Spring 5-22-2019

Over speed detection using Artificial Intelligence

Samkit Patira
San Jose State University

Over speed detection using Artificial Intelligence

A Project Presented to

The Faculty of Department of Computer Science

San Jose State University

In Partial Fulfilment of
the Requirements for the Degree
Master of Science

By

Samkit Patira

May 2019

The Designated Project Committee Approves the Master's Project Titled

Overspeed detection using Artificial Intelligence

By

Samkit Patira

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2019

ABSTRACT

Real Time Overspeed Detection using Artificial Intelligence

By Samkit Patira

Over speeding is one of the most common traffic violations. Around 41 million people are issued speeding tickets each year in USA i.e one every second. Existing approaches to detect over-speeding are not scalable and require manual efforts. In this project, by the use of computer vision and artificial intelligence, I have tried to detect over speeding and report the violation to the law enforcement officer. It was observed that when predictions are done using YoloV3, we get the best results.

ACKNOWLEDGEMENTS

The past two years at San Jose State University were great. I learned a lot from professors and friends.

This project won't have been possible without the guidance and advise of Prof. Heller. I would like to express my special thanks to Dr. Philip Heller for his patience and his brilliant guidance throughout the duration of my project. My completion of the project would not be accomplished without his support.

I am also thankful to my committee members, Dr. Robert Chun and Vyas Bhagwat for providing their valuable feedback and guidance.

And finally, I would like to thank my parents and friends for helping me in gathering the data and guiding me in making this project despite of their busy schedules.

CONTENTS

CHAPTER 1

Introduction

1.1 Problem Statement

Speeding is one of the biggest traffic violations. It endangers everyone on the road. According to NHTSA [1], in year 2017 speeding killed 9717 people in the USA. Some of the consequences of over speeding are loss of vehicle control, increased stopping distance, economic losses, increased fuel consumption and loss of lives. As crash speeds get very high, airbags and seat belts may not work as well to protect the passengers from the collision. Over speeding costs of billions of dollars to the country's economy [1].

For example, Orland park, Illinois with population of 60000, police issued 26,821 citations. Out of those 4732 were for speeding. Law enforcement officers spend a significant amount of time in catching over speed violators. Also, a very small fraction of violators is caught by the existing system.

The aim of this project is to design an automated over speed detection system that would notify traffic police with the details of an over speeding vehicle. Once the details are sent to the regulatory authority, the driver can be charged for over speeding. The main objective of the project is to eliminate the manpower needed by the existing systems. Currently, officer has to hold the speeding gun to measure the speed of the vehicles. It should be noted that currently, only 5% of the violators get speeding tickets. This is because the number of vehicles is much more compared to law enforcement officers. Some of the cities like San Francisco, Stockton, Gilbert, etc. have less than 10 officers per 10K [2] population. In this situation, it’s very difficult to catch all the traffic violators.

In the proposed system, dashcam could be placed inside the vehicle which will read the traffic speed signs posted on the road. This speed would be compared with the actual speed of the vehicle using accelerometer present inside the mobile phone and if found very high it will notify the law enforcement agencies with the details like the place, driver, photo, etc. This will help in eliminating the manual efforts required. Also, this system can be used in remote places where there are no officers present.

The project uses deep learning technique that needs a significant amount of computation for training. Thus, we decided to reduce the scope of the project just to train a model that can read different traffic speed signs.

1.2 Related Work

Related work in over speed detection mainly involves the usage of devices that are not installed inside the vehicles. Pacing [3] is quite a common technique where cops drive behind the vehicle and accelerate until the speed of officer's vehicle matches the speed of the vehicle of the suspect and until the distance between both the cars is constant. Then by looking at the speedometer, the officer can determine the target vehicle speed. But this technique is prone to human errors. Many time officers may read the wrong reading or may read the reading while he is accelerating his vehicle.

Another such speed detection system involves the use of a speed gun which is placed in the direction of moving a vehicle and is based on the radio frequency or laser. Radio wave signal [4] is sent and then waiting for it to be reflected by the car. Using the timing of the wave signal, the speed of the vehicle is detected. This involves manual efforts with a person holding the gun.

VASCAR (Visual Average Speed Computer and Recorder) is a small processor that is placed inside the officer's car. An officer passes the vehicle at a very high speed than the speed of the vehicle of the suspect. Officer would be waiting few miles away and by doing some calculations, it's possible to calculate the speed of the suspect's vehicle. This is the type of speed trap.

Many research studies have been conducted in the field of vehicle speed detection. One such technique [5] proposed involving the comparison of the vehicle position between the current frame and the previous frame from video captured with a stationery camera. Another similar technique uses video surveillance system [6]. Frame of the camera covers specific area, then calculates the speed of the car on basis of the time the car was in that area.

