Object Classification and Road Side Signs Identification for Autonomous Vehicles

Project ID: 14271

B.Tech. Project Report
submitted for fulfillment of
the requirements for the
Degree of Bachelor of Technology
Under Biju Pattnaik University of Technology

Submitted By

Ashirbad Subudhi **Roll No. CSE201540397**

2018- 2019

Under the guidance of

Dr. K. Hemant K. Reddy

NATIONAL INSTITUTE OF SCIENCE & TECHNOLOGY

Palur Hills, Berhampur, Odisha – 761008, India

ABSTRACT

Identification of any object and traffic road sign is an important job for any autonomous vehicle. An automated driving system is a complex combination of various components that can be defined as systems where perception, decision making, and operation of the automobile are performed by electronics and machinery instead of a human driver, and as an introduction of automation into road traffic. This includes handling of the vehicle, destination, as well as awareness of surroundings. While the automated system has control over the vehicle, it allows the human operator to leave all responsibilities to the system.

The main objectives of this project are - Object identification and Traffic road sign identification.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank all those individuals whose invaluable contribution in a direct or indirect manner has gone into the making of this project a tremendous learning experience for me.

It is my proud privilege to epitomize my deepest sense of gratitude and indebtedness to my faculty guide, **Dr. K. Hemant K. Reddy** for his valuable guidance, keen and sustained interest, intuitive ideas and persistent endeavour. His guidance and inspirations enabled me to complete my report work successfully.

I give my sincere thanks to **Mr. Debananda Kanhar, CSE Project Coordinator and Dr. Sandipan Malik, B.Tech. Project Coordinator** for giving me the opportunity and motivating me to complete the project within stipulated period of time and providing a helping environment.

I acknowledge with immense pleasure the sustained interest, encouraging attitude and constant inspiration rendered by **Prof. Sangram Mudali** (Director), **Prof. (Dr.) Ajit Kumar Panda** (Dean), **Prof. (Dr.) Shom Prasad Dash** (HOD, School of Computer Science) N.I.S.T. Their continued drive for better quality in everything that happens at N.I.S.T. and selfless inspiration has always helped us to move ahead.

Ashirbad Subudhi

Roll No. 201540397

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Intelligent transportation systems are arguably the most anticipated smart city services. ITSs are built on the premise of endowing vehicles and transportation infrastructure with connectivity, sensing, and autonomy, so as to provide safer road travel and ensure effective transportation. To enable this ITS vision, there is a need to equip vehicles and transportation infrastructure with smart sensors that can collect and process a heterogeneous set of data on each vehicle, its passengers, and its environment. This information collection must be done at ultra-low latency and in real time, since ITSs must support autonomous features, such as self-driving vehicles. Indeed, monitoring and managing the operation of an ITS requires a reliable communication and data analytics infrastructure to transmit and process the various smart sensor data. However, due to the massive number of mobile sensors in an ITS, the transmission of all of the sensor measurements to a remotely located cloud as is done traditionally can result in high delays, communication network congestions, and computational overload. This will not be tolerable for an ITS since any delayed decision in the system might cause congestions, travel delays, and accidents. Therefore, efficient edge computing and analytics frameworks must be proposed to process much of the data at the level of each individual vehicle and send only the results to the cloud in order to meet the ITS latency and reliability challenges.

Current improvements in processing units of mobile devices, that can handle Gigabits of data in real-time, make them suitable for implementation of ITS edge analytics solutions. The passengers' mobile devices can be used as edge processors due to the available power supply from the vehicle. Moreover, every vehicle can be equipped with capable processing devices such as microprocessors, that can support complex, edge computing processes. However, for effective data processing, computationally capable edge devices cannot enable the vision of ITSs, unless they are coupled with artificially intelligent algorithms that can perform optimized edge analytics. In order to enable such smart, edge analytics, one can rely on the emerging frameworks from deep learning, that have shown their effectiveness in dealing with large image and signal datasets.

Deep learning techniques can perform sophisticated functions which cannot be analytically derived such as image and speech recognition. Therefore, if properly deployed, deep learning tools can be an effective edge analytics tool for ITSs, as they enable the optimization and processing of the heterogeneous ITS data. Deep learning approaches have already been proposed for various ITS applications such as in and. In, the authors proposed a novel deep-learning based traffic flow prediction method based on an autoencoder model that represents traffic flow features for prediction. The work in evaluated the temporal and spatial patterns of transportation network flow using a multi task learning architecture. These early works have shown that deep learning algorithms outperform conventional regression and time series methods in transportation systems, thus motivating further research in this area.

CHAPTER 2
LITERATURE REVIEW

Paper: "Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems"
Authors: Aidin Ferdowsi, Ursula Challita, and Walid Saad

This paper talked about Intelligent transport system. Where the component of a typical ITS is shown. My Project is a specific part of the whole system which deals with identifying objects and traffic signs using deep learning techniques. It tells how earlier technologies used cloud for ITS but it was underperforming. So, they showed how to improve the ITS. It talked about mobile edge analytics and it challenges and opportunity.

The Topics discussed were:

- Heterogeneous Data Sources
- Path Planning and Autonomous Control
- Vehicular Platoon Control
- Semi-autonomous ITSs
- ITS Security

Paper: "You Only Look Once: Unified Real-Time Object Detection"
Authors: Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

This paper talked about a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, it framed object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

CHAPTER 3
CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (for example SVM/Softmax) on the last i.e. fully-connected layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

3.1 Architecture Overview

Artificial Neural Networks receive an input as a single vector, and transform it through a series of *hidden layers*. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully connected layer is called the "output layer" and in classification settings it represents the class scores.

CNN take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. The layers of a CNN have neurons arranged in three dimensions: width, height, depth.

For example, the input images have dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. The final

output layer has dimensions 1x1x10, because by the end of the CNN architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension.

3.2 Layer used to build CNN

CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. There are three main types of layers to Build CNN architectures: Convolutional Layer**,** Pooling Layer**,** and Fully-Connected Layer. These layers are stacked to form a full CNN architecture.

- INPUT will hold the raw pixel values of the image, in this case an image of width size, height size, and with three color channels R, G, B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [n_width x n_height x n_filter] if n number of filters are used.
- RELU layer will apply an element wise activation function, such as the max (0, x) thresholding at zero. This leaves the size of the volume unchanged ([n_width x n_height x n_filter]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [n_width/2 x n_height/2 x n_filter].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1 x 1 x n_class], where there will be n categories. As with ANN and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, CNN transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient

descent so that the class scores that the CNN computes are consistent with the labels in the training set for each image.

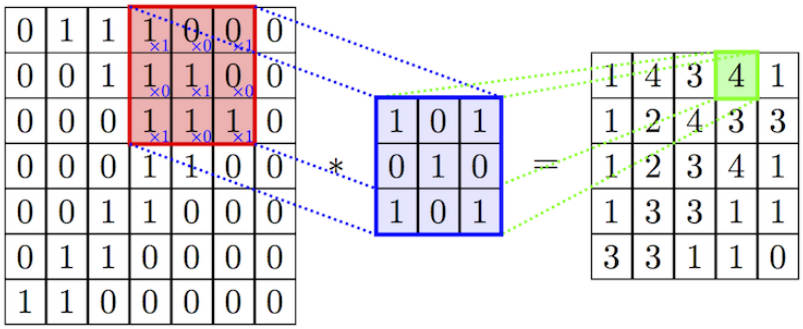


Figure 3.1 Convolution of a matrix using a filter

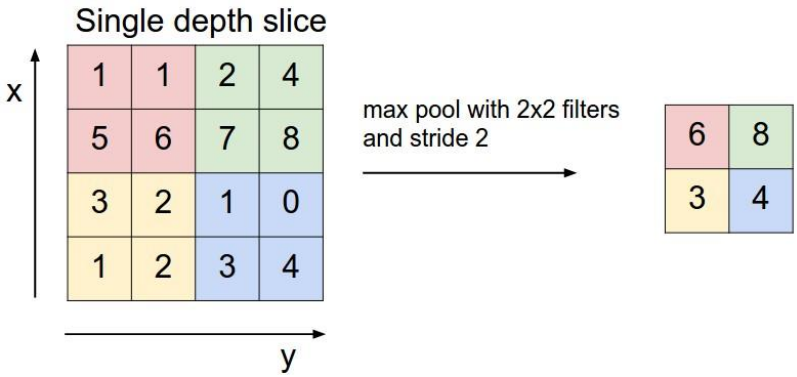


Figure 3.2 Max Pooling of a Convoluted Matrix

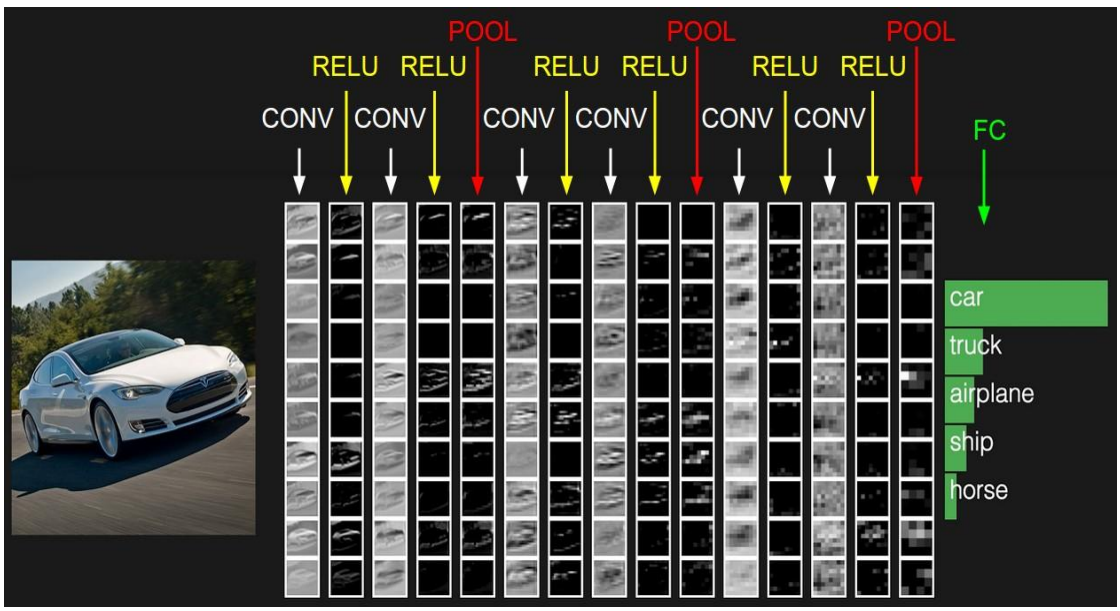


Figure 3.3 How a CNN predicts result from Input Image

CHAPTER 4
DIFFERENT CNN ARCHITECTURES

There are several architectures in the field of Convolutional Networks that have a name. The most common are:

- LeNet**:** The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990's. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.

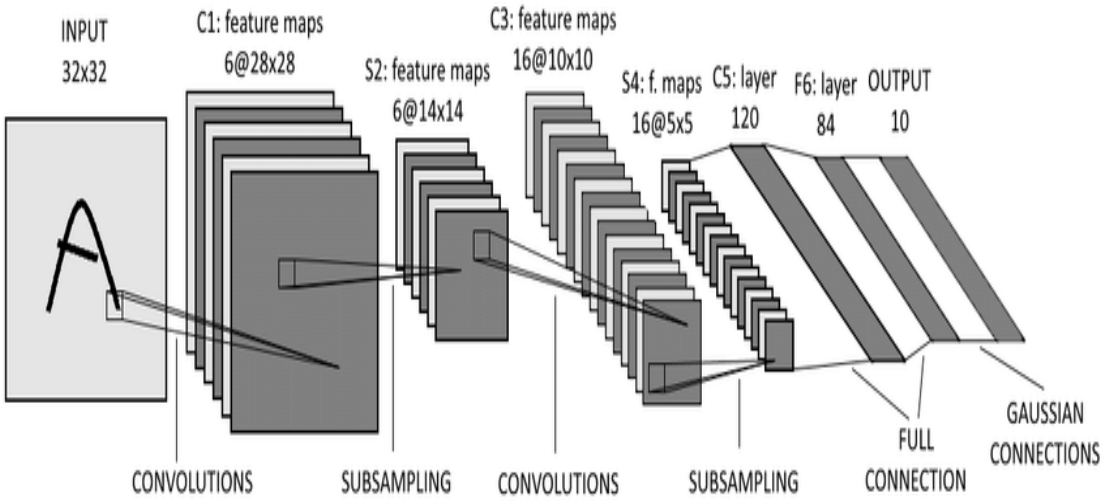


Figure 4.1 LeNet Architecture

- **AlexNet**: The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).

- **ZFNet**: The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

- **GoogLeNet**: The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the CNN, eliminating a large amount of parameters that do not seem to matter much. There are also several follow up versions to the GoogLeNet, most recently Inception-v4.

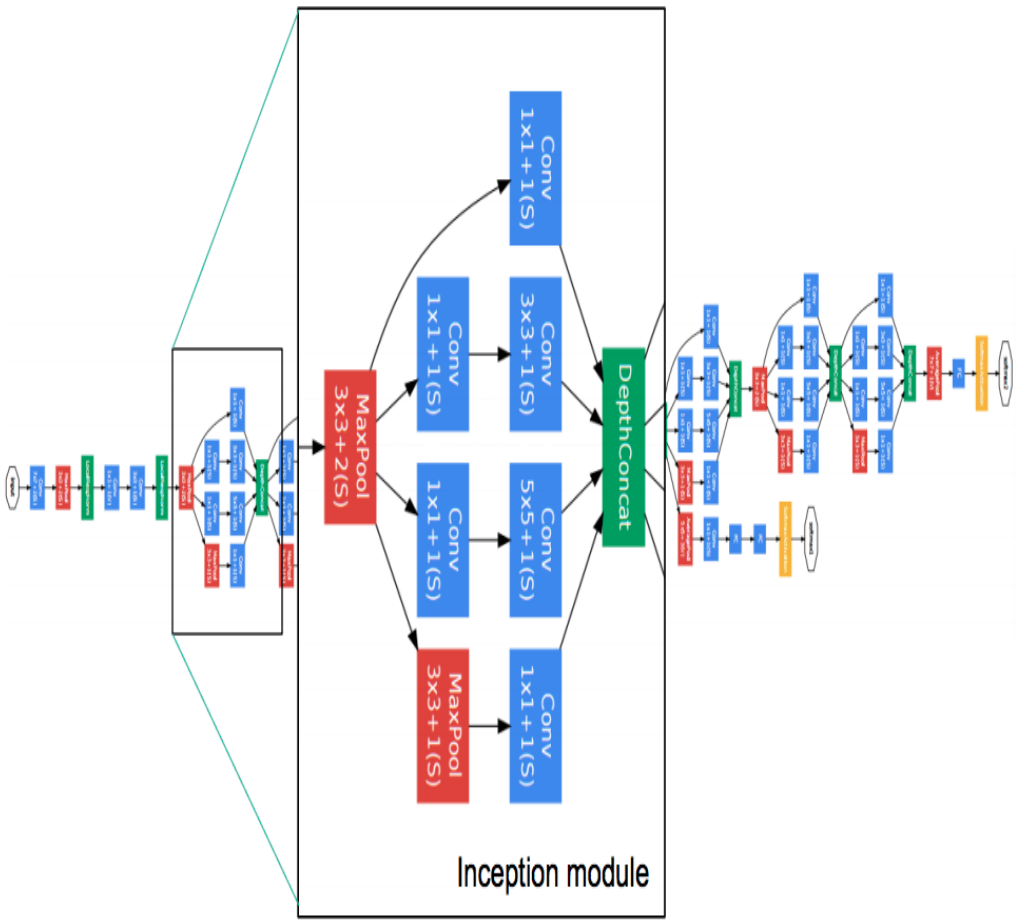


Figure 4.2 GoogLeNet Architecture

- **VGGNet**: The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. Their pretrained model is available for plug and play use in Caffe. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

- **ResNet**: Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special *skip connections* and a heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network. The reader is also referred to Kaiming's presentation (video, slides), and some recent experiments that reproduce these networks in Torch. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using CNN in practice (as of May 10, 2016). In particular, also see more recent developments that tweak the original architecture from Kaiming He et al. Identity Mappings in Deep Residual Networks (published March 2016).

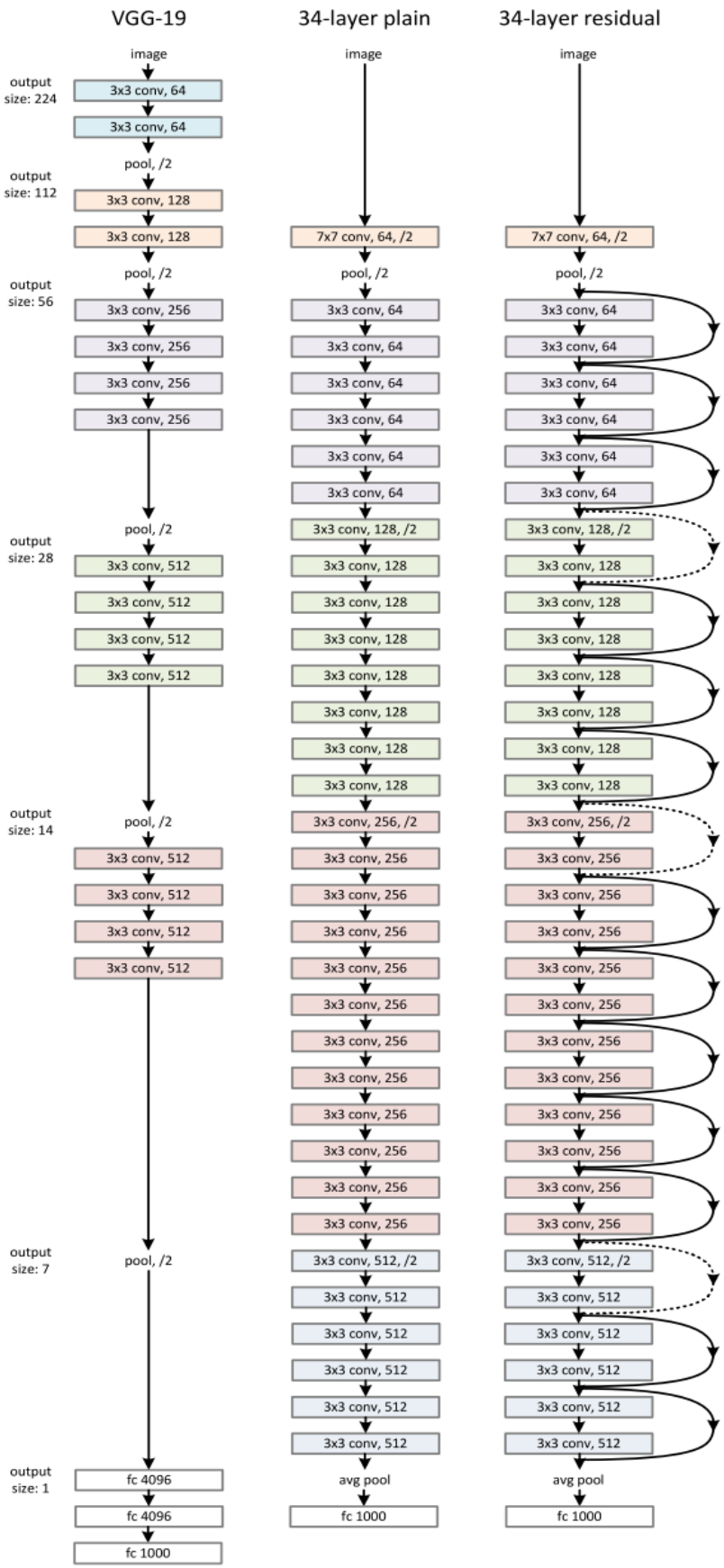


Figure 4.3 VGGNet and ResNet

CHAPTER 5
WORK DONE PREVIOUSLY

5.1 Preprocessing

The German Traffic Sign Recognition Benchmark (GTSRB) is the dataset used for this project which contains more than 38K images in 43 different classes.

Image Format:

The images contain one traffic sign each

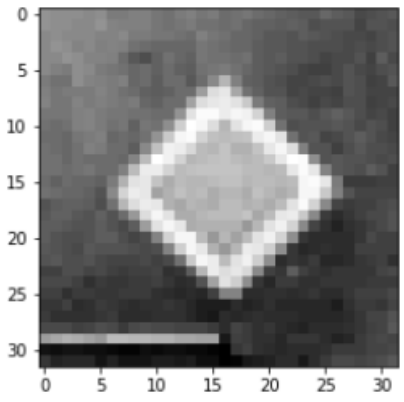
Images are stored in PPM format (Portable Pixmap)

Image sizes vary between 15x15 to 250x250 pixels

Since the images had different pixel values they were uniformly resized. The different sizes are 28x28, 32x32 and 50x50.

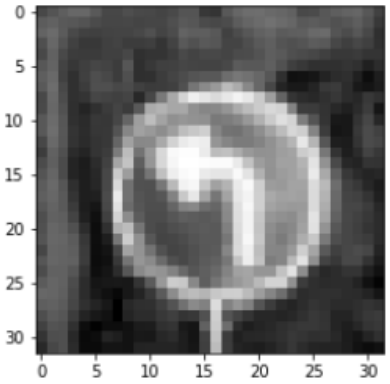
They were converted in Grayscale as well as RGB format for different architecture and stored in a numpy array with their Class Ids which is hot-encoded.

Examples of Input images:




```
[0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]
```

Figure 5.1 Input image in Grayscale Format

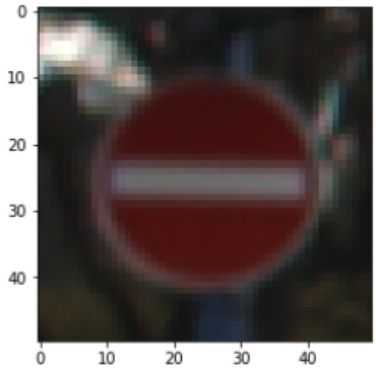


```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0]
```

Figure 5.2 Input image in Grayscale Format

```
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]
```

Figure 5.3 Input image in RGB Format



```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0]
```

Figure 5.4 Input image in RGB Format

5.2 Different CNN Architecture Used

The First architecture use in this project is a simple architecture which showed promising result to classify handwritten digits. The architecture is as follows:

Input: Image of 28x28 size with 1 channel (Grayscale)
Layer 1 (Convolutional): 32 filters of 5x5 size.
Activation: ReLu activation function.
Pooling: 2x2 filter with stride 2.
Layer 2 (Convolutional): 64 filters of 5x5 size.
Activation: ReLu activation function.
Pooling: 2x2 filter with stride 2.
Layer 5 (Fully Connected): This should have 1024 outputs.
Activation: ReLu activation function.
Layer 4 (Fully Connected): This should have 43 outputs.
Softmax Regression to find probability of all classes.
Optimizer: Adam Optimizer is used where learning rate is 0.001 beta_1 is 0.9 and beta_2 is 0.999
Loss function: Categorical Cross entropy.

Training Time: It took 70s to run each epoch NVidia GT920M graphics card.
Training Accuracy of this architecture was >5% after running 10 epochs. Which is very bad because this architecture helps to find digit not complex patterns.

The Second architecture use in this project is modified version of LeNet. The architecture is as follows:

Input: Image of 32x32 size with 1 channel (Grayscale)
Layer 1 (Convolutional): 6 filters of 5x5 size.
Activation: ReLu activation function.
Pooling: 2x2 filter with stride 2.
Layer 2 (Convolutional): 16 filters of 5x5 size.
Activation: ReLu activation function.
Pooling: 2x2 filter with stride 2.

Layer 5 (Fully Connected): This should have 120 outputs.
Activation: ReLu activation function.
Layer 4 (Fully Connected): This should have 43 outputs.
Softmax Regression to find probability of all classes.
Optimizer: Adam Optimizer
Loss function: Categorical Cross entropy.

Training Time: It took 20s to run each epoch NVidia GT920M graphics card.
Training Accuracy of this architecture was >93% after running 10 epochs.

The final CNN architecture that is used in this project is from VGGNet where there are 16 CONV/FC layers. Here it is modified and have 8 layers.

Input: Image of 50x50 size with 3 channels (R, G, B)
Layer 1 (Convolutional): The output shape should be 50x50x32.
Activation: ReLu activation function.
Layer 2 (Convolutional): The output shape should be 50x50x32.
Activation: ReLu activation function.
Pooling: The output shape should be 25x25x32.
Layer 3 (Convolutional): The output shape should be 25x25x64.
Activation: ReLu activation function.
Layer 4 (Convolutional): The output shape should be 25x25x64.
Activation: ReLu activation function.
Pooling: The output shape should be 12x12x64.
Layer 5 (Convolutional): The output shape should be 12x12x128.
Activation: ReLu activation function.
Layer 6 (Convolutional): The output shape should be 12x12x128.
Activation: ReLu activation function.
Pooling: The output shape should be 6x6x128.
Flattening: Flatten the output shape of the final pooling layer such that it's 1D instead of 3D.
Layer 7 (Fully Connected): This should have 128 outputs.
Activation: ReLu activation function.
Layer 8 (Fully Connected): This should have 43 outputs.

Softmax Regression to find probability of all classes.

Optimizer: Adam Optimizer

Loss function: Categorical Cross entropy.

Training Time: It took 100s to run each epoch NVidia GT920M graphics card.

Training Accuracy of this architecture was >95% after running 20 epochs. Which is quite good but not human level accurate.

5.2 Output

Model was evaluated on testing data set which was around 25%. The test dataset is a dataset that is independent of training dataset but follows same probability distribution as the training dataset.

Accuracy was around >94%.

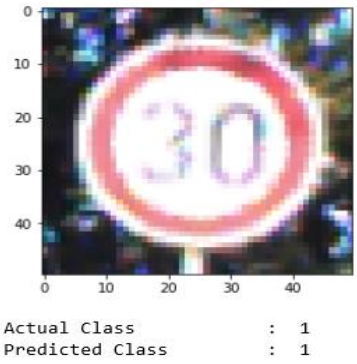


Figure 5.5 Predicted Results of an Image

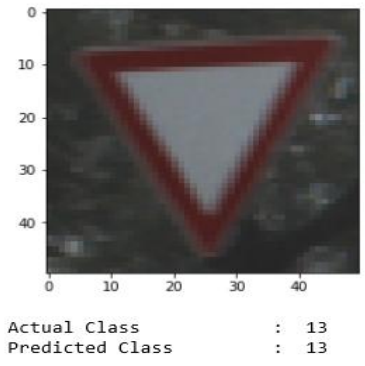


Figure 5.6 Predicted Results of an Image

Figure 5.5 and 5.6 shows how the model predict the results accurately where the images have better lighting conditions and better clarity.

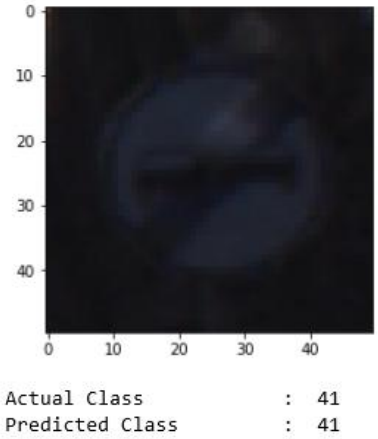


Figure 5.7 Predicted Results of an Image

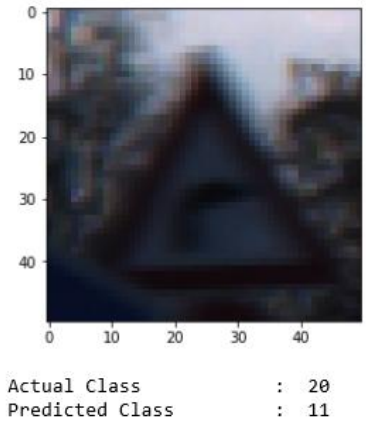


Figure 5.8 Predicted Results of an Image

Figure 5.7 show the model has predicted correctly even if the lighting conditions were very poor but the clarity was better.

Figure 5.8 show the model has incorrectly predicted the class of image because the resolution and the clarity of image was bad.

CHAPTER 6
YOLO

Object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using this system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is very simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

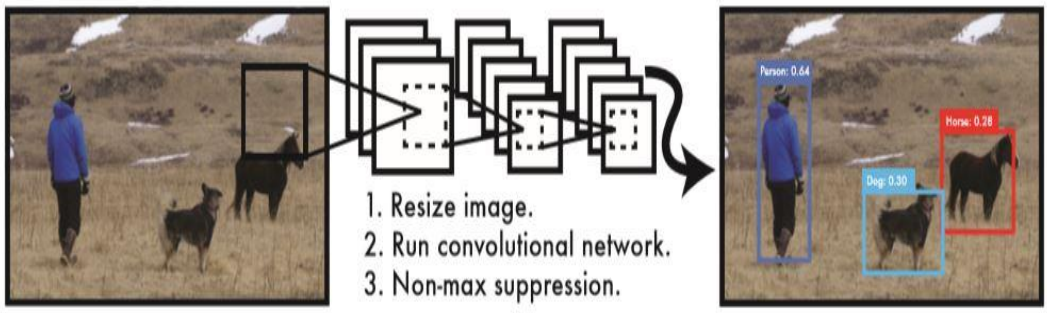


Figure 6.1 The YOLO Detection System

YOLO is extremely fast. Since we frame detection as a regression problem, we don't need a complex pipeline. The neural network was run on a new image at test time to predict detections. The network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means it can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method, mistakes background patches in an image for objects because it

can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

6.1 Working Principle

The network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. The YOLO design enables end-to-end training and real time speeds while maintaining high average precision.

The system divides the input image into an S ×S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally the confidence is defined as $Pr(Object)*IOU^{truth}_{pred}$. If no object exists in that cell, the confidence scores should be zero. Otherwise the confidence score is equal to the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box consists of 5 predictions: *x, y, w, h,* and confidence. The (*x,y*) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box.

Each grid cell also predicts C conditional class probabilities, $Pr(Class_i|Object)$. These probabilities are conditioned on the grid cell containing an object. Only one set of class probabilities per grid cell, regardless of the number of boxes B is predicted.

Sometimes, the conditional class probabilities are multiplied with individual box confidence predictions, which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object is shown in equation 6.1.

$$Pr(Class_i|Object)*Pr(Object)*IOU^{truth}_{pred}=Pr(Class_i)* IOU^{truth}_{pred} \quad (6.1)$$

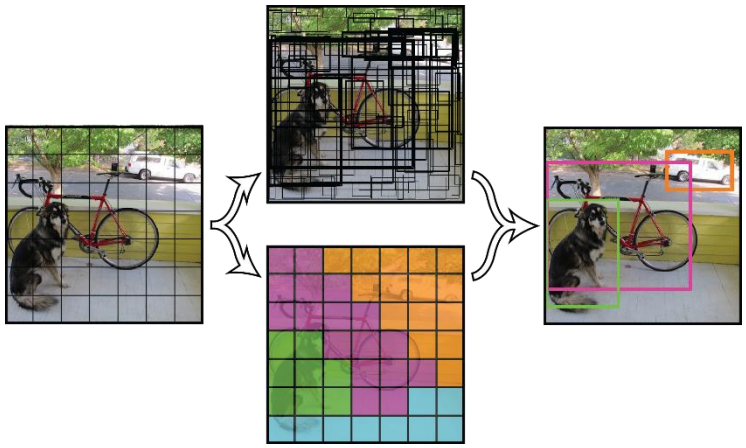


Figure 6.2 The Model

6.2 CNN for Feature Extraction

This network is a hybrid approach which is different from standard CNN models. The architecture is as follows:

Input: Image of 416x416 size
Layer (Convolutional): 32 filters of 3x3 size and output shape 256x256.
Pooling: 2x2 filter with stride 2.
Layer (Convolutional): 32 filters of 1x1 size.
Layer 3 (Convolutional): 64 filters of 3x3 size.
Residual Layer: Output size 128x128
Pooling: 2x2 filter with stride 2.
Layers given below were repeated 2 times
Layer (Convolutional): 64 filters of 1x1 size.
Layer (Convolutional): 128 filters of 3x3 size.
Residual Layer: Output size 64x64
Pooling: 2x2 filter with stride 2.
Layers given below were repeated 8 times
Layer (Convolutional): 128 filters of 1x1 size.
Layer (Convolutional): 256 filters of 3x3 size.
Residual Layer: Output size 32x32
Pooling: 2x2 filter with stride 2.
Layers given below were repeated 8 times
Layer (Convolutional): 256 filters of 1x1 size.

Layer (Convolutional): 512 filters of 3x3 size.

Residual Layer: Output size 16x16

Pooling: 2x2 filter with stride 2.

Layers given below were repeated 4 times

Layer (Convolutional): 512 filters of 1x1 size.

Layer (Convolutional): 1024 filters of 3x3 size.

Residual Layer: Output size 8x8

Pooling: Average pooling of global size.

Layer (Fully Connected): This should have 1000 outputs.

Softmax Regression to find probability of all classes.

It is using a linear activation function for the final layer and all other layers use the following leaky rectified linear activation is shown in equation 6.2:

$$\varphi(x) = \begin{cases} x, & if\ x > 0 \\ 0.1x, & otherwise \end{cases} \quad (6.2)$$

Sum-squared error is used as it is easy to optimize.

6.4 Loss Function

The loss functions used are:

Classification loss:

If an object is detected, the classification loss at each cell is the squared error of the class conditional probabilities for each class is shown in equation 6.3:

$$\sum_{i=0}^{S^2} 1_i^{obj} \sum_{C \epsilon classes} (p_i(c) - \hat{p}_i(c))^2 \quad (6.3)$$

1_i^{obj} = 1 if an object appears in cell *i*, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell *i*.

Localization loss:

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object is shown in equation 6.4.

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[(x_i-\hat{x_i})^2+(y_i-\hat{y_i})^2\right]$$
$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[(\sqrt{w_i}-\sqrt{\hat{w_i}})^2+(\sqrt{h_i}-\sqrt{\hat{h_i}})^2\right] \quad (6.4)$$

1_{ij}^{obj} = 1 if the *j*th bounding box in cell *i* is responsible for detecting the object, otherwise 0.

λ_{coord} increases the weight for the loss in the bounding box coordinates.

Absolute errors in large boxes and small boxes shouldn't weight equally. i.e. a 2-pixel error in a large box is the same for a small box. So, YOLO predicts the square root of the bounding box width and height instead of the width and height. In addition, to put more emphasis on the bounding box accuracy, we multiply the loss by $\lambda_{coord} = 5$.

Confidence loss:

If an object is detected in the box, the confidence loss (measuring the objectness of the box) is shown in equation 6.5:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C_i})^2 \quad (6.5)$$

1_{ij}^{obj} = 1 if the *j*th bounding box in cell *i* is responsible for detecting the object, otherwise 0.

$\hat{C_i}$ is the box confidence score of the box *j* in cell *i*.

If an object is not detected in the box, the confidence loss is shown in equation 6.6:

$$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{noobj}(C_i-\hat{C_i})^2 \quad (6.6)$$

λ_{noobj} weights down the loss when detecting background.

1_{ij}^{noobj} is the complement of 1_{ij}^{obj}.

$\hat{C}_i$ is the box confidence score of the box *j* in cell *i*.

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. the model is trained to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor $\lambda_{noobj} = 0.5$.

Loss:

The final loss adds localization, confidence and classification losses together. i.e., eq(6.3) + eq(6.4) + eq(6.5) + eq(6.6).

The training was dome in batch size of 64, a momentum of 0.9 and a decay of 0.0005. For first 75 epochs the learning rate was 0.01, then 0.001 for 30 epochs, and finally 0.0001 for 30 epochs.

6.4 Output

The highest score for a box is also called its confidence. If the confidence of a box is less than the given threshold, the bounding box is dropped and not considered for further processing.

The boxes with their confidence equal to or greater than the confidence threshold are then subjected to Non-Maximum Suppression. This would reduce the number of overlapping boxes.

Figure 6.3 Input Image

Figure 6.4 Output when NMS=0.1

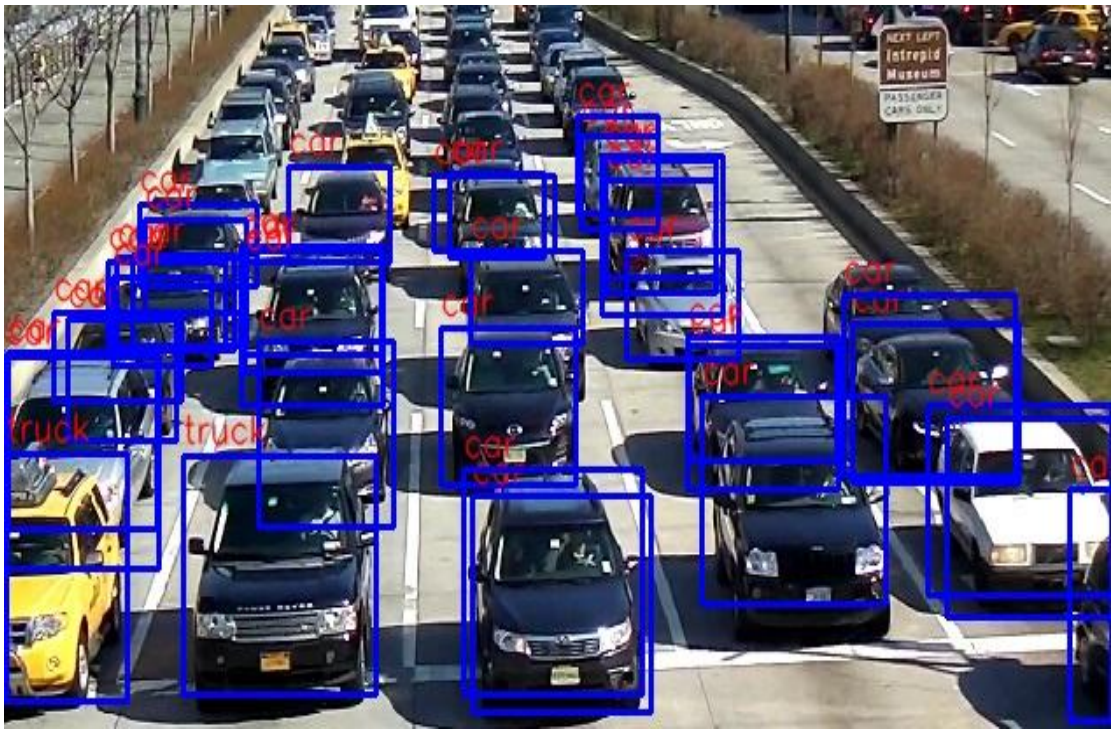


Figure 6.5 Output when NMS=0.8

Figure 6.6 Output when NMS=0.4

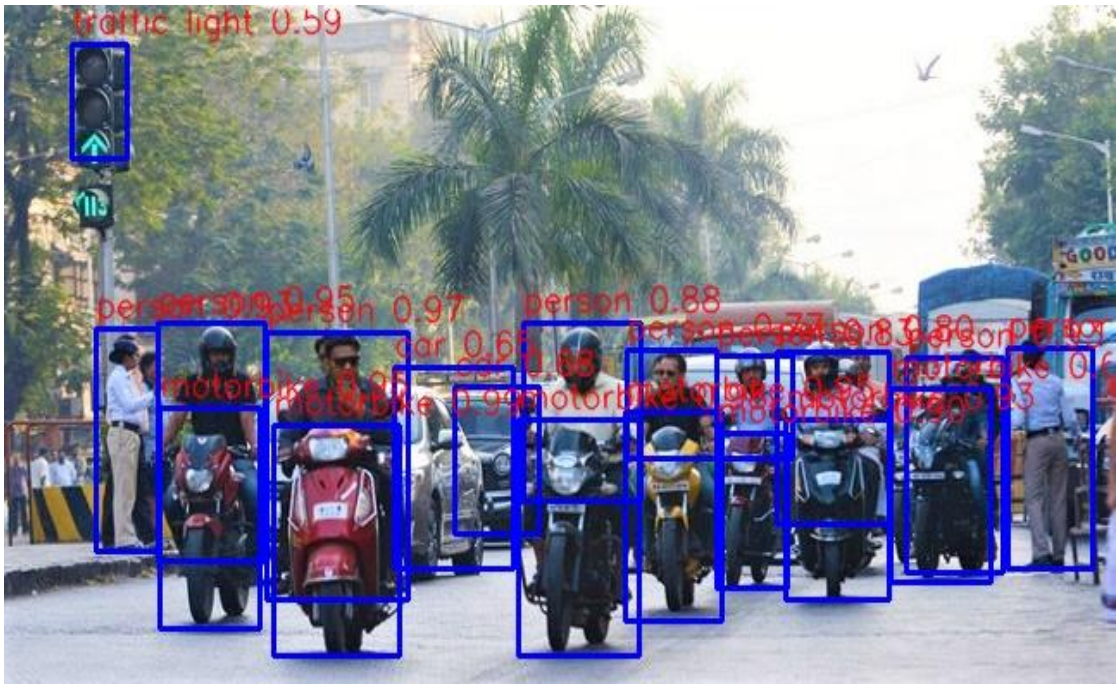


Figure 6.7 Output on different image

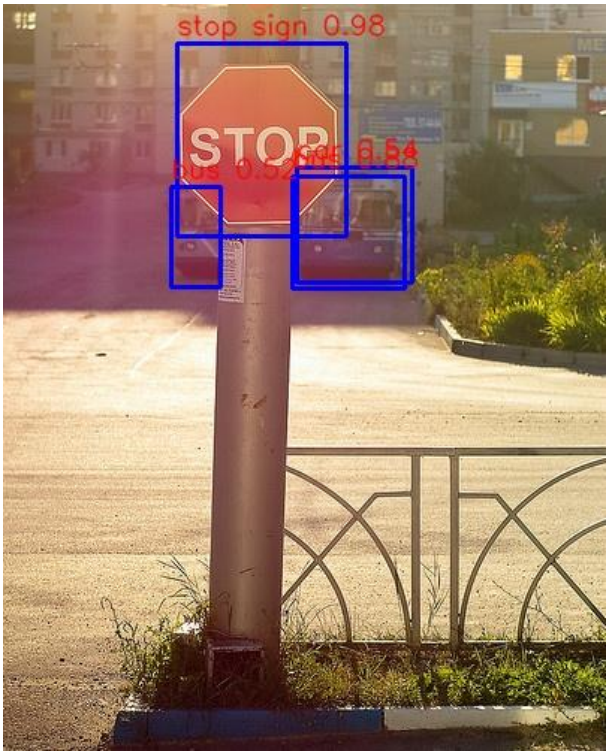


Figure 6.8 Output on Traffic Sign

Figure 6.9 Output on Traffic Signal

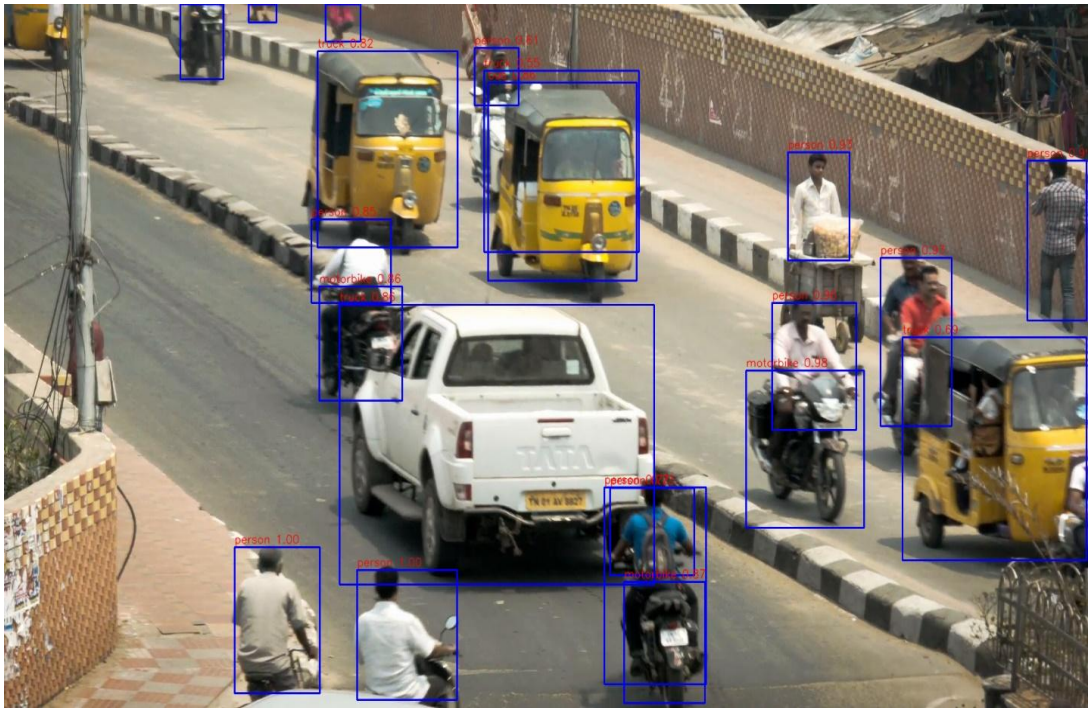


Figure 6.10 Output on different image

CHAPTER 7
CONCLUSION

Identification of any object and traffic road sign is an important for autonomous vehicles, at the same time achieving the 100% accuracy is also difficult and complex task. In order to achieve the efficacy, I have implemented the traffic road signs identification of different data set. To achieve the efficacy, the preprocessed dataset applied on different models.

The modified version VGGNet for CNN model which performed very good with RGB images showing an accuracy more than 95% which is 3% more accurate than the LeNet while using RGB images as input.

Then YOLO model was used which was able to identify different objects, traffic sign and traffic signal present on the road. It created bounding boxes for the objects and was successful in classifying them correctly. It has very fast inference time as compared to other models.

REFERENCES

[1] Ferdowsi, U. Challita,, W. Saad. "Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems"

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. "LeNet-5, Convolutional Neural Network"

[3] K. Simonyan, A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition"

[4] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You Only Look Once: Unified Real-Time Object Detection"

[5] J. Redmon, A. Farhadi, "YOLOv3: An Incremental Improvement"

[6] "Convolutional Neural Networks for Visual Recognition" website:http://cs231n.github.io.