Face Recognition Project

Vardan Papyan
CS Department
Technion - Israel
vardanp91@gmail.com

Emil Elizarov
CS Department
Technion - Israel
semil123@t2

October 31, 2013

Instructor: Igor Kviatkovsky
Prof. in change: Prof. Ehud Rivlin

1 Introduction

Our goal was to create a fully operational mobile application which could detect, recognize and track human faces.
In order to do that, we have decided to use the Android[3] platform combined with the opencv library[4][5].
The development of the application was made on Qualcomm MSM8960[6] mobile device which run a 4.0.3 Android OS.
In addition to the application we have built, we also did a research about how well we can use LDA[1] and PCA[2] in order to recognize faces and also about the use of LDA in order to do basic pose estimation.

1.1 Definitions

We will now explain some of the terms in which we will use from now on:

Android application Android is an operation system that is mostly used in mobile devices. The applications on Android are primary written in Java[7].

Android NDK/native methods The NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++ instead of using Java.[8]

Opencv library OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial

products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.[5]

Smartphone's buttons On every android smartphone there are three buttons that you must be familiar with in order to operate our application:

- Home button.
- Return button.
- Menu button.

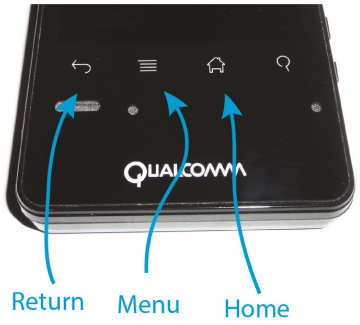


Figure 1: Qualcomm MSM8960 buttons

1.2 The application's running environment

We decided to write our application in Java with Android NDK functions which we wrote in C. What Android NDK means is that you can write functions in C/C++ in some special syntax, compile it seperatly and then link it to the Java application via what is called JNI[9] (Java Native Interface).
The main reasons that caused us to use native methods are:

Opencv's functionality most of the opencv library is written in C/C++, and there are large parts which have been converted into Java, but there were cases in which we wanted to use methods that were written only in C/C++, so we were obligated to use Android NDK.

Performance as you know, C/C++ code runs faster than Java code (C is considered as "low level" language).The heavy operations in our application are those that involve the calls to the opencv methods. For this reason most of the calls to the opencv methods are done via the native methods which allow us cosiderable boost in processing speed.

1.3 Use the application

Now we will show you how to use our application.
First of all you need to launch the application, it is done like launching any other Android application:

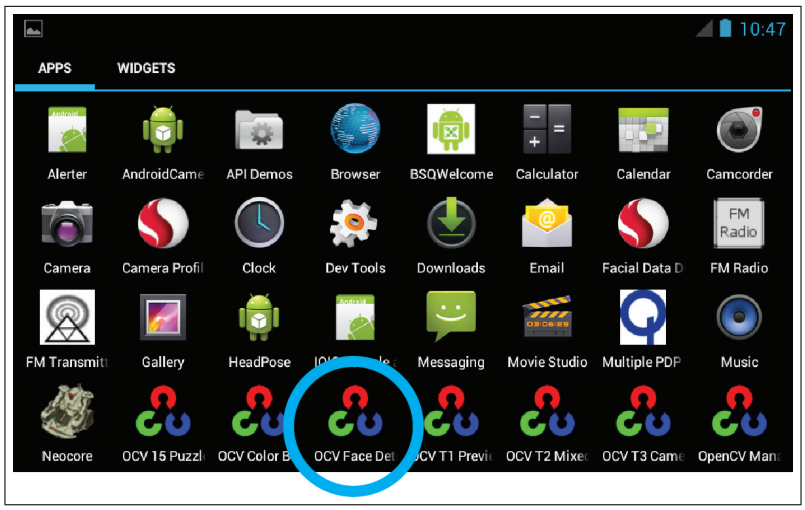


Figure 2: Launch the application

After launching the application we will see the application:

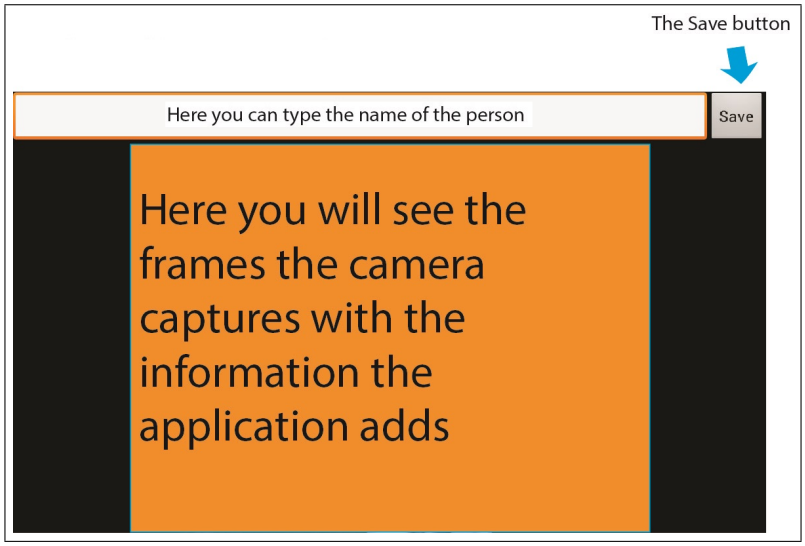
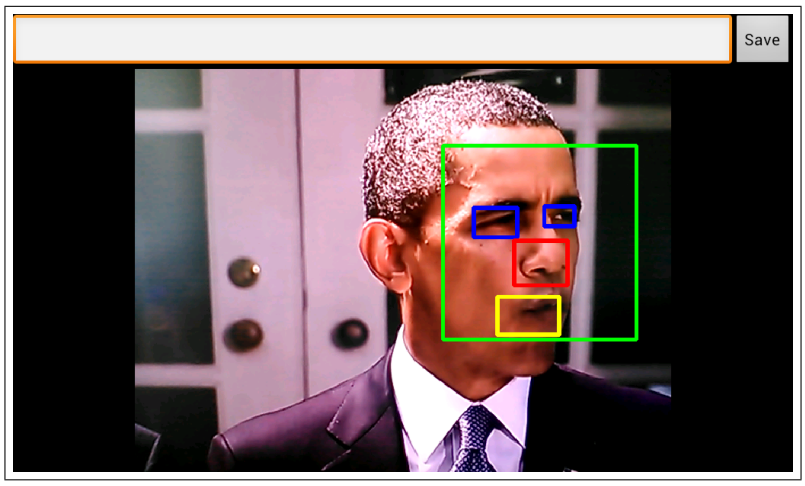
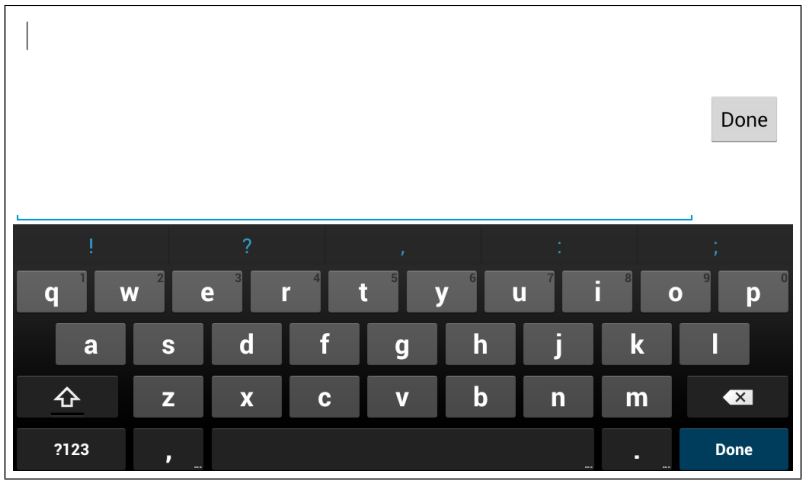


Figure 3: The application's layout

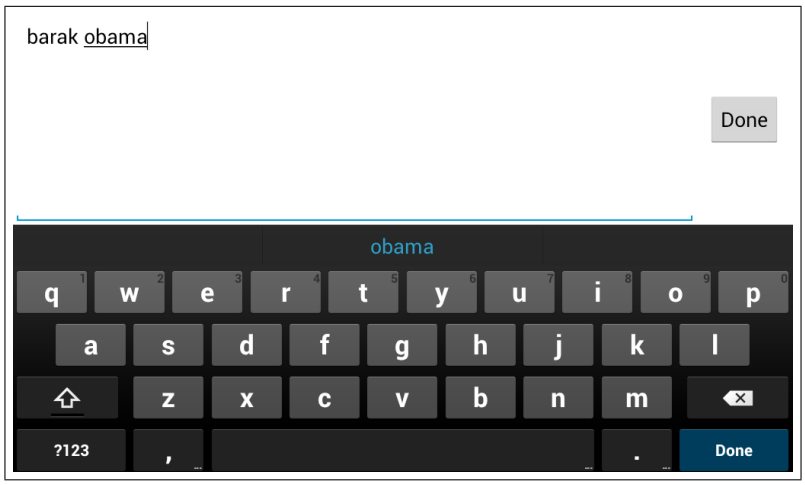
Now we will use the application on one of president Obama's speeches:



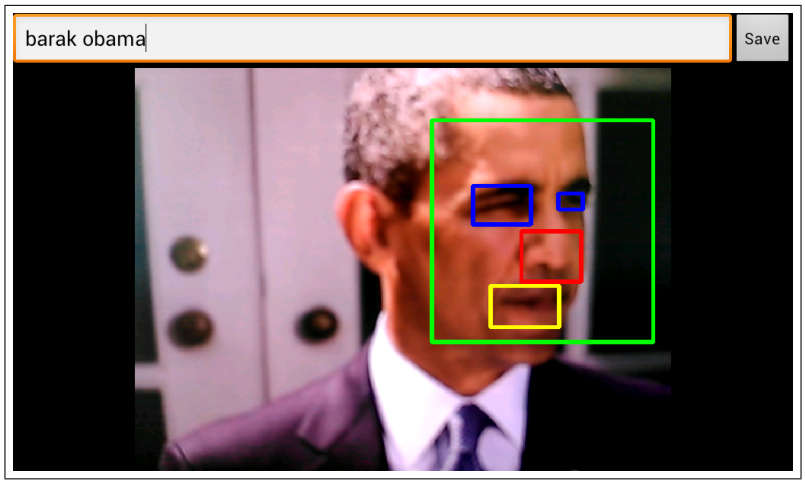
We can see that the application automatically detects faces that are seen in the frame, we can see that there is a green bounding box around the face, two blue bounding boxes around the eyes, a red bounding box around the nose and a yellow bounding box around the mouth.
We want to make the application to recognize Obama when his face is captured in the frame, in order to do that we need to save a several recordes of his face: Click on the white line that is on the top of the screen. The following window will open:



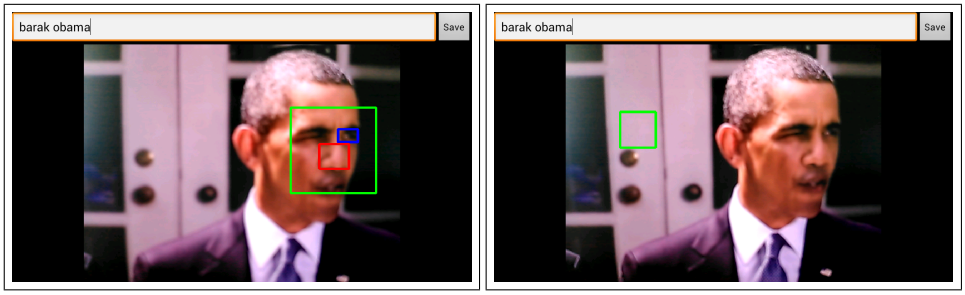
Now type the name of the person, we will type "barak obama", afterwards we will press the "Done" button.



Now we will see that Obama's name appears in the top white box.

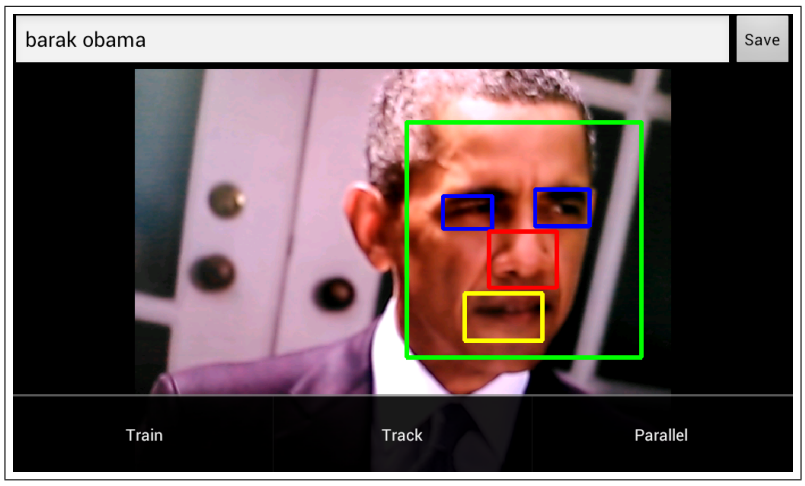


In order to record a face we need to press the "Save" button, when pressing the "Save" button the face which all it's parts (eyes, nose and mouth) were detected will be saved. But as you might have seen, there are some moments when the application losts the face or some of it's parts, for instance:

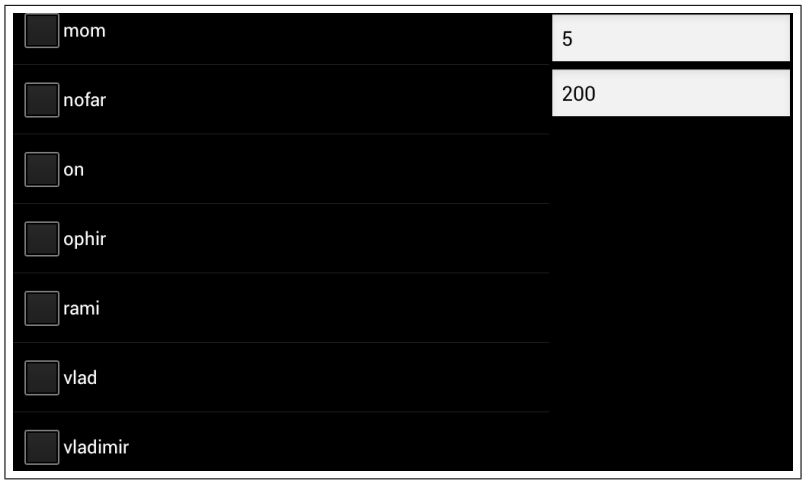


But do not worry, the application saves the last "complete" face which it detected, and when you press on the "Save" button that face is the one that is being saved to the database, for this reason you do not need to be "with the finger on the trigger".
We will save few samples of Obama's face, we will wait few second for Obama to change the position of his face and then hit the "Save" button. For example, these are three samples which we took:

Now we wish the application to recognize Obama's face, in order to do that we will press the smartphone's "Menu Button", the following menu will show up:



Press on the "Train" button and the following screen will show up:



In this screen you will see:

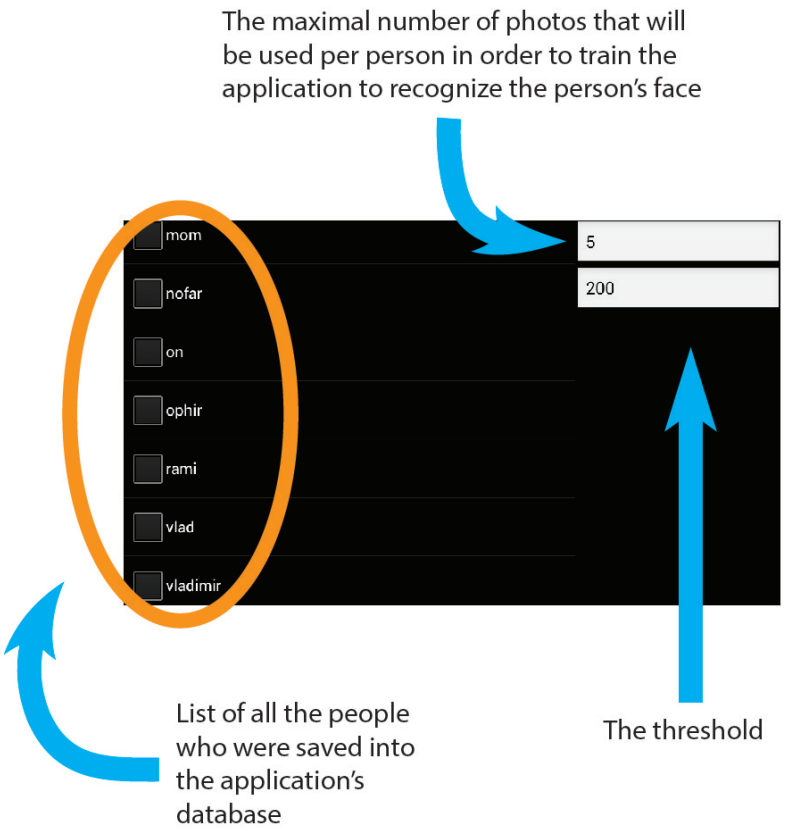
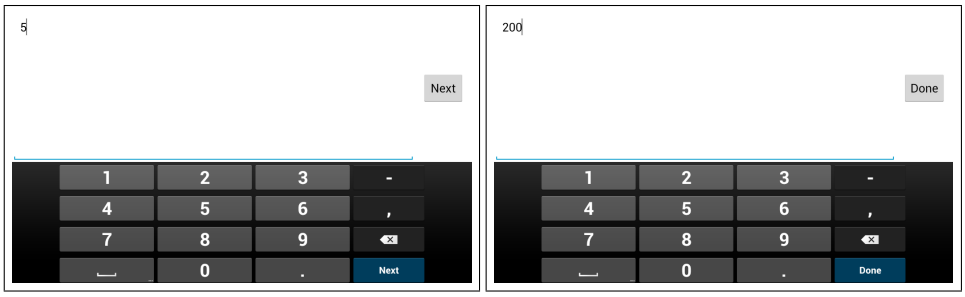


Figure 6: Train screen layout

Note: The "Threshold" thing will be explained in detail in the following sections of the article.
Scroll the list and choose all the people you want the application to recognize, in our case it will be just "barak obama".

