A Machine Learning Pose Detection Framework to Identify Suspicious Activity

Research Project – Configuration Manual
MSc Data Analytics

Rajat Deepak Agrawal
Student ID: x21172030

School of Computing
National College of Ireland

Supervisor: Paul Stynes

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……Rajat Deepak Agrawal……..………………………………………………
Student ID:	………………x21172030…………………………………………………………….……
Programme:	…MSc Data Analytics……… …………… **Year:** 2023……….
Module:	…Research Project -Configuration Manual……………………………….….………
Lecturer:	……Prof. Paul Stynes ………………………………………………………….………
Submission Due Date:	………………14/08/2023…………………………………………………………….………
Project Title:	……A Machine Learning Pose Detection Framework to Identify Suspicious Activity…………………………………………………………….………
Word Count:	1755………………………………… **Page Count:** ……12…………………………….….….………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Rajat Deepak Agrawal………………………………………………………………

Date: …………………………………………14/08/2023…………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Machine Learning Pose Detection Framework to Identify Suspicious Activity

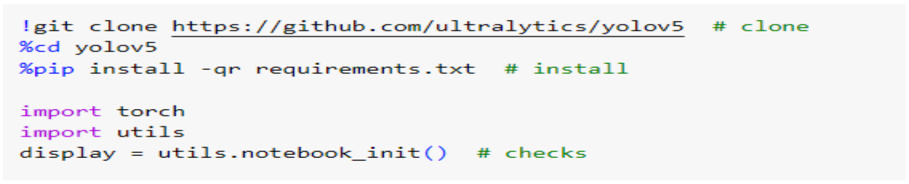
Rajat Deepak Agrawal
Student ID: x21172030

1 Introduction

This configuration manual describes the steps which were taken in the implementation of the research project ‘A Machine Learning Pose Detection Framework to Identify Suspicious Activity’. The document gives information about steps taken to acquire the data, system specifications for the project, libraries used, and code walkthrough which has been used for the implementation.
The document is divided into 6 sections. Section 2 consists of system requirements, section 3 gives information about data collection, section 4 gives information on data preprocessing and transformation, section 5 gives information about models which were analyzed, and section 6 provides information on alarm system.

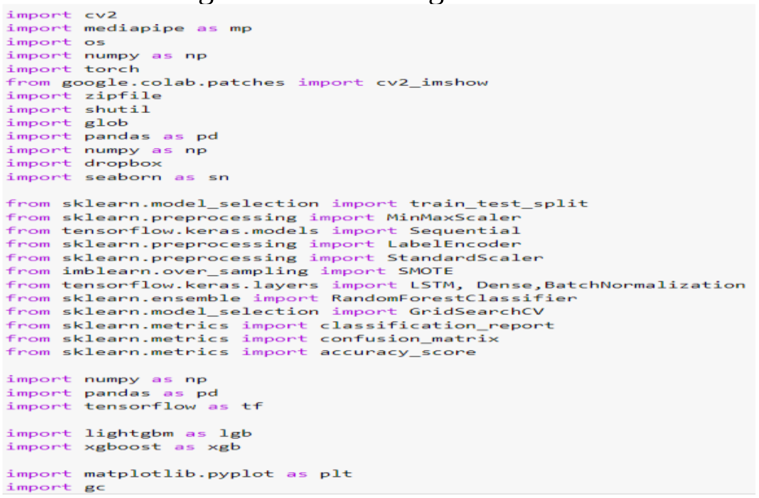
2 System requirements

The implementation of this research was done on google colab pro platform. The whole research was implemented using Python as programming language. The GPU hardware accelerator used was A100 on colab. The dependencies which needed to be installed was YOLO5 using git cloning.



```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt  # install

import torch
import utils
display = utils.notebook_init()  # checks
```

Figure 1 GIT cloning for YOLO



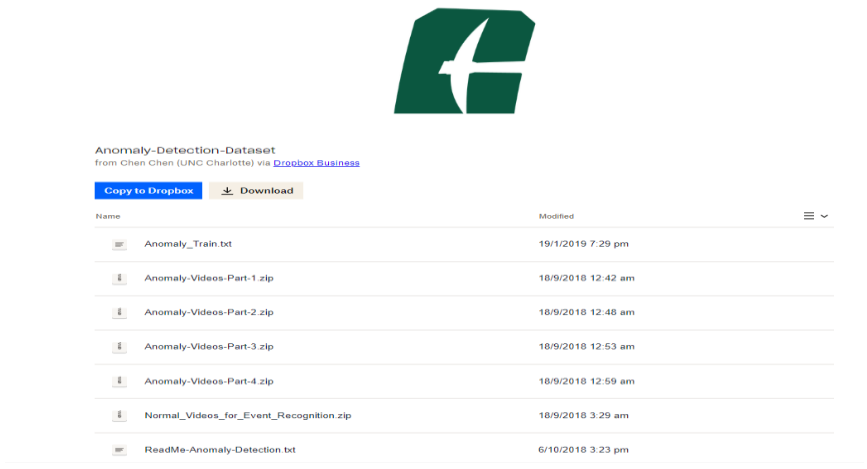
```
import cv2
import mediapipe as mp
import os
import numpy as np
import torch
from google.colab.patches import cv2_imshow
import zipfile
import shutil
import glob
import pandas as pd
import numpy as np
import dropbox
import seaborn as sn

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from tensorflow.keras.layers import LSTM, Dense,BatchNormalization
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

import numpy as np
import pandas as pd
import tensorflow as tf

import lightgbm as lgb
import xgboost as xgb

import matplotlib.pyplot as plt
import gc
```

Figure 2 All the Necessary Libraries

Figure 1 shows the commands to follow for YOLO installation while Figure 2 shows all the other libraries which needs to be installed before further processing.

3 Data Collection

To implement the study, a publicly available dataset called 'UCF-Crime' was used. The data consisted of 1900 surveillance videos which were 128 hours long in total. These videos consisted of 13 classes of suspicious activities such as Abuse, Arrest, Vandalism, etc. https://www.dropbox.com/sh/75v5ehq4cdg5g5g/AABvnJSwZI7zXb8_myBA0CLHa?dl=0)



Anomaly-Detection-Dataset
from Chen Chen (UNC Charlotte) via Dropbox Business

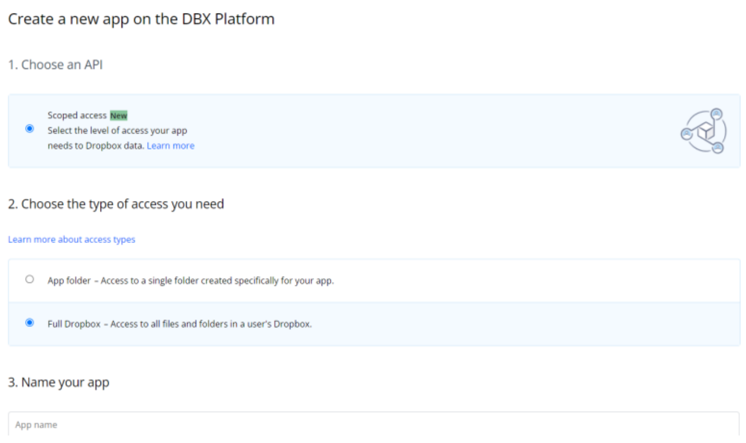
Copy to Dropbox | Download

Name	Modified
Anomaly_Train.txt	19/1/2019 7:29 pm
Anomaly-Videos-Part-1.zip	18/9/2018 12:42 am
Anomaly-Videos-Part-2.zip	18/9/2018 12:48 am
Anomaly-Videos-Part-3.zip	18/9/2018 12:53 am
Anomaly-Videos-Part-4.zip	18/9/2018 12:59 am
Normal_Videos_for_Event_Recognition.zip	18/9/2018 3:29 am
ReadMe-Anomaly-Detection.txt	6/10/2018 3:23 pm

Figure 3 Dataset

As the data is almost 100GB's of size it is present on the open Dropbox link provided by creators. This data needs to be copied onto our personal dropbox for the purpose of implementation.
To download the data onto google colab Dropbox API needs to be used using python. To access the data a scoped application needs to be created on Dropbox developer website. The steps are as follows: -

1. Navigate to the Dropbox App Console at https://www.dropbox.com/developers/apps/ .
2. Select the "Create app" option and select "Scoped access" from the menu.



