FYP code section.

Google Colab code is based on the reference. And models without convLSTM and LRCN model have been created by myself. Also functions after ‘!pip install pytube’ has been written by myself. All FastAPI and Flutter codes were written by myself as well.

FastAPI code – vsCode environment has been used

As FastAPI is based on Python, it is recommended to use virtual environment in order to deal with multiple dependencies which were already set in before Python project. Also the version of tensorflow should be same as the version used in Google colab. Otherwise, model is not deployed with an error. For other packages can be installed with recent version.

# YouTube video download function

def download\_youtube\_videos(youtube\_url, output\_path):

"""

Downloads the highest resolution progressive video from a YouTube URL.

Args:

youtube\_url (str): URL of the YouTube video to download.

output\_path (str): Directory where the downloaded video will be saved.

Returns:

str: Filename of the downloaded video.

"""

yt = YouTube(youtube\_url)

stream = yt.streams.filter(progressive=True, file\_extension='mp4').order\_by('resolution').desc().first()

video\_title = stream.default\_filename

original\_video\_title = video\_title

video\_number = 1

while os.path.exists(os.path.join(output\_path, video\_title)):

video\_number += 1

video\_title = f"{original\_video\_title.split('.mp4')[0]}\_{video\_number}.mp4"

stream.download(output\_path=output\_path, filename=video\_title)

return video\_title

# Frame extraction from video function

def frames\_extraction(video\_path: str, sequence\_length: int = SEQUENCE\_LENGTH) -> List[np.ndarray]:

"""

Extracts frames evenly spaced from a video file, resizing them and normalizing pixel values.

Args:

video\_path (str): Path to the video file.

sequence\_length (int): Number of frames to extract, ensuring they are evenly spaced throughout the video.

Returns:

List[np.ndarray]: A list of processed frames (numpy arrays).

"""

frames\_list = []

video\_reader = cv2.VideoCapture(video\_path)

video\_frames\_count = int(video\_reader.get(cv2.CAP\_PROP\_FRAME\_COUNT))

frame\_interval = int(np.floor(video\_frames\_count / sequence\_length)) if video\_frames\_count >= sequence\_length else 1

for frame\_count in range(sequence\_length):

video\_reader.set(cv2.CAP\_PROP\_POS\_FRAMES, frame\_count \* frame\_interval)

success, frame = video\_reader.read()

if not success:

break

resized\_frame = cv2.resize(frame, (IMAGE\_HEIGHT, IMAGE\_WIDTH))

normalized\_frame = resized\_frame / 255.0

frames\_list.append(normalized\_frame)

video\_reader.release()

return frames\_list

# Continuous prediction and overlay function

def continuous\_predict\_and\_overlay(input\_video\_path, model, output\_video\_path):

"""

Reads video, performs predictions on a sequence of frames, and overlays predictions on the video.

Args:

input\_video\_path (str): Path to the input video.

model (Model): Trained machine learning model for predictions.

output\_video\_path (str): Path where the output video with overlaid predictions will be saved.

"""

cap = cv2.VideoCapture(input\_video\_path)

fps, width, height = cap.get(cv2.CAP\_PROP\_FPS), int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)), int(cap.get(cv2.C

AP\_PROP\_FRAME\_HEIGHT))

out = cv2.VideoWriter(output\_video\_path, cv2.VideoWriter\_fourcc(\*'mp4v'), fps, (width, height))

sequence = deque(maxlen=SEQUENCE\_LENGTH)

while True:

ret, frame = cap.read()

if not ret:

break

resized\_frame = cv2.resize(frame, (IMAGE\_HEIGHT, IMAGE\_WIDTH))

normalized\_frame = resized\_frame / 255.0

sequence.append(normalized\_frame)

if len(sequence) == SEQUENCE\_LENGTH:

prediction\_input = np.expand\_dims(np.array(sequence), axis=0)

predictions = model.predict(prediction\_input)

predicted\_index = np.argmax(predictions, axis=1)[0]

class\_name = CLASSES\_LIST[predicted\_index]

cv2.putText(frame, class\_name, (50, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)

out.write(frame)

cap.release()

out.release()

# Root path to return a greeting, demonstrating basic FastAPI operation.

@app.get("/")

def read\_root():

"""

Basic API endpoint for testing that the API is operational.

Returns:

dict: A simple greeting message.

"""

return {"Hello": "This is your FastAPI application"}

# Endpoint for video analysis

@app.post("/analyze-video/")

async def analyze\_video(file: UploadFile = File(...)):

"""

Receives a video file, processes it to extract frames, performs predictions using three models, and returns a zip file containing analysis results.

Args:

file (UploadFile): The video file uploaded by the client.

Returns:

FileResponse: A zip file containing the output videos from different models.

