Pedestrian Detection for Autonomous Vehicles

N Prithvi Raju
School of Engineering and Technology
Asian Institute of Technology StudentID: ST120000
Email: nprithviraj24@gmail.com

***Abstract*—In this paper we present a method of detecting and tracking pedestrians from a moving vehicle. We accomplish this by using YOLO v3, the revolutionary convolutional neural network that has set the benchmark for object detection.The model is accurate and fast. It can be used to track the pedestrians on the sidewalk or those who are approaching the road from the side, where partial occlusion can occur. By displaying a bounding box around the subject, the model manages to improve the speed and accuracy since the detection algorithm detects the pedestrians.**

***Index Terms*—Pedestrian, YOLOv3, CNN, Detection**

I. Introduction

With the increase in applications of AI, automation has been the core idea of every AI product. Object tracking and detection has been one of the emerging fields in Machine Learning.

Autonomous vehicle monitoring systems using video cameras are already in use. It constantly records video, audio, GPS and accelerometer data and stores it in separate directories in a database. This system is usually applied as personal courier system or autonomous vehicles which detects the pedestrians and traverse through them.

The main objective of this project is to develop an autonomous system that would be able to detect pedestrians on the road in-front of a moving vehicle and on the sidewalk. This is achieved through the use of computer vision algorithms and image processing. The system did use an onboard optical camera to capture a live video footage of the area of interest. As this is a prototype system, a laptop computer was used to run the detection and tracking algorithms. The option of a low cost video capture, store and forward device is available. However, the dataset was gathered from Caltech Pedestrian Dataset. Basic theme of the project is to detect the pedestrian and map the distance between the device and the subject. Further use this information in SLAM based projects.

Owning a quality camera can be fairly useful by itself. Video stream can provide a lot of information not easily comprehensible by just using various sensors. However, not always there is a human eye to make a sense of it. Therefore, additional algorithms can be implemented to provide a lot of insights automatically. A lot of them can be extracted and tracked using detection algorithms. There are plenty of algorithms to detect objects of a choice in a photo or a video frame. Last five years saw a rise of convolutional neural networks. Their novel architecture enabled to make a detection model to learn high level abstracts by itself, only by using pictures as input data. However, there are a lot of different machine learning models, all incorporating convolutions, but none of them are as fast and precise as YOLOv3 (You Only Look Once). Differently from most of its competitors, it is built in a way that whole image is processed at once. That enables algorithm to work much faster while still offering comparable detection quality.

II. Related Work

- Faster R-CNN for Robust Pedestrian Detection Using Semantic Segmentation Network:
  Convolutional neural networks (CNN) have enabled significant improvements in pedestrian detection owing to the strong representation ability of the CNN features. However, it is generally difficult to reduce false positives on hard negative samples such as tree leaves, traffic lights, poles, etc. Some of these hard negatives can be removed by making use of high level semantic vision cues. In this paper, we propose a region-based CNN method which makes use of semantic cues for better pedestrian detection. This method extends the Faster R-CNN detection framework by adding a branch of network for semantic image segmentation. The semantic network aims to compute complementary higher level semantic features to be integrated with the convolutional features. They make use of multi-resolution feature maps extracted from different network layers in order to ensure good detection accuracy for pedestrians at different scales. Boosted forest is used for training the integrated features in a cascaded manner for hard negatives mining. Experiments on the Caltech pedestrian dataset show improvements on detection accuracy with the semantic network. With the deep VGG16 model, our pedestrian detection method achieves robust detection performance on the Caltech dataset.
- Region-CNN Based Pedestrian Detection Methods: Region-based convolutional neural networks (R-CNN) (Girshick et al., 2014) is a representative region-based detection method using deep neural network (DNN) features. The initial version of the R-CNN detector uses the selective search approach (Uijlings et al., 2013) for region proposal. Despite accurate, R-CNN is too slow for real-time applications even with high-end hardware. Faster R-CNN (Ren et al., 2015) improves R-CNN by replacing selective search (Uijlings et al., 2013) with a built-in network that can directly generate proposals. This sub-

network, referred to as region proposal network (RPN), is integrated with Fast R-CNN (Girshick, 2015) to pool candidate object bounding boxes with features extracted using region of interest (RoI) pooling.
Despite Faster R-CNN being particularly successful for object detection, the results for pedestrian detection are not satisfying on pedestrian benchmark (Dollr et al., 2009b). The anchors used in Ren et al. (2015) for generic object detection are of multiple aspect ratios, which may not be suitable for pedestrian detection. Anchors of inappropriate aspect ratios will induce false detections and are harmful for detection accuracy. In Zhang et al. (2016a), the anchors are tailored into a single aspect ratio of a wider range of scales to be suitable for pedestrian detection and this approach achieves promising results on the Caltech dataset.

