

# **TRAFFIC SIGN RECOGNITION USING DEEPLARNING BY NEURAL NETWORKS**

**A MAJOR PROJECT REPORT**

*Submitted by*

**V. KRANTHI KUMAR REDDY  
K. THIRUPATHI REDDY  
R. PRANAV KUMAR RAJU**

**[REGNO:RA1711003020467]  
[REGNO:RA1711003020456]  
[REGNO: RA1711003020473]**

*Under the guidance of*

**MS.V. Surya, Assistant Professor**

(Designation, Department of Computer Science and Engineering)

*in fulfillment for the award of the degree*

**BACHELOR OF TECHNOLOGY**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

*of*

**FACULTY OF ENGINEERING AND TECHNOLOGY**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**RAMAPURAM CAMPUS, CHENNAI -600089**

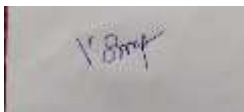
**MAY 2021**

# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Deemed to be University U/S 3 of UGC Act, 1956)**

## **BONAFIDE CERTIFICATE**

Certified that this project report titled **“TRAFFIC SIGN RECOGNITION USING DEEP LEARNING BY NEURAL NETWORKS”** is the bonafide work of **V. KRANTHI [RA1711003020467], THIRUPATHI REDDY [RA1711003020456] and PRANAV KUMAR RAJU [RA1711003020473]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an occasion on this or any other candidate.



**SIGNATURE**

**MS.V. Surya**

**Assistant Professor,**

Computer Science and Engineering,

SRM Institute of Science and Technology,

Ramapuram Campus, Chennai.

**SIGNATURE**

**Dr. K. RAJA, M.E., Ph.D.,**

**Professor and Head**

Computer Science and Engineering,

SRM Institute of Science and Technology,

Ramapuram Campus, Chennai.

Submitted for the project viva-voce held on **31.05.2021** at SRM Institute of Science and Technology, Ramapuram Campus, Chennai -600089.

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

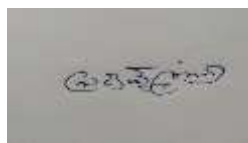
**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**  
**RAMAPURAM, CHENNAI - 89**

**DECLARATION**

We hereby declare that the entire work contained in this project report titled **“TRAFFIC SIGN RECOGNITION USING DEEP LEARNING BY NEURAL NETWORKS”** has been carried out by **V. KRANTHI [RA1711003020467], THIRUPATHI REDDY [RA1711003020456] and PRANAV KUMAR RAJU [RA1711003020473]** at SRM Institute of Science and Technology, Ramapuram Campus, Chennai-600089, under the guidance **MS.V. Surya, Assistant Professor**, Department of Computer Science and Engineering.



**V. KRANTHI REDDY**



**R. PRANAV RAJU**



**K. TIRUPATI REDDY**

**Place: Chennai**

**Date: 31.05.2021**



**Own Work Declaration**  
Department of Computer Science and Engineering  
**SRM Institute of Science & Technology**

**Own Work\* Declaration Form**

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

**Degree/ Course** : **B. Tech / CSE**  
**Student Name** : **K. Thirupathi, V. Kranthi, R. Pranav**  
**Registration Number** : **RA1711003020456, RA1711003020467, RA1711003020473**  
**Title of Work** : **TRAFFIC SIGN RECOGNITION USING DEEP LEARNING BY NEURAL NETWORKS**

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism\*\*, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

**DECLARATION:**

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

V. KRANTHI

R. PRANAV

K. THIRUPATHI

## **ABSTRACT**

Now a days with rapid development of society and economy, automobiles have become one of the most common modes of transportation for every individual. Traffic road sign detection is very crucial in transport system. The aim of the project is to detect the type of the traffic signal and act according to that. The entire action was recorded in a video sequence by an on-board camera attached to the vehicle. The main theme of the project is for the application of driverless cars and also traffic recognition is one of the most challenging problem for driving assistance system. The project also describes how deep learning is applied i.e.; the projectbased learning approach. The project uses neural networks. Convolution neural network algorithm is a multilayer perceptron that is the special design for identification of image information. Generally, has more layers: input layer, convolution layer, hidden layer and output layer. In addition, the deep network architecture has more number of convolutional layer. The algorithm is proposed to train traffic sign training sets to get a model that can classify traffic signs and to learn and identify the most critical traffic sign features, so as to achieve the purpose of identifying traffic signs in the real scenario. The project focuses for an intelligent driving system to be enabled and to ensure driver safety.

## ACKNOWLEDGEMENT

We place on record our deep sense of gratitude to our lionized Chairman **Dr. R. SHIVA KUMAR** for providing us with the requisite infrastructure throughout the course.

We take the opportunity to extend our hearty and sincere thanks to our Dean, **Dr.M. MURALI KRISHNA, B.E., M. Tech., Ph.D. MISTE, FIE, C.Engg**, for maneuvering us into accomplishing the project.

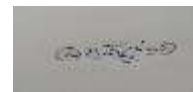
We take the privilege to extend our hearty and sincere ggratitude to the Professor and Head of the Department, **Dr.K. RAJA, M.E., PhD.**, for his suggestions, support and encouragement towards the completion of the project with perfection.

We express our hearty and sincere thanks to our guide **MS.V. Surya, Assistant Professor**, Computer Science and Engineering Department for her encouragement, consecutive criticism and constant guidance throughout this project work.

Our thanks to the teaching and non-teaching staff of the Computer Science and Engineering Department of SRM Institute of Science and Technology, Ramapuram Campus, for providing necessary resources for our project.