Create a new app on the DBX Platform

1. Choose an API

[✓ Scoped access New
Select the level of access your app needs to Dropbox data. Learn more]

2. Choose the type of access you need

Learn more about access types

App folder – Access to a single folder created specifically for your app.

[✓ Full Dropbox – Access to all files and folders in a user's Dropbox.]

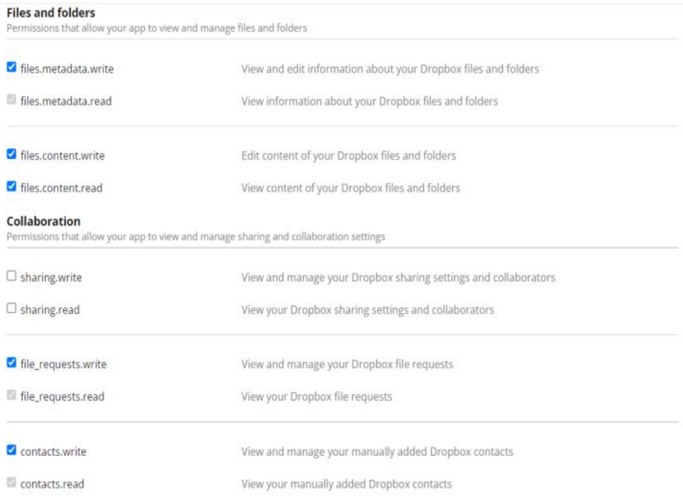
3. Name your app

App name

Figure 3 Creating Scoped App on Dropbox Developer Page

3. Select the type of access which is required (like accessing just files and folders, or team member file access or full Dropbox access etc).

4. Name the app and navigate to the permission sections.



Files and folders
Permissions that allow your app to view and manage files and folders

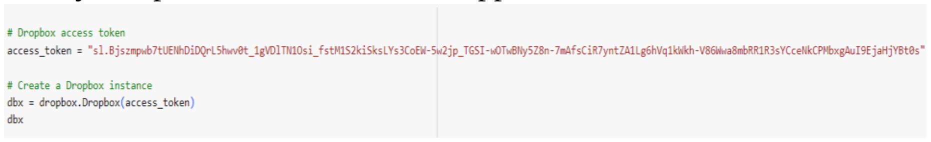
✓ files.metadata.write	View and edit information about your Dropbox files and folders
✓ files.metadata.read	View information about your Dropbox files and folders
✓ files.content.write	Edit content of your Dropbox files and folders
✓ files.content.read	View content of your Dropbox files and folders

Collaboration
Permissions that allow your app to view and manage sharing and collaboration settings

☐ sharing.write	View and manage your Dropbox sharing settings and collaborators
☐ sharing.read	View your Dropbox sharing settings and collaborators
✓ file_requests.write	View and manage your Dropbox file requests
✓ file_requests.read	View your Dropbox file requests
✓ contacts.write	View and manage your manually added Dropbox contacts
✓ contacts.read	View your manually added Dropbox contacts

Figure 4 Granting Permission

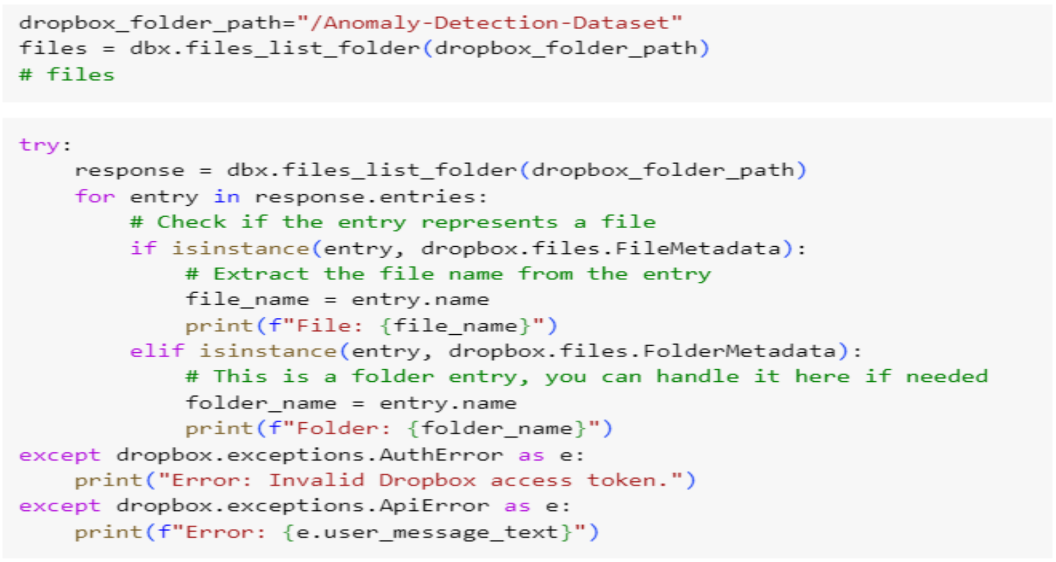
5. Provide permission for read/write.
6. After the permissions are given navigate to settings section and click on 'Generate' key.
7. Copy that key and paste it in below code snippet.



```
# Dropbox access token
access_token = "sl.Bjszmpwb7tUENhDiDQrL5hwv0t_1gVDlTN1Osi_fstM1S2kiSksLYs3CoEW-5w2jp_TGSI-wOTwBNy5Z8n-7mAfsCiR7yntZA1Lg6hVq1kWkh-V86Wwa8mbRR1R3sYCceNkCPMbxgAuI9EjaHjYBt0s"

# Create a Dropbox instance
dbx = dropbox.Dropbox(access_token)
dbx
```

Figure 5. Access the data using access token

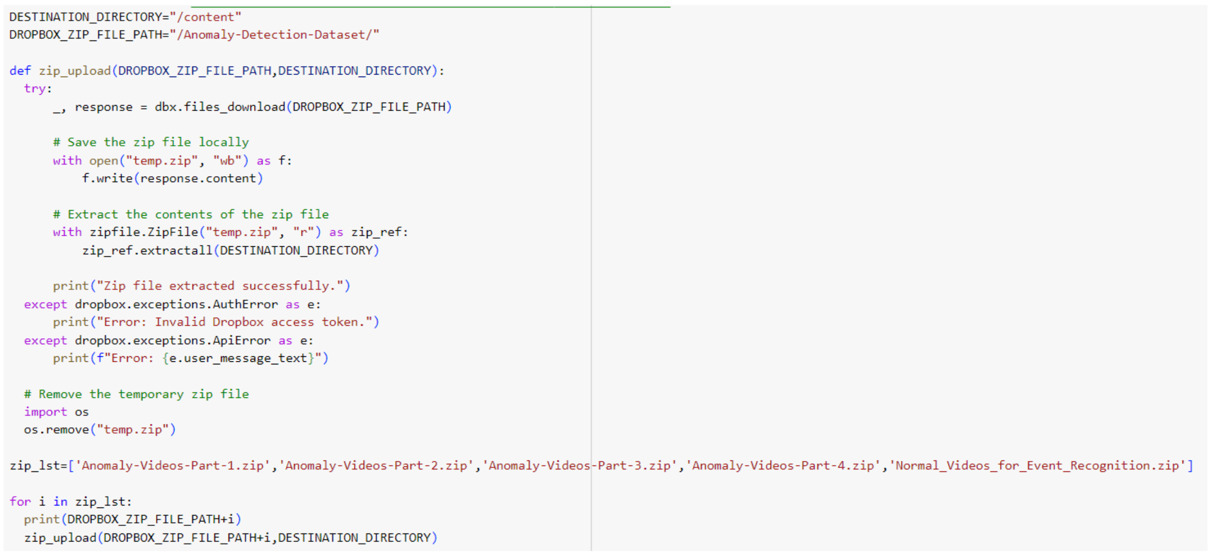
After the above code is executed, the list of files in the Dropbox can be seen using the code below. The code uses Dropbox APIs to download the data files from the Dropbox to google colab.



```
dropbox_folder_path="/Anomaly-Detection-Dataset"
files = dbx.files_list_folder(dropbox_folder_path)
# files
```

```
try:
    response = dbx.files_list_folder(dropbox_folder_path)
    for entry in response.entries:
        # Check if the entry represents a file
        if isinstance(entry, dropbox.files.FileMetadata):
            # Extract the file name from the entry
            file_name = entry.name
            print(f"File: {file_name}")
        elif isinstance(entry, dropbox.files.FolderMetadata):
            # This is a folder entry, you can handle it here if needed
            folder_name = entry.name
            print(f"Folder: {folder_name}")
except dropbox.exceptions.AuthError as e:
    print("Error: Invalid Dropbox access token.")
except dropbox.exceptions.ApiError as e:
    print(f"Error: {e.user_message_text}")
```

