Evaluating deep learning approaches for perceived age prediction

Callum Parton

170173276

BSc Computer Science

Supervisor: Dr Jaume Bacardit

Word Count: 14999

Abstract

With the rapid pace of advancement in deep learning research, a growing domain of problems are being solved, which would not be possible with traditional programming techniques. However, the robustness and accuracy of deep learning models is often intrinsically linked to the quality of available datasets that the models are trained on. This makes it difficult to build highly accurate and reliable deep learning models for niche fine-grained problems such as perceived age prediction. This dissertation project investigates and explores perceived age prediction, to determine which components in the deep learning pipeline can improve performance for this task when limited with a small, albeit clean target dataset. The explored pipeline components were transfer learning, data pre-processing, learning strategy, and hyperparameter adjustment.

Declaration

“I declare that this dissertation represents my own work, except where otherwise stated.”

Acknowledgements

I would like to thank my supervisor, Dr Jaume Bacardit, for his advice and support throughout this project.

Table of Contents

1 Introduction

1.1 Technological Context

Artificial Intelligence (AI) is the science of creating intelligent machines and computer programs to achieve complex goals typically reserved for human-level intelligence [1]. AI is a broad area, which overlaps and unites multiple disciplines, most notably computer science and mathematics/statistics. The concept of Artificial General Intelligence, which would involve computers performing at least on par with humans across a vast range of tasks, including artistic endeavors, is a fascinating area of research [2]. However, this is proposed to be at least decades away, and AI is currently limited to a much narrower set of capabilities [3]. Within AI, there is a popular subtopic of Machine Learning (ML). ML techniques allow computer programs to learn and improve at a given task from experience and data, forgoing the need to be explicitly programmed with what decisions to make [4]. Traditional ML algorithms use pre-selected 'features' to influence predictions for the given task [5]. A feature is simply a problem-specific variable. For instance, in predicting house price, a feature may be the location expressed in a real-valued longitude/latitude tuple.

Deep learning is currently a popular approach to ML, and it is biologically inspired, involving computations performed using Artificial Neural Networks (ANN). An ANN, consisting of many layers of neurons, is loosely inspired by the human brain. Each of these neurons processes data and transmits a signal downstream to connected neurons, gradually 'learning' something precise about the input data. Learning is represented through the update of neuron weights after each training cycle. This approach allows precise features to be engineered from complex high-dimensional data like images through a series of nonlinear transformations in several 'hidden' layers, differing from traditional ML [5]. The stacking of many of these hidden layers and mathematical transformations is unique to deep learning, as the top layers distil precise higher-order knowledge representations. The concept of ANNs has existed for a long time, but the *deep* aspect of learning is a relatively new area. In this context, *deep* refers to the number of layers in the network, with deep networks simply having many layers between the input and output, and therefore many weights [6].

Deep Learning has become much more accessible in recent times as the power of Graphical Processing Units (GPUs) has increased, and GPU vendors such as Nvidia have invested in providing a software platform for developers. Deep learning has applications across a myriad of domains and industries, from search engine improvement to medical image analysis. Many major companies use deep learning to improve and diversify their services, most notably Google with their DeepMind laboratories and the various algorithms used across their product family (e.g., Google Assistant). Deep learning models such as Convolutional Neural Networks are suited for problems involving image data as input, with the images being vectorised into a 2D or 3D matrix. In terms of output, ANNs may typically predict a category the input belongs to (classification) or a numerical value (regression) [7].

1.2 Application Context and Rationale

Predicting perceived age is an interesting problem that can be modelled as a classification or a regression task. A classification task as a person belongs to a particular age category or a regression task as a person's age can be thought of as a continuous number to reflect the continuous ageing process. Perceived age is how old one looks for one's age, which differs from an individual's chronological age, which is their actual age [8].

A highly accurate deep learning model for predicting perceived age across broad demographics would be very beneficial for health and wellbeing applications. A 2016 study entitled 'Mortality is Written on the Face` showed how a simple passport-style image of an individual's face could signify mortality risk [9]. The study used side-by-side photographs of Danish twins labelled with the human-assessed perceived age of the individuals. It concluded that information in the face drove the link between perceived age and survival. Ultimately, if a Deep Learning model is developed that is more accurate than human assessed perceived age, it could be integrated into a mobile/web application to provide users with a general metric to assess their current health from just a photo of their face. Model application integration is a relatively simple step compared to building an accurate model as there are various challenges involved with building models for this purpose.

The challenges are numerous with perceived age prediction. Dataset choice is one of the significant sources of issues. Firstly, it is likely to be a problem if the dataset does not have a wide, even distribution of genders, ethnicities, and ages. A biased distribution may lead to model bias, resulting in less accurate predictions for faces of the elderly or minority ethnicities if there is relatively less training data for these groups in the chosen dataset. There must be an approach devised to understand and mitigate this bias. This is a significant problem for the purposes of this model as, ultimately, the elderly are the most vulnerable group. Secondly, whilst there is a continuously growing wealth of face images on the internet, these images are not appropriately labelled. Labelling is an important concept when performing supervised learning. During the training process, the network will attempt to learn the input data features that map to its labelled output. Lots of data is already labelled as it is objective - an image of a cat will have a cat label - perceived age labelling requires human assessors, which naturally leads to a lack of data due to this manual process. Establishing that there is a lack of data is a vital part of the problem because deep learning networks are susceptible to poor performance when they are only trained on a small array of data that lacks diversity. It is also currently unknown which elements of the deep learning pipeline provide the most benefit for this task and which elements could be added to this pipeline or tweaked. Also, it is hard to determine this without the understanding of how the network makes predictions. Without a strategy for understanding the workings of the network, it becomes harder to tweak hyperparameters and add new layers as the network becomes a 'black-box'.

1.3 Aim and Objectives

Overall Aim

The overall aim of my project will be to evaluate deep learning approaches for predicting the perceived age of people, from images of their faces that are annotated with their perceived age. The project aims to investigate the individual components of a deep learning pipeline for this task: pre-processing data, augmenting this data (online or offline), adjusting hyperparameters, transfer learning, CNN feature extraction, and finally, classification or regression for the output.

Objectives

1. **Identify appropriate deep learning training strategies.**

This objective aims to identify the training strategies in a deep learning task, which can be modified to improve model accuracy. This objective will be satisfied if I have built several models, each of which attempt to use different strategies that I have understood from the literature.

2. **Identify suitable components of a deep learning pipeline to adjust.**

There are several components in a deep learning pipeline. From how data is processed before input to how it is augmented and normalised inside the model. With image data, there are many transformations that preserve the original labels. Additionally, there are numerous key parameters, known as hyperparameters, in the model. The hyperparameters of my model and the components that can be tweaked must be identified so that these different components can be implemented and evaluated against each other.

3. **Design appropriate experiments to evaluate components of pipelines.**

This objective relates to objective 2 as I will need to devise metrics to assess the performance of the models after tuning and when pitted against each other. This objective will be met by plotting incremental variable changes for experiments to track the values/components causing a performance uptick or loss.

4. **Compare and contrast final models with existing research.**

Once my models are finalised, I will compare and contrast the performance of the models that I have created to the state-of-the-art models that I have researched. I will find it interesting to see how significant the difference in accuracy is.

5. **Use existing techniques to determine how CNNs make decisions.**

Determining how CNNs make decisions via visualisation, allows a better understanding of the predictions the model makes. After my model has been finalised, I will then apply existing techniques to understand these predictions.

1.4 Dissertation Structure

The structure of this dissertation is as follows:

Introduction

This section contains the context that the project sits in, the rationale and motivation behind the project, and the aims and objectives.

Background

The following section, *background*, involves a brief introduction to the history of neural networks and the modern tools and platforms available for deep learning tasks. This section will also detail and review current approaches to the problem of perceived age prediction.

Design and Implementation

Next, the *design* section will detail the different components of the deep learning pipeline that have been investigated and the rationale for the tools and technologies chosen to complete this project.

Following this, the *implementation* section will describe, at a high-level, the code and process behind each of the experiments.

Results and Evaluation

The *results and evaluation* section will then proceed, showing the results of the various experiments that were performed and analysing these results in the evaluation. This section will conclude with an overall review of all the experiments performed.

Conclusion

Finally, the *conclusion* section will summarise the project and discuss the original goals to see if these were met and to what extent. Additionally, personal development will also be reviewed in this section. Finally, further work in this domain will be discussed to conclude this dissertation.

1.5 Structure Rationale

Due to this project's experimental and iterative data-driven nature, it is difficult to devise a readable structure without conflating multiple sections, which would make each section's overall message unclear. I have decided to have *design and implementation* as one section because the planning, order, and reasoning for the experiments I have designed are strongly coupled with the implementation. The implementation directly produces the results, but I have not joined these sections because it would make the implementation process difficult to follow. The remainder of the sections are structured as standard.

2 Background

This section will cover key publications relevant to the aims and objectives of my project. Firstly, there is a discussion on the brief history of Neural Networks with particular emphasis on Deep Learning and its applications in tasks involving image data. It is essential to consider the history, as the evolutions and past breakthroughs in this field are fundamental to modern techniques used today. Following this, the inner workings of convolutional networks will be discussed in detail. These networks are the backbone of my project and what I will be exploring the components of. Then, existing methods and approaches to perceived age prediction will be discussed, including the datasets for perceived age prediction. Finally, the tools and technologies that make this possible will be reviewed.

2.1 Brief History of Neural Networks

Deep learning is thought to be a very modern topic in Computer Science as it has only become feasible in the last decade due to advancements in GPU technology [10, 11]. GPUs are suited to these workloads due to the parallelisable nature of the matrix multiplication computations performed during training. Deep learning is a subset of ML (**Figure 1**), which dates to 1943, making the field much older than one may expect.

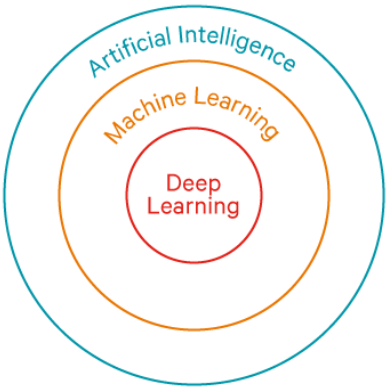


Figure 1: *How AI, ML and Deep learning relate source: https://www.qualcomm.com/news/onq/2017/09/01/whats-difference-between-artificial-intelligence-machine-learning-and-deep*

In 1943, a seminal paper by Warren McCulloch and Walter Pitts mathematically modelled neurons in the brain and grouped these neurons together to form an ANN [12]. This model of a neuron is used today as the units in a neural network and was used by Frank Rosenblatt in 1958 in which he modelled a neural network containing a single input and output layer, known as a *Perceptron*, with weights between interconnected layers and a learning rate [13]. These components are standard in a contemporary Deep Learning model, and the learning rate element is one hyperparameter that will be explored in this dissertation. Rosenblatt's model could classify real-valued inputs into one of two classes: known as binary classification. Modern deep learning networks use many layers, which is particularly suitable for image data tasks such as age prediction and often perform classification for many classes (e.g., ages from 0 – 100), rather than just two. Specialised deep networks for image tasks are known as convolutional

neural networks (CNN), and the early inspiration for these, known as the *neocognitron* (**Figure 2**), was proposed by Kunihiko Fukushima in 1980 [14]. This hierarchical multi-layered *neocognitron* could gradually extract higher-order features from the input to learn visual pattern recognition. The *neocognitron* could recognise handwritten digits and shares structural similarity to modern CNNs.

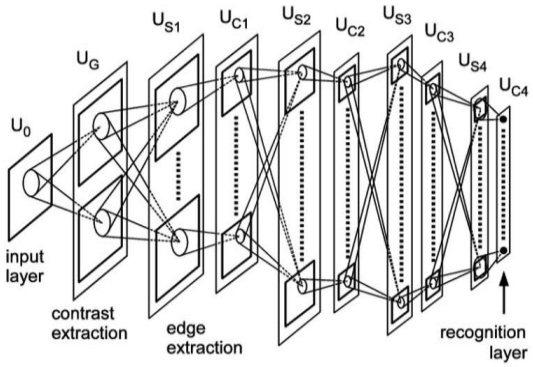


***Figure 2**. Fukushima's neocognitron [14]*

It was not until 2011 that the true capabilities of GPU accelerated CNNs were realised. At this time, the ImageNet database was launched, and it became a benchmark for image classification tasks [15]. CNNs greatly outperformed other models with their GPU accelerated performance, deep architectures, and unique convolutional layers. At the time of writing, ImageNet contains 14,197,122 labelled images. Famously, in 2012, the Deep Convolutional AlexNet architecture was used to win the ImageNet competition, with top-1 and top-5 accuracy rates of 63.3% and 84.6% [16]. Recently, the Meta Pseudo Labels approach, which uses the Google EfficientNet-L2 architecture, achieved top-1 and top-5 accuracy rates of 90.2% and 98.8% [17].

2.2 Convolutional Neural Networks

As stated, CNNs are deep feed-forward architectures suited for learning from raw image data. An example use-case could be simple binary classification. Is this image a cat, or is it a dog? The experiments that I perform will leverage convolutional architectures, as the input data will be images of faces in the ChaLearn dataset [18]. Several popular state-of-the-art implementations of CNNs, include VGG, Xception, and EfficientNet [19, 20, 21]. These architectures are particularly important in my project as understanding the underlying workings of the architectures that I am building on is particularly important for performing experiments. These architectures all take different approaches but fundamentally use the same building blocks of a convolutional architecture, which I will detail below.

2.2.1 Convolution layers

A convolutional layer convolves the input image to produce feature maps that represent what has been learnt. Images are represented as 2D (monotone) or 3D (RGB) arrays, with each element holding a value that corresponds to the pixel intensity. The convolutional operation

involves a kernel filter with smaller dimensions than the input image, which slides over the input for each set of pixels. The dot product is computed between the image and the filter as shown below:

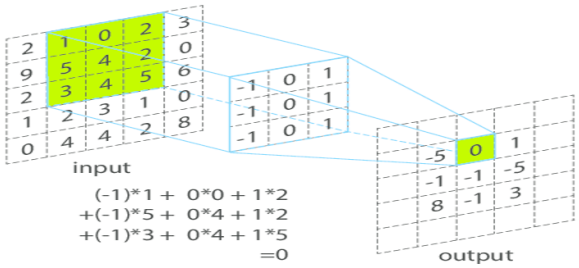


Figure 3. *Convolution operation on an image. Source: https://www.nicolasmine.com/old/scripts/solution/bokeh/Filter_decomposition.html*

These filters are organised into convolutional layers, with a filter for each colour channel. In the case of colour images, this means three channels for RGB [22]. The final feature maps are produced after the Rectified Linear Units (ReLu) function is applied, which only activates the neuron if the input is > 0. ReLu is computationally efficient while ensuring better error propagation, which is beneficial for training [20]. **Figure 4** below presents a visualisation of the activations across feature maps in the VGG16 architecture. It is important to understand the effect of each layer, as I will be modifying appropriate layers in my implementation when tuning the model's performance for perceived age prediction. For example, it may only be sensible to modify the weights of the more abstract features produced by the later convolutional blocks (blocks 4 and 5 below) if my network is pre-trained on ImageNet as described above, to preserve common features.

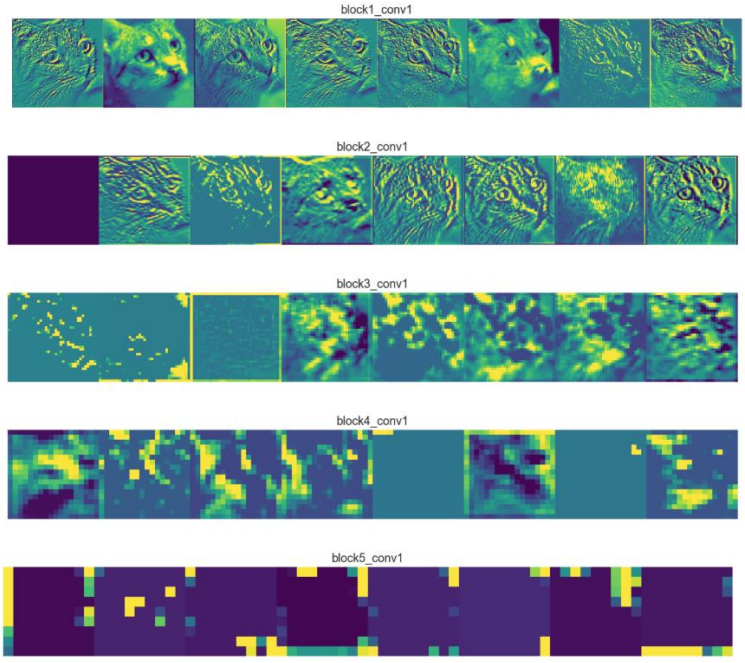


Figure 4: *Example feature maps from the VGG16 architecture: source https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2*

2.2.2 Padding

As seen in **figure 3**, the convolution operation decreases image resolution, from **5x5** to **3x3**. This is not desirable as there are often several convolutional layers in the network, decreasing the resolution further. A technique called padding can alleviate this concern; VGG16 uses a zero-padding technique that pads the outermost area of the image with zeros (black pixels) [19]. This technique is simple but preserves resolution after the dot product is calculated for each convolution.

2.2.3 Pooling

Pooling layers typically follow the convolution layers and are therefore used on the feature maps. The objective of the pooling layer is to downsample the convolutional output. VGG16 has five max-pooling layers, following the convolutional blocks. Max pooling is a type of pooling, which takes the maximum of a defined subset of the image to form a summary of a particular region of the input [22]. Max pooling is a useful technique because what is learnt from the feature map is incredibly precise – there is an output value for each pixel. Using pooling means that a more generalised understanding of features can be made. A photo of a face in ChaLearn may be at a slightly different angle than other samples, which could cause a very different output in terms of what is being learnt without pooling. Pooling accommodates these slight variances as they are inevitable when working with images.

2.2.4 Dropout

Dropout is a technique that can improve neural network performance by dropping the weighted connections between neurons with some chance p [23]. This strategy can be effective as it makes learning more robust. This is because neurons cannot simply rely on the weights from proceeding neurons to inform their activations as they may be dropped out of the network. Dropout is also a way to prevent over-fitting, which may occur for my project due to the small size of the ChaLearn dataset [18]. This is an important layer that I could add to the neural network architectures that I will use.

2.3 Data Augmentation

As stated, the ChaLearn dataset is small, and therefore the model is prone to overfitting. Data augmentation is a key element of a Deep Learning pipeline that involves modifying the original training data to inflate the volume of data fed to the model artificially. These data augmentations can involve simply cropping, flipping, or tweaking the RGB colour space of the image. Just a single consistent transformation can be effective at increasing data volumes. For example, if the number of images in the training set is N and some augmentation is applied to all of them, there is already a $2N$ training set size. Stacking multiple augmentations can then

significantly increase the cardinality of the dataset. However, not all augmentations are appropriate, as the objective is to create data that appears valid and respects the labelling of the original version. The augmentations mentioned are simple geometric transformations, but there are many more advanced techniques such as Neural Style Transfer or using Generative Adversarial Networks (GANs) to generate training samples [24]. This relates to my second objective, as data augmentation is one of the key components that I can exploit during the training process to attempt to improve model accuracy. Flipping the images on the horizontal axis to create a mirror effect would be a simple and appropriate form of augmentation on ChaLearn. However, in contrast, de-texturising images so that they lose their clarity and sharpness may not be applicable in my project. Local clarity reduction is a common technique when editing portraits to 'beautify' the subject and make them appear younger. If an older face in the dataset was de-textured, this could mean that the perceived age data that is associated with the label is lost as wrinkles in the face become smoothened. **Figure 5** presents some common augmentations.

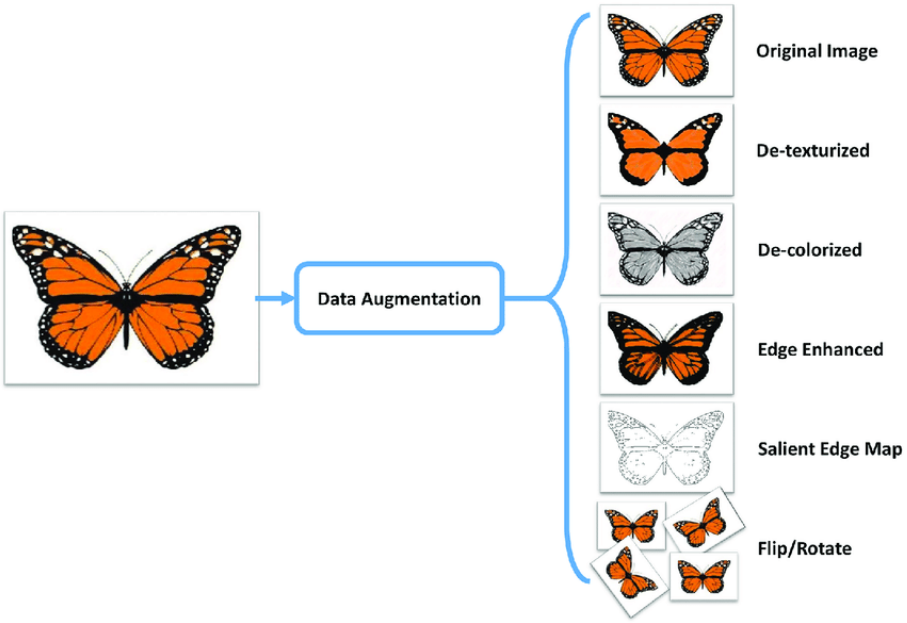


Figure 5: *Various examples of augmentation source: https://www.researchgate.net/publication/319413978/figure/fig2/AS:533727585333249@1504261980375/Data-augmentation-using-semantic-preserving-transformation-for-SBIR.png*

2.4 Face Detection

Face detection is the process of detecting a face or several faces in an image. It is a key pre-processing component in age prediction problems. This differs from face recognition, which first involves detection and then calculations to determine who the person is. In age prediction, face recognition is not necessary, but face detection algorithms are often applied to crop the input images to the face only. Face detection is difficult, as it needs to be robust enough to deal with a wide range of factors, namely, image resolution, exposure, angle of the face, and the focus on the face relative to other elements in the photograph.

An example of this is in the BridgeNet approach for age estimation [25]. This approach uses MTCNN for face detection. MTCNN is a CNN that first builds an image pyramid of 3 tiers from the input image [26]. The first stage uses a fully convolutional network to roughly estimate the

location of the face in the image by providing several candidate solutions. The next step then refines this estimation to focus on where the face is in the image by rejecting some candidate solutions. Finally, an accurate prediction is made with keypoints highlighting the key facial landmarks (eyes, nose, mouth, etc.). From these keypoints, a bounding box around the face can be constructed.

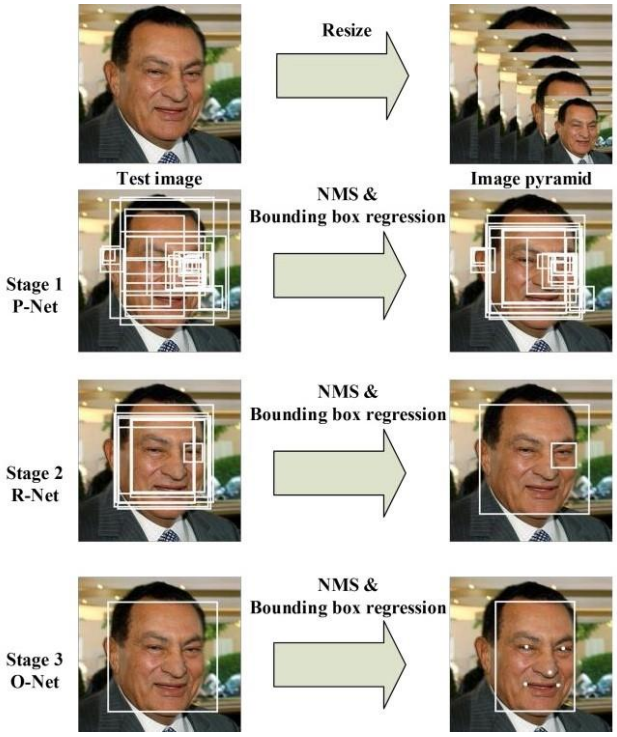


Figure 6: *MTCNN detection process [26]*

Modern CNN-based face detection algorithms are very effective. Again, another example of how deep learning methods can greatly increase the performance of tasks. Haar cascades are an older example of face detection from 2001. This method extracts haar-like features (shown below), which involves extracting sums of pixels, like a convolutional layer, as discussed previously [27].