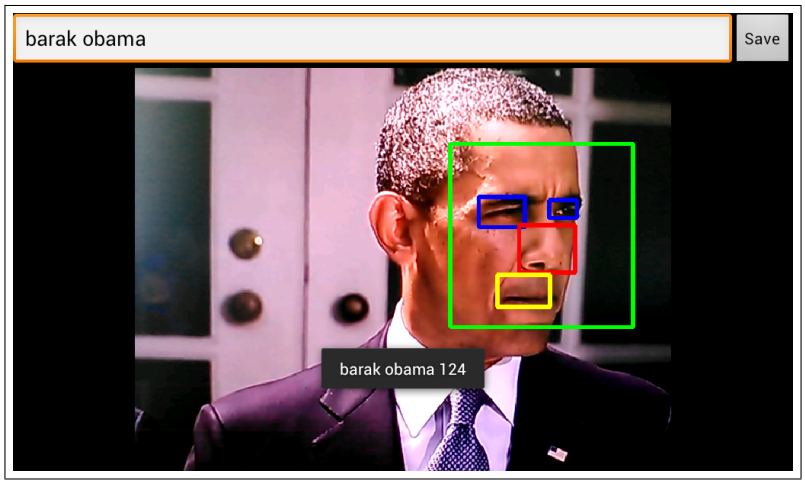
It is optional but you can also change the number of photos and the threshold. The reason why you would like to increase the "number of photos per person" is to make the recognition more accurate, and the reason to decrease it will be to make the training process and the application run faster. The reasons behind the increase and the decrease of the Threshold will be discussed in subsection 4.5.
In order to change the above values just press on the relevant one and then this screen will show up:



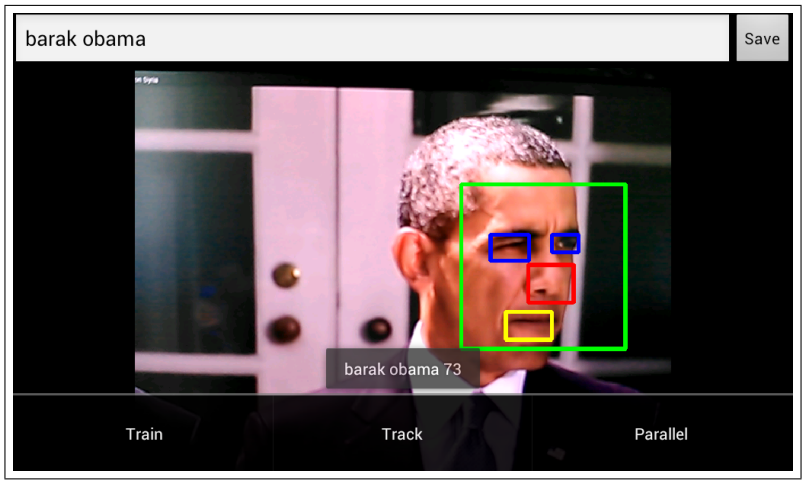
(a) Number of photos value change (b) Threshold value change

Now press on the smartphone's "Return Button" – this will invoke the training process which will train the application's face recognizer and it will as well return you to the application's main screen.
Now after the application detects a face it will check whether it is the face of one of the people that were marked in the "Recognition list", if it is, it will show the name of the person and the "difference" between the shown face and the face which the application considers to be the face of that person, if not it will act as it acted before – it will just show the bounding boxes.
Note: If this "difference" is greater than the threshold then the application will

act as if this face is not the face of the person, and you will just see the regular bounding box.

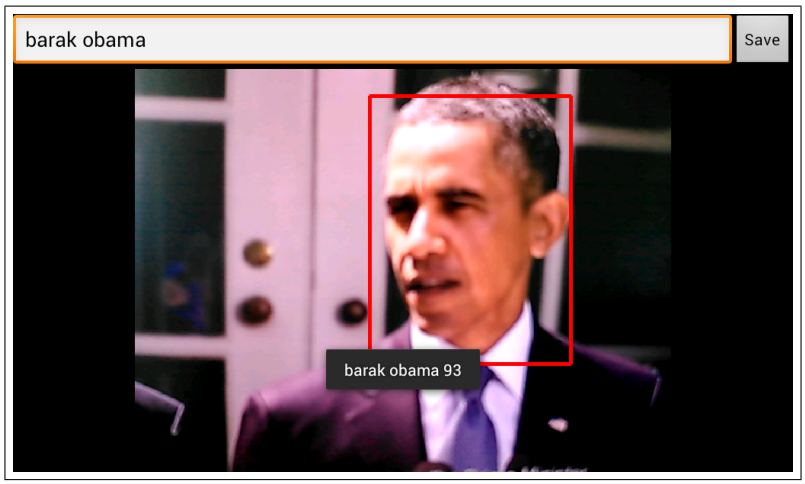


Now we wish to track Obama's movements.
We press on the smartphone's "Menu button".



Then we press on the "Tracking" button.
The multiple choise list will be shown and we will choose "barak obama":

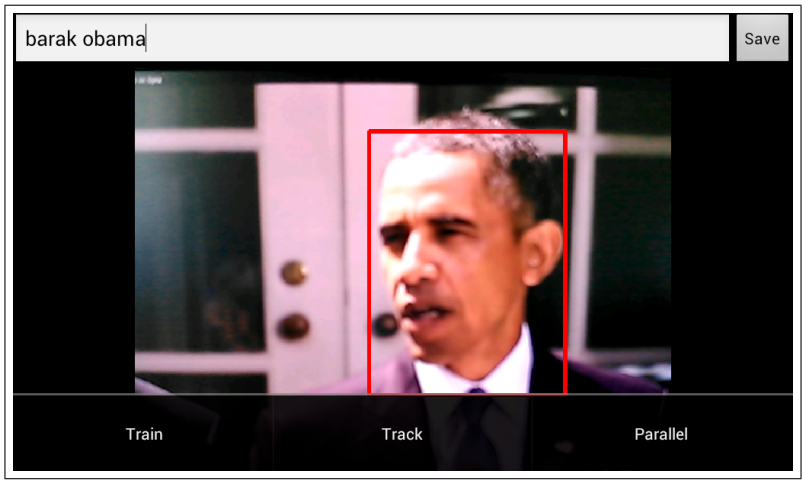
We will press on the smartphone's "Return Button".
This will bring us back to the main screen and now when Obama's face will be detected and recognized the tracking red counding box will show up:



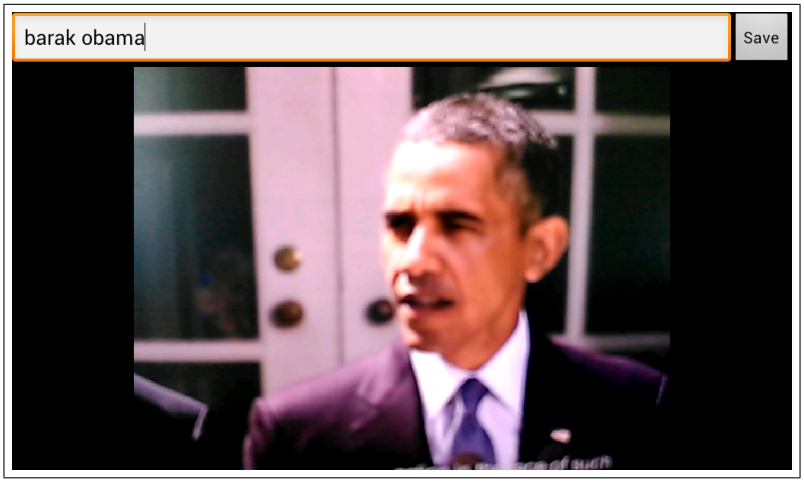
The box will follow Obama's face and will not let go, no metter what happens. The tracking is based on the face's color, so the face can tilt and turn to the other side and so on. If the person exits the frame and then enters again – the detection and the recognition will start again the the red tracking bounding box will be showed once again.

