MALCOM Project Report

Real-time object detection for autonomous cars using Deep Learning

Senane Zineb, Harkati Rayene Chiraz

Motivation

While passing by the St-Philippe bus stop, we discovered an advertising poster about autonomous cars. This poster attracted our intention and motivated us to learn more about object detection.
Self-driving or autonomous cars are one of the most research topics of interest nowadays and the most anticipated technologies of the current century. The idea is to navigate roadways without human intervention or control by performing a real-time detection of the different objects they can face. The sensing and perceiving tasks of AI-driven cars will be performed by integrating a deep learning architecture and model. However, this object detection problem is still a major challenge for computer vision and Machine Learning as it is hard to have a precise localization of the objects and not compromise the speed of execution and the accuracy of the models used. For this trade-off between speed and real-time detection, we will use the state-of-the-art model called YOLO introduced in 2015.

Related work

Since object detection is recently one of the topics of interest, there are many works on it. The object detectors are divided into one-stage and two-stage detectors. Two-stage detectors detect objects in two phases: a region proposal stage and a classification and localization stage while one-stage detectors achieve higher localization and object recognition accuracy but with lower inference speed. One-stage detectors are faster because they directly predict boxes from the input image without a region proposal therefore they are well suited for real-time use.
Two-stage detector R-CNN was the first region-based CNN for object detection using selective search to propose regions and a normal convolutional neural network to extract features on each region proposal. Due to its complex training and slow testing, a faster version called fast R-CNN was proposed. This time, features are extracted once, and then region proposals are produced, which are then classified. Mask R-CNN is an extension of Faster R-CNN by adding a fully connected mask head to the network. It is mainly used for instance segmentation tasks.
One-stage detectors YOLO and its last three versions (v2,v3,v4) are one-stage detectors that are very popular for real-time object detection because of their high speed. YOLO divides the image into a grid and predicts bounding boxes and class scores in each cell. The Single-shot multibox detector (SSD) directly predicts class scores and box offsets for a fixed set of default bounding boxes of different scales at each location in several feature maps with different scales.

Data set

We used the Berkeley Deep Drive dataset (BDD100K), which is a diverse driving dataset for heterogeneous multitask learning instances, constructed with 100K videos and 10 tasks to evaluate the exciting progress of image recognition algorithms in autonomous driving.
This dataset images contain different types of objects in different circumstances: geographic, environmental, and weather diversity. This diversified dataset will be useful for our task because the pre-trained models will be less likely to be surprised by new conditions.
We will use a few images from this dataset to perform object detection on images. For real-time object detection, we will use a live camera.

Methods

The goal of our project is to detect and classify objects with respect to their nature in an image and a real-time video using a DNN structure. We used the pre-trained state-of-the-art model YOLO trained on the Common Objects in Context (COCO) database composed of 123,287 images and 886,284 instances to detect 80 different object classes. YOLO uses darknet, an open-source, deep neural network framework that supports CPU and GPU computations. We first used the YOLOv3 version to detect objects in the BDD100K images and some images captured by our own cameras. Then we used the YOLOv4 version in order to perform real-time object detection, and we tested it with the live camera through google colab. The performance of the models will be based on the classification, localization of the bounding boxes, and the confidence metric.

Data Analysis and Preprocessing for images

We used cv2.dnn.blobFromImage method that returns a blob which is a transformation of our input image into a 4d NumPy array (images, channels, width, height) after mean subtraction, normalizing, and channel swapping.

Normalizing image inputs: Data normalization is an important step that ensures that each input parameter (pixel), has a similar data distribution. This makes convergence faster while training the network.

Data normalization is done by subtracting the mean from each pixel and then dividing the result by the standard deviation (255) in order to range the pixels from 0 to 1.

Resizing: Data resizing is also critical since we need to pass a specific input dimension to the convolutional neural layer. In our setting, we supplied the spatial size that the convolutional neural network expects, which is 416x416.

Swapping channels: As openCV assumes BGR channel order for images and the mean parameter assumes RGB order for mean values. To avoid this discrepancy problem, we need just to swap the B and R channels by setting `swapRB` to 'True'.

