Traffic Sign Detection- A Better Approach and Higher Recognition Using Convolution Neural Network

A PROJECT REPORT

Submitted by

Mervin Abey Abraham
Reg. No. 20MCA1011

in partial fulfillment for the award of the degree of

Master of Computer Applications

School of Computer Science and Engineering
Vellore Institute of Technology - Chennai Campus
Vandalur - Kelambakkam Road, Chennai - 600 127

June - 2022

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled **Traffic Sign Detection- A Better Approach and Higher Recognition Using Convolution Neural Network** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology - Chennai Campus, 600 127 in partial fulfillment of the requirements of the award of the degree of **Master of Computer Applications** is a bona-fide record of the work carried out by me under the supervision of **Prof. Vergin Raja Sarobin**. I further declare that the work reported in this project, has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Place: Chennai
Date: October 13, 2022

Signature of Candidate
(Mervin Abey Abraham)

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **Traffic Sign Detection- A Better Approach and Higher Recognition Using Convolution Neural Network** is prepared and submitted by **Mervin Abey Abraham (Reg. No. 20MCA1011)** to Vellore Institute of Technology - Chennai Campus, in partial fulfillment of the requirement for the award of the degree of **Master of Computer Applications** is a bona-fide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Guide/Supervisor	**Head Of The Department**
Name: Prof. Vergin Raja Sarobin	Name: Prof. Sivagami
Date:	Date:
Examiner	**Examiner**
Name:	Name:
Date:	Date:

(Seal of SCOPE)

Acknowledgement

It is my pleasure to express with deep sense of gratitude to Dr. Vergin Raja Sarobin M Assistant Professor, SCOPE, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of IoT and Artificial Intelligence. It is with gratitude that I would like to extend thanks to our honorable Chancellor Dr. G. Viswanathan, all the vice presidents Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Mr. G. V. Selvam, Assistant Vice–President Ms. Kadhambari S. Viswanathan, Vice-Chancellor Dr. Rambabu Kodali and Pro-Vice Chancellor Dr. Kanchana Bhaaskaran V. S. for providing an exceptional working environment and inspiring all of us during the course. In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Sivagami M., Head of the Department and Dr. Bhanu Chander Balusa Project Coordinator, M. C. A., SCOPE, Vellore Institiute of Technology, Chennai for their valuable support and encouragement to take up and complete the thesis. Special mention to Dean Dr. Ganesan R. and Associate Dean Dr. Geetha S., SCOPE, Vellore Institiute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Chennai
Date:

Mervin Abey Abraham
Reg. No. 20MCA1011

Abstract

In a Driving Assistance System (ADAS), one of the main system check is to find and recognize the road signs shown on the road to assist the driver ahead. This research paper aims to find traffic signs primary features and label them as well as focuses on recognizing them in oncoming traffic symbols and develop a Traffic Sign Recognition (TSR) system by performing image recognition and identifying the sign recognized using different filters and shapes , patterns etc. We have used CNN and different layers of CNN to understand why it has been the best model to categorize the images. In this research we will be creating all the different layers of CNN which will have 1 layer, 3 layer and 4 layer Convolutional Neural Network. We will attempt to create a faster model of CNN where we can classify images of 43 multi-class, single set images from a German Traffic Sign data set. We will do metric analysis on the all the convolutional neural network and find a test score which will have the highest accuracy on classification of traffic signs.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Road signs are an essential part of driving on the road since they are one of the primary ways in which the driver learns the route and its layout ahead of time. These signs are useful because they provide quick feedback to the driver and act as muscle memory, keeping the driver informed of what is about to unfold in terms of traffic maintenance, blocked roads, major accident signs, and so on. As a result, each government will produce traffic signs that each nation will use, and drivers will be able to grasp what the signals signify. On normal situations the signs will be able to see by the drivers and at low speeds as they will be zooming through the traffic in a highway and might miss the traffic sign and thus might cause an accident especially in a low lit roads and highways. The drivers who would be driving for long distances can really be tried from constantly looking what is there out on the road.

This is where the introduction of traffic detecting sensors and image classification which will come to help the drivers or assist them in driving better. For example, in an instance a truck driver is going at 80km/h, the driver gets distracted or doing any other activity like holding the phone and calling someone or in the worst case scenario the driver falls asleep and can cause the foot to pedal into more acceleration. The driver not being aware of what lies ahead is for sure dead on spot if he hits the side of the highway or a car which can even tip the truck and cause even more damage all around. Here, is where the Automatic Driving Assistance System will help the drivers from risking their lives. The split second action where the human can not take under high stress which is because the fingers and the nerve to stiff out of fear can be eliminated by the ADAS and move the vehicle out of imminent danger or can pull the brakes automatically.

This split second movement can be done in a better way where the image taken from the car can be classified and predict the traffic symbols, road lanes, barrier on the side, vehicles or objects present in front of the vehicle. The classification is one of the main parts where the prediction directly relies on it. If the prediction fails in identifying an image is completely because the algorithm was not taught properly and because the classification on the images have been failed. Therefore in this research paper we use convolutional neural networks to identify the traffic symbols and help one of the many parameters a camera sensor of a car would do.

1.1 Overview

Driver assistance in Traffic Sign identification and recognition also known as Advanced driver-assistance systems (ADAS) is a critical component of a sophisticated driver assistance system. Traffic symbols have various distinguishing characteristics that can be used to detect and identify them. They are created in precise colours and shapes, with the text or symbol standing out against the background. Because traffic signs are generally positioned upright and facing the camera, the incidence of circular and mathematical error is limited. Traffic symbol information, such as shape and colour, can be utilised to categorise traffic symbols.

1.2 Motivation

Many accidents happen because the drivers fail to see one of the simple and important steps that is reading the road properly. One can read the road majorly from traffic symbols which is telling what lies ahead but many of the drivers ignore it or completely miss them while going at huge speeds. This brings us the importance of having and ADAS where the car which is using has sensors around the car and is in an angle of viewing blind spots and traffic symbols ahead. This brings our research to have a traffic symbol classification and detection system.

1.3 Limitations

Limitations to ADAS is the highly incompetent drivers who advantage of the system, critical behaviour change in driving patterns can throw off the learning patterns of the system. 1.8 million accidents happen across the world and 90% of the accidents are caused by human error. These systems are created to help the drivers who are not able to see their blind spot or when there is sudden car lane change and driver is not able to come to a decision in a split second. These systems can help them and make the right choice ahead of time, where the human error in a car crash is minimized.

1.4 Advantages

In terms advantages a system can be more effective and intelligent in terms of the following factors:-

1. *Traffic Sign Identification*
2. *Traffic light prediction*
3. *Traffic-dependant cruise control*
4. *Parking spot availability monitoring*
5. *Increased safety by warning other drivers of dangerous conditions*
6. *Platooning for trucks and industrial vehicles*
7. *Pedestrian and personal mobility vehicle detection*
8. *Emergency vehicle approaching alerts*

1.5 Statement

In this research paper, our goals will be naturally to identify the signs well ahead using ROI which is Region of Interest of that traffic sign. Here, in order to find the region, we will have to do Data Augmentation to the images in the data set. Data augmentation is an essential technique in deep learning since deep

learning requires enormous quantities of data and it is not always possible to collect hundreds or millions of photos, hence data augmentation comes in handy. By doing so we will be able to find the traffic symbol much more easily. Therefore we use different operations like padding, sharpness, image enhancer which will deal in brightness of the image, truncate. Using this new set we will train the algorithm and help us classify the traffic symbols and make sure we have high accuracy and learn how the classification is in the algorithm.

1.6 Existing work

Most of the features are there in the current high end versions of electric vehicles where some vehicles have the most important features like:-

1. *Forward Collision Warning*
2. *Rear-View Cameras*
3. *Lane Departure Warning*
4. *Adaptive Headlights*
5. *Adaptive Cruise Control*

Forward collision warning is one of the demanding safety technology that every manufacturer would want for their vehicles. It will keep a watch on the road in front of the vehicle and initially warns the driver with flashing lights and loud noises if the sensors predict the vehicle is going to come in contact with an object. It would mainly rely on radar sensors and cameras around the vehicle.

Most rear-view cameras are manufacturer-installed, but you can also buy one as an aftermarket upgrade if your car didn't come with one. Like the forward collision warning system, this feature allows us to monitor how well we're staying in our lane using lane-detection technology. While adaptive headlights are available on many vehicles, they haven't yet become a standard feature.

In general vehicles have control system so that the driver does not have to keep punching on the pedal and the speed will be set on the highway and this feature uses senors around it which will have lasers, radar, cameras and other on-board

sensors to help the situation. Most of them are adaptive which means they will end as soon as the sensors detect something which will predict if its dangerous to the driver or not and pull the brake or disables the cruise so that the vehicle will slow down slowly. There are single radar systems which only detects the traffic in front of the vehicle.

Most forward driver assistance systems will either engage the brakes or vibrate the steering wheel so the that the driver is given a time of response and change directions in a split second of the time given to them. When to pull the brakes and how near one vehicle is to another or with an immovable objects are near to the vehicle are captured by its sensors which will be placed around the vehicle. This benefits lot of the cars on road and which makes driving on road much more safer for the drivers and pedestrians.

1.7 Challenges

Nevertheless, there are various issues that can impede the successful detection and recognition of traffic signals. These issues include changes in perspective, changes in lighting (including changes induced by shifting light levels, dusk, fog, and shadowing), occlusion of signs, motion blur, and weather worn degeneration of signs. Accuracy is critical since even a single misclassified or unnoticed indicator might have a negative impact on the motorist.

1.8 Approach

As this being one of the most critical problems to overcome in an ADAS which are the accuracy and recognition time the system took to identify the traffic sign. Most Systems have 89% - 95% accuracy where in this research, we have successfully attained to get a staggering 99% CNN model. With only 0.5% to misclassified images as wrong traffic signs.

- We first have to do the data augmentation to the data set which consists of:-
  - Sharpen the image
  - Brightness to the image

  - Cropping to the symbol
  - Padding around the image

- We then do split them into train set and test set
- We then compile the model in a Sequential order
- We then see if the model is a good fit or bad fit
- Visualize the data and find the better accuracy and loss trade off point.
- Compare it with the other models and see which convolutional model has the best accuracy and least loss

1.9 Statement of Assumptions

Note that weather conditions such as snowfall, frost, and even the sun might have an impact on how this function operates. One should check the car manual / handbook to find out where your sensors are so you can maintain them clean. It also goes without saying that you should not rely only on the collision system to avoid accidents, since the angle of the sun under certain situations may render it ineffective.

1.10 Research Objective

The aim of the research is to gain high accuracy and evolve the Convolutional Neutral Network using filters and parameters which helps us in the identification part. By doing so we can ensure that the identification is faster and easier for the system to recognize and predict on the outcome of the events taking place.

Chapter 2

Literature Review

As the world works to develop safer and more automated automobiles, advanced driver-assistance systems (ADAS) technology is rising to the top of the industry. With this requirement in mind, this paper outlines the significance of driver-assistance systems, their existing limits and shortcomings, and how sophisticated linked automobiles and roadside architecture might aid in the prevention of accidents.

TITLE: A Survey on Road Traffic Sign Recognition System using Convolution Neural Network
YEAR: 2018
AUTHOR: Hatolkar, Yuga and Agarwal, Poorva and Patil, Seema

Yuga Hatolkar of the Poorva Agrawal Symbiosis Institute of Technology (2018) [3] headed a team that investigated a survey on road traffic sign recognition systems using convolution neural networks. One of the solutions presented for Driver Assistance Systems (DAS) and Automated Driving is traffic symbol identification (AD). Almost a quarter of Indian drivers use Google Maps instead of paying attention to traffic signs on the road. This traffic sign recognition technology will make it easier for drivers to monitor traffic symbols and avoid accidents. The Fuzzy classification module functions as an optimizer module, enhancing the output of the convolution neural network.

TITLE: A Real Time Traffic Sign Detection and Recognition Algorithm based on Super Fuzzy Set
YEAR: 2017
AUTHOR: A. Khodayari

In a Real Time Traffic Sign Detection and Recognition Algorithm based on Super Fuzzy Set, a research group from the Mechanical Engineering Department led by A. Khodayari (2017) [6]. The intelligent driver adviser is traffic sign recognition (TSR) technology. Many academics have used fuzzy sets in their algorithms in TSR systems based on image processing. In this work, a novel machine vision method based on fuzzy logic for traffic sign detection will be presented. OpenCV was created to be computationally efficient, with a major emphasis on practical scenarios. It is utilised to create a traffic sign identification system by analysing photos collected on the road.

TITLE: Traffic Sign Detection- A New Approach and Recognition Using Convolution Neural Network
YEAR: 2017
AUTHOR: Dhar, Prashengit and Abedin

Vision-based traffic TSR is an important subject of study that draws the industry's research community on a regular basis. [1] The signs are for the drivers to understand and in a way manage the traffic on road and caution the drivers as well as pedestrians. There are signs of different kinds in their shapes, form, color, and so on of road traffic signs are different from country to country and some states have their own signs like Moose sign in America which is there in that county or state and not there in other parts of the world. Therefore creating a standard system for all the traffic signs is not possible. They have used real time traffic analysis and recognition as its an important part of the study. The suggested solution is for Bangladeshi traffic signs to have a red ring around them and a triangular shape. Color information is employed for segmentation, while CNN is used as a classifier.

TITLE: Automatic Detection and Recognition of Traffic Signs
YEAR: 2010
AUTHOR: Hossain, M Sajjad and Hasan

[4] We developed a novel algorithm for the automated detection and identification of traffic signs. We will continue to undertake research to increase the robustness of this new technique so that it can function better in a variety of atmospheres and brightness settings. Each government enforces a set of laws and regulations to create a safe transportation system. The categorization step is the second stage of automated detection and recognition of traffic signs. At this stage, each ROI is categorised based on its form and colour. Bangladeshi road markers are pre-defined in terms of colour and form. Red, blue, and yellow are the colours, while the shapes are circular, triangular, and rectangular.

TITLE: Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark
YEAR: 2013
AUTHOR: Houben, Sebastian and Stallkamp

[5] The German Traffic Sign Detection Benchmark (GTSDB) consists of a huge data collection of real-world photographs as well as a structured assessment process. We created a web interface that allows users to post, analyse, and score solutions. The benchmark asks all competitors to develop and train their detectors using the training set. We found it beneficial to categorise the indicators into three competition-relevant groups. Following the use of linear discriminant analysis, they had expanded the training set by P negative cases that were still identified by the existing intermediate classifier. The German Traffic Sign Detection Benchmark (GTSDB) consists of a huge data collection of real-world photographs as well as a structured assessment process. We created a web interface that allows users to post, analyse, and score solutions. The benchmark asks all competitors to develop and train their detectors using the training set. We found it beneficial to categorise the indicators into three competition-relevant groups. Following the use of linear discriminant analysis, they had expanded the training set by P negative cases that were still identified by the existing intermediate classifier.

TITLE: Real-time detection and recognition of road traffic signs
YEAR: 2012
AUTHOR: Greenhalgh, Jack and Mirmehdi, Majid

The proposed system consists of the following two main phases: detection and identification. Candidates for the traffic symbol are known as MSER, as described by Matas et al[2]. The identification phase is used to confirm the candidate's territory as a traffic sign and to classify a specific type of sign. The proposed system can operate in a range of vehicle speeds and lighting conditions. SVM with Gaussian kernel is used to classify each icon type based on shape and color. The total number of images in this data set was 43,509. The overall accuracy of this classifier was 89.2%, which was higher than the full synthetic or full actual data set. They had given an idea if one is to train the SVM classifier at a quick pace then it will contribute to the proposed method. They plan to give further comparisons with their other taxonomic methods later in their future work

TITLE: Road sign detection and recognition using colour segmentation, shape analysis and template matching
YEAR: 2007
AUTHOR: Malik, Rabia and Khurshid, Javaid and Ahmad, Sana Nazir

A team of academics has created an algorithm for recognising road signs with red borders and black symbols inside.

Automatic detection and identification of road signs is a crucial component in the design of the autonomous vehicle. Only the regions that are eligible for use as a road sign are included in the output image. The colour segmentation algorithm utilised is unaffected by illumination and shadows. This same technology is designed to operate with offline RGB photos. The system uses an RGB picture captured by a camera. It uses colour segmentation to extract all of the red zones. By sending each item through the form detector, the noncircular and nontriangular portions are deleted[7].

