**`Maria Mumtaz**

Schwarzenburgstrasse 217

3097 Bern

[maria.asad17@gmail.com](mailto:maria.asad17@gmail.com)

[maria.mumtaz@unibe.ch](mailto:maria.mumtaz@unibe.ch)

**Data Science Project**

**Object Detection and Localization for Autonomous Cars**

**Project Report**

**15 June 2022**

**Table of Contents**

[1 Project Overview 2](#_Toc106708108)

[1.1 Motivation 2](#_Toc106708109)

[2 Data collection 2](#_Toc106708110)

[2.1 Data labelling 3](#_Toc106708111)

[2.1.1 Internal labelling 3](#_Toc106708112)

[2.1.2 External labelling 3](#_Toc106708113)

[2.1.3 Semi supervised labelling 3](#_Toc106708114)

[3 Training the CNN model 4](#_Toc106708115)

[3.1 Image Augmentation with Albumentations 5](#_Toc106708116)

[3.2 Intersection over union 5](#_Toc106708117)

[4 Convolutional Neural Network model 7](#_Toc106708118)

[4.1 First evaluation on un-trained data 8](#_Toc106708119)

[4.2 Training the model 8](#_Toc106708120)

[5 The YOLO Algorithm 10](#_Toc106708121)

[5.1 Image sections 10](#_Toc106708122)

[5.2 Predictions and bounding box on image splits 11](#_Toc106708123)

[5.3 Merging overlapping bounding boxes 12](#_Toc106708124)

[6 Output video stream 13](#_Toc106708125)

[7 Conclusions 13](#_Toc106708126)

[8 Acknowledgements 14](#_Toc106708127)

[9 References and Bibliography 14](#_Toc106708128)

# Project Overview

In this project, object detection and localization techniques are utilized using supervised & semi-supervised machine learning to detect other vehicles on road for a self-driving car. A deep learning Convolutional Neural Network (CNN) classification model predicts if an image of the road has any car and localizes them by generating a bounding box around its whereabouts in that image. Images from camera mounted in the vehicle looking forward are fed in a Convolutional Neural Network to predict if there are other cars in the vicinity.

An implementation of YOLO algorithm [1] is used to detected vehicles and bounding boxes are generated to localize the detection. The state-of-the-art YOLO algorithm localizes multiple cars in a single image by performing a sliding window operation.

## Motivation

For self-driving vehicles, safety is of a prime concern. It is therefore very important that the object is precisely and accurately detected in real time. Since autonomous cars interact most with other cars, detecting other cars timely and accurately in the vicinity is the very first step in building a fully autonomous vehicle driving system. Accordingly, this project implements a YOLO based detection that recognize and localizes cars meticulously in real time when input video stream from the cameras is given to the deep learning network. The ‘You-Only-Look-Once’ (YOLO) algorithm for object localization is designed from scratch without using the built-in libraries to understand the intuition behind it. Since the project is based on videos and images, learning computer vision and image processing was also a motivating factor along with implementing machine learning and deep learning techniques. Moreover, the raw data used in the project is also collected from scratch to add robustness to the model. In addition, some already available car images datasets have also been unutilized to complement the train set and making it more comprehensive.

# Data collection

The data is primarily collected from the MP4 videos downloaded from a car’s dash-cam. Additionally, relevant images from Stanford Car Images Dataset [2] specifically those having bounding boxes of a particular proportion in the image were also added to the training data set. Furthermore, to increase car models diversity, several images from the Geneva car show dataset [3] with images of car fronts, backs and sides only were also included in the training set. A total of 5366 images were collected and labelled to train the model out of which 2810 images had cars in them, and 2556 images were no-car images.

## Data labelling

Almost all the training images were collected from scratch with front mounted car's dash cam and were manually to be labelled with the help of a custom labelling tool. Labelled images of cars with bounding box information were then fed to the neural network. The car images were collected and labelled manually because of the following reasons:

* The data is collected in the Swiss region only, to be more specific and for the model to perform better
* Only those images that had cars from the front and back were included from the Stanford dataset, to match with the data that was collected manually

All labelled images were stored as JPEG images with dimensions of 480 x 270 pixels. Following sections explain the different types of data labelling techniques employed.

