Internship Report - Second Project

Face Attributes Detection using Multi-task Deep Learning

Ismail OUSSAID

with the supervision of: Enrico Marchetto

Summary:

Facing dataset

The idea of the project is to find Multi-task Deep Learning solutions for Face Attributes Detection provided the CelebA dataset. Indeed, we want to classify 6 attributes: gender, hairstyle, wearing hat, beard, mustache & wearing eyeglasses in images.
First thing is to have a global analysis of the provided data. For the face extraction, I have used the csv file containing attributes labels. I have decided to test two methods:

1. Face extraction - face focus (type A)

For each picture, I take the coordinates of the nose, which becomes the center of the image and I crop the picture (Fig.1) using the parameter $\varepsilon = 0.5$:

$$
\begin{aligned}
y_{min} &= y_{nose} * (small - 0.2) \\
y_{max} &= y_{nose} * (grand - 0.2) \\
x_{min} &= x_{nose} * small \\
x_{max} &= x_{nose} * grand
\end{aligned}
$$

$$with: \ grand = \varepsilon + 0.2 \ \& \ small = \varepsilon - 0.2$$

2. Face extraction - larger vision (type B)

I have used the Quividi face detector to find coordinates of the faces in the dataset.
For each sample, I take the coordinates of the bounding box (corner's coordinates, width & height) and I extract the center's coordinates. I crop the picture (Fig.2) using the parameters $\alpha = 0.7, \ \beta = 0.9$ & $shift = h//16$:

$$
\begin{aligned}
y_{min} &= y_{center} - \beta * h \\
y_{max} &= y_{center} + \beta * h + shift \\
x_{min} &= x_{center} - \beta * w \\
x_{max} &= x_{center} + \beta * w
\end{aligned}
$$

Fig.1 A-cropped image (left) & B-cropped image (right)

After the face extraction & images reshaping to (36, 36, 1) which corresponds to surveillance camera images, I checked the distribution (Fig.3) of classes for each attribute of interest.

For 5 attributes, the distribution is very imbalanced as more than 83% of the dataset is composed of one class and *gender* is more well-balanced as 58% of the data is female samples. Then, using F1-Score to evaluate our future models performances for the 5 attributes detection and accuracy (possibly, F1-Score) for gender classification, seems more appropriate.

$$F1 = \frac{2TP}{2TP + FP + FN} \qquad Acc = \frac{TP + TN}{TP + FN + FP + TN}$$