Figure 8: The tracking in action.

The last thing that remains is to explain the "Parallel" button. Press on the smartphone's "Menu Button".



Then press on "Parallel".



The parallel mode is creating another thread which will do the analysis of the frame, i.e. all the detection, recognition and tracking are done by another thread, thus we will not see any bounding boxes and so. One can notice that the FPS (Frames Per Seconds) rate is significantly increased.

2 Android and class diagram in a nutshell

We will now introduce a detailed explanation about the application's structure.

2.1 Explanation about the basic form of an Android application in a nutshell

As introduced, the application is written in Java as an Android application. Now, intuitively what is the "use cycle" of every mobile application?, namely, what states the application can be at?
The main states the application can be at are:

- Running, it is the most essential state
- Starting for the first time since you permanently exited it
- Pausing
- Exiting
- Returning from pause

What the Android platform cleverly do is supplying an interface of methods that define/implement each of the above states and obligating the developer to implement each of the methods. We can see it more clearly using this chart[10]:

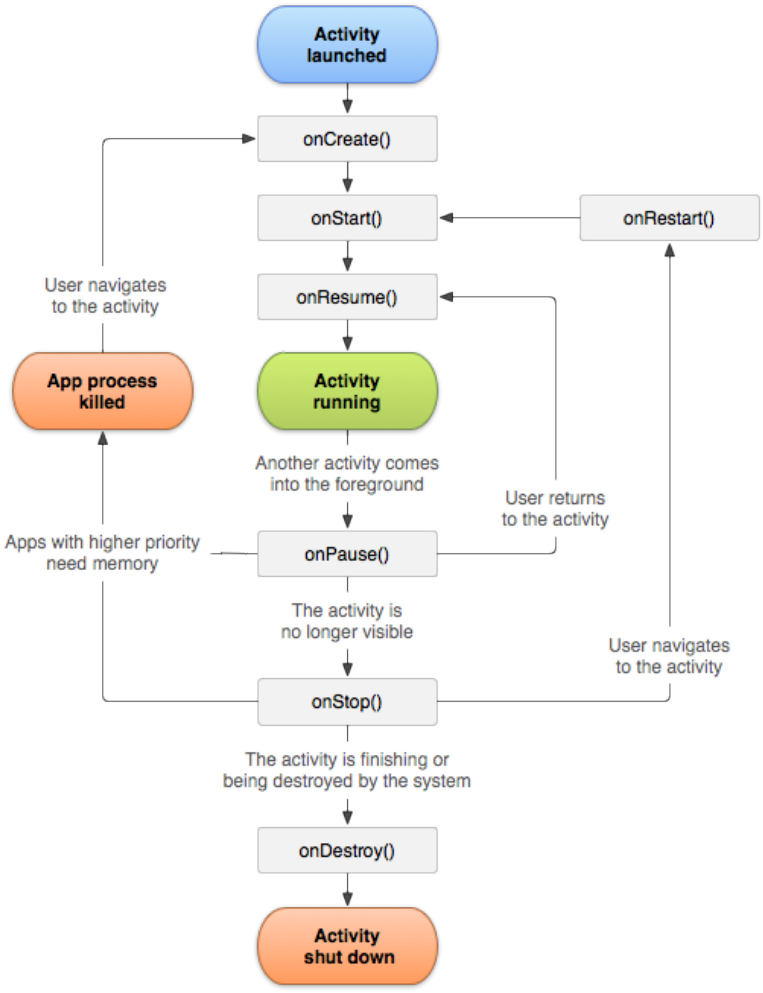


Figure 9: Activity lifecycle chart

We need to mention that we implemented more methods than described here. Methods like:

onCameraFrame which is called when a new frame comes from the camera

onCameraViewStopped which is called when the camera is stopping

etc.

are just exaples to those "additional methods".
All the above and the other methods that we implemented are essential in order to write an Android application which combines Opencv with the use of the smartphone's camera. We will not discuss these methods here due to an

excellent documentery that has been already written in the interfaces which define these methods.

2.2 Class diagram of our application

Understanding the basic form of an Android application, we now move to the class diagram of our applcation.

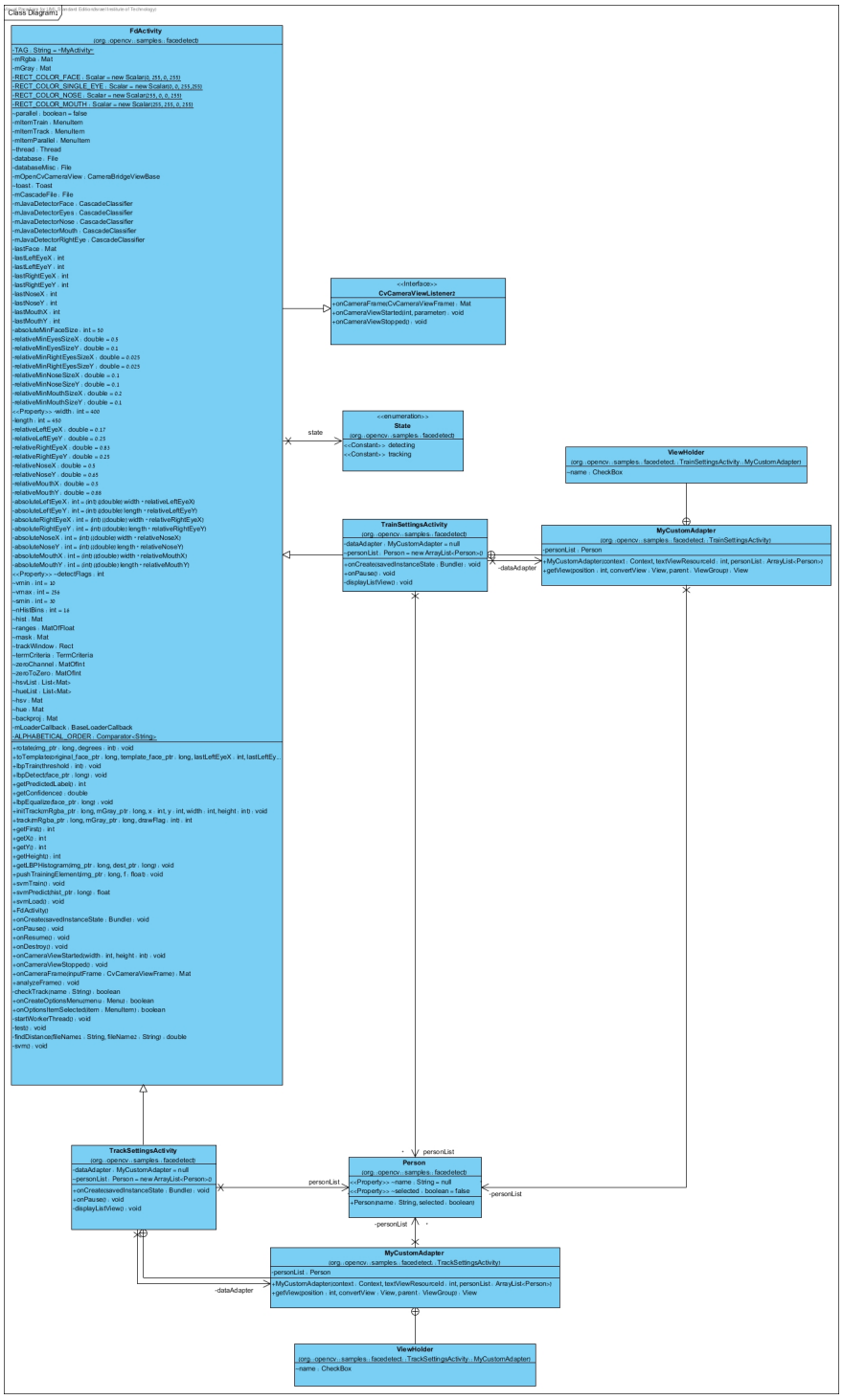


Figure 10: The application's class diagram

Explanations and details about the classes:
Note: Some details may not be perefectly clear now, because detailed explanation about the algorithms and their relations to the code is given in the next sections.

State This is an enum class which helps us to describe the state of the application – wheter we are "detecting" now, or if the detection is finished and we are "tracking" a person now.

CvCameraViewListener2 This is an interface that the class FdActivity have to implement in order to use the smartphone's camera.

FdActivity This is the most essential class in the project, you could say that all the other classes just support it. This class is doing or invoking other methods (mostly native methods) which are related to the detection, the tracking, the storing and all the other functions that have already been described and will describe throughout the article.
This class also contains the important method "analyzeFrame" which will be discussed later on.

TrackSettingsActivity This class is implementing all the screen which lets the user select which people he would like the application to track.

TrainSettingsActivity This class is implementing the screen which lets the user select people he would like the application to detect. In addition the "threshold" and the "number of photos per person" fields are handled here as well.