TITLE: Traffic sign detection based on color segmentation of obscure image candidates: a comprehensive study
YEAR: 2018
AUTHOR: Nandi, Dip and Saif

[8] Traffic sign detection researchers encounter challenges such as identifying the sign, categorising it, and differentiating one sign from another. For more than thirty years, the Embedded and Intelligent Automated Solution for Vehicles for Transportation Safety has been the focus of study in the Computer Vision and Pattern Recognition field. The most typical automated systems for Traffic Sign Detection and Recognition use one or two video cameras installed in front of the vehicle.Color Segmentation Accuracy = 93.3% in Daylight and 95% in Shadow respectively.Recognition Rate in Daylight 100%, Shadow 94.7% Training with only Positive samples i.e. real signs and after that mixing training data with 25,000 samples of real signs. They argue that at the current state of the art, no human machine interaction is seen, which might be a component of future study since it is critical for an effective Advanced Driving Assisting System. With a high-resolution camera and rapid frame rates, a quick system is required.

Chapter 3

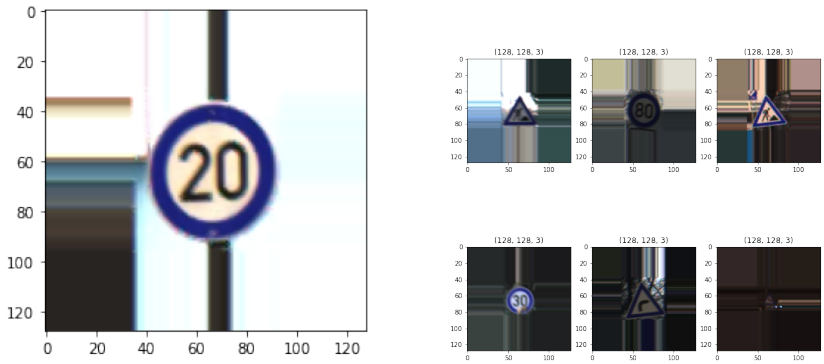
System Design

3.1 Proposed System

A automobile must be capable of understanding its surroundings in order to fully drive itself. Other cars, pedestrians, and road signs are also included. Road signs tell us about the legislation, notify us about harmful situations, and direct us to our intended destination. Many individuals might be badly wounded if an automobile cannot discern between signs, colours, and forms.

The way an automobile perceives the road is not the same as how we view it. We can all easily identify the difference between road signs and different traffic scenarios. When pictures are sent into a computer, they only perceive ones and zeros. That implies we'll have to educate the automobile to learn like people, or at the very least to recognise signs like us.

To tackle this issue, we have attempted to create a better and higher accurate convolutional neural network (CNN) to identify traffic signals. We have attempted to analyse different neural networks to understand and make a better system in classification. Since we are doing the CNN on 2 dimensional images, we will be using Conv2D and its different layers and how different parameters and filter will work on the convolutional neural network.



(a) Padding (b) Final Data Augmented Images

Figure 3.1: Few Data Augmented Images in the data set

The process of creating these convolutional network needs to have three major processes in this process: pre-processing the data, creating the convolutional neural network, and visualizing the result and understand how the neural networks are learning through its feature set. Therefore we will go step by step and explain in detail in the further below points.

The data set will first have to go through data augmentation and to do that each image in the data set will need to be data augmented which will go through a series of sharpening the image, brighten the dark images, cropping and padding around the image this will make the learning process more accurate for the feature set in the CNN.

Then we load in the data set into a train and test set by splitting them 80% and 20% ratio. After this we load the CNN model in a sequential order and add filters to the model. After this we will run a fit test to teach the algorithm and visualize if the data is being taught properly to the model and how the accuracy is having high accuracy and have minimal loss. More will be explained in further points below, let's first see more into the German traffic sign data set we will be using to teach the CNN models.

3.2 Data Set

The data set which we will be using is the German Traffic Sign Benchmark is a multi-class, single-image classification challenge which was held at the International Joint Conference on Neural Networks (IJCNN) in 2011.

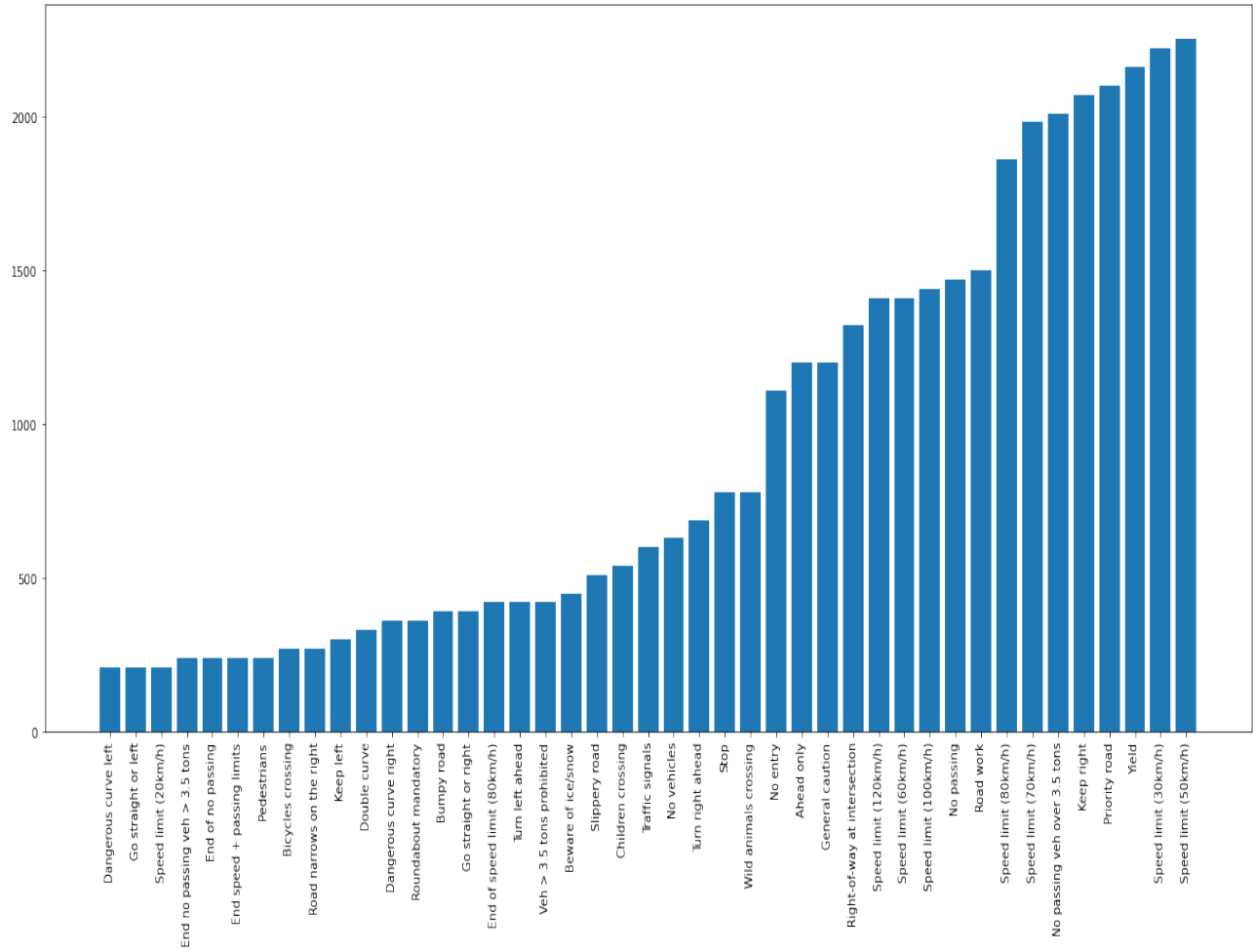


Figure 3.2: Plotting the data set on the basis of number of images in each class

This is a German Data set where there are 43 different classes of traffic signs.There are 39,209 train images and 12,630 test images. The following displayed image is the train data set. There are images ranging from 210 to as high as 2,250 images in a class. All the images in the data set are of different in nature to one another.

[12]

The 43 different classes of traffic signs are :-

1. Speed limit (20km/h
2. Speed limit (30km/h)
3. Speed limit (50km/h)
4. Speed limit (60km/h)
5. Speed limit (70km/h)
6. Speed limit (80km/h)
7. End of speed limit (80km/h)
8. Speed limit (100km/h)
9. Speed limit (120km/h)
10. No passing
11. No passing vehicle over 3.5 tons
12. Right-of-way at intersection
13. Priority road
14. Yield
15. Stop
16. No vehicles
17. Vehicle >3.5 tons prohibited
18. No entry
19. General caution
20. Dangerous curve left
21. Dangerous curve right

22. Double curve
23. Bumpy road
24. Slippery road
25. Road narrows on the right
26. Road work
27. Traffic signals
28. Pedestrians
29. Children crossing
30. Bicycles crossing
31. Beware of ice/snow
32. Wild animals crossing
33. End speed + passing limits
34. Turn right ahead
35. Turn left ahead
36. Ahead only
37. Go straight or right
38. Go straight or left
39. Keep right
40. Keep left
41. Roundabout mandatory
42. End of no passing
43. End no passing vehicle >3.5 tons

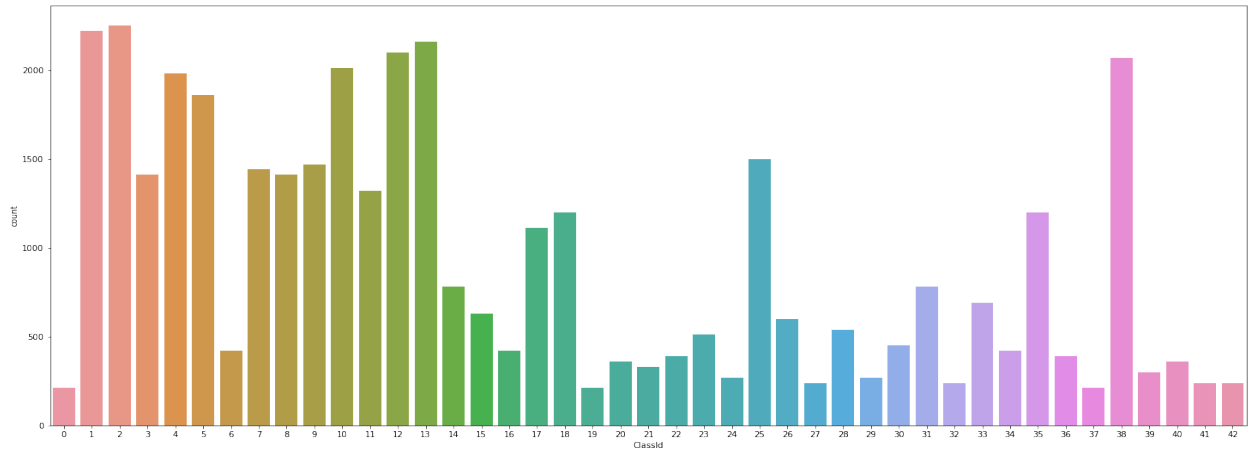


Figure 3.3: Checking class distribution in train data

Observations :

- The target classes are certainly not spread consistently.
- This is understandable given that warnings such as ”Keep speed below 30 km/h” or ”A bump ahead” appear more frequently than ones such as ”Road under construction ahead.”
- Each class has an unequal distribution of images. Some courses have about 2500 photos, while others only have 250.
- A smaller quantity of photographs might jeopardise the training process.
- We can utilise Data-Augmentation to compensate for a shortage of photos in some classes.

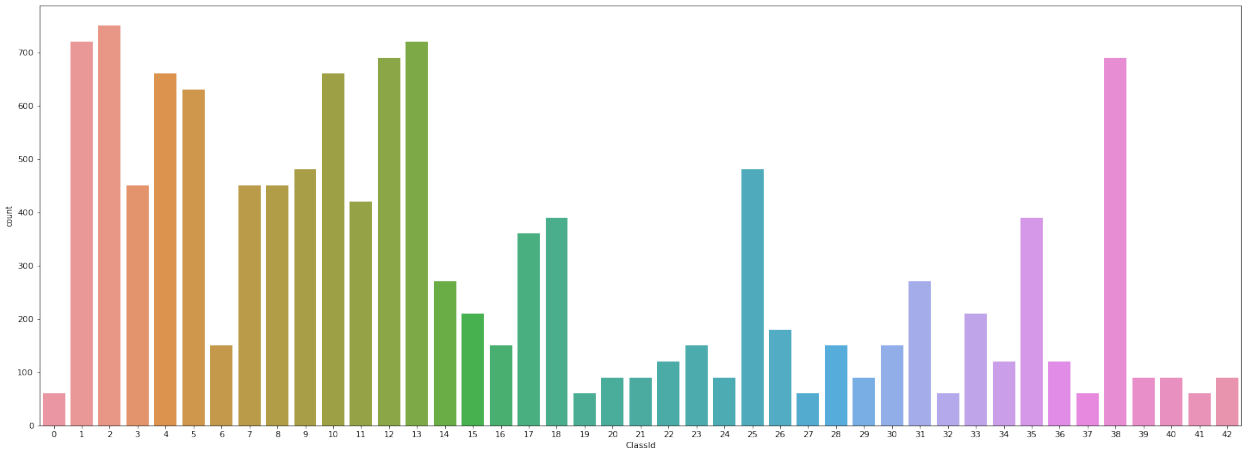


Figure 3.4: Checking class distribution in test data

Observations :

- The test data observations are all very identical to the train data observations.
- Some classes in this data are also having more frequent than others.

3.3 Data Flow Diagram

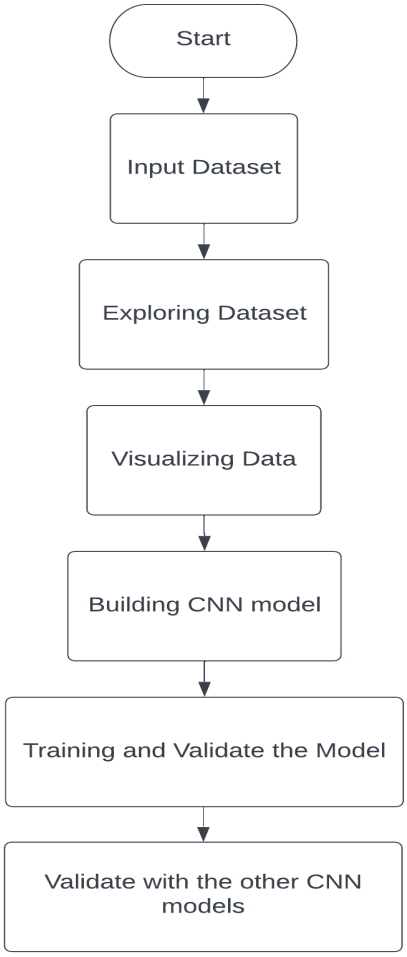


Figure 3.5: Data Flow Diagram - CNN Model

We input the images to a dataset and explore the dataset using matplotlib library and visualize the data . After doing the analysis on the dataset, we build the CNN models and split the data into train and test set for the CNN model to learn from the training images. Further we visualize if the model is over fitting or under fitting.

3.4 Convolutional Neural Network

3.4.1 2 dimensional CNN — Conv2D

In this research when we speak about Convolution Neural Network (CNN), we generally mean the 2-Dimensional CNN used for image categorization. This layer of convolutional layer is the one of the main element of CNN. It is a numerical technique that combines the collections of two data sets.To build a feature map, model parameters is convolutioned with a training data set which is then used on a convolution filter. There are numerous parameters of CNN and the following are the parameters used in this research. [10]

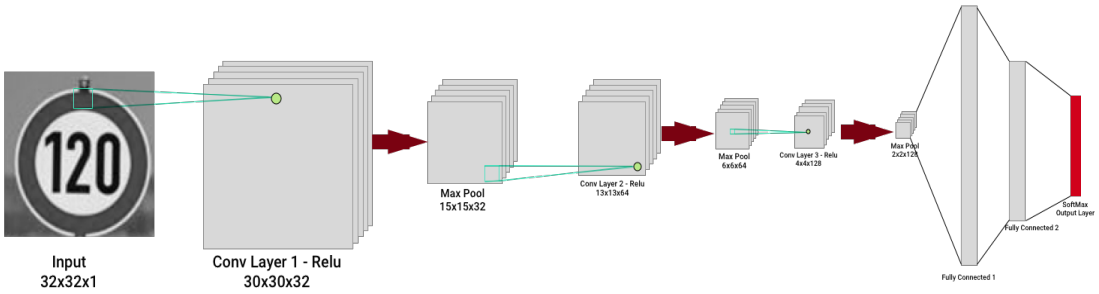


Figure 3.6: 3 layer CNN

- Conv2D: convolutional layer is created using this approach. The first parameter is the number of filters, and the second is the size of the filters. For example, in the first convolution layer, we generate 32 or 64 or 128 3x3 filters. As for activation, we employ real non-linearity.

