INFOSYS PROJECT - OBJECT TRACKING IN SURVEILLANCE VIDEOS

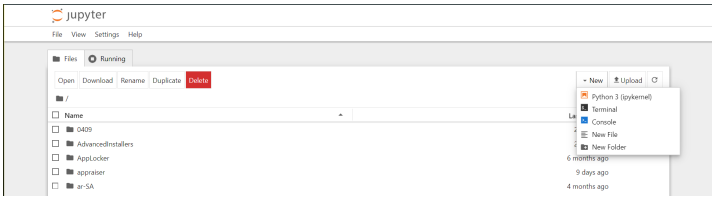
1. IMAGE PROCESSING BY USING OPENCV

INSTALL OPENCV IN JUPYTER NOTEBOOK BY USING THE COMMAND

COMMAND -

```
pip -q install opencv-python
```

YOU CAN USE THE TERMINAL OPTION ON JUPYTER NOTEBOOK.

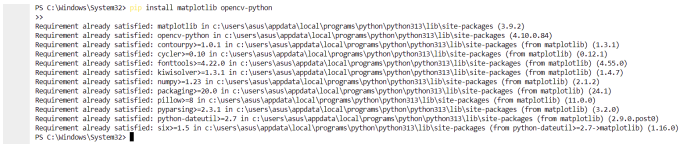


THEN INSTALL OPENCV IN THAT TERMINAL USING THE COMMAND

THEN BEFORE PROCEEDING TOWARDS NEXT STEP INSTALL MATPLOTLIB USING -

pip install matplotlib opencv-python

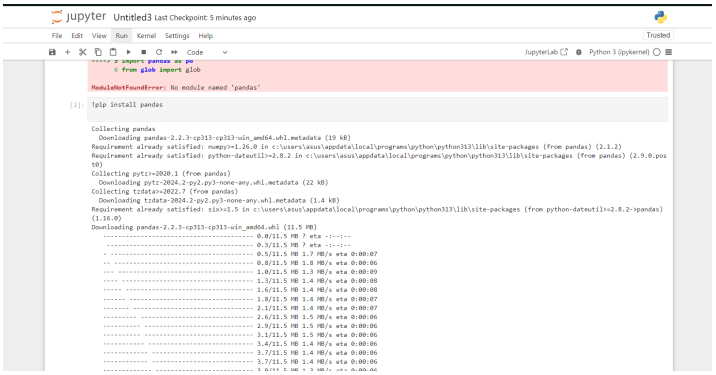
IN MY NOTEBOOK IT IS ALREADY INSTALLED .



NOW INSTALL PANDA ON THE NOTEBOOK -

!pip install pandas

HERE IS A CHANGE YOU DO NOT REQUIRE TO USE TERMINAL OF THE NOTEBOOK YOU CAN SIMPLY USE THE GIVEN COMMAND ON THE NOTEBOOK ITSELF



AS YOU MAY HAVE NOTICE THE RED BOC ABOVE , THAT IS BECAUSE THE PANDA MODULE WAS NOT INSTALLED ON MY NOTEBOOK BUT AFTER INSTALLANTION YOU HAVE TO RUN THE SAME CODE AGAIN AND HERE IS THE NEW RESULT -

THEN IN THE MAIN PROCESS USE THE GIVEN CODE BELOW -

```
import cv2
import numpy as np
# from google.colab.patches import cv2_imshow
import matplotlib.pylab as plt
import pandas as pd
from glob import glob
```

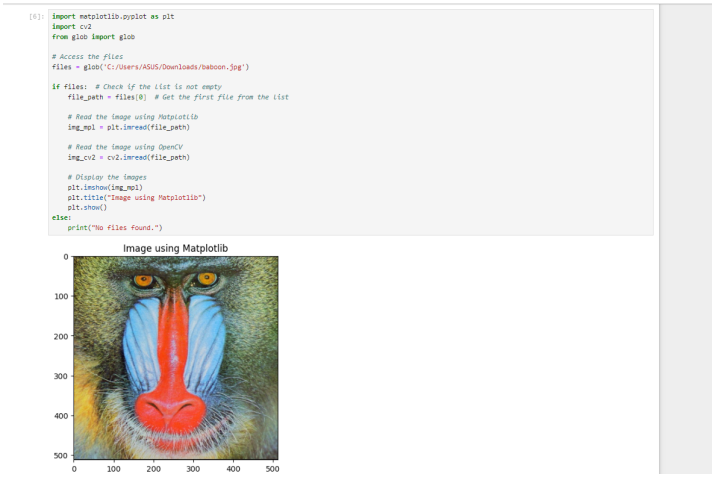
THEN IN THE NEXT STEP YOU WILL USE THE GIVEN CODE -

```
import matplotlib.pyplot as plt
import cv2
from glob import glob

# Access the files
files = glob('C:\Users\ASUS\Downloads\baboon.jpg')

if files:  # Check if the list is not empty
    file_path = files[0]  # Get the first file from the list

    # Read the image using Matplotlib
    img_mpl = plt.imread(file_path)

    # Read the image using OpenCV
    img_cv2 = cv2.imread(file_path)

    # Display the images
    plt.imshow(img_mpl)
    plt.title("Image using Matplotlib")
    plt.show()
else:
    print("No files found.")BUT IN THIS GIVEN CODE ON THE LINE - files = glob ( "C:\Users\ASUS\Downloads\baboon.jpg
```

BUT BEFORE USING THE COMMAND YOU CAN SEE THE LINE - files = glob ('C:\Users\ASUS\Downloads\baboon.jpg')
HERE YOU HAVE TO CHANGE THE LOCATION OF THE PATH WHERE YOU HAVE SAVED THE PHOTO WHICH YOU WANTED TO USE IN THE PROJECT

FOR EXAMPLE I HAVE USED THIS PICTURE -

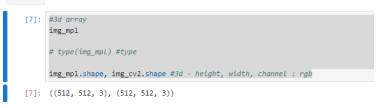
AFTER USING THE GIVEN CODE THE PHOTO WHILE CHANGE INTO THE GIVEN PHOTO -



AFTER GETTING THIS OUTPUT RUN ANOTHER SHELL AND USE THIS CODE -

```
#3d array
img_mpl

# type(img_mpl) #type

img_mpl.shape, img_cv2.shape #3d - height, width, channel : rgb
```

AND THIS IS THE OUTPUT -

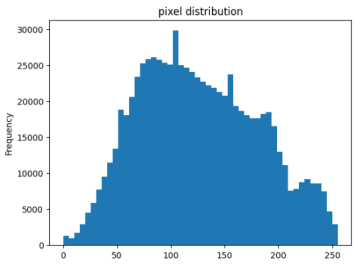


NOW AFTER THAT RUN ANOTHER CODE -

```
pd.Series(img_mpl.flatten()).plot(kind='hist', bins = 50, title='pixel distribution') #bins: specifies the number of bins (g
plt.show()
```

AND HERE IS THE PIXEL DISTRIBUTION OUTPUT -

```
[8]: pd.Series(img_mpl.flatten()).plot(kind='hist', bins = 50, title='pixel distribution') #bins: specifies the number of bins (groups) in the histogram, g(v
plt.show()
```