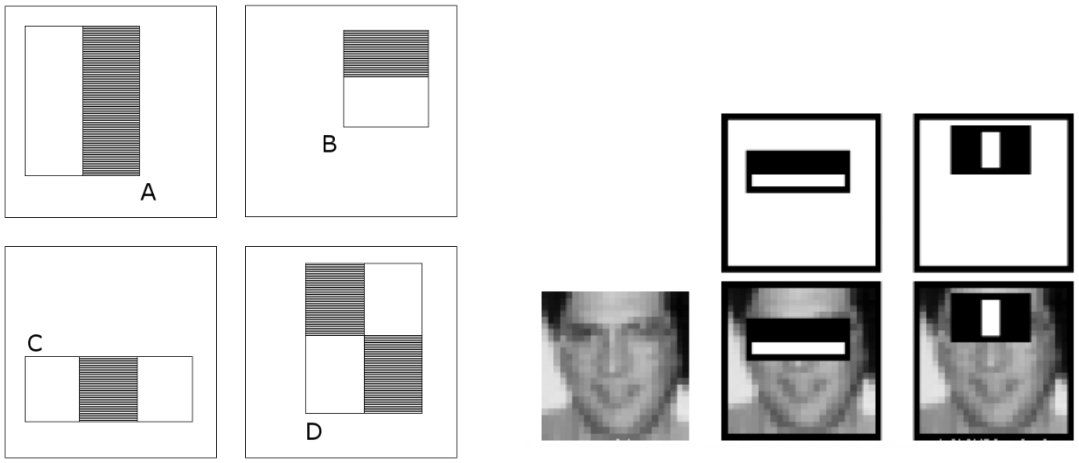


Figure 7: *Haar cascade feature maps, applied on the input image source [27]*

The cascade element improves performance by making feature extraction more efficient by doing this in stages. Each stage is a test of increasing feature complexity, and if the input fails at any one of these stages, it is discarded. Below is an example of haar cascades, using OpenCV. This method is not the most accurate, as several faces are not detected in the scene.

Figure 8: *Face detection with OpenCV using Haar Cascades source: https://stackabuse.com/facial-detection-in-python-with-opencv/*

In this project, the effectiveness of the chosen face detection approach will be evaluated in terms of robustness and improvements in prediction accuracy.

2.5 Existing Approaches to Perceived Age Prediction

2.5.1 Datasets for Perceived Age Prediction

The ChaLearn Apparent Age V2 dataset contains images labelled with the perceived age. Each perceived age label is the mean of the ages estimated by multiple human assessors using an Amazon Mechanical Turk [18]. This dataset is relatively small, containing ~4000 images for training and ~1500 images intended to validate the network. There is a fairly wide distribution of ages in the dataset, from 0 to 84. This is depicted below in **Figure 9.** This figure shows that there are by far the most data samples for younger people aged 20-30. The proportion of children in this data set, apart from 1-year olds, is much lower than the young adults. It is also notable that this dataset has a decreasing number of samples for older people, with very few samples for those aged 60+. Additionally, the distribution of ethnicities in this dataset is another issue. Ideally, this should be balanced with an equal number of samples per age category for every ethnicity represented.

The distribution across this dataset poses a problem in this project. The model may be much more accurate when predicting the perceived age of younger people due to the network having more of this data available to learn from.

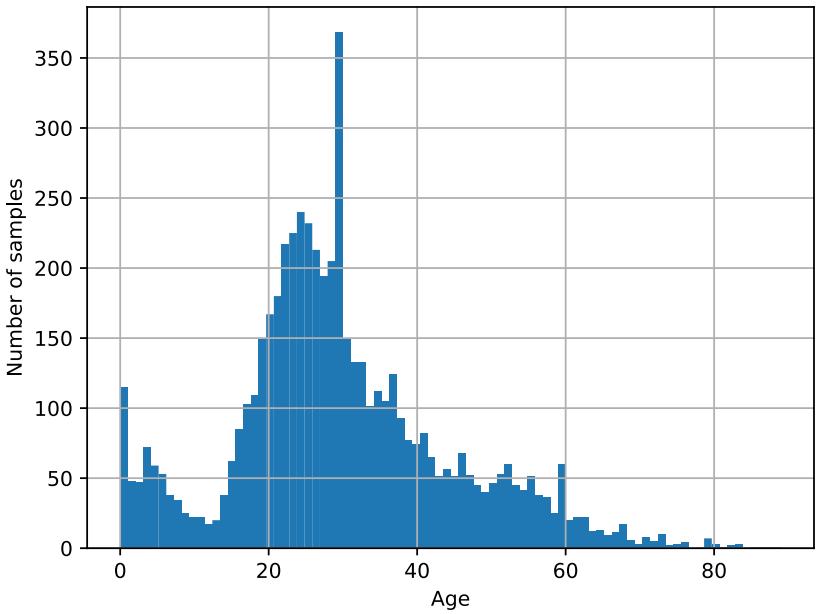


***Figure 9**: ChaLearn V2 age distribution across training and validation sets*

The size of this dataset may also become a problem, as for deep learning, it can be advantageous to have a large dataset. A network can overfit when trained on a smaller dataset. This could mean that it becomes very proficient at predicting the perceived ages of the labelled images that it has seen during the supervised training process but cannot accurately predict new, unseen images from validation or test data. I will be investigating the effects of this when using this small dataset in my project.

Despite these issues, ChaLearn is the first perceived age labelled dataset, and it is also very clean – most samples contain a good quality image of a face without issues such as blur or excessive compression artefacts.

ChaLearn was released as part of a competition, and therefore the approaches below will mention the results achieved using the Mean Absolute Error (MAE) metric.

2.5.2 Datasets for transfer learning

As ChaLearn is small, transfer learning is often used. Transfer learning takes knowledge a neural network learns in one task, saves this knowledge in weights, and applies this knowledge to a new task by initialising the new network with these saved weights. This technique can work if the data in the first dataset shares similarity with the target dataset. For perceived age prediction, much larger chronological age datasets can be used for pre-training before fine-tuning on ChaLearn's perceived age labelled images. Popular chronological age datasets include

MORPH, UTKFace, and CACD [28, 29, 30]. MORPH contains 55,134 facial images of 13,617 subjects aged between 16 and 77. MORPH contains multiple samples per subject as it was originally intended for studying face aging progression. UTKFace has a larger age range of facial imagery with a span of 0 to 116 years old, including various image resolutions, exposures, and poses. Finally, CACD contains 163,446 images from 2,000 celebrities collected through web scraping. This dataset calculates age by subtracting the birth year from the year the photo was taken.

2.5.3 BridgeNet

Recently, there have been several approaches to the problem of perceived age prediction: the most recent being BridgeNet and CORAL [25, 31].

BridgeNet acknowledges that predicting age is a complex problem in the domain of computer vision due to the nature of the ageing process. The data is diverse across a wide range of classes (ages), and ageing is a *non-stationary* continuous process with different ageing characteristics depending on the subject's age. This relationship is visually depicted below. Due to the continuous nature, the appearance of people aged a year apart may be very similar, which makes it harder for a CNN to differentiate the features that distinguish these categories. This is described in BridgeNet *'this similarity relationship caused by continuity causes the appearance of faces to be very similar at adjacent ages'* [25]. BridgeNet uses a divide and conquer approach and divides the data into age groups. Regression is then applied to these groups to model continuity between these ages. Gating networks then divide the weights between the ages in a binary-tree like structure. This data structure merges the rightmost leaf node and leftmost leaf node for nodes at the same depth to model the continuous ageing process. The underlying VGG16 architecture is used to build this network. Overall, this process results in a **MAE of 2.87** on the ChaLearn test set.

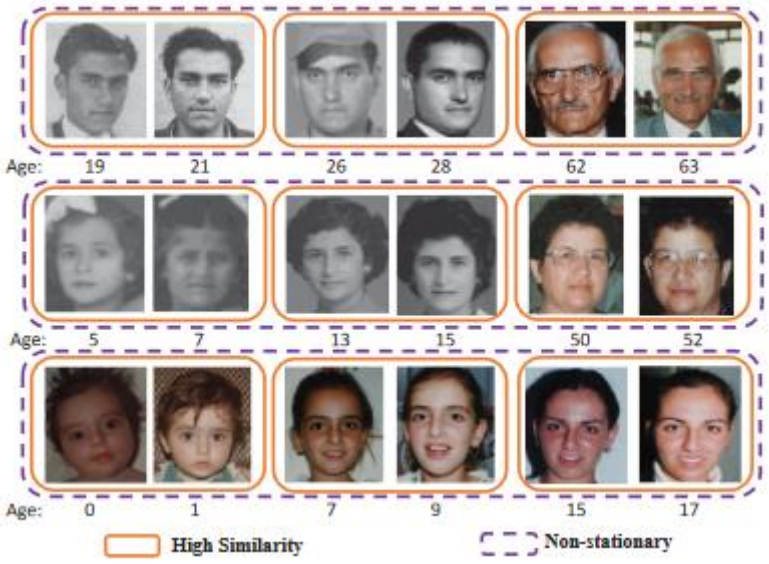


***Figure 10**: Representing age relationships [25]*

This is a key publication in relevance to my project objectives as it details hyperparameters and the architecture used for this problem of age prediction. Additionally, as stated earlier, it describes the MTCNN architecture used for facial detection, a key pre-processing stage that I will implement for the ChaLearn dataset.

2.5.4 CORAL

The consistent rank logits (CORAL) approach focuses specifically on improving the standard categorical cross-entropy loss function used for classification problems to accurately model the ordinal nature of age classes [31]. CORAL does this by devising a framework of three main components. These are a custom label encoding process, a modified network output layer, and a custom loss function. Ordinal data is data with naturally ordered categories, e.g., a person aged 30 is older than a person aged 20. Standard classification loss functions such as cross-entropy will treat the distinct ages as independent labels and not a strongly ordered set, which is undesirable as the ageing relationship between data points is lost. The CORAL approach ensures that classification is consistent while capturing the *non-stationary* ageing property as alluded to in BridgeNet previously.

This paper modified the ResNet-34 architecture. The last SoftMax output layer of ResNet-34 was replaced with the custom binary tasks layer. This output layer works in tandem with the extended binary label encoding that CORAL uses. Extended binary label encoding is similar to one-hot encoding. If there are four ages/classes, i.e., 1,2,3,4 then the age 1 would be converted to [1, 0, 0, 0], age 2 as [1, 1, 0, 0] and so forth. These label vectors referred to as "binary tasks" were treated with equal importance weights (λ). The output layer is fed these class labels and the total classes (K) – 1. The final component which ties the label encoding and custom output layer together is the custom loss function. The loss uses the output of each of the binary tasks $g(\boldsymbol{x_i}, \boldsymbol{W})$ with bias units and a logistic sigmoid function. The loss is minimised during training and defined below as:

$$L(\mathbf{W}, \mathbf{b}) = \\ -\sum_{i=1}^{N}\sum_{k=1}^{K-1} \lambda^{(k)}[\log(\sigma(g(\mathbf{x}_i, \mathbf{W}) + b_k))y_i^{(k)} \\ + \log(1 - \sigma(g(\mathbf{x}_i, \mathbf{W}) + b_k))(1 - y_i^{(k)})],$$

Figure 11: *CORAL loss [31]*

The use of ordinal regression outperformed the standard categorical cross-entropy loss used in classification tasks for age estimation across the MORPH, AFAD, and CACD datasets. Importantly, CORAL is architecture agnostic. Therefore, using CORAL could be beneficial towards my project objectives when looking at components within the CNN to improve prediction accuracy.

2.5.5 DEX

An older approach to this problem is Deep EXpectation of apparent age (DEX) which details the problem as a classification problem using a SoftMax activation function [32]. The SoftMax function outputs the probability of the input image belonging to each of the given classes, of which they are 101 in this approach. DEX recognises the issue of overfitting on the small ChaLearn dataset. It, therefore, builds a dataset of "*524,230 face images crawled from IMDB and Wikipedia* "to train the model on, which at the time of publication was the largest public age prediction dataset. IMDb-Wiki is similar to CACD but larger in scale. The DEX approach utilises the VGG16 architecture, and the Mathias et al. face detector [33]. DEX outperformed human age assessors and won the 2015 ChaLearn challenge with a **MAE of 3.22**. This paper is another essential publication for my project. It describes the approach in detail and the rationale for pre-training the network on IMDb-Wiki and then utilising transfer learning to apply the understanding of age estimation to the perceived age estimation problem in ChaLearn.

2.6 Tools and Technologies

Python is a language increasing in popularity, ranking fourth on Stack Overflow's annual survey, and it is the most common language for modern deep learning frameworks [34]. This is due to the language's readability and ML packages such as NumPy, OpenCV, TensorFlow, and PyTorch in the Python ecosystem [35, 36].

TensorFlow and PyTorch are two of the most popular deep learning frameworks, written by Google and Facebook. Tensorflow has the advantage of native visualisation and live logging support with TensorBoard, as currently, PyTorch does not offer this functionality. Visualisation is one of my objectives, which makes TensorFlow attractive for this task. However, PyTorch provides a more intuitive Pythonic API than TensorFlow as TensorFlow is written for several languages, whereas PyTorch, as the name suggests, is only for Python. For code deployment in production, TensorFlow is advantageous. It provides the TensorFlow Lite API for model production to mobile devices and the firebase backend for serving models in web applications. However, for my project, deployment is not a concern.

Tensorflow and PyTorch are both relatively low level on their own, so instead, higher-level frameworks have been researched, which provide abstractions for common operations in deep learning tasks. Fast AI sits on top of PyTorch and has accompanying forums and a tutorial series and book named Deep Learning for Coders; it is a very new framework and is currently undergoing many changes [37, 38]. Unfortunately, it is not currently available for Windows, so Windows users are forced to use a cloud environment. This is not ideal for my project because the free-tier of cloud environments often comes with strict quotas and limited GPU time. Alternatively, Keras is higher-level TensorFlow framework that is part of the core TensorFlow API [39]. Keras is a more mature library, and the integration with TensorFlow means that there is a wealth of documentation available. It is important to make a sensible decision about tooling in my project. A good set of tools will ultimately accelerate the development process, and the experiments that can be performed to satisfy the objectives.

3 Design and Implementation

3.1 Introduction

This section will detail the experimental plan that I have created for the project by visually depicting how the individual experiments fit into the deep learning pipeline. I will then enumerate through each of the testable components of the pipeline in detail, explaining the corresponding experiments that can be performed. Finally, the last part of the design section will focus on the tools and technologies selected to achieve the aims of the project and the methodology for data collection. In the implementation section, the process will be described for each of the components, with key figures illustrating image modifications as appropriate in some of the experiments.

3.2 The Deep Learning Pipeline and Evaluation Process

Figure 12 on the following page depicts a Deep Learning pipeline annotated with elements specific to this problem domain. These components can be tweaked and adjusted to gain better results for the problem.

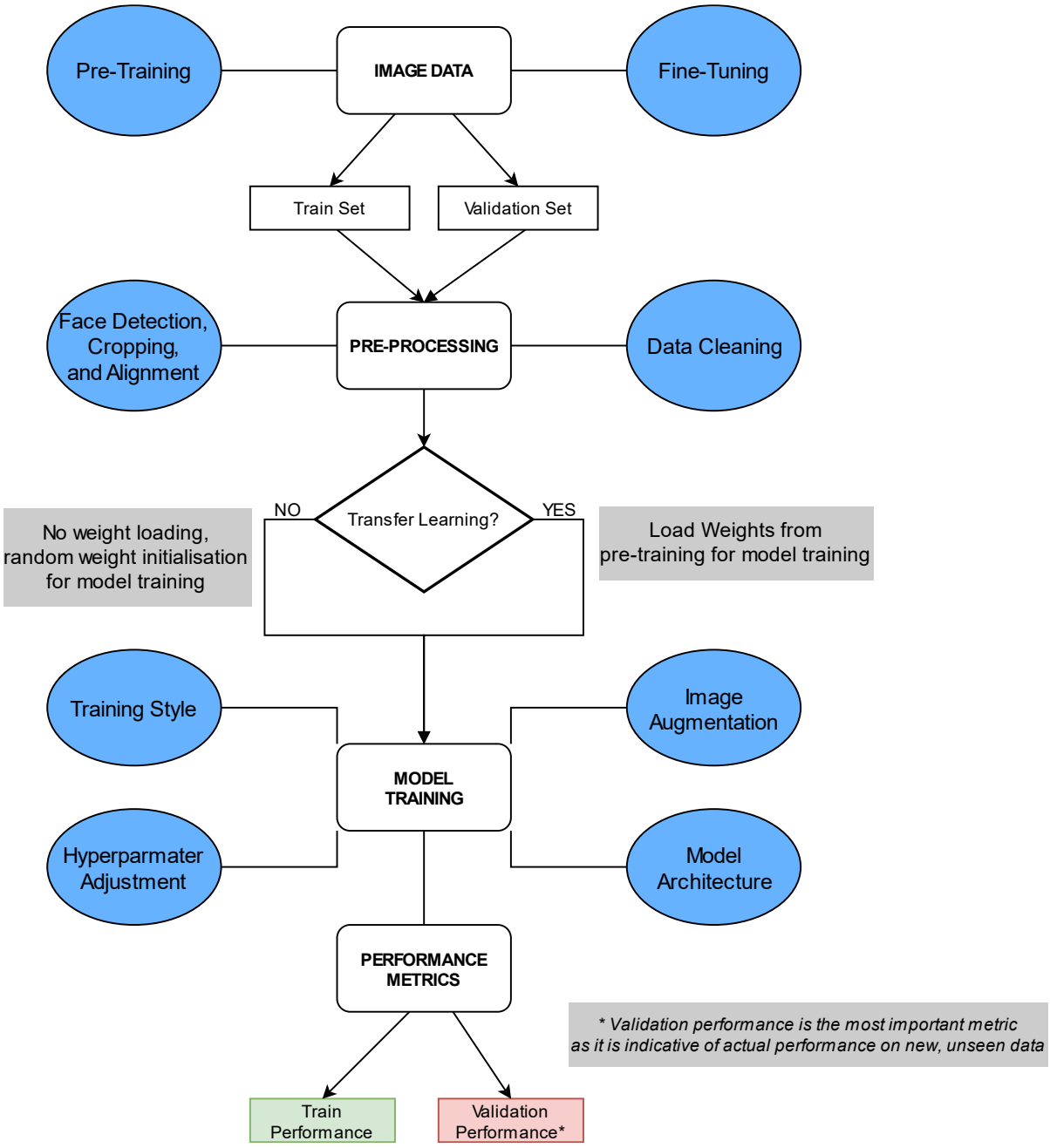


***Figure 12**: The Pipeline and Evaluation Process*

3.2.1 Gathering appropriate Image Datasets

The starting point for the implementation in this project will be selecting appropriate datasets. The final dataset which I will be predicting on is the perceived age ChaLearn dataset, as mentioned in the previous chapter [18]. To reiterate, this is a small dataset, so pre-training a model on a similar chronological age dataset with the same architecture before fine-tuning on ChaLearn could yield a significant performance improvement. This is described in detail in later sections. A suitable chronological age dataset of this nature would be IMDB-Wiki [40].

Therefore, I have added the blue labels to the "IMAGE DATA" component of the pipeline as shown above to separate these instances of pre-training and fine-tuning.

3.2.2 Pre-Processing Images

Experiments on pre-processing will be a significant area in my project after I have built the initial models and tuned relevant hyperparameters to gather a baseline in prediction accuracy. When working with image data, lots of pre-processing can be done akin to what is possible in photo editing suites such as Adobe Lightroom. In the figure above, I have split this investigation into two main components. These are face detection/alignment and data cleaning. Face detection involves transforming the raw data from the datasets into a cropped version that is focused on the face area, omitting irrelevant information in the scene. Many samples in the dataset may only have the face in a small area of the image, which is unsuitable for the network as it is not clear which features correlate with ageing.

Additionally, once a face is detected and the image is cropped, alignment could be performed to further normalise and improve data consistency. On the other hand, it will be important to perform data cleaning to remove noise from my datasets. Noise in a dataset refers to incorrect samples that are not representative of the problem that is trying to be solved [41]. Noise can significantly harm the results produced when training [42]. Intuitively, this makes sense as noisy data is by nature unclear regarding its class/label. If this data is unclear to a human, it will almost certainly be ambiguous when passed to a deep learning model.

3.2.3 Transfer Learning

As alluded to in the previous image dataset component, similar chronological age datasets to the target dataset can be utilised to perform pre-training. This provides a checkpoint in learning, and what is learnt from the upstream dataset can be transferred to the downstream model that is being trained to predict perceived age. This technique can be accomplished by saving the learned weights in the Hierarchical Data Format (HDF) [43]. Additionally, models pre-trained on ImageNet can be used as they have already gained knowledge on general image classification. Transfer learning is an optional but important area to experiment on as, ultimately, it provides the potential for larger datasets to be leveraged.

3.2.4 Model Training

Finally, the last stage of the pipeline is the actual training of the model with the pre-processed dataset and weights that have been selected. The training process itself is where the possible number of experiments explodes as many parameters and processes can be tweaked and controlled. For example, training style will determine the strategy the model will use to learn. For instance, the model could classify the face image into discrete pre-defined age classes or

perform a regression to output a real-valued age. I will explore both training strategies in my experiments.

Perhaps more important is the underlying architecture. Fortunately, many general-purpose convolutional architectures for tasks involving image data already exist that perform extremely well when classifying a diverse range of images, such as VGG16 [19], which has a Top-5 accuracy of 0.901% on the validation portion of ImageNet [15]. VGG16 has been chosen as it is used in both DEX and BridgeNet.

Several hyperparameters can be tested at this stage, such as the batch size, learning rate, total epochs, and the number of frozen layers in the network. Additionally, images can be appropriately augmented on-the-fly at train time to provide the network with more data than is present in the original dataset, to provide more diversity and variation between samples.

3.2.5 Performance Metrics

The output of the model itself, which is the output of my experiments, will be performance metrics during the training cycle. These metrics give an indication of the rate at which the model is learning and if it is overfitting to the training data. Overfitting occurs when a model does not generalise well, shown by poor performance on the validation portion of the input data compared to the training data. Therefore, the values achieved on the validation portion of the dataset are indicative of performance on unseen data. These metrics will be defined as appropriate for the task that is being performed. It is important to select correct metrics as part of my experimental design. Therefore, in this case, I will use the MAE metric, which measures the average of the absolute difference between the predictions made by the neural network and the actual ground truth values of the samples. This metric is used widely in the literature, becoming a *"de facto standard for age estimation"* [32]. As part of my experimental design, it is important to keep this metric constant across my experiments, regardless of the parameters being adjusted or the model itself. This will allow effective comparisons to be made between experiments to critically evaluate their results and identify the parameters and components of the model that are most effective for accurate perceived age prediction.

3.3 Experimental Plan

The experimental plan for this project has been carefully considered to ensure consistency between experiments, clear results, and concise conclusions to be made. My project is suited to Agile delivery, as there are no clear deliverables. Instead, many experiments are iteratively performed, with each experiment influencing the next set of experiments as combinations of the best parameters attained from previous experiments are re-used.