**V.KRANTHI KUMAR REDDY**



**R. PRANAV KUAMR RAJU**



**THIRUPATHI REDDY**

## TABLE OF CONTENTS

	TITLE		Page No.
<b>1</b>	<b>Introduction</b>		<b>11</b>
<b>2</b>	<b>Literature Survey</b>		<b>14</b>
	2.1	Traffic Sign Detection Algorithm Based on Deep Convolutional Neural Network.	<b>14</b>
	2.2	Traffic Sign Detection and Pattern Recognition using Support Vector Machine.	<b>15</b>
	2.3	Detection and Recognition of Traffic Signs based on RGB to Red Conversion.	<b>16</b>
	2.4	Simultaneous Traffic-Sign Detection and Real time communication using Dual Camera in ITS.	<b>17</b>
	2.5	Real-time Large-Scale Traffic Detection.	<b>18</b>
	2.6	Vision-Based Traffic Sign Detection and Recognition Systems.	<b>19</b>
<b>3</b>	<b>System Specification</b>		<b>20</b>
	3.1	Hard ware Specification	<b>20</b>
	3.2	Software Specification	<b>21</b>
<b>4</b>	<b>Proposed Methodology</b>		<b>23</b>
	4.1	Prepare a Dataset of images	<b>27</b>
	4.2	Image Pre processing	<b>29</b>
	4.3	Network Structure	<b>36</b>

	4.3.1	Convolution	<b>39</b>
	4.3.2	Normalization	<b>41</b>

	4.3.3	Pooling layer	<b>43</b>
	4.3.4	Output Layer	<b>46</b>
	4.4	Testing and Recognition	<b>48</b>
	4.5	Experimental Analysis	<b>38</b>
	4.6	Algorithm	<b>49</b>
	4.7	Results	<b>51</b>
<b>5</b>	<b>Conclusion</b>		<b>53</b>
	<b>References</b>		<b>54</b>
	<b>Appendix</b>		<b>56</b>



## LIST OF TABLES

<b>Figure no.</b>	<b>Title of the figure</b>	<b>Pg. No.</b>
4.4(a)	DATASET CONFIGURATION FOR DESIRED CLASSES	48
4.5(a)	COMPARISON OF EXISTING SYSTEM VS PROPOSED SYSTEM	49
4.6(a)	ALGORITHM OF SINGLE NEURAL NETWORK	51

## LIST OF FIGURES

<b>Figure no.</b>	<b>Title of the figure</b>	<b>Pg. No.</b>
4(a)	PROPOSED ARCHITECTURE MODEL	25
4(b)	BIAS IN NEURAL NETWORKS	26
4.1(a)	GTSRB DATASET OF TRAFFIC SIGN IMAGES	27
4.1(b)	ROI FORMAT OF IMAGE	29
4.2(a)	IMAGE PREPROCESSING BLOCK	31
4.2(b)	RGB IMAGE	34
4.3(c)	GREYSCALED IMAGE	34
4.2(d)	MATRIX FORM	36
4.3(a)	NETWORK STRUCTURE	37
4.3(b)	4*4*3 RGB IMAGE	39
4.3(c)	CONVOLUTIONING A 5*5*1 IMAGE WITH 3*3*1 KENNEL TO GET 3*3*1 CONVOLVED FEATURE	40
4.3(d)	CONVOLUTION OPERATION ON A M*N*3 IMAGE MATRIK WITH A 3*3*3 KENNEL	41
4.3(e)	RELU V/S LOGISTIC SIGMOID	43
4.3(f)	DOWNSAMPLING	46
4.3(g)	SOFTMAX REGRESSION IN OUTPUT LAYER	47

4.4(a)	TESTING OF THE TRAINED MODEL	48
--------	------------------------------	----

4.7	PROBABILITY OF PREDICTION RATE	52
-----	--------------------------------	----

# **CHAPTER 1**

## **INTRODUCTION**

Traffic-sign recognition (TSR) is a technology by which a vehicle is able to recognize the traffic signs put on the road e.g., "speed limit" or "children" or "turn ahead". This is part of the feature collectively called as Advanced Driving Assistance System (ADAS). The technology is being developed by a variety of automotive suppliers. It uses image processing techniques to detect the traffic signs. The detection methods can be generally divided into color based, shape based and learning based methods.

In general, Traffic signs are the silent speakers on the road. Be it the person behind the wheel or a pedestrian, having a sound knowledge about road safety is absolutely necessary for all before hitting the roads. Traffic signs give information about the road conditions ahead, provide instructions to be followed at the major crossroads or junctions, warn or guide drivers, and ensure proper functioning of road traffic. Being unaware of road signs is akin to throwing caution to the wind. It can lead to loss of life and property.

A person is supposed to be familiar (get through a written or oral test) with the traffic signs and symbols before acquiring a driving license in India. Road Safety Signs are Primarily of Three Types:1. Mandatory Signs: These traffic signs are used to ensure free movement of traffic and make the road users convenient of certain laws and regulations, restrictions and prohibitions. Violation of these road safety signs is an offense, as per law.2. Cautionary Signs: These traffic signs make the road users conscious of hazardous conditions on the road beforehand. The drivers, accordingly, take necessary actions to handle the situation,3. Informatory Signs: These traffic signs guide road users about destinations, distance, alternative routes, and prominent locations like food joints, public toilets, nearby hospitals, etc.

For an Intelligent Transportation System Smart vehicle are must. Smart vehicles reduce number of accidents happening around the world. Not only that Smart vehicles resembles the driver safety and also helps in the automatic driving and passenger's safety. The camera used for traffic sign detection captures an image with 1236\*968 pixels in a series of images just like a video format. For a safer journey to happen, one must follow the road traffic signs and obey the traffic rules. As a rapid increase in rate of population, number of motor vehicles used by individuals also increasing. As a result of that traffic is

increased. For application of smart vehicles, traffic sign detection and recognition is must. Traffic signs are of several types out of them three are important and mostly used. They are regulatory signs, warning signs (used to warn the driver about upcoming hazards, lane changes/merges etc.,) Guide signs (used to guide the driver regarding a distance such as distance signs and mile markers). Detection of traffic sign is the first step for autonomous vehicles. The image was detected by a camera. Some of the factors that involve during the detection of traffic sign was:

- 1) Fading of colors: Due to exposure of traffic sign to continuous sunlight may cause color fading which leads to correlation of input data. Also due to the pollution, causing fading of color of traffic sign which causes a problem for detection.
- 2) Weather condition: In some cases of rainy situations, detection of road signs might become difficult.

