# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

The face is our primary focus of attention in social life playing an important role in conveying identity and emotions. We can recognize a number of faces learned throughout our lifespan and identify faces at a glance even after years of separation. This skill is quite robust despite of large variations in visual stimulus due to changing condition, aging and distractions such as beard, glasses or changes in hairstyle. Computational models of face recognition are interesting because they can contribute not only to theoretical knowledge but also to practical applications. Computers that detect and recognize faces could be applied to a wide variety of tasks including criminal identification, security system, image and film processing, identity verification, tagging purposes and human-computer interaction. Unfortunately, developing a computational model of face detection and recognition is quite difficult because faces are complex, multidimensional and meaningful visual stimuli.

Face detection is used in many places now a days especially the websites hosting images like picassa, photobucket and facebook. The automatically tagging feature adds a new dimension to sharing pictures among the people who are in the picture and also gives the idea to other people about who the person is in the image.

Nowadays, the quality of our mobile phone cameras permit us to take high quality pictures. So it is possible to compute many kinds of recognition on those pictures. Face detection is a kind of recognition mostly used in our actual society. We use it all the days on Facebook to tag people on our pictures as mentioned earlier. It is also used in video game, with the kinect concept, or in security, to permit access to private spaces. And it's just few examples of face recognition uses, because in our modern society, face detection will be everywhere.

The main idea of this project was to create a mobile application for able to recognition of humans faces anywhere at any place on the basis of the his photo and profile present on the social media sites like facebook, linkedIn etc. But due to the limited period and less experience the applictaion is not fully developed, it is not yet linked to any of the social sites. The goal of this application is to allow one to retrieve information about any person just through this mobile application only. The reason for developing the mobile application is its flexibility and availability. So we decided to

use the android API to create this, we have used OpenCV library to develop this application.

## 1.2 Applications of the proposed system

The “Face Recognizer App” can have various applications in variety of fields, even can be useful in day-to-day life of the common people. Some applications are following:

- First and the foremost application is to recognize and identify the unknown people.
- The security of the mobile itself, it can be used in place of passwords and pin used to unlock the mobile devices now-a-days, using this app any unauthorized access to the mobile device can be prevented by controlling the access to the device based on face recognition.
- Specific application security by setting it as access medium to any application in mobile phones.
- Can prove as an effective method to find the lost children and person by matching their photos with the suspects which ensures access as well as data security.
- Can be further extended to as the attendance taking system.

There can be many more applications of this system in terms of security, authentication and authorization. Being a mobile application it will be within the reach of every person.

## 1.3 Motivation

The biggest motivation between behind this project is to provide a new dimension to the social media, where currently people search the person based on the name and other information but instead, we now provide them further flexible and easy way to search any unknown person based on his/her photo.

The person can easily get information about any person he suspects without much effort, just have to focus camera on his face without his/her concern and without any external help and extra investigation. With the advent of technology, there is no need for wastage of time and energy.

## 1.4 Problem Statement and Objective

To develop an android application for recognizing people anywhere by using camera through this android application in order to help people to recognize anyone even in the mob without offending security issues, just by extracting information about the person mentioned by him/her on any of the social media site. This type of system will be of great success seeing the increasing dependability of the people on social media. This application can be further extended for making Attendance Taking System and many more applications can be there.

## 1.5 Problem working with mobile phones

The incorporation of face recognition algorithms in to mobile devices has been a challenging problem due to the constraints on processing power, limited storage of the mobile device, limited network bandwidth and connection instability, privacy and security concerns. Hence client-server architecture needs to be developed for this purpose. The client side performs facial detection based on color segmentation, template matching, etc on the captured image and extracts the informational features. These features are then sent to the server which does the computationally intensive task of comparison with the database image set that would help in the face recognition and then sends the information back to the client. The client then displays the required information to the user.

## 1.6 Report Outline

The rest of the report is organized as follows – Chapter 2 contains the basic explanation of the face detection and face recognition. Chapter 3 contains all the algorithms studied by us during the project duration on face detection and recognition. Chapter 4 Background and Literature Overview of the technologies and languages used in our project and a brief explanation of FaceReconizer class. Chapter 5 contains the implementation details and algorithms used. Chapter 6 contains the working details of our application. Chapter 7 contains conclusion and future vision of our application.

# CHAPTER 2

# FACE DETECTION AND RECOGNITION

## 2.1 Face Detection

Face detection can be regarded as a specific case of object-class detection. Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Face detection is used in biometrics, often as a part of (or together with) a facial recognition system. It is also used in video surveillance, human computer interface and image database management.

## 2.2 Face Recognition

Face recognition is one of most relevant applications of image analysis. Face Recognition is a task of identifying an already detected face and telling exactly who's it is and also deals with unique facial characteristics of human beings. It can be applied in various challenging fields like video retrieval, security systems and identity authentication. It involves the pattern recognition and image processing. There are mainly two types of comparisons which are described as follows:

**Verification:** When the system compares the given individual with the individual whom he says he is and gives a yes or no decision.

**Identification:** When the system compares the given individual to all the other individuals stored in the database and gives a ranked list of matches.

Facial recognition methods involve a series of steps that are capturing, analyzing and comparing your face to a database of stored images. Below is the basic process that is used by the face recognition system to capture and compare images:

**Detection:** The recognition software searches the faces using the video camera when the system is attached to a video surveillance system. And if there is a face in the view, it is detected within a fraction of a second.

**Alignment:** Once the system has detected the face, then it determines the head's position, size and pose. For the system to register the face it needs to be turned at least 35 degrees towards the camera.

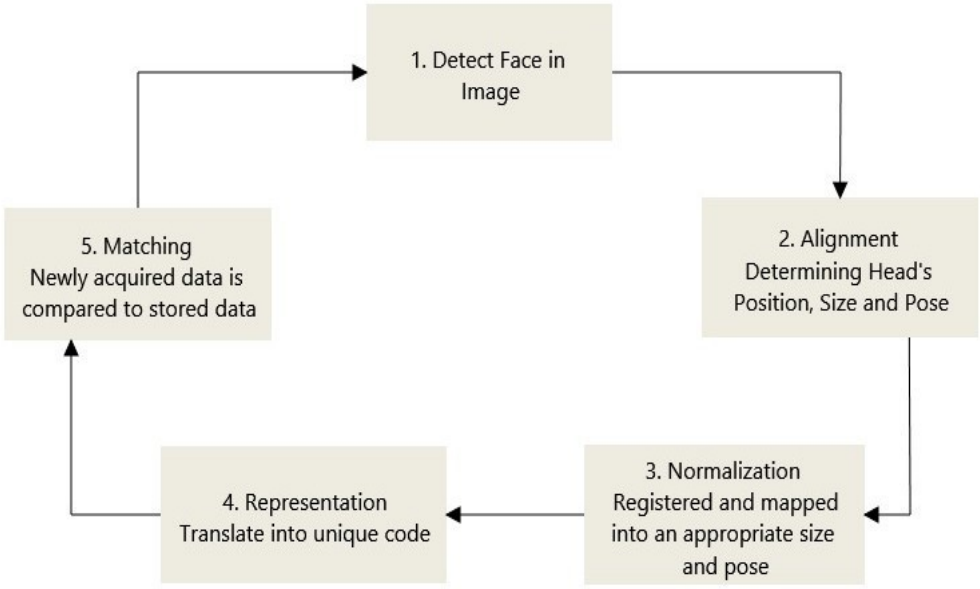


*Figure 2.1*

**Normalization:** For the image of the head to be registered and mapped into an appropriate size and pose, it is scaled and rotated. Normalization is performed despite of the head's location and distance from the camera. Light is not an issue in the normalization process.

**Representation:** After the normalization has been done, the system converts the facial data into a unique code. This coding process allows for easier representation and comparison of the newly acquired facial data to facial data which is already stored.

.

