University of Bath

CM50265

ML2 Coursework

Training of a CNN Built from Scratch and Through Transfer Learning

Authors - Group T

ID	Contribution
Ferdinand Krammer (219443933)	100
Toby Lewis-Atwell (179084636)	100
Thomas Ryder (219476757)	100
Douglas Tilley (209469615)	100

Reasons for the different weights: N/A

Do all the members agree with the above contributions? Yes

May 9, 2024

1 Introduction

Identification of a person's age or gender is a task that we, as humans, give little thought towards the underlying complexity of. Deep learning, but more specifically convolutional neural networks (CNNs), have revolutionised the field of computer vision. Their ability to extract hierarchies of features that are spatially invariant, but also reduce the impact of issues associated with the curse of dimensionality (large dimensionality from a flattened image), have overcome some of the issues that plagued image classification approaches previously.

Using a subset (5000 images) of the UTKFace dataset, containing face images with corresponding age and gender labels, we utilise a CNN to predict both these attributes on unseen images. For this, we take two approaches. First, we develop our own custom CNN architecture from scratch through a hyperparameter optimisation procedure. Second, we implement transfer learning, making use of pre-trained CNNs with demonstrable performance, with some fine-tuning to our own data.

2 My Own CNN

2.1 Data Preprocessing

Initially, the dataset was checked for images that were obviously not faces, and two such images (shown in Figure 1) were found and removed from the training data.

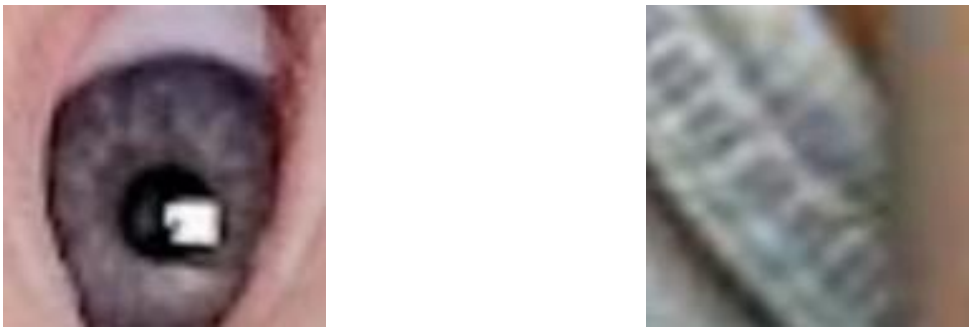

Figure 1: Images not of faces that were removed from the training data.

The distribution of age over the two genders was visualised (see Figure 2), as a sanity check that the distribution of age was roughly the same for both genders.

Table 1 shows the splitting of the dataset for hyperparameter tuning. Due to the limited size of the dataset, data augmentation was necessary to help reduce overfitting. This was performed through the use of the Keras ImageDataGenerator() class and its flow_from_dataframe() method. The augmentation steps were:

- Flipping along the y axis
- Shifting horizontally or vertically
- Zooming in
- Rotating by up to 10 degrees clockwise or anti-clockwise.

2.2 Modelling Approach

As per the assignment requirements, a single CNN was used to predict both age and gender. To do this, two modifications had to be taken compared to a standard CNN:

- *Output layers* - the CNN was built with two output layers: (1) a layer with a linear activation function for predicting age and (2) a layer with a sigmoid activation function for predicting gender.

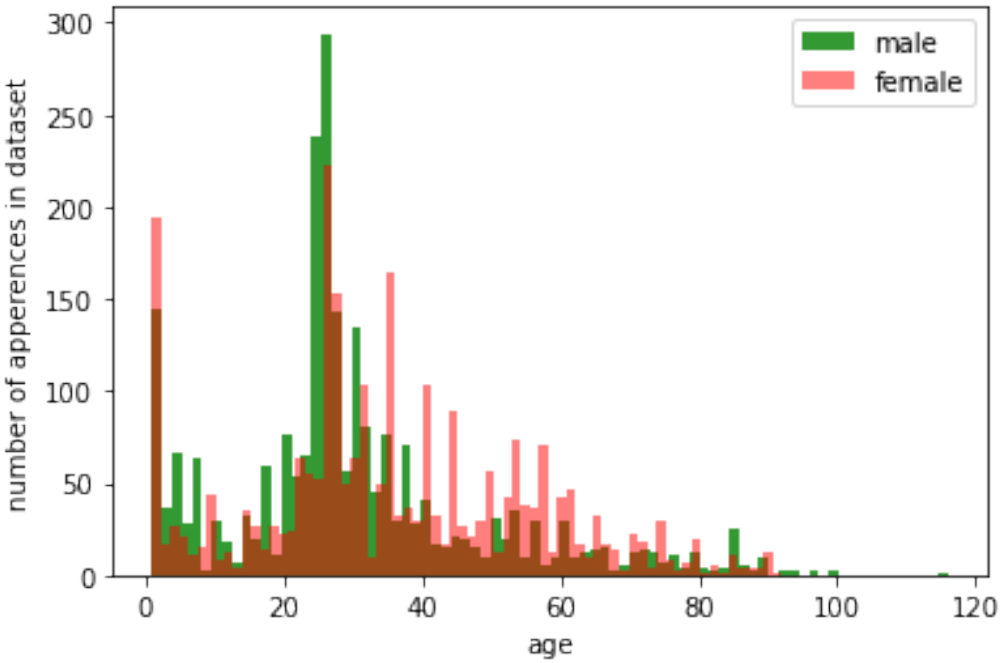

Figure 2: The distribution of ages for each of the two genders.

Table 1: Dataset splits. Note the total number of images is 4998 due to the two removed images.

Name	Percentage split	Image Count	Description
Train	80	4000	For training the model.
Val 1	10	500	For evaluating the model during training with early stopping.
Val 2	10	498	For evaluating a trained model in the grid search.

- *Splitting of the convolutional base* - it was assumed the features for predicting age and gender would be different, hence the network was split in two at the convolutional base, such that different features could be extracted from the images for the two different tasks.

