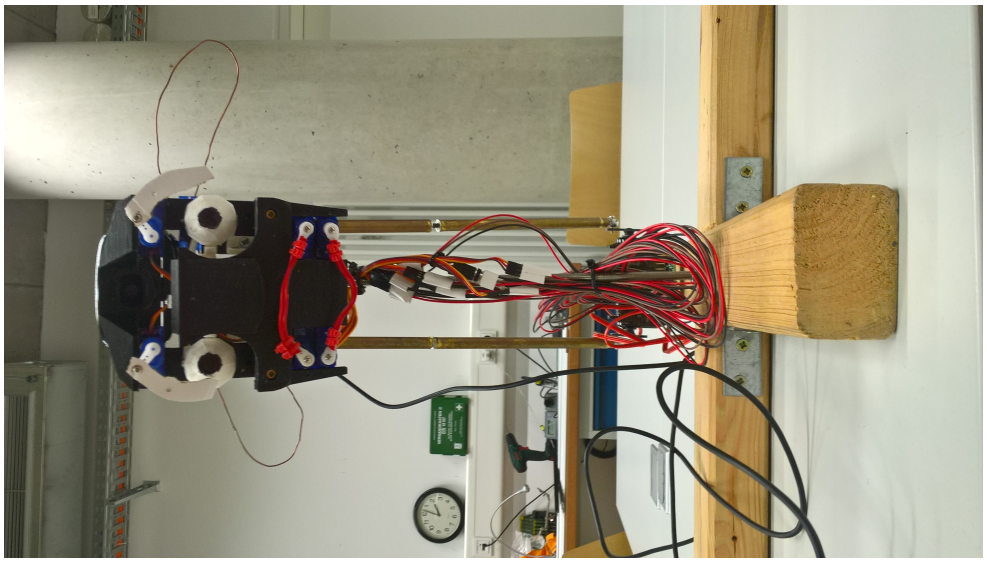
RoboFace



Kevin Kiefer
5th semester informatics

Letitia Parcalabescu
3rd semester informatics

Supervisors: Gero Plettenberg and Benjamin Reh

1 Task:

The existing robotic face reacts on visual stimuli by producing acoustic and mimic signals. In order to achieve this, we implemented face and attribute recognition recognition. Furthermore, a voice output should indicate a reaction on the analysed face that RoboFace contemplates. This face is also tracked by the robot in order to maintain eye contact.

2 Milestones:

- overwriting the existing codebase
- face recognition
- face attribute recognition
- voice/sound output
- face tracking

3 Setup guide for the robot

3.1 Hardware

3.2 Software

The elaborated C++ code for controlling RoboFace has beautiful user frienly python binaries. The following code example speaks for himself: »> import face
»> f = face.Face()
»> f.sad()
»> f.happy()
»> f.unsure()
»> f.angry()
»> f.neutral()

For using the face attribute detection software, a standard python (2.7 or 3.5) anaconda installation should additionally include *opencv, keras*. You can easily install them using conda with the following commands:

- conda install -c menpo opencv3=3.1.0
- conda install keras

Then, for running the software, execute the python script `face_detection/face_detection.py`

If you eventually want to retrain the network for further improvement of the results, then you should take a look in te `neural_network_training/train_normalised.py` script.

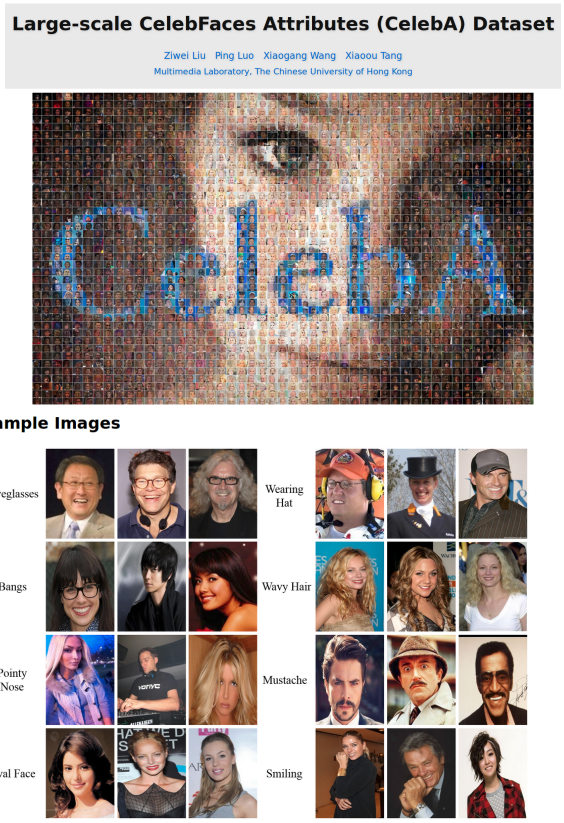
3.3 Face attribute detection - the magic behind neural networks