"""

# The detailed implementation would involve file handling and model prediction as previously outlined

1. **download\_youtube\_videos**:
   * **Role**: This function is responsible for downloading a video from a given YouTube URL. It downloads the highest resolution version available and ensures that the downloaded file is uniquely named if multiple downloads occur. This function is crucial for processing YouTube content within the application.
2. **frames\_extraction**:
   * **Role**: This function extracts a specific number of frames evenly spaced throughout a video file. It adjusts the frames by resizing them and normalizing their pixel values. This is a critical preprocessing step for video analysis, where these frames serve as the input data for machine learning models to make predictions.
3. **continuous\_predict\_and\_overlay**:
   * **Role**: This function continuously reads frames from a video, makes predictions on sequences of frames using a trained model, and overlays these predictions as text labels onto the video. This provides real-time video analysis and visualization of the model's predictions, creating an annotated video output that is saved to a specified path.
4. **read\_root**:
   * **Role**: Serves as a simple test endpoint for the FastAPI application. It returns a basic greeting, indicating that the API server is running and can respond to requests. This is useful for initial tests and health checks of the API.
5. **analyze\_video**:
   * **Role**: Handles the video analysis workflow for uploaded videos. This endpoint accepts a video file, performs frame extraction, uses multiple models to predict actions in the video, and then packages the videos with predictions into a zip file for the user to download. This function integrates multiple steps and models, demonstrating the application's capability to handle complex video processing tasks.

**Code Overview and Component Roles**

1. **FastAPI Initialization**:
   * **Role**: Initialize the FastAPI application which acts as the main object handling all routes and requests.
2. **Model Loading and Configuration**:
   * **Role**: Load the deep learning models and define configuration parameters like class labels and input dimensions. This setup is crucial for preparing the models for making predictions on video data.
3. **Directory Setup**:
   * **Role**: Ensures a temporary directory exists for storing videos and processed outputs. This is essential for file management within the application.
4. **CORS Middleware Configuration**:
   * **Role**: Configures Cross-Origin Resource Sharing (CORS) settings to allow interactions between the server and web clients from different domains. This is important for creating applications to function correctly when requests are made across domains.

**Function Roles**

1. **download\_youtube\_videos**:
   * **Function Role**: Downloads a YouTube video using the provided URL and saves it to a specified directory. It ensures the video filename is unique if similar files already exist. This function is key for obtaining video content for analysis.
2. **frames\_extraction**:
   * **Function Role**: Extracts a fixed number of frames from a video, evenly spaced throughout its duration. This function adjusts the frames by resizing and normalizing them, preparing them for input into the machine learning models. It's crucial for consistent model input preparation.
3. **continuous\_predict\_and\_overlay**:
   * **Function Role**: Processes a video by continuously reading its frames, making predictions with a given model, and overlaying these predictions onto the video. It outputs a new video with annotations indicating the predicted activities. This function is vital for providing real-time insights into the video content.
4. **analyze\_video**:
   * **Function Role**: Handles the video file upload, extracts frames from the video, performs predictions using the loaded models, and packages the analyzed videos into a zip file for download. This endpoint is a key feature of the application, offering end-to-end video analysis.
5. **download\_and\_analyze\_youtube**:
   * **Function Role**: Downloads a video from YouTube and analyzes it by predicting frame sequences and overlaying these predictions. It returns the processed video, allowing users to see the model's performance on YouTube content.
6. **stream\_video**:
   * **Function Role**: Streams a video file directly from the server to the client. This function supports streaming both a specific video by name and a default video, enabling flexible video access within the application.

Flutter – used in vsCode environment.

Other changes in flutter code.

In order to utilize the iphone, dependencies in /ios/Runner/info.plist has been changed.  
<key>NSAppTransportSecurity</key>

<dict>

<key>NSExceptionDomains</key>

<dict>

<key>192.168.140.135</key>

<dict>

<key>NSExceptionAllowsInsecureHTTPLoads</key>

<true/>

<key>NSIncludesSubdomains</key>

<true/>

</dict>

</dict>

</dict>

Above code has been added since it cannot access to local server without above mentioned.

In case of other dependencies, which allow to connect with localhost server with FastAPI in backend, download youtube video, and searching the path of files. Also, the version of each dependencies do not have to be specified as recent version will be downloaded if version is not mentioned. Details are mentioned below.

In pubspec.ymal,

http:

fluttertoast:

path\_provider:

video\_player:

flutter\_spinkit:

above codes have been added and their functions are described below.

1. **http**:
   * **Function**: This package provides a composable, Future-based library for making HTTP requests. It's used to interact with APIs or any network communication in your Flutter application.
2. **fluttertoast**:
   * **Function**: Fluttertoast is a plugin for Flutter that allows you to show toast messages (a short, non-invasive feedback message) in your application. It's commonly used to provide simple feedback about an operation, like a save or error message.
3. **path\_provider**:
   * **Function**: This plugin is used in Flutter to find commonly used locations on the filesystem. It supports finding directories like the temporary directory and the documents directory, which are used for storing files that are only needed temporarily or that are intended to be visible/managed by the user.
4. **video\_player**:
   * **Function**: This package provides video playback capabilities in a Flutter application. It supports both network and asset videos and offers a range of controls over the video playback, such as pausing, resuming, and seeking to specific points.
5. **flutter\_spinkit**:
   * **Function**: Flutter Spinkit is a collection of animated loading indicators for Flutter. It's useful for showing a visual indicator when your app is performing a task that requires the user to wait, such as loading data from a network.

Also, in order to utilize iphone, register to Apple developer with about 700HKD is necessary otherwise laptop would abandon the connection as Xcode will abandon the connection. This is because it is not available to specify the bundle identifier in Xcode/Runner/General part as team name, which should be registered in Apple developer, may not proceed the random user if not registered. Furthermore, bundle identifier should be same as the name of project otherwise error occurs.

Reference

Simplilearn. (2022). *Deep Learning (Full Course) | Deep Learning Tutorial for Beginners | Python Deep Learning Course* [Video]. YouTube. <https://www.youtube.com/watch?v=QmtSkq3DYko&t=2183s>