MyCustomAdapter This class is just technical necessity that alows us to implement these "multiple choise lists" in Android.

ViewHolder Implemented as a class just to allow convinient future changes to the application.

Person This class allows us to represents a person in the application. We need it because this application handles human faces, and we need organized way to store all the data on these people.

3 Face Detection

As seen, the application can detect human faces. In this section we will explain how this detection is implemented.
While the application is working, the camera is capturing frames and passing them to the application. When a frame arrives from the camera, the application moves the new frame to the analysis method "analyzeFrame". In this method, inter alia, the process of the face detection in the frame is being done.
In the default mode of operation, which is "detecting", the method "analyzeFrame" analyses the frame and creates an array of "bounding rectangles" which represents the exact parts of the frame that are strictly contains human faces.
In order to detect faces the application uses the OpenCV's method "detectMultiScale"[11].
(In short, the method is implementing a Viola-Jones algorithm[12] to detect objects in an image, given a cascade that was prepared in advance.)
The method "detectMultiScale" is getting, inter alia, the frame (the frame is represented by a matrix - the class "Mat") and a cascade (in our case we give the method a cascade of a human face) as parameters and the method returns an array of bounding boxes/rectangles of all the detected faces in the frame.
After the creation of the array of the bounding rectangles, the method iterates over all the faces that were detected and tries to detects the eyes, the nose and the mouth of each and every detected face. All the above "parts of face" are detected in the following way:
We set our main ROI (Region Of Interest) to be the bounding rectangle of the face meaning by any case we do not look or do actions on the parts of the frame that are out of the boundaries of the ROI. Now we divide the face into sub-ROIs:

- ROI of the left eye
- ROI of the right eye
- ROI of the nose
- ROI of the mouth

Now we use the "detectMultiScale" method on every ROI - we give the method as parameter the image/frame that represents the part of the frame that is related to the mentioned part of face, and we give the relevant cascade for this ROI in order to detect the relevant part of the face. E.g. when we wish to find the left eye, we set the left eye ROI, as will be described, and we use the "detectMultiScale" method on that ROI using the left eye cascade.
We found out that the best parts of face ROIs division is:

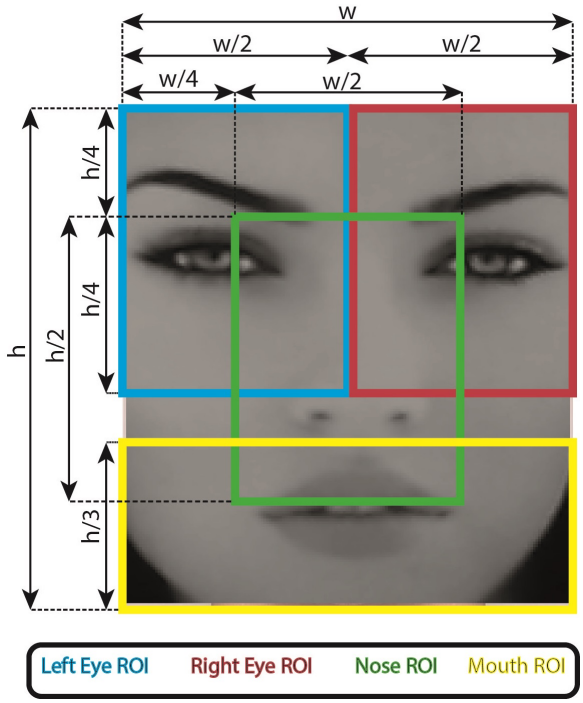


Figure 11: Relative parts of face ROI division (on a face that was detected by the first call to "detectMultiScale")

These ROIs division get very good results for most of the faces that are returned by the first call to "detectMultiScale".
The division to ROIs before running the detection of the parts of face is important! if we do not do the division the results will be:

considerably longer processing time due to search for the parts of face in non productive areas, e.g. the left eye cannot be in the bottom or in the rights side of the face, it will be a waste of resourses to search for it there.

enhanced false positive detection rate the detection is not perefect, there is a false positive rate – the larger the region we search an object on, the larger the probability for false positive detection will be. When we limit ourselves to only the most relevant areas we reduce the probability to get a false positive detection.

In case the detection of the parts of the face is erroneous (less than one of each part of the face was detected, e.g. no mouth was detected) - we abandon this face and do not continue to the recognition level with this face - most probably in a few frames we will get a better look of the face that will allow us to detect its parts without errors.

4 Face Recognition

4.1 Introduction

Our application supports recognition of faces of people who were recorded into the application's data base. In this section we will explain the implementation and the "field results" of the face recognition part in the application.
This section will be a direct continuation to the "face detection" section.

4.2 Initialization of the face recognition mode

As seen, we can press on the smartphone's "menu button" and choose the recognize option. Now we will:

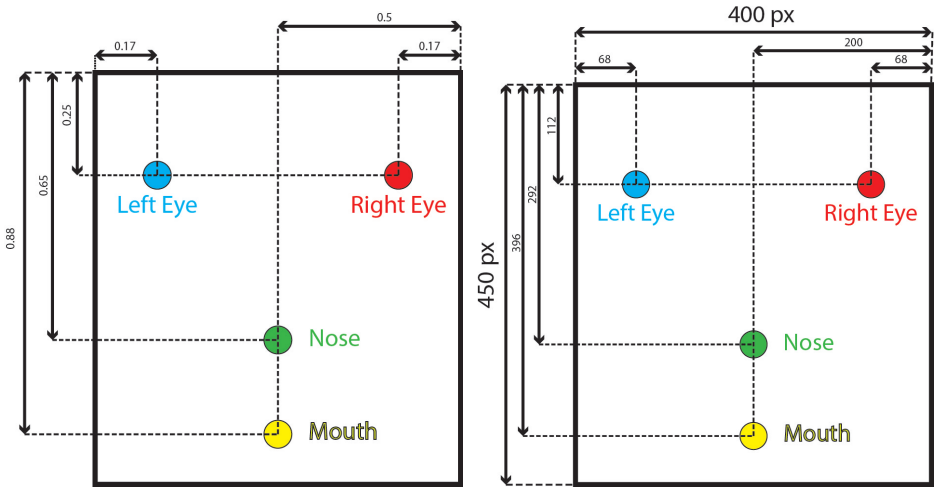
1. select all the people who we want the application to train on them, i.e. who we would like the application to recognize.
2. choose the wanted threshold.
3. choose the number of template face files (the meaning of "template face form/files" will be explained in detail in subsection 4.3) per each person that will be used by the application to train the face recognizer, when we have more template face files in the application database than required we will choose randomly the requested amount of files per person from the datavbase.
4. press on the "back button".

Note: there are default values for 2. and 3.
By doing so the application creates a special file named "TrainSettings.txt", the file is saved to the SD card. This file will contain paths to the files which contain the template forms of the faces of the people we would like the application to recognize.
When the choosing is finished the native method "lbpTrain" is invoked, in this method we use the "FaceRecognizer"[13] class of the OpenCV library which does the follows:
An object of the class "FaceRecognizer" gets all the photos of the faces in the template form, now the object devide each face to 8x8 equal pieces and extracts a normalized LBP[15] histograms (normalized means that the sum of all the entries of the histograms is 1) of size 1x256 for each of these pieces and concatenates the histograms into one histogram of size 1x(8*8*256)=1x16384. These histograms (one per each template face photo) are being stored to the smartphone's main memeory.

4.3 Template face form

As we already explained, the method "analyzeFrame" iterated over all the faces that were detected in the frame, and for each face the method finds the eyes,

the nose and the mouth of that face. If there are no errors in the detection then we continue to the recognition part.
In this part we take the face and find the center point of each part of face: it is simple - each part is already bounded by a detecting rectangle, if the width of the rectangle is w and the height of the rectangle is h then the center point of this part of the face will simply be ($\frac{w}{2} + topLeftXCoordinateOfTheFace$, $\frac{h}{2} + topLeftYCoordinateOfTheFace$). Now we use these points and transform the face into what we call a "template face":



(a) Relative face template (b) Non relative face template

Figure 12: Face Templates

After many trials we have decided that this "template face" (represented by a 400x450 pixels photo/matrix) is the most suitable for the faces that the detection method "detectMultiScale" is returning to us.
We have also decided that this size should be ideal in terms of storage space-recognition efficiency.
In order to transform the detected face to the template form we invoke the native method "toTemplate", this method calls the OpenCV method "warpAffine"[16] that applies an affin transformation to the face using the center points of every part of the detected face respectively to the center points of the template face and returns us the face in its "template form".
It is important to mention that all the faces in the application's database are saved in the template form.