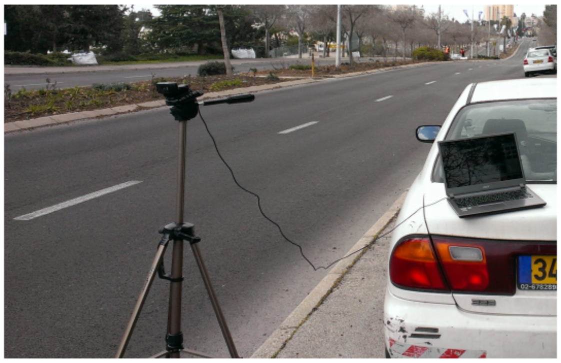


Figure 1.2 Speed detection using video surveillance

Moreover, most of the existing solutions need manual efforts or some kind of infrastructure setup. Such systems are either expensive or labor intensive and rely on outdated technology. All the solutions seen so far are based on physical methods and this is because of the limited computation capabilities in the ’90s and 2000s. Recent advances in high-performance computation and artificial intelligence can overcome these drawbacks. This will also ensure that all the over speeding violators are caught and are reported.

1.3 Contribution

The primary contribution of this project is to design a system that can read traffic speed signs using computer vision and artificial intelligence. I have tried object detection algorithm (Yolo) and also experimented with transfer learning so as to use already existing trained model. It was observed that the by the use of transfer learning, we can significantly improve the effectiveness of the trained model.

The project report is organized into chapters as follows: Chapter 2 defines common concepts and terms used in deep learning. Chapter 3 defines various data augmentation techniques used for training data. Chapter 4 describes algorithm used and experiments. Chapter 5 has the conclusion and future scope.

CHAPTER 2

Background

In this chapter, I provide background information related to deep learning and other related concepts. This background information is crucial to understand the algorithms and techniques used in the project.

2.1 Deep Learning

Artificial intelligence can be defined as a science or engineering of making machines smart and intelligent. Deep Learning is a part of artificial intelligence which primarily deals with the neural networks. Neural networks try to learn from the training data without being programmed explicitly. They have a variety of applications in domains like natural language processing, image processing, object detection, classification, speech recognition, text processing, and summarization, etc.

The basic building block of a neural network is called a neuron. A neuron can be thought as of biological neurons present inside the human brain. The neural network may have millions of the neurons. Each neuron is connected to other neurons by means of edges. They receive inputs from other elements or neurons and then the inputs are multiplied by the weights and result is then transformed by some mathematical function into the output.

Neural networks have one layer of input, multiple hidden layers, and one output layer.

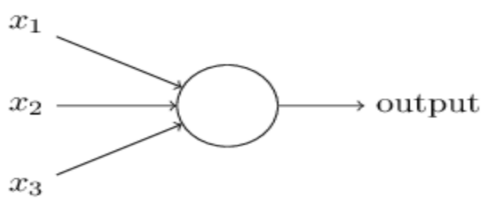


Figure 2.1 Neuron

In the above figure, neuron has 3 inputs but can have more or less inputs. Each input is associated with real numbers called weights.

2.2 Fully connected neural networks

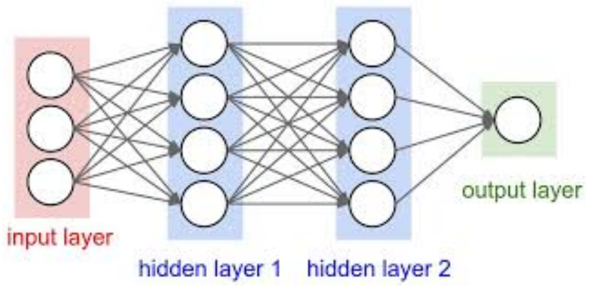


Figure 2.2 Fully connected layer

In fully connected neural networks, each neuron is connected to every other neuron of the next layer. Input and output length of the data is fixed.

Training:

Most of the time for training, we use labeled training data. This means that training data has known output. Functions in neural networks can be classified into a sequence of linear and non-linear functions.

$$x \rightarrow f(x)$$

$x \rightarrow W_1 \rightarrow h_1(W_1x) \rightarrow W_2h_1(W_1x) \rightarrow \;.... \sim f(x)$

During the training phase, we try to adjust and set the value of weight matrices. Weights during the start of training are randomly initialized. These weights are later used in the inference phase. The main goal of training is to start with the model with very low accuracy and eventually have a network with high accuracy. We need non-linear functions to make the network more powerful and means to introduce differentiable. Some of the key terms used during training:

Sigmoid Function: It is very similar to the step function in which output is between 0 and 1. It is used for models where

Activation: $sig(x) = \frac{1}{1+e^{-x}}$ Derivative: $sig'(x) = sig(x).(1 - sig(x))$ Range: [0:1]

Tanh Function:

Activation: $tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ Derivative: tanh\`(x) = 1 – $\tanh(x)^2$ Range: [-1:1]

ReLu Function: Also called as a rectified linear unit. It outputs the same input for all positive values other zero.

Activation: $ReLU(x) \begin{cases} x & x > 0 \\ 0 & x < 0 \end{cases}$ Derivative: $ReLU(x) \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$ Range: [0:∝]

Leaky ReLu Function: It is very similar to ReLU except it has a scaling factor.

Activation: $\mathrm{L}ReLU(x) \begin{cases} x & x > 0 \\ -\alpha x & < 0 \end{cases}$ Derivative: $\mathbf{L}ReLU(x) \begin{cases} 1 & x > 0 \\ -\alpha & < 0 \end{cases}$ Range: [-∝:∝]

Cross Entropy Loss: Cross entropy loss functions is mainly used in classification problems.

