AGE ESTIMATION & GENDER CLASSIFICATION

With Convolutional Neural Networks

Group ID	**A**
Student ID's	219501369
	219304092
	219496777
	219509000

Links:

age_gender_A.h5

age_gender_B.h5

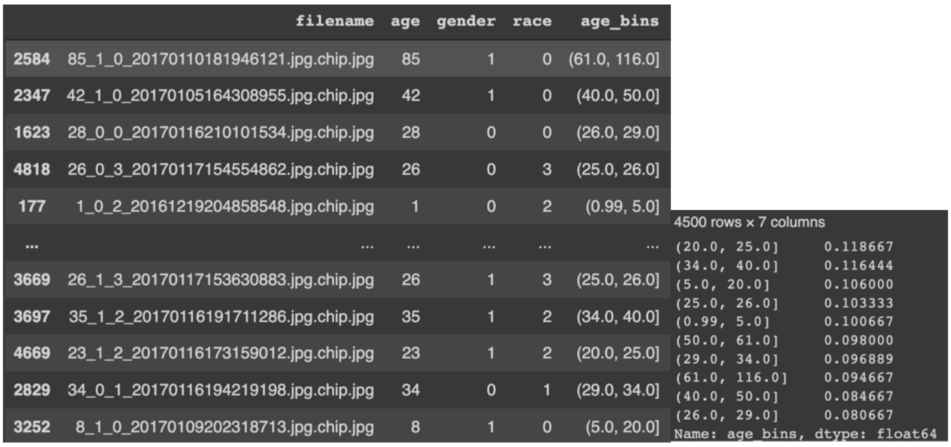
Section 1: Introduction

Convolutional neural networks (CNNs) are regularised multilayer perceptrons and are an integral part of modern deep learning computer vision applications. Considering each pixel as a feature, images have very high dimensionality, hence the convolutional layers in CNNs are very important in dimensionality reduction.

In this project, two CNNs will be trained for gender and age classification, using 5,000 labelled images from the UTKFace dataset. The first model (age_gender_A.h5) will be trained from scratch and the second (age_gender_B.h5) will involve fine-tuning a pre-trained model.

Section 2: My own CNN

The images were stratified by equal-sized age bins before splitting into training, validation and test sets to ensure each set had the same distribution of age ranges.



	filename	age	gender	race	age_bins
2584	85_1_0_20170110181946121.jpg.chip.jpg	85	1	0	(61.0, 116.0]
2347	42_1_0_20170105164308955.jpg.chip.jpg	42	1	0	(40.0, 50.0]
1623	28_0_0_20170116210101534.jpg.chip.jpg	28	0	0	(26.0, 29.0]
4818	26_0_3_20170117154554862.jpg.chip.jpg	26	0	3	(25.0, 26.0]
177	1_0_2_20161219204858548.jpg.chip.jpg	1	0	2	(0.99, 5.0]
...	...	...	...	...	...
3669	26_1_3_20170117153630883.jpg.chip.jpg	26	1	3	(25.0, 26.0]
3697	35_1_2_20170116191711286.jpg.chip.jpg	35	1	2	(34.0, 40.0]
4669	23_1_2_20170116173159012.jpg.chip.jpg	23	1	2	(20.0, 25.0]
2829	34_0_1_20170116194219198.jpg.chip.jpg	34	0	1	(29.0, 34.0]
3252	8_1_0_20170109202318713.jpg.chip.jpg	8	1	0	(5.0, 20.0]

4500 rows × 7 columns

```
(20.0, 25.0]     0.118667
(34.0, 40.0]     0.116444
(5.0, 20.0]      0.106000
(25.0, 26.0]     0.103333
(0.99, 5.0]      0.100667
(50.0, 61.0]     0.098000
(29.0, 34.0]     0.096889
(61.0, 116.0]    0.094667
(40.0, 50.0]     0.084667
(26.0, 29.0]     0.080667
Name: age_bins, dtype: float64
```

Figure 1: The training, validation and test sets were stratified by even age bins to ensure similar distributions for each set.

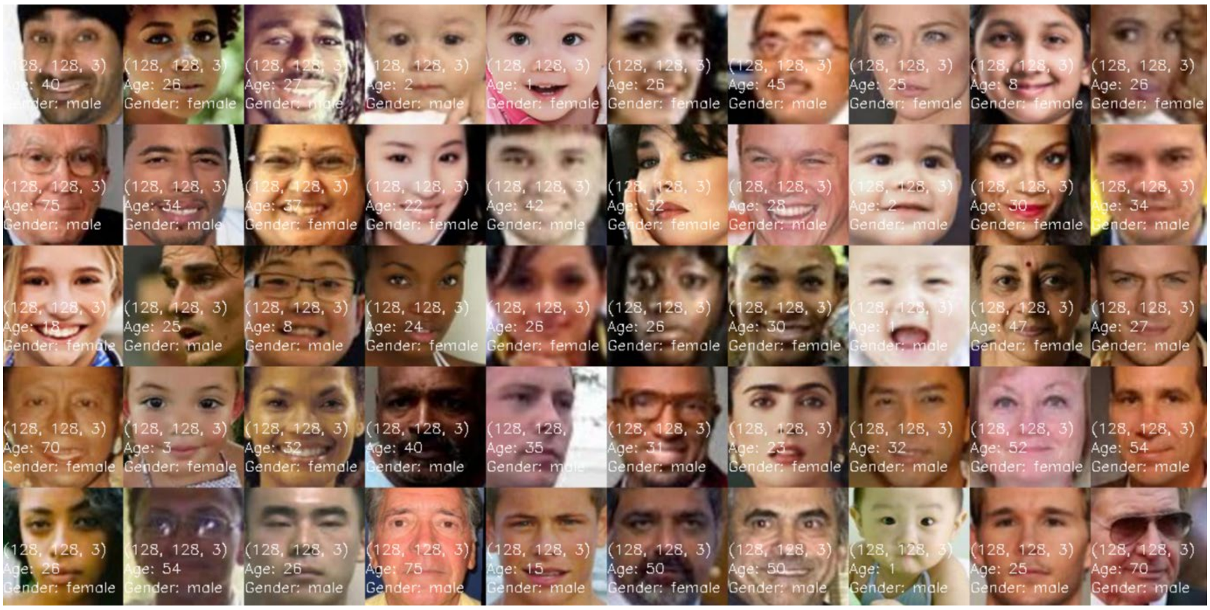
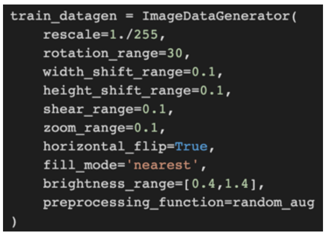