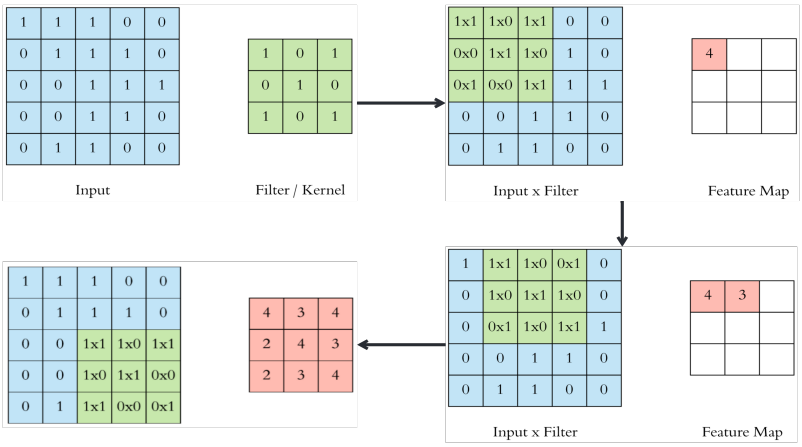


Figure 3.7: Input, Kernel and Feature Map

- MaxPooling2D: The sole argument is the window size, which is used to generate a max-pooling layer. Because it is the most frequent, we utilise a 2x2 window. We don't modify the default stride length, which is equal to the window size.

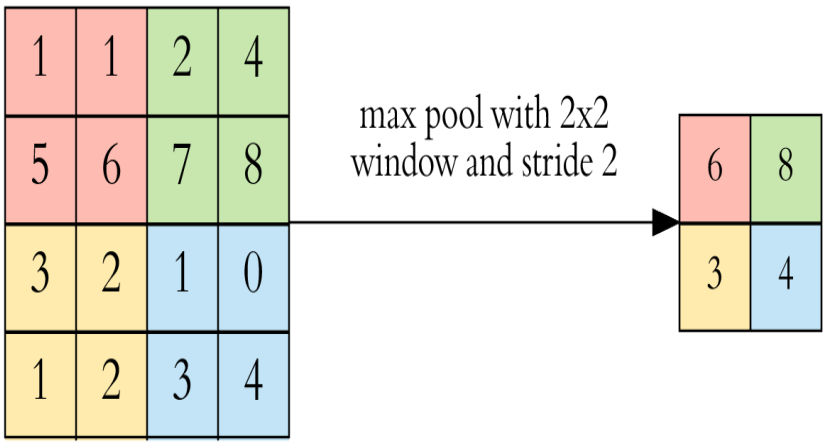


Figure 3.8: Input, Kernel and Feature Map

- Stride: Stride specifies how much to move to the CNN filter at each step. Therefore, stride is set to 1 by default for convolution layers, so its best we do not change its value.

- Flatten: After the process of the convolution and pooling layers, we have to flatten their output and feed into the fully connected layers. This is because the a fully connected layer expects the 1 dimensional vector of numbers but the output of both convolution and pooling are of #D volumes in nature. This causes us to flatten the output for the final polling layer to the vectors, this will turn it into a fully connected layer which will become the input. It is basically the procedure of arranging the 3-Dimensional volume into a 1-Dimensional vector

- Dropout: Dropout is a simple method for preventing over-fitting. During training, a neuron is momentarily ”dropped” or inhibited with probability p at each repetition. This signifies that all of this neuron's inputs and outputs will be inhibited for the current iteration. At each training step, the dropped-out neurons are re-sampled with probability p, thus a neuron that was dropped out in one step can be active in the next. For example, the 1st dropout layer will randomly disable 25% of the outputs.

- BatchNormalization: To normalize the input layers, we use the BatchNormalization layers to adjust and scale the activations. Batch Normalization reduces the amount by what the hidden unit values shift around (covariance shift). Also, it allows each layer of a network to learn by itself a little bit more independently of other layers.[11]

3.4.2 CNN - 1 Convolutional Layer

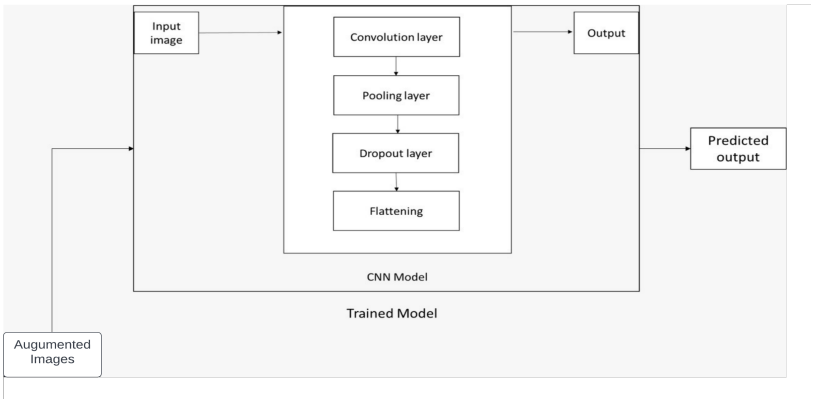


Figure 3.9: CNN - 1 Convolutional Layer

The augmented images after loading into the train set are called into the our 1st model of CNN where in the first Conv2D we go through in total of 34 filters, kernel size of 3*3 which specifies the height and width of the Conv2D convolution window, with activation of ReLu as it is much more faster in learning than sigmoid or Tanh and using Maxpooling we can effectively reduce the spacial values of the output we use powers of 2 as the spacial dimension values. Now to reduce the overfitting we use dropout of 0.2.

The next step is to feed the last output tensor into a stack of Dense layers, otherwise known as fully-connected layers. These densely connected classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Thus, we need to flatten the 3D outputs to 1D where we use flatten function to make the 3D volume of vector into 1D, and then add 2 dense layers on top. Now using keras compile function we can start compiling after specifying the architecture. We will use Adam optimizer with its default learning value which is 0.001

3.4.3 CNN - 3 Convolutional Layers

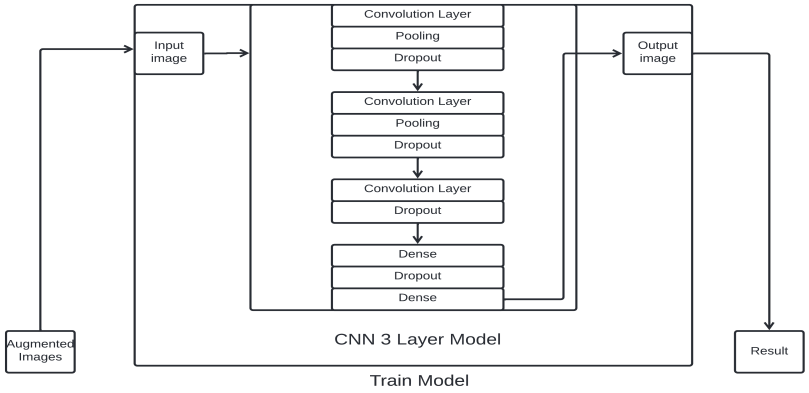


Figure 3.10: CNN - 3 Convolutional Layer

The augmented images after loading into the train set are called into the our model 3 of CNN where in the first Conv2D we go through in total of 34 filters, kernel size of 3*3 which specifies the height and width of the Conv2D convolution window, with activation of ReLu as it is much more faster in learning than sigmoid or Tanh and using Maxpooling we can effectively reduce the spacial values of the output we use powers of 2 as the spacial dimension values. Now to reduce the overfitting we use dropout of 0.25.

In the second Conv2D we go through in total of 64 filters, kernel size of 3*3 which specifies the height and width of the Conv2D convolution window, with activation of ReLu and using Maxpooling we can effectively reduce the spacial values of the output we use powers of 2 as the spacial dimension values. Now to reduce the overfitting we use dropout of 0.25.

In the third Conv2D we go through in total of 128 filters, kernel size of 3*3 which specifies the height and width of the Conv2D convolution window, with activation of ReLu. Now to reduce the overfitting we use dropout of 0.4.

The next step is to feed the last output tensor into a stack of Dense layers, otherwise known as fully-connected layers. These densely connected classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Thus, we need to flatten the 3D outputs to 1D where we use flatten function to make the 3D volume of vector into 1D, and then add 2 dense layers. Now using keras compile function we can start compiling after specifying the architecture. We will use Adam optimizer with its default learning value which is 0.001

3.4.4 CNN - 4 Convolutional Layers

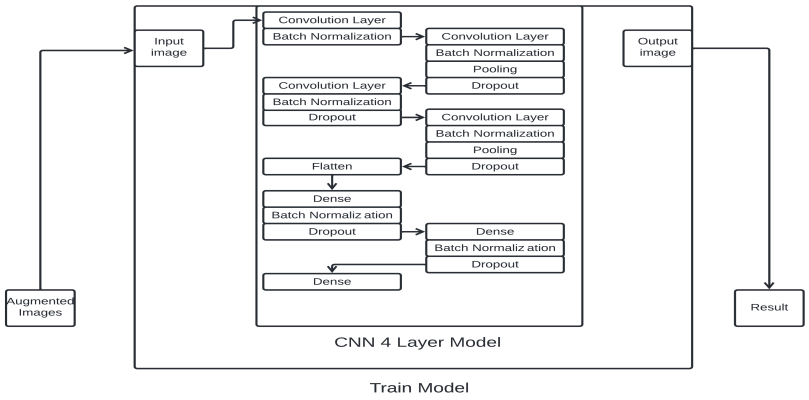


Figure 3.11: CNN - 4 Convolutional Layer

This CNN takes as input tensors of shape (Height, Width, image_channels). In this case, We configure the CNN to process inputs of size (28, 28, 1), which is the format of the German Traffic images. We do this by passing the argument input_shape=(28, 28, 1) to the first layer.

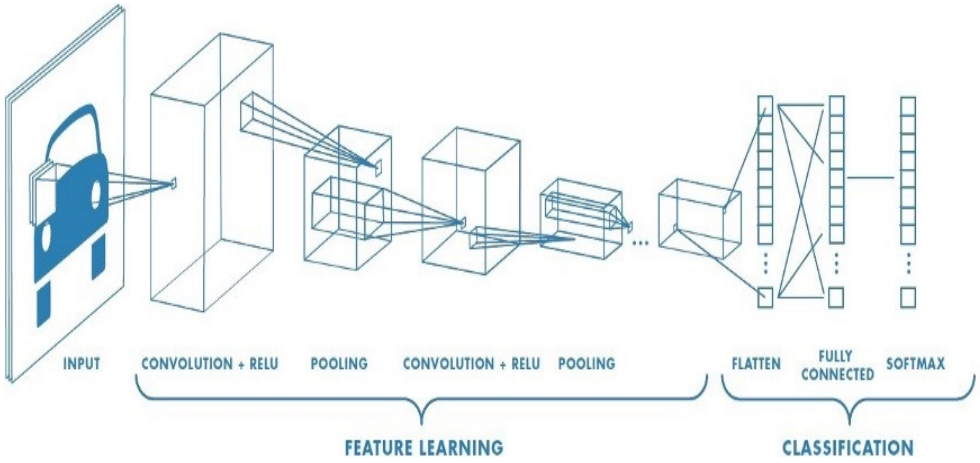


Figure 3.12: CNN - 4 Convolutional Layer Structure

The 1st layer is a Conv2D layer for the convolution operation that extracts features from the input images by sliding a convolution filter over the input to produce a feature map. Henceforth we have chosen feature map with size 3 x 3.

The 2nd layer is a MaxPooling2D layer for the max-pooling operation that reduces the dimensional of each feature, which helps shorten training time and reduce number of parameters. Therefore we have chosen the pooling window with size 2 x 2.

[13] To combat overfititng, we have added a Dropout layer as the 3rd layer, a powerful regularization technique. Dropout is the method used to reduce overfitting. It forces the model to learn multiple independent representations of the same data by randomly disabling neurons in the learning phase. In this model, dropout will randomly disable 20% of the neurons.

The next step is to feed the last output tensor into a stack of Dense layers, otherwise known as fully-connected layers. These densely connected classifiers process vectors, which are 1D, whereas the current output is a 3D tensor. Thus, we need to flatten the 3D outputs to 1D, and then add 2 Dense layers on top.[9]

We do a 43-way classification (as there are 43 classes of traffic sign images), using a final layer with 43 outputs and a softmax activation. Softmax activation enables me to calculate the output based on the probabilities. Each class is assigned a probability and the class with the maximum probability is the model's output for the input.

When compiling the model, we have chosen categorical_crossentropy as the loss function (which is relevant for multi-class, single-label classification problem) and Adam optimizer.

The cross-entropy loss calculates the error rate between the predicted value and the original value. The formula for calculating cross-entropy loss is given here. Categorical is used because there are 43 classes to predict from.

The Adam optimizer is an improvement over SGD(Stochastic Gradient Descent). The optimizer is responsible for updating the weights of the neurons via back-propagation. It calculates the derivative of the loss function with respect to each weight and subtracts it from the weight. That is how a neural network learns.

3.5 Requirement Specification

3.5.1 Recommended Hardware Requirement

- Processor : Intel Core i5 6th Generation processor or higher.
- HardDisk : 500GB HDD or 250GB SDD
- RAM : 16GB

3.5.2 Software Requirement

- Software : Jupyter Notebook
- Libraries : pandas, numpy, matplotlib.pyplot, tensorflow, seaborn, cv2, os, datetime, tqdm, sklearn, tensorflow, keras, PIL, math
- System Requirements: Windows 10
- Broswer: Chrome, Brave, any

Chapter 4

Implementation of system

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import cv2
import os
import datetime
from tqdm import tqdm
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from PIL import ImageEnhance, Image
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, concatenate, MaxPool2D, Dropout, Flatten, Dense
from tensorflow.keras.callbacks import ReduceLROnPlateau, TensorBoard, EarlyStopping, Callback
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
import math
import pickle
```

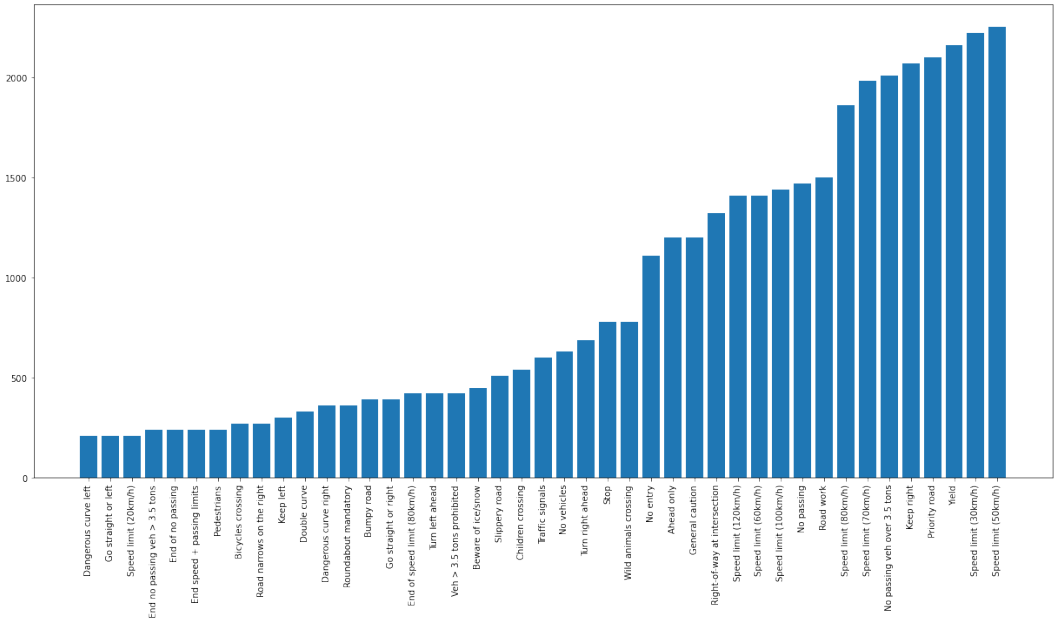
```
def load_train_images() :
    n_classes = 43
    data = []
    labels = []
    #Reading the train images
    for itr in range(n_classes) :
        path = 'Train/{0}/'.format(itr)
        Class = os.listdir(path)
        for c in Class :
            image = cv2.imread(path + c)
            image = Image.fromarray(image)
            data.append(np.array(image))
            labels.append(itr)
    data=np.array(data)
    labels=np.array(labels)
    return data, labels
```