Cross Entropy Loss = $-(y_i . \log(y`_i) + (1-y_i). \log(1-y_i))$

During the forward propagation, it takes inputs from the previous layer, and then each node computes z = Wx + b where W is weight, x is the input and b is the bias [N3]. After this, some activation function would be applied to the z. Different layers can have different activation functions. At the end of this process, a loss would be calculated using functions like cross-entropy loss.

After calculating the loss, back-propagation is carried so as to update the weights. It starts from reverse topological order to compute the derivative of the node with respect to the previous node. This will change the values of parameters.

Algorithm:

Step 1: Calculate the forward phase for each training data sample.

Step 2: Calculate the backward phase.

Step 3: Combine the individual gradients.

Step 4: Update the weights**.**

Experiment 1: A simple program was implemented to gain a better understanding of the forward and backward propagation. This program was written from scratch in python without using any library. I had used sigmoid as an activation function and mean square error as a loss function. Training set was XOR table. It was trained for 1250 iterations.

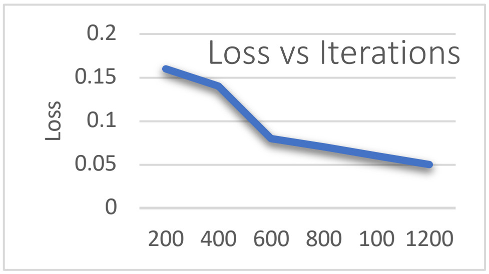


Figure 2.3 Loss vs Iterations

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.4 Input and Output

During the evaluation phase, model performed very well and predicted correct output for 12 entries out of 14 entries.

2.3 Convolution Neural Networks

Convolutional neural networks are inspired by the brain [7]. They have a different architecture than the previously seen fully connected neural networks. All the layers in CNN used for image processing have three dimensions and input and output is 3-dimensional. Unlike fully connected neural network, only some neurons present in a layer are connected to the next layer. They are mainly used in applications related to image classification and recognition.

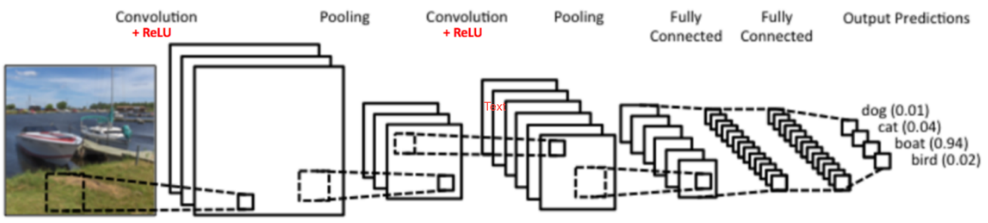


Figure 2.5 Convolutional Neural network

Convolutions are capable of extracting different features from an input image. It preserves the spatial relationship between pixels by learning features using small windows of input data. The motivation behind using CNN over a simple neural network is that they are capable to learn relevant features at different levels which was very similar to the human brain. A filter slides over the input to produce a feature map. Filters are associated with weights. We will get more feature maps if we use more number of filters. During the training phase, CNN learns or adjusts the values of weights.

One of the reasons for using CNN over a simple neural network is weight sharing in the CNN. CNN is more efficient in terms of memory, complexity and, computation. Consider we have 5 filters of size 3x3 in CNN. The number of parameters required would be 3*3*5 = 45 parameters. In case of traditional a neural network, we will require (45*h*w) parameters where h and w and are height and width of the image. Also, it is possible to do transfer learning by using CNN. Transfer learning is a machine learning technique where a model trained for some particular task can be re-used to perform a similar task. This helps in reducing the training costs.

Some of the layers used in CNN:

Convolution Layer: This layer [N12] does dot product between the input tensor and weight matrix. The weight matrix is also called as a kernel. A kernel is generally square in shape and is spatially smaller than input tensor. A kernel can be imagined as a cube which has more depth in comparison to other dimensions. Kernel slides over the image and each kernel act as a feature detector.

Max pool Layer: In max pool layer we move the window (size can be 2x2, 3x3, 4x4, etc.) over the image and take maximum value from the window as an output. Max pool reduces the number of parameters. It is down sampling layer which reduces the size of an output.

Average pool Layer: Average pool is very similar to the max pool layer. Instead of taking maximum value from the window, it takes the average of all the values present in the window.
It is also a down sampling layer, but it preserves the input information.

Up sample Layer: Up sample is a deconvolution layer which increases the size of output. It uses interpolation techniques like bilinear interpolation to produce the output.

Dropout: Drop out is used to prevent over-fitting. It simply ignores some units during the training. It makes the model more robust but takes more iterations for the model to converge.

Gradient Descent: Gradient descent can be imagined as a ball moving down the hill. The aim is to find the deepest point among all the hills. It can be seen from the picture. Gradient descent measures the change in the weights with respect to change in the difference of actual output and predicted output or error.

2.4 EXPERIMENT 2:

TRAFFIC SIGN CLASSIFIER.

