```
!pip install pafy youtube-dl moviepy


!pip install --upgrade pip


import h5py
import os
import cv2
import pafy
import numpy as np
from moviepy.editor import *
from collections import deque
from keras.models import load_model


# returns a compiled model
# identical to the previous one
model = load_model('models_output\LRCN_model___Date_Time_2022_08_09__20_26_39___Loss_0.491615

IMAGE_HEIGHT, IMAGE_WIDTH = 64, 64
CLASSES_LIST = ["bridge_connecting", "baggage_handling", "misclaneous"]
model_output_size = len(CLASSES_LIST)
SEQUENCE_LENGTH = 6


def predict_on_video(video_file_path, output_file_path, SEQUENCE_LENGTH):
    '''
    This function will perform action recognition on a video using the LRCN model.
    Args:
    video_file_path:  The path of the video stored in the disk on which the action recognitio
    output_file_path: The path where the ouput video with the predicted action being performe
    SEQUENCE_LENGTH:  The fixed number of frames of a video that can be passed to the model a
    '''

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture(video_file_path)

    # Get the width and height of the video.
    original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Initialize the VideoWriter Object to store the output video in the disk.
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P', '4', 'V
                                   video_reader.get(cv2.CAP_PROP_FPS), (original_video_width,

    # Declare a queue to store video frames.
    frames_queue = deque(maxlen=SEQUENCE_LENGTH)

    # Initialize a variable to store the predicted action being performed in the video.
    predicted_class_name = ''
```

```
    # Iterate until the video is accessed successfully.
    while video_reader.isOpened():

        # Read the frame.
        ok, frame = video_reader.read()

        # Check if frame is not read properly then break the loop.
        if not ok:
            break

        # Resize the Frame to fixed Dimensions.
        resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))

        # Normalize the resized frame by dividing it with 255 so that each pixel value then l
        normalized_frame = resized_frame / 255

        # Appending the pre-processed frame into the frames list.
        frames_queue.append(normalized_frame)

        # Check if the number of frames in the queue are equal to the fixed sequence length.
        if len(frames_queue) == SEQUENCE_LENGTH:
            # Pass the normalized frames to the model and get the predicted probabilities.
            predicted_labels_probabilities = model.predict(np.expand_dims(frames_queue, axis=

            # Get the index of class with highest probability.
            predicted_label = np.argmax(predicted_labels_probabilities)

            # Get the class name using the retrieved index.
            predicted_class_name = CLASSES_LIST[predicted_label]

        # Write predicted class name on top of the frame.
        cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 2

        # Write The frame into the disk using the VideoWriter Object.
        video_writer.write(frame)

    # Release the VideoCapture and VideoWriter objects.
    video_reader.release()
    video_writer.release()


def predict_single_action(video_file_path, SEQUENCE_LENGTH):
    '''
    This function will perform single action recognition prediction on a video using the LRCN
    Args:
    video_file_path:  The path of the video stored in the disk on which the action recognitio
    SEQUENCE_LENGTH:  The fixed number of frames of a video that can be passed to the model a
    '''

    # Initialize the VideoCapture object to read from the video file.
    video_reader = cv2.VideoCapture(video_file_path)
```

```
# Get the width and height of the video.
original_video_width = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
original_video_height = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Declare a list to store video frames we will extract.
frames_list = []

# Initialize a variable to store the predicted action being performed in the video.
predicted_class_name = ''

# Get the number of frames in the video.
video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))

# Calculate the interval after which frames will be added to the list.
skip_frames_window = max(int(video_frames_count / SEQUENCE_LENGTH), 1)

# Iterating the number of times equal to the fixed length of sequence.
for frame_counter in range(SEQUENCE_LENGTH):

    # Set the current frame position of the video.
    video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)

    # Read a frame.
    success, frame = video_reader.read()

    # Check if frame is not read properly then break the loop.
    if not success:
        break

    # Resize the Frame to fixed Dimensions.
    resized_frame = cv2.resize(frame, (IMAGE_HEIGHT, IMAGE_WIDTH))

    # Normalize the resized frame by dividing it with 255 so that each pixel value then l
    normalized_frame = resized_frame / 255

    # Appending the pre-processed frame into the frames list
    frames_list.append(normalized_frame)

# Passing the  pre-processed frames to the model and get the predicted probabilities.
predicted_labels_probabilities = model.predict(np.expand_dims(frames_list, axis=0))[0]

# Get the index of class with highest probability.
predicted_label = np.argmax(predicted_labels_probabilities)

# Get the class name using the retrieved index.
predicted_class_name = CLASSES_LIST[predicted_label]

# Display the predicted action along with the prediction confidence.
print(f'Action Predicted: {predicted_class_name}\nConfidence: {predicted_labels_probabili
```

```
    # Release the VideoCapture object.
    video_reader.release()
```

```
# Make the Output directory if it does not exist
test_videos_directory = 'test_videos'
os.makedirs(test_videos_directory, exist_ok = True)
```

