**OpenCV (cv2)**

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It includes hundreds of computer vision algorithms and is widely used in various real-time applications.

**Importing OpenCV and Matplotlib**

*python code*

*import cv2*

*import matplotlib.pyplot as plt*

* **cv2**: This is the Python binding for OpenCV.
* **matplotlib.pyplot as plt**: This imports the pyplot module from Matplotlib, a plotting library, and gives it the alias plt. It's used for creating static, interactive, and animated visualizations in Python.

**Configuration File and Frozen Model**

**Configuration File**

*python code*

*config\_file = 'ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt'*

* **config\_file**: This is the path to the configuration file (.pbtxt) for the SSD MobileNet v3 model.

**ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt**: This is a text file containing the network architecture, including layer definitions and parameters required to construct the network.

It’s specifically tailored for the SSD MobileNet v3 model trained on the COCO dataset (Common Objects in Context), which is a large-scale object detection, segmentation, and captioning dataset. **MobileNet v3** is an efficient deep learning model designed for mobile and embedded vision applications, providing a balance between speed and accuracy. The **COCO (Common Objects in Context) dataset (2020 version)** is a large-scale dataset used for object detection, segmentation, and captioning, serving as a benchmark for evaluating computer vision models.

**Key Points**

**MobileNet v3**

* Efficiency: Uses depthwise separable convolutions to reduce computational cost.
* SE Blocks: Enhances feature extraction by recalibrating channel-wise feature responses.
* NAS: Utilizes automated design techniques to optimize the architecture.
* Variants: MobileNet v3-Large and MobileNet v3-Small for different needs.
* Performance: Improved accuracy and efficiency over previous versions.

**COCO Dataset (2020 Version)**

* Diverse Categories: 80 object categories, including people, animals, and vehicles.
* Detailed Annotations: Bounding boxes, segmentation masks, keypoints, and captions.
* Large-Scale: Over 330,000 images with more than 1.5 million object instances.
* High-Quality: Images from diverse sources for a representative dataset.
* Updated Version: 2020 version includes improved annotations and additional images.

**Frozen Model**

*python code*

*frozen\_model = 'frozen\_inference\_graph.pb'*

* **frozen\_model**: This is the path to the frozen model file (.pb).
  + **frozen\_inference\_graph.pb**: This is a binary file that contains the pre-trained weights and the computational graph of the model. In TensorFlow, "freezing" a model means combining the graph structure and the trained weights into a single file, making it easier to deploy for inference. The pb stands for Protocol Buffers, a binary serialization format used by TensorFlow to store large-scale models.

**Key Points**

* **All-in-One File**: Combines the model's structure and learned information into one file.
* **Ready for Use**: Optimized to make predictions quickly, without extra training steps.
* **Portable**: Easy to move and use on different devices, like phones or servers.

**Loading the Model**

*python code*

*model = cv2.dnn\_DetectionModel(frozen\_model, config\_file)*

* **cv2.dnn\_DetectionModel**: This is a class in the OpenCV dnn (Deep Neural Network) module. It simplifies the process of loading pre-trained models and using them to detect objects in images or videos and designed for loading and running object detection models with minimal code.
  + **frozen\_model**: This argument specifies the path to the frozen inference graph (frozen\_inference\_graph.pb).
  + **config\_file**: This argument specifies the path to the configuration file (ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14.pbtxt).

This line initializes the SSD MobileNet v3 model with the pre-trained weights and the network architecture defined in the config file. It allows you to run object detection on images or video frames.

**Explanation of the SSD MobileNet v3 Model**

* **SSD (Single Shot MultiBox Detector)**: SSD is an object detection model that detects objects in images and video frames in a single pass, making it faster than many other detection models. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell.
* **MobileNet v3**: MobileNet is a family of efficient models designed for mobile and embedded vision applications. MobileNet v3 is an improved version that provides a good balance between speed and accuracy, making it suitable for real-time applications on resource-constrained devices.
* **COCO Dataset**: The COCO dataset is a large-scale dataset for object detection, segmentation, and captioning. Models trained on this dataset can detect a wide variety of everyday objects.