4.4 Recognize a face

Now the template face form can be used in order to do the recognition. It is important to understand that without the template face form the LBP histograms[15] that are used in order to recognize the face will not be accurate enough for the recognition to work properly, i.e. the probability of a correct recognition drops drastically.
One example for that is if the face is a little tilted then without the template form errors will arise and this "little" may be enough to get a recognition error.
We are using this template face form and by invoking the native method "lbpDetect" we find out if this face matches one of the faces of the people that we wish to detect. In order to do that, this method is using the OpenCV's "FaceRecognizer"[13]: the "FaceRecognizer" class has the mehod "predict" which gets a template face as parameter, converts it to LBP histogram of size 1x16384 and then using Nearest Neighbor with the help of the Chi-Square norm:

$$d_{Chi-Square}(H_1, H_2) = \frac{1}{N} \sum_{i=1}^{N} \frac{(H_1(i)-H_2(i))^2}{H_1(i)}$$

it finds the most resembling face to the face we are trying to detect.
In detail, we have a lot of histogrames already in the memory, and each histogram has a tag with the name of the person "attached" to it, what "perdict" does is to iterate over all the histograms that are in the smartphone's memory and compare these histogram to the histogram of the face which we are trying to recognize by using the Chi-Square norm. As you might have noticed, the Chi-Square norm is not a symmetric norm, the OpenCV implementation using the histograms in the memory as H_1 and the histogram of the face which we are trying to detect as H_2. The two compared histograms are considered to be histograms of the same face if the Chi–Sqare distance between the two histograms is less than the threshold. Do not worry, the threshold issue will be explained in detail in the next subsection. Note that we find the nearest neighbor – i.e. the histogram in the memory which gives us the minimum Chi-Square distance, and the tag of that histogram is deciding the recognition (of course only if less than the threshold).
Now lets go back to the application: if we find a match then we are writing the name of the person under the bounding box and we show the user the distance between the given face and the person's actual face (the minimum distance).
We want to emphasize again that if the faces are not similar enough (more than the threshold permit) then we won't show any recognition, as if the application is still "searching" who is this person.

4.5 Experimental results of the face recognizer

In this subsection we will try to understand the efficiency of the face recognizer we have built, and the impact of changes on the threshold.
As seen in the last subsection, the distance between the LBP histograms of two face (the expected face and the face which we are looking at now) is represented

by a numerical value, high distance signify that the faces are very different from each other, and a low distance signify that the faces are similar one to the other. Hereby the threshold is a numerical value that represents how much unsimilarity we can tolerate,on the one hand if the threshold is high so is our tolerance – we can tolerate massive unsimilarity between two faces, on the other hand, if our threshold is low then we can tolarate only little portion of unsimilarity between two faces.
The recognition is influenced by the threshold, and we may say that the recognition is based on the threshold, for this reason it is essential to determine the most suitable threshold to recignize human faces using our application. That may not be an easy task: on the one hand, if the threshold is too high then two different faces might be classified to be the face of the same person, on the other hand if the threshold is too low then two faces of the same person may be classified as faces of two different people.
Thus, it is important to determine which threshold would give us the best trade-off between the two situations.
Because of that, we performed an experiment: we went out and used the application to take out ≈4-5 portrait face photos of 22 people. We used these pictures to analyze and determine the best threshold.
We divided those photos into pairs and sorted the pairs into two classes: "Same" - the pair of faces belongs to the same person and "Not Same" – the pair of faces does not belong to the same person. Afterwards we built a histogram of the distances. We transformed the histogram into distance-probability histogram (normalized the histogram) and drew two graphs:

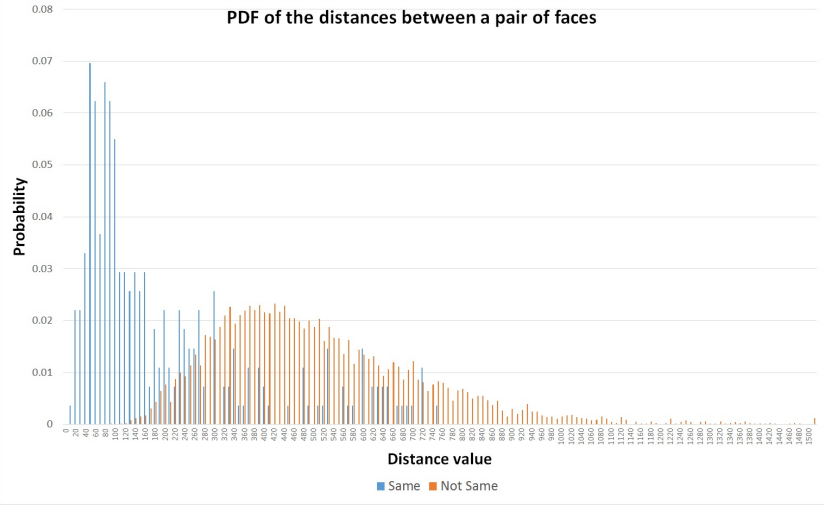


Figure 13: Probability distribution function of the distances between a pair of faces

Now let us look at the ROC[17] curve which represents the true positive rate and the false negative rate as a function of the threshold:

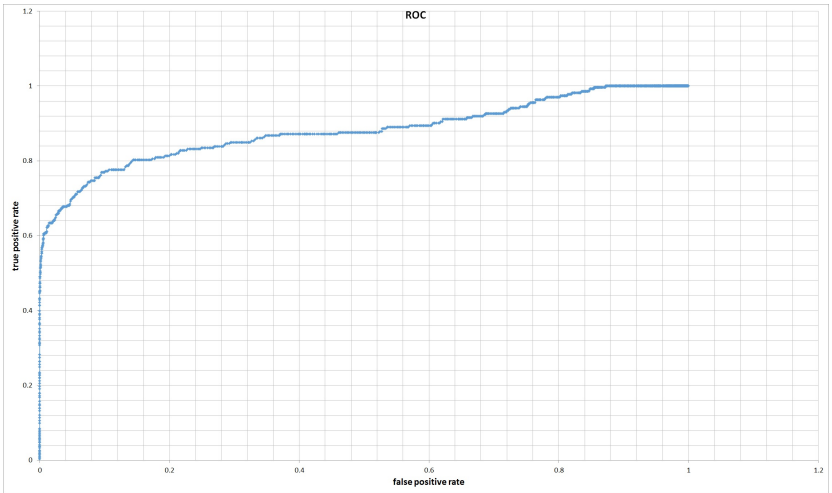


Figure 14: Threshold ROC using Chi-Square norm

From the ROC curve we can learn that a threshold of around ≈290-350 can give us a TPR of 0.78-0.83 and FPR of 0.13-0.25 which is very good. The table below should detail the top left area of the ROC graph:

Threshold	TPR	FPR	Threshold	TPR	FPR
291	0.783883	0.132249	321	0.809524	0.188682
292	0.787546	0.133813	322	0.809524	0.191027
293	0.787546	0.13522	323	0.813187	0.193216
294	0.787546	0.136939	324	0.813187	0.19556
295	0.791209	0.137877	325	0.813187	0.197436
296	0.794872	0.140066	326	0.813187	0.199781
297	0.798535	0.141942	327	0.81685	0.202282
298	0.802198	0.143817	328	0.81685	0.203689
299	0.802198	0.145224	329	0.81685	0.206972
300	0.802198	0.146631	330	0.81685	0.209004
301	0.802198	0.148194	331	0.820513	0.211505
302	0.802198	0.150696	332	0.820513	0.213381
303	0.802198	0.152103	333	0.824176	0.214632
304	0.802198	0.153509	334	0.827839	0.216351
305	0.802198	0.154916	335	0.827839	0.217758
306	0.802198	0.156949	336	0.827839	0.219947
307	0.802198	0.159293	337	0.827839	0.221666
308	0.802198	0.161169	338	0.827839	0.223386
309	0.802198	0.163514	339	0.827839	0.225731
310	0.802198	0.16539	340	0.831502	0.228388
311	0.802198	0.168204	341	0.831502	0.231202
312	0.802198	0.169298	342	0.831502	0.232922
313	0.802198	0.171799	343	0.831502	0.234798
314	0.805861	0.173988	344	0.831502	0.236986
315	0.805861	0.177271	345	0.831502	0.239018
316	0.809524	0.179146	346	0.831502	0.241363
317	0.809524	0.181022	347	0.831502	0.244177
318	0.809524	0.182898	348	0.831502	0.245897
319	0.809524	0.18493	349	0.831502	0.24746
320	0.809524	0.186337	350	0.835165	0.249492

Table 1: ROC's top left area description

5 Tracking

If the face that we recognized is on the list of the faces that needs to be tracked (this list is in the file "TrackingSettings.txt" and it is changed if we press on the "menu button" and then press on "tracking") then we will start the tracking - the user will see a red bounding box around the recognized face. Whenever the recognized person tilts his face, or turns his face - the bounding box will track his face and won't let go.
Now to the implementation: after recognizing the face we check whether we need to track this person or not we invoke the method "checkTrack" in order to do that. After a conformation we do two important things:

1. We change the state of operation from "detecting" to "tracking".
2. We invoke the native method "initTrack" which initializes the tracking algorithm which locks on the face.

