# REFLECTIVE JOURNAL

**GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY**

**CS3042**

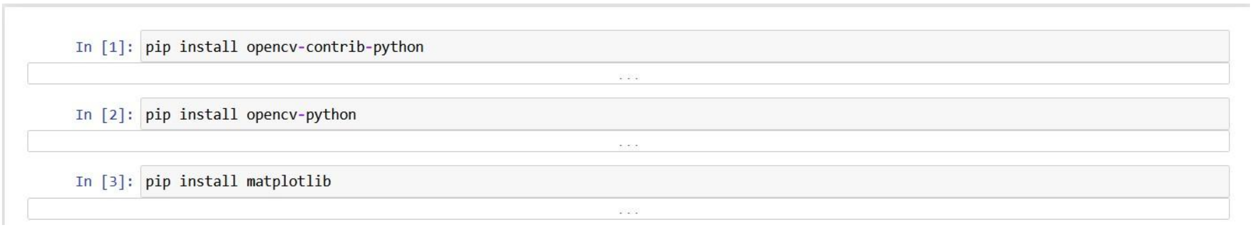
**IMAGE PROCESSING AND**

**COMPUTER VISION**

**INTAKE 39**

Author : NPHP Hemantha

# SHAPES

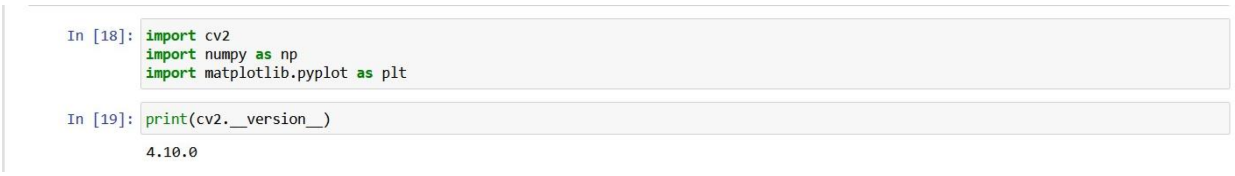


```
In [1]: pip install opencv-contrib-python
```

```
In [2]: pip install opencv-python
```

```
In [3]: pip install matplotlib
```

- If you need to track an object across multiple video frames or recognize handwritten text, ***opencv-contrib-python*** includes tools that can help.
- ***Open Source Computer Vision*** Library is used for computer vision tasks like image processing, video capture, and object detection.
- ***Matplotlib*** is used for creating visualizations in Python, such as graphs, charts, and plots.



```
In [18]: import cv2
         import numpy as np
         import matplotlib.pyplot as plt
```

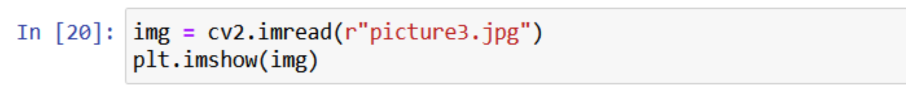
```
In [19]: print(cv2.__version__)

         4.10.0
```

Import the necessary libraries for image processing and plotting.

- cv2 is OpenCV, a library for computer vision.
- numpy (imported as np) is used for numerical operations.
- matplotlib.pyplot (imported as plt) is used for displaying images.

Prints the version of OpenCV installed.

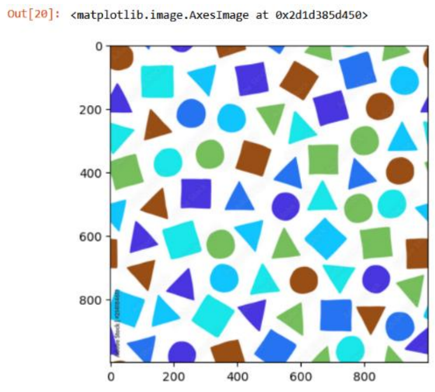
- Reason: Useful for ensuring compatibility, as some functions might behave differently in different versions.



```
In [20]: img = cv2.imread(r"picture3.jpg")
         plt.imshow(img)
```

Reads the image from the specified path and displays it using Matplotlib.

- cv2.imread() loads the image.
- plt.imshow() shows the image.



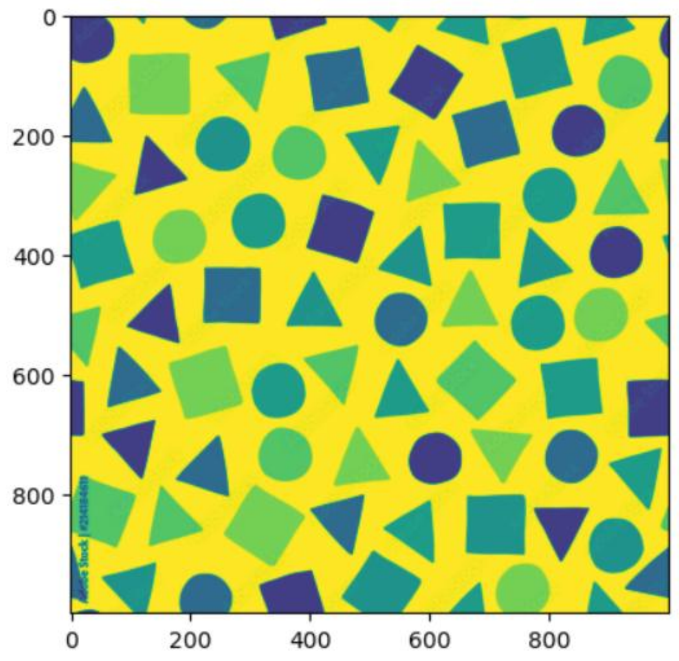
```
Out[20]: <matplotlib.image.AxesImage at 0x2d1d385d450>
```

```
In [21]: gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
         plt.imshow(gray)
```

Converts the image to grayscale.

- cv2.cvtColor() changes the color space of the image from BGR to grayscale.

```
Out[21]: <matplotlib.image.AxesImage at 0x2d1d54f4450>
```

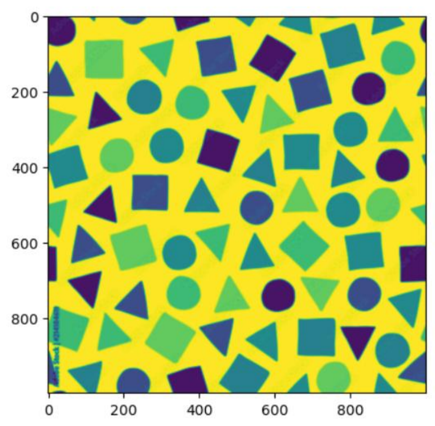


```
In [22]: blured = cv2.GaussianBlur(gray,(7,7),0)
         plt.imshow(blured)
```

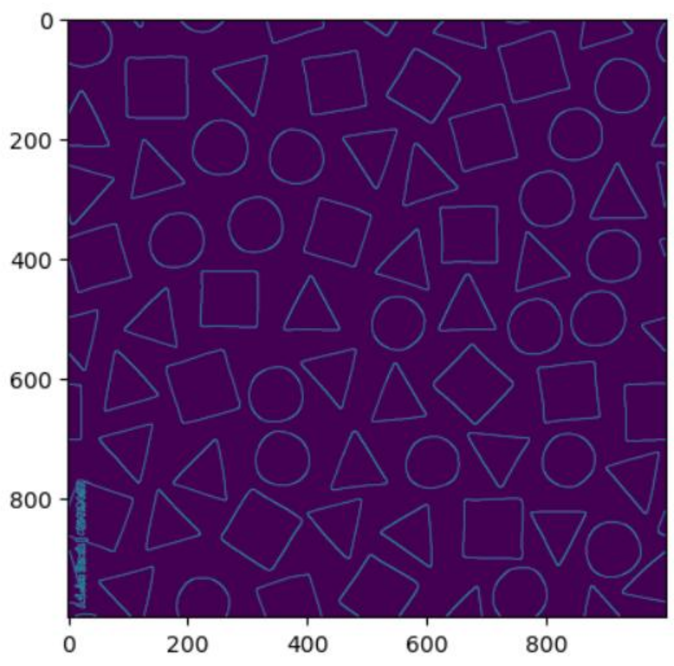
Blurs the grayscale image to reduce noise.