### Internal labelling

Using CV2 library, the videos are streamed frame by frame and then a bounding box is dragged and drawn manually. The images of cars and no-car (background) are then saved, whereas the bounding box information is stored in a csv representing the coordinates of the four corners of the bounding box.

For image processing, utility functions were built to rotate image, rescale image, and crop image.

### External labelling

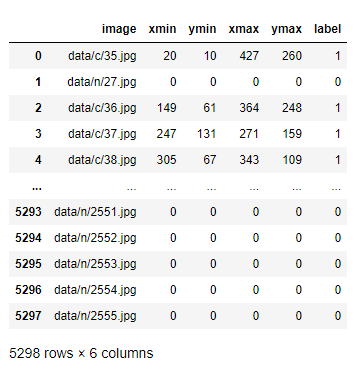
A hand full car images from other car datasets having bounding box information were also used. Qualifying images had car bounding boxes between 80 and 40% of the total dimension of 480 x 270 pixels. For this, the center point of the bounding box was calculated and then the frame was added covering the bounding box. Also, only those images were taken that mostly had car fronts and backs, to make it suitable for a forward mounted vehicle detection system.

### Semi supervised labelling

In semi supervised labelling, the trained model was iteratively run-on new videos and classified images with cars having high confidence were saved. These images were then verified and re-labelled for output labels and bounding boxes. The re-labelled images were then added back to the input data set to improve the training accuracy, and the model was re-trained. This resulted in an increase of overall accuracy of the model.

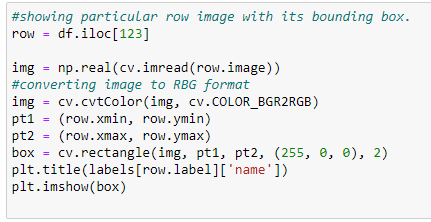
# Training the CNN model

All labelled images for training were added in the data directory, with images having cars in 'data\c' subdirectory and those without cars in 'data\n' subdirectory. An index CSV file contained the relative link to the image directory and the bounding box dimensions. For images with cars, the points with upper left corner and bottom right corner of the bounding boxes were stored in the index file. For images with no car, zeros were stored for bounding box points. For training the model, this file was loaded in data a frame and an additional label field was calculated for class label, as depicted in figure below.

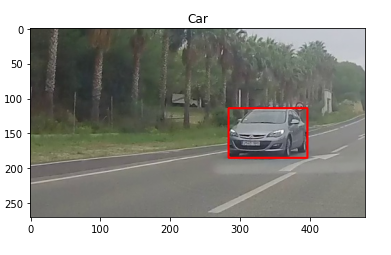


*Fig 3.1: Dataframe with each image path, bounding box information and label*

Once images are read in the data frame, any single image may be projected on screen with this code snippet:



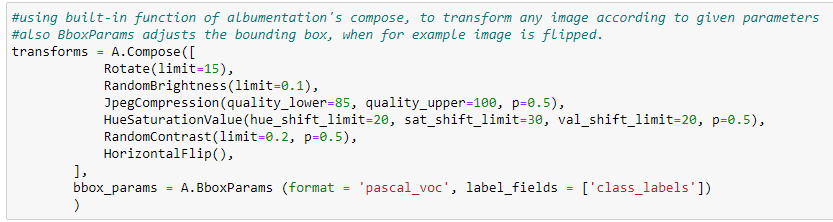
*Fig 3.2: Code for showing a particular image of a particular row in the dataframe*



*Fig 3.3: Image of row number 123 with bounding box*

## Image Augmentation with Albumentations

Albumentations is a Python library for image augmentation. The purpose of the image augmentation is to create new training samples from the existing data. From this library, functions like Compose, RandomBrightness, JpegCompression, HueSaturationValue, RandomContrast, HorizontalFlip and Rotate are used. Each training image was a slight variation of the original images with some tilt, rotation and/or tweak in contrast or brightness. As shown in the following code snippet, the images are transformed when the function is called.



*Fig 3.1.1: Code for applying Albumentations library built-in functions to transform image*

## Intersection over union

To compute the accuracy of bounding box, Intersection over Union (IoU) class is defined. This class is used as a custom evaluation metric to measure the accuracy of the bounding box. It is used as a function of the ground-truth bounding box and the predicted bounding box. If the predicted and the ground-truth bounding boxes overlap perfectly, the IoU will be 1. For better prediction, IoU should be greater than 0.5. IoU is used as a metric and defined when the model is compiled.