Figure 6. Downloading the files on google colab

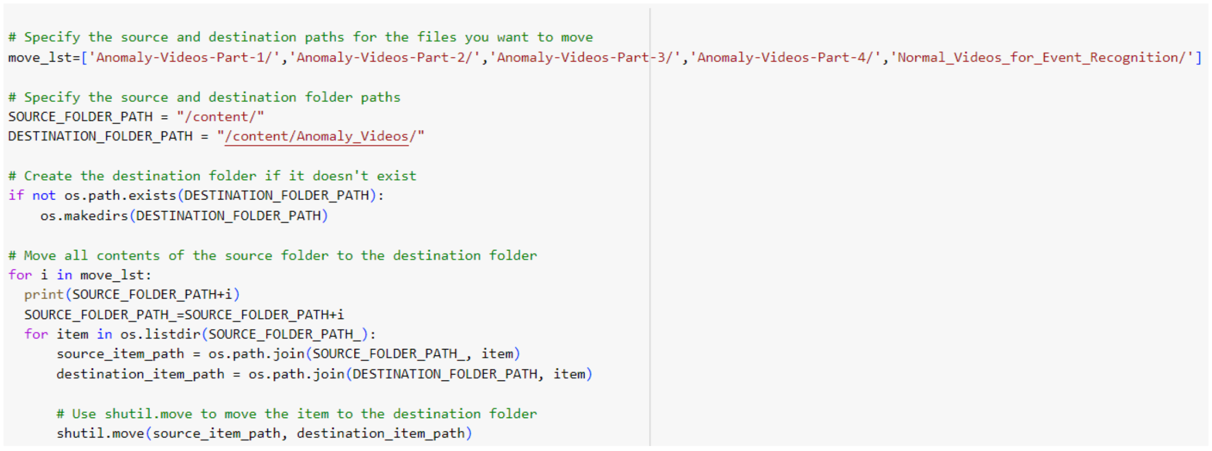


```python
DESTINATION_DIRECTORY="/content"
DROPBOX_ZIP_FILE_PATH="/Anomaly-Detection-Dataset/"

def zip_upload(DROPBOX_ZIP_FILE_PATH,DESTINATION_DIRECTORY):
  try:
      _, response = dbx.files_download(DROPBOX_ZIP_FILE_PATH)

      # Save the zip file locally
      with open("temp.zip", "wb") as f:
          f.write(response.content)

      # Extract the contents of the zip file
      with zipfile.ZipFile("temp.zip", "r") as zip_ref:
          zip_ref.extractall(DESTINATION_DIRECTORY)

      print("Zip file extracted successfully.")
  except dropbox.exceptions.AuthError as e:
      print("Error: Invalid Dropbox access token.")
  except dropbox.exceptions.ApiError as e:
      print(f"Error: {e.user_message_text}")

  # Remove the temporary zip file
  import os
  os.remove("temp.zip")

zip_lst=['Anomaly-Videos-Part-1.zip','Anomaly-Videos-Part-2.zip','Anomaly-Videos-Part-3.zip','Anomaly-Videos-Part-4.zip','Normal_Videos_for_Event_Recognition.zip']

for i in zip_lst:
  print(DROPBOX_ZIP_FILE_PATH+i)
  zip_upload(DROPBOX_ZIP_FILE_PATH+i,DESTINATION_DIRECTORY)
```

Figure 7. Extracting the downloaded files

s

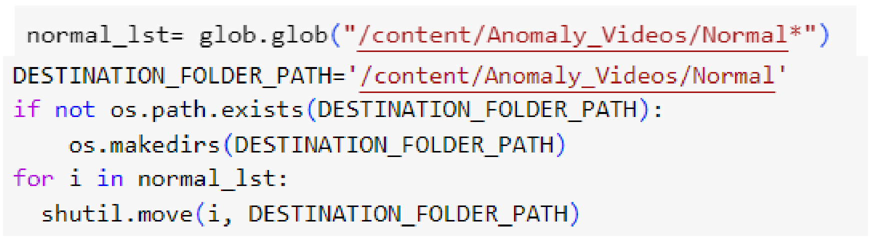
In the next step, the data zip files are extracted, and the folders of the same name are created (Figure 6).



```python
# Specify the source and destination paths for the files you want to move
move_lst=['Anomaly-Videos-Part-1/','Anomaly-Videos-Part-2/','Anomaly-Videos-Part-3/','Anomaly-Videos-Part-4/','Normal_Videos_for_Event_Recognition/']

# Specify the source and destination folder paths
SOURCE_FOLDER_PATH = "/content/"
DESTINATION_FOLDER_PATH = "/content/Anomaly_Videos/"

# Create the destination folder if it doesn't exist
if not os.path.exists(DESTINATION_FOLDER_PATH):
    os.makedirs(DESTINATION_FOLDER_PATH)

# Move all contents of the source folder to the destination folder
for i in move_lst:
  print(SOURCE_FOLDER_PATH+i)
  SOURCE_FOLDER_PATH_=SOURCE_FOLDER_PATH+i
  for item in os.listdir(SOURCE_FOLDER_PATH_):
      source_item_path = os.path.join(SOURCE_FOLDER_PATH_, item)
      destination_item_path = os.path.join(DESTINATION_FOLDER_PATH, item)

      # Use shutil.move to move the item to the destination folder
      shutil.move(source_item_path, destination_item_path)
```

Figure 8. Moving all class files in one folder

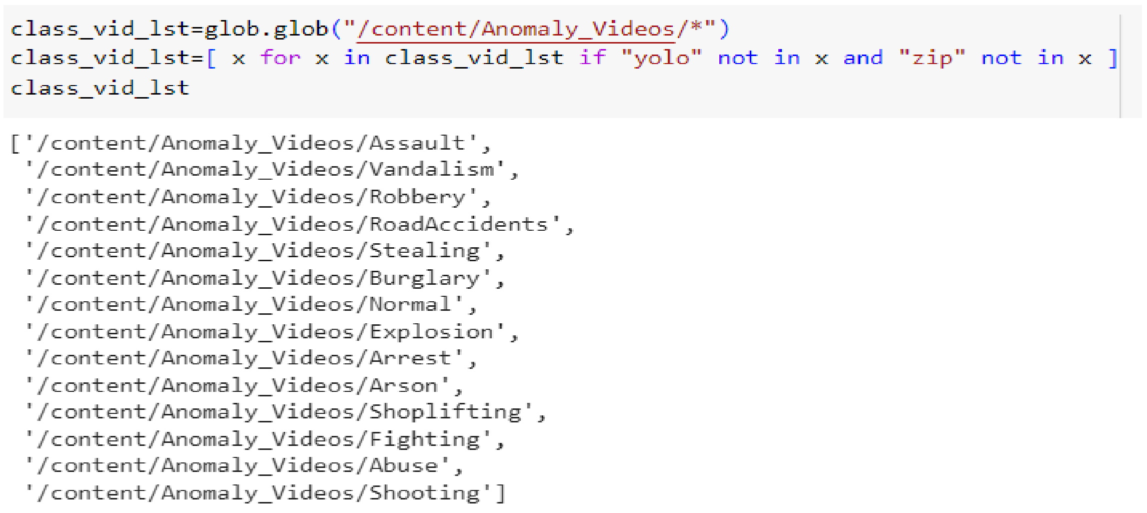
The data is divided into 5 folders. The first four folders starting with the name anomaly consist of videos of 13 different types while the fifth folder consists of normal event videos. As these classes are in different folders, these folders are moved into one single folder.



```python
normal_lst= glob.glob("/content/Anomaly_Videos/Normal*")
DESTINATION_FOLDER_PATH='/content/Anomaly_Videos/Normal'
if not os.path.exists(DESTINATION_FOLDER_PATH):
    os.makedirs(DESTINATION_FOLDER_PATH)
for i in normal_lst:
  shutil.move(i, DESTINATION_FOLDER_PATH)
```