**In brief:-**

1. **cv2**: Python binding for OpenCV, used for computer vision tasks.
2. **matplotlib.pyplot**: Module from Matplotlib for creating visualizations.
3. **config\_file**: Path to the configuration file for the SSD MobileNet v3 model, containing network architecture.
4. **frozen\_model**: Path to the frozen inference graph, containing pre-trained weights and the computational graph.
5. **cv2.dnn\_DetectionModel**: Class to initialize the SSD model for object detection using the specified model and configuration files.

**DetectionModel Class**

**cv2.dnn\_DetectionModel**: A class in the OpenCV dnn module that helps you use pre-trained object detection models. It allows you to load a model, configure it, and run object detection with minimal code.

**Key Functions and Capabilities**

1. **Loading Pre-trained Models**:
   * It allows you to load pre-trained object detection models, such as SSD (Single Shot MultiBox Detector), YOLO (You Only Look Once), and others, using their configuration and weights files.

**Setting Input Parameters**:

* You can configure the input parameters of the model, such as input size, scaling factor, mean values, and channel swapping. These parameters ensure that the input image is preprocessed correctly before being passed to the model.

**Running Object Detection**:

* The class provides a detect method to perform object detection on an input image. It returns the class IDs, confidence scores, and bounding boxes of the detected objects.

**Post-processing and Visualization**:

* Once you have the detection results, you can process and visualize them by drawing bounding boxes and labels on the image.

*Python code*

*classLabels = fpt.read().rstrip('\n').split('\n')*

* **Purpose**: This reads the contents of the file and processes it to populate the classLabels list.
* **Details**:
  + fpt.read(): Reads the entire content of the file into a single string.
  + .rstrip('\n'): Removes any trailing newline characters (\n) from the end of the string. This ensures there are no empty strings at the end of the list after splitting.
  + .split('\n'): Splits the string into a list where each element corresponds to a line in the file. Each line in the labels.txt file should represent a different class label.

*Python code*

*model.setInputSize(320, 320)*

*model.setInputScale(1.0 / 127.5)*

*model.setInputMean((127.5, 127.5, 127.5))*

*model.setInputSwapRB(True)*

These lines of code configure the input settings for the cv2.dnn\_DetectionModel in OpenCV. Each method call sets a specific parameter for the input image before it is processed by the object detection model. Here is a detailed explanation of each method:

**1. model.setInputSize(320, 320)**

* **Purpose**: Sets the size of the input image for the model.
* This resizing is necessary because most deep learning models require a fixed input size. The image will be resized to 320x320 pixels before being processed by the model.

**2. model.setInputScale(1.0 / 127.5)**

* **Purpose**: Sets the scaling factor for the input image.
* Pixel values are typically in the range [0, 255].By scaling with 1.0 / 127.5, you transform the pixel values to the range [-1, 1]. This is often done to normalize the data, which can help improve the performance and stability of the model.

**3. model.setInputMean((127.5, 127.5, 127.5))**

* **Purpose**: Sets the mean subtraction values for the input image (blue, green, and red channels, respectively)
* Subtracting these mean values from each channel of the image helps to center the data around zero. This can help the model converge faster and improve accuracy.

**4. model.setInputSwapRB(True)**

**Purpose**: Swaps the red and blue channels of the input image.

*Python code*

*img = cv2.imread('GIRL.PNG')*

*plt.imshow(img)*

**1. img = cv2.imread('GIRL.PNG')**

* **Purpose**: Reads an image from a file.

And show in your notebook.