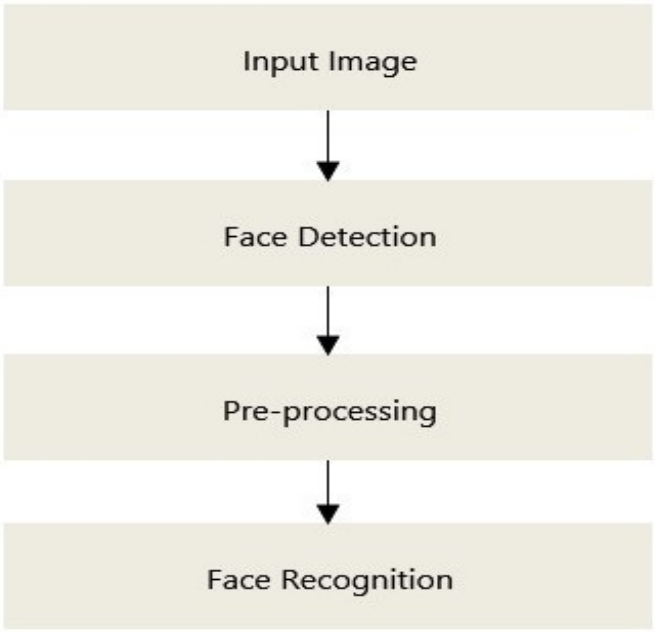


*Figure 2.2*

**Matching:** The newly acquired facial data is compared to the stored data and linked to at least one stored facial representation. The system decides if the features extracted from the newly acquired facial data are a match or not. If a score is above a predetermined threshold, a match is declared.

# CHAPTER 3

# FACE DETECTION AND RECOGNITION ALGORITHM

## 3.1 Face Detection Algorithm

Some of the common approaches used for face detection are Haar-cascades and scanning images using an increasing window, skin color segmentation, template matching and morphological processing algorithms. An impressive achievement in face detection would be efficient face detection at frame rate.

**3.1.1 Haar-cascades algorithm:** Haar-cascades algorithm is also referred to as Viola-Jones face detector and is very effective due to it's robust real-time processing capabilities. Firstly let us discuss Haar-like features, they are the digital image features that are mainly used in object recognition. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.
Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here only face detection is under consideration. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. After that we need to extract features from it. For this, haar features shown in figure 3.1 are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

Now all possible sizes and locations of each kernel is used to calculate plenty of features. It needs lot of computation, even a 24x24 window results over 160000 features. For each feature calculation, we need to find sum of pixels under white and black rectangles. But to reduce the computation they introduced the concept of integral images. Integral images can be defined as two-dimensional lookup tables in the form of a matrix with the same size of the original image. Each element of the integral image contains the sum of all pixels located on the up-left region of the

original image. It simplifies calculation of sum of pixels, how large may be the number of pixels, to an operation involving just four pixels.

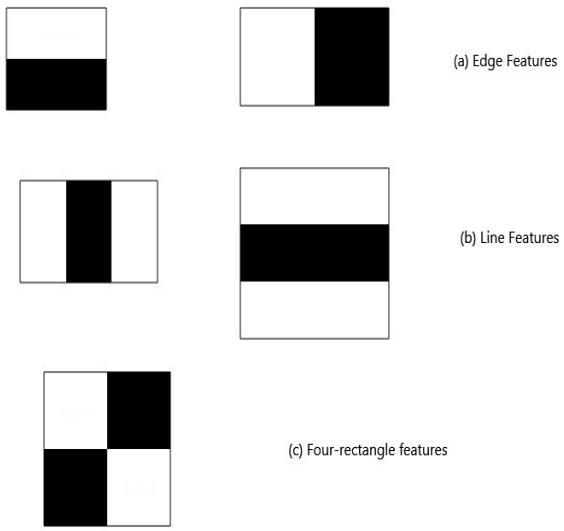


*Figure 3.1*

Most of them are irrelevant among the calculated features. For example, consider the figure 3.2. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. The selection of the best features out of 160000+ features is achieved by Adaboost.

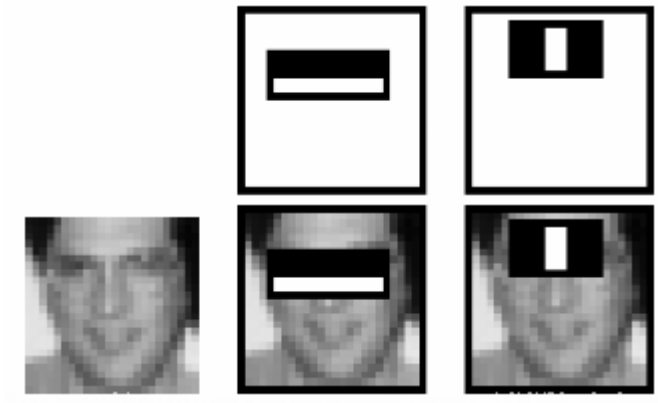

*Figure 3.2*

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But

obviously, there will be errors or misclassifications. We select the features with minimum error rate, they are the features that best classify the face and non-face images. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found.

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. Even 200 features provide detection with 95% accuracy. The final setup had around 6000 features.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Instead focus on region where there can be a face. This way, we can get more time to check a possible face region.

For this the concept of Cascade of Classifiers is introduced. Instead of applying all the 6000 features on a window, group the features into different stages of classifiers and apply one-by-one. Normally first few stages will contain very less number of features. If a window fails the first stage, discard it. We don't consider remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

**Haar-cascade Detection in OpenCV**

OpenCV comes with a trainer as well as detector. If you want to train your own classifier for any object like car, planes etc. you can use OpenCV to create one. Its full details are given at Cascade Classifier Training.

Here we will deal with detection. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in opencv/data /haarcascades/ folder. Let's create face and eye detector with OpenCV.

First we need to load the required XML classifiers. Then load our input image (or video) in grayscale mode.

```
import numpy as np
```

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Now we find the faces in the image. If faces are found, it returns the positions of detected faces as Rect(x,y,w,h). Once we get these locations, we can create a ROI for the face and apply eye detection on this ROI (since eyes are always on the face ).

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
 cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
 roi_gray = gray[y:y+h, x:x+w]
 roi_color = img[y:y+h, x:x+w]
 eyes = eye_cascade.detectMultiScale(roi_gray)
 for (ex,ey,ew,eh) in eyes:
 cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

**3.1.2 Skin color segmentation:** In isolating skin color region normalized RGB, HIS and YCrCb color space transformations are commonly used. Based on the color component values, each pixel is classified as skin or non-skin. The number of pixels used to represent the face in mobile phone photos dominates the pixels used to represent other part of the body. Therefore these approaches are likely to give good results on handsets. To reduce the effects of illumination variations scale-by-max color balancing is done. Many complex methods have been used in the literature to do the processing of images in extreme illumination conditions. One of the simplest methods to implement on a mobile phone could be do the calculations in RGB space.

A pixel is classified as skin if its (R, G, B) values.

**3.1.3 Morphological Image Processing:-** This process helps to regroup the skin pixels by eliminating the non-skin visible pixels. Sparse non-skin pixels can be removed by performing erosion. Smoothing of the contours and regrouping the skin regions is attained by doing dilation.

**3.1.4 Template Matching:-** This approach is for finding an object represented by the template in a given image. Using FFT and IFFT, cross correlation can be implemented in the frequency domain as it is computationally more efficient. Color segmentation and morphological processing will ideally leave only the face portion. But template matching will be helpful in cases where skin is exposed like hands, neck, etc.

## 3.2 Face Recognition Algorithm

**3.2.1 PCA (Principal Component Analysis):-** Commonly uses the eigenfaces in which the probe and gallery images must be the same size as well as normalized to line up the eyes and mouth of the subjects whining the images. Approach is then used to reduce the dimension of data by the means of image compression basics and provides most effective low dimensional structure of facial pattern. This reduction drops the unuseful information and decomposes the face structure into orthogonal (uncorrelated) components known as eigenfaces. Each face image is represented as weighted sum feature vector of eigenfaces which are stored in 1-D array. A probe image is compared against the gallery image by measuring the distance between their respective feature vectors then matching result has been disclosed. The main advantage of this technique is that it can reduce the data needed to identify the individual to 1/1000 of the data presented.