The aim of this experiment was to become familiar with the convolution neural networks and Keras library. In this experiment, I have classified traffic signs by using convolutional neural networks. This can have application in the autonomous vehicles. Dataset was taken from German Traffic Signs [8]. Dataset consisted of around 20000 images belonging to 43 different categories. These categories included traffic signs like the left turn, right turn, stop sign, one way, etc. Images just had traffic sign and nothing else in the background.

Images present in the dataset were of different sizes. As a neural network need a fix size input, images were resized to 48x48. Later, images were converted into greyscale so as to reduce the input tensor size. Data was split in the ratio of 4:1 i.e 16000 images were used for training and 4000 images were used for validation.

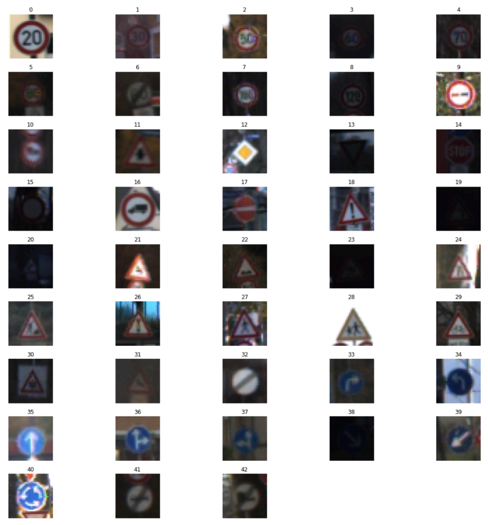
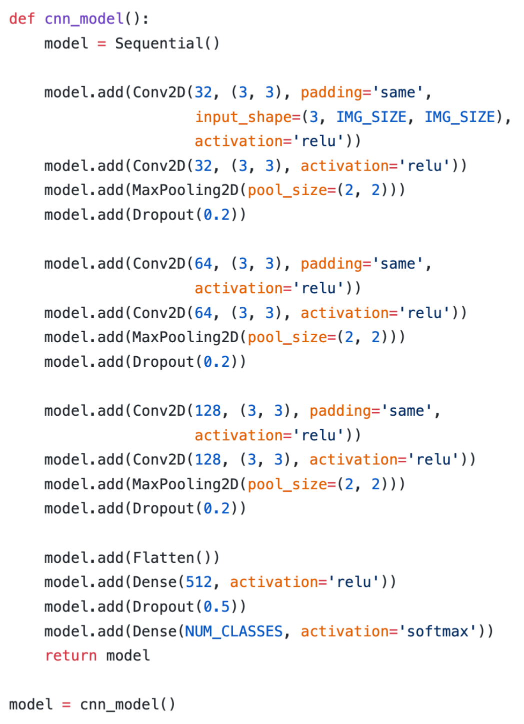


Figure 2.6 Snapshot of images

For doing this, I have used pandas, NumPy, SkImage, SkLearn, h5py, glob, Keras, Matplot lib, OpenCV and python. Keras is an open source library which is developed by Google. It uses Tensorflow at the backend and was installed using pip command. Keras provides support for various neural network models and is more user-friendly compared to other deep learning libraries. It supports training on GPU (Graphical Processing Unit) and TPU (Tensor Processing Unit). OpenCV is an open source library developed by Intel. It is used for computer vision and I have used it for various image transformation operations. Skimage provides support for image processing and has inbuilt functionality for geometric transformations, morphing, segmentation, color changes, etc. 9

I have used CNN architecture to build the model. This is because our aim here is to do image classification. As seen previously CNN is a much better option than using multi perceptron network.



```
def cnn_model():
    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same',
                     input_shape=(3, IMG_SIZE, IMG_SIZE),
                     activation='relu'))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(64, (3, 3), padding='same',
                     activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(128, (3, 3), padding='same',
                     activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(NUM_CLASSES, activation='softmax'))
    return model

model = cnn_model()
```

Figure 2.7 Snapshot of model used using Keras

I have used convolution 3x3 with different channels length, max pool, and dropout. To introduce non-linearity, I have used ReLU as an activation function. Below I have explained some of the hyperparameters that I have tuned.

I had initially used gradient descent [9]. In this kind of gradient descent, we calculate an error for each sample but updates the model only after all the samples are evaluated. This is computationally very efficient but sometimes can lead to wrong convergence. Later, I used stochastic gradient descent [10] which updates the model after each training sample. This is very slow, but it leads to the right state of convergence. It should be noted that stochastic gradient descent is computationally more costly. During inference, it was observed that accuracy was improved by 14% when a stochastic gradient was used.

I tried experimenting with mean square error and cross-entropy as a loss function. Mean square takes the square of difference between the actual output and predicted output whereas cross entropy is the logarithmic function. It turned out that cross entropy was a better option. This is because mean square error is suited for regression whereas cross-entropy is well for classification problems.

Learning rate used in the model was dynamic. Learning rate determines how fast the model would be trained. It is also called as step size and it governs how weights are updated during the training. Range of learning rate is between 0 to 1. If the learning rate is low, it will take more time for the model to converge. Initially, learning rate was set as 0.01. Learning rate used is given as

LR = LR * (0.1 * (epoch/10))

Batch size refers to the number of samples present in a single batch. I have taken batch size as 30.