III. Proposed Method

YOLOv3 is not first of its kind, but an improved technology from its predecessors. For its time YOLO9000 was the fastest, and also one of the most accurate algorithm. However, a couple of years down the line and its no longer the most accurate with algorithms like RetinaNet, and SSD outperforming it in terms of accuracy. It still, however, was one of the fastest. But that speed has been traded off for boosts in accuracy in YOLO v3. While the earlier variant ran on 45 FPS on a Titan X, the current version clocks about 30 FPS. This has to do with the increase in complexity of underlying architecture called Darknet.

- Darknet-53 YOLO v2 has struggled with detection of small structures, due to its 30 layer architecture. This was attributed to loss of fine-grained features as the layers downsampled the input. To remedy this, YOLO v2 used an identity mapping, concatenating feature maps from from a previous layer to capture low level features. YOLO v2s architecture was lacking some of the most important elements that are now staple in most of state-of-the art algorithms. No residual blocks, no skip connections and no upsampling. YOLO v3 incorporates all of these. First, YOLO v3 uses a variant of Darknet, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is the reason behind the slowness of YOLO v3 compared to YOLO v2. Here is how the architecture of YOLO now looks like.

A. Prerequisites

Before getting started, GCC toolchain must be installed. Latest OpenCV version is also required if one opts to use the tools for displaying images or videos. OpenCV should be compiled for applicable Nvidia GPU if one can be used. Python isnt required, but highly advised for image dataset manipulations, anchor box generation and other things. Lastly, if Nvidia GPU is used and CUDA with Compute Capability ¿3.0 is supported it is highly advised to also install CUDA and CuDNN libraries to make execution much faster. To download files needed to run YOLO, visit https://github.com/pjreddie/darknet. Git is to be used to clone files or just download all the latest files. Alternatively, YOLOv3 files from https://github.com/AlexeyAB/darknet which also contain additional options and instructions about how files should be compiled, how images and videos can be processed as well fine tuning on a custom dataset.

B. Transfer Learning

Deep neural networks are a lot of times trained from scratch using huge datasets such as ImageNet (containing millions of images) and usually generalize well for a huge amount of classes. However, we usually dont need most (if not all) of them and sometimes we need classes that are not in these huge datasets. Training network from scratch on a small custom batch of images would result in overfitting poor generalization in real life conditions even though training accuracy would be very high. Turns out a lot of images share some similarities and features for one object detector usually work well while searching for another object. That means only several last layers could be retrained, taking the rest of neural network as an already built feature extractor.

C. Getting dataset ready

A great dataset for our domain i.e pedestrian detection is called Caltech Pedestrian Dataset (http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/). It consists of 350.000 bounding boxes for 2300 unique pedestrians over 10 hours of videos. For Caltech Pedestrian Dataset it needs to be converted to VOC and later to YOLO format.

Without Data Annotations, we used the tool called YOLO Annotation Tool (https://github.com/ManivannanMurugavel/YOLO-Annotation-Tool). Objectives are:

1. Create `.txt`-file for each `.jpg`-image-file - in the same directory and with the same name, but with `.txt`-extension, and put to file: object number and object coordinates on this image, for each object in new line:
   `<object-class> <x> <y> <width> <height>`

Where:

- `<object-class>` - integer number of object from `0` to `(classes-1)`
- `<x> <y> <width> <height>` - float values relative to width and height of image, it can be equal from 0.0 to 1.0
- for example: `<x> = <absolute_x> / <image_width>` or `<height> = <absolute_height> / <image_height>`
- attention: `<x> <y>` - are center of rectangle (are not top-left corner)