(128, 128, 3) Age: 40 Gender: male
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 27 Gender: male
(128, 128, 3) Age: 2 Gender: male
(128, 128, 3) Age: 1 Gender: female
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 45 Gender: male
(128, 128, 3) Age: 25 Gender: female
(128, 128, 3) Age: 8 Gender: female
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 75 Gender: male
(128, 128, 3) Age: 34 Gender: male
(128, 128, 3) Age: 37 Gender: female
(128, 128, 3) Age: 22 Gender: female
(128, 128, 3) Age: 42 Gender: male
(128, 128, 3) Age: 32 Gender: female
(128, 128, 3) Age: 28 Gender: male
(128, 128, 3) Age: 2 Gender: male
(128, 128, 3) Age: 30 Gender: female
(128, 128, 3) Age: 34 Gender: male
(128, 128, 3) Age: 18 Gender: female
(128, 128, 3) Age: 25 Gender: male
(128, 128, 3) Age: 8 Gender: male
(128, 128, 3) Age: 24 Gender: female
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 30 Gender: female
(128, 128, 3) Age: 1 Gender: male
(128, 128, 3) Age: 47 Gender: female
(128, 128, 3) Age: 27 Gender: male
(128, 128, 3) Age: 70 Gender: female
(128, 128, 3) Age: 3 Gender: female
(128, 128, 3) Age: 32 Gender: female
(128, 128, 3) Age: 40 Gender: male
(128, 128, Age: 35 Gender: male
(128, 128, 3) Age: 31 Gender: male
(128, 128, 3) Age: 23 Gender: female
(128, 128, 3) Age: 32 Gender: male
(128, 128, 3) Age: 52 Gender: female
(128, 128, 3) Age: 54 Gender: male
(128, 128, 3) Age: 26 Gender: female
(128, 128, 3) Age: 54 Gender: male
(128, 128, 3) Age: 26 Gender: male
(128, 128, 3) Age: 75 Gender: male
(128, 128, 3) Age: 15 Gender: male
(128, 128, 3) Age: 50 Gender: female
(128, 128, 3) Age: 50 Gender: male
(128, 128, 3) Age: 1 Gender: male
(128, 128, 3) Age: 25 Gender: male
(128, 128, 3) Age: 70 Gender: male

Figure 1: The raw UTKFace dataset.

The training data was then fed into the augmentation pipeline so that for each epoch, the model would receive 4500 unique images during training, generated from ImageDataGenerator.



```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest',
    brightness_range=[0.4,1.4],
    preprocessing_function=random_aug
)
```

Figure 2: The random Data Augmentations applied to the training dataset. Additional custom blur transformations were also used.

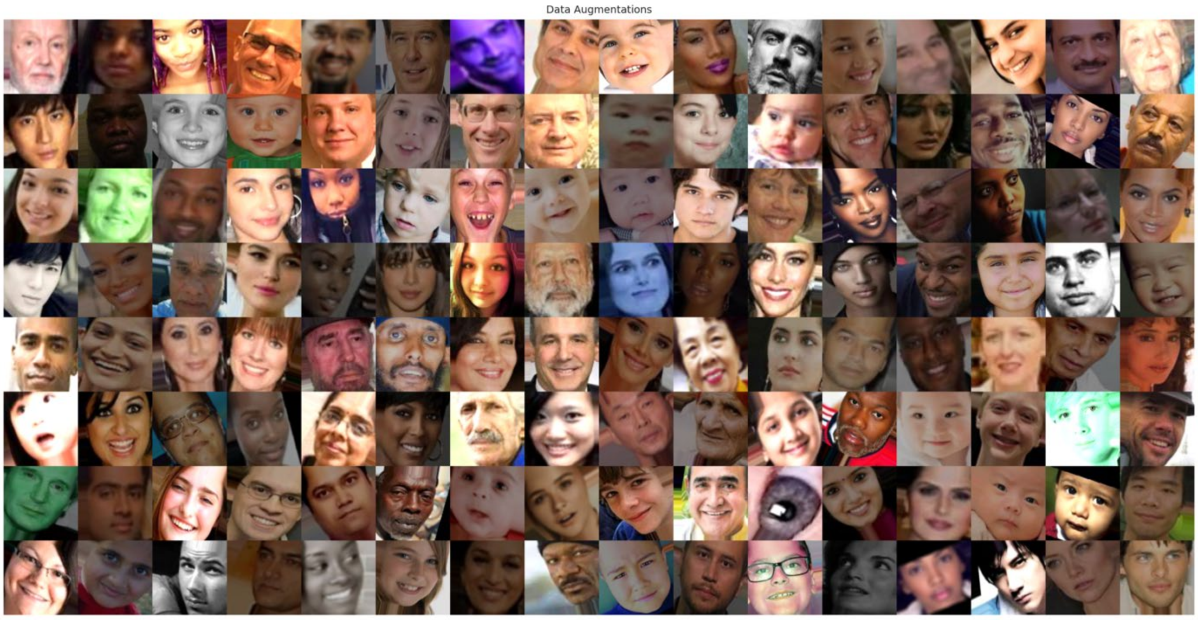
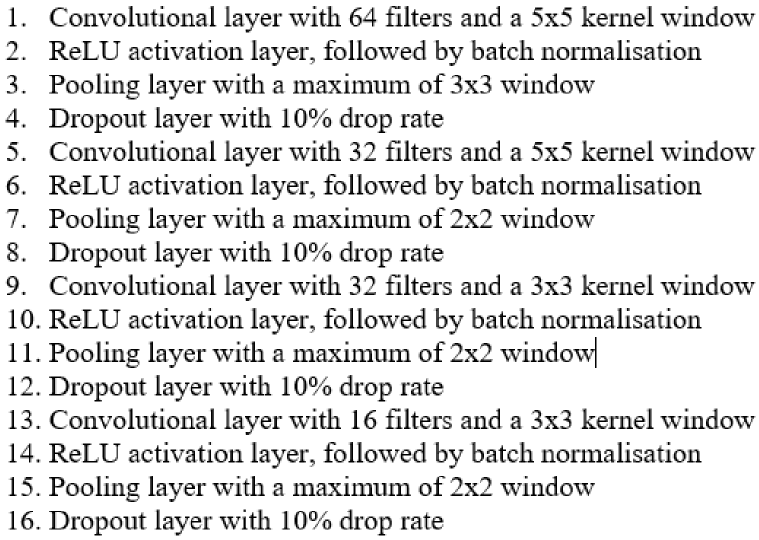