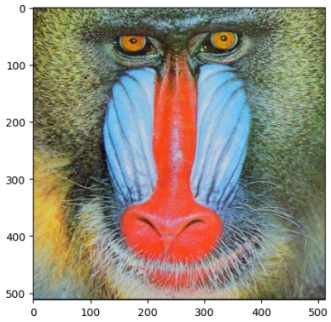


AFTER THAT THE NEXT STEP IS TO CREATE FIGURE SIZE OF THE PICTURE IN AXIS TABLE -

```
fig, ax = plt.subplots(figsize= (5,5))
ax.imshow(img_mpl)
# ax.axis('off')
plt.show()
```

AND HERE IS THE OUTPUT -

```
[9]: fig, ax = plt.subplots(figsize= (5,5))
ax.imshow(img_mpl)
# ax.axis('off')
plt.show()
```



AFTER THAT , THE NEXT STEP IS TO PUT RGB COLOURS IN THE PHOTO -

```
#Display RGB channels

fig, axs = plt.subplots(1,3,figsize = (10,5))
axs[0].imshow(img_mpl[:,:,0], cmap='Reds')
axs[1].imshow(img_mpl[:,:,1], cmap='Greens')
axs[2].imshow(img_mpl[:,:,2], cmap='Blues')
axs[0].axis('off')
axs[1].axis('off')
axs[2].axis('off')
plt.show()
```

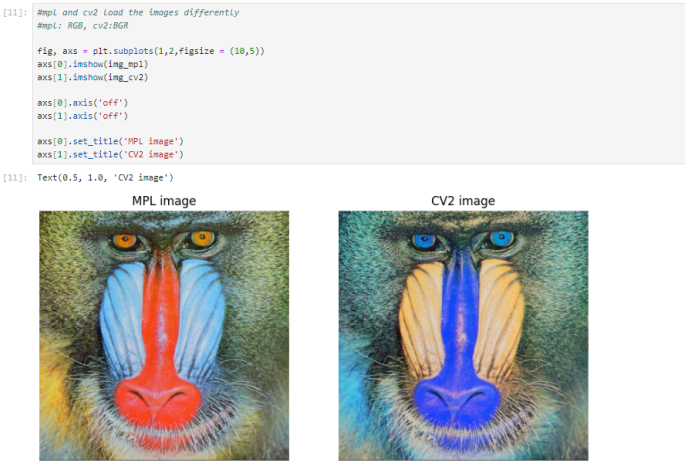
OUTPUT -

```
[10]: #Display RGB channels

fig, axs = plt.subplots(1,3,figsize = (10,5))
axs[0].imshow(img_mpl[:,:,0], cmap='Reds')
axs[1].imshow(img_mpl[:,:,1], cmap='Greens')
axs[2].imshow(img_mpl[:,:,2], cmap='Blues')
axs[0].axis('off')
axs[1].axis('off')
axs[2].axis('off')
plt.show()
```

IN THE NEXT PROCESS WE USE THIS CODE -

```
#mpl and cv2 load the images differently
#mpl: RGB, cv2:BGR

fig, axs = plt.subplots(1,2,figsize = (10,5))
axs[0].imshow(img_mpl)
axs[1].imshow(img_cv2)

axs[0].axis('off')
axs[1].axis('off')

axs[0].set_title('MPL image')
axs[1].set_title('CV2 image')
```

THE OUTPUT -




```
#Converting from BGR to RGB

img_cv2_rgb = cv2.cvtColor(img_cv2, cv2.COLOR_BGR2RGB)
fig, ax = plt.subplots()
ax.imshow(img_cv2_rgb)
ax.axis('off')
plt.show()
```

OUTPUT -

AFTER THIS , IN THE NEXT STEP WE WILL CHECK IF THE IMAGE IS BLUR OR NOT , SO USE THIS CODE - cv2.imwrite('cv2_monkey.jpg', blur)

BUT IF AFTER USING THIS CODE , IF YOU GOT A PROBLEM LIKE THIS -

```
[13]: cv2.imwrite('cv2_monkey.jpg', blur)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 cv2.imwrite('cv2_monkey.jpg', blur)

NameError: name 'blur' is not defined
```

THEN USE THIS CODE TO RECTIFY THIS PROCESS -

```
import cv2

# Read the image
img = cv2.imread('C:/Users/ASUS/Downloads/baboon.jpg')

if img is not None:  # Ensure the image was successfully loaded
    # Apply Gaussian Blur
    blur = cv2.GaussianBlur(img, (15, 15), 0)

    # Save the blurred image
    save_status = cv2.imwrite('C:/Users/ASUS/Downloads/cv2_monkey.jpg', blur)

    if save_status:
        print("Image successfully saved as 'cv2_monkey.jpg'")
    else:
        print("Error: Could not save the image.")
else:
    print("Error: Could not load the image. Check the file path.")
```

THIS CODE WILL FIX THE ERROR AND ALSO DO NOT FORGET TO CHANGE THE IMAGE PATH WHICH I HAVE MENTIONED ON THE CODE .

BUT THERE WAS NO ERROR THEN THEIR IS NO NEED TO USE THIS CODE .

THE OUTPUT OF THE CODE - cv2.imwrite('cv2_monkey.jpg', blur) IS -

```
[16]: cv2.imwrite('cv2_monkey.jpg', blur)

[16]: True
```

IN THIS STEP WE WILL UPGRADE THE PICTURE USING -

!pip install imageai --upgrade

```
[17]: !pip install imageai --upgrade

Collecting imageai
  Downloading imageai-3.0.3-py3-none-any.whl.metadata (340 bytes)
Downloading imageai-3.0.3-py3-none-any.whl (69 kB)
Installing collected packages: imageai
Successfully installed imageai-3.0.3
```

2. OBJECT DETECTION USING TINY YOLO

HERE YOU HAVE TO DOWNLOAD THE YOLO FILE - tiny-yolov3.pt

download

I HAVE EMBEDED A LINK WHICH WILL REDIRECT YOU TO THE DOWNLOADING PAGE OF THE YOLOV3 FILE.

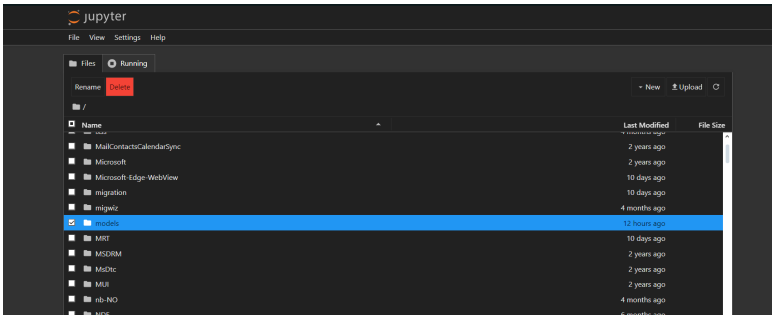
AFTER DOWNLOADING THE FILE , IN THE NEXT STEP YOU HAVE TO CREATE THE MODELS FOLDER ON JUPYTER.

```
import os

# Create the 'models' directory if it doesn't exist
models_dir = "models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

print("Models directory is ready!")
```

```
[32]: import os

      # Create the 'models' directory if it doesn't exist
      models_dir = "models"
      if not os.path.exists(models_dir):
          os.makedirs(models_dir)

      print("Models directory is ready!")

      Models directory is ready!
```

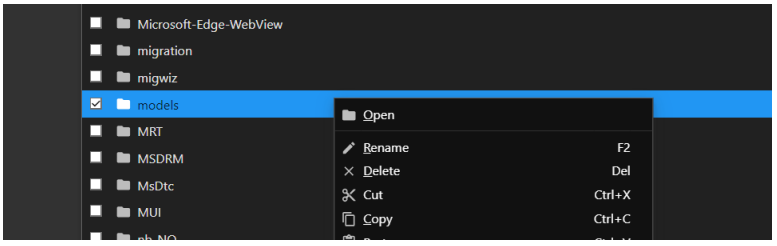
USING THE GIIVEN CODE YOUR MODELS FOLDER WILL BE READY .

