# CASA Project – CNN for Face Recognition

The goal of the project was to build a Convolutional Neural Network which can distinguish between 62 people from the private database we were given (GOTCHA DB).

The original database consisted of 9 maps (types) of videos recorded in 9 different conditions. In the first 7 maps there was a video of each subject (62 in total), in the 8th and 9th maps there were videos of only 6 subjects. Duration of the videos varied around 4s and all of them were recorded at 60 frames per second. Using GOTCHA DB, new database, which consists of 62 folders, was created and used for the project. Name of each folder corresponded to the ID of one of the subjects, which made the labelling process easier. In every folder of the new database there are all the frames extracted from the original videos in which *one particular subject* appears (except the ones from maps 8 and 9), cropped in the way that only the face of the subject is visible.

False positive and false negative detections were minimised using multiple haarcascade files, optimizing the parameters and using histogram equalization. One haarcascade file was used for detecting the face. Once the face was detected, we used another haarcascade file in order to search for the mouth in the area of the detected face. If the mouth was found, we would suggest we have detected a human face. Histogram equalization was necessary when extracting the frames from videos of subject 33, because of his darker skin tone. We were still left with a small number of false positive detections (faces on the shirts, pavement, ripped jeans), which we manually deleted. Final result was database which consists of 62, 355 images of 62 subjects.