PCA solves the recognition problem within a representation space of lower dimension than image space. PCA is an eigenface method which helps in the reduction of the dimensionality of the original data space. But there is a disadvantage of PCA which says that recognition rate decreases under varying pose and illumination. A face recognition system can be considered as a good system if we extract with the help of Principal Component Analysis and for recognition back propagation Neural Network are used.

**3.2.2 LDA (Linear Discriminant Analysis):-** LDA is an appearance based technique used for dimensionality reduction and recorded a great performance in face recognition. It provides us with a small set of features that carry the most relevant information for classification purposes. LDA is a statistical approach for classifying samples of unknown classes based on training samples with known classes. This technique aims to maximum between class (across users) variance and minimum within class (within
user) variance. In these techniques a block represents a class, and there are a large variations between blocks but little variations within classes. It searches for those vectors in underlying space that best discriminate among classes (rather than those that best describe the data). More formally given a number of independent features relative to which the data is described. LDA creates a linear combination of these which yields the largest mean difference between desire classes.

**3.2.3 SVM (Support Vector Machine):** SVM is a binary classifier as a method for learning. SVM is a classification method that separate two data sets with maximum distance between them. The concept is to extend the spatial resolution around the margin by a conformal mapping, such that the divisibility between classes is increased. SVM cannot be applied directly when some of the features (face pixels) are occluded. In this case, values for those dimensions are unknown. SVM cannot be used when the feature vectors defining our samples have missing entries. Support vector machines (SVMs) are formulated to solve a classical two class pattern recognition problem.

**3.2.4 SIFT(Scale Invariant Feature Transforms):** SIFT was invented by Lowe. The SIFT approach, for image feature generation, takes an image and transforms it into a large collection of local feature vectors. SIFT descriptor which is invariant to scale, rotation, affine transformation, noise, occlusions and is highly distinctive. SIFT features consist of four major stages in detection and representation; they are:

- finding scale-space extrema;
- key point localization and filtering;
- orientation assignment;
- key point descriptor.

In the first stage the key points of images are constructed by using Difference-of-Gaussian (DoG) function. In second stage, candidate key points are restricted to sub-pixel accuracy and removed if found to be unreliable. The third stage represents the dominant orientations for each essential point of the images. The final stage constructs a descriptor for each key point location depends upon the image gradients in its local neighborhood. Then the SIFT descriptor is accepting the 128-dimensional vector which used to identify the neighborhood around a pixel. The SIFT extracts the key points (locations and descriptors) for all the database images. Then given an altered image SIFT extracts the key point for that image and compares that point to the dataset.

**3.5 SURF (Speeded Up Robust Features):** The SURF is local feature detector and descriptor. It also extracts the key points from both the database images and the altered images. This method matches the key points between altered image and each database image. H. Bay invents SURF descriptor which is invariant to a scale and in-plane rotation features. It consists of two stages such as interest point detector and interest point descriptor. In the first stage, locate the interest point in the image. Use the Hessian matrix to find the approximate detection.

It can be used for the task such as object recognition, image registration, classification or 3D construction. SURF detectors find the interest points in an image, and descriptors are used to extract the feature vectors at each interest point just as in SIFT. SURF uses Hessian-matrix approximation to locate the interest points instead of Difference of Gaussians (DoG) filter used in SIFT. SURF as a descriptor uses the first-order Haar wavelet responses in x and y, whereas the gradient is used by SIFT. SURF usually uses 64 dimensions in SURF to reduce the time cost for both feature matching and computation. SURF has three times better performance as compared to SIFT.

# CHAPTER 4
# BACKGROUND AND LITERATURE OVERVIEW

## 4.1 Introduction to the programming languages

### 4.1.1 The JAVA programming language

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. The Java language was created by James Gosling and Patrick Naughton, Sun Microsystems employees in 1995. This is the language we have use in our project. Java can work on all the operating system. Its motto: ”write once, run anywhere”. Java is one of the most popular programming language during the ten last years. All the Java components are objects that interact with each other. Those objects are represented by classes. All classes have their own attributes and methods.

The keyword like “private” doesn’t allow the other classes to use “class attribute”. It is a way to hide important attributes or methods. Indeed, with this key-word, the other classes can’t use those. Sometimes it is necessary to improve the security of a class. Normally, all attributes have to be private, and programmers must add “getters” methods to the class to catch them. We have to pass by the method “class methods” to catch “class attribute”. It is the principle of encapsulation. The “public” keyword allows all the other classes to use methods/attributes.

The last important concept of Java is the principle of inheritance. It is possible to reuse the old Java code thanks to the keyword “extends” during the declaration of the class. The keyword “protected” can be used also to declare functions or attributes. This keyword allows access only to the children classes. Only the public/protected attributes and methods can be inherited. Thanks to inheritance, developers can always reuse their codes.

### 4.1.2 XML Language

The XML (Extensible Markup Language) is a computer markup language. It permits to organize information in a way that allow both computer and human to understand. XML has been declined in a lot of formats, like XHTML, used for the internet. The information are organized as a tree form. That means a XML file contain a root

markup who include all the other markups of the file. Thanks to this, there is a real hierarchy of information.

## 4.2 Introduction to the work environment

### 4.2.1 Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

**Project Structure**

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view. This view is organized by modules to provide quick access to your project's key source files.All the build files are visible at the top level under Gradle Scripts and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown.

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

The Android Studio main window is made up of several logical areas.

- The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
- The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
- The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
- The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
- The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
- The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This

can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

**Tool Windows**

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.
- To show or hide the entire tool window bar, click the window icon in the bottom left-hand corner of the Android Studio window.
- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

You can also use keyboard shortcuts to open tool windows.

If you want to hide all toolbars, tool windows, and editor tabs, click **View > Enter Distraction Free Mode**. This enables Distraction Free Mode. To exit Distraction Free Mode, click **View > Exit Distraction Free Mode**.

You can use Speed Search to search and filter within most tool windows in Android Studio. To use Speed Search, select the tool window and then type your search query.

**Version Control Basics**

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.

After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations:

- From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
- From the drop-down menu, select a version control system to associate with the project root, and then click **OK**.

The VCS menu now displays a number of version control options based on the system you selected.

**Note:** You can also use the **File > Settings > Version Control** menu option to set up and modify the version control settings.

### Gradle Build System

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named build.gradle. They are plain text files that use Groovy syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

### Build Variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

### APK Splits

APK splits allow you to efficiently create multiple APKs based on screen density or ABI. For example, APK splits allow you to create separate hdpi and mdpi versions of an app while still considering them a single variant and allowing them to share a test app, javac, dx, and ProGuard settings.

**Managing Dependencies**

Dependencies for your project are specified by name in the build.gradle file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your build.gradle file. Android Studio configures projects to use the Maven Central Repository by default. (This configuration is included in the top-level build file for the project.)

"Face Recognizer App" supports Android 4.0 (IceCream Sandwich) and higher and it is not backward compatible with 2.3.x (Gingerbread), as 2.3.x version of Android is outdated and isn't being currently used anywhere.

### 4.2.2 Android SDK

The Android SDK Manager separates the SDK tools, platforms, and other components into packages for easy access and management. We used API 18 for our app's development environment. It is particularly useful for simplifying the steps of retrieving the Build Tools, SDK version, or support libraries used in your project.
We have used the OpenCV library in our project.

## 4.3 The 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.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track

moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million. The library is used extensively in companies, research groups and by governmental bodies.

OpenCV 2.4 now comes with the very new FaceRecognizer class for face recognition, so you can start experimenting with face recognition right away.

The currently available algorithms are:

- Eigenfaces ( createEigenFaceRecognizer())
- Fisherfaces (see createFisherFaceRecognizer())
- Local Binary Patterns Histograms (see createLBPHFaceRecognizer())

### 4.3.1 FaceRecognizer Class

All face recognition models in OpenCV are derived from the abstract base class FaceRecognizer, which provides a unified access to all face recognition algorithms in OpenCV.

Every FaceRecognizer is an Algorithm, so you can easily get/set all model internals (if allowed by the implementation). Algorithm is a relatively new OpenCV concept, which is available since the 2.4 release.