Fig.2 F1-Score & Accuracy formulas

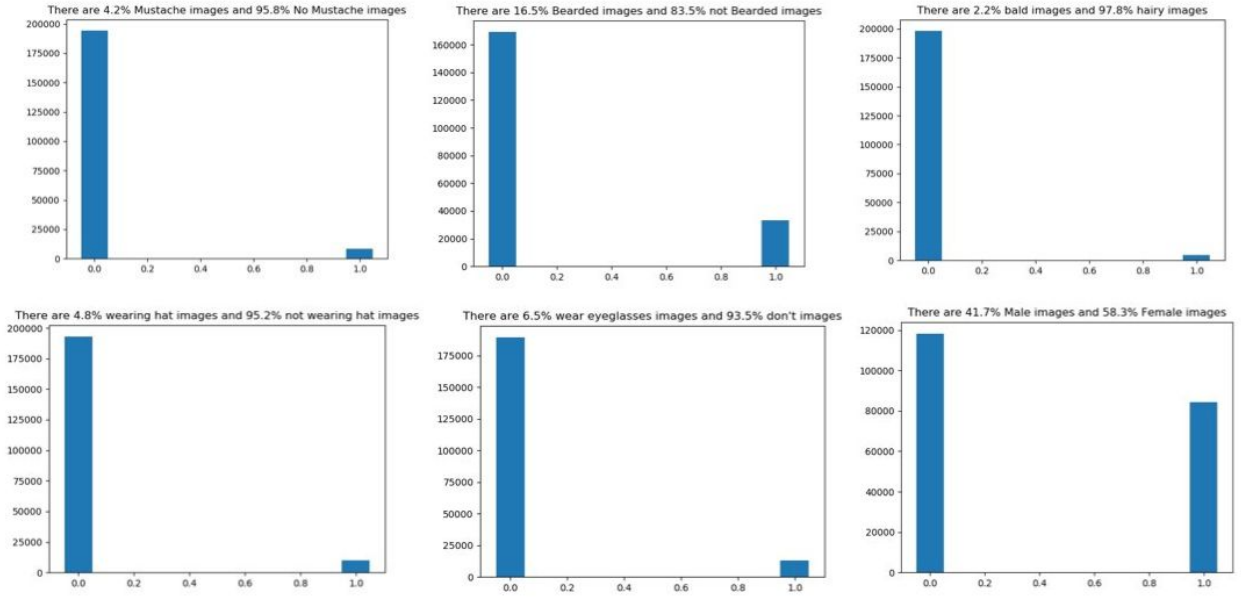

Fig.3 Histograms for *gender, beard, bald, hat, mustache, eyeglasses* distribution

Just to have reference scores, I choose to check the scores of the minority estimator as I have chosen to take minority class as the positive label. The exception is *gender* attribute, which is well balanced (58% female, 42% male) so then, we consider accuracy score for majority estimator as a reference.

```
For attribute: Male, the minority estimator has an accuracy of: 0.5832457218446291
For attribute: Mustache, the minority estimator has a F1-Score of: 0.07977594116085986
For attribute: Bald, the minority estimator has a F1-Score of: 0.04390140287526672
For attribute: Eyeglasses, the minority estimator has a F1-Score of: 0.12227515385185733
For attribute: No_Beard, the minority estimator has a F1-Score of: 0.2833502796136248
For attribute: Wearing_Hat, the minority estimator has a F1-Score of: 0.09244081217604994
```

To train the models, we will augment our dataset by adding horizontally flipped images (mirror-effect), blurred images (with average filter of size (2,2)) and horizontally flipped blurred images so we have four times more images of shape (36, 36, 1).

FaceNet: Implementation & Results

In this section, we will focus on finding functional multi task learning solutions with the help of a customized CelebA dataset and we will focus on having an interpretable model with a very few computational cost. To get inspiration for neural architectures to consider for our tasks, I check recent research papers about classification of one or many of *gender, beard, bald, hat, mustache, eyeglasses* classification. So, the very next thing is to find a functional neural architecture.

Very few, if no one, has worked on beard, bald, hat, mustache or eyeglasses detection in research. However, gender (like age) has already been worked on by researchers. So, to decide on the architecture, I have got inspiration from many papers and finally I have decided to have 2 Conv2D/MaxPool2D/BatchNorm (I have decided to add BatchNorm to avoid overfitting).

The objective is to have a multi-task learning network. So, I decided to consider the common network as Convolutional + Flatten and in order to have the smallest flop, I made just one intermediate layer and one final 2-node layer (Fig.4) per attribute.

Also, the loss function is a very important asset. We are working on classification so I take categorical cross entropy for each attribute. Then, I have to make a linear sum of these losses to make the MTL loss. After many attempts and the scrutiny of validation losses' effect on attributes scores during trainings, I have chosen:

$$Loss = 10.L_{gender} + 5L_{eyeglasses} + 5.L_{beard} + 5L_{bald} + L_{mustache} + L_{hat}$$

The architecture is organized like this :

```
inputs = Input(shape=inputShape)
x = Conv2D(self.first, size, padding="same")(inputs)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Conv2D(self.second, size, padding="same")(x)
x = BatchNormalization()(x)
x = Activation("relu")(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
```

I have taken *relu* as an activation function in every common & intermediate layer and the padding is always *same* so that the output in convolution doesn't differ in size as input. In simple terms, we add a n-pixel borders of zero to tell that don't reduce the dimension and have the same as input so we keep as much info as we can.