Now every time we invoke the "analyzeFrame" method we would go to the section which handles the "tracking" mode of operation instead of the "detecting" mode of operation.
In this mode of operation we invoke the native method "track" which essentially invokes the method "CamShift"[18] of the OpenCV library. This method tracks the object (the face in our case) using color histgorams. It distinguishes the face from the environment around the face using the unique colors of the face. It returns the new bounding box of the face in the new frame by searching around the previous color blob. You may say that it does "color tracking" of the human face skin color. It is also important to understand that the tracking is done gradually frame after frame.
If the "CamShift" algorithm has lost the face, then we will return -1 and the calling method "analyzedFrame" will change the mode of operation from "tracking" to "detecting" to try and redetect the lost face, and then start tracking them again when the face is rerecognized.

6 Face recognition and Pose Estimation using PCA and LDA

6.1 Introduction

In the application we used the Nearest Neighbour with Chi-Square norm to recognize person's face. In this section we will see how good is the face recognizer using LDA[1] and PCA[2] and how can we use LDA in order to do a simple pose estimation.

6.2 Pose estimation

We used 3 classes:

Frontal face a portrait face photo.

Right profile face a profile photo of the face where the nose and the eyes are turned to the right.

Left profile face a profile photo of the face where the nose and the eyes are turned to the left.

Each class contains ≈35 photos. We wanted to see how well does the LDA can distinguish between these three classes.
We used the OpenCV library in order to get the same LBP histogram representation as we got in the previous sections (a 1x16384 histogram per each face).

Then we have written these histograms into a file and transformed all the data to Matlab where we ran for several hours a simple LDA on these three classes. Afterwards, as we have done in subsection 4.5 we created pairs of "Same" and "Not same".
Using the eighenvectors that the LDA returned we reduced our 1x16384 histograms to 1x1 and 1x2 matrixes. At this point it is important to say that the dimension reduction the LDA does can be used only to reduce the data to ((number of classes) -1) dimensions or less. After that we used the L2 norm in order to get the distance between each "same" pair and each "not same" pair. We have built two graphes – one for the reduction to one dimension and the other one for the reduction to two dimensions:

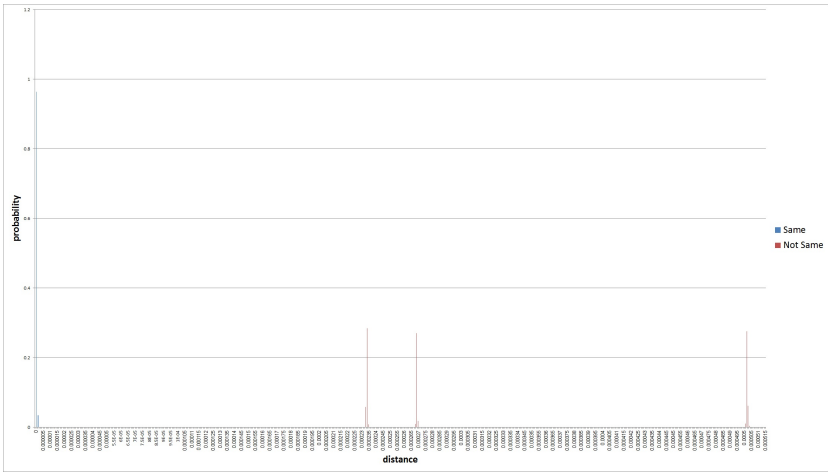


Figure 15: Probability distribution function of the distances between a pair of faces where the dimension reduction of the LDA was 1

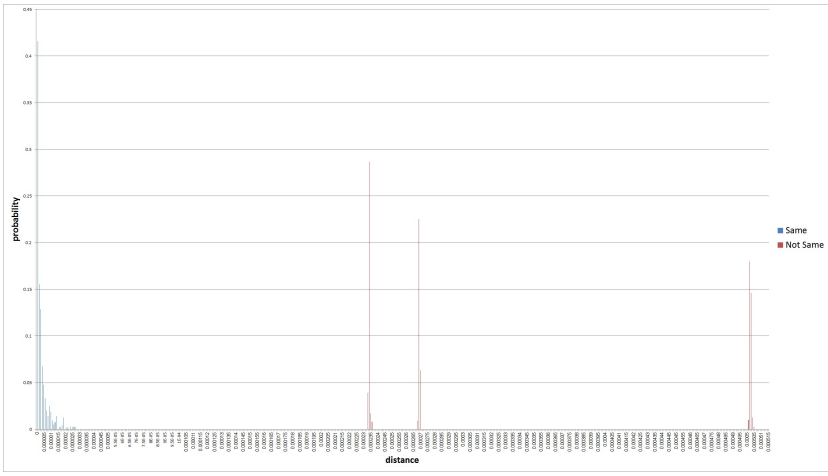


Figure 16: Probability distribution function of the distances between a pair of faces where the dimension reduction of the LDA was 2

As we can see the pose estimation using LDA was just perefect, the results for reduction to 1 dimension were better than those when we reduced to 2 dimension, nonetheless the two graphs gives us 100% classification accuracy.
Furthermore, we can see the area of the "same" class – near the 0, and three occurrences of the "not same" class:

- the first peak corresponds to the first and the second classes.
- the first peak corresponds to the second and the third classes.
- the first peak corresponds to the first and the third classes.

6.3 Face recognition – LDA only

We used a dataset of 25 people, each having ≈4-5 portrait photos of their faces and tried to see if LDA can give us good recognition.We used the same methods as before – extracted LBP histograms as before, converted them to Matlab and ran the LDA for few hours. Then reducing the original LBP histograms of size 1x16834 to 1-24 dimentions (remember – you can lower only to maximum of the number of classes minus one dimensions), created "same" and "not same" pairs, calculated the L2 distances and built the graphs.
The results were quite poor – no matter to how much dimentions we lowered the dataset, we only got separation of ≈53% (we mean that the best separation is a distance where 53% of the first graph are to its right and 53% of the second graph are to its left) in all the cases.

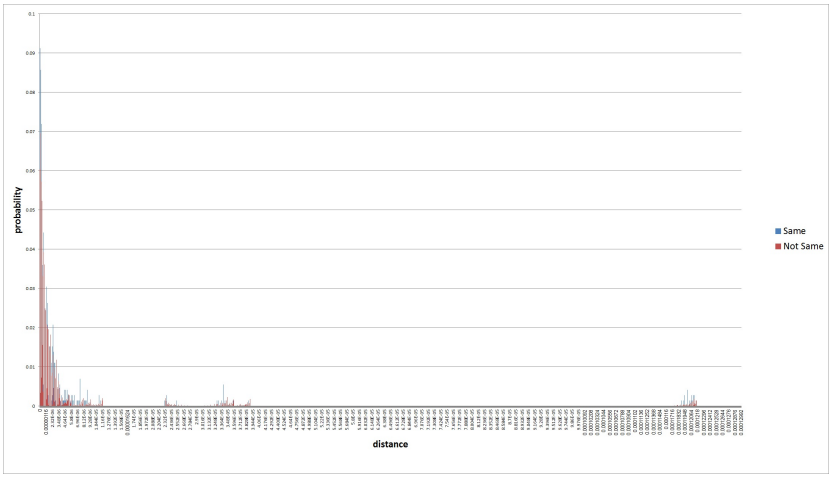


Figure 17: Probability distribution function of the distances between a pair of faces where the dimension reduction of the LDA was 10

6.4 Face recognition – PCA and LDA combined

After the failure in using only LDA we tried to first reduce initial dataset of the histograms of size 1x16384 using PCA and only after that using the LDA.
By using the "princomp" comment in Matlab we were able to reduce the histograms to a much lower dimension:

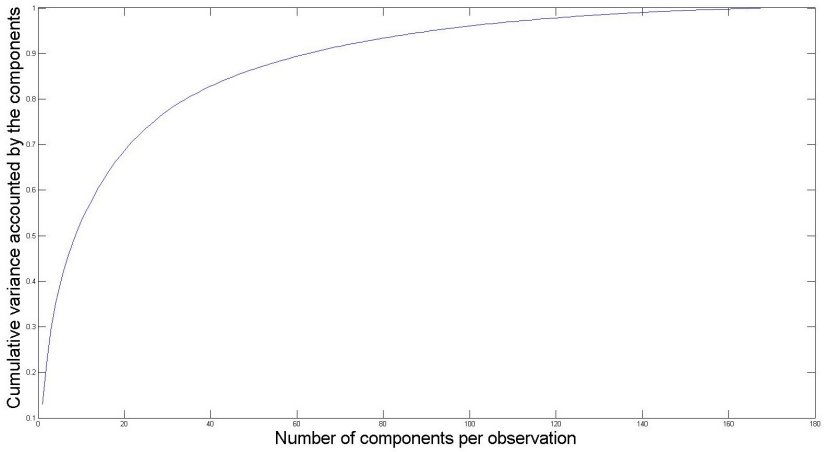


Figure 18: Cumulative variance accounted by the components as a function of the number of components

As seen in this graph, 171 components for each are enough to get variance of 100%, i.e. we can lower the 1x16384 histograms to 171 dimensions (1x171)

without losing variance.
From this point on we did the same thing as before – extracted the LBP histograms as before. Afterwards we did two loops – the outer is over the number of dimentions that the PCA reduces, while the inner loop is over the number of dimensions the LDA reduces the data which the PCA reduced .
When we used 23 of the PCA componenets – i.e. we reduced the LBP histograms from 1x16384 to 1x23 and we ran LDA on that data and reduced the data of the LDA to 22 dimensions, built the "same" "not same" graphs using the L2 norm we got 86% separation.