Epochs refer to the number of iterations of training data. The model was trained for different values of the epochs but best results were achieved when the number of epochs was 30.

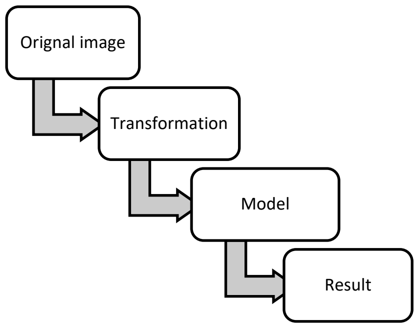


Figure 2.8 Data Flow

The model was trained on MacBook Pro which had 16 GB of RAM, 2.9 GHz Intel processor and Radeon Pro 4 GB graphics card. It took a little more than 9 hours to train the model. To improve the model, I had used k–fold cross validation where k was 5.

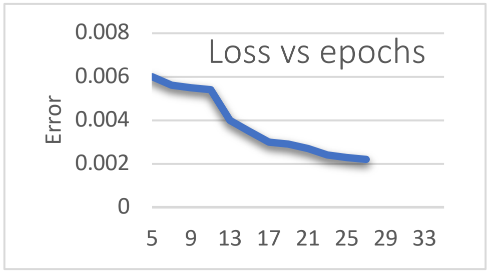


Figure 2.8 Graph of Error vs Epochs

Observation: Over the whole course of training, it was observed that loss was decreasing with increase in the epochs. Initially, loss was very high and was decreasing non-linearly. After 30 epochs there was a negligible change in the loss and therefore training the model for 30 epochs would be the most efficient. The best accuracy I was able to achieve was 96% on the validation set. It takes ~2 seconds for inference of single image using the trained model on the same machine. For testing, I had clicked manually 25 images and these images were cropped so just as to get the traffic sign. When testing was done over these images, 21 images predicted the correct categories whereas 4 images predicted the invalid categories.

CHAPTER 3

Dataset

3.1 DATA COLLECTION

For training data most of the images I have collected using google map’s street view. Some of the images were also taken from videos available on YouTube. By doing so I was able to generate the training dataset of around 530 images. After getting all the images, it is important to create labels. Bounding boxes were drawn over the speed limit sign and top left and bottom right coordinates of the boxes were stored in the .txt file. This file also contains information about the speed limit (category) and the size of the image.

3.2 DATA AUGMENTATION

Training deep learning model on a large number of training samples is always helpful. It makes the model more robust and prevents overfitting. By exposing the model to the different set of images with different variations and environment, it helps to achieve more accuracy [13]. I did data augmentation to generate more training data. Data augmentation is a technique to generate automatically more images by doing some transformations on images.

I have used several data augmentation techniques. Scaling [12] is done to make the image dataset diverse. Sometimes object to be detected in the image can be a very small or could be very large. Rotation rotates the image on x or y axis. Images captured from the camera have a various angle of rotations. I have added generated images by adding the Gaussian noise [N11]. This will produce images with different lighting conditions. It will stimulate the practical scenario of images taken by device for different times of the day.

In contrast stretching technique we change the contrast of an image by altering range of intensity levels. Histogram equalization is another such technique based on the histogram. We first find the histogram and then normalize it over the probability distribution.

Tools used to achieve image data augmentation consists of MATLAB, python, SkiImage, NumPy and OpenCV (Computer Vision library). All the tools used except MATLAB are open source. By doing so, I was able to create a database of 1930 images from the original 530 images. All the images in the dataset had speed signs. To prevent overfitting of the model and make the model more robust, I added some negative images. Negative images consisted of invalid categories and some images had no traffic speed signs. Final dataset consisted of 2078 images.

CHAPTER 4

4.1 IMAGE CLASSIFICATION VS OBJECT DETECTION

Previously seen CNN for traffic light does image classification. Classification can be defined as a process of categorizing an image into one of the pre-defined group of classes. Classification models try to find the most dominating object (area wise) in the image and classify it [14]. The dominating portion gets the highest score or priority. It does not consider transformation properties like scaling, location, color changes, rotation, etc. Let's say we have trained our classification model for the stop sign and the car. If the car is present in the majority portion of the image, the image would be classified as a car and will ignore the stop sign. It should be noted that in our current use case, speed sign will have area less than 5% of total image area.

Image classification also does not care about the location of the object in the frame. Sometimes we would like to know the location and number of cars or the number of known objects present in the image. Solution for this is to use object detection model. Classification differentiates two objects whereas object detection tries to find particular features of objects in the images.

Object detection can be defined as a combination of classification and localization. Object detection considers all the objects and their location. It is programmed to categorize each known present object and give details about the location. Well, known application of an object detection system is Amazon Go stores.

4.2 You Only Look Once (YoLo)

Humans look at the image frame and easily detect what different objects are present in the image. Traditional computer vision techniques are able to detect the objects in the image only if the image has a majority portion of that object and nothing else in the background. They use properties of objects like image color, shape, etc. These algorithms work for a constrained environment and fail if the images have variations. They may be able to detect a single large-sized soccer ball in the image very accurately but won't work if we wish to detect many small size soccer balls of different variations present in the image.

