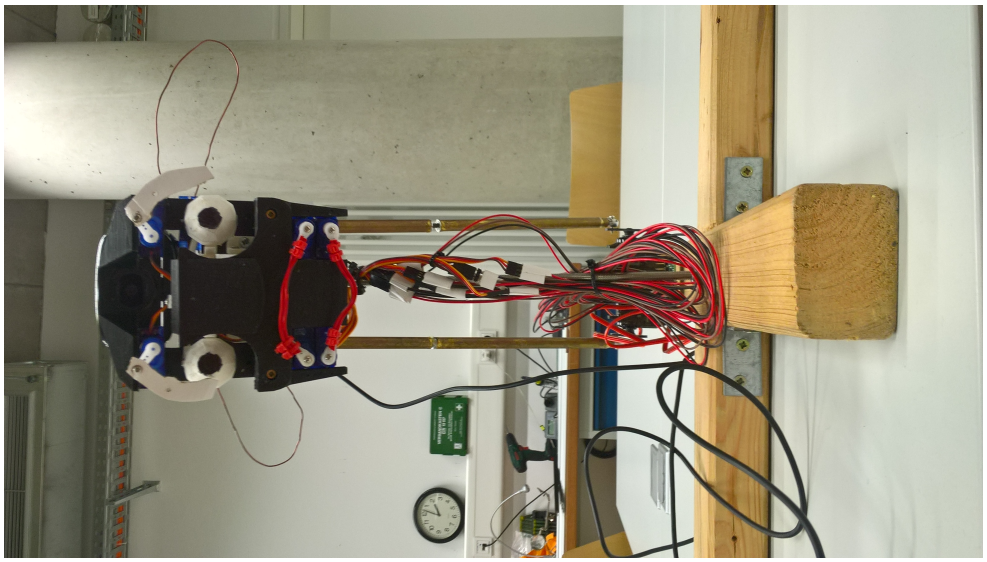
RoboFace



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.

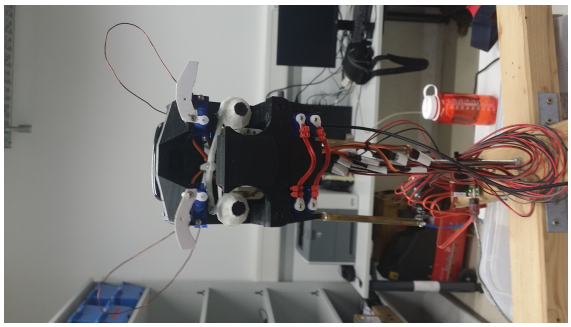
2 Milestones:

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

3 Setup guide for the robot

3.1 Hardware

There are 2 servos used to move the head and 10 more to display facial expressions. The mini maestro 18-channel usb servo controller is used to control them.



3.2 Software

A python interface is build on top of the C++ code controlling the robots hardware. Which may be used like this: »> import face
»> f = face.Face()
»> f.happy() # look happy (possibly moves the head)
»> f.angry(moveHead=False) # look angry but don't move the head
»> f.moveHead(x, y) # move the head and possibly the eyes to the position (x, y),
»> # where (x, y) are coordinates within the current webcam frame
Further facial expressions and head movement functions as well as optional configuration parameters for the constructor are provided. Please call help(Face) from within your python interpreter for more information.

In case you need a more specific control over each servo you must use the C++ api, which is a superset of the python api. The major addition here, which is visible to the user, is the notion of a "ServoConfig" and the corresponding applyConfig() function. For example:

```
#include <face.h>

int main()
{
    Face f = Face();
    ServoConfig<3> config = {{1, 5, 10}, {4500, 8000, 6000}};
    // set servo 1, 5, and 10 to the positions 4500, 8000 and 6000
    f.applyConfig(config);
}
```

The code depends on the pololu-usb-sdk and RapaPololuMaestro library. Which are available on github.

- https://github.com/pololu/pololu-usb-sdk
- https://github.com/jbitoniau/RapaPololuMaestr://github.com/jbitoniau/RapaPololuMaestro

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

The task to give RoboFace his understanding of the world through the webcam he uses to perceive, belongs to the problems that computer vision tries to solve. The algorithm of our choice is a convolutional neural network (CNN), since neural networks are the state of the art today in computer vision.

Our implementation is a supervised learning algorithm, since we feed in labels (groundtruth) for every training image. We define a loss function that minimises the distance between the prediction and the true label. In this way, we learn the actual representation of the training data and can apply it to our test images and on RoboFace.