2.3 Hyperparameter Optimisation

Obtaining the maximum possible performance from a CNN is dependent on finding the optimal hyperparameters for the dataset at hand. A small grid search was carried out, which involved training multiple architectures, and selecting the best based on the performance on held out validation data. The hyperparameters that were selected for tuning were those defining the architecture of the CNN:

- The number of convolutional layers - drawing on inspiration from the VGG architectures, the convolutional base was a repeating structure of two 3×3 convolutions followed by a max pooling layer. In addition, all architectures had $64 \rightarrow 128 \rightarrow 256 \rightarrow 512$ filters for the first four pairs of kernel convolutions, and 512 for every pair of layers after.
- The number of dense layers and the number of units in each dense layer.
- The split point in the convolutional base. In theory, splitting earlier should improve performance, as features specific to the individual tasks can be extracted at an earlier level. However, this increases compute time as the number of learnable parameters increases. There could also potentially be some benefit to sharing the earliest (and hence lowest-level) features.

In determining the best model, accuracy and mean absolute error (MAE) were computed on the predictions made on the second validation set for gender and age prediction, respectively.

Table 2 shows each of the 24 combinations of hyperparameters that were tested, along with the gender accuracy and age MAE on the second held-out validation set. CNN 10 (shown in Figure 3) was selected, since it offered one of the best balances between gender accuracy and age MAE.

Table 2: The hyperparameters for each of the models trained in the grid search. Columns are the number of pairs of kernel convolutions followed by max pooling layers, the number of dense layers following the convolutional layers, the number of nodes in each of the dense layers, the number of pairs of convoluational layers until the network was split into two branches, the accuracy of the network for gender prediction on the Val 2 validation set and the MAE of the network for age prediction on the Val 2 validation set.

Model Number	Convolutions	Dense Layers	Hidden Nodes	Split Point	Accuracy	MAE
CNN 1	3	1	100	3	0.8996	6.321
CNN 2	3	1	300	3	0.8855	6.661
CNN 3	3	3	100	3	0.8795	6.082
CNN 4	3	3	300	3	0.8635	6.137
CNN 5	5	1	100	3	0.8976	5.664
CNN 6	5	1	300	3	0.9056	5.868
CNN 7	5	3	100	3	0.8835	6.238
CNN 8	5	3	300	3	0.8936	5.946
CNN 9	5	3	100	1	0.8896	6.031
CNN 10	**5**	**3**	**300**	**1**	**0.9036**	**5.715**
CNN 11	5	3	100	4	0.9056	6.262
CNN 12	5	3	300	4	0.8876	6.103
CNN 13	3	2	200	3	0.8916	5.91
CNN 14	3	2	200	1	0.9016	6.18
CNN 15	3	5	200	3	0.8735	6.228
CNN 16	5	5	200	1	0.8936	6.196
CNN 17	5	5	200	5	0.8273	5.955
CNN 18	2	2	150	1	0.8815	6.377
CNN 19	5	2	100	3	0.9197	6.143
CNN 20	5	2	300	3	0.8876	5.851
CNN 21	5	2	200	4	0.8635	6.299
CNN 22	5	4	200	3	0.8876	5.943
CNN 23	3	2	200	1	0.8594	6.703
CNN 24	3	4	100	1	0.8916	5.96

2.4 Additional Overfitting Measures

Further measures were taken to reduce overfitting to the training set. To reduce compute time, parameter values were fixed with sensible values (determined through trial-and-error) rather than in-

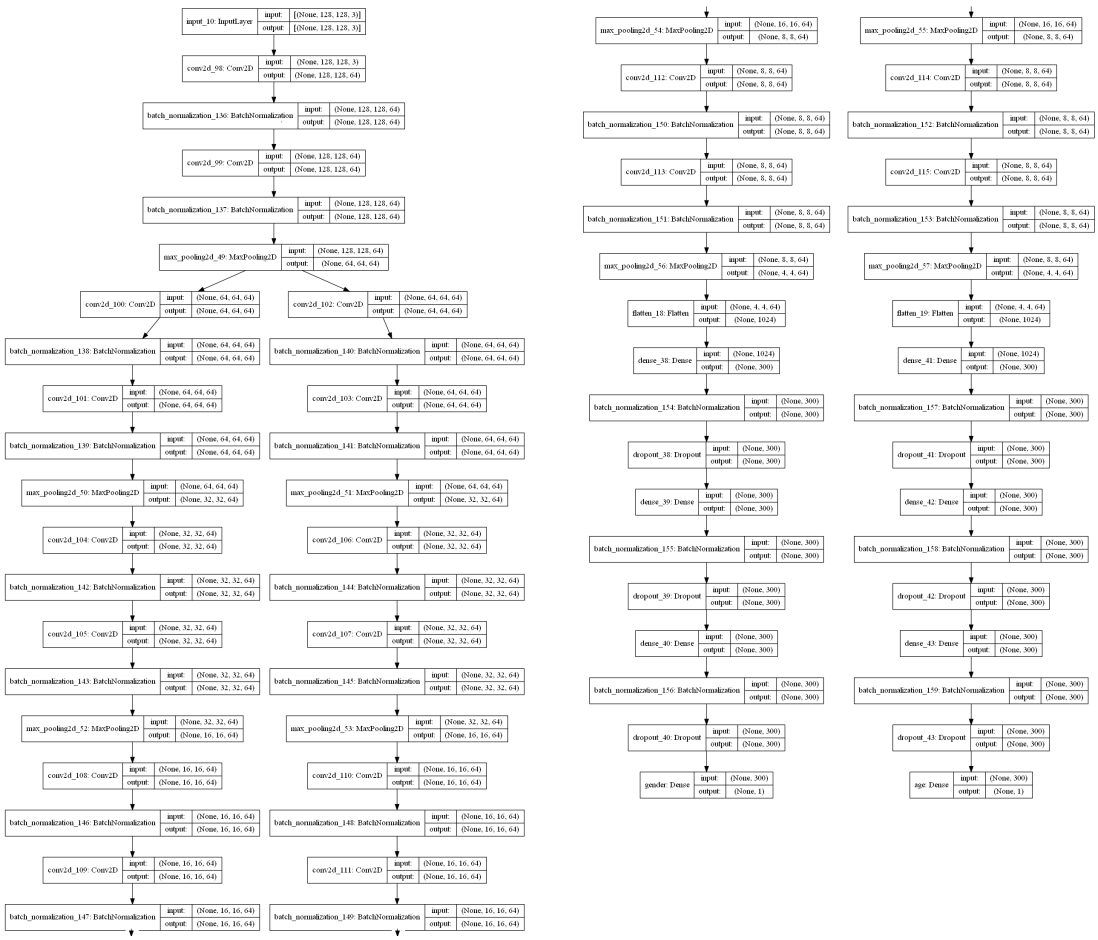

Figure 3: The architecture of CNN 10.