```
def randomize_data(data, labels) :
    s=np.arange(data.shape[0])
    np.random.shuffle(s)
    data=data[s]
    labels=labels[s]
    return data, labels
```

```
def split_data(data, labels, val_size) :

    #Splitting images into train and validation sets
    (X_train, X_val) = data[(int)(val_size*len(labels)):], data[:(int)(val_size*len(labels))]

    #Normalization
    X_train = X_train.astype('float32')/255
    X_val = X_val.astype('float32')/255

    #Splitting labels
    (y_train, y_val) = labels[(int)(val_size*len(labels)):], labels[:(int)(val_size*len(labels))]

    #Performing OneHotEncoding
    enc = OneHotEncoder()
    enc.fit(y_train.reshape(-1,1))

    y_train = enc.transform(y_train.reshape(-1,1))
    y_val = enc.transform(y_val.reshape(-1,1))
    y_train = y_train.toarray()
    y_val = y_val.toarray()

    return X_train, X_val, y_train, y_val
```

```
# Label Overview
classes = { 0:'Speed limit (20km/h)',
            1:'Speed limit (30km/h)',
            2:'Speed limit (50km/h)',
            3:'Speed limit (60km/h)',
            4:'Speed limit (70km/h)',
            5:'Speed limit (80km/h)',
            6:'End of speed limit (80km/h)',
            7:'Speed limit (100km/h)',
            8:'Speed limit (120km/h)',
            9:'No passing',
            10:'No passing veh over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield',
            14:'Stop',
            15:'No vehicles',
            16:'Veh > 3.5 tons prohibited',
            17:'No entry',
            18:'General caution',
            19:'Dangerous curve left',
            20:'Dangerous curve right',
            21:'Double curve',
            22:'Bumpy road',
            23:'Slippery road',
            24:'Road narrows on the right',
            25:'Road work',
            26:'Traffic signals',
            27:'Pedestrians',
            28:'Children crossing',
            29:'Bicycles crossing',
            30:'Beware of ice/snow',
            31:'Wild animals crossing',
            32:'End speed + passing limits',
            33:'Turn right ahead',
            34:'Turn left ahead',
            35:'Ahead only',
            36:'Go straight or right',
            37:'Go straight or left',
            38:'Keep right',
            39:'Keep left',
            40:'Roundabout mandatory',
            41:'End of no passing',
            42:'End no passing veh > 3.5 tons' }
```

```
train_path = "Train"
folders = os.listdir(train_path)

train_number = []
class_num = []

for folder in folders:
    train_files = os.listdir(train_path + '/' + folder)
    train_number.append(len(train_files))
    class_num.append(classes[int(folder)])

# Sorting the dataset on the basis of number of images in each class
zipped_lists = zip(train_number, class_num)
sorted_pairs = sorted(zipped_lists)
tuples = zip(*sorted_pairs)
train_number, class_num = [ list(tuple) for tuple in  tuples]

# Plotting the number of images in each class
plt.figure(figsize=(21,10))
plt.bar(class_num, train_number)
plt.xticks(class_num, rotation='vertical')
plt.show()
```




```
import shutil
#Run this only once
#Creating validation directory
val_dir = "object_detection/data_sets/Validation"
os.mkdir(val_dir)

n_classes = 43
train_dir = "object_detection/data_sets/Train{0}"

#Moving files from train to validation directory
for n in tqdm(range(n_classes)) :
  path = os.path.join(val_dir, str(n))
  os.mkdir(path)
  src_path = train_dir.format('/' + str(n))
  files = os.listdir(src_path)
  rand_idx = random.sample(range(len(files)), math.ceil(len(files)/4))
  for idx in rand_idx :
    src = src_path + "/" + files[idx]
    shutil.move(src, path)
```

```
df_train = pd.read_csv('Train.csv')
labels_train = df_train['ClassId']

df_test = pd.read_csv('Test.csv')
labels_test = df_test['ClassId']
df_train.head()
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	27	26	5	5	22	20	20	Train/20/00020_00000_00000.png
1	28	27	5	6	23	22	20	Train/20/00020_00000_00001.png
2	29	26	6	5	24	21	20	Train/20/00020_00000_00002.png
3	28	27	5	6	23	22	20	Train/20/00020_00000_00003.png
4	28	26	5	5	23	21	20	Train/20/00020_00000_00004.png

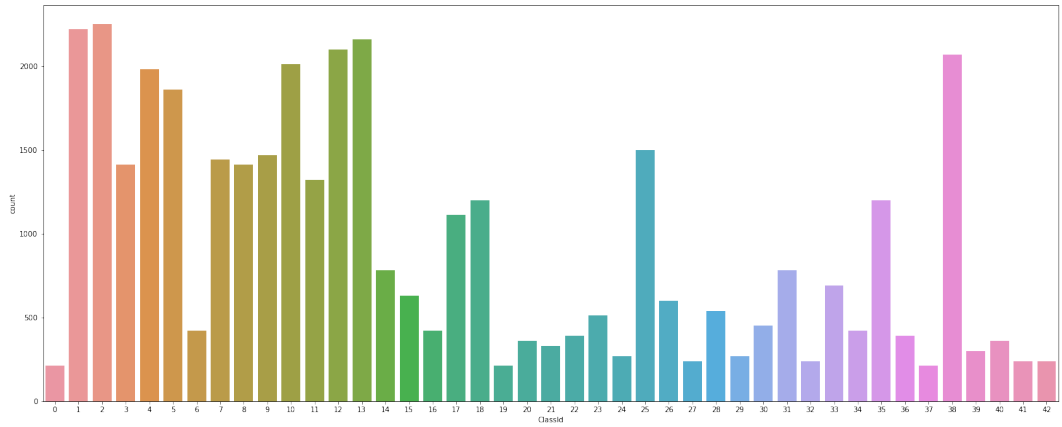
```
df_test.head()
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	53	54	6	5	48	49	16	Test/00000.png
1	42	45	5	5	36	40	1	Test/00001.png
2	48	52	6	6	43	47	38	Test/00002.png
3	27	29	5	5	22	24	33	Test/00003.png
4	60	57	5	5	55	52	11	Test/00004.png

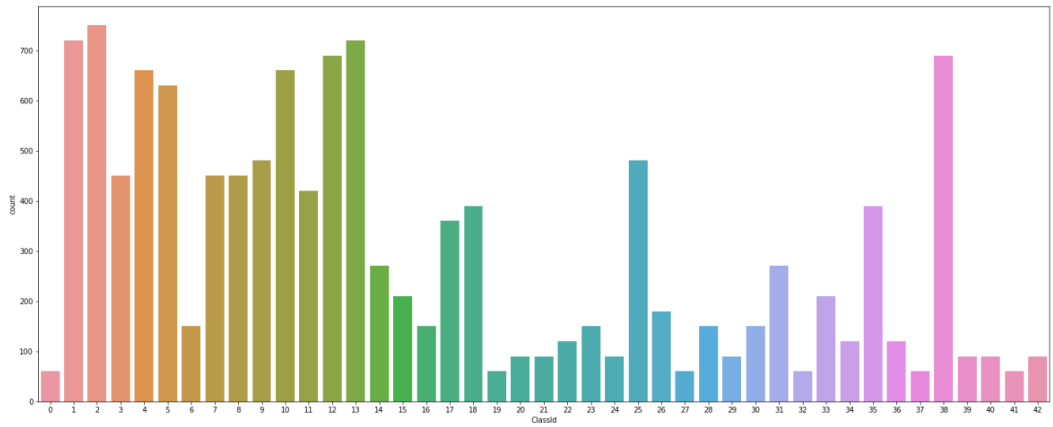
```
print("No of images in train data : {0}".format(df_train.shape[0]))
print("No of images in test data : {0}".format(df_test.shape[0]))
```

```
No of images in train data : 39209
No of images in test data : 12630
```

```
#Checking class distribution in train data
fig, ax = plt.subplots(1, 1)
fig.set_size_inches((25,10))
sns.countplot(data = df_train, x = 'ClassId')
plt.show()
```




```
#Checking class distribution in test data
fig, ax = plt.subplots(1, 1)
fig.set_size_inches((25,10))
sns.countplot(data = df_test, x = 'ClassId')
plt.show()
```



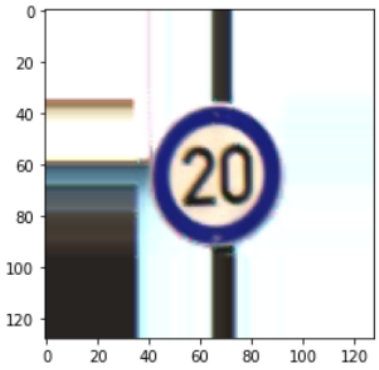

```
import random
from matplotlib.image import imread

test_file = pd.read_csv('Test.csv')
imgs = test_file["Path"].values

plt.figure(figsize=(25,25))

for i in range(1,26):
    plt.subplot(5,5,i)
    random_img_path = random.choice(imgs)
    rand_img = imread(random_img_path)
    plt.imshow(rand_img)
    plt.grid(b=None)
    plt.xlabel(rand_img.shape[1], fontsize = 20)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 20)#height of image
```


STOP
80
70
53
52
133
31
82
52
54
140
30
74
38
37
32
43
45
37
40
32
43
46
32
59
70
35
27
34
58
79
35
27
26
70
42
54
43
26
68
41
54
42
63
40
34
42
58
62
37
32
41
54

```
data, labels = load_train_images()
```

```
path = 'Train/0/00000_00004_00007.png'
image = cv2.imread(path)
if image.shape[0] <128 :
    diff = 128 - image.shape[0]
    top = math.ceil(diff/2)
    bottom = math.floor(diff/2)
if image.shape[1] <128 :
    diff = 128 - image.shape[1]
    left = math.ceil(diff/2)
    right = math.floor(diff/2)
image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_REPLICATE)
print(image.shape)
image = Image.fromarray(image)
plt.imshow(image)
plt.show()
```

```
(128, 128, 3)
```

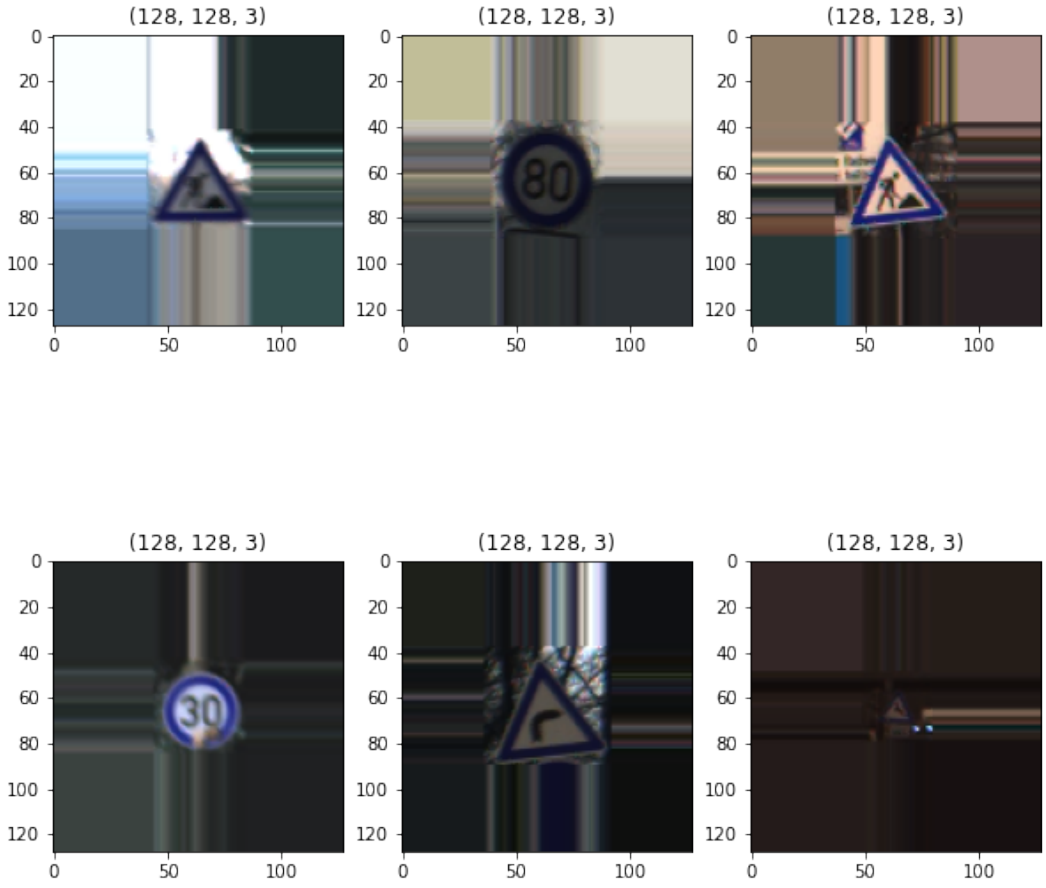


```
def padding(image) :
    top, bottom, left, right = 0, 0, 0, 0
    if image.shape[0] < 128 :
        diff = 128 - image.shape[0]
        top = math.ceil(diff/2)
        bottom = math.floor(diff/2)
    if image.shape[1] < 128 :
        diff = 128 - image.shape[1]
        left = math.ceil(diff/2)
        right = math.floor(diff/2)
    image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_REPLICATE)
    return image

def truncate(image) :
    image = image[0:128, 0:128]
    return image

def load_train_images_preprocessed() :

    n_classes = 43
    data = []
    labels = []
    #Reading the train images
    for itr in tqdm(range(n_classes)) :
        path = 'Train/{0}/'.format(itr)
        Class = os.listdir(path)

        for c in Class :
            image = cv2.imread(path + c)
            image = padding(image)
            image = truncate(image)
            image = Image.fromarray(image, 'RGB')
            data.append(np.array(image))
            labels.append(itr)

    data=np.array(data)
    labels=np.array(labels)
    return data, labels
```

```
#Loading the preprocessed images
import random
data_preprocessed, new_labels = load_train_images_preprocessed()
```

```
100%|██████████████████████████████████████████████████████████████████████████| 43/43 [00:48<00:00,  1.12s/it]
```

```
fig, ax = plt.subplots(2,3)
height, width = 10, 10

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[0][0].title.set_text(data_preprocessed[seed].shape)
ax[0][0].imshow(data_preprocessed[seed])

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[0][1].title.set_text(data_preprocessed[seed].shape)
ax[0][1].imshow(data_preprocessed[seed])

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[0][2].title.set_text(data_preprocessed[seed].shape)
ax[0][2].imshow(data_preprocessed[seed])

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[1][0].title.set_text(data_preprocessed[seed].shape)
ax[1][0].imshow(data_preprocessed[seed])

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[1][1].title.set_text(data_preprocessed[seed].shape)
ax[1][1].imshow(data_preprocessed[seed])

fig.set_size_inches(height, width)
seed = random.randrange(0, data_preprocessed.shape[0])
ax[1][2].title.set_text(data_preprocessed[seed].shape)
ax[1][2].imshow(data_preprocessed[seed])
```