NOW YOU CAN ACCESS THE MODEL FOLDER ON THE HOME OF JUPYTER NOTEBOOK.

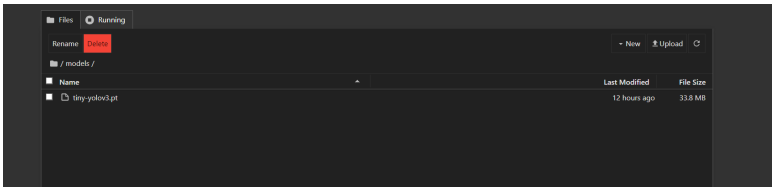


YOU CAN SEE A ✓ MARK ON THE SIDE OF THE FOLDER

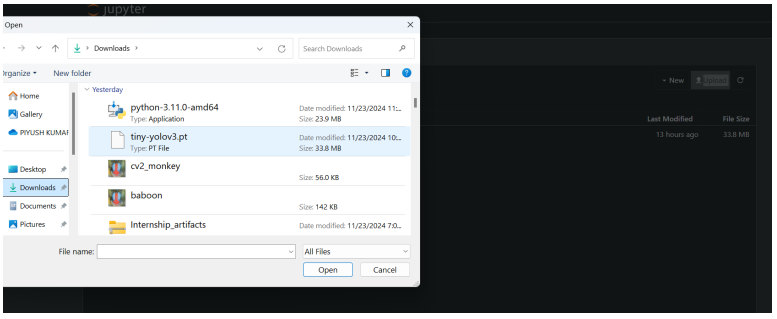
THEN RIGHT-CLICK ON THE FOLDER THEN OPEN THE FOLDER



AFTER THIS YOU CAN SEE I HAVE ALREADY UPLOADED THE DOWNLOADED YOLO FILE IN THE MODEL FOLDER



AFTER OPENING THE MODEL FOLDER THEN CLICK UPLOAD AND IT WILL NAVIGATE YOU TO THE FILE EXPLORER THEN CHOSE THE DOWNLOADED FILE AND IT WILL BE UPLOADED TO THE FOLDER.



THEN YOU ARE READY TO GO !

IN THIS VERY NEXT STEP YOU NEED TO DO SOME MORE INSTALLATION OF MODULES LIKE - torch torchvision torchaudio

```
!pip install torch torchvision torchaudio
```

AFTER INSTALLATION USE THE GIVEN CODE FOR CONFIRMATION -

```
import torch
print(torch.__version__)
print("CUDA available:", torch.cuda.is_available())
```

IN THIS PROCESS WE WILL INSTALL OPEN-CV - PYTHON - HEADLESS

```
!pip install opencv-python-headless
```

WHY IT IS NEED BECAUSE IT IS THE FULL VERSION OF OPEN-CV WITH ADDITIONAL FEATURES

LAST AND FINAL STEP -

CODE -

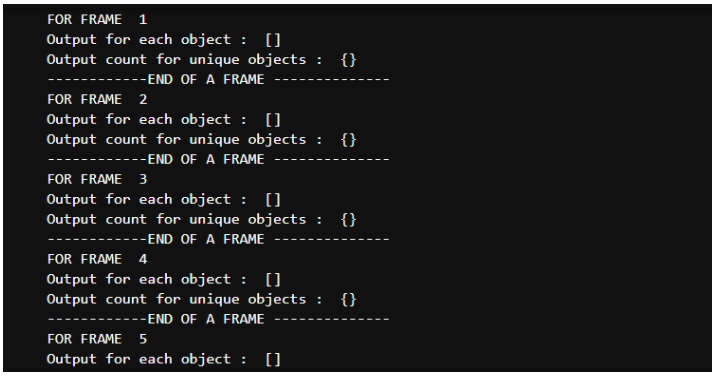
```
!pip install imageai --upgrade

from imageai.Detection import VideoObjectDetection
import os
import time

# Start timing
start_time = time.time()

# Set the execution path
execution_path = os.getcwd()

# Create the 'models' directory if it doesn't exist
os.makedirs("models", exist_ok=True)

def forFrame(frame_number, output_array, output_count):
    print("FOR FRAME ", frame_number)
    print("Output for each object : ", output_array)
    print("Output count for unique objects : ", output_count)
    print("------------END OF A FRAME --------------")

def forSeconds(second_number, output_arrays, count_arrays, average_output_count):
    print("SECOND : ", second_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last second: ", average_output_count)
    print("------------END OF A SECOND --------------")

def forMinute(minute_number, output_arrays, count_arrays, average_output_count):
    print("MINUTE : ", minute_number)
    print("Array for the outputs of each frame ", output_arrays)
    print("Array for output count for unique objects in each frame : ", count_arrays)
    print("Output average count for unique objects in the last minute: ", average_output_count)
    print("------------END OF A MINUTE --------------")

# Initialize the VideoObjectDetection
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsTinyYOLOv3()

# Path to the model file
model_path = os.path.join(execution_path, "models/tiny-yolov3.pt")
if not os.path.exists(model_path):
    print("Model file not found. Please download tiny-yolov3.pt and place it in the 'models' directory.")
else:
    video_detector.setModelPath(model_path)
    video_detector.loadModel()

    # Path to input video
    input_video_path = os.path.join(execution_path, "C:/Users/ASUS/Downloads/videoplayback.mp4")
    if not os.path.exists(input_video_path):
        print(f"Input video file not found at {input_video_path}. Please provide a valid video file.")
    else:
        # Detect objects in the video
        output_video_path = os.path.join(execution_path, "output_video.mp4")
```

```
video_detector.detectObjectsFromVideo(
    input_file_path=input_video_path,
    output_file_path=output_video_path,
    frames_per_second=10,
    per_second_function=forSeconds,
    per_frame_function=forFrame,
    per_minute_function=forMinute,
    minimum_percentage_probability=30
)

# End timing and calculate the duration
end_time = time.time()
execution_duration = end_time - start_time

print("Video saved at:", output_video_path)
print("Time taken to run the code:", execution_duration, "seconds")
```

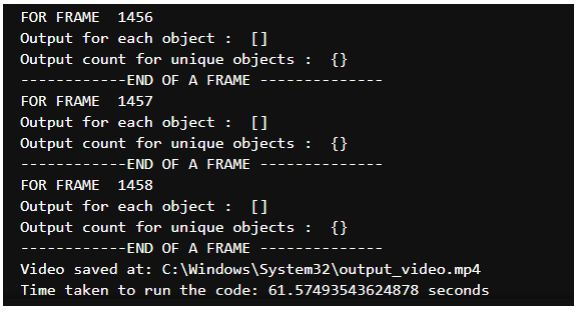
BEFORE EXECUTING THE ENTIRE CODE YOU NEED TO DOWNLOAD ANY SAMPLE VIDEO FROM ONLINE AND THEN YOU HAVE TO COPY THE PATH OF THE VIDEO FROM THE FILE EXPLORER ON YOUR SYSTEM AND PASTE IT ONE THE CODE WHERE I HAVE MENTIONED THE CODE -

"C:\Users\ASUS\Downloads\videoplayback.mp4" - THIS IS THE PATH OF MY VIDEO WHICH I HAVE INSERTED ON THE CODE MENTIONED ABOVE .

