Maynooth University
National University of Ireland Maynooth

MSc Data Science & Analytics Dissertation

Traffic Sign Recognition using Deep learning in MATLAB (CNN using Transfer Learning)

Author:
SUJIT KRISHNANKUTTY
Student Number: 20250254

Supervisor:
Dr. ADAM WINSTANLEY

A thesis submitted in fulfillment of the requirements for the degree of MSc in Data Science and Analytics
2020-2021

in the

Department of Computer Science
Maynooth University

ACKNOWLEDGEMENTS

The completion of this project has required an amalgamation of efforts and ideas from a lot of sources. In the end, I wish to acknowledge the intellectual input and support extended to me by all those directly or indirectly involved with this project.

I owe a profound sense of gratitude towards my supervisor Dr. Adam Winstanley for giving me an opportunity to work under his guidance and for all the valuable guidance provided by him during the course of the project work. Without his informative suggestions and critical analysis, this project could not have been done. I thoroughly enjoyed those seasons where he helped me to resolve critical issues with simplest of approaches.

I would especially like to extend my heartfelt thankfulness towards my mentor for his constant encouragement and critical comments throughout the project. All through the duration of the project; his approachability made it remarkably easy for me to work and seek his inputs. I would also like to thank open-source development platforms like Kaggle, Github, and forums like Mathworks from the bottom of my heart. These platforms really helped me to learn new technologies and saved my life when I had to fix critical issues while in the development phase.

DECLARATION

I hereby certify that this material, which I now submit for the assessment on the program of study as part of MSc in Data Science and Analytics qualification, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: SUJIT KRISHNANKUTTY Date: 08-08-2021

ABSTRACT

Traffic-Sign Recognition is a widely studied domain by researchers, especially in the development of automated driving and ADAS (Advanced driver-assistance systems) technologies. Traditional computer-vision solutions and image recognition algorithms fails to recognize road signs accurately due to its limitations. As a result, there is a need to build an intelligent and automated system that can accurately recognize and detect the traffic-signs, ultimately aiding the vehicle to make strategic decisions and executing it without human intervention. In this research work, a Deep Learning solution using Transfer learning approach is introduced for developing a robust traffic-sign image classification model. A modified version of pre-trained convolutional neural network GoogleNet is employed for the implementation of Transfer Learning. The proposed CNN architecture is designed to extract key features from the traffic-sign images and differentiate the images under labelled classes of traffic signs. An overall accuracy of (87% - 94%) were evidently exhibited by the network, and with the inclusion of validation set the model exhibited more than 95%, tested under different hyperparameter constraints. The standard GTSRB dataset containing multiple classes of traffic signs is used for conducting the network training and for achieving a good overall accuracy.

Keywords – Traffic-Sign Recognition, Automated Driving, Deep Learning, Image Recognition, CNN, Transfer Learning, GoogleNet

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

Chapter 1 - Introduction

1.1 Autonomous and Assisted Driving

Today, driving a vehicle has become increasingly common in people's lives. As a result, the number of vehicles on the road today is increasing at a rapid pace. Simultaneously, the number of roads and traffic signs has increased significantly in different countries. Drivers are expected to be aware of all road traffic signs and pay close attention to these signs while driving their cars. Traffic sign proves to be an important aspect for controlling and managing traffic, providing various indicators such as warning, road conditions, speed limit and much more which ultimately helps in preventing road accidents (Ai, C., 2013). Taking its significance into account, we can develop an intelligent transportation system that helps in recognizing road traffic signs using computer vision and image processing techniques which would help an individual in making better decisions. These systems are deployed in autonomous vehicles as a part of Advanced Driver Assistance Systems (ADAS).
(Fan & Zhang 2015).

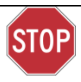
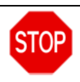
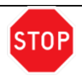
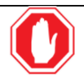
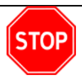
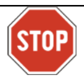
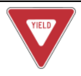
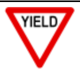
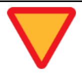
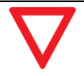
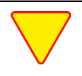
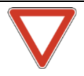
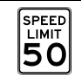
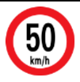
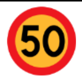
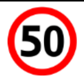
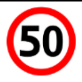
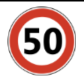
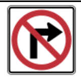
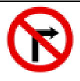
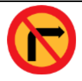
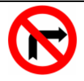
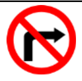
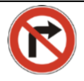
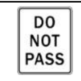
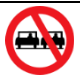
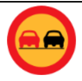
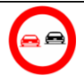
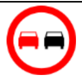
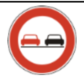
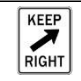
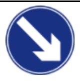
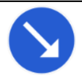
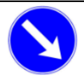
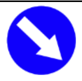
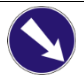
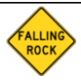
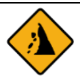
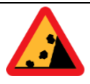
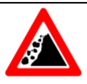
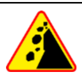
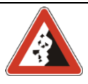
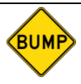
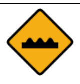
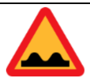
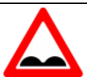
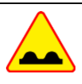
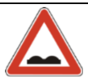
1.2 Challenges of Driver Assistance Systems

With the rapid advancement of science and technology, autonomous vehicles will be widely used and developed in the future, employing sensors and computer vision software for improved safety and control as predicted by (Ramler and Ziebermayr 2017). Based on previous studies, researchers are trying to implement and develop different methods, algorithms, and innovative approaches to build an optimized traffic sign recognition system. Understanding the environment around a moving vehicle is a complex multi-faceted problem.

A small part of the problem of (semi-)automatic vehicles is the recognition and identification of traffic signs. In a traffic-sign recognition system, a camera is mounted in a vehicle which helps to capture an image of a traffic scene, which is then processed to detect and recognize traffic signs. At present, there are quite a few car manufacturing companies that have installed such systems in the cars, which can detect and recognize at the most 2-3 categories of traffic signs- max speed limit, stop sign, no vehicle overtaking signs. It is quite a challenging task to add more classes of traffic signs as different countries have different representations of road signs in terms of colour, shape, and pictorial representation. (Toth 2012)

TABLE I

TRAFFIC SIGNS COMPARISON FOR DIFFERENT COUNTRIES

	USA	**Ireland**	**Sweden**	**Israel**	**Poland**	**Slovakia**
Stop and give way						
Yield (give way)						
Maximum speed limit						
No turn						
No overtaking						
Keep right						
Falling rocks						
Bump						

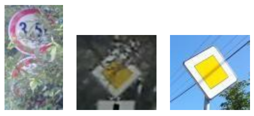
This is a Table from the research work by (Toth 2012)

The (Table-I) gives an apt representation of a comparison of different classes of traffic signs for different countries. It can be observed, even though the traffic signs seem to be similar for various countries, there are slight differences in terms of representation. Along with that, if we consider the additional factors such as camera resolution, image/video quality, climate conditions, wear and tear of signs, it is quite difficult to build a robust system that accurately detects and recognizes the traffic signs.

We would need to create an image datastore containing millions of images for each country individually as there is a high possibility that the images may get misclassified if we merge traffic sign images for every countries together. The model detects the traffic sign and extracts the key features from each image and classifies it accordingly. As simple as it may sound, research studies show that in real world, it is quite a cumbersome task to collect an image dataset from different angles, at different weather conditions, different camera resolutions. Some of the difficulties and challenges faced while detecting and recognizing the road traffic signs have been discussed in (Table-II). (Toth 2012)

TABLE II

DIFFICULTIES AT TRAFFIC SIGNS RECOGNITION

Difficulties	**Traffic signs images**
Different Lighting conditions (Day, night, backlight)	
Different Climate conditions(rain / snow / fog)	
Color fading	
Occlusion and shadows (trees/object)	
Damaged Traffic-Signs (vandalised / wear and tear / rusted / over time)	
Complex traffic signs – Multiple Road signs	
Incorrect traffic signs	

This is a Table from the research work by (Toth 2012)

- Different Lighting conditions – Due to different lighting scenarios, images captured during day and night would have some differences and the key features extracted the model would vary. Shape based recognition would seem to be a good option for handling such scenarios.
- Different Climate conditions – The images captured in different weather conditions are also influenced by heavy rain, fog, snow which makes it difficult to recognize the exact traffic sign.
- Color Fading – Over time, the colours of the traffic signs may fade away due to seasonal weather changes.
- Shadows / Object blocking – It is one of the most common issues faced while image capturing as a lot of objects may be blocking the traffic signs or responsible for casting their shadow on road signs. For instance, shadows cast on traffic signs due to transmission lines can result in different interpretations. Due to object blocking, the images will not be detected and may result in the driver making wrong decisions.
- Damaged traffic signs – The traffic signs can also be damaged by other factors such as road accidents, strong rains, wind, storms, rust, tilt. The road signs can also be vandalized by some people.
- Complex Road signs – In some of the traffic scenes, it can happen multiple images are detected and for the model to make decisions in real-time, it would need high computational complexity for image processing. It can also result in misclassification of traffic signs.
- Incorrect signs – Sometimes, traffic signs are placed in an incorrect way which may result in different meanings.

1.3 Traffic Sign Recognition using Deep learning

Motivated by the success of Deep Learning-based classification and recognition methods in various domains, I chose to implement this approach using Convolutional Neural Networks (CNN) and Transfer Learning with an aim to create a robust classification model focussing only on traffic sign recognition without considering the traffic sign detection step. This project can be directly used in self-driving cars and the automotive industry to improve computer vision models in automation systems.

For developing a Deep Learning Network specifically for Image classification, we would be using convolutional neural networks which work best for Image classification problems because of their high accuracy. A Transfer Learning approach is preferred as it would prove to be an efficient solution in this Deep Learning problem instead of training the network model from the scratch as in the latter case, we need millions of images to train, one should have good knowledge and experience working with network architecture and also a higher computational power (GPU) is required. With the help of MATLAB Deep Learning Toolbox and pre-trained networks available we have achieved an optimized Deep Learning Model by testing with different tuning options. Also, the error rate or misclassification rate have been identified with the help of a confusion matrix of predicted results with the actual results.

1.4 Objectives

In this research, a Deep Learning approach is used to build a Traffic Sign Recognition system designed to extract the key features from the images and classify them according to different categories. The goal of this research is to design a Deep Learning neural network that can easily recognize some of the basic road-traffic signs and should be able to distinguish various categories of road signs. The main objectives of the project are listed out below:

- To develop a Deep Neural Network designed for Image Recognition and Classification.
- To identify whether the trained network performs well on Test data with good overall accuracy.
- To check for the performance of the trained network on images with multiple road signs in a single image.
- To investigate the misclassified images obtained on testing with new data and the impact of testing the model with different tuning options available on the performance of the Neural Network.

1.5 Outline of Thesis

In chapter 2, we focus on the background and related work of Traffic-Sign Recognition and its past implementations using different approaches. Some work related to Transfer Learning and Image Recognition is described, along with the programming language used and the tools available for Deep Learning. In chapter 3, we discuss the architecture of the network, GoogleNet architecture, different layers of the pre-trained network in detail, conceptual design working of the model, the MATLAB Deep Learning Toolbox, and the working of Convolutional Neural Networks.

In chapter 4, we describe the actual implementation of Deep learning using convolutional neural network, setting up the environment, data collection, image pre-processing techniques applied, image categorization, pre-processing data using image datastore, training simulation in MATLAB, the impact of varying the tuning options available in Deep learning toolbox. In chapter 5, the analysis and the evaluation of the results are discussed in detail. In Chapter 6, the final chapter, we conclude the work, including suggestions for future work and applications.

Chapter 2

Background and Related Work

Over the last decade, we have seen an increase in the development of innovative and intelligent transportation systems and has been an area of intensive research particularly self-driving cars and ADAS. Many researchers have explored different ways to tackle the image detection and recognition problem for traffic signs using various datasets, machine learning techniques, Deep Learning approaches, and computer vision to achieve a model with good overall accuracy and appropriate methodology. This section covers some of the previous studies and research work related to traffic sign detection and recognition using Image Recognition and Transfer Learning.

2.1 Traffic sign detection using Image Recognition

Image Recognition is playing an increasingly larger role in modern life, like driver assistance systems, which need to know whether the camera installed in the car for detecting the images is looking at another vehicle, a pedestrian, a tree, or a traffic road sign.

Firstly, we will focus on the research work of authors (Djebbara Yasmina and Rebai Karima et al. 2018), where the paper talks about creating a traffic sign recognition system using the Deep Learning approach. A rich dataset was captured for the research purpose of road-traffic sign classification. The neural network used for the classification was LeNet-5 which utilizes two main layers constructs, one for performing convolution and the second one for subsampling. Tensorflow was utilized for building and training the LeNet-5 network. The results obtained from the LeNet-5 network demonstrate the efficacy of the developed method, as 97% validation accuracy was achieved, though the network faced some challenges to correctly classify some categories of images.

The research work of (Saleh K.H. Aly et al. 2013) describes a new approach for traffic signs recognition by utilizing image tampering and augmentation techniques based on the Gaussian blur principle. The author attempts to resolve the inconsistencies caused by Gaussian blur with the help of motion blur and defocussing blur methods for obtaining the blurred image dataset. As in most cases, the vehicle would be in motion, and images obtained while image capturing would be probably blurred by a variety of elements such as camera exposure, low image resolution, bad weather conditions, vehicle motion. The model works on the Local phase quantization method for capturing blurred invariant features of images and it evidently proved to be very effective at recognizing different types of blurring such as Gaussian blur and motion blurring.

The next paper from (Xiong Changzhen et al. 2016) describes a recognition model for road signs with the highest accuracy of 99% which focuses on a Deep convolutional neural-network approach with the help of RPN (Region Proposal Network) implemented in Faster R-CNN. The proposed method was capable of detecting the seven major categories of Chinese road signs. The results evidently show that our algorithm detects traffic signs at a rate greater than 99 percent of the time and in video sequences where the image detection is in real-time. A pre-trained network was used for initializing the neural-network weights and the Faster R-CNN method was utilized for fine-tuning the model.

In the research work by (Mahatme and Kuwelkar 2017), a system was developed for performing the two tasks, road sign detection, and recognition. The algorithm employed was different from the standard shape detection and colour segmentation techniques as it focussed on converting the RGB images to Red colour segmented images. The implementation of this model was performed in MATLAB and a single layer feed-forward perceptron neural network was designed for implementing the image recognition part. The trained network successfully detected Indian road signs with an overall detection accuracy of 93% with an excellent computational power as MATLAB Parallel-processing toolbox was used which helped the model to be trained on GPU instead of a single core.

2.2 Transfer Learning

The process of adapting a previously trained neural network to new test data by transferring the learned features to the modified network is referred to as Transfer Learning. There have been various studies and research conducted for implementing Transfer learning to different domains, as the time required to train the model is comparatively less as compared to training the models from the scratch. Also, anyone with a basic understanding of Deep learning can built a robust and effective neural network suited for any specific use-case.

The model proposed by (Chunmian Lin et al. 2019) uses an Inception-v3 model for recognizing and classifying new images by modifying the components of the network architecture i.e fully connected layers and tuning parameters of network layers. The network was trained for recognizing Belgium traffic signs and as transfer learning approach was used, the model was retrained at different epoch values and with lower learning rates. An accuracy of more than 99% was obtained for the trained network while the learning rate was set at 0.05, which evidently shows that transfer learning is a very powerful and robust method for road sign recognition. The author has also recommended implementing hyperparameter optimization which would improve the overall efficiency of the trained model.