The same type of traffic sign may have different consistency in color, in the appearance. Since the camera mounted on the vehicle will not be at a 90 deg angle sight to the traffic signs. The paper comes up with a solution that works in real time recognition of traffic signs which is consistent in detection of road signs and classifies them instantly. The outlets of the paper are as follows: The system uses a RGB image as an input source for detection. A simplified filter is used to strengthen the edges of the image, also the size, the shape. It also reduces the noise in the darker or more brighter areas. Region of interest are founded abnormally by the use of maximally stable external regions algorithm.

Artificial neural network is made up of multiple hidden layers. The pre-processed image was given as input to the convolutional layer of the artificial neural network. This layer helps for the classification of traffic signs. Training of neural network algorithm takes a lot of time. Since we are using a training data set of over 5000 traffic sign images. More amount of input data makes the algorithm to provide an efficient output. Thus, the system enables for an efficient and correct prediction of traffic sign in the real time. In order to solve the concerns over road and transportation safety, automatic traffic sign detection and recognition (TSDR) system has been introduced.

An automatic TSDR system can detect and recognize traffic signs from and within images captured by cameras or imaging sensors. In adverse traffic conditions, the driver may not notice traffic signs, which may cause accidents. In such scenarios, the TSDR

system comes into action. The main objective of the research on TSDR is to improve the robustness and efficiency of the TSDR system. To develop an automatic TSDR system is a tedious job given the continuous changes in the environment and lighting conditions. Among the other issues that also need to be addressed are partial obscuring, multiple traffic signs appearing at a single time, and blurring and fading of traffic signs, which can also create problem for the detection purpose. For applying the TSDR system in real-time environment, a fast algorithm is needed. As well as dealing with these issues, a recognition system should also avoid erroneous recognition of non-signs. The aim of this research is to develop an efficient TSDR system which can detect and classify traffic signs into different classes in real-time environment.

In order to solve the concerns over road and transportation safety, automatic traffic sign detection and recognition (TSDR) system has been introduced. An automatic TSDR system can detect and recognize traffic signs from and within images captured by cameras or imaging sensors. In adverse traffic conditions, the driver may not notice traffic signs, which may cause accidents. In such scenarios, the TSDR system comes into action. The main objective of the research on TSDR is to improve the robustness and efficiency of the TSDR system. To develop an automatic TSDR system is a tedious job given the continuous changes in the environment and lighting conditions. Among the other issues that also need to be addressed are partial obscuring, multiple traffic signs appearing at a single time, and blurring and fading of traffic signs, which can also create problem for the detection purpose. For applying the TSDR system in real-time environment, a fast algorithm is needed. As well as dealing with these issues, a recognition system should also avoid erroneous recognition of non-signs.

A visual-based traffic sign recognition system can be implemented on the automobile with an aim of detecting and recognizing all emerging traffic signs. The same would be displayed to the driver with alarm-triggering features if the driver refuses to follow the traffic signs.

## **CHAPTER 2**

### **LITERATURE SURVEY**

#### **2.1 Traffic Sign Detection Algorithm Based on DeepConvolutional Neural Network.**

**Authors:** Xiong Changzhen, Wang Cong and Ma Weixin.

**Year of Publishing:** 2016

The paper used the concept of machine learning with a deep convolutional neural network algorithm using region proposal network in Faster R-CNN. Faster R-CNN has two networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects. The main different here with Fast R-CNN is that the later uses selective search to generate region proposals. The time cost of generating region proposals is much smaller in RPN than selective search, when RPN shares the most computation with the object detection network. Briefly, RPN ranks region boxes (called anchors) and proposes the ones most likely containing objects. Fast R-CNN have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck.

The experimental results show that the traffic sign detection rate of the algorithm is above 97% and the detection time is very less towards real-time in the format of video sequences. It takes only about 51.5 MS average detection time for one frame with the resolution of 640h 480 video sequence. And it is robust to various interferences. Its drawbacks are the system doesn't show how to obtain accurate ground truth and the class of the sign detected automatically. Also, the system is not responding for a multiple set of images in a single frame.

## **2.2 Traffic Sign Detection and Pattern Recognition using Support Vector Machine.**

**Authors:** Kiran C.G, V. Prabhu, Abdu Rahiman V. and Rajeev K.

**Year of Publishing:** 2009

The paper used the basis of AI with Machine Learning and Support Vector Machines algorithm. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

In other words, Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

The advantages of the system are it uses color segmentation, shape classification and pattern recognition for detection and classification of images. The system produces an output of 75.19% achievement in signal and sign recognition. The main advantage is that response time is of 64.2 MS. The system has certain drawbacks that its achievement rate is less and miss recognition of certain signs when it is left for real-time scenario. Also, the huge saturations cause delay in recognition of data and processing of data.



## **2.3 Detection and Recognition of Traffic Signs based on RGB to Red Conversion.**

**Authors:** Mr. Mohith Bhairav Mahatme, Mrs. Sonia Kuwelkar

**Year of Publishing:** 2017

Concept behind the paper is Image Processing by computer vision and Single Layer Perception algorithm are used. A single layer perceptron (SLP) is a feed-forward network based on a threshold transfer function. SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1, 0). The single layer perceptron does not have a priori knowledge, so the initial weights are assigned randomly. SLP sums all the weighted inputs and if the sum is above the threshold (some predetermined value), SLP is said to be activated (output=1).