Algorithm provides the following features for all derived classes:

- So called “virtual constructor”. That is, each Algorithm derivative is registered at program start and you can get the list of registered algorithms and create instance of a particular algorithm by its name (Algorithm::create()). If you plan to add your own algorithms, it is good practice to add a unique prefix to your algorithms to distinguish them from other algorithms.
- Setting/Retrieving algorithm parameters by name. If you used video capturing functionality from OpenCV high gui module, you are probably familar with cvSetCaptureProperty(), cvGetCaptureProperty(), VideoCapture::set() and VideoCapture::get(). Algorithm provides similar method where instead of

integer id's you specify the parameter names as text strings. See Algorithm::set() and Algorithm::get() for details.

- Reading and writing parameters from/to XML or YAML files. Every Algorithm derivative can store all its parameters and then read them back. There is no need to re-implement it each time.

Moreover every FaceRecognizer supports the:

- **Training** of a FaceRecognizer with FaceRecognizer::train() on a given set of images (your face database).
- **Prediction** of a given sample image, that means a face. The image is given as a Mat.
- **Loading/Saving** the model state from/to a given XML or YAML.
- **Setting/Getting labels info**, that is stored as a string. String labels info is useful for keeping names of the recognized people.

**Setting the Thresholds**

Sometimes you run into the situation, when you want to apply a threshold on the prediction. A common scenario in face recognition is to tell, whether a face belongs to the training dataset or if it is unknown. You might wonder, why there's no public API in FaceRecognizer to set the threshold for the prediction, but rest assured: It's supported. It just means there's no generic way in an abstract class to provide an interface for setting/getting the thresholds of every possible FaceRecognizer algorithm. The appropriate place to set the thresholds is in the constructor of the specific FaceRecognizer and since every FaceRecognizer is a Algorithm (see above), you can get/set the thresholds at runtime.

Sometimes it's impossible to train the model, just to experiment with threshold values. Thanks to Algorithm it's possible to set internal model thresholds during runtime. If you've set the threshold to 0.0 then is going to yield -1 as predicted label, which states this face is unknown.

**Getting the name of the FaceRecognizer**

Since every FaceRecognizer is a Algorithm, you can use Algorithm::name() to get the name of a FaceRecognizer.

```
Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
```

```
std::string name = model->name();
```

**FaceRecognizer::train**

Trains a FaceRecognizer with given data and associated labels.

C++: void FaceRecognizer::train(InputArrayOfArrays **src**, InputArray **labels**) = 0

Parameters :

- **src:** The training images, that means the faces you want to learn. The data has to be given as a vector<Mat>.
- **Labels:** The labels corresponding to the images have to be given as a vector<int>.

The images are read with imread() and pushed into a std::vector<Mat>. The labels of each image are stored within a std::vector<int> (you could also use a Mat of type CV_32SC1). Think of the label as the subject (the person) this image belongs to, so same subjects (persons) should have the same label. For the available FaceRecognizer you don't have to pay any attention to the order of the labels, just make sure same persons have the same label:

Now that you have read some images, we can create a new FaceRecognizer. In this example I'll create a Fisherfaces model and decide to keep all of the possible Fisherfaces:

```
Ptr<FaceRecognizer> model =  createFisherFaceRecognizer();
```

And finally train it on the given dataset (the face images and labels):

```
model->train(images, labels);
```

**FaceRecognizer::update**

Updates a FaceRecognizer with given data and associated labels.

C++: void FaceRecognizer::update(InputArrayOfArrays src, InputArray labels)

Parameters:

- **src :** The training images, that means the faces you want to learn. The data has to be given as a vector<Mat>.
- **Labels:** The labels corresponding to the images have to be given as a vector<int> .

This method updates a (probably trained) FaceRecognizer, but only if the algorithm supports it. The Local Binary Patterns Histograms (LBPH) recognizer (see createLBPHFaceRecognizer()) can be updated. For the Eigenfaces and Fisherfaces method, this is algorithmically not possible and you have to re-estimate the model with FaceRecognizer::train(). In any case, a call to train empties the existing model and learns a new model, while update does not delete any model data.

Calling update on an Eigenfaces model (see createEigenFaceRecognizer()), which doesn't support updating, will throw an error.

Note: The FaceRecognizer does not store your training images, because this would be very memory intense and it's not the responsibility of the FaceRecognizer to do so. The caller is responsible for maintaining the dataset, he want to work with.

**FaceRecognizer::predict**

C++: int FaceRecognizer::predict(InputArray src) const = 0 .
C++: void FaceRecognizer::predict(InputArray src, int& label, double& confidence) const = 0 .

Predicts a label and associated confidence (e.g. distance) for a given input image.

Parameters:

- **src:** Sample image to get a prediction from.
- **Label:** The predicted label for the given image.
- **Confidence:** Associated confidence (e.g. distance) for the predicted label.

The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.

The following example shows how to get a prediction from a trained model:

```
Mat img = imread("person1/3.jpg", CV_LOAD_IMAGE_GRAYSCALE);
int predicted = model->predict(img);
```

Or to get a prediction and the associated confidence (e.g. distance):

```
Mat img = imread("person1/3.jpg", CV_LOAD_IMAGE_GRAYSCALE);
int predicted_label = -1;
double predicted_confidence = 0.0;

model->predict(img, predicted_label, predicted_confidence);
```

**FaceRecognizer::save**

Saves a FaceRecognizer and its model state.

C++: void FaceRecognizer::save(const string& **filename**) const

Saves this model to a given filename, either as XML or YAML.

Parameters:

- **filename:** The filename to store this FaceRecognizer to (either XML/YAML).

C++: void FaceRecognizer::save(FileStorage& **fs**) const

Saves this model to a given FileStorage.

Parameters:

- **fs:** The FileStorage to store this FaceRecognizer to.

Every FaceRecognizer overwrites FaceRecognizer::save(FileStorage& fs) to save the internal model state. FaceRecognizer::save(const string& filename) saves the state of a model to the given filename.

The suffix const means that prediction does not affect the internal model state, so the method can be safely called from within different threads.

**FaceRecognizer::load**

Loads a FaceRecognizer and its model state.

C++: void FaceRecognizer::load(const string& filename)

C++: void FaceRecognizer::load(const FileStorage& fs) = 0

Loads a persisted model and state from a given XML or YAML file . Every FaceRecognizer has to overwrite FaceRecognizer::load(FileStorage& fs) to enable loading the model state. FaceRecognizer::load(FileStorage& fs) in turn gets called by FaceRecognizer::load(const string& filename), to ease saving a model.

**FaceRecognizer::setLabelsInfo**

Sets string information about labels into the model. .. ocv:function:: void FaceRecognizer::setLabelsInfo(const std::map<int, string>& labelsInfo)

Information about the label loads as a pair “label id - string info”.

**FaceRecognizer::getLabelInfo**

Gets string information by label. .. ocv:function:: string FaceRecognizer::getLabelInfo(const int &label)

If an unknown label id is provided or there is no label information assosiated with the specified label id the method returns an empty string.

**FaceRecognizer::getLabelsByString**

Gets vector of labels by string.

C++: vector<int> FaceRecognizer::getLabelsByString(const string& str)

The function searches for the labels containing the specified substring in the associated string info.

## 4.4 createEigenFaceRecognizer

C++: Ptr<FaceRecognizer> createEigenFaceRecognizer(int num_components=0, double threshold=DBL_MAX)

Parameters:

- **num_components:** The number of components (read: Eigenfaces) kept for this Prinicpal Component Analysis. As a hint: There’s no rule how many components (read: Eigenfaces) should be kept for good reconstruction capabilities. It is based on your input data, so experiment with the number. Keeping 80 components should almost always be sufficient.
- **threshold:** The threshold applied in the prediciton.

Training and prediction must be done on grayscale images, use cvtColor() to convert between the color spaces.

The eigenfaces method makes the assumption, that the training and test images are of equal size. You have to make sure your input data has the correct shape, else a meaningful exception is thrown. Use resize() to resize the images.
This model does not support updating.