cluded in the grid search. The measures taken were:

- Batch normalisation between all convolutional and dense layers.
- L2 regularisation in dense layers with $\lambda = 10^{-5}$.
- Dropout in dense layers with $p(\text{drop unit}) = 0.2$.
- Early stopping with minimum improvement of 0.1 and patience of 50 epochs.

2.5 Final Performance

CNN 10 was retrained with a training set of size 4499 images and tested with a held-out validation set of size 499 images. The final part of the training output for this network is shown in the Appendix Section A. On the validation set, this model achieved a final accuracy of 90.98% for gender prediction and a MAE of 5.949 years for age prediction. The learning curves for this model's training are shown in Figures 4 and 5.

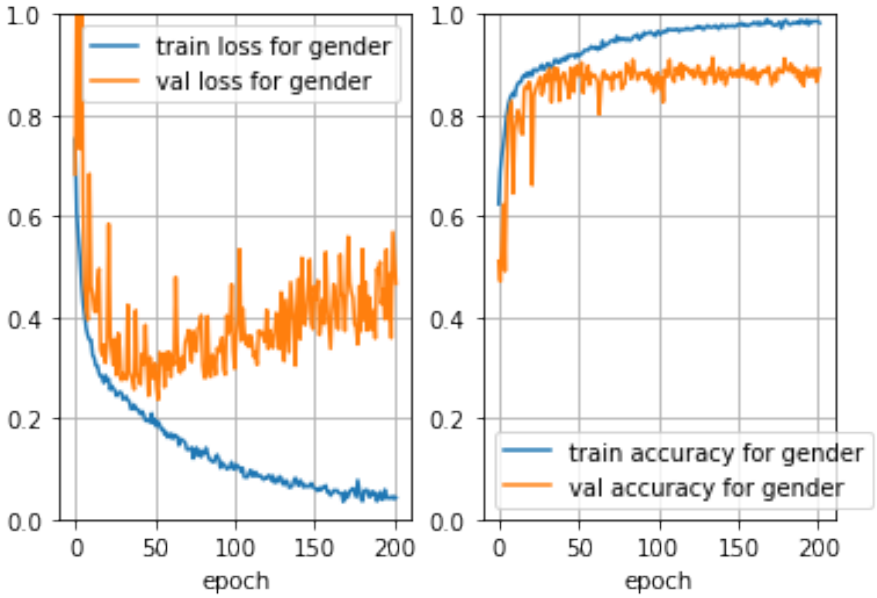

Figure 4: The learning curves for gender prediction from CNN 10. Note the early stopping meant that the final weights used were from before overfitting started to occur at around 50 epochs.

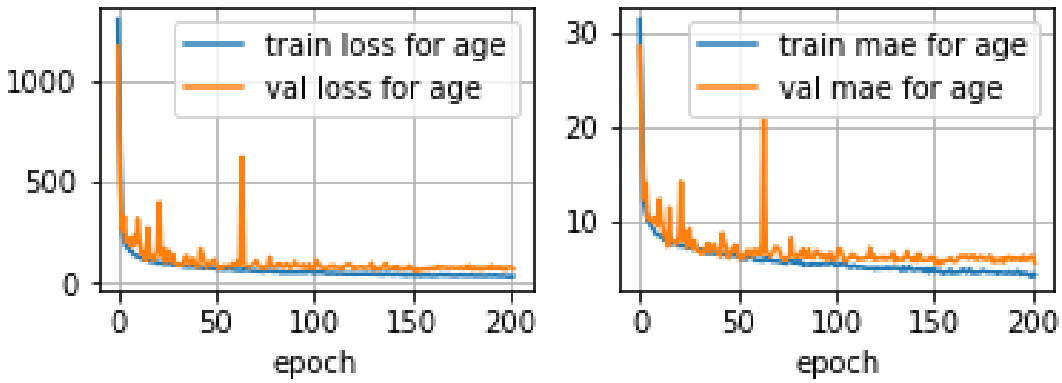

Figure 5: The learning curves for age prediction from CNN 10. Note the early stopping meant that the final weights used were from before overfitting started to occur at around 50 epochs.

3 Transfer learning and fine-tuning

Six different pre-trained CNNs were tested for transfer learning. For each, the dense structure of the models were replaced with new dense layers, batch normalisation, dropout, and split at the start of the dense layers for the regression and classification tasks. During training of the dense layers, the convolutional layers can be made trainable and refined for our specific data. Performance was best when the only the final layers were unfrozen, with the rest of the convolutional structure remaining frozen. Two trainable convolutional blocks were added to the top model, but the performance of the classifier and regression dropped and hence these additional layers were removed. Reasonable starting hyperparameters for the dense structure of each pre-trained CNN were selected, inspired on the paper by Smith and Chen[1], though with some further trial-and-error.

Table 3 shows the results of a brief comparison of the six models, each with the same dense layers and trained for 10 epochs. The model selected for further training was Xception [2] which is a 71 Layer pre-trained network with the weights from ImageNet. The model is a stack of 36 convolutional layers to obtain feature extraction and has dropout values at 0.5 with Batch Normalization layers at each stage. The model structure can be found in Figure 7.

Inspired by Li et al [3], we experimented with various combinations of dense structures. The hyperparameters of the dense structure were optimised through another grid search operation, with results shown in Table 4. However, due to the compute required with these larger models, we couldn't be as exhaustive as before. With the best structure, we performed further hyperparameter optimisation on learning rate and number of dense units with the hyperband algorithm [4].