SLP is a linear classifier and if the cases are not linearly separable the learning process will never reach a point where all the cases are classified properly. The most famous example of the inability of perceptron to solve problems with linearly non-separable cases is the XOR problem. It has advantage like the system has an accuracy of 93.25% in detection and recognition of traffic signs. Certain drawbacks are that it uses HSV color segmentation process (Hue-Saturation-Value). In case of outdoor Images, the color is sensitive to variation in lightening. Hue becomes meaningless. So, it becomes hard for the light color to find out.

## **2.4 Simultaneous Traffic-Sign Detection and Real time communication using Dual Camera in ITS.**

**Authors:** Mohd. Khalid Hasan, Md. Shahjahal, Mostafa Zaman Chowdary, Nam Tuan Le and Yeong Min Jang.

**Year of Publishing:** 2019

The system uses the concept of optimal camera communications(occ) by the help of computer vision. The fundamentals of optical camera communication (OCC), with the OCC system employing the pervasive image sensor assembled in consumer electrons as the receiver. It occupies a wide spectrum, and can be easily built upon pervasive optical light sources and the pervasive consumer cameras. OCC emerges as a new form of visible light communication. It employs an image sensor assembled in consumer electronic devices. The imaging lens projects light on to the image sensor, which is comprised of multiple PD based pixels to detect the incident optical (photon) radiation. Each activated pixel generates a voltage proportional to the number of impinging photons

The paper uses deep learning algorithms such as neural networks called as Deep convolution neural network algorithm is applied. The advantage of the paper is that it uses dual camera system for an accurate information of the data which can be recognized by the system for recognition with a precision rate of 95%. Certain drawbacks like the developed algorithm utilizes a dual-camera system, which in turn can increase the implementation cost. In some cases of multiple set of different size images, it leads to the delay in getting the input from the camera system.

## **2.5 Real-time Large-Scale Traffic Detection.**

**Authors: Aleksej Avramovic, Domen Tabernik, and Danijel Skocaj**

**Year of Publishing:2018**

Concept used in this system is that the optimal camera communications(occ), computer vision and deep learning. The fundamentals of optical camera communication (OCC), with the OCC system employing the pervasive image sensor assembled in consumer electronics as the receiver. It occupies a wide spectrum, and can be easily built upon pervasive optical light sources and the pervasive consumer cameras. OCC emerges as a new form of visible light communication. It employs an image sensor assembled in consumer electronic devices.

Deep learning algorithms are applied. Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. The artificial neural networks are built like the human brain, with neuron nodes connected together like a web. It has advantage like the major advantage is that the results obtained from the experimental analysis shows that real time TSD is possible even for the HD images, with the mean averaged precision of 88%. It has drawbacks like the trade-off between accuracy is inevitable and the result with faster version is less accurate.

## **2.6 Vision-Based Traffic Sign Detection and Recognition Systems.**

**Authors:** Safat B. Wali , Majid A. Abdullah , Mahammad A. Hannan,Aini Hussain, Salina A,Samad, Pin J. Ker and Muhamad Bin Mansor

**Year of Publishing:** 2016

The system uses the concept of neural networks, computer vision and svm algorithm. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Computer Vision is the process of using machines to understand and analyze imagery (both photos and videos). While these types of algorithms have been around in various forms since the 1960's, recent advances in Machine Learning, as well as leaps forward in data storage, computing capabilities, and cheap high-quality input devices, have driven major improvements in how well our software can explore this kind of content.

Computer Vision is the broad parent's name for any computations involving visual content – that means images, videos, icons, and anything else with pixels involved. Its major advantage is results obtained from the experimental analysis shows that real time TSD is possible with the mean averaged precision of 89.2%. Its major drawback is there are a number of external non-technical challenges that may face this system in the real environment degrading its performance significantly. Among the many issues that needed to be addressed while developing a TSDR system are multiple appearance of sign, fading of colors.

## CHAPTER 3

### SYSTEM SPECIFICATION

Advances in massive knowledge technologies and AI algorithms like neural networks, and density-based applications will improve the user expertise, additional customized through the user expertise. This increase is achieved by increasing the worth of the business, grouping in depth historical knowledge and distinctive valuable rules. When Smart Vehicles technology is used, we increase driving assistance intelligence, the market prospect of a safety driving is possible to succeed. As a result, several researchers specialize in predicting various road traffic signs, road signals followed by multiple sensors for the application of driverless cars. A digital image processing system is the combination of the computer hardware and the image processing software. Selection of the camera, selection of the lens and lighting source, evaluation of image quality, selection of PC hardware and software and the configuration of all components – all of those are important steps toward an effective image processing system.

#### 3.1 Hard ware Specification

While considering the hardware aspects, some algorithms have to be modified to overcome the real time constraint. To implement the TSR system to hardware, we are using Simulink graphical tool for modeling, simulation which is suitable for making image processing block. Proposed FPGA (Field Programmable Gate Array) based hardware implementation of road signs. Then we are also using MATLAB system Generator as it has a big advantage in terms of the conception time. This tool provides the user some special building blocks to create the optimized designs for FPGA's. Moreover, to that the system uses a depth sensor and an ultra-wide angle camera lens for capturing the images in a high dynamic and stability mode. Most of the deep learning algorithms are seamlessly tested on the GPU. So a Gpu with high support is required for the System as a hardware requirement side to that Cpu also. Here, the main component of the system is the camera. The system a high-dynamic resolution camera with wide angle support for detecting and recognizing the images of traffic signs. A wide angle lens is also one that projects a substantially larger image circle than would be typical for a standard design lens of the same focal length. This large image circle enables either large tilt & shift movements with a view camera, or a wide field of view. Another result of using a wide-angle lens is a greater apparent perspective distortion when the camera is not aligned perpendicularly to the subject: parallel lines converge at the same rate as with a normal lens, but converge more due to the wider total field. For example, buildings appear

