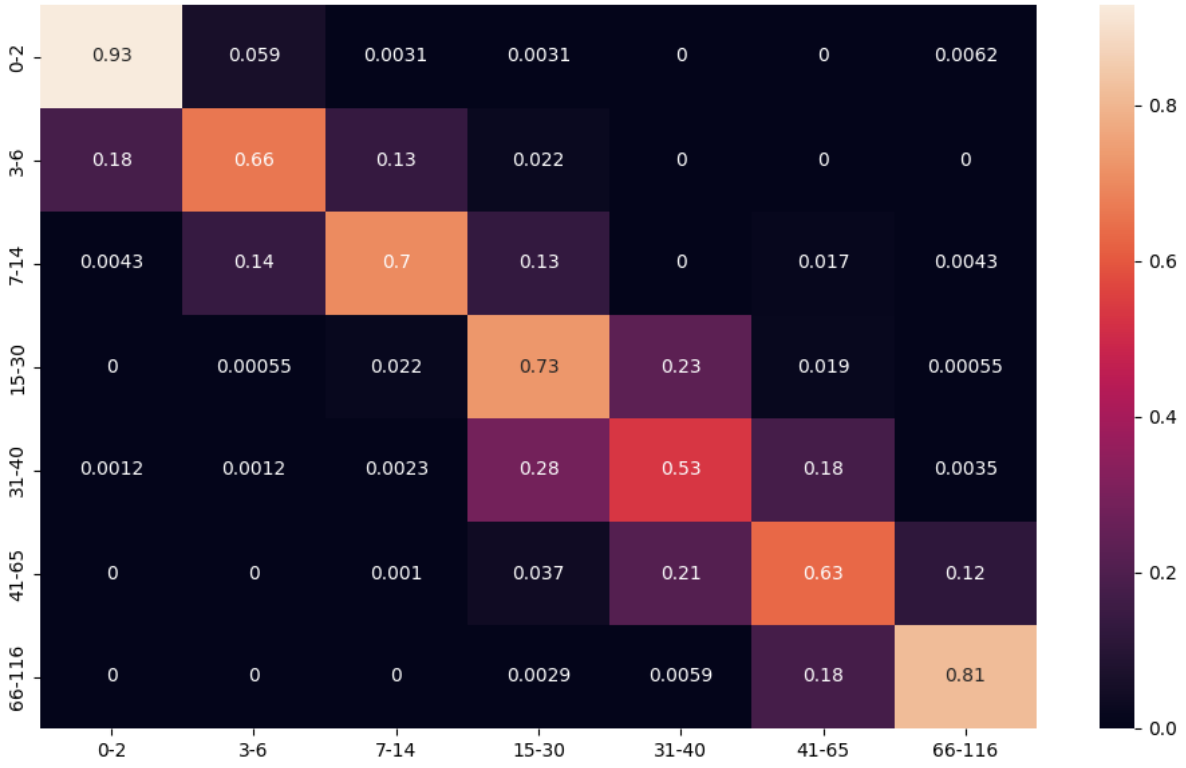

Figure 90 RMSProb confusion matrix

5.13 NAdam

After trying RMSProb in the previous trial, that optimizer got swapped with NAdam, which is very similar to Adam but incorporates Nesterov momentum explained earlier in chapter three with a learning rate of 0.001 and betas 0.9 and 0.999 respectively. This model achieves a maximum of 72.65% F1 score, and early stops at epoch 57. The confusion matrix is presented in Figure 92

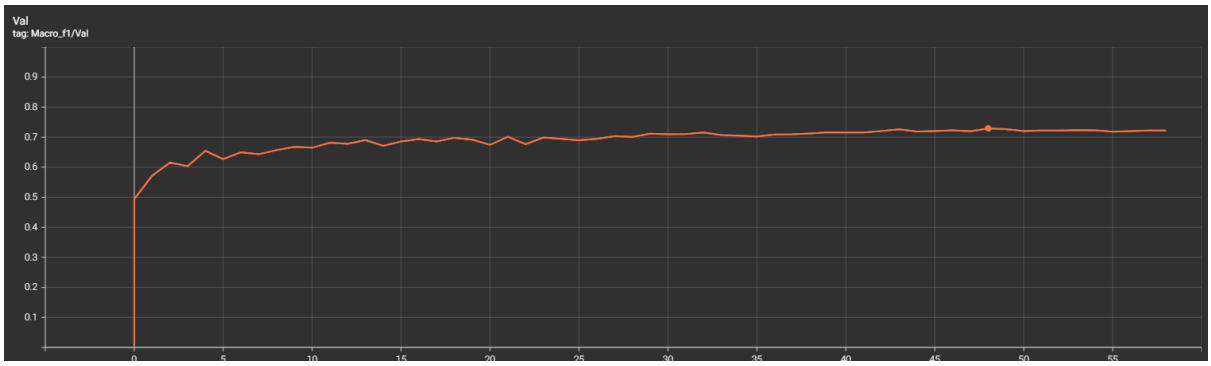

Figure 91 NAdam. F1 72.65%

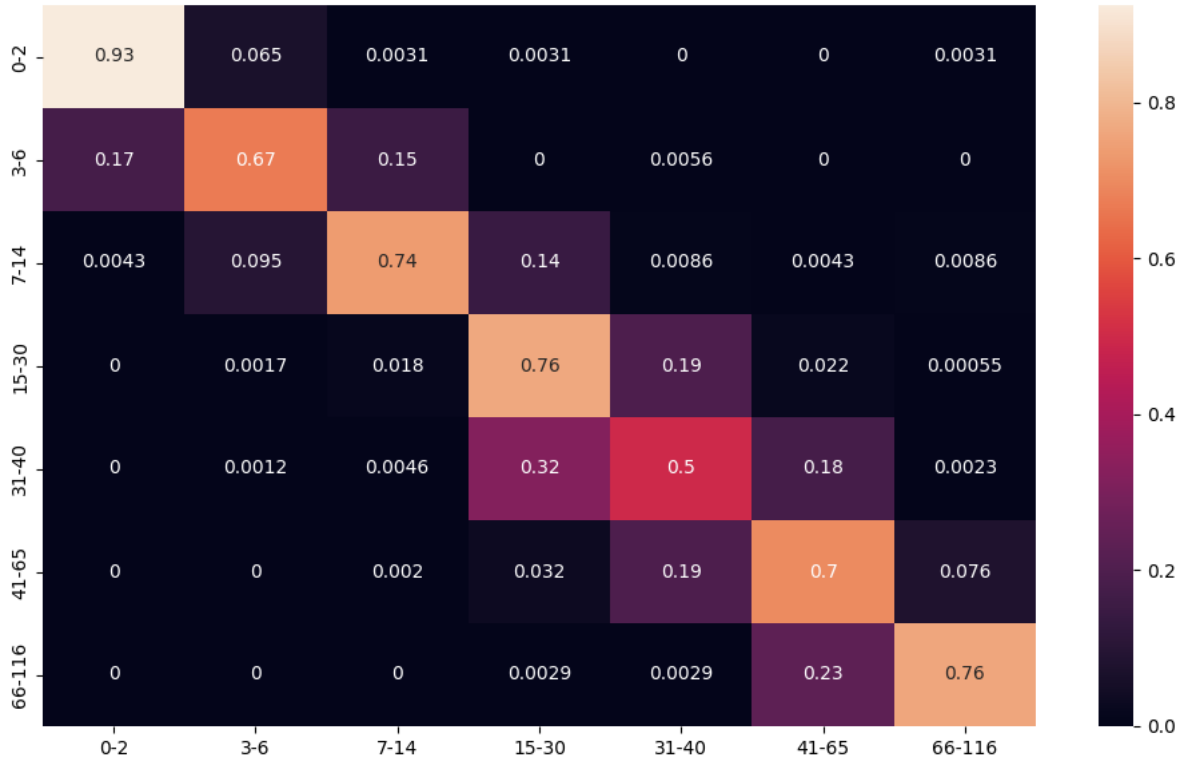

Figure 92 NAdam confusion matrix

5.14 Hierarchy - gender - misleading (data leakage)

Starting here, other techniques beside traditional training with basic CNNs will be discussed. For the next 3 trials, the focus will be on hierarchical learning where age is classified based on gender or ethnicity. The first model here predicts gender then age. The results look very promising but there is a **grave error of data leakage that causes inflation of results making them unreliable** that will be explained further in the analysis section. This model achieves 90.00% F1 score, and early stops at epoch 49.

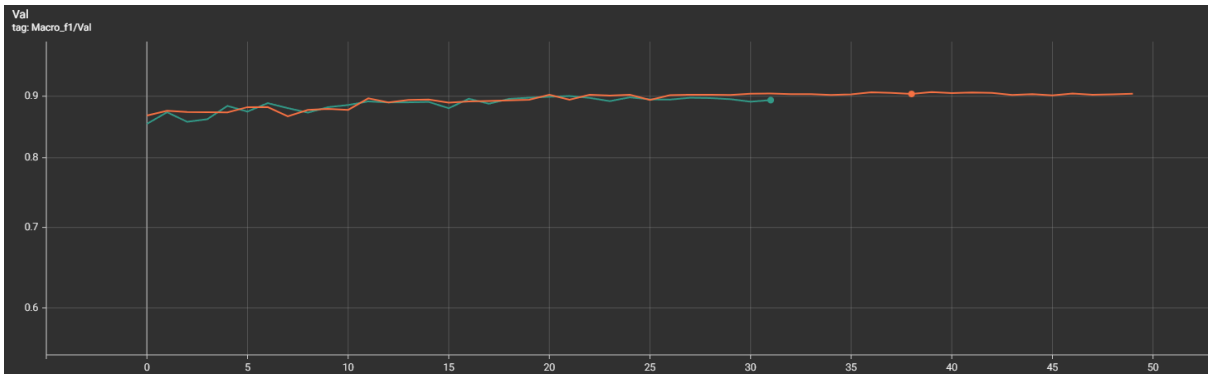

Figure 93 Hierarchy - gender - leakage. Avg F1 90%

5.15 Hierarchy - gender - fixed

After fixing the data leakage mentioned above (how it is fixed is mentioned in chapter 6 below), these are the reliable results for hierarchy based on age, where there are two different age models trained. The first one is trained on male images with an f1 score of 72.95%, while the second one is trained only on images of females achieving 71.97%, resulting in an average of 72.45%.

