Facial Matching in Electronic Voting Systems

Gama Marius Catalin

1 Introduction

Facial recognition is present in numerous systems and procedures in the modern day, and the fact that consumer products such as mobile phones have implemented it successfully is proof that something led by big organizations such as companies and governments could yield positive results and a decrease in frauds when implemented for an electronic voting system.[6] The current romanian voting system has a lot of flaws, however I believe that the biggest one is the fact that there are only a few central locations where people may go and cast their vote, and also a lot of bureaucracy is involved, taking a lot of time from the voters and the managing staff. Small kiosks and stations could be scattered around general points of interest (town halls, public parks - for government managed polls; main halls, cafeterias - for companies) and here the voting system might be present, where the user will go and scan their IDs, and a camera will take the live feed of the person casting the vote and compare it to the picture found on the ID card or in a central database, only getting the vote through if the software gives it the green light.

2 Approach

2.1 Detecting faces

A big part of setting up a facial recognition system is detecting if there is actually a face in the current frame, and that can be quite easily done with a series of cascading feature maps [5][1] that go over each part of the input image and check if the given section passes all the given checks.

This approach is very reliable and it can yield very accurate results as the subject doesn't need to go out of it's way to stand a specific way in order to match the processed section of the video feed, however the process is quite computationally expensive, and as such it lead to unoperabe loading times for each frame, thus it came to my attention that as a proof of concept it shouldn't be used, however in a real application it might be a good idea to implement such a thing.

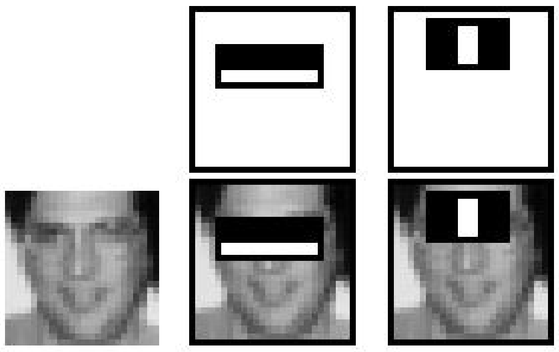

Figure 1: Cascading feature maps used to detect faces

2.2 Comparing faces

Another key aspect of a facial matching system is that of actually managing to check if two given images share any kind of similarity, and as such an easy way of converting the given input to something that a computer can easily understand was needed, and as such the best approach was converting each image to a mathemathical equation [2][3] that could be easily transformed and compared to another one of the same layout.

3 Architecture

Given that the algorithm needed to take two images and process each one such that it could compare them in the end an architecture that took the greatest area of interest from each image and process it in a way that makes it able to be compared to the other one, such that a great approach that I found was using a convolutional siamese network [4] that works on two separete streams, one for each image, and in the end merges them to compare the two inputs. The network takes as input two images of shape 100x100x3 and processes them on three different sets of convolutional and max-pooling layers and then in the end another an additional convolutional and dense layer. This dense layer acts as the "equation" [2][3] that defines the given image. The comparison comes up as a L1 distance layer.