Another research work by (Liu Wei et al. 2018), proposed a Transfer learning solution based on an SVM classifier for classifying the different categories of Chinese road signs. The data was collected from multiple sources and the neural network was trained on the R-CNN algorithm. For image detection and higher computational power, different toolboxes such as Image Processing, Parallel Computing, GPU Coder, Computer Vision, available in MATLAB software were utilized for this research. With the help of Parallel computing and GPU, the author was able to train more images which aided the model to achieve 99% Precision and 95% recall as calculated from the confusion matrix.

An efficient approach by scaling the Convolutional neural networks is proposed by authors (Tan and Le .,2019) which focuses on identifying and scaling the dimensions of the network such as the network depth, network width, and resolution coefficients and ultimately improving the performance of the neural networks. A new neural network architecture called EfficientNets was implemented for this research work which demonstrated much better accuracy and efficiency over the traditional Convolutional Neural networks. An effective compound scaling method was implemented which enabled to build a new baseline network by scaling up the resource constraints of the original network to gain a higher accuracy over other networks.

There was an alternative approach to traditional Transfer learning methods suggested by the author (Xishuai Peng et al. 2017) which has been implemented using the Generalized Auto-Encoder algorithm. In this method, both the source and the target data were corrupted, and the domain-invariant data was achieved by filtering out the noiseless data which was the closest target data. A Deep non-linear architecture was designed in order to preserve both statistical and geometric properties at the same time. The model was compared with some of the standard traditional Transfer learning methods such as PCA, ITML and as predicted GAE algorithm outperforms all of the methods in recognizing most of the images.

2.3 Deep Learning with MATLAB

In this research work, a traffic sign image recognition system is built using Deep Learning with a convolutional neural network and the implementation is performed in MATLAB. Deep Learning techniques can be applied to any domain or any project using MATLAB language. MATLAB provides a platform for developing and implementing algorithms, pre-processing and labelling the data as per specific requirements, code-based simulation, and installing it in complex and standalone AI systems.

There are different in-built Deep Learning architectures, pre-trained models (trained with millions of images), and Deep Network Designer tools which help us to develop, modify and optimize a Deep Learning network as per specific user needs. The algorithms can also be accelerated with the help of graphical computational hardware such as NVIDIA GPU which are detected by MATLAB software once installed in any machine and requires no special programming. MATLAB provides a wide range of integrating options with other platforms such as the TensorFlow framework, PyTorch, Keras, and MxNet. MATLAB provides a simulation-based environment for testing the architecture with optimizations and for testing out the network with a new test dataset. For implementing Deep Learning in MATLAB there are some pre-requisites in the form of packages and toolboxes which need to be installed prior to writing out the MATLAB code such as Deep Learning Toolbox, Parallel computing, MATLAB coder, Image processing toolbox, Reinforcement Learning.

There are other programming languages such as Python and R language which performs image processing, Deep Learning, and artificial intelligence and provide excellent results. Implementation of Deep learning and image processing in Python and R are quite easier as both have easy to read interface and both are open source, i.e., free for use, whereas MATLAB is a closed proprietary product. In Python, image processing is implemented using external libraries and packages. Numpy and SciPy libraries of Python contain a plethora of image-processing algorithms. On the other hand, MATLAB provides a multi-paradigm computing environment combined with a desktop interface. It also contains a live simulation environment for performing model training on different algorithms, iterative analysis, and a live editor that combines code and output in a notebook-like interface. Hence, we have implemented Deep Learning for traffic sign recognition in MATLAB due to its exceptionally advanced features available for computing and image processing.

Chapter 3 Conceptual Design and Architecture

3.1 Network Architecture

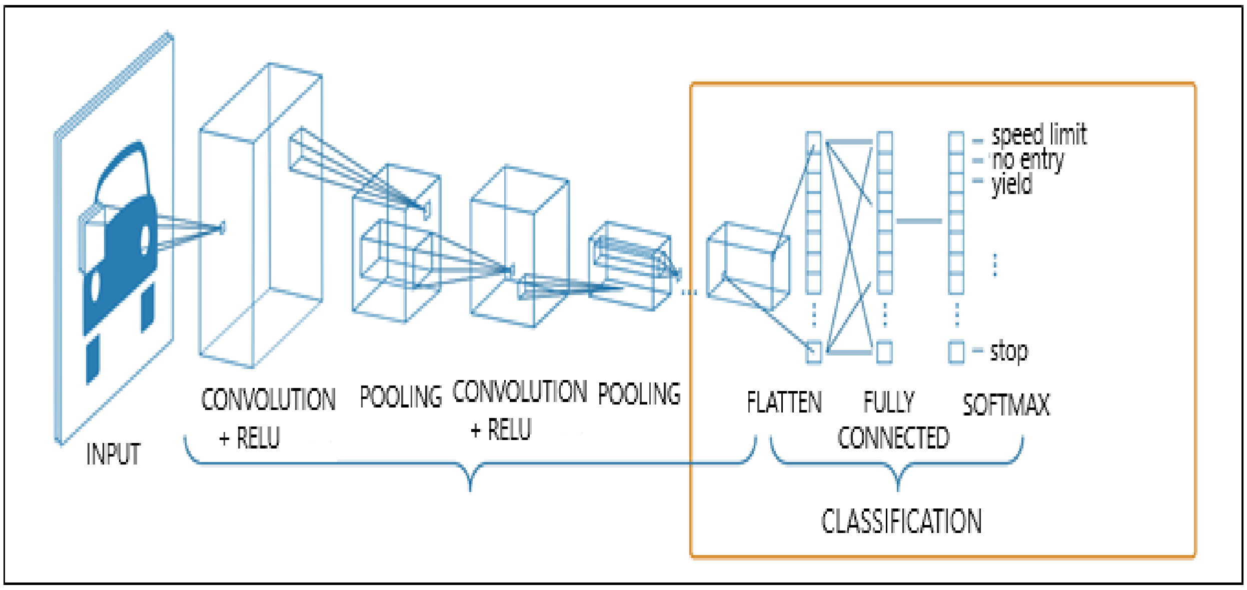


Fig 1. Deep Learning Network Architecture for Image Classification

In this research project, Deep Learning is implemented using pre-trained convolutional neural networks. Convolutional neural networks are a powerful and widely used type of Deep Learning network that is developed for specialized applications in which the input has a 2D structure, for instance, image dataset. The figure-1 gives a brief idea about the network architecture of CNN. It comprises the input layer, the intermediate layers (referred to as hidden layers where the activation function performs the masking of inputs and outputs), and an output layer. The activation function normally used in most of the scenarios is the ReLU function as it outputs the input directly with no transformations if the input is positive, else it returns an output zero. The ReLU function solves the issue of vanishing gradients which allows the network to learn faster resulting in better performance.

A Convo-Net architecture can vary depending upon the number of layers present and the types of components incorporated in the network. As we are interested in obtaining a categorical response, i.e., recognizing the traffic sign from a given set of traffic sign images, we must have a classification function as well as a classification layer for classifying the images into different categories. The neurons present in each network layer are arranged in a three-dimensional way, converting a three-dimensional input into a three-dimensional output.

For instance, consider the 1st layer i.e., the input layer holding the image data as 3-D input, while their height, width and color are considered as their dimensions, the layer's neurons connect to the areas of the images and transforming them into a three-dimensional output with the help of feature extraction.

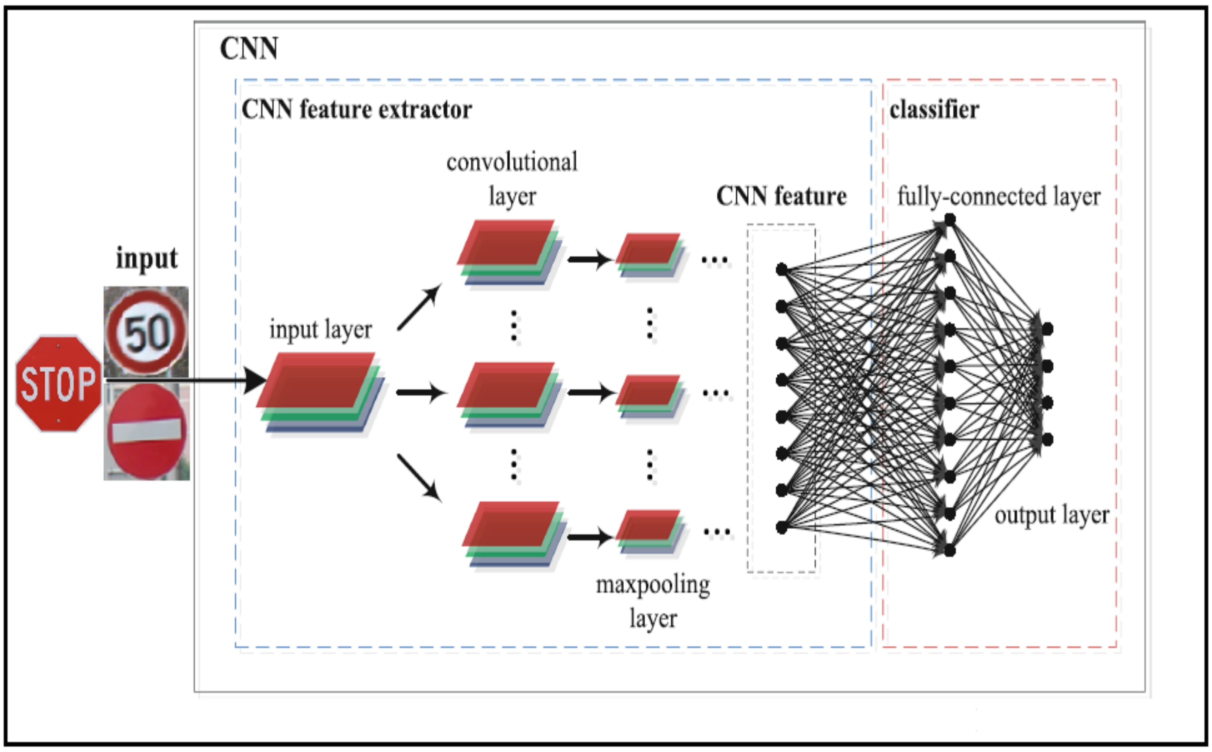


Fig 2. Feature Extractor Architecture (Zeng and Fang et al. 2015)

The figure-2 gives another perspective to look at the architecture where it is comprised of two major components. The first component has layers that alternate between convolution and max-pooling. The pooling layer is responsible for performing a specified functions such as max pooling, where the layer takes the maximum value from a particular filtered region. Each layer's input is simply fed from the output of the preceding layer. Hence, a hierarchical feature extractor is generated which converts the original input into feature vectors. The extracted feature vectors are then classified with the help of the second component i.e., fully connected layers which are also responsible for flattening the results before classification.

Feature extraction is the process of decreasing the number of resources needed to describe a huge amount of data. One of the primary issues with completing a complicated data analysis is the large number of variables being involved. Hence, when we train a network with more complex datasets which contain millions of RGB coloured images, we might need to consider a complex network model with multiple convolution and fully connected layers.

3.2 Conceptual Design Working

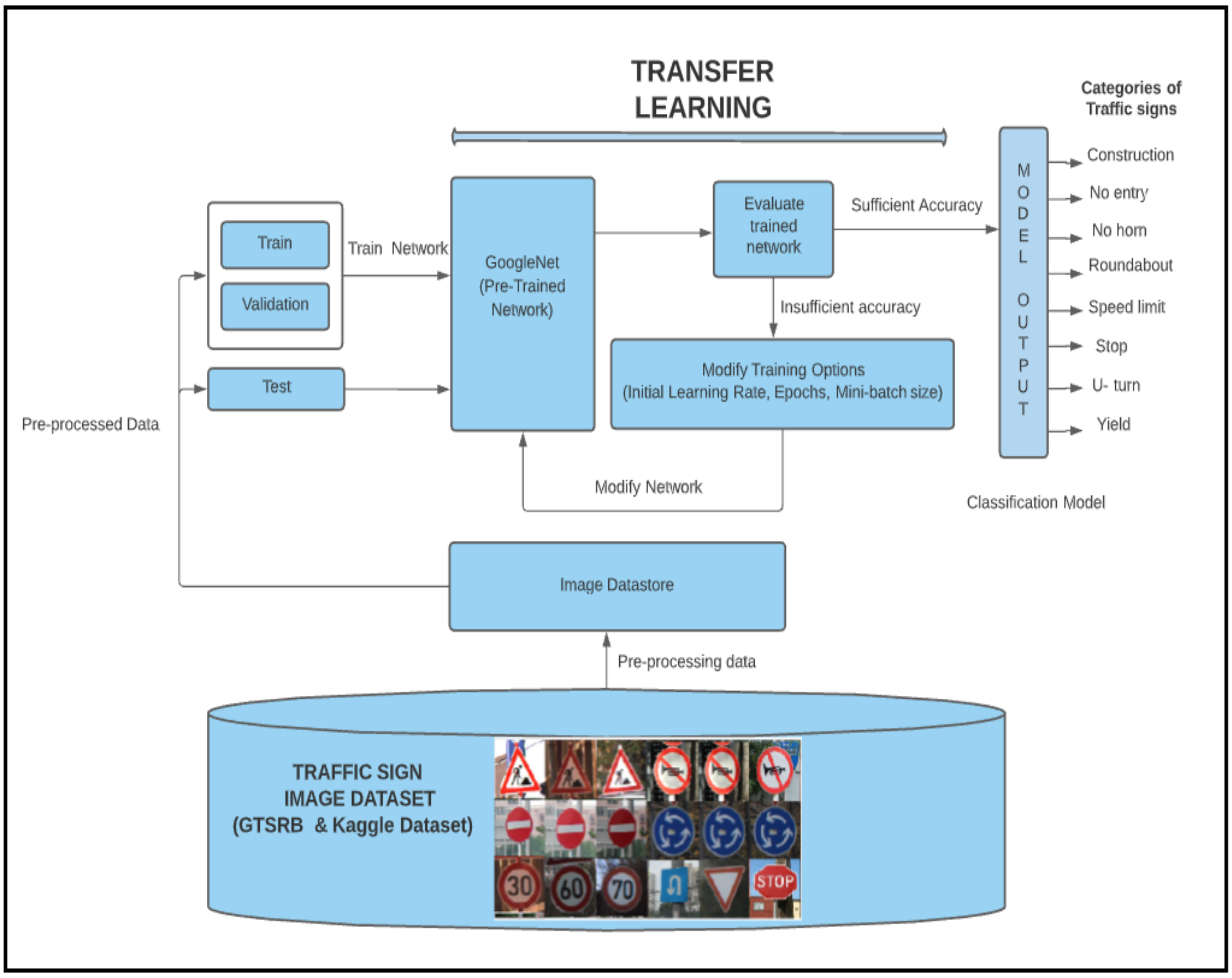


Fig 3. Workflow Design of Traffic Sign Recognition system

In this section of our research, we cover the conceptual working of the traffic sign recognition model with a brief introduction of every step involved in it. The traffic sign image datasets are gathered from multiple sources (Kaggle & GTSRB). Once the data is collected, we need to process it before the training stage as there is a high probability that the images might have different dimensions. As we need to process all the images as per specified required criteria of the pre-trained convolution neural network, the dataset is passed on to the imageDatastore stage. *(Fig 3 gives a brief idea about working design of the model)*

Basic pre-processing steps can be performed using imageDatastore function. The transformed and pre-processed data is then split into training data, testing data, and validation data in a ratio of 70:15:15. This step is performed to avoid model overfitting and enhancing the model accuracy. In the next stage using trainNetwork function in MATLAB, and using GoogleNet (pre-trained network), the model is re-trained with traffic signs images. Once the model is trained, the model is evaluated using test data and checked for its accuracy. The training options can be modified using the training options available in the trainNetwork function. Finally, an output model is obtained after achieving a good accuracy score which can confidently recognize some of the basic traffic road signs. Using the confusion matrix, we can identify the misclassified images and study patterns to improve the overall efficiency of the traffic sign recognition model.