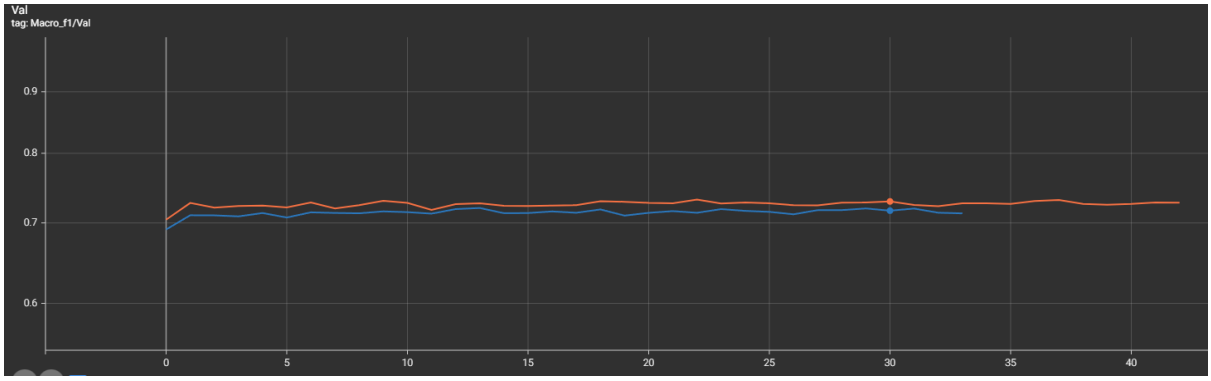

Figure 94 Hierarchy - gender - fixed. Avg F1 72.45%

5.16 Hierarchy - ethnicity - fixed

This trial shares most of its setup with the previous trial, the difference is the hierarchy in this trial is based on ethnicity instead of gender, which means that there are five age models, one for each ethnicity:

white, black, Asian, Indian and others (include Hispanic, middle eastern, etc…). The five models achieve an average of 71.24% with the highest ethnicity being white and the lowest being the 'other' label.

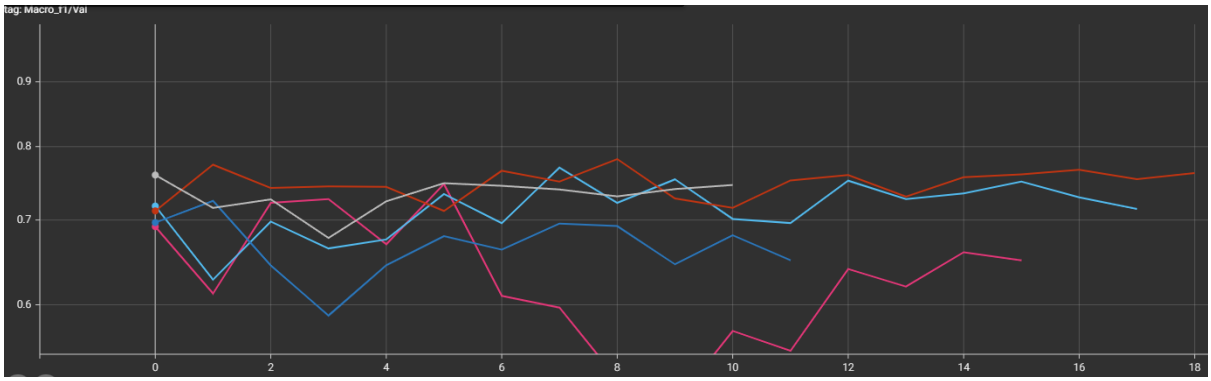

Figure 95 Hierarchy - ethnicity - fixed. Avg F1 71.24%

5.17 Transfer learning - FaceNet

For this trial, transfer learning was used. Instead of using the usual ImageNet weights, the weights of FaceNet will be used, which was pretrained on facial recognition using CASIA-WebFace dataset. The idea is features learnt for face recognition should be similar, and helpful, in classifying age from faces. This model achieves a maximum of 69.04% F1 score, and early stops at epoch 48. The confusion matrix is presented in Figure 97

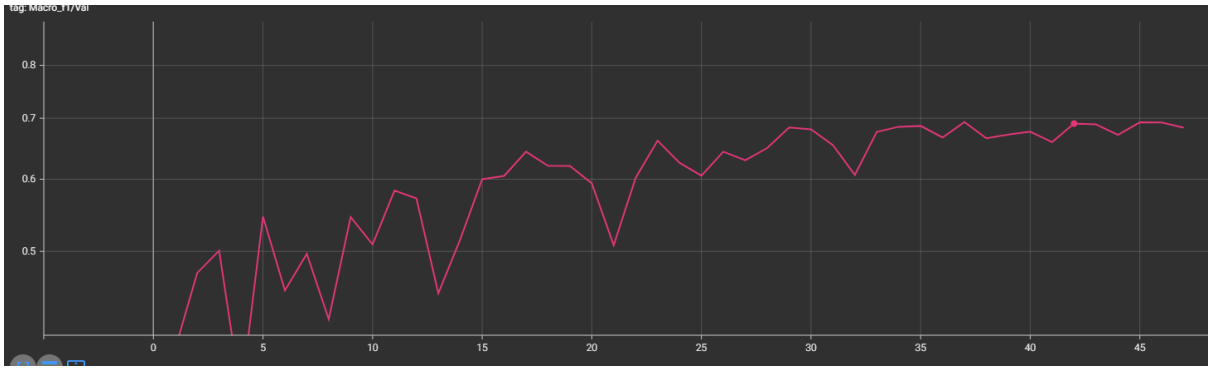

Figure 96 InceptionResnetv1 facenet. F1 69.04%

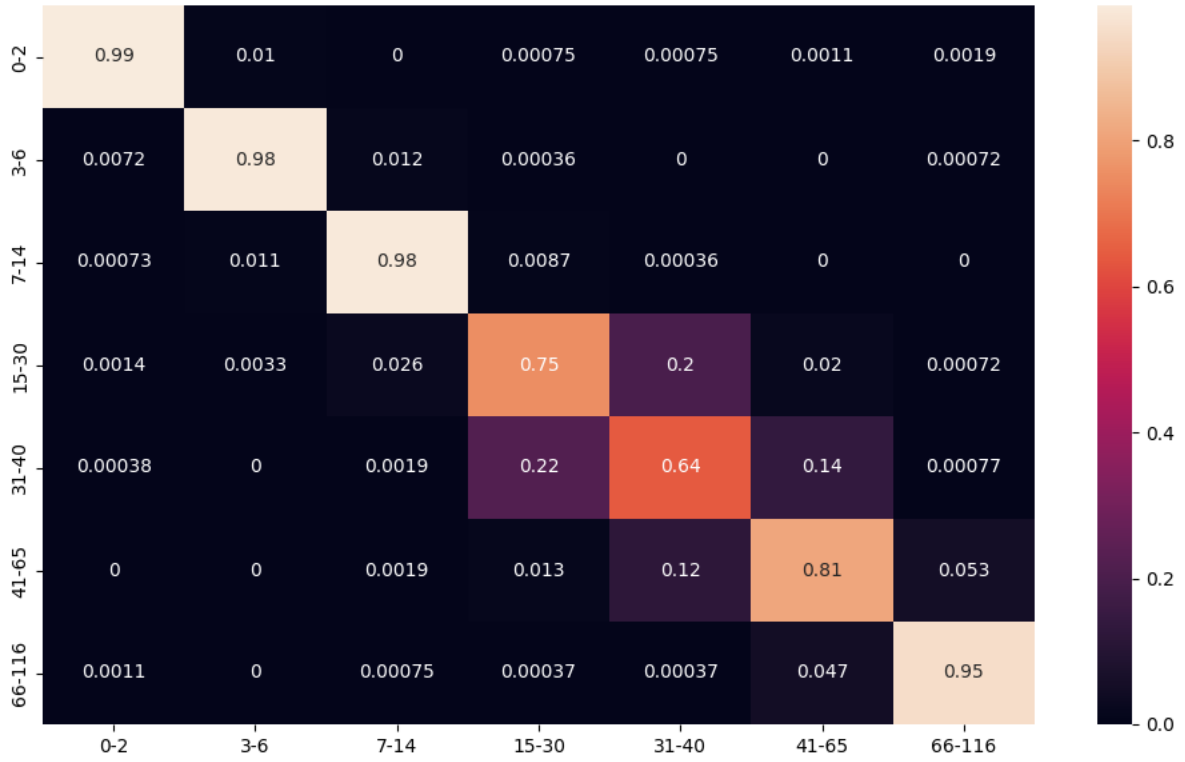

Figure 97 Transfer learning - FaceNet confusion matrix

5.18 Residual attention

For this trial, a more complex model was used. Namely ResnetInception V1 with residual attention added between its layers. Incorporating attention modules within residual blocks, allows the model to selectively enhance or diminish features based on their importance as explained previously in chapter 3. Using this model with residual attention achieved an F1 score of 69.95%.

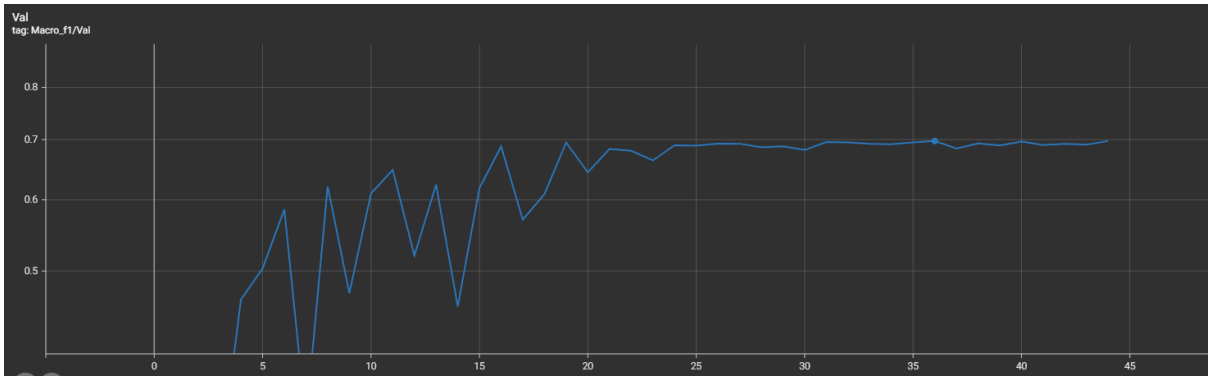

Figure 98 Residual Attention. F1 69.95%

5.19 Multitasking

In this trial, multitasking was used. Where the three tasks of AGE were classified at the same time using shared features and hard parameter sharing. Each task is classified by a fully connected layer at the end with 132 hidden neutons. The F1 score results for age, gender, and ethnicity are 70.71%, 93.01% and 74.95% respectively.