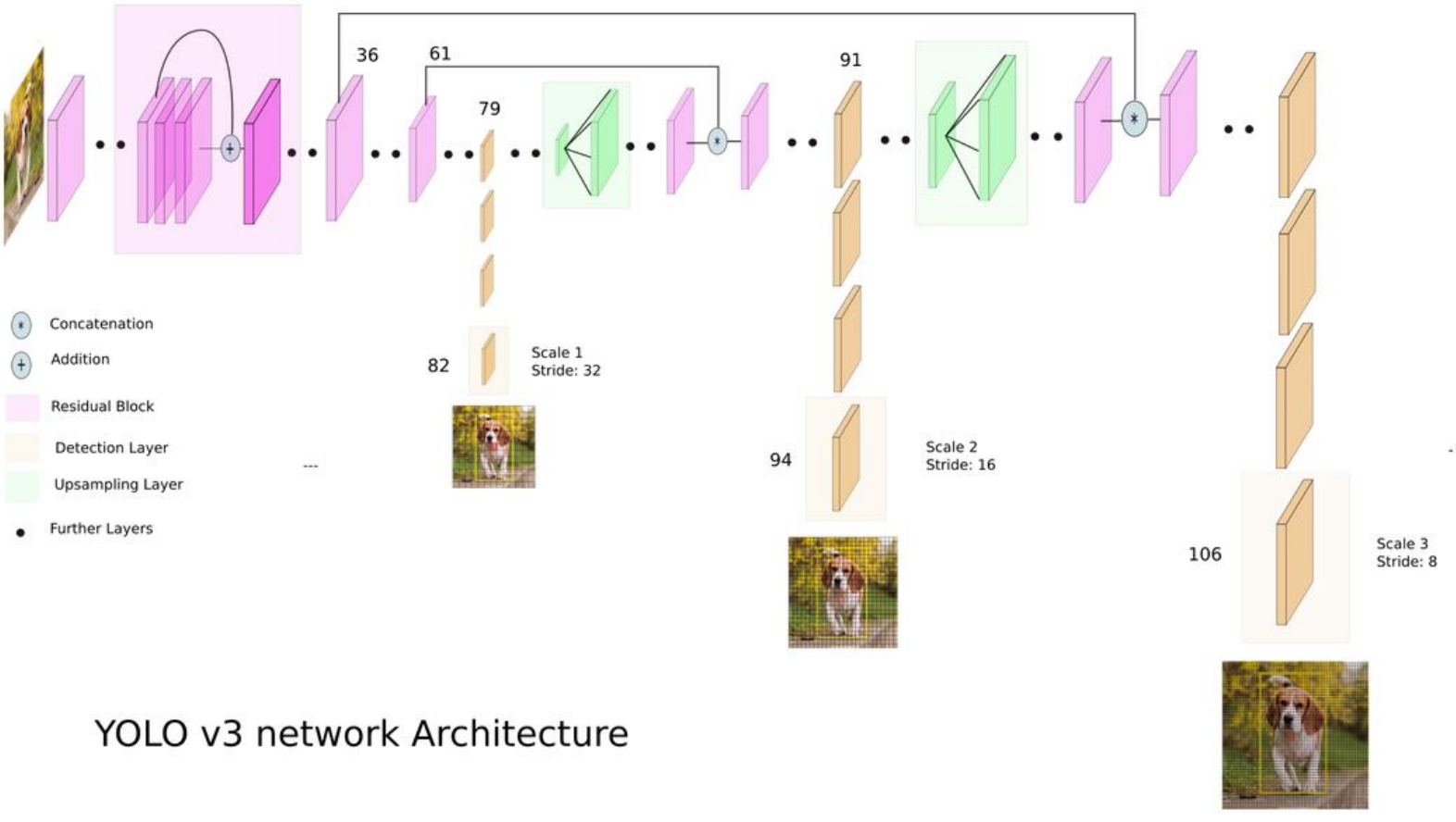
For example for `img1.jpg` you should create `img1.txt` containing:

```
1 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
1 0.420312 0.395833 0.140625 0.166667
```

- Pre-trained weights for the convolutional layers (154 MB) (https://pjreddie.com/media/files/darknet53.conv.74) must put to the directory where darknet file has been compiled.
- Training the model by using the command line: ‘./darknet detector train data/obj.data yolo-obj.cfg darknet53.conv.74‘
- After training is complete get result in .weights file

Additional Details:

- Almost 60k training images, with annotations of each subject(person, car) in each file.
- Yolo-cfg file mention path of each image and its corresponding annotation file.
- Trained on NVIDIA Tesla K80, for 8 straight days. No validation set given.



YOLOv3 ARCHITECTURE

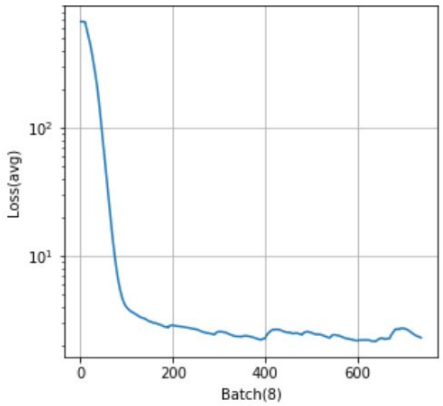
D. Fine Tuning

Steps needed to training YOLOv3 for pedestrian detection:

- Customized configuration file with modified batch size as 64, subdivisions as 8 and class size as 3 as we are detecting bicycle, car and person.