3.3.1 Architecture

1. 32 of neurons in each convolutional layer
2. Activations: Relu
3. 9 x 9 convolution → Max Pooling
4. 7 x 7 convolution → Max Pooling
5. 5 x 5 convolution → Max Pooling
6. 3 x 3 convolution → Max Pooling
7. 3 x 3 convolution → Max Pooling
8. Dropout(0.25)
9. 512 Dense
10. Dropout(0.5)
11. 13 Dense
12. Sigmoid
13. Binary Crossentropy
14. Adadelta optimiser
15. Overall: 125,709 parameters

3.3.2 Training dataset

For training a neural network that generalises as much as possible, we take the big CelebA dataset, which is freely available at http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html . It contains 202.599 images of 10.177 celebrities annotated with 40 attributes per image.



From this attributes, we take 13 that are based only of the face region (so not taking the neck, shoulders or head top into account) and those where we think RoboFace could make something of. These are:

1. Black Hair
2. Blond Hair
3. Brown Hair
4. Eyeglasses
5. Gray Hair
6. Male
7. Mouth Slightly Open
8. No Beard
9. Smiling
10. Straight Hair
11. Wavy Hair

12. Wearing Earrings

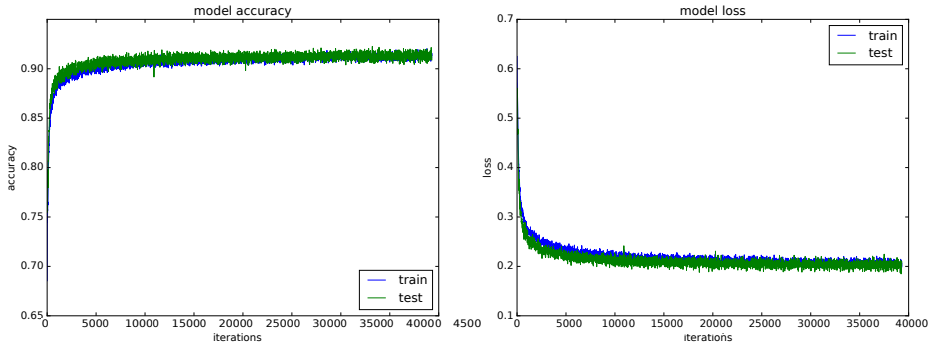
13. Wearing Lipstick

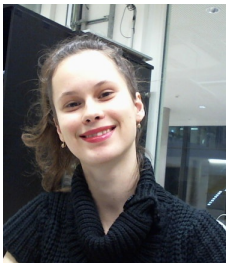
3.3.3 Training the neural network

For training the neural network, we make use of the parallelizing capacity of the GPU, which results into a training time of 2 seconds per 1024 images. CPU training is obviously much slower. The optimiser for finding a good local optimum of the loss function is the Adadelta optimiser from keras, which stands out for his adaptive learning rate and faster convergence than normal stochastic gradient descent SGD. The overall 125.709 parameters are found in this way.