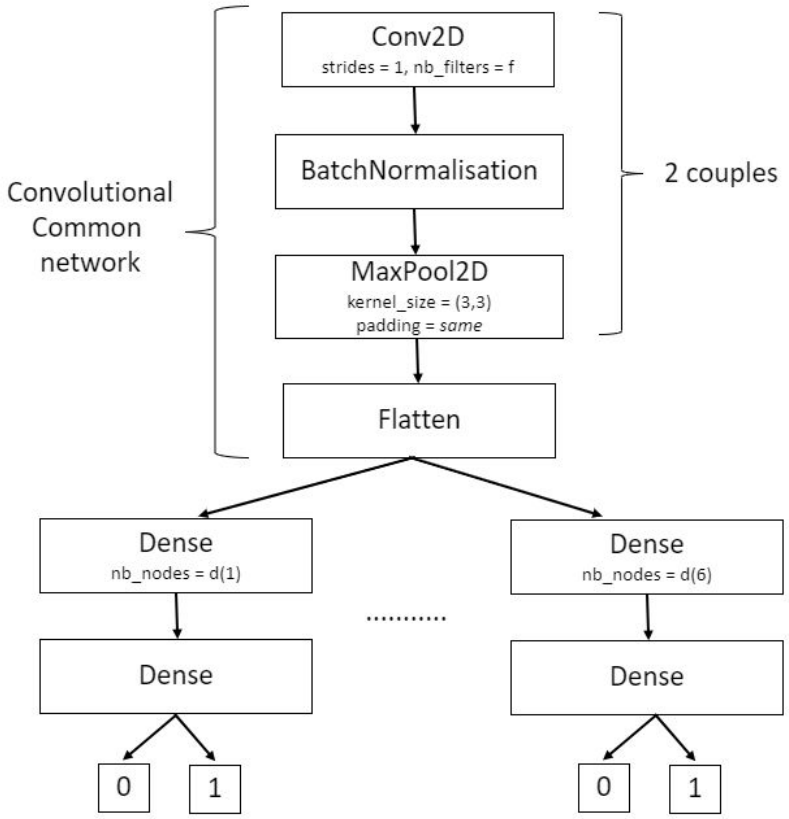

Fig.4 Neural architecture of attribute k classifier with hyperparameters (f1, f2, d(k))

To simplify the network, I have taken d(k)=d constant for every k. Then, for every attribute, there is the same number of nodes for each dense network. With such a choice, I go from 6 to 3 hyperparameters: first_conv (f1), second_conv (f2) and unit (d).

During training, it appears the hardest task (as expected) is *gender* classification. Surprisingly, *beard* & *bald* happen to be hard but after reviewing the dataset type A, the first is expectable as the cropping hides the beards but for the second, it just seems to be a hard task and I don't find any clear explanation, so far.

Fig.5 Beard labeled image cropped

To find an interesting architecture, I have chosen the following process:

- Split the dataset into 90 % training set and 10 % testing set (for each dataset)
- Train the architecture on Tensorflow & Keras with epochs = 10 for different values of the hyperparameters: f1, f2, d, batch_size (Grid Search with one fold)
- Evaluate each model (architecture with fixed hyperparameters) on testing set
- Compute test scores for each model

kernel size	first conv	second conv	unit	batch_size	gender	mustach	eyeglasses	beard	hat	bald	flop
(3, 3)	4	8	8	64	0,9210789204	0,956494391	0,9667814374	0,8949261904	0,9639551044	0,9775560498	11338
(3, 3)	4	8	16	64	0,927490592	0,9562598467	0,9718231559	0,9010108113	0,9686636329	0,9775621891	33276
(3, 3)	4	16	8	64	0,9380183816	0,9570189118	0,975112319	0,9081383348	0,9637082815	0,9787964225	22314
(3, 3)	4	16	16	64	0,9449669123	0,9561672807	0,9767044187	0,9030656815	0,9724587798	0,9806908965	65596

Fig.6 Extract of excel tab output of this process (for type A)

This process is not the most precise as it is not a cross validation, the kernel_size is unique (=(3,3)) and there are very few epochs (10 epochs) but this is due to a lack of time as the CPU takes too much time to run such amount of CNNs on a +800k A-type dataset (after augmentation) and 760k B-type dataset (after augmentation).
After reading the results, the best model (Fig.7) is: (f1, f2, d, batch_size)=(4, 16, 16, 64).