Model internal data:

- num_components,see createEigenFaceRecognizer().
- Threshold, see createEigenFaceRecognizer().
- Eigenvalues, The eigenvalues for this Principal Component Analysis (ordered descending).
- Eigenvectors, The eigenvectors for this Principal Component Analysis (ordered by their eigenvalue).
- Mean, the sample mean calculated from the training data.
- Projections, the projections of the training data.
- Labels, the threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.

## 4.5 createFisherFaceRecognizer

C++: Ptr<FaceRecognizer> createFisherFaceRecognizer(int num_components=0, double threshold=DBL_MAX)
Parameters:

- **num_components:** The number of components (read: Fisherfaces) kept for this Linear Discriminant Analysis with the Fisherfaces criterion. It's useful to keep all components, that means the number of your classes c (read: subjects, persons you want to recognize). If you leave this at the default (0) or set it to a value less-equal 0 or greater (c-1), it will be set to the correct number (c-1) automatically.
- **threshold:** The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.

Training and prediction must be done on grayscale images, use cvtColor() to convert between the color spaces.

- The fisherfaces method makes the assumption, that the training and test images are of equal size. You have to make sure your input data has the correct shape, else a meaningful exception is thrown. Use resize() to resize the images.
- This model does not support updating.

Model internal data

- num_components, see createFisherFaceRecognizer().
- threshold, see createFisherFaceRecognizer().
- eigenvalues, the eigenvalues for this Linear Discriminant Analysis (ordered descending).
- eigenvectors, the eigenvectors for this Linear Discriminant Analysis (ordered by their eigenvalue).
- mean, the sample mean calculated from the training data.
- projections, the projections of the training data.
- labels , the labels corresponding to the projections.

## 4.6 createLBPHFaceRecognizer

C++: Ptr<FaceRecognizer> createLBPHFaceRecognizer(int radius=1, int neighbors=8, int grid_x=8, int grid_y=8, double threshold=DBL_MAX)

Parameters:

- **radius:** The radius used for building the Circular Local Binary Pattern. The greater the radius, the
- **neighbors:** The number of sample points to build a Circular Local Binary Pattern from. An appropriate value is to use `` 8`` sample points. Keep in mind: the more sample points you include, the higher the computational cost.
- **grid_x:** The number of cells in the horizontal direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.

- **grid_y:** The number of cells in the vertical direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.
- **threshold:** The threshold applied in the prediction. If the distance to the nearest neighbor is larger than the threshold, this method returns -1.

The Circular Local Binary Patterns (used in training and prediction) expect the data given as grayscale images, use cvtColor() to convert between the color spaces.

- This model supports updating.

Model internal data:

- radius, see createLBPHFaceRecognizer().
- neighbors, see createLBPHFaceRecognizer().
- grid_x, see createLBPHFaceRecognizer().
- grid_y, see createLBPHFaceRecognizer().
- threshold, see createLBPHFaceRecognizer().
- histograms Local Binary Patterns Histograms calculated from the given training data (empty if none was given).
- labels, labels corresponding to the calculated Local Binary Patterns Histograms.

# CHAPTER 5

# IMPLEMENTATION

## 5.1 Haar Feature based cascade classifier

The object detector described below has been initially proposed by Paul Viola [Viola01] and improved by Rainer Lienhart [Lienhart02].

First, a classifier (namely a cascade of boosted classifiers working with haar-like features) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below. The current algorithm uses the Haar-like features as shown in *Figure 5.1*.

The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the

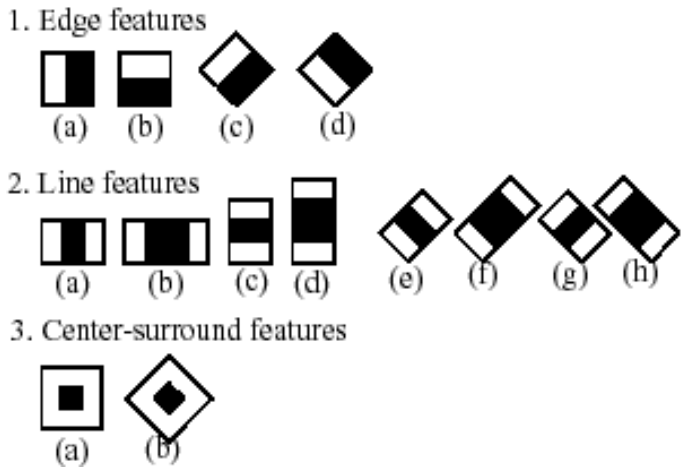


*Figure 5.1*

difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular regions are calculated rapidly using integral images.

## 5.2 AT & T

The AT&T Facedatabase, sometimes also referred to as ORL Database of Faces, contains ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

## 5.3 EigenFaces

We can use the jet colormap, so we can see how the grayscale values are distributed within the specific Eigenfaces. We can see, that the Eigenfaces do not only encode facial features, but also the illumination in the images.

10 Eigenvectors are not sufficient for a good image reconstruction, 50 Eigenvectors may already be sufficient to encode important facial features. You'll get a good reconstruction with approximately 300 Eigenvectors for the AT&T Facedatabase.

There are rule of thumbs how many Eigenfaces you should choose for a successful face recognition, but it heavily depends on the input data.

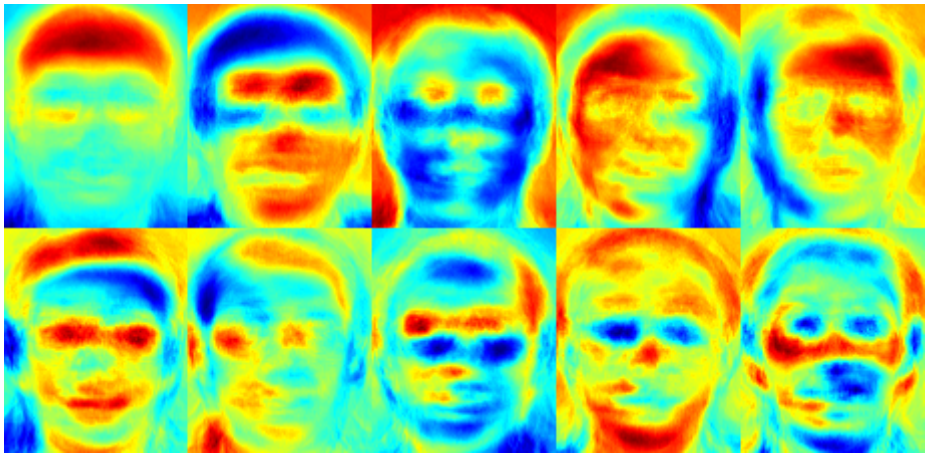

*Figure 5.2*

## 5.4 FisherFaces In OpenCV

The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information may be lost when throwing components away.

The Fisherfaces method learns a class-specific transformation matrix, so the they do not capture illumination as obviously as the Eigenfaces method. The Discriminant Analysis instead finds the facial features to discriminate between the persons. It's important to mention, that the performance of the Fisherfaces heavily depends on the input data as well. Practically said: if you learn the Fisherfaces for well-illuminated pictures only and you try to recognize faces in bad-illuminated scenes, then method is likely to find the wrong components (just because those features may not be predominant on bad illuminated images). This is somewhat logical, since the method had no chance to learn the illumination.

The Fisherfaces allow a reconstruction of the projected image, just like the Eigenfaces did. But since we only identified the features to distinguish between subjects, we can't expect a nice reconstruction of the original image. For the Fisherfaces method we'll project the sample image onto each of the Fisherfaces instead. So we'll have a nice visualization, which feature each of the Fisherfaces describes:

```
for(int num_component = 0; num_component < min(16, W.cols);
        num_component++) {
```

```
// Slice the Fisherface from the model:
Mat ev = W.col(num_component);
Mat projection = subspaceProject(ev, mean, images[0].reshape(1,1));
Mat reconstruction = subspaceReconstruct(ev, mean, projection);

// Normalize the result:
reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));

// Display or save:
if(argc == 2) {
imshow(format("fisherface_reconstruction_%d", num_component),
        reconstruction);
 } else {
  imwrite(format("%s/fisherface_reconstruction_%d.png",
   output_folder.c_str(), num_component), reconstruction);
  }
 }
```

The differences may be subtle for the human eyes, but we should be able to see some differences:

*Figure 5.3*

## 5.5 Integration of OpenCV with Android Studio

To use OpenCV we first need to integrate OpenCV with our android application. First you need to create a new project with the specific name. While creating the new project you should specific the minimum android version which your application will

support. For our application “4.4-4.4.4 KitKat” is the minimum android version required.

