```
!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
```

```
model = load_model('models_output\Model___Date_Time_2022_08_09__12_13_56___Loss_0.09544385224

image_height, image_width = 64, 64
classes_list = ["bridge_connecting", "baggage_handling", "misclaneous"]
model_output_size = len(classes_list)
```

```
def predict_on_live_video(video_file_path, output_file_path, window_size):
    # Initialize a Deque Object with a fixed size which will be used to implement moving/roll
    predicted_labels_probabilities_deque = deque(maxlen=window_size)

    # Reading the Video File using the VideoCapture Object
    video_reader = cv2.VideoCapture(video_file_path)

    # Getting 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))

    # Writing the Overlayed Video Files Using the VideoWriter Object
    video_writer = cv2.VideoWriter(output_file_path, cv2.VideoWriter_fourcc('M', 'P', '4', 'V
                               (original_video_width, original_video_height))

    while True:

        # Reading The Frame
        status, frame = video_reader.read()

        if not status:
            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
        # Passing the Image Normalized Frame to the model and receiving Predicted Probabiliti
        predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis=

        # Appending predicted label probabilities to the deque object
        predicted_labels_probabilities_deque.append(predicted_labels_probabilities)

        # Assuring that the Deque is completely filled before starting the averaging process
        if len(predicted_labels_probabilities_deque) == window_size:
            # Converting Predicted Labels Probabilities Deque into Numpy array
            predicted_labels_probabilities_np = np.array(predicted_labels_probabilities_deque

            # Calculating Average of Predicted Labels Probabilities Column Wise
            predicted_labels_probabilities_averaged = predicted_labels_probabilities_np.mean(

            # Converting the predicted probabilities into labels by returning the index of th
            predicted_label = np.argmax(predicted_labels_probabilities_averaged)

            # Accessing The Class Name using predicted label.
            predicted_class_name = classes_list[predicted_label]

            # Overlaying Class Name Text Ontop of the Frame
            cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (

        # Writing The Frame
        video_writer.write(frame)

        cv2.imshow('Predicted Frames', frame)

        key_pressed = cv2.waitKey(10)

        if key_pressed == ord('q'):
             break

    cv2.destroyAllWindows()

    # Closing the VideoCapture and VideoWriter objects and releasing all resources held by th
    video_reader.release()
    video_writer.release()


def make_average_predictions(video_file_path, predictions_frames_count):
    # Initializing the Numpy array which will store Prediction Probabilities
    predicted_labels_probabilities_np = np.zeros((predictions_frames_count, model_output_size

    # Reading the Video File using the VideoCapture Object
    video_reader = cv2.VideoCapture(video_file_path)

    # Getting The Total Frames present in the video
    video_frames_count = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
```

```
# Calculating The Number of Frames to skip Before reading a frame
skip_frames_window = video_frames_count // predictions_frames_count

for frame_counter in range(predictions_frames_count):
    # Setting Frame Position
    video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_counter * skip_frames_window)

    # Reading The Frame
    _, frame = video_reader.read()

    try:
       resized_frame = cv2.resize(frame, (image_height, image_width), interpolation=cv2.I
       # print(resized_frame.shape)
    except:
        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


    # Passing the Image Normalized Frame to the model and receiving Predicted Probabiliti
    predicted_labels_probabilities = model.predict(np.expand_dims(normalized_frame, axis=

    # Overlaying Class Name Text Ontop of the Frame
    # cv2.putText(frame, predicted_class_name, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,

    # Writing The Frame
    # video_writer.write(frame)

    # cv2.imshow('Predicted Frames', frame)

    # key_pressed = cv2.waitKey(10)

    # if key_pressed == ord('q'):
        #  break

    # Appending predicted label probabilities to the deque object
    predicted_labels_probabilities_np[frame_counter] = predicted_labels_probabilities

#  cv2.destroyAllWindows()

# Calculating Average of Predicted Labels Probabilities Column Wise
predicted_labels_probabilities_averaged = predicted_labels_probabilities_np.mean(axis=0)

# Sorting the Averaged Predicted Labels Probabilities
predicted_labels_probabilities_averaged_sorted_indexes = np.argsort(predicted_labels_prob

# Iterating Over All Averaged Predicted Label Probabilities
```

```
    for predicted_label in predicted_labels_probabilities_averaged_sorted_indexes:
        # Accessing The Class Name using predicted label.
        predicted_class_name = classes_list[predicted_label]

        # Accessing The Averaged Probability using predicted label.
        predicted_probability = predicted_labels_probabilities_averaged[predicted_label]

        print(f"CLASS NAME: {predicted_class_name}   AVERAGED PROBABILITY: {(predicted_probab

    # Closing the VideoCapture Object and releasing all resources held by it.
    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
```

```
print("\n CASE 1: Results Without Using Moving Average ")
# First let us see the results when we are not using moving
# average, we can do this by setting the window_size to 1.
# Settings the Window Size which will be used by the Rolling Average Process
window_size = 1

# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_without_moving_avg_bridge_connecting.mp4"

# Calling the predict_on_live_video method to start the Prediction.
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