- cv2.GaussianBlur() applies a Gaussian filter with a kernel size of 7x7.

```
Out[22]: <matplotlib.image.AxesImage at 0x2d1d5518f10>
```



```
In [23]: edges = cv2.Canny(blured,30,120)
         plt.imshow(edges)

Out[23]: <matplotlib.image.AxesImage at 0x2d1d559cad0>
```



Detects edges in the blurred image.

- cv2.Canny() performs Canny edge detection with thresholds of 30 and 120.

```
In [24]: contours, _= cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Finds contours in the edge-detected image.

- cv2.findContours() retrieves the contours from the edges.

```
In [25]: circles = cv2.HoughCircles(
                 edges,
                 cv2.HOUGH_GRADIENT,
                 dp=1,
                 minDist=50,
                 param1=100,
                 param2=30,
                 minRadius=5,
                 maxRadius=50
         )

         if circles is not None:
                 circles = np.uint16(np.around(circles))
                 for circle in circles[0, :]:
                     center = (circle[0], circle[1])
                     radius = circle[2]

                     # Compute the top-left and bottom-right coordinates of the bounding rectangle
                     x, y = center[0] - radius, center[1] - radius
                     w, h = 2 * radius, 2 * radius

                     # Draw the bounding rectangle
                     cv2.rectangle(img, (int(x), int(y)), (int(x + w), int(y + h)), (0, 0, 0), 2)


         cv2.imshow('Circles with Bounding Boxes', img)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
```

Detects circles in the edge-detected image using the Hough Circle Transform.

cv2.HoughCircles() finds circles with:

- dp=1: The inverse ratio of the accumulator resolution to the image resolution.
- minDist=50: Minimum distance between the centers of detected circles.
- param1=100: Higher threshold for the Canny edge detector (lower threshold is half of this).
- param2=30: Accumulator threshold for the circle centers at the detection stage.
- minRadius=5: Minimum circle radius.
- maxRadius=50: Maximum circle radius.

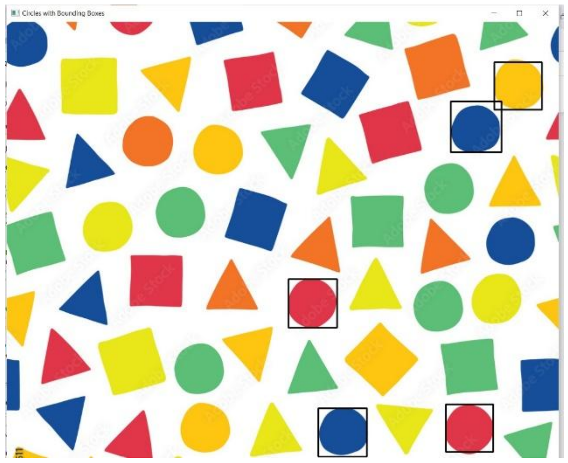
Draws rectangles around detected circles.

- np.uint16(np.around(circles)): Converts the circle parameters to integer and rounds them.
- Iterates through each detected circle:
- circle[0], circle[1]: Coordinates of the circle's center.
- circle[2]: Radius of the circle.
- Computes the top-left and bottom-right coordinates of a rectangle around the circle.
- cv2.rectangle(): Draws the rectangle on the original image.

Shows the image with the detected circles and bounding boxes in a window.

- cv2.imshow(): Displays the image in a window.
- cv2.waitKey(0): Waits indefinitely for a key press.
- cv2.destroyAllWindows(): Closes the window.

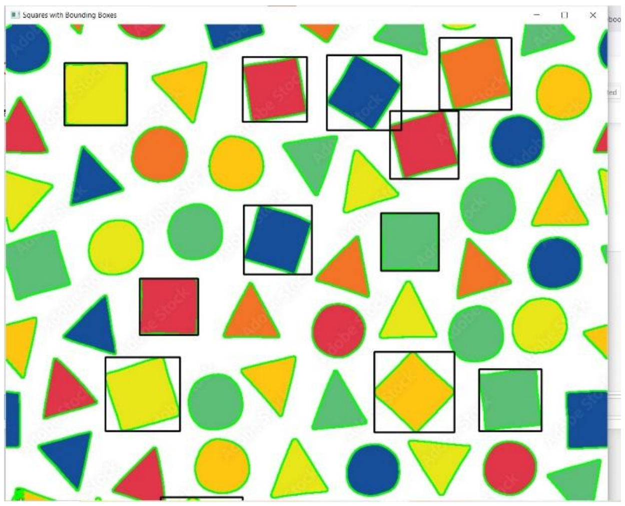
Output:

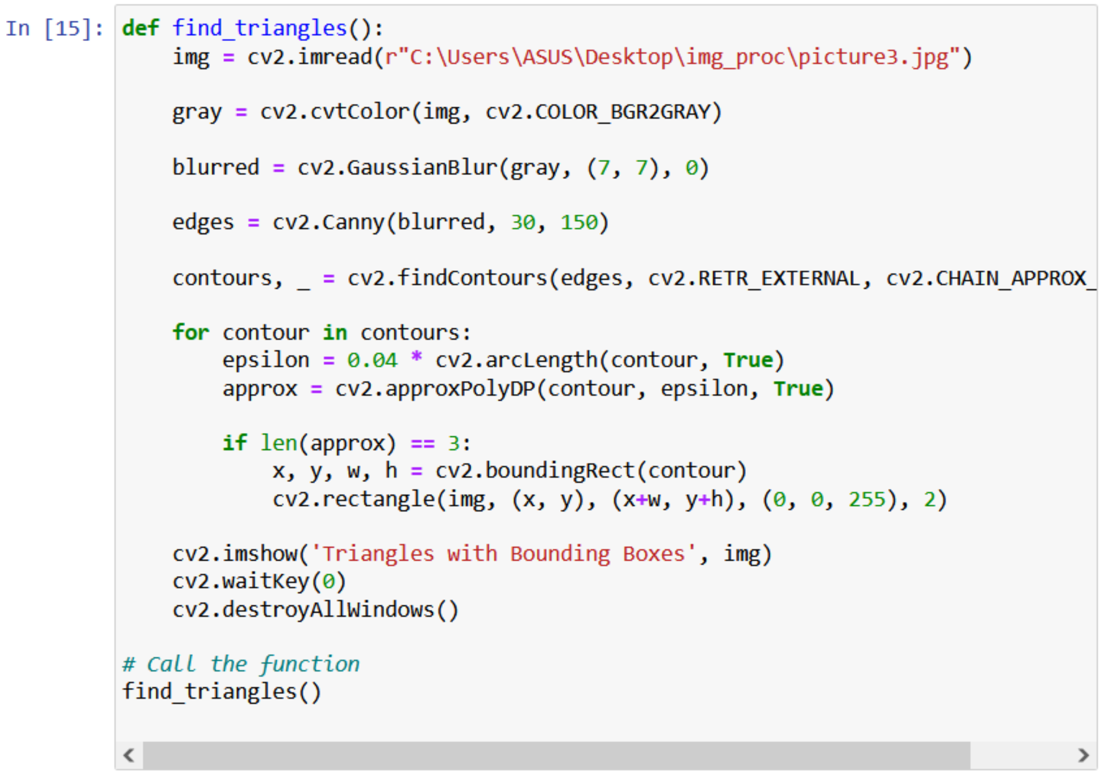