A neural network is composed of linear functions called neurons, followed by a nonlinear transformation. We can not only have many neurons per layer, but also stack layers consecutively. This results in a much better and faster training, since the minimum of the non-convex function we want to minimise in training is a better one , so closer to the global optimum we can not find. But interesting is, that it was mathematically proven, that a single hidden layer (so a neural network containing input, output and one middle layer) of finite number of neurons, is enough for approximating any continous function. For more details, see the universal approximation theorem.

The architecture defines how this neurons are stacked in layers. A widely used layer in vision solving algorithms is implementing convolutions. This is inspired by the human vision system, that also bundles a region of photosensitive cells and combine their signal into one neuron that send his own output to be processed further.

3.3.1 Architecture

We take more of this convolutional layers and stack them together, to generate a so called deep neural network. The deepness shows its effectiveness in maximising the field of view of the network. Since a single convolution has for example the size 9 x 9, one layer can see only this much context. But stacking a next convolutional layer, the output of the previous 9 x 9 is considered again in 7 x 7 window for example, grows the field of view of the second layer considerably.

In the architecture we implemented, there is a Max Pooling Layer following every convolution step. This layer is actually downscaling the image to the half, in order to reduce the number of parameters and avoid redundancy of information.

The activations representing the nonlinearity is the ReLU (rectified linear unit) which penalises for wrong answers of each neuron during training.

The dropout layers in our architecture shut down randomly 25% and 50% of the neurons in order to make a better generalising neural network, since the neurons do not get the chance to learn the data by heart and over-fit.

The dense layer is connecting every neuron in the previous one with the ones in the current layer. Hence it contains more parameters than the convolutional ones, where the weights are vastly shared. We can interpret the role of the dense layers as the classification step in the network, whereas the convolutions did the feature extraction. So it is sensible to choose the number of neurons in the last layer as the amount of classes we want to

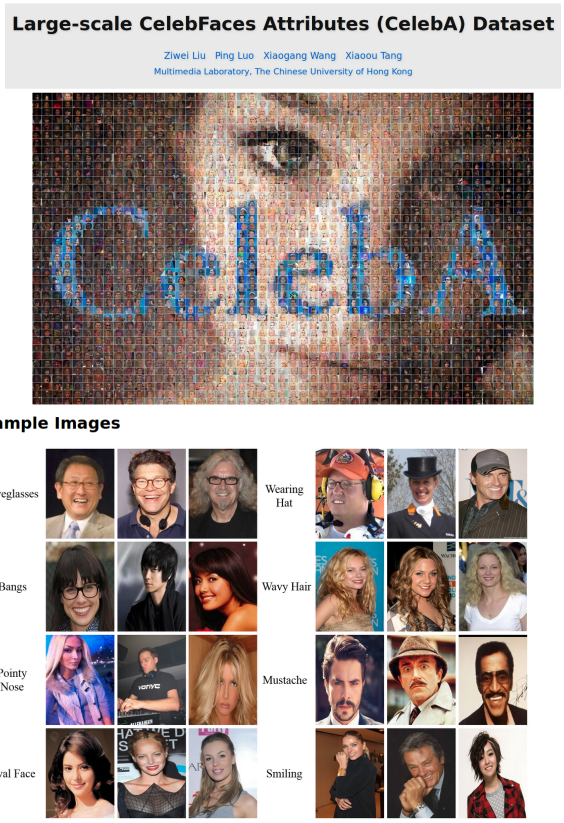
predict, which is 13 in our case.

The last activation function in the network penalises for wrong classification. We use the binary crossentropy and not the categorical one, since the classes are not mutually dependent. We can have a male with straight hair, this attributes do not exclude each other. There are some which do, like straight and wavy hair, but firstly, a good trained network should not predict both at the same time, since it learned to distinguish one from another. Secondly, this can be avoided in a postprocessing step, at prediction time.

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

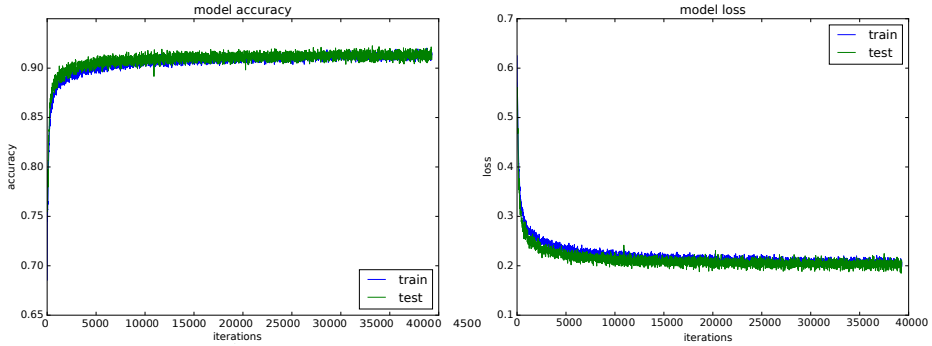
12. Wearing Earrings
13. Wearing Lipstick