```
 CASE 1: Results Without Using Moving Average
t:   1%|█                                                                     | 3/526 [0
Moviepy - Writing video __temp__.mp4

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

```
print("\n CASE 2: Results When Using Moving Average ")

# Now let us use moving average with a window size of 25

# Setting the Window Size which will be used by the Rolling Average Process
window_size = 25

# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_moving_avg_bridge_connecting.mp4"

# Calling the predict_on_live_video method to start the Prediction and Rolling Average Proces
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

```
 CASE 2: Results When Using Moving Average
t:   0%|▌                                                                          | 2/526 [0
Moviepy - Writing video __temp__.mp4

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

```
print("\n CASE 3: Using Single-Frame CNN Method")

# Calling The Make Average Method To Start The Process
make_average_predictions(input_video_file_path, 50)
```

```
 CASE 3: Using Single-Frame CNN Method
CLASS NAME: bridge_connecting   AVERAGED PROBABILITY: 9.6e+01
CLASS NAME: misclaneous   AVERAGED PROBABILITY: 0.13
CLASS NAME: baggage_handling   AVERAGED PROBABILITY: 0.003
```

Input: Baggage handling Video

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

```
print("\n CASE 1: Results Without Using Moving Average ")
# First let us see the results when we are not using moving
# average, we can do this by setting the window_size to 1.
# Settings the Window Size which will be used by the Rolling Average Process
window_size = 1
```

```
# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_without_moving_avg_baggage_handling.mp4"

# Calling the predict_on_live_video method to start the Prediction.
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

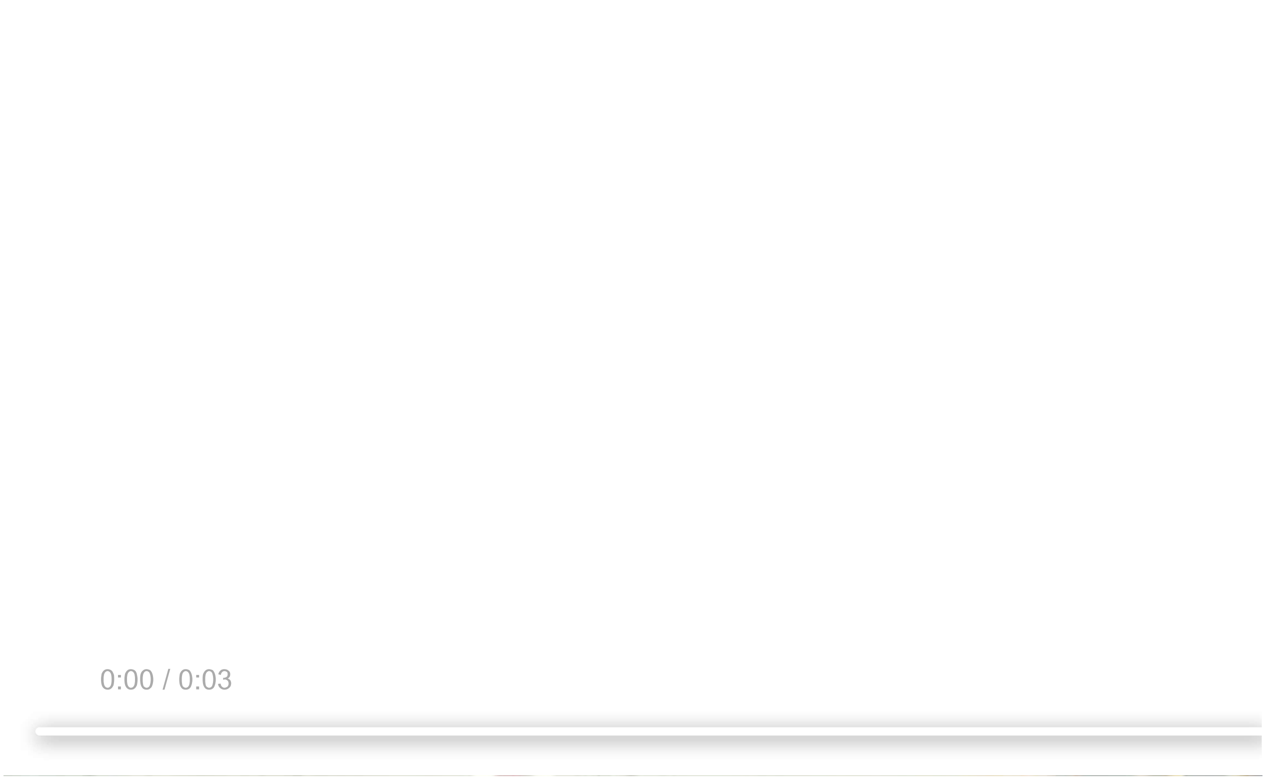
```
 CASE 1: Results Without Using Moving Average
t:   2%|█                                                                         | 2/91 [0
Moviepy - Writing video __temp__.mp4

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

0:00 / 0:03

```
print("\n CASE 2: Results When Using Moving Average ")

# Now let us use moving average with a window size of 25

# Setting the Window Size which will be used by the Rolling Average Process
window_size = 25

# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_moving_avg_baggage_handling.mp4"
```

```
# Calling the predict_on_live_video method to start the Prediction and Rolling Average Proces
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

```
 CASE 2: Results When Using Moving Average
t:   2%|█                                                                         | 2/91 [0
Moviepy - Writing video __temp__.mp4