```
In [17]: for contour in contours:
             epsilon = 0.04 * cv2.arcLength(contour, True)
             approx = cv2.approxPolyDP(contour, epsilon, True)

             if len(approx) == 4:  # square has 4
                 area = cv2.contourArea(contour)
                 if area > 2000:
                     x, y, w, h = cv2.boundingRect(contour)
                     cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 0), 2)

         cv2.imshow('Squares with Bounding Boxes', img)
         cv2.waitKey(0)
         cv2.destroyAllWindows()
```

- First line starts a loop that goes through each contour found in the previous findContours step.
  Approximates each contour to a polygon with fewer vertices.
- epsilon is a parameter that specifies the maximum distance between the original contour and its approximation, calculated as 4% of the contour's perimeter.
- cv2.arcLength(contour, True) calculates the perimeter of the contour.
- cv2.approxPolyDP(contour, epsilon, True) approximates the contour to a polygon with fewer vertices based on epsilon.
- Checks if the approximated polygon has 4 vertices.
  Computes the area of the contour and checks if it is larger than a threshold (2000 in this case).
- cv2.contourArea(contour) calculates the area enclosed by the contour.
  Draws a rectangle around the detected square.
- cv2.boundingRect(contour) finds the bounding rectangle for the contour.
- x, y are the top-left corner coordinates of the rectangle.
- w, h are the width and height of the rectangle.
- cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 0), 2) draws the rectangle on the original image with a black border of thickness 2.

Output:

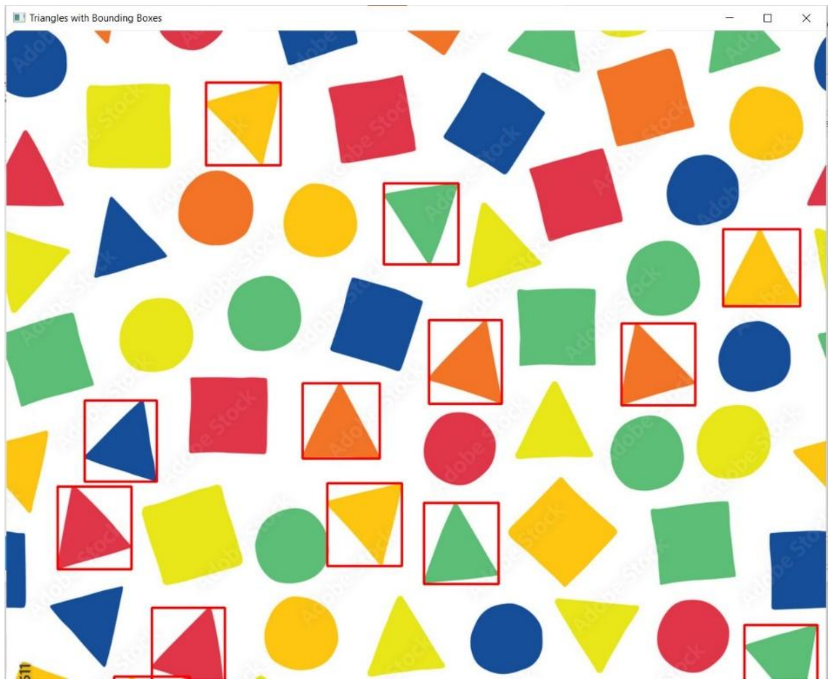




```
In [15]: def find_triangles():
             img = cv2.imread(r"C:\Users\ASUS\Desktop\img_proc\picture3.jpg")

             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

             blurred = cv2.GaussianBlur(gray, (7, 7), 0)

             edges = cv2.Canny(blurred, 30, 150)

             contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_

             for contour in contours:
                 epsilon = 0.04 * cv2.arcLength(contour, True)
                 approx = cv2.approxPolyDP(contour, epsilon, True)

                 if len(approx) == 3:
                     x, y, w, h = cv2.boundingRect(contour)
                     cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)

             cv2.imshow('Triangles with Bounding Boxes', img)
             cv2.waitKey(0)
             cv2.destroyAllWindows()

         # Call the function
         find_triangles()
```

- epsilon determines the precision for approximating the shape.
- cv2.approxPolyDP simplifies the contour shape.
- if len(approx) == 3 checks if the shape has 3 corners (i.e., it's a triangle).
- cv2.boundingRect computes a rectangle around the triangle.
- cv2.rectangle draws this rectangle on the image.

Output:



# FACE DETECTION

**Install packages:**

face_recognition: A library for recognizing and manipulating faces.

opencv-python: A library for computer vision tasks.

Reason: Installing face_recognition ensures you have the necessary tools for face detection and comparison.

**Import Libraries**

Imports libraries for face recognition, image processing, and plotting.

- face_recognition: For detecting and comparing faces.
- cv2: For image manipulation.
- numpy: For numerical operations.
- matplotlib.pyplot: For displaying images.

**Load and Display the Main Image**

Loads and displays the main image using Matplotlib.

- %matplotlib inline: Ensures Matplotlib plots are shown inline in Jupyter notebooks.
- mpimg.imread('duals.jpg'): Reads the image file.
- plt.imshow(img): Displays the image.

Reason: Shows the main image to visually confirm the image to be processed.

**Define Function for Face Comparison**

Defines a function that compares faces in two images and marks matches.

Reason: Encapsulates the face detection and comparison logic in a reusable function.

**Load Images Inside Function**

Loads the main and target images using face_recognition.

Reason: You need the images loaded into memory to process them for face recognition.

**Find Face Locations and Encodings**

Detects faces and generates face encodings for both images.

- face_recognition.face_locations(): Finds face positions in the images.

- face_recognition.face_encodings(): Generates numerical representations (encodings) of the faces.

Reason: Face locations help to crop the face regions, and encodings are used for comparison to identify matches.

**Create Copy of Main Image for Marking**:

Creates a copy of the main image to draw markings without changing the original image.

Reason: Helps in visualizing the results without modifying the input image.

**Compare Faces and Mark Matches**

Compares each face in the main image to faces in the target image and marks matches.

- for: Iterates through detected faces in the main image.
- face_recognition.compare_faces(): Compares each main face encoding to all target face encodings to check for matches.
- If a match (True) is found:
- ➢ target_index = results.index(True): Finds the first index where a match occurs.
- ➢ Draws a red rectangle around the matching face in the main image.
- ➢ Target_index is the position of the first match in the list of results returned by the face_recognition.compare_faces function. This index helps identify which face in the target image matches a face in the main image.

Reason: To visually indicate which faces in the main image match any face in the target image.

**Display Marked Image**

Displays the main image with marked faces using Matplotlib.

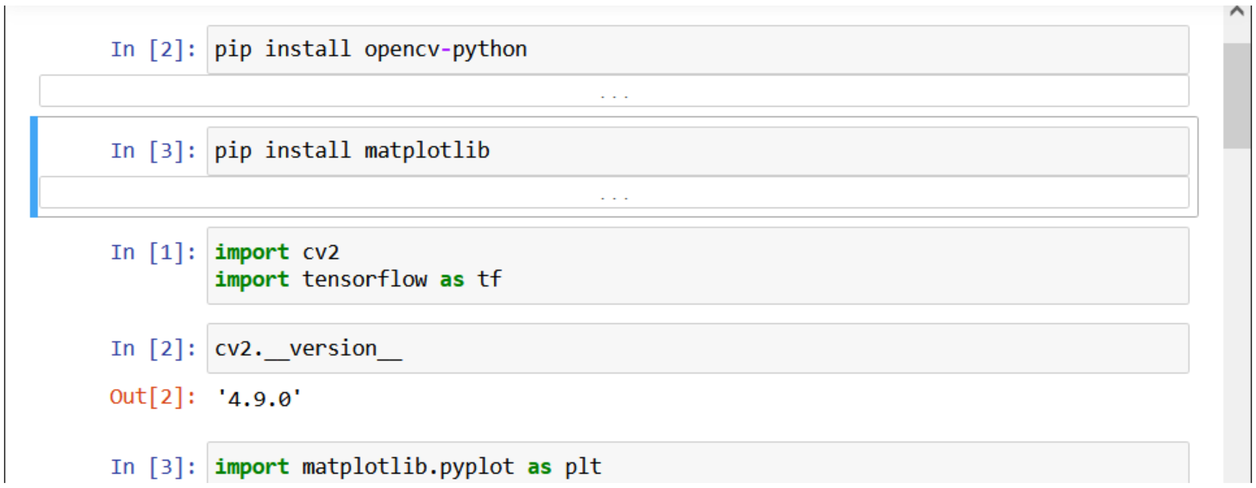
Reason: To visualize the final result with matches highlighted.

**Specify Image Paths and Call Function**

Specifies the paths to the main and target images and calls the function to compare faces and mark matches.

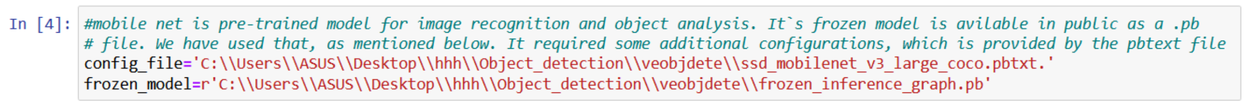
Reason: Provides the input to the function and triggers the face comparison and marking process.

## OBJECT DETECTION



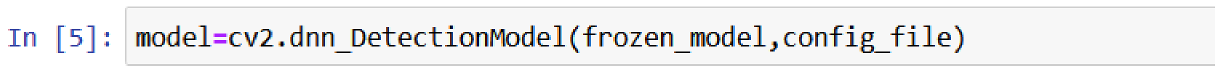
```
In [2]: pip install opencv-python

