CHAPTER ONE

INTRODUCTION

1.1 Background of study

Artificial intelligence (AI) is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by animals including humans. Leading AI textbooks define the field as the study of "intelligent agents": any system that perceives its environment and takes actions that maximize its chance of achieving its goals. Some of the benefits of AI are automation enhancement, weather forecasting, smart decision making, medical advancement and so much more.

Security is the major concern of today. The need for security is required everywhere from home to commercial, official and from defense point of view. Over the days the security provided or accessing system is developed based on password access, mechanical system-based access and biometrical features-based access. In biometrical based systems, fingerprint, iris, and face recognition and detection are widely used. Security systems have been developed over the years with different discrete access codes being employed, but also in the recent past, with the ability of face recognition principles or ideas being applied to a wide range of problems like criminal identification and crowd surveillance. Online Biometric security is one of the tools that Information Technology (IT) is trying to maximize to full potential. Today, security is one of the most important challenges we face in the day-to-day activities of life; there are several security systems, most of which are either electronic or manual. However, the most significant security check systems are those that use biometrics – which is unique to each individual. Iris, fingerprints, palm check, face recognition are all examples of biometric tools. Face recognition is a popular type of biometric system. Such systems are essentially pattern recognition systems that operate by fetching an individual's biometric data, extracting a feature set from the acquired data (in this case the face), and comparing this feature against the template set in the database. Depending on the application context, a biometric system may operate either in verification mode or identification mode.

Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast). Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

1.2 Problem Statement

Security is one of the main objectives of every individual, everyone wants to feel safe within their home and work environment. This security requirement increases as the importance of the premises increases. For this reason, security personnel in places such as banks and other establishment of high importance such as military departments and many others may need assistance when it comes to security control. An efficient face recognition system can be of great help in forensic sciences, identification for law enforcement, surveillance, and authentication for banking and security system. The use of face recognition system can help to limit or inhibit the disadvantages and weaknesses of human

based security systems thereby solving many of the problems and providing a much better solution for security control.

1.3 Aim and Objective

Here are few objectives and aim of (Optical character recognition and Face Recognition System):

- To design real time-face detection and recognition system.
- To design a real time- Optical character recognition system

1.4 Significance of the study

- Helps find missing people and identify perpetrators.
- Protects businesses against theft.
- Strengthens security measures in banks and airports.
- Improves photo organization and many more.

1.5 Scope of the study

The scope of this study covers only all the two listed system, face detection and recognition, Optical character recognition accessing different trained Models and packages in order to provide it main aim of production.

1.6 Limitation of the study

1. Saving captured images for long time keeping.
2. Unable to detect a face with excessive saturations or facial damage.
3. Saves one image at a time.
4. Uploading saved images to a cloud database for future uses.

1.7 Organization of the Report

The project is organized in five chapters.

Chapter One: and the whole idea of this research work presentation in chapter one, like objective of the study, statement of the research area of coverage limitation and definition of terms all this makes up the chapter one.

Chapter Two: this section deals with the review of study, review of concept theories upon which this work is built on, the potential issues in the Optical character recognition and Face Recognition System.

Chapter Three: talks about the software tools used in the project mainly related to programming language. The methodology at which this research work will be implemented.

Chapter Four: the system is implemented and presented with its analysis. Functions of the system and the operation of the system is also, in depth explained for reader understating and comprehension. The system requirement is also detailed and the platform at which the system can run on.

Chapter Five: summaries the whole work done and make possible recommendation and suggest other points to be included into the work for future propose

1.8 Definition of Terms

1 Attribute: A data item that characterize an object
2 Data flow: Movement of data in a system from a point of origin to specific destination indicated by a line and arrow
3 Data Security: Protection of data from loss, disclosure, modification or destruction.
4 Design: Process of developing the technical and operational specification of a candidate system for implements.
5 File: Collection of related records organized for a particular purpose also called dataset.
6 Flow Chart: A graphical picture of the logical steps and sequence involved in a procedure or a program.
7 Implementation: In system development-phase that focuses on user training, site preparation and file conversion for installing a candidate system.

8 Normalization: A process of replacing a given file with its logical equivalent the object is to derive simple files with no redundant elements.

9 Operation System: In database – machine-based software that facilitates the availability of information or reports through the DBMS.

10 Password: Identity authenticators a key that allow access to a program system a procedure.

11 Record: A collection of aggregates or related items of a data treated as a unit.

12 Source Code: A procedure or format that allow enhancements on a software package.

13 System: A regular or orderly arrangements of components or parts in a connected and interrelated series or whole a group of components necessary to some operation.

14 System Design: Detailed concentration on the technical and other specification that will make the new system operational.

15 OCR: Optical character recognition.

16 Optical character recognition: is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

17 Java: is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible.

18 JDK: The Java Development Kit (JDK) is a distribution of Java Technology by Oracle Corporation. It implements the Java Language Specification (JLS) and the Java Virtual Machine Specification (JVMS) and provides the Standard Edition (SE) of the Java Application Programming Interface (API). tem (called the guest).

19 Debug: is an act used to test and debug other programs

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview of Relevant Technology

Early optical character recognition may be traced to technologies involving telegraphy and creating reading devices for the blind. In 1914, Emanuel Goldberg developed a machine that read characters and converted them into standard telegraph code. Concurrently, Edmund Fournier d'Albe developed the Optophone, a handheld scanner that when moved across a printed page, produced tones that corresponded to specific letters or characters. In the late 1920s and into the 1930s Emanuel Goldberg developed what he called a "Statistical Machine" for searching microfilm archives using an optical code recognition system. In 1931 he was granted USA Patent number 1,838,389 for the invention. The patent was acquired by IBM.

In 1974, Ray Kurzweil started the company Kurzweil Computer Products, Inc. and continued development of omni-font OCR, which could recognize text printed in virtually any font (Kurzweil is often credited with inventing omni-font OCR, but it was in use by companies, including CompuScan, in the late 1960s and 1970s). Kurzweil decided that the best application of this technology would be to create a reading machine for the blind, which would allow blind people to have a computer read text to them out loud. This device required the invention of two enabling technologies – the CCD flatbed scanner and the text-to-speech synthesizer. On January 13, 1976, the successful finished product was unveiled during a widely reported news conference headed by Kurzweil and the leaders of the National Federation of the Blind.In 1978, Kurzweil Computer Products began selling a commercial version of the optical character recognition computer program. LexisNexis was one of the first customers, and bought the program to upload legal paper and news documents onto its nascent online databases. Two years later, Kurzweil sold his company to Xerox, which had an interest in further commercializing paper-to-computer text

conversion. Xerox eventually spun it off as Scansoft, which merged with Nuance Communications.

In the 2000s, OCR was made available online as a service (Weber), in a cloud computing environment, and in mobile applications like real-time translation of foreign-language signs on a smartphone. With the advent of smart-phones and smartglasses, OCR can be used in internet connected mobile device applications that extract text captured using the device's camera. These devices that do not have OCR functionality built into the operating system will typically use an OCR API to extract the text from the image file captured and provided by the device. The OCR API returns the extracted text, along with information about the location of the detected text in the original image back to the device app for further processing (such as text-to-speech) or display.

OCR engines have been developed into many kinds of domain-specific OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- Data entry for business documents, e.g. Cheque, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- In airports, for passport recognition and information extraction
- Automatic insurance documents key information extraction
- Traffic sign recognition
- Extracting business card information into a contact list

A facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces, typically employed to authenticate users through ID verification services, works by pinpointing and measuring facial features from a given image.

Facial recognition systems are employed throughout the world today by governments and private companies. Their effectiveness varies, and some systems have previously been scrapped because of their ineffectiveness. The use of facial recognition systems has also raised controversy, with claims that the systems violate citizens' privacy, commonly make incorrect identifications, encourage gender norms and racial profiling, and do not protect important biometric data. These claims have led to the ban of facial recognition systems in several cities in the United States. As a result of growing societal concerns, Meta announced that it plans to shut down Facebook facial recognition system, deleting the face scan data of more than one billion users. This change will represent one of the largest shifts in facial recognition usage in the technology's history.

Automated facial recognition was pioneered in the 1960s. Woody Bledsoe, Helen Chan Wolf, and Charles Bisson worked on using the computer to recognize human faces. Their early facial recognition project was dubbed "man-machine" because the coordinates of the facial features in a photograph had to be established by a human before they could be used by the computer for recognition. On a graphics tablet a human had to pinpoint the coordinates of facial features such as the pupil centers, the inside and outside corner of eyes, and the widows peak in the hairline. The coordinates were used to calculate 20 distances, including the width of the mouth and of the eyes. A human could process about 40 pictures an hour in this manner and so build a database of the computed distances. A computer would then automatically compare the distances for each photograph, calculate the difference between the distances and return the closed records as a possible match.

In 1970, Takeo Kanade publicly demonstrated a face matching system that located anatomical features such as the chin and calculated the distance ratio between facial features without human intervention. Later tests revealed that the system could not always reliably identify facial features. Nonetheless, interest in the subject grew and in 1977 Kanade published the first detailed book on facial recognition technology.

2.2 Review of the related work

For the study of an existing system or technology, Take Amazon Rekognition Software as a case of study

Amazon Rekognition is a cloud-based software as a service (SaaS) computer vision platform that was launched in 2016. It has been sold and used by a number of United States government agencies, including U.S. Immigration and Customs Enforcement (ICE) and Orlando, Florida police, as well as private entities. Amazon Rekognition makes it easy to add image and video analysis to its applications using proven, highly scalable, deep learning technology that requires no machine learning expertise to use. With Amazon Rekognition, it can identify objects, people, text, scenes, and activities in images and videos, as well as detect any inappropriate content. Amazon Rekognition also provides highly accurate facial analysis and facial search capabilities that it can use to detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

2.2.1 Capabilities of the Amazon Rekognition

Rekognition provides a number of computer vision capabilities, which can be divided into two categories: Algorithms that are pre-trained on data collected by Amazon or its partners, and algorithms that a user can train on a custom dataset. As of July 2019, Rekognition provides the following computer vision capabilities.

Pre-trained algorithms:

1. Celebrity recognition in images
2. People Pathing enables tracking of people through a video. An advertised use-case of this capability is to track sports players for post-game analysis.
3. Text detection and classification in images
4. Unsafe visual content detection

2.2.2 Benefits of Using the Amazon Rekognition

1. **Automate visual analysis tasks at scale:** With Amazon Rekognition, it can tackle tasks that cannot be performed manually such as analyzing millions of images and videos within minutes. it can also increase workforce productivity and minimize errors by augmenting cumbersome, repetitive human visual review tasks with machine learning.
2. **Get started without ML expertise**: With Amazon Rekognition, it can quickly add a wide range of pre-trained computer vision APIs to one's applications within hours, without the need to collect data and build computer vision models from scratch. If it has unique requirements, it can use AutoML to automatically train & host custom ML models by uploading as few as ten labeled images.
3. **Reduce ML infrastructure costs:** Amazon Rekognition is a fully managed service that can automatically scale up and down based on companies, individual and people business needs. it does not need to build and manage its own ML infrastructure. it can quickly deliver reliable, scalable, and secure applications powered by computer vision and pay only for the images and videos that it analyzes.

2.2.3 Application or Use Cases of the Amazon Rekognition

1. Media Analysis.
2. Make content searchable.
3. Flag inappropriate content.
4. Enable digital identity verification.
5. Workplace Safety.

2.3 Summary of the problem of the existing system

1. The cost is bit more for small scale companies.
2. For text processing, OCR is not available in amazon rekognition.
3. Only facial search is available in image search.
4. Expensive to maintain

2.4 Summary

This application is to be integrated with four different systems, which are the pose estimate, face mask detection, face detection and recognition and then the object detection system, for the pose estimation, it is very useful in augmented reality, gaming, and robotics, for the face mask detection, this system is very useful especially during an epidemic era or the covid-19 era, the face detection and recognition systems, detects face and recognize it according the captured image stored in the trained centered and then finally the object detection system, this on the other hand detects an object and produce the name and the rate of the accuracy.

CHAPTER THREE

METHODOLGY

3.1 Design Consideration

The Proposed Face Recognition and Optical character recognition (Project), is an android application, that detects a human face and attempts to recognize it by producing a name or an id of the face detected Optical character recognition detects an image and prints out all text that are available in the image. The application will be developed by using, TensorFlow, Java and Xml, with the android studio environment. At the end of the development, the application will be able to detect a face using real time and local disk to detect and to recognize a human face, detect image and produce text (Image to Text).

3.2 Summary of project methodology

Face Recognition and Optical character recognition System is an android application, that detects a human face and attempts to recognize it by producing a name or an id of the face detected, and detects an image and prints out all text that are available in the image.

3.3 Data Source/Collection

1. fEMG: As with EMG, the data refers to the level of electrical activity generated by the muscles, after a baseline has been determined.
2. 2. SCface - Surveillance Cameras Face Database
3. Multi-PIE
4. The Yale Face Database.

3.4 Hardware Requirement

1 GHz or better processor is recommended.

2 At least 512 MB of free RAM should be available for the application.

3 A phone camera

4 Minimum OS of Android 5.0 "Lollipop

5 Maximum OS of Android 9.0” Pie”

3.5 Software Requirement

1 Android Studio: Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

2 OpenCv: OpenCV is a simple to use library in python (written in C++) that allows us to do various image processing like blurring, denoising, gray scaling, finding contours, drawing shapes, etc. within just 10 lines of code.

3 TensorFlow: TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

3.6 Design Architecture

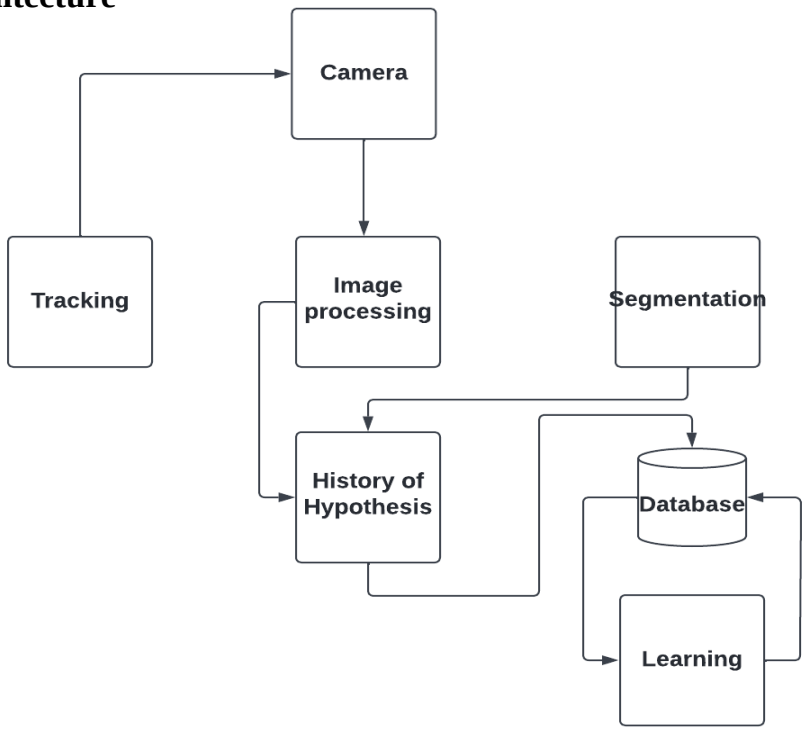


Fig 4: Design Architecture

3.7 Block diagram of proposed system

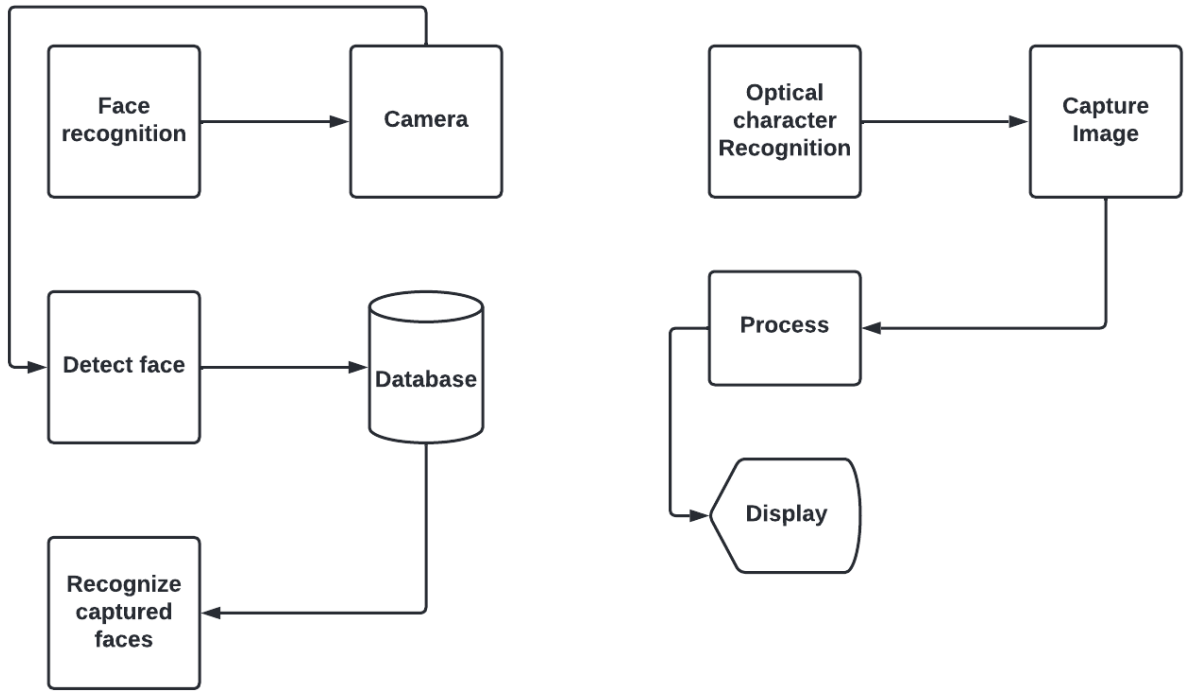


Fig 5: Block diagram of all the systems

3.8 Detailed Design of each subsystem

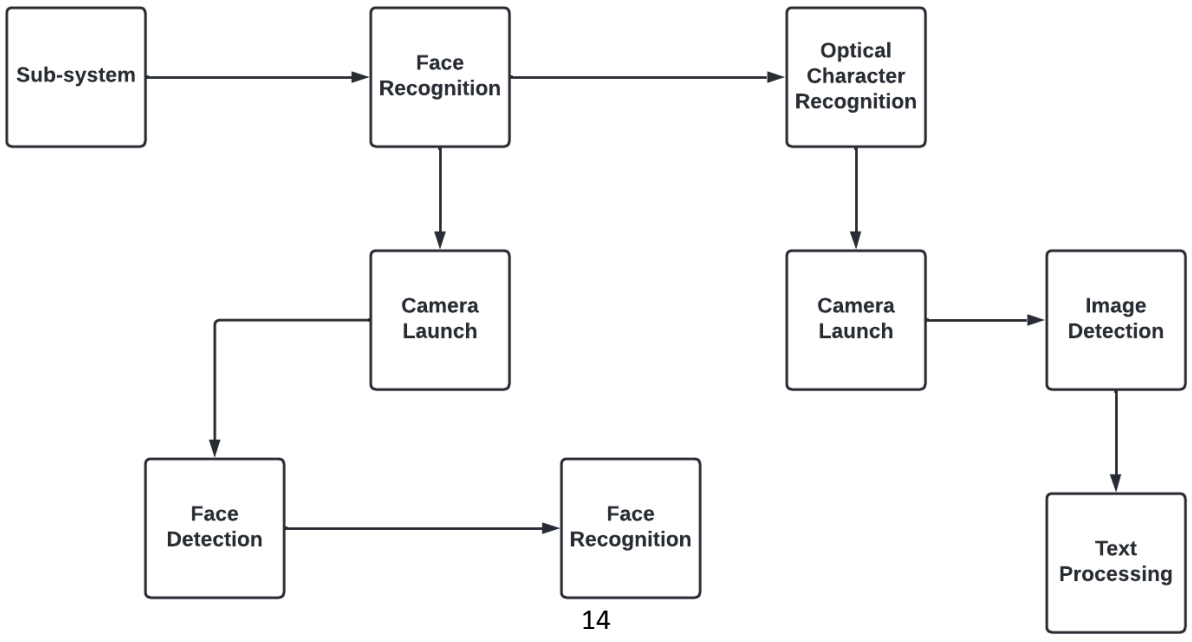


Fig 6: Sub-system Design

Facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces, typically employed to authenticate users through ID verification services, works by pinpointing and measuring facial features from a given image.

Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).

CHAPTER FOUR

RESULT AND DISCUSSION

4.1 Implementation Procedures

Implementation and development of this project was done with the use of the Android Studio IDE, TensorFlow, OpenCV and Firebase. The android studio was used in writing the codes, the entire project was written in Java, Xml, Python.

Tensor flow dependency was added to the android studio Build.gradle, this dependency contains resource files and API that was added to the app directly. Below is the list of dependencies that was used during implementation.

For the face recognition system:

```
implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation
'com.google.android.material:material:1.5.0'
    implementation
'androidx.constraintlayout:constraintlayout:2.1.3'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation
'androidx.test.ext:junit:1.1.3'
    androidTestImplementation
'androidx.test.espresso:espresso-core:3.4.0'
    implementation('org.tensorflow:tensorflow-lite:0.0.0-
nightly') { changing = true }
    androidTestImplementation
'com.google.truth:truth:1.0.1'
// added MLKit dependencies for face detector
    implementation 'com.google.mlkit:face-detection:16.1.5'
```

For the Optical Character Recognition:

```
implementation 'com.google.android.gms:play-services-
vision:20.1.3'
implementation 'org.tensorflow:tensorflow-lite:0.0.0-
```

```
nightly-SNAPSHOT'
implementation 'org.tensorflow:tensorflow-lite-
support:0.3.0'
```
Appendices

b. Code used for each subsystem

Camera Activity code:

```
package com.pack.michealapp;

import android.app.Fragment;
import android.graphics.SurfaceTexture;
import android.hardware.Camera;
import android.hardware.Camera.CameraInfo;
import android.os.Bundle;

import com.faceexpression.face.customview.AutoFitTextureView;
import com.faceexpression.face.env.ImageUtils;
import com.faceexpression.face.env.Logger;

import java.io.IOException;
import java.util.List;

public class LegacyCameraConnectionFragment extends Fragment {
 private static final Logger LOGGER = new Logger();
 /** Conversion from screen rotation to JPEG orientation. */
 private static final SparseIntArray ORIENTATIONS = new SparseIntArray();
 private static final String KEY_FACING = "camera_facing";

 static {
  ORIENTATIONS.append(Surface.ROTATION_0, 90);
  ORIENTATIONS.append(Surface.ROTATION_90, 0);
  ORIENTATIONS.append(Surface.ROTATION_180, 270);
  ORIENTATIONS.append(Surface.ROTATION_270, 180);
 }

 private Camera camera;
 private Camera.PreviewCallback imageListener;
 private Size desiredSize;
 private int facing;
 /** The layout identifier to inflate for this Fragment. */
 private int layout;
```

```
 /** An {@link AutoFitTextureView} for camera preview. */
 private AutoFitTextureView textureView;
 /**
  * {@link TextureView.SurfaceTextureListener} handles several lifecycle events on a
{@link
  * TextureView}.
  */
 private final TextureView.SurfaceTextureListener surfaceTextureListener =
    new TextureView.SurfaceTextureListener() {
      @Override
      public void onSurfaceTextureAvailable(
         final SurfaceTexture texture, final int width, final int height) {

       int index = getCameraId();
       camera = Camera.open(index);

       try {
        Camera.Parameters parameters = camera.getParameters();
        List<String> focusModes = parameters.getSupportedFocusModes();
        if (focusModes != null
           &&
focusModes.contains(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE)) {

parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE);
        }
        List<Camera.Size> cameraSizes = parameters.getSupportedPreviewSizes();
        Size[] sizes = new Size[cameraSizes.size()];
        int i = 0;
        for (Camera.Size size : cameraSizes) {
         sizes[i++] = new Size(size.width, size.height);
        }
        Size previewSize =
           CameraConnectionFragment.chooseOptimalSize(
              sizes, desiredSize.getWidth(), desiredSize.getHeight());
        parameters.setPreviewSize(previewSize.getWidth(), previewSize.getHeight());
        camera.setDisplayOrientation(90);
        camera.setParameters(parameters);
        camera.setPreviewTexture(texture);
       } catch (IOException exception) {
        camera.release();
```

```
        }

        camera.setPreviewCallbackWithBuffer(imageListener);
        Camera.Size s = camera.getParameters().getPreviewSize();
        camera.addCallbackBuffer(new byte[ImageUtils.getYUVByteSize(s.height, s.width)]);

        textureView.setAspectRatio(s.height, s.width);

        camera.startPreview();
      }

      @Override
      public void onSurfaceTextureSizeChanged(
          final SurfaceTexture texture, final int width, final int height) {}

      @Override
      public boolean onSurfaceTextureDestroyed(final SurfaceTexture texture) {
        return true;
      }

      @Override
      public void onSurfaceTextureUpdated(final SurfaceTexture texture) {}
    };
  /** An additional thread for running tasks that shouldn't block the UI. */

}
```

LagacyCameraConnectionFragment code

```
package com.pack.michealapp;;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.app.Fragment;
import android.content.Context;
import android.content.DialogInterface;
import android.content.res.Configuration;
import android.graphics.ImageFormat;
```

```
import android.graphics.Matrix;
import android.graphics.RectF;
import android.graphics.SurfaceTexture;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.CaptureResult;

import com.faceexpression.face.customview.AutoFitTextureView;
import com.faceexpression.face.env.Logger;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;


@SuppressLint("ValidFragment")
public class CameraConnectionFragment extends Fragment {
  private static final Logger LOGGER = new Logger();

  /**
   * The camera preview size will be chosen to be the smallest frame by pixel size capable
of
   * containing a DESIRED_SIZE x DESIRED_SIZE square.
   */
  private static final int MINIMUM_PREVIEW_SIZE = 320;

  /** Conversion from screen rotation to JPEG orientation. */
  private static final SparseIntArray ORIENTATIONS = new SparseIntArray();

  private static final String FRAGMENT_DIALOG = "dialog";

  static {
```

```
  ORIENTATIONS.append(Surface.ROTATION_0, 90);
  ORIENTATIONS.append(Surface.ROTATION_90, 0);
  ORIENTATIONS.append(Surface.ROTATION_180, 270);
  ORIENTATIONS.append(Surface.ROTATION_270, 180);
 }

 /** A {@link Semaphore} to prevent the app from exiting before closing the camera. */
 private final Semaphore cameraOpenCloseLock = new Semaphore(1);
 /** A {@link OnImageAvailableListener} to receive frames as they are available. */
 private final OnImageAvailableListener imageListener;
 /** The input size in pixels desired by TensorFlow (width and height of a square bitmap).
*/
 private final Size inputSize;
 /** The layout identifier to inflate for this Fragment. */
 private final int layout;

 private final ConnectionCallback cameraConnectionCallback;
 private final CameraCaptureSession.CaptureCallback captureCallback =
    new CameraCaptureSession.CaptureCallback() {
      @Override
      public void onCaptureProgressed(
          final CameraCaptureSession session,
          final CaptureRequest request,
          final CaptureResult partialResult) {}

      @Override
      public void onCaptureCompleted(
          final CameraCaptureSession session,
          final CaptureRequest request,
          final TotalCaptureResult result) {}
    };
 /** ID of the current {@link CameraDevice}. */
 private String cameraId;
 /** An {@link AutoFitTextureView} for camera preview. */
 private AutoFitTextureView textureView;
 /** A {@link CameraCaptureSession } for camera preview. */
 private CameraCaptureSession captureSession;
 /** A reference to the opened {@link CameraDevice}. */
 private CameraDevice cameraDevice;
 /** The rotation in degrees of the camera sensor from the display. */
```

```
 private Integer sensorOrientation;
 /** The {@link Size} of camera preview. */
 private Size previewSize;
 /** An additional thread for running tasks that shouldn't block the UI. */
 private HandlerThread backgroundThread;
 /** A {@link Handler} for running tasks in the background. */
 private Handler backgroundHandler;
 /** An {@link ImageReader} that handles preview frame capture. */
 private ImageReader previewReader;
 /** {@link CaptureRequest.Builder} for the camera preview */
 private CaptureRequest.Builder previewRequestBuilder;
 /** {@link CaptureRequest} generated by {@link #previewRequestBuilder} */
 private CaptureRequest previewRequest;
 /** {@link CameraDevice.StateCallback} is called when {@link CameraDevice} changes
its state. */
 private final CameraDevice.StateCallback stateCallback =
    new CameraDevice.StateCallback() {
      @Override
      public void onOpened(final CameraDevice cd) {
        // This method is called when the camera is opened.  We start camera preview
here.
        cameraOpenCloseLock.release();
        cameraDevice = cd;
        createCameraPreviewSession();
      }
// We set up a CaptureRequest.Builder with the output Surface.
previewRequestBuilder =
cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
previewRequestBuilder.addTarget(surface);

LOGGER.i("Opening camera preview: " + previewSize.getWidth() + "x" +
previewSize.getHeight());

// Create the reader for the preview frames.
previewReader =
   ImageReader.newInstance(
      previewSize.getWidth(), previewSize.getHeight(), ImageFormat.YUV_420_888, 2);

previewReader.setOnImageAvailableListener(imageListener, backgroundHandler);
previewRequestBuilder.addTarget(previewReader.getSurface());
```

```
// Here, we create a CameraCaptureSession for camera preview.
cameraDevice.createCaptureSession(
   Arrays.asList(surface, previewReader.getSurface()),
   new CameraCaptureSession.StateCallback() {

     @Override
     public void onConfigured(final CameraCaptureSession cameraCaptureSession) {
      // The camera is already closed
      if (null == cameraDevice) {
       return;
      }

      // When the session is ready, we start displaying the preview.
      captureSession = cameraCaptureSession;
      try {
       // Auto focus should be continuous for camera preview.
       previewRequestBuilder.set(
          CaptureRequest.CONTROL_AF_MODE,
          CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
       // Flash is automatically enabled when necessary.
       previewRequestBuilder.set(
          CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_ON_AUTO_FLASH);

       // Finally, we start displaying the camera preview.
       previewRequest = previewRequestBuilder.build();
       captureSession.setRepeatingRequest(
          previewRequest, captureCallback, backgroundHandler);
      } catch (final CameraAccessException e) {
       LOGGER.e(e, "Exception!");

}
```

DetectorActivity Codes

```
package com.pack.michealapp;;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
```

```
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.hardware.camera2.CameraCharacteristics;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.Bundle;
import android.os.SystemClock;
import android.util.Size;
import android.util.TypedValue;
import android.view.LayoutInflater
;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import com.faceexpression.face.customview.OverlayView;
import com.faceexpression.face.env.BorderedText;
import com.faceexpression.face.env.ImageUtils;
import com.faceexpression.face.env.Logger;
import com.faceexpression.face.tflite.SimilarityClassifier;
import com.faceexpression.face.tflite.TFLiteObjectDetectionAPIModel;
import com.faceexpression.face.tracking.MultiBoxTracker;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.mlkit.vision.common.InputImage;
import com.google.mlkit.vision.face.Face;
import com.google.mlkit.vision.face.FaceDetection;
import com.google.mlkit.vision.face.FaceDetector;
import com.google.mlkit.vision.face.FaceDetectorOptions;

import java.io.IOException;
import java.util.LinkedList;
import java.util.List;
```

```
/**
 * An activity that uses a TensorFlowMultiBoxDetector and ObjectTracker to detect and
then track
 * objects.
 */
public class DetectorActivity extends CameraActivity implements
OnImageAvailableListener {
  private static final Logger LOGGER = new Logger();


  // FaceNet
//  private static final int TF_OD_API_INPUT_SIZE = 160;
//  private static final boolean TF_OD_API_IS_QUANTIZED = false;
//  private static final String TF_OD_API_MODEL_FILE = "facenet.tflite";
//  //private static final String TF_OD_API_MODEL_FILE = "facenet_hiroki.tflite";

  // MobileFaceNet
  private static final int TF_OD_API_INPUT_SIZE = 112;
  private static final boolean TF_OD_API_IS_QUANTIZED = false;
  private static final String TF_OD_API_MODEL_FILE = "mobile_face_net.tflite";


  private static final String TF_OD_API_LABELS_FILE =
"file:///android_asset/labelmap.txt";

  private static final DetectorMode MODE = DetectorMode.TF_OD_API;
  // Minimum detection confidence to track a detection.
  private static final float MINIMUM_CONFIDENCE_TF_OD_API = 0.5f;
  private static final boolean MAINTAIN_ASPECT = false;

  private static final Size DESIRED_PREVIEW_SIZE = new Size(640, 480);
  //private static final int CROP_SIZE = 320;
  //private static final Size CROP_SIZE = new Size(320, 320);


  private static final boolean SAVE_PREVIEW_BITMAP = false;
  private static final float TEXT_SIZE_DIP = 10;
  OverlayView trackingOverlay;
  private Integer sensorOrientation;
```

```
private SimilarityClassifier detector;

private long lastProcessingTimeMs;
private Bitmap rgbFrameBitmap = null;
private Bitmap croppedBitmap = null;
private Bitmap cropCopyBitmap = null;

private boolean computingDetection = false;
private boolean addPending = false;
//private boolean adding = false;

private long timestamp = 0;



private FloatingActionButton fabAdd;

//private HashMap<String, Classifier.Recognition> knownFaces = new HashMap<>();


@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);

 fabAdd = findViewById(R.id.fab_add);
 fabAdd.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
   onAddClick();
  }
 });

 // Real-time contour detection of multiple faces
 FaceDetectorOptions options =
       new FaceDetectorOptions.Builder()
            .setPerformanceMode(FaceDetectorOptions.PERFORMANCE_MODE_FAST)
            .setContourMode(FaceDetectorOptions.LANDMARK_MODE_NONE)
            .setClassificationMode(FaceDetectorOptions.CLASSIFICATION_MODE_NONE)
            .build();
```

```
    FaceDetector detector = FaceDetection.getClient(options);

    faceDetector = detector;


    //checkWritePermission();


  @Override
  public void onPreviewSizeChosen(final Size size, final int rotation) {
    final float textSizePx =
        TypedValue.applyDimension(
            TypedValue.COMPLEX_UNIT_DIP, TEXT_SIZE_DIP,
getResources().getDisplayMetrics());
    borderedText = new BorderedText(textSizePx);
    borderedText.setTypeface(Typeface.MONOSPACE);

    tracker = new MultiBoxTracker(this);


    try {
      detector =
          TFLiteObjectDetectionAPIModel.create(
              getAssets(),
              TF_OD_API_MODEL_FILE,
              TF_OD_API_LABELS_FILE,
              TF_OD_API_INPUT_SIZE,
              TF_OD_API_IS_QUANTIZED);
      //cropSize = TF_OD_API_INPUT_SIZE;
    } catch (final IOException e) {
      e.printStackTrace();
      LOGGER.e(e, "Exception initializing classifier!");
      Toast toast =
          Toast.makeText(
              getApplicationContext(), "Classifier could not be initialized",
Toast.LENGTH_SHORT);
      toast.show();
      finish();
```

```
    }

    previewWidth = size.getWidth();
    previewHeight = size.getHeight();

    sensorOrientation = rotation - getScreenOrientation();
    LOGGER.i("Camera orientation relative to screen canvas: %d", sensorOrientation);

    LOGGER.i("Initializing at size %dx%d", previewWidth, previewHeight);
    rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight,
Config.ARGB_8888);


    int targetW, targetH;
    if (sensorOrientation == 90 || sensorOrientation == 270) {
      targetH = previewWidth;
      targetW = previewHeight;
    }
    else {
      targetW = previewWidth;
      targetH = previewHeight;
    }
    int cropW = (int) (targetW / 2.0);
    int cropH = (int) (targetH / 2.0);

    croppedBitmap = Bitmap.createBitmap(cropW, cropH, Config.ARGB_8888);

    portraitBmp = Bitmap.createBitmap(targetW, targetH, Config.ARGB_8888);
    faceBmp = Bitmap.createBitmap(TF_OD_API_INPUT_SIZE, TF_OD_API_INPUT_SIZE,
Config.ARGB_8888);

    frameToCropTransform =
            ImageUtils.getTransformationMatrix(
                    previewWidth, previewHeight,
                    cropW, cropH,
                    sensorOrientation, MAINTAIN_ASPECT);

//    frameToCropTransform =
//            ImageUtils.getTransformationMatrix(
//                    previewWidth, previewHeight,
```

```
//                previewWidth, previewHeight,
//                sensorOrientation, MAINTAIN_ASPECT);

    cropToFrameTransform = new Matrix();
    frameToCropTransform.invert(cropToFrameTransform);


    Matrix frameToPortraitTransform =
            ImageUtils.getTransformationMatrix(
                    previewWidth, previewHeight,
                    targetW, targetH,
                    sensorOrientation, MAINTAIN_ASPECT);



    trackingOverlay = (OverlayView) findViewById(R.id.tracking_overlay);
    trackingOverlay.addCallback(
            new OverlayView.DrawCallback() {
              @Override
              public void drawCallback(final Canvas canvas) {
                tracker.draw(canvas);
                if (isDebug()) {
                  tracker.drawDebug(canvas);
                }
              }
            });

    tracker.setFrameConfiguration(previewWidth, previewHeight, sensorOrientation);
  }


  @Override
  protected void processImage() {
    ++timestamp;
    final long currTimestamp = timestamp;
    trackingOverlay.postInvalidate();

    // No mutex needed as this method is not reentrant.
    if (computingDetection) {
      readyForNextImage();
```

```
    return;
  }
  computingDetection = true;

  LOGGER.i("Preparing image " + currTimestamp + " for detection in bg thread.");

  rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0, previewWidth,
previewHeight);

  readyForNextImage();

  final Canvas canvas = new Canvas(croppedBitmap);
  canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);
  // For examining the actual TF input.
  if (SAVE_PREVIEW_BITMAP) {
    ImageUtils.saveBitmap(croppedBitmap);
  }

  InputImage image = InputImage.fromBitmap(croppedBitmap, 0);
  faceDetector
        .process(image)
        .addOnSuccessListener(new OnSuccessListener<List<Face>>() {
          @Override
          public void onSuccess(List<Face> faces) {
            if (faces.size() == 0) {
              updateResults(currTimestamp, new LinkedList<>());
              return;
            }
            runInBackground(
                    new Runnable() {
                      @Override
                      public void run() {
                        onFacesDetected(currTimestamp, faces, addPending);
                        addPending = false;
                      }
                    });
          }

        });
```

```
}

@Override
protected int getLayoutId() {
  return R.layout.tfe_od_camera_connection_fragment_tracking;
}

@Override
protected Size getDesiredPreviewFrameSize() {
  return DESIRED_PREVIEW_SIZE;
}

// Which detection model to use: by default uses Tensorflow Object Detection API frozen
// checkpoints.
private enum DetectorMode {
  TF_OD_API;
}

@Override
protected void setUseNNAPI(final boolean isChecked) {
  runInBackground(() -> detector.setUseNNAPI(isChecked));
}

@Override
protected void setNumThreads(final int numThreads) {
  runInBackground(() -> detector.setNumThreads(numThreads));
}


// Face Processing
private Matrix createTransform(
        final int srcWidth,
        final int srcHeight,
        final int dstWidth,
        final int dstHeight,
        final int applyRotation) {

  Matrix matrix = new Matrix();
  if (applyRotation != 0) {
```

```
    if (applyRotation % 90 != 0) {
      LOGGER.w("Rotation of %d % 90 != 0", applyRotation);
    }

    // Translate so center of image is at origin.
    matrix.postTranslate(-srcWidth / 2.0f, -srcHeight / 2.0f);

    // Rotate around origin.
    matrix.postRotate(applyRotation);
   }


   if (applyRotation != 0) {

    // Translate back from origin centered reference to destination frame.
    matrix.postTranslate(dstWidth / 2.0f, dstHeight / 2.0f);
   }

   return matrix;

  }

  private void showAddFaceDialog(SimilarityClassifier.Recognition rec) {

   AlertDialog.Builder builder = new AlertDialog.Builder(this);
   LayoutInflater inflater = getLayoutInflater();
   View dialogLayout = inflater.inflate(R.layout.image_edit_dialog, null);
   ImageView ivFace = dialogLayout.findViewById(R.id.dlg_image);
   TextView tvTitle = dialogLayout.findViewById(R.id.dlg_title);
   EditText etName = dialogLayout.findViewById(R.id.dlg_input);

   tvTitle.setText("Add Face");
   ivFace.setImageBitmap(rec.getCrop());

   });

     }
   }
}
```

Main3Activity code:

```
package com.pack.michealapp;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.annotation.SuppressLint;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.graphics.drawable.BitmapDrawable;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.SparseArray;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.Toast;

import com.google.android.gms.vision.Frame;
import com.google.android.gms.vision.text.TextBlock;
import com.google.android.gms.vision.text.TextRecognizer;
import com.theartofdev.edmodo.cropper.CropImage;
import com.theartofdev.edmodo.cropper.CropImageView;

public class Main3Activity extends AppCompatActivity {

    EditText mResultEt;
```

```
ImageView mPreviewIv;

private static final int CAMERA_REQUEST_CODE = 200;
private static final int STORAGE_REQUEST_CODE = 400;
private static final int IMAGE_PICK_GALLERY_CODE = 1002;
private static final int IMAGE_PICK_CAMERA_CODE = 1001;

String[] cameraPermission;{} ;
String[] storagePermission;{} ;

Uri image_uri;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main3);

    Button button = (Button)findViewById(R.id.addimg);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){
            showImageImportDialog();
        }
    });


    mResultEt = findViewById(R.id.resultEt);
    mPreviewIv = findViewById(R.id.imageIv);

    //camera permission
    cameraPermission = new String[]{Manifest.permission.CAMERA,
            Manifest.permission.WRITE_EXTERNAL_STORAGE};
    storagePermission = new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE};
}

//actionbar menu
 /*
 @Override
 public boolean onCreateOptionsMenu(Menu menu) {
  //inflate menu
```

```
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
//handle actionbar items clicks

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();
    if (id== R.id.addimage){
        showImageImportDialog();
    }
    return super.onOptionsItemSelected(item);
}*/

private void showImageImportDialog() {
    String[] items = {"Camera", "Gallery"};
    AlertDialog.Builder dialog = new AlertDialog.Builder(this);
    //Set title
    dialog.setTitle("Select image");
    dialog.setItems(items, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int which) {
            if (which == 0) {
                //camera option clicked
                if (!checkCameraPermission()) {
                    //camera permission not allowed, request it
                    requestCameraPermission();

                } else {
                    //permission allowed, take picture
                    pickCamera();
                }
            }
            if (which == 1) {
                //gallery option clicked
                    //camera option clicked
                /*for OS marshmallow and above we need to ask runtime permission
                for camera and storage*/
                    if (!checkStoragePermission()) {
                        //Storage permission not allowed , request it
```

```
                    requestStoragePermission();
                } else {
                    //permisssion allowed take picture
                    pickGallery();
                }
            }
        }

    });
    dialog.create() .show();
}

private void pickGallery() {
    //intent to pick image from gallery
    Intent intent = new Intent(Intent.ACTION_PICK);
    //set intent type to image
    intent.setType("image/+");
    startActivityForResult(intent, IMAGE_PICK_GALLERY_CODE);
}

private void pickCamera() {
    ContentValues values = new ContentValues();
    values.put(MediaStore.Images.Media.TITLE, "NewPic");
    values.put(MediaStore.Images.Media.DESCRIPTION, "MichealApp");
    image_uri =
getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,values)
;

    Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, image_uri);
    startActivityForResult(cameraIntent,IMAGE_PICK_CAMERA_CODE);
}

private void requestStoragePermission() {
    ActivityCompat.requestPermissions(this,
storagePermission,STORAGE_REQUEST_CODE);


}
```

```
    private boolean checkStoragePermission() {
        boolean result = ContextCompat.checkSelfPermission(this,
                Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
(PackageManager.PERMISSION_GRANTED);
        return result;
    }

    private void requestCameraPermission() {
        ActivityCompat.requestPermissions(this,
cameraPermission,CAMERA_REQUEST_CODE);
    }

    private boolean checkCameraPermission() {
        boolean result = ContextCompat.checkSelfPermission(this,
                Manifest.permission.CAMERA) == (PackageManager.PERMISSION_GRANTED);
        boolean result1 = ContextCompat.checkSelfPermission(this,
                Manifest.permission.WRITE_EXTERNAL_STORAGE) ==
(PackageManager.PERMISSION_GRANTED);
        return result && result1;

    }

    //handle permission result
    @SuppressLint("MissingSuperCall")
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        switch ( requestCode) {
            case CAMERA_REQUEST_CODE:
                if (grantResults.length > 0) {
                    boolean cameraAccepted = grantResults[0] ==
                            PackageManager.PERMISSION_GRANTED;
                    boolean writeStorageAccepted = grantResults[0] ==
                            PackageManager.PERMISSION_GRANTED;
                    if (cameraAccepted && writeStorageAccepted) {
                        pickCamera();
                    }
                    else {
                        Toast.makeText(this, "permission denied", Toast.LENGTH_SHORT).show();
                    }
```

```
            }
            break;
        case STORAGE_REQUEST_CODE:
            if (grantResults.length > 0) {
                boolean writeStorageAccepted = grantResults[0] ==
                        PackageManager.PERMISSION_GRANTED;
                if (writeStorageAccepted) {
                    pickCamera();
                }
                else {
                    Toast.makeText(this, "permission denied", Toast.LENGTH_SHORT).show();
                }
            }
            break;
    }
  }
  //handle image result

  @SuppressLint("MissingSuperCall")
  @Override
  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
    //got image from camera
    if (resultCode == RESULT_OK) {
        if (requestCode == IMAGE_PICK_GALLERY_CODE){
            //got image from gallery now crop it
            CropImage.activity(data.getData())
                    .setGuidelines(CropImageView.Guidelines.ON) //enable image guidlines
            .start(this);
        }
        if (requestCode == IMAGE_PICK_CAMERA_CODE){
            CropImage.activity(image_uri)
                    .setGuidelines(CropImageView.Guidelines.ON) //enable image guidlines
                    .start(this);

        }
    }
    //get cropped image
    if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE) {
        CropImage.ActivityResult result = CropImage.getActivityResult(data);
```

```
        if (resultCode == RESULT_OK){
            Uri resuultUri = result.getUri();
            //set image to image view
            mPreviewIv.setImageURI(resuultUri);

            //get
            BitmapDrawable bitmapDrawable =
(BitmapDrawable)mPreviewIv.getDrawable();
            Bitmap bitmap = bitmapDrawable.getBitmap();
            TextRecognizer recogniser = new
TextRecognizer.Builder(getApplicationContext()).build();
            if (!recogniser.isOperational()){
                Toast.makeText(this, "Error", Toast.LENGTH_SHORT).show(); }
            else
            {
                Frame frame = new Frame.Builder().setBitmap(bitmap).build();
                SparseArray<TextBlock> item = recogniser.detect(frame);
                StringBuilder sb = new StringBuilder();
                //get text from sb until there is no text
                for (int i=0; i<item.size(); i++)
                {
                    TextBlock myItem = item.valueAt(i);
                    sb.append(myItem.getValue());
                    sb.append("\n");
                }
                //set text to edit
                mResultEt.setText(sb.toString());
            }
        }
        else if (resultCode == CropImage.CROP_IMAGE_ACTIVITY_RESULT_ERROR_CODE){
            //if there is any error show it
            Exception error = result.getError();
            Toast.makeText(this, "+error", Toast.LENGTH_SHORT).show();




        }
    }
```

```
    }
}
```

4.2 Coding environment and implementation

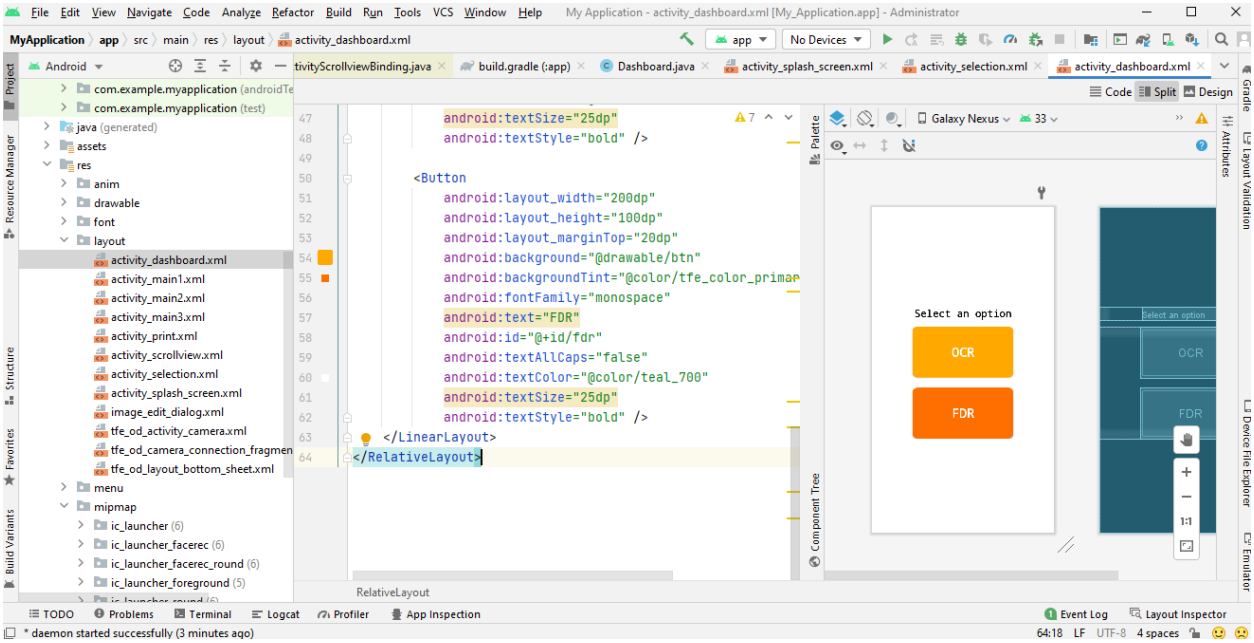


Fig 7: Android studio environment

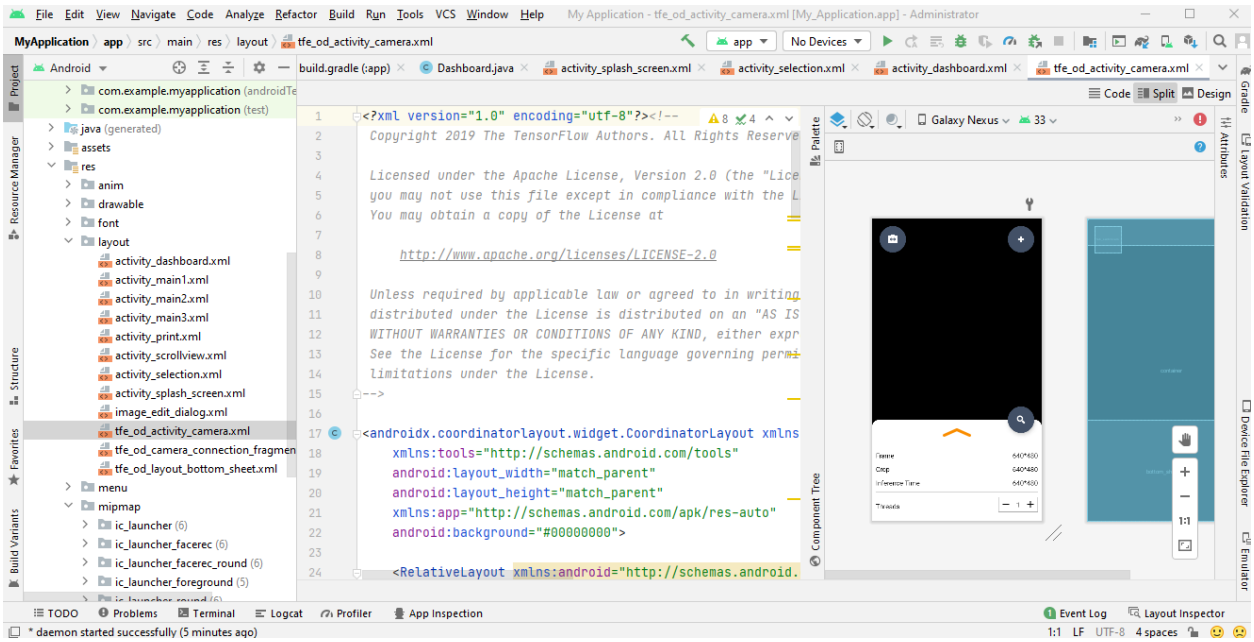


Fig 8: Android studio environment

In fig 8, it shows the android studio environment of the face recognition system, the android is and IDE for developing android application, but of recent it is proven to also develop IOS application there by making it a cross-platform IDE.

The camera activity in fig 8, this is almost the main activity because it initializes the camera permission on the device thereby making the devices to start by activate its camera on load in other to detect a face or an object.

Table 1.0:

S/N	Activities	Layouts
1	Camera Activity	camera_activity.xml
2	Detector Activity	dectector_activity.xml
3	Main Activity	main_activty.xml
4	Camera Connection Fragment	tfe_od_main_activity.xml
5	Legacy Camera Connection Fragment	tfe_od_layout_bottom_sheet.xml
6	Firstclass	Activity_first_class
7	Main3Activty	Activtity_main3

The table above contain a list of both activity and layout used throughout the entire project. An activity is written in only Java programming language, as well as a class in android studio, for a layout it is written in Xml.

4.3 Implementation Result

After the entire coding, error handling and debugging, here are the result of the app working properly on a mobile device.

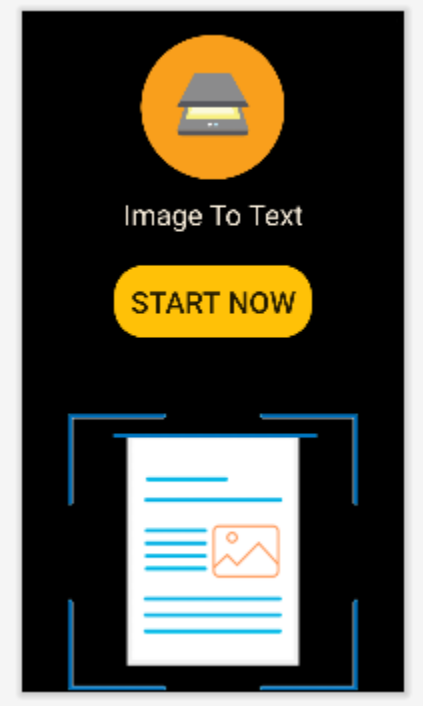


Image To Text
START NOW

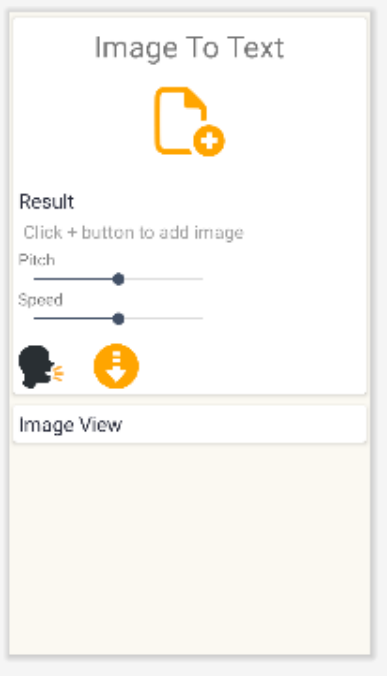
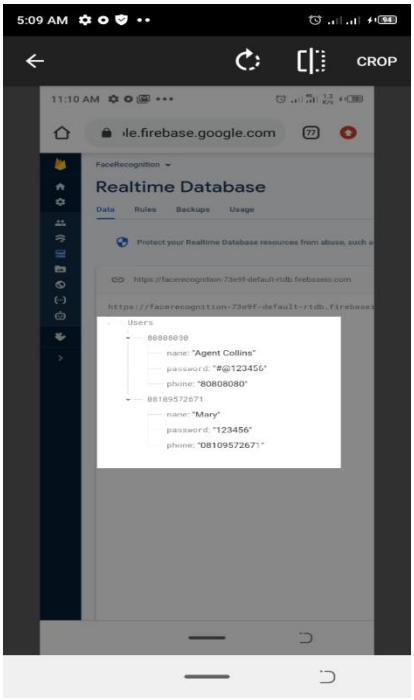
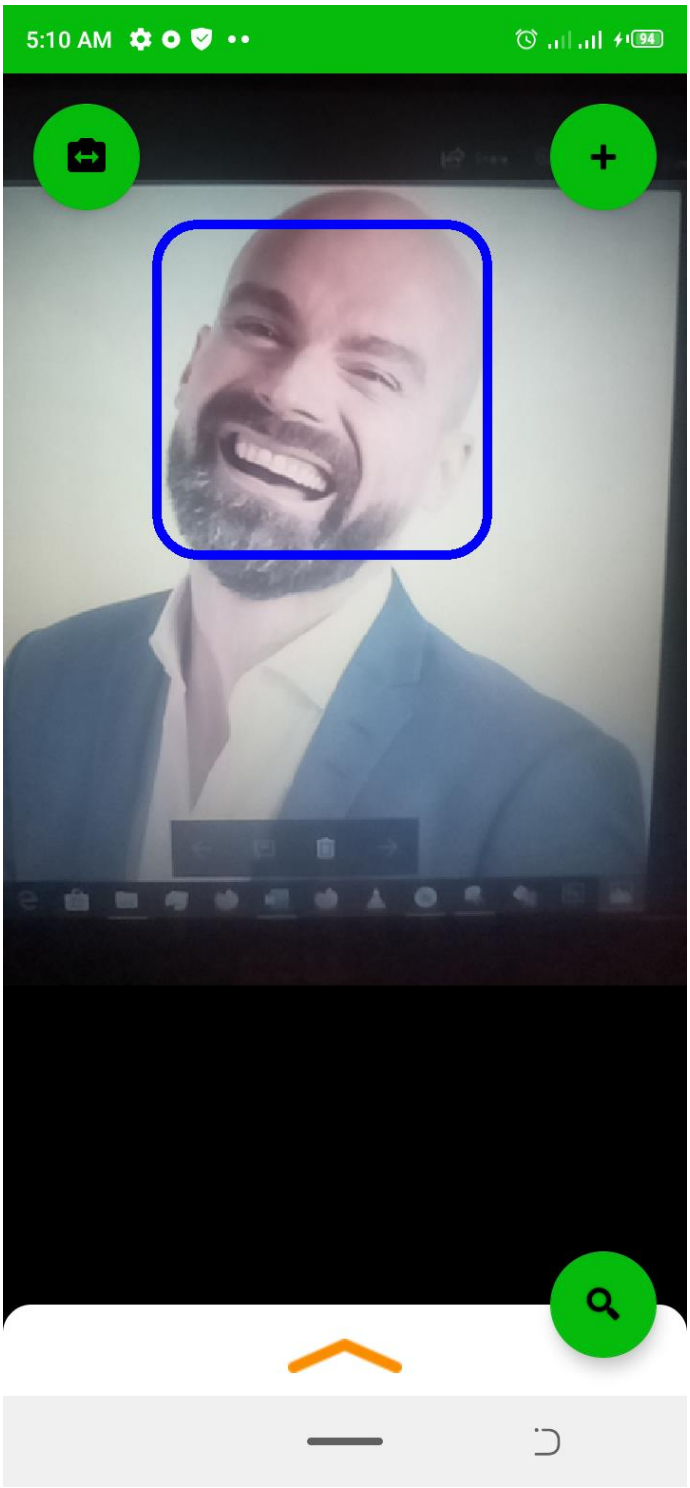


Image To Text
Result
Click + button to add image
Pitch
Speed
Image View


5:09 AM
CROP
Realtime Database


5:10 AM
Detected Text
Users
80808080
name: "Agent Collins"
password: "#@123456"
phone: "80808080"
08109572671
name: "Mary"
password: "123456"
phone: "08109572671"
Image Preview



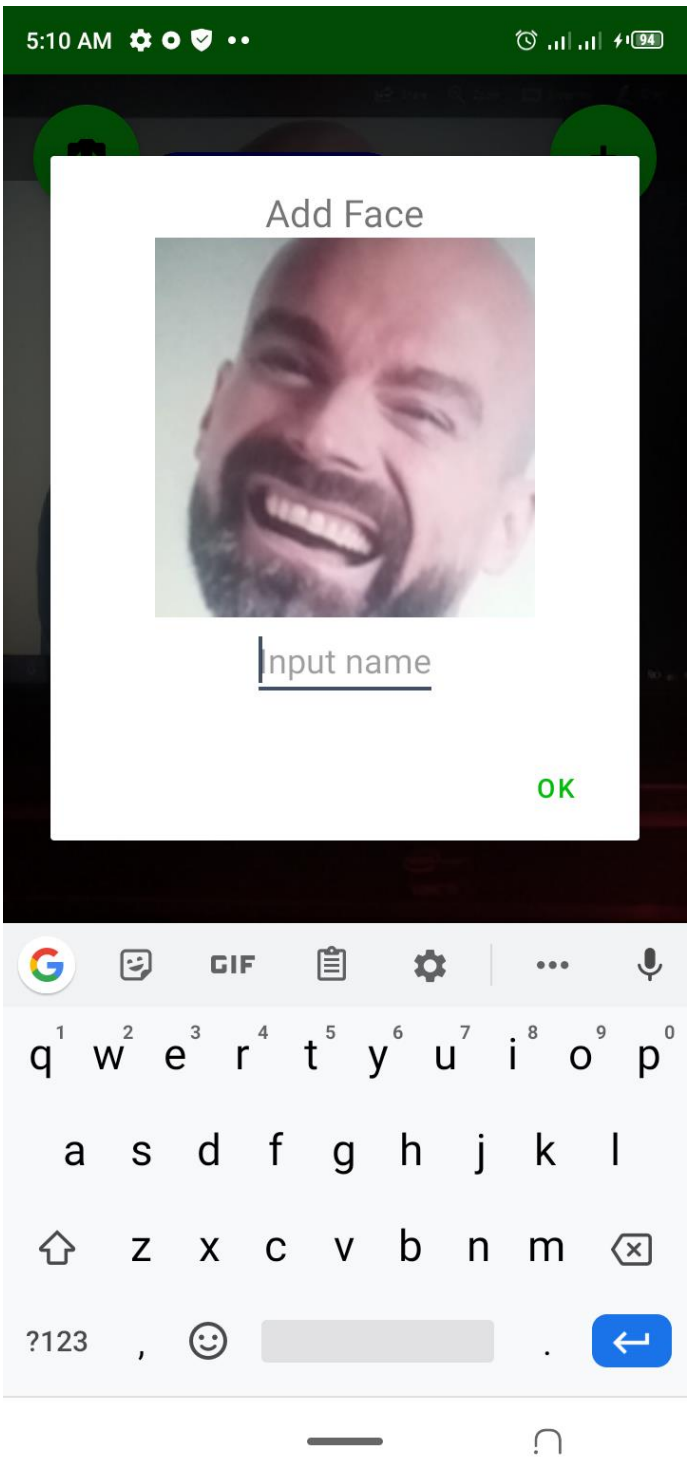


Fig 11: Face detection

Fig 12: Face being captured

In fig 11, the camera by launch detects a face and put it on auto focus mode, on the auto focus mode, it remains on the detected face as far as the camera is on and is focused on the face, in Fig 12, the face is captured and stored in temporary database for a short period of

time. The captured face is given a name, which will be used to remember it when next the camera sees that particular face.

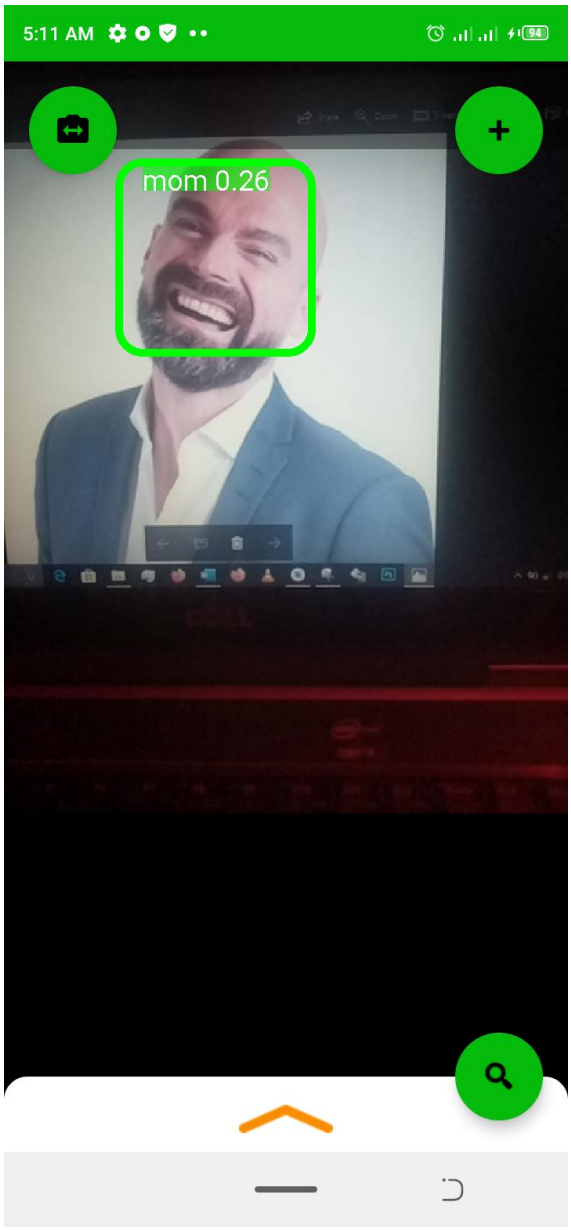


Fig 13: A detected and recognized face

4.4 Summary of Result

The idea and aim of this project are to build a face detection and face recognition as well as an optical character recognition (OCR) application using android studio and other framework, finally I can say that this application and project has reached its main purpose as stated in Chapter one.

After building and running the application for the first time, here are the of result that was seen:

1. Automatic face detection and recognition for the FRS (facial recognition system).
2. Optical character recognition (OCR)

CHAPTER FIVE

CONCLUSION

5.1 Conclusion

This paper presents a comprehensive survey of AI features ranging from, face recognition and Optical character recognition (OCR) various challenges in the last 10 years. This literature analysis showed continuously increasing interest in the field of face recognition. During training some well-known problems such as pose, facial expression, illumination, occlusion, different facial features etc. have attained a lot of attention in the research community of computer vision and pattern recognition. Various types of techniques have been proposed to compensate to all these challenges but still there are some unsolved challenges, so there is a scope of optimization. All these analyses will give a right direction to the researcher in future to resolve the unsolved challenges.

5.2 Problem encountered/Limitations

Below are the list of issues or problem I encountered when making the mobile application of face detection and recognition:

1. TensorFlow installation

2. Gradle version control

3. Internet connection due to poor services

4. Google vision integration problem

5. 3 Recommendation/suggestion for future work

I strongly recommend that more research should be carried out in this very project, as further development will strongly improve AI awareness and of a more benefit for companies, industries and small-scale business to start up with.

References

- Definition of AI as the study of intelligent agents, drawn from the leading AI textbooks.
- Dasiopoulou, Stamatia, et al. "Knowledge-assisted semantic video object detection." IEEE Transactions on Circuits and Systems for Video Technology 15.10 (2005): 1210–1224
- Ling Guan; Yifeng He; Sun-Yuan Kung (1 March 2012). Multimedia Image and Video Processing. CRC Press. pp. 331–. ISBN 978-1-4398-3087-1.
- M. Turk, A. Pentland: Face Recognition using Eigenfaces, Conference on Computer Vision and Pattern Recognition, 3 – 6 June 1991, Maui, HI , USA, pp. 586 – 591.
- S. Gupta , O.P. Sahoo, A. Goel, R. Gupta: A New Optimized Approach to Face Recognition using EigenFaces, Global Journal of Computer Science and Technology, Vol. 10, No. 1, April 2010, pp. 15 – 17.
- V. Perlibakas: Face Recognition using Principal Component Analysis and Wavelet Packet Decomposition, Informatica, Vol. 15, No. 2, 2004, pp. 243 – 250.
- V. Perlibakas: Distance Measures for PCA-based Face Recognition, Pattern Recognition Letters, Vol. 25, No. 6, April 2004, pp. 711 – 724.

- https://en.wikipedia.org/wiki/Artificial_intelligence#cite_note-Definition_of_AI-1
- http://www.iti.gr/~bmezaris/publications/csvt05.pdf
- https://books.google.com/books?id=eTfNBQAAQBAJ&q=%22object+detection%22+%22image+annotation%22&pg=PA331
- http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.