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



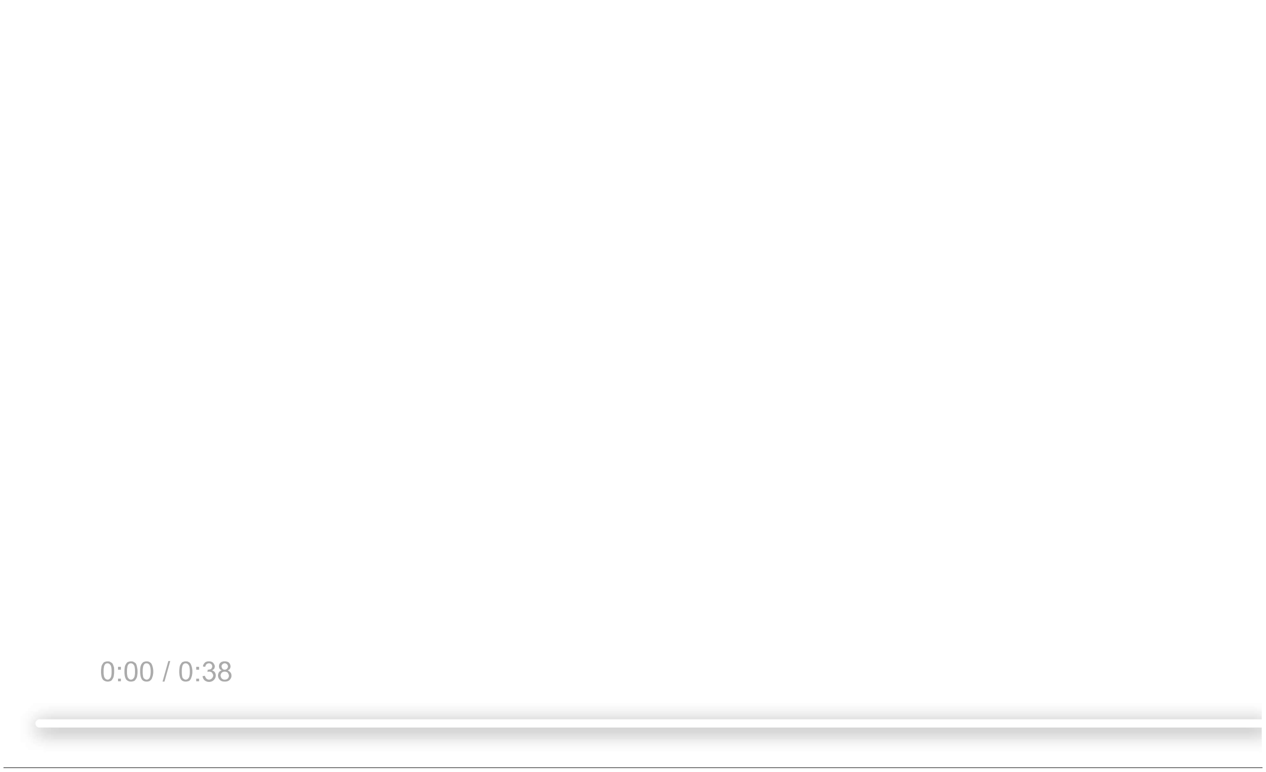
```
print("\n CASE 3: Using Single-Frame CNN Method")

# Calling The Make Average Method To Start The Process
make_average_predictions(input_video_file_path, 50)
```

```
 CASE 3: Using Single-Frame CNN Method
CLASS NAME: baggage_handling   AVERAGED PROBABILITY: 9.2e+01
CLASS NAME: bridge_connecting   AVERAGED PROBABILITY: 6.2
CLASS NAME: misclaneous   AVERAGED PROBABILITY: 1.5
```

Input: Misclaneous Video

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


print("\n CASE 1: Results Without Using Moving Average ")
# First let us see the results when we are not using moving
# average, we can do this by setting the window_size to 1.
# Settings the Window Size which will be used by the Rolling Average Process
window_size = 1

# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_without_moving_avg_misclaneous.mp4"

# Calling the predict_on_live_video method to start the Prediction.
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

```
 CASE 1: Results Without Using Moving Average
t:   0%|▌                                                                   | 2/925 [0
Moviepy - Writing video __temp__.mp4

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



```
