Object Detection Image Model

Arijeet Goswami

Student, University of Engineering and Management Kolkata

ABSTRACT
Object detection in images is a critical task in computer vision with wide-ranging applications in autonomous driving, robotics, healthcare, surveillance, and more. Leveraging Artificial Intelligence (AI) and Machine Learning (ML) techniques, significant advancements have been made in accurately identifying and localizing objects within complex scene.
The integration of AI and ML allows these models to learn discriminative features through large annotated datasets, reducing the need for manual feature engineering. Transfer learning and fine- tuning of pre-trained models further improve efficiency and accuracy, especially in domain- specific applications where data is scarce. Additionally, we discuss real-time detection techniques that balance accuracy with speed, making them suitable for time-sensitive applications like real- time video analysis. Challenges such as false positives, overlapping object detection, and the trade-off between precision and recall are examined, along with emerging techniques like transformer-based models and the impact of AI on future object detection tasks.

1. INTRODUCTION

1.1 About The Project:

Object detection is a pivotal task in computer vision, involving the identification and localization of objects within images. With the advancement of Artificial Intelligence (AI) and Machine Learning (ML), particularly deep learning techniques, object detection models have achieved remarkable accuracy and efficiency.
AI-powered object detection is widely applied in fields like autonomous vehicles, surveillance, healthcare, and augmented reality. The integration of ML allows these models to continually improve by learning from large datasets, making them highly adaptable to new scenarios. Despite their successes, challenges such as detecting objects in cluttered or dynamic environments, improving real-time performance, and handling small or overlapping objects remain areas of ongoing research.

1.2 Objectives And Deliverables:

Object detection is a fundamental challenge in computer vision that involves identifying and localizing objects within an image, often with the aim of recognizing multiple objects simultaneously. With the rise of Artificial Intelligence (AI) and Machine Learning (ML), object detection techniques have seen significant advancements, transforming applications across industries such as autonomous driving, security surveillance, medical imaging, robotics, and retail.

2. METHEDOLOGY

2.1 Flow Of Project:

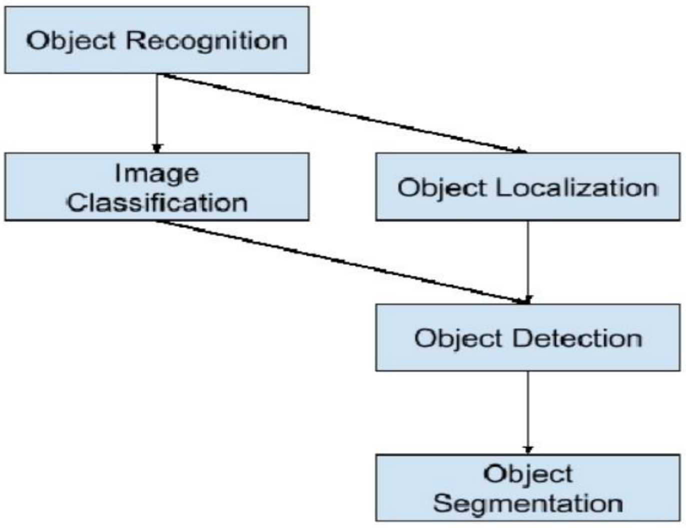


Figure 1: IMAGE-RECOGNITION-Flow-Chart (Source: riset.guru)

The diagram represents a hierarchical process of object recognition in computer vision, breaking it down into several stages:

1. **Object Recognition**: The broader task of identifying objects in an image.
2. **Image Classification**: A sub-task where an image is assigned a label based on the object it contains.
3. **Object Localization**: Involves determining the location of an object within the image, often by drawing bounding boxes.
4. **Object Detection**: Combines object classification and localization to detect multiple objects and their positions in an image.
5. **Object Segmentation**: The most detailed step, which involves partitioning the image into segments where each pixel is assigned to an object, allowing for precise boundaries and distinctions.

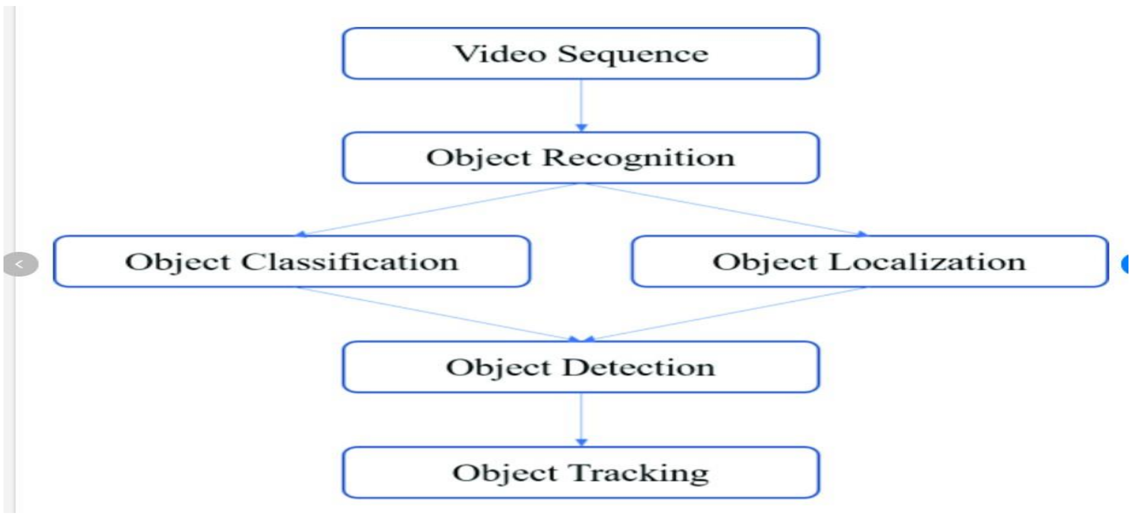


Figure 1: VIDEO-RECOGNITION-Flow-Chart (Source: ResearchGate)