3.3 GoogleNet Architecture

One of the greatest discoveries in the study of Deep Neural Networks was the Inception Network. GoogleNet is one of the variants of the Inception network and as the name suggests, it was introduced by the Google research team. With the introduction of Conv-net, large datasets, accessibility to huge computational power, and inherent network architectures, efficient solutions have been developed in the field of computer vision. The researchers have discovered that increasing the number of layers and components in a network improved the performance significantly. However, everything comes at a cost, and it was found that adding layers to larger networks make them prone to overfitting the data and are sometimes affected by vanishing gradient problem.

The GoogleNet architecture solved the majority of the problems faced by complex networks, primarily through the use of the Inception module. The inception module is a network architecture that uses convolutions with multiple size filters to identify features at various scales which makes the network architecture wider instead of deeper and it also utilizes dimensionality reduction techniques to lower the computing cost of a large network.

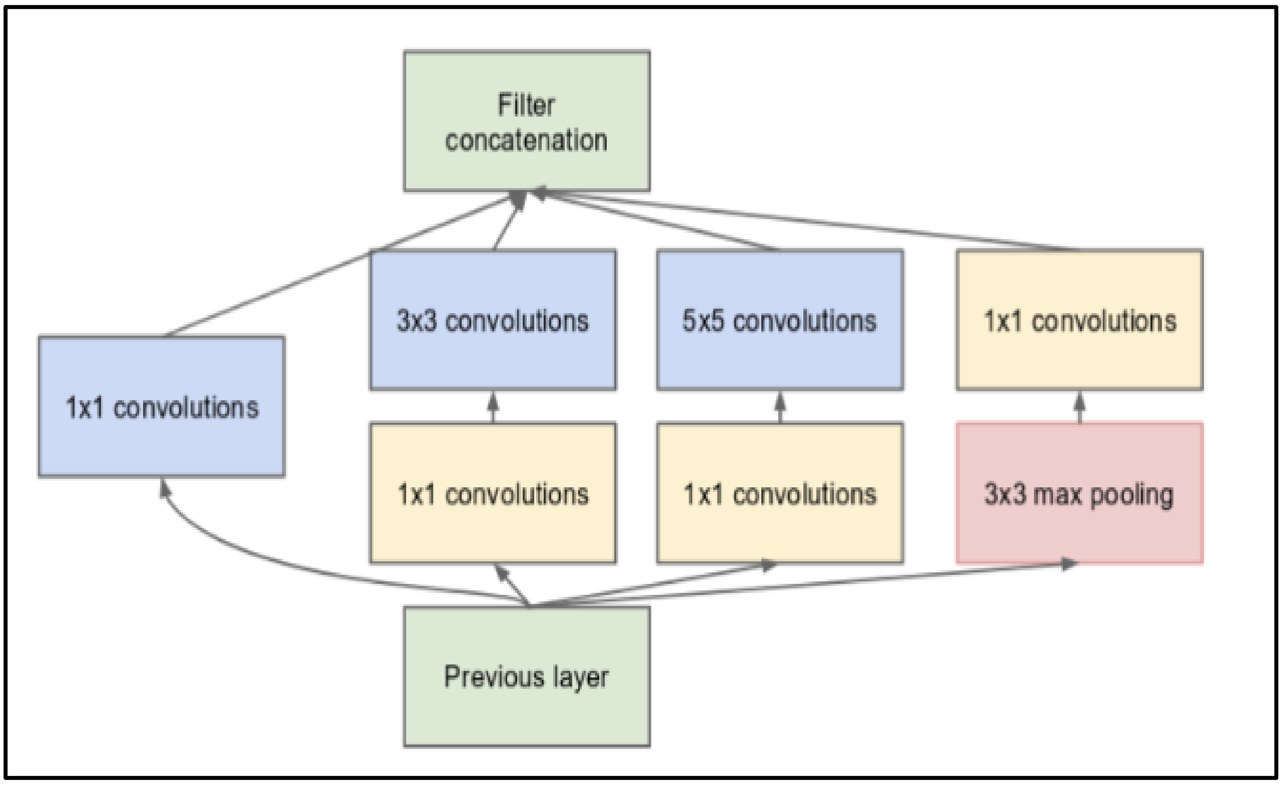
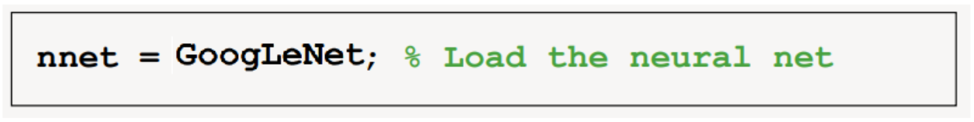


Fig 4. Building blocks of Google Architecture [*(Kölsch and Liwicki et al. 2017)*]

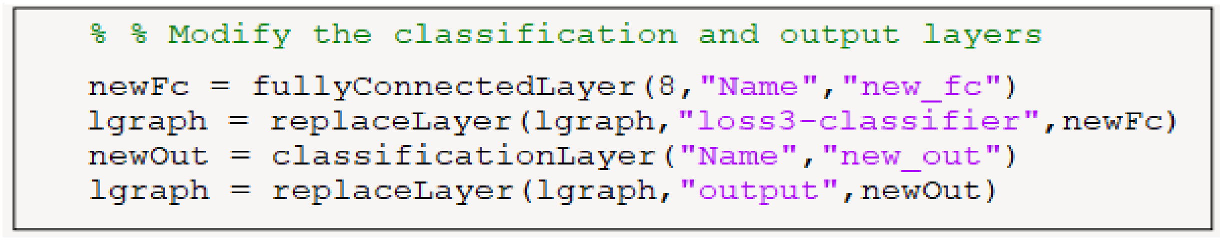
The GoogleNet Architecture has a total of 22 layers, including 27 pooling layers thereby comprising of 144 components in total. There are a total of 9 inception modules layered in a linear fashion. One of the techniques GoogleNet adopts to accomplish high efficiency is by reducing the input image data while maintaining key spatial details. Between the inception modules, there are 2 max-pooling layers. These max-pooling layers' job is to perform the down-sampling of the input as it travels across the network which is accomplished by compressing the dimensions of the input i.e., height and width of the images. After all the max-pooling stages, a global average pooling layer is connected at the end of the inception modules which averages all of the feature maps. A layer called the Dropout layer is placed just before the linear layer, responsible for regularization is utilized while training the network to prevent it from overfitting. The regularization technique reduces the number of interconnected neurons in a neural network at random. During each training iteration, there is a chance for every neuron of being excluded from the aggregated contribution from the connected neurons.

GoogleNet is a conv-net architecture trained on more than a million image-datasets widely used for image classification problems. In MATLAB, GoogleNet can be used as a pre-trained network for building a classification model. For the specific requirements of working on a new image-dataset, GoogleNet can be retrained through transfer learning. The most commonly used approach in Transfer learning is to pre-train the network on the ImageNet dataset. In MATLAB, pre-trained GoogleNet will be utilized for implementing the Deep Network.



```
nnet = GoogLeNet; % Load the neural net
```

The above command returns a pre-trained GoogleNet CNN that has been trained on the ImageNet dataset. For the execution of this command, a pre-requisite of Deep Learning Toolbox support package is required by this function.



```
% % Modify the classification and output layers
newFc = fullyConnectedLayer(8,"Name","new_fc")
lgraph = replaceLayer(lgraph,"loss3-classifier",newFc)
newOut = classificationLayer("Name","new_out")
lgraph = replaceLayer(lgraph,"output",newOut)
```

In this research, for the classification of road-traffic signs, the last layers of GoogleNet are replaced as per our requirement of classifying different categories of road signs i.e. a loss3-classifier is replaced with a fully-connected layer, and the output layer is replaced with a new classification output layer with the help of *replaceLayer* function.

Chapter 4

Implementation

Implementation is the most crucial part of this research project. In this chapter, we cover all the steps involved in the implementation of the Traffic Sign Recognition system by applying Deep Learning in MATLAB.

4.1 Setting up the environment

In the initial environmental setup, we install the latest version of MATLAB desktop software. As MATLAB is licensed software, a valid license is required for implementing any kind of project. In this project, the MATLAB R2020b version (for academic use) is installed on the local desktop for carrying out the actual implementation. An online version of MATLAB is also available for academic use which provides access to MATLAB and Simulink with no additional installation for the toolbox. MATLAB online also provides access to cloud storage for managing large files.

In MATLAB, for implementing Deep Learning there are some additional pre-requisites that need to be installed while installing the MATLAB software such as Parallel Computing Toolbox, Computer Vision Toolbox, GPU Coder, MATLAB Coder, Reinforcement Learning Toolbox, Deep Learning HDL Toolbox, Simulink (for using Deep Learning Toolbox block library). We can face some issues if we fail to install any of the above dependencies, hence it is very important to ensure all the dependencies are selected while installation. In Deep Learning as model training plays a vital role in the implementation, it is important to select the appropriate computational resources as per the requirement of the project. MATLAB offers 3 types of computation options: - CPU-based, GPU-based, Cloud-based. In our research, CPU-based computation is selected for training the neural network as there were some challenges faced while opting for the GPU based computation.

4.2 Data collection

For this Deep Learning project, the image datasets have been downloaded from multiple sources like Kaggle, GitHub, and GTSRB dataset. As the name suggests GTSRB (German Traffic Sign Recognition Benchmark), this dataset is considered as the benchmark dataset used for image classification and computer vision applications. The image format for all image data sets is in PNG format.

Different categories of traffic signs are readily available in a sub-folder format where each sub-folder represents a specific road sign and contains a set of captured traffic sign images. The GTSRB dataset contains around 43 categories of traffic signs containing approximately 50,000 image files. As CPU-based computation is used for this project, we have considered some of the basic categories of traffic signs – Speed Limit, No-entry, No-horn, Construction work, Roundabout, Stop sign, U-turn, Yield. Once data is collected from all sources, we proceed to the next stage where we create sub-folders for different categories for implementing the Transfer Learning approach for our traffic sign recognition system.

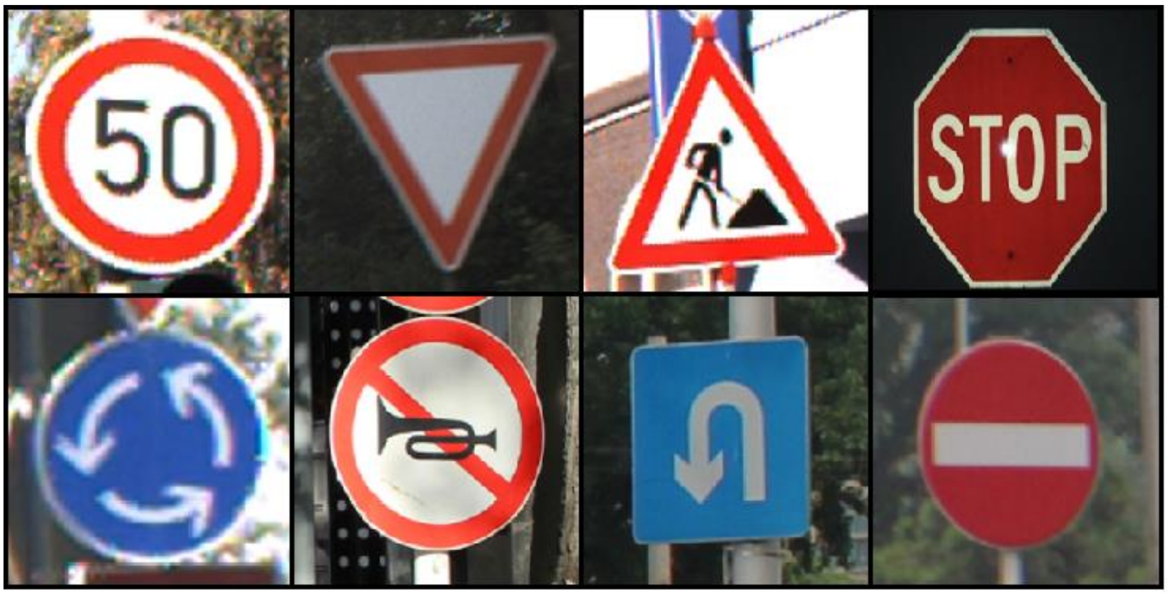


Fig 5. Categories of Traffic Signs

4.3 Image Categorization

In this image classification project, all the images are classified in an organized fashion by class into different folders. As shown in the figure 6, every category of traffic signs are segregated into different folders.

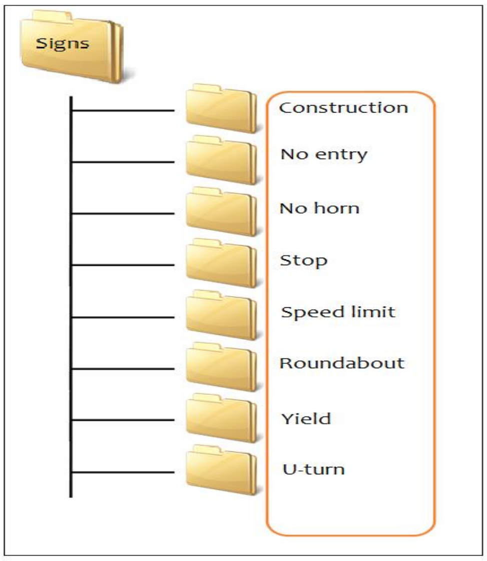
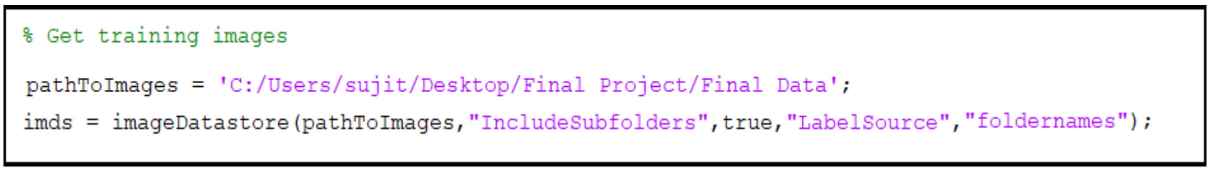


Fig 6. Labelling categories of Traffic-Signs

The segregation shown in Fig 6 along with labelling the categories of road signs is very crucial while training a network, as the model classifies new images based on the categories with the known labels as specified by the user. The labels are automatically determined by the image- datastore based on the folder names when the "LabelSource" option is specified. The labels are extracted directly from the folder names and saved as a categorical array. The Labels property must be manually modified if the image files are not stored in this manner.



```
% Get training images

pathToImages = 'C:/Users/sujit/Desktop/Final Project/Final Data';
imds = imageDatastore(pathToImages,"IncludeSubfolders",true,"LabelSource","foldernames");
```

In the above-shared code section, *imds* variable stores the Image-datastore where the first argument contains the path for the folder containing all images. *"IncludeSubfolders"* & *"LabelSource"* arguments are specified for the model to categorize the images based on their folder names.

4.4 Pre-processing image data

In MATLAB, we have Image-datastores which are a convenient way to manage, process, and work with large image datasets. For image classification using CNN, each image must be size-specified as per the input layer of the network. All images require basic pre-processing before they can be directly used for training and classification. Instead of pre-processing images individually, the most preferred and efficient way is to apply the pre-processing to the entire image-datastore.