NOW AFTER RUNNING THE WHOLE PROGRAM IF YOU GOT THE OUTPUT LIKE THIS -



THEN YOU ARE GOOD TO GO BUT THIS IS THE BEGINNING OUTPUT , AFTER SCROLLING DOWN TO THE END YOU WILL SEE -

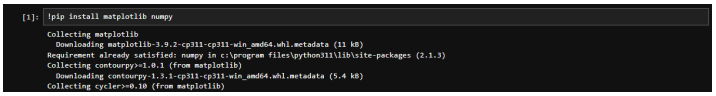


THIS SHOWS THE CODE EXECUTION IS COMPLETED.

3. OBJECT TRACKING (DNN MODULE) PROJECT

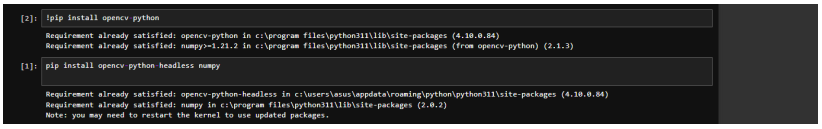
IN ORDER TO DO THIS MODEL PROJECT YOU NEED TO HAVE SOME MODULES INSTALLED ON YOUR JUPYTER NOTEBOOK.

FIRST ONE IS - !pip install matplotlib numpy



SECOND IS - !pip install opencv-python

pip install opencv-python-headless numpy



AFTER THAT THERE ARE SOME CHANGES YOU NEED TO DO IN YOUR CODE , BUT FIRST YOU NEED TO DOWNLOAD 3 FILES

coco.names.txt

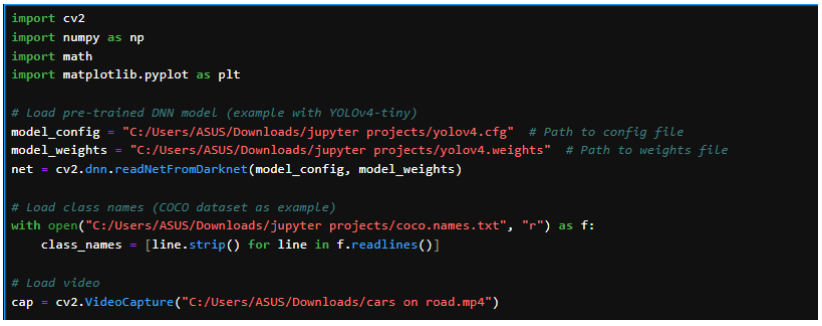
yolov4.cfg

AND THE NEXT IS OF YOLO WEIGHTS FILES - https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights

USE THIS GIVEN LINK AS THE FILE SIZE IS 245 MB

NOW CHANGES NEED TO BE DONE ON THE CODE -

AS YOU CAN SEE ON THE GIVEN PICTURE IN —



```python
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

# Load pre-trained DNN model (example with YOLOv4-tiny)
model_config = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.cfg"  # Path to config file
model_weights = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.weights"  # Path to weights file
net = cv2.dnn.readNetFromDarknet(model_config, model_weights)

# Load class names (COCO dataset as example)
with open("C:/Users/ASUS/Downloads/jupyter projects/coco.names.txt", "r") as f:
    class_names = [line.strip() for line in f.readlines()]

# Load video
cap = cv2.VideoCapture("C:/Users/ASUS/Downloads/cars on road.mp4")
```

model_config = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.cfg" # Path to config file
model_weights = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.weights" # Path to weights file

with open("C:/Users/ASUS/Downloads/jupyter projects/coco.names.txt", "r") as f:

cap = cv2.VideoCapture("C:/Users/ASUS/Downloads/cars on road.mp4")

YOU NEED TO SET THE PATH OF THE DOWNLOADED FILES FROM YOUR SYSTEM THEN THE CODE WILL WORK. AND FOR THE 4TH LINE ABOUT VIDEO CAPTURE YOU HAVE TO USE A SAMPLE VIDEO FOR THAT CODE BUT IT SHOULD BE DIFFERENT FROM THE PREVIOUS ONE .

AFTER THE CODE IS EXECUTED YOU CAN SEE THAT THE CODE WILL CAPTURE THE OBJECTS FROM THE VIDEO AND IT WILL CONVERT THE VIDEO IN MULTIPLE PICTURES.

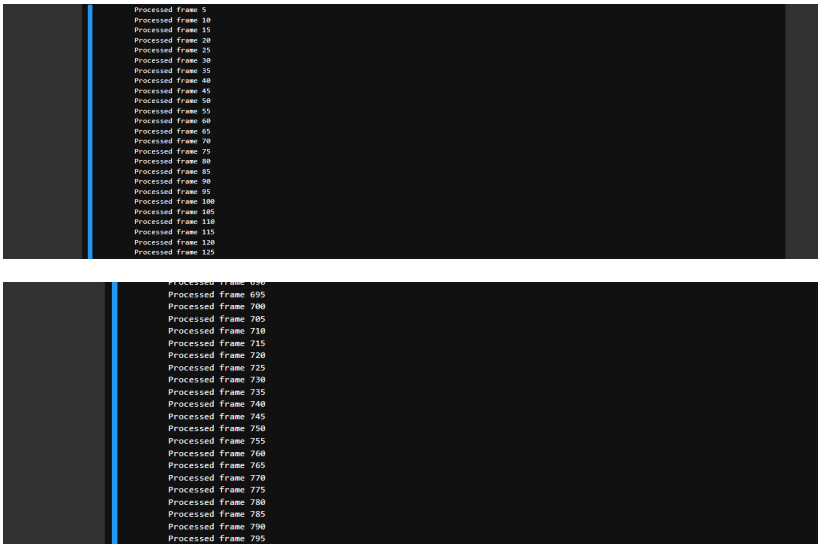
NOW USE THIS CODE -

```
import cv2
import numpy as np
import math

# Initialize video capture
input_video_path = "C:/Users/ASUS/Downloads/jupyter projects/Car Road Transportation.mp4"
output_video_path = "output.avi"

cap = cv2.VideoCapture(input_video_path)
if not cap.isOpened():
    print("Error: Unable to open video file.")
```