For A-type dataset, the performance is:

- Gender accuracy: 94.5%
- Mustache f1: 95.6%
- Eyeglasses f1: 97.6%
- Beard f1: 90.3%
- Hat f1: 97.2%
- Bald f1: 98.1%

There has not been a hyperparameters tuning with B-type (lack of time and type A is more significant) so the best type-A performer will be tested on type-B.
Considering the references I have given in page 3, this is quite astounding so we need to cross validate the model with a 5-Fold to ensure it is consistent (small standard deviation) with 25 epochs on the augmented data.

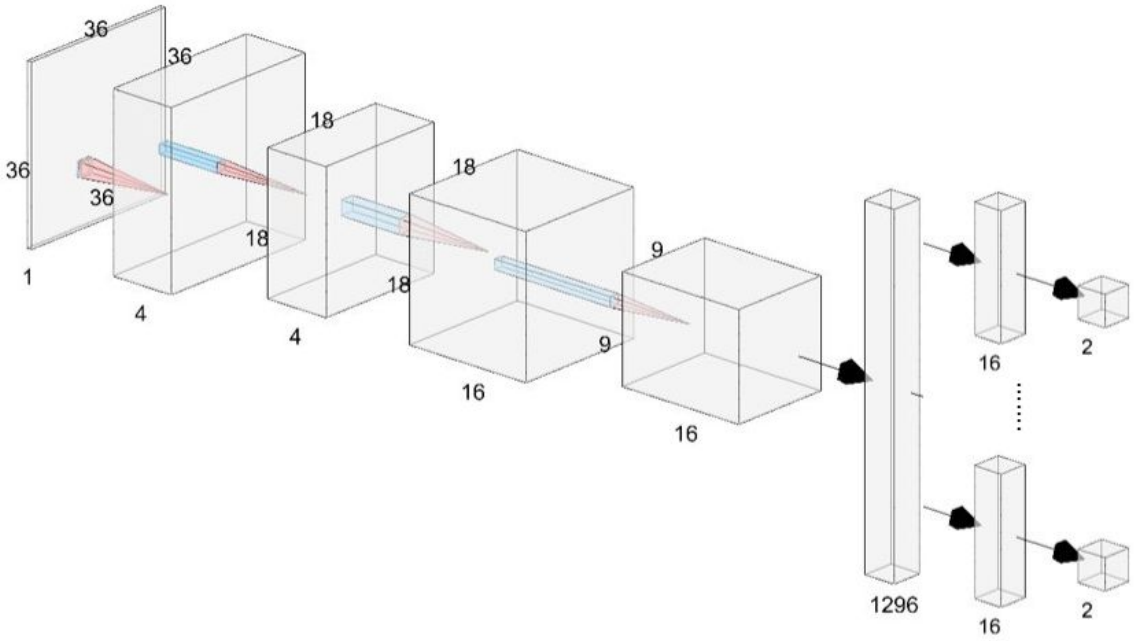

Fig.7 MTL FaceNet architecture

After testing phase, the performance is:

	type-A	type-B
• Gender accuracy:	94.06% (1.17%)	95.15% (0.27%)
• Mustache f1:	95.69% (0.08%)	95.72% (0.07%)
• Eyeglasses f1:	97.9% (0.11%)	97.62% (0.17%)
• Beard f1:	90.8% (0.81%)	91.32% (0.17%)
• Hat f1:	97.11% (0.07%)	97.50% (0.08%)
• Bald f1:	98.06% (0.05%)	97.96% (0.12%)

- Floating points operations: 43282

The model is very good in classifying and very consistent as it has a very low variance of performance. Also, flop are not high so FaceNet is clearly doing the job well.

To understand how FaceNet works in detail, I will take a grayscale image randomly in our dataset (Fig.8) and show how it is processed since it gets into the first layer.
First, the network processes the 36x36x1 image into 4 feature maps size (36,36) which highlights some details. Eyes, top head & face variations are highlighted, which is expectable as our attributes are detectable through such details. After this, the image goes through Max Pooling that stresses out sharp & smooth features by taking maximum pixel value in each square of 4 pixels and one gets a (18,18) size image.
Next thing is the exact same process but with 16 feature maps which highlights a lot more features as one can see. But I have to admit that features are not very visible, which explains the difficulty for the network to train well as the images are so small.