```
def channel_count(data) :
    n_channels = []
    ret_dict = {}
    for itr in range(len(data)) :
        img = data[itr]
        if img.ndim == 2 :
            channels = 1
        if img.ndim == 3 :
            channels = img.shape[-1]
        n_channels.append(channels)
    tmp = set(n_channels)
    for ele in tmp :
        ret_dict[ele] = n_channels.count(ele)
    return ret_dict
```

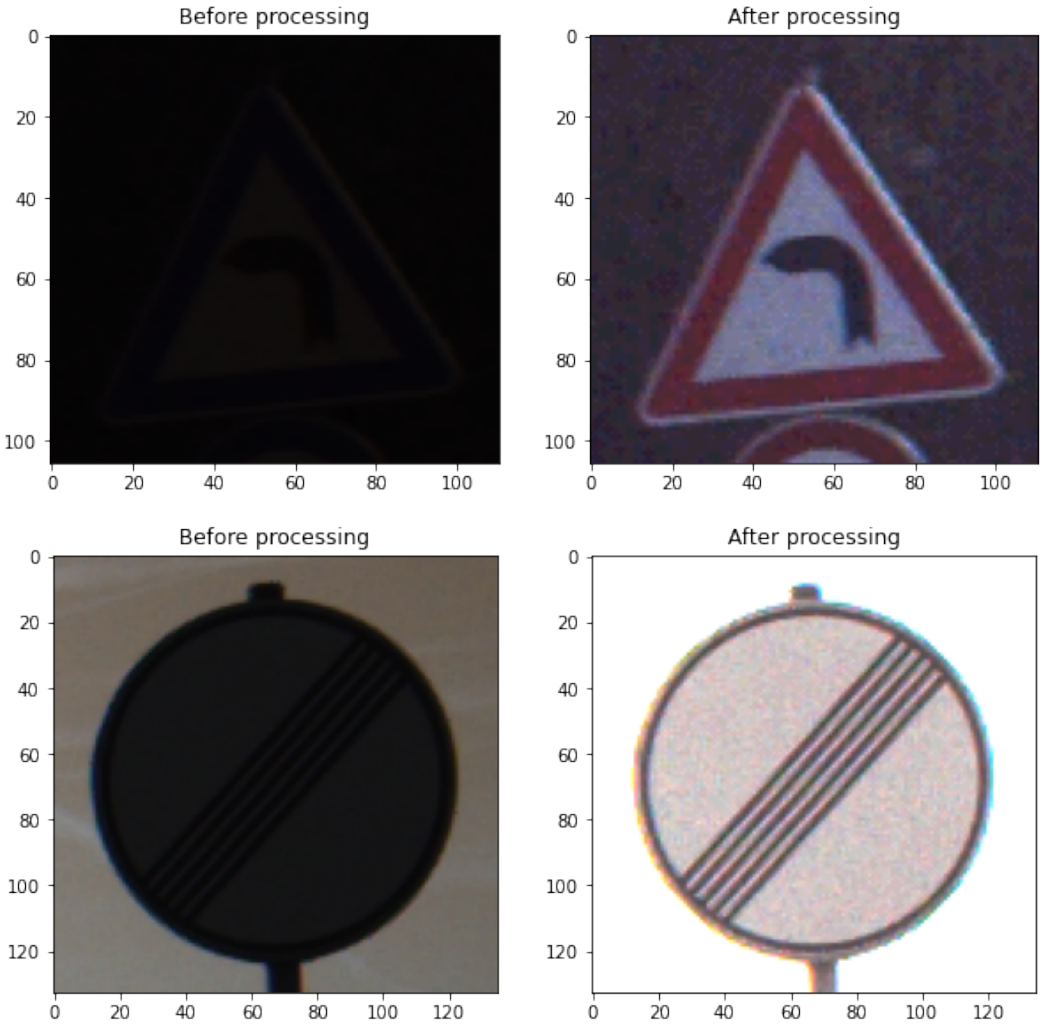
```
n_channels = channel_count(data)
for idx, ele in n_channels.items() :
  print("There are {0} {1}-channeled images".format(ele, idx))
```

```
There are 39209 3-channeled images
```

```
#Dealing with bright and dark images :

height, width = 10, 10
fig, ax = plt.subplots(2,3)
fig.set_size_inches(height, width)
image = cv2.imread("Train/19/00019_00003_00027.png")
image = Image.fromarray(image)
ax[0][0].imshow(image)
fig.set_size_inches(height, width)
image = cv2.imread("Train/19/00019_00004_00029.png")
image = Image.fromarray(image)
ax[0][1].imshow(image)
fig.set_size_inches(height, width)
image = cv2.imread("Train/25/00025_00000_00026.png")
image = Image.fromarray(image)
ax[0][2].imshow(image)
fig.set_size_inches(height, width)
image = cv2.imread("Train/32/00032_00000_00029.png")
image = Image.fromarray(image)
ax[1][0].imshow(image)
fig.set_size_inches(height, width)
image = cv2.imread("Train/34/00034_00000_00028.png")
image = Image.fromarray(image)
ax[1][1].imshow(image)
fig.set_size_inches(height, width)
image = cv2.imread("Train/3/00003_00000_00028.png")
image = Image.fromarray(image)
ax[1][2].imshow(image)
```

```
#Using Data Augmentation
im = Image.open('Train/19/00019_00004_00029.png')
enhancer = ImageEnhance.Brightness(im)
enhanced_im1 = enhancer.enhance(10)
im = Image.open("Train/32/00032_00000_00029.png")
enhancer = ImageEnhance.Brightness(im)
enhanced_im2 = enhancer.enhance(10)
```

```
height, width = 10, 10
fig, ax = plt.subplots(2,2)
fig.set_size_inches(height, width)

image = cv2.imread('Train/19/00019_00004_00029.png')
image = Image.fromarray(image)
ax[0][0].title.set_text('Before processing')
ax[0][0].imshow(image)
ax[0][1].title.set_text('After processing')
ax[0][1].imshow(enhanced_im1)

image = cv2.imread("Train/32/00032_00000_00029.png")
image = Image.fromarray(image)
ax[1][0].title.set_text('Before processing')
ax[1][0].imshow(image)
ax[1][1].title.set_text('After processing')
ax[1][1].imshow(enhanced_im2)
```


Before processing
After processing
Before processing
After processing

```
data = []
labels = []
classes = 43

for i in range(classes):
    path = os.path.join(os.getcwd(),'Train',str(i))
    images = os.listdir(path)

    for j in images:
        try:
            image = Image.open(path + '\\'+ j)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

```
data = np.array(data)
labels = np.array(labels)
```

```
print(data.shape, labels.shape)
```

```
(39209, 30, 30, 3) (39209,)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=68) # 0.25 x 0.8 = 0.2

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=68) # 0.25 x 0.8 = 0.2

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape,X_val.shape,y_val.shape)
```

```
(25093, 30, 30, 3) (7842, 30, 30, 3) (25093, 43) (7842, 43) (6274, 30, 30, 3) (6274, 43)
```

```
from tensorflow.keras.utils import to_categorical
#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367, 43) (7842, 43)
```

```
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential, load_model
from keras.optimizers import adam_v2

input_shape = X_train.shape[1:]

def CNN1():
    cnn1 = Sequential()
    cnn1.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    cnn1.add(MaxPooling2D(pool_size=(2, 2)))
    cnn1.add(Dropout(0.2))

    cnn1.add(Flatten())

    cnn1.add(Dense(256, activation='relu'))
    cnn1.add(Dense(43, activation='softmax'))

    cnn1.compile(adam_v2.Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

    return cnn1
```

```
cnn1 = CNN1()
```

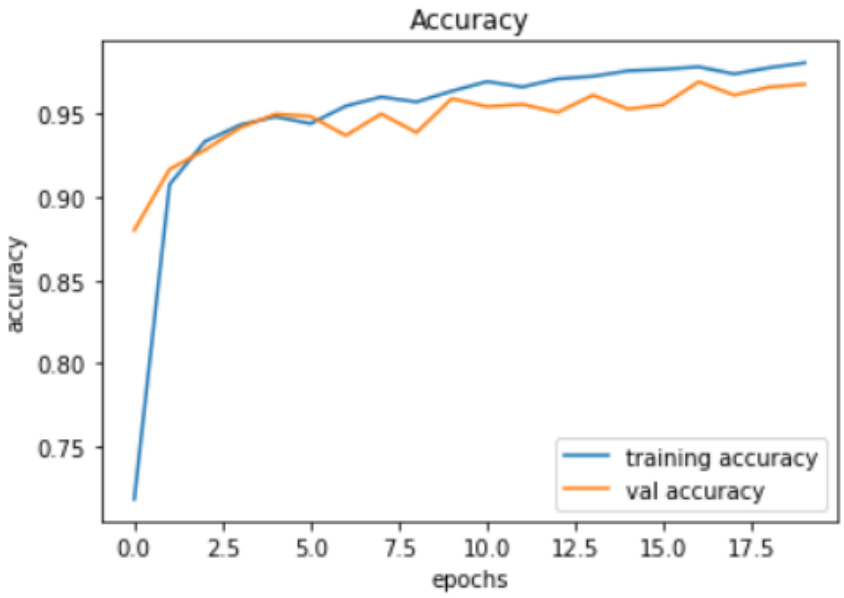
```
cnn1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        896

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 dropout (Dropout)           (None, 14, 14, 32)        0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 256)               1605888

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 1,617,835
Trainable params: 1,617,835
Non-trainable params: 0
_________________________________________________________________
```

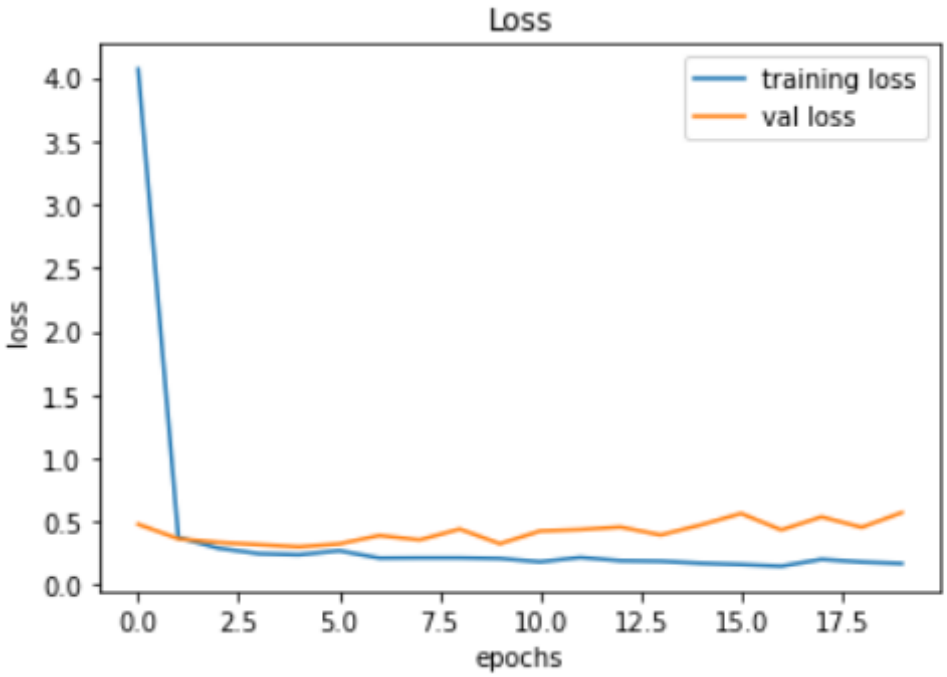
```
#Final trainig of model
history1 = cnn1.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_val, y_val))

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Epoch 1/20
981/981 [==============================] - 15s 15ms/step - loss: 4.0651 - accuracy: 0.7186 - val_loss: 0.4802 - val_accuracy:
0.8800
Epoch 2/20
981/981 [==============================] - 14s 15ms/step - loss: 0.3761 - accuracy: 0.9074 - val_loss: 0.3656 - val_accuracy:
0.9165
Epoch 3/20
981/981 [==============================] - 15s 15ms/step - loss: 0.2908 - accuracy: 0.9331 - val_loss: 0.3369 - val_accuracy:
0.9281
Epoch 4/20
981/981 [==============================] - 15s 15ms/step - loss: 0.2484 - accuracy: 0.9431 - val_loss: 0.3194 - val_accuracy:
0.9415
Epoch 5/20
981/981 [==============================] - 16s 16ms/step - loss: 0.2398 - accuracy: 0.9480 - val_loss: 0.3023 - val_accuracy:
0.9496
Epoch 6/20
981/981 [==============================] - 15s 16ms/step - loss: 0.2711 - accuracy: 0.9439 - val_loss: 0.3251 - val_accuracy:
0.9482
Epoch 7/20
981/981 [==============================] - 16s 17ms/step - loss: 0.2136 - accuracy: 0.9545 - val_loss: 0.3917 - val_accuracy:
0.9368
Epoch 8/20
981/981 [==============================] - 17s 17ms/step - loss: 0.2152 - accuracy: 0.9599 - val_loss: 0.3567 - val_accuracy:
0.9498
Epoch 9/20
981/981 [==============================] - 17s 18ms/step - loss: 0.2153 - accuracy: 0.9569 - val_loss: 0.4421 - val_accuracy:
0.9385
Epoch 10/20
981/981 [==============================] - 17s 18ms/step - loss: 0.2101 - accuracy: 0.9634 - val_loss: 0.3266 - val_accuracy:
```

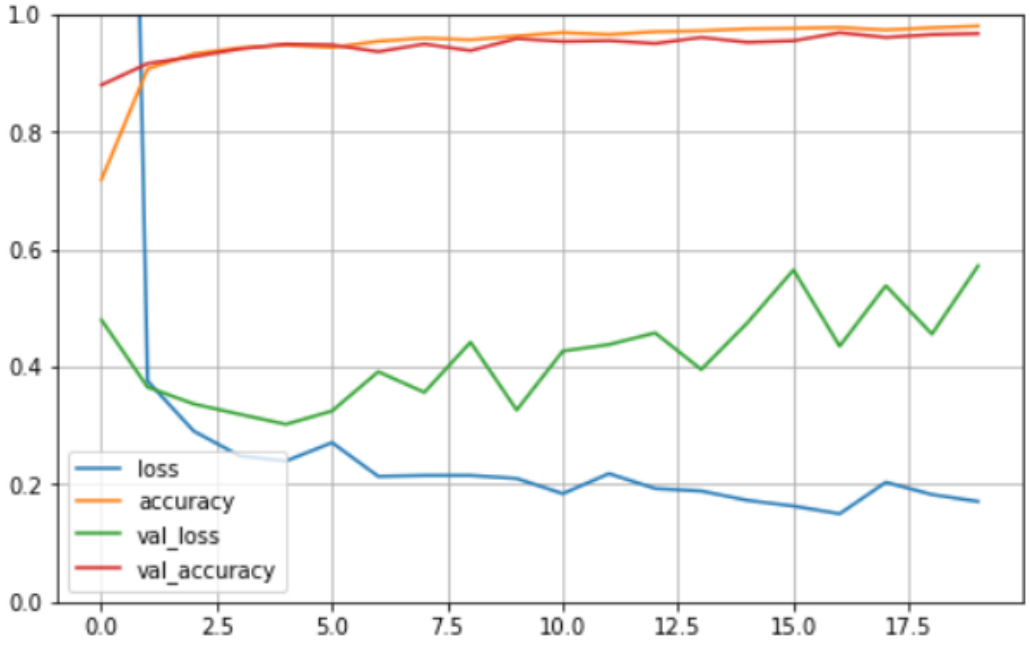
```
#plotting graphs for accuracy
plt.figure(0)
plt.plot(history1.history['accuracy'], label='training accuracy')
plt.plot(history1.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



```
#plotting graphs for loss
plt.figure(1)
plt.plot(history1.history['loss'], label='training loss')
plt.plot(history1.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



```
pd.DataFrame(history1.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```




```
score = cnn1.evaluate(X_test, y_test, verbose=0)

print('Test Score:', score[0])
print('Test Accuracy:', score[1])
```

```
Test Score: 0.5722211003303528
Test Accuracy: 0.9674828052520752
```

2. CNN with 3 Convolutional Layer