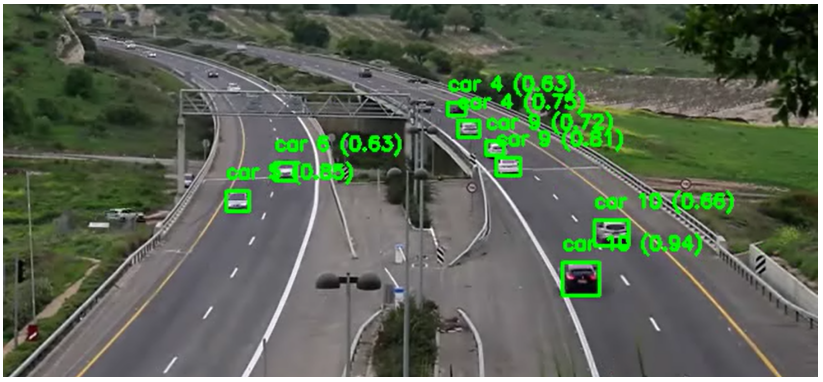
```
    exit()

frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Initialize video writer
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_video = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

# Load YOLO model
model_cfg = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.cfg"
model_weights = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.weights"
labels_path = "C:/Users/ASUS/Downloads/jupyter projects/coco.names.txt"

net = cv2.dnn.readNetFromDarknet(model_cfg, model_weights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

with open(labels_path, "r") as f:
    labels = f.read().strip().split("\n")

# Parameters for detection
conf_threshold = 0.6
nms_threshold = 0.3
blob_size = (320, 320)

# Object tracking variables
tracked_objects = {}
object_id = 0

# Function to calculate Euclidean distance
def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)

# Function to perform object detection
def detect_objects(frame):
    blob = cv2.dnn.blobFromImage(frame, scalefactor=1 / 255.0, size=blob_size, swapRB=True, crop=False)
    net.setInput(blob)
    layer_names = net.getUnconnectedOutLayersNames()
    layer_outputs = net.forward(layer_names)

    boxes = []
    confidences = []
    class_ids = []

    for output in layer_outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            if confidence > conf_threshold:
                box = detection[0:4] * np.array([frame_width, frame_height, frame_width, frame_height])
                (center_x, center_y, width, height) = box.astype("int")

                x = int(center_x - (width / 2))
                y = int(center_y - (height / 2))

                boxes.append([x, y, int(width), int(height)])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
    filtered_boxes = []
    filtered_class_ids = []
    filtered_confidences = []

    if len(indices) > 0:
        for i in indices.flatten():
            filtered_boxes.append(boxes[i])
            filtered_class_ids.append(class_ids[i])
            filtered_confidences.append(confidences[i])

    return filtered_boxes, filtered_class_ids, filtered_confidences

# Main loop
```

```
frame_count = 0
frame_skip = 5

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    if frame_count % frame_skip != 0:
        continue

    boxes, class_ids, confidences = detect_objects(frame)

    for box, class_id, confidence in zip(boxes, class_ids, confidences):
        (x, y, w, h) = box
        center_x, center_y = x + w // 2, y + h // 2

        # Tracking logic
        min_distance = float("inf")
        matched_id = None

        for obj_id, (prev_x, prev_y) in tracked_objects.items():
            distance = euclidean_distance((center_x, center_y), (prev_x, prev_y))
            if distance < min_distance and distance < 50:
                min_distance = distance
                matched_id = obj_id

        if matched_id is None:
            object_id += 1
            matched_id = object_id

        tracked_objects[matched_id] = (center_x, center_y)

        # Draw bounding box and label
        label = f"{labels[class_id]} {matched_id} ({confidence:.2f})"
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Write processed frame
    output_video.write(frame)
    print(f"Processed frame {frame_count}")

cap.release()
output_video.release()
print("Processing complete. Output saved as output.avi")
```

THE OUTPUT AFTER EXECUTION OF CODE -



```
Processed frame 1115
Processed frame 1120
Processed frame 1125
Processed frame 1130
Processed frame 1135
Processed frame 1140
Processed frame 1145
Processed frame 1150
Processed frame 1155
Processed frame 1160
Processed frame 1165
Processed frame 1170
Processing complete. Output saved as output.avi
```



4. DENSE OPTICAL FLOW

As a student, understanding **Dense Optical Flow** can help you learn how motion is tracked in videos. It's a technique in computer vision used to analyze **pixel-by-pixel movement** between two consecutive frames. Essentially, it allows us to see how objects or areas in a video move over time.

What Happens in the Code?

1. **Video Input**: The code reads a video file (`marathon.mp4`) frame by frame.
2. **Grayscale Conversion**: Each video frame is converted to **grayscale** to simplify calculations. Dense Optical Flow works on intensity differences between frames.
3. **Farneback Method**: This is the algorithm being used to calculate the **motion**:
   - It calculates how much each pixel moves (magnitude).
   - It also determines the direction (angle) in which the pixel moves.
4. **Visualization**:
   - The motion is visualized as **color flow**:
     - Colors represent the **direction** of movement.
     - The brightness of the color shows the **speed** (magnitude) of movement.
5. **Output Video**: The processed video with visualized motion (flow field) is saved as `dense_optical_flow_output.mp4`.

How Can You Relate to This?

- Imagine you're watching a **race video** (like the marathon used in the code). Dense Optical Flow helps you identify **how runners are moving** frame by frame. Faster movement appears brighter, and the direction of motion is shown by colors.
- It's like creating a **map of motion** across every frame in the video.

What Can You Use It For?

As a student, you can apply Dense Optical Flow to:

- **Object Tracking**: Track the motion of people, vehicles, or objects.
- **Activity Recognition**: Understand movements like walking, running, or cycling.
- **Sports Analysis**: Analyze player or ball movement in games.
- **Motion Detection**: Identify areas where movement occurs, like in surveillance videos.

In this project, you can experiment with videos that have:

- **Clear movement** (e.g., sports, traffic, or people walking).
- **Steady camera movement** (avoid shaky videos for better results).

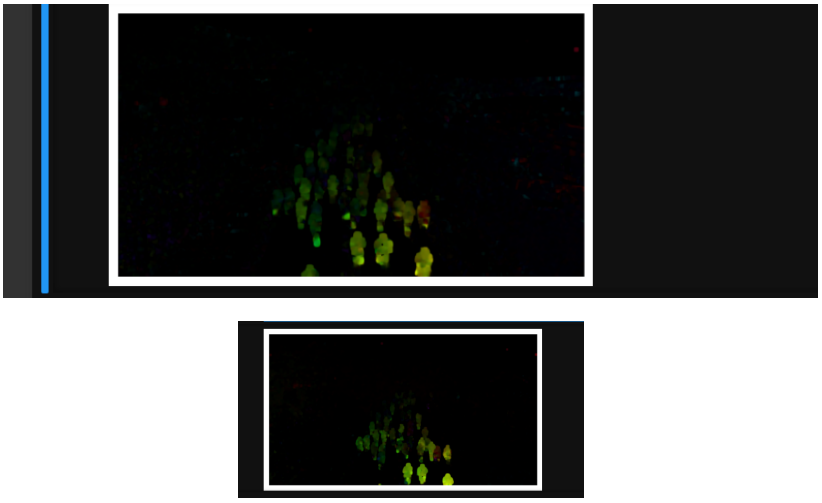
This exercise will give you a deeper understanding of motion analysis and how computer vision works to process video data.