The diagram illustrates a sequential process for object recognition in video sequences, breaking down the steps as follows:

1. **Video Sequence**: The input to the process is a sequence of video frames.
2. **Object Recognition**: Identifies and interprets objects within the video frames.
3. **Object Classification and Localization**: These two parallel tasks involve classifying detected objects into categories (Object Classification) and determining their precise location in the frames (Object Localization).
4. **Object Detection**: Combines classification and localization to detect multiple objects and their positions within the video frames.
5. **Object Tracking**: Once objects are detected, they are tracked across the video sequence, ensuring consistent identification as they move through frames.

This flow emphasizes how video-based object detection extends from recognition to tracking, facilitating continuous monitoring of objects in motion.

2.2 Language And Platform

Python and Google Colab

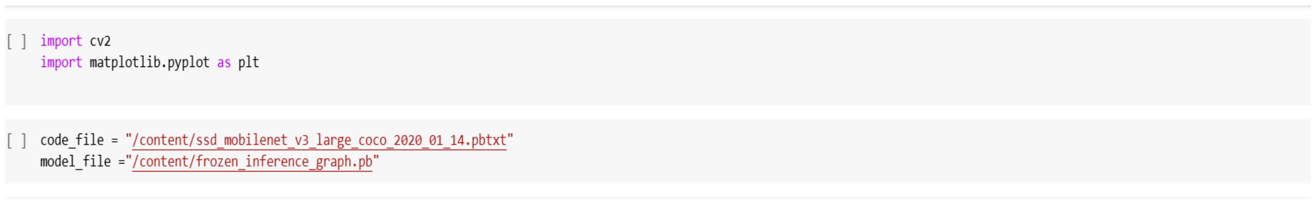
Python is a powerful, high-level programming language known for its simplicity, readability, and extensive libraries, making it a popular choice for tasks ranging from web development to data science and machine learning. Its versatility and supportive community have contributed to the rise of libraries such as TensorFlow, PyTorch, and Scikit-learn, which are widely used in artificial intelligence (AI) and machine learning (ML) projects.

Google Colab, or Google Colaboratory, is a cloud-based platform that allows users to write and execute Python code in Jupyter notebooks. It is particularly useful for data scientists and ML practitioners, as it provides free access to computational resources such as GPUs (Graphical Processing Units) and TPUs (Tensor Processing Units). Colab is equipped with pre-installed libraries, making it easier to start coding without the need for local setup.

Additionally, its integration with Google Drive allows for easy storage and collaboration, making it ideal for research, experimentation, and team-based projects in AI and ML. Together, Python and Google Colab provide a powerful, accessible environment for coding, experimentation, and collaboration in machine learning, data analysis, and other computational tasks.

3. IMPLEMENTATION:

3.1 Model Creation



```
[ ] import cv2
    import matplotlib.pyplot as plt
```

```
[ ] code_file = "/content/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
    model_file ="/content/frozen_inference_graph.pb"
```

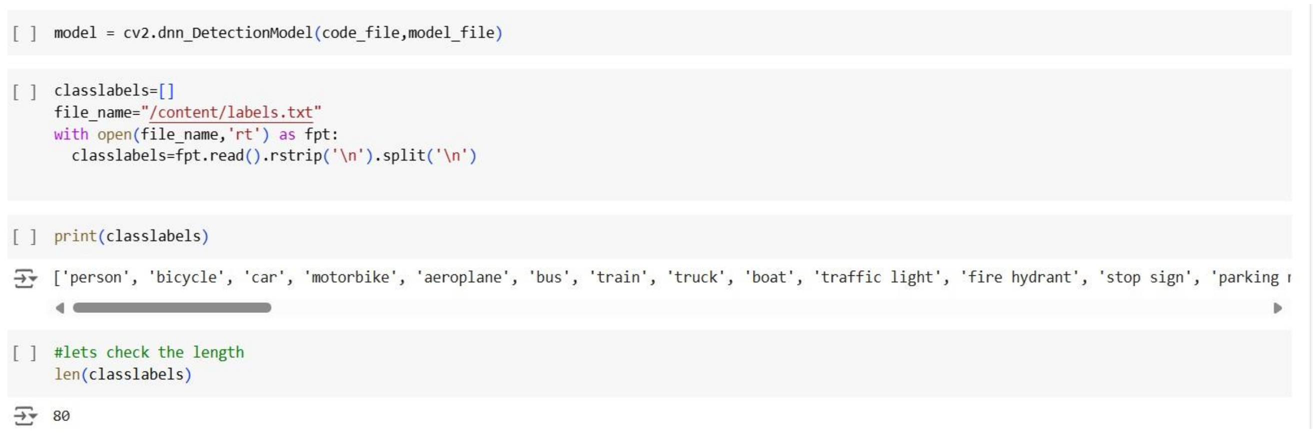
Imports:

- cv2: This is part of the OpenCV library, commonly used for computer vision tasks such as image processing, object detection, and image classification.
- matplotlib.pyplot as plt: This is used to visualize images, graphs, or other data plots.