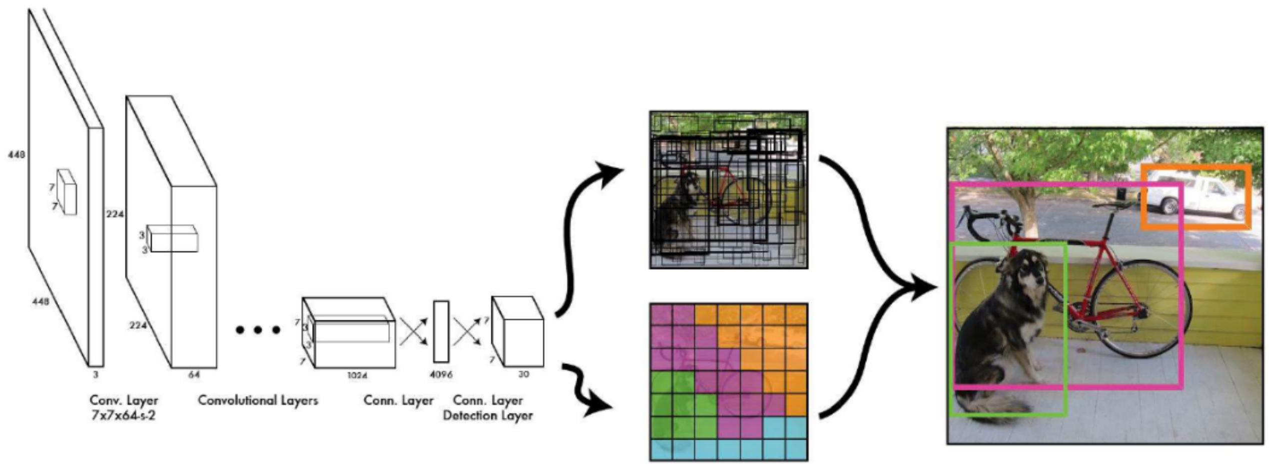
Models

You Look Only Once (YOLO) is an algorithm for object detection tasks, it uses the features learned by a deep convolutional neural network to detect the objects. The name of the model comes from the fact that it looks at the entire image only once and goes through the network once to detect the objects, which was not the case for the other existing models used in object detection as they work with region proposals and look at the image multiple times.

YOLO is very popular for object detection due to its high speed, it divides the input image into grid cells and for each grid cell, it predicts a given number of bounding boxes, where each bounding box consists of these coordinates X, Y which are the center of the bounding box relative to the cell, the width and the height of the bounding box relative to the whole image and a confidence score (that should be above the value we fixed which is 0.5) for the prediction, and given class probabilities per grid cell then takes the highest one as the final class.

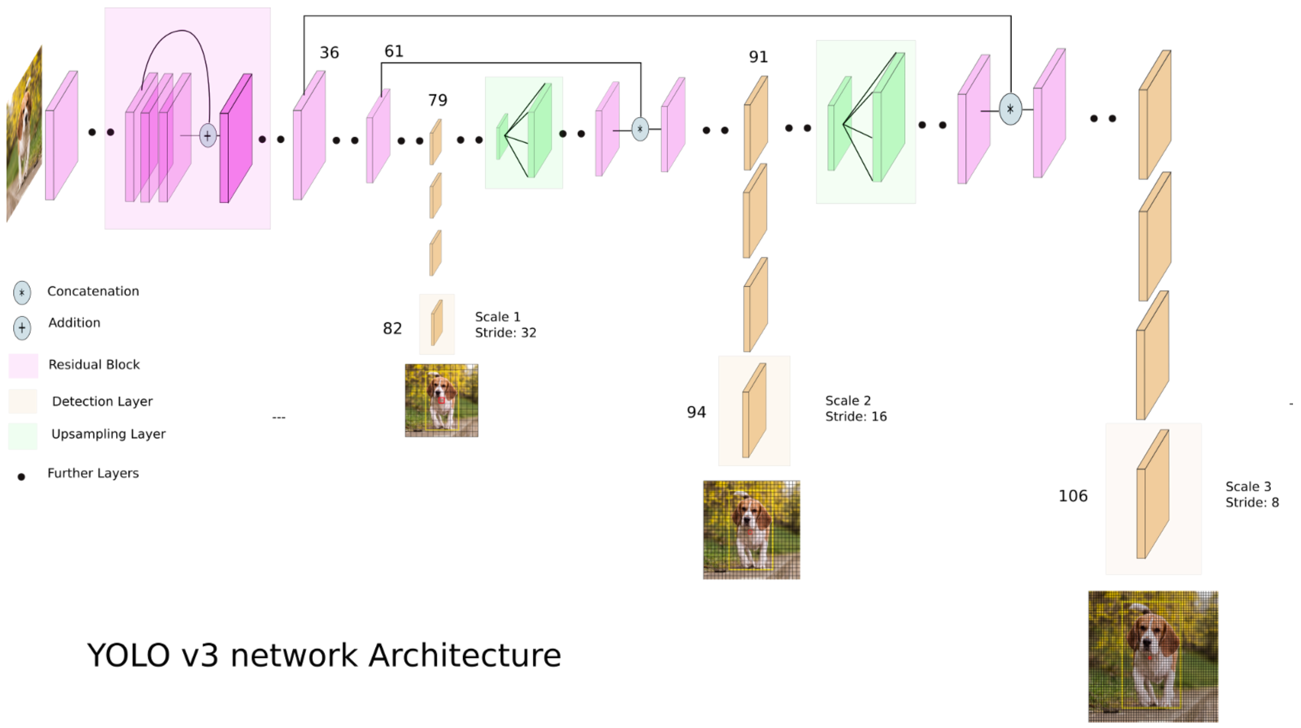
The YOLO algorithm uses a custom loss function in order to control the different output domains and their influence on the final loss by using special hyperparameters: It first penalizes bad locations for the center coordinates if the cell contains an object, then it penalizes bad bounding box width and height values. The square root is present so that errors in small bounding boxes are more penalizing than errors in big bounding boxes. After that, it passes to the confidence scores where it penalizes the small scores for cells containing an object and penalizes the big scores for cells containing no object, it discards out weak predictions by ensuring the detected probability is greater than the minimum probability.