Firstly, the base models are built from a suitable backbone architecture. These models will employ different training strategies, namely classification and regression-based outputs, which provides a wider range of results and an increased combination of experiments. These two models are a prerequisite to the other experiments that are outlined above. Following this, experiments progressively analyse different components of the pipeline/model in turn to improve results. The parameters are tested independently at first and then combined to explore

what combinations are effective in producing the lowest MAE across the validation set of the original data.

3.3.1 Data collection

This project produces a great deal of data due to the number of iterations each time a model is trained. Each cycle produces data about training and validation, and this is repeated twice for the classification and regression-based models. To store this data, my models will use TensorBoard callbacks during the training cycle. This stores data in a hierarchical structure in named directories. This ensures experimental data is not lost, which is important as experiments take a long time to complete while also providing a method for combining experimental results by copying necessary runs between directories.

3.4 Chosen Technologies

To achieve the aims of this project, the following tools and technologies have been chosen. These technologies have been selected for various reasons, including available libraries relevant to the problem domain, ease of use and debugging features.

3.4.1 Python

The programming language selected to drive this project is Python. Python is an interpreted and dynamically typed language especially suited for Data Science and AI applications due to the vast array of Python modules suitable for these workloads. These modules are distributed through The Python Package Index (PyPI) or, for scientific computing, Conda is a rich alternative repository. The Anaconda distribution of Python can be installed to use Conda packages, which features many pre-installed common packages and a robust package manager and virtual environment solution that handles and avoids dependency conflicts by checking for any conflicts installing, updating, or removing packages.

3.4.2 PyCharm and Jupyter Notebook

PyCharm was the chosen IDE for developing my code. It provides excellent integration with Git for version control as well as Anaconda virtual environments. Additionally, PyCharm has specific features for scientific computing. For instance, a panel that helps visualise the contents of an N-dimensional NumPy array. This is useful as NumPy arrays encode images, which I will be transforming. Finally, PyCharm also has support for Jupyter Notebooks. Notebooks help with experimentation as it is an environment where you can visualise images and only run

specific parts of the code known as “cells.” This is productive when you are often revisiting parts of the code to make minor changes to variables.

3.4.3 TensorFlow and Keras

I have chosen to use TensorFlow as the primary framework behind my project [35]. A key factor in this decision was the integration with Keras [39]. Keras, built by François Chollet, was once a separate higher-level library as an abstraction layer on top of TensorFlow to simplify developing, training, and evaluating Deep Learning models. However, since Tensorflow 2.x, Keras is integrated into TensorFlow. This integration is particularly useful. It eases an already complex Deep Learning setup procedure that requires specific versions of a Nvidia CUDA installation alongside the Nvidia CUDA Deep Neural Network Library (cuDNN) and manual drag-and-drop installation for auxiliary libraries. Key reasons for choosing Keras were its maturity, intuitive model building APIs and quality documentation.

3.5 Implementation

This sub-section goes through the various stages of the implementation phase. Models were built, datasets were processed, and experiments were undertaken to evaluate the different elements of the pipeline.

3.5.1 Creation of Deep Learning models

The first stage of this project involved building two models with which the input data of facial imagery will be passed in to train the model and evaluate its performance. The models used two distinct strategies to produce the predicted age result.

Firstly, a classification model was implemented using the chosen Keras framework. This model was intended to be trained on the ChaLearn target dataset to provide a perceived age prediction baseline for future experiments. Several utility functions from Keras were chosen to aid experiments planned in the future to produce this model. I ensured the models were flexible and extensible to support this. The ChaLearn dataset was then downloaded into separate training and validation folders, with 20% of the data reserved for validation purposes. Once this data is on disk, the images are loaded into the model using the *ImageDataGenerator* class to yield batches of the training and validation data to the model during the training process. This class was researched in depth to assess the suitability of it for future experiments. I chose this class as there are appropriate *flow* methods for loading data from various sources, including directories, CSV files, or simply NumPy arrays. Also, batch size can be set through a parameter – this was key when it came to performing my initial experiments of hyperparameter tuning. Additionally, batching data is important for memory efficiency; holding the entire dataset in memory is not feasible. Another important aspect of this class is the ability to implement train-time data augmentation. While this does increase the training time per epoch, it is a more

productive approach than augmenting data offline and saving/loading these samples to/from disk. Before training, the initial classification model pre-processes the input data into a form suitable for the network. In this case, pre-processing involved subtracting the mean RGB value from each pixel for each image with examples shown below [19].

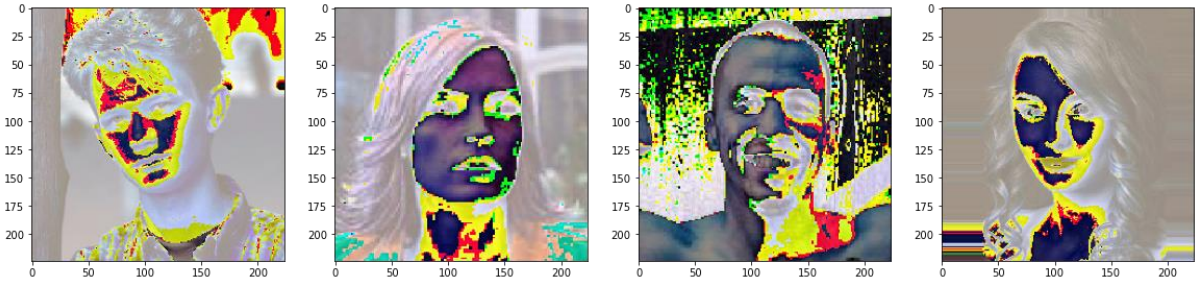


***Figure 13**: VGG16 Pixel Scaling performed on ChaLearn images.*

After performing this input-processing, the discrete age labels for each image were encoded using One Hot Encoding. One hot encoding transforms the labels into a binary matrix whereby the index of the '1' bit represents the value. The cardinality of this matrix is the total number of classes or ages in this instance. For example, if there were 10 different ages in the dataset from 0-9 the age 1 would be encoded as [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]. One hot encoded data is easier for the model to work with than raw categorical data for classification like in this instance. Once the generator was defined for loading the image dataset, the network was built using the Keras sequential API. The sequential API allows models to be built linearly with layers organised in a stack structure. For this project, the sequential model was a suitable level of complexity to appropriately define my model. The images were going through a linear series of transformations through the network.

The pre-processing performed on the images is the same as what was used during training the VGG16 architecture. This model leverages the VGG16 architecture due to its performance in image classification tasks. The weights of the model were initialised with the ImageNet weights by using the Keras Applications API. This pre-training was done to improve performance and reduce the training time as there were time constraints for the project, and the aim was to focus broadly on the components rather than delve into building novel models from scratch. By default, the last layer of the pre-trained VGG16 model has 1000 output classes corresponding to the ImageNet categories originally used. The model I have built removes this last layer and replaces it with a dense layer with the number of neurons equal to the number of distinct ages in ChaLearn. Additionally, by default, all the layers that are present in the initial model are not frozen. This behaviour is not desirable as it means that the calibrated weights will be modified, and the knowledge they represent is lost [44]. Therefore, the baseline model that was created only leaves the final dense classification layer trainable to combat this behaviour. The SoftMax activation function was selected for this final layer, which outputs the probabilities of the input belonging to each class.

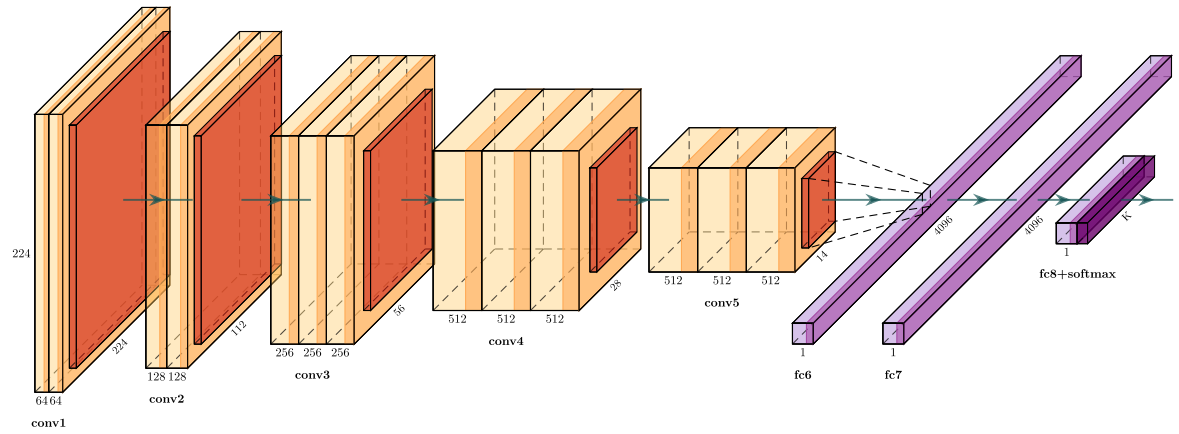


Figure 14: *Visualisation of the classification architecture excluding top age classifier, created with PlotNeuralNet [45]*

After defining the architecture, the loss function, performance metrics, and optimiser to train the model can be set. For this classification-based model, the loss function is the standard categorical cross-entropy appropriate for classification tasks with a SoftMax output and one-hot encoded class labels. The metric for this models is the MAE. This is typically used for regression, but the model employs a custom metric that translates the output probabilities into an expected value formation by computing the sum of the age classes multiplied with the SoftMax age probabilities. This refinement was used in DEX and allows use of MAE by creating a regression output for the classification model [32]. Finally, an optimiser is defined for this model. The optimiser is responsible for reaching the minima of the loss function by adjusting the weights after each epoch. The rate of adjustment is controlled by the learning rate hyperparameter, which facilitates gradient descent. This model uses the Adaptive Moment Estimation (Adam) optimiser. The adaptive in this optimiser refers to the fact that it does not use a static learning rate. Instead, the learning rate may increase as momentum is gained, speeding up convergence [46].

The second model which I created used a regression-based learning strategy. The process for creating this model shared similarities with the classification model. Again, the *ImageDataGenerator* class was used to allow comparisons to be made between the two models when experimenting on augmentation. This model used the floating-point age labels directly from the ChaLearn *train_gt.csv* file; therefore, no one hot encoding took place. The architecture layout for this model used VGG16 with a modified final layer. This was a single neuron with a linear activation function to represent the real-valued regression output. The other aspect that differentiates this model from the classification model is the loss function. In this case, the MAE is used as the loss function and the metric. This is an appropriate approach as the model's output can be directly compared to the ground truth for each sample present in the *valid_gt.csv* and *train_gt.csv* files, respectively.

3.5.2 Hyperparameter Tuning

Following the initial creation of both the regression-based and classification models, the first set of experiments took place. Deep learning models need to be calibrated by providing values for key *hyperparameters* that are typically constant throughout the training process, which differs from the weights and biases of neurons inside the network that are updated at each epoch. The hyperparameters are problem and architecture-specific. Therefore, a rule-of-thumb approach is not typically suitable when finding the optimal values. I used an empirical approach to tune the hyperparameters of my models. Using pre-trained architectures such as VGG16 reduces the number of elements that benefit from tuning (e.g., layer types and kernel sizes do not require tuning). Automated tuning strategies usually involve a domain of parameters to test and an objective function to maximise or minimise. Automated approaches can involve random searches through pre-defined ranges, an inefficient exhaustive combinatorial approach in which all possible combinations are tested, or more efficient state-of-the-art methods such as HyperBand [47, 48].

The empirical approach I used to tune my VGG16 classification and VGG16 regression models focused on three key hyperparameters. These are the *batch size*, *learning rate,* and *trainable layers*.

The batch size hyperparameter is passed to the *ImageDataGenerator* during model configuration. This allows for mini-batch gradient descent to occur during the training process. The model can then 'see' more than a single sample per epoch, which allows for more accurate iterative updates of the weights inside the network as loss measurements become more accurate with more data to work with. GPUs are suited to working with batches of images as this is naturally a task that can be parallelised. However, there is an upper bound to the batch size as batches must fit into GPU VRAM. The feasibility of the manual approach to tuning batch size was reaffirmed due to the limited range of values for the batch size, due to VRAM memory limitations, and the fact that batch sizes should normally be defined in powers of 2 to comply with the memory architecture.

The learning rate hyperparameter is passed to the *Adam Optimisation* algorithm in my models. The learning rate hyperparameter is essential and defines the relative amount the weights should be updated during back-propagation regarding the error rate reported by the loss function. In the book *Deep Learning,* the following is said about the learning rate hyperparameter *"The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate."* [49]. The importance of this hyperparameter is simple. A learning rate too low will not allow convergence and progress to be made with learning, while a high learning rate will result in instability as weights become adjusted too strongly at each epoch. Again, this is a suitable parameter for testing through trial and observation. The learning rate is commonly defined in orders of magnitude, with a default being 0.001 for the *Adam Optimiser* in Keras.

Finally, as mentioned previously, the 16 layers in the VGG16 architecture, when imported, are left unfrozen. This means that for each epoch, *all* weights will be updated. This is desirable when training a model from scratch using an initial random initialisation of the weights. However, my model takes advantage of transfer learning as it is initiated with the weights from ImageNet training [15]. The initial convolutional layers learn important generic features that can be

applied to this perceived age prediction problem. Conversely, only leaving the last layer trainable provides great stability in training but leaves the parameter exploration space too narrow to achieve good results. This was reflected in my findings when testing orders of magnitude for the learning rate. The network stops quickly stops learning when all layers are left unfrozen. The performance during this training phase is poor and falls far behind the models with a restriction on the number of unfrozen layers. The most effective number of layers to be left unfrozen was eight from the top of the architecture. In the context of VGG16, this means leaving the last convolutional block unfrozen and trainable. The later convolutional blocks learn representations of the most abstract problem-specific areas. For example, this could involve creases in the skin and other indicators of ageing.

Overall, discovering the optimal combination of hyperparameters for both the regression model and the classification model was a crucial first round of experimentation. These parameters need to be set every time a model is trained. They are fundamental to obtaining good performance levels.

3.5.3 Transfer Learning

At this stage in the project, the models were only being trained on the ChaLearn dataset. This dataset is biased with face samples of subjects aged 20-30 and only contains 5600 samples. Therefore, the next set of experimentation involved pre-training on the IMDB-Wiki dataset that has been identified during research. This dataset has > 0.5 million samples, which can be annotated with the individual's chronological age. Despite this dataset being annotated with chronological age, it has a high similarity to ChaLearn and therefore suitable for knowledge transfer during fine-tuning.

Pre-Processing IMDb-Wiki

Firstly, the dataset was downloaded from the official source [40]. Two options are provided, one with the raw images and a version with the images cropped to the face with a 40% margin. The latter version was downloaded due to the cumulative size of 8GB compared to the ~270GB variant.

The IMDb and Wiki datasets came bundled with a Matlab (.mat) file that held the image metadata and data about the sample, including the birth year and confidence scores. The confidence scores (face score, secondary face score) pertained to the likelihood of a face being present in the image and the likelihood of more than a single face being in the image, respectively.

This Matlab file was read using the *loadmat* utility in SciPy. This function converted the Matlab file into a more readable Python format – a dictionary. Following this, the dataset was cleaned to remove noise and labelled with chronological age. The cleaning process involved three main checks. Firstly, the face score of the sample was checked. If the score was *inf,* then the sample was skipped as *inf* implies a face is not present in the image. Then, the secondary face score was checked. If there was a secondary face score, the sample was skipped as images containing more than a single face are not suitable because it is not clear which face belongs to the age label. Finally, if the sample passed the prior two checks, the subject's age was checked; if the age was in the range 0 – 101, then the sample was retained. This final age check is necessary as the Wiki

portion of the dataset contains historical figures, which would lead to class imbalance, and this type of data would not be passed in at inference time. The age was calculated using *capture time of photograph – birth year* and then saved in the filename for later parsing.

Following this data cleaning process, 224, 568 samples remained. This roughly halves the raw data but ensures the data is clean. **Figure 15** below shows that the processed data still retained age distribution with a much greater number of samples compared to ChaLearn.

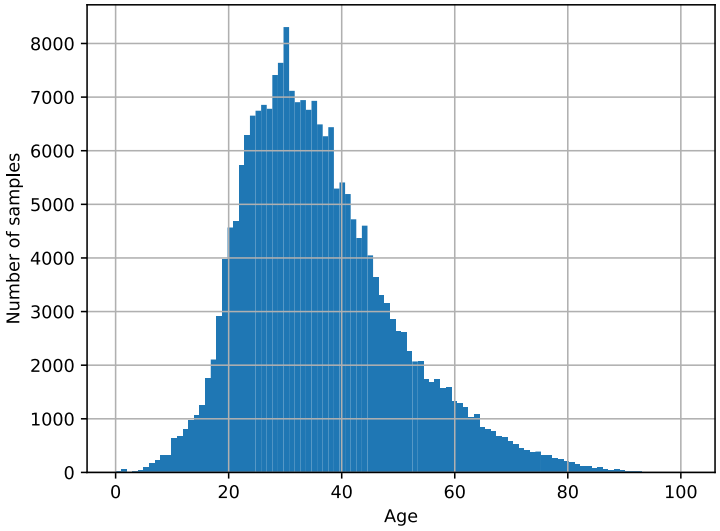


***Figure 15**: Distribution of age in processed version of Imdb-Wiki*

Transfer Learning

After this, a CSV file was generated, mapping the discrete age labels in the filename to each image. This file was used to generate the image label batches in the regression model, which was left unchanged from the model used for ChaLearn.

The file was also used in the classification model. This model had to be refactored to work with the IMDB-Wiki data due to the format of the labels. Previously, labels were inferred from the directory structure (e.g., all subjects aged ten would be in a folder named "10"), but the IMDb-Wiki data resided in a root folder. Therefore, the data was loaded into a pandas dataframe and reshaped to generate NumPy arrays of train/validation samples and labels.

For each model, transfer learning was then performed. This involved training on the large dataset and creating checkpoints when a better value was achieved on the validation set for the aforementioned age MAE metric. These checkpoints, using the Keras callbacks API, saved the weights only in a .h5 file.

Finally, when fine-tuning the model on the ChaLearn dataset, these weights were loaded into the models to kick-start training from using this previously saved knowledge.

3.5.4 Image Manipulation

After establishing an appropriate transfer learning strategy by creating the base models and manipulating the IMDb-Wiki dataset, the focus was shifted deeper into the underlying data itself. Experiments were performed on face detection involving varying degrees of crop factors, alignment of these cropped faces, and finally, data augmentation.

3.5.4.1 Face Cropping

The first area of pre-processing that was focused on was cropping the images to the face. The rationale for this was to improve data consistency while also removing some irrelevant information in the photographs. The image from ChaLearn below is just one example of an image with a significant amount of data that is irrelevant to the problem.

Figure 16: *Sample from the ChaLearn labelled 22.8*

The method of face cropping had to be robust enough to positively detect faces in over 230, 000 images from both ChaLearn and IMDb-Wiki, respectively. The Haar-cascade OpenCV method mentioned in research was considered initially, but this is an old method of face detection, so implementing this would not be a good usage of time as more accurate methods are available. The chosen underlying face detection approach utilised the DLIB C++ based CNN Face detector, which has Python bindings [50]. This method was chosen as accuracy is important, and images in the datasets had varying challenging factors such as different head positions, angles, and focus. Cropping the image datasets of this size would take considerable time on the CPU. To alleviate this concern, images were processed in batches of 32 images on the GPU. This gives great speed improvements. For some samples in ChaLearn, the image file size was greater than 1MB, which led to CUDA out-of-memory errors. Due to this, image resolution was reduced using OpenCV by resizing to 299 x 299 before being passed to the face detector. The existing face detector returns an array of facial landmarks in the form of pixel coordinates. These were used to slice the NumPy array representation of the input image to achieve the crop.

Face detection only failed on a very small subset of the IMDB-Wiki dataset, as this data has already been processed depending on if a face was present in the image as described previously.

On the smaller ChaLearn dataset, there was a higher percentage of failures. The 18 failures, in this case, were due to poor quality samples in ChaLearn, so this was a helpful way to identify and remove these. Below is a subset of these poor samples.

Figure 17: *Poor ChaLearn samples, labelled left-to-right 19, 1 and 3*

Initially, face detection was only applied to produce a single type of crop. This crop was a very tight crop to the face resulting from not adding any additional margin to the bounding box constructed from the coordinates returned by the face detection algorithm. Therefore, to test the impact of cropping on the face further, additional crops for each image were produced in this experimentation phase. An example of these various crop factors is displayed below.

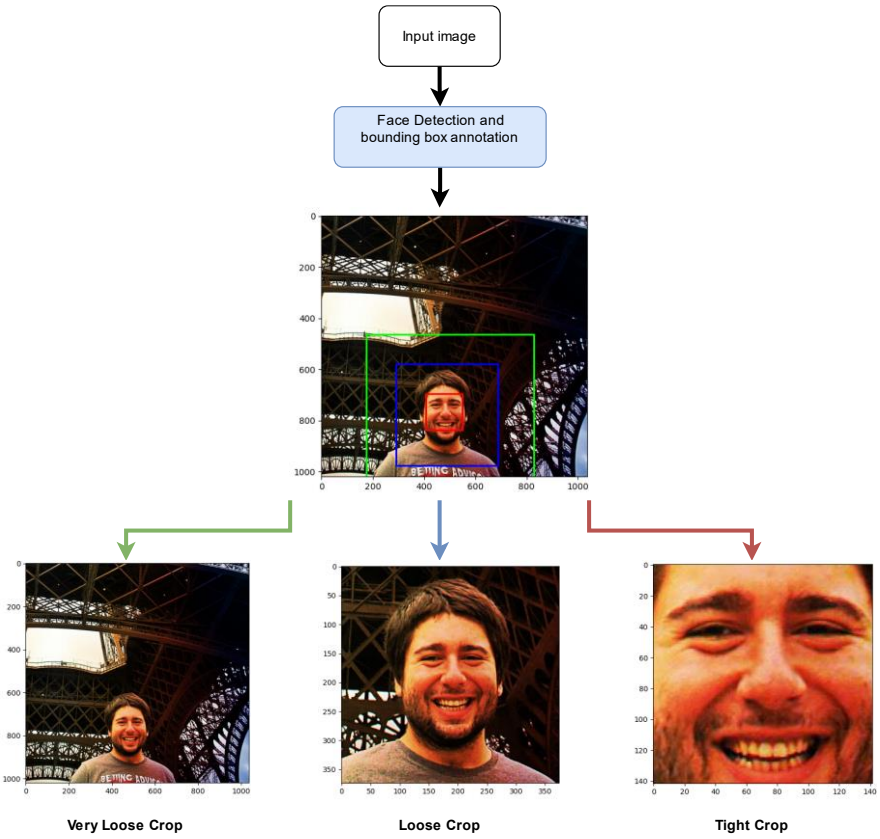


Figure 18*: Visualisation of face detection and cropping*

These different crops were created with specific elements of the images in mind. The loose crop was not expected to perform significantly better than the original image but still necessary to

test. This crop factor often contains additional information about the subject's clothing – clothing may influence perceived age. The middle 'loose' crop aimed to test if there is an impact when including the neck of the subject and their hair. Finally, the tight crop focused on just the face, providing data about facial ageing specifically.

3.5.4.2 Face Alignment

Following the successful implementation of facial cropping across the datasets, facial alignment was also applied. A WIP Python module was leveraged to align faces, which acted as a wrapper around the *face_alignment* library that uses the state-of-the-art FAN-Face model [51, 52]. The wrapper module contained an algorithm that used facial landmarks to align the image. To elaborate, the landmarks (coordinates) for the eye positions were used, as an aligned image will have the eyes on a level horizontal plane, and the centre point between the eyes can be used as a rotation point. Finally, the angle between the eyes and this centre point was used in the module to create an OpenCV rotation matrix, which is then passed to OpenCV's *warpAffine* function to perform the correct degree of rotation to align the image.