Model and Configuration Files:

- code_file: Refers to the location of a .pbtxt file. This is likely a configuration file that defines the architecture of the object detection model.

- model_file: Refers to a .pb file. This is the frozen inference graph, which contains the pre- trained weights for a model, most likely the SSD MobileNet V3 model, trained on the COCO dataset.



```
[ ] model = cv2.dnn_DetectionModel(code_file,model_file)

[ ] classlabels=[]
    file_name="/content/labels.txt"
    with open(file_name,'rt') as fpt:
      classlabels=fpt.read().rstrip('\n').split('\n')

[ ] print(classlabels)

    ['person', 'bicycle', 'car', 'motorbike', 'aeroplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking

[ ] #lets check the length
    len(classlabels)

    80
```

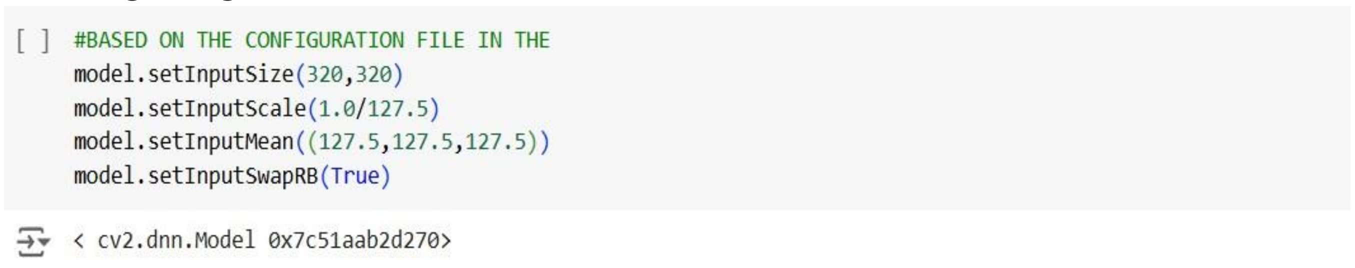
Loading the Model:

- The line model = cv2.dnn_DetectionModel(code_file, model_file) loads the object detection model using OpenCV's DNN (Deep Neural Network) module. The code_file and model_file parameters refer to the configuration and model weights files, respectively, as seen earlier.

Loading Class Labels:

- A list classLabels is initialized as empty.
- The code then opens a file named labels.txt located at "/content/labels.txt", which contains the labels of the object categories (like 'person', 'bicycle', 'car', etc.).
- The file is read, and its contents are split by newline characters to create a list of labels that correspond to the categories that the model can detect.

3.2 Setting Configuration For The Model



```
[ ] #BASED ON THE CONFIGURATION FILE IN THE
    model.setInputSize(320,320)
    model.setInputScale(1.0/127.5)
    model.setInputMean((127.5,127.5,127.5))
    model.setInputSwapRB(True)

    < cv2.dnn.Model 0x7c51aab2d270>
```

The model is configured with the following parameters:

- setInputSize(320, 320): Sets the input size of the model to 320x320 pixels.
- setInputScale(1.0/127.5): Normalizes the input by scaling the pixel values by dividing by 127.5.
- setInputMean((127.5, 127.5, 127.5)): Sets the mean values for each channel (RGB) to 127.5, which is used for mean subtraction.
- setInputSwapRB(True): Swaps the Red and Blue channels, which is necessary for converting BGR to RGB format.

The model object shown is of type cv2.dnn.Model.

This snippet is using OpenCV's **Deep Neural Network (DNN)** module to configure a pre- trained deep learning model for inference, likely for tasks such as object detection, image classification, or image segmentation. Here's a more detailed breakdown of each part:

1. **Input Size**:

model.setInputSize(320, 320): The model's input size is set to 320x320 pixels. This resizes the input image to match the expected input dimensions of the model. Neural networks often require a fixed-size input, and resizing ensures that the image dimensions conform to what the model was trained on.

2. **Input Scale**:

model.setInputScale(1.0/127.5): This scales the input image's pixel values by dividing them by 127.5. The typical pixel range for an image is between 0 and 255. Dividing by 127.5 shifts the range to approximately [-1, 1]. This normalization improves the model's performance by ensuring that the input data has a consistent scale, which is important for network stability during inference.

3. **Input Mean**:

model.setInputMean((127.5, 127.5, 127.5)): This subtracts the mean values for each channel (RGB). By subtracting 127.5 from each channel, the pixel values are further centered around 0 (i.e., mean-centered). This step ensures that the input image has values more similar to the training distribution, as many models are trained on normalized data.