The pre-trained GoogleNet takes the image data with an input layer having the dimensions [224 224 3]. Hence, for re-sizing and standardizing all images as per the dimensions of the network, a custom function is created, and the processed images are stored in an image datastore. There are also other alternatives for image-datastore such as *augmentedImageDatastore* which performs augmentation of images with a randomized combination of resizing, reflection, rotation, shear, and translation. Augmented Image-datastore works well while training the model as it performs image processing operations while they read the image data in mini-batches. Therefore, for each epoch, the network uses a slightly different image dataset.

4.5 Implementation of Deep Learning using Transfer Learning

As we are dealing with the image classification problem, a Deep learning approach is used which allows the model to recognize and extract key features from the image-dataset. The set of images used for this research contains eight different categories of traffic signs, and the Deep learning network will recognize the signs automatically with the help of categorized label names specified to folders. With the help of the train data, the network tries to learn the specific features of the image and correlate them with its corresponding category.

The feature detection layers present in the network perform three kinds of operations on the dataset: convolution, pooling, and ReLU function.

- Convolution processes the incoming images through a series of convolutional filters, where each filter activates different features of the images.
- Pooling reduces the number of constraints, the network needs to learn from, by carrying out non-linear down sampling.
- ReLU activation function maps negative values to zero and preserves only positive values, enabling the network to learn faster and efficiently by training on tens and hundreds of layers, where each layer learns to recognize distinct features.

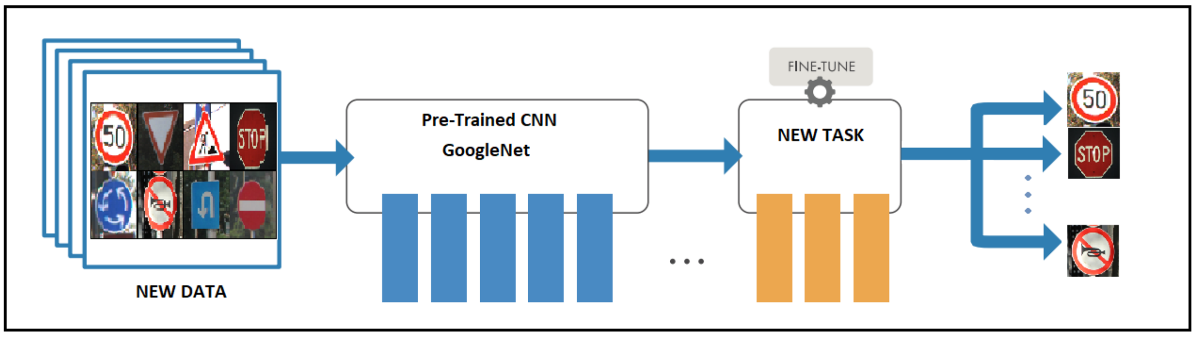


Fig 7. Transfer Learning Methodology

It is relatively simple to use a pre-trained network like GoogleNet rather than training the model from the scratch. However, there is no control over how the network runs and there is also a possibility that the network is unlikely to solve the problem that needs to be resolved. Anyone can build and train a network with simple network design with random weights. However, getting suitable results takes a lot of work in terms of network architecture expertise, large training datasets, and finally high computational power.

Transfer learning is an effective solution to a variety of issues. It takes little data and less computation time as compared to training the model from the scratch, and the outcome of the model can be tailored to the specific use-case.

4.6 Generate train and test data

The concept of train-test validation influences how data should be processed for training the network and deployment of the computer-vision model. During training, the network tries to correlate the given inputs with the given outputs. Therefore, the network can easily classify the training data. However, for evaluating the network, the model needs to be tested on a new test dataset. Hence, we need to split the dataset into training and test data. In most of the cases, the ratio selected for the train-test split is 70:30, but in cases where we include the validation set in the model, the proportion would change to 70:15:15.

```
% % Split into training and testing sets
[trainImgs,testImgs] = splitEachLabel(imds,0.8);

[trainImag,testImag,validation] = splitEachLabel(imds,0.7,0.15);
```

For splitting the images into the Train-Test dataset, MATLAB provides a function called as *splitEachLabel*, where the function divides all the images stored in an imageDatastore into two separate imageDatastores. As shown in the above code editor, in the first case only two arguments are provided, image-datastore and the proportion value (0 to 1) which determines the proportion of images from each category that should be included in the training dataset. The remaining proportion of images are stored in the test dataset. In the second case, as the validation set is included, three arguments are fed to the function where the additional argument is assigned for the proportion for the validation set. Instead of specifying proportion, MATLAB also provides an option to take an exact count of the files to be taken from each category, which ensures every label in the training datastore has *n* images, even if each category does not have the same number of images.

4.7 Loading & Training the pre-trained network

The pre-trained network used for our image-classification problem has already been trained to extract powerful and key features from natural images. These trained networks have various characteristics which need to be considered while applying them directly to a specific problem. The most crucial aspects that should be considered while selecting a model are network accuracy, computational speed, size of the network. MATLAB Deep Network Designer app provides a platform to create, import, modify networks as well as load pre-trained neural networks. In our research, we have loaded GoogleNet for classifying the traffic signs.

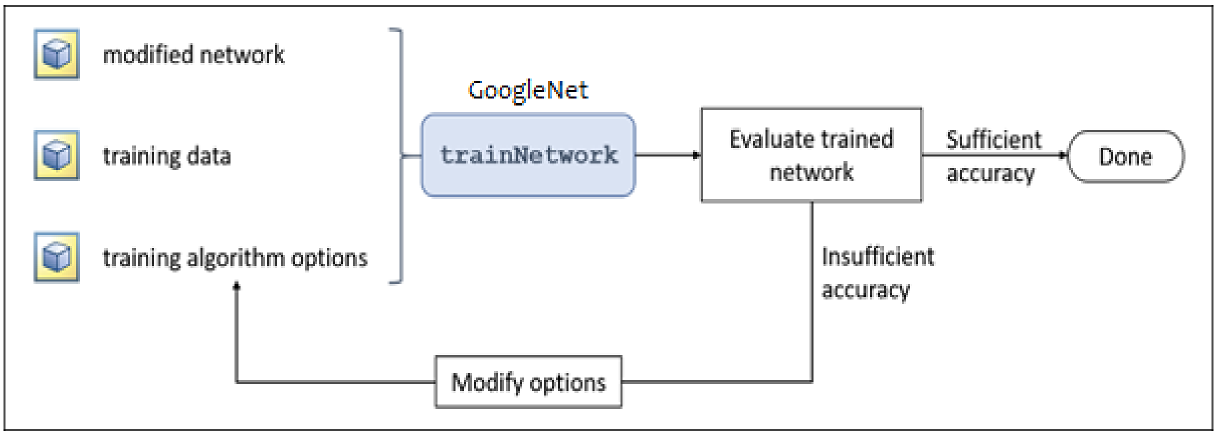


Fig 8. Training the Pre-trained network (Referred from [MATLAB for Engineers](#))

As shown in the Fig 8 architecture, training a pre-trained network requires three important components:

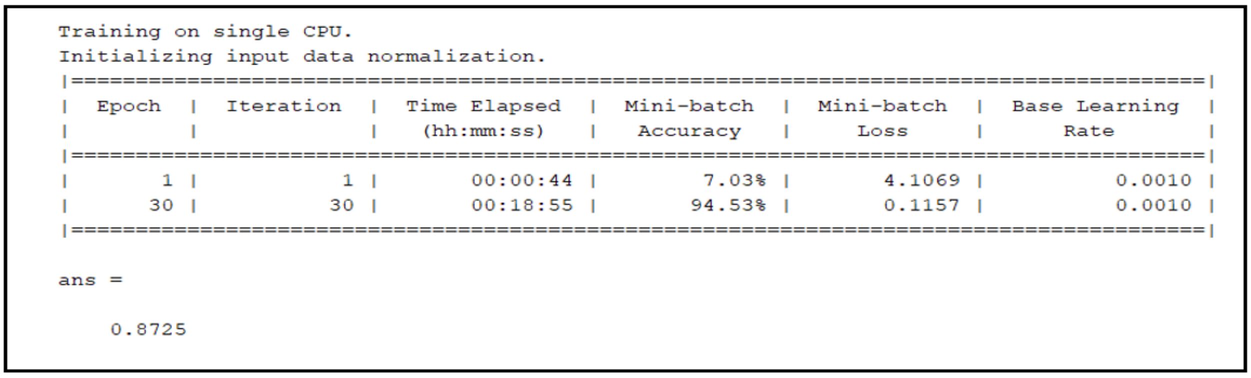
- An array of layers that represents the neural net architecture. It is created by modifying an existing network (GoogleNet).
- Training dataset. In our case, images with known labels will be used as image-datastore.
- A variable that stores the options which are responsible for controlling the behaviour of the training algorithm.

The above three components are passed as inputs to the *trainNetwork* function, and a trained network is obtained as the output. If the performance of the trained network is not satisfactory, the available tuning options are tweaked, and network is re-trained, and the process is repeated until a good accuracy is achieved by the network. MATLAB provides a wide range of options while training the network such as *optimizer* required, *Initial learning rate, MaxEpochs, MiniBatchSize, ValidationFrequency,* and much more. These training options have default values that are tweaked while performing model training to achieve a suitable accuracy.

In our research, we have selected SGDM optimizer, and the Initial learning rate is set to a lesser value (0.001) than the default value (0.01). SGDM is used for model training as it helps models to converge faster when large datasets. A subset of training data, known as mini-batches, is used for updating the network weights at each iteration, and each iteration employs a distinct mini-batch. When mini batches along with SGDM are implemented, the parameters frequently get updated for faster computations. An epoch occurs when the entire training dataset has been used. The learning rate determines how frequently the algorithm modifies the network weights of the model. As the main goal of the transfer learning approach is to fine-tune a pre-trained neural net, the weights should be modified less frequently than when the model is trained from scratch.

The *trainNetwork* function automatically detects for GPU, if the prerequisites of Parallel Computing Toolbox and other toolboxes are installed. In our project, MATLAB failed to detect specific GPU on the local machine. Hence, the model is trained on a single CPU which is the default option when it comes to training a network. Although training the network on a single CPU machine is time-consuming, we have trained the network at different epoch values. The model is trained at different epoch values for attaining higher accuracy.

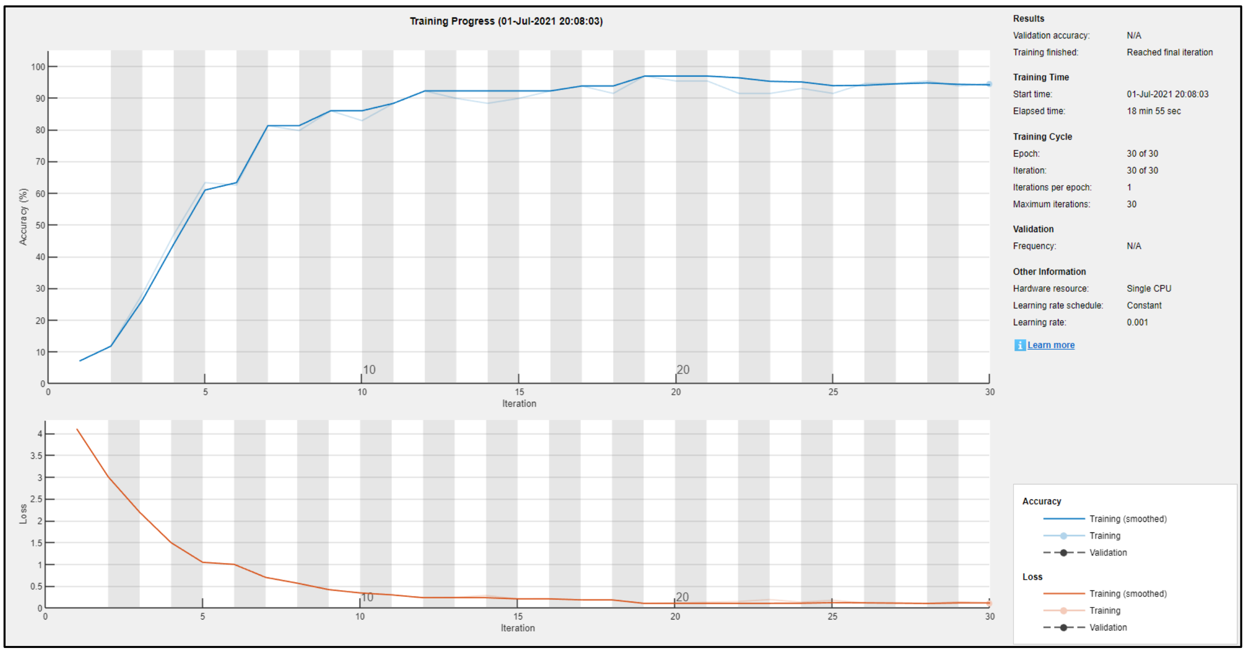
4.8 Model Training Simulation



```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:44 |        7.03% |       4.1069 |          0.0010 |
|      30 |          30 |       00:18:55 |       94.53% |       0.1157 |          0.0010 |
|=========================================================================================|

ans =

    0.8725
```

Fig 9. Network Training on Single CPU

The network training progress appears on the MATLAB command window as shown in figure 9, which contains the classification accuracy for each mini-batch as well the overall accuracy of the trained network.



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 01-Jul-2021 20:08:03
Elapsed time: 18 min 55 sec

Training Cycle
Epoch: 30 of 30
Iteration: 30 of 30
Iterations per epoch: 1
Maximum iterations: 30

Validation
Frequency: N/A

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

Fig 10. Training Progress

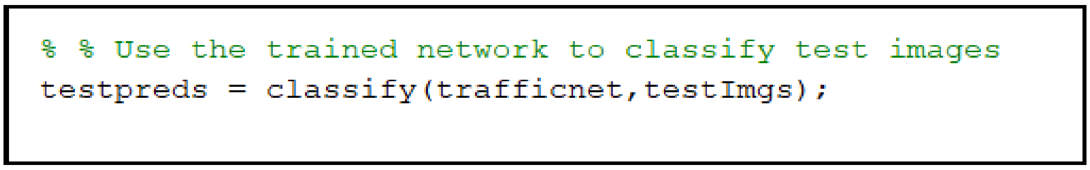
MATLAB also provides a simulation window for monitoring the Deep Learning training progress of the neural network. A real-time training simulation as shown in the fig-10, containing different training metrics is displayed which helps to understand how the training is progressing at every iteration. Each iteration involves estimating the gradient and updating the network parameters. If the validation set is specified in the training options, validation metrics are displayed each occurrence when the *trainNetwork* function validates the network.

The training progress window plots some of the training metrics as shown below:

- Training accuracy – Accuracy obtained for each mini-batches.
- Training accuracy with smoothing – Accuracy achieved by applying a smoothing algorithm. Trends are observed with lesser noise with smoothed accuracy.
- Validation accuracy – Accuracy obtained for entire validation dataset.
- Training loss, smoothed training loss, and validation loss – The loss for ever mini-batch, loss for smoothed version, loss on the validated dataset.

It can be observed that as the number of iterations increases with time, the accuracy of the model increases reaching more than 90% after 15-20 iterations. Along with accuracy, the overall loss decreases over time and reaches a minimum once the training is complete.

The network would definitely perform well on the training dataset, but the real problem is how well the network performs on the test dataset. For this purpose, *classify* function is used for obtaining the classifications, the *trainedNetwork* predicts for images in test image-datastore. The classify function will run through all images in test images-datastore and performs classification of test images as per labelled categories.



```
% % Use the trained network to classify test images
testpreds = classify(trafficnet,testImgs);
```

4.9 Model Output

The trained network model implemented using the transfer learning approach by applying GoogleNet model can predict the traffic sign images and categorize them according to the labelled folders as stored inside the image-datastore. As the network predictions for test images have been determined with the help of labelled test images, the predictions can be compared with true classes to evaluate how the network is performing. With the help of the confusion matrix and overall accuracy obtained for the model, the performance of the network can be assessed. The training progress visualization window helps us to understand the trend and impact of testing the model with different tuning options such as how quickly the model learns the main features, how the network accuracy is impacted, determine whether the network is overfitting the train data.