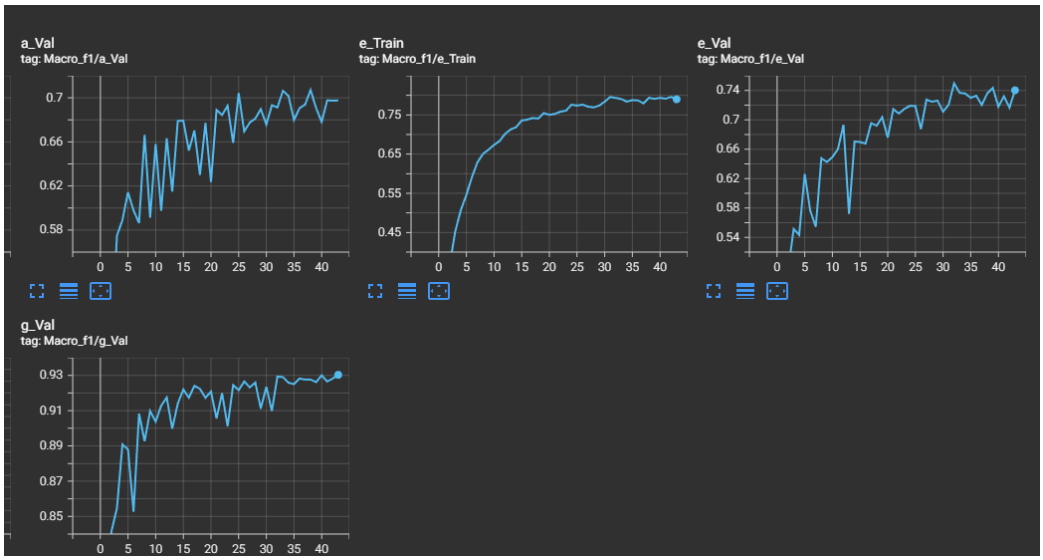

Figure 99 Multitasking. F1 70.71%

5.20 Final optimal model AGECNN

The final trial shows the optimal model after making the best decisions based on the 400+ previous models. The final hyperparameters are (copied from tensorboard):

{'IMG_SIZE': 224, 'WEIGHTED_SAMPLER': True, 'USE_TRANSFORMS': True, 'USE_GREYSCALE': True, 'LR': 0.0001, 'MODEL_NAME': 'Basic_4cnv', 'CLASS_RANGES': ['0-2', '3-6', '7-14', '15-30', '31-40', '41-65', '66-116'], 'OPTIM': {'optimizer': 'NAdam', 'betas': (0.9, 0.999), 'eps': 1e-08, 'weight_decay': 0.1}, 'base_features': 64, 'hidden_neurons': 132}

The hyperparameters are the same for gender and ethnicity model. The only difference is that gender has base features = 32 and hidden neurons = 64, while ethnicity has both equal 64. The results for age, gender and ethnicity are 73.27%, 80.25%, and 93.41% respectively. The confusion matrices for the three tasks are displayed below.