```
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential, load_model
from keras.optimizers import adam_v2

input_shape = X_train.shape[1:]

def CNN3():
    cnn3 = Sequential()
    cnn3.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    cnn3.add(MaxPooling2D((2, 2)))
    cnn3.add(Dropout(0.25))

    cnn3.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    cnn3.add(MaxPooling2D(pool_size=(2, 2)))
    cnn3.add(Dropout(0.25))

    cnn3.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    cnn3.add(Dropout(0.4))

    cnn3.add(Flatten())

    cnn3.add(Dense(256, activation='relu'))
    cnn3.add(Dropout(0.3))
    cnn3.add(Dense(43, activation='softmax'))

    cnn3.compile(adam_v2.Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])


    return cnn3
```

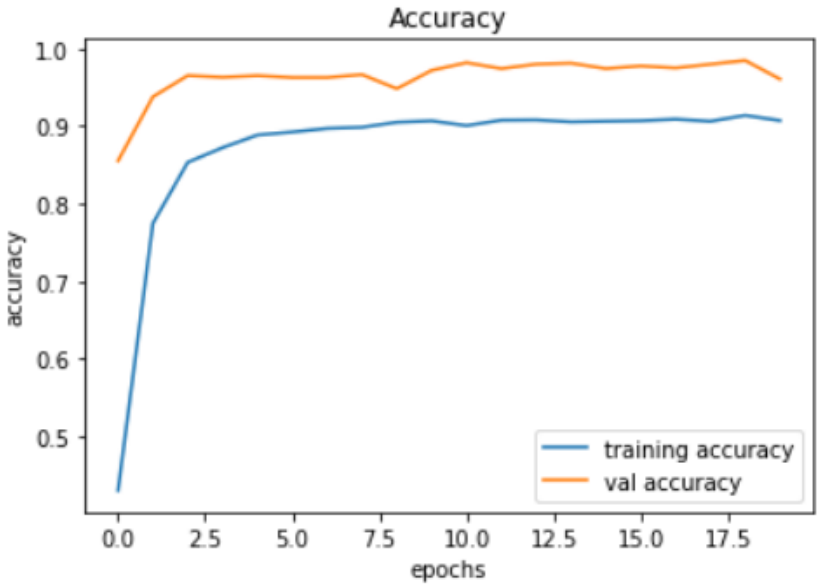
```
cnn3.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)            (None, 28, 28, 32)        896

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 dropout_1 (Dropout)          (None, 14, 14, 32)        0

 conv2d_2 (Conv2D)            (None, 12, 12, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 dropout_2 (Dropout)          (None, 6, 6, 64)          0

 conv2d_3 (Conv2D)            (None, 4, 4, 128)         73856

 dropout_3 (Dropout)          (None, 4, 4, 128)         0

 flatten_1 (Flatten)          (None, 2048)              0

 dense_2 (Dense)              (None, 256)               524544

 dropout_4 (Dropout)          (None, 256)               0

 dense_3 (Dense)              (None, 43)                11051

=================================================================
Total params: 628,843
Trainable params: 628,843
Non-trainable params: 0
_________________________________________________________________
```

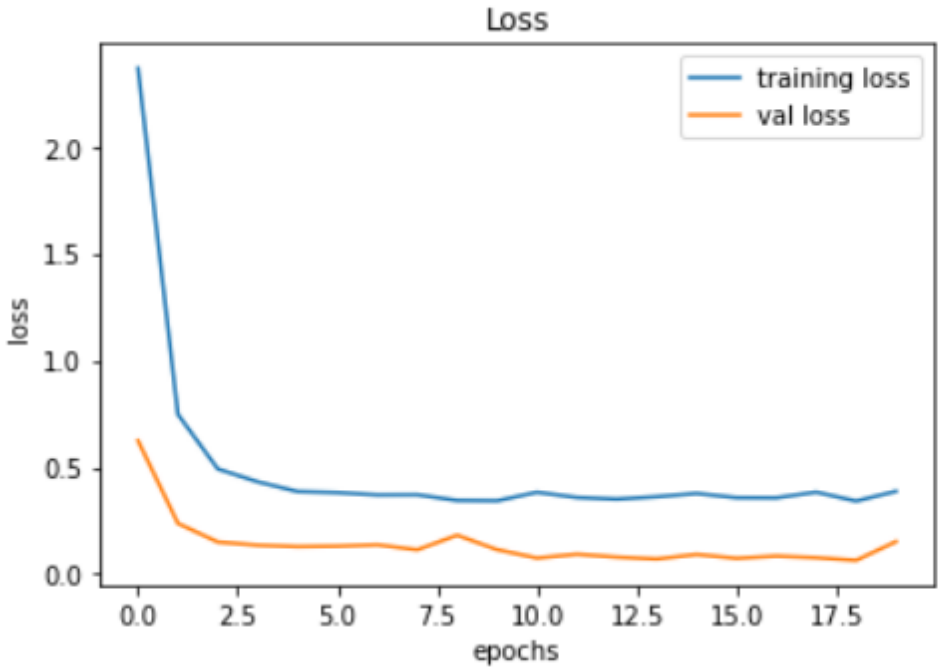
```
#Final trainig of model
history2 = cnn3.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_val, y_val))

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Epoch 1/20
981/981 [==============================] - 22s 22ms/step - loss: 2.3660 - accuracy: 0.4305 - val_loss: 0.6242 - val_accuracy: 0.8554
Epoch 2/20
981/981 [==============================] - 22s 23ms/step - loss: 0.7472 - accuracy: 0.7748 - val_loss: 0.2373 - val_accuracy: 0.9379
Epoch 3/20
981/981 [==============================] - 21s 21ms/step - loss: 0.4918 - accuracy: 0.8535 - val_loss: 0.1502 - val_accuracy: 0.9653
Epoch 4/20
981/981 [==============================] - 21s 22ms/step - loss: 0.4318 - accuracy: 0.8724 - val_loss: 0.1355 - val_accuracy: 0.9631
Epoch 5/20
981/981 [==============================] - 21s 22ms/step - loss: 0.3862 - accuracy: 0.8886 - val_loss: 0.1290 - val_accuracy: 0.9652
Epoch 6/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3812 - accuracy: 0.8925 - val_loss: 0.1311 - val_accuracy: 0.9628
Epoch 7/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3705 - accuracy: 0.8971 - val_loss: 0.1374 - val_accuracy: 0.9628
Epoch 8/20
981/981 [==============================] - 20s 21ms/step - loss: 0.3711 - accuracy: 0.8986 - val_loss: 0.1140 - val_accuracy: 0.9666
Epoch 9/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3446 - accuracy: 0.9051 - val_loss: 0.1822 - val_accuracy: 0.9484
Epoch 10/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3431 - accuracy: 0.9068 - val_loss: 0.1145 - val_accuracy: 0.9719
Epoch 11/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3828 - accuracy: 0.9008 - val_loss: 0.0755 - val_accuracy: 0.9819
Epoch 12/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3586 - accuracy: 0.9078 - val_loss: 0.0930 - val_accuracy: 0.9742
Epoch 13/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3509 - accuracy: 0.9081 - val_loss: 0.0798 - val_accuracy: 0.9800
Epoch 14/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3621 - accuracy: 0.9056 - val_loss: 0.0711 - val_accuracy: 0.9813
Epoch 15/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3773 - accuracy: 0.9064 - val_loss: 0.0919 - val_accuracy: 0.9742
Epoch 16/20
981/981 [==============================] - 21s 22ms/step - loss: 0.3576 - accuracy: 0.9070 - val_loss: 0.0744 - val_accuracy: 0.9777
Epoch 17/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3570 - accuracy: 0.9090 - val_loss: 0.0845 - val_accuracy: 0.9751
Epoch 18/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3834 - accuracy: 0.9063 - val_loss: 0.0772 - val_accuracy: 0.9799
Epoch 19/20
981/981 [==============================] - 20s 21ms/step - loss: 0.3419 - accuracy: 0.9139 - val_loss: 0.0644 - val_accuracy: 0.9847
Epoch 20/20
981/981 [==============================] - 21s 21ms/step - loss: 0.3872 - accuracy: 0.9073 - val_loss: 0.1523 - val_accuracy: 0.9607
```

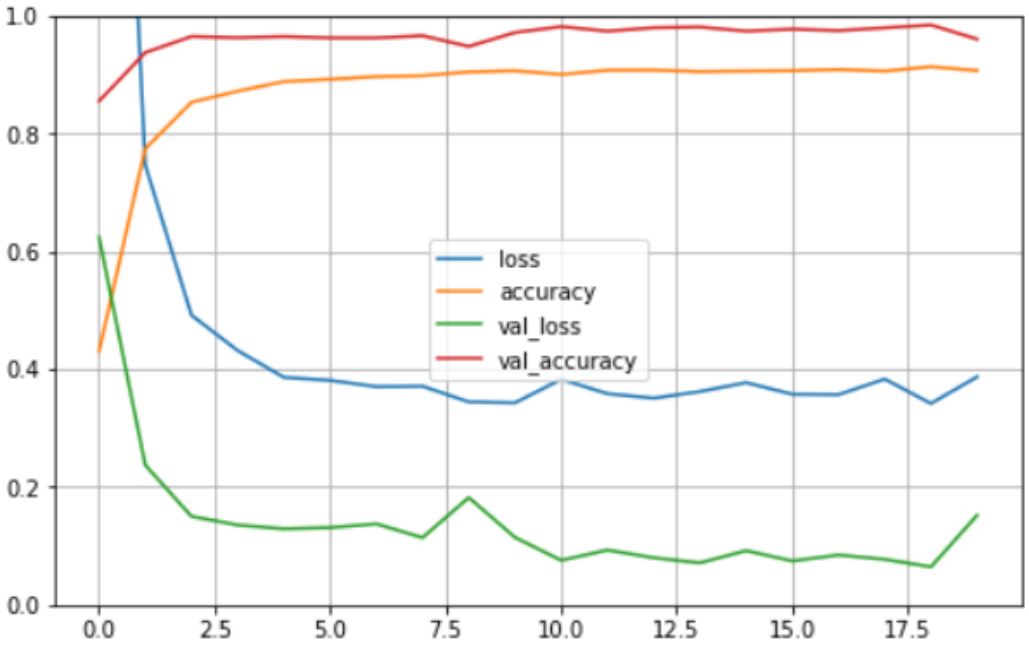
```
#plotting graphs for accuracy
plt.figure(0)
plt.plot(history2.history['accuracy'], label='training accuracy')
plt.plot(history2.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



```
#plotting graphs for loss
plt.figure(1)
plt.plot(history2.history['loss'], label='training loss')
plt.plot(history2.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



```
pd.DataFrame(history2.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



3. CNN with 4 Convolutional Layer

```
from tensorflow.keras.models import Sequential, load_model
from keras.optimizers import adam_v2

input_shape = X_train.shape[1:]

#Building the model
def CNN4():
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=input_shape))
    model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(rate=0.5))
    model.add(Dense(43, activation='softmax'))

    warnings.filterwarnings("ignore", category=DeprecationWarning)
    model.compile(adam_v2.Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

```
cnn4.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        2432

 conv2d_5 (Conv2D)           (None, 22, 22, 32)        25632

 max_pooling2d_3 (MaxPooling  (None, 11, 11, 32)       0
 2D)

 dropout_5 (Dropout)         (None, 11, 11, 32)        0

 conv2d_6 (Conv2D)           (None, 9, 9, 64)          18496

 conv2d_7 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_4 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_6 (Dropout)         (None, 3, 3, 64)          0

 flatten_2 (Flatten)         (None, 576)               0

 dense_4 (Dense)             (None, 256)               147712

 dropout_7 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 43)                11051

=================================================================
Total params: 242,251
Trainable params: 242,251
Non-trainable params: 0
_________________________________________________________________
```

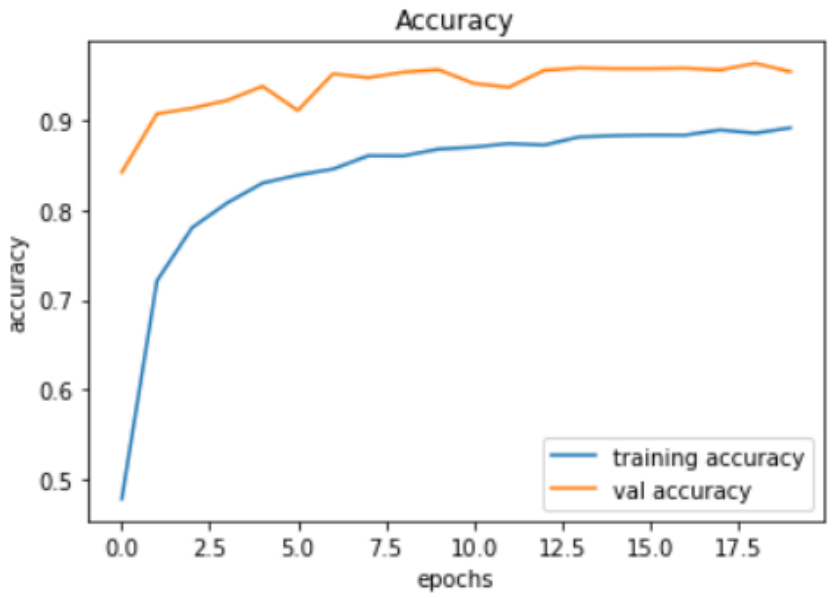
```
cnn4.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        2432

 conv2d_5 (Conv2D)           (None, 22, 22, 32)        25632

 max_pooling2d_3 (MaxPooling  (None, 11, 11, 32)       0
 2D)

 dropout_5 (Dropout)         (None, 11, 11, 32)        0

 conv2d_6 (Conv2D)           (None, 9, 9, 64)          18496

 conv2d_7 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_4 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_6 (Dropout)         (None, 3, 3, 64)          0

 flatten_2 (Flatten)         (None, 576)               0

 dense_4 (Dense)             (None, 256)               147712

 dropout_7 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 43)                11051

=================================================================
Total params: 242,251
Trainable params: 242,251
Non-trainable params: 0
_________________________________________________________________
```

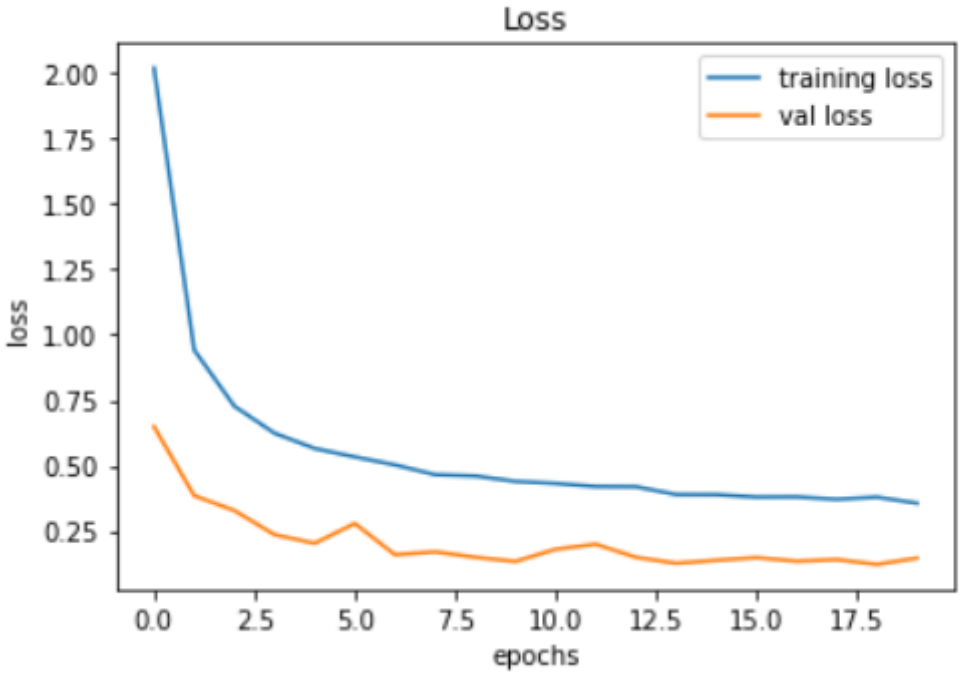
```
#Final trainig of model
history3 = cnn4.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_val, y_val))