One approach considered was weighting the losses for both classification and regression tasks. However, this had a negligible effect on the classification.

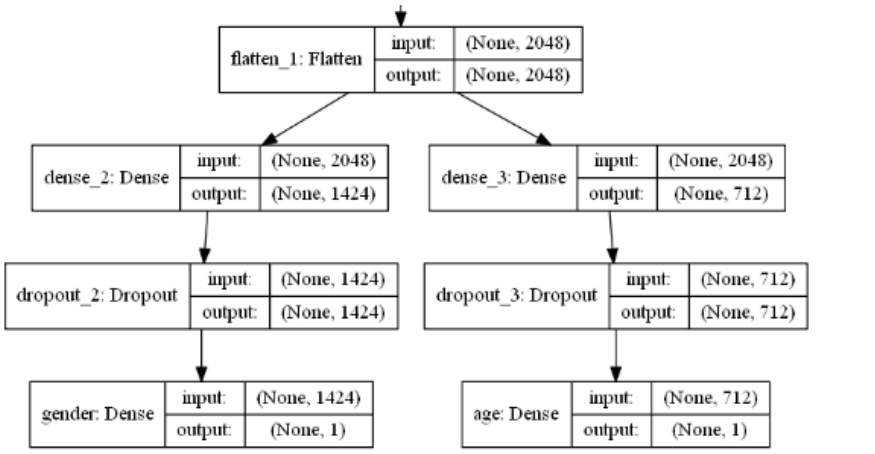

Figure 6: The top model with dense layers and dropout included.

Table 3: A comparison of the different pre-trained models tested, comparing VGG16, VGG19, Xception, ResNet50V2, EfficientNetB4 and InceptionV3, initially selected from the Keras applications API [2], [5]–[8]

Model	Gender Result	MAE Age Result
EfficientNetB4	0.52906	15.1119
VGG19	0.75351	11.8049
VGG16	0.77555	11.9737
InceptionV3	0.79359	9.2118
ResNet50V2	0.81363	9.7123
Xception	0.83968	9.4344

Different techniques were tested for freezing and unfreezing layers of the pre-trained model, fine-tuning was initially done by freezing whole convolutional blocks but leaving the Batch Normalization un-frozen. After some trial and error, it was found that freezing only 1 or 2 layers provided the largest improvement in accuracy. Figures 8 and 9 show the learning curves for the training and fine-tuning of the best transfer learning model. The final part of the training output for this network is shown in the Appendix Section B. The final model achieved an accuracy of 83.97% for gender prediction and a MAE of 5.639 years on the held-out validation set.

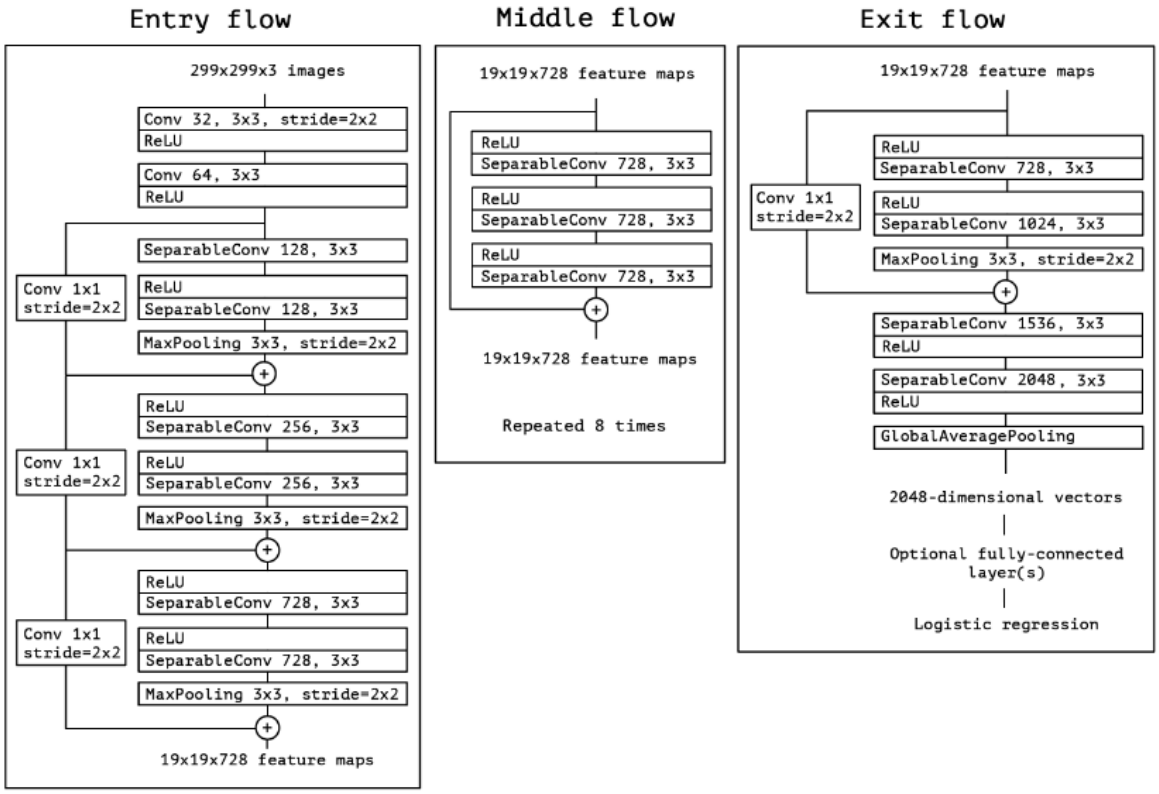

Figure 7: The original Xception Structure. Reproduced from reference [2].

Table 4: A comparison of the top model concatenated to the transfer learning model with the different model orderings.

Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Gender Classification Result
Dense	Dense	Dropout	Classifier		0.81882
Dense	Batch Normalization	Dropout	Classifier		0.81982
Batch Normalization	Dense	Dropout	Classifier		0.82182
Dense	Dropout	Batch Normalization	Classifier		0.82182
Dense	Dropout	Batch Normalization	Dropout	Classifier	0.83083
Dense	Batch Normalization	Classifier			0.82583
Dense	Dropout	Classifier			0.83483

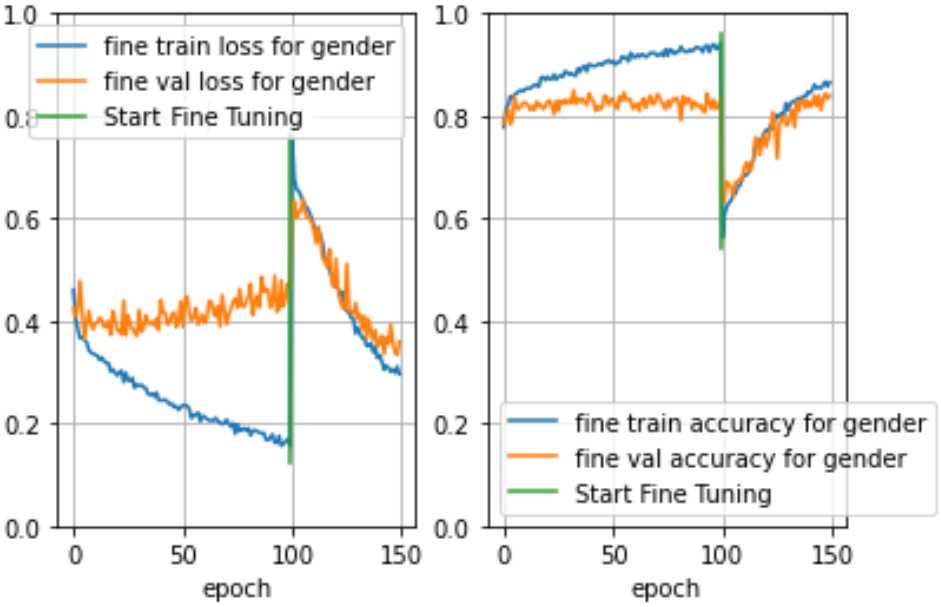

Figure 8: The learning curves for gender prediction from the transfer learning model. Note the early stopping meant that the final weights used were from before overfitting started to occur at around 50 epochs.

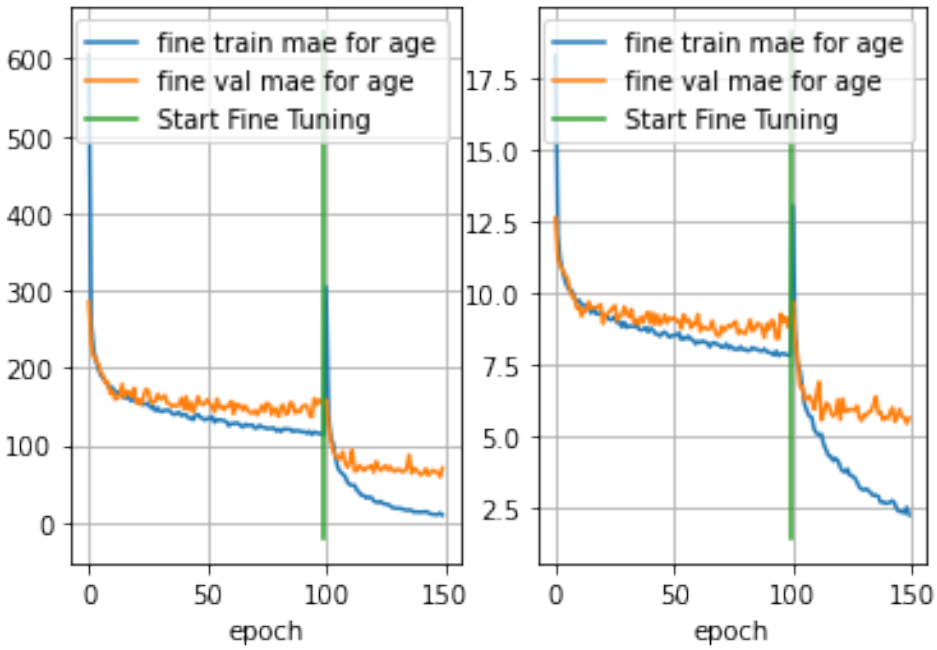

Figure 9: The learning curves for gender prediction from transfer learning model. Note the early stopping meant that the final weights used were from before overfitting started to occur at around 50 epochs.

4 Summary and Discussion

Comparing a self-built CNN to a pre-trained network is a comparison of a specifically built network to a generic one. By utilising a pre-trained network, the performance and application of a CNN can, in theory, be rapidly performed with a small number of epochs. It was found that the use of a pre-trained model resulted in lower accuracy (90.98% vs 83.97%) but slightly lower MAE (5.949 vs 5.639) than was achieved with a custom model. Transfer learning models also have the advantage over models built from scratch in that they do not require as much training time. The images in Figure 10 show both good and bad predictions of age and gender, as given by the self built CNN. When compared to the outputs from the pre-trained network as given in Figure 11, both networks appear to make similar predictions; as seen from the accuracy and MAE, the predictions for the self built model are slightly better.

There is potential to increase the network performance further through the use of cross-stitch networks as presented by Misra [9] which allow for the sharing of more specific features for a given output. However, since the GPUs provided by Colab have limited GPU RAM and run times, we were unable to implement cross-stitching with high performance in this work. The plots for the loss, accuracy and MAE are given in the code-appendix Figures 12 and 13 with the cross-stitching only being implemented in the fully-connected layers.

Figure 10: Selection of predictions from model A to show cases where the model was able to predict both the gender and age well as well as when it was unable to.

Figure 11: Selection of predictions from model B to show cases where the model was able to predict both the gender and age well as well as when it was unable to.

