Age-gender classification challenge

Prepared by Denis Chainikov

Multi output model

Preprocessing

Firstly all the images of UTKFace dataset were resized from 200 x 200 to 128 x 128 and then they were formatted to numpy array for the following utilisation. Dataset was separated on three parts: train, validation and test. Age labels were changed from [1, 116] to [0,1] to solve this problem like regression one.

Model

The architecture of the model is mainly based on convolutional layers, just in the end there are two dense layers for age and gender (regression and binary classification). For gender binary classification, model uses Binary Cross Entropy as a loss and Accuracy as a metrics for validation. For age regression, model uses Mean Squared Error loss and Mean Absolute error as a metrics for validation. It's better to use regression for age problem cause age classes are very close to each other (45 and 46 years for example). Also loss weight coefficient for age loss is multiplied by two in order to make it comparable to gender loss.

The number of layers and filters were chosen by experimenting different architectures.

Model	3 CL 128 64 32	4 CL 128 64 32 16	4 CL 128 64 32 32	4 CL 128 64 64 32	5 CL 128 64 64 32 32
Test Loss	0.2924	0.2963	0.2865	0.3969	0.6172

Next question was about the order of Batch Normalisation (BN) and Relu activation function. The model with BN followed by Relu showed better results.

Order	BN -> Relu	Relu -> BN
Test Loss	0.3031	0.5348

The idea to put a Relu activation function at the last neuron for age regression hasn't given a good result. The idea behind was to force the neuron to give only the range of values between [0, 1].

Last neuron age Relu	True	False
Test Loss	0.3031	0.2924

The number of neurons in the biggest Dense layer was also tested and found that 512 neurons are optimal.

Dense neurons	512	1024
Test Loss	0.2796	0.3145

Next optimization was about dropout. In this model dropout is used just before the dense layer (512) as a regularisation technique. So dropout rate 0.6 seems optimal in our model.

Dropout rate	0.4	0.5	0.6	0.7
Test Loss	0.2717	0.2796	0.2472	0.2589

Another point which can change the performance dramatically is a choice of optimizer. For all of calculations before RMSProp has been used, but what about his newer version ADAM optimizer?

Optimizer	RMSProp	Adam
Test Loss	0.2472	0.2352

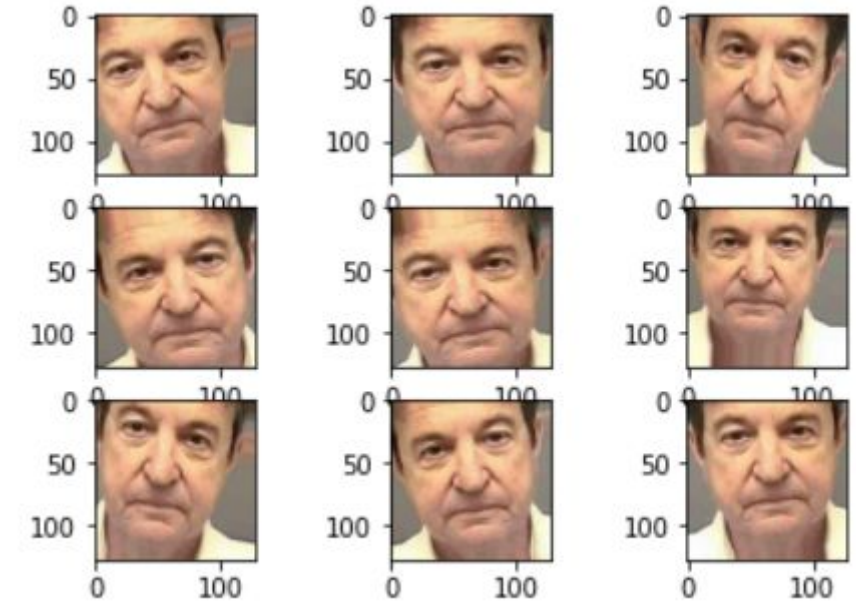
So ADAM optimizer helped us to win some points!

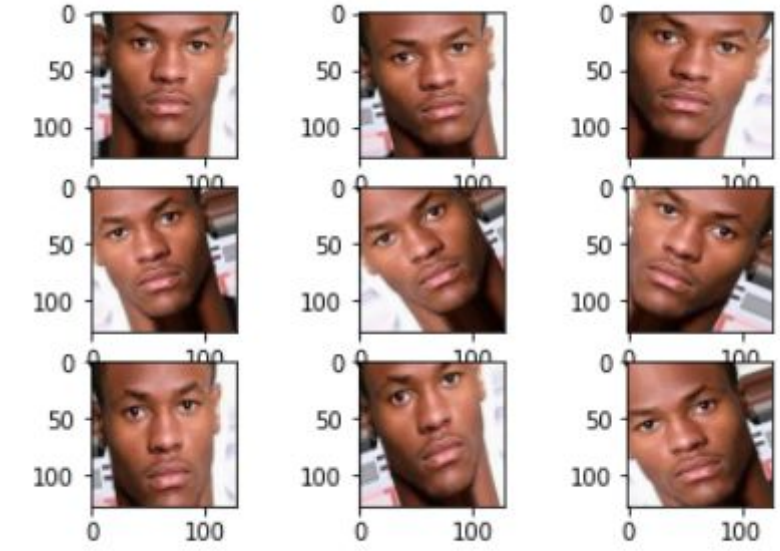
To sum up after all of these optimizations we came up with a following model.

```
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', input_shape=(128,128,3))(images)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size= (2, 2))(x)

x = layers.Conv2D(filters=64, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size= (2, 2))(x)

x = layers.Conv2D(filters=32, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size= (2, 2))(x)

x = layers.Conv2D(filters=32, kernel_size=3, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size= (2, 2))(x)

x = layers.Flatten()(x)
x = layers.Dropout(0.6)(x)
x = layers.Dense(512)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)

age_prediction = layers.Dense(1, name = 'age')(x) #

gender_prediction = layers.Dense(1, activation='sigmoid', name = 'gender')(x) #

model = Model(images,
              [age_prediction, gender_prediction])
if show: model.summary()

model.compile(optimizer='adam',
              loss={'age': 'mse', 'gender': 'binary_crossentropy'},
              loss_weights={'age': 2., 'gender': 1.},
              metrics={'age': 'mae', 'gender': 'accuracy'})
```