Chapter 5

Analysis and Evaluation

The results and evaluations of the research work are assessed using confusion matrix and analysing the impact of changing the options on its performance. A comparative study on the network with validation set and without validation set is also performed for evaluating the obtained results. The number of misclassified images obtained while testing the network with new test images can be obtained using a confusion-matrix chart.

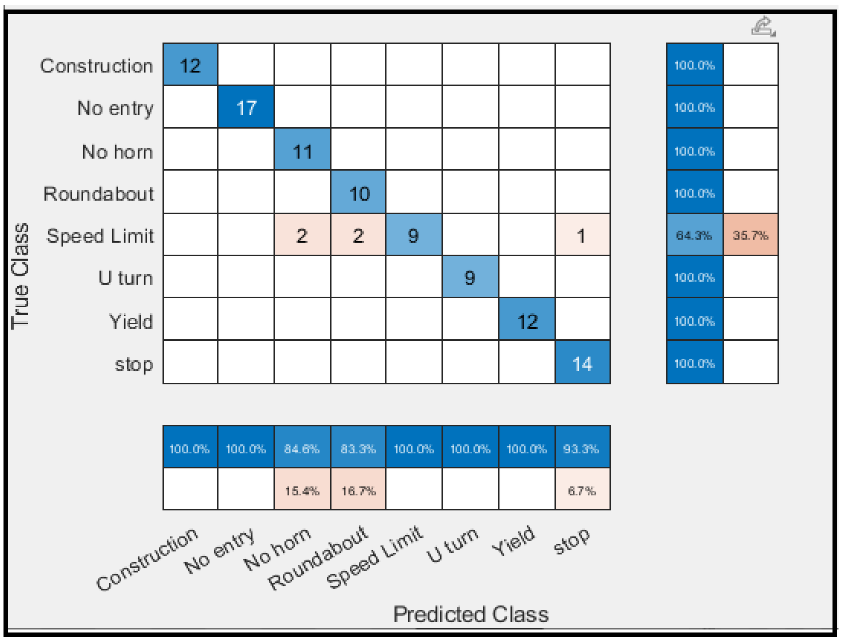


Fig 11. Confusion-Matrix chart

```
%
% % Visualize the confusion matrix
confusionchart(trueLabels,predictedLabels,'RowSummary','row-normalized','ColumnSummary','column-normalized');
```

The *confusionchart* function calculates and creates a confusion matrix plot for the predicted classifications. The rows of the confusion matrix relate to the true class, while its columns to the predicted class. The correct classified observations are represented by diagonal cells, and the incorrect ones are represented by off-diagonal elements.

The confusion chart shown in figure 11 illustrates that there are five test images misclassified in the speed limit traffic sign category. The misclassified images are mostly the complex traffic-signs which contain more than one traffic-signs. It is quite difficult to deal with images that contain multiple road-signs, as there can be multiple permutations and combinations of traffic signs captured in a single traffic scene. The figure 11 confusion chart has been tested without a validation set which indirectly hampers the network accuracy.

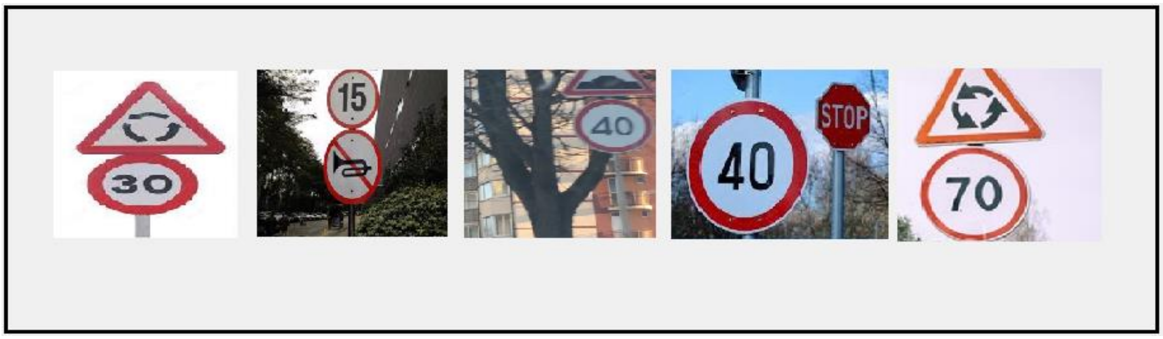


Fig 12. Mis-classified images

The inclusion of a validation set in the network training improves the overall accuracy as the validation set is responsible for providing unbiased and frequent evaluation i.e., the validation data set is used for fine-tuning the hyper-parameters of the network model. The main reason to include validation set in model training is to avoid model over-fitting, providing some helpful insights in terms of optimizing the hyperparameters and how the optimization of constraints influences the trained network.

MODEL ACCURACY

Without Validation Set

Epochs	**Accuracy**	**Misclassification Rate**
30	87.25 %	5.05 %
40	94.95 %	5.05 %

With Validation Set

Epochs	**Accuracy**	**Misclassification Rate**
20	96.08 %	2.02 %
30	98.04 %	1.01 %
40	98.04 %	1.01 %

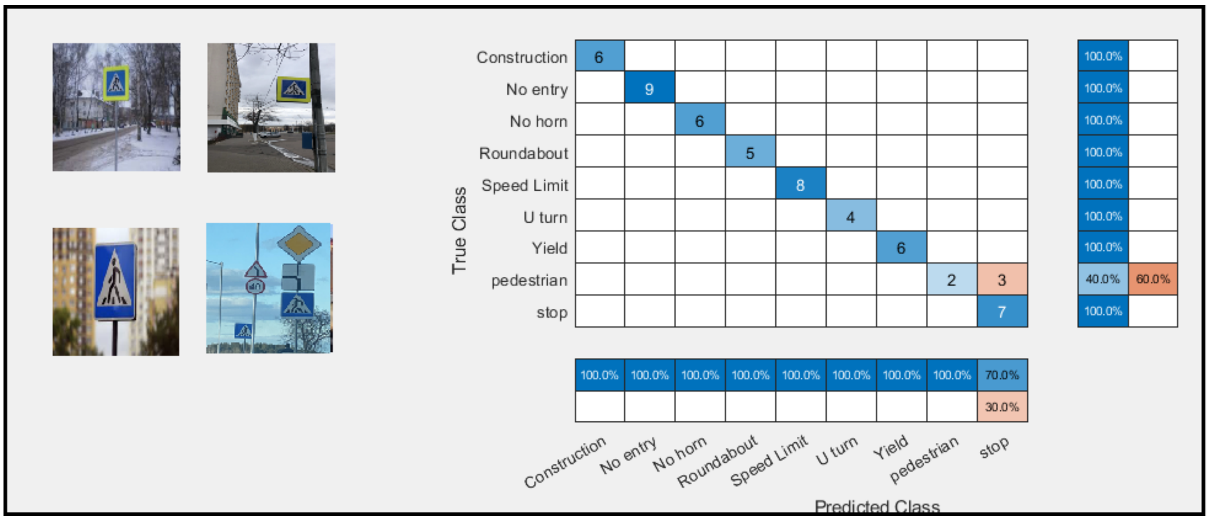
TABLE III – Comparative study of the model with and without Validation set

From the Table-III, we have evidently found that with the help of the validation set, the overall accuracy of the network is improved when training the model with different epochs. The network took single iterations per epochs for training the model on all the different constraints the network was trained upon. *(see Appendix for all the calculated results)*

The model has been trained on a smaller number of epochs while keeping the learning rate to a constant value (0.001) and other tuning constraints as default values, due to the fact that increasing the number of epochs would result in model over-fitting i.e., the validation metrics start to a show stagnating trend after a certain number of epochs and makes the model irrelevant to other datasets. The image mis-classification rate has been calculated from the confusion matrix for each training scenario. *(see Appendix for confusion matrix charts)*

Other Challenges

There were some other challenging tasks faced during the course of our research work. For instance, we have not included the pedestrian data in this project as the misclassified images were on the higher side than the classified ones. These misclassifications are due to the fact that the images captured were not at proper angles or low camera resolution, traffic signs detected at the border of the image, multiple signs detected in a single image (Fig 13). Due to some of these issues, the network fails to extract the key features from the image and result in misclassification of traffic signs.



True Class	Construction	No entry	No horn	Roundabout	Speed Limit	U turn	Yield	pedestrian	stop		
Construction	6									100.0%	
No entry		9								100.0%	
No horn			6							100.0%	
Roundabout				5						100.0%	
Speed Limit					8					100.0%	
U turn						4				100.0%	
Yield							6			100.0%	
pedestrian								2	3	40.0%	60.0%
stop									7	100.0%	
	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	70.0%		
									30.0%		

Fig 13. Pedestrian Dataset challenges

Other than images, videos can also serve as an input to the network. Some of the difficulties faced while sign recognition is poor video quality (due to the low quality of image sensor), image output format, compression rate, blurring effect caused due to motion of the vehicle, and much more. The image captured always have different resolutions and every pre-trained network have their specific input dimensions to store the image/video data inside the input layer. After the pre-processing stage, in most cases, it happens that the image loses its quality and ultimately results in misclassification.

One of the main challenges faced in this project work is not having enough computational power. Although MATLAB was installed with its required pre-requisites and additional toolboxes for Deep Learning, MATLAB failed to detect GPU in the local machine. For implementing GPU within MATLAB, a recent graphics driver is required. MATLAB supports NVIDIA Drivers for GPU computation. MATLAB online currently has no specific full-fledged cloud computing environment for executing projects as it is still in the development phase. Due to the lack of computational power, the project has been trained on a single CPU which is quite a challenging task.

Chapter 6 - Conclusion and Future Work

6.1 Summary of Thesis

This thesis presented a Deep Learning solution for developing a Traffic Sign Recognition system using Transfer Learning Approach. Different steps have been carried out for implementing a Deep Neural Network using MATLAB, that helps in recognizing and classifying some of the basic road signs to the user.

The documentation has been divided into 5 main chapters. The first chapter covers a brief discussion on Automated Driving systems and some of the difficulties and challenges faced by automated driver assisted cars. It gives a brief idea about building an Image Recognition system using the Transfer Learning approach and the objectives to achieve this goal have also been outlined in this chapter.

In chapter 2, we have explained some of the previous related research works and various published academic IEEE papers that have helped indirectly in achieving our project objectives. The research works show different Image-detection algorithms, machine learning models, Deep Learning networks and computer vision have been used by researchers for building robust traffic sign detection and recognition systems. It also covers some of the advantages of using MATLAB over other languages for its implementation.

In chapter 3, we discussed the Deep Learning Network architecture used for the image classification problems. Implementation of a pre-trained network in MATLAB, the architecture of the pre-trained network, the layers involved in the network, the functionality of each layer, the conceptual design working of the model are some of the other aspects covered in this chapter.

Chapter 4 serves as one of the most important chapters in this thesis. This chapter covers our main objective of implementing of the Deep Neural Network for traffic sign recognition and it covers every step involved in the process. From setting up the environment to the final classified output, each step involved in the workflow were explained with its importance supported by suitable diagrams.

Chapter 5 is another crucial chapter that covers the analysis and evaluation of the network model in detail. With the help of a confusion matrix and calculated results of overall accuracy, the performance of the trained network has been assessed. Some of the factors that hamper the overall network accuracy are also discussed in this chapter. Comparisons for training the network with and without validation sets were analysed and explained. The other challenges faced during this course of this thesis project were also covered in this chapter.

6.2 Future Works

As a Deep Learning project, this project has a huge scope for future works that could be carried out. Some of the works that could be taken upon are discussed below:

- Develop a fully automated system for traffic-sign detection and recognition by integrating it GPS and other navigation systems.
- Training with different pre-trained networks available in MATLAB and performing a comparative study for determining which model performs better in terms of execution time, accuracy as well as computational power required.
- Exploring the network with different hyperparameter optimizations for improving the efficiency of the network.
- Image recognition stage can be processed at a faster pace by reducing the dimensions of the feature vectors and providing additional computational power with higher GPU configuration.
- Implementation of the network by training with real-time videos instead of image datasets.

References

[1] Ai, C., 2013. A sensing methodology for an intelligent traffic sign inventory and condition assessment using GPS/GIS, computer vision and mobile LiDAR technologies (Doctoral dissertation, Georgia Institute of Technology).

[2] Ramler, R. and Ziebermayr, T., 2017, March. What You See Is What You Test-Augmenting Software Testing with Computer Vision. In 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 398-400). IEEE.

[3] Yasmina, D., Karima, R. and Ouahiba, A., 2018, November. Traffic signs recognition with deep learning. In 2018 International Conference on Applied Smart Systems (ICASS) (pp. 1-5). IEEE.

[4] Aly, S., Deguchi, D. and Murase, H., 2013, October. Blur-invariant traffic sign recognition using compact local phase quantization. In 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013) (pp. 821-827). IEEE.

[5] Zeng, Y., Xu, X., Fang, Y. and Zhao, K., 2015, June. Traffic sign recognition using deep convolutional networks and extreme learning machine. In International Conference on Intelligent Science and Big Data Engineering (pp. 272-280). Springer, Cham.

[6] Changzhen, X., Cong, W., Weixin, M. and Yanmei, S., 2016, August. A traffic sign detection algorithm based on deep convolutional neural network. In 2016 IEEE International Conference on Signal and Image Processing (ICSIP) (pp. 676-679). IEEE.

[7] Mahatme, M.B. and Kuwelkar, M.S., 2017, July. Detection and Recognition of Traffic Signs based on RGB to RED Conversion. In 2017 International Conference on Computing Methodologies and Communication (ICCMC) (pp. 447-451). IEEE.

[8] Lin, C., Li, L., Luo, W., Wang, K.C. and Guo, J., 2019. Transfer learning based traffic sign recognition using inception-v3 model. Periodica Polytechnica Transportation Engineering, 47(3), pp.242-250.

[9] Peng, X., Li, Y., Wei, X., Luo, J. and Murphey, Y.L., 2017, November. Traffic sign recognition with transfer learning. In 2017 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1-7). IEEE.

[10] Wei, L., Runge, L. and Xiaolei, L., 2018, June. Traffic sign detection and recognition via transfer learning. In 2018 Chinese Control And Decision Conference (CCDC) (pp. 5884-5887). IEEE.

[11] Tan, M. and Le, Q., 2019, May. Efficientnet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.

[12] Paperspace Blog. (2020). A Guide to AlexNet, VGG16, and GoogleNet. [online] Available at: https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/.

[13] Alake, R. (2020). Deep Learning: GoogLeNet Explained. [online] Medium. Available at: https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765.

[14] Fan, Y. and Zhang, W., 2015, August. Traffic sign detection and classification for Advanced Driver Assistant Systems. In 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) (pp. 1335-1339). IEEE.

[15] Toth, Š., 2012. Difficulties of traffic sign recognition. 7th Winter School of Mathematics Applied to ICT, MICT.

[16] Introducing Deep Learning with MATLAB. (n.d.). [online]. Available at: https://www.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf

[17] www.mathworks.com. (n.d.). Options for training deep learning neural network - MATLAB trainingOptions. [online] Available at: https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html.

[18] Kölsch, A., Afzal, M.Z. and Liwicki, M., 2017, November. Multilevel context representation for improving object recognition. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) (Vol. 5, pp. 10-15). IEEE.