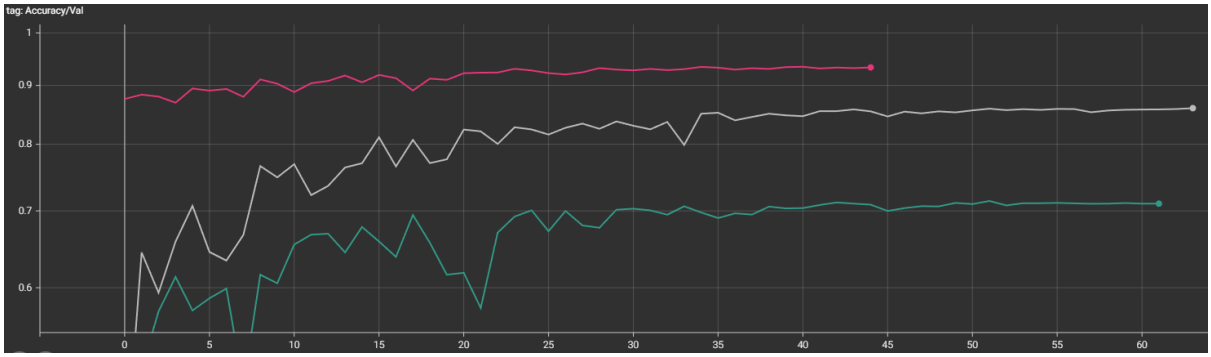

Figure 100 Final optimal model. F1 73.27%

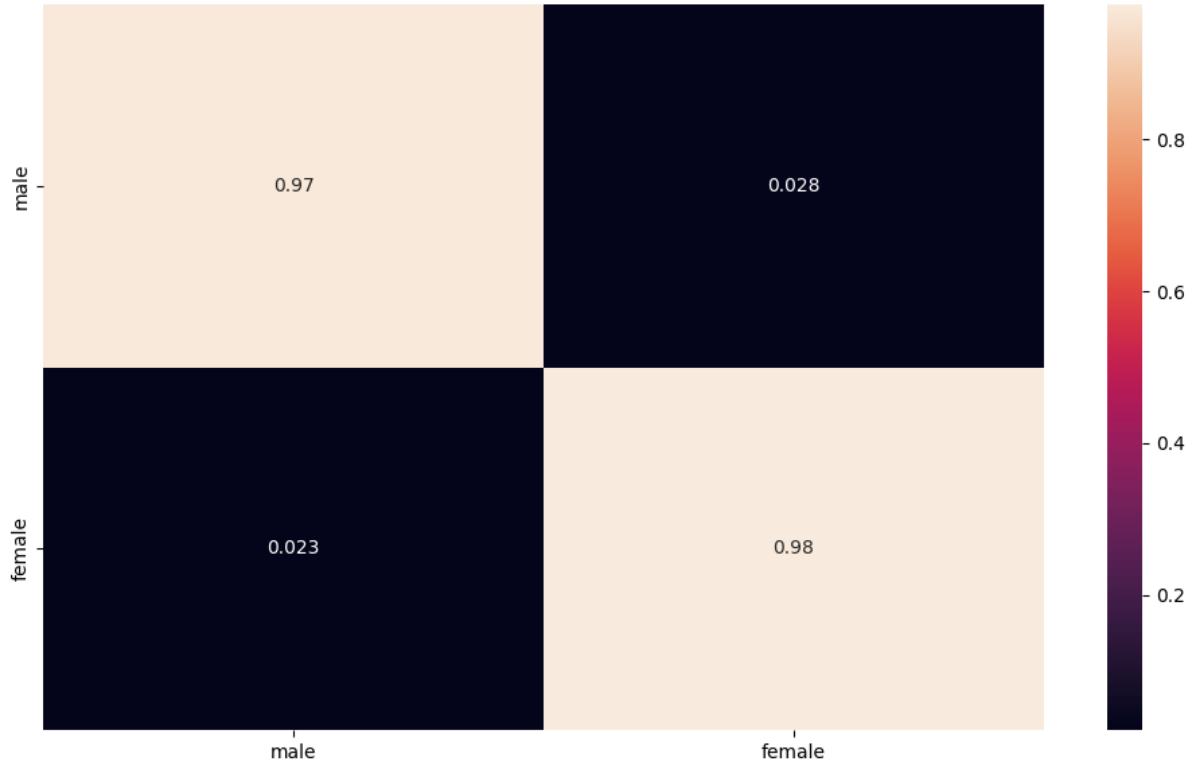

Figure 101 Optimal model gender confusion matrix

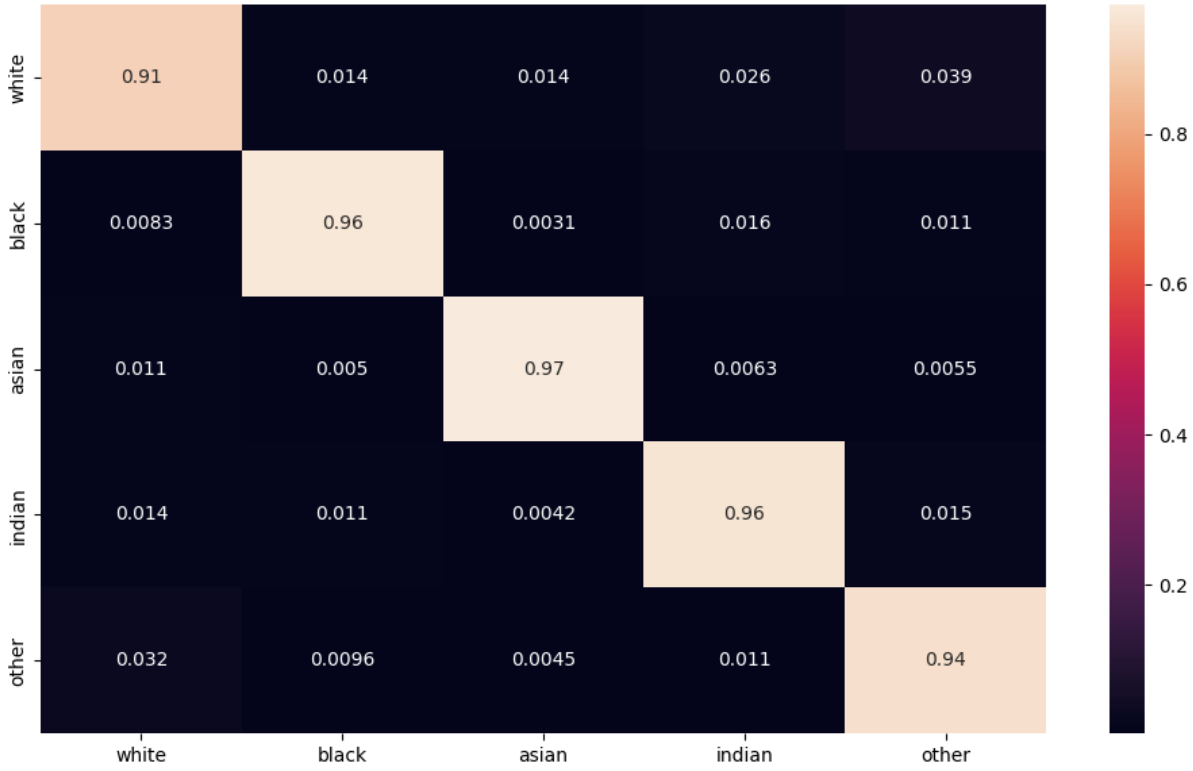

Figure 102 optimal model ethnicity confusion matrix

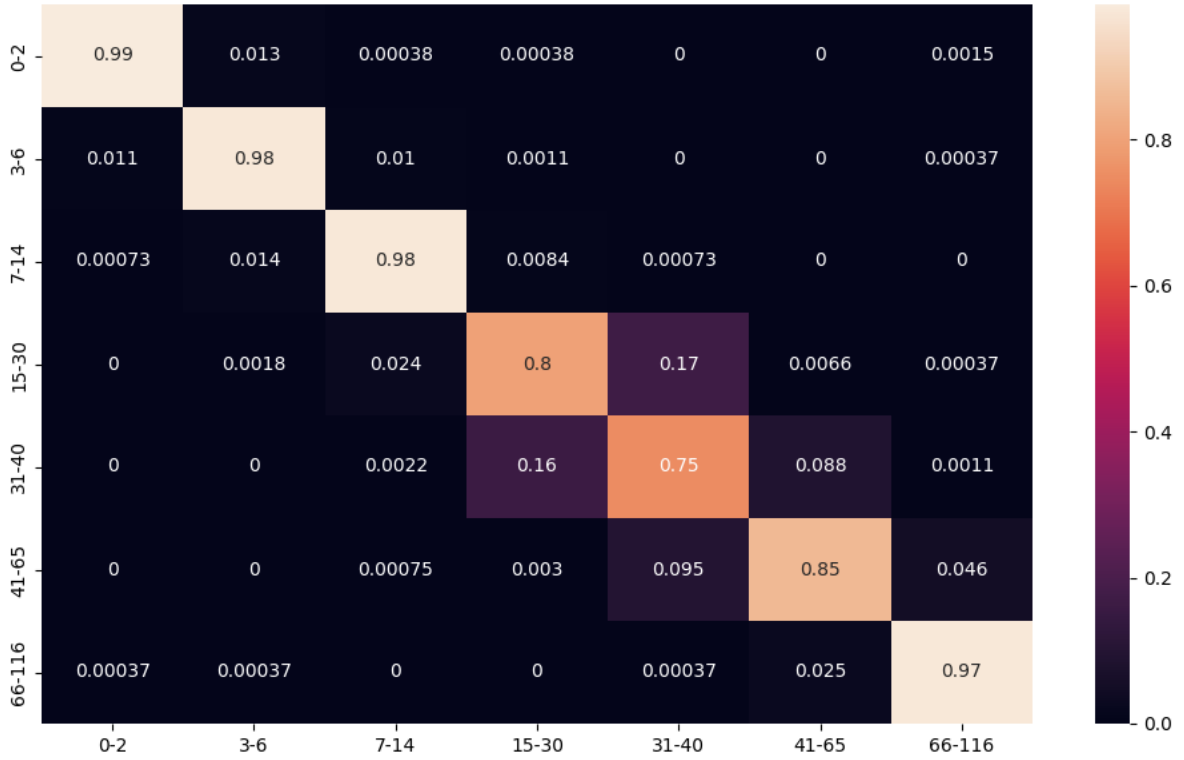

Figure 103 optimal model age confusion matrix

6 Analysis and discussion

In this section, the above results will be analysed and discussed to state what choices were made, and more importantly, why they were made. I will refer to one of the models as V## that means variant number ## where the number is the same as the subheading number in the previous section. For example, the base model is V01 while the last optimal model is V20. The analysis will consist of six parts. The first one compares the previously mentioned 20 models and states what's been learnt from them. The second part touches on why more advanced learning techniques like multitasking and residual attention were not the optimal models. The third part discusses different age groups and how the optimal age group was selected. The fourth part goes into detail about the data leakage in V14 and why data separation from code is crucial. The fifth part analyses and discusses the optimal model and insights on what is the reason of these results. It will investigate the relationship between gender and ethnicity with age, and the choice of these specific age categories. Finally, the last part will compare this project's results with other academic papers who have taken on the same task.

6.1 Previous 20 variants

I have listed the variants in the previous section in a way that can be easily logically followed for readers to draw their own analysis and thoughts. Starting with V01 with the model architecture, weighted loss, and other building blocks, it gave a very first base line to set expectations. Then the second variant proved that sticking to one dataset and decreasing data variance helped the model avoid some confusion. After that, more complex models were tried like DenseNet and ResNet to check with the traditional CNN was too simple. However, given the results it was very clear that the task at hand does not require these complex models and performs better with simpler models. This becomes even clearer when the residual attention is tried in V18.

Then a few more techniques were added to counter the data imbalance and make the model more generalizable. Namely the data augmentation and weighted sampler that exposed the model more frequently to the minority class and ensured the data is always slightly different thanks to the data augmentations. For generalization, several regularization techniques were employed. Including dropouts, L2 regularization in the optimizer, and batch normalization. Then, trying different optimisers lead to choosing NAdam. Since Adam performed very well, it makes sense that using Nesterov's momentum resulted in even a better performance.

6.2 Why not the more advanced learning techniques

To ensure that the traditional CNN model was not too simple, and the project was not trivial, a lot of effort was put into trying other learning techniques, namely hierarchical, transfer, residual attention, and

multitask learning. Even though none of them performed better than the 4-layered CNN, it can confidently be confirmed that they have been studied, implemented, and tested. A hypothesis why these techniques did not work goes as follows: For the hierarchical, the choice of age groups eliminated the variance caused by gender and ethnicity. For transfer learning, since the models were pretrained on different tasks, and the current dataset has enough data, it was always better to learn from scratch. For residual attention, it was shown that even the traditional ResNet was too much, so it makes sense that residual attention does the same, especially since the images were already cropped and aligned to the faces, so there was not much the attention mechanism could do. Finally, for multitasking, while it is definitely much faster to train a single model for three tasks, and it did help with decreasing overfitting, it is predictable that a model focusing on three tasks at once would behave worse that a model focusing on a single task.

6.3 Selecting suitable age groups

It was apparent that different age groups result in completely different results. That was tricky to determine since many papers do not state how many ages groups, they used so some results might be inflated. The process of picking the bins took a lot of trial and error as well. While the F1 scores were taken into account, on their own, they are not an enough indicator of which age groups were harder to detect. That is why the confusion matrix was very helpful here. The way this problem was tackled was first, the model attempted to classify ages in groups of 2 (1-2,3-4,5-6, …, 61-116) The purpose of this is to see which classes are classified by the model as which groups as shown in Figure 104.The lighter the square, the better the model is detecting that class. For example it is already visible that the model is very good at detecting very young ages. And it makes sense since they are very clear.

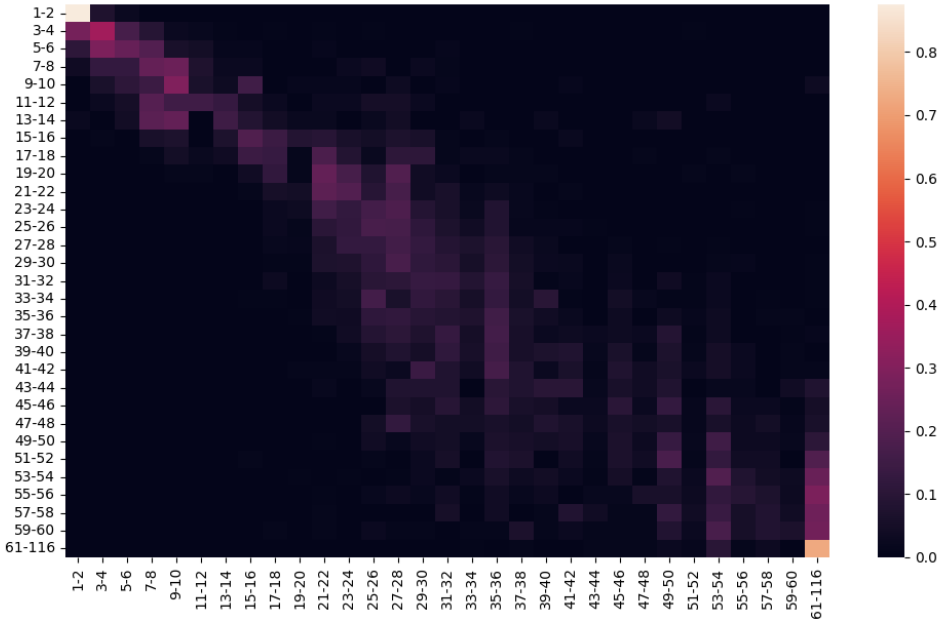

Figure 104 Confusion matrix of age classes in groups of 2

Then, a very interesting analysis was made where the confusion matrix was used to visually determine the best age groups as shown in Figure 105. The goal was to create squares that have the highest ratio of light to dark squares, since that would indicate the best age grouping. Finally, after having a very good idea on how to assign groups, it was determined to go with 7 bins since it is not too few for the model to be useful, or too many that the model gets confused. With more trial and error around these groups, the final result was ['0-2', '3-6', '7-14', '15-30', '31-40', '41-65', '66-116'] and it made a huge difference towards the optimal model.

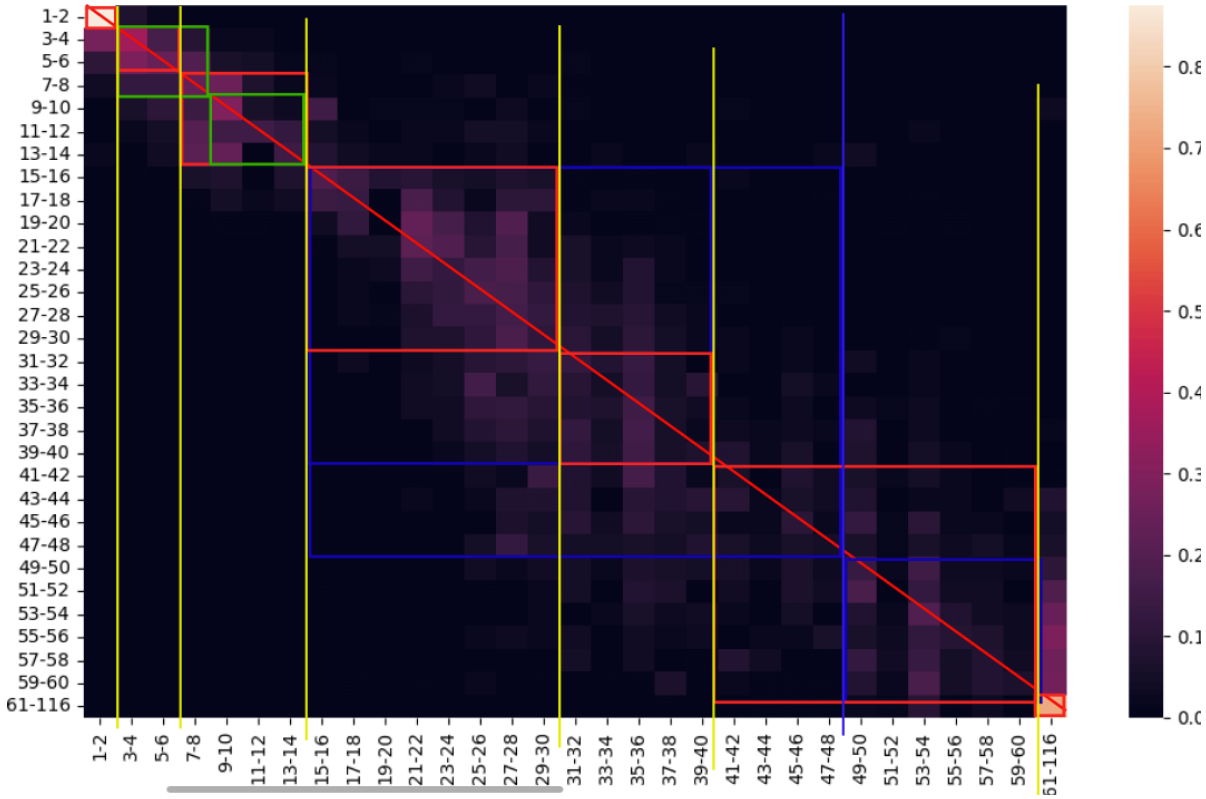

Figure 105 Visually determine best age groups from confusion matrix.