3.3.4 Normalising the data

A wrong way to train the neural network is to feed in the CelebA images from the dataset exactly as they are. Even though the training curve looks very fine, since we achieve a low loss value and a high accuracy on the training set and on the test set, the neural network over-fits the CelebA dataset and does not generalise at all on real world images as taken from RoboFace. He completely fails by saying that in the following image, there is a male with no beard with probabilities around 90%

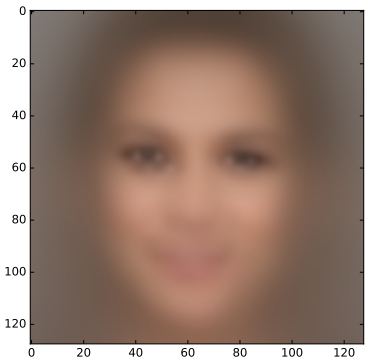




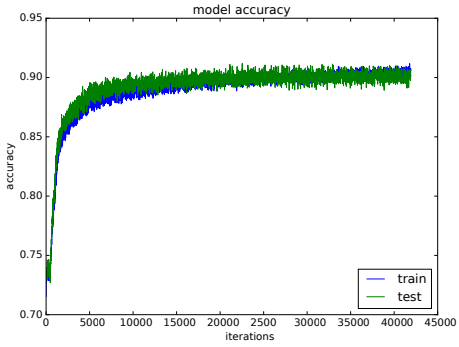
The write way too do it, is by normalising the training and the test set, as well as the images taken by RoboFace. This is done in `neural_network_training/normalisation.py` the following steps:

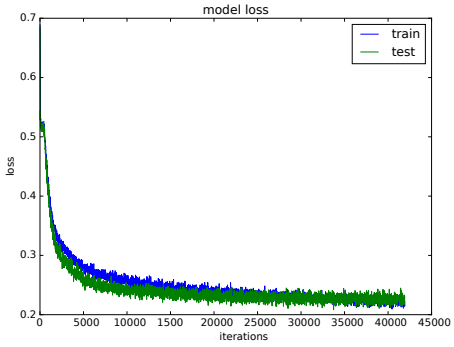
1. center the eyes

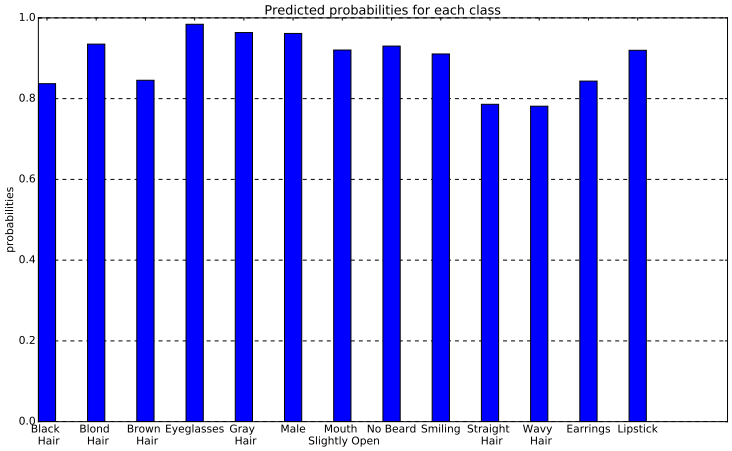
2. resize images to have the same inter ocular distance
3. rotate image to make the inter ocular line look horizontal
4. resize image to 128 x 128 pixels
5. subtract the mean face



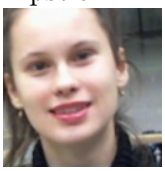
Now, the loss-accuracy curve looks as well as before, but the generalisation capabilities are vastly enhanced.



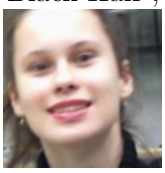




'Black Hair', 'Mouth Slightly Open', 'No Beard', 'Smiling', 'Wearing Earrings', 'Wearing Lipstick'



'Black Hair', 'Mouth Slightly Open', 'No Beard', 'Straight Hair', 'Wearing Lipstick'



On another day, when having another look, he also performs fine:

'Black Hair', 'Mouth Slightly Open', 'No Beard', 'Smiling', 'Straight Hair', 'Wearing Earrings', *'Wearing Lipstick*

Indeed, the following errors are done in 2 out of 45 examples. But this small amount is no problem, since we have a continuous frame and prediction stream from the robot. If for example, he predicts the stream 'male, male, female, male, male', it is clear that we can exclude this error in a post-processing step.

'Black Hair', *'Male'*, 'Straight Hair'

4 Outlook

Hardware improvements

5 Source code

Can be found on https://github.com/LetiP/RoboFace