Figure 3: The randomly Augmented dataset.

Data Augmentation is used to prevent the model from memorising the training set and thus overfitting. The validation and test sets both have 250 images reserved, which are unmodified but still rescaled within the range 0 to 1. In the identification of gender from facial images, our model initially splits the age and sex features as different branches in CNN. These two branches are later combined so the model can make better-informed decisions for age prediction based on gender.

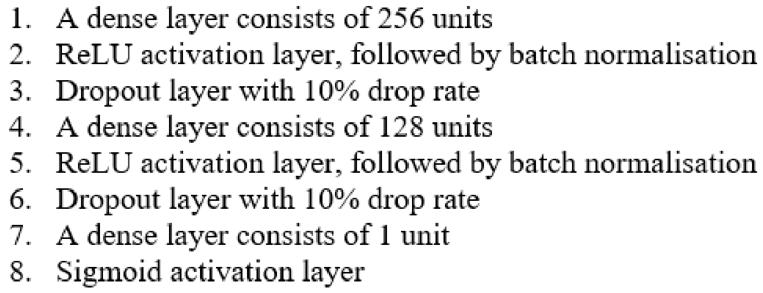
For each branch, the following sequence of hidden layers was adopted:



1. Convolutional layer with 64 filters and a 5x5 kernel window
2. ReLU activation layer, followed by batch normalisation
3. Pooling layer with a maximum of 3x3 window
4. Dropout layer with 10% drop rate
5. Convolutional layer with 32 filters and a 5x5 kernel window
6. ReLU activation layer, followed by batch normalisation
7. Pooling layer with a maximum of 2x2 window
8. Dropout layer with 10% drop rate
9. Convolutional layer with 32 filters and a 3x3 kernel window
10. ReLU activation layer, followed by batch normalisation
11. Pooling layer with a maximum of 2x2 window
12. Dropout layer with 10% drop rate
13. Convolutional layer with 16 filters and a 3x3 kernel window
14. ReLU activation layer, followed by batch normalisation
15. Pooling layer with a maximum of 2x2 window
16. Dropout layer with 10% drop rate

The hidden layer followed the pattern of Convolutional 2D layers, ReLU activation layers, batch normalisation layers, Max pooling layers, and dropout layers. These 5 layers are repeated 4 times, with each time having smaller filters and kernel windows. The pooling layer picks the maximum value of the feature map and passes to the dropout, where 10% of the node is dropped to prevent overfitting. This dropout rate is set at a constant 10%.

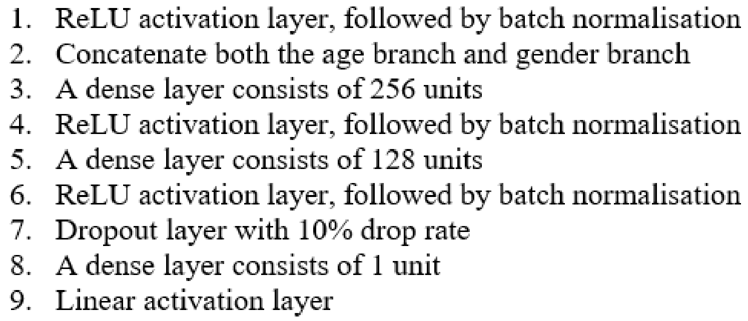
In the gender branch, the output of the model is first flattened, followed by:



1. A dense layer consists of 256 units
2. ReLU activation layer, followed by batch normalisation
3. Dropout layer with 10% drop rate
4. A dense layer consists of 128 units
5. ReLU activation layer, followed by batch normalisation
6. Dropout layer with 10% drop rate
7. A dense layer consists of 1 unit
8. Sigmoid activation layer

For the gender branch, the final layer is a sigmoid activation function, as gender is defined as a binary output of male and female and hence is a classification problem.

For the age branch, the output from the hidden layer model is first flattened, then:

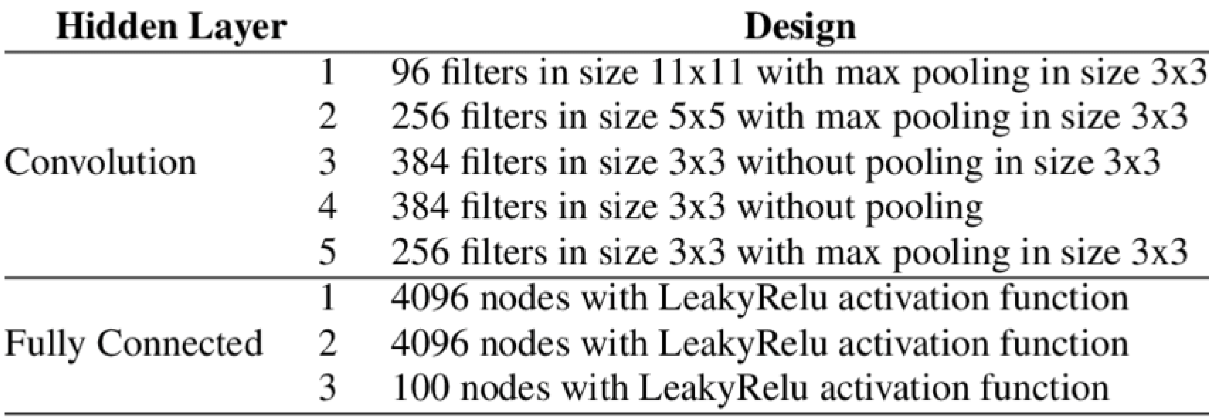


1. ReLU activation layer, followed by batch normalisation
2. Concatenate both the age branch and gender branch
3. A dense layer consists of 256 units
4. ReLU activation layer, followed by batch normalisation
5. A dense layer consists of 128 units
6. ReLU activation layer, followed by batch normalisation
7. Dropout layer with 10% drop rate
8. A dense layer consists of 1 unit
9. Linear activation layer