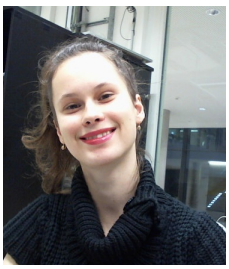
3.3.3 Training the neural network

The neural network implementation is done with the keras module using the TensorFlow backend. 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%

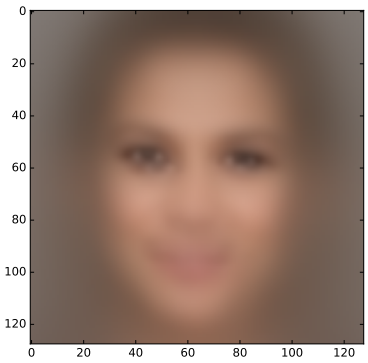




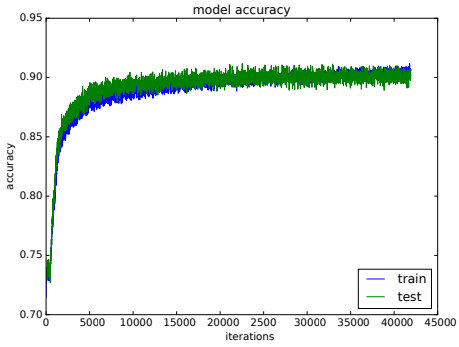
The write way to do it, is by normalising the training and the test set, as well as the images taken by RoboFace. This is done in `neural_network_training/normalisation.py` the following steps:

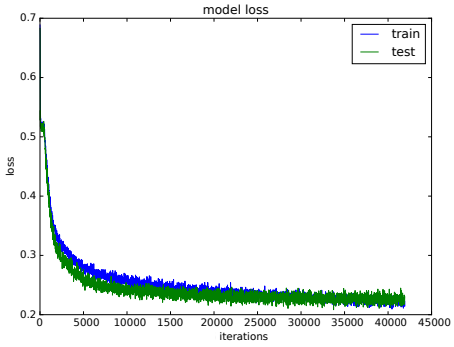
1. center the eyes

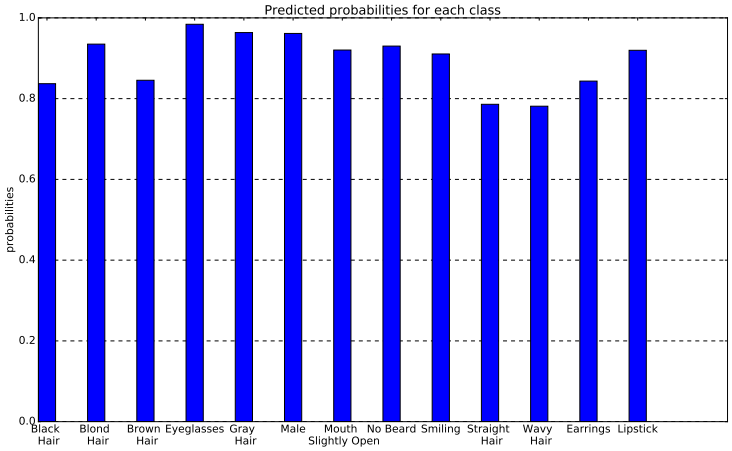
2. resize images to have the same inter ocular distance
3. rotate image to make the inter ocular line look horizontal
4. resize image to 128 x 128 pixels
5. subtract the mean face



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





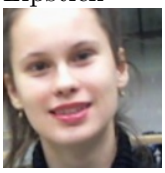


In the plot showing the accuracy per each class, we see as expected, that wavy and straight hair are comparably not distinguished that well. This comes form the fact, that the normalisation step crops out most of the region that does not contain the face, so a lot of hair information gets lost. The eyeglasses class, which is in the center of the face image, gets the best accuracy.

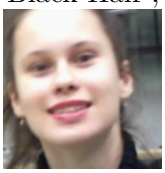
3.3.5 Results

In the following are some examples of the neural network predicting on images taken by RoboFace.

'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

Improving the socket on which the head is placed as well as the way it moves would provide the biggest benefit for the hardware. Software wise it is mostly an issue of getting enough labeled images to train the neural network. We already used hundreds of thousands of them tough. Improving much without going into the millions is unlikely.

5 Source code

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