Figure 9. Moving the Normal event files

The same is done with normal event videos. They are also moved into the same folder with all the other classes. After all the videos are moved in appropriate places, we can see the video list using the snippet below.

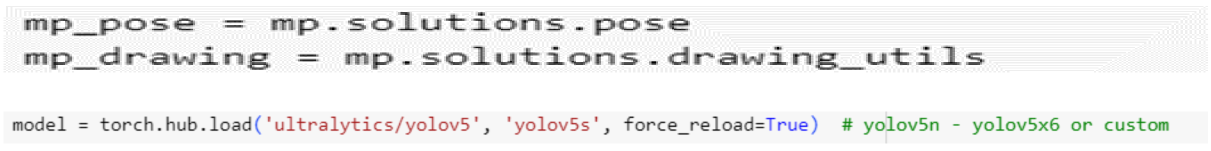


```
class_vid_lst=glob.glob("/content/Anomaly_Videos/*")
class_vid_lst=[ x for x in class_vid_lst if "yolo" not in x and "zip" not in x ]
class_vid_lst
```

```
['/content/Anomaly_Videos/Assault',
 '/content/Anomaly_Videos/Vandalism',
 '/content/Anomaly_Videos/Robbery',
 '/content/Anomaly_Videos/RoadAccidents',
 '/content/Anomaly_Videos/Stealing',
 '/content/Anomaly_Videos/Burglary',
 '/content/Anomaly_Videos/Normal',
 '/content/Anomaly_Videos/Explosion',
 '/content/Anomaly_Videos/Arrest',
 '/content/Anomaly_Videos/Arson',
 '/content/Anomaly_Videos/Shoplifting',
 '/content/Anomaly_Videos/Fighting',
 '/content/Anomaly_Videos/Abuse',
 '/content/Anomaly_Videos/Shooting']
```

Figure 10. List of all class files

4 Data Pre-Processing, Feature Creation and Transformation

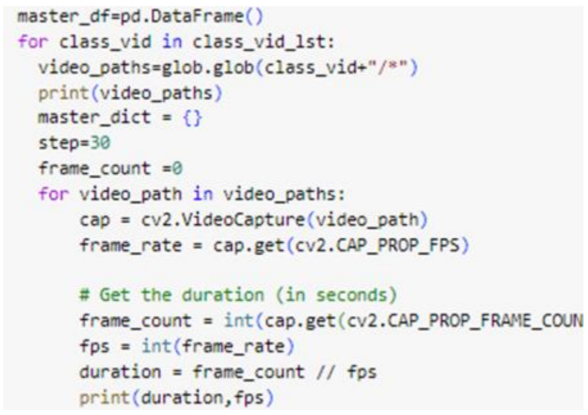


```
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
```

```
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', force_reload=True)  # yolov5n - yolov5x6 or custom
```

Figure 11. Loading of Mediapipe BlazePose and YOLO

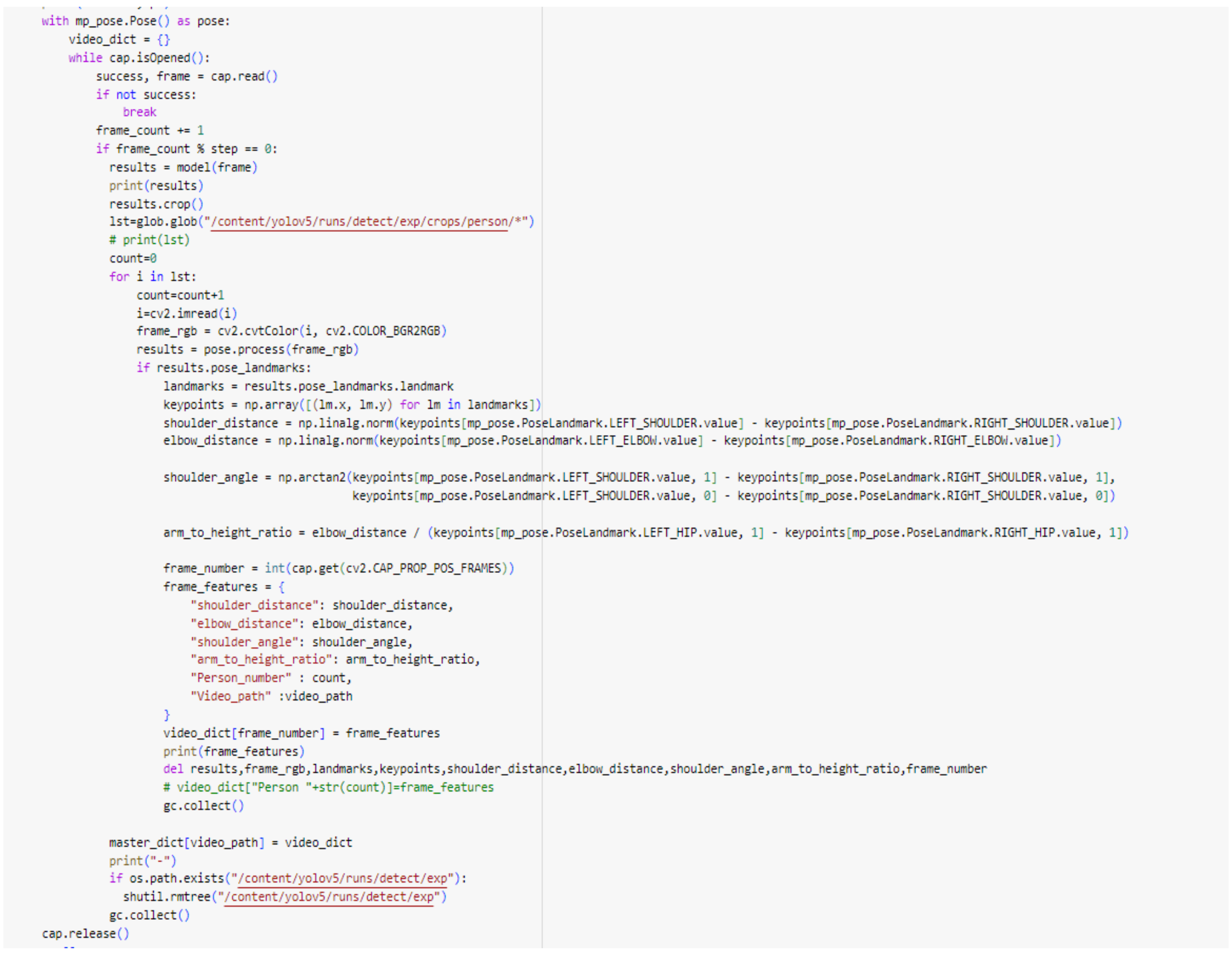
As the first step in data preprocessing, Mediapipe Blazepose model instance is initialized using mp. solution. pose. Similarly, for the people detection part in further script 'YOLO5' model is loaded.



```
master_df=pd.DataFrame()
for class_vid in class_vid_lst:
  video_paths=glob.glob(class_vid+"/*")
  print(video_paths)
  master_dict = {}
  step=30
  frame_count =0
  for video_path in video_paths:
      cap = cv2.VideoCapture(video_path)
      frame_rate = cap.get(cv2.CAP_PROP_FPS)

      # Get the duration (in seconds)
      frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUN
      fps = int(frame_rate)
      duration = frame_count // fps
      print(duration,fps)
```

