A PRELIMINARY REPORT ON

TRAFFIC SIGN DETECTION AND CLASSIFICATION BY YOLOv2 USING DEEP LEARNING

BY

ROHIT AGARWAL **19CS06002**

SHOBHIT **19CS06008**

DEPARTMENT OF COMPUTER ENGINEERING
SCHOOL OF ELECTRICAL SCIENCES
IIT BHUBANESWAR

Abstract:
The enhancements in the field of computer vision and deep learning led to a renaissance in the autonomous domain. These advancements can be applied in a number of ways where a driver from the car can be replaced by the autonomous module which can learn how to drive and act on the given condition. One such module in driverless cars is traffic sign detection and recognition of class. The traffic sign detection could help in making the car more efficient for autonomous driving. The traffic sign detection contains four phases namely data collection, data and image processing, convolutional neural network and output processing. The dataset here we have considered is of street where there are some traffic signs and each image is annotated with bounding box coordinates and the class labels of that particular bounding box. The bounding boxes in the annotations have the format of representation as x_min, y_min, x_max and y_max and the class labels. Each image id has a list of objects having these coordinates and class labels as their values where image id are their key. The next phase is the image processing and data processing where the images are resized and normalized to make them compatible with the model and the data processing is done in order to convert the representation which is more stable with respect to learning of the model. The data processing is also done in other perspectives such that it can be useful for the calculation of the loss function. The next is the convolutional neural network where the model is the same as that of the YOLOv2 model which has been proven as a better algorithm over others when it comes to object detection. The YOLOv2 has proven to be better at detecting multiple objects in a particular part of image and can detect a maximum of prespecified number. The main focus of this project would be to detect the traffic signs and the class which is obtained from the CNN model. Since this model is considering the task of localizing the classifying the traffic signs at the same time appropriately there is a need for a different loss function, thus an additional layer of lambda function is added to compute the loss value. This loss value helps in training the neural networks. The output of the CNN model is raw and needs to be transformed so the next stage is the output processing. Finally we represent the bounding boxes in the form of rectangular coordinates with class labels and a visual judgement can be made accordingly.

Keywords:
Convolutional Neural Networks, BackPropagation, Loss function, Processing the data, anchors, IOU, Non-max suppression, Normalizing, Localization, Classification, YOLOv2.

Chapter 1: Introduction

Overview:

A system that allows a vehicle to drive safely on public roads without human intervention and involvement is known as “autonomous driving”. Terms like automated driving, driverless vehicle and self driving car are all describing the same technology. A computer brain and sensors that can drive a car in place of humans, at least under some circumstances. Under no or minimal human involvement vehicles can operate, similar to autopilot in aeroplanes. As these vehicles will be able to handle various circumstances appropriately without any accidents, higher speed limits and different restrictions denoted by traffic signs may be put into considerations. One way to make the restrictions possible is to make its detection automatically in a driverless car is through deep learning. Deep Learning has some very interesting properties such as being able to automatically learn complex mapping functions and has the ability to scale up easily. Such properties are important in many real world applications such as large scale image classification and recognition.

The goal of this project is to find a model that when given an image taken while driving detects the traffic sign in the image with minimal loss and classify the traffic sign so that the system can decide what to be the next step. In this project, we implement the applications of techniques in supervised learning to predict the bounding box for the traffic sign and the type of traffic sign which is inside the bounding box. The implementation of the same is done using a well structured dataset which is in a key value pairs in json format. The primary goal is to design a well sophisticated model that should encapsulate real life scenarios and should generate results within acceptable errors. Additionally, the model should be free of assumption about specific hard coded templates, rules or categories and instead rely on learning from the training data. Each entry in the dataset is well annotated with the bounding boxes and their respective classes along with the image id to distinctly identify the image. For this we apply a procedure of four phases to carry out the same.

The Data collection phase includes the collection of images where a significant number of images contain traffic signs for their recognition. Each image in the dataset should have annotations specifying whether the image contains the traffic signs or not and if it contains then how many traffic signs are in it. The dataset collected cannot be directly used and is in the pure raw format. For better results, the dataset needs to be pre-processed in order to make the predictions stable and in a proper format. The data pre-processing stages also create some volumes known as detector masks and matching true boxes which are explained in detail in the succeeding sections. The bounding box coordinates also need to be adjusted depending upon the model's stability and requirements. These are useful for training in terms of calculation of the loss function and backpropagating the same. The images along with the dataset needs some preprocessing as they are also in their raw formats. The image pre-processing includes steps like normalization and resizing of the images. The image collected in their raw format is in the shape of 2048 x 2048 and can be a load on training such a large image. Thus we need to resize the image in order to train the model faster. The images are then fed to the neural network part which is the main model mapping which tries to predict the bounding boxes and the classes of the traffic signs in the images.

The model that we have used here is very similar to the YOLOv2 model and has the same characteristics of the YOLOv model. The model has 23 layers of convolution operations with some layers of max pooling and batch normalization. The model uses the activation function as leaky relu which is used after every convolution operation and softmax function for the final layer to predict the class. It also contains a skip connection where a forward feedback is given to a layer in the further model. The reason for this is to avoid the loss of gradient problem during backpropagating the error in the model. The skip connection also enables us to learn such a deep network without losing the main goal of the model. Since, the aim of the model is to predict the bounding box and the classes of it we need to rely on a different loss function which is a combination of regression as well as classification loss where regression loss is for the bounding box prediction and classification loss is for the class prediction of the traffic sign. This loss function takes input as the actual boxes, the model's output, the preprocessing results obtained from the data preprocessing and produces a single real valued mean loss number. This loss can help us to determine the accuracy of the model. The model's output cannot be directly used for prediction and localization, instead it needs some processing.

This is the final phase of the module where the output volume of the model is converted into a human understandable and representable form. Even the model predicts many extra boxes which need to be removed from the prediction using the methods of non-max suppression and IOU. These methods are described in later sections. Then we draw the final output boxes after this processing and a human level accuracy can be checked accordingly. The final boxes before drawing need some conversion in the reverse

process as we did in the preprocessing phase to obtain the actual sized image coordinates. Thus the boxes are ready for the drawing on the image with the labels of it.

Motivation:

Traffic sign detection becomes important while following the traffic rules. Autonomous traffic sign detection is useful when the driver is too careless to notice the traffic sign while driving or when the autonomous driving techniques are used. This helped us to design a system where an efficient system needs to be created which not only helps the car to locate the traffic sign but also classify the sign and act according to the sign in the image captured by the car. The main aim of this is to integrate it with a driverless along with say steering angle prediction module which can help the other module to work seamlessly. This interwork could help a driverless car possible in real life. Not only this but also more modules like object detection can be added to this system so that all can work towards the same goal.

Problem Statement:

Given an image at real time, detect and classify the traffic sign if any in the image seamlessly so that the driverless car can learn how to predict the traffic signs and act accordingly in order to avoid any catastrophic event. The images in the dataset are 2048 x 2048 dimensioned and the final output is the same image 2048 x 2048 with labelled bounding boxes on it which encloses the traffic signs. The bounding boxes must also denote the class to which the traffic signs belong to. The problem is supervised learning technique where the input is the image (independent variable) and the output labels (dependent variables) are the annotations of the image where each annotation is the traffic signs box coordinates and the classes. This is a problem of localization where we need to localize an object given at hand and classify them based on the assumptions while developing the system.

Problem Scope:

Every project when developed cannot be a full fledged and has some bounded features and scopes. The scope of this project is to identify only the traffic signs and not any other objects. The model trained and generated is trained in such a manner which is useful only for the prediction of traffic signs and to effectively classify them. The model does not consider any other objects. But by extending the dataset that we consider for training, we can extend the same model for object detection as well but with the expense of more training time. As the model is large, there are more parameters to be trained thus by considering a more general identification of objects by this model this can cause to increase the training time significantly. However the trained model weights can be considered as it is and the concept of transfer learning can be applied in order to detect more general objects rather than just traffic signs.

Methodologies of Problem Solving:

1) Supervised Deep Learning
2) Annotating the dataset
3) YOLOv2 Model (CNN Model)
4) IOU
5) Non-max Suppression
6) Anchor boxes
7) Loss Function
8) Localization and classification

Chapter 2: Software Requirement Specification

User Classes and Characteristics:

1) Normal Users:

   This class consists of the end users who are benefited by this module when integrated with the other module which helps in making the driverless car possible. This class is concerned with the external functionality provided by the system and not the internal details of how the internal implementation is. This work is next generation technology where each user will have a car with this module implemented in it. The users use the system and expect no bugs in any regards and that the system works flawless.

2) Companies:

   This module can work as a building block for other types of systems and can be considered as a module of that system individually. The companies working in the field of computer vision, deep learning and automobiles will have a good usage of this system for automating daily life transportation and in robotics. The companies can also use this pretrained model and apply the concepts of transfer learning so that the same model can predict other objects.

3) Researchers:

   The project is prone to research as there are many parameters, we can change the parameters and try to find even better optimized ones which can lead to better accuracy. The researchers can also change models for their research perspectives and find better ones when compared to this which can improve the capability of localization and classification of the traffic signs in the image.

Assumptions and Dependencies:

1) The system needs to have Numpy, pandas, python, tensorflow, keras installed so that its functionalities can provide an easier way of implementation.
2) The model is responsible for predictions of only traffic signs and no other objects can be detected using this trained model. However, we can extend the same to detect other objects as well.
3) The dataset needs to be annotated in the specified format and each annotation needs to correct else can cause problems in detecting.

Functional Requirements:

- Dataset:

  Every deep learning when needed to train requires an extensive amount of data such that it can draw mappings related to the input and the outputs. For this mapping to happen we need a proper dataset with proper representations which would need minimum handling and more information. For this application, we consider our dataset to be annotated as well. An the primary requirement is that the input independent variable of the model is the input image which has the size of 2048 x 2048 and the dependent variables are the annotations of the image which consists of coordinates of bounding boxes which are used to enclose the traffic sign and class label of the traffic sign bounded by the corresponding bounding box. The dataset annotations news to be in a key value pair which is a better way for representation of such a large amount of annotations file. This forms the basis of model training and generation of mappings from input image to output images.

- Pre-Processing:

  Even though the dataset is well annotated the first thing we need to do after reading the dataset is to preprocess the data such that the data is relevant to be trained and does not cause any problems while training. For this we convert the data from corners coordinates to midpoint, height and width notations which proves to be better when compared to the training done on corner coordinates. Moreover the data points need to be normalized such that the training does not cause problems. Additionally, data needs to be preprocessed for the calculation of the loss that is the difference in the output and the actual values. The data is processed in such a manner that it creates a volume of the annotations which is very similar to the underlying model. This can directly be compared with the predicted value in order to calculate losses. The image input needs to also be reshaped to adjust it according to the YOLOv2 model that we have used. The images are resized to 608 x 608 and then are normalized to make the pixel value range from 0 to 1 so that it becomes easier and faster for training.

- Model:

  The model here used is the YOLOv2 model which consists of 22 layers of convolution, batch-normalization and the activation function of leaky relu. This network is fully convolutional and does not include any flattening or single dimensional vector, it only deals with the volume. The network has a softmax activation function as the last layer activation function so that it can classify to which class the bounding box belongs. The model contains a skip connection where a certain number of layers are skipped and the output of it is forwarded to layers which come after that layer. This helps in learning the very deep network.

- Loss Function:

  The problem at hand does not only consider detecting but also classification. Thus we require a better loss function which can consider both regression and the classification losses and a proper weightage for each loss. The losses that we should consider here are the coordinate loss, classification loss and the confidence loss. The output that we obtained from the preprocessing stage and the output stage are used here to calculate the loss value which is a single valued real number. The coordinate loss is the loss from the prediction of the coordinates of bounding boxes and the actual data volumes generated from the data preprocessing. The confidence loss signifies that if an object is present how confidently the model is able to predict that object which is a probabilistic value. The last is the classification loss which is the wrong class generation errors.

- Output Processing:

  The model predicts many boxes which some are noises and some are just redundant boxes. Thus we need to eliminate these boxes which are redundant. The redundancy can be eliminated by using the concept of IOU and non-max suppression. The boxes which are false positives can be eliminated by simply checking the confidence field of the prediction volume. If the confidence is less than the threshold than the object is simply eliminated and the boxes are not considered. Next the important boxes are converted into the form which can be drawn on the output image and can be represented to the users.

- Draw Boxes:

  The final boxes are drawn on the original size image which is the 2048 x 2048. The boxes can be drawn using the openCV libraries which could help the clients to manually judge whether the predictions are within the acceptable limits and if not the model needs to be evaluated once again for its correctness.

Non - Functional Requirements:

- Performance:

  For increasing the speed of the model to be trained we need a GPU with a minimum of Nvidia Geforce GTX 1050Ti or higher which can provide better speed while training. The detection i.e outputs needs to be also faster for which we need a faster CPU.

- Safety:

  Since the model predicts the traffic sign way before the model predictions can be considered as safer. Even the smallest traffic sign can be detected with this model. The project is safer to use as the loss will be as less as possible and the alerting system can be easily integrated with this.

- Security:

  The project is secured with consideration that the model which it is integrated in is secured. The model when given any random images does not predict false and simply ignores the random noises. Thus changing the image and doing modifications to the image captured needs to be avoided and directly it should be fed to the model. Moreover developing a deep learning application requires a large dataset, thus dataset manipulations need to be taken care such that the model does not learn any garbage.

- Software Quality Attributes:

  1) Maintainability:

     Maintainability is the ability of the system to undergo changes with the degree of ease. These changes could impact components, services, features and interfaces when adding or changing functionalities, fixing errors and meeting new business requirements. In our project we have used standard design patterns, also we have used various libraries and modules instead of sequential code wherever possible to ensure that software can undergo changes with degree of ease.

  2) Reusability:

     Reusability defines capability for components and subsystems to be suitable for use in other applications and scenarios. It minimizes the implementation of already implemented modules and encourages the usage of the same module again. We have used the modular approach of designing which can be broken and used individually,

  3) Extensibility:

     Extensibility is system design principle where implementation takes future growth. Extension can be through the addition of new functionality or through modification of existing functionality. Our project is although limited to only prediction and detection of traffic sign it can easily be extended for detecting the other objects as well considering a different dataset altogether.

System Requirements:

- Database Requirements:

  The dataset is stored in a json file format which consists of key value pairs. It should be formatted properly to avoid any last minute surprises while handling the dataset in this file format.

- Software Requirements:

  1) Tensorflow
  2) Keras
  3) Jupyter Notebook

4) Python
5) Numpy Pandas
6) OpenCv
7) Json File format handler

- Hardware Requirements:
  1) External GPU GTX 1050Ti minimum
  2) 16 GB DDR4 RAM
  3) SSD Memory
  4) I7 Processor

Chapter 3: System Designs

System Architecture:

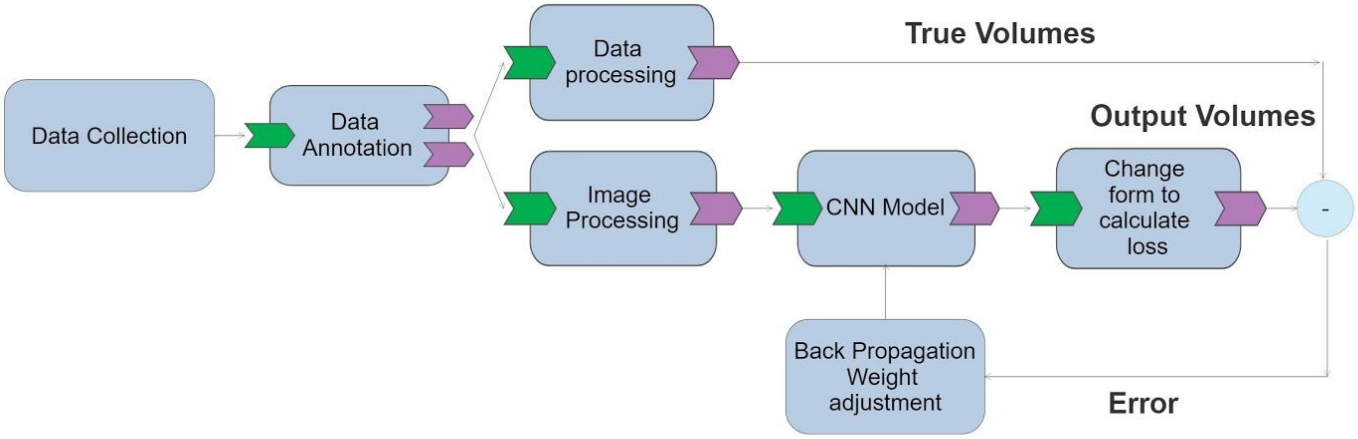


Fig 3.1: System Architecture (Training)

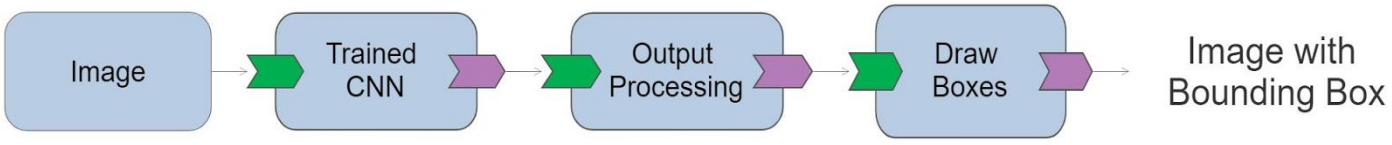


Fig 3.2: System Architecture (Testing)

Class Diagram:

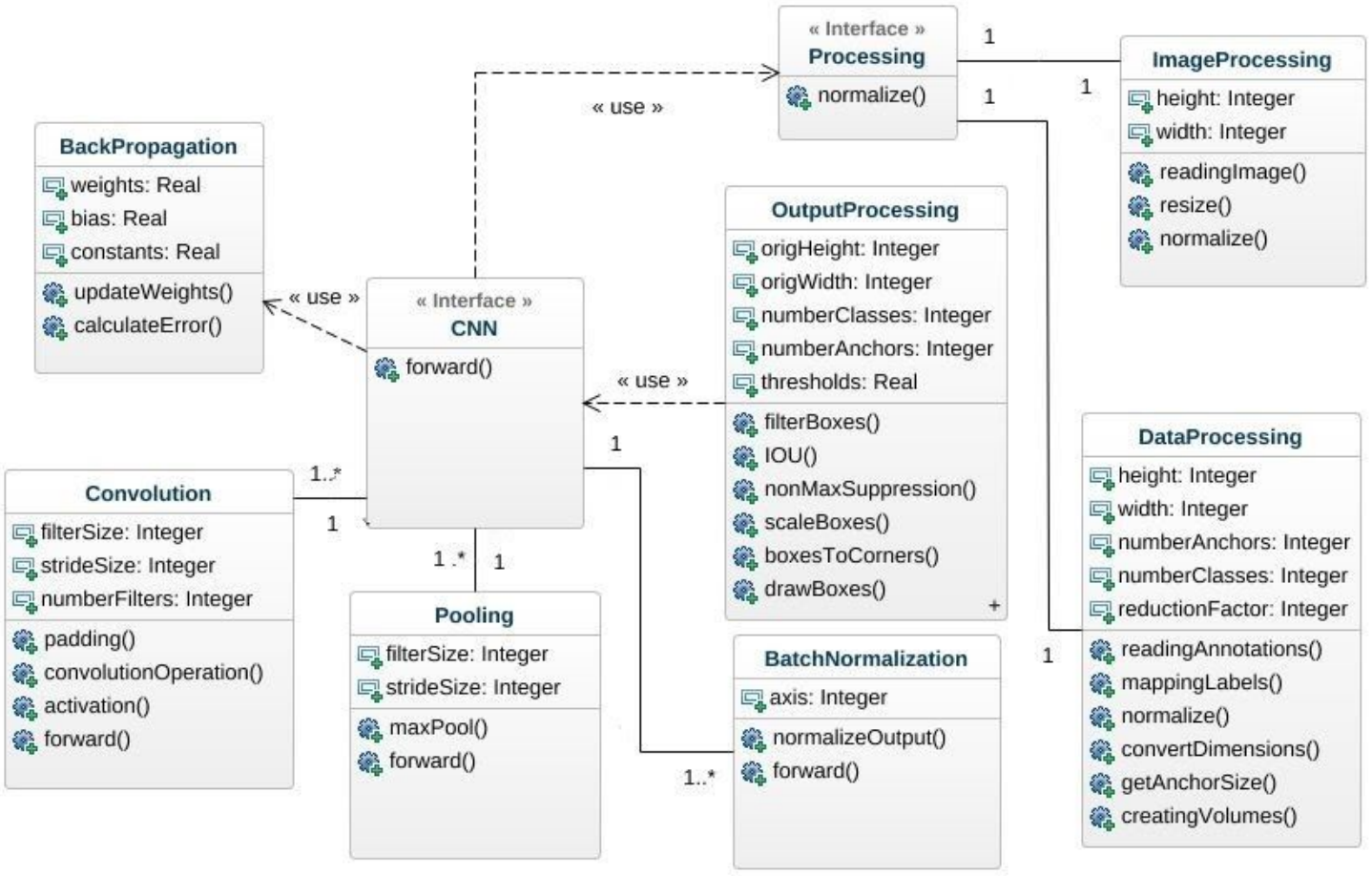


Fig 3.3: Class Diagram

Sequence Diagram:

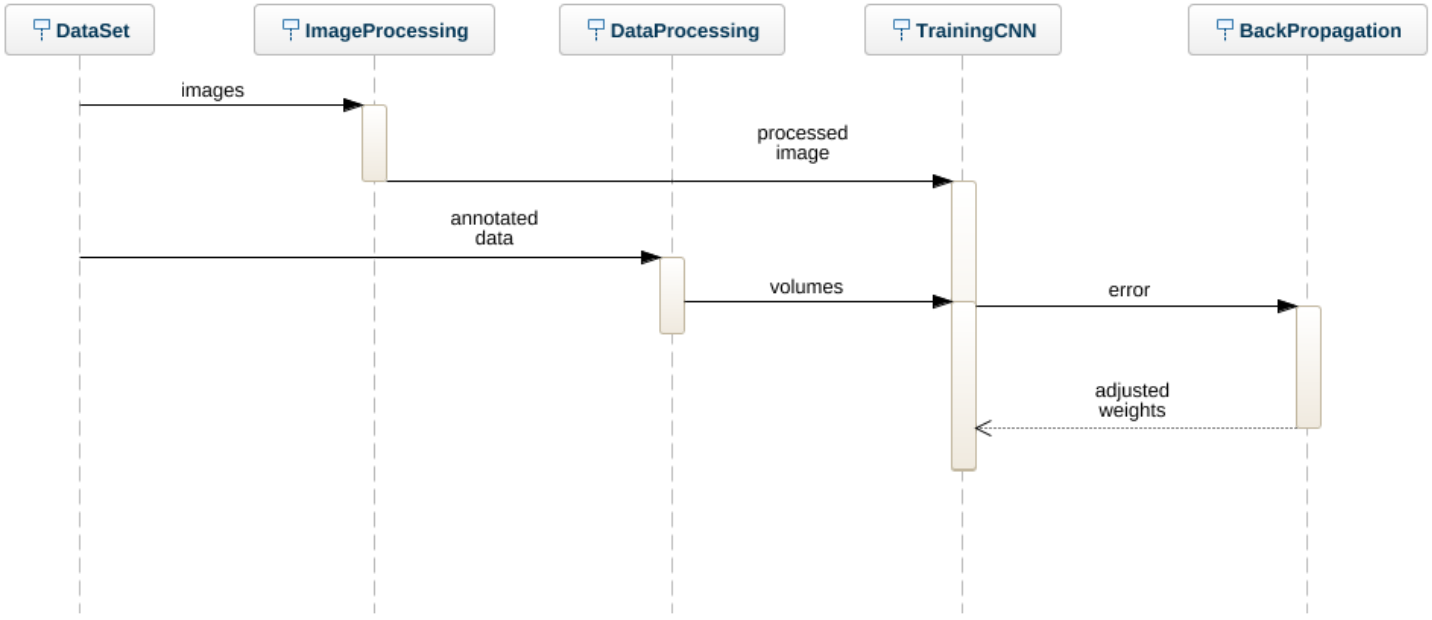


Fig 3.4: Sequence Diagram (Training)

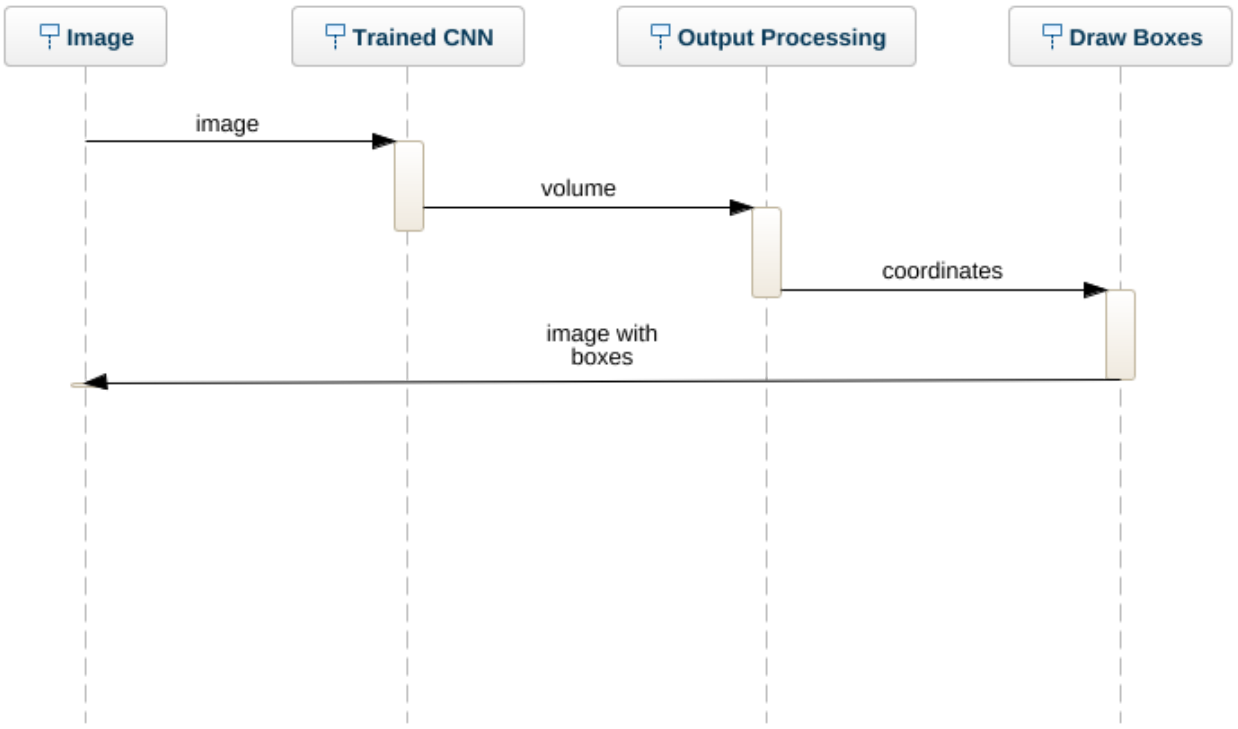


Fig 3.5: Sequence Diagram (Testing)

Deployment Diagram:

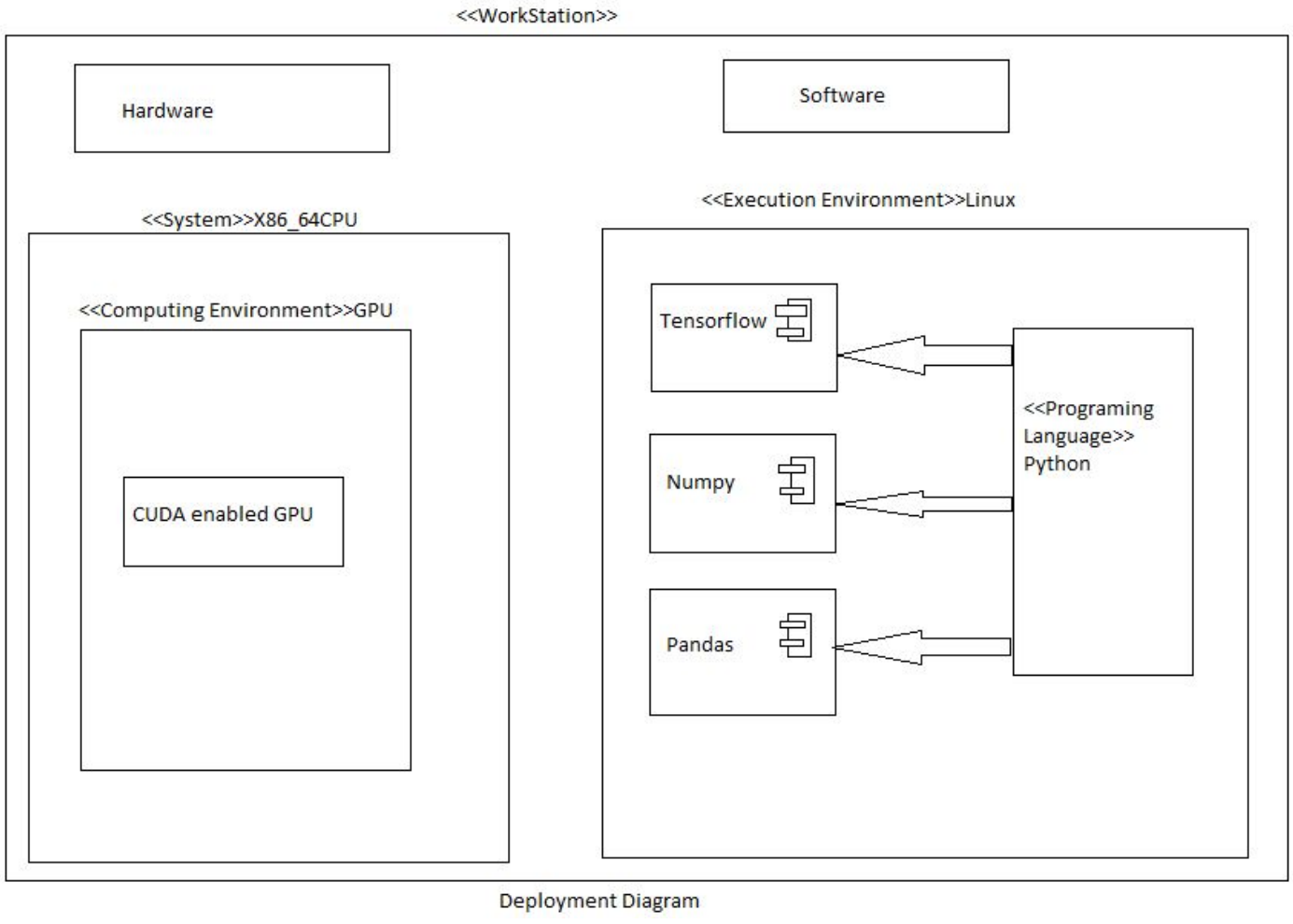


Fig 3.6 Deployment Diagram

Use Case:

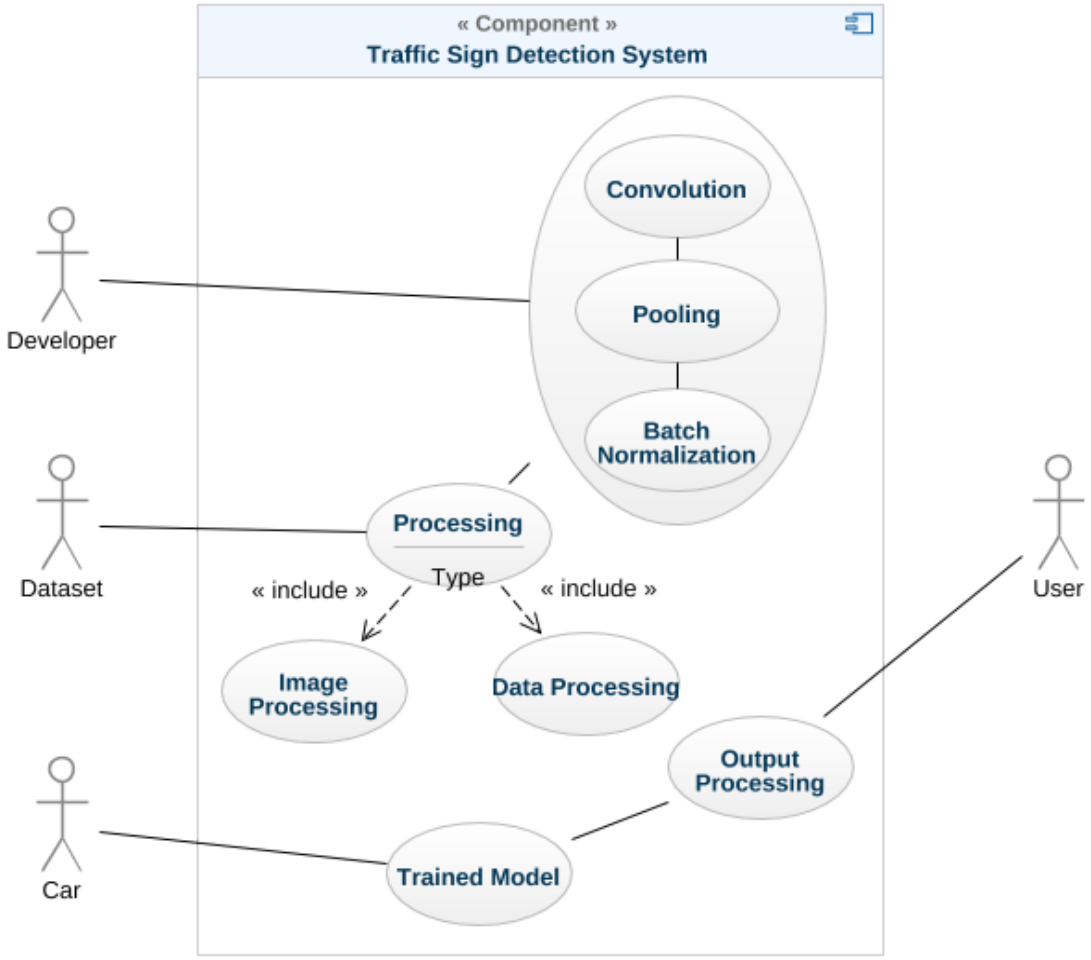


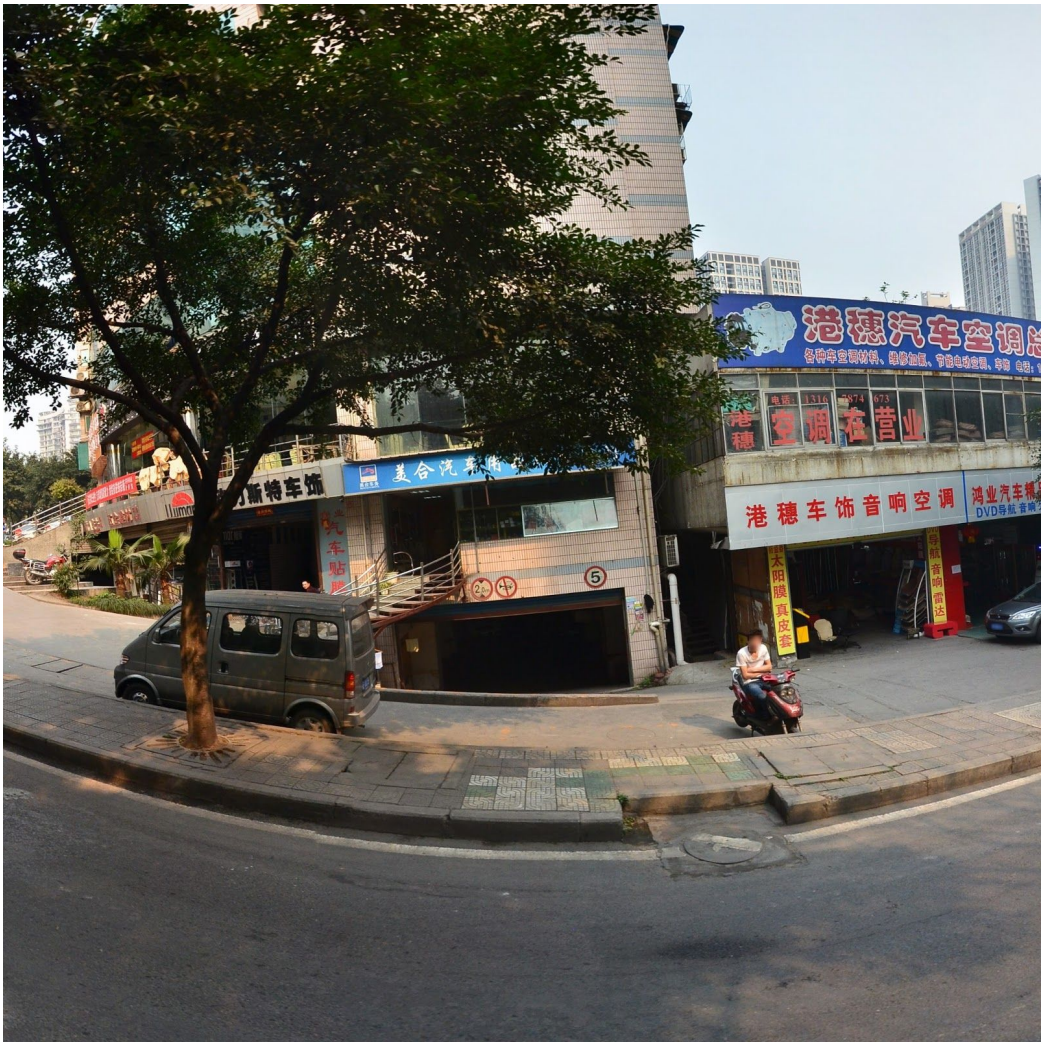
Fig 3.7: Use Case Diagram

Chapter 4: Project Implementation

Data Set:

The most important thing while developing a deep learning application is the dataset. Thus understanding the dataset for creation of the application is very important. The dataset considered for this application contains two things: images and the annotations of the images. The dataset contains many instances of each traffic sign, images were captured both from vehicles and shoulder-mounted equipment at the interval of 10m, and images are captured in different environmental conditions which improve the accuracy of the system. The dataset consists of around 6000 training images and around 3000 testing images of size 2048 x 2048 and the images are in RGB format and each training example in the dataset has the particular annotations stored in the json file. Both the training and testing image folder contains the file ids.txt which contains the ids of the images which is helpful in image processing and reading the annotations from the json file becomes helpful using this file. Each image has zero or more numbers of traffic signs which can be on a different portion of the image and for each traffic sign there is a bounding box coordinate and the class of the box is stored in the json file. Due to the uneven numbers of examples of different classes of traffic

signs, we used a data augmentation technique during training [14, 22]. We simply ignored classes with fewer than 100 instances. This left 45 classes to classify. Classes with between 100 and 1000 instances in the training set were augmented to give them 1000 instances. Other classes that have more than 1000 instances remain unchanged. The example shows how the dataset is created. The annotations of the image below is as follows:



imgs: {"32773": {"path": "test/32773.jpg", "objects": [{"category": "ph2", "bbox": {"xmin": 924.0, "ymin": 1132.0, "ymax": 1177.3333, "xmax": 966.6667}, {"category": "p11", "bbox": {"xmin": 970.667, "ymin": 1128.0, "ymax": 1170.6667, "xmax": 1013.3333}, {"category": "pl5", "bbox": {"xmin": 1146.67, "ymin": 1108.0, "ymax": 1150.6667, "xmax": 1190.6733000000002}], "id": 32773}

Here we can see there are many key value pairs. We will see each key and their particular value one by one. The “imgs” key contains the list of images as the value. The value contains the image id as the key here for example the image id 32773 is the image id and its value is again a list of key-value pairs. The key in this image id value is “path” and “objects”. The value of path is the actual directory where the images are stored in the computer and objects are the lists of bounding boxes and the class labels here denoted as bbox and category respectively. The bbox in a list are the coordinates of the actual rectangular box denoted by the pixel. The bounding boxes have the coordinates of top left corner and the bottom right corner of the rectangular region. The category key has the value as a string which denotes the class of the traffic sign corresponding to that traffic sign. Here in this example we can see the objects list contains three values which denotes there are three bounding boxes in the image. Similarly the whole data is annotated by hand to obtain such a kind of annotations which is useful for further implementations.

Image Processing:

The images in the dataset have size 2048 x 2048 which needs to be reduced to 608 x 608 so that it becomes compatible with the YOLO model. The other reason for decreasing the size of the image is to reduce the number of parameters to be trained while actually training the model. The effects of reducing the image size is not significant as almost all of the high resolution image contains minute details which are irrelevant to the output we want and creates unnecessary burden for training and increases the training time of the model. Thus reducing the size of the image is a better trade off between the training time and capturing the minute details of the image. The image pixel values usually range from 0 to 255 and directly this as an input may cause explosions of number and we may get a not a number error. So it is always better to have a normalization technique such that the pixel values are always in between 0 to 1 real valued. These do not cause explosions and are better for predictions as well. The normalization technique helps in representing all the features using a single scale and equal error weightage. Thus this all causes a way in reducing the training time.

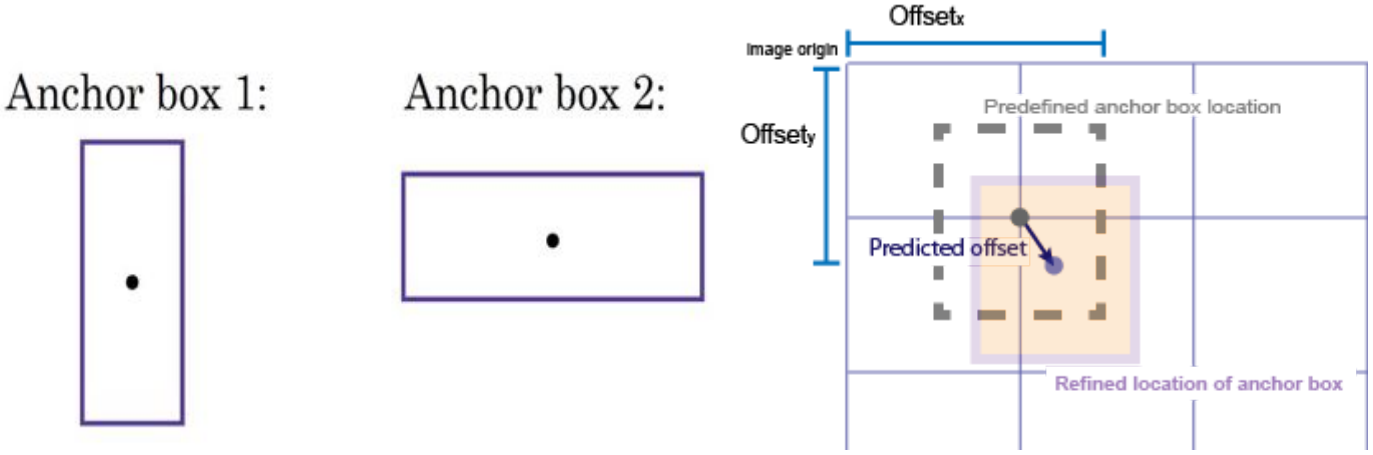
Data Processing:

The data is first read from the json file in a structured way and assumed that the maximum number of bounding boxes in the image are no more than 50. The boxes are read in the format of xmax, xmin, ymax and ymin and the class labels. The mapping is done from the string label into an integer number for better prediction of the class while later representing it in one hot encoding format. The list's first four values are xmin, ymin, xmax,ymax and the fifth value is the class label. After reading and mapping, the next task is to convert the coordinates into a different format that is in the form of midpoint, height and width. The formula used to convert in this are as follows:

x_mid = (x_max - x_min) / 2
y_mid = (y_max - y_min) / 2
height = y_max - y_min
width = x_max - x_min

The reason for conversion is that it provides better stability while prediction and wandering is the lowest thus reducing the error. The other reason is that the actual size of the traffic sign can be greater than the grid cell size thus the entire coordinates finding would become difficult. With using the midpoint exactly one grid cell is responsible for detecting the traffic sign. We can use this grid cell for predicting the bounding box and giving other parameters like height, width and class of the image. The next step of data processing is to normalise the value of pixels which are ranging from 0 to 255. So we divide the value of the pixel by 255 and obtain the pixel values in the range 0 to 1, this would allow the same benefit as that of an image. The data processing is mainly done to help in training by comparing the model output with the preprocessed data. To allow this we convert our data into the exact format which would help in calculating the loss which we further would call as yolo_loss. In this part we find the grid cell to which the bounding box will be in and make a volume having shape 19 x 19 x 5 x 5, where 19 x 19 are the number of grid cells after the convolution model output as the factor of 32 is reduced from the resized image. The next 5 is the number of anchor boxes which are the predefined boxes which defines the shape of the traffic sign. The coordinates of anchor boxes are obtained by analyzing the dataset (i.e we can use k-means algorithm) for size and shape of the bounding boxes. According to this YOLO decided to have 5 anchor boxes based on the shapes of the bounding box in the dataset. Each anchor box is responsible for detection of a particular shape of bounding box. So to calculate which one of the 5 anchor boxes will contain this traffic sign details we calculate the IOU and among the 5 anchors whichever has the maximum IOU we put the coordinates in that anchor. Thus

here we can see each grid cell can predict a maximum of 5 boxes of different shapes. The next dimension is the actual coordinate values which we have obtained from the pre stage where 4 of them are for x_mid, y_mid, height and width for representation of the bounding box and the next one is for class label which is an integer. The important point in this to note is that the midpoints and height and width are respective to the grid cell and not the whole image. We will call this 19 x 19 x 5 x 5 volume as matching true boxes for future references. The height and width in matching true boxes are the log space transforms of the actual values, the reason for this is it provides a better stability and faster convergence. Similarly we create a volume of 19 x 19 x 5 x 1 which we will call a detector mask. This detector mask will have a boolean value as 1 if the object is present in that grid cell and in one of the anchors else it will have the value 0. Thus we create matching true boxes and detector masks for each image in the training dataset and this is used further to calculate the loss and to know the accuracy of the system. Thus the shape of the matching true boxes and detector masks would be m x 19 x 19 x 5 x 5 and m x 19 x 19 x 5 x 1 respectively where m is the number of training examples considered for training.



Model:

The model that we have used is that of YOLOv2 which is generally considered as the state-of -the-art. The summary of the model is:

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            (None, 608, 608, 3)  0
__________________________________________________________________________________________________
conv_1 (Conv2D)                 (None, 608, 608, 32) 864         input_5[0][0]
__________________________________________________________________________________________________
norm_1 (BatchNormalization)     (None, 608, 608, 32) 128         conv_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_23 (LeakyReLU)      (None, 608, 608, 32) 0           norm_1[0][0]
```

```
_____________
max_pooling2d_6(MaxPooling2D) (None, 304, 304, 32) 0           leaky_re_lu_23[0][0]
_____________
_____________
conv_2 (Conv2D)                (None, 304, 304, 64) 18432
max_pooling2d_6[0][0]
_____________
_____________
norm_2 (BatchNormalization)    (None, 304, 304, 64) 256         conv_2[0][0]
_____________
_____________
leaky_re_lu_24 (LeakyReLU)     (None, 304, 304, 64) 0           norm_2[0][0]
_____________
_____________
max_pooling2d_7 (MaxPooling2D) (None, 152, 152, 64) 0
leaky_re_lu_24[0][0]
_____________
_____________
conv_3 (Conv2D)                (None, 152, 152, 128 73728
max_pooling2d_7[0][0]
_____________
_____________
norm_3 (BatchNormalization)    (None, 152, 152, 128 512         conv_3[0][0]
_____________
_____________
leaky_re_lu_25 (LeakyReLU)     (None, 152, 152, 128 0           norm_3[0][0]
_____________
_____________
conv_4 (Conv2D)                (None, 152, 152, 64) 8192
leaky_re_lu_25[0][0]
_____________
_____________
norm_4 (BatchNormalization)    (None, 152, 152, 64) 256         conv_4[0][0]
_____________
_____________
leaky_re_lu_26 (LeakyReLU)     (None, 152, 152, 64) 0           norm_4[0][0]
_____________
_____________
conv_5 (Conv2D)                (None, 152, 152, 128 73728
leaky_re_lu_26[0][0]
_____________
_____________
norm_5 (BatchNormalization)    (None, 152, 152, 128 512         conv_5[0][0]
_____________
_____________
leaky_re_lu_27 (LeakyReLU)     (None, 152, 152, 128 0           norm_5[0][0]
_____________
_____________
max_pooling2d_8 (MaxPooling2D) (None, 76, 76, 128)  0
leaky_re_lu_27[0][0]
```

```
_____________
conv_6 (Conv2D)                 (None, 76, 76, 256)  294912
max_pooling2d_8[0][0]
_____________
norm_6 (BatchNormalization)     (None, 76, 76, 256)  1024        conv_6[0][0]
_____________
leaky_re_lu_28 (LeakyReLU)      (None, 76, 76, 256)  0           norm_6[0][0]
_____________
conv_7 (Conv2D)                 (None, 76, 76, 128)  32768
leaky_re_lu_28[0][0]
_____________
norm_7 (BatchNormalization)     (None, 76, 76, 128)  512         conv_7[0][0]
_____________
leaky_re_lu_29 (LeakyReLU)      (None, 76, 76, 128)  0           norm_7[0][0]
_____________
conv_8 (Conv2D)                 (None, 76, 76, 256)  294912
leaky_re_lu_29[0][0]
_____________
norm_8 (BatchNormalization)     (None, 76, 76, 256)  1024        conv_8[0][0]
_____________
leaky_re_lu_30 (LeakyReLU)      (None, 76, 76, 256)  0           norm_8[0][0]
_____________
max_pooling2d_9 (MaxPooling2D)  (None, 38, 38, 256)  0
leaky_re_lu_30[0][0]
_____________
conv_9 (Conv2D)                 (None, 38, 38, 512)  1179648
max_pooling2d_9[0][0]
_____________
norm_9 (BatchNormalization)     (None, 38, 38, 512)  2048        conv_9[0][0]
_____________
leaky_re_lu_31 (LeakyReLU)      (None, 38, 38, 512)  0           norm_9[0][0]
_____________
conv_10 (Conv2D)                (None, 38, 38, 256)  131072
leaky_re_lu_31[0][0]
_____________
norm_10 (BatchNormalization)    (None, 38, 38, 256)  1024        conv_10[0][0]
```

```
__________________________________________________________________________________________________
_______________
leaky_re_lu_32 (LeakyReLU)      (None, 38, 38, 256)  0           norm_10[0][0]
__________________________________________________________________________________________________
_______________
conv_11 (Conv2D)                (None, 38, 38, 512)  1179648
leaky_re_lu_32[0][0]
__________________________________________________________________________________________________
_______________
norm_11 (BatchNormalization)    (None, 38, 38, 512)  2048        conv_11[0][0]
__________________________________________________________________________________________________
_______________
leaky_re_lu_33 (LeakyReLU)      (None, 38, 38, 512)  0           norm_11[0][0]
__________________________________________________________________________________________________
_______________
conv_12 (Conv2D)                (None, 38, 38, 256)  131072
leaky_re_lu_33[0][0]
__________________________________________________________________________________________________
_______________
norm_12 (BatchNormalization)    (None, 38, 38, 256)  1024        conv_12[0][0]
__________________________________________________________________________________________________
_______________
leaky_re_lu_34 (LeakyReLU)      (None, 38, 38, 256)  0           norm_12[0][0]
__________________________________________________________________________________________________
_______________
conv_13 (Conv2D)                (None, 38, 38, 512)  1179648
leaky_re_lu_34[0][0]
__________________________________________________________________________________________________
_______________
norm_13 (BatchNormalization)    (None, 38, 38, 512)  2048        conv_13[0][0]
__________________________________________________________________________________________________
_______________
leaky_re_lu_35 (LeakyReLU)      (None, 38, 38, 512)  0           norm_13[0][0]
__________________________________________________________________________________________________
_______________
max_pooling2d_10 (MaxPooling2D) (None, 19, 19, 512)  0
leaky_re_lu_35[0][0]
__________________________________________________________________________________________________
_______________
conv_14 (Conv2D)                (None, 19, 19, 1024) 4718592
max_pooling2d_10[0][0]
__________________________________________________________________________________________________
_______________
norm_14 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_14[0][0]
__________________________________________________________________________________________________
_______________
leaky_re_lu_36 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_14[0][0]
__________________________________________________________________________________________________
_______________
conv_15 (Conv2D)                (None, 19, 19, 512)  524288
leaky_re_lu_36[0][0]
```

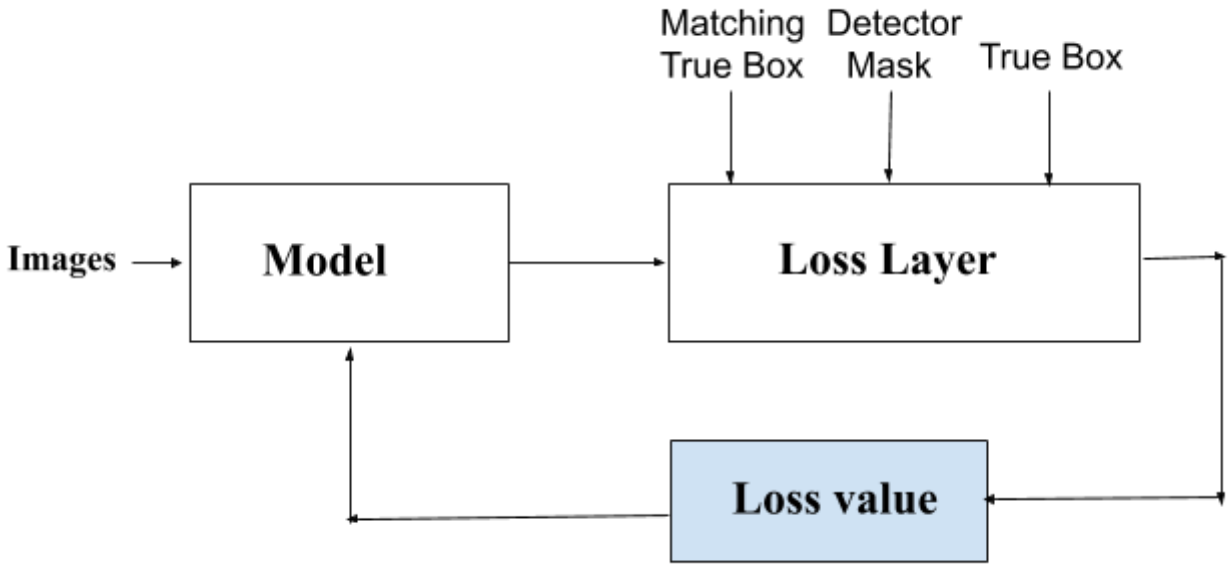
```
__________________________________________________________________________________________________
_________________
norm_15 (BatchNormalization)    (None, 19, 19, 512)  2048        conv_15[0][0]
__________________________________________________________________________________________________
_________________
leaky_re_lu_37 (LeakyReLU)      (None, 19, 19, 512)  0           norm_15[0][0]
__________________________________________________________________________________________________
_________________
conv_16 (Conv2D)                (None, 19, 19, 1024) 4718592
leaky_re_lu_37[0][0]
__________________________________________________________________________________________________
_________________
norm_16 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_16[0][0]
__________________________________________________________________________________________________
_________________
leaky_re_lu_38 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_16[0][0]
__________________________________________________________________________________________________
_________________
conv_17 (Conv2D)                (None, 19, 19, 512)  524288
leaky_re_lu_38[0][0]
__________________________________________________________________________________________________
_________________
norm_17 (BatchNormalization)    (None, 19, 19, 512)  2048        conv_17[0][0]
__________________________________________________________________________________________________
_________________
leaky_re_lu_39 (LeakyReLU)      (None, 19, 19, 512)  0           norm_17[0][0]
__________________________________________________________________________________________________
_________________
conv_18 (Conv2D)                (None, 19, 19, 1024) 4718592
leaky_re_lu_39[0][0]
__________________________________________________________________________________________________
_________________
norm_18 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_18[0][0]
__________________________________________________________________________________________________
_________________
leaky_re_lu_40 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_18[0][0]
__________________________________________________________________________________________________
_________________
conv_19 (Conv2D)                (None, 19, 19, 1024) 9437184
leaky_re_lu_40[0][0]
__________________________________________________________________________________________________
_________________
norm_19 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_19[0][0]
__________________________________________________________________________________________________
_________________
conv_21 (Conv2D)                (None, 38, 38, 64)   32768
leaky_re_lu_35[0][0]
__________________________________________________________________________________________________
_________________
leaky_re_lu_41 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_19[0][0]
__________________________________________________________________________________________________
_________________
```

```
norm_21 (BatchNormalization)    (None, 38, 38, 64)   256         conv_21[0][0]
__________________________________________________________________________________________________
conv_20 (Conv2D)                (None, 19, 19, 1024) 9437184
leaky_re_lu_41[0][0]
__________________________________________________________________________________________________
leaky_re_lu_43 (LeakyReLU)      (None, 38, 38, 64)   0           norm_21[0][0]
__________________________________________________________________________________________________
norm_20 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_20[0][0]
__________________________________________________________________________________________________
lambda_2 (Lambda)               (None, 19, 19, 256)  0
leaky_re_lu_43[0][0]
__________________________________________________________________________________________________
leaky_re_lu_42 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_20[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 19, 19, 1280) 0           lambda_2[0][0]

leaky_re_lu_42[0][0]
__________________________________________________________________________________________________
conv_22 (Conv2D)                (None, 19, 19, 1024) 11796480
concatenate_2[0][0]
__________________________________________________________________________________________________
norm_22 (BatchNormalization)    (None, 19, 19, 1024) 4096        conv_22[0][0]
__________________________________________________________________________________________________
leaky_re_lu_44 (LeakyReLU)      (None, 19, 19, 1024) 0           norm_22[0][0]
__________________________________________________________________________________________________
conv_23 (Conv2D)                (None, 19, 19, 250)  256250
leaky_re_lu_44[0][0]
__________________________________________________________________________________________________
input_6 (InputLayer)            (None, None, 5)      0
__________________________________________________________________________________________________
input_7 (InputLayer)            (None, 19, 19, 5, 1) 0
__________________________________________________________________________________________________
input_8 (InputLayer)            (None, 19, 19, 5, 5) 0
__________________________________________________________________________________________________
yolo_loss (Lambda)              (None, 1)            0           conv_23[0][0]
                                                                 input_6[0][0]
                                                                 input_7[0][0]
                                                                 input_8[0][0]
```

```
==========================================================================================
==============
Total params: 50,804,186
Trainable params: 50,783,514
Non-trainable params: 20,672
```

The model contains a skip connection which is useful to avoid the problem of loss of gradients. The concatenation shown in the above summary has the indication of skip connection. The difficulty of training can be avoided by using the concept of skip connection. The model also uses a lambda layer at the end to calculate the loss function that has resulted in a single value which is nothing but the loss of the model's output. The inputs for this layer are matching true boxes, detector mask, and the model's output. The similar can be represented in the following block diagram:



The last layer of the yolo model needs to be changed as per the number of classes by the formula given as
Number of channels = number of anchors * (5 + number of classes)
Therefore in our case it will become 5 *(5 + 45) = 250 which is the number of channels of the final convolution layer. So each grid cell has 250 dimensional vector which can be broken down into 5 * (5 + 45) for obtaining the dimensions of the bounding boxes. The 5 that is kept constant in the formula is as follows:

1) The first four points denote the midpoints (x, y), height and width.
2) The last one represents the confidence that the object is present in the anchor box of this grid cell. If the confidence (probabilistics value) is high that means the object is present in the grid cell of a particular anchor box. Again the anchor box is selected based on the highest value of IOU when compared to all other anchor boxes.

Loss Function and Training:

The loss function plays a very important role in training the model and adjusting the parameters. The loss function is responsible for transferring the gradient back in the network so that according to the error the weights update themselves in order to produce lesser error. This is known as back propagation. The method used for loss calculation thus becomes important so that ultimately the weights converge and give an output which is within acceptable limits. In this application there are two components which can generate error, those are: localization (detection) and classification. Thus we need to design the loss function in such a

manner that both these losses are given weightage. For this we consider the yolo loss function. The yolo loss function has three components which are:

1) Confidence loss
2) Classification loss
3) Coordinate loss

Before we start the explanation of these loss components, let us first try to understand each component of output and how we could extract each one of them from the output.

Coordinates are the actual value of the bounding boxes as explained earlier and is in the form of midpoints and height, width. The next is the classification where the class of the image is predicted in the one hot encoding form (i.e we apply a softmax function as an activation function for class prediction) and finally the confidence which indicates whether the object lies in the grid cell. The first thing that we should do to calculate the loss is to separate out the coordinates, classes and the confidence of predictions and convert them into percentages. First we create a tensor of 19 x 19 x 1 x 2 which contains the index value of that particular grid cell. Then we add the output from the model to this tensor and divide it by the number of grid cells. By this we obtain the predictions in percentages with respect to the whole image. When we extract each individual information we apply some functions to it so that the predictions are in the proper way. Sigmoid function is applied when we extract midpoint and confidence of the object. Softmax is applied when we try to predict the class of the object and exponential function for height and width to nullify the effect of log taken in matching true boxes. So, for calculating object loss the most important thing is to calculate whether the prediction has detected the object or not. For this we calculate IOU between the predicted boxes which we have extracted in the previous step just explained above and the true boxes. If the IOU is above some particular threshold then the object has been detected and if not there is no object. Thus we represent this IOU of each anchor of each grid cell by a tensor of shape 19 x 19 x 5 x 1 this is equivalent to the detector mask that has been obtained from the data processing stage. The no object loss is evaluated as the below formula.

$$\text{No\_object\_loss} = \lambda_{\text{no\_object}} * (1 - \text{object\_detections}) * (1 - \text{detector\_mask}) * (\text{confidence})^2$$

This loss indicates if the object is not present what is the confidence value of it which is considered as the loss of the model. (Only true negative case is considered here)

The object loss is similar to no object loss but the formula is as follows:

$$\text{Object\_loss} = \lambda_{\text{object}} * (\text{detector\_mask}) * (1 - \text{confidence})^2$$

Thus the confidence loss is given by the summation of object loss and no object loss. The weightage of these losses are decided by the lambda coefficient multiplied to the formula. The value denotes which loss is more important than the other and based on it the model will predict the object.

The classification loss is nothing but the square errors instead of using categorical crossentrophy yolo loss function uses this method. The loss is given by:

$$\text{Classification\_loss} = \lambda_{\text{class}} * (\text{detector\_mask}) * (\text{original class} - \text{predicted class})^2$$

The coordinate loss is calculated very similarly using the formula as shown:

$$\text{Localization\_loss} = \lambda_{\text{coord}} * (\text{detector\_mask}) * (\text{matching true boxes} - \text{prediction boxes})^2$$

Thus the total loss is calculated by adding up all the individual loss and the final loss becomes:

$$\text{Total loss} = \text{Classification Loss} + \text{Coordinate Loss} + \text{Confidence Loss}$$

To reduce this loss value, we must train the model in order to reduce the loss value. The training configuration used for this model is:

Learning Parameter = 0.01

Number_of_epochs = 5000

Optimizer = Adam

Batch_Size = 1
Validation_Size = 0.2
The model was trained for a particular chunk of data then we saved the model weights and again reloaded it for training on a newer chunk of data.

Output Processing:

This phase is done after the model has been successfully trained and to visualize the actual output of the model. After the model has predicted the boxes our job is to collect only those boxes which are correct. For this we perform some operations. First the predictions which are in the format of midpoint, height and width are converted back into left top corners and right bottom corners. After the conversion we eliminate all those boxes which have confidence less than a particular threshold which is in our case 0.5. The confidence is obtained by multiplying the class probability and object probability. Then we apply the generated mask to obtain the boxes, scores, classes. Henceforth we lose the concept of grids and volumes and we get a simple array of boxes which is similar to the dataset annotation after reading. Although the boxes predicted do not contain the actual coordinates of the image. Thus we need to scale up the boxes to the size as that of the original image. The boxes predicted are redundant and may have multiple bounding boxes for the same object. Thus we have to apply a technique of non- max suppression where the boxes which overlap significantly are deleted i.e. the bounding boxes which have IOU greater than 0.6 when compared to other relevant boxes are deleted. Thus we obtain the final bounding boxes of the traffic sign of a given input image.

Draw Boxes:

The final boxes obtained needs to be plotted on the image for visual correction of the model's prediction. By using this function we bound the prediction to that of image size that is to (2048, 2048) and we finally plot the rectangle of each traffic sign on an image. Moreover we also plot the predicted class label of the traffic sign and human level judgement can be performed.

Chapter 5: Results

The output of the input image is shown as below which was obtained after training the model. The below image depicts the same.

The training loss can be seen in the following execution of the fit function (Initial Stages):

```
Train on 282 samples, validate on 71 samples
Epoch 1/10
282/282 [==============================] - 174s 616ms/step - loss: 43.5412 - val_loss: 857265.5720
Epoch 2/10
282/282 [==============================] - 161s 572ms/step - loss: 34.0406 - val_loss: 155.0734
Epoch 3/10
282/282 [==============================] - 162s 573ms/step - loss: 21.6925 - val_loss: 28.1841
Epoch 4/10
282/282 [==============================] - 162s 573ms/step - loss: 18.9973 - val_loss: 21.7447
Epoch 5/10
282/282 [==============================] - 162s 573ms/step - loss: 18.9911 - val_loss: 25.5684
Epoch 6/10
282/282 [==============================] - 162s 573ms/step - loss: 19.7652 - val_loss: 18.3363
Epoch 7/10
282/282 [==============================] - 161s 573ms/step - loss: 18.8958 - val_loss: 19.2583
Epoch 8/10
282/282 [==============================] - 161s 569ms/step - loss: 18.5644 - val_loss: 17.9724
Epoch 9/10
282/282 [==============================] - 161s 569ms/step - loss: 17.7430 - val_loss: 17.6259
Epoch 10/10
282/282 [==============================] - 161s 570ms/step - loss: 17.6863 - val_loss: 17.1555
```

The training error can be reduced by further training and considering more dataset.

Chapter 6: Conclusion and Future Work

The above implemented project showed sufficient results and the bounding boxes obtained after the output processing and draw boxes were considerable. The trained model can easily be integrated with the other model which can help in making the vehicle autonomous. The model shows sufficient prediction results within acceptable limits and the classification of traffic signs shown on the image helps the other model to act accordingly. For example, a speed limit traffic sign could be properly classified and can help the

autonomous vehicle to lower the speed such that it can avoid catastrophic events. Such an integration module can help the evolution of driverless cars. The prediction time of all objects in an image is very less as the algorithm is you only look once and in one go all the objects can be detected within no time and minimal error. Although the project had some limitations like it can detect only traffic signs but it is an example of how a module can be implemented and easily integrated with the other modules which help in autonomous vehicles. The model can though be trained for other objects but this requires a lot of data and training time. Thus a full fledged model can be created out of it with some easy tuning. The model can even be extended for different traffic signs and more number of anchor boxes can be considered.

References:

[1] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li and S. Hu, "Traffic-Sign Detection and Classification in the Wild," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 2110-2118, doi: 10.1109/CVPR.2016.232.

[2] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.

[3] Y. Zhong, J. Wang, J. Peng and L. Zhang, "Anchor Box Optimization for Object Detection," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Village, CO, USA, 2020, pp. 1275-1283, doi: 10.1109/WACV45572.2020.9093498.

[4] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng. An empirical evaluation of deep learning on highway driving. CoRR, abs/1504.01716, 2015.