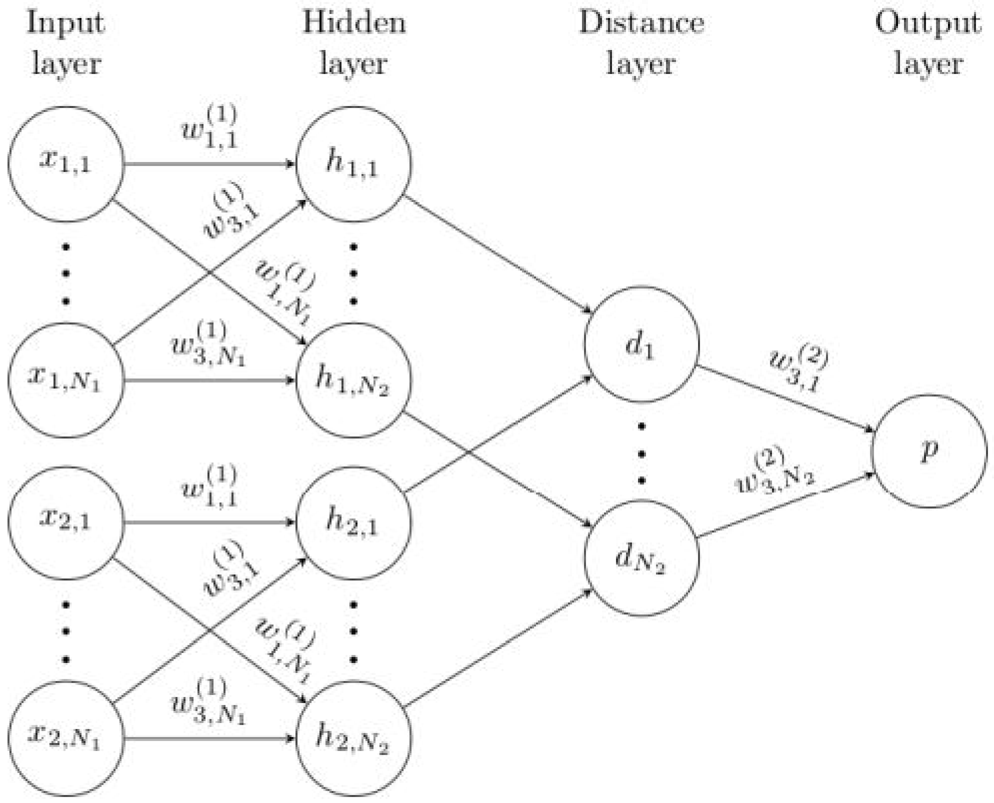

Figure 2: Basic layout of the siamese neural network

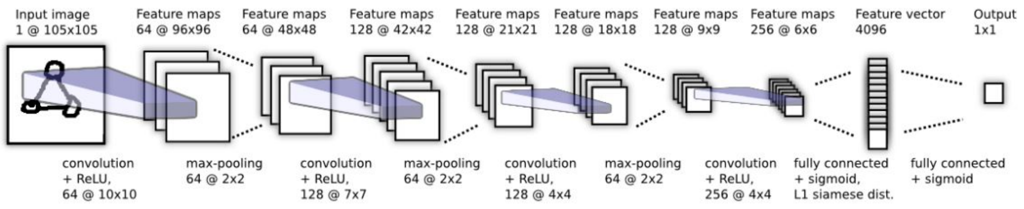

Figure 3: Layered view of the siamese network for one input, the siamese twin joins after the 4096 dense layer

4 Experimentation

4.1 Training

The network was trained using sets of 300 anchor, positive and negative images containing faces. Both the anchor and positive sets of images were recorded on the spot by taking pictures using the CV2 Python library through the webcam of the device, and the negative samples were part of the ”Faces in The Wild” dataset.

The training procedure was run for 50 epochs, and each pair of images took my system 14-17ms, definetely could be improved if the training was done on a GPU, and the accuracy of the network could have been even greater of more epochs or more samples were used.

4.2 Testing

4.2.1 Automated testing

A batch of 30% of the used dataset was reserved for testing, and after two training iterations the results were almost 100% for each test run that I did

4.2.2 Live testing

For live testing the siamese network was embeded into a live iteration of the application, where the user could validate the input taken from the device's webcam, and it was checked against a preset image that was used for validation (something similar to what a mobile phone has for FaceID). As expected, the failure rate of this iteration of the application was higher than that of the automated testing, resulting in 70-80% accuracy, however this number can be improved using a combination of more training runs or different threshold values for the validation of the image.

5 Closing Statement

The siamese neural network [4] that I have used for this experiment needs a little bit more polish and an integration with a facial detection algorithm[1][5] in order to be let out in the real world, however as a proof of concept for what can be done with relative ease for integrating a facial matcher into different applications it should be considered a successful experiment.

References

[1] Computerphile. Detecting faces (viola jones algorithm) - computerphile. 2018.

[2] Computerphile. How face id works... probably - computerphile. 2018.

[3] William Crumpler and James A. Lewis. How does facial recognition work? 2021.

[4] Ruslan Salakhutdinov Gregory Koch, Richard Zemel. Siamese neural networks for one-shot image recognition. 2021.

[5] Michael Jones Paul Viola. Rapid object detection using a boosted cascade of simple features. 2001.

[6] Jan Willemson Sven Heiberg, Kristjan Krips and Priit Vinkel. Facial recognition for remote electronic voting – missing piece of the puzzle or yet another liability? 2021.