Figure 12. Video preprocessing and information retrieval

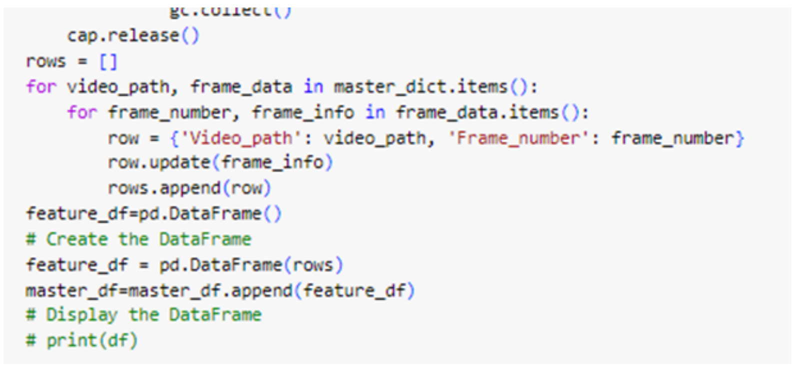
In the above nested loop, glob function is used to find out video paths in all the class folders. The above section of code is used to capture video information such as number of frames in total, frame rate per second and duration of video. The variable 'step' works as parameter in further code where it makes the algorithm consider every n'th frame for processing. Here in the above code 'n' is value 30.



```
with mp_pose.Pose() as pose:
    video_dict = {}
    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break
        frame_count += 1
        if frame_count % step == 0:
          results = model(frame)
          print(results)
          results.crop()
          lst=glob.glob("/content/yolov5/runs/detect/exp/crops/person/*")
          # print(lst)
          count=0
          for i in lst:
              count=count+1
              i=cv2.imread(i)
              frame_rgb = cv2.cvtColor(i, cv2.COLOR_BGR2RGB)
              results = pose.process(frame_rgb)
              if results.pose_landmarks:
                  landmarks = results.pose_landmarks.landmark
                  keypoints = np.array([(lm.x, lm.y) for lm in landmarks])
                  shoulder_distance = np.linalg.norm(keypoints[mp_pose.PoseLandmark.LEFT_SHOULDER.value] - keypoints[mp_pose.PoseLandmark.RIGHT_SHOULDER.value])
                  elbow_distance = np.linalg.norm(keypoints[mp_pose.PoseLandmark.LEFT_ELBOW.value] - keypoints[mp_pose.PoseLandmark.RIGHT_ELBOW.value])

                  shoulder_angle = np.arctan2(keypoints[mp_pose.PoseLandmark.LEFT_SHOULDER.value, 1] - keypoints[mp_pose.PoseLandmark.RIGHT_SHOULDER.value, 1],
                                              keypoints[mp_pose.PoseLandmark.LEFT_SHOULDER.value, 0] - keypoints[mp_pose.PoseLandmark.RIGHT_SHOULDER.value, 0])

                  arm_to_height_ratio = elbow_distance / (keypoints[mp_pose.PoseLandmark.LEFT_HIP.value, 1] - keypoints[mp_pose.PoseLandmark.RIGHT_HIP.value, 1])

                  frame_number = int(cap.get(cv2.CAP_PROP_POS_FRAMES))
                  frame_features = {
                      "shoulder_distance": shoulder_distance,
                      "elbow_distance": elbow_distance,
                      "shoulder_angle": shoulder_angle,
                      "arm_to_height_ratio": arm_to_height_ratio,
                      "Person_number" : count,
                      "Video_path" :video_path
                  }
                  video_dict[frame_number] = frame_features
                  print(frame_features)
                  del results,frame_rgb,landmarks,keypoints,shoulder_distance,elbow_distance,shoulder_angle,arm_to_height_ratio,frame_number
                  # video_dict["Person "+str(count)]=frame_features
                  gc.collect()

          master_dict[video_path] = video_dict
          print("-")
          if os.path.exists("/content/yolov5/runs/detect/exp"):
            shutil.rmtree("/content/yolov5/runs/detect/exp")
          gc.collect()
cap.release()
```

Figure 13. Video preprocessing and information retrieval

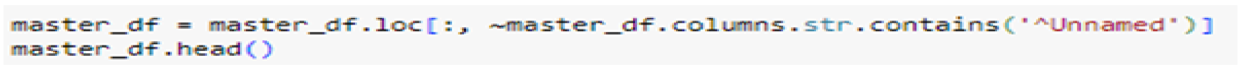
For the next step, for every video BlazePose model is initialized. For every video OpenCv opens a video cap and starts converting videos into frames. When the frame number matches the multiple of step variable provided, that frame is extracted for pose estimation. Then the YOLO model is applied on this frame using model(frame) function. After that the results are stored in ‘results. If there are multiple people in the frame, the crop () function of YOLO crops these people and stores it in the form of image at the path ‘content/yolo5/runs/ detect/exp/crops/ person’. Every image from the path is processed using OpenCV. Every cropped image is then converted from BGR format to RGB before being sent for pose estimation.
When sent to pose estimation, pose. landmarks are stored in variables called results. These landmarks consist of 33 keypoints in the form of x, y, z. These landmarks are then converted into customised geometric features such as shoulder distance, elbow distance, shoulder angle and arm to height ratio. After these features are calculated they are stored in the form of dictionary. After they are safely stored in dictionary all the unnecessary variables are deleted and garbage collector is called. After the processing is done for every frame, its cropped images are deleted for optimal use of storage.



```
            gc.collect()
    cap.release()
rows = []
for video_path, frame_data in master_dict.items():
    for frame_number, frame_info in frame_data.items():
        row = {'Video_path': video_path, 'Frame_number': frame_number}
        row.update(frame_info)
        rows.append(row)
feature_df=pd.DataFrame()
# Create the DataFrame
feature_df = pd.DataFrame(rows)
master_df=master_df.append(feature_df)
# Display the DataFrame
# print(df)
```

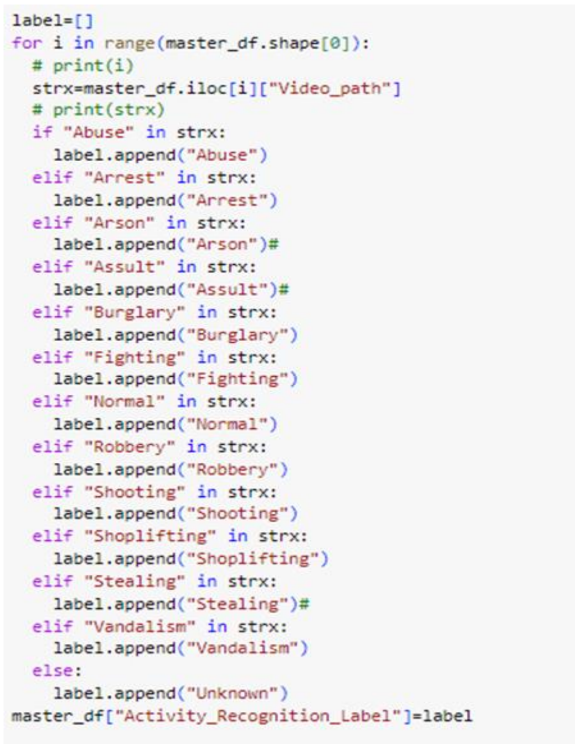
Figure 14. Store features into dataframe

For every frame the results are stored in a master dictionary and later combined into a dataframe.



```
master_df = master_df.loc[:, ~master_df.columns.str.contains('^Unnamed')]
master_df.head()
```