The .h5 files for the trained models may be found at the following links:

- *age_gender_A.h5* - `https://drive.google.com/file/d/1cmAwjXSK6G654Y79adkqp9X5zo5_H--C/view?usp=sharing`

- *age_gender_B.h5* - https://drive.google.com/file/d/1NW39JQqDEIWLO6Q5WCYtFQ7rGQbXUuMy/view?usp=sharing

References

[1] P. Smith and C. Chen, "Transfer learning with deep cnns for gender recognition and age estimation," in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 2564–2571.

[2] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[3] X. Li, S. Chen, X. Hu, and J. Yang, "Understanding the disharmony between dropout and batch normalization by variance shift," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2677–2685. DOI: 10.1109/CVPR.2019.00279.

[4] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

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

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, Springer, 2016, pp. 630–645.

[7] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[9] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3994–4003.

A Training Output from the Custom CNN

```
y: 0.9847 - age_mae: 4.5723 - val_loss: 74.0096 - val_gender_loss: 0.4126 - val_age_loss: 73.4822 - val_gender_accuracy: 0.8737 - val
_age_mae: 6.2634
Epoch 189/1000
141/141 [==============================] - 22s 158ms/step - loss: 37.4121 - gender_loss: 0.0526 - age_loss: 37.2445 - gender_accurac
y: 0.9807 - age_mae: 4.7837 - val_loss: 79.2801 - val_gender_loss: 0.3581 - val_age_loss: 78.8067 - val_gender_accuracy: 0.8878 - val
_age_mae: 6.3636
Epoch 190/1000
141/141 [==============================] - 14s 96ms/step - loss: 36.4860 - gender_loss: 0.0358 - age_loss: 36.3347 - gender_accuracy:
0.9878 - age_mae: 4.6682 - val_loss: 70.3724 - val_gender_loss: 0.4948 - val_age_loss: 69.7617 - val_gender_accuracy: 0.8657 - val_ag
e_mae: 6.0162
Epoch 191/1000
141/141 [==============================] - 13s 95ms/step - loss: 34.2367 - gender_loss: 0.0435 - age_loss: 34.0771 - gender_accuracy:
0.9864 - age_mae: 4.5582 - val_loss: 90.6833 - val_gender_loss: 0.4881 - val_age_loss: 90.0788 - val_gender_accuracy: 0.8838 - val_ag
e_mae: 6.2961
Epoch 192/1000
141/141 [==============================] - 13s 95ms/step - loss: 35.6611 - gender_loss: 0.0535 - age_loss: 35.4908 - gender_accuracy:
0.9816 - age_mae: 4.6152 - val_loss: 75.5442 - val_gender_loss: 0.5096 - val_age_loss: 74.9175 - val_gender_accuracy: 0.8577 - val_ag
e_mae: 6.1750
Epoch 193/1000
141/141 [==============================] - 14s 95ms/step - loss: 38.6940 - gender_loss: 0.0387 - age_loss: 38.5380 - gender_accuracy:
0.9856 - age_mae: 4.8468 - val_loss: 72.0281 - val_gender_loss: 0.4260 - val_age_loss: 71.4846 - val_gender_accuracy: 0.8818 - val_ag
e_mae: 6.1610
Epoch 194/1000
141/141 [==============================] - 13s 95ms/step - loss: 36.3513 - gender_loss: 0.0399 - age_loss: 36.1937 - gender_accuracy:
0.9878 - age_mae: 4.6774 - val_loss: 78.5391 - val_gender_loss: 0.4537 - val_age_loss: 77.9676 - val_gender_accuracy: 0.8958 - val_ag
e_mae: 6.2457
Epoch 195/1000
141/141 [==============================] - 14s 96ms/step - loss: 33.4488 - gender_loss: 0.0585 - age_loss: 33.2720 - gender_accuracy:
0.9787 - age_mae: 4.4685 - val_loss: 71.3493 - val_gender_loss: 0.3961 - val_age_loss: 70.8346 - val_gender_accuracy: 0.8778 - val_ag
e_mae: 6.0021
Epoch 196/1000
141/141 [==============================] - 14s 96ms/step - loss: 35.9395 - gender_loss: 0.0418 - age_loss: 35.7788 - gender_accuracy:
0.9869 - age_mae: 4.6320 - val_loss: 80.1395 - val_gender_loss: 0.5339 - val_age_loss: 79.4864 - val_gender_accuracy: 0.8758 - val_ag
e_mae: 6.4171
Epoch 197/1000
141/141 [==============================] - 14s 96ms/step - loss: 35.5633 - gender_loss: 0.0445 - age_loss: 35.3994 - gender_accuracy:
0.9836 - age_mae: 4.6169 - val_loss: 68.4947 - val_gender_loss: 0.3965 - val_age_loss: 67.9786 - val_gender_accuracy: 0.8898 - val_ag
e_mae: 6.1515
Epoch 198/1000
141/141 [==============================] - 13s 95ms/step - loss: 33.3884 - gender_loss: 0.0411 - age_loss: 33.2275 - gender_accuracy:
0.9856 - age_mae: 4.5073 - val_loss: 66.2704 - val_gender_loss: 0.4857 - val_age_loss: 65.6647 - val_gender_accuracy: 0.8778 - val_ag
e_mae: 6.0516
Epoch 199/1000
141/141 [==============================] - 14s 96ms/step - loss: 31.3645 - gender_loss: 0.0419 - age_loss: 31.2024 - gender_accuracy:
0.9856 - age_mae: 4.3525 - val_loss: 80.1507 - val_gender_loss: 0.3589 - val_age_loss: 79.6713 - val_gender_accuracy: 0.8918 - val_ag
e_mae: 6.3065
Epoch 200/1000
141/141 [==============================] - 13s 95ms/step - loss: 34.6782 - gender_loss: 0.0457 - age_loss: 34.5117 - gender_accuracy:
0.9856 - age_mae: 4.5435 - val_loss: 79.6341 - val_gender_loss: 0.5685 - val_age_loss: 78.9444 - val_gender_accuracy: 0.8657 - val_ag
e_mae: 6.2658
Epoch 201/1000
141/141 [==============================] - 14s 96ms/step - loss: 30.1481 - gender_loss: 0.0401 - age_loss: 29.9865 - gender_accuracy:
0.9858 - age_mae: 4.2333 - val_loss: 78.4012 - val_gender_loss: 0.5166 - val_age_loss: 77.7629 - val_gender_accuracy: 0.8737 - val_ag
e_mae: 6.4520
Epoch 202/1000
141/141 [==============================] - 13s 94ms/step - loss: 33.8899 - gender_loss: 0.0428 - age_loss: 33.7250 - gender_accuracy:
0.9824 - age_mae: 4.4579 - val_loss: 69.9704 - val_gender_loss: 0.4663 - val_age_loss: 69.3816 - val_gender_accuracy: 0.8918 - val_ag
e_mae: 5.7560
```