In [3]: pip install matplotlib

In [1]: import cv2
        import tensorflow as tf

In [2]: cv2.__version__

Out[2]: '4.9.0'

In [3]: import matplotlib.pyplot as plt
```

- Installs or updates the TensorFlow, OpenCV, and Matplotlib libraries.
- Imports OpenCV and TensorFlow libraries. cv2.__version__ prints the version of OpenCV.
- Imports Matplotlib for plotting images.



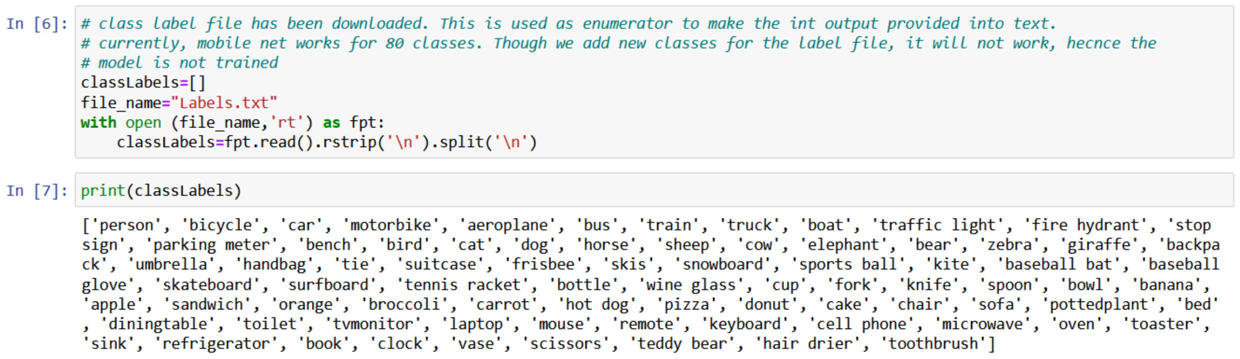
```
In [4]: #mobile net is pre-trained model for image recognition and object analysis. It`s frozen model is avilable in public as a .pb
        # file. We have used that, as mentioned below. It required some additional configurations, which is provided by the pbtext file
        config_file='C:\\Users\\ASUS\\Desktop\\hhh\\Object_detection\\veobjdete\\ssd_mobilenet_v3_large_coco.pbtxt.'
        frozen_model=r'C:\\Users\\ASUS\\Desktop\\hhh\\Object_detection\\veobjdete\\frozen_inference_graph.pb'
```

- Specifies paths to the model configuration file (.pbtxt) and the pre-trained model (.pb).



```
In [5]: model=cv2.dnn_DetectionModel(frozen_model,config_file)
```

-  Loads the model using OpenCV's deep neural network module (dnn).
- Initializes the model for object detection using the specified configuration and weights.



```
In [6]: # class label file has been downloaded. This is used as enumerator to make the int output provided into text.
        # currently, mobile net works for 80 classes. Though we add new classes for the label file, it will not work, hecnce the
        # model is not trained
        classLabels=[]
        file_name="Labels.txt"
        with open (file_name,'rt') as fpt:
            classLabels=fpt.read().rstrip('\n').split('\n')

In [7]: print(classLabels)

        ['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop
        sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpa
        ck', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball
        glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana',
        'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'sofa', 'pottedplant', 'bed'
        , 'diningtable', 'toilet', 'tvmonitor', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
        'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']
```

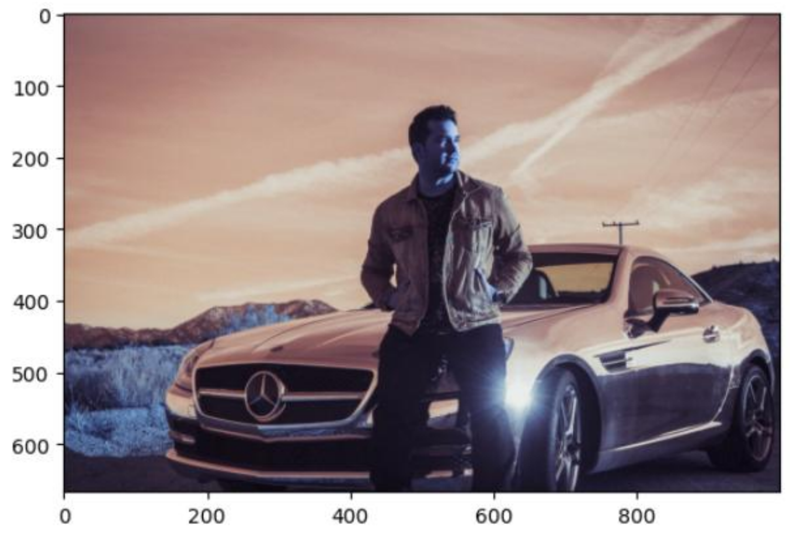
- Reads class labels from a file and stores them in a list. Class labels are used to convert numeric class predictions into human-readable text. The file typically contains names of objects that the model can detect (e.g., "person", "car").
- Prints the list of class labels. To verify that class labels are loaded correctly.

```
In [8]: # some adjustments are made to the input.
        # file is organized and , once satisfying
        model.setInputSize(320,320)
        model.setInputScale(1.0/127.5)
        model.setInputMean((127.5,127.5,127.5))
        model.setInputSwapRB(True)
```

Sets parameters for model input:

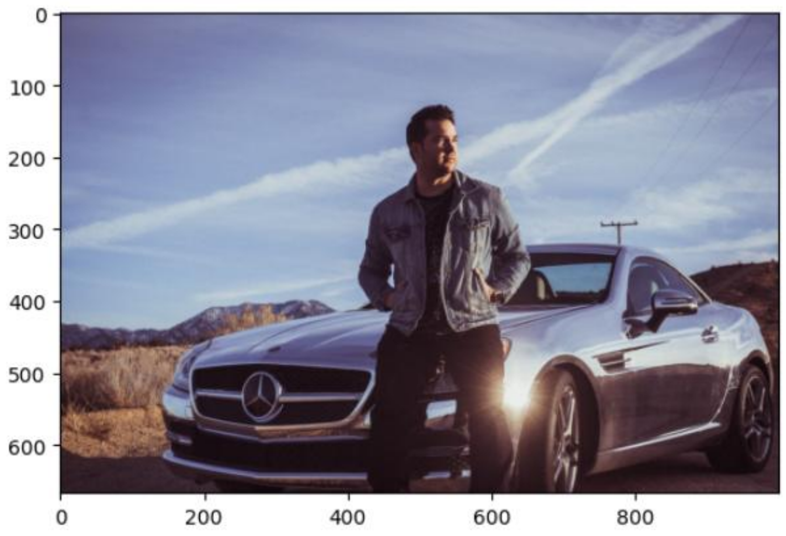
- setInputSize(320,320): Resizes the input image to 320x320 pixels.
- setInputScale(1.0/127.5): Scales input pixel values.
- setInputMean((127.5,127.5,127.5)): Subtracts a mean value to normalize input.
- setInputSwapRB(True): Swaps the red and blue channels (OpenCV uses BGR, while most models use RGB).
- These adjustments ensure the input image matches the format expected by the model.