4. **Swapping Color Channels**:

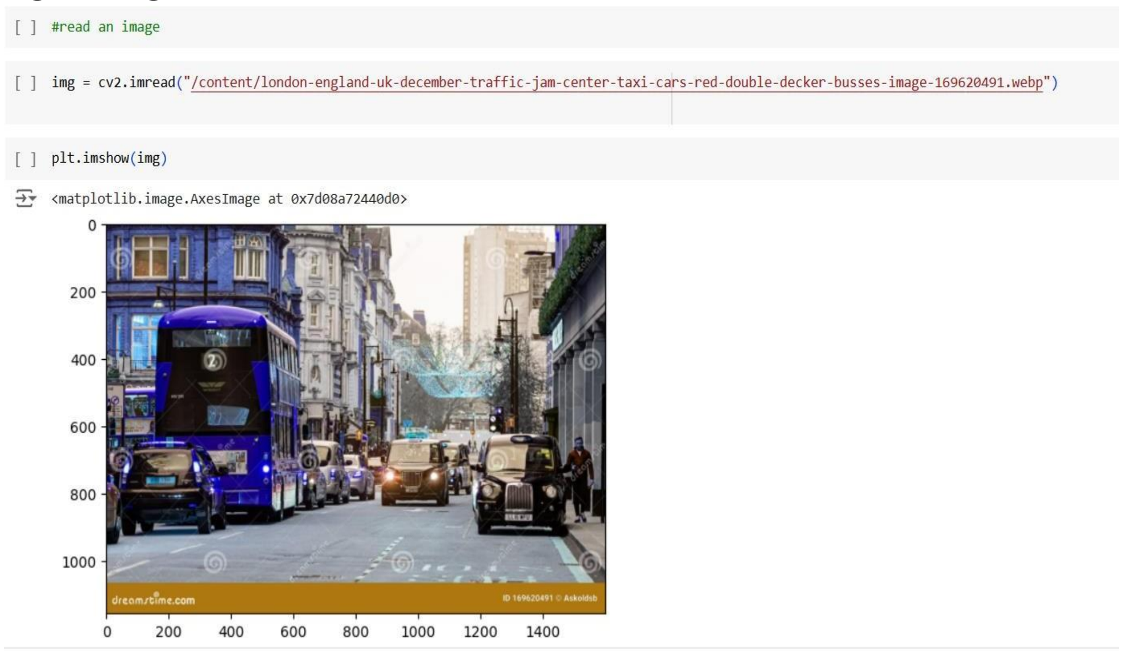
model.setInputSwapRB(True): This swaps the Red and Blue channels in the image. OpenCV loads images in BGR format by default, but many deep learning models, especially those trained with libraries like TensorFlow or PyTorch, expect images in RGB format. This opkeration ensures the image channels are aligned with the model's expectations.

Additional Context:

- These transformations (resize, normalization, mean subtraction, and channel swapping) are typical preprocessing steps required before passing an image through a deep learning model.
- The cv2.dnn.Model is the OpenCV class that handles loading and running a pre- trained neural network model for inference. This could be a model trained using popular frameworks like TensorFlow, Caffe, or ONNX.

These settings are crucial for ensuring the input data matches the training data distribution, which improves accuracy and reliability when making predictions or inferences with the model.

3.3 Loading Of Image Insde, The Model



```
[ ] #read an image

[ ] img = cv2.imread("/content/london-england-uk-december-traffic-jam-center-taxi-cars-red-double-decker-busses-image-169620491.webp")

[ ] plt.imshow(img)

<matplotlib.image.AxesImage at 0x7d08a72440d0>
```

Python code snippet that loads and displays an image using OpenCV and Matplotlib. Here is a summary of the process:

1. **Reading an Image**:

- The code uses cv2.imread() to load an image from a file path. The path points to an image showing a busy street in London, featuring double-decker buses, taxis, and other vehicles.
- The image format is .webp, a compressed image format commonly used for web images.

2. **Displaying the Image**:

- The loaded image is displayed using plt.imshow(img), a function from the Matplotlib library that renders the image in the current plotting window.
- The displayed image showcases a London street scene, likely in December, as described by the file name, with notable vehicles like taxis and double-decker buses.

The output shows a plot of the image with the axes (x and y) representing the pixel dimensions. The file path hints at the image being related to London traffic and iconic vehicles like red double- decker buses.

```
[ ]  #convert the image form bgr to rgb
     plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7d08a4568be0>
```

code snippet that performs color conversion and displays the result using OpenCV and Matplotlib:

1. **Color Conversion**:

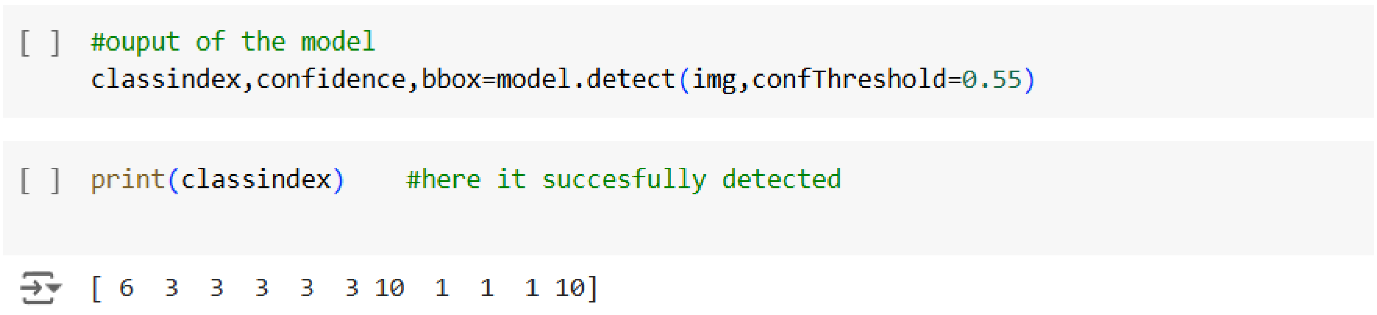
- The code uses cv2.cvtColor(img, cv2.COLOR_BGR2RGB) to convert the image from BGR (Blue-Green-Red) to RGB (Red-Green-Blue) color format.
- OpenCV loads images in BGR format by default, but many plotting libraries like Matplotlib expect images in RGB format. This conversion ensures proper color representation when displaying the image.