What the algorithm finally does, is identify objects in the image and map them to the grid cell containing the center of the object. This grid cell will be responsible for predicting the final bounding box of the object such that it will select the bounding box with the highest confidence then it compares all the resting bounding boxes with the selected one and eliminates the ones that have a high IoU value where IoU is a metric called intersection over union used to measure how close two different bounding boxes are. The higher the IoU, the closer the bounding boxes are and an IoU equal to 1 means that two bounding boxes are identical while an IoU equal to 0 means that they are not even intersected, in our case we fixed the IoU threshold to 0.5, which means that we eliminate any bounding box below this value compared to the maximal probability bounding box, while the score threshold set to 0.5 will eliminate any bounding box that has confidence below that value.



1. <u>YOLOv3</u>

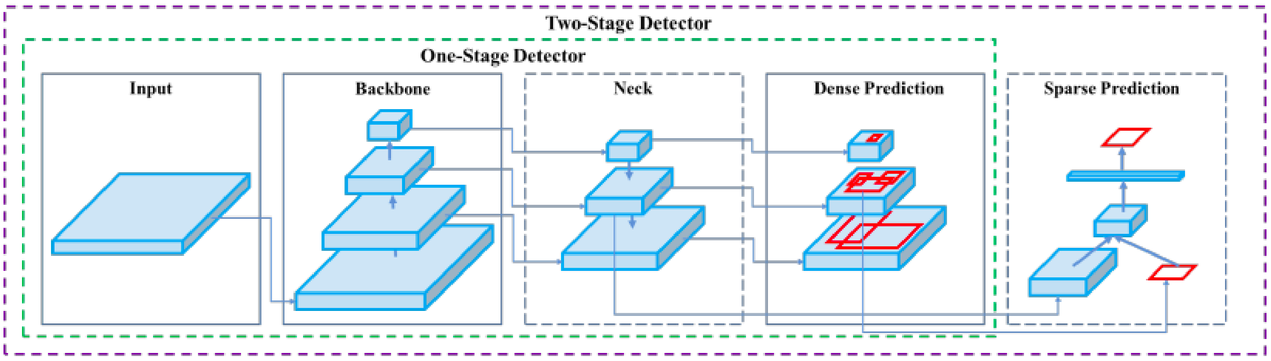
The main idea of YOLOv3 is residual skip connections and upsampling. It consists mainly of convolutional layers, with some shortcut connections and upsample layers. It makes detections at three different scales. using 1x1 kernel on the feature maps and fully connected layers. The shape of the kernel for detection is 1x1x255. Once we pass the input image, YOLOv3 downsamples it by 32,16, and 8 and performs the detection. The second feature map (obtained after a stride of 16) is upsampled by a factor of 2 and compared to the first resulting feature map, the same process is used for the third feature map and these comparisons give the final bounding box and the predicted classes.
To determine the class of objects it uses independent logistic classifiers and binary cross-entropy loss instead of softmax and mean squared error.