Input: Bridge connecting video

```
input_video_file_path = r"C:\Users\vsriniva\Desktop\Action_recognition\Identify_Gate_Operatio
```

```
output_video_file_path = "result_LRCN_bridge_connecting.mp4"

# Perform Action Recognition on the Test Video.
predict_on_video(input_video_file_path, output_video_file_path, SEQUENCE_LENGTH)

# Display the output video.
VideoFileClip(output_video_file_path, audio=False, target_resolution=(300,None)).ipython_disp
```

```
    t:   0%||                                                                | 3/1561 [0
    Moviepy - Writing video __temp__.mp4

    Moviepy - Done !
    Moviepy - video ready __temp__.mp4
```

```
# Perform Single Prediction on the Test Video.
predict_single_action(input_video_file_path, SEQUENCE_LENGTH)

# Display the input video.
# VideoFileClip(input_video_file_path, audio=False, target_resolution=(300,None)).ipython_dis
```

```
    Action Predicted: bridge_connecting
    Confidence: 0.8911964893341064
```

Input: Baggage handling Video

```
input_video_file_path = r"C:\Users\vsriniva\Desktop\Action_recognition\Identify_Gate_Operatio


output_video_file_path = "result_LRCN_baggage_handling.mp4"

# Perform Action Recognition on the Test Video.
predict_on_video(input_video_file_path, output_video_file_path, SEQUENCE_LENGTH)

# Display the output video.
VideoFileClip(output_video_file_path, audio=False, target_resolution=(300,None)).ipython_disp
```

```
t:   3%|▉                                                                  | 3/91 [0
Moviepy - Writing video __temp__.mp4

Moviepy - Done !
Moviepy - video ready __temp__.mp4
```

```
# Perform Single Prediction on the Test Video.
predict_single_action(input_video_file_path, SEQUENCE_LENGTH)

# Display the input video.
# VideoFileClip(input_video_file_path, audio=False, target_resolution=(300,None)).ipython_dis
```

```
Action Predicted: baggage_handling
Confidence: 0.8003833293914795
```

Input: Misclaneous Video

```
input_video_file_path = r"C:\Users\vsriniva\Desktop\Action_recognition\Identify_Gate_Operatio


output_video_file_path = "result_LRCN_misclaneous.mp4"

# Perform Action Recognition on the Test Video.
predict_on_video(input_video_file_path, output_video_file_path, SEQUENCE_LENGTH)

# Display the output video.
VideoFileClip(output_video_file_path, audio=False, target_resolution=(300,None)).ipython_disp
```

```
t:   0%|                                                                    | 3/925 [0
Moviepy - Writing video __temp__.mp4

Moviepy - Done !
Moviepy - video ready __temp__.mp4
```

```
# Perform Single Prediction on the Test Video.
predict_single_action(input_video_file_path, SEQUENCE_LENGTH)

# Display the input video.
# VideoFileClip(input_video_file_path, audio=False, target_resolution=(300,None)).ipython_dis
```

```
Action Predicted: misclaneous
Confidence: 0.9992884397506714
```