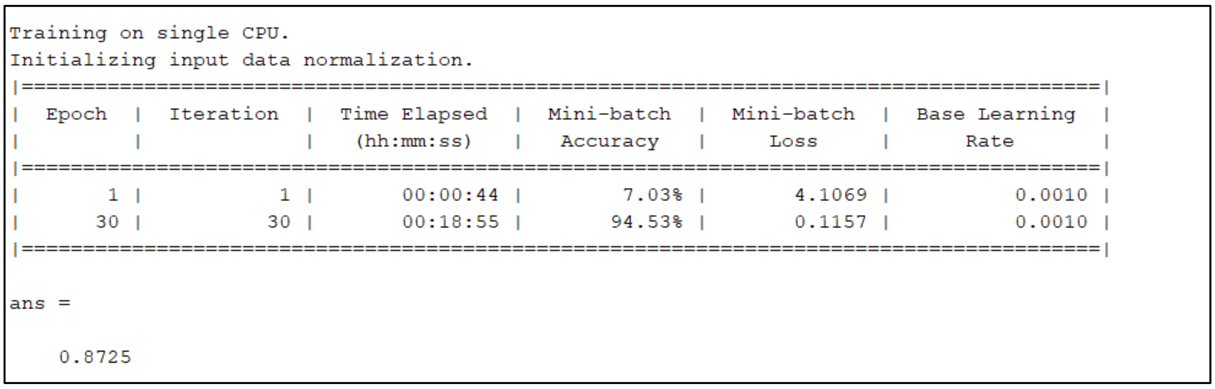
[19] MATLAB For Engineers. (n.d.). Components Needed for Transfer Learning. [online] Available at: https://matlab4engineers.com/lesson/components-needed-transfer-learning/?lang=en&v=d2cb7bbc0d23 [Accessed 4 Aug. 2021].

APPENDIX

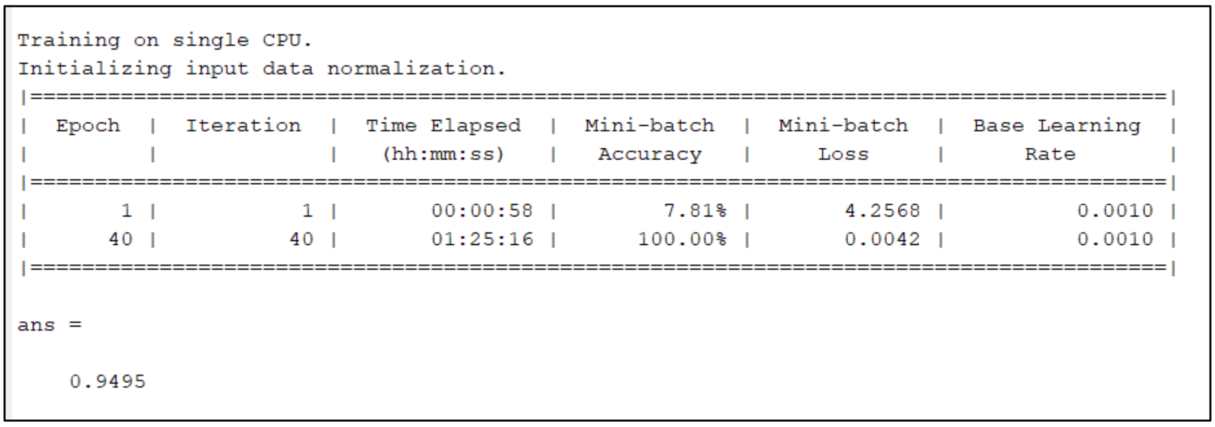
- The MATLAB files are submitted as supplementary files, and they are kept in MATLAB Scripts folder. It contains two MATLAB files; one contains the main code for execution and the second one contains image resizing function called within the MATLAB code for processing images. There is another folder named Final Data containing different categories of traffic sign images.

Calculated Results

- Network Tested without validation set
- EPOCHS- (30, 40), Learning Rate-0.001, SGDM optimizer



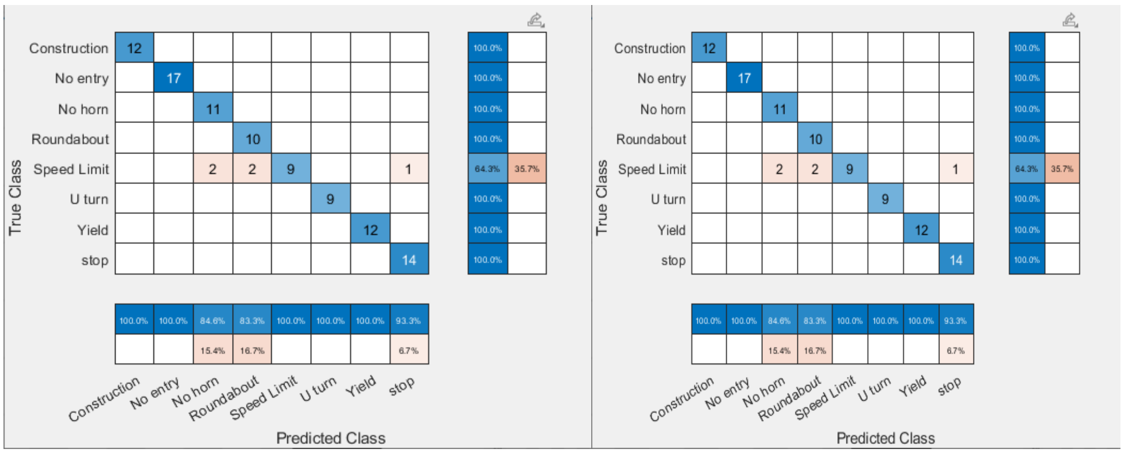
```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:44 |        7.03% |       4.1069 |          0.0010 |
|      30 |          30 |       00:18:55 |       94.53% |       0.1157 |          0.0010 |
|=========================================================================================|

ans =

    0.8725
```



```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:58 |        7.81% |       4.2568 |          0.0010 |
|      40 |          40 |       01:25:16 |      100.00% |       0.0042 |          0.0010 |
|=========================================================================================|

ans =

    0.9495
```

Even though the overall accuracy obtained for network trained with 40 epochs is more the network trained with 30 Epochs, the misclassified images obtained in both the scenarios are the same.

Confusion Matrix - Model trained without validation set



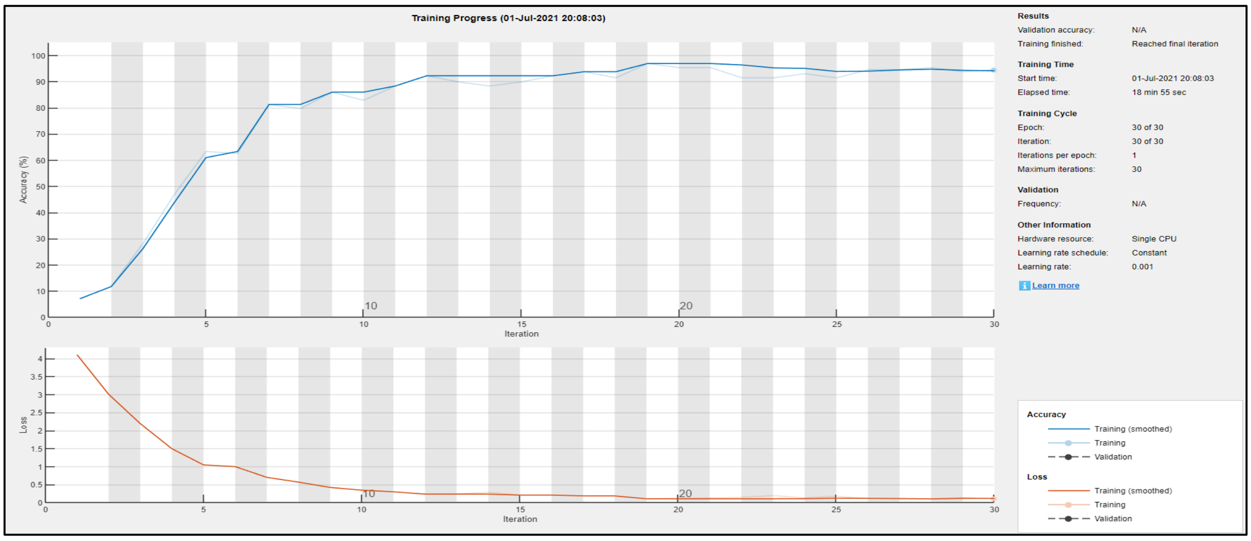
True Class	Construction	No entry	No horn	Roundabout	Speed Limit	U turn	Yield	stop		
Construction	12								100.0%	
No entry		17							100.0%	
No horn			11						100.0%	
Roundabout				10					100.0%	
Speed Limit			2	2	9			1	64.3%	35.7%
U turn						9			100.0%	
Yield							12		100.0%	
stop								14	100.0%	
	100.0%	100.0%	84.6%	83.3%	100.0%	100.0%	100.0%	93.3%		
			15.4%	16.7%				6.7%		

Predicted Class

True Class	Construction	No entry	No horn	Roundabout	Speed Limit	U turn	Yield	stop		
Construction	12								100.0%	
No entry		17							100.0%	
No horn			11						100.0%	
Roundabout				10					100.0%	
Speed Limit			2	2	9			1	64.3%	35.7%
U turn						9			100.0%	
Yield							12		100.0%	
stop								14	100.0%	
	100.0%	100.0%	84.6%	83.3%	100.0%	100.0%	100.0%	93.3%		
			15.4%	16.7%				6.7%		

Predicted Class

Training Progress without Validation set



Results

Validation accuracy:	N/A
Training finished:	Reached final iteration

Training Time

Start time:	01-Jul-2021 20:08:03
Elapsed time:	18 min 55 sec

Training Cycle

Epoch:	30 of 30
Iteration:	30 of 30
Iterations per epoch:	1
Maximum iterations:	30

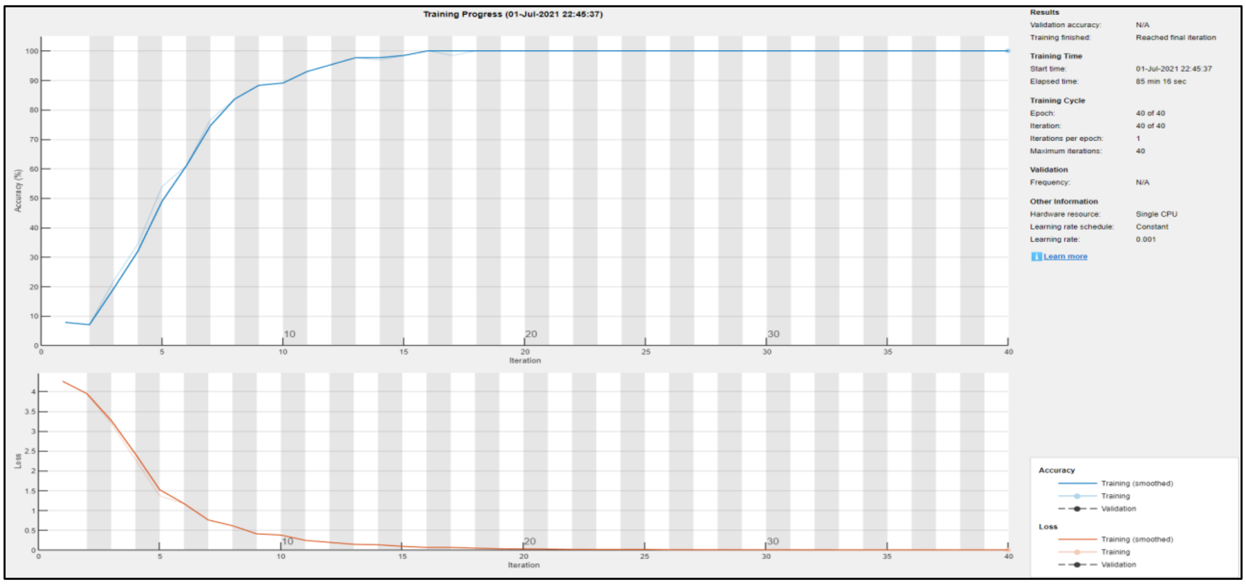
Validation

Frequency:	N/A

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more



Results

Validation accuracy:	N/A
Training finished:	Reached final iteration

Training Time

Start time:	01-Jul-2021 22:45:37
Elapsed time:	85 min 16 sec

Training Cycle

Epoch:	40 of 40
Iteration:	40 of 40
Iterations per epoch:	1
Maximum iterations:	40

Validation

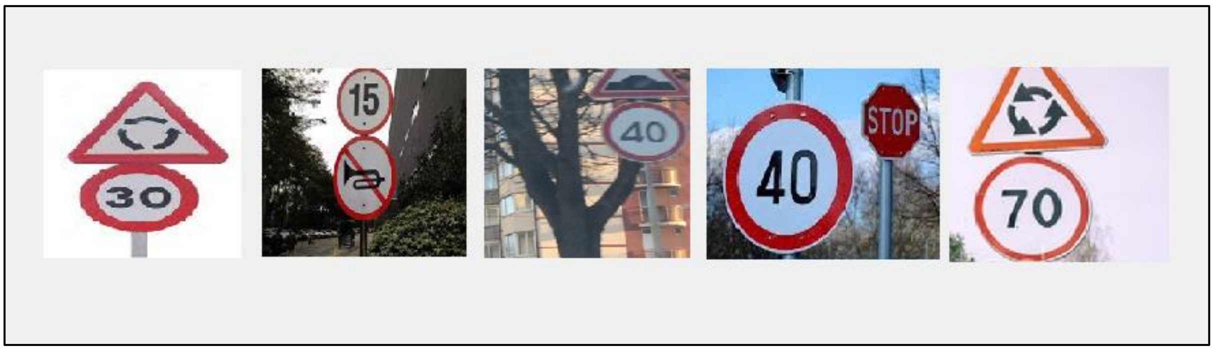
Frequency:	N/A

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

Learn more

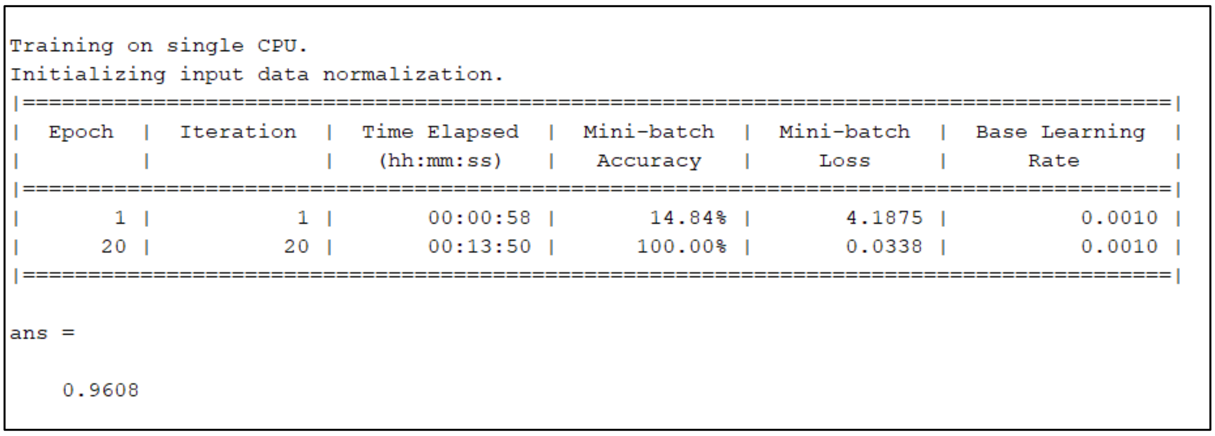
Mis-classified Images (Without Validation set)