6.4 Data leakage - most valuable lesson I have learned.

I have dedicated a subsection for this matter because of how important it is, and because it was one of the best lessons, I have learned working on this project, and I will never forget it. As seen in V14, the results went up hugely. From 70ish to 90%. At first, I thought the hierarchy I have added was ground-breaking and that I cracked the code. As proud and ecstatic as I was, I have learnt to doubt my results over the years, especially if they are too good to be true, and it's always better safe than sorry. So, I looked very deep into the code, followed it line by line, everything was going very well, until I saw it. The data leakage was there. It was in a very tiny line, inside a function, inside a function, inside a function. If I haven't had absorbed every line of code while working on the project, I would have missed

it, so would anyone who is seeing my code for the first time. But it was there, and it was inflating my results. The problem was as follows:

The way I have implemented hierarchy was that I first trained the age model on the whole dataset. Once I had that, I saved its weights. Then I started my hierarchy. I first trained a gender classification model on my data. After that, I used it to predict the gender of every sample in the data and save the predicted gender along with the original data. Finally, I split my data into a male and female dataset. Then passed each one into an age model that is pretrained with the weights from the very first step. So far so good. What I did not notice, however, is the fact that my training/test split was not consistent. That means that the way the data was split into training and testing in the first age detection model is not the same as the one used in the hierarchical age detection models. By now you have already guessed. Some of the training data in the first age model got in as validation data in the second model. That explained why the model was very good at classifying them the second time, it is because it saw those samples before. So, to remedy the situation, I did what I have should've done since the beginning, I split the data from the code. Before even training the models, I split the data into 2 different csv files, and then every time I train, I read the data from the csv files. This ensures that no matter how many models I train, the training data is always used for, and only for, training. While the results have decreased a lot, which is very logical, I am still glad I saw through the leakage, I am glad I was able to fix it, I am glad my results are now much more reliable, and most of all, I am glad I have learned something new.

6.5 Final optimal model

Now that all the results were discussed separately and together, it is time to discuss the final model on its own, to try to understand the results, limitations and what can be done to result in better classification performance. This part will consist of three mini-parts, one for each task of the AGE classification project.

6.5.1 Age

It is important to remember that age is the primary goal of this project, and as a result the model was mostly optimized for age, also age is the trickiest because there are so many different ways to split into age categories, and thus it is very challenging. I will not discuss splitting into age categories further since I have discussed it in detail earlier in this section. What I would like to discuss however is the classification of the current age groups. The confusion matrix of the optimal model for age classification is shown in Figure 106. It can be noted that the model is very good at detecting the extreme ages (0-14 and 66-116). However, it is not as good in the middle groups, and that makes sense, since it can sometimes be hard even for humans to detect those ages accurately.

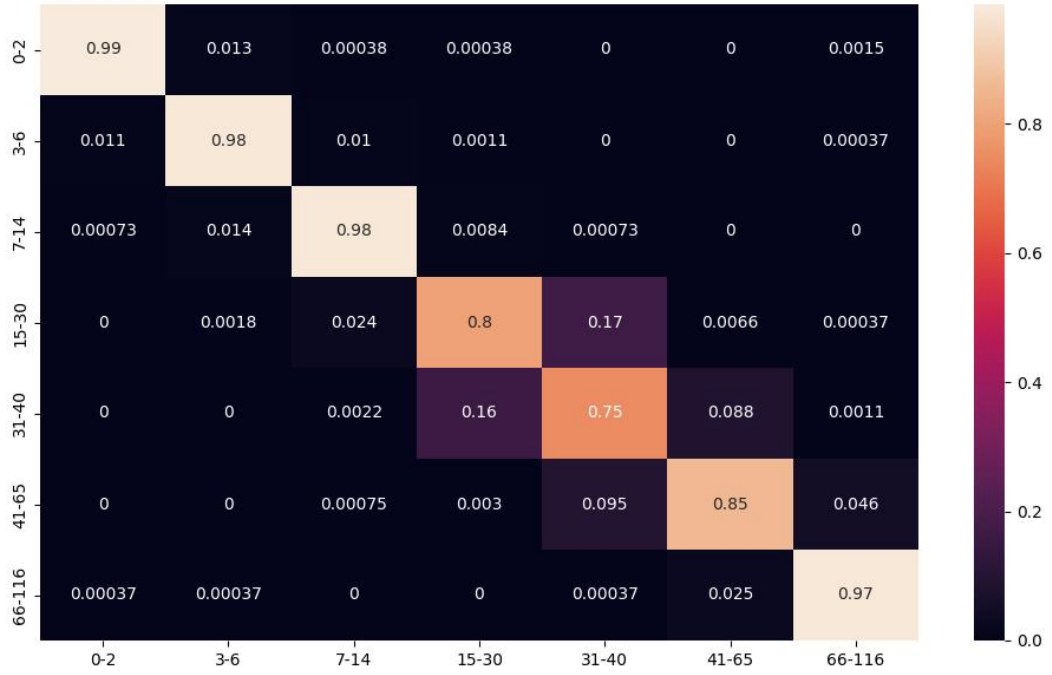

Figure 106 Age confusion matrix for optimal model

It is visible specifically that the model confuses 15-30 as 31-40 and vice versa. So, let us look further into those two groups. First, it is predictable that the model would face issues with these two groups from Figure 104 since there was no way to draw squares that cover a lot of light squares without grouping in a lot of dark squares as well. Now, if we look further into these two groups, the data shown in Figure 107 indicates the mode and mean of the samples in these two groups, and how close they are to one another. So that explains the fact that they are causing the most confusion.

age_cat	mean	mode
0-2	1.300312	1
3-6	4.190101	3
7-14	10.075194	8
15-30	24.855938	26
31-40	35.379668	35
41-65	52.474536	45
66-116	77.551501	85

Figure 107 Age categories mean and mode.

More analysis into why age is the least predictable of the tasks is the high variance in gender and ethnicity as shown previously in Figure 13 and Figure 14. In the three middle classes for example, it is very clear that they have a much higher variance of ethnicity compared to the very young and very old as shown in Figure 108

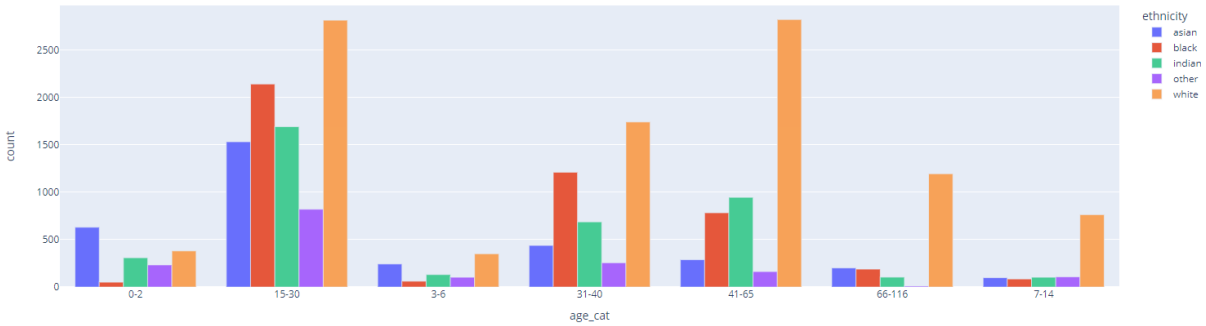

Figure 108 Ethnicity distribution grouped by age categories.

6.5.2 Gender

As for the gender, it is already at 93% so there is not much room of improvement there. Not only gender is notoriously easier to classify, but also the data is almost balanced around gender ad indicated earlier, resulting in the beautiful confusion matrix shown in

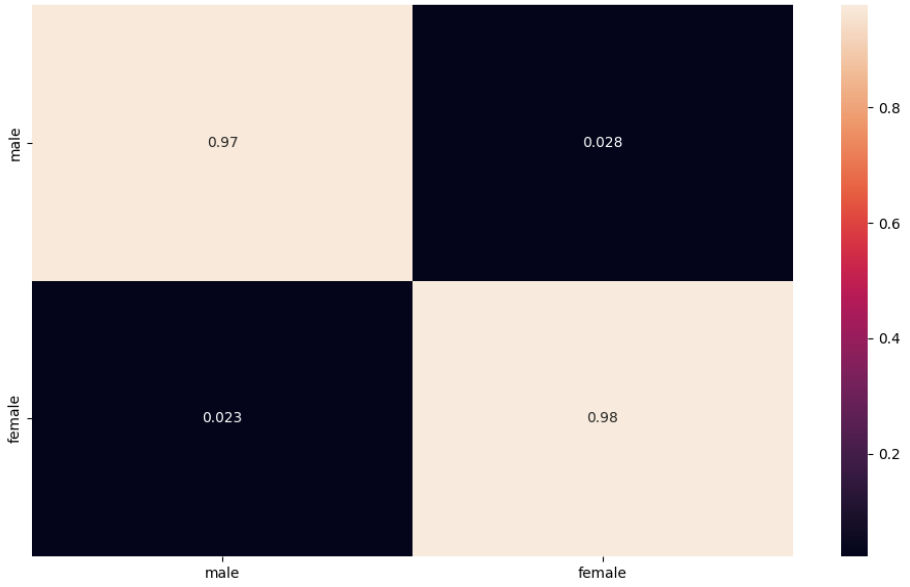

Figure 109 Gender confusion matrix for optimal model

6.5.3 Ethnicity

Finally, while the ethnicity is higher than the age, at around 79%, there is more potential to improve it. As mentioned before, this project heavily focused on age, and thus the model can certainly be optimized for ethnicity. One example would be to further balance the data since ethnicity is awfully unbalanced compared to the other tasks. As can be seen in Figure 110, most of the 'other' ethnicity is predicted as white and Indian. Making this task the only one where accuracy is much higher than F1 score, residing at 85%, which is 5% higher than the F1 score.