Under this kind of situations, Yolo comes to rescue. Yolo is an object detection system and is able to detect a wide variety of the objects present in the real time. Because of its unified architecture, it is extremely fast in detection. Existing deep learning classifier models like Regions with Convolutional Neural Network [15] (R-CNN) are capable of performing object detection. For object detection, these systems use sliding window i.e they consider a classifier for every object to be detected and slide it over all possible window locations on the image. Once the classification is done, post-processing is carried out and bounding boxes are redefined. Post processing is also done to remove duplicated detections. This increases the complexity, computation and time it takes for the detection.

The basic motivation for using Yolo is the speed and complexity of the system. Instead of sliding over an image many times, Yolo [16] only looks once and detects all the objects present in the image. Yolo defines the detection problem as a regression problem and uses features from an entire image at the time of training. Unlike RCNN, it looks at the entire image during the time of training and testing That means Yolo predicts all the different object categories present on an image simultaneously.

4.3 MODEL

Before understanding the working of Yolo, let us understand input to the model. In my current experiment, input to the neural network consists of the image, type or category of the speed sign present in the image and bounding box details. Bounding box is a square drawn over the circular speed signboard. A bounding box has details like normalized x and y position and also normalized the height and width of the box. All values present in the bounding box have range from 0 to 1. I have stored all these details in the form of text file. If the image has two different speed signs (rare case) then there will be two text files. The training dataset consists of the speed signs of 7 different categories.

Unlike fast R-CNN [15] which produces the output by performing prediction multiple times for the same image, Yolo passes the image only once and produces the output. Yolo divides the input into grid cells X*Y. Each grid cell is associated with n bounding boxes and would predict the following parameters:

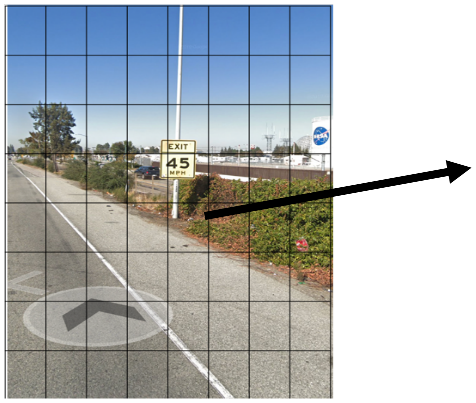
$y = (p_c, b_x, b_y, b_w, b_h, c)$

where p_c = *confidence score of the bounding box*

b_x, b_y = *location of the box*

b_w, b_h = *dimensions of the box*

c = *class predicted.*

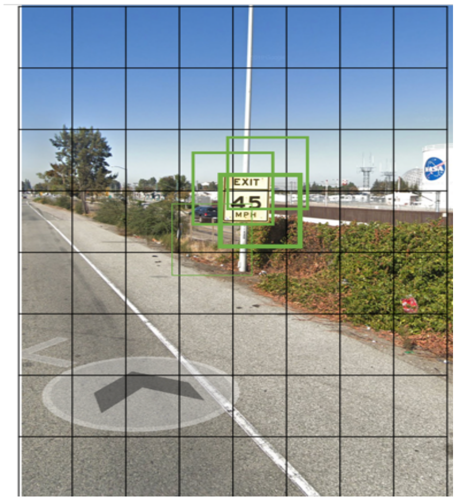


	p_c	b_x	b_y	b_w	b_h	c_1	*cn*
Box1							
Box2							

Figure 4.1 Division of image into grids

4.4 NON-MAX SUPPRESSION ALGORITHM:

A number of bounding box produced by the model would be very high. This is because if there are m*m grids and each grid is producing n boxes then total bounding boxes would be m*m*n. Only very few numbers of boxes would have actual object present in it whereas other boxes would be empty. One or more grid cells may predict the same object and will have different bounding boxes.



Here in the picture, it can be seen that many grid cells predict the same speed sign with different confidence score. Box with the thickest outline border has the highest confidence score.

Figure 4.2 Multiple bounding boxes for same object

To overcome this, Yolo uses non-maximal suppression [N13].

Algorithm:

Step 1: Sort all the bounding boxes where objects are present by the p_c confidence score.

Step2: Start from first box and ignore the next bounding box if it has the same object and IoU (Intersection over Union) > 0.5.