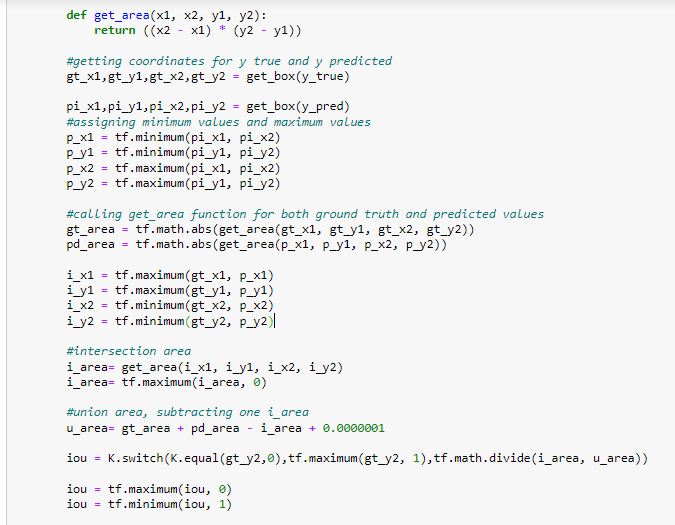
To calculate the IoU, the areas of ground truth bounding box and predicted bounding box were calculated. The intersection is the area of overlap in the bounding box while union is the combined area of bounding boxes minus the area of intersection:

Intersection = (Min(x2gt, x2p) - Max(x1gt, x1p)) \* (Min(y2gt, y2p) - Max(y1gt, y1p))

Union = Area(bboxgt) + Area(bboxp) – Intersection

IoU = Intersection / Union

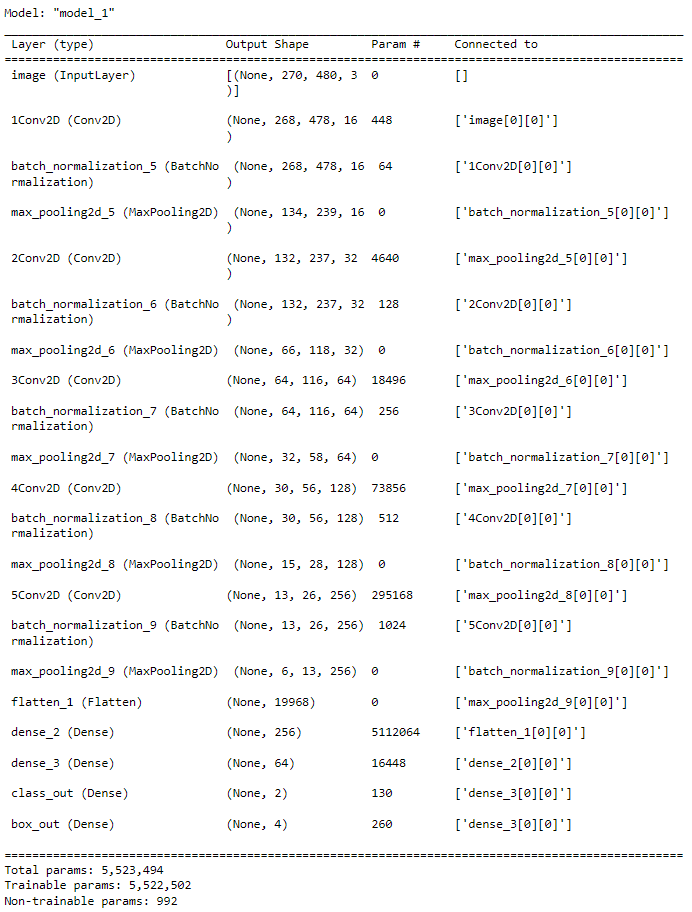
Following code snippet represents the implementation of IoU.