But even with all of these optimization techniques overfitting takes place in that model (already at 30 epochs). So the next step is to make some data augmentation.

Data augmentation works in this way:





With the help of data augmentation and early stopping the best result that has been obtained with this model for 90 epochs is the following:

Model	With Data Augmentation	Without Data Augmentation
Test Loss	0.203753	0.222669
Test Age Loss	0.008003	0.008399
Test Gender Loss	0.187746	0.205870
Test Age MAE	0.067269	0.068024
Test Gender Accuracy	0.922705	0.916264

So data augmentation techniques gave us a really good performance in comparing with ordinary data method. It's the best result I've obtained with multi output model. Of course there are still ways to improve this model for example: trying to adjust not only one parameter but a group of parameters at the same time; add a dropout layer after each convolution block, reduce the batch size (from 64 to 32 for example), controller learning rate etc. Also fine-tuning approach can be tried, for example: download ResNet50, add some dense layer of the top of it and train it (first 4-5 epochs ResNet should be frozen and for the rest of training epochs it should be unfrozen)

Let's try separate neural networks.

Separate age network

This network has used the same structure with the same parameters. The only thing that has been changed is the final layer. Now we have only one dense layer (age layer) instead of two layers (age and gender).

Let's train it with augmented data and compare the results with multi output network.

Model	Separate age	Multi output best
Test Age Loss	0.005592	0.008003
MAE	0.054874	0.067269

Separate gender network

The same idea as with the previous one.

Model	Separate gender	Multi output best
Test Gender Loss	0.194972	0.187746
Accuracy	0.918276	0.922705

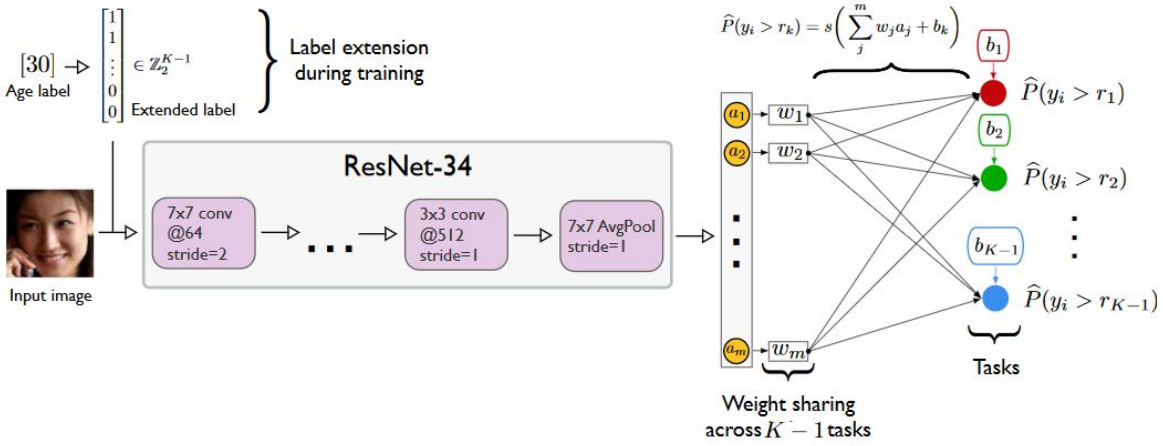
Logically separate networks should have performed better than a multi output one. It may be related with some ''randomness'' of training a neural network (weight initialisation for example) so it's important to train a network several times and estimate the error of final results.

State of the art

Consistent rank logits (CORAL)

In this approach for age prediction model researches propose a framework for extending an ordinal regression problem into several binary classification problems.

Firstly they create a label vector for each age comparing age with ordered ranks. Then they train a single CNN with these binary classifiers in the output layer. Those binary tasks share the same weight parameter but have independent bias units. This approach guarantees that the predictions are consistent.



This approach performs really well on UTKFace in comparison with others Methods like CE-CNN and OR-CNN.

Method	CE-CNN	OR-CNN	CORAL-CNN
Test Loss MAE	0.046	0.0412	0.038

CORAL: https://arxiv.org/pdf/1901.07884.pdf

Google Collab Notebook:
https://colab.research.google.com/drive/1px--Jh0io8jHtLNWPm4oSwT7god8xGNk