For the age branch, a linear activation function is used instead because age is continuous and hence a regression problem. During the initial training, the initial learning rate is set at 0.008 with epochs at 200 and epsilon at 0.1. These hyperparameters were decided through several trials and errors testing. We chose to use Adam for the optimizer rather than standard Gradient Descent as Adam converges faster which we needed due to limited GPU time. Adam is an algorithm for gradient-based optimization of stochastic objective functions. It combines the advantages of RMSProp and AdaGrad and computes individual adaptive learning rates for different parameters. In figure 6 of the training curve, the model was further trained for another 50 epochs, meaning the total epochs trained were 250.

We chose to minimise the loss for the age branch with the mean squared error and the gender branch with binary cross-entropy. The output metrics chosen were also the mean average error and accuracy respectively.

Our model architecture was inspired by the 2012 ImageNet competition winner AlexNet [1] but scaled down to tackle the given problem efficiently.



Hidden Layer		Design
Convolution	1	96 filters in size 11x11 with max pooling in size 3x3
	2	256 filters in size 5x5 with max pooling in size 3x3
	3	384 filters in size 3x3 without pooling in size 3x3
	4	384 filters in size 3x3 without pooling
	5	256 filters in size 3x3 with max pooling in size 3x3
Fully Connected	1	4096 nodes with LeakyRelu activation function
	2	4096 nodes with LeakyRelu activation function
	3	100 nodes with LeakyRelu activation function

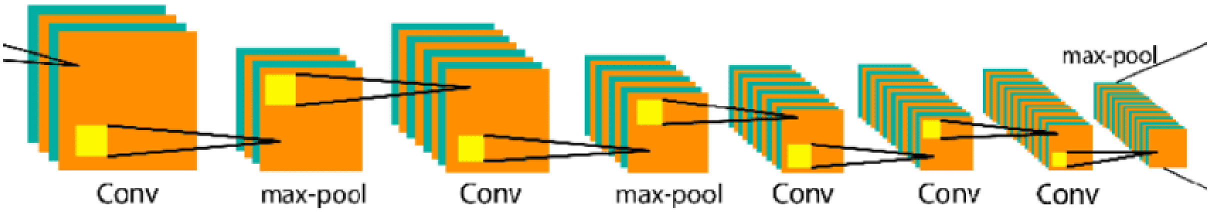


Figure 4: AlexNet architecture.

The result of our validation has an MAE (mean absolute error) of 5.65 years with an accuracy of the gender classification of 94.53%. The test MAE for age is about 4.60 years with an accuracy of gender classification of 92.19%.

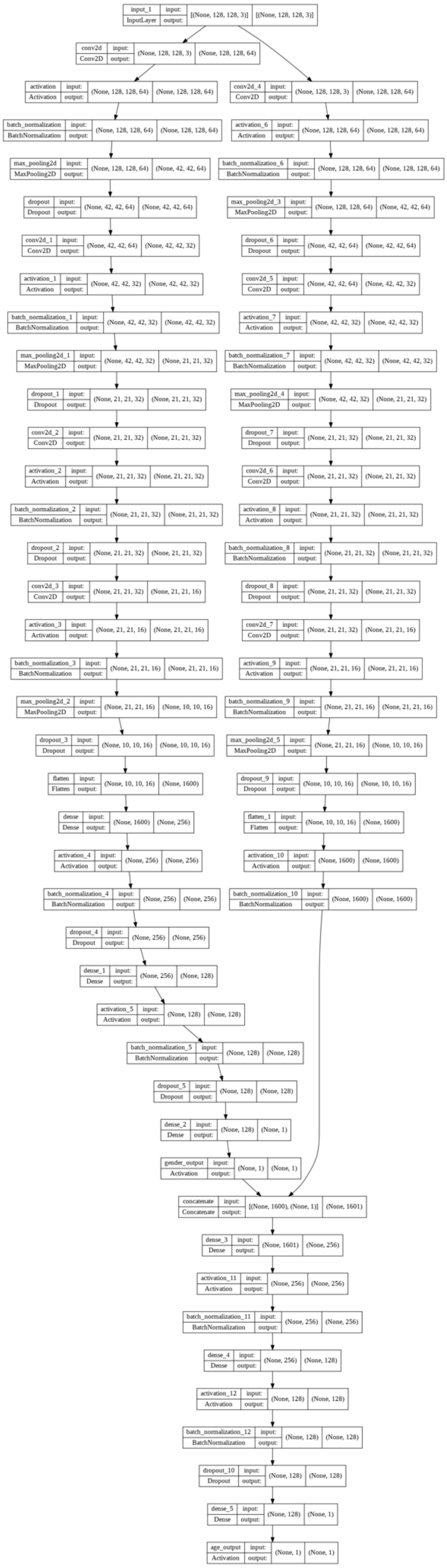
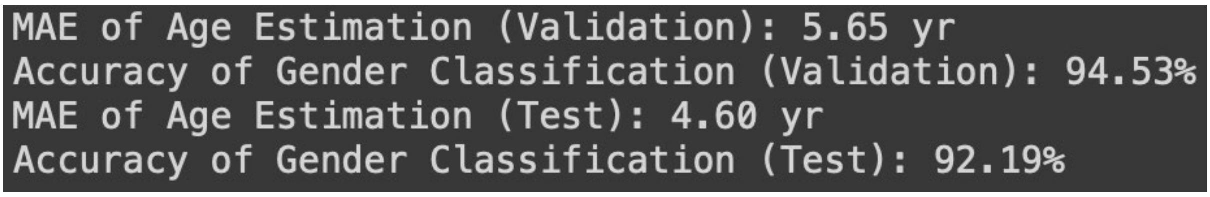

