Vehicle Counting, Classification & Detection using OpenCV & Python

TLDR!!

- This documentation will show you how to use OpenCV and Python to count vehicles, classify them, and detect them.
- Using OpenCV, we'll create a sophisticated car detection and categorization project.
- We'll utilize OpenCV-python and the YOLOv3 model. A Python library for real-time computer vision is called Open-CV. With OpenCV, we can use YOLO directly.

What exactly is YOLO?

You Only Look Once is an acronym that stands for You Only Look Once. It is an object recognition algorithm that operates in real time. It is capable of classifying and localizing multiple objects in a single frame. Because of its simpler network architecture, YOLO is a very fast and accurate algorithm.

How does YOLO work?

These are the primary techniques employed by YOLO.

1. Residual Blocks - This function divides an image into NxN grids.
2. Bounding Box Regression - Each grid cell is sent to the model. Then YOLO calculates the probability that the cell contains a specific class, and the class with the highest probability is chosen.
3. Intersection Over Union (IOU) - IOU is a metric that evaluates the intersection of the predicted and ground truth bounding boxes.
4. A Non-max suppression technique is used to eliminate the very close bounding boxes by performing the IoU with the one with the highest class probability among them.

- The YOLO network is made up of 24 convolutional layers that are followed by two fully connected layers. The convolutional layers are trained on the ImageNet classification task at half the resolution (224x224 input image) before being double-trained for detection.
- The various layers To reduce the feature space from previous layers, alternate 1 1 reduction layer and 33 convolutional layers.
- The final four layers are added to train the network to detect objects.
- The final layer forecasts the object class and bounding box probabilities.

Vehicle Detection and Classification Project Using OpenCV

In this project, we will detect and classify cars, HMV (Heavy Motor Vehicle), and LMV (Light Motor Vehicle) on the road, as well as count the number of vehicles on the road. And the data will be saved in order to analyze various vehicles on the road.

To complete this project, we will develop two programmes. The first will be a vehicle detection tracker that uses OpenCV to keep track of every detected vehicle on the road, and the second will be the main detection programme.

Prerequisites for Vehicle Detection and Classification Project using OpenCV:

1. Python – 3.x (We used python 3.8.8 in this project)
2. OpenCV – 4.4.0

★ It is strongly recommended to run DNN models on GPU.
You can install OpenCV via "pip install opencv-python opencv_contrib-python".

3. Numpy – 1.20.3
4. YOLOv3 Pre-trained model weights and Config Files.

Tracker:

- To keep track of an object, the tracker employs the Euclidean distance concept.
- It computes the distance between two center points of an object in the current frame and the previous frame, and if the distance is less than the threshold distance, it confirms that the object in the previous frame is the same object in the current frame.
- The Euclidean distance is returned by the math.hypot() method.
- If the distance is less than 25, then the object is the same as in the previous frame.

Counter for Vehicles:

Steps for Detection and Classification of Vehicles Using OpenCV:

1. Import the necessary packages and start the network.
2. Retrieve frames from a video file.
3. Run the detection after pre-processing the frame.

4. Perform post-processing on the output data.
5. Count and track all vehicles on the road.
6. Save the completed data as a CSV file.

❖ Import the necessary packages and start the network.

- First, we import all of the project's required packages.
- Then, from the tracker programme, we initialize the EuclideanDistTracker() object and set the object to "tracker."
- confThreshold and nmsThreshold are the detection and suppression minimum confidence score thresholds, respectively.

➔ There are the crossing line positions that will be used to count the vehicles.
    - ◆ **Note**: Modify middle_line_position according to your need.

- Because YOLOv3 is trained on the COCO dataset, we read the file containing all of the class names and store them in a list.
- The COCO dataset includes 80 distinct classes.

★ For this project, we only need to detect cars, motorcycles, buses, and trucks, so the required class index contains the index of those classes from the coco dataset.

- Use the cv2.dnn.readNetFromDarknet() function to configure the network.
- We're using GPU in this case, so we set "net.setPreferableBackend" to DNN BACKEND CUDA. as well as net.setPreferableTarget to DNN TARGET CUDA.
- Set the DNN backend to CUDA if you're using GPU, and comment out those lines if you're using CPU.
- We generate a random color for each class in our dataset using the np.random.randint() function. These colors will be used to draw the rectangles around the objects.
- The random.seed() function saves the state of a random function so that it can generate some random number on each execution, even if it generates the same random numbers on other machines as well.

❖ **Retrieve** frames from a video file.

- Cap.read() reads each frame from the capture object after reading the video file through the videoCapture object.
- We cut our frame in half by using cv2.reshape().
- The crossing lines are then drawn in the frame using the cv2.line() function.
- Finally, we displayed the output image using the cv2.imshow() function.

❖ **Run the detection** after pre-processing the frame.

- Our YOLO version accepts 320x320 image objects as input. The network's input is a blob object. The function dnn.blobFromImage() accepts an image as input and returns a blob object that has been resized and normalized.

- The image is fed into the network using net.forward(). And it produces a result.
- Finally, to post-process the output, we invoke our custom postProcess() function.

❖ Perform **post-processing** on the output data.

- First, we created an empty list called 'detected classNames,' in which we will store all of the detected classes in a frame.
- We iterate through each vector of each output using two for loops to collect the confidence score and classId index.
- Then we check to see if the class confidence score is higher than our predefined confThreshold. The information about the class is then collected and stored in three separate lists: box coordinate points, class-Id, and confidence score.

❖ **Non-Maximal Suppression**:

- Because YOLO occasionally returns multiple bounding boxes for a single object, we must reduce the number of detection boxes and select the best detection box for each class.
-
- We reduce the number of boxes and take only the best detection box for the class using the NMSBoxes() method.
- Text is drawn in the frame by cv2.putText.
- We draw a bounding box around the detected object using cv2.rectangle().

❖ **Count and track** all vehicles on the road.

- After receiving all of the detections, we use the tracker object to keep track of those objects. The tracker.update() function keeps track of all detected objects and updates their positions.
- The custom function Count vehicle counts the number of vehicles that have passed through the road.

❖ The **count_vehicle** Function:

- Make two temporary empty lists to hold the vehicle ids that cross the entry crossing line.
- Up list and down list are used to count the four vehicle classes in the up and down routes.
- The center point of a rectangle box is returned by the find center function.
- In this section, we keep track of each vehicle's position and Id.
- First, we check to see if the object is between the up-crossing line and the middle crossing line, and then we store the object's id in the up list for up route vehicle counting. We also do the opposite for down route vehicles.
- Then we check to see if the object has crossed the down line. If the object crossed the down line, its id is counted as an up route vehicle, and we add 1 with the specific type of class counter.
- Because we're counting vehicles on the Y-axis, we only need y coordinate points.

- In the frame, Cv2.circle() draws a circle. In this case, we're drawing the car's center of gravity.
- Finally, draw the counts in real-time to show the vehicle counting on the frame.

❖ Save the completed data as a **CSV file**.

- We open a new file data.csv with write permission only using the open function.
- Then we write three rows: the first with class names and directions, the second with up and down route counts, and the third with both.
- The writerow() function saves a row of data to a file.

Conclusion

In this project, we developed an advanced vehicle detection and classification system using OpenCV. We used the YOLOv3 algorithm in conjunction with OpenCV to detect and classify objects. We also studied deep neural networks, file systems, and advanced computer vision techniques.

Using OpenCV, we created an advanced vehicle detection and classification system for this project. To detect and classify objects, we used the YOLOv3 algorithm in conjunction with OpenCV. In addition, we learned about deep neural networks, file systems, and advanced computer vision techniques.