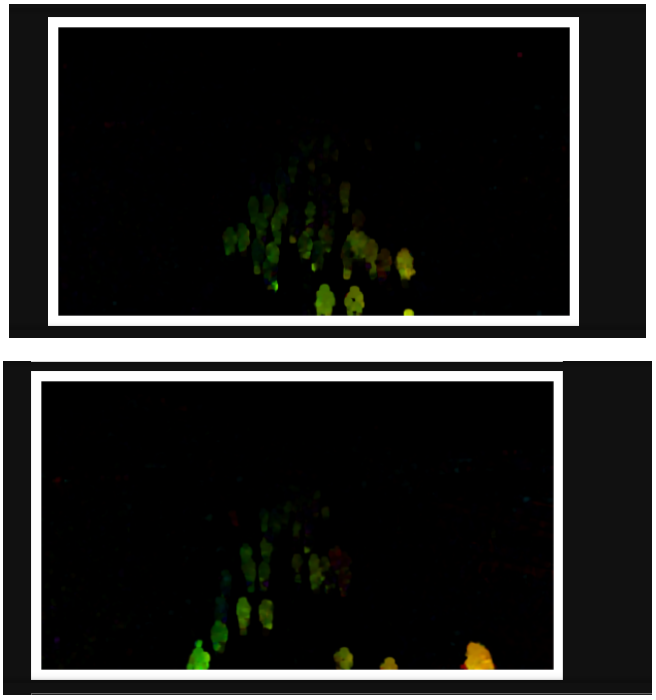
NOW HERE IS THE CODE FOR THE PROCESS ➖

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Load the video
cap = cv2.VideoCapture("C:/Users/ASUS/Downloads/marathon.mp4")

# Get video properties
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Initialize VideoWriter
out = cv2.VideoWriter('dense_optical_flow_output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))

# Read the first frame and convert it to grayscale
ret, first_frame = cap.read()
prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Convert the current frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate dense optical flow using Farneback method
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

    # Visualize optical flow
    magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])
    mask = np.zeros_like(frame)
    mask[..., 1] = 255
    mask[..., 0] = angle * 180 / np.pi / 2
    mask[..., 2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
    rgb = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)

    # Write the frame to the output video
    out.write(rgb)

    # Display the frame using Matplotlib
    plt.imshow(cv2.cvtColor(rgb, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.pause(0.001)  # Pause to simulate real-time playback

    # Update the previous frame
    prev_gray = gray

cap.release()
out.release()
plt.close()
```

NOW THE OUTPUT OF THE CODE ➖





5. SPARSE OPTICAL FLOW

Imagine you're watching a video of **moving cars** on a street. What if you wanted to know **how objects (like cars) move** from one frame of the video to the next? This is where **Optical Flow** comes in!

What is Optical Flow?

Optical Flow is a technique that helps track **how parts of an image (pixels)** move between frames of a video. For example:

- You see a car in one frame.
- In the next frame, the car has moved slightly to the right.
- Optical flow calculates this movement.

Why "Sparse" Optical Flow?

There are two types of optical flow:

1. **Dense Optical Flow**: Tracks **every pixel** in the image → **Very slow** and uses a lot of memory.
2. **Sparse Optical Flow**: Tracks only a **few important points** (like corners or edges) → **Faster and efficient**.

In simple words:

- Sparse Optical Flow focuses on **key points** (features) that are easier to track.
- It doesn't waste time on pixels that don't matter (like flat areas of a road).

How the Code Works (Step-by-Step)

1. **Load the Video**:
   - The program reads a video file (like a busy traffic video).
2. **Find Good Features to Track**:
   - It uses **Shi-Tomasi Corner Detection** to find "interesting points" in the video, such as corners or edges of objects.
   - These points are easier to follow as they are unique and stand out.
3. **Track the Motion of Points** (Using Lucas-Kanade Optical Flow):
   - The program calculates **where these points move** in the next frame.
   - For example, if a corner of a car was at position (100, 200) in Frame 1, it might move to (110, 205) in Frame 2.
4. **Draw Motion on the Video**:
   - **Green Lines**: Show the path of movement for each point.
   - **Red Circles**: Show the current position of the points.
5. **Output the Video**:
   - A new video is saved that shows the motion paths of the tracked points.

Why is this Useful?

- **Efficiency**:
  - Instead of tracking every pixel, we focus on a few key points.
  - This makes the program **faster** and less demanding on your computer.
- **Real-World Applications**:
  - **Traffic Monitoring**: Track cars or people moving.
  - **Robots**: Help robots follow moving objects.
  - **Sports Analytics**: Track players or balls on a field.
- **Easy to Learn**:
  - Sparse optical flow is a simple way to understand motion tracking before diving into advanced techniques.

How to Imagine It?

- Think of it like **attaching small markers** to the corners of a moving car.
- In every video frame, you check where those markers go.
- By connecting the markers' positions, you get a **path** that shows the car's movement.

In Short:

- Sparse Optical Flow tracks only **a few key points** in a video.
- It calculates **where those points move** between frames.
- This is done using Lucas-Kanade Optical Flow, a popular and fast technique.

It's a great starting point for understanding how computers "see" motion! 🚗💡

NOW LETS SEE THE CODE ➖

```
import cv2
import numpy as np

# Load the video
video_path = "C:/Users/ASUS/Downloads/jupyter projects/Busy Traffic street.mp4"
cap = cv2.VideoCapture(video_path)

# Check if the video opened successfully
if not cap.isOpened():
    print("Error: Cannot open the video file. Check the file path.")
    exit()

# Get video properties
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Initialize VideoWriter
out = cv2.VideoWriter('sparse_optical_flow_output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (frame_width, frame_height))

# Parameters for Lucas-Kanade optical flow
lk_params = dict(winSize=(15, 15), maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Parameters for Shi-Tomasi corner detection
feature_params = dict(maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

# Read the first frame and find corners
ret, old_frame = cap.read()
if not ret:
    print("Error: Cannot read the first frame of the video.")
    cap.release()
    exit()

old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

# Check if features are found
if p0 is None:
    print("Error: No features to track. Adjust the feature_params.")
    cap.release()
    exit()

# Create a mask for drawing
mask = np.zeros_like(old_frame)

while cap.isOpened():
```

```
    ret, frame = cap.read()
    if not ret:
        print("End of video reached or cannot read a frame.")
        break

    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate optical flow
    if p0 is not None:
        p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

        # Select good points
        if p1 is not None and st is not None:
            good_new = p1[st == 1]
            good_old = p0[st == 1]

            # Draw the tracks
            for i, (new, old) in enumerate(zip(good_new, good_old)):
                a, b = new.ravel()
                c, d = old.ravel()
                mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
                frame = cv2.circle(frame, (int(a), int(b)), 5, (0, 0, 255), -1)

            img = cv2.add(frame, mask)