input_1 | input: | [(None, 128, 128, 3)] | [(None, 128, 128, 3)]
InputLayer | output:
conv2d | input: | (None, 128, 128, 3) | (None, 128, 128, 64)
Conv2D | output:
activation | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
Activation | output:
batch_normalization | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
BatchNormalization | output:
max_pooling2d | input: | (None, 128, 128, 64) | (None, 42, 42, 64)
MaxPooling2D | output:
dropout | input: | (None, 42, 42, 64) | (None, 42, 42, 64)
Dropout | output:
conv2d_1 | input: | (None, 42, 42, 64) | (None, 42, 42, 32)
Conv2D | output:
activation_1 | input: | (None, 42, 42, 32) | (None, 42, 42, 32)
Activation | output:
batch_normalization_1 | input: | (None, 42, 42, 32) | (None, 42, 42, 32)
BatchNormalization | output:
max_pooling2d_1 | input: | (None, 42, 42, 32) | (None, 21, 21, 32)
MaxPooling2D | output:
dropout_1 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Dropout | output:
conv2d_2 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Conv2D | output:
activation_2 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Activation | output:
batch_normalization_2 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
BatchNormalization | output:
dropout_2 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Dropout | output:
conv2d_3 | input: | (None, 21, 21, 32) | (None, 21, 21, 16)
Conv2D | output:
activation_3 | input: | (None, 21, 21, 16) | (None, 21, 21, 16)
Activation | output:
batch_normalization_3 | input: | (None, 21, 21, 16) | (None, 21, 21, 16)
BatchNormalization | output:
max_pooling2d_2 | input: | (None, 21, 21, 16) | (None, 10, 10, 16)
MaxPooling2D | output:
dropout_3 | input: | (None, 10, 10, 16) | (None, 10, 10, 16)
Dropout | output:
flatten | input: | (None, 10, 10, 16) | (None, 1600)
Flatten | output:
dense | input: | (None, 1600) | (None, 256)
Dense | output:
activation_4 | input: | (None, 256) | (None, 256)
Activation | output:
batch_normalization_4 | input: | (None, 256) | (None, 256)
BatchNormalization | output:
dropout_4 | input: | (None, 256) | (None, 256)
Dropout | output:
dense_1 | input: | (None, 256) | (None, 128)
Dense | output:
activation_5 | input: | (None, 128) | (None, 128)
Activation | output:
batch_normalization_5 | input: | (None, 128) | (None, 128)
BatchNormalization | output:
dropout_5 | input: | (None, 128) | (None, 128)
Dropout | output:
dense_2 | input: | (None, 128) | (None, 1)
Dense | output:
gender_output | input: | (None, 1) | (None, 1)
Activation | output:
conv2d_4 | input: | (None, 128, 128, 3) | (None, 128, 128, 64)
Conv2D | output:
activation_6 | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
Activation | output:
batch_normalization_6 | input: | (None, 128, 128, 64) | (None, 128, 128, 64)
BatchNormalization | output:
max_pooling2d_3 | input: | (None, 128, 128, 64) | (None, 42, 42, 64)
MaxPooling2D | output:
dropout_6 | input: | (None, 42, 42, 64) | (None, 42, 42, 64)
Dropout | output:
conv2d_5 | input: | (None, 42, 42, 64) | (None, 42, 42, 32)
Conv2D | output:
activation_7 | input: | (None, 42, 42, 32) | (None, 42, 42, 32)
Activation | output:
batch_normalization_7 | input: | (None, 42, 42, 32) | (None, 42, 42, 32)
BatchNormalization | output:
max_pooling2d_4 | input: | (None, 42, 42, 32) | (None, 21, 21, 32)
MaxPooling2D | output:
dropout_7 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Dropout | output:
conv2d_6 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Conv2D | output:
activation_8 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Activation | output:
batch_normalization_8 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
BatchNormalization | output:
dropout_8 | input: | (None, 21, 21, 32) | (None, 21, 21, 32)
Dropout | output:
conv2d_7 | input: | (None, 21, 21, 32) | (None, 21, 21, 16)
Conv2D | output:
activation_9 | input: | (None, 21, 21, 16) | (None, 21, 21, 16)
Activation | output:
batch_normalization_9 | input: | (None, 21, 21, 16) | (None, 21, 21, 16)
BatchNormalization | output:
max_pooling2d_5 | input: | (None, 21, 21, 16) | (None, 10, 10, 16)
MaxPooling2D | output:
dropout_9 | input: | (None, 10, 10, 16) | (None, 10, 10, 16)
Dropout | output:
flatten_1 | input: | (None, 10, 10, 16) | (None, 1600)
Flatten | output:
activation_10 | input: | (None, 1600) | (None, 1600)
Activation | output:
batch_normalization_10 | input: | (None, 1600) | (None, 1600)
BatchNormalization | output:
concatenate | input: | [(None, 1600), (None, 1)] | (None, 1601)
Concatenate | output:
dense_3 | input: | (None, 1601) | (None, 256)
Dense | output:
activation_11 | input: | (None, 256) | (None, 256)
Activation | output:
batch_normalization_11 | input: | (None, 256) | (None, 256)
BatchNormalization | output:
dense_4 | input: | (None, 256) | (None, 128)
Dense | output:
activation_12 | input: | (None, 128) | (None, 128)
Activation | output:
batch_normalization_12 | input: | (None, 128) | (None, 128)
BatchNormalization | output:
dropout_10 | input: | (None, 128) | (None, 128)
Dropout | output:
dense_5 | input: | (None, 128) | (None, 1)
Dense | output:
age_output | input: | (None, 1) | (None, 1)
Activation | output:

Figure 5: ModelA



```
MAE of Age Estimation (Validation): 5.65 yr
Accuracy of Gender Classification (Validation): 94.53%
MAE of Age Estimation (Test): 4.60 yr
Accuracy of Gender Classification (Test): 92.19%
```

Figure 6: Age MAE and Gender accuracy for our model.