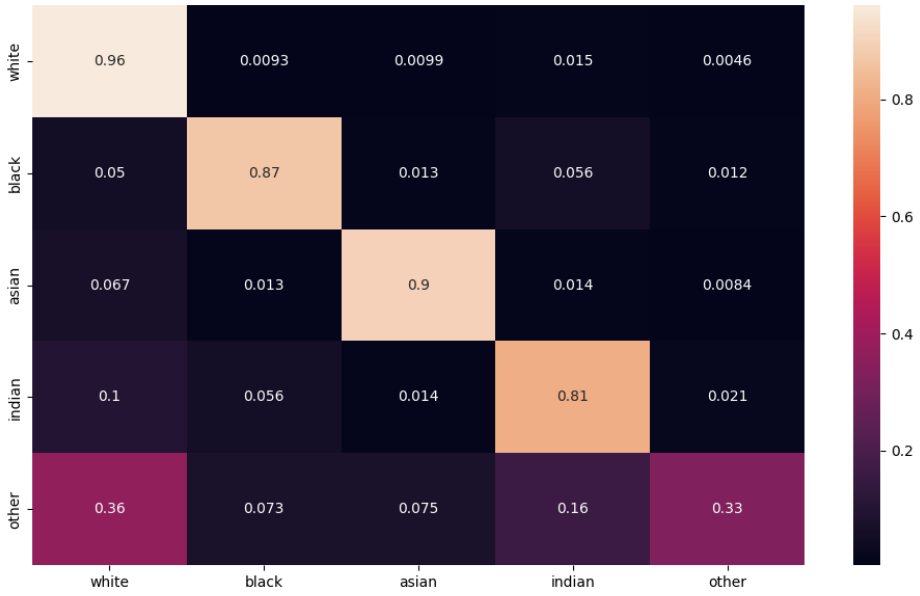

Figure 110 Ethnicity confusion matrix for optimal model

6.6 Comparison with previous work

Finally, in this subsection we compare the results to previous work. Again, it is important to note that the project focuses mainly on age, and that is the one that will have the most comparisons and exceeding previous work. However, the gender and ethnicity are still competitive, even though they were not the primary focus of the project and not as much time went into optimizing them. As before, I will refer to one of the models as V## that means variant number ## where the number is the same as the subheading number in the "Trials and Results" section. Note that they all are using accuracy, while I am using F1 score, which is more telling for classification, especially with unbalanced data. For the ethnicity, to convert from F1 to accuracy, add about 5% to my results, so that it is comparable to other papers.

6.6.1 Age

Model	Metric	Value
Hierarchy VGG16 (only 4 bins) [8]	Accuracy	80.76%
Senet50 [63]	Accuracy	61.96%
Facenet [23]	Accuracy	56.9%
Finetune Facenet (FFNet) [23]	Accuracy	64.00%
MTCNN [23]	Accuracy	70.1%
ResNet50 [63]	Accuracy	64.00%
VGG16(DEX) [64]	Accuracy	67.30%
xAlexNet + GoogLeNet + LeNet [65]	Accuracy	69.59%
My Proposed Hierarchical model V15	F1 score	72.1%
My Proposed Residual Attention V18	F1 score	69.95%
My Proposed optimal model V20	F1 score	**73.27%**

Table 3 Age classification comparison with previous work.

6.6.2 Gender

Model	Metric	Value
VGG16 [8]	Accuracy	90.04%
VGGFace [66]	Accuracy	91.60%
MobileNet [67]	Accuracy	91.95%
MTCNN [23]	Accuracy	**98.23%**
StarGAN [70]	Accuracy	91.70
My Proposed Multitasking model V19	F1 score	93.03%
My Proposed optimal model V20	F1 score	93.41%

Table 4 Gender classification comparison with previous work.

6.6.3 Ethnicity

Model	Metric	Value
VGG16 [68]	Accuracy	80.00%
VGGFace [68]	Accuracy	79.00%
ResNet-34 [69]	Accuracy	83.50%
VGG-Face [69]	Accuracy	**89.30%**
StarGAN [70]	Accuracy	87.20
My Proposed Multitasking model V19	F1 score	74.35%
My Proposed optimal model V20	F1 score	80.25%

Table 5 Ethnicity classification comparison with previous work.

7 Conclusions and Future Work

7.1 Conclusion

In conclusion, this project focused on human profiling, specifically AGE (age, gender, and ethnicity) estimation, due to its practical applications in various domains such as security, marketing, and health. The research aimed to investigate state-of-the-art age classification models and explore the impact of other soft biometrics, such as gender and ethnicity, on age detection. The project identified a gap in optimizing age classification models for high-variance datasets, like UTKFace, with realistic age bins. To address this gap, the project followed a systematic approach, beginning with a comprehensive literature review of previous work in the field. A base model was then developed, and realistic age bins were empirically determined. Several model architectures and learning methods were experimented with, including transfer learning, multitasking, and hierarchical learning. Ultimately, optimized classification models were developed to classify a person's age, gender, and ethnicity. Moreover, a dangerous case of data leak was encountered, detected, and handled appropriately.

The results of the project were grand. AGECNN achieved F1 scores of 73.27% for age, 93.41% for gender, and 80.25% for ethnicity. While the model demonstrated competitive gender classification performance comparable to state-of-the-art methods, further optimization is needed for ethnicity classification, particularly in terms of data balancing and accurately classifying the "other" ethnicity label. Most notably, the age classification results surpassed those of existing state-of-the-art models that utilized the same age categories. Not only, the model achieved state-of-the-art results, but it also deeply analysed the results, explaining their weak points and suggestions to overcome them. This accomplishment highlights the project's success in filling the identified gap and improving age classification accuracy while also maintaining competitive gender and ethnicity classification. Moving forward, future research can focus on refining ethnicity classification and exploring additional techniques to enhance age classification further.

7.2 Contributions / what I learnt.

- AGECNN model achieved state-of-the-art results in age classification using this set of age bins, while still maintaining competitive results in gender and ethnicity classification
- Selected optimized age category bins based on both trial and error, and on visualizations such as the confusion matrix.
- Experimented with several model architectures and learning techniques, all of which are compared and reproducible using the linked github code, including:
    - Residual attention

    - Transfer learning with FaceNet
    - Multitasking
    - Hierarchical models
- Detected a data leak while optimizing the models and handled it by separating the data from the code logic.
- Deep analysis that explains the current results state and discusses limitations and future work

7.3 Future Work

As mentioned in the Analysis and Discussion section, the model was mainly optimized on age, so that leaves a lot of improvement space for the other two tasks, especially ethnicity detection since the data is highly imbalanced. One recommendation would be to enforce a threshold on predicting the 'white' and 'black' labels, with anything below the threshold to be classified as 'other' which might show great increase in performance.

Moreover, access was unavailable to MORPHII dataset due to its expensive license, a great dataset with age, gender, and ethnicity labels like UTKFace. With access to that dataset, oversampling can be achieved to specially fill for the imbalanced age and ethnicity labels. In addition, the trained models can be tested on MORPHII to determine their generalization on a completely different dataset.

Furthermore, using ensemble between the optimized model and residual attention or multitasking is an area worth exploring. Another addition to the project can be visualizing the feature maps of the different models, especially the optimized model and the residual attention models, to compare what each model focuses on when looking at an image since I was limited by the computational power. Finally, some sort of face detection can be fine-tuned on the data to ensure that all image faces are aligned, especially in deployment.

References

[1] A. Dantcheva, P. Elia and A. Ross, "What Else Does Your Biometric Data Reveal? A Survey on Soft Biometrics," in IEEE Transactions on Information Forensics and Security, vol. 11, no. 3, pp. 441-467, March 2016, doi: 10.1109/TIFS.2015.2480381.

[2] X. Geng, Z. H. Zhou, και K. Smith—Miles, 'Automatic age estimation based on facial ageing patterns (vol 29, pg 2234, 2007)', Ieee Transactions on Pattern Analysis and Machine Intelligence, τ. 30, τχ. 2, σσ. 368–368, 2008.

[3] E. Agustsson, R. Timofte, S. Escalera, X. Baro, I. Guyon and R. Rothe, "Apparent and Real Age Estimation in Still Images with Deep Residual Regressors on Appa-Real Database," 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), 2017, pp. 87-94, doi: 10.1109/FG.2017.20.

[4] G. Guo and X. Wang, "A study on human age estimation under facial expression changes," 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 2547-2553, doi: 10.1109/CVPR.2012.6247972.

[5] D. G. Ganakwar and V. K. Kadam, "Comparative Analysis of Various Face Detection Methods," 2019 IEEE Pune Section International Conference (PuneCon), 2019, pp. 1-4, doi: 10.1109/PuneCon46936.2019.9105893.

[6] Í. d. P. Oliveira, J. L. P. Medeiros, V. F. d. Sousa, A. Gomes Teixeira Junior, E. Torres Pereira and H. Martins Gomes, "A Data Augmentation Methodology to Improve Age Estimation Using Convolutional Neural Networks," 2016 29th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), 2016, pp. 88-95, doi: 10.1109/SIBGRAPI.2016.021.

[7] N. L. Ajit Krisshna, V. K. Deepak, K. Manikantan, και S. Ramachandran, 'Face recognition using transform domain feature extraction and PSO-based feature selection', Applied Soft Computing, τ. 22, σσ. 141–161, 2014.

[8] D. Yi, Z. Lei, και S. Li, 'Age Estimation by Multi-scale Convolutional Network', 04 2015, τ. 9005, σσ. 144–158.

[9] Raman, K. Elkarazle, and P. Then, "Gender-specific Facial Age Group Classification Using Deep Learning," Intelligent Automation and Soft Computing, vol. 34, pp. 106–118, Apr. 2022, doi: 10.32604/iasc.2022.025608.

[10] P. Rodríguez, G. Cucurull, J. M. Gonfaus, F. X. Roca, and J. Gonzàlez, "Age and gender recognition in the wild with deep attention," Pattern Recognition, vol. 72, pp. 563–571, 2017.

[11] S. Zaghbani, N. Boujneh, M. S. Bouhlel, 'Age estimation using deep learning', Computers & Electrical Engineering, 68. 337–347, 2018.