            # Update the previous frame and points
            old_gray = frame_gray.copy()
            p0 = good_new.reshape(-1, 1, 2)
        else:
            print("Warning: No good points to track. Resetting points.")
            mask = np.zeros_like(frame)  # Reset mask
            p0 = cv2.goodFeaturesToTrack(frame_gray, mask=None, **feature_params)
            old_gray = frame_gray.copy()
    else:
        print("Warning: Points lost, re-detecting features.")
        mask = np.zeros_like(frame)
        p0 = cv2.goodFeaturesToTrack(frame_gray, mask=None, **feature_params)
        old_gray = frame_gray.copy()

    # Write the frame to the output video
    out.write(frame)

    # Display the frame (optional for testing)
    # cv2.imshow('Sparse Optical Flow', img)
    # if cv2.waitKey(1) & 0xFF == ord('q'):
    #     break

cap.release()
out.release()
print("Sparse Optical Flow processing complete. Output saved as 'sparse_optical_flow_output.mp4'.")
```

NOW THE OUTPUT OF THE CODE ➖

```
End of video reached or cannot read a frame.
Sparse Optical Flow processing complete. Output saved as 'sparse_optical_flow_output.mp4'.
```

IMPLEMENTATION OF UI ON PROJECTS

TO IMPLEMENT THE UI FOR THE PROJECTS INSTALL - GRADIO ON GOOGLE COLLAB OR JUPYTER NOTEBOOK

```
!pip install gradio
```

AFTER INSTALLATION IS COMPLETE DO USE THIS COMMAND TO CHECK FOR UPGRADED VERSION -

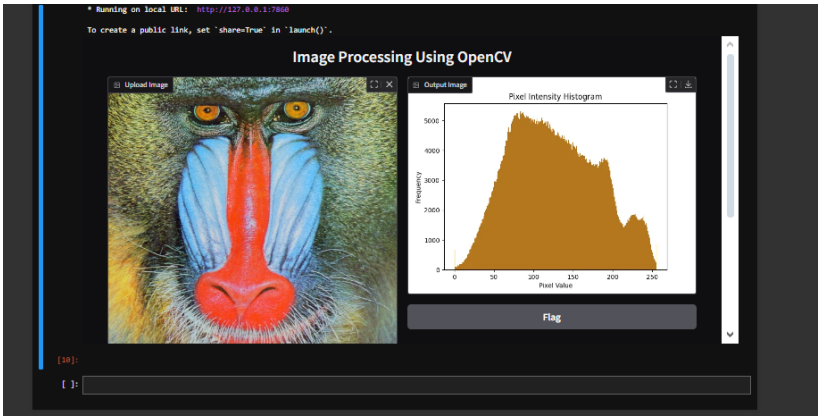
```
!pip install --upgrade gradio
```

1. IMAGE PROCESSING USING OPENCV

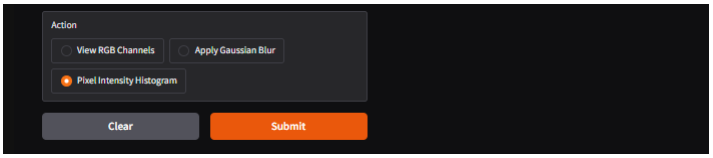
NOW USE THIS CODE FOR GENERATING THE USER INTERFACE ON YOUR GOOGLE COLLAB OR JUPYTER NOTEBOOK SHELL TO IMPLEMENT THE U.I

```
import gradio as gr
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Function for processing the image
def process_image(image, action):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB for display
    if action == "View RGB Channels":
        channels = [image[:, :, i] for i in range(3)]  # Split into R, G, B
        fig, axs = plt.subplots(1, 3, figsize=(10, 5))
        for i, channel in enumerate(channels):
            axs[i].imshow(channel, cmap=["Reds", "Greens", "Blues"][i])
            axs[i].axis('off')
            axs[i].set_title(["Red", "Green", "Blue"][i])
        plt.tight_layout()
        plt.savefig("rgb_channels.png")
        plt.close()
        return cv2.imread("rgb_channels.png")  # Return the saved image as output
    elif action == "Apply Gaussian Blur":
        blurred = cv2.GaussianBlur(image, (15, 15), 0)
        return blurred  # Return blurred image
    elif action == "Pixel Intensity Histogram":
        plt.hist(image.ravel(), bins=256, range=(0, 256))
        plt.title("Pixel Intensity Histogram")
        plt.xlabel("Pixel Value")
        plt.ylabel("Frequency")
        plt.savefig("histogram.png")
        plt.close()
        return cv2.imread("histogram.png")  # Return the saved histogram image

# Gradio interface
interface1 = gr.Interface(
    fn=process_image,
    inputs=[
        gr.Image(type="numpy", label="Upload Image"),
        gr.Radio(["View RGB Channels", "Apply Gaussian Blur", "Pixel Intensity Histogram"], label="Action")
    ],
    outputs=gr.Image(type="numpy", label="Output Image"),
    title="Image Processing Using OpenCV"
)

# Launch
interface1.launch()
```

AND HERE I GOT MY RESULTS . ITS REALLY AMAZING -



AND THERE WILL BE A LOCAL URL FOR YOUR SYSTEM SAVE IT ON YOUR BROWSER WHICH COULD BE USEFUL FOR FUTURE USE.



AFTER SCROLLING DOWN YOU WILL GET THESE OPTIONS LIKE WE HAVE MADE IN OUR CODING AND THOSE IMAGES CAN BE DOWNLOADED .

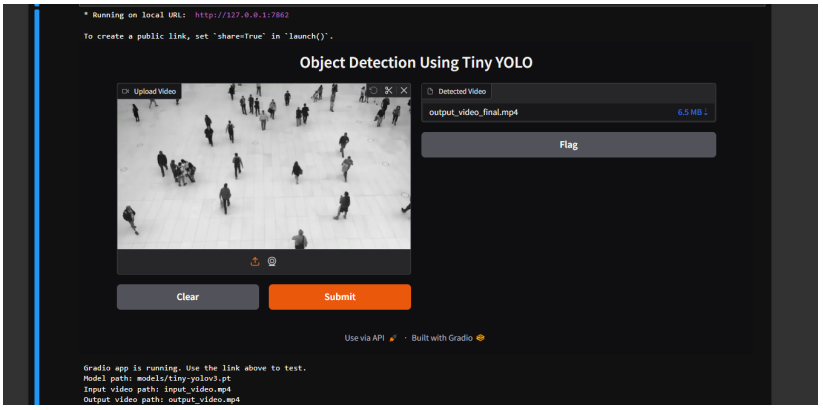
2. OBJECT DETECTION USING TINY YOLO

CODE -

```
import gradio as gr
from imageai.Detection import VideoObjectDetection
import os
import shutil
import warnings

# Suppress ResourceWarning if non-critical
warnings.filterwarnings("ignore", category=ResourceWarning)

# Function for object detection
def detect_objects(video_path):
    detector = VideoObjectDetection()
    model_path = "models/tiny-yolov3.pt"

    # Check if the model file exists
    if not os.path.exists(model_path):
        return "Error: Model file not found!"

    print("Model path:", model_path)
    detector.setModelTypeAsTinyYOLOv3()
    detector.setModelPath(model_path)
    detector.loadModel()