2. **Displaying the Image**:

- The converted image is displayed using plt.imshow(), which is part of the Matplotlib library.
- The displayed image shows a typical London street scene with iconic vehicles such as red double-decker buses and black cabs in traffic, possibly during winter, as hinted by the festive lights above the street.

The color conversion ensures that the image appears correctly (with the correct colors) in the Matplotlib plot.

3.4 Detection Of Object By The Model



```
[ ]  #ouput of the model
     classindex,confidence,bbox=model.detect(img,confThreshold=0.55)
```

```
[ ]  print(classindex)    #here it succesfully detected
```

```
[ 6  3  3  3  3  3 10  1  1  1 10]
```

1. **Detection Function**:

- The function model.detect(img, confThreshold=0.55) is executed to perform object detection on the provided image (img). The confidence threshold is set to 0.55, meaning only objects detected with a confidence score of at least 55% will be considered valid.

2. **Outputs**:

- The function returns three values:
  - **classindex**: The class labels of the detected objects, represented as indices.
  - **confidence**: The confidence scores corresponding to each detection, indicating how certain the model is about the class of each object.
  - **bbox**: The bounding boxes for each detected object, describing their position and size in the image.

3. **classindex Output**:

- When printing classindex, the output is [6, 3, 3, 3, 3, 10, 1, 1, 1, 10]. This array indicates the detected class labels in the image.
  - Class **6** appears once.
  - Class **3** is detected four times, suggesting multiple objects of this class in the image.
  - Class **10** appears twice.
  - Class **1** is detected three times.

4. **Success**:

- The model has successfully detected multiple instances of various classes. This suggests that the image contains objects from several categories, with class 3 being the most frequent.

5. **Interpretation**:

- These class indices likely map to specific categories (e.g., person, car, dog, etc.) based on the model's training dataset. For further analysis, you'd need to reference the model's class labels to understand what each index (e.g., 3, 6, 10, 1) corresponds to in real-world terms.

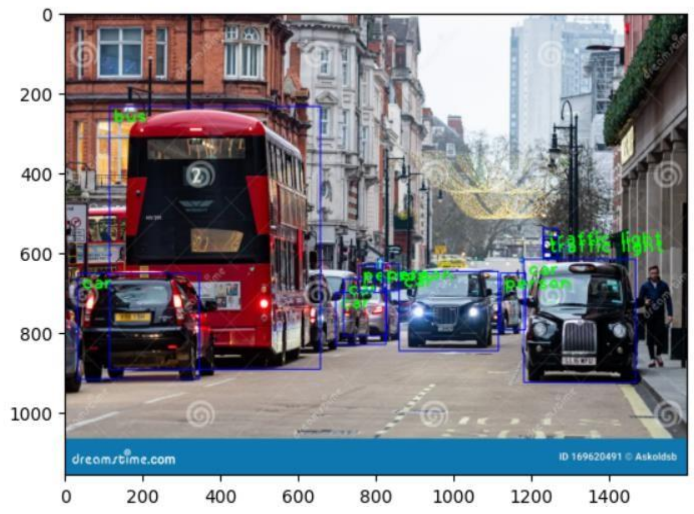
This setup indicates the model is functioning correctly, as it has detected multiple objects with varying class indices in the image, demonstrating its ability to process and analyze the input data effectively.

3.5 Providing Description For The Object Detected By Model For The Image

```
[ ] #lets define the text boxes
    font_scale=3
    font=cv2.FONT_HERSHEY_PLAIN
    for classindex,confidence,bbox in zip(classindex.flatten(),confidence.flatten(),bbox):
      cv2.rectangle(img,bbox,(255,0,0),2)
      cv2.putText(img,classlabels[classindex-1],(bbox[0]+10,bbox[1]+40),font,fontScale=font_scale,color=(0,255,0),thickness=3)
```

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

```
<matplotlib.image.AxesImage at 0x7d08a3b1a770>
```



1. **Code Overview:**

- The code uses OpenCV functions like cv2.rectangle() and cv2.putText() to draw bounding boxes around detected objects and label them with their respective class names.
- The bounding boxes are drawn in blue, and the labels (class names) are displayed in green text at the top-left corner of each bounding box.
- The font used is FONT_HERSHEY_PLAIN, with a font scale of 3, and the text has a thickness of 3.

2. **Visualization**:

- The image shows a busy street scene with several objects detected and highlighted by the model.
- Detected objects include:
   1. **Bus**: A red double-decker bus, which is detected and labeled in the middle of the image.
   2. **Cars**: Several cars on the street are also detected and labeled.
   3. **Traffic Light**: A traffic light is detected in the upper-right area of the image.
   4. **Person**: A person is detected and labeled as well.