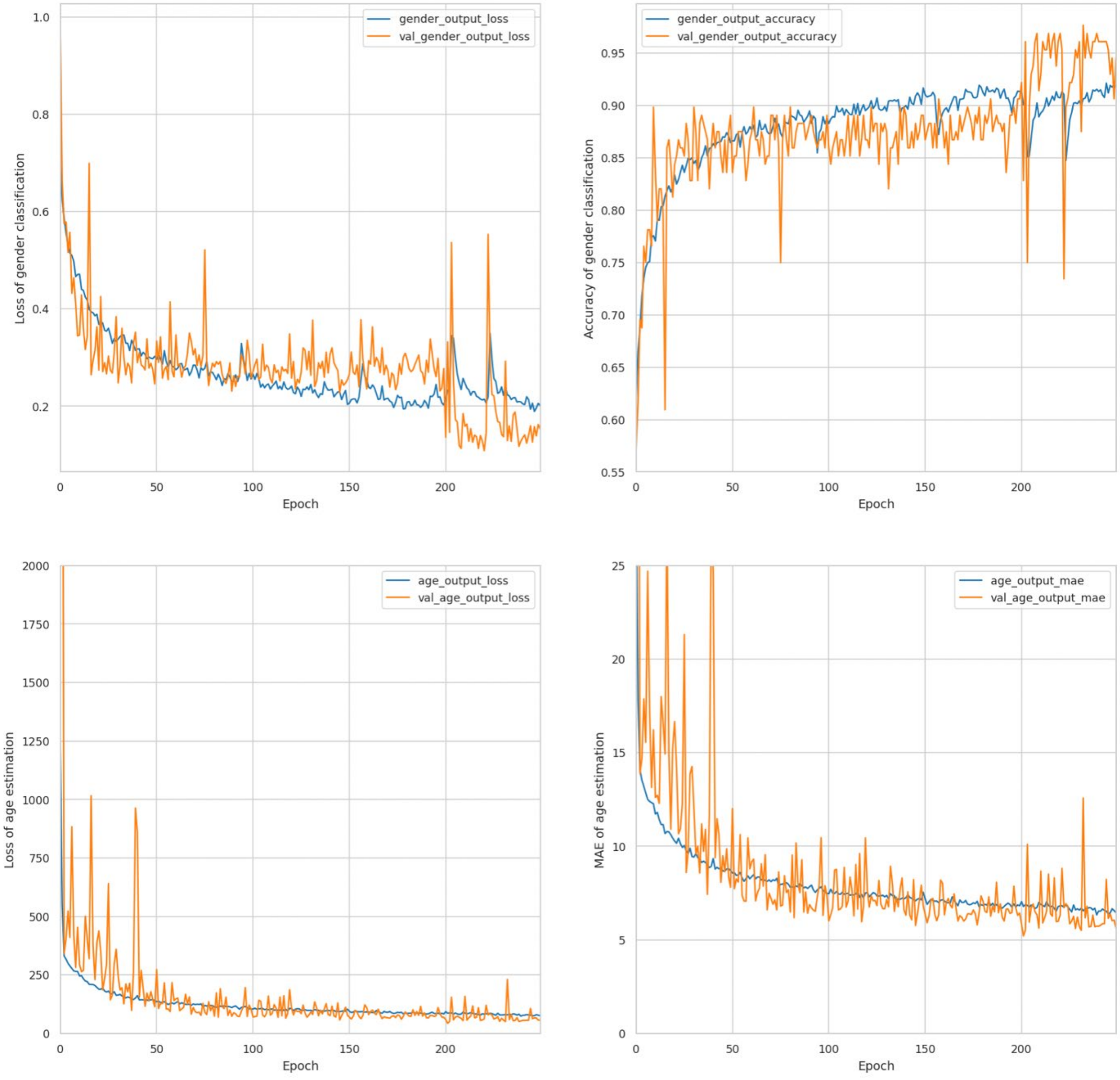
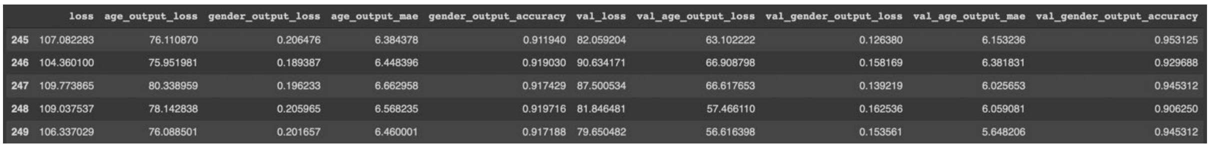


Figure 7: Training/Validation curves.



	loss	age_output_loss	gender_output_loss	age_output_mae	gender_output_accuracy	val_loss	val_age_output_loss	val_gender_output_loss	val_age_output_mae	val_gender_output_accuracy
245	107.082283	76.110870	0.206476	6.384378	0.911940	82.059204	63.102222	0.126380	6.153236	0.953125
246	104.360100	75.951981	0.189387	6.448396	0.919030	90.634171	66.908798	0.158169	6.381831	0.929688
247	109.773865	80.338959	0.196233	6.662958	0.917429	87.500534	66.617653	0.139219	6.025653	0.945312
248	109.037537	78.142838	0.205965	6.568235	0.919716	81.846481	57.466110	0.162536	6.059081	0.906250
249	106.337029	76.088501	0.201657	6.460001	0.917188	79.650482	56.616398	0.153561	5.648206	0.945312

Figure 8: The age and gender loss.

Section 3: Pre-trained CNN

In the pre-train part of the CNN, we used Xception, which is a CNN that consists of 71 layers[2,3]. The pre-trained version of this network has been trained on more than 1 million images from the ImageNet itself [4]. We assume that the pre-train CNN would return with much higher accuracy. For the training in this pre-trained CNN, the initial learning rate is set the same as in our CNN with epochs at 150 and epsilon at 0.1. Except for the hidden layer, both the age and gender branch are the same as the one used in our CNN model. Adam is still used as the optimizer in the compiler. The result from this pre-trained model gave an MAE of 8.38 years with an accuracy of the gender classification of 82.03% in the validation model. The test MAE for age is about 8.50 years with an accuracy of 83.59% in gender classification.