B Training Output from the Transfer Learning Model

```
Epoch 00041: val_gender_accuracy did not improve from 0.83166
Epoch 42/50
141/141 [==============================] - 17s 120ms/step - loss: 13.4563 - gender_loss: 0.3209 - age_loss: 13.1354 - gender_accurac
y: 0.8504 - age_mae: 2.7134 - val_loss: 64.1204 - val_gender_loss: 0.3713 - val_age_loss: 63.7492 - val_gender_accuracy: 0.8216 - val
_age_mae: 5.8433

Epoch 00042: val_gender_accuracy did not improve from 0.83166
Epoch 43/50
141/141 [==============================] - 17s 122ms/step - loss: 14.1930 - gender_loss: 0.3052 - age_loss: 13.8878 - gender_accurac
y: 0.8613 - age_mae: 2.7647 - val_loss: 69.4923 - val_gender_loss: 0.3353 - val_age_loss: 69.1570 - val_gender_accuracy: 0.8397 - val
_age_mae: 6.0079

Epoch 00043: val_gender_accuracy improved from 0.83166 to 0.83968, saving model to Checkpoints\ftmodel.ckpt
Epoch 44/50
141/141 [==============================] - 17s 120ms/step - loss: 12.8240 - gender_loss: 0.3126 - age_loss: 12.5114 - gender_accurac
y: 0.8542 - age_mae: 2.6548 - val_loss: 60.9262 - val_gender_loss: 0.3679 - val_age_loss: 60.5583 - val_gender_accuracy: 0.8196 - val
_age_mae: 5.5220

Epoch 00044: val_gender_accuracy did not improve from 0.83968
Epoch 45/50
141/141 [==============================] - 17s 120ms/step - loss: 11.0176 - gender_loss: 0.3093 - age_loss: 10.7083 - gender_accurac
y: 0.8560 - age_mae: 2.4405 - val_loss: 65.0747 - val_gender_loss: 0.3569 - val_age_loss: 64.7178 - val_gender_accuracy: 0.8397 - val
_age_mae: 5.7210

Epoch 00045: val_gender_accuracy did not improve from 0.83968
Epoch 46/50
141/141 [==============================] - 17s 122ms/step - loss: 10.4485 - gender_loss: 0.3010 - age_loss: 10.1475 - gender_accurac
y: 0.8540 - age_mae: 2.3874 - val_loss: 67.6571 - val_gender_loss: 0.3927 - val_age_loss: 67.2645 - val_gender_accuracy: 0.8116 - val
_age_mae: 5.9579

Epoch 00046: val_gender_accuracy did not improve from 0.83968
Epoch 47/50
141/141 [==============================] - 17s 120ms/step - loss: 10.3240 - gender_loss: 0.3051 - age_loss: 10.0189 - gender_accurac
y: 0.8604 - age_mae: 2.3801 - val_loss: 65.2510 - val_gender_loss: 0.3594 - val_age_loss: 64.8916 - val_gender_accuracy: 0.8317 - val
_age_mae: 5.6927

Epoch 00047: val_gender_accuracy did not improve from 0.83968
Epoch 48/50
141/141 [==============================] - 17s 120ms/step - loss: 10.0327 - gender_loss: 0.2989 - age_loss: 9.7339 - gender_accuracy:
0.8669 - age_mae: 2.3320 - val_loss: 65.1090 - val_gender_loss: 0.3381 - val_age_loss: 64.7709 - val_gender_accuracy: 0.8437 - val_ag
e_mae: 5.6583

Epoch 00048: val_gender_accuracy improved from 0.83968 to 0.84369, saving model to Checkpoints\ftmodel.ckpt
Epoch 49/50
141/141 [==============================] - 17s 121ms/step - loss: 12.1076 - gender_loss: 0.3114 - age_loss: 11.7961 - gender_accurac
y: 0.8571 - age_mae: 2.5292 - val_loss: 58.8399 - val_gender_loss: 0.3338 - val_age_loss: 58.5061 - val_gender_accuracy: 0.8337 - val
_age_mae: 5.4558

Epoch 00049: val_gender_accuracy did not improve from 0.84369
Epoch 50/50
141/141 [==============================] - 17s 122ms/step - loss: 9.4867 - gender_loss: 0.2962 - age_loss: 9.1904 - gender_accuracy:
0.8655 - age_mae: 2.2603 - val_loss: 70.4980 - val_gender_loss: 0.3595 - val_age_loss: 70.1385 - val_gender_accuracy: 0.8397 - val_ag
e_mae: 5.6534
```

C Code Appendix for Cross stitching

```
# best model reproduced with cross stitching in the FC layers only as when
# implementing cross stitching in the convectional layers there was not
# enough ram for it to run

class cross_stich(keras.layers.Layer):
    def __init__(self):
        super(cross_stich,self).__init__()


    def call(self, inpits):
        inputA, inputB=inpits
        flatInputA=Flatten()(inputA)
        flatInputB=Flatten()(inputB)
        inputarray=tf.concat((flatInputA,flatInputB),axis=1)
        output=tf.matmul(inputarray,self.alpha)
        inputA_shape = [-1 if dimen is None else dimen for dimen in inputA.shape]
        inputB_shape = [-1 if dimen is None else dimen for dimen in inputB.shape]
        outputA=tf.reshape(output[:,:flatInputA.shape[1]],inputA_shape)
        outputB=tf.reshape(output[:,flatInputB.shape[1]:],inputB_shape)
        return outputA, outputB

    def build(self,input_shape):
        inShape=input_shape[1]
        dim=1
        for i in range(len(inShape)):
            if inShape[i] ==None:
                dim*=1
            else:
                dim*=inShape[i]


        self.alpha=self.add_weight(
                                    shape=(dim*2,dim*2),
                                    initializer='identity',
                                    trainable=True)

cross_stich_layer1=cross_stich()
cross_stich_layer2=cross_stich()
cross_stich_layer3=cross_stich()
def modl_V3():
    input_layer=x=Input(shape=(128,128,3))
    # layer1
    x= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(x)
    x= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(x)
    x= BatchNormalization()(x)
    x= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(x)
    #layer 2

    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(x)
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= BatchNormalization()(a)
    a= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(a)

    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(x)
    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= BatchNormalization()(g)
    g= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(g)

    #layer 3
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= BatchNormalization()(a)
    a= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(a)