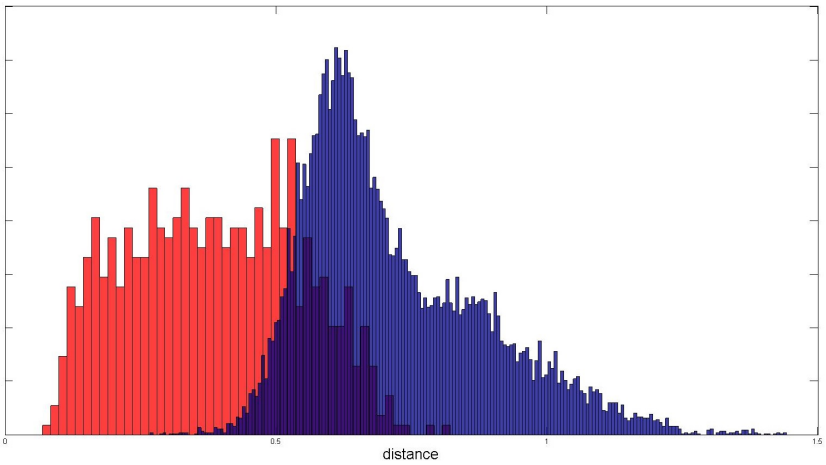


Figure 19: Probability distribution function of the distances between a pair of faces where the dimension reduction of the PCA was 23 and the dimension reduction of the LDA was 22

PCA dimension reduction	LDA dimension reduction	separation
13	12	0.82
14	13	0.82
15	13	0.81
15	14	0.82
16	14	0.82
16	15	0.83
17	15	0.81
17	16	0.83
18	17	0.82
19	17	0.81
19	18	0.82
20	18	0.82
20	19	0.84
21	19	0.82
21	20	0.84
22	20	0.83
22	21	0.85
23	20	0.82
23	21	0.84
23	**22**	**0.86**
24	21	0.83
24	22	0.85
24	23	0.85
25	22	0.82
25	23	0.84
25	24	0.84
26	24	0.83
27	24	0.81

Table 2: Table of all the pairs of PCA-LDA dimension reduction which resulted in more than 80% separation, we plotted the bold pair

We did not encounter any 80%+ separations in the range of 28-180 PCA dimension reduction.

6.5 Face recognition – PCA only

We wanted to check whether doing dimension reduction using only PCA could get us to good results.
We extracted the LBP histograms as earlier, converted them to Matlab, ran the PCA in iterations from 1 to 172, built the "same" "not same" graphs per each iteration and checked where the separation rate of the graphs, the results are presented in the following graph:

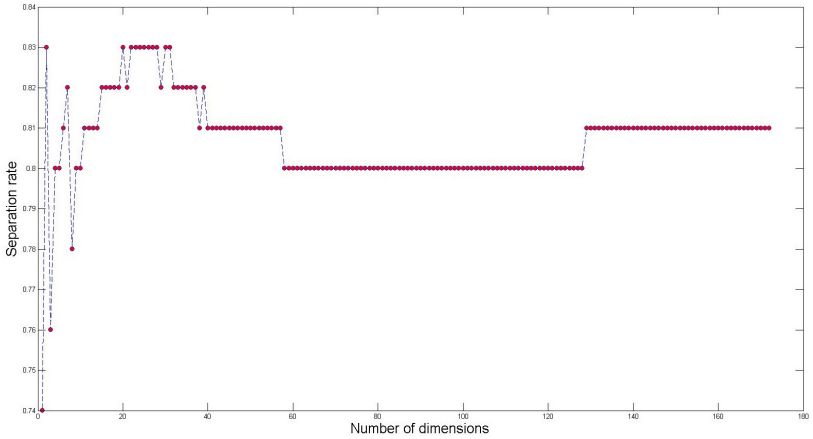


Figure 20: Separation rate as a function of the number of dimensions. The red dots are the results.

As we can see the best separation rate is about 83%.
We will show the graphs of the point 31,0.83 as an example.

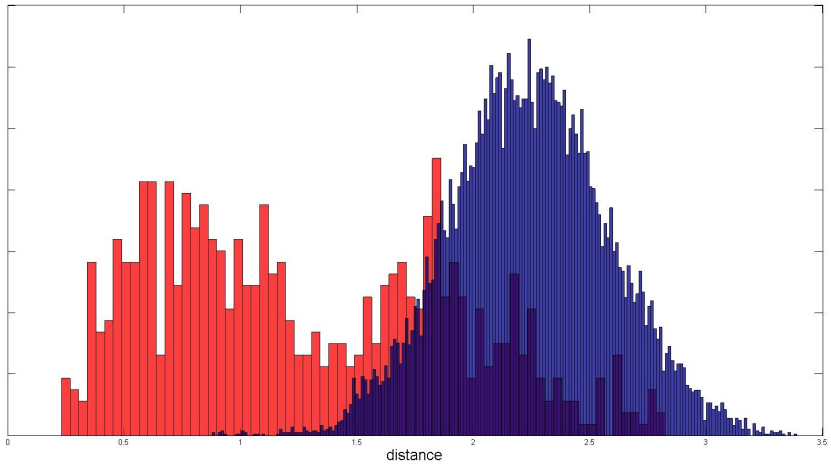


Figure 21: Probability distribution function of the distances between a pair of faces where the dimension reduction of the PCA was 31 and the separation was 83%

6.6 Conclution

We saw that using soley LDA gives us poor results. Using soley PCA gives us much better results, and the the combination of the two gives us the best results, although not far from the results gotten by using soley PCA. All these

conslusions can be seen clearly in the combined ROC graphs:

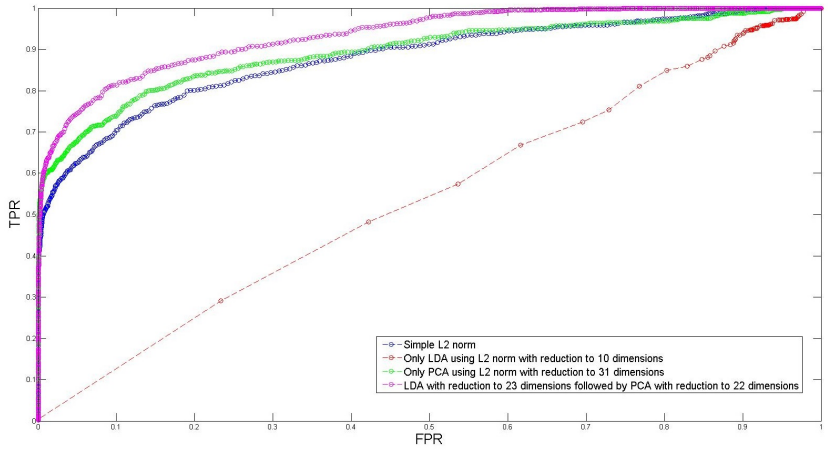


Figure 22: ROC of all the methods we used to do the face recognition in this section

References

[1] `http://en.wikipedia.org/wiki/Linear_discriminant_analysis`.

[2] `http://en.wikipedia.org/wiki/Principal_component_analysis`.

[3] `http://developer.android.com/index.html`.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] `http://opencv.org/`.

[6] `https://developer.qualcomm.com/mobile-development/development-devices/snapdragon-s4-msm8960-mdps`.

[7] `http://java.com/`.

[8] `http://developer.android.com/tools/sdk/ndk/index.html`.

[9] `http://en.wikipedia.org/wiki/Java_Native_Interface`.

[10] `http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle`.

[11] `http://www.cs.york.ac.uk/hise/cadiz/home.html`.

[12] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[13] http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_api.html.

[14] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines, 2002.

[15] Timo Ahonen, Student Member, Abdenour Hadid, Matti Pietikinen, and Senior Member. Face description with local binary patterns: Application to face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:2037–2041, 2006.

[16] http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#warpaffine.

[17] http://en.wikipedia.org/wiki/Receiver_operating_characteristic.

[18] http://opencv.willowgarage.com/documentation/cpp/motion_analysis_and_object_tracking.html#cv-camshift.