E. Train Loss

Please check log file attached with the drive link to know further. Heres the snippet of Loss curve from train/$log file generated$:



IV. Results

From the pretrained weights and Coco.classes I have kept three classes that will be detected throughout the video to be detected, car, person and a bicycle.

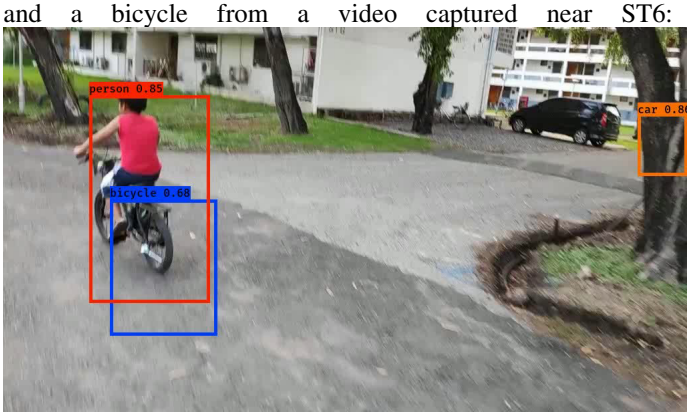
YOLOv3 is considerably big network, which runs objects decently at 30fps. Though YOLOv3 could detect the person and bicycle in a frame, the bounding boxes werent mapped accurately in most of the cases. To run the code, I have taken 3 videos in AIT campus and divided them into frame, code can be found in the directory frames/ written in iPython.

Process: Get the video -¿ Break it into frame -¿ Predict on each frame -¿ get the output frames -¿ combine these frame to form video.

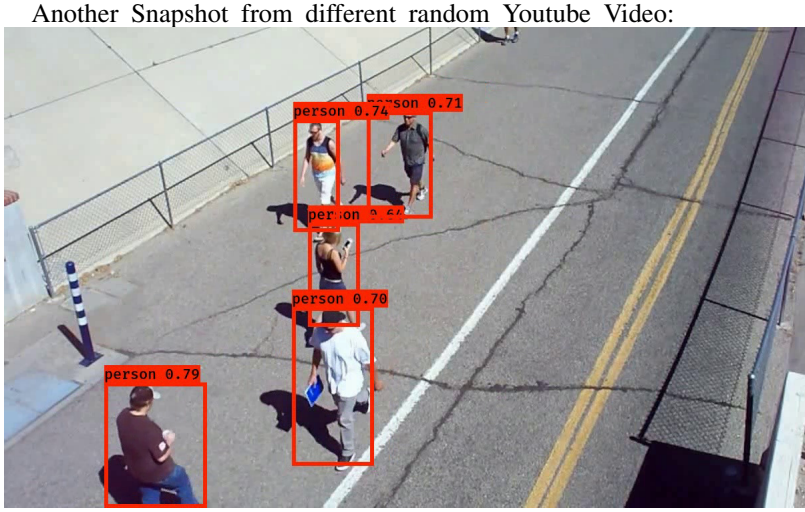
A. Prediction

Andrew Ng has given excellent tutorial on YOLOv3 (one of the weekly assignments in Deep Learning), and I have referred the code from there to predict the bounding box. However the code is written in Keras with tensorflow backend, and Keras wants weights in .h5 file, so YAD2K tool came in handy, which converts .weights to .h5. Code can be found in test/ folder.

Heres the case of accurately detecting a person and a bicycle from a video captured near ST6:

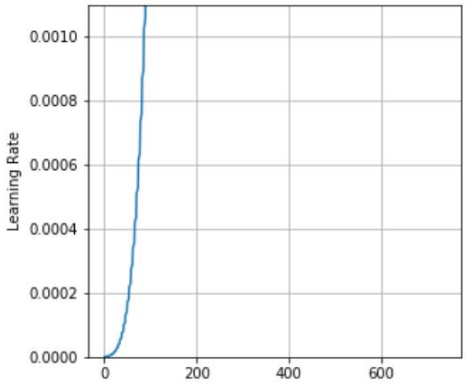


Another Snapshot from different random Youtube Video:



B. Learning Rate

Learning Rate of the system:



V. Future Work

Pedestrian technology can always be used in Simultaneous Localisation and Mapping applications. One of the application is personal courier that follows the owner in a fairly crowded street. Other applications include Autonomous Vehicle System, detecting number of people crossing a street which can then be used to monitor traffic violations by pedestrians.

Sky's the limit for YOLOv3 based systems, because object detection is one of the emerging fields in Computer Science. Commercialisation of such systems has been the trademark of huge IT MNCs.

VI. Conclusion