to be falling backwards much more severely when the camera is pointed upward from ground level than they would if photographed with a normal lens at the same distance from the subject, because more of the subject building is visible in the wide-angle shot. Also supported by a low light image sensor. Image sensors work in much the same way as film, allowing light to enter and exposing the image to the sensor. These devices are available in two flavors — CMOS image sensors or charge-coupled devices (CCDs). While CCDs have traditionally held the upper hand because of light sensitivity and higher resolution, developments in CMOS image sensors are changing the game. Combined with their cheaper price, they also offer faster processing, lower power usage, and integrated camera functions. The NV-CMOS® solution will integrate a low-light CMOS imager with video processing capabilities for image enhancement. This powerful, compact sensor has the potential to achieve unprecedented performance in low-light conditions for night vision and ground-based and airborne intelligence, surveillance, and reconnaissance applications—as well as in microscopy, biofluorescence, and DNA sequencing. Because different lenses generally require a different camera–subject distance to preserve the size of a subject, changing the angle of view can indirectly distort perspective, changing the apparent relative size of the subject and foreground.

### **3.2 Software Specification**

The software specification involves the use of software for the implementation of the idea through deep learning for the detection and recognition of traffic sign. We use tensorflow as AI development tool and python. The softwares which support for the implementation are keras by google, tensorflow, pytorch and etc. keras is on the top of all other deep learning library. To run keras on the frontend tensorflow must be runned on the backened. Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility). Supports both convolutional networks and recurrent networks, as well as combinations of the two. Runs seamlessly on CPU and GPU. Most of the deep learning algorithms are seamlessly tested on the GPU. Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations. To give a concrete example, Google users can experience a faster and more refined the search with AI. If the user types a keyword the search bar, Google provides a recommendation about what could be the next word. TensorFlow is a computational framework for building machine learning models. You can use lower-level APIs to build models by defining a series of mathematical operations. Alternatively, you can use higherlevel APIs (like tf.estimator) to specify predefined architectures, such as linear regressors or neural networks. It is implmented through a jupyter server for the python code implementation. JupyterLab is a web-based interactive

development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

## **CHAPTER 4**

### **PROPOSED METHODOLOGY**

#### **(MODULE DESCRIPTION)**

In a word, accuracy. Deep learning achieves recognition accuracy at higher levels than ever before. This helps consumer electronics meet user expectations, and it is crucial for safety-critical applications like driverless cars. Recent advances in deep learning have improved to the point where deep learning outperforms humans in some tasks like classifying objects in images. Deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

It requires large amounts of labeled data. For example, driverless car development requires millions of images and thousands of hours of video. It also requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

Deep learning subset of Machine learning provide systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning



algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

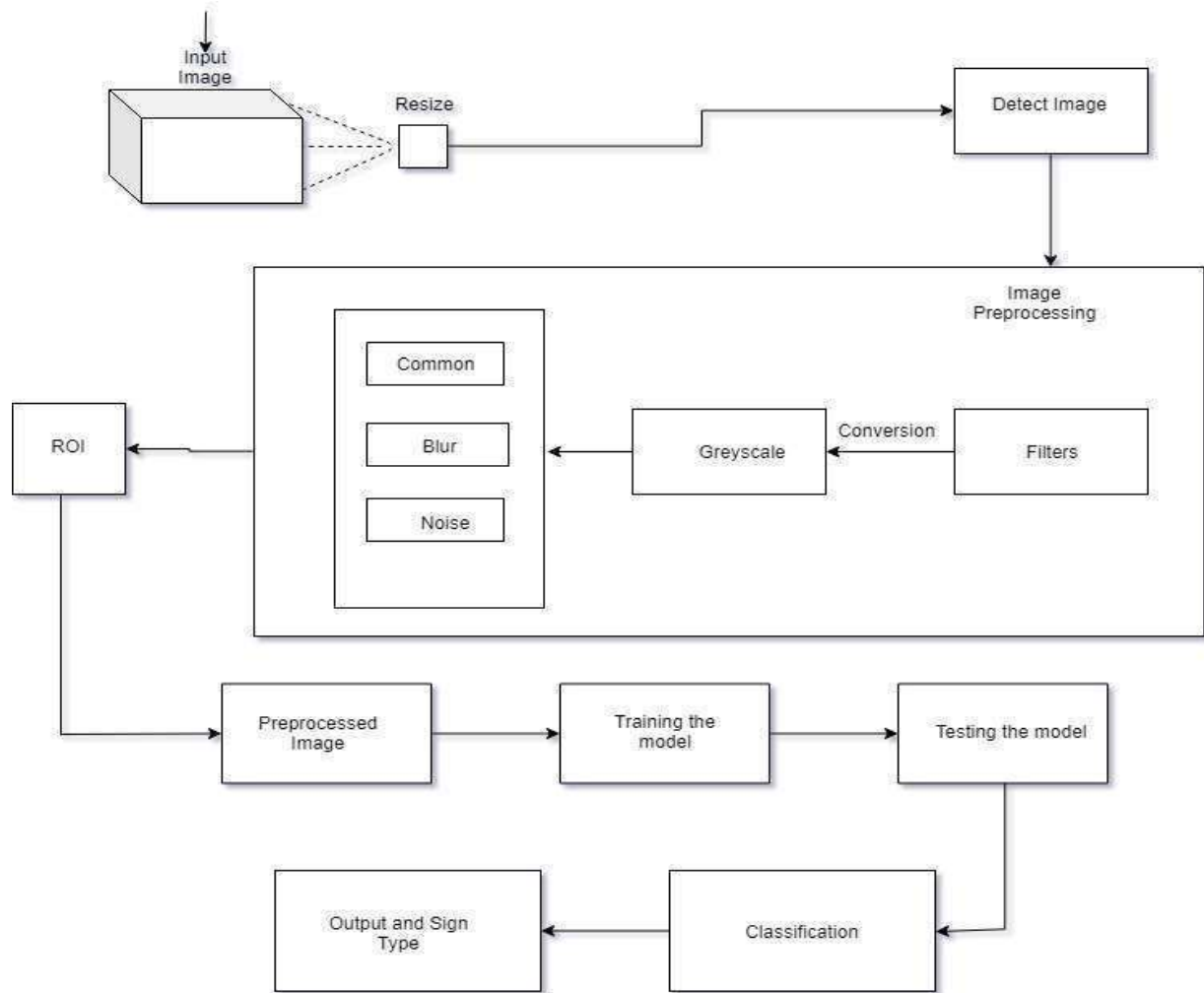
In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Deep learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining deep learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images. CNNs eliminate the need for manual feature extraction, so you do not need to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pretrained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.