Functionality was then added to this existing WIP module to increase the speed and reliability of performing the alignment, which was important for the large IMDb-Wiki dataset. This additional functionality enabled greyscale images to be aligned, as there was a significant portion of grayscale images in the dataset, and *face_alignment* did not accept these. This was done through conversion of the underlying NumPy shape and saving these images to a temporary location. Additionally, changes were made to enable batch processing on the GPU, as once again, it would take a considerable length of time to align all faces on the CPU only.

Initially, only the tightest cropped images were aligned as these performed the best in the prior experiments. However, it was noted with these images that information was lost with noticeable zero-padding. Therefore, the previous 'loose' crop images were also aligned to mitigate this information loss. This is displayed below in **Figure 19**.

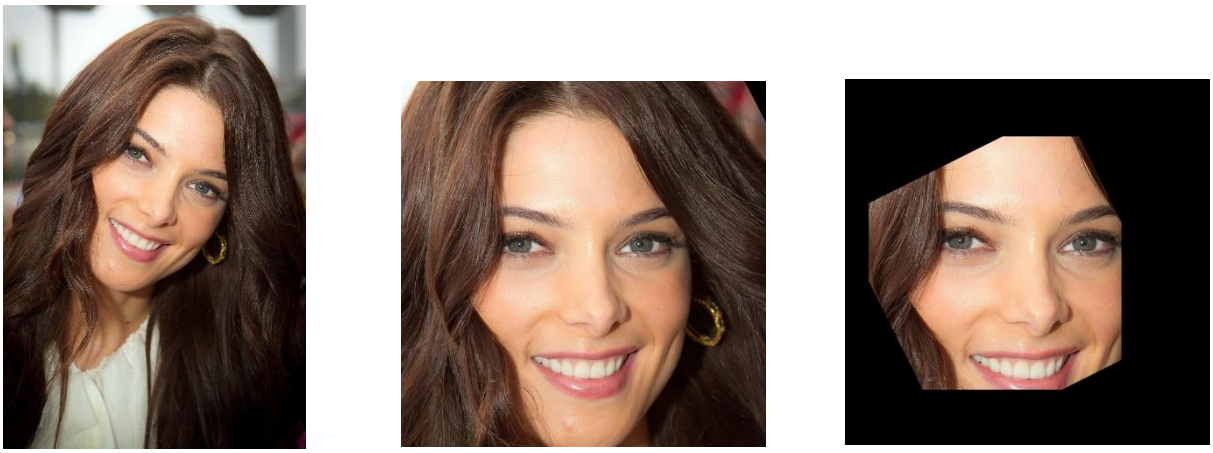


Figure 19: *Left: Original Image, Center: Aligned Loose Crop, Right: Aligned Tight Crop*

3.5.4.3 Data Augmentation

Following the image pre-processing implementation and experiments described above, train-time on the fly data augmentation was implemented for the next round of experimentation**.** This phase took a significant amount of time as augmenting data on the fly is an expensive operation, resulting in a noticeably slower time to complete each training epoch. Augmentation was implemented using the *ImageDataGenerator* class on the training partition of the dataset to increase the size of the small ChaLearn dataset. Additionally, augmentation was implemented on IMDB-Wiki, to produce better weights for transfer learning.

A myriad of augmentations were tested, all of which were selected to modify the images whilst preserving the age-related features of the image. The most effective augmentations were subtle transformations of the original images. The applied augmentations were random rotation with bounds of +10/-10 degrees, a 10% shift of the image in height or width, a 10% zoom in/out factor, and finally mirroring the image on the horizontal axis. Additionally, some images combined these augmentations. For instance, flipping the original image and applying height and width shifts. A visualisation of these augmentations performed inside the model is shown below in **figure 20**.

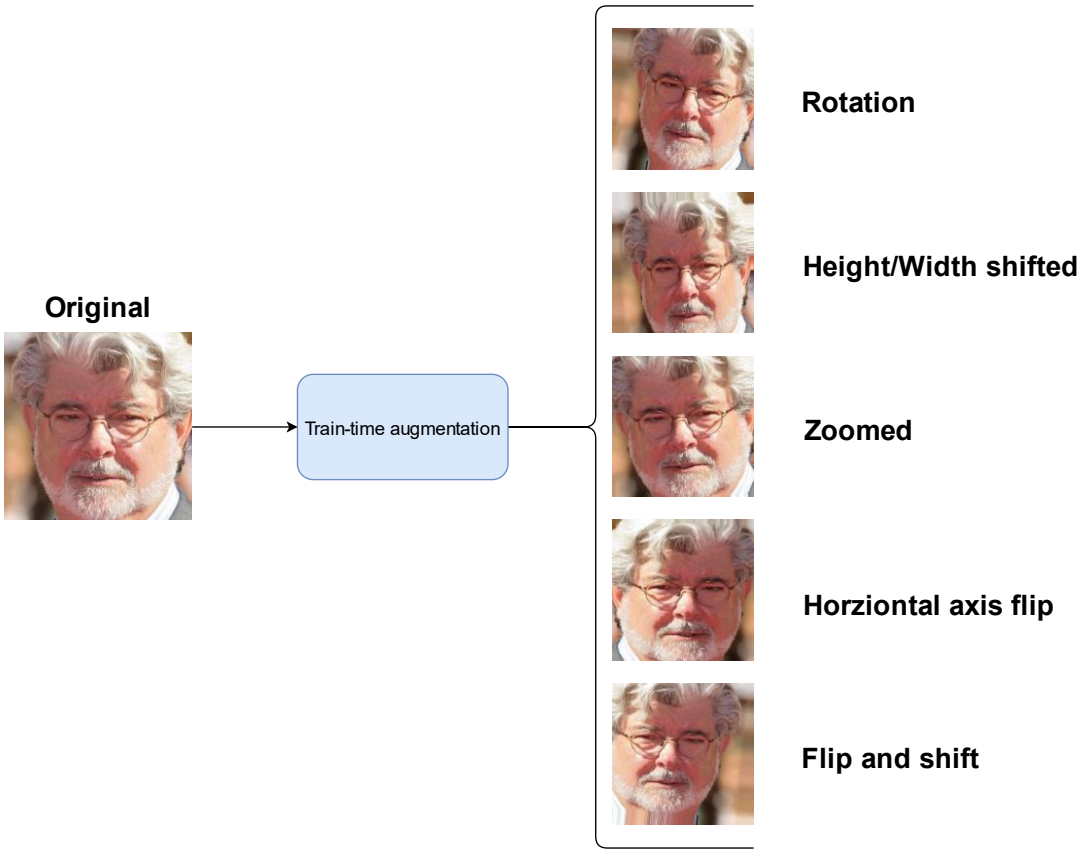


Figure 20: *Implemented ChaLearn augmentations*

3.5.5 Network Visualisation

To add interpretability to the models that have been built and meet project objectives, visualisation was implemented on the finalised models. The visualisation was done using two existing techniques with the first technique using a well-established library to ensure accuracy in these results. The techniques performed were activation visualisation using Keract and Occlusion Sensitivity[53].

Activation visualisation is a technique that allows the feature maps that are created inside the convolutional layers of the network to be displayed [6]. This then adds meaning to the 5 convolutional blocks in the finalised VGG16-based model, specifically, the individual filters. The results created from these visualisations are essentially a decomposition of the important parts of the image, according to the filters.

Occlusion sensitivity is a method that systematically hides each portion of the input image with a grey square patch. Firstly, a prediction is made on the input image in its original state and the age predicted is persisted. Then, for each occluded area of the image, a new prediction is made on perceived age, with the probability of membership to the original class stored in a 2D array (height, width) at the index of the current position of the grey square patch. This then allows comparison to be made between the unobscured image and the image with different elements hidden, to see which segments are important for prediction accuracy towards the given class indicated by the original image.

This technique was implemented on the classification model, as the final SoftMax layer gives probabilities of each age class. The confidence of the target age class for each region, can then be displayed as a heatmap overlaid on the original image. **Figure 21** below visualises the occlusion for different input images.

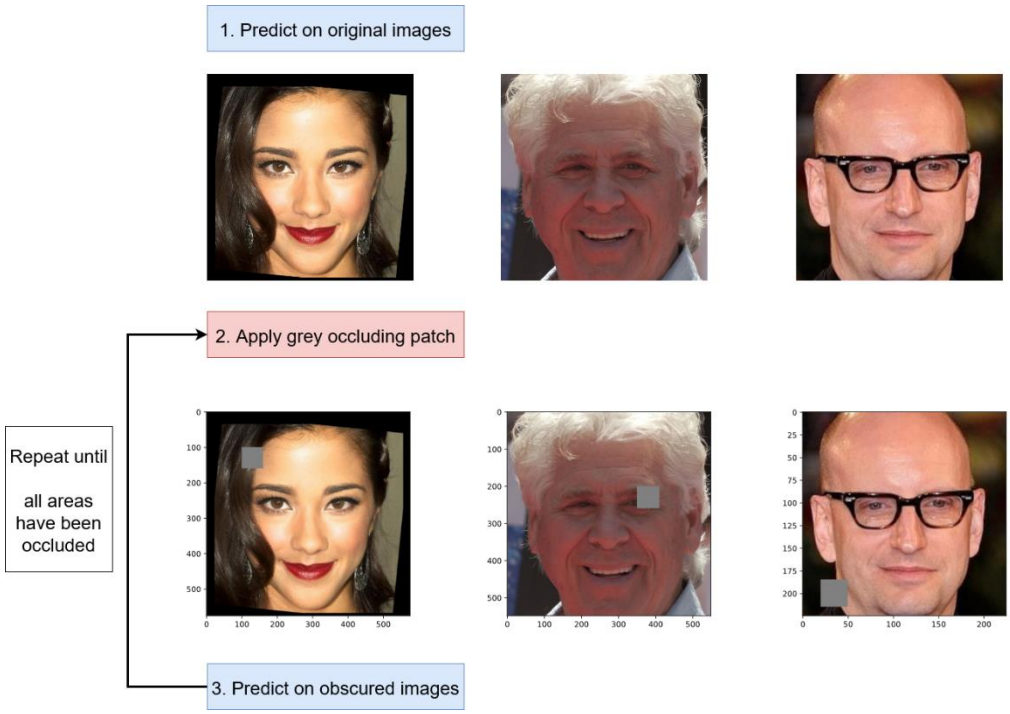


***Figure 21**: Process for building occlusion sensitivity heatmap*

4 Results and Evaluation

This section will detail the results of the experiments that were outlined in the previous chapter. Following each set of results, a concise evaluation will follow. This section concludes with an overall evaluation and critical comparison with the models identified in the literature. All experiments show both the train and validation results on ChaLearn, measuring the MAE.

4.1 Environment

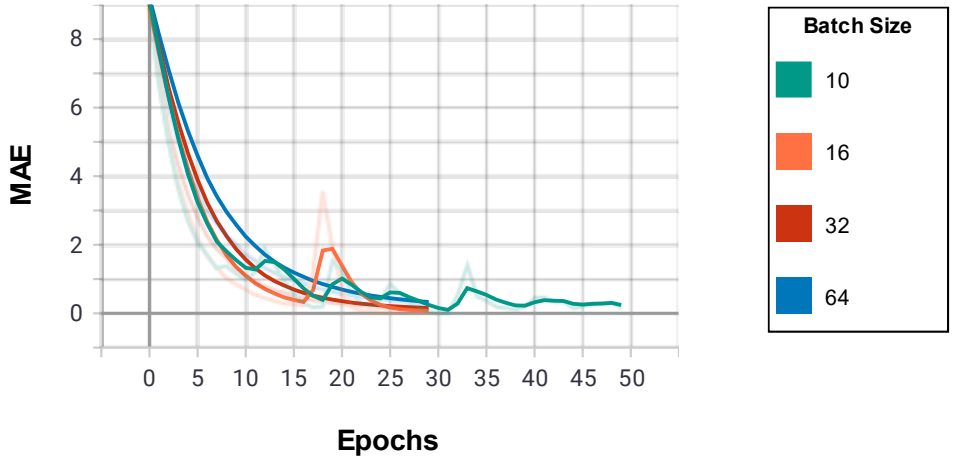
Experiments were performed on two machines. Initial experiments on hyperparameter adjustment utilised a Nvidia GTX 2070 Super with 8GB of VRAM. Additionally, the bulk of the data pre-processing, cleaning, and organisation work was performed on this machine. All subsequent experimentation took advantage of the Bede (NICE) HPC [54] whereby jobs were run on nodes containing 4 Nvidia Tesla V100 data centre GPUs, each with 32GB of VRAM.

4.2 Hyperparameter Adjustment

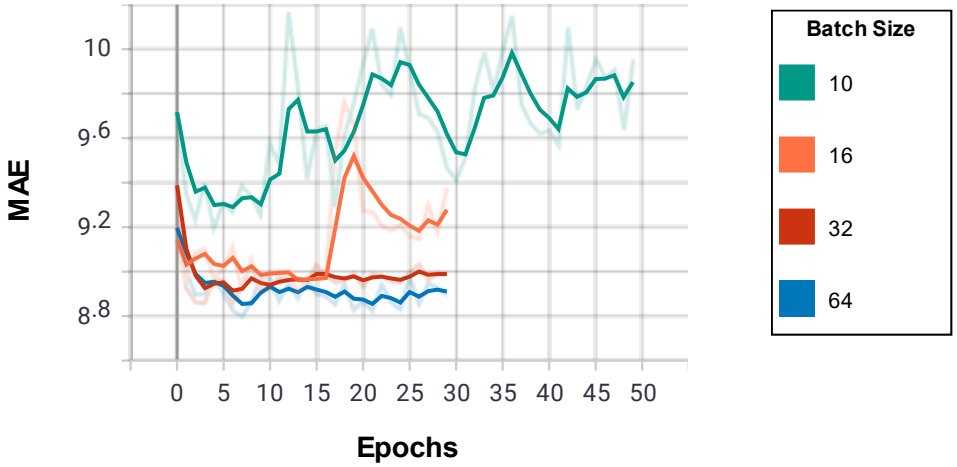
4.2.1 Batch Size

Below are the results for the adjustment of the batch size for the classification and regression models. Other hyperparameters were left at their default values, as defined by Keras.

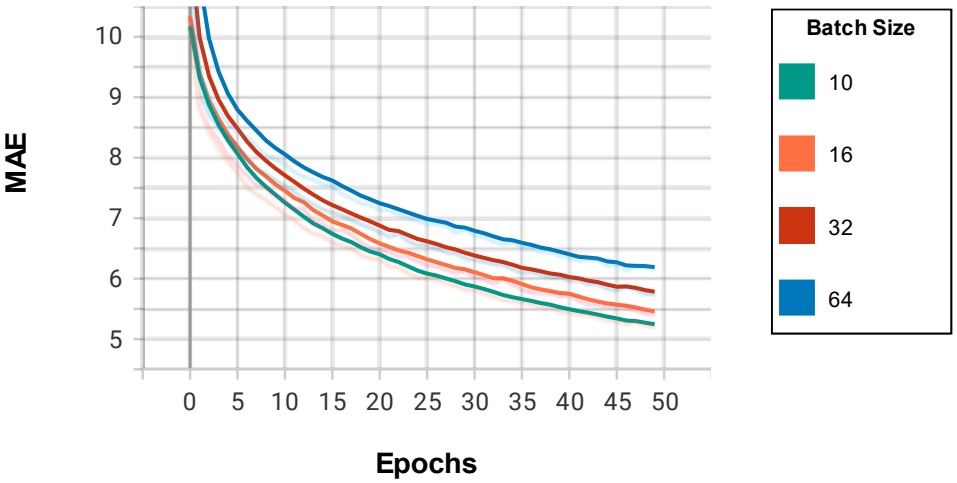
Classification Training



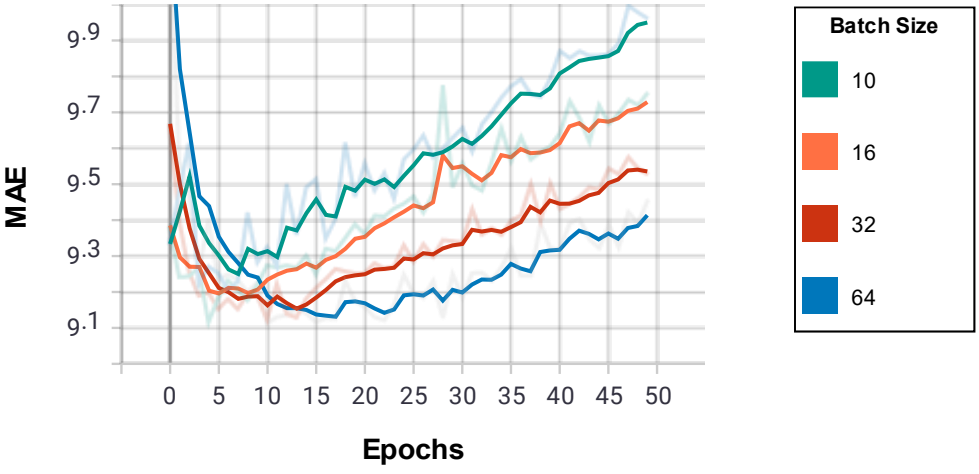
Classification Validation



Regression Training



Regression Validation



Evaluation: These results show that loading images in larger batches per epoch reduces the error rate while also reducing over-fitting. The batch size of 64 is the best value for the hyperparameter for both models.

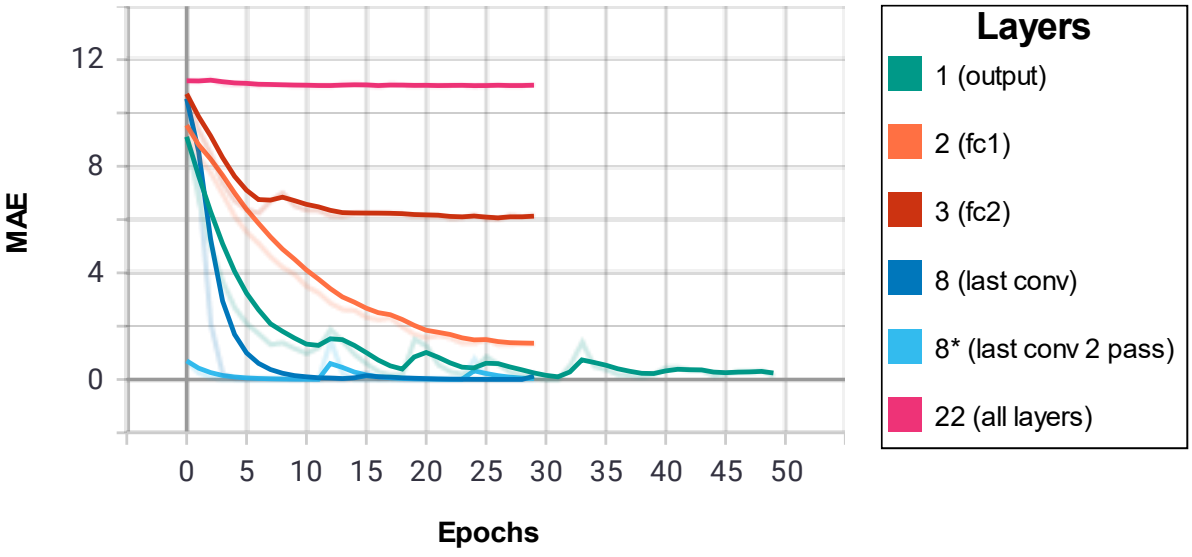
4.2.2 Trainable Layers

In a neural network, the layer weights are updated after each epoch using the back-propagation algorithm. However, some of these layers can be set not to be trainable or 'frozen', so the weights are not updated. This series of experiments explores the trainable layers. For some experiments, one training cycle was completed with only the last layer trainable and then another cycle with more layers trainable and a low learning rate of 0.00001. I have indicated this approach by using the * in the legend.

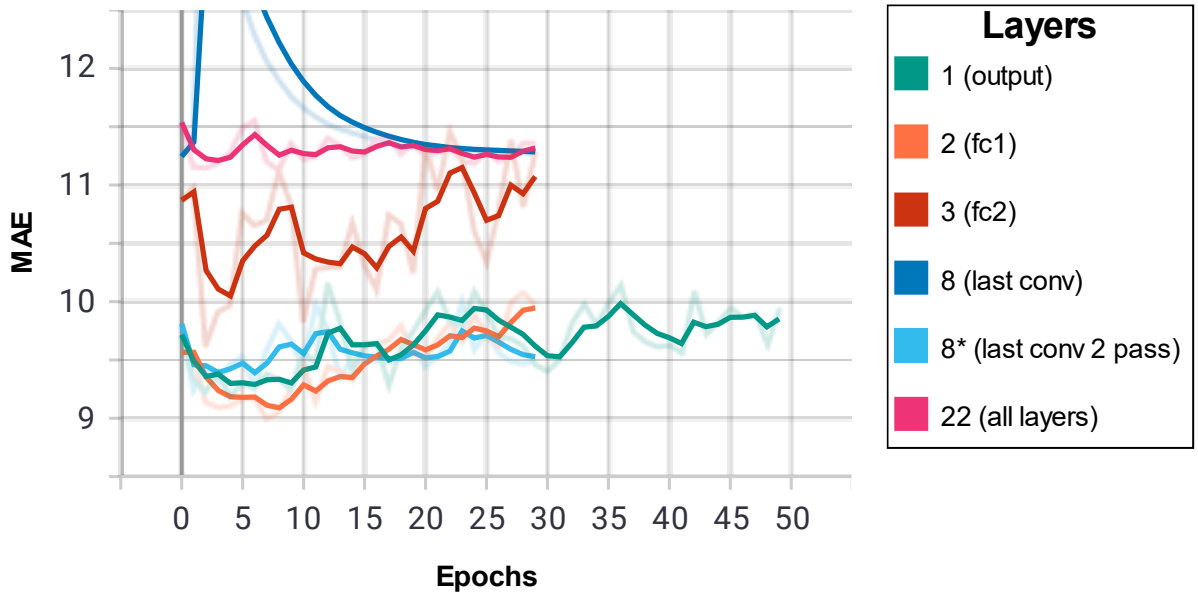
Below are the results for the adjustment of how many layers are trainable for both models. The table below indicates the type of layers that are being modified in the chosen *VGG16* architecture.