```
In [9]: img=cv2.imread('download.jpg')
        plt.imshow(img) # this is why we need pyplotlib
Out[9]: <matplotlib.image.AxesImage at 0x20944f56590>
```



Load the image.

```
In [10]: plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB)) # some color cha

Out[10]: <matplotlib.image.AxesImage at 0x209487e0f10>
```



- Converts the image from BGR to RGB for correct color display in Matplotlib.

```
In [11]: ClassIndex,confidence,bbox=model.detect(img,confThreshold=0.5) # if the detection confidence is more than 50% display the o/p

In [12]: print(ClassIndex) # numerical predications of classes. Check with the class label list, strating from 1. Then, those are
         # man and car

         [1 3]
```

Detects objects in the image with a confidence threshold of 50%.

- model.detect(): Performs object detection.
- confThreshold=0.5: Only considers detections with confidence above 50%.

Prints the indices of detected classes.

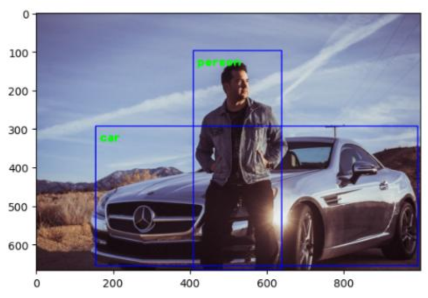
```
In [13]: font_scale=2
         font=cv2.FONT_HERSHEY_PLAIN # bounding box text
         for ClassInd,conf,boxes in zip(ClassIndex.flatten(),confidence.flatten(),bbox):
             # configuration of bounding box and associated text
             cv2.rectangle(img,boxes,(255,0,0),2)
             cv2.putText(img,classLabels[ClassInd-1],(boxes[0]+10,boxes[1]+40),font,fontScale=font_scale,color=(0,255,0),thickness=3)

In [14]: plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

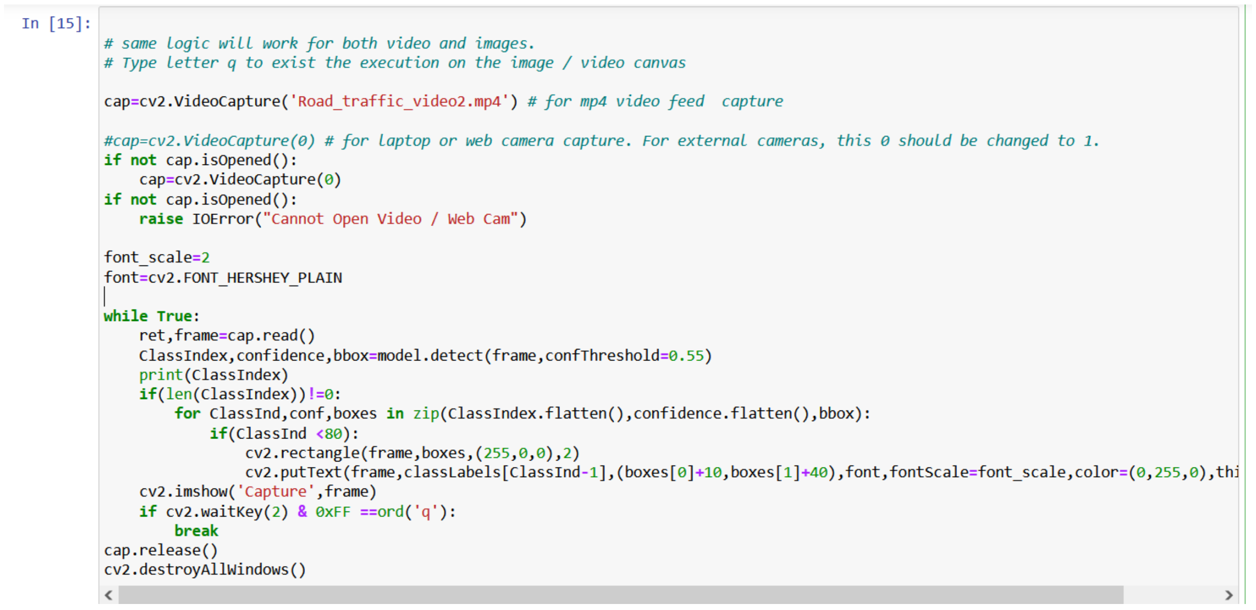
Draws rectangles and labels for detected objects on the image.

- cv2.rectangle(): Draws a bounding box.
- cv2.putText(): Adds a label near the bounding box.

Displays the image with bounding boxes and labels.

```
Out[14]: <matplotlib.image.AxesImage at 0x20948843490>
```

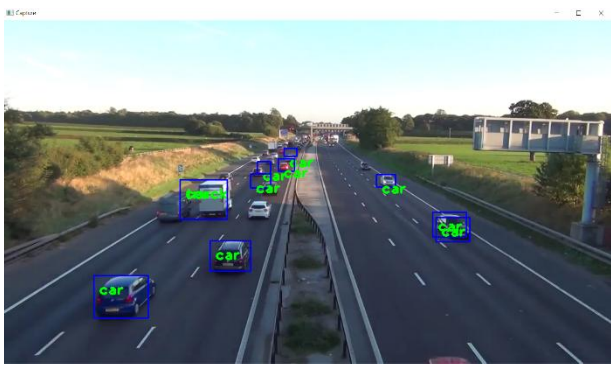


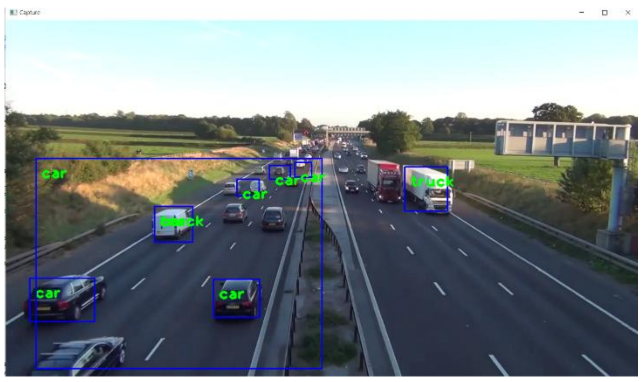

```
In [15]:
# same logic will work for both video and images.
# Type letter q to exist the execution on the image / video canvas

cap=cv2.VideoCapture('Road_traffic_video2.mp4') # for mp4 video feed  capture

#cap=cv2.VideoCapture(0) # for laptop or web camera capture. For external cameras, this 0 should be changed to 1.
if not cap.isOpened():
    cap=cv2.VideoCapture(0)
if not cap.isOpened():
    raise IOError("Cannot Open Video / Web Cam")

font_scale=2
font=cv2.FONT_HERSHEY_PLAIN

while True:
    ret,frame=cap.read()
    ClassIndex,confidence,bbox=model.detect(frame,confThreshold=0.55)
    print(ClassIndex)
    if(len(ClassIndex))!=0:
        for ClassInd,conf,boxes in zip(ClassIndex.flatten(),confidence.flatten(),bbox):
            if(ClassInd <80):
                cv2.rectangle(frame,boxes,(255,0,0),2)
                cv2.putText(frame,classLabels[ClassInd-1],(boxes[0]+10,boxes[1]+40),font,fontScale=font_scale,color=(0,255,0),thi
    cv2.imshow('Capture',frame)
    if cv2.waitKey(2) & 0xFF ==ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Opens a video file or webcam feed for processing.

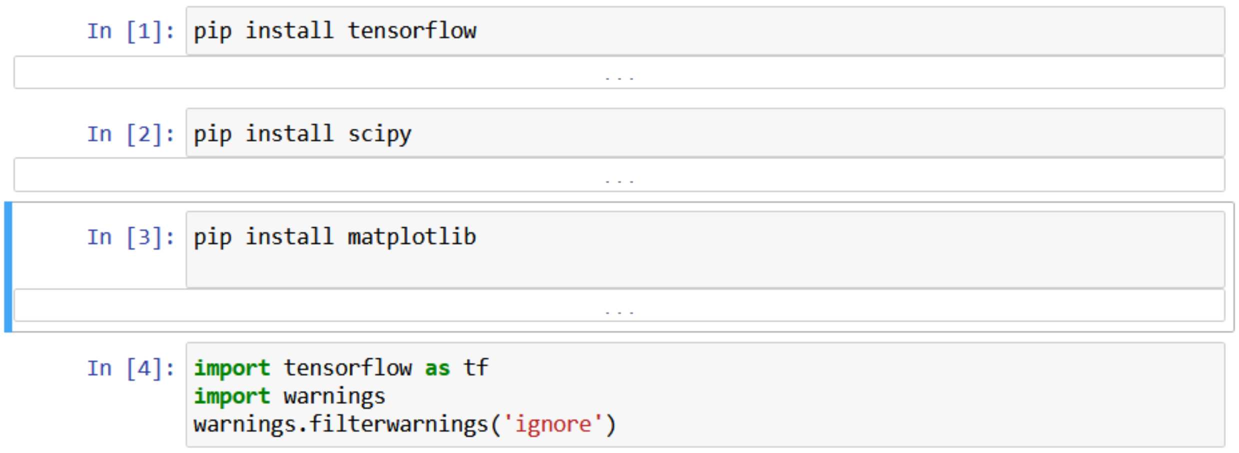
- cv2.VideoCapture(): Opens the video source.
- while True: Continuously processes each frame.
- cap.read(): Reads the current frame.
- model.detect(): Detects objects in the frame.
- print(ClassIndex): Prints detected classes.
- Draws bounding boxes and labels for detected objects in each frame.
- cv2.imshow(): Displays the processed frame.
- cv2.waitKey(2) & 0xFF == ord('q'): Waits for 'q' key press to exit.
- cap.release(): Releases the video capture object.
- cv2.destroyAllWindows(): Closes all OpenCV windows.

Output:



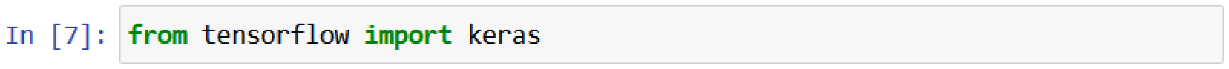


## PNEUMONIA DETECTION

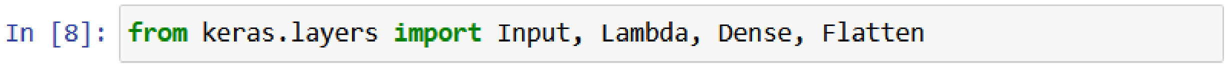