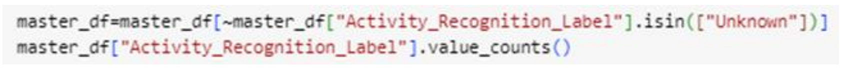
The dataframe is checked if any empty column has been generated in it or not before further processing.



```
label=[]
for i in range(master_df.shape[0]):
  # print(i)
  strx=master_df.iloc[i]["Video_path"]
  # print(strx)
  if "Abuse" in strx:
    label.append("Abuse")
  elif "Arrest" in strx:
    label.append("Arrest")
  elif "Arson" in strx:
    label.append("Arson")#
  elif "Assult" in strx:
    label.append("Assult")#
  elif "Burglary" in strx:
    label.append("Burglary")
  elif "Fighting" in strx:
    label.append("Fighting")
  elif "Normal" in strx:
    label.append("Normal")
  elif "Robbery" in strx:
    label.append("Robbery")
  elif "Shooting" in strx:
    label.append("Shooting")
  elif "Shoplifting" in strx:
    label.append("Shoplifting")
  elif "Stealing" in strx:
    label.append("Stealing")#
  elif "Vandalism" in strx:
    label.append("Vandalism")
  else:
    label.append("Unknown")
master_df["Activity_Recognition_Label"]=label
```

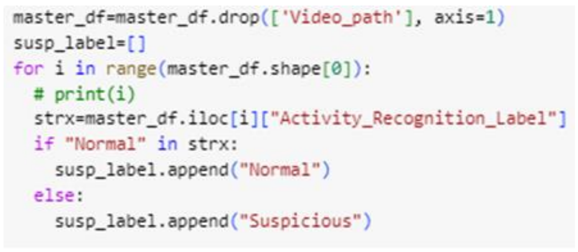
Figure 15. Label Annotation

After that, a column named ‘Activity_Recognition_Label’ is generated for the annotation of activities based on the video paths as those paths contain the information about subfolders. These subfolders' names contain the names of the classes.



```
master_df=master_df[~master_df["Activity_Recognition_Label"].isin(["Unknown"])]
master_df["Activity_Recognition_Label"].value_counts()
```

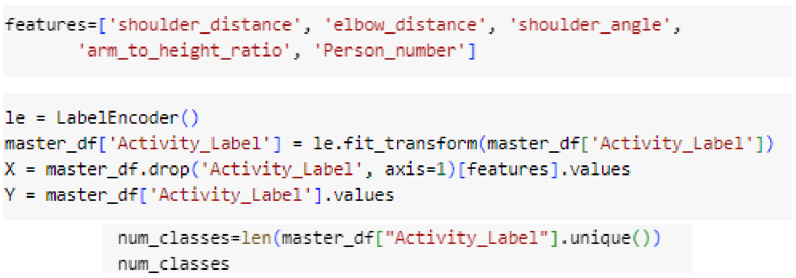
After names are converted the dataframe is checked again for ‘unknown’ labels.



```python
master_df=master_df.drop(['Video_path'], axis=1)
susp_label=[]
for i in range(master_df.shape[0]):
  # print(i)
  strx=master_df.iloc[i]["Activity_Recognition_Label"]
  if "Normal" in strx:
    susp_label.append("Normal")
  else:
    susp_label.append("Suspicious")
```

Figure 16. Changing Labels to Suspicious or Normal

Create another label column called ‘Activity_Label’ which contains information of if the activity is suspicious or not.
If we are trying to create a binary classification model,



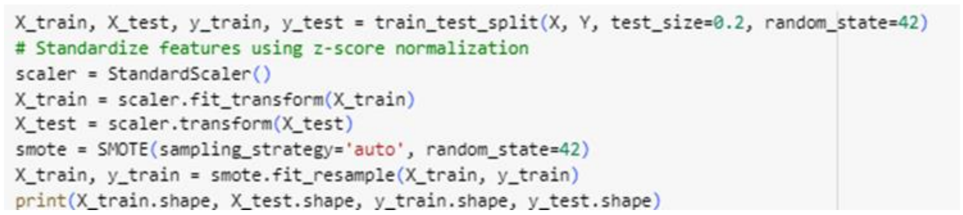
```python
features=['shoulder_distance', 'elbow_distance', 'shoulder_angle',
       'arm_to_height_ratio', 'Person_number']
```

```python
le = LabelEncoder()
master_df['Activity_Label'] = le.fit_transform(master_df['Activity_Label'])
X = master_df.drop('Activity_Label', axis=1)[features].values
Y = master_df['Activity_Label'].values
```

```python
num_classes=len(master_df["Activity_Label"].unique())
num_classes
```

Figure 17. Label Encoding

Here the labels of the data are converted from strings to numbers using a label encoder.



```python
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Standardize features using z-score normalization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Figure 18. Scaling and handling imbalance data

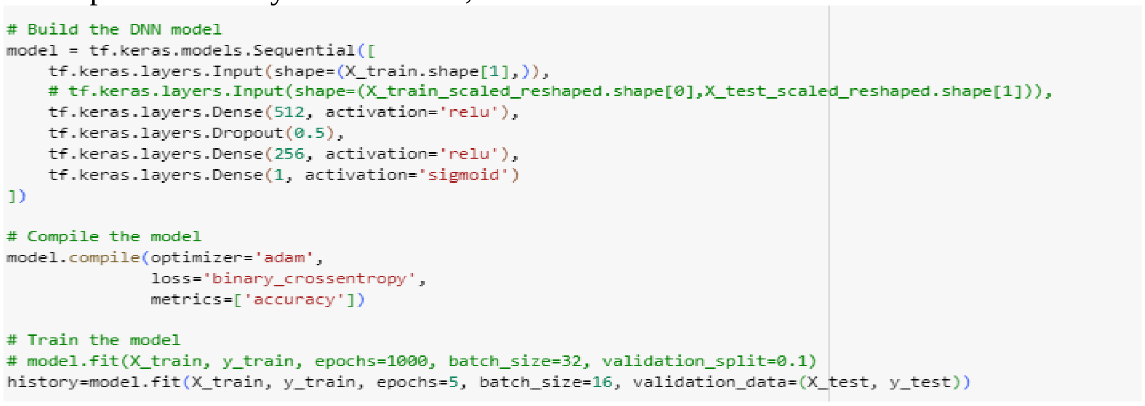
After label encoding the data is split into two parts training and testing. Later standard scalar is applied to the data for scaling the data between 0 to 1 so that one feature doesn’t dominate the model result because its range is high. Later SMOTE technology is used to create synthetic data which is integrated into original to address the problem of data imbalance.

5 Data Modelling and Evaluation

This study proposes to create two different models where the primary model is used to detect if the activity is suspicious or not and the second model is used to recognize the type of activity. All the machine learning models were created in the same way but for deep learning, the number of neurons and number of layers were changed to get better results.

5.1 DNN model

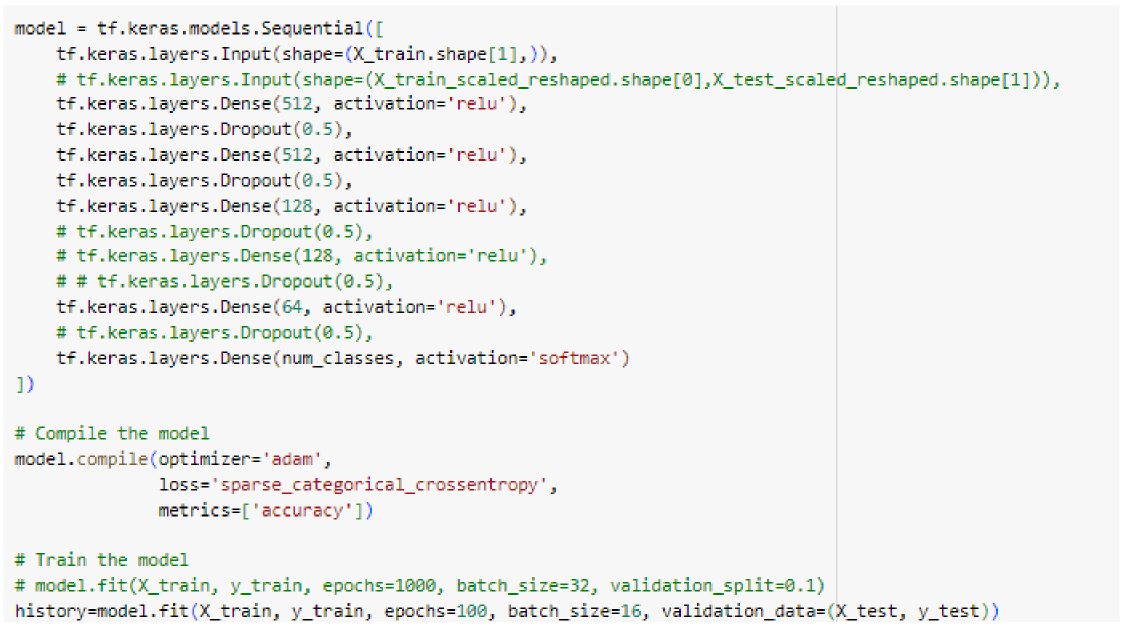
for suspicious activity identification,