4.5 NEURAL NETWORK:

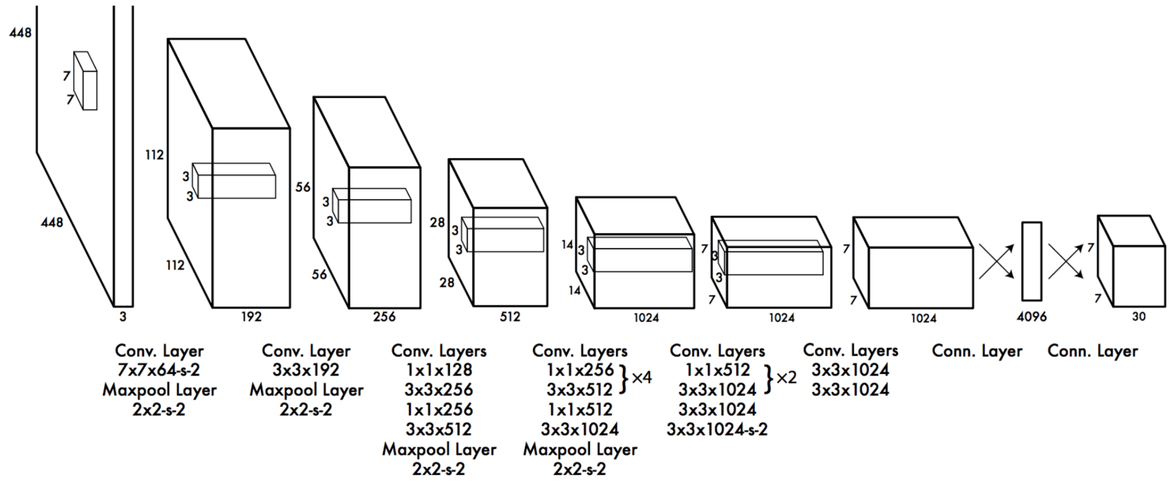


Figure 4.3 Neural network

Neural network of Yolo is very similar to GoogLeNet [17]. GoogLeNet has 22 convolutional layers whereas Yolo has 24 convolutional layers. Convolution layer in Yolo is followed by the two fully connected layers. Size of the kernel used in convolution layers is 3*3 or 5*5. This causes the weights of convolution layer to be less dependent on the location of the objects in the image and weights do not have spatial information. Fully connected layer takes into consideration spatial far-away features.

4.6 EXPERIMENTS:

All the experiments were carried on the Google cloud platform. Compute Engine had NVIDIA Tesla P100 processor with 4 GPUs. P100 is based on Pascal architecture. It's capable of performing 21 TeraFLOPs of performance. For testing I have used 34 images. Some part of the code is adapted from the [20].

a) Loss function:

Yolo uses mean square loss function [18]. Outcome of Yolo is N – vector where

N = (m*n) * (B + 5) * C

*where m*n are number of grid cells*

B is number of bounding boxes per grid. (B = 2)

C is the number of categories (C = 8)

5 is for the b_x, b_y, b_w, b_h, p_c. This is explained in previous section.

$$
\begin{aligned}
&\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2] \\
&+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2] \\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2 \\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2
\end{aligned}
$$

Here λ is the constant.

When the model was trained from scratch using above loss function, it made correct bounding boxes for 29 images. Model took 234 minutes to train.

b) Transfer Learning:

Transfer learning is the process of training the model with the help of the other model which is already trained for some dataset. Transfer learning is done to reduce the time required for training. Here I did transfer learning using Yolo trained for COCO dataset. Weights were obtained from the official Yolo website. Model has the same number of neural network layers as the one in previous experiment. Training was faster as it took 190 minutes to train the network.

c) Changing the number of convolution layers:

In this experiment, I changed the number of convolution layers. Initially, I reduced the number of convolution layers. I tried with 14, 16 and 18 number of layers. It was observed that the more the number of layers, better was the accuracy. When the number of layers was 14, model was doing false predictions and drawing the bounding boxes at the random places. Accuracy improved when the number of layers was 18 but still, it was not satisfactory.

Next, I tried to increase the number of convolution layers to 28 from the original 22. Training the network with 28 layers took around 310 minutes. Accuracy of the prediction of the model was almost as same as the original model.

It can be concluded that increasing the number of layers, does not help us to improve the accuracy.

d) Different batch sizes:

I tried training the model with different batch sizes. Batch size has a direct effect on the convergence [18]. It was observed that greater the batch size, higher would be the accuracy. Increasing batch size beyond 64 did not have a drastic change in the performance. Optimal batch size for the current was found to be 64. With the increase in the batch size, computation cost also increased.

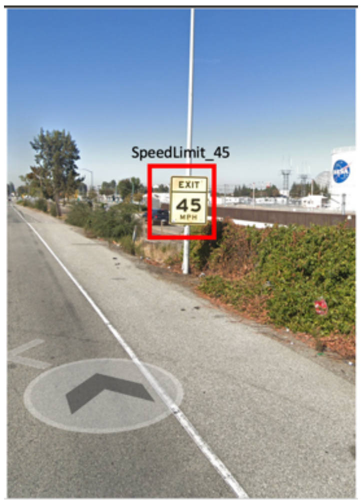


Figure 4.4 Final output

4.7 DECISION FACTOR

During the inference, it was seen that for the same speed sign the machine learning model was giving different outputs. Consider that we are using a camera for detection of speed sign is producing 30 frames per seconds. The time for which a driver or camera is able to see the speed sign would depend on the speed of the car. We will calculate the time when the speed of the car is 60 miles per hour or 96 kilometers per hour and the driver is able to see the speed sign from 50 meters away.

$$t = \frac{distance}{speed} = \frac{0.05}{96} = 1.8\ secs$$

That means the camera will produce ~50 frames that will have speed sign in it. Assuming our model is able to process images in real-time, the model will produce 50 outputs. These outputs may not have the same predictions. Under such circumstances, we can use different approaches to predict the correct speed sign reading:

a. Based on frequency: Category having the maximum count of output from the model would be the final speed. If more than one category has the same maximum count, we can consider the average of such speeds.

b. Based on the car speed: We can use the speed of the car to find out the final speed. The predicted speed which is having a minimum difference with actual car speed would be the right value. Here we are assuming that the driver is not over-speeding under-speeding.

c. Gaussian Distribution: Normal distribution has the bell curve and probability of speed sign can be given as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where, μ = average of speed
σ = variation

FUTURE WORK AND CONCLUSION

The goal of this project is to detect over-speeding by placing the device on the car dashboard and report the violation to the officers. For this purpose, we identified and explored computer vision technique using deep learning. Yolo is the best-suited model as it has the capability to add more categories. We were able to achieve accuracy of around 90% for the images in day time but accuracy reduces if it is night time. By adding more nighttime images to the training set, this can be improved. Using our designed system, it's possible to catch over speeding violators in the remote areas where cops are not present.

Although we were able to train the model, it needs a very high amount of computation for inference. One way to improve our model would be to optimize the neural network used, so it needs less computation. We can also try out different networks like Deep Multibox, OverFeat, Multi grasp, etc.

If we are able to create a model which can run on a normal phone which has limited resources, we can create a phone application. By doing so, we will not need any external device. Further, it is possible to detect a traffic light violation using the same model. The system would identify the red and green light and then check if the car is stationary or not. But this will require a change in the decision logic. It is possible that our model detects red light falsely reports while the car is far away from the traffic light and is moving. I have collected the dataset which has green, yellow and red traffic light images.

REFERENCES

[1] -amy.lee.ctr@dot.gov. ‘Speeding’. *NHTSA*, 9 Sept. 2016, Retrieved from https://www.nhtsa.gov/risky-driving/speeding.

[2] -*Law Enforcement Officers Per Capita for Cities, Local Departments*. Retrieved from https://www.governing.com/gov-data/safety-justice/law-enforcement-police-department-employee-totals-for-cities.html.

[3] -A. G. Rad, A. Dehghani, and M. R. Karim. “*Vehicle speed detection in video image sequences using cvs method*”, *International Journal of Physical Sciences*, 5(17):2555–2563, 2010.

[4] -Pornpanomchai, C., & Kongkittisan, K. (2009). Vehicle speed detection system. *2009 IEEE International Conference on Signal and Image Processing Applications*. doi:10.1109/icsipa.2009.5478629

[5] -Ginzburg, Chaim, et al. “*A Cheap System for Vehicle Speed Detection”*, *arxiv.org*, https://arxiv.org/abs/1501.06751v1. Jan. 2015

[6] -Jozef Gerát, Dominik Sopiak, Miloš Oravec, Jarmila Pavlovicová, "*Vehicle speed detection from camera stream using image processing methods*", *ELMAR 2017 International Symposium*, pp. 201-204, 2017.

[7] -Huang, Xiaobo. *“Convolutional Neural Networks In Convolution”*. Oct. 2018. arxiv.org,

[8] -“Dataset,” *German Traffic Sign Benchmarks*. [Online]. Retrieved from http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset. [Accessed: 12-March-2019].

[9] -L. Simon S., L. Jason D., Wang, Liwei, Zhai, and Xiyu, “*Gradient Descent Finds Global Minima of Deep Neural Networks*,” *arXiv.org*, 04-Feb-2019.

[10] -Ting, Li, Guiying, Tang, and Ke, “*Stochastic Gradient Descent for Nonconvex Learning without Bounded Gradient Assumptions,*” *arXiv.org*, 10-Mar-2019

[11] -S. Lau and S. Lau, “Image Augmentation for Deep Learning,” *Towards Data Science*, 10-Jul-2017. [Online]. Available: https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2.

[12] -B. Raj and B. Raj, “Data Augmentation | How to use Deep Learning when you have Limited Data - Part 2,” *Medium*, 11-Apr-2018. [Online]. Retrieved from https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

[13] -Luis, Wang, and Jason, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning,” *arXiv.org*, 13-Dec-2017. Available: https://arxiv.org/abs/1712.04621.

[14] -Girshick et al., "*Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." arXiv.org*

[15] -Parthasarathy, Dhruv. '*A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN*'. *Athelas*, 22 Apr. 2017

[16] -Redmon, Joseph, and Ali Farhadi. '*YOLO9000: Better, Faster, Stronger'*. 2017 *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), IEEE, 2017, pp. 6517–25. Crossref, doi:10.1109/CVPR.2017.690.

[17] -A Krizhevsky, Sutskever and G. Hinton, "*ImageNet Classification with Deep convolutional Neural Networks*", Retrieved from http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

[18] -J. Hui and J. Hui, "Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3," *Medium*, 18-Mar-2018. [Online]. Retrieved from https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088.

[19] -Playing around with RCNN, State of the Art Object Detector Retrieved from https://cs.stanford.edu/people/karpathy/rcnn/

[20] -*Convolutional Neural Networks. Contribute to Pjreddie/ GitHub*, Retrieved from https://github.com/pjreddie/darknet.

[21] -Mehta, Rakesh, and Cemalettin Ozturk. 'Object Detection at 200 Frames Per Second'. *ArXiv:1805.06361 [Cs]*, May 2018. *arXiv.org*, Retrieved from http://arxiv.org/abs/1805.06361.