```
In [1]: pip install tensorflow
...
In [2]: pip install scipy
...
In [3]: pip install matplotlib
...
In [4]: import tensorflow as tf
        import warnings
        warnings.filterwarnings('ignore')
```

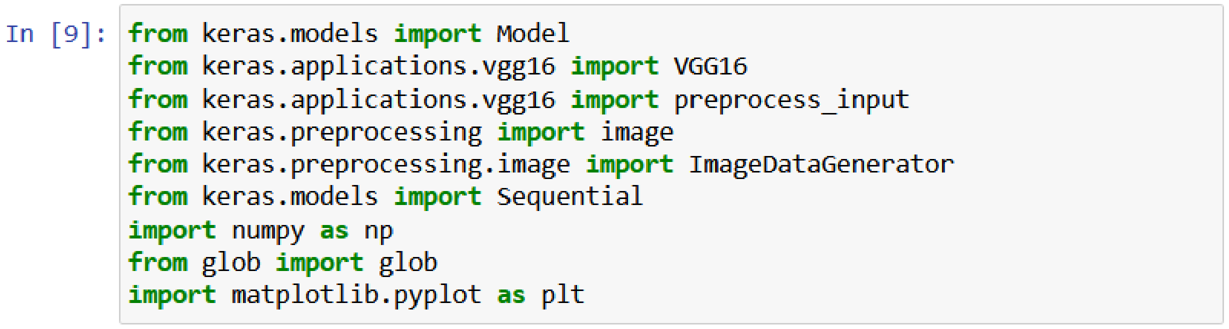
- Installs TensorFlow and SciPy libraries.
- Imports libraries.



```
In [7]: from tensorflow import keras
```

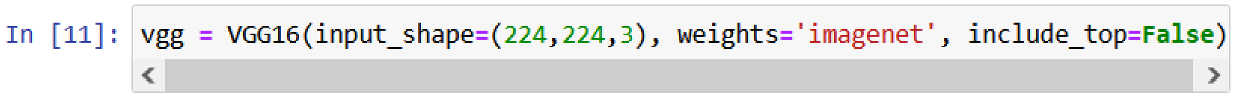


```
In [8]: from keras.layers import Input, Lambda, Dense, Flatten
```



```
In [9]: from keras.models import Model
        from keras.applications.vgg16 import VGG16
        from keras.applications.vgg16 import preprocess_input
        from keras.preprocessing import image
        from keras.preprocessing.image import ImageDataGenerator
        from keras.models import Sequential
        import numpy as np
        from glob import glob
        import matplotlib.pyplot as plt
```

- Imports various modules from TensorFlow and Keras for building the model, processing images, and plotting.
- These modules provide functions to create layers, models, handle images, and visualize data.



```
In [11]: vgg = VGG16(input_shape=(224,224,3), weights='imagenet', include_top=False)
```

- Loads the VGG16 model pre-trained on ImageNet without the top classification layer.
- Reason: Uses the convolutional layers of VGG16 as a feature extractor for our images. We exclude the top layer because we need to create our own custom classifier for pneumonia detection.

```
In [12]: for layer in vgg.layers:
             layer.trainable = False
```

- Freezes the weights of all layers in the VGG16 model.
- Prevents changes to the pre-trained weights during training to retain learned features from the ImageNet dataset.

```
In [13]: folders = glob('chest_xray/train/*')
         x = Flatten()(vgg.output)
```

- Gets the number of folders (classes) in the training directory and flattens the output of the VGG16 model.
- Converts the output from the convolutional layers into a single long vector to be fed into the new classification layer.

```
In [14]: # use the derived flattened tensor and re-use it to create a custom and tra
         # output we need as penumonia positive / negative. Hence it`s binary classe
         prediction = Dense(len(folders), activation='softmax')(x)
         # create a model object
         model = Model(inputs=vgg.input, outputs=prediction)
         # view the structure of the model
         model.summary()
```

- Adds a fully connected (dense) layer with a softmax activation function.
- Creates a new output layer to predict the probability of each class (Pneumonia or Normal).
- Creates a new model combining VGG16 and the custom output layer, then prints a summary of the model structure.

```
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2)                 50178

=================================================================
Total params: 14764866 (56.32 MB)
Trainable params: 50178 (196.01 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
In [15]: model.compile(
           loss='categorical_crossentropy',
           optimizer='adam',
           metrics=['accuracy']
         )
```

- Configures the model with a loss function, optimizer, and metric.
- categorical_crossentropy is suitable for multi-class classification, adam optimizer helps in training, and accuracy measures the model's performance.

```
In [16]: from keras.preprocessing.image import ImageDataGenerator
```

- import Imagedata generator

```
In [17]: train_datagen = ImageDataGenerator(rescale = 1./255,
                                            shear_range = 0.2,
                                            zoom_range = 0.2,
                                            horizontal_flip = True)

         test_datagen = ImageDataGenerator(rescale = 1./255)

         training_set = train_datagen.flow_from_directory('chest_xray/train',
                                                          target_size = (224, 224),
                                                          batch_size = 10,
                                                          class_mode = 'categorical')



         test_set = test_datagen.flow_from_directory('chest_xray/test',
                                                     target_size = (224, 224),
                                                     batch_size = 10,
                                                     class_mode = 'categorical')

         Found 5216 images belonging to 2 classes.
         Found 624 images belonging to 2 classes.
```

Sets up image augmentation for the training set.

- Reason: Augments the training images by scaling, shearing, zooming, and flipping to improve model robustness.

Sets up image preprocessing for the test set by scaling pixel values.

- Normalizes the images for consistent input to the model.

Loads and augments images from the training directory.

- Creates batches of augmented images for training, with each image resized to 224x224 pixels.

Loads and preprocesses images from the test directory.

- Creates batches of normalized images for validation/testing, also resized to 224x224 pixels.