```
# Build the DNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    # tf.keras.layers.Input(shape=(X_train_scaled_reshaped.shape[0],X_test_scaled_reshaped.shape[1])),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
# model.fit(X_train, y_train, epochs=1000, batch_size=32, validation_split=0.1)
history=model.fit(X_train, y_train, epochs=5, batch_size=16, validation_data=(X_test, y_test))
```

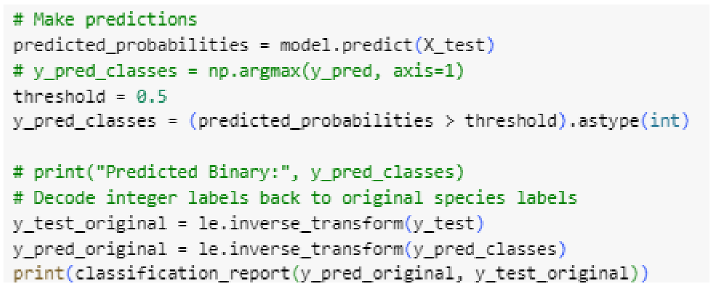
Figure 19. DNN to predict if actvity is suspicious or not

For activity type recognition,



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),
    # tf.keras.layers.Input(shape=(X_train_scaled_reshaped.shape[0],X_test_scaled_reshaped.shape[1])),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    # tf.keras.layers.Dropout(0.5),
    # tf.keras.layers.Dense(128, activation='relu'),
    # # tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu'),
    # tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
# model.fit(X_train, y_train, epochs=1000, batch_size=32, validation_split=0.1)
history=model.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=(X_test, y_test))
```

Figure 20. DNN to predict activity type



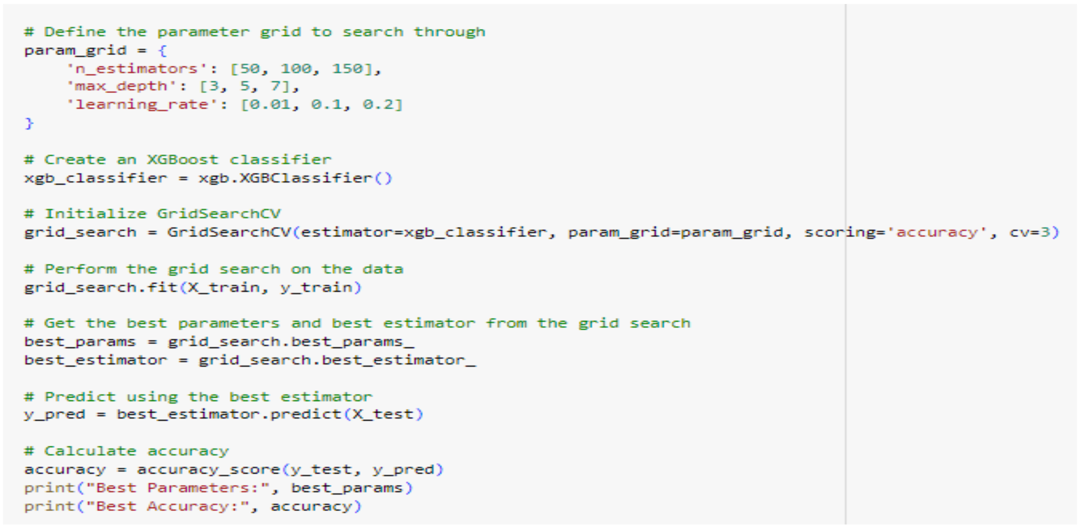
```
# Make predictions
predicted_probabilities = model.predict(X_test)
# y_pred_classes = np.argmax(y_pred, axis=1)
threshold = 0.5
y_pred_classes = (predicted_probabilities > threshold).astype(int)

# print("Predicted Binary:", y_pred_classes)
# Decode integer labels back to original species labels
y_test_original = le.inverse_transform(y_test)
y_pred_original = le.inverse_transform(y_pred_classes)
print(classification_report(y_pred_original, y_test_original))
```

Figure 21. Classification Report

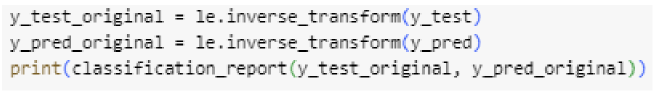
Here is the model. predict () function is used for predicting the data while le. inverse_transform () is used for inverse transformation of the converted labels.

5.2 XgBoost model

In the case of machine learning the models followed the same approach where gridsearch () was applied to models to find the best parameters which is then used to create the model. For evaluation part the models used classification report and accuracy score.



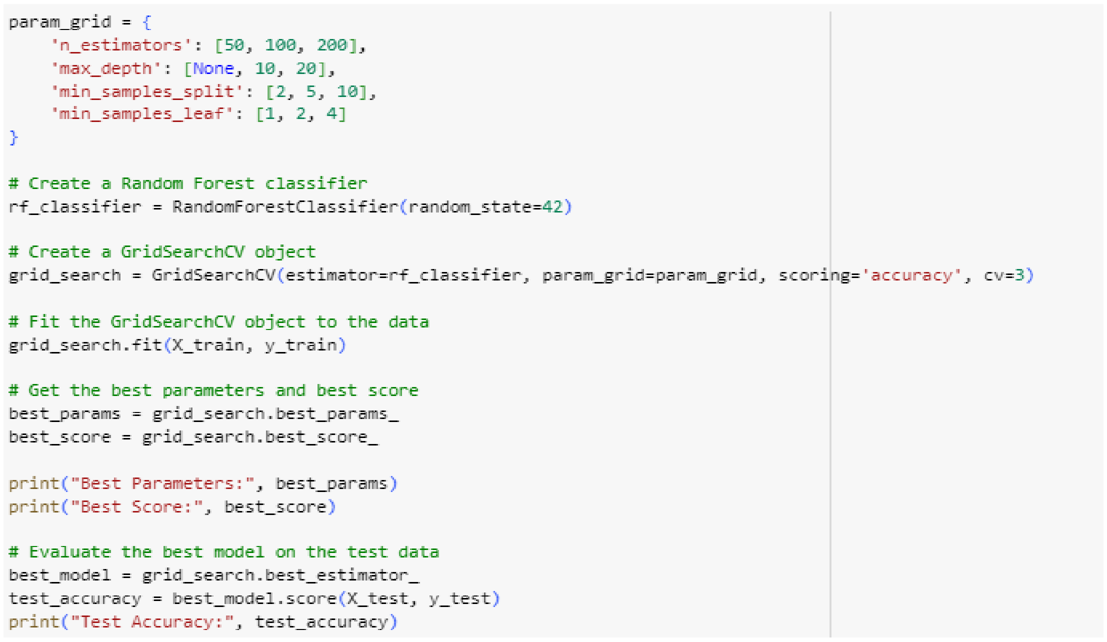
```
# Define the parameter grid to search through
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2]
}

# Create an XGBoost classifier
xgb_classifier = xgb.XGBClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, scoring='accuracy', cv=3)

# Perform the grid search on the data
grid_search.fit(X_train, y_train)

# Get the best parameters and best estimator from the grid search
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Predict using the best estimator
y_pred = best_estimator.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Best Parameters:", best_params)
print("Best Accuracy:", accuracy)
```



```
y_test_original = le.inverse_transform(y_test)
y_pred_original = le.inverse_transform(y_pred)
print(classification_report(y_test_original, y_pred_original))
```