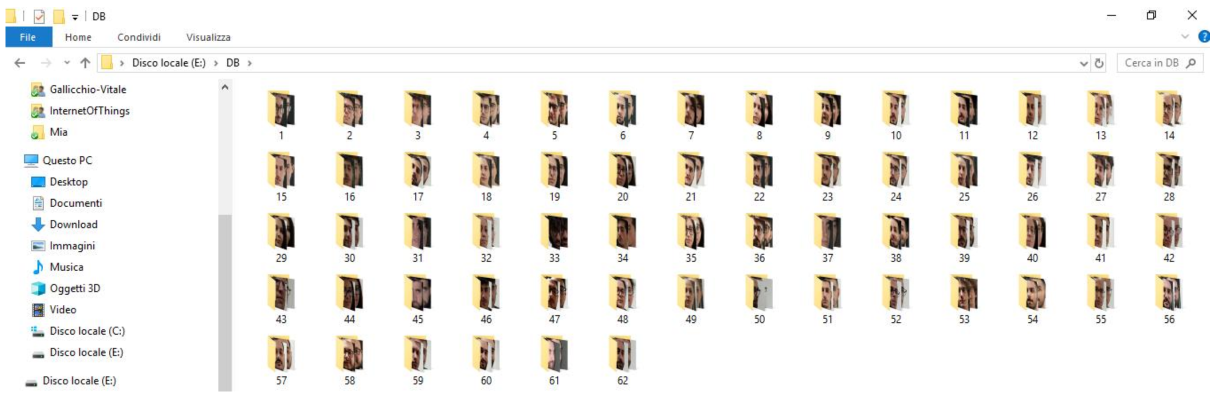

*Figure 1. Structure of DB used for training, validation and testing*

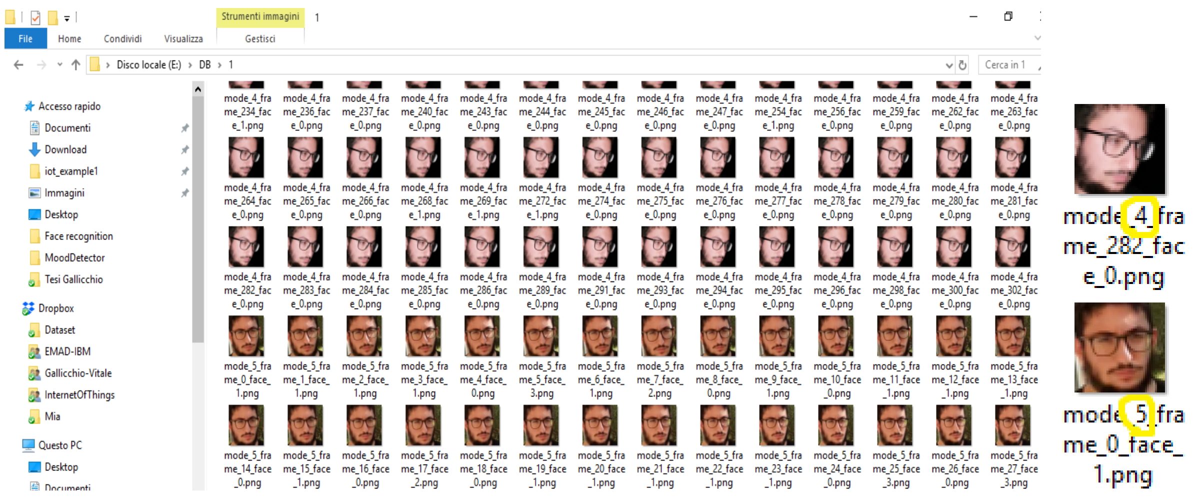

*Figure 2. Structure of the folder related to person one, the number x after mode_ represents the video from which the image was taken.*

Before using our images as inputs to the neural network, it was necessary to resize them, as we didn't have enough of the computational power to load and work with images of the original size. We resized them to 102x102 pixels, aware of the fact that our neural network might work even better with images of the original size (512x512). Unfortunately, we weren't able to test that assumption.

The data also had to be reshaped and modelled in order to be used as an input to the CNN. After normalizing and transforming it, we had image 3D tensors (width * height * depth), which were used in convolutional layers to obtain a discriminant image "feature vector". Image feature vectors and labels were used in deep layers for learning.

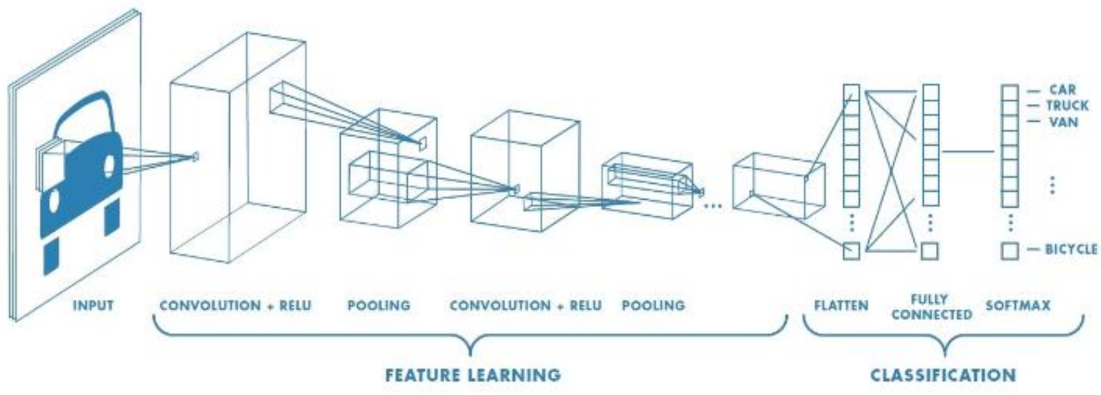

*Figure 3. Typical CNN structure*

The first idea was to use all of the 60 fps from all of the videos, but it resulted in overfitting the model, in fact, after just a couple of epochs, most of the images used for training and validation were pretty much the same. Although we randomly divided images in training e validation the redundancy of many similar images brought the model to train and validate on the same examples and so overfit. After experimenting with different rates, we decided to use the one of 10 fps, that happens to be, at the time of writing, the optimal rate.

In the trials which follow, we tested the behaviour of the model when using different videos for training and validating, while always preserving one (usually 1st or 7th one) for testing. The result was too dependent on the choice we made when splitting videos in a group for training or validation, so the final decision was to use a set of videos for training *and* validation and one video for testing. In final model, frames from videos 2-7 for all of the subjects (at the rate of 10 fps) were used for training and validate the CNN. The data was shuffled and split in training (5551 images) and validation set (2379 images). Frames from the 1st video (3634 of them) were used for testing the model.