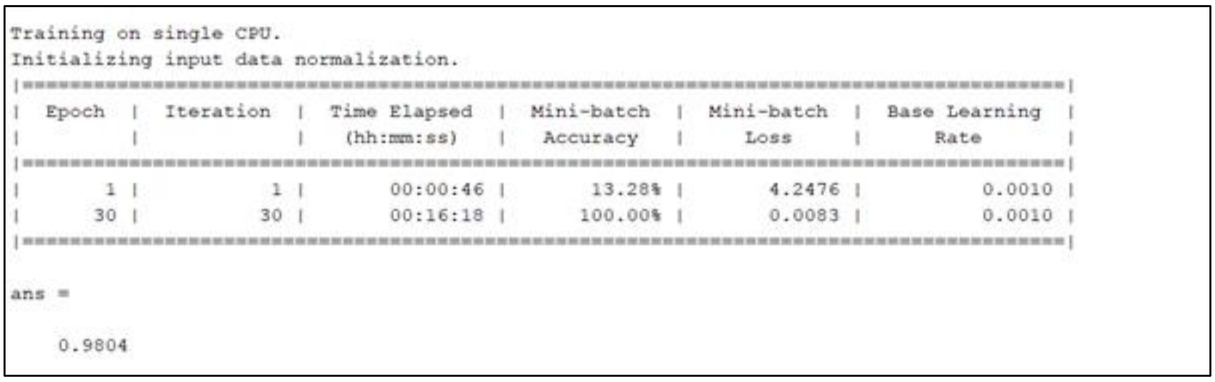
- Network Tested with Validation set

  The network trained with Validation set exhibits accuracy more the previous trained model without validation set

Model tested with Epochs (20, 30, 40)



```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:58 |       14.84% |       4.1875 |          0.0010 |
|      20 |          20 |       00:13:50 |      100.00% |       0.0338 |          0.0010 |
|=========================================================================================|

ans =

    0.9608
```



```
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:46 |       13.28% |       4.2476 |          0.0010 |
|      30 |          30 |       00:16:18 |      100.00% |       0.0083 |          0.0010 |
|=========================================================================================|

ans =

    0.9804
```

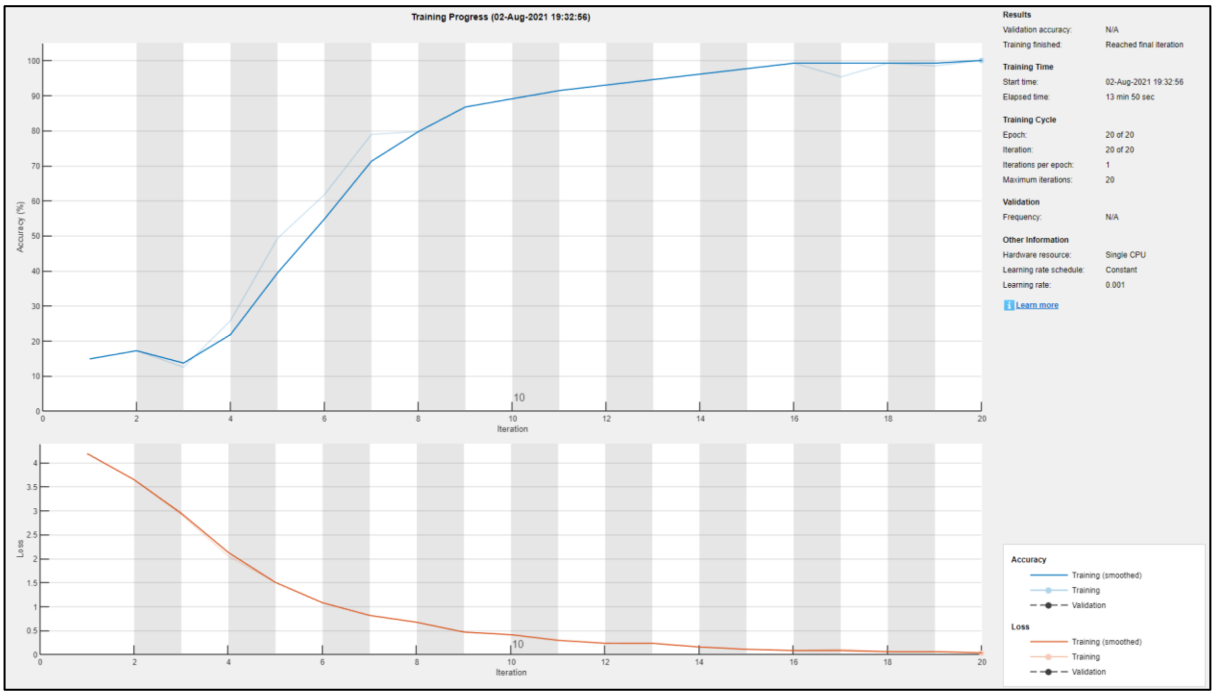
```
Training on single CPU.
Initializing input data normalization.
|==========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|==========================================================================================|
|       1 |           1 |       00:00:53 |       10.94% |       4.0581 |          0.0010 |
|      40 |          40 |       00:28:20 |      100.00% |       0.0091 |          0.0010 |
|==========================================================================================|

ans =

    0.9804
```

It can be observed that the training time for model at higher number of epochs is more as compared to the lower ones. Even if the number of epochs is increased further, the accuracy of the network remains the same, which suggests to stop increasing the epochs as further optimizing would result in model over-fitting.

Training Progress with Validation set

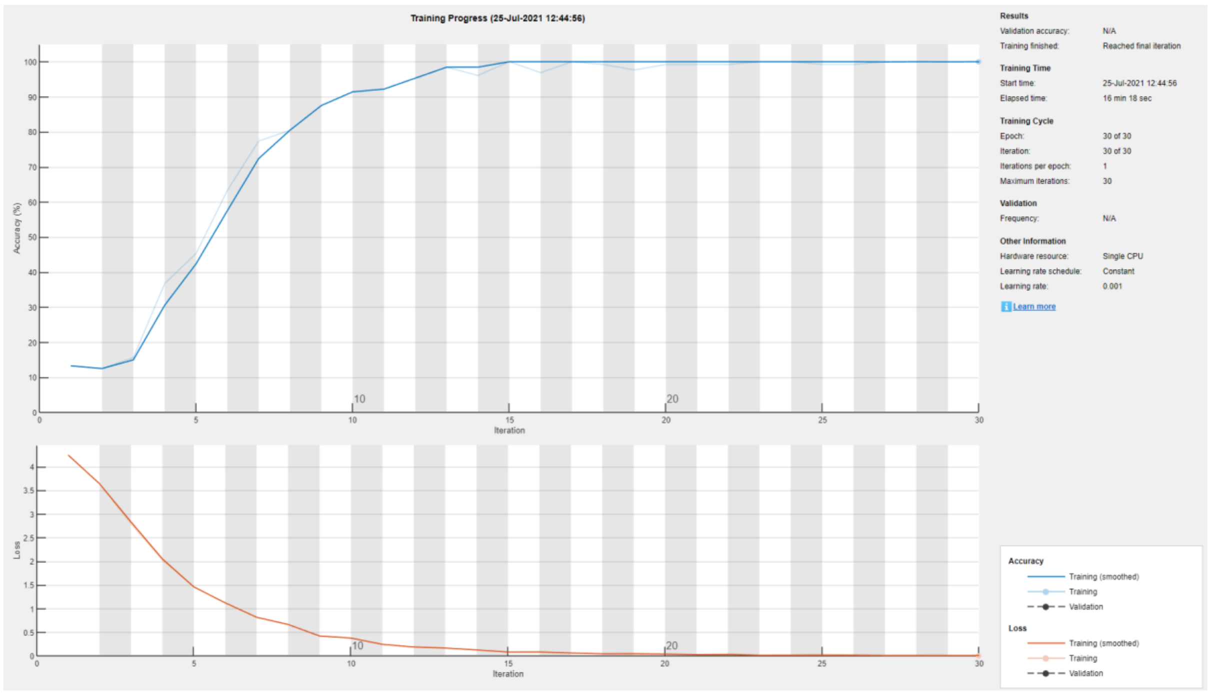
The training time is significantly reduced when the network is trained with the validation set.

- EPOCH- 20, Learning Rate-0.001, SGDM optimizer



Results	
Validation accuracy:	N/A
Training finished:	Reached final iteration
Training Time	
Start time:	02-Aug-2021 19:32:56
Elapsed time:	13 min 50 sec
Training Cycle	
Epoch:	20 of 20
Iteration:	20 of 20
Iterations per epoch:	1
Maximum iterations:	20
Validation	
Frequency:	N/A
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

- EPOCH- 30, Learning Rate-0.001, SGDM optimizer



Results
Validation accuracy: N/A
Training finished: Reached final iteration

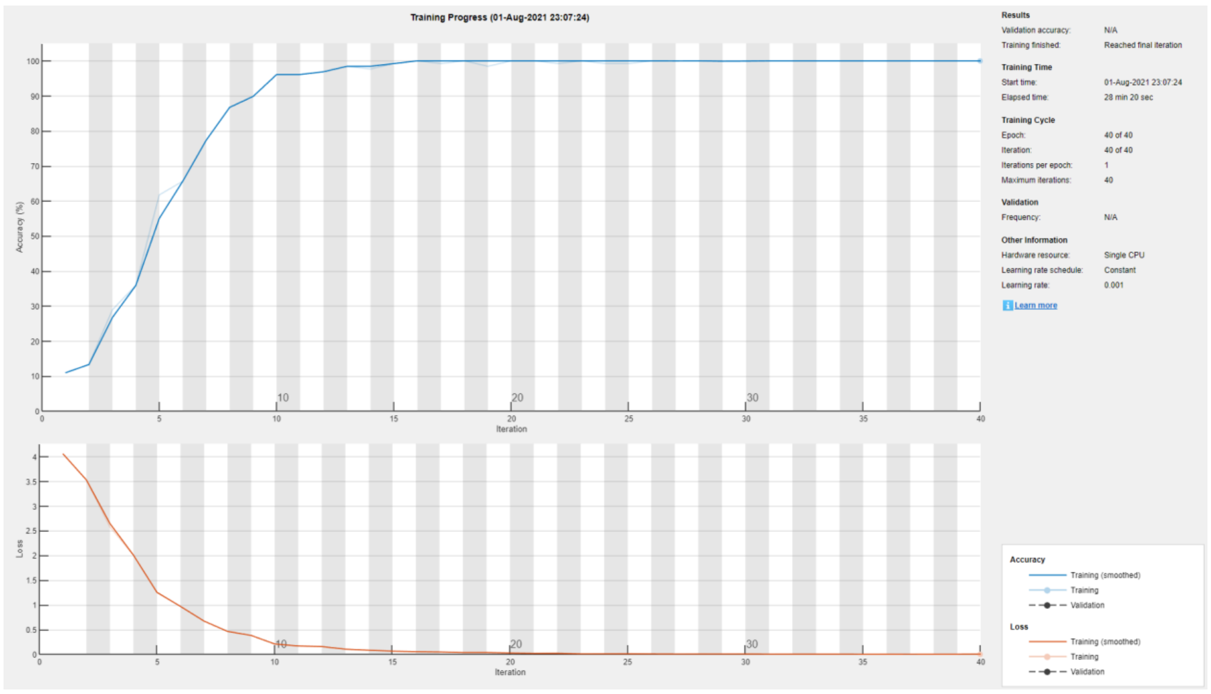
Training Time
Start time: 25-Jul-2021 12:44:56
Elapsed time: 16 min 18 sec

Training Cycle
Epoch: 30 of 30
Iteration: 30 of 30
Iterations per epoch: 1
Maximum iterations: 30

Validation
Frequency: N/A

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

- EPOCH- 40, Learning Rate-0.001, SGDM optimizer



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 01-Aug-2021 23:07:24
Elapsed time: 28 min 20 sec

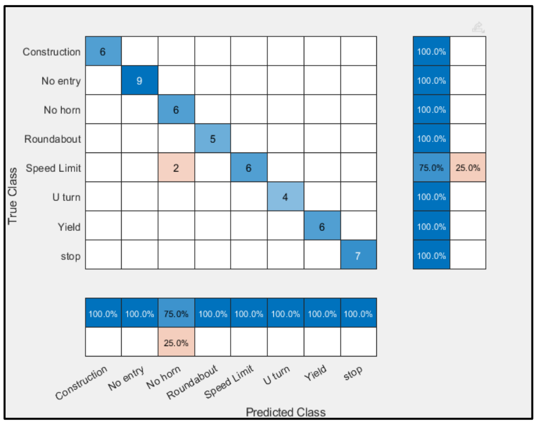
Training Cycle
Epoch: 40 of 40
Iteration: 40 of 40
Iterations per epoch: 1
Maximum iterations: 40

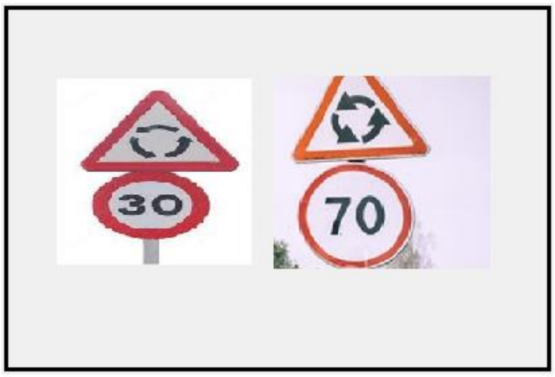
Validation
Frequency: N/A

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

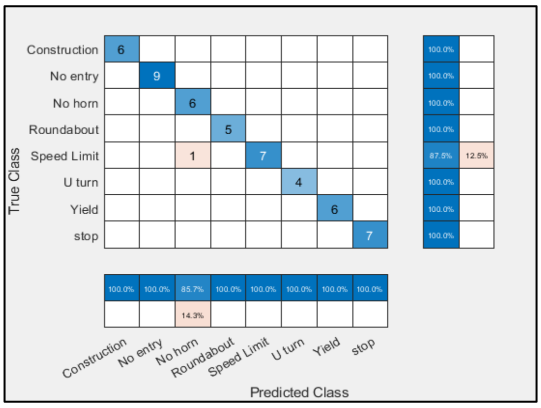
Confusion Matrix with Validation Set with Misclassified Images

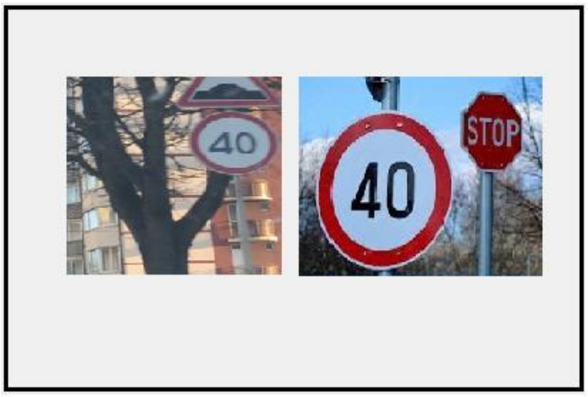
- EPOCH- 20, Learning Rate-0.001, SGDM optimizer