Finally, the feature maps are flattened to a vector of 1296 coordinates, that goes through dense fully-connected multi-task layers that provides 6 probability arrays outputs for our 6 attributes.

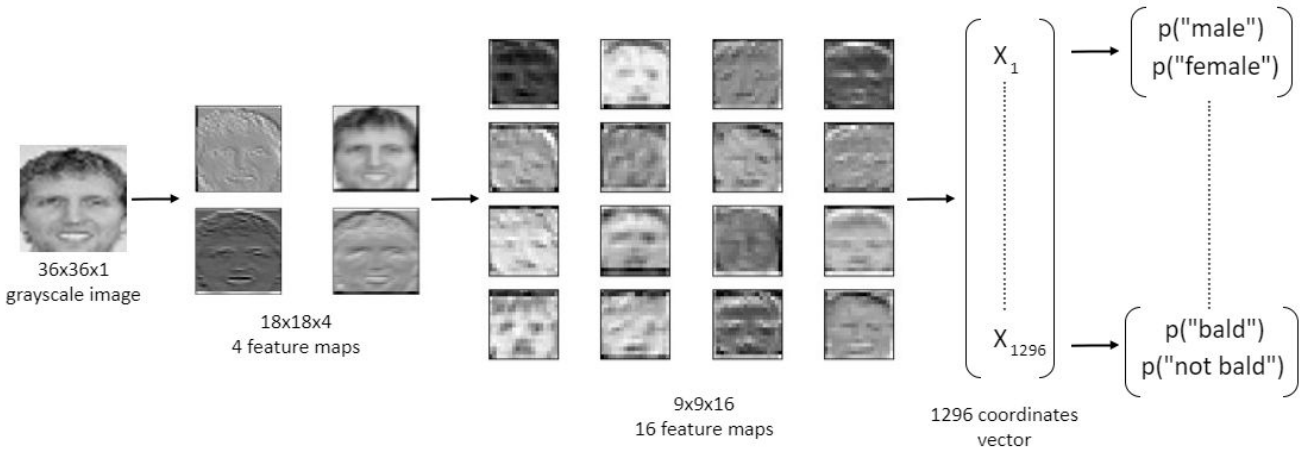

Fig.8 Image processing by FaceNet

Study of FaceNet bis:

In this section, we will focus on finding functional multi task learning solutions with the help of a customized CelebA dataset (type B) to detect *beard, bald, hat, mustache, eyeglasses* classification. To get inspiration for neural architectures to consider for our tasks, I use the precedent architecture for the FaceNet bis.

Also, the loss function is a very important asset. We are working on classification so I take categorical cross entropy for each attribute. Then, I have to make a linear sum of these losses to make the MTL loss. I have chosen:

$$Loss = 5L_{eyeglasses} + 5.L_{beard} + 5L_{bald} + L_{mustache} + L_{hat}$$

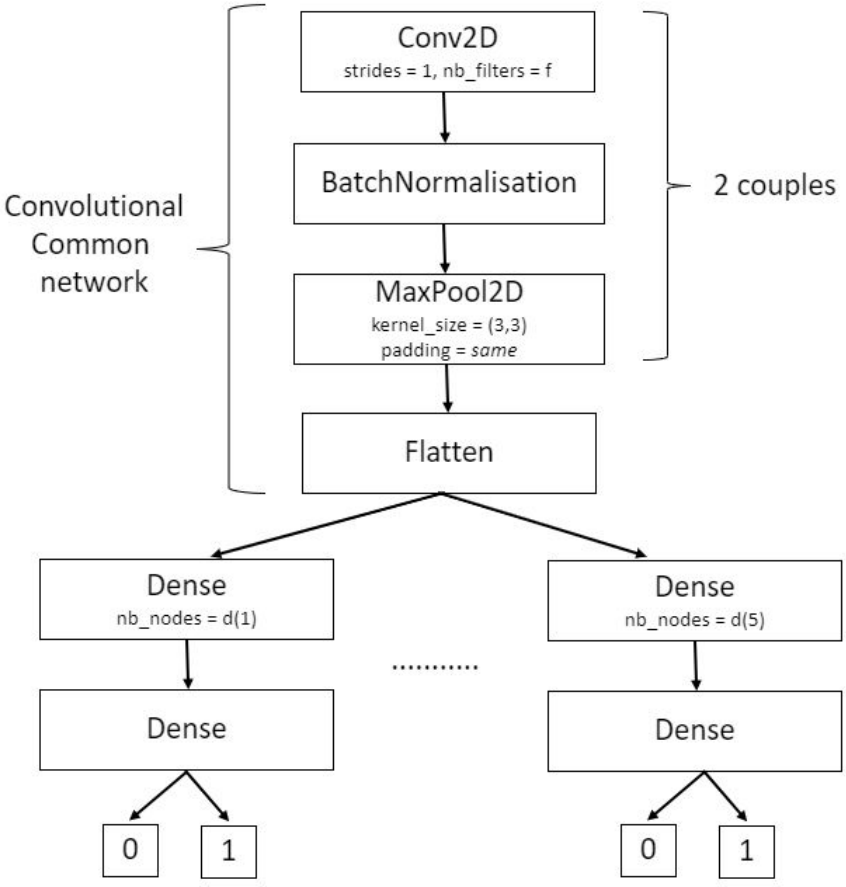

Fig.9 Neural architecture of attribute k classifier with hyperparameters (f1, f2, d(k))

To simplify the network, I have taken d(k)=d constant for every k. Then, for each attribute, there is the same number of nodes. With such a choice, I have only 3 hyperparameters: first_conv (f1), second_conv (f2) and unit (d).

To find an interesting architecture, I have chosen the following process:

- Split the dataset into 90 % training set and 10 % testing set (for each dataset)
- Train the architecture on Tensorflow & Keras with epochs = 10 for different values of the hyperparameters: f1, f2, d, batch_size (Grid Search with one fold)
- Evaluate each model (architecture with fixed hyperparameters) on testing set
- Compute test scores for each model

kernel size	first conv	second conv	unit	batch_size	gender	mustach	eyeglasses	beard	hat	bald	flop
(3, 3)	4	8	8	64	0,9210789204	0,956494391	0,9667814374	0,8949261904	0,9639551044	0,9775560498	11338
(3, 3)	4	8	16	64	0,927490592	0,9562598467	0,9718231559	0,9010108113	0,9686636329	0,9775621891	33276
(3, 3)	4	16	8	64	0,9380183816	0,9570189118	0,975112319	0,9081383348	0,9637082815	0,9787964225	22314
(3, 3)	4	16	16	64	0,9449669123	0,9561672807	0,9767044187	0,9030656815	0,9724587798	0,9806908965	65596

Fig.10 Extract of excel tab output of this process (for type A)

This process is not the most precise as it is not a cross validation, the kernel_size is unique (=(3,3)) and there are very few epochs (10 epochs) but this is due to a lack of time as the CPU takes too much time to run such amount of CNNs on a 760k B-type dataset (after augmentation).
Regarding the results, the best model is: (f1, f2, d, batch_size)=(4, 16, 16, 64).
For A-type dataset, the performance is:

- Mustache f1: 95.8%
- Eyeglasses f1: 97.8%
- Beard f1: 91.1%
- Hat f1: 97.3%
- Bald f1: 98.1%

There has not been a hyperparameters tuning with A-type (useless as we consider the A-type to be a non-realistic dataset) so the best classifier is the best type-A performer. Considering the references I have given in page 3, this is quite astounding so we need to cross validate the model with a 5-Fold to ensure it is consistent (small standard deviation) with 25 epochs on the augmented data.