Trainable Layer Value	**Affected Layers**
1	SoftMax Classification Dense Layer / Single Neuron Regression Dense Layer
2	Fully Connected (fc) 4096 neurons
3	Fully Connected (fc 2) 4096 neurons
8	Final Conv Block - Block5 conv1, conv2, conv3
22	All layers of VGG16

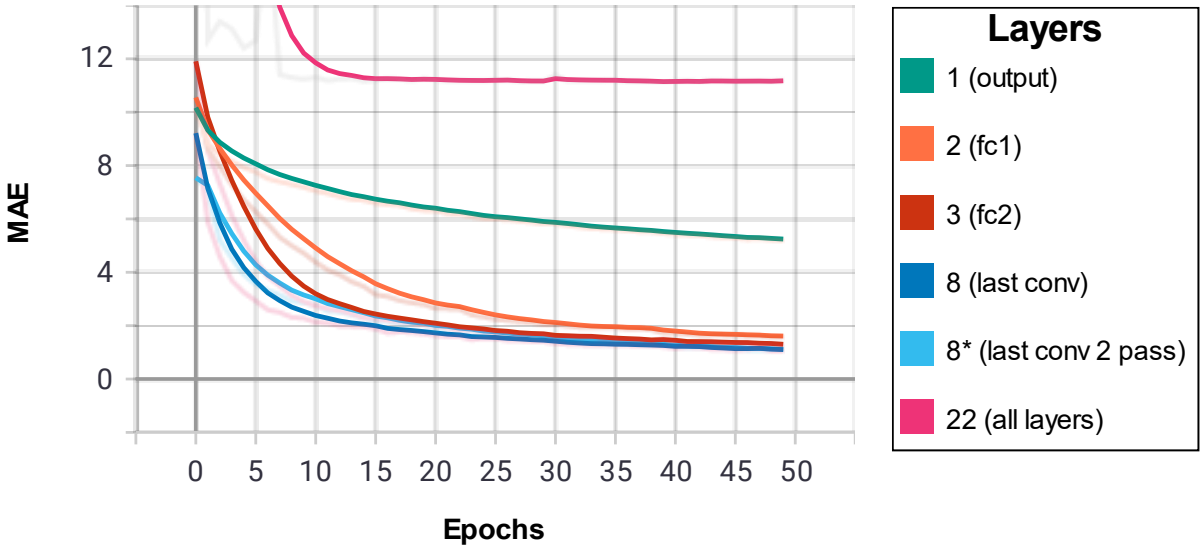
Classification Training



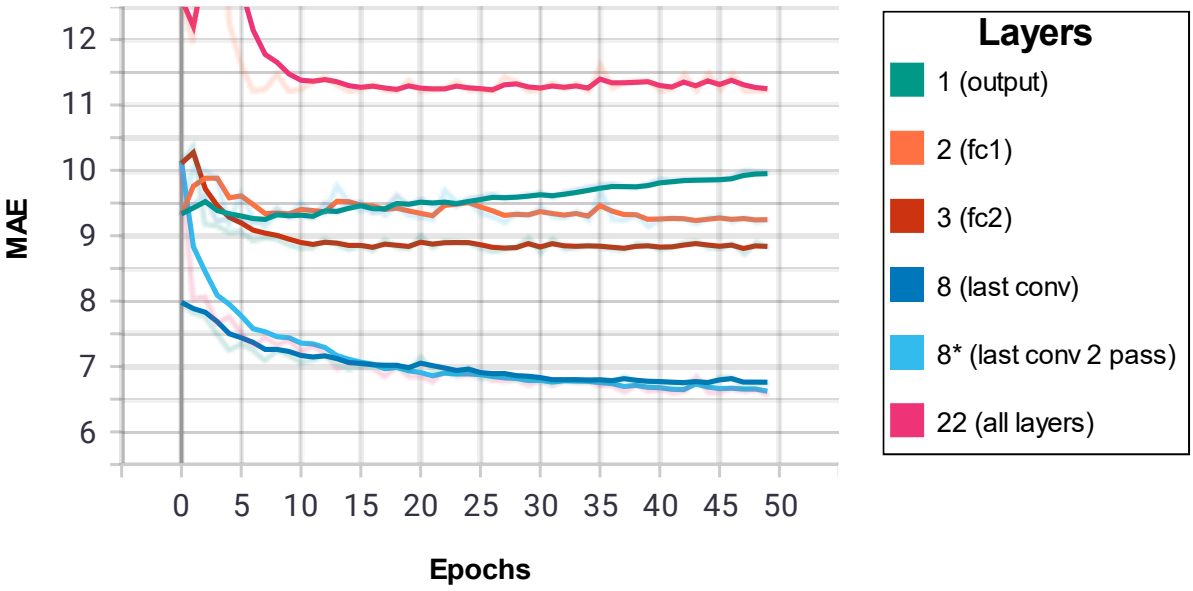
Classification Validation



Regression Training



Regression Validation



Evaluation: For classification, it initially appears that leaving more layer's trainable nets a performance loss. However, this is only when leaving a greater number of layers trainable with the default learning rate of 0.01 for the Adam optimiser in Keras. The best result is achieved when leaving the last convolutional block trainable using the two-cycle approach as indicated by the *.

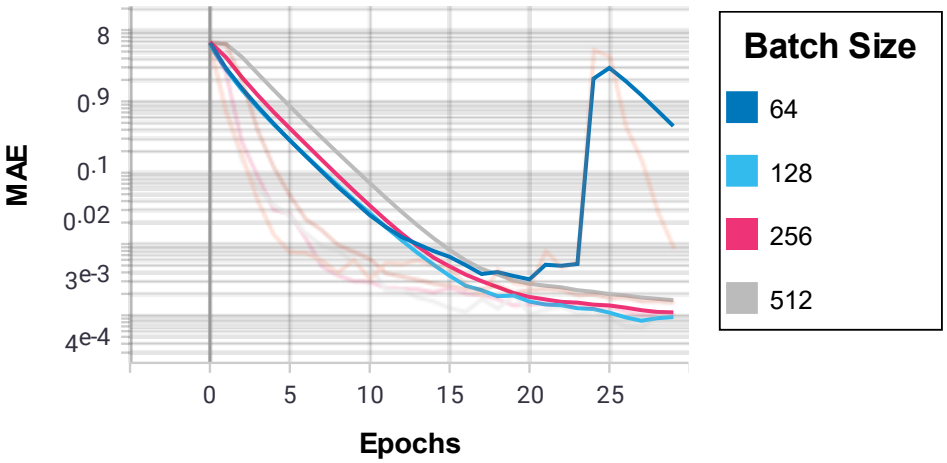
For the regression model, leaving the last convolutional block trainable again results in a performance increase. Unlike the classification model, this is also the case with both the default learning rate and the reduced learning rate in the two-cycle approach. The worst results in both models occur with the default training rate and leaving all layers in the architecture unfrozen. This is likely due to the ImageNet weight values being lost. The initial convolutional blocks will have distilled generic image information that applies to this problem. The best results occur when the last convolutional block is left trainable. This can be explained as the last block learns abstract features that will pertain to this input data.

4.2.3 Extended Batch Size and Learning Rate

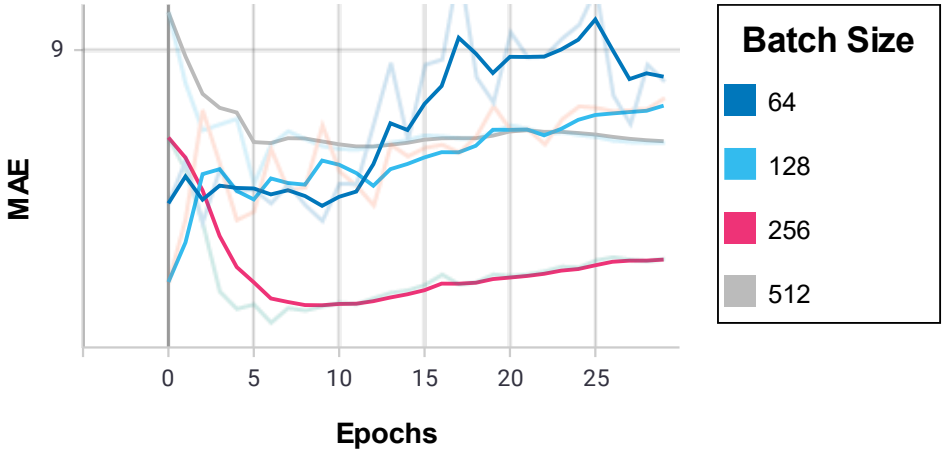
This next set of results focused on observing the impact of increasing the batch size to larger amounts, made possible by the additional VRAM of the HPC hardware. Additionally, the effect of the learning rate hyperparameter is shown. The batch size experiments leave the learning rate fixed at 0.0001, while for the learning rate experiments, the batch size was fixed at the best values found.

Extended Batch Size

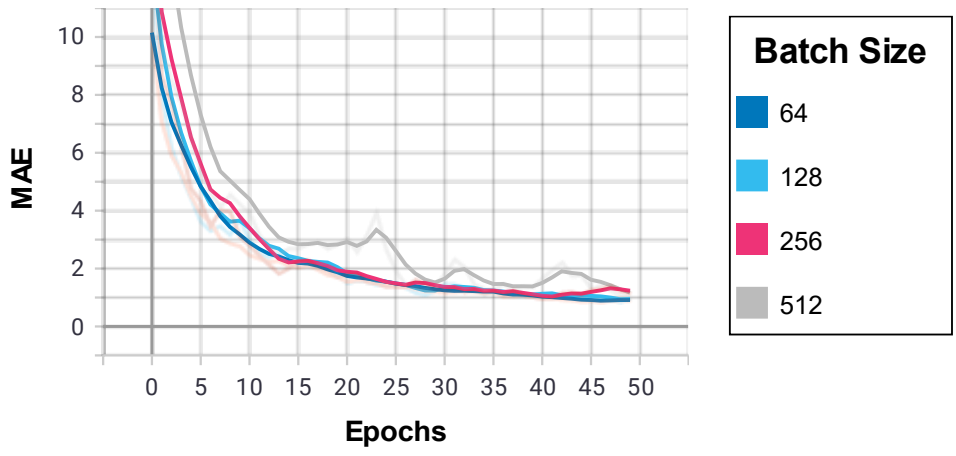
Classification Training



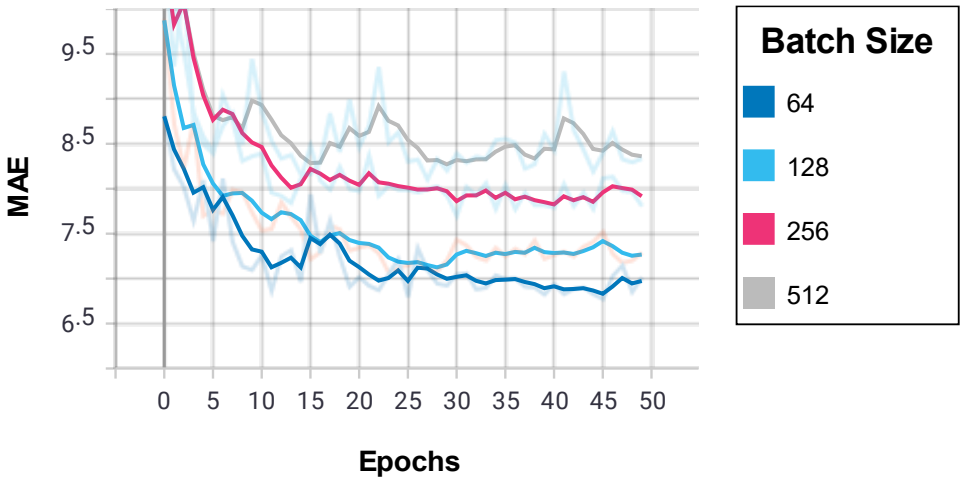
Classification Validation



Regression Training



Regression Validation

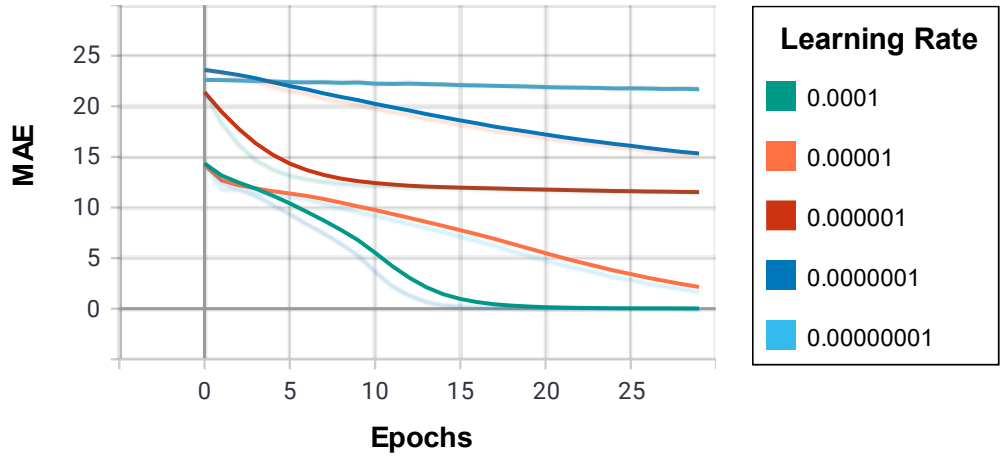


Evaluation: The regression model benefits the most with the smaller batch size of **64** over the total epochs. The classification model is most performant at the larger **256** batch size. It is also notable that the values decrease and then stabilise for regression whereby batches of 64 and 128 show rapid performance decrease after ten epochs for classification.

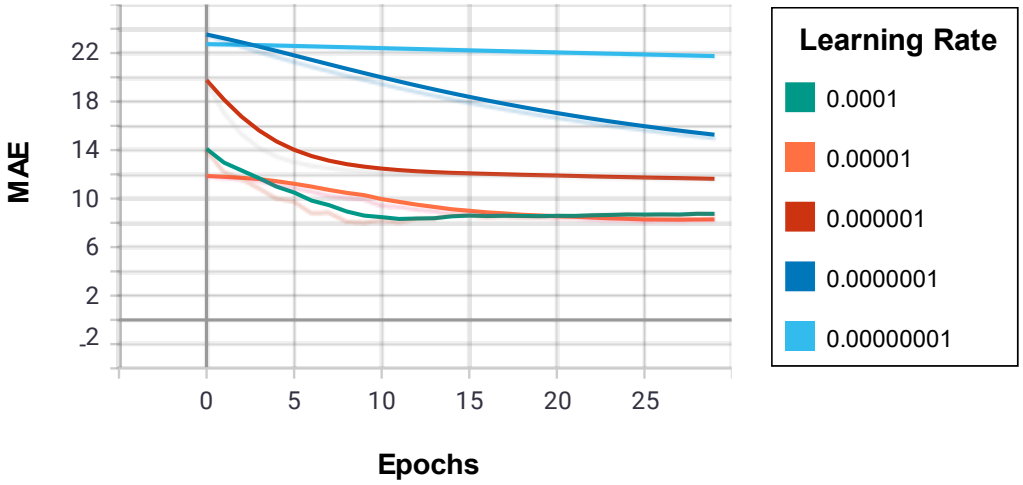
Extended Learning Rate

The classification results were combined with a batch size of 256 whilst the regression model used a batch size of 64.

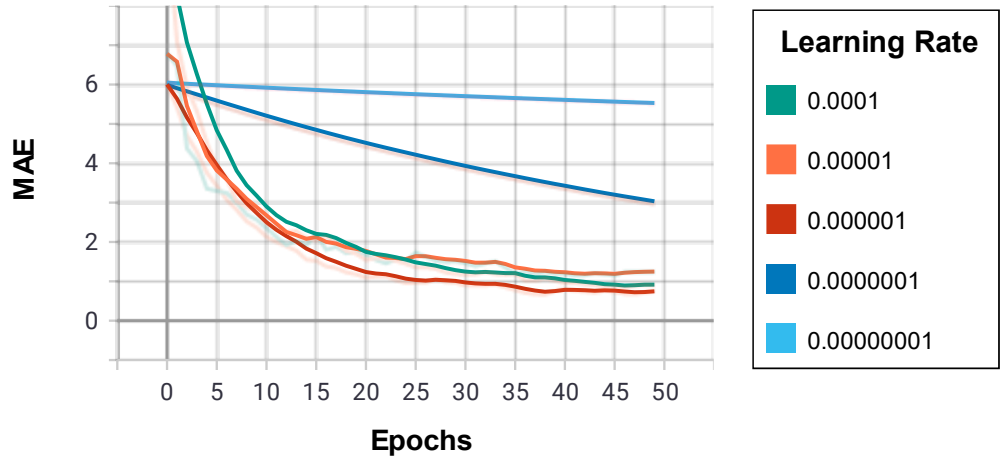
Classification Training



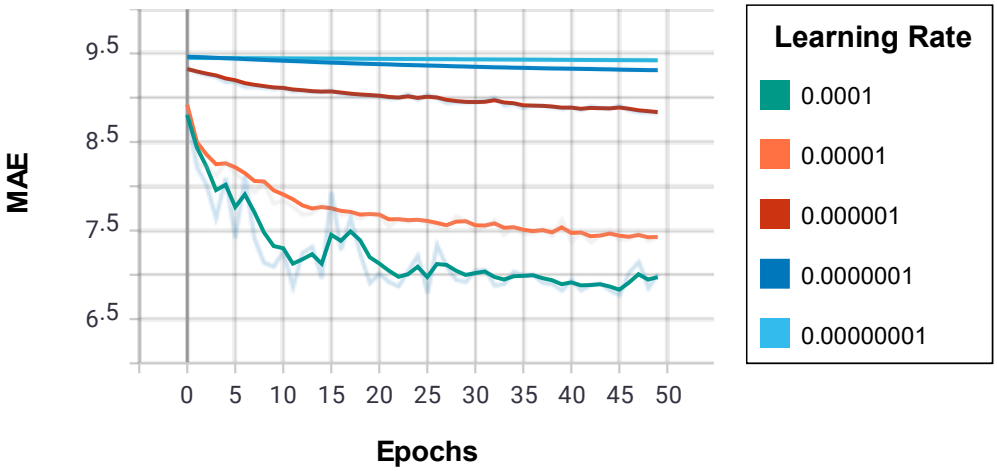
Classification Validation



Regression Training



Regression Validation



Evaluation: The best learning rate found was 0.0001 for both models. Gradually, results got worse as 'slower' learning rates were applied. Larger increments need to be taken to progress learning at each epoch.

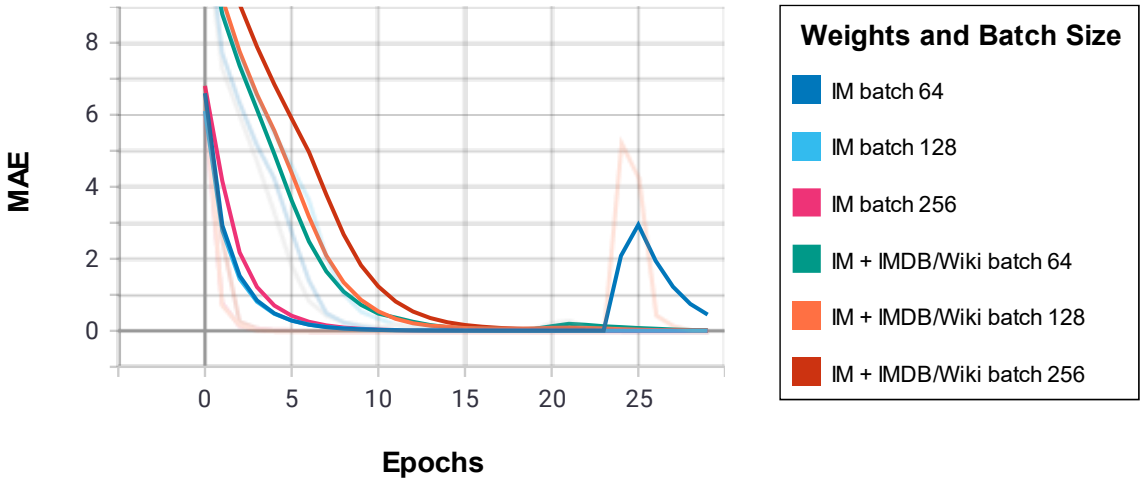
4.3 Transfer Learning and Pre-Training

These tests were performed with a learning rate of 0.0001 as this was the best result in the prior experiments. Additionally, the last convolutional block was left trainable as this also had a positive impact. The IMDb Wiki pre-training was performing with a batch size of 512.

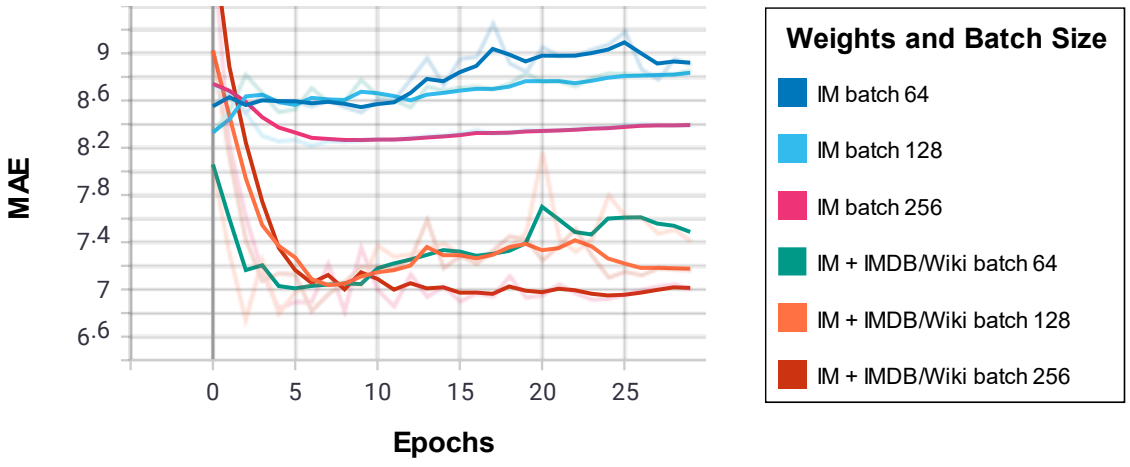
4.3.1 ImageNet (IM) + IMDb Wiki

Impact of Pre-Training

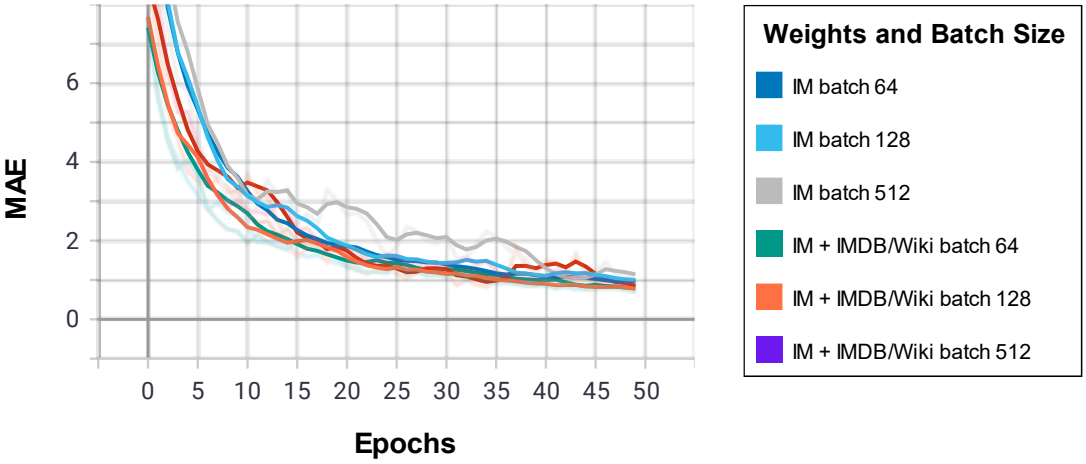
Classification Training



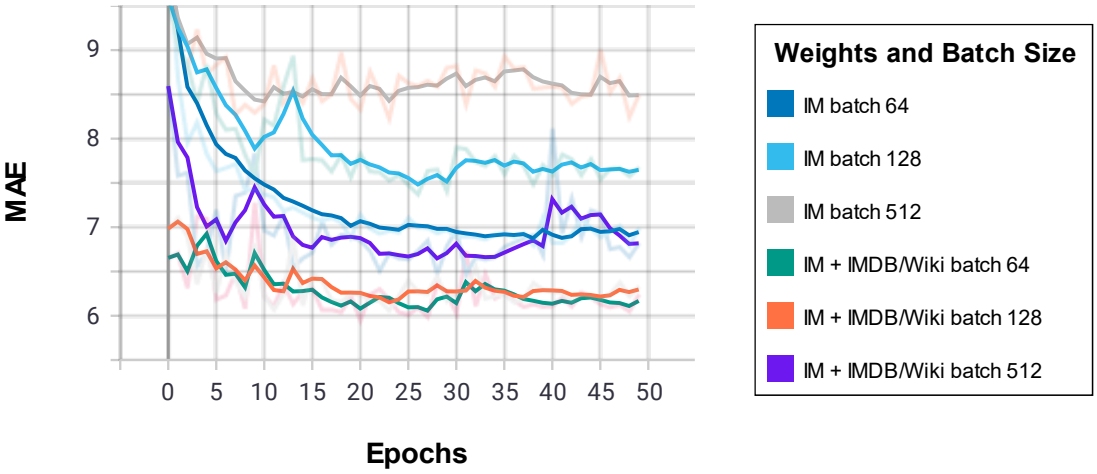
Classification Validation



Regression Training



Regression Validation



Evaluation: For both models pre-training on IMDb-Wiki and then fine-tuning on ChaLearn nets ~ 1-year decrease in the MAE across the validation set.