*Fig 3.2.1: Intermediate code for IoU class implementation*

# Convolutional Neural Network model

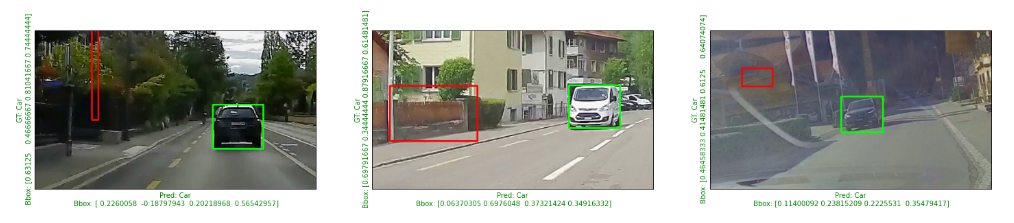
A simple model is defined with i=5 blocks of convolutional layers each having 2i filters and Relu activation, batch normalization and Max pooling 2D. In the end its flattened and two dense layers of size 256 and 64 were added respectively. The model summary is as follows:



*Fig 4.1: A multi-layer Conv2D model*

## First evaluation on un-trained data

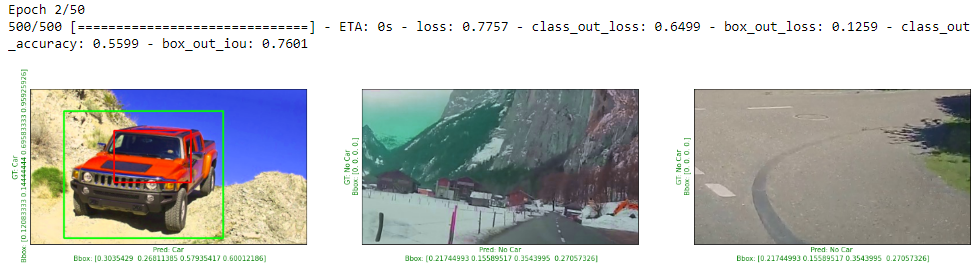
At first the model is tested without any training on the data. test(model) function is called and the first output turns out like this:



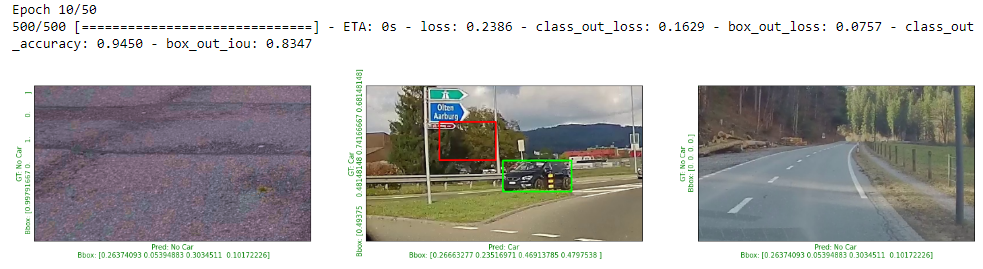
Obviously, the prediction was random, and the locations of bounding boxes was totally wrong. The green bounding box in the above picture is the ground truth and the red one is the predicted bounding box. In this example all three images were coincidently rightly predicted, showing car when there was car in the image. However, the bounding box prediction is totally random because the data is not trained yet.

## Training the model

The model is trained for 50 epochs, with 500 iterations in each epoch. At the end of each epoch, the model is evaluated with test(model) function, that shows three images with green ground truth bounding box, red predicted bounding box and actual and predicted labelling on x and y axis (green if rightly predicted and red if wrongly predicted). The evaluation at the second epoch had 55.9% accuracy for class prediction and 76% for bounding box IoU.



At the end of 10th epoch, the accuracy increased up to 94.5% with bounding box IoU 83.4%.



By the end of 30th epoch, the accuracy increased up to 99.5% with bounding box IoU 83.26%.



By the end of 40th epoch, the accuracy increased up to 99.80% with bounding box IoU 95.26%.



At the end of the training, the 50th epoch, the accuracy increased up to 99.95% with bounding box IoU 90.24%. Be informed that 90% IoU includes the IoU for non-car images as well which is always near 1. Thus, the true value of IoU for images with car could be calculated to be around 0.8.



