**Backend LSTM integration and Testing**

**Loading Pre-trained Model and Model Integration**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.models import load\_model

import tensorflow as tf

from tensorflow.keras.layers import LSTM, Dense, GlobalAveragePooling3D, Reshape

# Load the pre-trained R(2+1)D model

r2plus1d\_model = tf.keras.models.load\_model(r"E:\Final Year Project\fyp\r\_2\_1\_d\_3d\_cnn\_model\_final.h5",

custom\_objects={'Conv2Plus1D': Conv2Plus1D,

'ResizeVideo': ResizeVideo,

'ResidualMain': ResidualMain,

'Project': Project})

# Freeze the weights of the pre-trained R(2+1)D layers

for layer in r2plus1d\_model.layers:

layer.trainable = False

# Reshape the output of the R(2+1)D model to include spatial dimensions

reshaped\_output = Reshape((-1, r2plus1d\_model.output\_shape[-1]))(r2plus1d\_model.output)

# Add LSTM layer on top of the reshaped output

lstm\_units = 64

lstm\_layer = LSTM(lstm\_units)(reshaped\_output)

# Add classification layer

num\_classes = 2 # Number of classes: 'fights' and 'nofights'

output\_layer = Dense(num\_classes, activation='softmax')(lstm\_layer)

# Create the combined model

combined\_model = tf.keras.Model(inputs=r2plus1d\_model.input, outputs=output\_layer)

# Compile the model

combined\_model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Print model summary

combined\_model.summary()

**Training**

# Assuming you have already loaded and compiled your combined\_model

# Define your dataset paths

train\_path = r"F:\FYP\Real Life Violence Dataset subset\train"

val\_path = r"F:\FYP\Real Life Violence Dataset subset\val"

test\_path = r"F:\FYP\Real Life Violence Dataset subset\test"

# Define other parameters

batch\_size = 32

epochs = 10

# Create TensorFlow datasets

train\_ds = tf.data.Dataset.from\_generator(FrameGenerator(train\_path, n\_frames, training=True),

output\_signature=output\_signature)

train\_ds = train\_ds.batch(batch\_size)

val\_ds = tf.data.Dataset.from\_generator(FrameGenerator(val\_path, n\_frames),

output\_signature=output\_signature)

val\_ds = val\_ds.batch(batch\_size)

test\_ds = tf.data.Dataset.from\_generator(FrameGenerator(test\_path, n\_frames),

output\_signature=output\_signature)

test\_ds = test\_ds.batch(batch\_size)

# Train the model

history = combined\_model.fit(train\_ds,

epochs=epochs,

validation\_data=val\_ds)

# Evaluate the model

loss, accuracy = combined\_model.evaluate(test\_ds)

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

**Training Output**  
Epoch 1/10

50/50 [==============================] - 1008s 20s/step - loss: 0.3574 - accuracy: 0.9187 - val\_loss: 0.1858 - val\_accuracy: 0.9312

Epoch 2/10

50/50 [==============================] - 1046s 21s/step - loss: 0.1559 - accuracy: 0.9464 - val\_loss: 0.1311 - val\_accuracy: 0.9518

Epoch 3/10

50/50 [==============================] - 1420s 29s/step - loss: 0.1330 - accuracy: 0.9477 - val\_loss: 0.1271 - val\_accuracy: 0.9404

Epoch 4/10

50/50 [==============================] - 1970s 40s/step - loss: 0.1349 - accuracy: 0.9489 - val\_loss: 0.1502 - val\_accuracy: 0.9381

Epoch 5/10

50/50 [==============================] - 1414s 28s/step - loss: 0.1318 - accuracy: 0.9489 - val\_loss: 0.1190 - val\_accuracy: 0.9610

Epoch 6/10

50/50 [==============================] - 1166s 23s/step - loss: 0.1243 - accuracy: 0.9533 - val\_loss: 0.1210 - val\_accuracy: 0.9564

Epoch 7/10

50/50 [==============================] - 1049s 21s/step - loss: 0.1264 - accuracy: 0.9552 - val\_loss: 0.1271 - val\_accuracy: 0.9587

Epoch 8/10

50/50 [==============================] - 904s 18s/step - loss: 0.1253 - accuracy: 0.9540 - val\_loss: 0.1515 - val\_accuracy: 0.9289