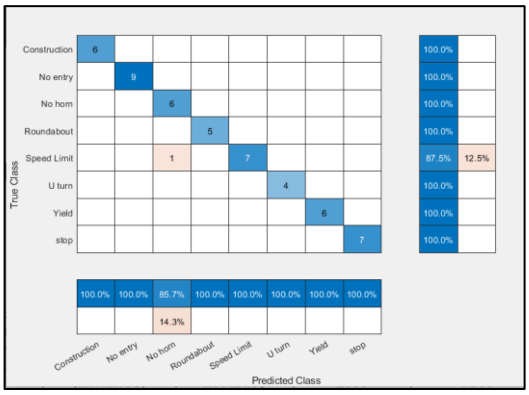


- EPOCH- 30, Learning Rate-0.001, SGDM optimizer





- EPOCH- 40, Learning Rate-0.001, SGDM optimizer



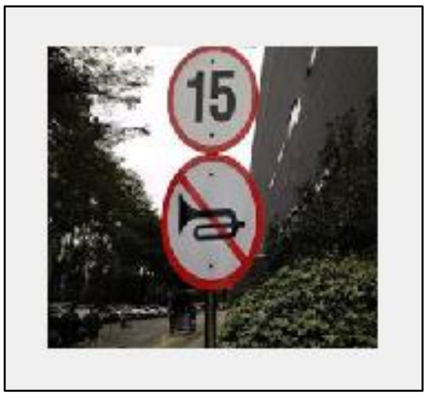
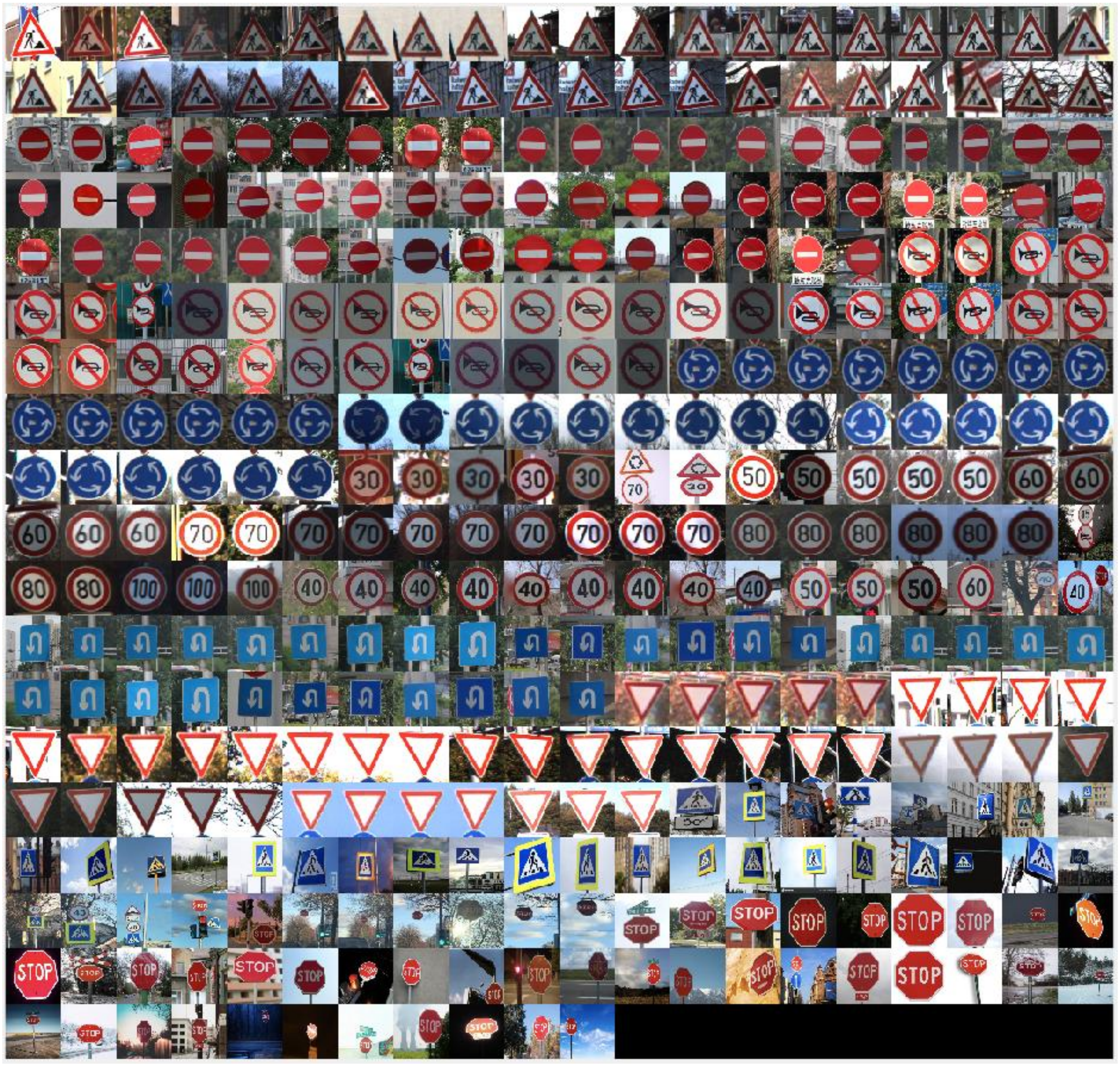


Image Datastore



GoogleNet Layers with specific layers updated as per categories of Traffic signs

	constraint	type	description
1	'data'	Image Input	224×224×3 images with 'zerocenter' normalization
2	'conv1-7x7_s2'	Convolution	64 7×7×3 convolutions with stride [2 2] and padding [3 3 3 3]
3	'conv1-relu_7x7'	ReLU	ReLU
4	'pool1-3x3_s2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 1 0 1]
5	'pool1-norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
6	'conv2-3x3_reduce'	Convolution	64 1×1×64 convolutions with stride [1 1] and padding [0 0 0 0]
7	'conv2-relu_3x3_reduce'	ReLU	ReLU
8	'conv2-3x3'	Convolution	192 3×3×64 convolutions with stride [1 1] and padding [1 1 1 1]
9	'conv2-relu_3x3'	ReLU	ReLU
10	'conv2-norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
11	'pool2-3x3_s2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 1 0 1]
12	'inception_3a-1x1'	Convolution	64 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]
13	'inception_3a-relu_1x1'	ReLU	ReLU
14	'inception_3a-3x3_reduce'	Convolution	96 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]
15	'inception_3a-relu_3x3_reduce'	ReLU	ReLU
16	'inception_3a-3x3'	Convolution	128 3×3×96 convolutions with stride [1 1] and padding [1 1 1 1]
17	'inception_3a-relu_3x3'	ReLU	ReLU
18	'inception_3a-5x5_reduce'	Convolution	16 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]
19	'inception_3a-relu_5x5_reduce'	ReLU	ReLU
20	'inception_3a-5x5'	Convolution	32 5×5×16 convolutions with stride [1 1] and padding [2 2 2 2]

21	'inception_3a-relu_5x5'	ReLU	ReLU
22	'inception_3a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
23	'inception_3a-pool_proj'	Convolution	32 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]
24	'inception_3a-relu_pool_proj'	ReLU	ReLU
25	'inception_3a-output'	Depth concatenation	Depth concatenation of 4 inputs
26	'inception_3b-1x1'	Convolution	128 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]
27	'inception_3b-relu_1x1'	ReLU	ReLU
28	'inception_3b-3x3_reduce'	Convolution	128 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]
29	'inception_3b-relu_3x3_reduce'	ReLU	ReLU
30	'inception_3b-3x3'	Convolution	192 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]
31	'inception_3b-relu_3x3'	ReLU	ReLU
32	'inception_3b-5x5_reduce'	Convolution	32 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]
33	'inception_3b-relu_5x5_reduce'	ReLU	ReLU
34	'inception_3b-5x5'	Convolution	96 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]
35	'inception_3b-relu_5x5'	ReLU	ReLU
36	'inception_3b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
37	'inception_3b-pool_proj'	Convolution	64 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]
38	'inception_3b-relu_pool_proj'	ReLU	ReLU
39	'inception_3b-output'	Depth concatenation	Depth concatenation of 4 inputs
40	'pool3-3x3_s2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 1 0 1]

41	'inception_4a-1x1'	Convolution	192 1×1×480 convolutions with stride [1 1] and padding [0 0 0 0]
42	'inception_4a-relu_1x1'	ReLU	ReLU
43	'inception_4a-3x3_reduce'	Convolution	96 1×1×480 convolutions with stride [1 1] and padding [0 0 0 0]
44	'inception_4a-relu_3x3_reduce'	ReLU	ReLU
45	'inception_4a-3x3'	Convolution	208 3×3×96 convolutions with stride [1 1] and padding [1 1 1 1]
46	'inception_4a-relu_3x3'	ReLU	ReLU
47	'inception_4a-5x5_reduce'	Convolution	16 1×1×480 convolutions with stride [1 1] and padding [0 0 0 0]
48	'inception_4a-relu_5x5_reduce'	ReLU	ReLU
49	'inception_4a-5x5'	Convolution	48 5×5×16 convolutions with stride [1 1] and padding [2 2 2 2]
50	'inception_4a-relu_5x5'	ReLU	ReLU
51	'inception_4a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
52	'inception_4a-pool_proj'	Convolution	64 1×1×480 convolutions with stride [1 1] and padding [0 0 0 0]
53	'inception_4a-relu_pool_proj'	ReLU	ReLU
54	'inception_4a-output'	Depth concatenation	Depth concatenation of 4 inputs
55	'inception_4b-1x1'	Convolution	160 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
56	'inception_4b-relu_1x1'	ReLU	ReLU
57	'inception_4b-3x3_reduce'	Convolution	112 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
58	'inception_4b-relu_3x3_reduce'	ReLU	ReLU
59	'inception_4b-3x3'	Convolution	224 3×3×112 convolutions with stride [1 1] and padding [1 1 1 1]
60	'inception_4b-relu_3x3'	ReLU	ReLU

61	'inception_4b-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
62	'inception_4b-relu_5x5_reduce'	ReLU	ReLU
63	'inception_4b-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] and padding [2 2 2 2]
64	'inception_4b-relu_5x5'	ReLU	ReLU
65	'inception_4b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
66	'inception_4b-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
67	'inception_4b-relu_pool_proj'	ReLU	ReLU
68	'inception_4b-output'	Depth concatenation	Depth concatenation of 4 inputs
69	'inception_4c-1x1'	Convolution	128 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
70	'inception_4c-relu_1x1'	ReLU	ReLU
71	'inception_4c-3x3_reduce'	Convolution	128 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
72	'inception_4c-relu_3x3_reduce'	ReLU	ReLU
73	'inception_4c-3x3'	Convolution	256 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]
74	'inception_4c-relu_3x3'	ReLU	ReLU
75	'inception_4c-5x5_reduce'	Convolution	24 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
76	'inception_4c-relu_5x5_reduce'	ReLU	ReLU
77	'inception_4c-5x5'	Convolution	64 5×5×24 convolutions with stride [1 1] and padding [2 2 2 2]
78	'inception_4c-relu_5x5'	ReLU	ReLU
79	'inception_4c-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
80	'inception_4c-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]

81	'inception_4c-relu_pool_proj'	ReLU	ReLU
82	'inception_4c-output'	Depth concatenation	Depth concatenation of 4 inputs
83	'inception_4d-1x1'	Convolution	112 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
84	'inception_4d-relu_1x1'	ReLU	ReLU
85	'inception_4d-3x3_reduce'	Convolution	144 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
86	'inception_4d-relu_3x3_reduce'	ReLU	ReLU
87	'inception_4d-3x3'	Convolution	288 3×3×144 convolutions with stride [1 1] and padding [1 1 1 1]
88	'inception_4d-relu_3x3'	ReLU	ReLU
89	'inception_4d-5x5_reduce'	Convolution	32 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
90	'inception_4d-relu_5x5_reduce'	ReLU	ReLU
91	'inception_4d-5x5'	Convolution	64 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]
92	'inception_4d-relu_5x5'	ReLU	ReLU
93	'inception_4d-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
94	'inception_4d-pool_proj'	Convolution	64 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
95	'inception_4d-relu_pool_proj'	ReLU	ReLU
96	'inception_4d-output'	Depth concatenation	Depth concatenation of 4 inputs
97	'inception_4e-1x1'	Convolution	256 1×1×528 convolutions with stride [1 1] and padding [0 0 0 0]
98	'inception_4e-relu_1x1'	ReLU	ReLU
99	'inception_4e-3x3_reduce'	Convolution	160 1×1×528 convolutions with stride [1 1] and padding [0 0 0 0]
100	'inception_4e-relu_3x3_reduce'	ReLU	ReLU

101	'inception_4e-3x3'	Convolution	320 3×3×160 convolutions with stride [1 1] and padding [1 1 1 1]
102	'inception_4e-relu_3x3'	ReLU	ReLU
103	'inception_4e-5x5_reduce'	Convolution	32 1×1×528 convolutions with stride [1 1] and padding [0 0 0 0]
104	'inception_4e-relu_5x5_reduce'	ReLU	ReLU
105	'inception_4e-5x5'	Convolution	128 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]
106	'inception_4e-relu_5x5'	ReLU	ReLU
107	'inception_4e-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
108	'inception_4e-pool_proj'	Convolution	128 1×1×528 convolutions with stride [1 1] and padding [0 0 0 0]
109	'inception_4e-relu_pool_proj'	ReLU	ReLU
110	'inception_4e-output'	Depth concatenation	Depth concatenation of 4 inputs
111	'pool4-3x3_s2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 1 0 1]
112	'inception_5a-1x1'	Convolution	256 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
113	'inception_5a-relu_1x1'	ReLU	ReLU
114	'inception_5a-3x3_reduce'	Convolution	160 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
115	'inception_5a-relu_3x3_reduce'	ReLU	ReLU
116	'inception_5a-3x3'	Convolution	320 3×3×160 convolutions with stride [1 1] and padding [1 1 1 1]
117	'inception_5a-relu_3x3'	ReLU	ReLU
118	'inception_5a-5x5_reduce'	Convolution	32 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
119	'inception_5a-relu_5x5_reduce'	ReLU	ReLU
120	'inception_5a-5x5'	Convolution	128 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]

121	'inception_5a-relu_5x5'	ReLU	ReLU
122	'inception_5a-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
123	'inception_5a-pool_proj'	Convolution	128 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
124	'inception_5a-relu_pool_proj'	ReLU	ReLU
125	'inception_5a-output'	Depth concatenation	Depth concatenation of 4 inputs
126	'inception_5b-1x1'	Convolution	384 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
127	'inception_5b-relu_1x1'	ReLU	ReLU
128	'inception_5b-3x3_reduce'	Convolution	192 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
129	'inception_5b-relu_3x3_reduce'	ReLU	ReLU
130	'inception_5b-3x3'	Convolution	384 3×3×192 convolutions with stride [1 1] and padding [1 1 1 1]
131	'inception_5b-relu_3x3'	ReLU	ReLU
132	'inception_5b-5x5_reduce'	Convolution	48 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
133	'inception_5b-relu_5x5_reduce'	ReLU	ReLU
134	'inception_5b-5x5'	Convolution	128 5×5×48 convolutions with stride [1 1] and padding [2 2 2 2]
135	'inception_5b-relu_5x5'	ReLU	ReLU
136	'inception_5b-pool'	Max Pooling	3×3 max pooling with stride [1 1] and padding [1 1 1 1]
137	'inception_5b-pool_proj'	Convolution	128 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]
138	'inception_5b-relu_pool_proj'	ReLU	ReLU
139	'inception_5b-output'	Depth concatenation	Depth concatenation of 4 inputs
140	'pool5-7x7_s1'	Global Average Pooling	Global average pooling

141	'pool5-drop_7x7_s1'	Dropout	40% dropout
142	'new_fc'	Fully Connected	8 fully connected layer
143	'prob'	Softmax	softmax
144	'new_out'	Classification Output	crossentropyex