3. **Bounding Boxes**:

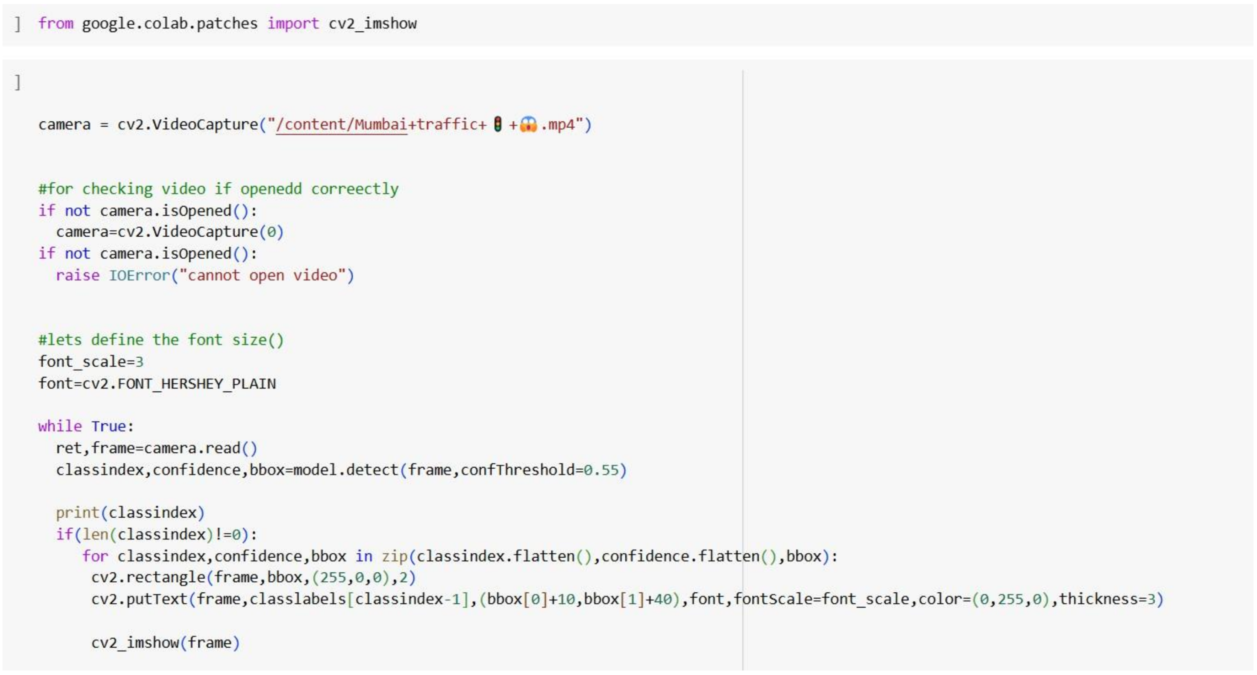
- The bounding boxes drawn around each detected object help to visually differentiate and localize the objects in the scene.
- The detected objects are clearly labeled with their corresponding class names such as "Bus," "Car," "Person," and "Traffic Light."

4. **Display**:

- The processed image is displayed using matplotlib (plt.imshow()), converting the BGR image (OpenCV default color space) to RGB for accurate display in matplotlib.

This visualization confirms the success of the object detection model in identifying and labeling multiple objects in a real-world urban scene.

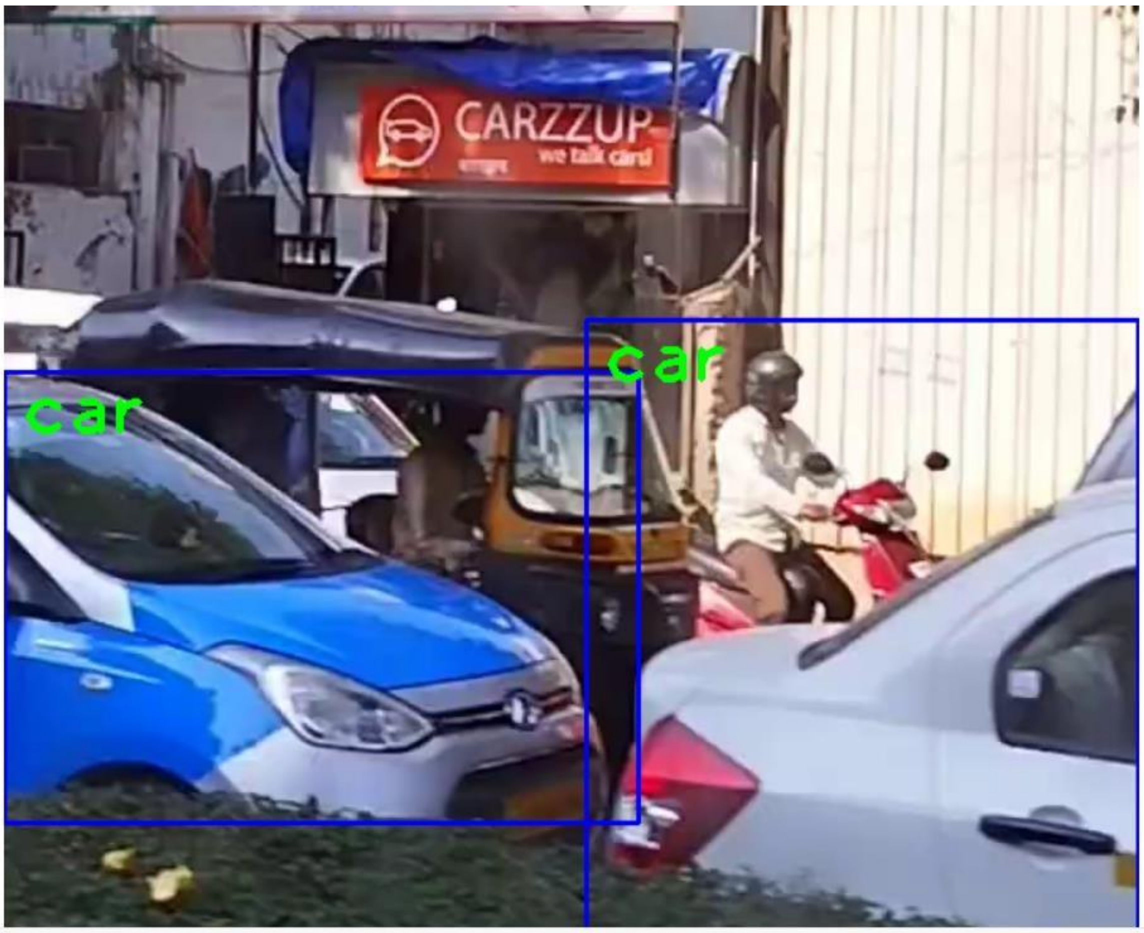
3.6 Object Detection By Video:



```
] from google.colab.patches import cv2_imshow
```

```
]

camera = cv2.VideoCapture("/content/Mumbai+traffic+🚦+😱.mp4")


#for checking video if openedd correectly
if not camera.isOpened():
  camera=cv2.VideoCapture(0)
if not camera.isOpened():
  raise IOError("cannot open video")


#lets define the font size()
font_scale=3
font=cv2.FONT_HERSHEY_PLAIN

while True:
  ret,frame=camera.read()
  classindex,confidence,bbox=model.detect(frame,confThreshold=0.55)

  print(classindex)
  if(len(classindex)!=0):
     for classindex,confidence,bbox in zip(classindex.flatten(),confidence.flatten(),bbox):
      cv2.rectangle(frame,bbox,(255,0,0),2)
      cv2.putText(frame,classlabels[classindex-1],(bbox[0]+10,bbox[1]+40),font,fontScale=font_scale,color=(0,255,0),thickness=3)

      cv2_imshow(frame)
```