YOLO v3 network Architecture

2. YOLOv4

Yolov4 is an improvement on the Yolov3 algorithm by having an improvement in the mean average precision(mAP) by as much as 10% and the number of frames per second by 12%. The Yolov4 architecture has 4 distinct blocks as shown in the image above: the backbone, the neck, the dense prediction, and the sparse prediction.



Backbone:

Is the feature extraction architecture which is used to split the current layer into two parts, one to pass through convolution layers and the other that would not pass through convolutions, after which the results are aggregated.

The neck:

Helps to add layers between the backbone and the dense prediction block(head), which is a bit like what the ResNet architecture does. The yolov4 architecture uses a modified Path aggregation network, a modified spatial attention module, and a modified spatial pyramid pooling, which are all used to aggregate the information to improve accuracy

The head (Dense prediction):

is used for locating bounding boxes and for classification. The process is the same as the one described for Yolo v3, the bounding box coordinates(x,y, height, and width) are detected as well as the score.

In our project, we will use the YOLO model with its two versions YOLOv3 for object detection from images and YOLOv4 for real-time object detection

Results

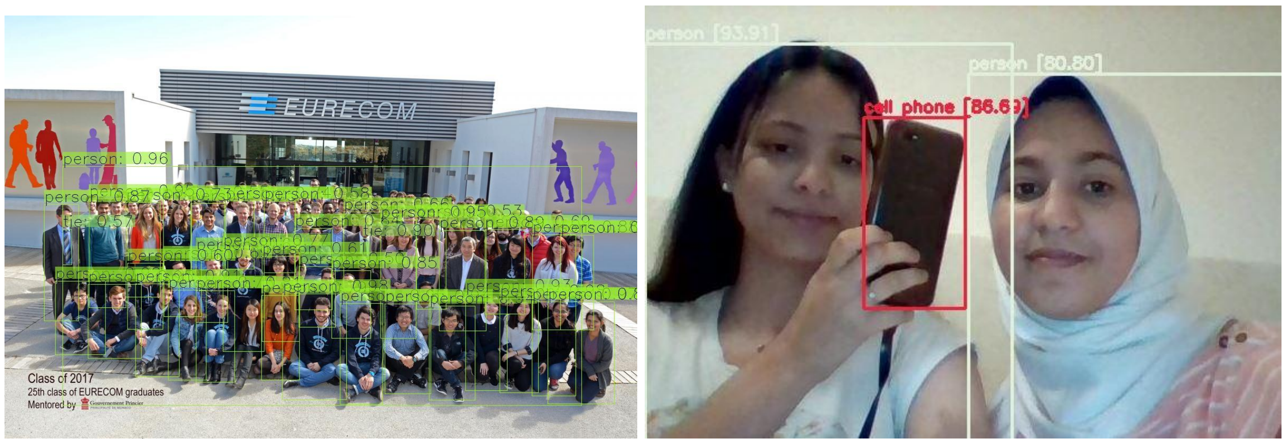


Fig 3: Results for object detection using YOLOv3 (left) and YOLOv4(right)

The implementation of the models and the images used for our experiments are found in our gitlab repository. Furthermore, we provide live videos using our laptop camera for real-time object detection with YOLOv4. We measured the confidence level for the evaluation of YOLO.

Conclusion&Further improvements

As we can see in the image obtained using YOLOv3, the YOLO object detector has also its own drawbacks. First of all, it struggles to detect objects grouped close together. Second, it also has difficulties to detect small objects. However, it has a good tradeoff in terms of speed and accuracy. As future work, we can implement and train a two-stage detector on the BDD100K dataset in order to achieve higher accuracy which is suitable for autonomous car driving purposes.

Gitlab link of our repository

https://gitlab.eurecom.fr/senane/malcomproject/

Bibliography

[1] Yolov3 and Yolov4 in Object Detection
https://ai-pool.com/a/s/yolov3-and-yolov4-in-object-detection

[2] Real-time-Object-Detection-for-Autonomous-Driving-using-Deep-Learning
https://github.com/alen-smajic/Real-time-Object-Detection-for-Autonomous-Driving-using-Deep-Learning