    # Define paths for input and output
    input_path = "input_video.mp4"
    output_path = "output_video.mp4"  # Expected output path
    final_output_path = "output_video_final.mp4"  # Renamed output path

    # Copy uploaded video to the input path
    shutil.copy(video_path, input_path)
    print("Input video path:", input_path)
    print("Output video path:", output_path)

    # Perform object detection
    try:
        detection_results = detector.detectObjectsFromVideo(
            input_file_path=input_path,
            output_file_path=output_path,
            frames_per_second=10,
            minimum_percentage_probability=20,  # Lowered for better detection
            save_detected_video=True
        )
        print("Detection results:", detection_results)
    except Exception as e:
        print("Error during detection:", str(e))
        return f"Error during detection: {str(e)}"

    # Check if the file with double extension exists
    if os.path.exists(output_path + ".mp4"):
        shutil.move(output_path + ".mp4", final_output_path)
        return final_output_path

    # Check if the regular output file exists
    if os.path.exists(output_path):
        shutil.move(output_path, final_output_path)
        return final_output_path

    return "Error: Output video was not generated!"

# Gradio interface
interface2 = gr.Interface(
```

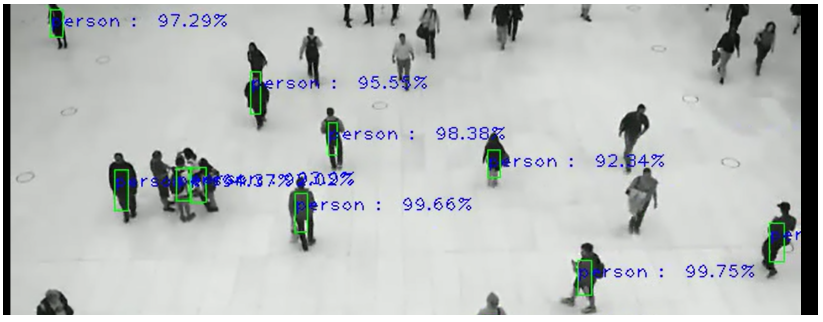
```
    fn=detect_objects,
    inputs=gr.Video(label="Upload Video"),
    outputs=gr.File(label="Detected Video"),
    title="Object Detection Using Tiny YOLO"
)

# Launch Gradio app
app = interface2.launch()

# Ensure cleanup on exit
try:
    print("Gradio app is running. Use the link above to test.")
    input("Press Enter to stop the app...\n")
finally:
    app.close()
    print("Gradio app closed.")
```

AFTER USING THE CODE YOU WILL GET THIS RESULT -



ALSO IT GAVE ME THE OUTPUT VIDEO FILE WHICH I CAN INSTALL .

AND THE RESULT IT ALSO PERFECT -



3. OBJECT TRACKING

IN THIS ON FOR THE CODE YOU NEED THE 3 FILES WHICH WE HAVE USED ON THE OBJECT TRACKING PROJECT -

```
# Load YOLO model
model_cfg = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.cfg"
model_weights = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.weights"
labels_path = "C:/Users/ASUS/Downloads/jupyter projects/coco.names.txt"
net = cv2.dnn.readNetFromDarknet(model_cfg, model_weights)
```

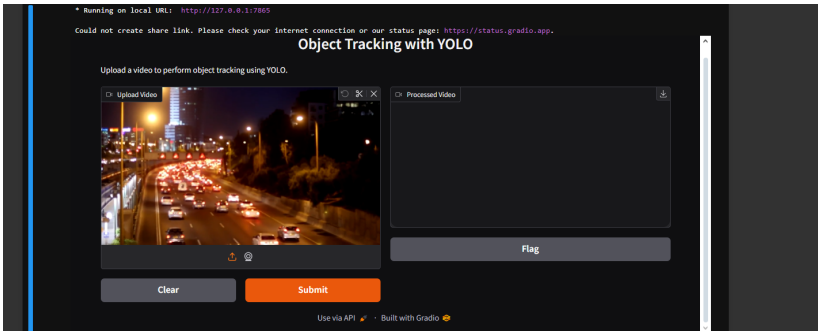
NOW THE CODE TO DO THE U.I. PART FOR THE 3RD PROJECT —

```
import gradio as gr
import cv2
import os
import numpy as np
import math

def process_video(input_video_path):
    # Set output video path
    output_video_path = "output.avi"
```

```
    # Initialize video capture
    cap = cv2.VideoCapture(input_video_path)
    if not cap.isOpened():
        return "Error: Unable to open video file."

    # Video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Initialize VideoWriter
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    output_video = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

    # Load YOLO model
    model_cfg = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.cfg"
    model_weights = "C:/Users/ASUS/Downloads/jupyter projects/yolov4.weights"
    labels_path = "C:/Users/ASUS/Downloads/jupyter projects/coco.names.txt"
    net = cv2.dnn.readNetFromDarknet(model_cfg, model_weights)

    with open(labels_path, "r") as f:
        labels = f.read().strip().split("\n")

    # Detection and tracking parameters
    conf_threshold = 0.6
    nms_threshold = 0.3
    blob_size = (320, 320)

    def detect_objects(frame):
        blob = cv2.dnn.blobFromImage(frame, scalefactor=1 / 255.0, size=blob_size)
        net.setInput(blob)
        layer_names = net.getUnconnectedOutLayersNames()
        layer_outputs = net.forward(layer_names)
        boxes, confidences, class_ids = [], [], []
        for output in layer_outputs:
            for detection in output:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > conf_threshold:
                    box = detection[0:4] * np.array([frame_width, frame_height, frame_width, frame_height])
                    (center_x, center_y, width, height) = box.astype("int")
                    x = int(center_x - (width / 2))
                    y = int(center_y - (height / 2))
                    boxes.append([x, y, int(width), int(height)])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)
        indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
        filtered_boxes, filtered_class_ids, filtered_confidences = [], [], []
        if len(indices) > 0:
            for i in indices.flatten():
                filtered_boxes.append(boxes[i])
                filtered_class_ids.append(class_ids[i])
                filtered_confidences.append(confidences[i])
        return filtered_boxes, filtered_class_ids, filtered_confidences

    # Main processing loop
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        boxes, class_ids, confidences = detect_objects(frame)
        for box, class_id, confidence in zip(boxes, class_ids, confidences):
            (x, y, w, h) = box
            label = f"{labels[class_id]} ({confidence:.2f})"
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        output_video.write(frame)

    cap.release()
    output_video.release()
    return output_video_path

# Gradio interface
def gradio_interface(video_file):
    output_path = process_video(video_file)  # Directly use the file path
```

```
    return output_path

# Create the Gradio interface
interface = gr.Interface(
    fn=gradio_interface,
    inputs=gr.Video(label="Upload Video"),
    outputs=gr.Video(label="Processed Video"),
    title="Object Tracking with YOLO",
    description="Upload a video to perform object tracking using YOLO."
)

# Launch the interface on default port 7860
interface.launch(share=True)
```

THE OUT PUT OF THE CODE -



THAT'S HOW THE WHOLE PROJECT IS COMPLETED .