# The YOLO Algorithm

The ‘You Only Look Once’ or YOLO algorithm sees the entire image and requires only a single forward propagation to detect objects on run-time. In this algorithm, a grid is placed on the image and then a single neural network is applied to the entire image. This is an overlap in successive images, both horizontally and vertically to cater for split cars by the grid slicing which was kept as 25% of image height and length by default. This outputs the image classification and localization for each cell of the grid. Hence for each grid cell, there is a Y label.

## Image sections

For the YOLO algorithm, first the image is divided into sections with the get\_image\_section function. This function takes an image and overlaps value as arguments. With a 50% overlap, there is 50% from old grid and 50% from the new grid. For the next image, the start point would now be 25% of the image and the end point would be 75% of the image. This depicts a sliding window, making a grid and covering the entire image. The image is sectioned according to the overlap value and is now ready to be fed all at once to the model.

The figure below was the original image given to the YOLO algorithm.



*Fig 5.1.1: Original image before sectioning*

With an overlap of 0.45, it made the sections of the image as follows.



*Fig 5.1.2: Image after sectioning with an overlap of 0.45*

## Predictions and bounding box on image splits

Now, the sectioned image is sent to the predict function and the predictions along with the bounding box for each split of the image is returned. The image after the predictions with YOLO algorithm looked like this:

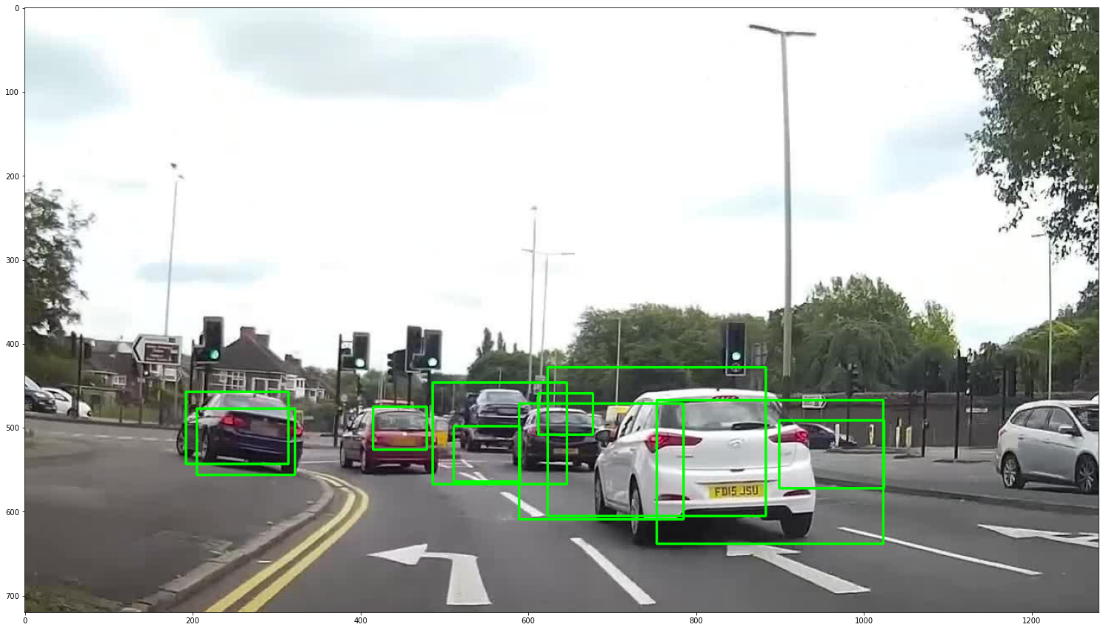


*Fig 5.2.2: Image after applying YOLO algorithm with predictions on each section*