[12] P. Smith, C. Chen, 'Transfer Learning with Deep CNNs for Gender Recognition and Age Estimation', CoRR. abs/1811.07344, 2018. - 16 -

[13] S. Taheri, Ö. Toygar, ‘On the use of DAG-CNN architecture for age estimation with multi-stage features fusion’, Neurocomputing, 329,. 300–310, 2019.

[14] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[15] K. Simonyan και A. Zisserman, ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’, CoRR, τ. abs/1409.1556, 2015.

[16] S. Chen, C. Zhang, M. Dong, and J. Le, “Using Ranking CNN for Age Estimation,” Jan. 2017. doi: 10.1109/CVPR.2017.86.

[17] B. I. Yoo, Y. Kwak, Y. Kim, C. Choi, and J. Kim, “Deep facial age estimation using conditional multitask learning with weak label expansion,” IEEE Signal Processing Letters, vol. 25, no. 6, pp. 808–812, 2018.

[18] S. Taheri O. Toygar, ‘Multi-Stage Age Estimation Using Two Level Fusions of Handcrafted and Learned Features on Facial Images’, IET Biometrics, τ. 8, 03 2019.

[19] A. González-Briones, G. Villarrubia, J. F. De Paz, and J. M. Corchado, “A multi-agent system for the classification of gender and age from images,” Computer Vision and Image Understanding, vol. 172, pp. 98–106, 2018.

[20] G. Villarrubia, J. F. De Paz, F. De La Prieta and J. Bajo, "Hybrid indoor location system for museum tourist routes in augmented reality," 17th International Conference on Information Fusion (FUSION), 2014, pp. 1-8.

[21] D. Delgado-Gomez, J. Fagertun, B. Ersbøll, F. M. Sukno, and A. F. Frangi, “Similarity-based fisherfaces,” Pattern Recognition Letters, vol. 30, no. 12, pp. 1110–1116, 2009.

[22] P. Mazurek, T. Hachaj, ‘Robustness of Haar Feature-Based Cascade Classifier for Face Detection Under Presence of Image Distortions’, 01 2020. 14–21.

[23] A. Das and A. Dantcheva, “Mitigating Bias in Gender, Age, and Ethnicity Classification: a Multi-Task Convolution Neural Network Approach,” Oct. 2018.

[24] Z. Zhifei, S. Yang, and Q. Hairong, “Age progression/regression by conditional adversarial autoencoder,” in IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2017

[25] V. Andreieva and N. Shvai, “Generalization of Cross-Entropy Loss Function for Image Classification,” Mohyla Mathematical Journal, vol. 3, pp. 3–10, Jan. 2021, doi: 10.18523/2617-7080320203-10.

[26] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” Journal of big data, vol. 6, no. 1, pp. 1–48, 2019.

[27] K. J. Friston, J. Ashburner, C. D. Frith, J.-B. Poline, J. D. Heather, and R. S. Frackowiak, “Spatial registration and normalization of images,” Human brain mapping, vol. 3, no. 3, pp. 165–189, 1995.

[28] H. M. Bui, M. Lech, E. Cheng, K. Neville, and I. S. Burnett, “Using grayscale images for object recognition with convolutional-recursive neural network,” in 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE), 2016, pp. 321–325.

[29] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in 2017 international conference on engineering and technology (ICET), 2017, pp. 1–6.

[30] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in Proceedings of the AAAI conference on artificial intelligence, 2017, vol. 31, no. 1.

[33] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[34] M. Tan and Q. Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 2019.

[35] S. Amari, “Backpropagation and stochastic gradient descent method,” Neurocomputing, vol. 5, no. 4–5, pp. 185–196, 1993.

[36] Y. Liu, Y. Gao, and W. Yin, “An improved analysis of stochastic gradient descent with momentum,” Advances in Neural Information Processing Systems, vol. 33, pp. 18261–18271, 2020.

[37] D. Xu, S. Zhang, H. Zhang, and D. P. Mandic, “Convergence of the RMSProp deep learning method with penalty for nonconvex optimization,” Neural Networks, vol. 139, pp. 17–23, 2021.

[38] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” International Conference on Learning Representations, Dec. 2014.

[39] T. Dozat, “Incorporating Nesterov Momentum into Adam,” 2016.

[40] J. Cusido, J. Comalrena, H. Alavi, and L. Llunas, “Predicting Hospital Admissions to Reduce Crowding in the Emergency Departments,” Applied Sciences, vol. 12, p. 10764, Oct. 2022, doi: 10.3390/app122110764.

[41] A. F. Agarap, “Deep learning using rectified linear units (relu),” arXiv preprint arXiv:1803.08375, 2018.

[42] J. Feng, X. He, Q. Teng, C. Ren, H. Chen, and Y. Li, “Reconstruction of porous media from extremely limited information using conditional generative adversarial networks,” Physical Review E, vol. 100, Sep. 2019, doi: 10.1103/PhysRevE.100.033308.

[43] J. Bridle, “Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters,” Advances in neural information processing systems, vol. 2, 1989.

[44] A. Lewkowycz, “How to decay your learning rate,” arXiv preprint arXiv:2103.12682, 2021.

[45] L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” in Artificial intelligence and machine learning for multi-domain operations applications, 2019, vol. 11006, pp. 369–386.

[46] A. Meiliana, N. Dewi, and A. Wijaya, “Car T Cells: Precision Cancer Immunotherapy,” The Indonesian Biomedical Journal, vol. 10, pp. 203–16, Dec. 2018, doi: 10.18585/inabj.v10i3.635.

[47] A. Agrawal and N. Mittal, “Using CNN for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy,” The Visual Computer, vol. 36, no. 2, pp. 405–412, 2020.

[48] Hughes, C. (2022) Demystifying pytorch’s Weightedrandomsampler by example, Medium. Available at: https://towardsdatascience.com/demystifying-pytorchs-weightedrandomsampler-by-example-a68aceccb452 (Accessed: 10 June 2023).

[49] M. Ren, W. Zeng, B. Yang, and R. Urtasun, “Learning to reweight examples for robust deep learning,” in International conference on machine learning, 2018, pp. 4334–4343.

[50]]I. Goodfellow, Y. Bengio, and A. Courville, “Regularization for deep learning,” Deep learning, pp. 216–261, 2016.

[51] T. Van Laarhoven, “L2 regularization versus batch and weight normalization,” arXiv preprint arXiv:1706.05350, 2017.

[52] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” The journal of machine learning research, vol. 15, no. 1, pp. 1929–1958, 2014.

[53] L. Prechelt, “Early stopping-but when?,” in Neural Networks: Tricks of the trade, Springer, 2002, pp. 55–69.

[54] R. Müller, S. Kornblith, and G. E. Hinton, “When does label smoothing help?,” Advances in neural information processing systems, vol. 32, 2019.

[55] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in International conference on machine learning, 2015, pp. 448–456.

[56] M. Iman, H. Arabnia, and K. Rasheed, “A Review of Deep Transfer Learning and Recent Advancements,” Technologies, vol. 11, p. 40, Mar. 2023, doi: 10.3390/technologies11020040.

[57] J. Deng, W. Dong, R. Socher, L. -J. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.

[58] D. Yi, Z. Lei, S. Liao, and S. Z. Li, “Learning face representation from scratch,” arXiv preprint arXiv:1411.7923, 2014.

[59] F. Wang et al., “Residual attention network for image classification,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3156–3164.

[60] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba, “Learning with hierarchical-deep models,” IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1958–1971, 2012.

[61] R. Caruana, “Multitask learning,” Machine learning, vol. 28, pp. 41–75, 1997.

[62] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, “Averaging weights leads to wider optima and better generalization,” arXiv preprint arXiv:1803.05407, 2018.

[63] S. M. Uddin, M. Morshed, M. Prottoy, and A. B. M. A. Rahman, “Age Estimation from Facial Images using Transfer Learning and K-fold Cross-Validation,” Jul. 2021, pp. 33–36. doi: 10.1145/3480651.3480659.

[64] R. Rothe, R. Timofte, and L. Van Gool, “Deep expectation of real and apparent age from a single image without facial landmarks,” International Journal of Computer Vision, vol. 126, no. 2, pp. 144–157, 2018.

[65] C.-J. Lin, C.-H. Lin, C.-C. Sun, and S.-H. Wang, “Evolutionary-fuzzyintegral-based convolutional neural networks for facial image classification,” Electronics, vol. 8, no. 9, p. 997, 2019.

[66] A. Krishnan, A. Almadan and A. Rattani, "Understanding Fairness of Gender Classification Algorithms Across Gender-Race Groups," 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 2020, pp. 1028-1035, doi: 10.1109/ICMLA51294.2020.00167.

[67] A. Savchenko, “Efficient facial representations for age, gender and identity recognition in organizing photo albums using multi-output ConvNet,” PeerJ Computer Science, vol. 5, p. e197, Jun. 2019, doi: 10.7717/peerj-cs.197.

[68] M. A. Ahmed, R. D. Choudhury, and K. Kashyap, “Race estimation with deep networks,” Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 7, pp. 4579–4591, 2022, doi: https://doi.org/10.1016/j.jksuci.2020.11.029

[69] Greco, A., Percannella, G., Vento, M. et al. Benchmarking deep network architectures for ethnicity recognition using a new large face dataset. Machine Vision and Applications 31, 67 (2020). https://doi.org/10.1007/s00138-020-01123-z

[70] A. Deviyani, Assessing Dataset Bias in Computer Vision. 2022.