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Epoch 1/20
981/981 [==============================] - 47s 47ms/step - loss: 2.0197 - accuracy: 0.4780 - val_loss: 0.6485 - val_accuracy: 0.8421
Epoch 2/20
981/981 [==============================] - 46s 47ms/step - loss: 0.9420 - accuracy: 0.7212 - val_loss: 0.3850 - val_accuracy: 0.9073
Epoch 3/20
981/981 [==============================] - 51s 52ms/step - loss: 0.7271 - accuracy: 0.7804 - val_loss: 0.3280 - val_accuracy: 0.9135
Epoch 4/20
981/981 [==============================] - 48s 49ms/step - loss: 0.6238 - accuracy: 0.8081 - val_loss: 0.2358 - val_accuracy: 0.9223
Epoch 5/20
981/981 [==============================] - 48s 49ms/step - loss: 0.5655 - accuracy: 0.8300 - val_loss: 0.2027 - val_accuracy: 0.9380
Epoch 6/20
981/981 [==============================] - 47s 48ms/step - loss: 0.5331 - accuracy: 0.8390 - val_loss: 0.2786 - val_accuracy: 0.9110
Epoch 7/20
981/981 [==============================] - 47s 48ms/step - loss: 0.5026 - accuracy: 0.8457 - val_loss: 0.1591 - val_accuracy: 0.9519
Epoch 8/20
981/981 [==============================] - 47s 48ms/step - loss: 0.4651 - accuracy: 0.8605 - val_loss: 0.1701 - val_accuracy: 0.9476
Epoch 9/20
981/981 [==============================] - 49s 50ms/step - loss: 0.4602 - accuracy: 0.8604 - val_loss: 0.1492 - val_accuracy: 0.9537
Epoch 10/20
981/981 [==============================] - 51s 52ms/step - loss: 0.4396 - accuracy: 0.8680 - val_loss: 0.1334 - val_accuracy: 0.9565
Epoch 11/20
981/981 [==============================] - 50s 51ms/step - loss: 0.4317 - accuracy: 0.8702 - val_loss: 0.1801 - val_accuracy: 0.9412
Epoch 12/20
981/981 [==============================] - 46s 47ms/step - loss: 0.4199 - accuracy: 0.8741 - val_loss: 0.1992 - val_accuracy: 0.9369
Epoch 13/20
981/981 [==============================] - 48s 49ms/step - loss: 0.4190 - accuracy: 0.8724 - val_loss: 0.1492 - val_accuracy: 0.9558
Epoch 14/20
981/981 [==============================] - 46s 47ms/step - loss: 0.3894 - accuracy: 0.8815 - val_loss: 0.1264 - val_accuracy: 0.9584
Epoch 15/20
981/981 [==============================] - 50s 51ms/step - loss: 0.3888 - accuracy: 0.8829 - val_loss: 0.1378 - val_accuracy: 0.9577
Epoch 16/20
981/981 [==============================] - 51s 52ms/step - loss: 0.3795 - accuracy: 0.8837 - val_loss: 0.1480 - val_accuracy: 0.9577
Epoch 17/20
981/981 [==============================] - 50s 51ms/step - loss: 0.3801 - accuracy: 0.8835 - val_loss: 0.1344 - val_accuracy: 0.9582
Epoch 18/20
981/981 [==============================] - 50s 51ms/step - loss: 0.3711 - accuracy: 0.8893 - val_loss: 0.1406 - val_accuracy: 0.9561
Epoch 19/20
981/981 [==============================] - 47s 48ms/step - loss: 0.3793 - accuracy: 0.8858 - val_loss: 0.1223 - val_accuracy: 0.9637
Epoch 20/20
981/981 [==============================] - 51s 52ms/step - loss: 0.3559 - accuracy: 0.8917 - val_loss: 0.1461 - val_accuracy: 0.9542
```

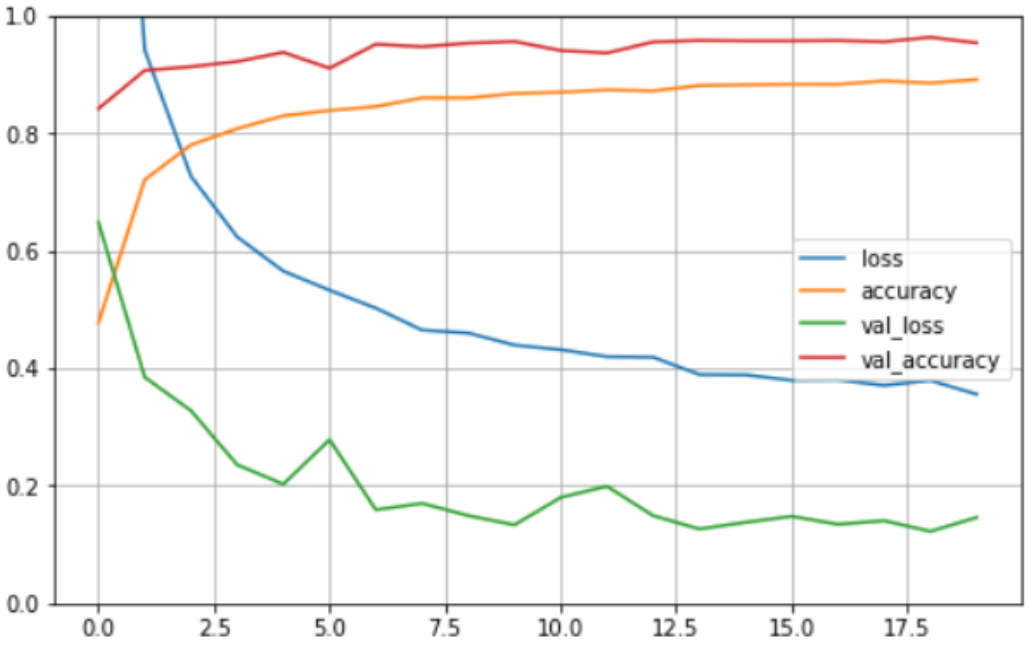
```
#plotting graphs for accuracy
plt.figure(0)
plt.plot(history3.history['accuracy'], label='training accuracy')
plt.plot(history3.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



```
#plotting graphs for loss
plt.figure(1)
plt.plot(history3.history['loss'], label='training loss')
plt.plot(history3.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



```
pd.DataFrame(history3.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



```
score = cnn4.evaluate(X_test, y_test, verbose=0)

print('Test Score:', score[0])
print('Test Accuracy:', score[1])
```

```
Test Score: 0.14613008499145508
Test Accuracy: 0.9542208909988403
```

Proposed 4 Dimentional CNN

```
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.models import Sequential, load_model
from keras.optimizers import adam_v2

input_shape = X_train.shape[1:]

def CNN4_model():

    cnn4 = Sequential()
    cnn4.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    cnn4.add(BatchNormalization())

    cnn4.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
    cnn4.add(BatchNormalization())
    cnn4.add(MaxPooling2D(pool_size=(2, 2)))
    cnn4.add(Dropout(0.25))

    cnn4.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    cnn4.add(BatchNormalization())
    cnn4.add(Dropout(0.25))

    cnn4.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    cnn4.add(BatchNormalization())
    cnn4.add(MaxPooling2D(pool_size=(2, 2)))
    cnn4.add(Dropout(0.25))

    cnn4.add(Flatten())

    cnn4.add(Dense(512, activation='relu'))
    cnn4.add(BatchNormalization())
    cnn4.add(Dropout(0.5))

    cnn4.add(Dense(256, activation='relu'))
    cnn4.add(BatchNormalization())
    cnn4.add(Dropout(0.5))

    cnn4.add(Dense(43, activation='softmax'))

    cnn4.compile(adam_v2.Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

    return cnn4
```

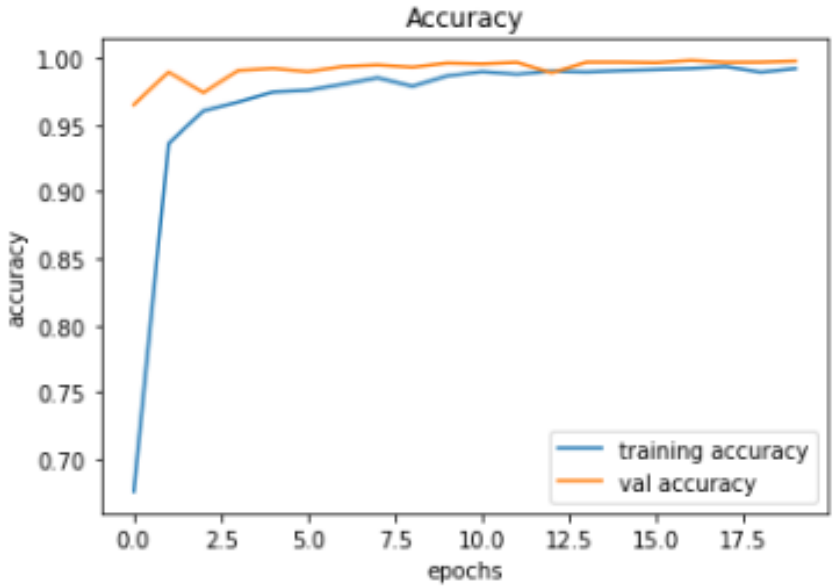
```
cnn4_1.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 32)        896

 batch_normalization (BatchN  (None, 28, 28, 32)       128
 ormalization)

 conv2d_9 (Conv2D)           (None, 26, 26, 32)        9248

 batch_normalization_1 (Batc  (None, 26, 26, 32)       128
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 dropout_8 (Dropout)         (None, 13, 13, 32)        0

 conv2d_10 (Conv2D)          (None, 11, 11, 64)        18496

 batch_normalization_2 (Batc  (None, 11, 11, 64)       256
 hNormalization)

 dropout_9 (Dropout)         (None, 11, 11, 64)        0

 conv2d_11 (Conv2D)          (None, 9, 9, 128)         73856

 batch_normalization_3 (Batc  (None, 9, 9, 128)        512
 hNormalization)

 max_pooling2d_6 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_10 (Dropout)        (None, 4, 4, 128)         0

 flatten_3 (Flatten)         (None, 2048)              0

 dense_6 (Dense)             (None, 512)               1049088

 batch_normalization_4 (Batc  (None, 512)              2048
 hNormalization)

 dropout_11 (Dropout)        (None, 512)               0

 dense_7 (Dense)             (None, 256)               131328

 batch_normalization_5 (Batc  (None, 256)              1024
 hNormalization)

 dropout_12 (Dropout)        (None, 256)               0

 dense_8 (Dense)             (None, 43)                11051

=================================================================
Total params: 1,298,059
Trainable params: 1,296,011
Non-trainable params: 2,048
_________________________________________________________________
```

```
#Final trainig of model
history4 = cnn4_1.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_val, y_val))

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
Epoch 1/20
981/981 [==============================] - 68s 68ms/step - loss: 1.1966 - accuracy: 0.6757 - val_loss: 0.1205 - val_accuracy: 0.9652
Epoch 2/20
981/981 [==============================] - 65s 66ms/step - loss: 0.2140 - accuracy: 0.9361 - val_loss: 0.0449 - val_accuracy: 0.9897
Epoch 3/20
981/981 [==============================] - 66s 68ms/step - loss: 0.1306 - accuracy: 0.9607 - val_loss: 0.1583 - val_accuracy: 0.9741
Epoch 4/20
981/981 [==============================] - 64s 65ms/step - loss: 0.1095 - accuracy: 0.9673 - val_loss: 0.0330 - val_accuracy: 0.9909
Epoch 5/20
981/981 [==============================] - 66s 67ms/step - loss: 0.0868 - accuracy: 0.9746 - val_loss: 0.0290 - val_accuracy: 0.9925
Epoch 6/20
981/981 [==============================] - 61s 63ms/step - loss: 0.0770 - accuracy: 0.9759 - val_loss: 0.0335 - val_accuracy: 0.9901
Epoch 7/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0664 - accuracy: 0.9804 - val_loss: 0.0208 - val_accuracy: 0.9939
Epoch 8/20
981/981 [==============================] - 64s 66ms/step - loss: 0.0495 - accuracy: 0.9853 - val_loss: 0.0177 - val_accuracy: 0.9950
Epoch 9/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0667 - accuracy: 0.9789 - val_loss: 0.0226 - val_accuracy: 0.9934
Epoch 10/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0437 - accuracy: 0.9868 - val_loss: 0.0142 - val_accuracy: 0.9964
Epoch 11/20
981/981 [==============================] - 61s 62ms/step - loss: 0.0311 - accuracy: 0.9900 - val_loss: 0.0153 - val_accuracy: 0.9958
Epoch 12/20
981/981 [==============================] - 62s 64ms/step - loss: 0.0377 - accuracy: 0.9881 - val_loss: 0.0113 - val_accuracy: 0.9969
Epoch 13/20
981/981 [==============================] - 60s 62ms/step - loss: 0.0288 - accuracy: 0.9904 - val_loss: 0.0415 - val_accuracy: 0.9892
Epoch 14/20
981/981 [==============================] - 61s 62ms/step - loss: 0.0324 - accuracy: 0.9898 - val_loss: 0.0133 - val_accuracy: 0.9971
Epoch 15/20
981/981 [==============================] - 61s 62ms/step - loss: 0.0296 - accuracy: 0.9909 - val_loss: 0.0130 - val_accuracy: 0.9971
Epoch 16/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0267 - accuracy: 0.9916 - val_loss: 0.0123 - val_accuracy: 0.9967
Epoch 17/20
981/981 [==============================] - 63s 64ms/step - loss: 0.0245 - accuracy: 0.9925 - val_loss: 0.0077 - val_accuracy: 0.9983
Epoch 18/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0192 - accuracy: 0.9939 - val_loss: 0.0118 - val_accuracy: 0.9969
Epoch 19/20
981/981 [==============================] - 61s 62ms/step - loss: 0.0374 - accuracy: 0.9897 - val_loss: 0.0097 - val_accuracy: 0.9971
Epoch 20/20
981/981 [==============================] - 62s 63ms/step - loss: 0.0259 - accuracy: 0.9924 - val_loss: 0.0098 - val_accuracy: 0.9978
```

```
#plotting graphs for accuracy
plt.figure(0)
plt.plot(history4.history['accuracy'], label='training accuracy')
plt.plot(history4.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```




```
score = cnn4_1.evaluate(X_test, y_test, verbose=0)

print('Test Score:', score[0])
print('Test Accuracy:', score[1])
```

```
Test Score: 0.009770429693162441
Test Accuracy: 0.997832179069519
```

Chapter 5

Results and Discussions

In this research it was a study on variety of CNN-based classification modes to evaluate on the German dataset. We were able to build the final model of CNN using Keras framework. We initially find out what the dataset contains of, did data augmentation to it and did a fit study on the models.

Here we will be using different metrics like Test Score, Test Accuracy, Validation Accuracy, Test Loss and Validation Loss.

Table 5.1: Metric

Metrics				
Model	**loss**	**accuracy**	**val_loss**	**val_loss**
CNN1 (1 Layer)	0.1708	0.9803	0.5722	0.9675
CNN2 (3 Layer)	0.3872	0.9073	0.1523	0.9607
CNN3 (4 Layer)	0.3559	0.8917	0.1461	0.9542
CNN4 (4 Layer)	0.0259	0.9924	0.0098	0.9978

In our first model we have given a 1 layer architecture of CNN and got a result of 98% accuracy which is surprisingly well for a single layer Conv2D

Table 5.2: Test Score and Accuracy

Test Score and Accuracy		
Model	**Test Score**	**Test Accuracy**
CNN1 (1 Layer)	0.5722	0.9674
CNN2 (3 Layer)	0.1522	0.9607
CNN3 (4 Layer)	0.1461	0.9542
CNN4 (4 Layer)	0.0097	0.9978

In our first model we have given a 1 layer architecture of CNN and got a result of 98% accuracy.

5.0.1 Visualizing What My Models Learns

It's often said that deep-learning models are "black boxes": learning representations that are difficult to extract and present in a human-readable form. Although this is partially true for certain types of deep-learning models, it's definitely not true for convnets. The representations learned by convnets are highly amenable to visualization, in large part because they're representations of visual concepts.

Here we attempt to visualize the intermediate CNN outputs (intermediate activations). Visualizing intermediate activations consists of displaying the feature maps that are output by various convolution and pooling layers in a network, given a certain input (the output of a layer is often called its activation, the output of the activation function). This gives a view into how an input is decomposed into the different filters learned by the network.

We want to visualize feature maps with three dimensions: width, height, and depth (channels). Each channel encodes relatively independent features, so the proper way to visualize these feature maps is by independently plotting the contents of every channel as a 2D image.

Lets see on how it will look on few of the images from the train set and we shall reshape it and use viridis color map to view it.