After testing phase, the performance is:

- Mustache f1: 95.81% (0.07%)
- Eyeglasses f1: 98.05% (0.22%)
- Beard f1: 92.07% (0.25%)
- Hat f1: 97.73% (0.13%)
- Bald f1: 98.16% (0.10%)
- Floating points operations: 43212

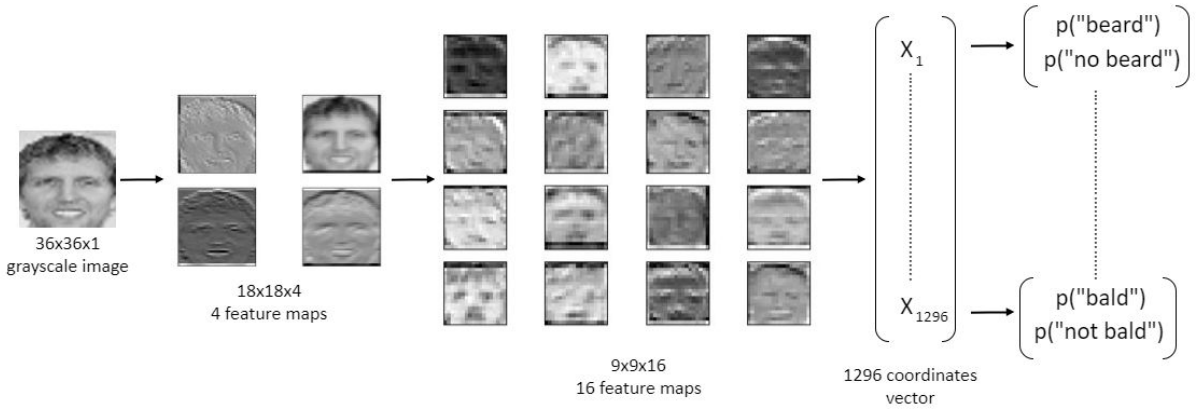

Fig.8 Image processing by FaceNet bis

Finally, the feature maps are flattened to a vector of 1296 coordinates, that goes through dense fully-connected multi-task layers that provides 5 probability arrays outputs for our 5 attributes.

Concerning the common attributes detected, FaceNet bis is better than FaceNet. Concerning the way the FaceNet bis works, it is similar to FaceNet.

Appendix

Model: "FaceNet"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 36, 36, 1)]	0	
conv2d (Conv2D)	(None, 36, 36, 4)	40	input_1[0][0]
batch_normalization	(None, 36, 36, 4)	16	conv2d[0][0]
activation (Activation)	(None, 36, 36, 4)	0	batch_normalization[0][0]
max_pooling2d	(None, 18, 18, 4)	0	activation[0][0]
conv2d_1 (Conv2D)	(None, 18, 18, 16)	592	max_pooling2d[0][0]
batch_normalization_1	(None, 18, 18, 16)	64	conv2d_1[0][0]
activation_1 (Activation)	(None, 18, 18, 16)	0	batch_normalization_1[0][0]
max_pooling2d_1	(None, 9, 9, 16)	0	activation_1[0][0]
flatten (Flatten)	(None, 1296)	0	max_pooling2d_1[0][0]
dense (Dense)	(None, 16)	20752	flatten[0][0]
batch_normalization_2	(None, 16)	64	dense[0][0]
activation_2 (Activation)	(None, 16)	0	batch_normalization_2[0][0]
dense_1 (Dense)	(None, 2)	34	activation_2[0][0]
dense_2 (Dense)	(None, 2)	34	activation_2[0][0]
dense_3 (Dense)	(None, 2)	34	activation_2[0][0]
dense_4 (Dense)	(None, 2)	34	activation_2[0][0]
dense_5 (Dense)	(None, 2)	34	activation_2[0][0]
dense_6 (Dense)	(None, 2)	34	activation_2[0][0]
gender (Activation)	(None, 2)	0	dense_1[0][0]
mustache (Activation)	(None, 2)	0	dense_2[0][0]
eyeglasses (Activation)	(None, 2)	0	dense_3[0][0]
beard (Activation)	(None, 2)	0	dense_4[0][0]
hat (Activation)	(None, 2)	0	dense_5[0][0]
bald (Activation)	(None, 2)	0	dense_6[0][0]

Total params: 21,732
Trainable params: 21,660
Non-trainable params: 72