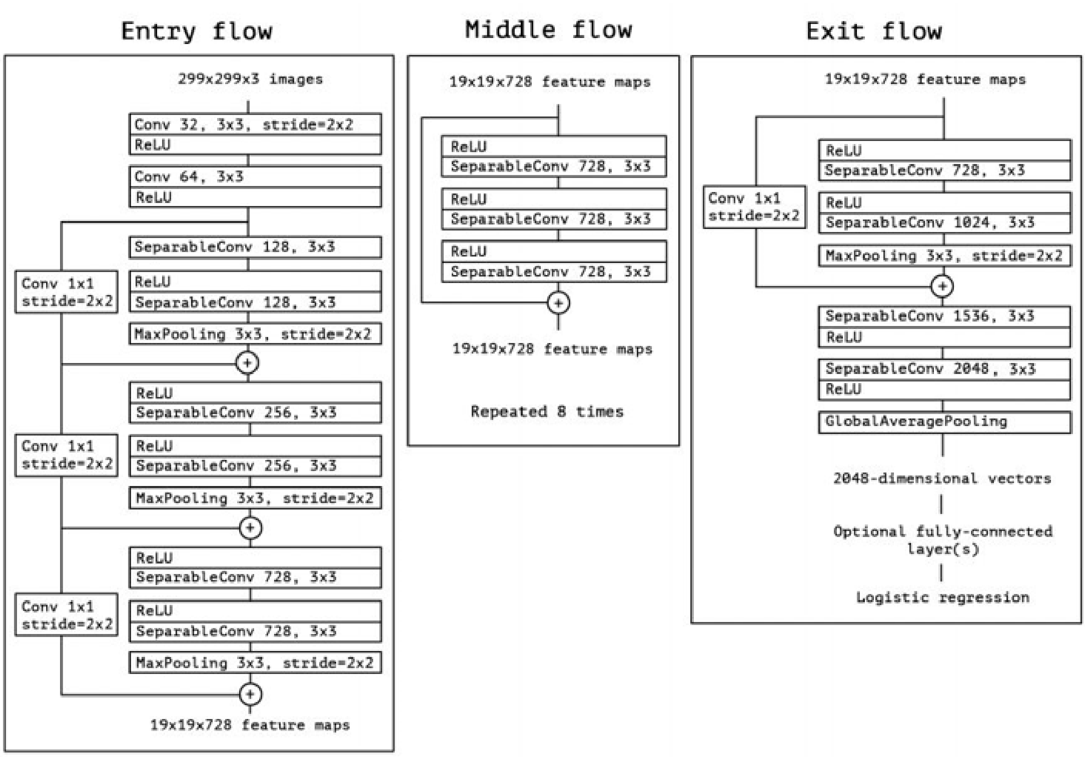
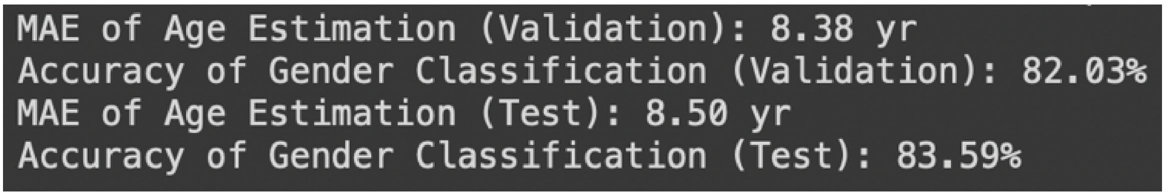


Figure 10: The Xception model pre-trained with ImageNet.



```
MAE of Age Estimation (Validation): 8.38 yr
Accuracy of Gender Classification (Validation): 82.03%
MAE of Age Estimation (Test): 8.50 yr
Accuracy of Gender Classification (Test): 83.59%
```

Figure 11: Age MAE and Gender accuracy in validation and test sets for the pre-trained model.