```

```
    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= BatchNormalization()(g)
    g= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(g)

    #layer 4
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= BatchNormalization()(a)
    a= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(a)

    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= BatchNormalization()(g)
    g= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(g)

    #layer 5
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(a)
    a= BatchNormalization()(a)
    a= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(a)

    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= Conv2D(64, (3, 3), activation='relu', padding='same', strides=1)(g)
    g= BatchNormalization()(g)
    g= MaxPooling2D(pool_size=(2, 2), strides=2, padding='same')(g)

    a=Flatten()(a)
    g=Flatten()(g)

    #fc1
    a = Dense(300,activation='relu')(a)
    a = Dropout(0.2)(a)
    a = BatchNormalization()(a)

    g = Dense(300,activation='relu')(g)
    g = Dropout(0.2)(g)
    g = BatchNormalization()(g)

    a,g=cross_stich_layer1([a,g])

    #fc 2
    a = Dense(300,activation='relu')(a)
    a = Dropout(0.2)(a)
    a = BatchNormalization()(a)

    g = Dense(300,activation='relu')(g)
    g = Dropout(0.2)(g)
    g = BatchNormalization()(g)

    a,g=cross_stich_layer2([a,g])

    #fc3

    a = Dense(300,activation='relu')(a)
    a = Dropout(0.2)(a)
    g = Dense(300,activation='relu')(g)
    g = Dropout(0.2)(g)

    a,g=cross_stich_layer3([a,g])

    age = Dense(1,activation='linear', name = 'age')(a)

    gender = Dense(1, activation='sigmoid', name='gender')(g)

    model = Model(inputs=input_layer, outputs =[gender,age])
```

```
131
132     model.compile(loss={'gender':'binary_crossentropy', 'age':'mse'}, optimizer=Adam(
        learning_rate=1e-4),
133                   metrics={'gender':'accuracy', 'age':'mae'},loss_weights={'gender'
        :100, 'age':1})
134     return model
135
136 model_3 = modl_V3()
137 model_3.summary()
138 plot_model(model_3, show_shapes=True)
139
140
141 early_stopping_callback = EarlyStopping(monitor='val_loss', min_delta=0.01, patience
        =40, restore_best_weights=True)
142 history = model_3.fit(train_datagen, epochs=400, validation_data=val_datagen,
143                       callbacks=[early_stopping_callback],verbose=2)
```

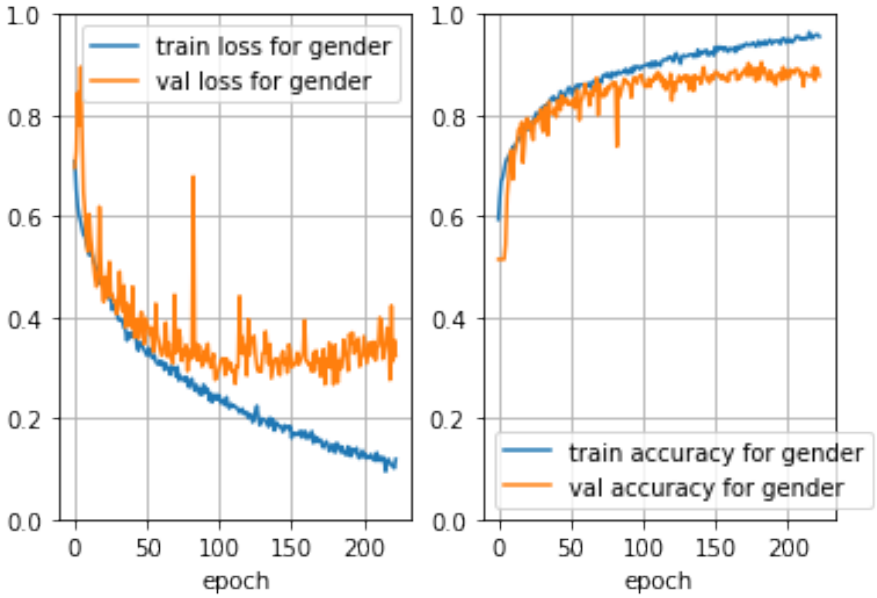

Figure 12: Loss and accuracy training curves for gender predictions from the cross-stitch network.

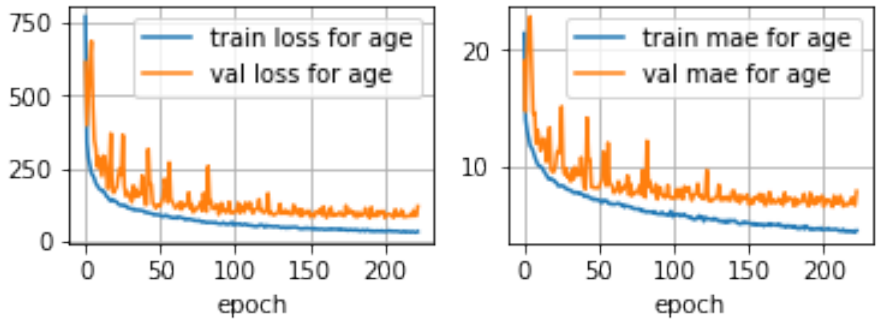

Figure 13: Loss and MAE training curves for age predictions from the cross-stitch network.