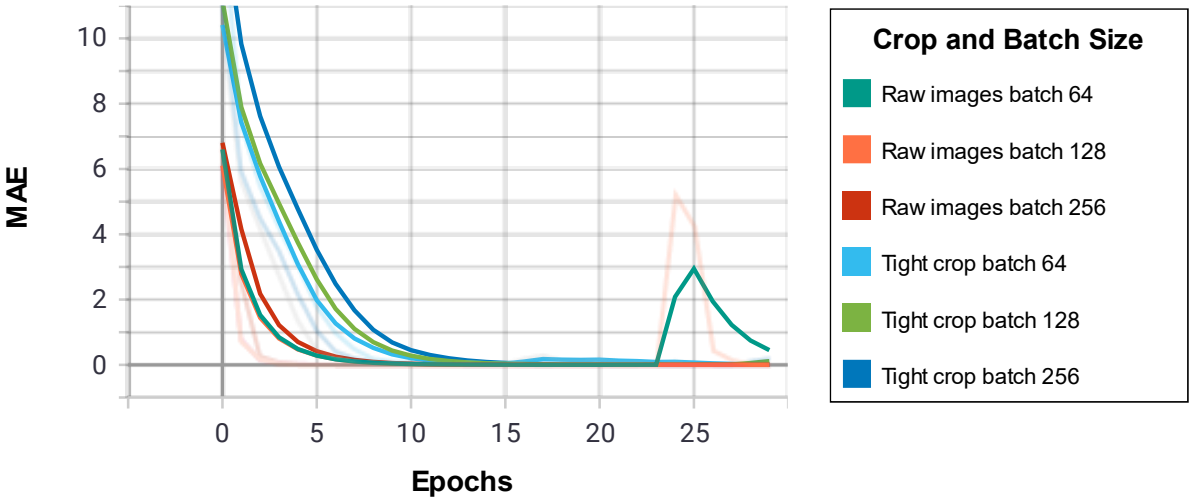
4.4 Image Manipulation

This next set of results are from the various manipulations of the images that were performed. This round of experiments shows the results of tightly cropping to the face, then varying crop factors, and finally facial alignment. The previously selected hyperparameters and layer adjustments were used in this round. It should also be noted that the manipulations on IMDb-Wiki were the same as on ChaLearn.

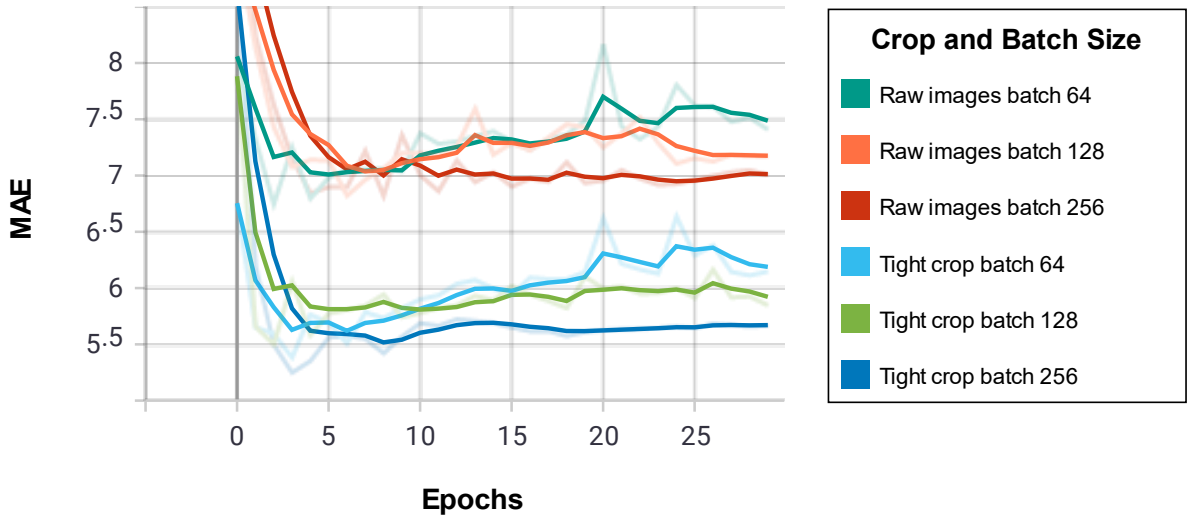
4.4.1 Tight image cropping

In the legend, "raw" refers to the image directly from the dataset without cropping. The tight crop was displayed in **Figure 18.**

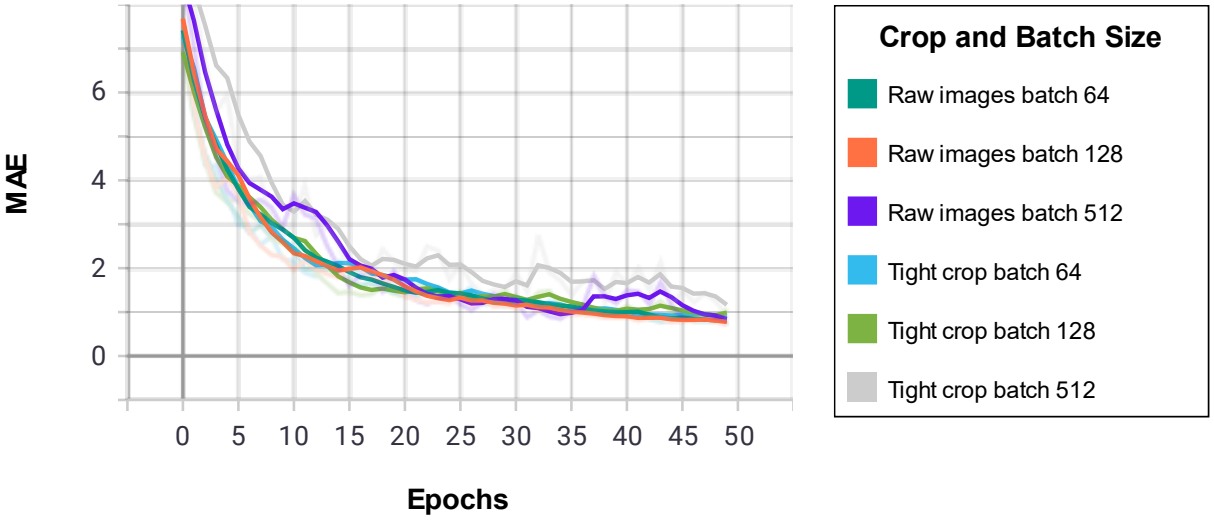
Classification Training



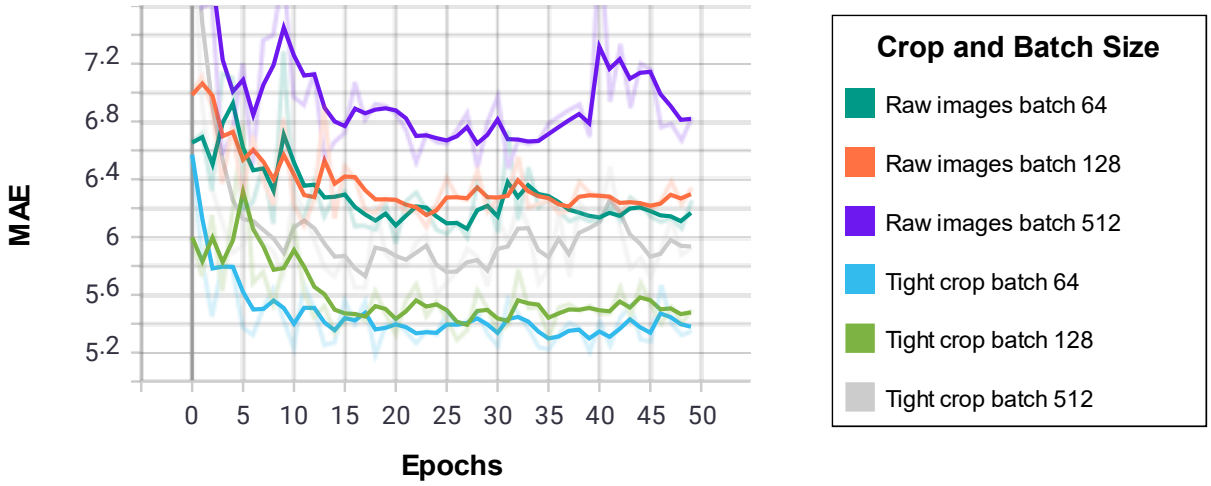
Classification Validation



Regression Training



Regression Validation



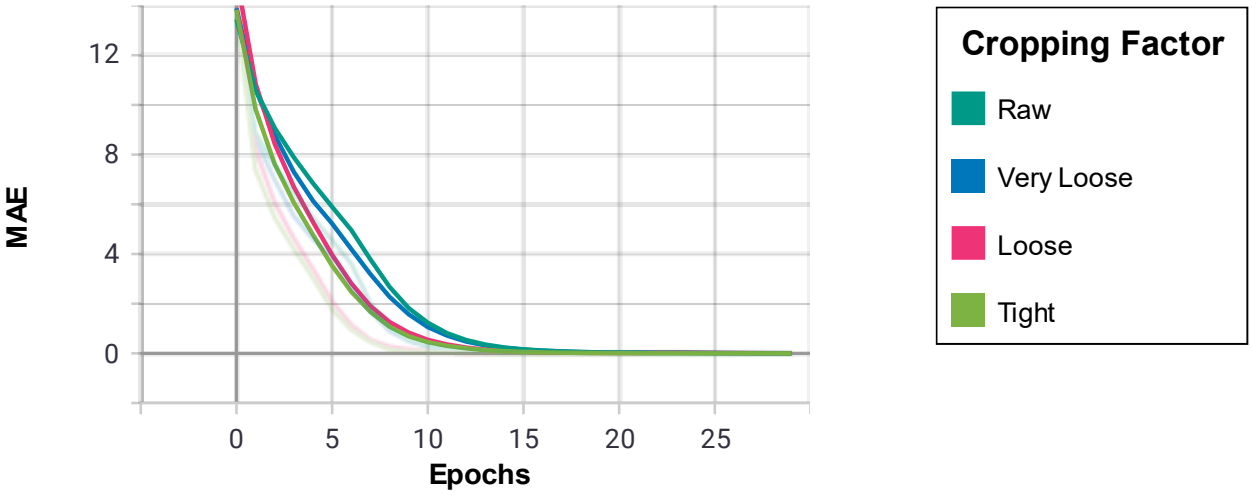
Evaluation: Both models benefited from tight cropping compared with training simply on the dataset's raw images. The regression model achieved the lowest MAE overall. Still, the classification model displayed the largest improvement with a 1.5 drop in the error rate in the validation set for the best results with/without cropping. As these experiments utilised a range of batch sizes, this round of experimentation supported the previous batch test experiments as batch sizes of 64 and 256 performed the best for the regression and classification models, respectively.

4.4.2 Testing additional cropping factors

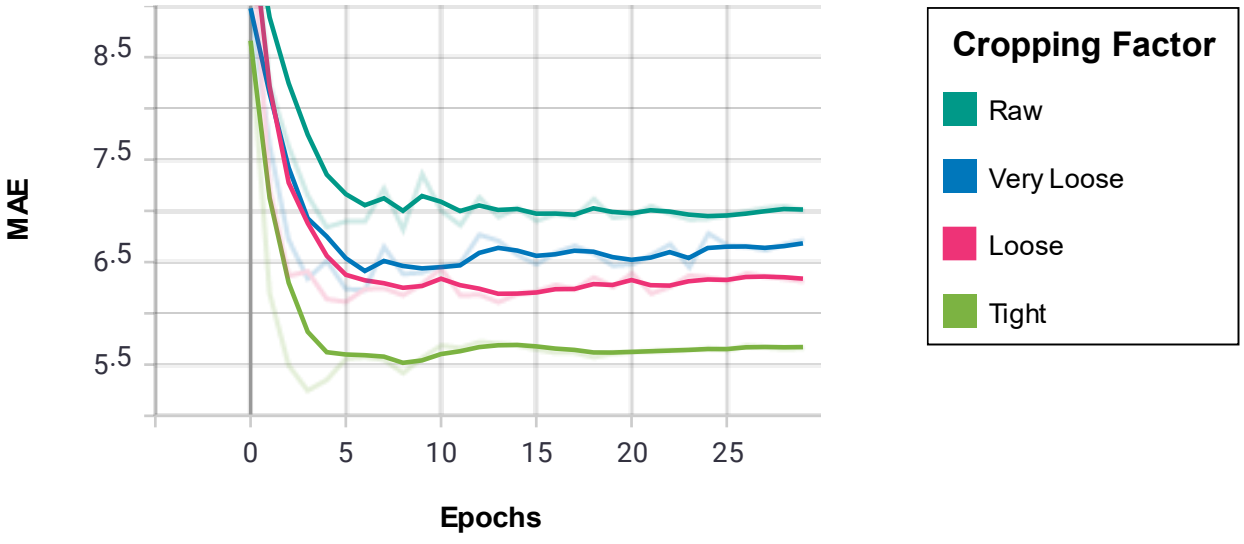
This series of results follows the experiments testing the cropping factors, explained in the table below. The experiments were performed using the optimal value for the batch size and learning rate, as discovered through previous experiments. Again, pre-training on IMDB-Wiki was used.

Crop Type	Explanation
Raw	The 'wild' image direct from the dataset, with no cropping, applied.
Very Loose	A crop containing the full face as well as the clothing detail of the subject combined with some background in the scene.
Loose	A crop containing the full face of the subject as well as all the subject's hair and neck detail.
Tight	A very tight crop to the face. Some, but not all, hair detail will be present, and the shape of the subject's face will be present.

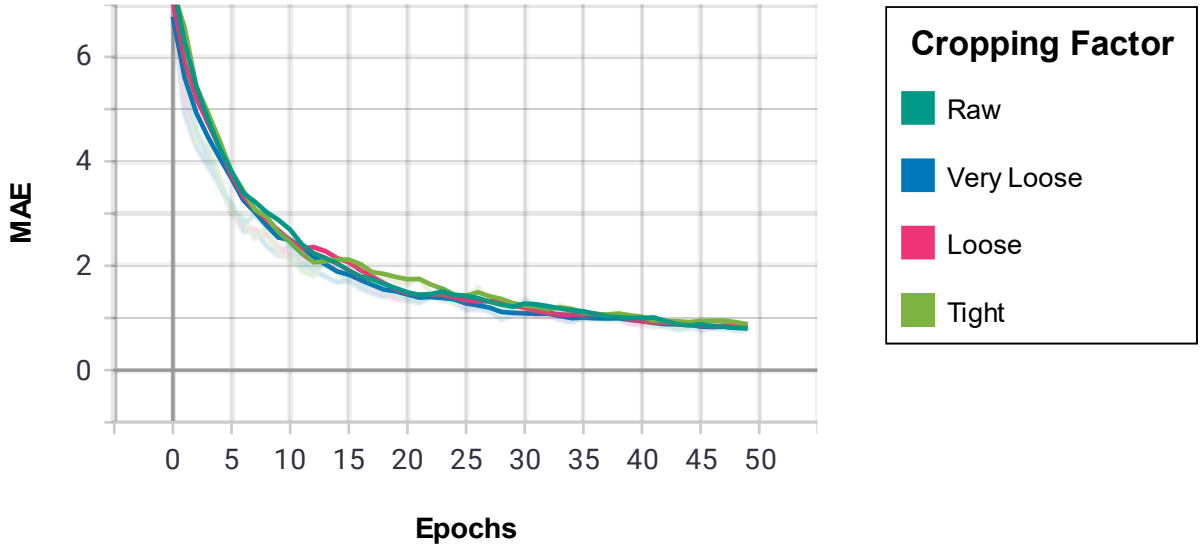
Classification Training



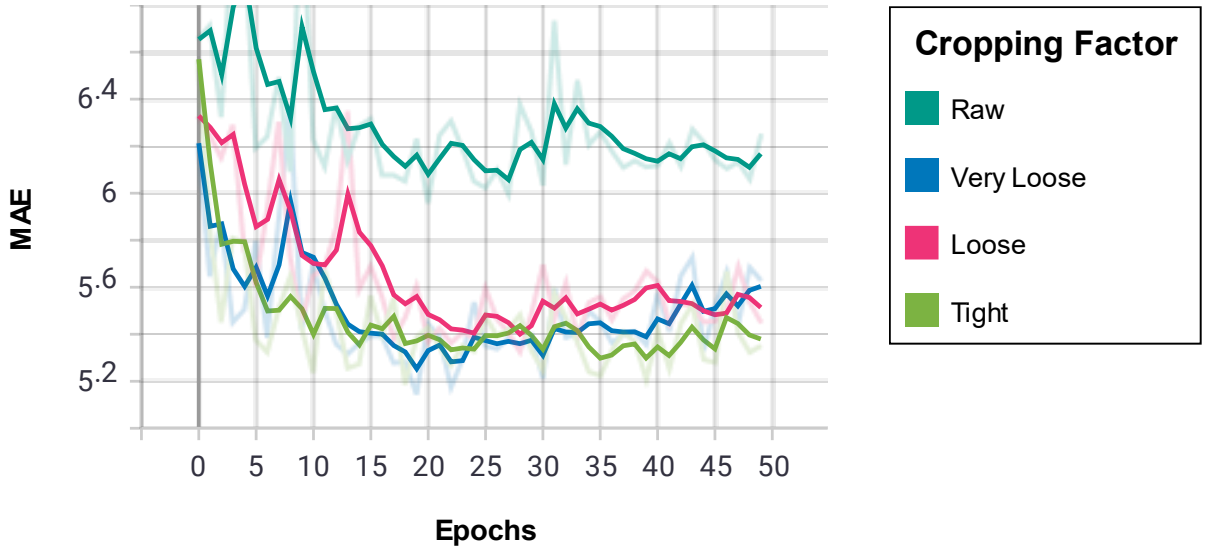
Classification Validation



Regression Training



Regression Validation

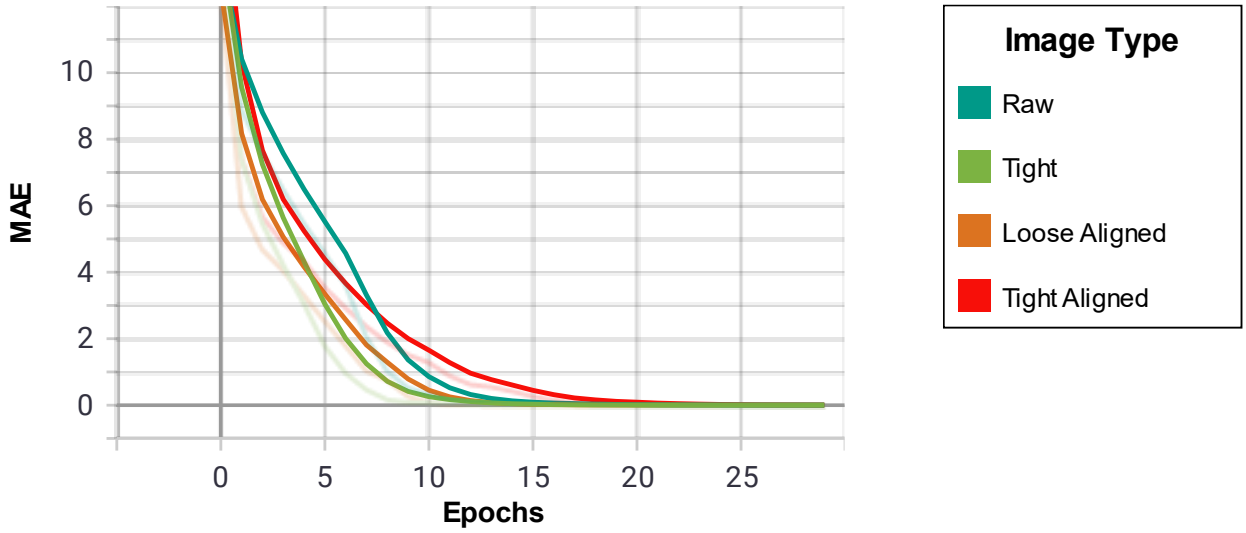


Evaluation: In the classification model, there is a clear link between the tightness of the crop and the performance. As the crop becomes tighter, the results improve, with the largest difference between "loose" and "tight" crops. The regression model differs. Whilst the raw images do perform worse than all the crop factors. The various crops exhibit very comparable performance.

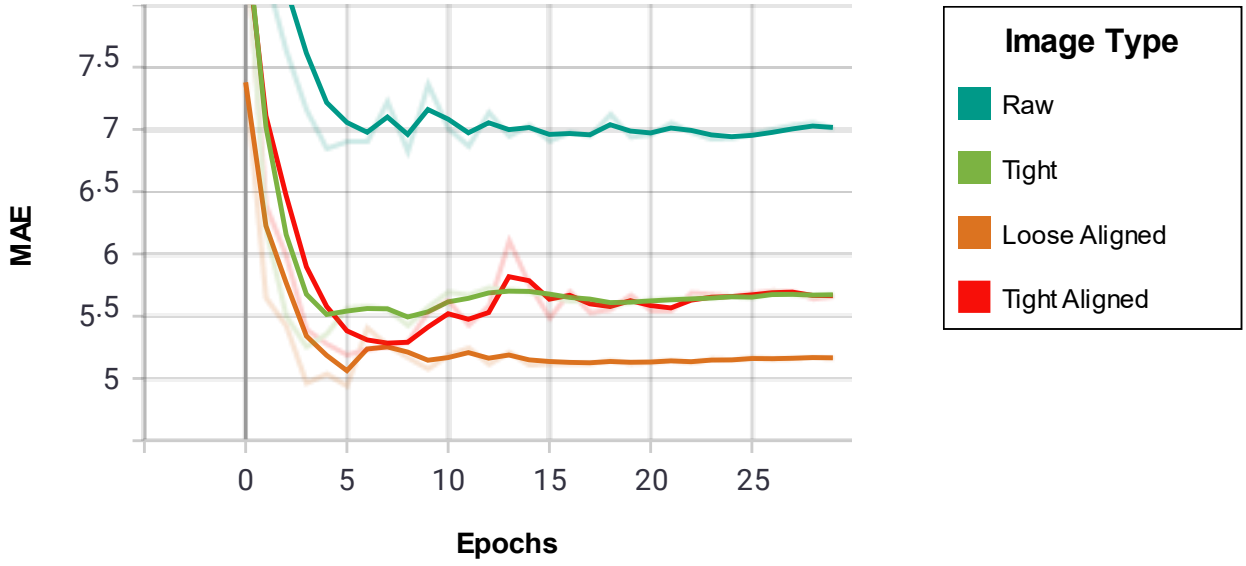
4.4.3 Face alignment

Below are the results from performing the facial alignment, with "tight" and "loose" alignments as depicted in **Figure 19.** Pre-training was performed, and these results again are compiled with the best hyperparameters and configurations found previously.