```
In [21]: r = model.fit_generator(
           training_set,
           validation_data=test_set,
           epochs=1,
           steps_per_epoch=len(training_set),
           validation_steps=len(test_set)
         )

         522/522 [==============================] - 522s 1s/step - loss: 0.1853 - accura
         cy: 0.9396 - val_loss: 0.6350 - val_accuracy: 0.8109
```

- Trains the model on the training data and evaluates it on the test data. Fits the model to the training data while validating its performance on the test set. This uses one epoch for demonstration.

```
In [22]: model.save('chest_xray.h5') # save the trained modle

In [23]: from keras.models import load_model

In [24]: from keras.preprocessing import image

In [25]: from keras.applications.vgg16 import preprocess_input # import the model`s skele

In [26]: model=load_model('chest_xray.h5')
```

Trains the model on the training data and evaluates it on the test data.

- Fits the model to the training data while validating its performance on the test set. This uses one epoch for demonstration.

Loads the saved model.

- To perform predictions using the pre-trained model.

```
In [27]: img=tf.keras.utils.load_img('chest_xray/chest_xray/test/PNEUMONIA/person1_virus_

         x=tf.keras.preprocessing.image.img_to_array(img) # image as a numpy array

         x=np.expand_dims(x, axis=0)

         img_data=preprocess_input(x) # organize for the prediction
```

Loads and preprocesses an image for prediction.

- load_img(): Loads the image.
- img_to_array(): Converts the image to a numpy array.
- expand_dims(): Adds a batch dimension.

Preprocesses the image data to match the model's input format.

- Applies necessary transformations for VGG16 compatibility.

```
In [28]: classes=model.predict(img_data)

         1/1 [==============================] - 0s 448ms/step
```

- Predicts the class of the image.
- Uses the model to classify the input image.

```
In [29]: result=int(classes[0][0])
```

- Converts the predicted class from a probability to an integer.
- Determines if the prediction is Pneumonia or Normal.

```
In [30]: if result== 0:
             print("Person is Affected By PNEUMONIA")
         else:
             print("Result is Normal")

         Person is Affected By PNEUMONIA
```

- Interprets the prediction and prints the result. Provides a human-readable output based on the model's prediction.

## GUN DETECTION

```
In [1]: pip install Pillow
```

```
In [1]: pip install SciPy
```

```
In [2]: pip install tensorflow
```

- Installs the Pillow library for image handling.
- Pillow is used for image processing tasks, such as opening, manipulating, and saving image files.

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
import os
from PIL import Image
```

- Imports TensorFlow for building and training neural networks, utilities for image processing, and Pillow for image handling.

```python
# Get the absolute path to the dataset
dataset_path = r'C:\Users\ASUS\Desktop\hhh\gun_detect\gunenv\dataset'
absolute_path = os.path.abspath(dataset_path)
```

- Sets up the absolute path to the dataset directory.

```python
# Load the pre-trained VGG16 model
base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
```

- Loads a VGG16 model pre-trained on ImageNet without the final classification layer and freezes its weights.
- Uses a well-established model to leverage its feature extraction capabilities. Freezing the weights prevents them from being updated during training.

```python
# Create a custom model for fine-tuning
model = tf.keras.Sequential([
    base_model,
    # reduce the complexity of the prev tensor derived from the input
    # by averaging the pixel distributions. So learning will be easy
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation='relu'),
    # randomly deactivate nerouns to prevent overfitting on features by
    # some neurons. 0.5 represent the % of neurons impacted by the dropout
    #feature
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Binary classification (gun or not)
])
```

Builds a new model by adding layers to the pre-trained base:

- GlobalAveragePooling2D: Reduces the spatial dimensions of the features by averaging.
- Dense(256): Adds a fully connected layer with 256 neurons and ReLU activation.
- Dropout(0.5): Randomly deactivates 50% of neurons to prevent overfitting.
- Dense(1): Adds a final layer for binary classification (sigmoid activation for binary output).
- Customizes the model to classify images as containing a gun or not.

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Data preparation with data augmentation. Larger batch sizes are good to
# make the models to be more mature. However, it could lead to stuck due to
# memory shortages. Hence, trial and error expertimentations are required.
batch_size = 32
# augment the images on the go as per the specs given during the training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Compiles the model with the Adam optimizer, binary cross-entropy loss, and accuracy metric.

- Configures the training process with appropriate settings for binary classification.

Creates an ImageDataGenerator for augmenting images during training:

- rescale: Normalizes pixel values to [0, 1].
- validation_split: Sets aside 20% of the data for validation.
- rotation_range, width_shift_range, etc.: Applies random transformations to the images.

```
# configured pipeline of datagenerator is directly linked to the folder
train_generator = train_datagen.flow_from_directory(
    absolute_path,
    target_size=(224, 224),
    batch_size=batch_size,
    # augment the images considering correct and wrong representations.
    # this allow the model to learn with a comparative assesment.
    class_mode='binary',
    # can be either training or validation. In validation mode, adjust the
    # images to support the inferencing purposes.
    subset='training'
)
```

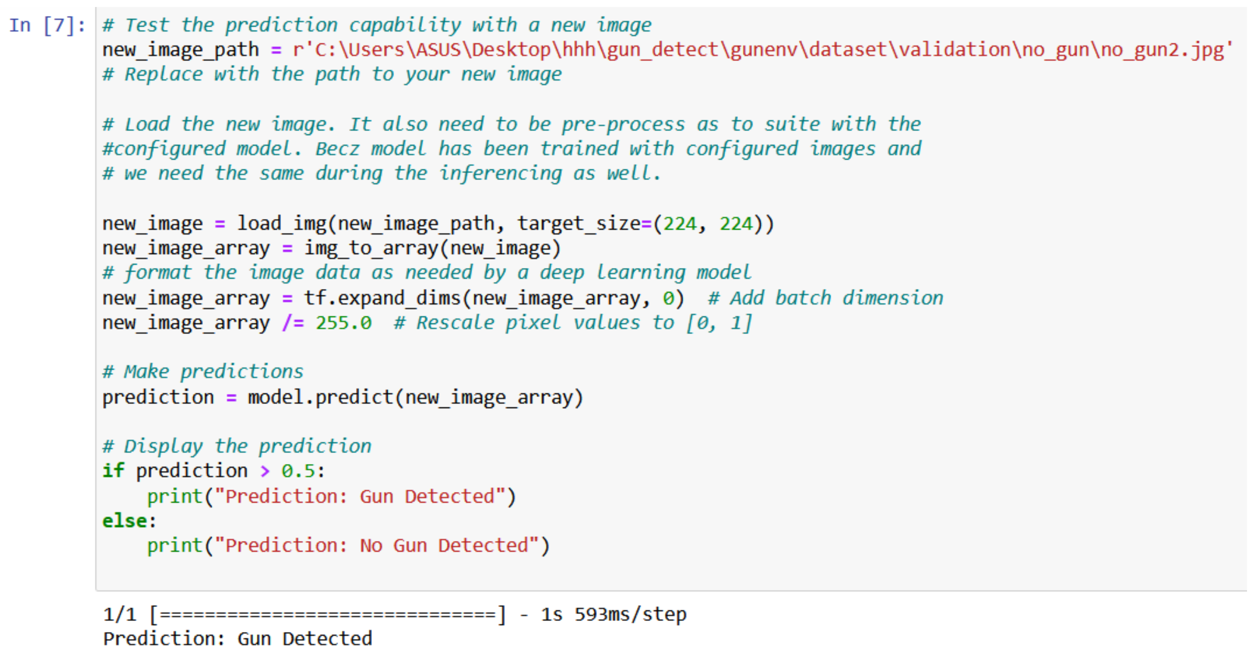
```
# arrangement of the validation dataset.
validation_generator = train_datagen.flow_from_directory(
    absolute_path,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

Connects the ImageDataGenerator to the dataset directory for training and validation subsets.

- Reads images from directories, applies augmentations, and prepares batches for training and validation.