The system takes a set of input images which are adjusted, resized in terms of shape and configuration. Images are converted in to greyscale format for the computer vision called preprocessing stage of the image. The greyscaled image is converted in to pixel values ranging from 0 to 255. Pixel values for a greyscaled image are put in a matrix form of representation. The values are arranged based upon the regions of interest. The process was done for all the images which are passed as input. Now, the entire set of preprocessed 15 images are passed in to the convolutional layer of the network model for training and testing the model for output and classification of images.

The architecture of the model looks like:



**Fig 4(a) Proposed Architecture Model**

The fundamental function of the system is to capture the image or video so that it can be further processed. In fact, the camera should be properly calibrated and its dynamic range should be adjusted before capturing any scene. Recently, high dynamic range cameras (HDR) based on CMOS technology had begun to develop in field of driver assistance systems since they provide high contrast images during night. The position and resolution of cameras acquiring the images of signs is the main concern. If the cameras are placed too far away, then the image captured will be blurred and cannot be detected accurately. But if the camera is placed too close (maybe on bumper of car); it will capture images which can be affected by the climatic conditions. Hence the distance of camera from the image determines the quality of image. So, an ultra-wide-angle camera supporting HDR is used for the project for detection and classification of road traffic signs.

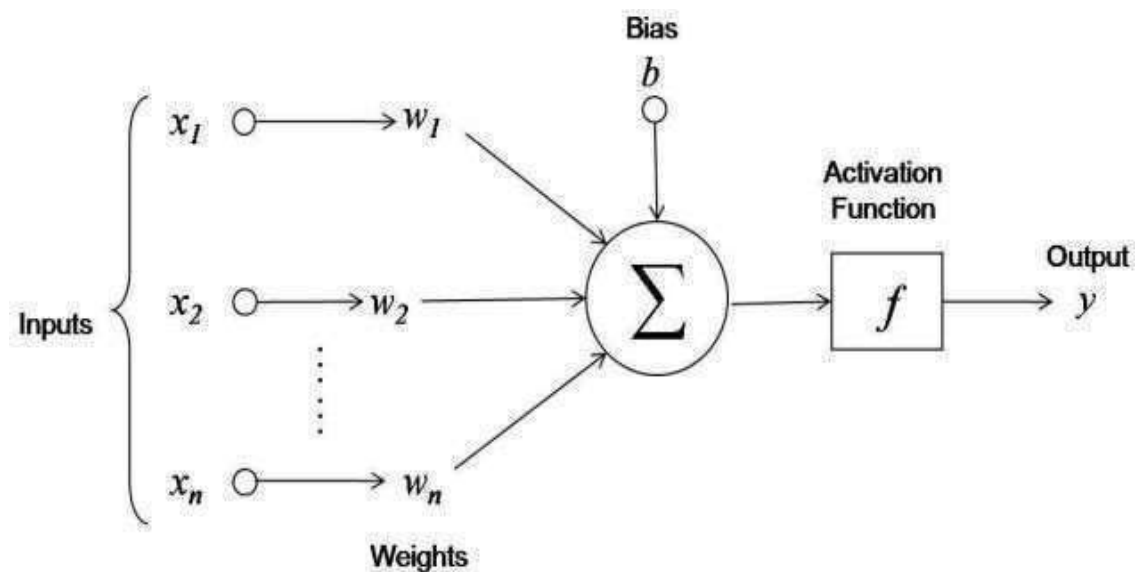
The traffic sign detected in the image is small and noisy if captured with a low-resolution camera and the image would not be detected efficiently. Hence many researchers have tried to use cameras of resolution more than 640\*480 pixels. But practically this resolution is insufficient while dealing with signs which are far away from the camera. There can also be excessive tilt in camera plane while capturing the image. All of these factors can make it impossible to recover the useful information from the images. The system uses tensorflow and python as software requirements for implementing neural networks algorithm on training the data and testing it by the providing a set of testing images.

The model takes set of input images for processing and training the model. The bias function in neural networks takes the set of input values from the matrix format of the image and assigns certain weights where the summation of all weights and inputs form a bias function. The bias function is later converted in to activation function for linear model of the outputs on the graph.

The mathematical expression can be given as

$$\Rightarrow Y = F(x) = \sum x_i w_i$$

$$\Rightarrow \text{Output} = \text{sum}(\text{weights} * \text{inputs}) + \text{Bias function}$$



**Fig 4(b) Bias in Neural Networks**

#### 4.1 Prepare a Dataset of images

The Deep neural network has set a benchmark in recognition of images and voices. The Current paper uses a convolutional layer of neural network architecture to detect and classify the road traffic signs .The Proposed model uses a dataset of traffic sign images named GTSRB.