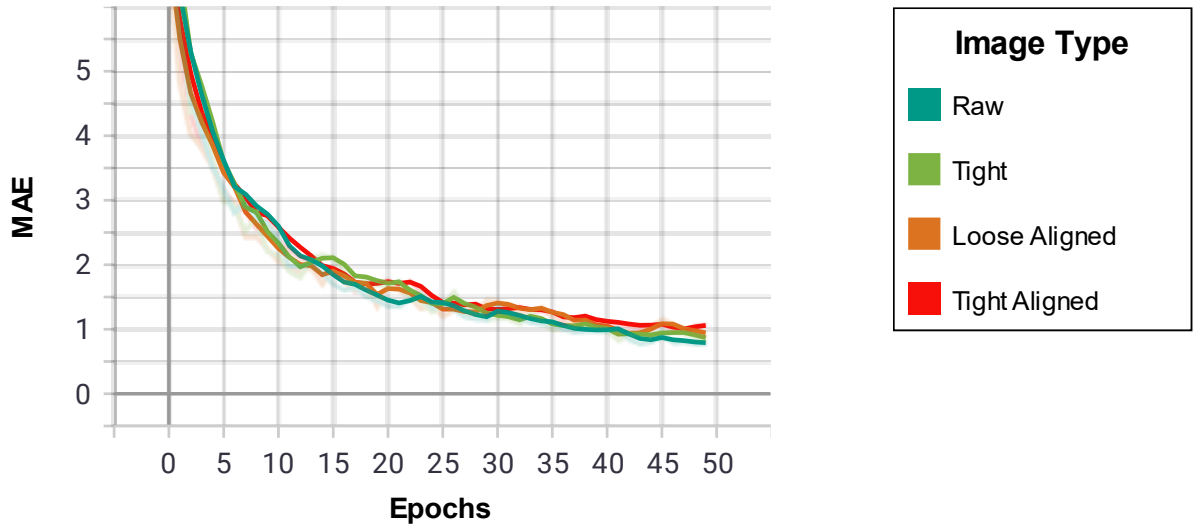
Classification Training



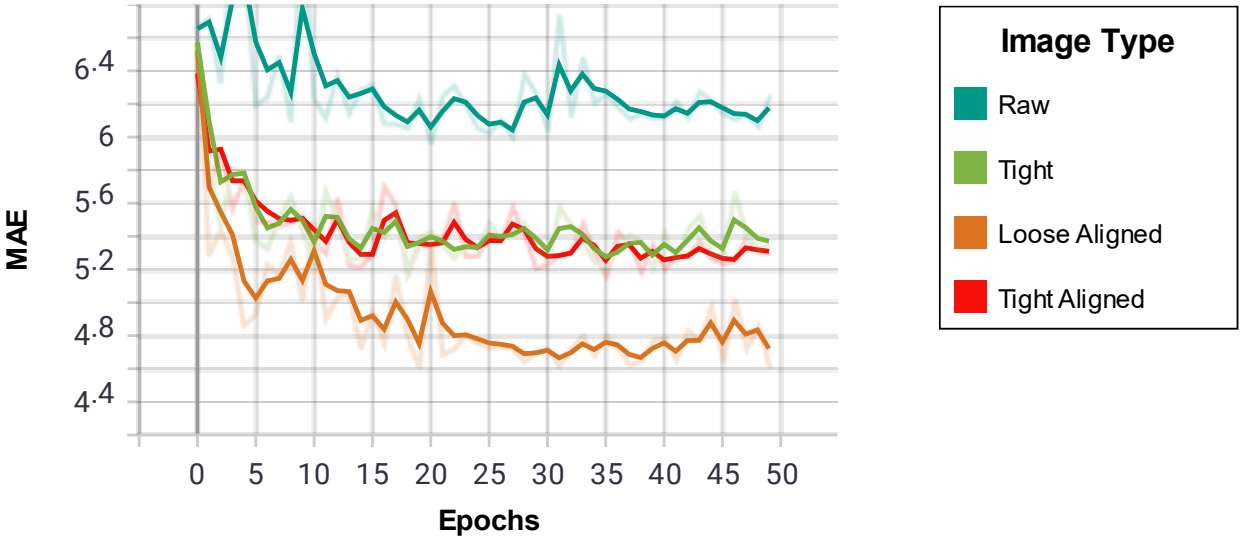
Classification Validation



Regression Training



Regression Validation

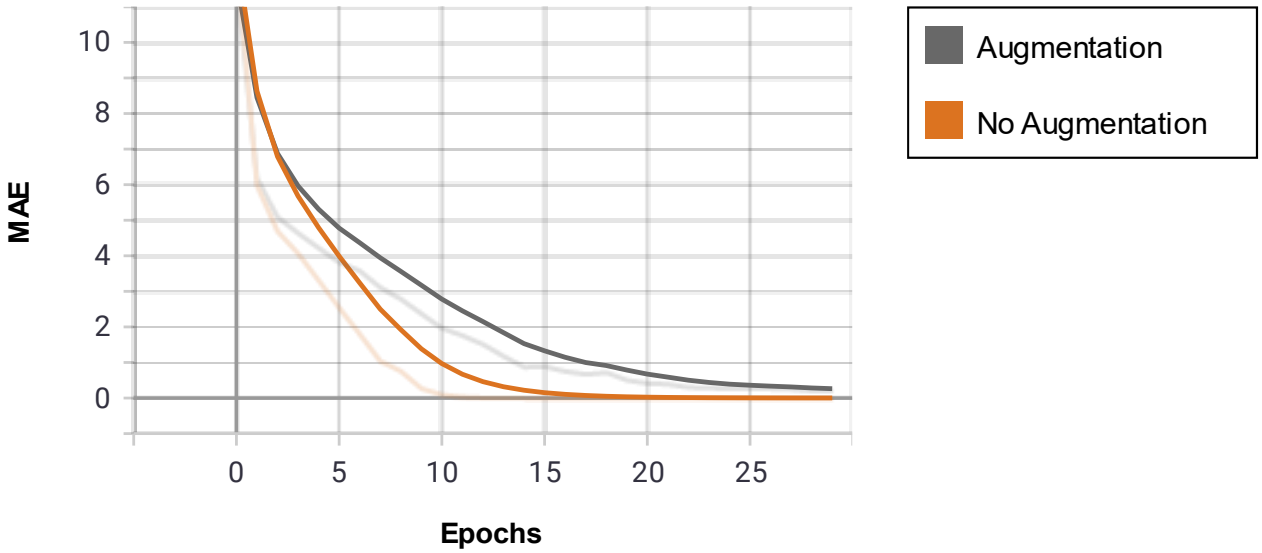


Evaluation: These results show that both models benefit from alignment, although aligning the tightly cropped images only results in a negligible performance gain. The loose alignment is effective, and on the regression model, this results in a consistent MAE below five years, which is a considerable improvement.

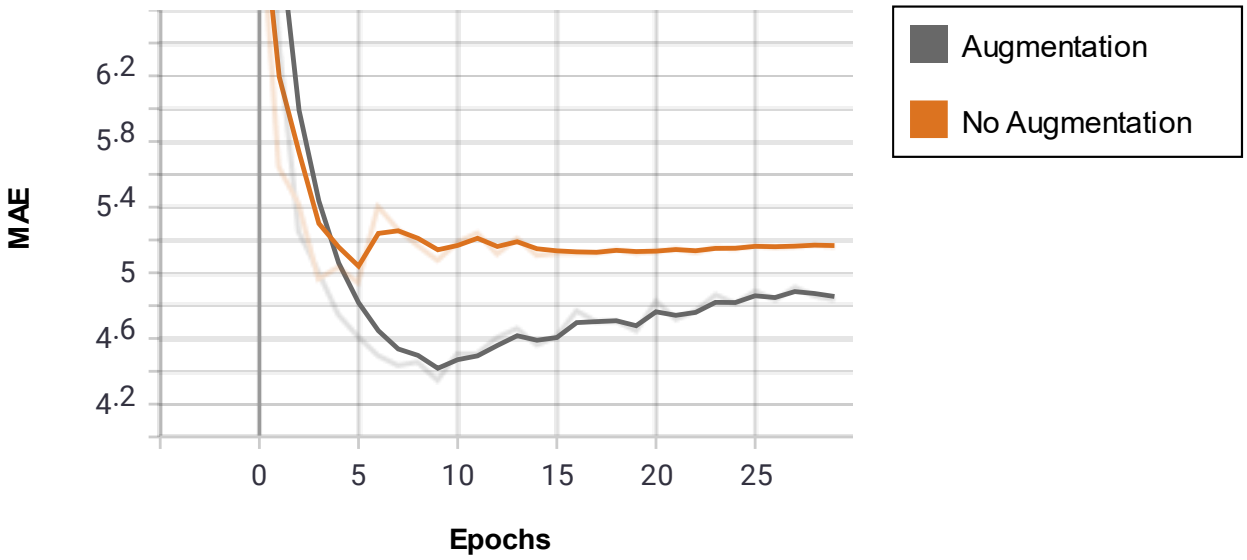
4.4.4 Data Augmentation

The results for the experiments involving data augmentation are below. It should be noted that these results only present the best augmentation configuration found, as referenced in implementation **section 3.5.4.3**. Augmentation was applied with the aligned images as the original data, and the best hyperparameter configuration found previously.

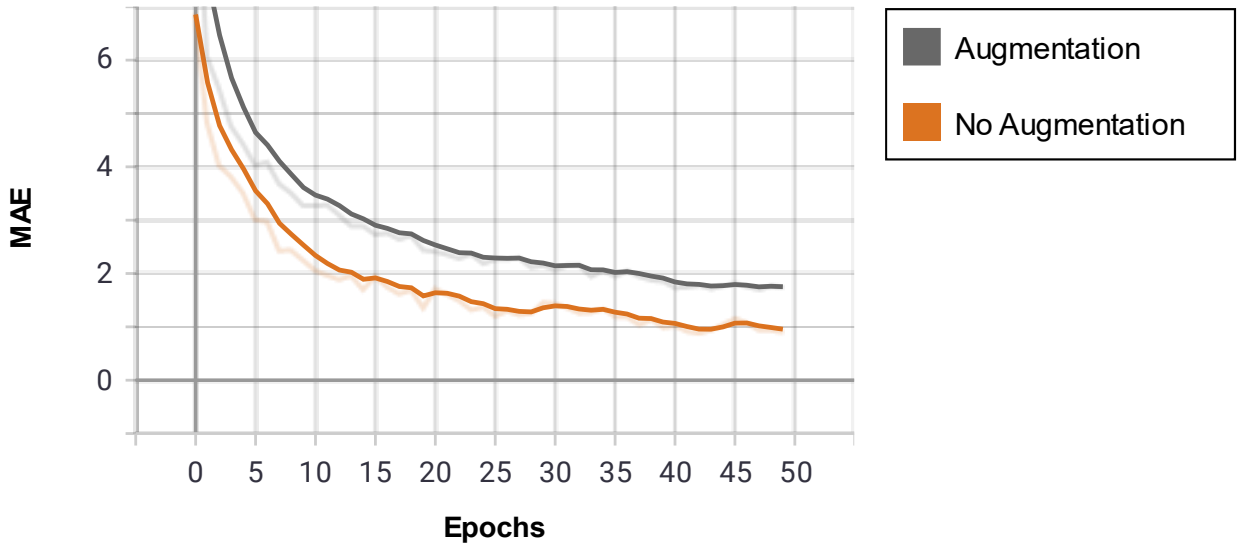
Classification Training



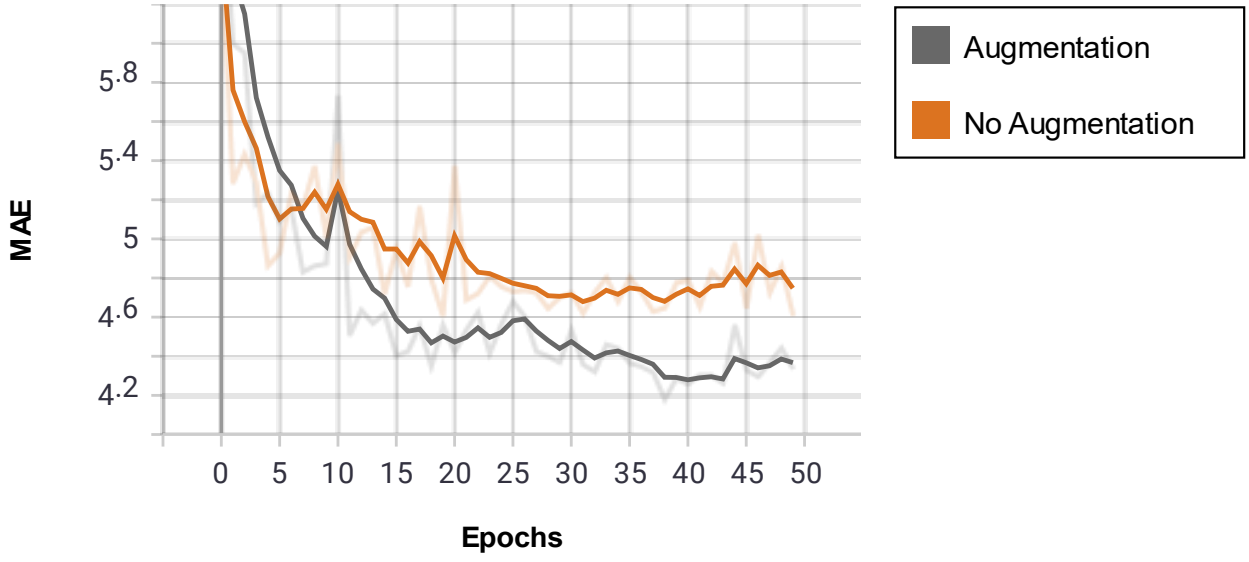
Classification Validation



Regression Training



Regression Validation



Evaluation: Although data augmentation introduced significant penalties in terms of execution time, the performance benefits are clear. On the classification model, adding data augmentation brings the MAE under 5 for the first time. It should also be noted that data augmentation slows down performance on the training set, but this can be explained as there is simply a greater volume of training data for the model to interpret.

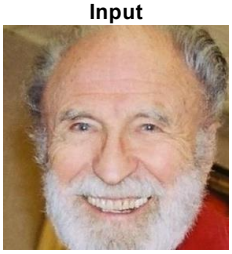
4.5 Visualisation

The image visualisation techniques shown below were performed after loading the finalised classification model. All the images shown are samples from the ChaLearn test set that were not seen during training/validation.

4.5.1 Activation Visualisation

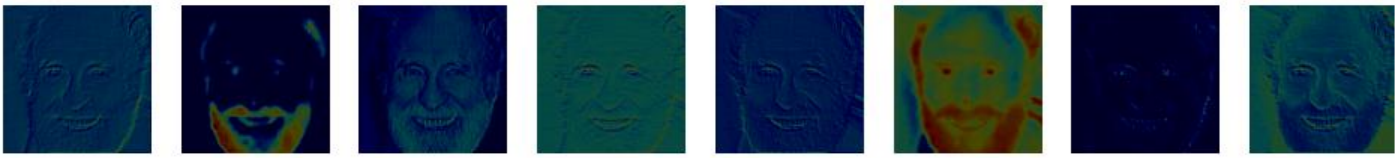
Each of the plotted feature maps uses heatmap overlays over the input image to visualise the activations.

Input



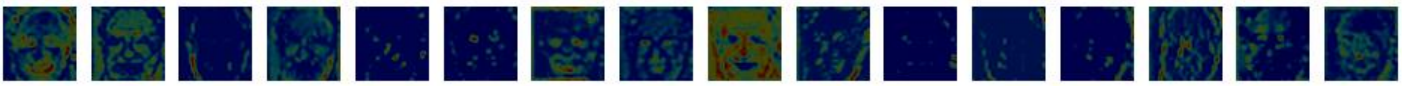
70 / 69.4
Predicted / Ground Truth

Conv Block 1 Pool

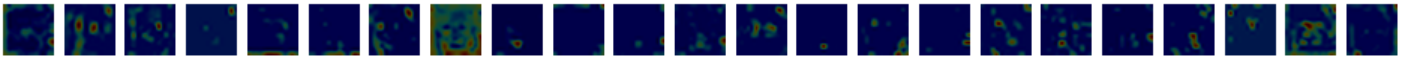


Conv Block 2 Pool

Conv Block 3 Pool

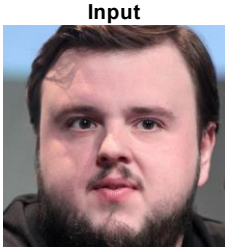


Conv Block 4 Pool



Conv Block 5 Pool

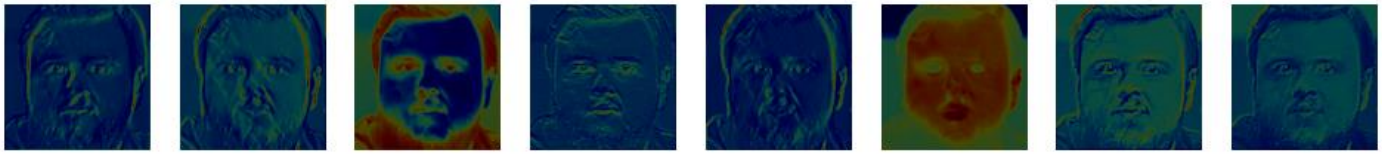
Input



29 / 30.2

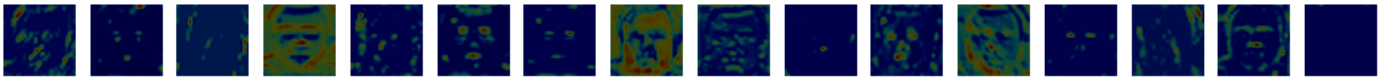
Predicted / Ground Truth

Conv Block 1 Pool

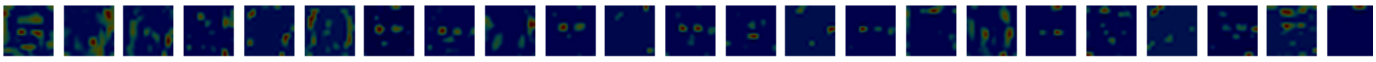


Conv Block 2 Pool

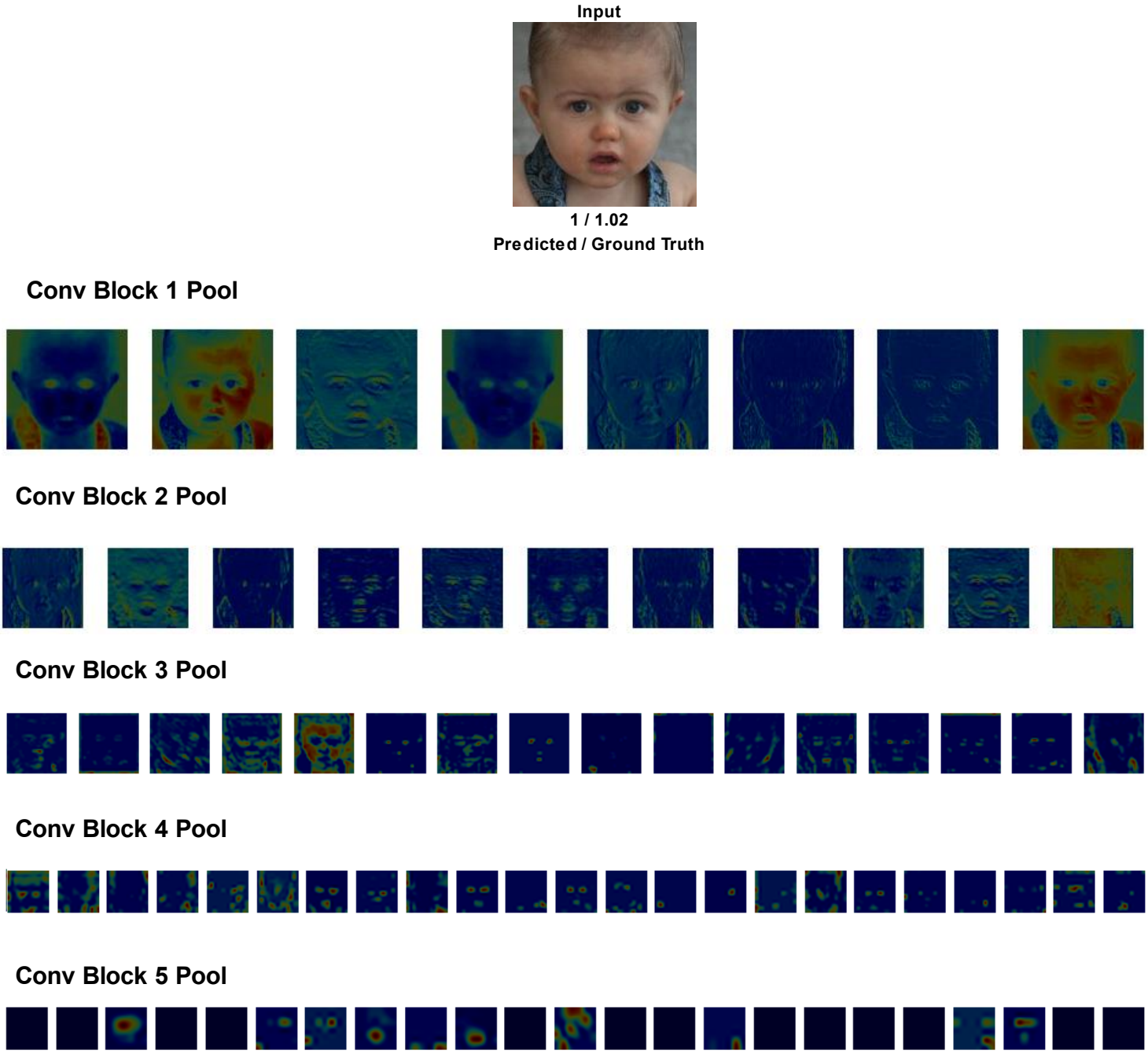
Conv Block 3 Pool



Conv Block 4 Pool



Conv Block 5 Pool



Evaluation: The activation visualisations on the three images shown above are fascinating, displaying progressive abstraction of the input and resolution reduction caused by the decreasing filter size for each convolutional block and subsequent pooling layers.