```
# Train the model with augmented data
model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator
)

# Save the model
model.save('custom_gun_model.h5')
```

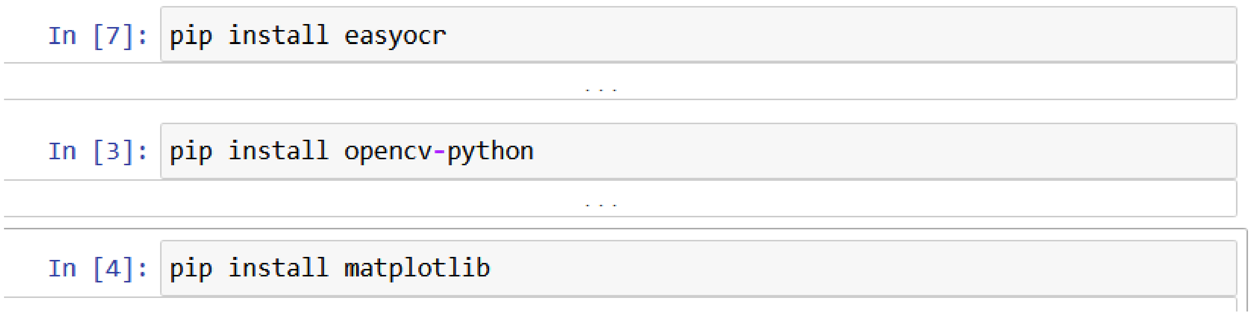
- Trains the model using the training generator and validates it using the validation generator for 50 epochs.
- Fits the model to the training data and evaluates it on the validation data to check its performance.
- Saves the trained model to a file. Stores the model for later use or deployment.



```
In [7]: # Test the prediction capability with a new image
        new_image_path = r'C:\Users\ASUS\Desktop\hhh\gun_detect\gunenv\dataset\validation\no_gun\no_gun2.jpg'
        # Replace with the path to your new image

        # Load the new image. It also need to be pre-process as to suite with the
        #configured model. Becz model has been trained with configured images and
        # we need the same during the inferencing as well.

        new_image = load_img(new_image_path, target_size=(224, 224))
        new_image_array = img_to_array(new_image)
        # format the image data as needed by a deep learning model
        new_image_array = tf.expand_dims(new_image_array, 0)  # Add batch dimension
        new_image_array /= 255.0  # Rescale pixel values to [0, 1]

        # Make predictions
        prediction = model.predict(new_image_array)

        # Display the prediction
        if prediction > 0.5:
            print("Prediction: Gun Detected")
        else:
            print("Prediction: No Gun Detected")

        1/1 [==============================] - 1s 593ms/step
        Prediction: Gun Detected
```

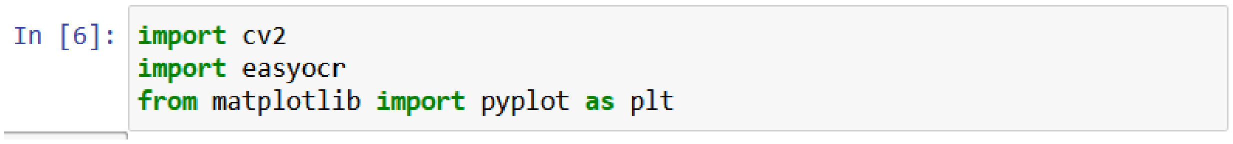
- Loads and preprocesses a new image for testing, makes a prediction, and displays the result. Evaluates the model’s ability to classify new, unseen images correctly.

## NUMBER PLATE DETECTION



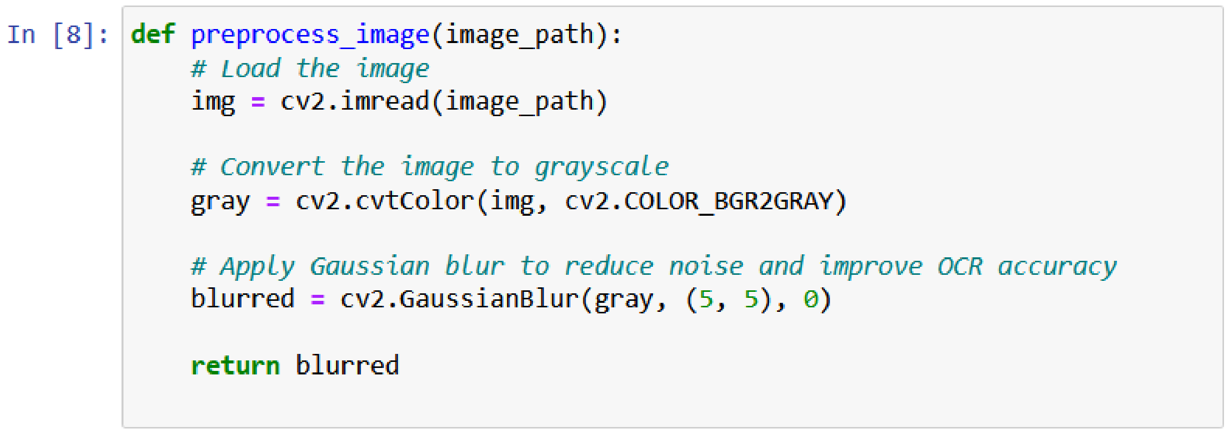
```
In [7]: pip install easyocr
...
In [3]: pip install opencv-python
...
In [4]: pip install matplotlib
```

- Installs the EasyOCR library.
- EasyOCR is a library used for Optical Character Recognition (OCR), allowing us to read text from images.



```
In [6]: import cv2
        import easyocr
        from matplotlib import pyplot as plt
```

- Imports OpenCV for image processing, EasyOCR for text recognition, and Matplotlib for displaying images.
- OpenCV is used for image manipulation, EasyOCR for extracting text from images, and Matplotlib for visualization.



```
In [8]: def preprocess_image(image_path):
            # Load the image
            img = cv2.imread(image_path)

            # Convert the image to grayscale
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            # Apply Gaussian blur to reduce noise and improve OCR accuracy
            blurred = cv2.GaussianBlur(gray, (5, 5), 0)

            return blurred
```

This function takes an image path, processes the image, and returns a blurred grayscale image.

- Load the image: Reads the image from the given path using OpenCV.

- Convert to grayscale: Converts the image to grayscale to simplify processing (OCR works better on grayscale images).
- Apply Gaussian blur: Smooths the image to reduce noise, helping improve the accuracy of text detection.

```
In [9]: def perform_ocr(image, reader):
            # Perform OCR on the preprocessed image
            result = reader.readtext(image)

            # Extract and return the recognized text
            recognized_text = ' '.join([entry[1] for entry in result])
            return recognized_text
```

This function takes a preprocessed image and an EasyOCR reader object, performs OCR, and returns the recognized text.

- Perform OCR: Uses EasyOCR to read text from the image.
- Extract text: Collects the text from the OCR results and joins them into a single string.

```
In [10]: def display_image_with_results(image, recognized_text):
             # Display the image with recognized text
             plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
             plt.title(f"Recognized Text: {recognized_text}")
             plt.axis('off')
             plt.show()
```

This function displays the original image and overlays the recognized text on it.

- Convert color: Changes the color format from BGR (used by OpenCV) to RGB (used by Matplotlib) for correct color display.
- Display image: Shows the image with the title displaying the recognized text.

```
In [11]: def number_plate_recognition(image_path):
             # Preprocess the image
             preprocessed_image = preprocess_image(image_path)

             # Create an OCR reader using the 'en' language for English text
             reader = easyocr.Reader(['en'])

             # Perform OCR on the preprocessed image
             recognized_text = perform_ocr(preprocessed_image, reader)

             # Display the results
             display_image_with_results(cv2.imread(image_path), recognized_text)
```

This function puts together the entire number plate recognition process.

- Preprocess the image: Prepares the image for OCR by converting to grayscale and blurring.
- Create OCR reader: Initializes an EasyOCR reader for English text recognition.
- Perform OCR: Detects text in the preprocessed image.
- Display results: Shows the original image along with the recognized text.

```
In [12]: # Example usage
         image_path = 'plate.jpg'
         number_plate_recognition(image_path)
```

- Sets the path to the image file and calls the number_plate_recognition function with this path.
- Provides an example to test the number plate recognition system with a sample image.