Epoch 9/10

50/50 [==============================] - 891s 18s/step - loss: 0.1294 - accuracy: 0.9477 - val\_loss: 0.1169 - val\_accuracy: 0.9587

Epoch 10/10

50/50 [==============================] - 1033s 21s/step - loss: 0.1233 - accuracy: 0.9540 - val\_loss: 0.1225 - val\_accuracy: 0.9472

14/14 [==============================] - 250s 18s/step - loss: 0.1229 - accuracy: 0.9476

Test Loss: 0.12289391458034515

Test Accuracy: 0.9476081728935242

**Testing Using WebCam**

import cv2

import numpy as np

import tensorflow as tf

# Define the function to preprocess frames

def format\_frames(frame, output\_size):

"""

Pad and resize an image from a video.

Args:

frame: Image that needs to be resized and padded.

output\_size: Pixel size of the output frame image.

Return:

Formatted frame with padding of specified output size.

"""

frame = tf.image.convert\_image\_dtype(frame, tf.float32)

frame = tf.image.resize\_with\_pad(frame, \*output\_size)

return frame

# Define the function to predict violence in a video

def predict\_violence(frames, model):

frames = np.array(frames)

predictions = model.predict(frames)

return predictions

# Load the trained model

model = new\_model # You need to have `new\_model` defined with your trained model

# Function to determine violence detection based on predictions

def detect\_violence(predictions):

violence\_scores = [pred[:2] for pred in predictions]

average\_scores = np.mean(np.array(violence\_scores), axis=0)

if average\_scores[0] > average\_scores[1]:

return True

else:

return False

# Function to mark violence in the video

def mark\_violence(frame, is\_violent):

if is\_violent:

# Draw a red rectangle on the frame

cv2.rectangle(frame, (0, 0), (frame.shape[1], frame.shape[0]), (0, 0, 255), thickness=2) # Adjust thickness

cv2.putText(frame, "Violence Detected", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

else:

# Draw a green rectangle on the frame

cv2.rectangle(frame, (0, 0), (frame.shape[1], frame.shape[0]), (0, 255, 0), thickness=2) # Adjust thickness

cv2.putText(frame, "No Violence Detected", (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2)

return frame

# Function to preprocess frames for prediction

def preprocess\_frame(frame):

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame = cv2.resize(frame, (224, 224)) # Resize the frame to match the model's input size

frame = frame.astype(np.float32) / 255.0 # Normalize pixel values to [0, 1]

return frame

# Start video capture from default camera (0)

cap = cv2.VideoCapture(0)

buffer\_size = 10 # Number of frames to consider for prediction

frame\_buffer = []

while True:

ret, frame = cap.read()

if not ret:

break

# Preprocess the frame

processed\_frame = preprocess\_frame(frame)

# Add the processed frame to the buffer

frame\_buffer.append(processed\_frame)

# If the buffer is filled, make a prediction

if len(frame\_buffer) == buffer\_size:

# Convert the frame buffer to a numpy array

frame\_sequence = np.array(frame\_buffer)

# Predict violence

predictions = model.predict(np.expand\_dims(frame\_sequence, axis=0))

violence\_detected = detect\_violence(predictions)

# Mark violence in the frame

marked\_frame = mark\_violence(frame, violence\_detected)

# Display the frame

cv2.imshow('Video', marked\_frame)

# Clear the buffer for the next sequence of frames

frame\_buffer = []

# Break the loop if 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release video capture

cap.release()

cv2.destroyAllWindows()

**Output**

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 971ms/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 0s 474ms/step

1/1 [==============================] - 0s 360ms/step

1/1 [==============================] - 0s 350ms/step

1/1 [==============================] - 0s 350ms/step

1/1 [==============================] - 0s 359ms/step

1/1 [==============================] - 0s 355ms/step

1/1 [==============================] - 0s 351ms/step

1/1 [==============================] - 0s 368ms/step

1/1 [==============================] - 0s 378ms/step

1/1 [==============================] - 0s 356ms/step

1/1 [==============================] - 0s 373ms/step

1/1 [==============================] - 0s 390ms/step

1/1 [==============================] - 0s 365ms/step

1/1 [==============================] - 0s 385ms/step

1/1 [==============================] - 0s 384ms/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 1s 1s/step