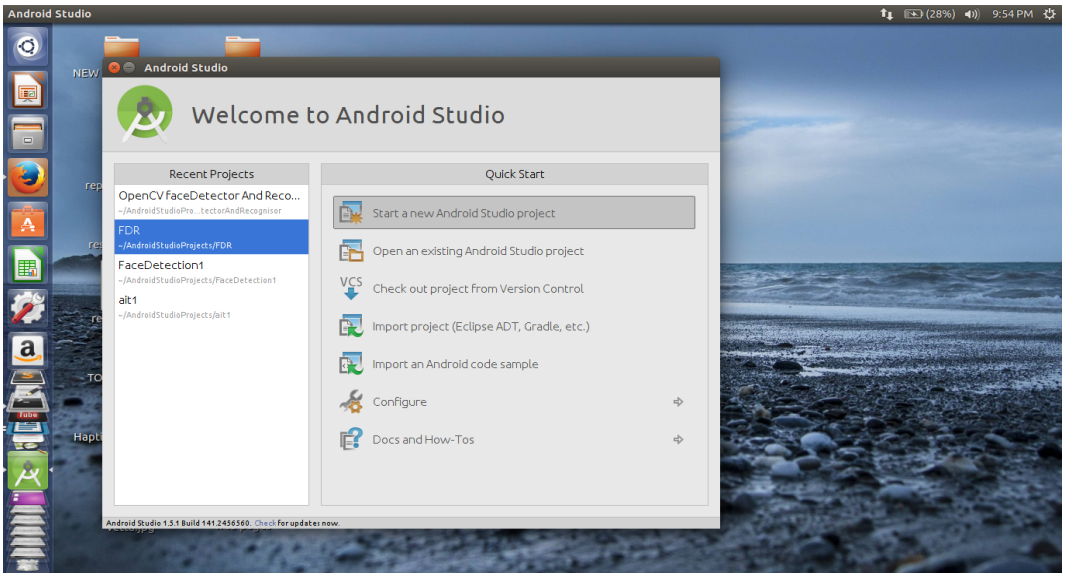


*Figure 5.4*

Once the project is successfully created, now the procedure for integrating the module starts. Follow the below procedure:-

**File > New > Import Module**

The above procedure will direct you to the screen shown in *Figure 5.5*. further steps are shown on *Figure 5.6,Figure 5.7* . From there select the “Java” module from the desired folder i.e. where OpenCV is stored as shown in *Figure 5.6*. After the above procedure if you encounter any error that means your SDK and OpenCV versions are different. The version of “compileSdkVersion” and “buildToolsVersion” must be same as that of the SDK.

To remove this error just modify their values in **openCVLibrary > build.gradle**

In our application the values of these two parameters are following:

**“compileSdkVersion” 23**

**“buildToolsVersion” 23.0.2**

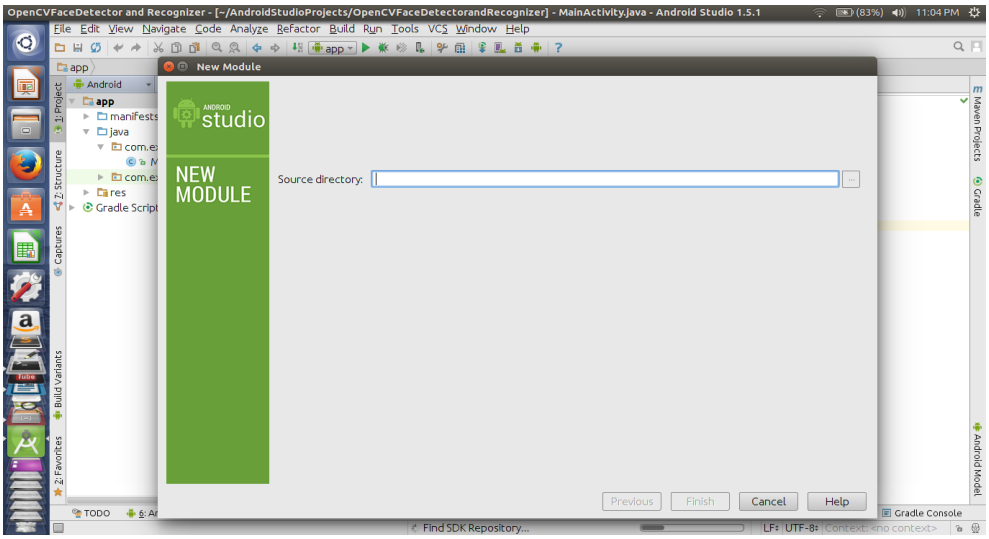


*Figure 5.5*

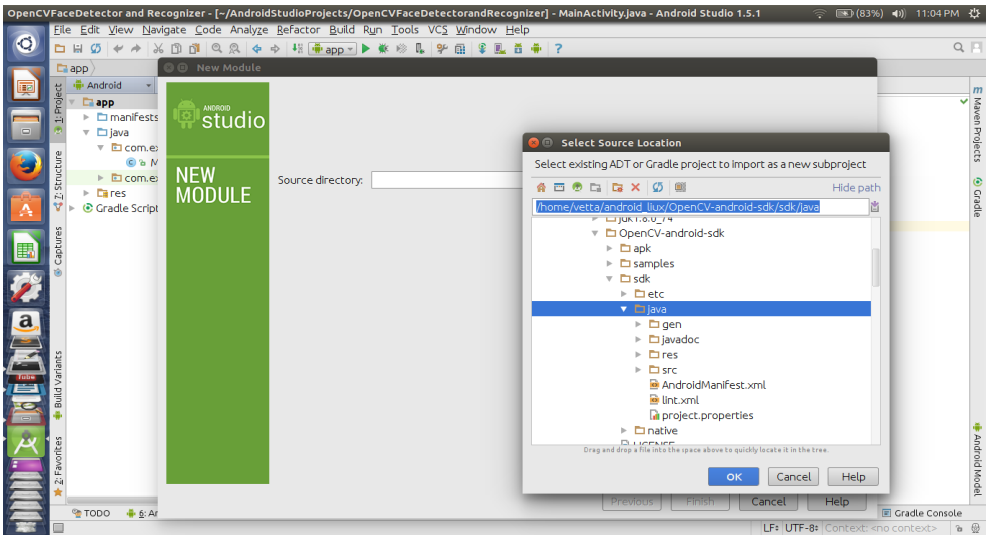


*Figure 5.6*

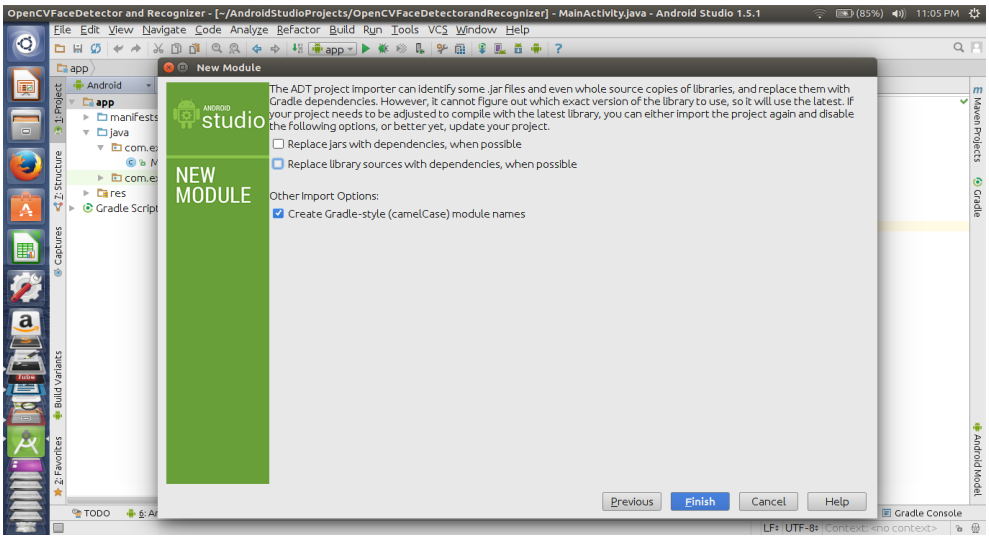


*Figure 5.7*

After changing their values rebuild the project again. Now there will be no errors. Next step include adding dependencies. To add dependencies follow:

**File > Project Structure > Module > App > Dependencies**

Further step are shown in *Figure 5.8* and *Figure 5.9*

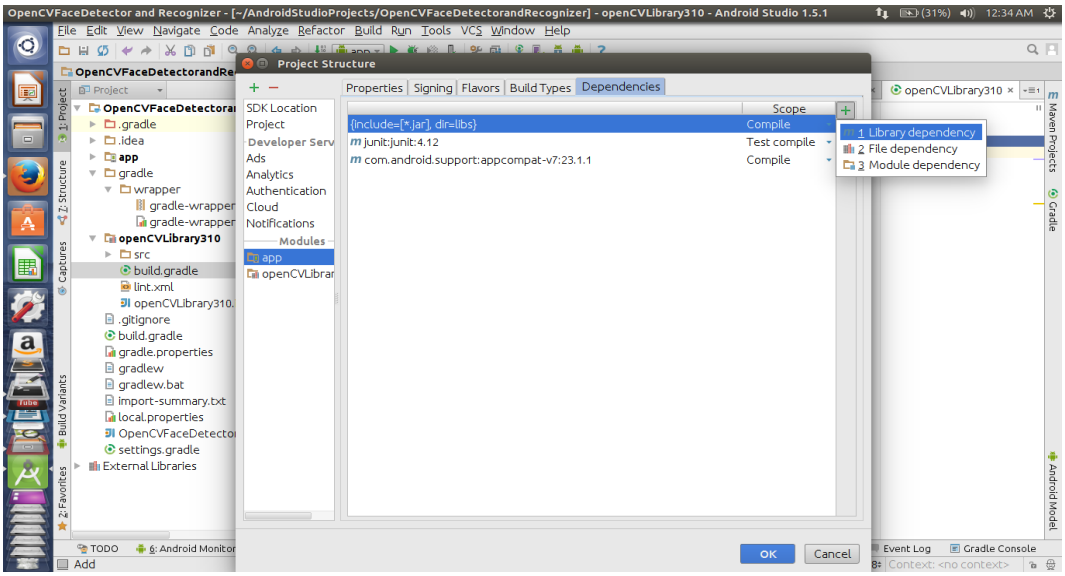


*Figure 5.8*

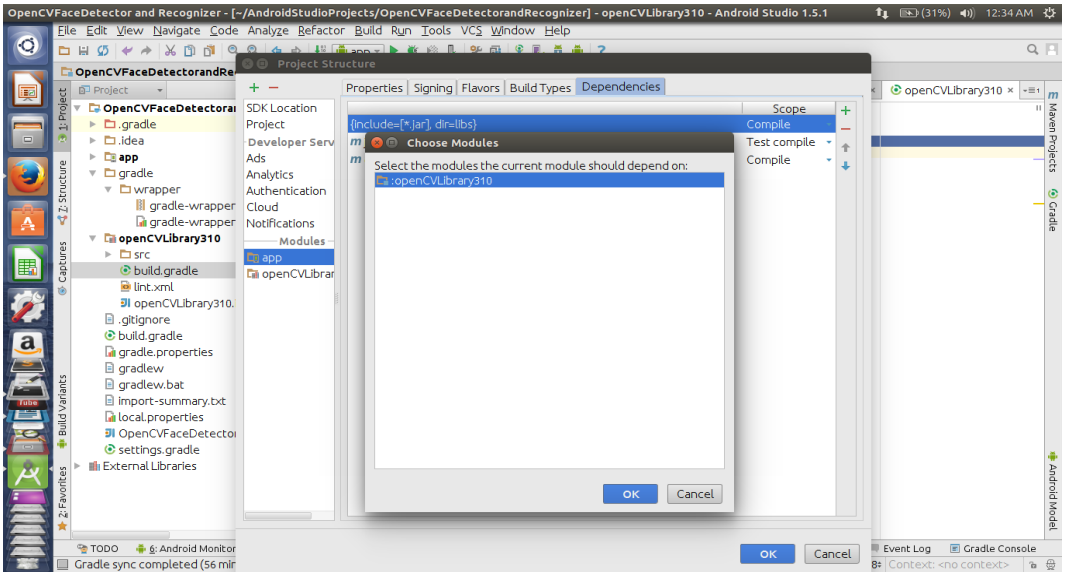


*Figure 5.9*

After successfully adding the dependencies, now copy the "libs" folder from

**OpenCVandroidsdk310 > native > libs**

to the project's main directory **app > src > main** and paste it in main folder and rename libs as JniLibs.Rebuild the project and you are ready to use OpenCV in android.

# CHAPTER 6

# WORKING DETAILS

Basic functionality of our application is that it recognizes the face of an individual and stores it photos for training and recognizing purpose. The graphical user interface of our application provides four buttons. These four buttons are:

- Scan
- Rec
- Train
- Identify

The user first need to add the photos of an individual which is accomplished by Scan button and then click on Rec button to save the features of the face. It is preferable to take atleast 10 photos for the better training.

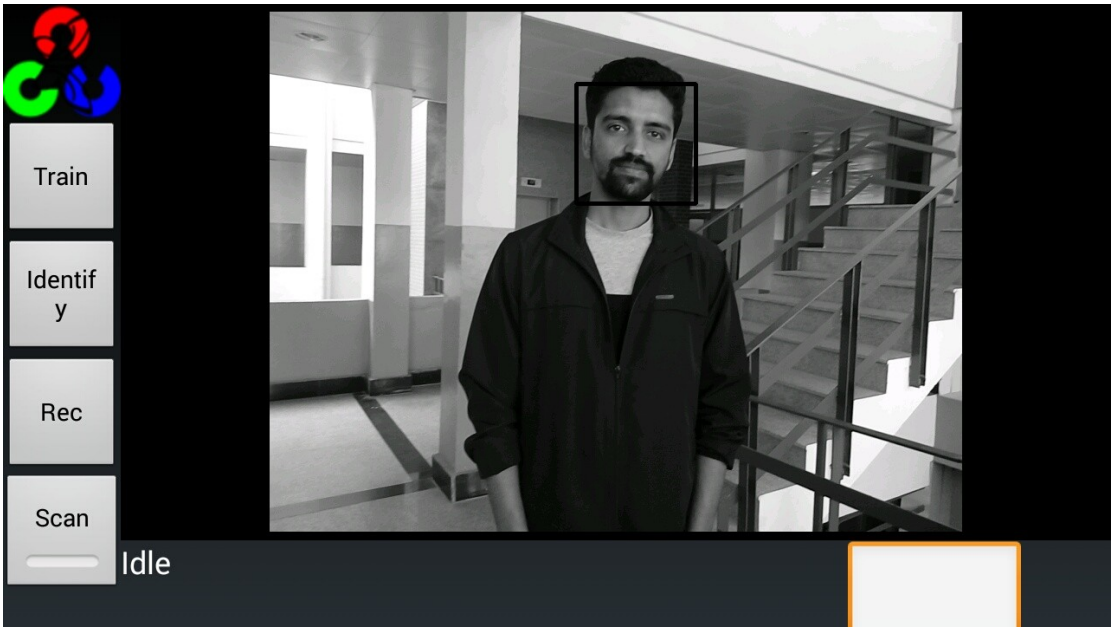


*Figure 6.1*

The application automatically recognizes the face on the screen and highlights it with this black square as shown in *Figure 6.1 and Figure 6.2.*

Even if the face is not align properly it can easily detect any alignment of the face, side, upward, downward any type of alignment.