The German Traffic sign Benchmarks dataset used for the training of the model.The GTSRB dataset was splitted in to training set and testing set of which training set has a 4167 images in a 32\*32 format for training of model and for testing and validation of the trained model,the paper uses a 1994 images to get a desired output. The images are of 1236\*968 pixels with a measured ground truth value. The training and testing datasets may have different set of images .One with an uneven color while the other with sign tilt or distortion etc., The problem statements of the above kind are come across the preprocessing stat of the image

About GTSRB : German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. They cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge.



**Fig 4.1(a) GTSRB Dataset of Traffic Sign Images**

Our benchmark has the following properties:

->Single-image, multi-class classification problem

->More than 40 classes

->More than 50,000 images in total

->Large, lifelike database

#### **4.1.1 Overview of Dataset**

- Single-image, multi-class classification problem
- More than 40 classes
- 4,000 images in total
- Reliable ground-truth data due to semi-automatic annotation
- Physical traffic sign instances are unique within the dataset (i.e., each real-world traffic sign only occurs once)

#### **4.1.2 Structure**

- The training set archive is structured as follows:
- One directory per class
- Each directory contains one CSV file with annotations ("GT-<ClassID>.csv") and the training images
- Training images are grouped by tracks

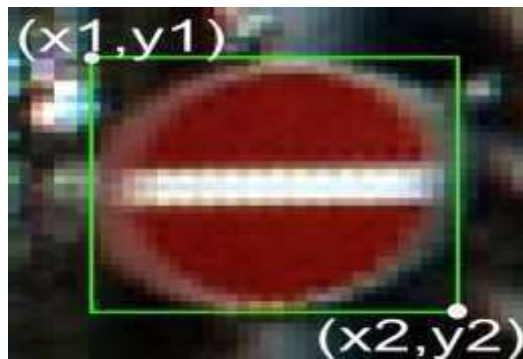
#### **4.1.3 Image format**

- The images contain traffic sign each with various image formats.
- Images contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches
- Images are stored in PPM format (Portable Pixmap, P6)
- Image sizes vary between 15x15 to 250x250 pixels
- Images are not necessarily squared

- The actual traffic sign is not necessarily centered within the image. This is true for images that were close to the image border in the full camera image
- The bounding box of the traffic sign is part of the annotations.

#### 4.1.4 Annotation format

- Annotations are provided in CSV files. Fields are separated by ";" (semicolon).
- Filename: Filename of corresponding image
- Width: Width of the image
- Height: Height of the image



**Fig 4.1(b) ROI Format of Image**

## 4.2 Image Pre processing

Computers today cannot only automatically classify photos, but can also describe the various elements in pictures and write short sentences describing each segment with proper English grammar. This is done by the Deep Learning network (CNN) which actually learns patterns that naturally occur in photos. Imagenet is one of the biggest databases of labeled images to train the Convolutional Neural Networks using GPU-accelerated deep learning frameworks such as Caffe2, Chainer, Microsoft Cognitive Toolkit, MXNet, PaddlePaddle, Pytorch, TensorFlow, and inference optimizers such as TensorRT. Deep learning, also called neural networks, is a subset of machine learning that uses a model of computing that's very much inspired by the structure of the brain.

Convolutional neural networks (CNNs) represent an interesting method for adaptive image processing, and form a link between general feed-forward neural networks and adaptive filters. Two dimensional CNNs are formed by one or more layers of two dimensional filters, with possible non-linear activation functions and/or down-sampling. CNNs possess key properties of translation invariance and spatially local connections (receptive fields). We present a description of the convolutional network architecture, and an application to practical image processing on a mobile robot. A CNN is used to detect and characterize cracks on an autonomous sewer inspection robot. The filter sizes used in all cases were 4x4, with non-linear activations between each layer.

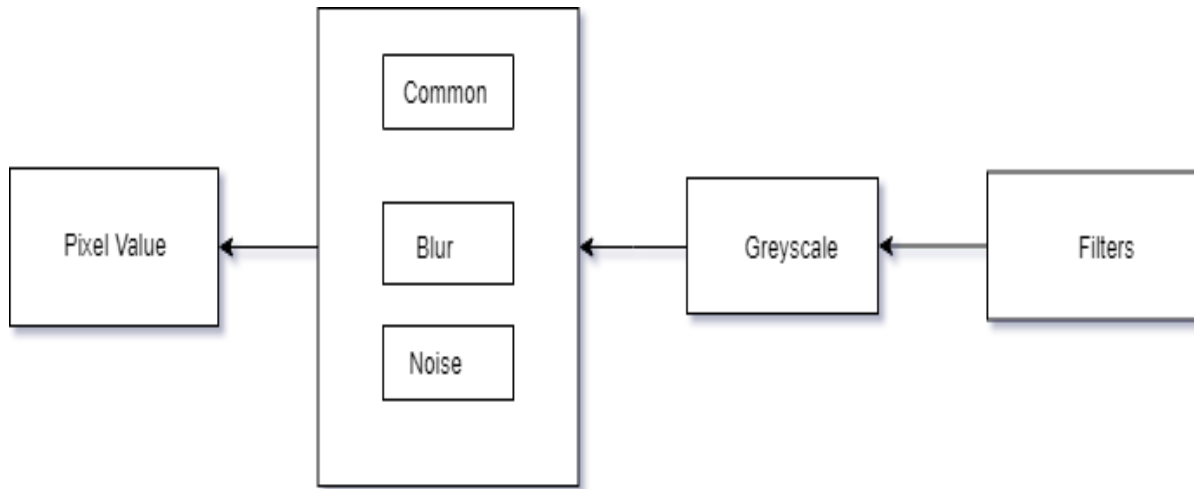
The number of feature maps used in the three hidden layers was, from input to output, 4, 4, 4. The network was trained using a dataset of 48x48 sub-regions drawn from 30 still image 1236\*968 pixel frames sampled from a pre-recorded sewer pipe inspection video. 15 frames were used for training and 15 for validation of network performance. Although development of a CNN system for civil use is on-going, the results support the notion that data-based adaptive image processing methods such as CNNs are useful for image processing, or other applications where the input arrays are large, and spatially / temporally distributed. Further refinements of the CNN architecture, such as the implementation of separable filters, or extensions to three dimensional (ie. video) processing, are suggested.