1. **Video Input:**
   - The cv2.VideoCapture() function is used to open a video file (in this case, "Mumbai traffic.mp4") or alternatively a webcam feed (cv2.VideoCapture(0)).
   - If the video fails to open, an IOError is raised.
2. **Font Settings:**
   - Font size and type are defined using font_scale=3 and cv2.FONT_HERSHEY_PLAIN to display text on the video frames.
3. **Object Detection Loop:**
   - The code enters an infinite loop (while True), where it continuously captures frames from the video.
   - The detection model (model.detect()) is used to detect objects in the frame. It returns the class index (object type), confidence score, and bounding box (bbox) coordinates for each detected object.
4. **Processing Detections:**
   - For each detected object (if classindex is not empty), the following steps occur:
   - ▪ A rectangle is drawn around the detected object using cv2.rectangle(), with color (255, 0, 0) (red) and a thickness of 2.
   - ▪ The detected object's class label (from classlabels[classindex-1]) is displayed above the bounding box using cv2.putText(), along with the confidence score.
5. **Frame Display:**
   - The updated frame, with detection annotations, is shown using cv2_imshow() (a Google Colab-specific function to display images or video frames within the notebook environment).
6. **Confidence Threshold:**
   - The confThreshold=0.55 ensures that only objects with a detection confidence greater than 55% are considered valid for displaying.