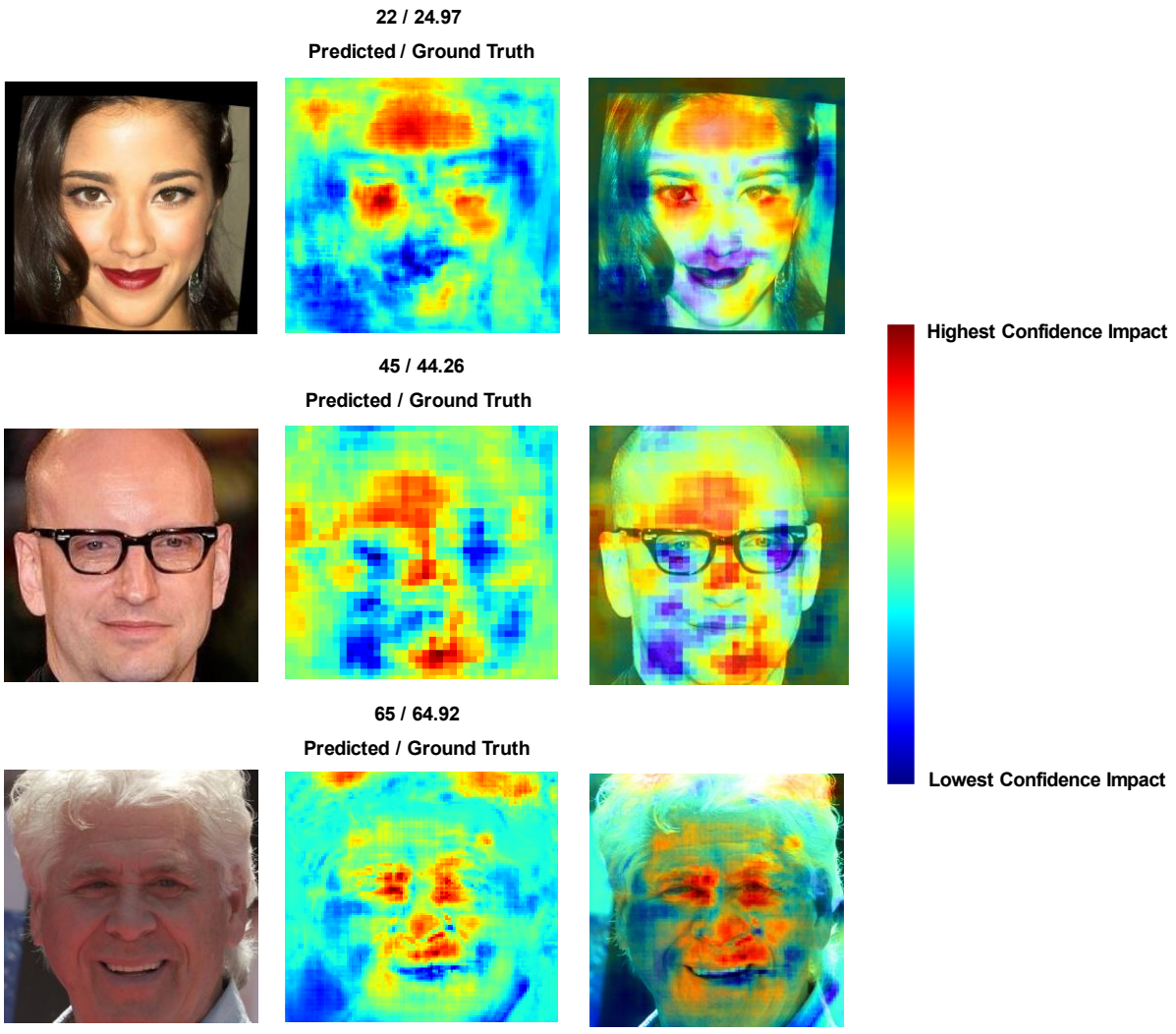
The edges outlining the face and the facial features are displayed in convolutional layers one and two. The third and fourth layers then feature the eyes heavily while also strongly activating on the facial and head hair in the first two images. As expected, the final block is very abstract, with some feature maps displaying no information. These filters predominantly activate on the

face only and are likely encoding precise features pertaining to facial shape, particularly around the eyes.

Breaking these results down further, the facial features that each individual filter specialises in can be seen. The second and sixth filter in block 1 for the first image clearly activates on the beard. The third filter in block 1 of the second image identifies the hair and eyes and some clothing detail, the sixth filter in this block does the inverse identifying the shape of the face. In block 1 of the third image, the second filter particularly activates on the right side of the face with the third filter isolating all the facial features.

4.5.2 Occlusion Sensitivity

Occlusion sensitivity was applied on the images below, using the probability of the expected class from the predictions to calculate the heatmap intensity for each patch.



Evaluation: All three images clearly show that the network is correctly using features in the face area to influence prediction confidence. For instance, the lower left corner of the bottom image is a deep blue, indicating that this background region has very minimal impact on the age prediction. This can also be seen in the top image, which shows the prediction accuracy is relatively unaffected when the black pixel borders created by alignment are hidden.

Interestingly, the forehead appears to be an area which is important for prediction confidence for all three images. The network could be focusing on skin texture and wrinkles or lack thereof. The youngest subject has very smooth skin, exacerbated with makeup. Makeup may influence perceived age prediction by the network in a similar way to how it influences perceived age prediction by humans. The elder subjects have wrinkles in the skin in this region, which is particularly focused on in the bottom image. The bottom image also shows portions of the subject's grey hair greatly impacting confidence when occluded. Finally, the eyes and the facial shape also appear to be important features, that when occluded drop accuracy relative to the expected age class.

In conclusion, occlusion sensitivity gives insight into what impacts predictions for different ages and gives a degree of explanation as to *why* the network makes each age prediction through showing the important image regions. Conversely, activation visualisation does not explain the decision process, but it does show *how* the incrementally abstract representations of the faces are constructed inside the convolutional blocks.

4.6 Overall Evaluation

Overall, the series of experiments performed has shown that each component can impact performance for the objective task when tuned appropriately.

Hyperparameter tuning was effective, but the experiments leveraging transfer learning and manipulating the raw images, including train-time augmentation, proved the most impactful with drastic performance improvements. A two-year improvement in the MAE resulted from the image manipulations, when compared to the raw images and transfer learning on IMDb-Wiki, brought a 1.5-year MAE reduction.

The best model achieved a **MAE of 4.25** on the ChaLearn test set, consisting of 1,935 samples. The final model architecture is displayed in **figure 21** below, and the table following this summarises the tweaked components and hyperparameters.

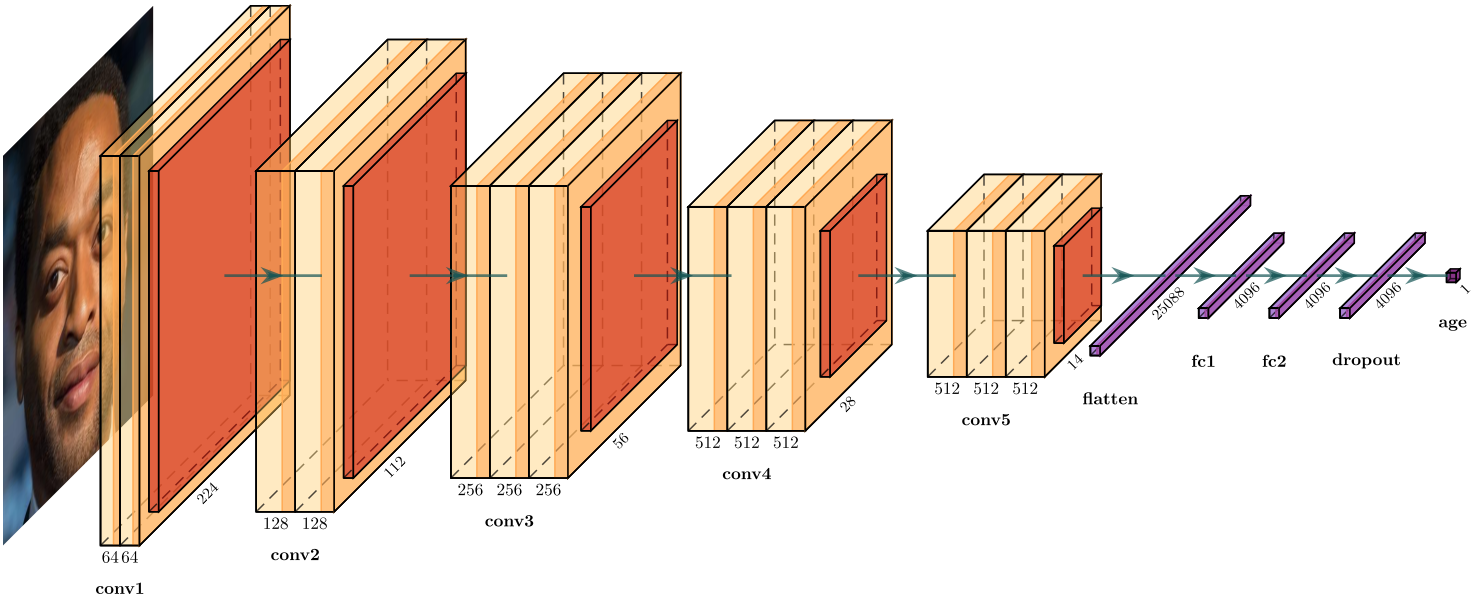


Figure 21: *Best performing architecture, created with PlotNeuralNet[45]*

Batch Size	**Learning Rate**	**Unfrozen Layers**	**Transfer Learning**	**Image pre-processing**	**Data Augmentation**
64	0.0001	• Conv block 5 • Flatten layer • fc1 • fc2 • dropout • final classifier layer	ImageNet + IMDb-Wiki	Loose crop border with landmark facial alignment	• random rotation (+10/-10) • 10% height shift • 10% width shift • 90%-110% zoom range • Horizontal image flipping

The improvement in model performance after tweaking the components of the pipeline was significant. To exemplify, the uncalibrated classification model has a MAE five years higher than the calibrated model. This model has given some accurate predictions at inference time on unseen data, as shown in the visualisation results.

It is important to compare the model I have created against approaches identified in the literature. These results are summarised in the table below.

Method	**ChaLearn MAE**
My model	*4.25*
DLDL-v2 (ThinAgeNet) [55]	3.45
DEX [32]	3.22
BridgeNet [25]	2.87

My model does fall behind the recent state-of-the-art perceived age prediction models shown above in terms of ChaLearn performance. Although my approach used the IMDb-Wiki dataset, my performance is below what was reported in the DEX paper. This result may be due to changes in pre-processing that were not reported in the paper, but it is more likely that this difference is linked with the ensemble learning approach used in DEX, with each model using a different train/test split. DEX used an ensemble of 20 models, whereas my approach used only a single model. It should be noted that this score is on the ChaLearn V1 test set, which overlaps with ChaLearn V2 that my model was evaluated on but is 862 samples smaller.

The performance of my model is closer to the DLDL-V2 approach. Interestingly, DLDL-v2 achieves this score without the large external data of IMDB-Wiki or an ensemble approach. The performance difference from my model can be explained as DLDL-V2 used effective ordinal ranking, similar to CORAL and the custom ThinAgeNet architecture which could handle age distribution bias and the non-stationary ageing property as with BridgeNet.

Finally, the largest performance difference exists between my model and BridgeNet. BridgeNet used IMDb-Wiki but only used VGG16 to extract features to pass to BridgeNet instead of as the main end-to-end architecture to obtain predictions from. The complex BridgeNet structure as discussed, enabled this large performance difference.

5 Conclusion

This section will reflect on the project, from the satisfaction of the aim and original objectives to discussion of personal development and future work.

5.1 Satisfaction of Aims and Objectives

This project aimed to evaluate deep learning approaches for predicting the perceived age of people using facial imagery. This aim was broken down into smaller objectives, with the satisfaction of these objectives ensuring overall project completion.

Objectives 1 and 2 focused on research to identify training strategies and pipeline components to conduct experiments on. These objectives were met as I identified that this problem could be modelled as a classification, regression, or ordinal classification problem. Then, I researched several distinct components that are suitable for adjustment when working with perceived age annotated images and CNNs.

This research then allowed me to meet **objective 3.** I achieved this objective by devising an appropriate experimental plan, which factored in expected complexity and was designed to allow incremental construction of the two models that used diverse training strategies.

The results of these experiments were evaluated independently at first and then evaluated against each other. The final best-performing model was evaluated against existing research to meet **objective 4.**

Finally, interpretability methods were implemented using existing techniques to visualise the inner workings of the CNN and the decision-making process. This allowed **objective 5** to be met.

5.2 What went well

Overall, I believe the project has been a success. The overall aim has been met, alongside the individual objectives. The experiments performed covered a wide range of CNN components, and careful thought was taken into analysing the results and ensuring the experimental results were clear.

The techniques I researched and used resulted in a significant increase in model performance from the initial versions to the finalised models. The evaluation allowed me to recognise shortcomings in the performance of the models whilst also appreciating the techniques that can improve perceived age prediction accuracy.

Additionally, the visualisation techniques were successfully utilised. Firstly, extracting knowledge to show the feature maps that were produced by the intermediate convolutional layers and more importantly showing *why* the network make predictions for differing ages. The

occlusion sensitivity results showed that the model was successful in identifying the face region and facial features, which adds more reliability and trust to the results achieved.

5.3 Improvements

Although the aim and objectives of this project have been accomplished, the aim could have been exceeded by performing additional experiments, which would result in a deeper evaluation. This could have resulted in final performance closer to the models identified during the research phase.

It was difficult to estimate how long each experiment would take in terms of implementation, as this was a new area for me, which brought difficulties such as spending significant time stuck on problems and trying to understand documentation. Moreover, it was unknown how long training would take for each experiment, which put pressure on time management. Data augmentation resulted in each epoch taking 3x as long, reducing the time planned for subsequent work. If I were to do this again, I would factor into the experimental plan this extra time for understanding and running experiments.

Some aspects were not successful. Implementation of CORAL ordinal being an area that failed due to my lack of experience in this area and time pressures. This is unfortunate, as an ordinal loss function would have modelled the problem closely and brought performance improvements.

5.4 Personal development

Completing this project has greatly developed and expanded my skills, which I attribute to the fact that deep learning was a new area that I had no prior experience in. These skills are summarised below.

Deep Learning: As the overarching theme of this project, I have gained considerable theoretical knowledge from the research phase and practical skill in implementing models using Keras and TensorFlow. The theoretical and practical knowledge I have gained pertains to CNN architectures.

Data manipulation: This was a general skill that I had to learn to complete this project and handle the ramifications of working with image data. Data manipulation mainly involved using Python packages such as Pandas, NumPy and OpenCV. Gaining experience with these packages to handle large datasets like IMDb-Wiki and manipulate images has increased my confidence with Python and given me the knowledge to use the language for future data wrangling problems. It was interesting to learn how mature Python's ecosystem is for data science and deep learning workloads.

Research and independent learning: I have developed these skills substantially throughout the project. I have learnt how to analyse papers and use my existing programming knowledge to adapt to new frameworks and apply the theoretical knowledge gained through research.

5.5 Future Work

I believe that there is still a great deal of further work that could be undertaken in this domain. One key area of experimentation could be comparing the impact of different multi-model ensemble approaches against using a single model.

It would also be interesting to implement novel image transformations at the pre-processing stage and at train-time during augmentation to add further understanding to the predictions made and improve performance. This could involve semantic segmentation and swapping of facial features in images to fully understand which features have the largest influence on the prediction of certain ages. Additionally, a more robust alignment algorithm could be implemented as the naïve one currently used is not perfect and failed on some samples.

It was clearly identified that the ChaLearn dataset has a poor, unequal age distribution (**Figure 9).** The impact of this could have been studied by comparing average test predictions for different age groups, rather than computing the MAE across the entire test set. The distribution of ethnicities could also have been investigated in a similar manner. Techniques to mitigate the impact of this biased distribution, as used in BridgeNet and DLDL-v2, could then be explored.

Finally, a major component that is yet to be tweaked is the underlying architecture. The implemented VGG16 architecture has an expensive computational cost and storage overhead, with its 138M parameters equating to around 500MB of disk space. It would be good to implement smaller modern networks such as MobileNetV2 and EfficientNet both for comparison to VGG16 and to open the gateway to allow the perceived age prediction model to run on mobile devices with TensorFlow Lite. However, before deployment to a production environment like a mobile application, performance would have to be improved.

6 References

[1] McCarthy, J. (2019) *What is AI? / Basic Questions*. [Online] [online]. Available from: http://jmc.stanford.edu/artificial-intelligence/what-is-ai/index.html.

[2] Müller, V.C. & Bostrom, N. (2014) Future progress in artificial intelligence. AI Matters. 1 (1), 9–11.

[3] AIMultiple (2021) *995 Expert opinions: AGI / Singularity by 2060*. [Online] [online]. Available from: https://research.aimultiple.com/artificial-general-intelligence-singularity-timing/.

[4] Manning, C. (2020) *Artificial Intelligence Definitions*.

[5] Witten, I.H. & Al, E. (2017) Data Mining: Practical Machine Learning Tools and Techniques. Vol. 4. Amsterdam: Morgan Kaufmann.

[6] Chollet, F. (2017) Deep Learning with Python.

[7] Heaton, J. (2021) Applications of Deep Neural Networks. arXiv:2009.05673 [cs].

[8] Gunn, D.A., Murray, P.G., Tomlin, C.C., Rexbye, H., Christensen, K. & Mayes, A.E. (2008) Perceived age as a biomarker of ageing: a clinical methodology. Biogerontology. 9 (5), 357–364.

[9] Gunn, D.A., Larsen, L.A., Lall, J.S., Rexbye, H. & Christensen, K. (2016) Mortality is Written on the Face. The Journals of Gerontology Series A: Biological Sciences and Medical Sciences. 71 (1), 72–77.

[10] Raina, R., Madhavan, A. & Ng, A.Y. (2009) Large-scale deep unsupervised learning using graphics processors. Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09.

[11] Oh, K.-S. & Jung, K. (2004) GPU implementation of neural networks. Pattern Recognition. 37 (6), 1311–1314.

[12] McCulloch, W.S. & Pitts, W. (1943) A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology. 52 (1-2), 99–115.

[13] Rosenblatt, F. (1958) The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review. 65 (6), 386–408.

[14] Fukushima, K. (1980) Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics. 36 (4), 193–202.

[15] Fei-Fei, L., Deng, J. & Li, K. (2010) ImageNet: Constructing a large-scale image database. Journal of Vision. 9 (8), 1037–1037.

[16] Krizhevsky, A., Sutskever, I. & Hinton, G.E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems. 251097–1105.

[17] Pham, H., Dai, Z., Xie, Q., Luong, M.-T. & Le, Q. (2020) Meta Pseudo Labels.

[18] Escalera, S. (2016) ChaLearn. [Online] [online]. Available from: http://chalearnlap.cvc.uab.es/dataset/19/description/ (Accessed 14 January 2021).

[19] Simonyan, K. & Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition. [Online] [online]. Available from: https://arxiv.org/abs/1409.1556.

[20] Chollet, F. (2017) Xception: Deep Learning with Depthwise Separable Convolutions.

[21] Tan, M. & Le, Q.V. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. [Online] [online]. Available from: https://arxiv.org/abs/1905.11946.

[22] Brownlee, J. (2019) A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. [Online] [online]. Available from: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

[23] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. & Bengio, Y. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 151929–1958.

[24] Shorten, C. & Khoshgoftaar, T.M. (2019) A survey on Image Data Augmentation for Deep Learning. Journal of Big Data. 6 (1),.

[25] Li, W., Lu, J., Feng, J., Xu, C., Zhou, J. & Tian, Q. (2019) BridgeNet: A Continuity-Aware Probabilistic Network for Age Estimation.

[26] Zhang, K., Zhang, Z., Li, Z. & Qiao, Y. (2016) Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. IEEE Signal Processing Letters. 23 (10), 1499–1503.

[27] Viola, P. & Jones, M. (2001) Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.

[28] Ricanek, K. & Tesafaye, T. (2006) MORPH: A Longitudinal Image Database of Normal Adult Age-Progression. *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*.

[29] Song, Y., Zhang, Z. & Qi, H. (2017) *UTKFace*. [Online] [online]. Available from: https://susanqq.github.io/UTKFace/.

[30] Chen, B.-C., Chen, C.-S. & Hsu, W.H. (2014) Cross-Age Reference Coding for Age-Invariant Face Recognition and Retrieval. *Computer Vision – ECCV 2014*. 768–783.

[31] Cao, W., Mirjalili, V. & Raschka, S. (2020) Rank consistent ordinal regression for neural networks with application to age estimation. Pattern Recognition Letters. 140325–331.

[32] Rothe, R., Timofte, R. & Van Gool, L. (2016) Deep Expectation of Real and Apparent Age from a Single Image Without Facial Landmarks. International Journal of Computer Vision. 126 (2-4), 144–157.

[33] Mathias, M., Benenson, R., Pedersoli, M. & Van Gool, L. (2014) Face Detection without Bells and Whistles. Computer Vision – ECCV 2014. 720–735.

[34] Stack Overflow (2020) Stack Overflow Developer Survey 2020. [Online] [online]. Available from: https://insights.stackoverflow.com/survey/2020#most-popular-technologies.

[35] Google Tensorflow (2021) *tensorflow/tensorflow*. [Online] [online]. Available from: https://github.com/tensorflow/tensorflow.

[36] Facebook PyTorch (2021) *pytorch/pytorch*. [Online] [online]. Available from: https://github.com/pytorch/pytorch.

[37] Howard, J., Thomas, R. (2021) Fast AI. [Online] [online]. Available from: https://docs.fast.ai/.

[38] Howard, J., Gugger, S. (2020) Deep Learning With FastAI and PyTorch: AI Applications without a PhD. S.L.: O'reilly Media.

[39] Chollet, F. (2019) Keras. [Online] [online]. Available from: https://keras.io.

[40] Rothe, R., Timofte, R. & Gool, L.V. (2015) IMDB-WIKI - 500k+ face images with age and gender labels. [Online] [online]. Available from: https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/.

[41] Elite Data Science (2017) Overfitting in Machine Learning: What It Is and How to Prevent It. [Online] [online]. Available from: https://elitedatascience.com/overfitting-in-machine-learning#signal-vs-noise.

[42] Gupta, S. & Gupta, A. (2019) Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review. Procedia Computer Science. 161466–474.

[43] The HDF Group (2021) The HDF5® Library & File Format. [Online] [online]. Available from: https://www.hdfgroup.org/solutions/hdf5/.

[44] Chollet, F. (2020) Keras documentation: Transfer learning & fine-tuning. [Online] [online]. Available from: https://keras.io/guides/transfer_learning/.

[45] Iqbal, H. (2021) HarisIqbal88/PlotNeuralNet. [Online] [online]. Available from: https://github.com/HarisIqbal88/PlotNeuralNet (Accessed 30 March 2021).

[46] Brownlee, J. (2017) Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. [Online] [online]. Available from: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[47] Bergstra, J., Ca, J. & Ca, Y. (2012) Random Search for Hyper-Parameter Optimization Yoshua Bengio. Journal of Machine Learning Research. 13281–305.

[48] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. (2018) Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. arXiv:1603.06560 [cs, stat].

[49] Goodfellow, I., Yoshua Bengio & Courville, A. (2017) Deep Learning. Cambridge, Massachusetts: The Mit Press.

[50] King, D. (2021) dlib: A toolkit for making real world machine learning and data analysis applications. [Online] [online]. Available from: https://pypi.org/project/dlib/ (Accessed 1 April 2021).

[51] Yang, J., Bulat, A. & Tzimiropoulos, G. (2020) FAN-Face: a Simple Orthogonal Improvement to Deep Face Recognition. Proceedings of the AAAI Conference on Artificial Intelligence. 34 (07), 12621–12628.

[52] Bulat, A. (2021) 1adrianb/face-alignment. [Online] [online]. Available from: https://github.com/1adrianb/face-alignment (Accessed 7 April 2021).

[53] Rémy, P. (2021) philipperemy/keract. [Online] [online]. Available from: https://github.com/philipperemy/keract (Accessed 8 April 2021).

[54] Bede (n.d.) BEDE N8 CIR. [Online] [online]. Available from: https://n8cir.org.uk/ (Accessed 8 April 2021)

[55] Gao, B.-B., Zhou, H.-Y., Wu, J. & Geng, X. (2018) Age Estimation Using Expectation of Label Distribution Learning. *www.ijcai.org*. 712–718.

7 Glossary

Activation – The output of a neuron in a neural network. The output of a neuron, given a set of inputs (weights) (e.g. Sigmoid, ReLu).

Artificial Neural Network - A form of processing data, inspired by the human brain, involving many interconnected layers of neurons processing data downstream to further layers before outputting a prediction.

Classification - A form of learning which aims to classify the input into a particular category.

Model - An encapsulation of the artificial neural network and what it has learnt during training.

Neuron - A unit of work in the artificial neural network to process some part of the input data.

Regression - A form of learning which aims to predict a continuous output from the input data.

Supervised Learning - A machine learning technique in which the training data is labelled with the target value.

Test Set - An unlabelled, unseen set of data to test the trained model on at inference time.

Training Set - Typically, the largest split of the dataset that the model learns from during each epoch.

Validation set - A smaller split of data to validate what the model is learning from the training set.