```
test_im1 = X_train[100]
plt.imshow(test_im1.reshape(30,30,3), cmap='viridis', interpolation='none')
plt.show()
```

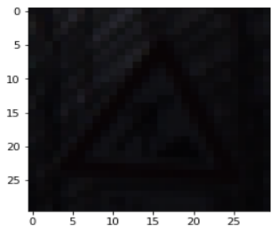


Figure 5.1: An image after data augmentation from X_train set

```
from keras import models
# extracts the outputs of the top 8 layers
layer_outputs = [layer.output for layer in cnn4_1.layers[:8]]

# creates a model that will return these outputs, given the model input
activation_model = models.Model(cnn4_1.input, layer_outputs)

# returns a list of Numpy arrays: one array per layer activation
activations = activation_model.predict(test_im1.reshape(1,30,30,3))

# activation of the 1st convolution layer
first_layer_activation = activations[0]

# display the 4th channel of the activation of the 1st layer of the original model
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

```
<matplotlib.image.AxesImage at 0x220cf527fd0>
```

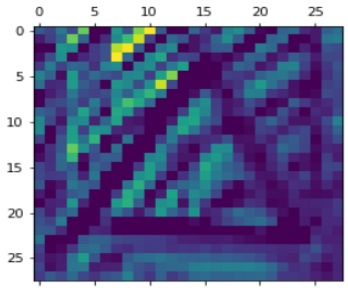


Figure 5.2: Using CNN model on the image

```
test_im1 = X_train[500]
plt.imshow(test_im1.reshape(30,30,3), cmap='viridis', interpolation='none')
plt.show()
```

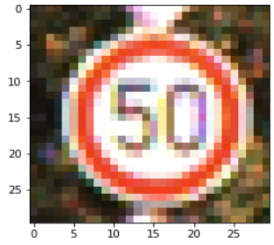


Figure 5.3: An image after data augmentation from X_train set

```
activations = activation_model.predict(test_im1.reshape(1,30,30,3))
first_layer_activation = activations[0]
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

```
<matplotlib.image.AxesImage at 0x221b2f6b670>
```

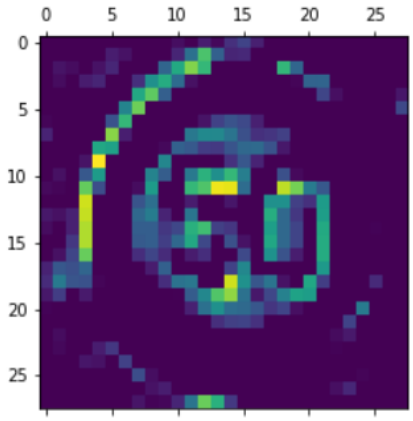


Figure 5.4: Using CNN model on the image

The feature set our first CNN model which has a single layer convolutional will be seeing the models like the below picture.

1 Layer convolutional

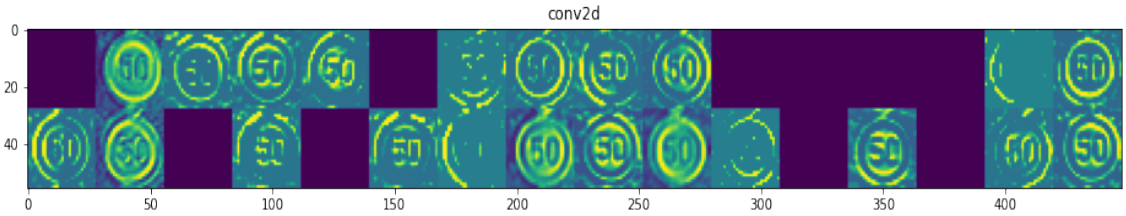


Figure 5.5: 1st Conv2D CNN 1 model

3 layer convolutional

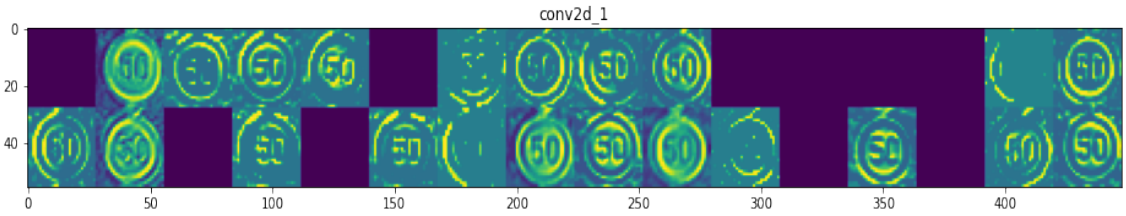


Figure 5.6: 1st Conv2D CNN 2nd model

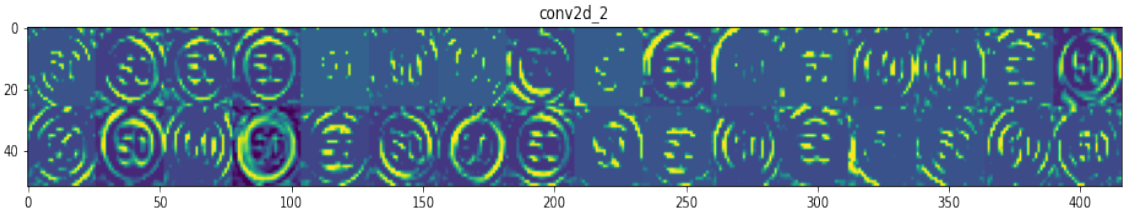


Figure 5.7: 2nd Conv2D 2nd model

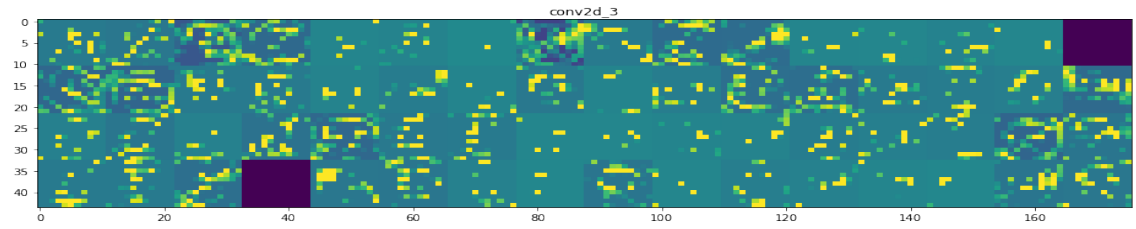


Figure 5.8: 3rd layer Conv2D 2nd model

4 layer convolutional in its feature set

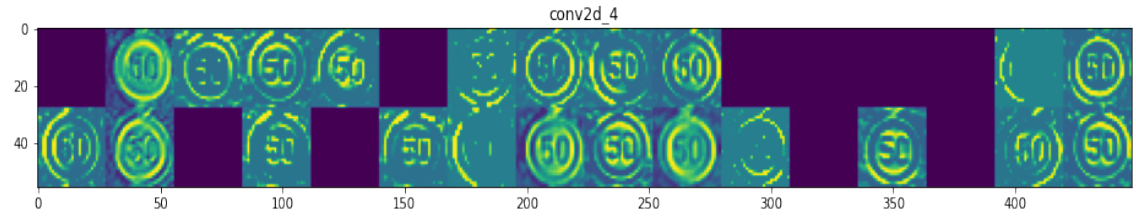


Figure 5.9: 1st Conv2D 3rd model

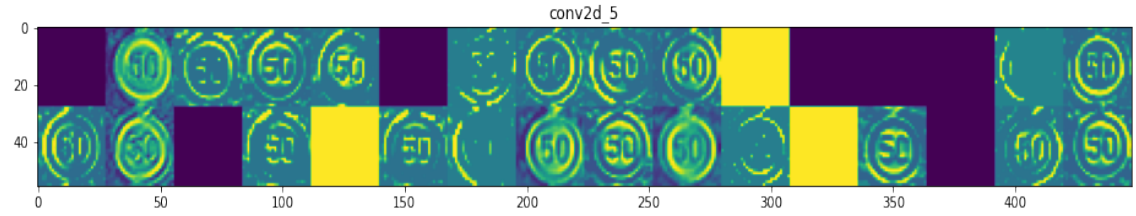


Figure 5.10: 2nd Conv2D 3rd model

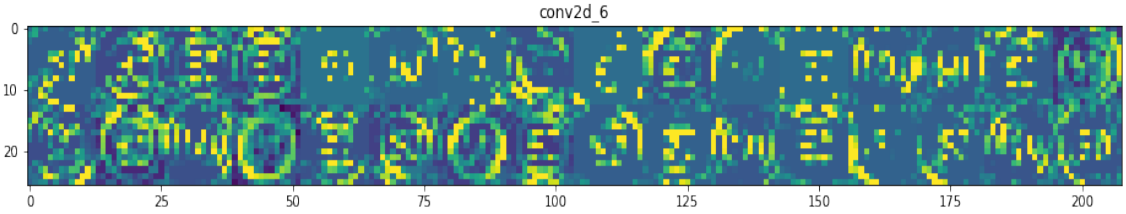


Figure 5.11: 3rd layer Conv2D 3rd model

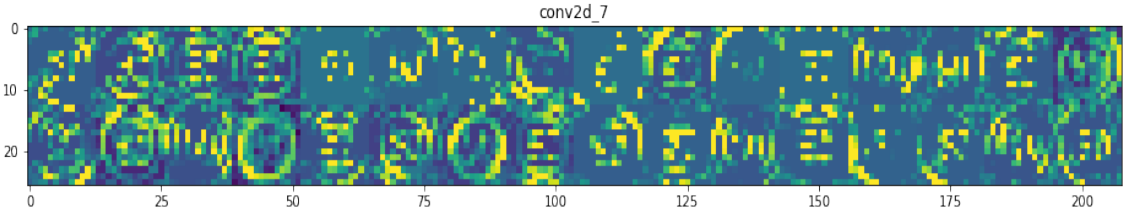


Figure 5.12: 4th layer Conv2D 3rd model

Our proposed method of 4 layer con2d in its feature set

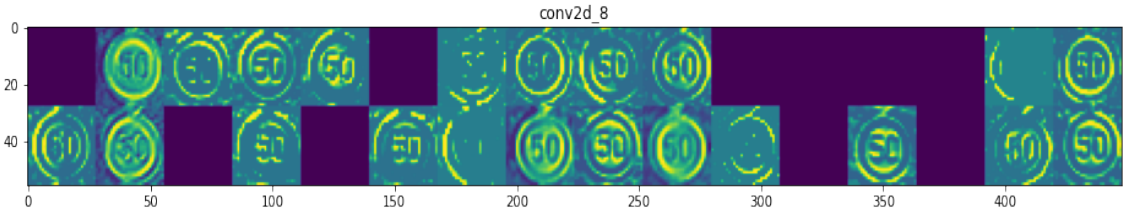


Figure 5.13: 1st Conv2D 4th model

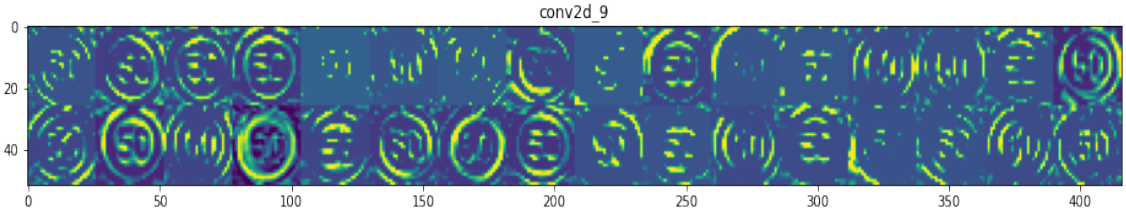


Figure 5.14: 2nd Conv2D 4th model

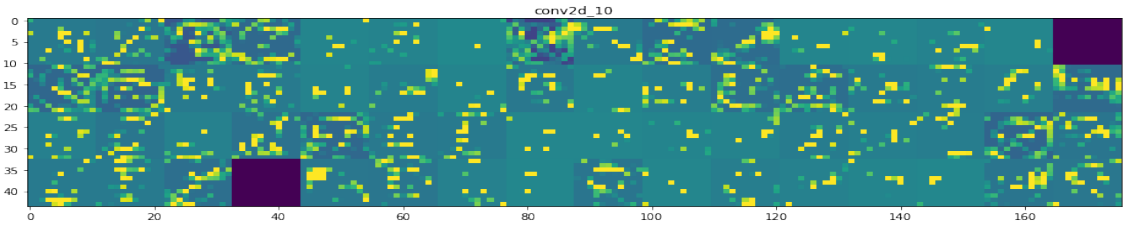


Figure 5.15: 3rd layer Conv2D 4th model

Chapter 6

Conclusion and Future Work

6.0.1 Conclusion

A major and main development that can be done is real time analysis of traffic symbols with network algorithms. A parallel algorithm can be introduced to speed up the processing time of fixed and trained layers. Schemes can be fixed to run combination of parallel algorithms for faster and accurate prediction. Further development can be done with text interpretation from traffic sign boards, to recognize text and translate the meaning of traffic sign. Additional database can be populated pictures of traffic symbols of varied sizes , pictures captured under different weathers like foggy or rainy, for better accuracy.During the project period, we tried to implement four types of CNN models and, after some research, noticed VGG-19 has a better image recognition pattern, but were not able to complete it in time.

6.0.2 Future Work

Therefore in future developments, we have been planning to add Video Capture which would take real-time analysis of the traffic and predict with a high accuracy and high fps. In real time data high fps predictive analysis is necessary. In few recent papers, FR-CNN and VGG-19 were showing improved accuracy from other papers which we have went through so far. Therefore, we will be adding FR-CNN and VGG-19 models to compare and analyse if they these models are better than our improved CNN model which has 99% accuracy.

Bibliography

[1] Prashengit Dhar, Md Zainal Abedin, Tonoy Biswas, and Anish Datta. Traffic sign detection—a new approach and recognition using convolution neural network. pages 416–419, 2017.

[2] Jack Greenhalgh and Majid Mirmehdi. Real-time detection and recognition of road traffic signs. *IEEE transactions on intelligent transportation systems*, 13(4):1498–1506, 2012.

[3] Yuga Hatolkar, Poorva Agarwal, and Seema Patil. A survey on road traffic sign recognition system using convolution neural network. *International Journal of Current Engineering and Technology*, 8(1):104–108, 2018.

[4] M Sajjad Hossain, M Mahmudul Hasan, M Ameer Ali, Md Humayun Kabir, and ABM Shawkat Ali. Automatic detection and recognition of traffic signs. pages 286–291, 2010.

[5] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. pages 1–8, 2013.

[6] A Khodayari, A Ghaffari, and F Fanni. A real time traffic sign detection and recognition algorithm based on super fuzzy set. *Automotive Science and Engineering*, 7(1):2346–2355, 2017.

[7] Rabia Malik, Javaid Khurshid, and Sana Nazir Ahmad. Road sign detection and recognition using colour segmentation, shape analysis and template matching. In *2007 international conference on machine learning and cybernetics*, volume 6, pages 3556–3560. IEEE, 2007.

[8] Dip Nandi, AFM Saifuddin Saif, Paul Prottoy, Kazi Md Zubair, and Seemanta Ahmed Shubho. Traffic sign detection based on color segmentation

of obscure image candidates: a comprehensive study. *International Journal of Modern Education and Computer Science*, 11(6):35, 2018.

[9] Alexander Shustanov and Pavel Yakimov. Cnn design for real-time traffic sign recognition. *Procedia Engineering*, 201:718–725, 2017. 3rd International Conference "Information Technology and Nanotechnology", ITNT-2017, 25-27 April 2017, Samara, Russia.

[10] Aashrith Vennelakanti, Smriti Shreya, Resmi Rajendran, Debasis Sarkar, Deepak Muddegowda, and Phanish Hanagal. Traffic sign detection and recognition using a cnn ensemble. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4, 2019.

[11] Tahmina Zebin, Patricia J Scully, Niels Peek, Alexander J Casson, and Krikor B Ozanyan. Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition. *IEEE Access*, 7:133509–133520, 2019.

[12] Jianming Zhang, Zhipeng Xie, Juan Sun, Xin Zou, and Jin Wang. A cascaded r-cnn with multiscale attention and imbalanced samples for traffic sign detection. *IEEE Access*, 8:29742–29754, 2020.

[13] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

School of Computer Science & Engineering
VIT University
Vandalur - Kelambakkam Road, Chennai - 600 127
(http://www.vit.ac.in)