Appendix I

Examples of model results

- **Age**

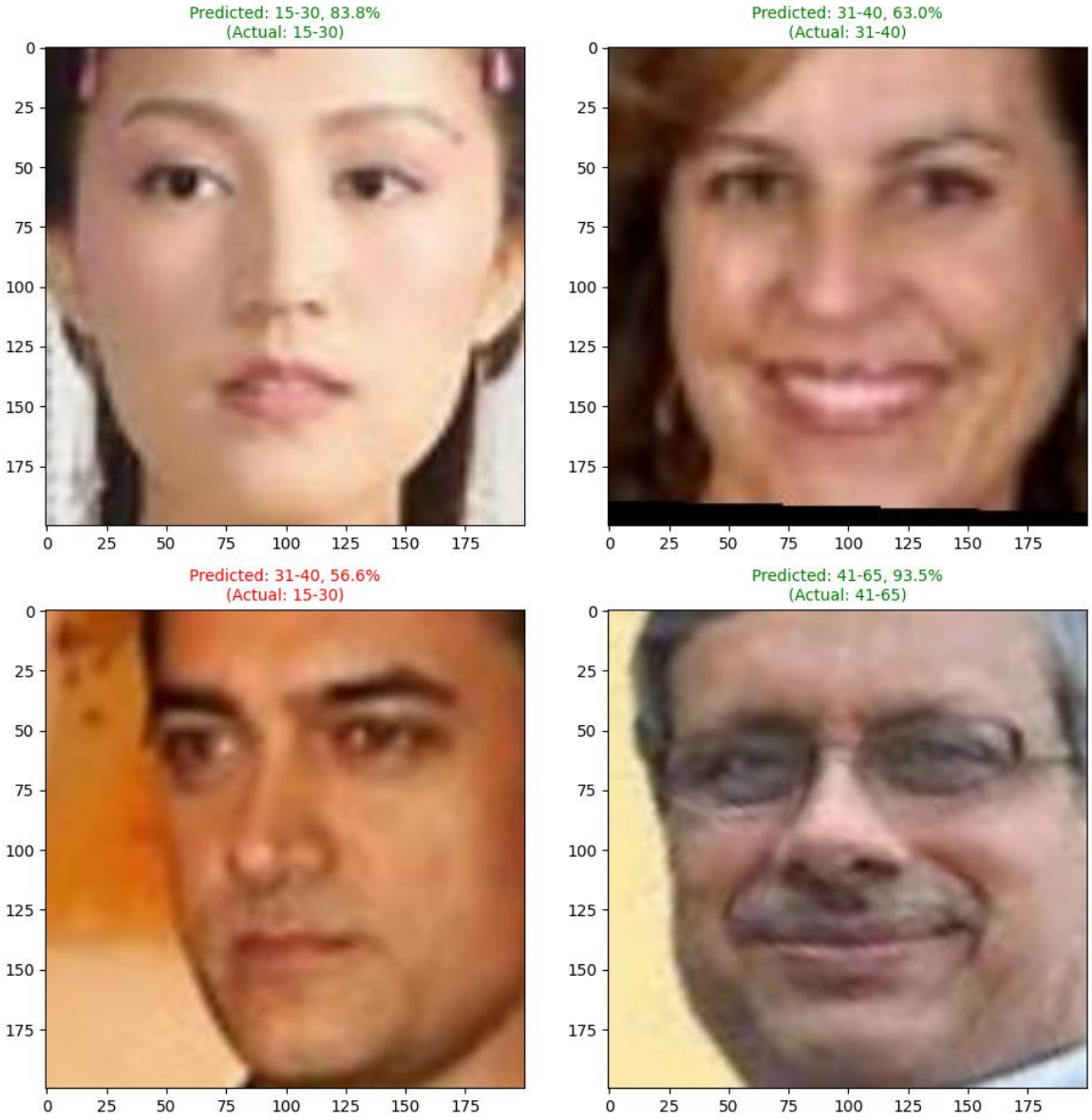

Figure 111 Model age classification

- **Gender**

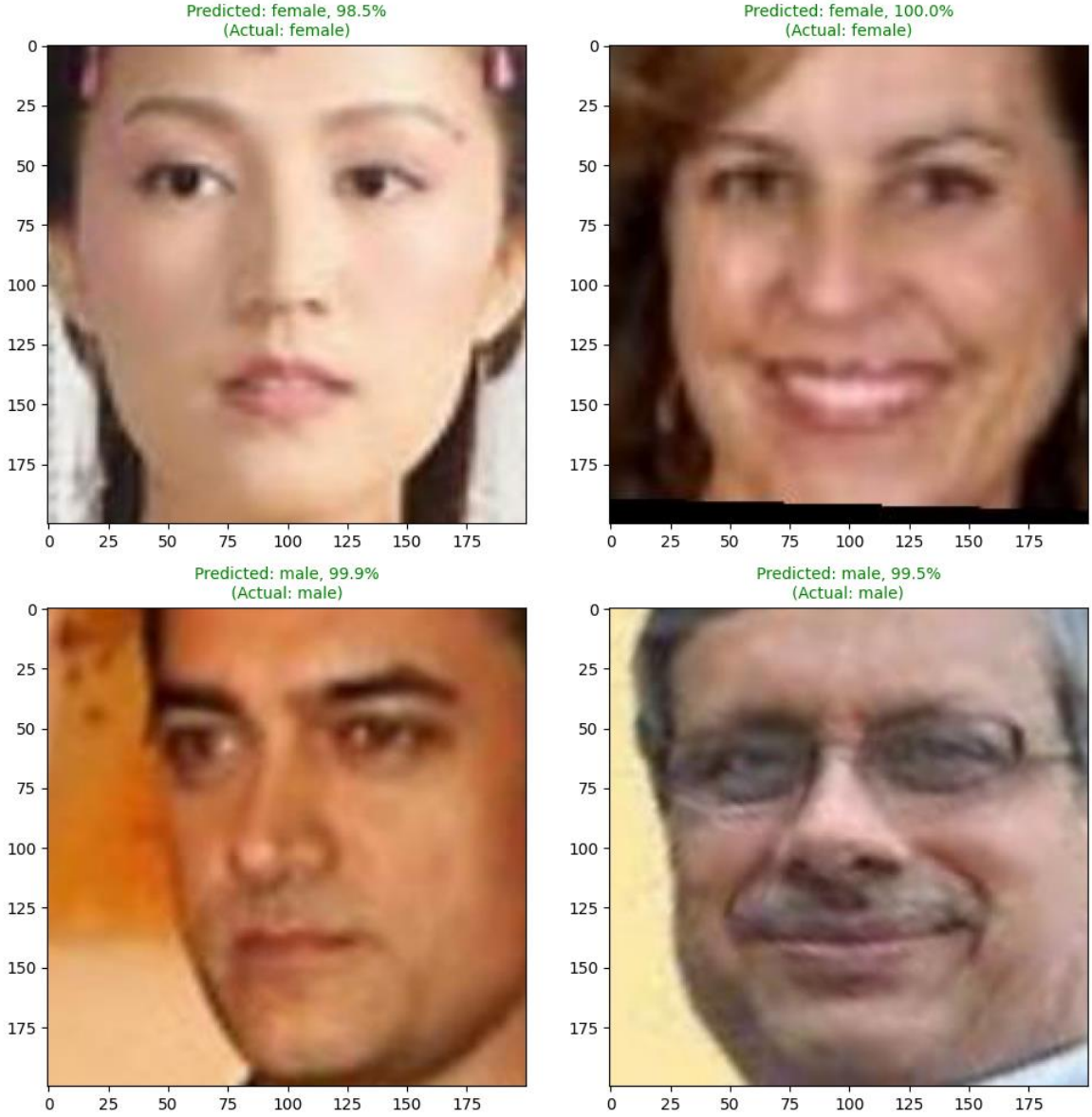

Figure 112 Model gender classification

- **Ethnicity**

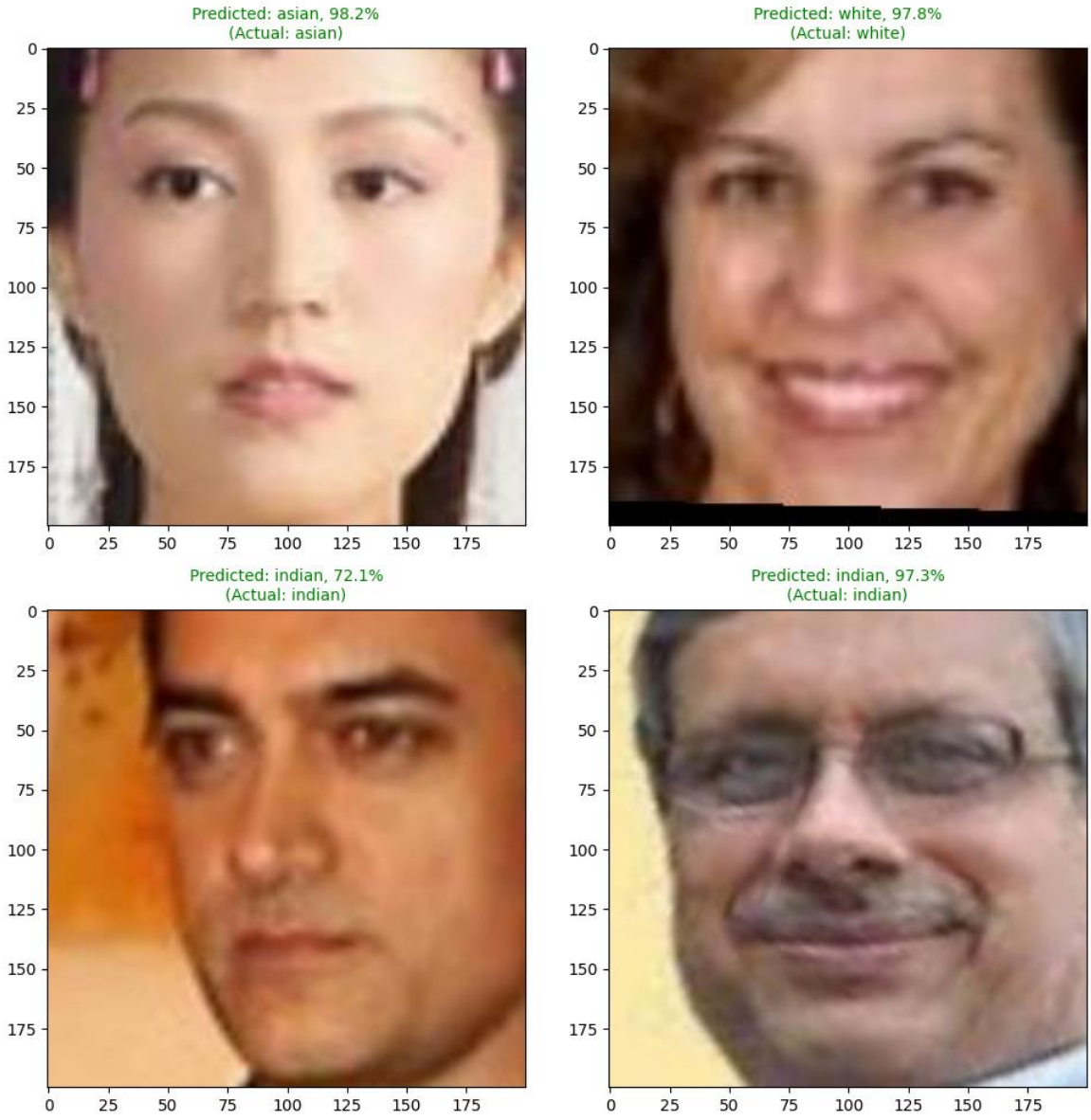

Figure 113 Model ethnicity classification