*python code*

*ClassIndex, confidence, bbox = model.detect(img, confThreshold=0.5)*

*print(ClassIndex)*

* **Purpose**: This line performs object detection on the input image img and print the value.
* **Details**:
  + model.detect(img, confThreshold=0.5): Uses the pre-trained detection model to find objects in the image.
  + confThreshold=0.5: Sets the confidence threshold to 0.5. Only detections with confidence scores above this threshold will be considered.
  + Returns three arrays:
    - ClassIndex: The class IDs of the detected objects.
    - confidence: The confidence scores for each detection.
    - bbox: The bounding boxes for each detection, given as a list of rectangles.

*Python code*

*font\_scale = 3*

*font = cv2.FONT\_HERSHEY\_PLAIN*

* **Purpose**: Specifies the font type and scale to be used for text annotations on the image.

*python code*

*for ClassInd, conf, boxes in zip(ClassIndex.flatten(), confidence.flatten(), bbox):*

*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)*

**Purpose**: Iterates through each detection and draws a bounding box and class label on the image.

**Details**:

* zip(ClassIndex.flatten(), confidence.flatten(), bbox): Combines the flattened arrays of class indices, confidence scores, and bounding boxes into tuples for iteration.
* for ClassInd, conf, boxes in ...: Iterates over each detection.
* cv2.rectangle(img, boxes, (255, 0, 0), 2): Draws a rectangle around the detected object.
* boxes: The coordinates of the bounding box.
* (255, 0, 0): The color of the rectangle in BGR format (blue).
* 2: The thickness of the rectangle's border.
* cv2.putText(img, classLabels[ClassInd-1], (boxes[0]+10, boxes[1]+40), font, fontScale=font\_scale, color=(0, 255, 0), thickness=3):
* Adds the class label text near the bounding box.
* classLabels[ClassInd-1]: Retrieves the class label corresponding to the detected object's class index (note that class indices are typically 1-based, hence the -1 adjustment).
* (boxes[0]+10, boxes[1]+40): The position to place the text (adjusted slightly from the top-left corner of the bounding box).
* font: The font type.
* fontScale=font\_scale: The scale of the font.
* color=(0, 255, 0): The color of the text in BGR format (green).
* thickness=3: The thickness of the text.

*python code*

*plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))*

* **Purpose**: Converts the image from BGR to RGB format and displays it using Matplotlib.
* **Details**:
  + cv2.cvtColor(img, cv2.COLOR\_BGR2RGB): Converts the image from BGR format (used by OpenCV) to RGB format (used by Matplotlib).
  + plt.imshow(...): Displays the image.

*Python code*

*cap = cv2.VideoCapture('4440930-hd\_1920\_1080\_25fps.mp4') #path is given from local machine & it is downloaded video of any street side*

*if not cap.isOpened():*

*cap= cv2.VideoCapture(0)*

*if not cap.isOpened():*

*raise IOError('Cant open the video')*

*font\_scale= 3*

*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),thickness = 3)*

*cv2.imshow('objection', frame)*

*if cv2.waitKey(2) & 0xff == ord('q'):*

*break*

*cap.release()*

cv2.destroyaLLWindows()

1. **Load Video File**
   * Purpose: Load a video file from the specified path.
2. **Fallback to Webcam if Video Not Opened**
   * Purpose: Attempt to open the default webcam if the video file cannot be opened.
   * Raise an error if neither the video file nor the webcam can be opened.
3. **Set Font Parameters**
   * Purpose: Specify the font type and scale for text annotations on video frames.
4. **Read and Process Video Frames in a Loop**
   * Purpose: Continuously read frames from the video or webcam.
5. **Perform Object Detection**
   * Purpose: Detect objects in the current frame using the pre-trained model with a confidence threshold of 0.55.
6. **Draw Bounding Boxes and Labels for Detected Objects**
   * Purpose: Iterate over detected objects and draw bounding boxes and class labels on the frame.
7. **Display the Annotated Frame**
   * Purpose: Display the frame with annotated bounding boxes and labels in a window.
8. **Break Loop on 'q' Key Press**
   * Purpose: Exit the loop and stop the video processing when the 'q' key is pressed.
9. **Release Resources and Close Windows**
   * Purpose: Release the video capture object and close all OpenCV windows.