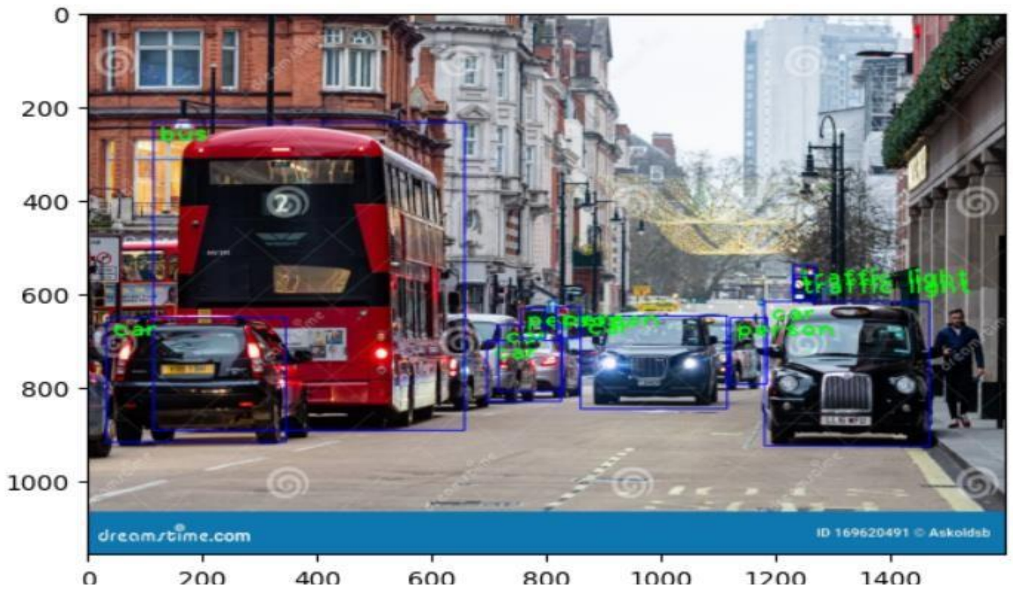
OBJECT-DETECTION OVER THE VIDEO

4. SAMPLE SCREENSHOTS AND OBSERVATIONS:

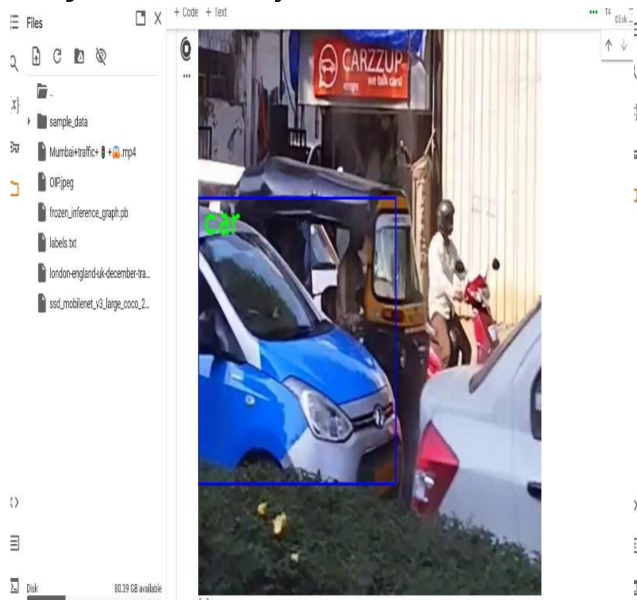
4.1 Detection of Object by the Model:

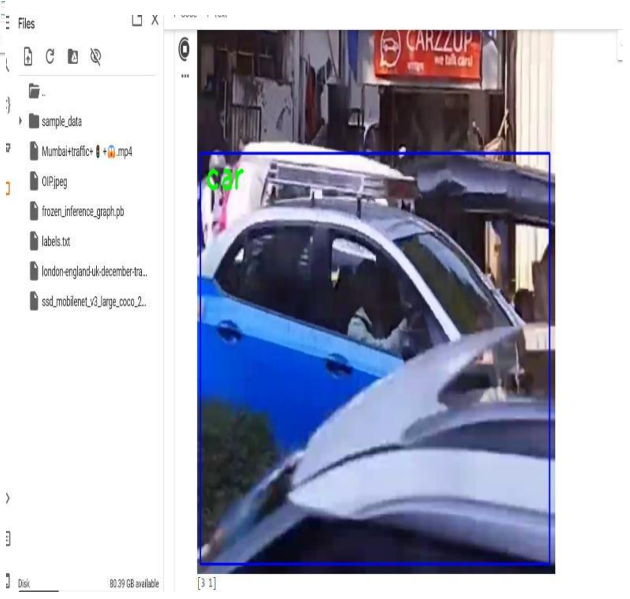
```
[ ]  plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7d08a3b1a770>
```



4.1 Object Detection By Video:




Files
sample_data
Mumbai+traffic+.mp4
OIP.jpeg
frozen_inference_graph.pb
labels.txt
london-england-uk-december-tra...
ssd_mobilenet_v3_large_coco_2...
CARZZUP
car
Disk
80.39 GB available

CONCLUSION

In conclusion, the object detection model demonstrated in the code efficiently captures and processes video frames to identify and classify objects in real-time. By leveraging OpenCV for video processing and drawing bounding boxes around detected objects, it provides a clear visual representation of detected items along with their class labels and confidence scores. The use of a confidence threshold ensures that only reliable detections are displayed. This setup serves as a strong foundation for further enhancements, such as integrating advanced detection models or improving the user interface for broader applications in real-world scenarios like traffic monitoring, security surveillance, or autonomous driving systems

FUTURE SCOPE

Integration with Advanced Deep Learning Models:

The future of object detection will see increased integration with more advanced deep learning models such as YOLO (You Only Look Once), Faster R-CNN, or Vision Transformers (ViT). These models will enable faster, more accurate detection of objects in real-time video streams, with applications ranging from autonomous vehicles to industrial automation.

3D Object Detection and Augmented Reality (AR):

Future object detection models will move from 2D to 3D space, allowing for better depth perception and interaction with the environment. This will open up new possibilities for AR and VR applications, where real-time detection of objects in a 3D environment will enhance user experience in gaming, training simulations, and interactive user interfaces.

Self-Supervised and Unsupervised Learning:

Traditional object detection models rely heavily on labeled datasets for training. The future will see an increase in self-supervised or unsupervised learning methods, where models can learn to detect and classify objects without extensive labeled data, making the training process more scalable and reducing the dependence on manually annotated datasets.

8. REFERENCE:

1. ReserchGate. VIDEO-RECOGNITION-Flow-Chart. Available at https://www.researchgate.net/figure/Flowchart-of-an-MOT-system_fig1_369644129

2. MUNGFALI. IMAGE-RECOGNITION-Flow-Chart. Available at https://mungfali.com/post/886609AC84E63F5A59636F88CE7266AEE4A187E9/Object+Detection+Flowchart

3. OBJECT DETECTION IMAGE MODEL PROJECT. Available at https://github.dev/ArijeetGoswami/OBJECT-DETECTION-MODEL/blob/main/OPEN_CV_FINAL_PROJECT.ipynb