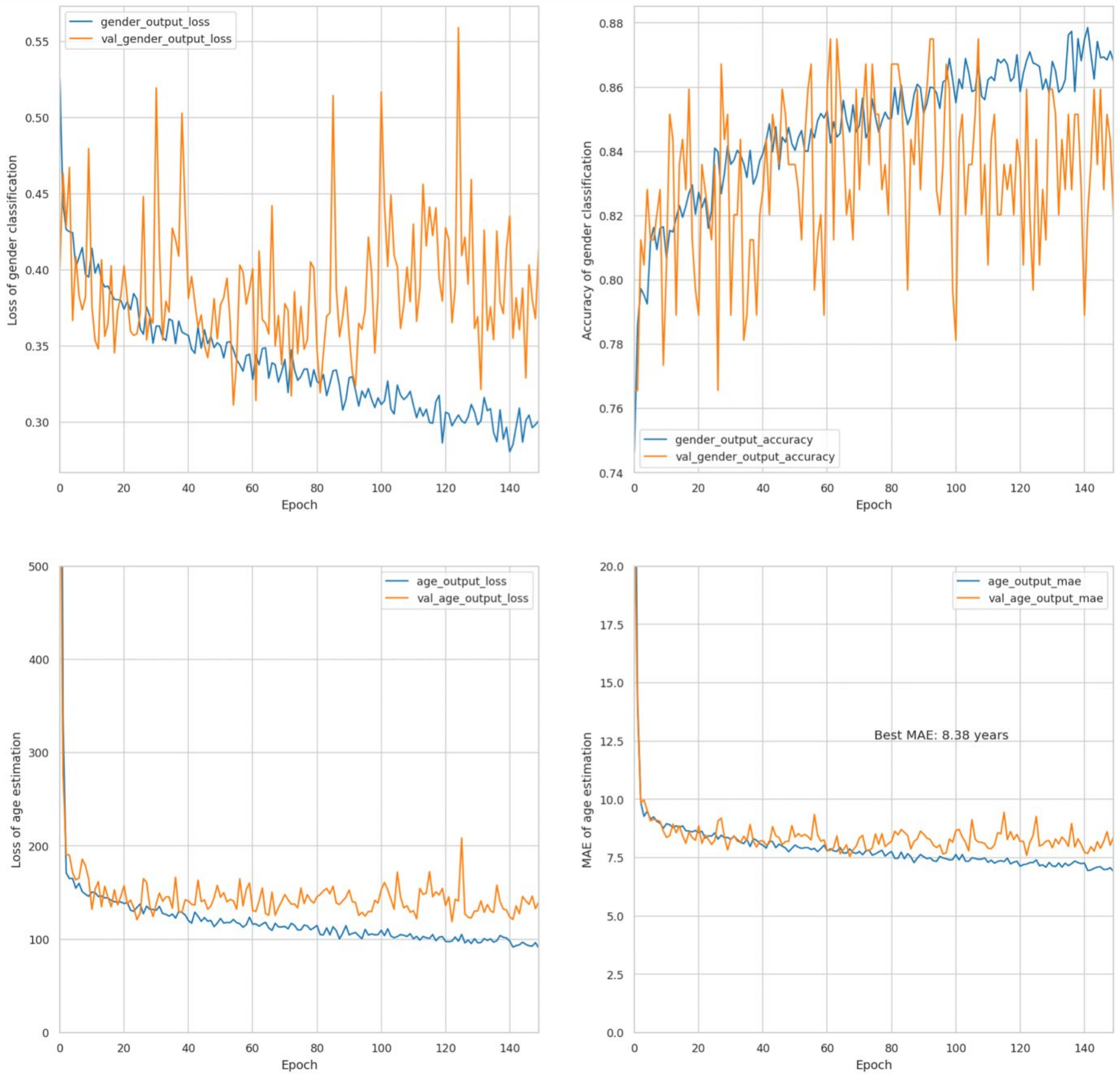
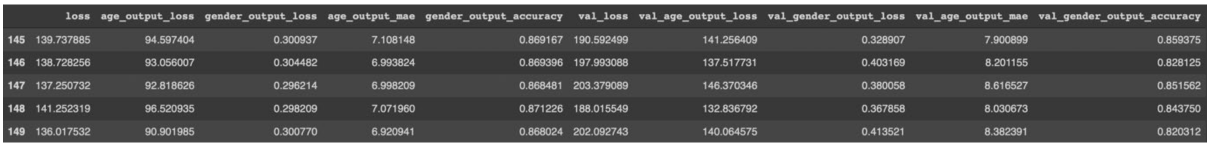


Figure 9: Training/Validation curves for Xception



	loss	age_output_loss	gender_output_loss	age_output_mae	gender_output_accuracy	val_loss	val_age_output_loss	val_gender_output_loss	val_age_output_mae	val_gender_output_accuracy
145	139.737885	94.597404	0.300937	7.108148	0.869167	190.592499	141.256409	0.328907	7.900899	0.859375
146	138.728256	93.056007	0.304482	6.993824	0.869396	197.993088	137.517731	0.403169	8.201155	0.828125
147	137.250732	92.818626	0.296214	6.998209	0.868481	203.379089	146.370346	0.380058	8.616527	0.851562
148	141.252319	96.520935	0.298209	7.071960	0.871226	188.015549	132.836792	0.367858	8.030673	0.843750
149	136.017532	90.901985	0.300770	6.920941	0.868024	202.092743	140.064575	0.413521	8.382391	0.820312

Figure 10: The age and gender loss.

Section 4: Summary and Discussion

For the test sets, our model achieved 92.2% gender classification accuracy and 4.60 years MAE for age estimation, while the pre-trained model performed significantly worse, getting 83.6% gender accuracy and 8.50 years MAE for age. This result was surprising but can be explained by the fact the Xception model (pre-trained) is for general object recognition and not solely face/age recognition. In comparison, our model has been adjusted for maximum return in face/age recognition, hence is specialised for this purpose and less generalisable.

For the validation sets, our model returned 5.65 year MAE for age and 94.5% gender accuracy. The pre-trained model had an age MAE of 8.38 years and 82.0% gender accuracy. These results are similar to the test set values, hence one can conclude overfitting was not an issue.

Following a similar architecture to the widely popular AlexNet, albeit scaled-down, proved to be very successful. Considering AlexNet achieved up to 81.4% accuracy in the ImageNet competition, our 92.2% gender accuracy is impressive, although these are not directly comparable. Xception's failings may be due in part to its aggressive downsampling of high-resolution images, potentially losing important information about the original image.

In this assignment, we successfully utilised the power of CNNs for image recognition and used our knowledge of the behaviour of convolutional, ReLU and pooling layers to create a model which outperformed the industry standard Xception by 8.6% for gender classification and an incredible 54% improvement for age estimation MAE. This can be explained by the fact Xception has 71 layers, compared to our model's 41 layers. Given Xception's increased model complexity, more training data is required to utilise its full potential. Otherwise, as seen in this project, Xception is overfitting the training data and hence reducing the accuracy of the test results. As age estimation is a regression problem, a very large sample would be needed to improve Xception's age accuracy.

Given more time, our model could have been improved further by testing more combinations of hidden layer architectures and carrying out greater data augmentation. Further research into attention layers may have also allowed us to adopt them more successfully.

Reference

[1] Detail of the AlexNet architecture. ResearchGate n.d. https://www.researchgate.net/figure/Detail-of-the-AlexNet-architecture_tbl1_332333495 (accessed March 20, 2022).

[2] Team K. Keras documentation: Xception n.d. https://keras.io/api/applications/xception/ (accessed March 20, 2022).

[3] Xception: Implementing from scratch using Tensorflow | by Arjun Sarkar | Towards Data Science n.d. https://towardsdatascience.com/xception-from-scratch-using-tensorflow-even-better-than-inception-940fb231ced9 (accessed March 20, 2022).

[4] Xception convolutional neural network - MATLAB xception - MathWorks United Kingdom n.d. https://uk.mathworks.com/help/deeplearning/ref/xception.html (accessed March 20, 2022).