print("\n CASE 2: Results When Using Moving Average ")
```

```
# Now let us use moving average with a window size of 25

# Setting the Window Size which will be used by the Rolling Average Process
window_size = 25

# Constructing The Output YouTube Video Path
# output_video_file_path = f'{output_directory}/{video_title} -Output-WSize {window_size}.mp4

output_video_file_path = "result_moving_avg_misclaneous.mp4"

# Calling the predict_on_live_video method to start the Prediction and Rolling Average Proces
predict_on_live_video(input_video_file_path, output_video_file_path, window_size)

# Play Video File in the Notebook
VideoFileClip(output_video_file_path).ipython_display(width = 700)
```

```
 CASE 2: Results When Using Moving Average
t:   0%||                                                                    | 2/925 [6
Moviepy - Writing video __temp__.mp4

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

```
print("\n CASE 3: Using Single-Frame CNN Method")

# Calling The Make Average Method To Start The Process
make_average_predictions(input_video_file_path, 50)
```

```
 CASE 3: Using Single-Frame CNN Method
```

```
CLASS NAME: misclaneous   AVERAGED PROBABILITY: 1e+02
CLASS NAME: bridge_connecting   AVERAGED PROBABILITY: 0.13
CLASS NAME: baggage_handling   AVERAGED PROBABILITY: 0.00011
```

+ Code   + Text