Figure 22. XgBoost Model creation and Evaluation

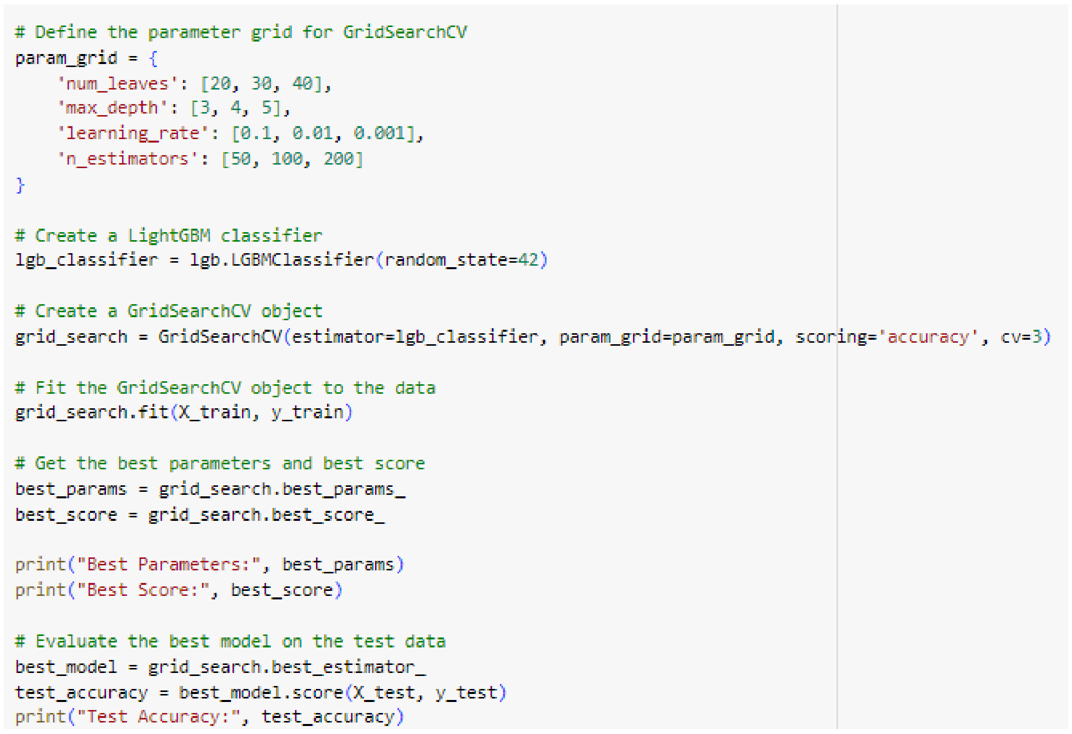
5.3 Random Forest Model



```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, scoring='accuracy', cv=3)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Evaluate the best model on the test data
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

Figure 23. Random Forest Model creation and Evaluation

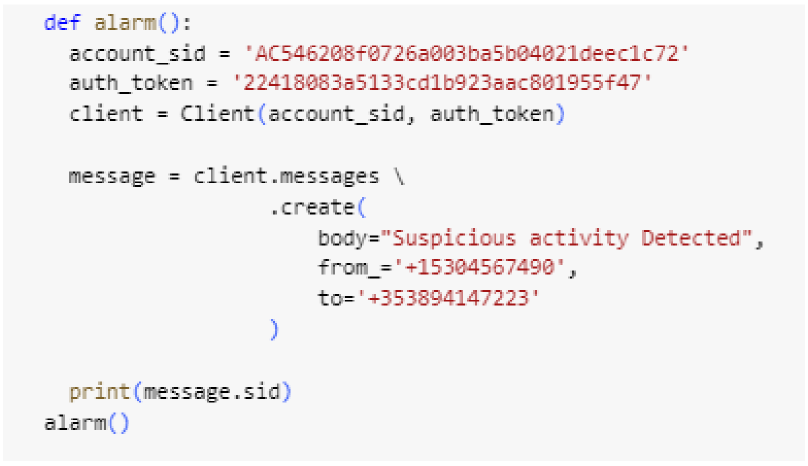
5.4 LightGBM Model



```
# Define the parameter grid for GridSearchCV
param_grid = {
    'num_leaves': [20, 30, 40],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [50, 100, 200]
}

# Create a LightGBM classifier
lgb_classifier = lgb.LGBMClassifier(random_state=42)

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=lgb_classifier, param_grid=param_grid, scoring='accuracy', cv=3)

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Evaluate the best model on the test data
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

Figure 24. LightGBM Model creation and Evaluation

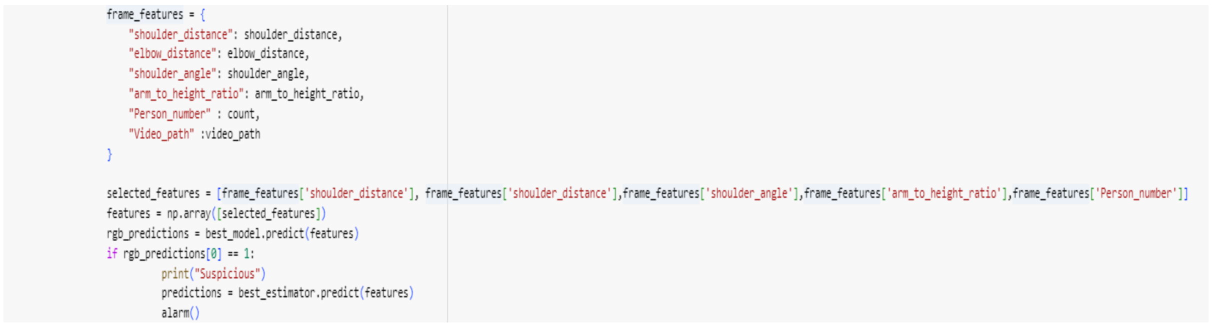
6 Alarm System



```
def alarm():
  account_sid = 'AC546208f0726a003ba5b04021deec1c72'
  auth_token = '22418083a5133cd1b923aac801955f47'
  client = Client(account_sid, auth_token)

  message = client.messages \
                .create(
                    body="Suspicious activity Detected",
                    from_='+15304567490',
                    to='+353894147223'
                )

  print(message.sid)
alarm()
```

Figure 25. Alarm system using Twilio

As an alarm system the framework uses twilio library of python. This library provides a facility of audio, video or email using python. To access this one must register on twilio and open a free account. The twilio provides an account number and authentication token which needs to be used as shown in the above diagram.



```
frame_features = {
    "shoulder_distance": shoulder_distance,
    "elbow_distance": elbow_distance,
    "shoulder_angle": shoulder_angle,
    "arm_to_height_ratio": arm_to_height_ratio,
    "Person_number" : count,
    "Video_path" :video_path
}

selected_features = [frame_features['shoulder_distance'], frame_features['shoulder_distance'],frame_features['shoulder_angle'],frame_features['arm_to_height_ratio'],frame_features['Person_number']]
features = np.array([selected_features])
rgb_predictions = best_model.predict(features)
if rgb_predictions[0] == 1:
        print("Suspicious")
        predictions = best_estimator.predict(features)
        alarm()
```

Figure 26. Real time prediction

The above method is called when multiple frames at a time are predicted as suspicious consecutively.