## Merging overlapping bounding boxes

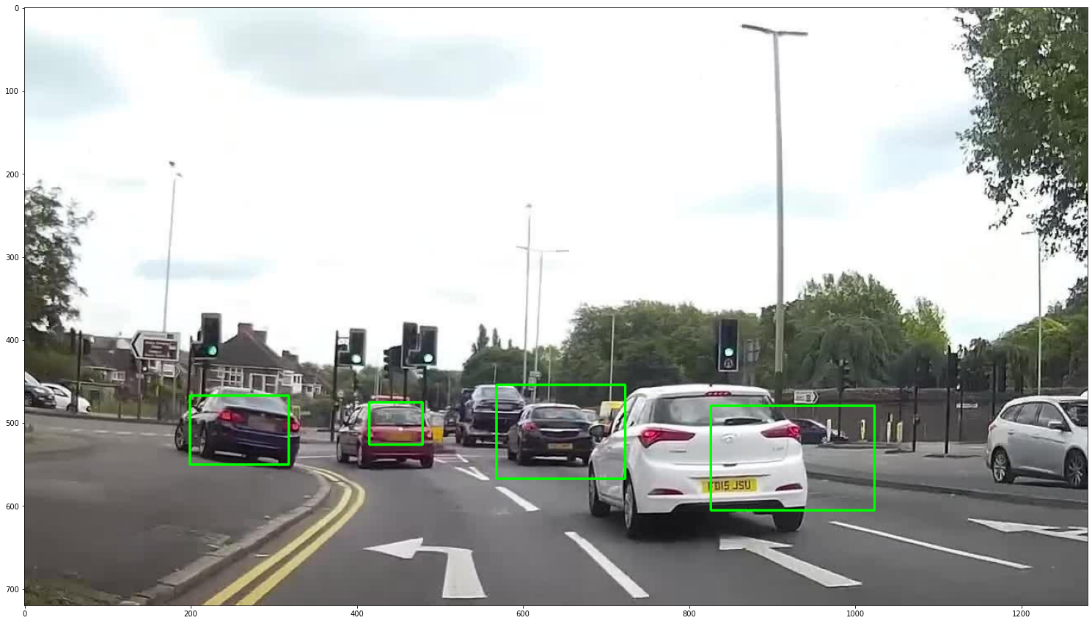
Once the predictions are made, the splits of the image are combined into single image. It can be seen that more than one grid identified the bounding box for the single car. This resulted in multiple bounding boxes for one car which needed to be merged into one. Merge\_overlaps function merges all overlapping bounding boxes in array of bounding boxes if the overlap is more than the overlap defined earlier i.e., 0.45.

The image before merging looked like this:



*Fig 5.3.1: Image with redundant overlapping bounding boxes*

With merging function, it resulted into this:



*Fig 5.3.2: Image after merging overlapping bounding boxes*

# Output video stream

To test the model, an unseen video stream is given as an input, it is then fed to the YOLO algorithm, which means that each frame of the video is sectioned according to the overlap value given, and then fed to the Convolutional neural network, where the model predicts and outputs bounding boxes for cars present in each section of the image. The grid sections of the images with prediction are then combined back in the single frame. All the frames with predictions made in run-time may be saved in an output MP4 file.

# Conclusions

In this project, Car detection with 99% accuracy and localization with 80% IoU was achieved. The data was collected from scratch and was region specific. YOLO algorithm was designed to make predictions in real-time.

The algorithm was able to make accurate predictions of cars. However, there were few wrong predictions, long vehicles such as trucks were not detected and road signs were sometimes wrongly detected as car. This is because, the data used was limited. With more data, other vehicles such as bicycle, motorcycle and buses, the project can be further improved and used as crucial part of self-driving vehicle along with other specialized autonomous functions.

# Acknowledgements

The YOLO algorithm part of the project is inspired by the guided project “Object localization with TensorFlow” by Amit Yadav, Coursera Project Network [4] and Convolutional neural network is inspired by Deep learning specialization on Coursera by Andrew Ng [5].

# References and Bibliography

[1] https://arxiv.org/pdf/1506.02640.pdf

[2] http://ai.stanford.edu/~jkrause/cars/car\_dataset.html

[3] https://www.epfl.ch/labs/cvlab/data/data-pose-index-php/

[4] https://www.coursera.org/projects/object-localization-tensorflow

**Link to GitHub Project repository:**

<https://github.com/MariaMumtaz/Uni-Bern/tree/main/CAS%20ADS/Final%20Project>

**Other useful links:**

<https://github.com/TheClub4/car-detection-yolov2/blob/master/Car%20detection%20for%20Autonomous%20Driving/Autonomous_driving_application_Car_detection_v3a.ipynb>

<https://towardsdatascience.com/deep-learning-for-self-driving-cars-7f198ef4cfa2>