1/1 [==============================] - 0s 363ms/step

1/1 [==============================] - 0s 364ms/step

1/1 [==============================] - 0s 352ms/step

1/1 [==============================] - 0s 352ms/step

1/1 [==============================] - 0s 373ms/step

1/1 [==============================] - 0s 357ms/step

1/1 [==============================] - 0s 418ms/step

1/1 [==============================] - 0s 368ms/step

1/1 [==============================] - 0s 386ms/step

1/1 [==============================] - 0s 373ms/step

1/1 [==============================] - 0s 380ms/step

1/1 [==============================] - 0s 401ms/step

**Testing Using Video Input**

import cv2

import numpy as np

import tensorflow as tf

from pathlib import Path

# Define the function to preprocess frames

def format\_frames(frame, output\_size):

"""

Pad and resize an image from a video.

Args:

frame: Image that needs to be resized and padded.

output\_size: Pixel size of the output frame image.

Return:

Formatted frame with padding of specified output size.

"""

frame = tf.image.convert\_image\_dtype(frame, tf.float32)

frame = tf.image.resize\_with\_pad(frame, \*output\_size)

return frame

# Define the function to predict violence in a video

def predict\_violence(video\_path, model, n\_frames=10, output\_size=(224, 224), batch\_size=8):

# Read the video file

src = cv2.VideoCapture(video\_path)

video\_length = int(src.get(cv2.CAP\_PROP\_FRAME\_COUNT))

# Create batches of frames

frames\_batch = []

for \_ in range(0, video\_length, n\_frames):

batch\_frames = []

for \_ in range(n\_frames):

ret, frame = src.read()

if not ret:

break

frame = format\_frames(frame, output\_size)

batch\_frames.append(frame)

frames\_batch.append(batch\_frames)

src.release()

# Pad the last batch if necessary

if len(frames\_batch[-1]) < n\_frames:

padding = n\_frames - len(frames\_batch[-1])

frames\_batch[-1].extend([np.zeros\_like(frames\_batch[-1][0])] \* padding)

# Convert to numpy array

frames\_batch = np.array(frames\_batch)

# Predict violence for each batch

predictions = []

for batch in range(0, len(frames\_batch), batch\_size):

batch\_predictions = model.predict(frames\_batch[batch:batch+batch\_size])

predictions.extend(batch\_predictions)

return predictions

# Load the trained model

model = new\_model

# Path to the video file you want to predict

video\_path = r"F:\FYP\Real Life Violence Dataset\nofights\NV\_15.mp4" # Replace 'path\_to\_video.mp4' with the actual path to your video file

# Predict violence in the video

violence\_predictions = predict\_violence(video\_path, model)

violence\_scores = [pred[:2] for pred in violence\_predictions]

# Print the predictions

print(violence\_scores)

# Calculate the mean of both classes from the output predictions

average\_scores = np.mean(np.array(violence\_scores), axis=0)

# Print the average scores

print("Average violence score:", average\_scores[0])

print("Average non-violence score:", average\_scores[1])

# Function to determine violence detection based on predictions

def detect\_violence(average\_scores\_0, average\_scores\_1):

if average\_scores[0] > average\_scores[1]:

return True

else:

return False

# Determine violence detection based on predictions

violence\_detected = detect\_violence(average\_scores[0], average\_scores[1])

# Display the result

if violence\_detected:

print("Violence detected in the video.")

# Extract frames and store them in the output folder

output\_folder = r"E:\Frames"

if not Path(output\_folder).exists():

Path(output\_folder).mkdir(parents=True, exist\_ok=True)

# Read the video file

src = cv2.VideoCapture(video\_path)

frame\_count = 0

while True:

ret, frame = src.read()

if not ret:

break

frame\_count += 1

frame\_path = Path(output\_folder) / f"frame\_{frame\_count}.jpg"

cv2.imwrite(str(frame\_path), frame)

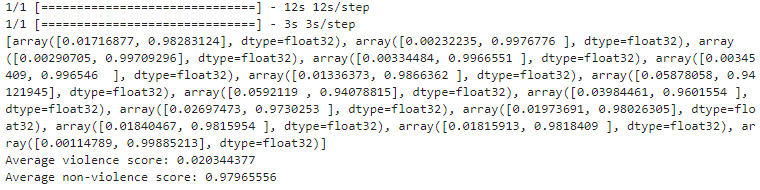
src.release()

print("Frames extracted and stored in the output folder.")

else:

print("No violence detected in the video.")

**Output**

****

****