Though our result was improving and the above mentioned option seemed the best possible one, we still had problems with overfitting. Finally, we applied the dropout regularization, and came up with the model with the following architecture:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 100, 100, 32)      896
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 50, 50, 32)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 48, 48, 68)        19652
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 24, 24, 68)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 24, 24, 68)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 22, 22, 68)        41684
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 11, 11, 68)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 11, 11, 68)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 8228)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               1053312
_________________________________________________________________
dropout_12 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 62)                7998
=================================================================
Total params: 1,123,542
Trainable params: 1,123,542
Non-trainable params: 0
```

Additionally, we tested different kinds of optimizers. Although ‘adam’ was often mentioned in the literature, we had the best results when using ‘rmsprop’, so we kept it in the final model. We limited the number of epochs to 100, which is not a big number, but both accuracy and loss seemed to have almost constant value after less than 40 epochs.

With the presented model, we achieved a 99% accuracy for training, 96% for validation and 86% for testing. Training and validation accuracy and loss can be seen on the graphs.

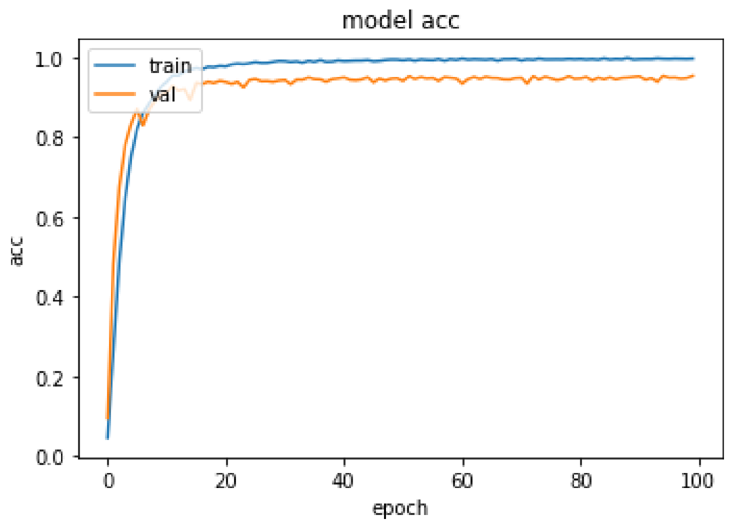

*Figure 4. Final model accuracy*

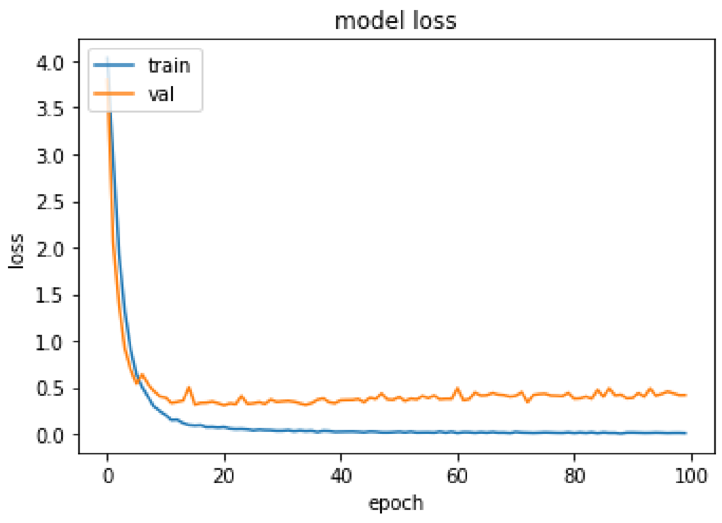

*Figure 5. Final model loss*

For experimenting purposes, we also tested the version were the model was trained and validated on frames from only the 7th, 3D video, and afterwards tested on remaining 6 videos, but we came to conclusion that 7th video didn't have enough information of the subjects to be able to generalize to all other conditions, or that our model was too simple for such experiment. In this case, we had the testing accuracy of 25%, which is more than random guessing, but still a lot less than expected.

Finally, we prepared the function for saving the models (neuron weights) we tested and the function for saving predictions on unlabelled data as .cvs file. The .cvs file created for the final model showed us some interesting trends. For example, the model was often identifying person 29 as person 8, but never vice versa.

In order to be able to test the model with one, unlabelled image, we created the prediction model, which allows the user to load one image by searching for it in the file explorer and then it outputs the prediction of it by printing out the top five IDs it predicted, along with the possibilities for each of them.

In order to further improve our result, we suggest using more images, adding new subjects, using bigger images and maybe use a more complex model which requires bigger computational power.