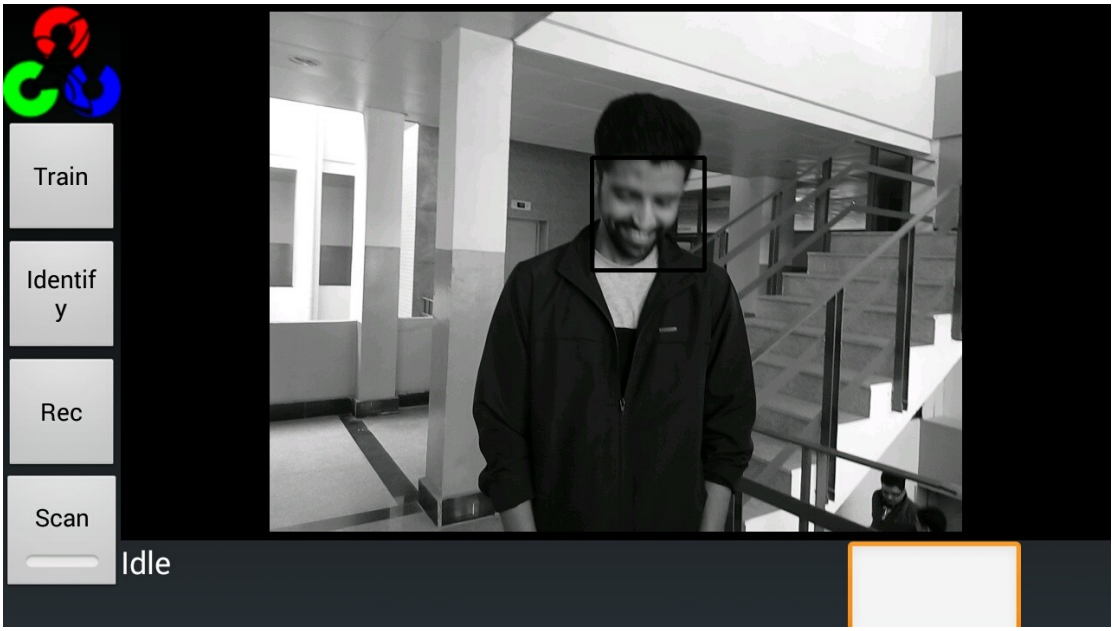


*Figure 6.2*

After taking the photos just add the name of the person in the textbox provided on the right hand side and then click on train button to train the application for this person as shown in *Figure 6.3*.

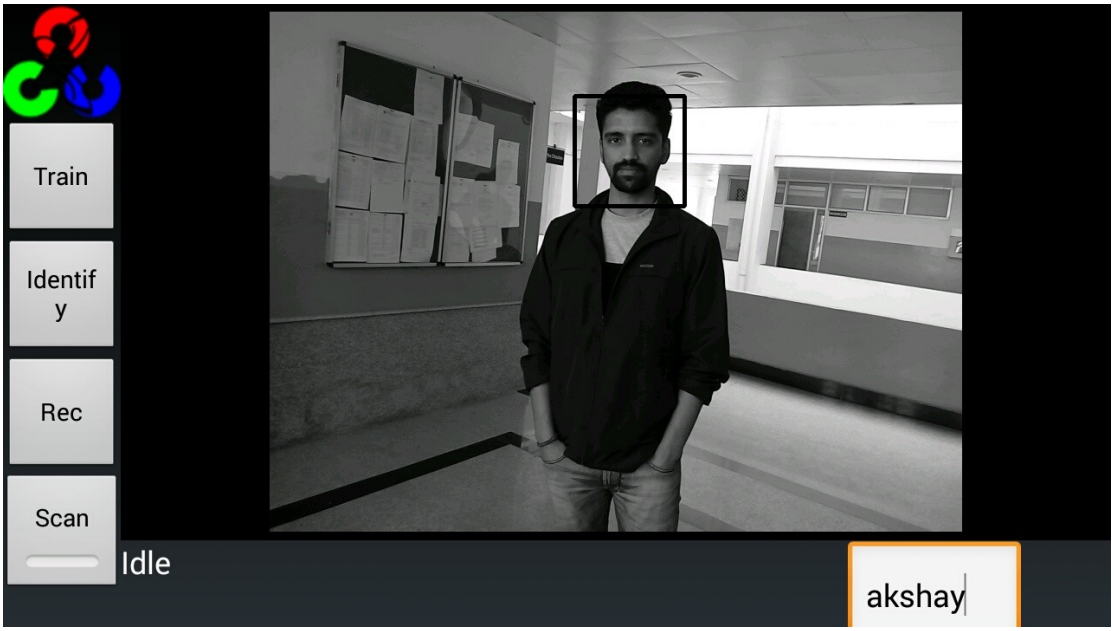


*Figure 6.3*

After training when you again focus on that person it can recognize that person and displays his/her name which you entered at the time of training.

*Figure 6.4* shows the name being displayed on the screen same as that which was entered during training. The name is displayed on left hand side.


*Figure 6.4*

# CHAPTER 7

# CONCLUSION AND FUTURE VISION

The aim of our project was to create an Android application for recognizing faces. After studying various algorithms of face detection and face recognition and analysing various technologies that can be used, we decided to use OpenCV Library for our application. The incorporation of face recognition algorithms into mobile devices has been a challenging problem due to the constraints on processing power, limited storage in the mobile devices, limited network bandwidth and connection instability, privacy and security concerns. The result of our project is the working app which detects and recognizes any person.

For now, this App is a Stand-alone application but it can further be made more useful by intergrating and extending further like it can be used to develop an efficient automated attendance taking system. Moreover it can be further extended to integrate with the social media websites such that our input image can be match with already existing profile on various social media sites, which will help any person anywhere. This type of App has various other applications some of which are already discussed in Chapter 1.

This kind of Android application will be of great use in various aspects of the lives of people and being an android application it will be easily affordable and convenient source for common people.

# REFERENCES

[1] Priyanka, Dr. Yashpal Singh “A Study on Facial Feature Extraction and Facial Recognition Approaches”International Journal of computer Science and Mobile Computing, Vol. 4, pp 166-174, 2015.

[2] Vikram Solunke, Pratik Kudle, Abhijit Bhise, Adil Naik, Prof. J.R. Prasad “A Comparison between Feature Extraction Techniques for Face Recognition” International Journal of Emerging Research in Management &Technology, Volume 3, pp 38-41, 201

[3] M. Turk and A. Pentland, “Eigenfaces for recognition”, Journal of Cognitive Neuroscience 3, 72-86. [Online]. Available: http://www.face-rec.org/algorithms/ PCA/jcn.pdf

[4] Kevin Letupe, “ Android application for Face Recognition” , 2013. [Online]. Available: https://ri.iut-info.univ-lille1.fr/2013-Avril/Herning/LETUP%20Rapport.pdf

[5] Guillaume Dave, Xing Chao, Kishore Sriadibhatla, “ Face Recognition in Mobile Phones” , 2008. [Online]. Available: https://stacks.stanford.edu/file/druid.../ Sriadibhatla_Davo_Chao_FaceRecognition.pdf

[6] Monica Chillaron, Larisa Dunai, Guillermo Peris Fajarnes, Ismael Lengua Lengua, “Face detection and recognition application for Android”, 2015. [Online]. Available: https://www.researchgate.net/publication/284253914_Face_detection_and_recognitio n_application_for_Android

[7] “OpenCV Open Source Computer Vision”. [Online]. Available: https://docs.opencv.org

[8] “Developers”. [Online]. Available: https://developer.android.com

[10] Ambika Ramchandra, Ravindra Kumar “Overview of Face Recognition System Challenges” International Journal of Scientific and Technology Research, Vol. 2, pp 234-236, 2013.

[11] Paul Viola, Michael Jones "Rapid Object Detection using a Boosted Cascade of Simple Features" Conference On Computer Vision And Pattern Recognition ,2001. [Online]. Available: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf