AMRITA SCHOOL OF ENGINEERING, BENGALURU CAMPUS

ELECTRONICS AND COMMUNICATION ENGINEERING

15ECE386 | OPEN LAB

PROJECT REPORT:
Human PokéDex

Submitted by

Registration Number	Name
BL.EN.U4ECE18172	Vikas Reddy S.
BL.EN.U4ECE18176	Vrushali Yegambhatla
BL.EN.U4ECE18182	Yash Rajesh Umale

Faculty in charge

Dr. N. Neelima
Assistant Professor (Sr. Gr.), ECE

Dr. T. K. Ramesh
Associate Professor, ECE

Contents

Chapter 1
Introduction

1.1 Abstract

Automation and autonomous systems are among the few powerhouses of innovation that drive entire domains towards advancing further in leaps and bounds. Great technological innovations can be attributed to tasks that are made easier and more perceptible by automation, and **artificial intelligence** is here to make these automated systems smart enough to perform their tasks with due diligence and the power of decision-making, thereby greatly reducing human intervention in redundant processes.

Our project follows the aforementioned ideals: building a product to minimize manual labour (both physical and mental) for tasks that can be seamlessly automated and processed, while solving the main problem statement at hand.

The main aim of the project focusses on **integrating the existent campus management system** with the **physical CCTV network on-campus** and further enabling the collective system to autonomously provide surveillance in the truest sense of the word by employing **deep learning techniques for smarter college management and surveillance.**

Video classification is a computer vision problem that was presented with the purpose of being able to automate classification tasks concerning real-time live video. Given the fact that the problem is considerably recent, there still exist quite a lot of gaps left to be discovered. Nevertheless, its applications are becoming largely varied, starting from only detecting the type of sports or daily activity that is happening on the scene, to actual health and security problems, to name a few.

It is an interesting fact to note that although surveillance cameras play an important role as a part of services to ensure the safety of citizens, they're plain video-providing entities with no smart decision-making mechanisms of their own. Due to the growth of image/video data collected from surveillance cameras, automated video analysis has become necessary in order to detect automatically abnormal events.

With our work, CCTV footage will find more meaning than just a video stream; cameras can autonomously trigger actions to help curb crime real-time, and the integrated system will result in greater convenience to everyone on campus.

1.2 Objectives

Our project aims at promoting safety on campus by automating the task of monitoring and reporting crimes by assigning the responsibility of detecting criminal or abnormal activity to a system which is well-versed in **deducing patterns** that distinguish criminal activity from normal activity.

In addition to detecting abnormality from footage, the vast CCTV network intertwined with the campus management system can be used for further implementations:

- Detecting crimes in a footage fed from a camera and recognizing people involved in the crime
- Enabling a student tracker system
  (Since CCTVs can now recognize faces, the vast CCTV network can maintain timestamps on a student's whereabouts at any given instant of time)
- A one-stop app which
  - leverages the same face recognition model from CCTVs to recognize criminals from a mobile phone
  - provides data from student tracker log in order to find the whereabouts of students/professors
  - retrieves relevant data on students
    (recognized by face or from a dropdown list)
  - receives alert notifications from the nearest CCTV camera witnessing a crime
  - stays in sync with the database linked with the CCTV network for better management of complaints

In order to fulfil these objectives, a greater objective was to generate video classification inferences from a normal 2D CNN, along with a minor objective of recognizing faces using simple vector-based classification algorithms.

1.3 What's New?

Traditional surveillance systems deal with a plethora of short-comings that do not sync well with the current support facilities available in today's day and age. One of the most notable flaws in vanilla surveillance systems is the **heavy dependence on an attentive supervisor monitoring footage** and ensuring that any abnormal activity is duly noted and taken care of.

In a bid to reduce manual intervention and labour, the project not only eliminates the need for additional supervision, but it also:

- Immediately detects the occurrence of a crime and takes note of people involved
- Triggers actions that initiate mitigation measures on the crime scene, real-time
- Maintain a log of complaints and tracked movements of students on campus with specific timestamps
- Builds an ecosystem that makes campus management, information retrieval and general safety a lot more accessible.

1.4 Project Subdivisions

The project has been divided into separate sub-problems, each dealing with a mutually exclusive aspect of the net project.

The subdivisions are as follows:

- Develop a robust face recognition model to accurately classify faces from an incoming video stream. Compare and contrast different face recognition models based on accuracy and frequency of false positives.

- Build an Android application that handles
    - registration of faces (further on point 3)
    - display of tracker log from CCTV networks
    - face recognition and student indexing on-device
    - data retrieval of concerned subject (via face recognition or otherwise)
    - synchronization with database and Cloud Storage location used by the campus management system and CCTV network

- Collect face datasets from the Android app and create a system that
    - asynchronously performs **dataset augmentation, face re-alignment** and further pre-processing on datasets
    - trains the concerned face recognition models over the newly generated datasets **over the Cloud**
    - pushes the updated model to the Android application as a package update

- Develop and train a CNN (either pre-trained over an activity classification dataset or related datasets) over the **UCF-Crimes dataset** for performing **video classification** (after cleaning and annotating datasets according to their corresponding labels).

- Maintain the backend's structural integrity for asynchronous calls to database, Cloud Storage and Cloud ML Vision.

1.5 Tech Stack, Services, Languages and Frameworks

Languages
• Python • Java • XML • Shell Script

Services
• Google Cloud Platform • Firebase Storage • Cloud Firestore • Docker and Kubernetes

Frameworks
• TensorFlow • Gradle • Android SDK

- Visual Studio Code
- Android Studio
- OpenCV
- Torch
- Keras

- CamXAPI
- WayScript
- Jupyter Notebooks
- IP Camera and RTSP URLs

1.6 Project Outcomes

The outcomes of the project are listed below as follows:

- Data continuously logged to a database on a student's whereabouts
- Easier student indexing and campus management
- Efficient anomaly/unrest detection and mitigation
- Accessibility (and clearance level-based abstraction) to relevant on-campus data
- Automation of complaint and grievance management
- Android application to alert users of unrest in their proximity, access data on colleagues or students, lodge/ solve complaints or automatically register a crime when using the app's face recognition feature on a crime scene.

—

CHAPTER 2
Description of Tech Stack

This chapter deals with presenting the conceptual fundamentals of image and video classification with deep learning, and describing the software, frameworks, tools, languages and the mathematics involved behind the superficial concepts implemented.

In the following subtopics, we will cover in detail the following:

- **Software Used:**
  - Jupyter Notebooks and Command Line
  - Visual Studio Code, Extensions and Remote Scripting
  - WayScript with containerized applications
  - Flask and Docker
- **Services Used:**
  - Google Cloud Platform
  - Firebase Services (Cloud Firestore, Cloud Storage, Crashlytics)
- **Languages:**
  - Python
  - Java
  - XML
- **Frameworks:**
  - TensorFlow
  - Gradle and Android SDK (for v27: Oreo)
- **Concepts and Mathematical Basis:**
  - Basic Linear Algebra and Classification Algorithms
  - Neural Networks and CNNs
  - Feature Extraction and Classification

2.1 Software Used

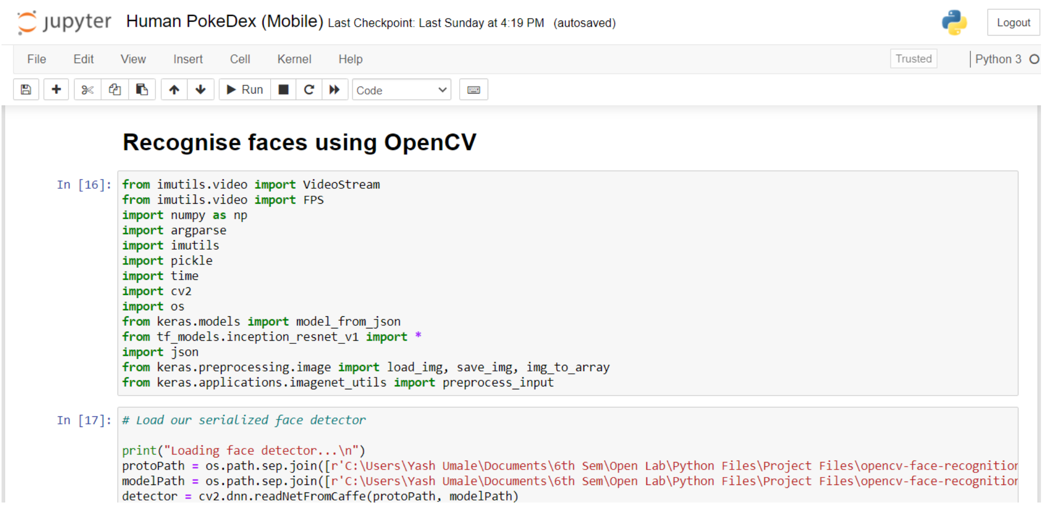
2.1.1 Jupyter Notebooks and Command Line

Jupyter Notebook is a web-application developed by a non-profit open source community called Project Jupyter.

This application allows developers to create and share documents that contain live code, equations, visualizations and narrative text. It is used to execute Python code on a need-to-run basis, and is greatly effective in testing and debugging sections of code before pushing the codebase to production.

Uses include: Data cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning etc.

Our project used Jupyter Notebooks (and a widely used variant of Jupyter Notebooks: Google Colaboratory) to execute pattern recognition algorithms on a collaborative basis. **Google Colaboratory** was used to initiate Python runtimes on virtual machines hosted on the Cloud, so as to leverage their compute power and resources to execute processes such as **training** and **dataset augmentation faster.**


jupyter
Human PokeDex (Mobile) Last Checkpoint: Last Sunday at 4:19 PM (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code

Recognise faces using OpenCV

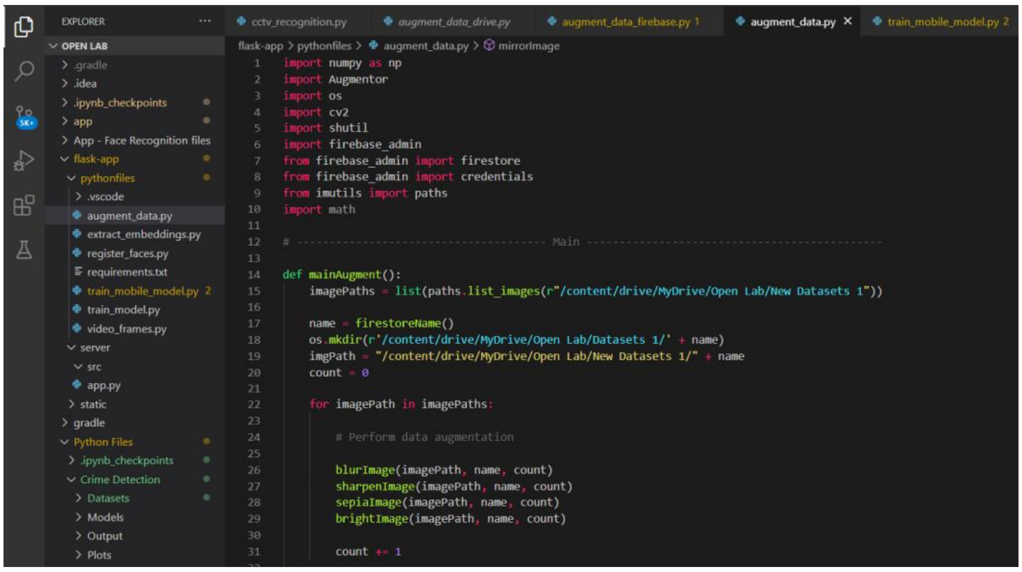
```
In [16]: from imutils.video import VideoStream
         from imutils.video import FPS
         import numpy as np
         import argparse
         import imutils
         import pickle
         import time
         import cv2
         import os
         from keras.models import model_from_json
         from tf_models.inception_resnet_v1 import *
         import json
         from keras.preprocessing.image import load_img, save_img, img_to_array
         from keras.applications.imagenet_utils import preprocess_input

In [17]: # Load our serialized face detector

         print("Loading face detector...\n")
         protoPath = os.path.sep.join([r'C:\Users\Yash Umale\Documents\6th Sem\Open Lab\Python Files\Project Files\opencv-face-recognitior
         modelPath = os.path.sep.join([r'C:\Users\Yash Umale\Documents\6th Sem\Open Lab\Python Files\Project Files\opencv-face-recognitior
         detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
```

Fig. 2.1: Jupyter Notebooks

2.1.2 Visual Studio (Extensions and Terminal)

Visual Studio is a streamlined code editor redefined and optimized for building, debugging and shipping web and cloud-based applications. With support for easier debugging with breakpoints, call stacks and an interactive console, along with add-ons like version control and extensions for efficient development, VSCode provides the tools needed by a developer for a quick code-build cycle and leaves complex workflows to IDEs.

Extensions on VSCode (such as IntelliSense, JVM, Docker Boilerplate and Debuggers) help in going beyond syntax highlighting to provide intelligent solutions for optimized code, function definitions and imported modules.



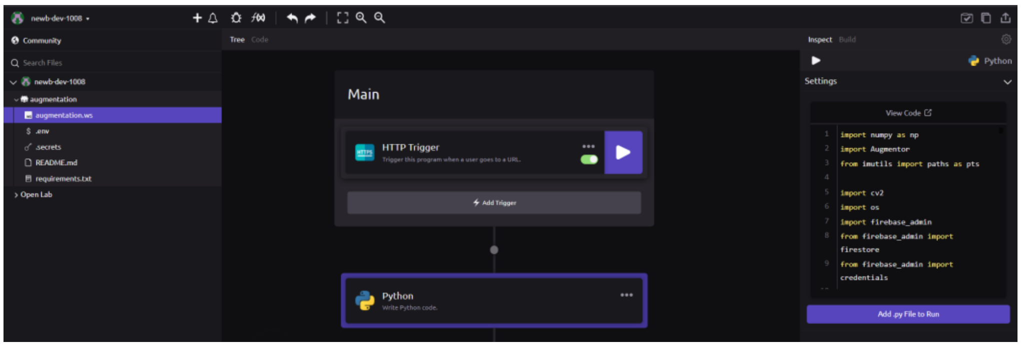
```python
import numpy as np
import Augmentor
import os
import cv2
import shutil
import firebase_admin
from firebase_admin import firestore
from firebase_admin import credentials
from imutils import paths
import math

# ------------------------------------ Main ------------------------------------------

def mainAugment():
    imagePaths = list(paths.list_images(r"/content/drive/MyDrive/Open Lab/New Datasets 1"))

    name = firestoreName()
    os.mkdir(r'/content/drive/MyDrive/Open Lab/Datasets 1/' + name)
    imgPath = "/content/drive/MyDrive/Open Lab/New Datasets 1/" + name
    count = 0

    for imagePath in imagePaths:

        # Perform data augmentation

        blurImage(imagePath, name, count)
        sharpenImage(imagePath, name, count)
        sepiaImage(imagePath, name, count)
        brightImage(imagePath, name, count)

        count += 1
```

Fig. 2.2: Visual Studio Code

2.1.3 WayScript and Flask server (with Docker container)

A rapid scripting platform for developers, WayScript allows developers to run Python in the cloud, and seamlessly integrate with your favourite APIs. It functions like a web host for Python files, without specifically requiring containers.

We used WayScript in our project to trigger a Python script for dataset augmentation when a user registers his/her face from the Android app. After dataset augmentation, another Python script is triggered to train the SVM on new embeddings and update the app with the new model.



```python
import numpy as np
import Augmentor
from imutils import paths as pts

import cv2
import os
import firebase_admin
from firebase_admin import
firestore
from firebase_admin import
credentials
```

Fig. 2.3: WayScript Interactive Menu

Flask is a lightweight web-application framework designed to scale complex applications quickly, using nothing more than a wrapper to run containers which house Python scripts. It provides simplicity, flexibility and fine-grained control over the local-host server which runs the Python file.

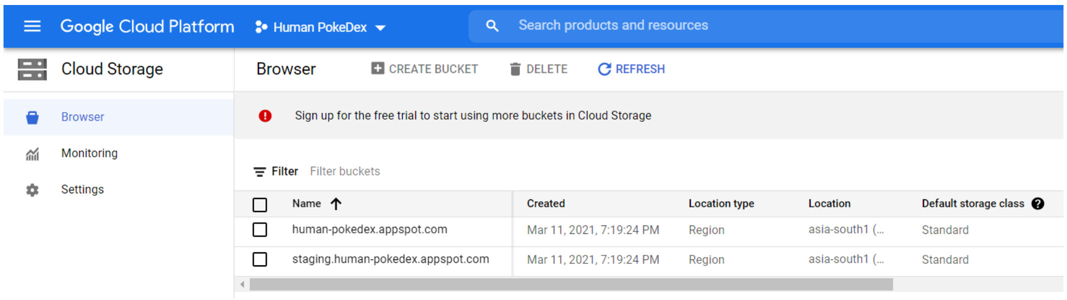
Docker is a tool used to ease deployment and run applications on a remote computer with the help of **containers.** Docker and Flask were used as alternatives to WayScript, as WayScript had limitations on installing new Python libraries.

2.2 Services Used

2.2.1 Google Cloud Platform

Google Cloud Platform is a suite of cloud computing services that run on the same infrastructure used by Google for their end-to-end applications. Developers can access IaaS (infrastructure as a service) using command-line tools, SDKs from GCP or from the virtual terminal in the Google Cloud console.

We used Google Cloud to run a virtual machine/instance with substantial GPU and CPU power in order to train our video classification model faster for crime detection.



Sign up for the free trial to start using more buckets in Cloud Storage

Name	Created	Location type	Location	Default storage class
human-pokedex.appspot.com	Mar 11, 2021, 7:19:24 PM	Region	asia-south1 (...	Standard
staging.human-pokedex.appspot.com	Mar 11, 2021, 7:19:24 PM	Region	asia-south1 (...	Standard

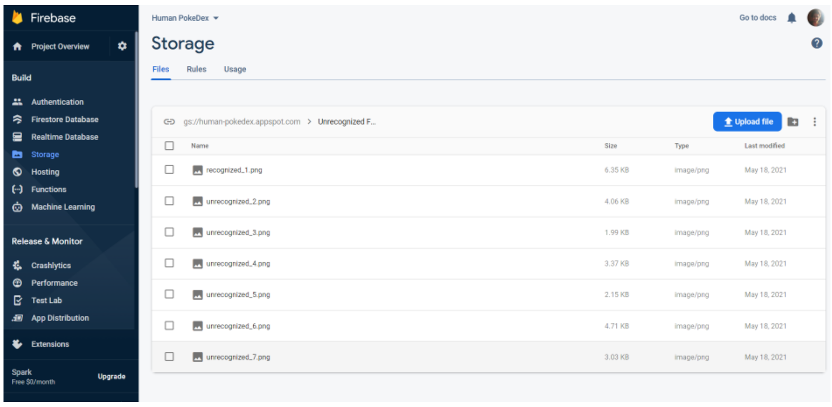
Fig. 2.4: Google Cloud Console

2.2.2 Firebase

Firebase is a platform built for scaling mobile and web applications, and providing backend services so that developers can focus on deployment with minimal maintenance of databases, storage and API callbacks.

We used:

- **Cloud Firestore** as an integrated backend for CCTV network as well as the Android application
- **Firebase Storage** for storing new datasets (Bitmaps), models, unrecognized images and files required for execution of Python scripts on Docker
- **Crashlytics** to keep track of faults/crashes reported by the app on any end user's device
- **Firebase ML Kit** to leverage advanced detection techniques for obtaining bounding boxes that distinguish faces from the rest of the image.



gs://human-pokedex.appspot.com > Unrecognized F...

Name	Size	Type	Last modified
recognized_1.png	6.35 KB	image/png	May 18, 2021
unrecognized_2.png	4.06 KB	image/png	May 18, 2021
unrecognized_3.png	1.99 KB	image/png	May 18, 2021
unrecognized_4.png	3.37 KB	image/png	May 18, 2021
unrecognized_5.png	2.15 KB	image/png	May 18, 2021
unrecognized_6.png	4.71 KB	image/png	May 18, 2021
unrecognized_7.png	3.03 KB	image/png	May 18, 2021

Fig. 2.5: Firebase Console

2.3 Languages

As the project has a multi-faceted approach with different components and scripts, we've used a couple of high-level languages along with markup languages for frontend purposes.

2.3.1 Python

Our project uses Python for the following purposes:

- Implementing face detection and recognition
- Training models for image and video classification
- Dataset augmentation
- Running the Flask server

Since Python is an interpreted language, multithreading is not entirely supported by the language. Hence, distributed systems are required in order to execute face recognition and crime detection simultaneously.

2.3.2 Java

Java v11 is used for Android App Development and implementing a similarity classifier for the application.

CameraXAPI (responsible for input stream from on-device camera module) is interfaced using Java and the frontend of the application is made responsive and fluid due to code written in the language.

Furthermore, Java on Android is known to be highly efficient as it automatically triggers such tasks over a background thread and executes API calls and database inferences asynchronously. Java's garbage collection and memory leak management makes it an ideal choice for Android App Development as well, in contrast to PWAs (progressive web-apps) developed using web technologies like React Native and Angular.js.

2.3.3 XML (Extended Markup Language)

XML primarily focusses on defining the user interface for the Android application. Inherently, we use certain libraries (Material Design, AndroidX, Jitpack.io etc.) to enhance the UI while ensuring optimal performance which enables the application to perform in sync with the Java codebase.

2.3.4 Shell/ Bash Script

A shell script is a list of commands in a computer program that is run by a Unix shell. These commands are required to trigger a chain of Python scripts, version control, etc.

2.4 Frameworks

2.4.1 TensorFlow

TensorFlow is an end-to-end open source platform built for simplifying and optimizing tasks that involve Machine Learning and Neural Networks. It is built on C++, thereby ensuring faster computation, multithreading, efficient memory management etc.

We used TensorFlow's framework to reach our objective using different models which implemented Keras (using TensorFlow backend). For face recognition, we tried our results with the **InceptionResNetV1** model and the **VGGFace2** model, both with TensorFlow backend.

For crime detection, we used a ResNet with TensorFlow backend for training and predicting criminal activity.

2.4.2 Gradle

Gradle is a build system used to automate building, testing and deployment. A build.gradle file automates all tasks related to dependency management and downloading requisite libraries for the native Android app.

Gradle runs on the JVM (Java Virtual Machine) and can be extended to build models as well. It uses the Groovy language to write all its dependency scripts.

Our project uses Gradle to sync the Android app files with the requisite dependencies and acquire user permissions for access to modules on the device such as camera, cellular network, Wi-Fi, etc.

CHAPTER 3

Design and Development

3.1 Prerequisites

Our project involves advanced computer vision applications such as video classification and face recognition, and these techniques require the knowledge of linear algebra, vector calculus, neural networks, convolutions and convolutional neural networks among many other mathematical fundamentals. These will be covered in the upcoming sections.

3.2 Embeddings, Norms and some Linear Algebra

Embeddings can be considered as compact entities that quantify faces and extract their distinct features from photos. They're generally used to represent faces with a 128-d vector in a 128 unit hypersphere.

Each embedding is essentially a vector with its corresponding position in the hypersphere. Clusters of these embeddings represent similar faces or classes huddled together in the 128-d space. In a nutshell, **an embedding is a mapping of discrete – categorical – variable to a vector of continuous numbers.** They're low dimensional, learned continuous vector representations of discrete variables.

These embeddings are useful because:

- They help in finding nearest neighbours in the embedding space, which can work for making recommendations based on user interests
- They reduce dimensionality of categorical variables and meaningfully represent categories in a transformed space
- They can be used as inputs to a machine learning model for a supervised task
- They provide better visualization of concepts and relations between categories

To compute these embeddings, we use a CNN which can accurately detect and compute an appropriate representation for the corresponding face.

In order to gauge similarities and differences between embeddings, we need some defined metrics. These metrics can be derived from distances between two or more embeddings (**L2 Norm**) or the angle of deviation between two embeddings from a reference vector (**cosine similarity**).

The **L2 Norm** returns the Euclidean distance between two vectors. This norm can be used to determine the distance between embeddings to further measure the dissimilarities between computed embeddings.

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{N} |x_i|^2\right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}$$

Cosine Similarity is the cosine of the angle between two n-dimensional vectors in an n-dimensional space.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Along with these metrics, it is important to consider the implications of **matrix multiplications, gradients, inner products and normalization.**

3.3 Theoretical Foundation of CNNs

A CNN is a type of artificial neural network that uses **convolutional layers** to filter inputs for obtaining useful information for the network such as edges and shapes among other features. This type of multi-layered network is widely used to recognize visual patterns such as characters, symbols, figures, etc. from pixel images.
A CNN is commonly compounded with many repeated layers and **activation functions** such as **sigmoid, tanh, relu (rectified linear unit), pooling layer** and a **fully connected layer** as shown in the figure.

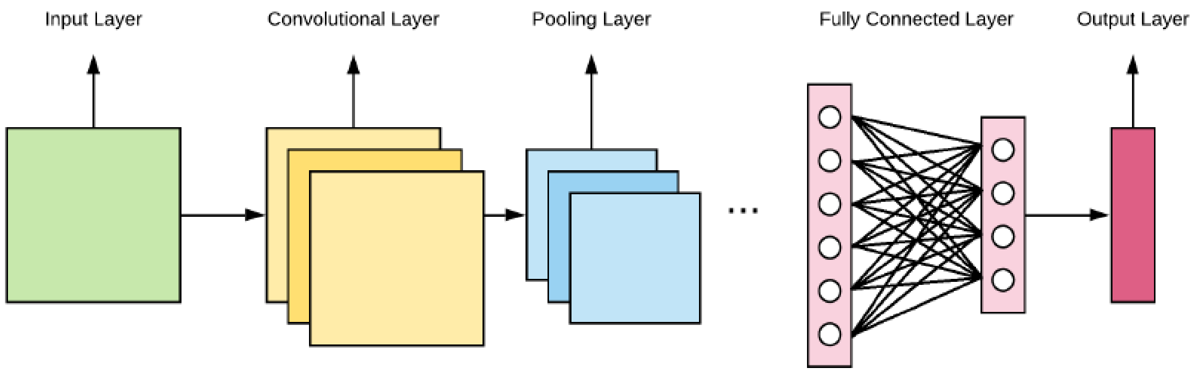


Fig 3.1: General Architecture of a CNN

- **Input Layer:**
  Contains image data represented by a 3D matrix. This data needs to be converted into a single column of dimension **(width × height × number of channels)**

- **Convolutional Layer:**
  Uses convolutional filters (called kernels) with a defined size to go over the entire input data and perform convolution operations.
  The filter slides over the data with a stride **S**. This process is done to learn and detect patterns from previous layers.
  The result is a feature map.

- **Pooling Layer:**
  Referred to as the **downsampling layer**, it is used to reduce spatial dimensions but not the depth of a CNN.

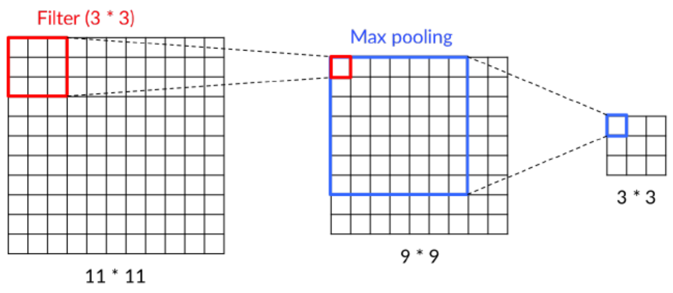


Fig. 3.2: Convolution and Pooling Layers

- **Activation Function:**
  Normally present after the pooling or fully connected layer, its objective is to apply a non-linear transfer function to encode patterns through transformations.
  Few common activation functions include **sigmoid function, tanh function** and the **relu (rectified linear unit) function.**

$$f(z) = \left\{ \begin{matrix} \frac{1}{(1+e^{-z})}\ ,\ \ f(z) = sigmoid \\ \frac{(e^{z}-e^{-z})}{(e^{z}+e^{-z})},\ \ f(z) = tanh \\ \max(0, z)\ ,\ \ f(z) = relu \end{matrix} \right\}$$

- **Fully Connected Layer:**
  A fully connected layer is a function mapping R^m to R^n. Each neuron, connected to each neuron in the previous layer has its corresponding weights. The input of the fully connected layer corresponds to the output of the of the last pooling/ convolution. Each of the FC layers perform calculations based on a specific activation function.
  FC layers are important for learning the features presented by pooling and convolution layers, thereby playing a major role in the classification stage.

- **Output Layer:**
  The final layer of the network (and the FC layer), this layer contains the activation functions used to obtain the probabilities of the given input belonging to a particular class with its corresponding label and output neurons.

3.4 Face Recognition

In applications of Computer Vision, there exist slight differences between terms that are often used interchangeably: face detection, face recognition and face verification. While they involve similar tech stack, the end result and objective are different.

Face Detection is the technique of detecting and returning the location of a face inside the frame from a given photo or video stream.

Face Verification takes it a step further and checks if the detected face holds any resemblance with another face stored in memory. This is done by gauging the similarity between the two faces using distance metrics like the **L2 norm** or the **cosine similarity**.

Finally, **face recognition** cumulatively involves both the above techniques to extract prominent features from the detected face and identify who the person is from a set of labels obtained from the dataset which the model is trained on.

In order to extract features from incoming frames, we need to first understand the working principle of a **convolutional neural network (CNN),** along with some linear algebra.

In a nutshell, here is our proposed solution for face recognition:

- Detecting faces
- Computing 128-d face embeddings to quantify a face
- Train a Support Vector Machine (SVM) on top of the embeddings
- Recognize faces in image and video streams

All of these tasks will be achieved with OpenCV, enabling us to obtain a pure OpenCV face recognition pipeline. This is achieved in two key steps:

- Applying **face detection**, which detects the presence and location of the face in an image without identifying it
- Extracting a 128-d feature vector (embedding) that quantify each face in an image

The model responsible for quantifying each face in an image is from the **OpenFace Project**, a Python and Torch implementation of face recognition and deep learning.

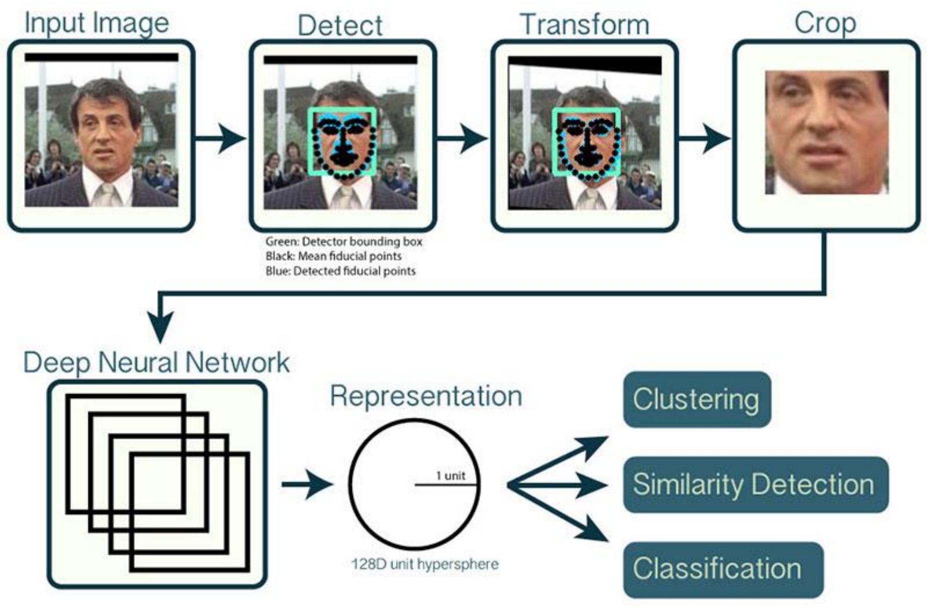


Fig 3.3: An overview of the OpenCV face recognition pipeline

First, we input an image or a video frame to the pipeline, which applies face detection to detect the location of the face in the image.

Face detection is performed by a Caffe model in OpenCV’s DNN module.

OpenCV’s deep learning face detector is based on the **Single Shot Detector (SSD) framework** with a ResNet base network. A single shot detection refers to a technique wherein the model needs only a single shot to detect multiple objects within the image. It discretizes the input image into certain bounding boxes around regions with feature maps of high confidence and generates multiple boxes around such region maps. **The confidence for each of these boxes** is calculated and the box dimensions are adjusted to obtain the best fit for detection.

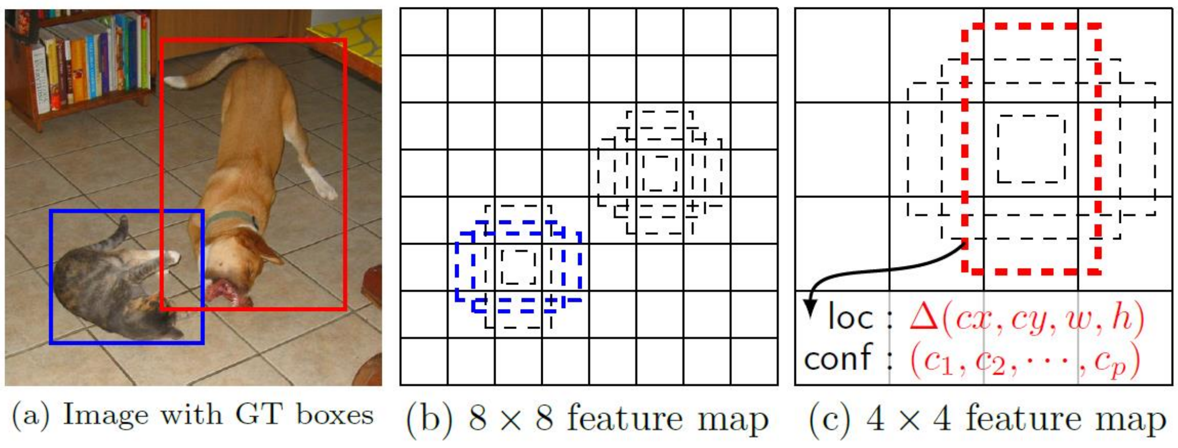


Fig 3.4: SSD – Multiple Bounding Boxes for Localization and Confidence

Additionally, we can compute facial landmarks (mouth, right/left eyebrows, eyes, nose, jawline) using **dlib**, which will futher enable us to preprocess the images and perform **face alignment** on datasets for better results.

Face alignment is the process of

- Identifying the geometric structure of the face
- Attempting to obtain the canonical alignment of the face based on translation, rotation and scale.

After applying face alignment and cropping, we pass the input face through the deep neural network.

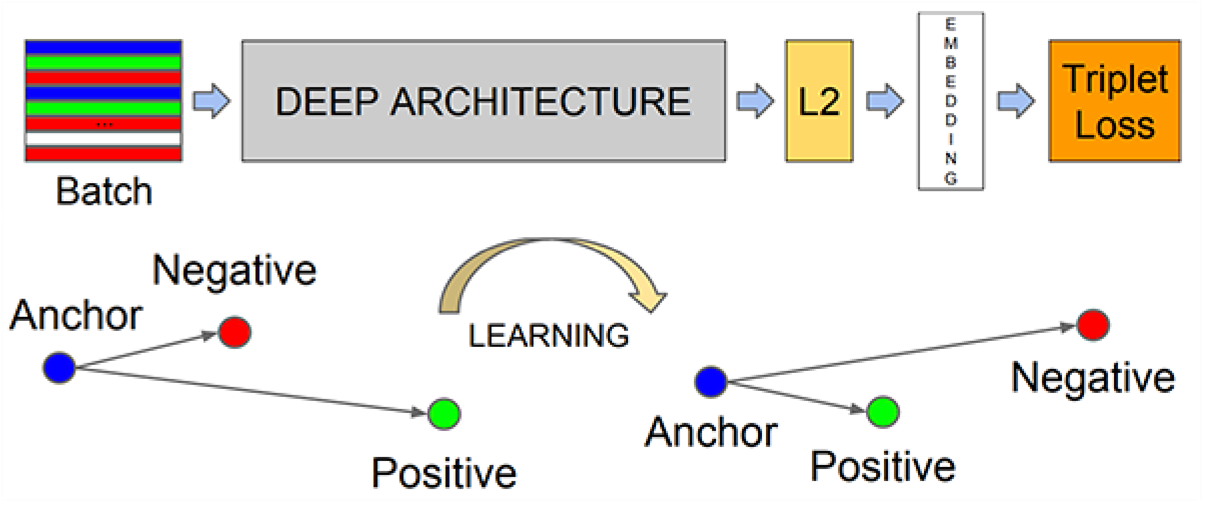


Fig. 3.5: How the Deep Learning model computes face embeddings

To train the face recognition model with deep learning, **each input batch of data must contain three images:**

- Anchor
- Positive Image
- Negative Image

The **anchor** is our current image (let's say of person A), whereas the **positive image** is also an image of person A. The **negative image** in each batch is that of a person other than A.

The point is that the anchor and positive images both belong to the same person whereas the negative image does not contain the same face.

The neural network computes the face embeddings of each of these faces and **tweaks the weights of the network** using **triplet loss** in such a way that:

- The 128-d embeddings of the anchor and the positive image lie closer
- At the same time, pushing the embedding of the negative image far away

In this manner, the network is able to learn to quantify faces and return highly robust and discriminating embeddings suitable for face recognition.

A CNN model computes the embeddings for all input images, and these embeddings are sufficiently different to train a machine learning classifier such as SVMs, SGD Classifiers, Random Forests, etc. on top of the face embeddings, and therefore obtain our face recognition pipeline.

3.4.1 How the Triplet Loss function works

An embedding in our project is represented by **$f(x) \in R^d$,** where **$d = 128$.** It embeds an image into a 128-d Euclidean space and is constrained to live on the 128-d hypersphere (i.e. norm of the embedding **$\|f(x)\| = 1$**).

Our objective is to ensure that an image $\boldsymbol{x_i^a}$ **(anchor)** of a specific person is closer to all other images $\boldsymbol{x_i^p}$ **(positive)** of the same person than it is to any image $\boldsymbol{x_i^n}$ **(negative)** of any other person.

In mathematical terms, **our objective:**

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$$

$$\forall \left(f(x_i^a), f(x_i^p), f(x_i^n)\right) \in \mathcal{T}$$

Where:

α = margin enforced between positive and negative pairs

$\mathcal{T}$ = set of all possible triplets in training set

Hence, the loss function that should be minimized by the network is:

$$\boxed{\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+}$$

3.4.2 Data Augmentation

If your neural nets are getting larger but the training sets aren't, a point in training is approached where the learning gradient hits a saturation point and vanishes. Eventually, the model hits an **accuracy wall**.

Dataset Augmentation – the process of applying simple and complex transformations like flipping and style transfer to your data – can help overcome the increasingly large requirements of deep learning models.

Deep Learning models particularly benefit from data augmentation as they try to understand relationships between the training examples provided on a pixel-to-pixel level. Hence, it needs a lot of examples to recognise and derive patterns from the data it observes.

Dataset Augmentation can multiply the effectiveness of present data.

The augmentation techniques applied in this project include:

- Flipping (both horizontally and vertically)
- Rotating
- Zooming and scaling
- Cropping
- Translating (moving along **x** and **y axis**)
- Adding Gaussian noise
- Shearing
- Skewing
- Black and White filters
- Sepia filters
- Blurring images

While most of these transformations can be obtained from fairly simple implementations like the **ImageDataGenerator** module of TensorFlow, other functions have been designed (mentioned forthwith) to perform data augmentation and upload augmented data to the Cloud for automatic training once new data has been generated.

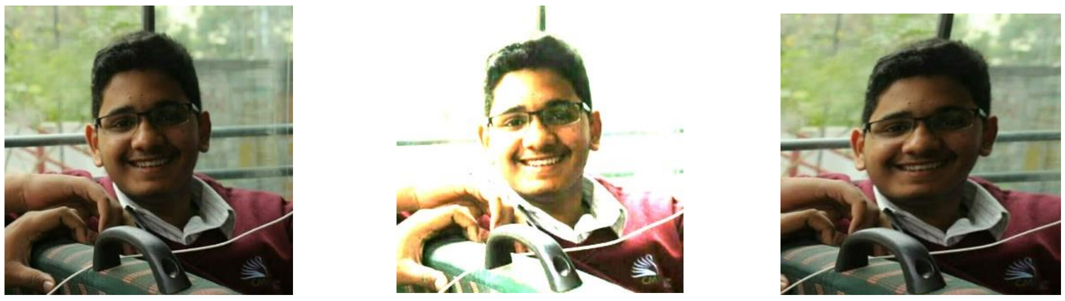


Fig. 3.7: Original Image, Sepia Image and Skewed Image

3.4.3 Hyperparameter Tuning

For training, the number and diversity of hyperparameters such as batch size, learning rate, number of epochs and number of layers are quite specific to each model. These **hyperparameters** must be taken into account while training a CNN to improve performance.

- **Learning Rate:**
  In charge to quantify the learning progress of a model in such a way that will be used to optimize its capacity. It is also considered as the most important one due to the fact that it affects the training time and accuracy.

- **Batch Size:**
  Defines the number of training samples to work before updating the model parameters. Its size must be greater than or equal to 1, and less than the training set size. It effects the **computational resources** and **time taken per epoch**.

- **Number of epochs:**
  Number of iterations the CNN works through the whole dataset. It can be a fixed number of iterations, or techniques such as **early stopping** can be employed by using a combination of training error and validation error to stop the model when the error is over a specific threshold.

- **Number of layers:**
  While going deeper within the network does not usually bring drastically changing results, in the case of CNNs the networks perform better with every additional layer due to better feature extraction.

- **Convolution Kernel Width:**
  The width of the kernel used during convolution operations influences the number of parameters in a model which in turn influences its capacity. The kernel size can be changed across the network.

3.5 Types of classification algorithms

As mentioned earlier, the OpenCV face recognition pipeline works on computing embeddings for faces detected in all images and training classic Machine Learning classifiers like SVM (Support Vector Machines), Random Forest Classifiers, SGD Classifiers etc. for predicting the label of a new face.

Our project implements SVMs to classify between embeddings as they are considered one of the best classification algorithms for reasons that are discussed below.

The SVM model is trained on the obtained embeddings and pickled (converted to ByteStream) along with a label-encoded file for each class.

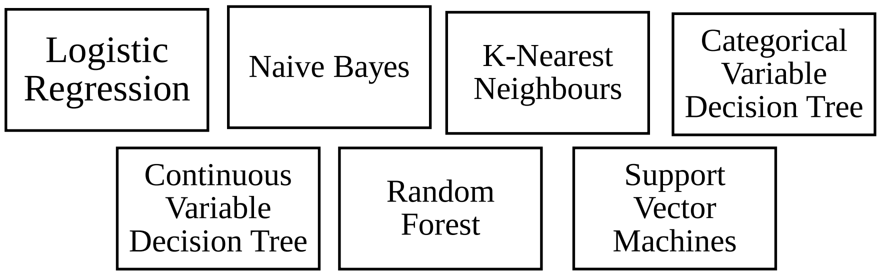


Fig. 3.8: Types of classification algorithms

- **Logistic Regression:** Logistic regression is used for binary classification. This algorithm employs a logistic function to model the probability of an outcome happening.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

  It is most useful when you want to understand how several independent variables affect a single outcome variable. Logistic regression has limitations; all predictors should be independent, and there should be no missing values. This algorithm will fail when there is no linear separation of values.

- **Support Vector Machines:** Support vector machines use a hyperplane in an N-dimensional space to classify the data points. N here is the number of features. It can be any number, but greater the dimensionality, the harder it becomes to build a model.

  The hyperplane that we choose directly affects the accuracy of the results. So, we search for the plane that has the maximum distance between data points of all classes.

  SVMs show accurate results with minimal computation power when you have a lot of features. They train on the **worst-case set of data points** to obtain the threshold required for accurately classifying the embeddings of different classes to alternating sides of the hyperplane.

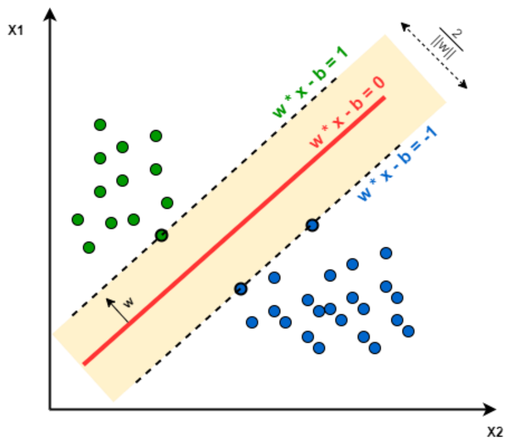


Fig. 3.9: SVM in action (2 classes)

3.6 Model Architectures (Face Recognition)

The models used for the Face Recognition component of the project have been summarised and illustrated as block diagrams. They cannot be included in the document due to their sizes which may compromise text quality within the image, and hence their respective clear PNG images have been hosted and their URLs have been listed forthwith.

3.6.1 Caffe Model (Face Detection) [Clear PNG Image]

3.6.2 OpenFace Model (Feature Extraction) [Clear PNG Image]

3.6.3 TFLite Model (Face Recognition on phones) [Clear PNG Image]

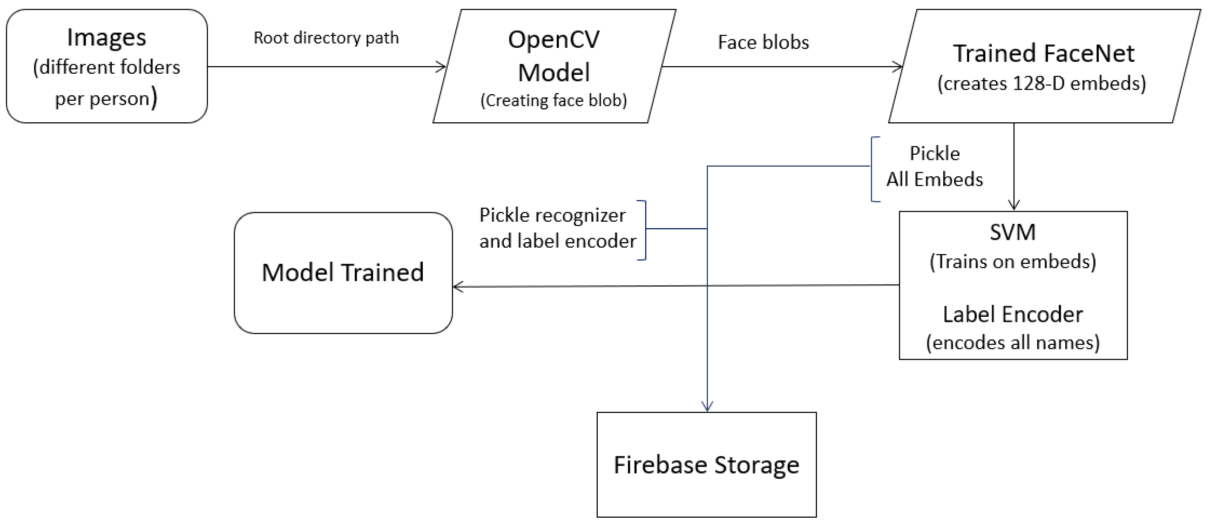

Images (different folders per person)
Root directory path
OpenCV Model (Creating face blob)
Face blobs
Trained FaceNet (creates 128-D embeds)
Pickle All Embeds
SVM (Trains on embeds)
Label Encoder (encodes all names)
Pickle recognizer and label encoder
Model Trained
Firebase Storage

Fig. 3.10: Overall Face Recognition Pipeline with OpenCV

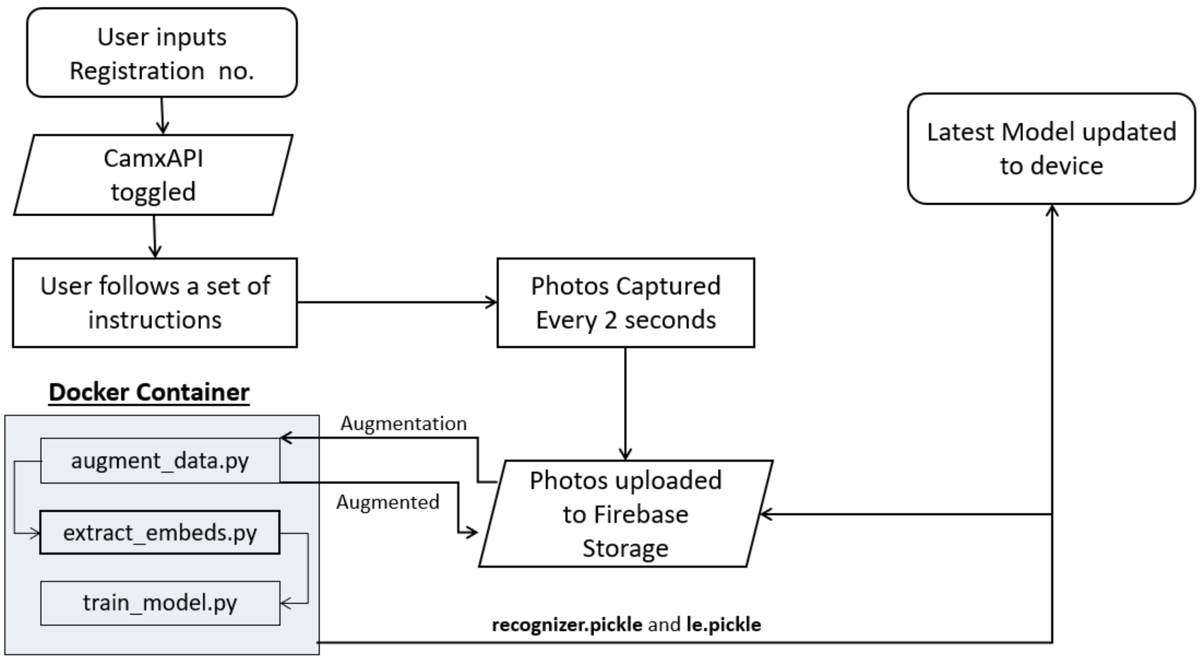

User inputs Registration no.
CamxAPI toggled
User follows a set of instructions
Photos Captured Every 2 seconds
Latest Model updated to device
Docker Container
augment_data.py
extract_embeds.py
train_model.py
Augmentation
Augmented
Photos uploaded to Firebase Storage
recognizer.pickle and le.pickle

Fig. 3.11: Flowchart depicting Face Registration from Android App

3.7 Video Classification for Crime Detection

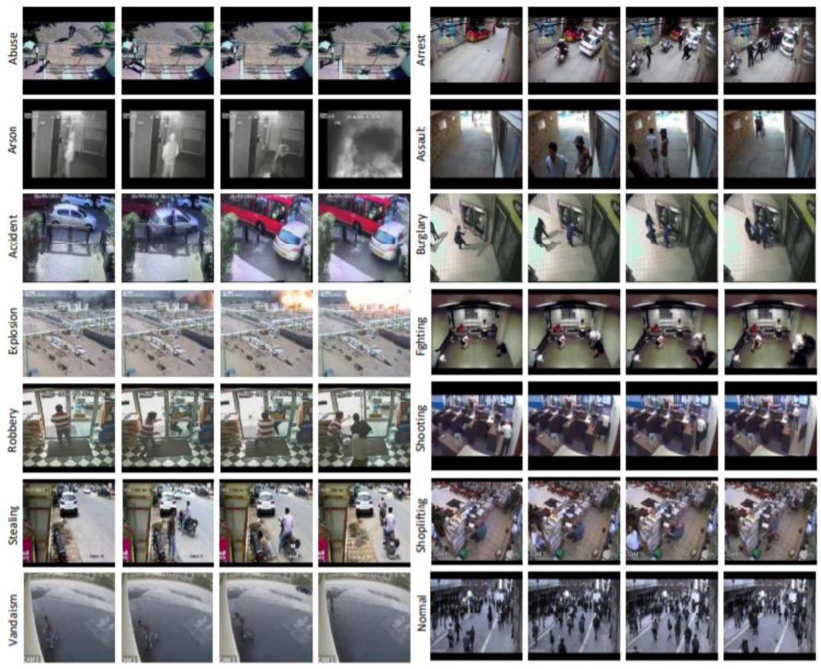
In this segment of the project, we aim towards detecting anomalies in videos of footage and normal activity. The dataset at play here used for training is the **UCF Crimes Dataset**, which is the only dataset that contains videos of diverse classes of crimes, each replete with valuable and distinctive features.

Main features of the dataset:

- It contains 13 classes in total: Abuse, Arrest, Arson, Assault, Accidents, Burglary, Explosion, Fighting, Robbery, Shooting, Stealing, Shoplifting and Vandalism.

- In total, it consists of 1900 real-world surveillance videos, all obtained under different environments and each video containing a specific realistic anomaly.

- This leads to a total of 128 hours of videos, cumulatively amounting to 95GB of storage capacity.



3.7.1 Dataset Preparation and Pre-processing

Before entering images into the network for training purposes, a set of pre-processing operations are applied in order to obtain cleaner information.

For this project, three different techniques have been applied to identify each scene, enhance images, and augment data:

- **Video Editing and Trimming:**
  Prior to training a CV model with the obtained datasets, it was important to trim unnecessary portions of the video which did not portray the occurrence of the crime labelled.
  This technique helped in the reduction of redundant and misleading features, as videos that were 5 minutes long were trimmed to 45 seconds focussing

merely on the duration when the crime occurred as the irrelevant portions had been discarded.
Videos with low resolutions and portions where the crime was not imperative or apparent were sharpened and cropped to highlight portions of the crime scene.

- **Data Labelling:**
  It is decisive to make annotations in each video in order to differentiate the abnormal scenes from the normal ones. For this, it is necessary to manually describe the parts of each video containing an anomaly. The remaining parts of the video can then be labelled under the normal class.

- **Data Augmentation:**
  Allows enlargement of the variety of data available to train a specific model without gathering new information.

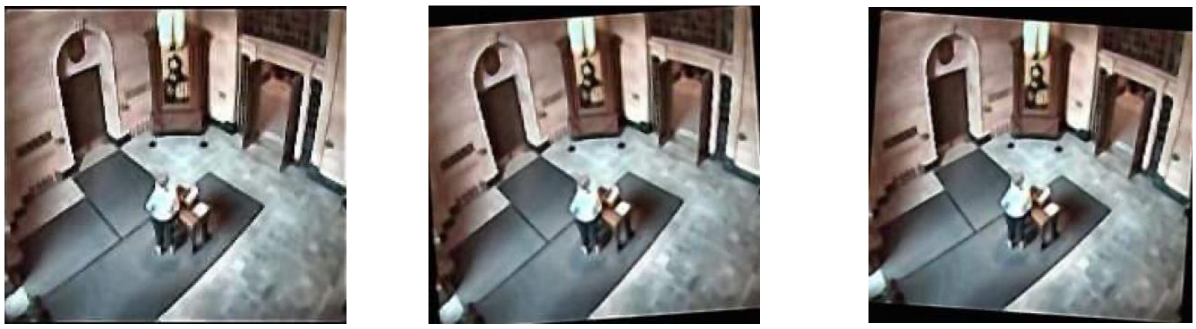


Fig. 3.10: Normal Image, Rotated Left, Rotated Right

- **Contrast Limited Adaptive Histogram (CLAHE)**
  Used to improve contrast in images, this is achieved by **elongating the most frequent intensity values.** This is used in the first step in order to grant areas of lower contrast to achieve a higher contrast overall and to avoid over-amplification related to noise.

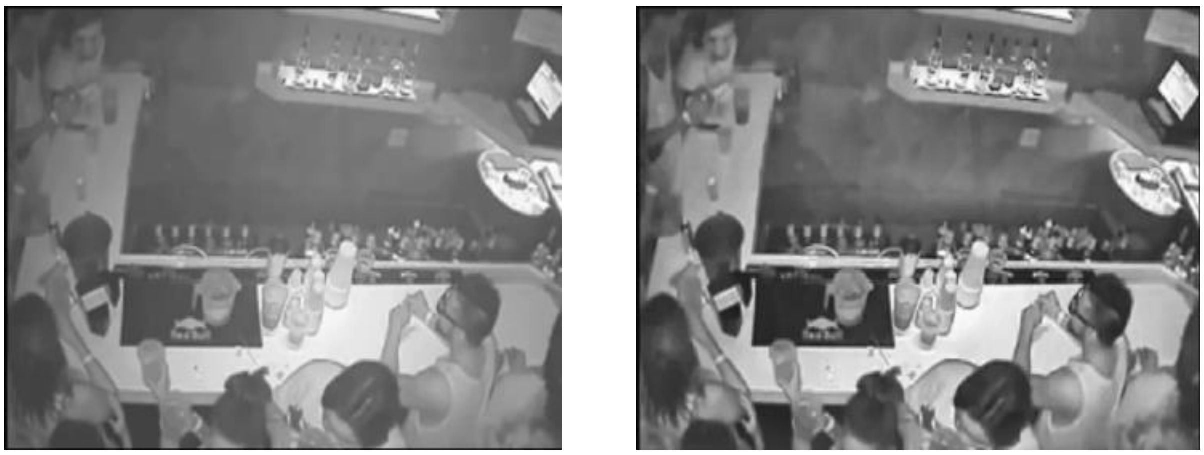
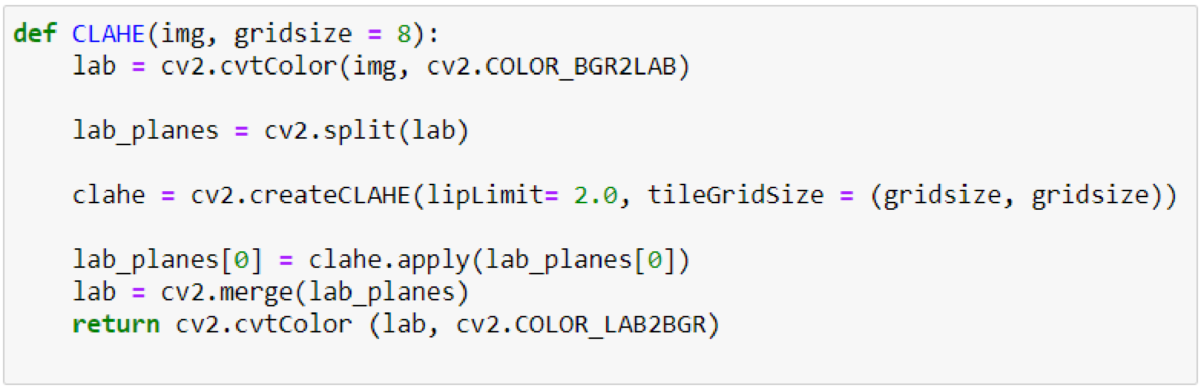


Fig. 3.11: Original Image vs Image with CLAHE

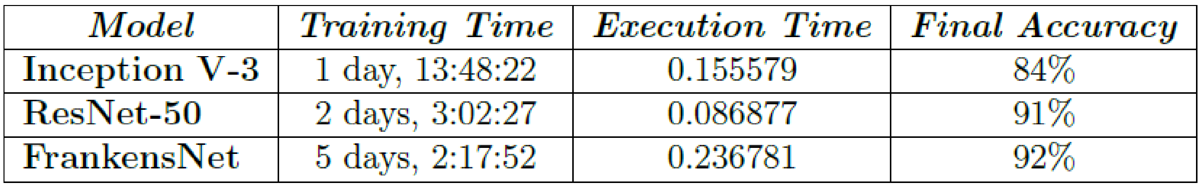
Code Snippet for CLAHE:



```
def CLAHE(img, gridsize = 8):
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    lab_planes = cv2.split(lab)

    clahe = cv2.createCLAHE(lipLimit= 2.0, tileGridSize = (gridsize, gridsize))

    lab_planes[0] = clahe.apply(lab_planes[0])
    lab = cv2.merge(lab_planes)
    return cv2.cvtColor (lab, cv2.COLOR_LAB2BGR)
```

3.7.2 Choosing the right model for classification

As per a series of tests conducted in this paper, we noticed the following end results for the three models trained against the same dataset: InceptionV3, ResNet-50 and their custom model, FrankensNet.



Model	*Training Time*	*Execution Time*	*Final Accuracy*
Inception V-3	1 day, 13:48:22	0.155579	84%
ResNet-50	2 days, 3:02:27	0.086877	91%
FrankensNet	5 days, 2:17:52	0.236781	92%

Fig. 3.12: Model Training Durations

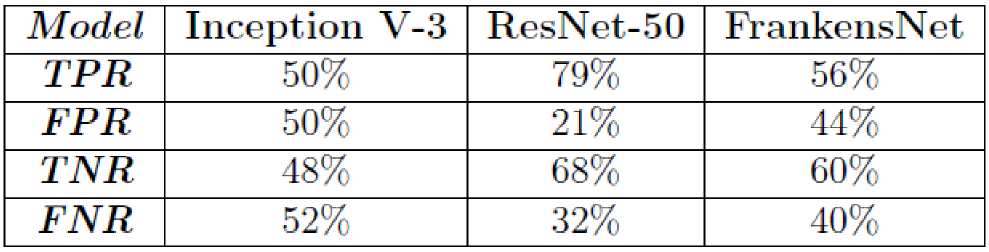
Moreover, it was also found that the ResNet was lesser prone to delivering **false positive results.**

- **False Positive:** When no crime is happening on the scene, but the model considers it as one.

- **False Negative:** When an anomaly is happening on the scene, but the model does not detect it.

- **True Positive:** When an anomaly occurs on the scene and the model accurately considers it as one.

- **True Negative:** When an anomaly is not happening on the scene and the model does not detect one.



Model	Inception V-3	ResNet-50	FrankensNet
TPR	50%	79%	56%
FPR	50%	21%	44%
TNR	48%	68%	60%
FNR	52%	32%	40%

Fig. 3.13: Classification Rates

Based on these insights, we chose the ResNet-50 for our project on Crime Detection. The ResNet-50 we chose has been pre-trained with an **activity dataset** in order to spare computational costs for training a model from scratch on the UCF Crimes Dataset.

3.8 Residual Network (ResNet)

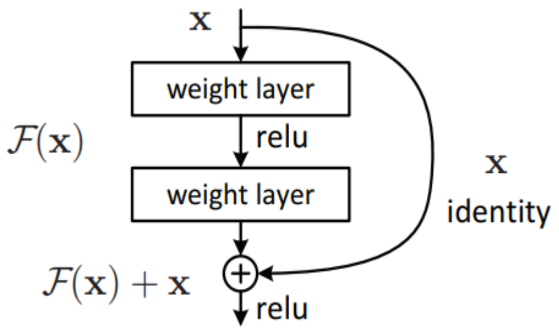
The ResNet layers are explicitly formulated as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions.

Its depth varies between 18 and 152 convolutional layers. ResNet introduces shortcut connections that bypass a signal from one layer to the next. Such connections pass through the gradient flows of networks from later layers to early layers, and ease the training of very deep networks.

Residual Block illustrated below allows the connection to bypass a signal from the top of the block to the tail.

3.8.1 Residual Block

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Network. In this network we use a technique called **skip connections**. The skip connection skips training from a few layers and connects directly to the output.
The approach behind this network is instead of layers learn the underlying mapping, we allow network fit the residual mapping.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 3.14: Residual Block

3.9 Proposed Video Classification Technique

Since a video is a **series of frames**, a naïve video classification method would be:

- Loop over all frames in a video file.
- For each frame, pass the frame through the CNN.
- Classify each frame individually and independent of each other.
- Choose the label with the largest corresponding probability.
- Label the frame and write the output frame to disk.

However, there are a set of problems encountered – one of the problems that does not require much attention for small scale crime detection is that **there is no correlation preserved between subsequent frames.** In order to achieve this, we require a sequence model (RNNs, LSTMs, GRUs) which can repeatedly run for all previous outputs as input to the recurring network. As this requires great computation and results in increased complexity, this solution is not implemented.

Another problem that occurs even in the case of image classification is that of **prediction flickering**, wherein the output label predicted by the model flickers as and when the output of the model changes.

A simple, yet elegant solution is to utilize the **rolling prediction averaging.** Under this technique, we slightly modify the processes at the output as follows:

- Obtain prediction from CNN
- Maintain a list of last K predictions
- Compute the average of last K predictions and choose the labels with the largest corresponding probability
- Label the frame and write the output to disk

3.10 Training the ResNet on UCF-Crimes Dataset

Steps involved in training and performing validation of the ResNet:

- Establish the following parameters: directories where the training and validation images are located, batch size, number of epochs, height and width of images, learning rate, etc.

- Generate training and validation data from ImageDataGenerator module of TensorFlow.

- Establishing behaviour after each epoch. (Saving information on training and validation accuracies at a given stage and plotting figures at the end of training)

- Storing the model and weights with varying extensions (.savedmodel, .pb, .h5)

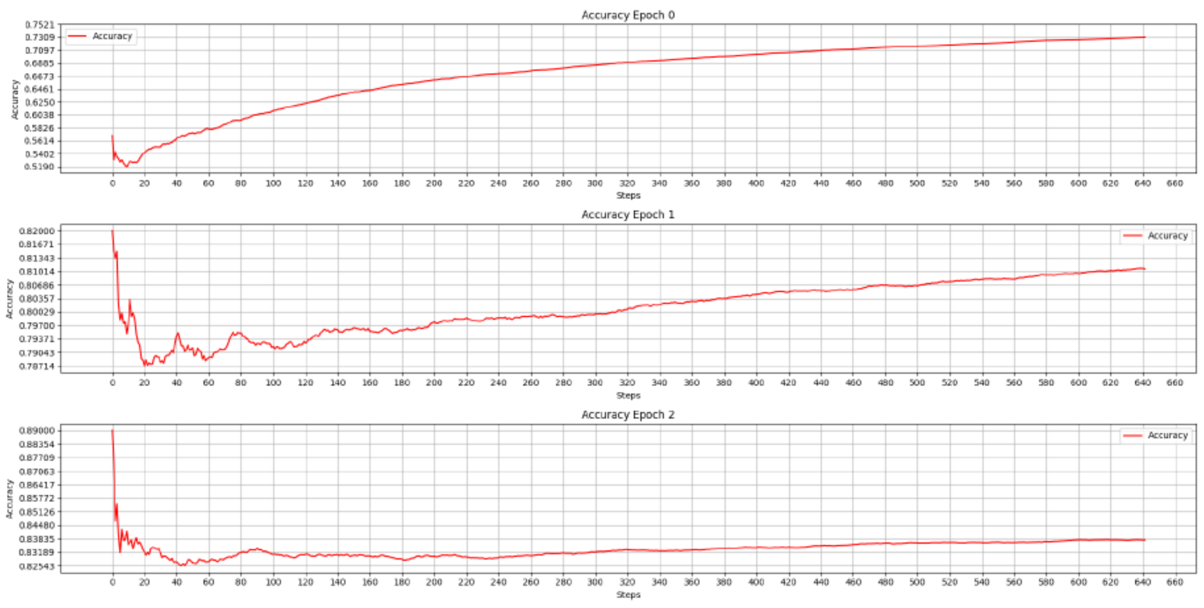
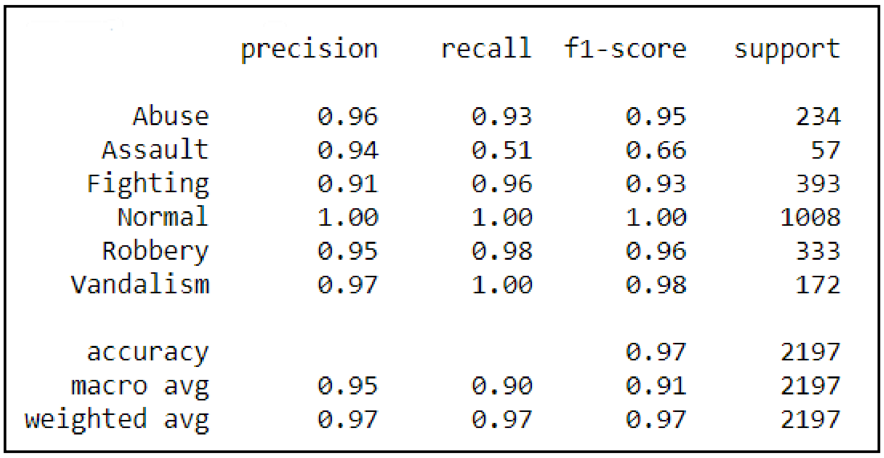


Fig. 3.15: Minimum Accuracy (LR: 10^{-4}, Batch Size = 32)

```
Epoch 1/50
205/205 [==============================] - 119s 512ms/step - loss: 1.4309 - accuracy: 0.5
046 - val_loss: 0.8264 - val_accuracy: 0.6967
Epoch 2/50
205/205 [==============================] - 104s 507ms/step - loss: 0.8809 - accuracy: 0.6
794 - val_loss: 0.6620 - val_accuracy: 0.7461
Epoch 3/50
205/205 [==============================] - 103s 503ms/step - loss: 0.7386 - accuracy: 0.7
386 - val_loss: 0.5724 - val_accuracy: 0.8024
Epoch 4/50
205/205 [==============================] - 105s 513ms/step - loss: 0.6517 - accuracy: 0.7
697 - val_loss: 0.5136 - val_accuracy: 0.8235
Epoch 5/50
205/205 [==============================] - 105s 512ms/step - loss: 0.6065 - accuracy: 0.7
881 - val_loss: 0.4645 - val_accuracy: 0.8488


                                            …

                                            …

                                            …


Epoch 45/50
205/205 [==============================] - 97s 475ms/step - loss: 0.1821 - accuracy: 0.94
69 - val_loss: 0.1257 - val_accuracy: 0.9646
Epoch 46/50
205/205 [==============================] - 98s 476ms/step - loss: 0.1722 - accuracy: 0.94
84 - val_loss: 0.1233 - val_accuracy: 0.9651
Epoch 47/50
205/205 [==============================] - 97s 473ms/step - loss: 0.1741 - accuracy: 0.94
86 - val_loss: 0.1224 - val_accuracy: 0.9646
Epoch 48/50
205/205 [==============================] - 101s 494ms/step - loss: 0.1669 - accuracy: 0.9
503 - val_loss: 0.1207 - val_accuracy: 0.9665
Epoch 49/50
205/205 [==============================] - 105s 514ms/step - loss: 0.1677 - accuracy: 0.9
500 - val_loss: 0.1179 - val_accuracy: 0.9678
Epoch 50/50
205/205 [==============================] - 181s 886ms/step - loss: 0.1581 - accuracy: 0.9
535 - val_loss: 0.1168 - val_accuracy: 0.9669
```

Fig. 3.16: Training the classifier (Epochs = 50)



```
              precision    recall  f1-score   support

       Abuse       0.96      0.93      0.95       234
     Assault       0.94      0.51      0.66        57
    Fighting       0.91      0.96      0.93       393
      Normal       1.00      1.00      1.00      1008
     Robbery       0.95      0.98      0.96       333
   Vandalism       0.97      1.00      0.98       172

    accuracy                           0.97      2197
   macro avg       0.95      0.90      0.91      2197
weighted avg       0.97      0.97      0.97      2197
```

Fig. 3.17: Evaluating the classifier

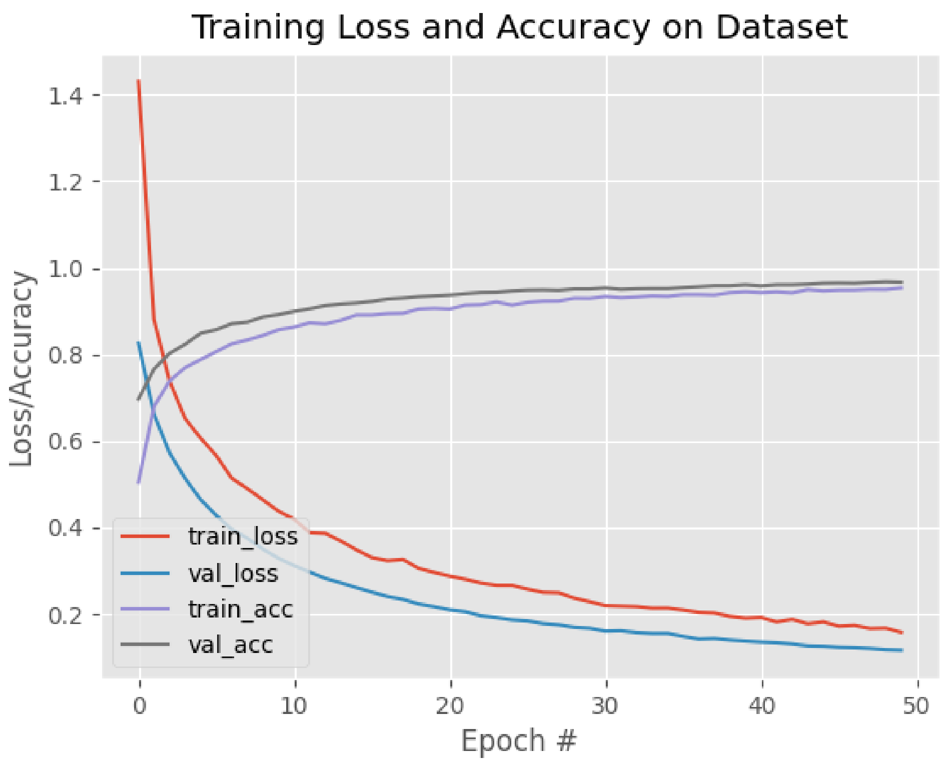


Fig. 3.18: Loss and Accuracy as a function of epochs

3.10.1 Model Architectures (Crime Detection)

The clear PNG image of the classifier based on the ResNet-50 architecture and trained on UCF-Crimes Dataset is listed below.

The ResNet-50 architecture could not be exported to PNG due to its large size. However, it is among the most common architectures and can be found online.

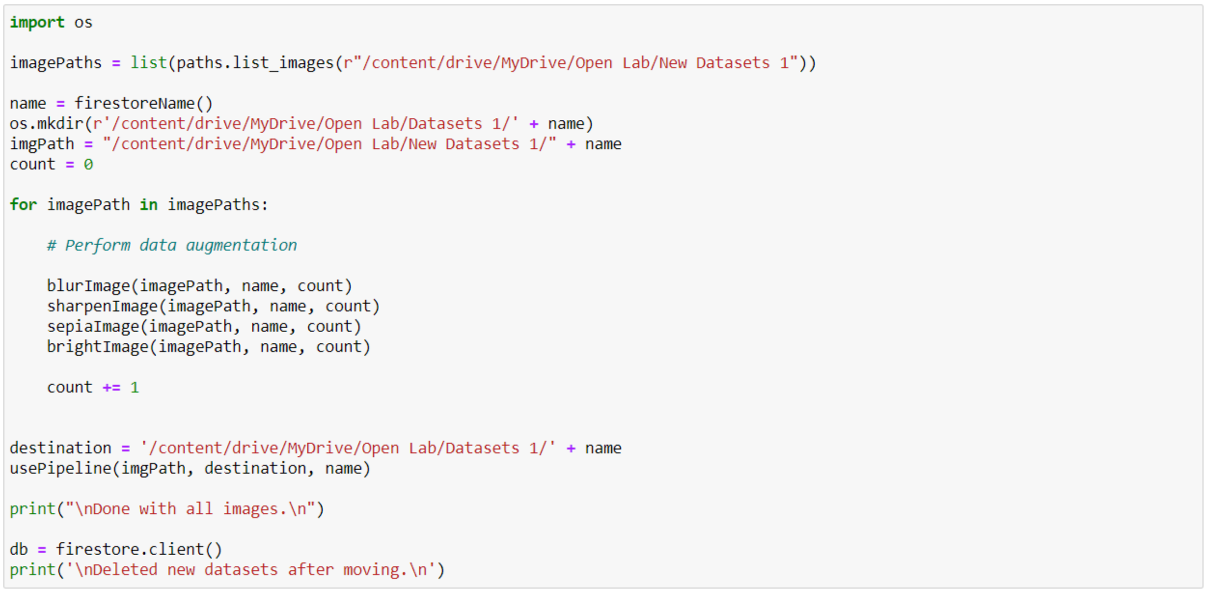
- Model Architecture for crime classifier based on ResNet-50 [Clear PNG Image]

CHAPTER 4

Demonstration and Results

4.1 Dataset Augmentation

The automated data augmentation pipeline handles new photos registered by the Android application and saves them in Drive and Cloud Firestore.



```python
import os

imagePaths = list(paths.list_images(r"/content/drive/MyDrive/Open Lab/New Datasets 1"))

name = firestoreName()
os.mkdir(r'/content/drive/MyDrive/Open Lab/Datasets 1/' + name)
imgPath = "/content/drive/MyDrive/Open Lab/New Datasets 1/" + name
count = 0

for imagePath in imagePaths:

    # Perform data augmentation

    blurImage(imagePath, name, count)
    sharpenImage(imagePath, name, count)
    sepiaImage(imagePath, name, count)
    brightImage(imagePath, name, count)

    count += 1


destination = '/content/drive/MyDrive/Open Lab/Datasets 1/' + name
usePipeline(imgPath, destination, name)

print("\nDone with all images.\n")

db = firestore.client()
print('\nDeleted new datasets after moving.\n')
```

Fig. 4.1: Code snippet responsible for handling Data Augmentation

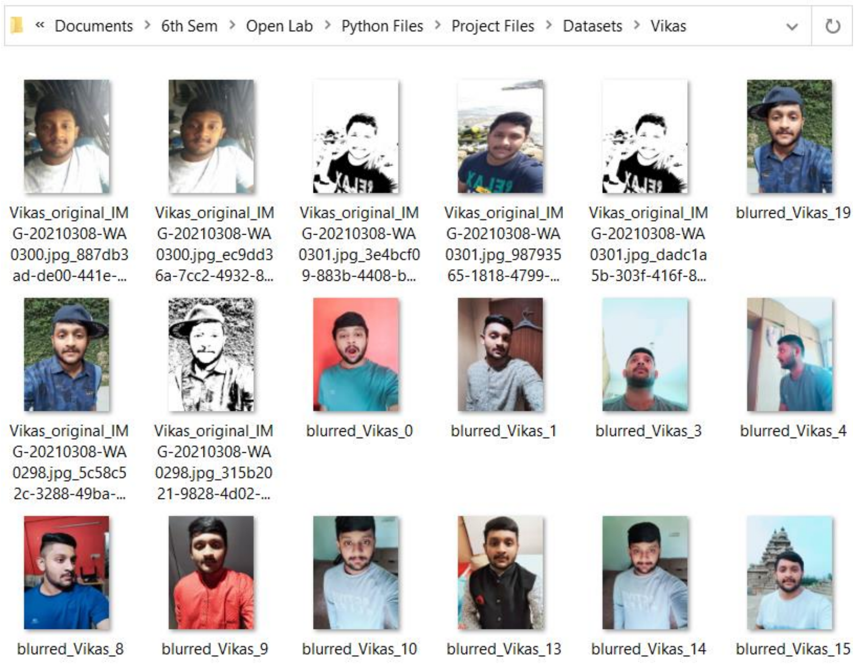

« Documents › 6th Sem › Open Lab › Python Files › Project Files › Datasets › Vikas
Vikas_original_IMG-20210308-WA0300.jpg_887db3ad-de00-441e-...
Vikas_original_IMG-20210308-WA0300.jpg_ec9dd36a-7cc2-4932-8...
Vikas_original_IMG-20210308-WA0301.jpg_3e4bcf09-883b-4408-b...
Vikas_original_IMG-20210308-WA0301.jpg_98793565-1818-4799-...
Vikas_original_IMG-20210308-WA0301.jpg_dadc1a5b-303f-416f-8...
blurred_Vikas_19
Vikas_original_IMG-20210308-WA0298.jpg_5c58c52c-3288-49ba-...
Vikas_original_IMG-20210308-WA0298.jpg_315b2021-9828-4d02-...
blurred_Vikas_0
blurred_Vikas_1
blurred_Vikas_3
blurred_Vikas_4
blurred_Vikas_8
blurred_Vikas_9
blurred_Vikas_10
blurred_Vikas_13
blurred_Vikas_14
blurred_Vikas_15

Fig. 4.2: Augmented Data

4.2 Face Recognition

4.2.1 Face Recognition on Android Application

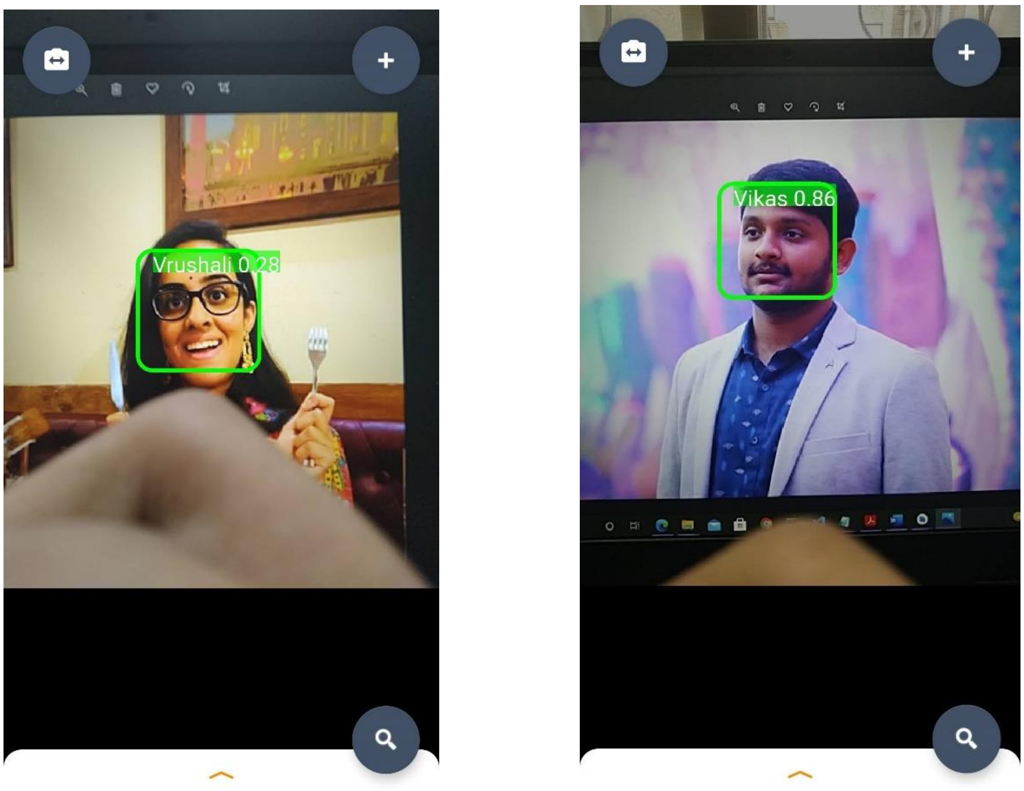
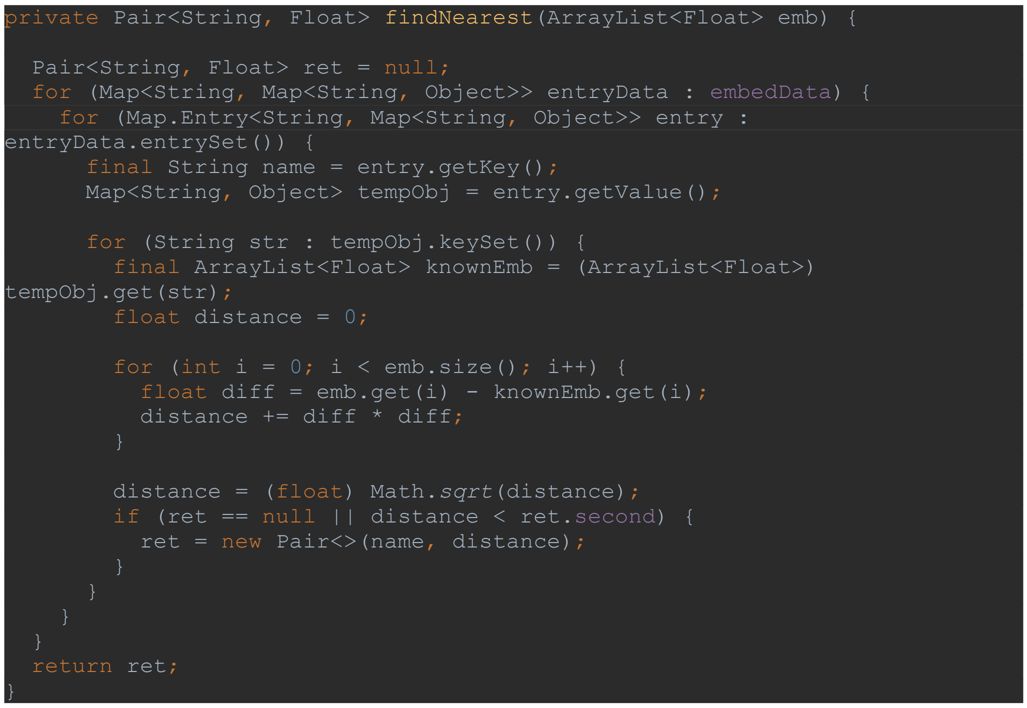


Fig. 4.3: Face Recognition on Android App



```java
private Pair<String, Float> findNearest(ArrayList<Float> emb) {

  Pair<String, Float> ret = null;
  for (Map<String, Map<String, Object>> entryData : embedData) {
    for (Map.Entry<String, Map<String, Object>> entry :
entryData.entrySet()) {
      final String name = entry.getKey();
      Map<String, Object> tempObj = entry.getValue();

      for (String str : tempObj.keySet()) {
        final ArrayList<Float> knownEmb = (ArrayList<Float>)
tempObj.get(str);
        float distance = 0;

        for (int i = 0; i < emb.size(); i++) {
          float diff = emb.get(i) - knownEmb.get(i);
          distance += diff * diff;
        }

        distance = (float) Math.sqrt(distance);
        if (ret == null || distance < ret.second) {
          ret = new Pair<>(name, distance);
        }
      }
    }
  }
  return ret;
}
```

Fig. 4.4: Code snippet responsible for classification on Android (Java)

4.2.2 Face Recognition and logging (from CCTV footage)

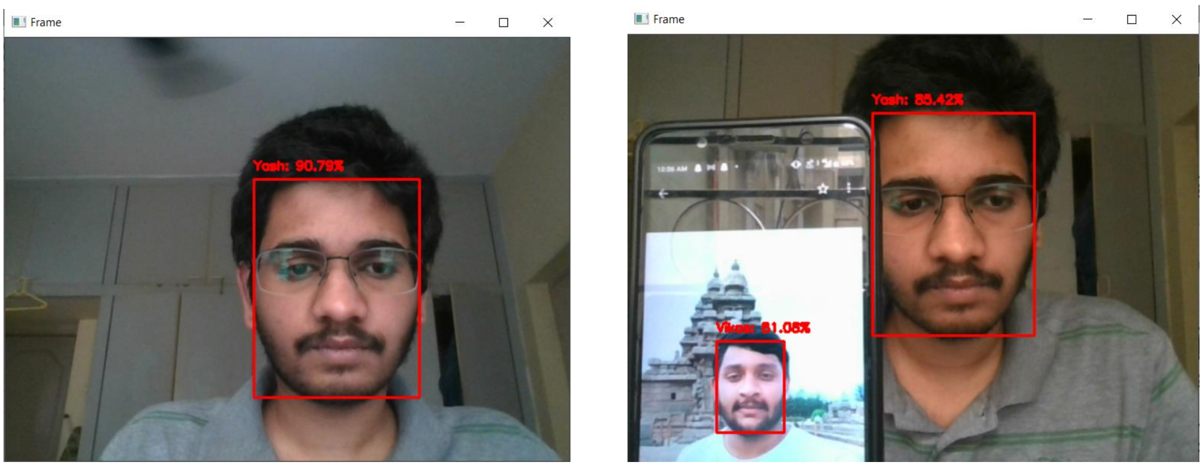
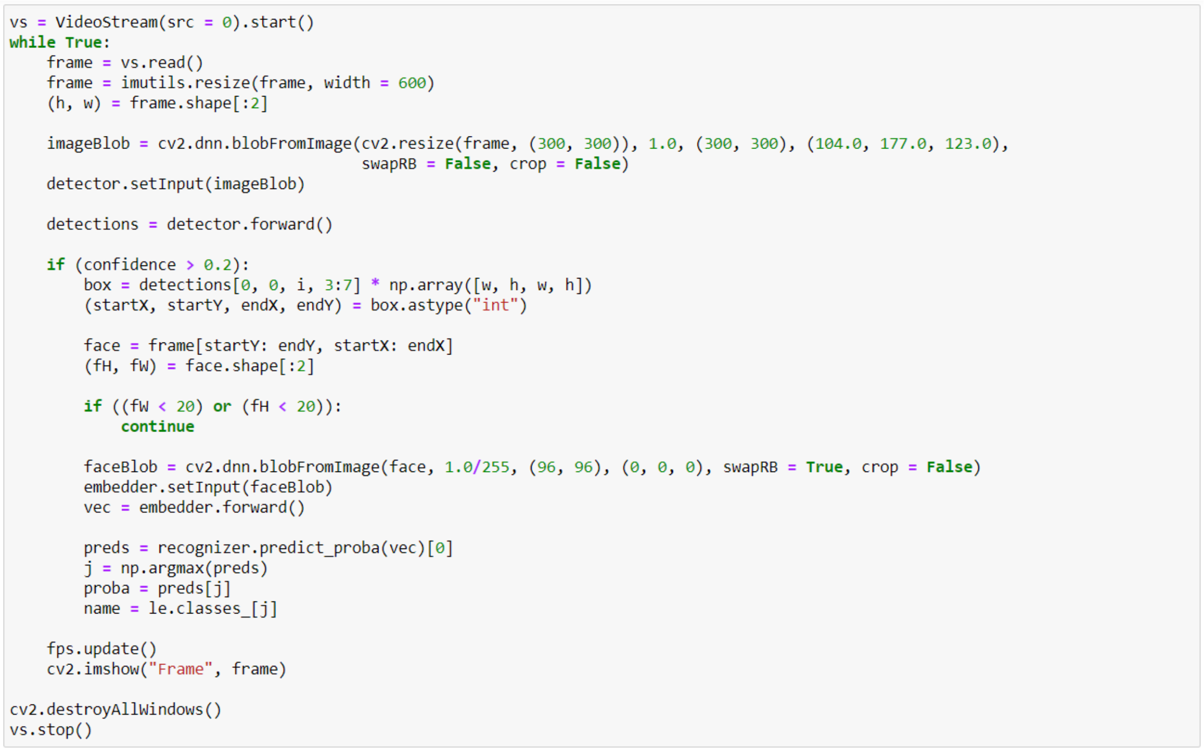
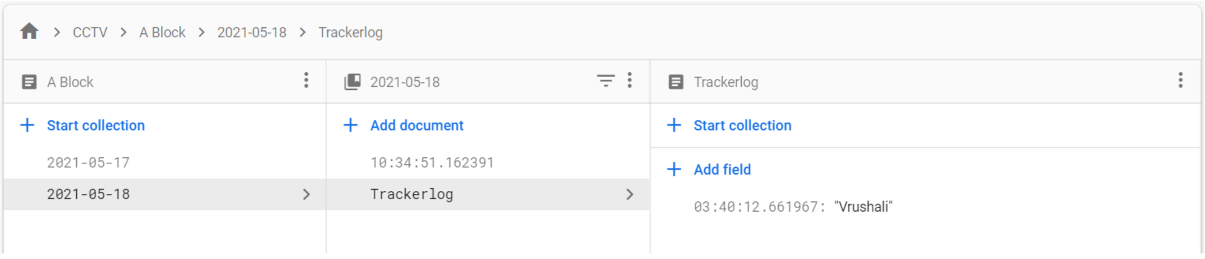


Fig. 4.5: Face Recognition from a remote mobile camera (simulating a CCTV) obtained via an RTSP URL



```python
vs = VideoStream(src = 0).start()
while True:
    frame = vs.read()
    frame = imutils.resize(frame, width = 600)
    (h, w) = frame.shape[:2]

    imageBlob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300, 300), (104.0, 177.0, 123.0),
                                      swapRB = False, crop = False)
    detector.setInput(imageBlob)

    detections = detector.forward()

    if (confidence > 0.2):
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        face = frame[startY: endY, startX: endX]
        (fH, fW) = face.shape[:2]

        if ((fW < 20) or (fH < 20)):
            continue

        faceBlob = cv2.dnn.blobFromImage(face, 1.0/255, (96, 96), (0, 0, 0), swapRB = True, crop = False)
        embedder.setInput(faceBlob)
        vec = embedder.forward()

        preds = recognizer.predict_proba(vec)[0]
        j = np.argmax(preds)
        proba = preds[j]
        name = le.classes_[j]

    fps.update()
    cv2.imshow("Frame", frame)

cv2.destroyAllWindows()
vs.stop()
```

Fig. 4.6: Code snippet responsible for face recognition (Python)



2021-05-17
2021-05-18

10:34:51.162391
Trackerlog

03:40:12.661967: "Vrushali"

Fig. 4.7: A-Block CCTV recognizes Vrushali at 03:40am on 18th May and logs it into Cloud Firestore database

4.3 Crime Detection

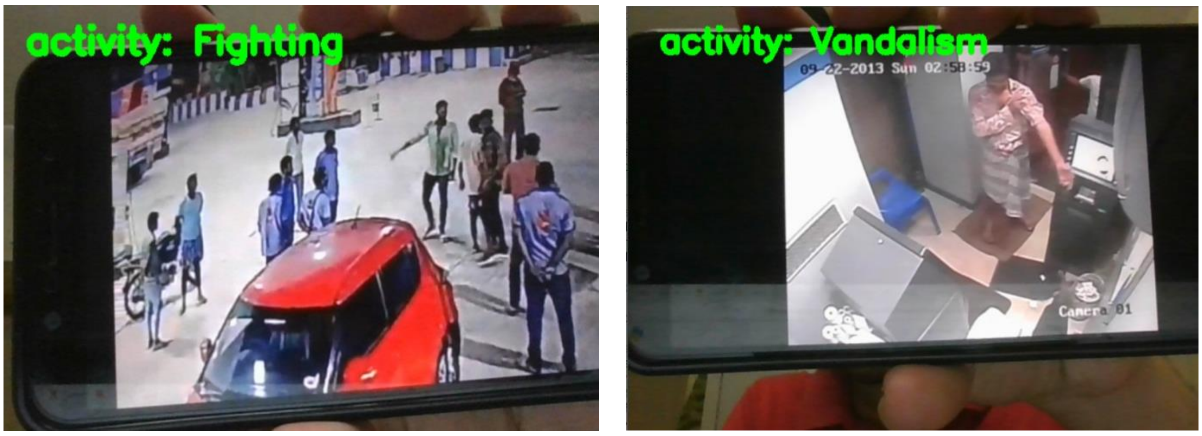
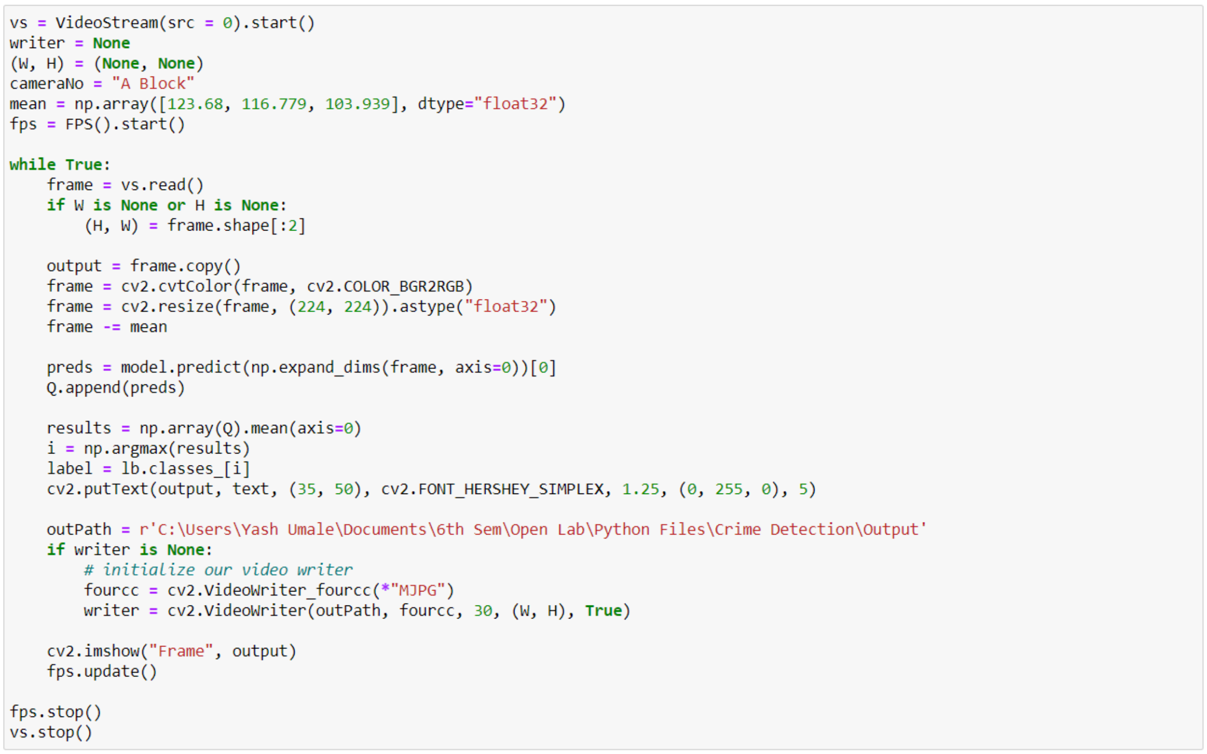


Fig. 4.8: Detecting and classifying crimes from footage
(Recognition occurs simultaneously and is logged in to the database as shown below)



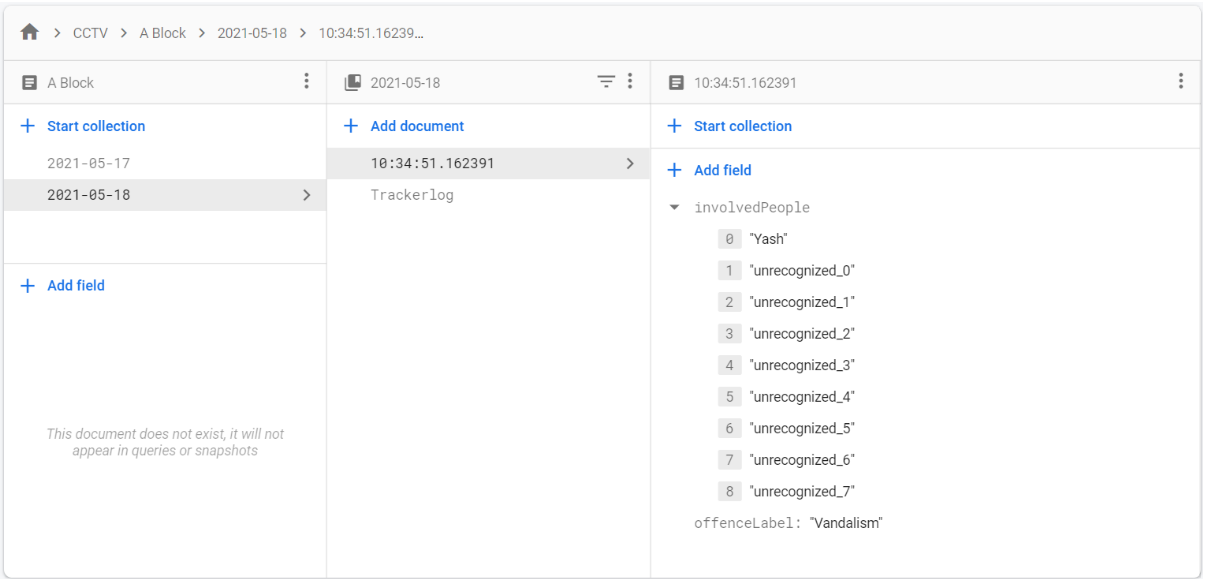
```
vs = VideoStream(src = 0).start()
writer = None
(W, H) = (None, None)
cameraNo = "A Block"
mean = np.array([123.68, 116.779, 103.939], dtype="float32")
fps = FPS().start()

while True:
    frame = vs.read()
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    output = frame.copy()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (224, 224)).astype("float32")
    frame -= mean

    preds = model.predict(np.expand_dims(frame, axis=0))[0]
    Q.append(preds)

    results = np.array(Q).mean(axis=0)
    i = np.argmax(results)
    label = lb.classes_[i]
    cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX, 1.25, (0, 255, 0), 5)

    outPath = r'C:\Users\Yash Umale\Documents\6th Sem\Open Lab\Python Files\Crime Detection\Output'
    if writer is None:
        # initialize our video writer
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(outPath, fourcc, 30, (W, H), True)

    cv2.imshow("Frame", output)
    fps.update()

fps.stop()
vs.stop()
```

Fig. 4.9: Code snippet responsible for **only detecting and classifying** the crime without recognition

Application:

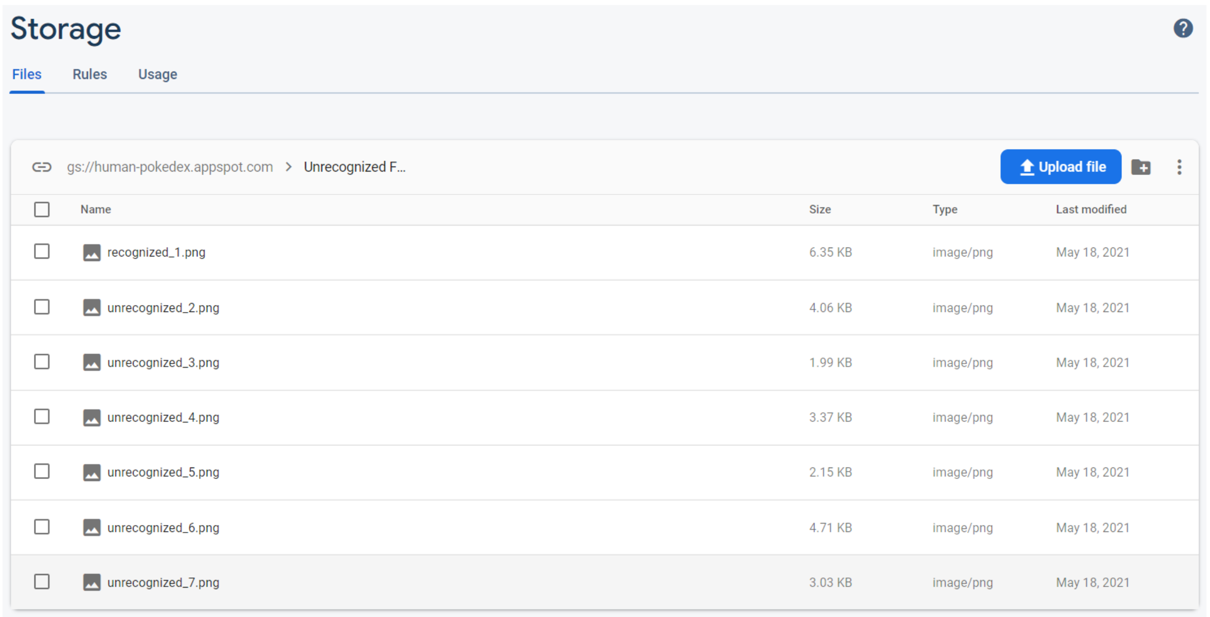
When a crime is detected by a CCTV camera, it is immediately logged into Firestore along with a timestamp and the list of people involved in the frame after performing recognition.

A notification is received by the app users who're in the vicinity of the camera (as they've been recognised and tracked a couple of minutes earlier) about the occurrence of the crime. The following screenshots display crime detection, recognition and database logging.


CCTV > A Block > 2021-05-18 > 10:34:51.16239...

A Block	2021-05-18	10:34:51.162391
+ Start collection	+ Add document	+ Start collection
2021-05-17	10:34:51.162391	+ Add field
2021-05-18	Trackerlog	involvedPeople
+ Add field		0 "Yash"
This document does not exist, it will not appear in queries or snapshots		1 "unrecognized_0"
		2 "unrecognized_1"
		3 "unrecognized_2"
		4 "unrecognized_3"
		5 "unrecognized_4"
		6 "unrecognized_5"
		7 "unrecognized_6"
		8 "unrecognized_7"
		offenceLabel: "Vandalism"

Fig. 4.10: Output after detecting vandalism. Faces unrecognized in the video are saved as Bitmaps for further recognition by concerned authorities.


Storage

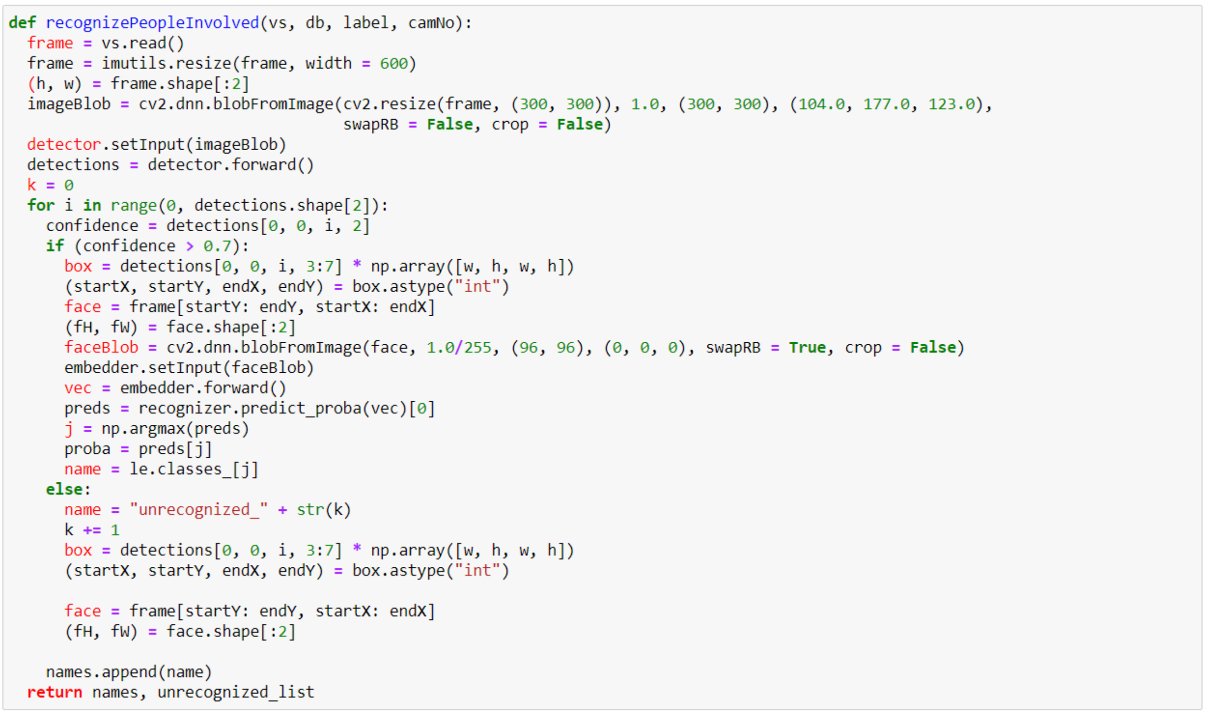
Files Rules Usage

gs://human-pokedex.appspot.com > Unrecognized F...

Upload file

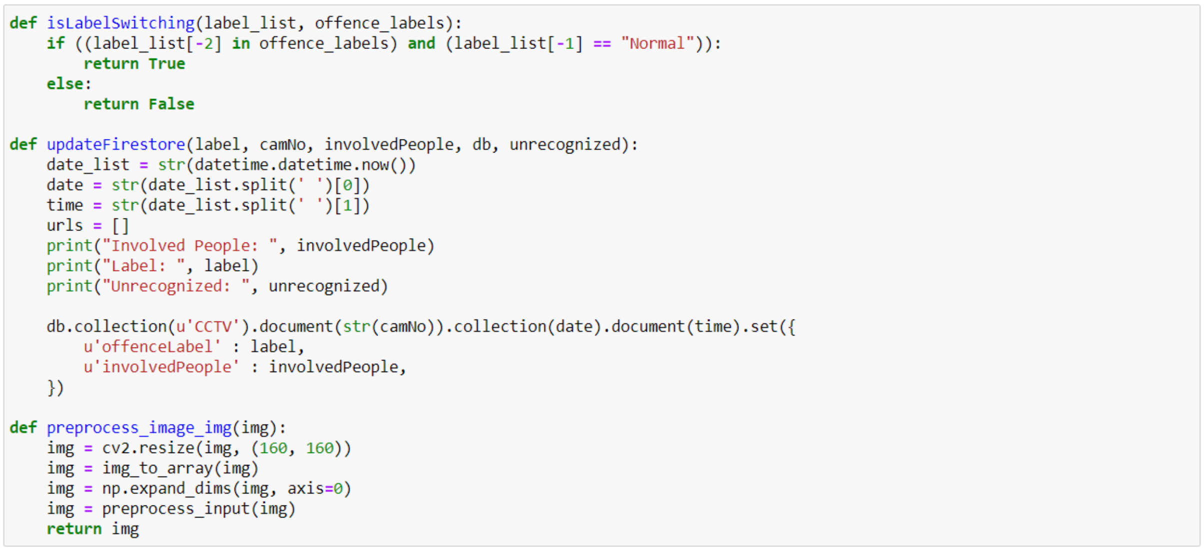
Name	Size	Type	Last modified
recognized_1.png	6.35 KB	image/png	May 18, 2021
unrecognized_2.png	4.06 KB	image/png	May 18, 2021
unrecognized_3.png	1.99 KB	image/png	May 18, 2021
unrecognized_4.png	3.37 KB	image/png	May 18, 2021
unrecognized_5.png	2.15 KB	image/png	May 18, 2021
unrecognized_6.png	4.71 KB	image/png	May 18, 2021
unrecognized_7.png	3.03 KB	image/png	May 18, 2021

Fig. 4.11: Firebase Storage contains the Bitmaps of unrecognized criminals. These photos are displayed to users on the Android app.



```python
def recognizePeopleInvolved(vs, db, label, camNo):
  frame = vs.read()
  frame = imutils.resize(frame, width = 600)
  (h, w) = frame.shape[:2]
  imageBlob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300, 300), (104.0, 177.0, 123.0),
                                    swapRB = False, crop = False)
  detector.setInput(imageBlob)
  detections = detector.forward()
  k = 0
  for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if (confidence > 0.7):
      box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
      (startX, startY, endX, endY) = box.astype("int")
      face = frame[startY: endY, startX: endX]
      (fH, fW) = face.shape[:2]
      faceBlob = cv2.dnn.blobFromImage(face, 1.0/255, (96, 96), (0, 0, 0), swapRB = True, crop = False)
      embedder.setInput(faceBlob)
      vec = embedder.forward()
      preds = recognizer.predict_proba(vec)[0]
      j = np.argmax(preds)
      proba = preds[j]
      name = le.classes_[j]
    else:
      name = "unrecognized_" + str(k)
      k += 1
      box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
      (startX, startY, endX, endY) = box.astype("int")

      face = frame[startY: endY, startX: endX]
      (fH, fW) = face.shape[:2]

    names.append(name)
  return names, unrecognized_list
```

Fig. 4.12: Snippet responsible for face recognition while detecting crimes



```python
def isLabelSwitching(label_list, offence_labels):
    if ((label_list[-2] in offence_labels) and (label_list[-1] == "Normal")):
        return True
    else:
        return False

def updateFirestore(label, camNo, involvedPeople, db, unrecognized):
    date_list = str(datetime.datetime.now())
    date = str(date_list.split(' ')[0])
    time = str(date_list.split(' ')[1])
    urls = []
    print("Involved People: ", involvedPeople)
    print("Label: ", label)
    print("Unrecognized: ", unrecognized)

    db.collection(u'CCTV').document(str(camNo)).collection(date).document(time).set({
        u'offenceLabel' : label,
        u'involvedPeople' : involvedPeople,
    })

def preprocess_image_img(img):
    img = cv2.resize(img, (160, 160))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    return img
```

Fig. 4.13: Function for keeping Firestore up-to-date and other helper functions

CHAPTER 5

Conclusion and Future Enhancements

In general terms, this chapter presents a summary of every step executed to fulfill the project, starting from literature review, followed by development of the Android application, setting up the backend on Firebase, implementing face recognition on Android and PC, and detecting crimes from CCTV footage.

- In recent years, several deep learning approaches have been presented for video classification because of its successful reach in object detection, image classification and pattern recognition. However, those techniques still face challenges with video classification due to a set of factors, such as capturing spatio-temporal information adequately.

- After literature review, ResNet-50 was selected for crime detection whereas OpenFace's FaceNet implementation and InceptionResNetV1 were chosen for Face Recognition. InceptionResNetV1 was converted to a TFLite model to satisfy the requirement of face recognition on an edge device.

- A CNN requires a wide range of data to be trained on, and in many cases the data was either unfit for consumption, was poorly recorded or didn't match the label it was uploaded under. Hence, such data had to be processed, cleaned, trimmed, augmented and re-labelled under requisite criteria and constraints.

- CNN models are not created with general purpose applications. Their parameters need to be adjusted to work adequately with respect to a given specific task. For our project, as per literature we noticed that ResNet took higher training time but also returned an increased accuracy.

- Creating and curating the Android application to share a backend that is used by the Python scripts for crime and face classification also meant that a lot of additional features pertaining to the same tech stack could be added. A student tracker log was implemented to leverage the consistent connection between the CCTV network and the application along with the face recognition algorithm executed upon each video stream in the network.

Future Enhancements

The combination of an Android app and an integrated campus management system can be utilized to its full potential if some more work is done on the app. As the app still needs some more work to become fully functional, completing the features that truly include campus management can help the project achieve its real purpose.

On the computer vision front, it would be great to obtain better datasets, try out other models and hyperparameters to achieve improved accuracy.

Glossary

• **Classification:** It is the set of categories (sub-populations) where a new observation belongs, on the basis of a training set of data containing observations.

• **Convolutional Kernel:** It consists of a small numerical matrix, mostly used in image processing for blurring, sharpening, embossing, edge detection, etc.

• **Convolutional Layer:** It uses convolutional kernels in order to extract features from input data through an activation map of that kernel.

• **Convolutional Neural Network:** It is a class of deep neural networks, in various cases is applied to analyzing visual imagery.

• **CPU:** It is known as the central processing unit, it is the computer component that is responsible for interpreting and executing most of the commands from the computer's hardware and software.

• **Dataset:** It is a collection of related sets of information that is composed of separated elements but can be manipulated as a unit by a computer.

• **Deep Learning:** It is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that ins unstructured or unlabelled.

• **Deep Neural Network:** It is a neural network with more than two layers.

• **Feature Extraction:** It is the generation of derived values coming from an initial dataset, intended to be informative to facilitate the subsequent learning and generalization steps.

• **Feature Map:** It is a matrix, or a set of matrices used for the mapping of where a certain kind of feature is found on the image which can be of interest for the Network.

• **Gaussian Mixture Model:** It is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters.

• **Graphics processing unit (GPU):** It is a computer chip that performs rapid mathematical calculations, primarily for the purpose of rendering images. Also widely used for deep learning purposes.

• **FLOPS:** Floating point operations per second is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations.

• **Hyperparameters:** They are model-specific properties that are fixed before the network model starts to train.

• **Localization:** It consists of finding/detecting where the object of interest is and drawing a bounding box around it.

• **One Class SVM:** It is an unsupervised learning algorithm based on deep learning approaches commonly used for classification tasks.

• **Output Layer:** It refers to the final layer of the network which produces given outputs for the program for future interpretation.

• **Optimization:** It is a process to find an alternative with the most cost effective or highest achievable performance under the given constraints, by maximizing desired factors and minimizing undesired ones.

• **Over-Fit:** A modeling error which occurs when a function is too closely fit to a limited set of data points.

• **Parameter:** The parameters of a neural network are typically the weights of the connections. In this case, these parameters are learned during the training stage.

• **Post-processing:** Term used for quality improvement image processing methods.

• **Pre-processing:** It refers to the transformations applied to input data before feeding it to the algorithm. It is a technique used to convert the raw data into a clean data set.

• **Stride:** It corresponds to the number of steps that the filter will move each time on a given direction.

• **Testing dataset:** It is a dataset independent of the training dataset that follows the same probability distribution as the training dataset to validation purposes.

• **Training dataset:** It is a sample dataset used for learning purposes to fit the parameters (e.g., weights) of, for example, a classifier.

• **Validation dataset:** It is a dataset of examples used to tune the hyperparameters (i.e. the architecture) of a classifier.

References

FaceNet: A Unified Embedding for Face Recognition and Clustering:
https://www.cv-foundation.org/openaccess/content_cvpr_2015/app/1A_089.pdf

OpenFace Project:
https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/

Implementing Multiple face recognition using OpenCV:
https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

Caffe Framework:
https://caffe.berkeleyvision.org/tutorial/net_layer_blob.html

Anomaly Detection Paper:
https://repositorio.yachaytech.edu.ec/bitstream/123456789/183/1/ECMC0027.pdf
https://www.ijeat.org/wp-content/uploads/papers/v8i5S/E10220585S19.pdf

MobileFaceNets:
https://arxiv.org/pdf/1804.07573.pdf

Video Classification techniques:
https://youtu.be/dY7j5dBqS5g

Netron (visualizing models):
https://github.com/lutzroeder/netron

SSD (Single-shot multibox Detector): https://arxiv.org/abs/1512.02325

Best models for face detection:
https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/

Pyrebase:

https://github.com/thisbejim/Pyrebase#storage

TensorFlow subroutines:

https://github.com/serengil/tensorflow-101

On-device face recognition:

https://towardsdatascience.com/using-facenet-for-on-device-face-recognition-with-android-f84e36e19761

https://medium.com/@estebanuri/real-time-face-recognition-with-android-tensorflow-lite-14e9c6cc53a5

Fast extraction of frames (OpenCV):

https://medium.com/@haydenfaulkner/extracting-frames-fast-from-a-video-using-opencv-and-python-73b9b7dc9661

RecyclerView Primer:

https://www.google.com/search?q=RecyclerView%3A+No+adapter+attached%3B+skipping+layout&rlz=1C1ONGR_enIN929IN929&oq=RecyclerView%3A+No+adapter+attached%3B+skipping+layout&aqs=chrome..69i57j69i58.172j0j7&sourceid=chrome&ie=UTF-8

Firestore with Python:

https://saveyourtime.medium.com/firebase-cloud-firestore-add-set-update-delete-get-data-6da566513b1b

Dataset Augmentation:

https://algorithmia.com/blog/introduction-to-dataset-augmentation-and-expansion#:~:text=Dataset%20augmentation%20%E2%80%93%20the%20process%20of,requirements%20of%20Deep%20Learning%20models

RTSP Streams and URLs:

https://www.getscw.com/decoding/rtsp

[1] “Ucf-crime dataset (real-world anomalies detection in videos), Jun. 2019.

[2] A. R. Zamir, “Action recognition in realistic sports videos,” in Computer vision in sports. Springer, 2014, pp. 181–208.

[3] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large scale video classification with convolutional neural networks,” in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[4] M. Sabokrou, M. Fayyaz, M. Fathy, Z. Moayed, and R. Klette, “Deep-anomaly: Fully convolutional neural network for fast anomaly detection in crowded scenes,” Computer Vision and Image Understanding, vol. 172, pp. 88–97, 2018.

[5] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” ArXiv e-prints, 11 2015.

[6] L. Prechelt, “Early stopping-but when?” in Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop. Springer, 1998, pp. 55–69.

[7] Will Kay, Joao Carreira, Sudheendra Vijayanarasimhan, Fabio Viola, Brian Zhang, “The Kinetics Action Video Dataset”, May 2017.

[8] Mei Wang and Weihong Deng, “Deep Face Recognition: A Survey”, Aug 2020.

[9] Keval Doshi and Yasin Yilmaz, “Online Anomaly Detection in Surveillance Videos with Asymptotic Bounds on False Alarm Rate”, Oct 2020

[10] Waqas Sultani, Chen Chen, Mubarak Shah, “Real-world Anomaly Detection in Surveillance Videos”, Feb 2019.