The training set provides the input image for the model. For training the neural network, the input image must be of a normalized form with a pixel value ranging from 0 to 255. Input images are adjusted and reshaped by sharpening the edges and by removing any disturbance further if any such as high saturation. For filtering the images the system uses Gaussian Filter algorithm which helps in reduction of noise in blur areas of the image. First, we need to add a little bit of variance to the data since the images from the dataset are very organized and contain little to no noise. We're going to artificially add noise using a Python library named PIL. We're going to do a random combination of the following to the images.

- Crop parts of the image
- Flip image horizontally
- Adjust hue, contrast and saturation

For increased accuracy, Image classification using CNN is most effective. First and foremost, we need a set of images. In this case, we take images of beauty and pharmacy products, as our initial training data set. The most common image data input parameters are the number of images, image dimensions, number of channels, and number of levels per pixel.

The Preprocessing Image stage can be shown as below:



**Fig 4.2 (a) Image Preprocessing Block**

#### **4.2.1 Filters**

Image filtering is useful for many applications, including smoothing, sharpening, removing noise, and edge detection. A filter is defined by a kernel, which is a small array applied to each pixel and its neighbors within an image. In most applications, the center of the kernel is aligned with the current pixel, and is a square with an odd number (3, 5, 7, etc.) of elements in each dimension. The process used to apply filters to an image is known as convolution, and may be applied in either the spatial or frequency domain.

Within the spatial domain, the first part of the convolution process multiplies the elements of the kernel by the matching pixel values when the kernel is centered over a pixel. The elements of the resulting array (which is the same size as the kernel) are averaged, and the original pixel value is replaced with this result. The convol function performs this convolution process for an entire image.

Within the frequency domain, convolution can be performed by multiplying the FFT (Fast Fourier Transform) of the image by the FFT of the kernel, and then transforming back into the spatial domain. The kernel is padded with zero values to enlarge it to the same size as



the image before the forward FFT is applied. These types of filters are usually specified within the frequency domain and do not need to be transformed. IDL's `DIST` and `hanning` functions are examples of filters already transformed into the frequency domain. See [Windowing to Remove Noise](#) for more information on these types of filters. The following examples in this section will focus on some of the basic filters applied within the spatial domain using the `convol` function:

- Low Pass Filtering
- High Pass Filtering
- Directional Filtering
- Laplacian Filtering

Since filters are the building blocks of many image processing methods, these examples merely show how to apply filters, as opposed to showing how a specific filter may be used to enhance a specific image or extract a specific shape. This basic introduction provides the information necessary to accomplish more advanced image-specific processing.

One way of thinking of Filtering is as an operation of removing frequency components from an image – ‘selectively’. For filtering the images the system uses Gaussian Filter algorithm which helps in reduction of noise in blur areas of the image. The system also uses a Gabor filter, named after Dennis Gabor, is a linear filter used for texture analysis, which means that it basically analyzes whether there are any specific frequency content in the image in specific directions in a localized region around the point or region of analysis. It Basically, forms a feature map for the convolutional layer of the neural network by a dot product with input image in matrix format of pixel values.

In General, CNNs compare images piece by piece. The pieces that it looks for are called features. By finding rough feature matches in roughly the same positions in two images, CNNs get a lot better at seeing similarity than whole-image matching schemes. Each feature is like a mini-image—a small two-dimensional array of values. Features match common aspects of the images. In the case of X images, features consisting of diagonal lines and a crossing capture all the important characteristics of most X's. These features will probably match up to the arms and center of any image of an X.

1..

#### **4.2.2 Greyscale Conversion**

Conversion of a color image into a grayscale image inclusive of salient features is a complicated process. The converted grayscale image may lose contrasts, sharpness, shadow, and structure of the color image. To preserve contrasts, sharpness, shadow, and structure of the color image a new algorithm has proposed. To convert the color image into grayscale image the algorithm performs RGB approximation, reduction, and addition of chrominance and luminance. The grayscale images generated using the algorithm in the experiment confirms that the algorithm has preserved the salient features of the color image such as contrasts, sharpness, shadow, and image structure.

The Gray World Assumption is a white balance method that assumes that your scene, on average, is a neutral gray. Gray-world assumption hold if we have a good distribution of colors in the scene. ... Gray world algorithm produces an estimate of illumination by computing the mean of each channel of the image. In digital photography, computer-generated imagery, and colorimetry, a grayscale or greyscale image is one in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information. Grayscale images, a kind of black-and-white or gray monochrome, are composed exclusively of shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest. Grayscale images are distinct from one-bit

bi-tonal black-and-white images which, in the context of computer imaging, are images with only two colors: black and white (also called bilevel or binary images).

Grayscale images have many shades of gray in between. Grayscale images can be the result of measuring the intensity of light at each pixel according to a particular weighted combination of frequencies (or wavelengths), and in such cases they are monochromatic proper when only a single frequency (in practice, a narrow band of frequencies) is captured. The frequencies can in principle be from anywhere in the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.). A colorimetric (or more specifically photometric) grayscale image is an image that has a defined grayscale colorspace, which maps the stored numeric sample values to the achromatic channel of a standard colorspace, which itself is based on measured properties of human vision.



**Fig 4.2(b) RGB Image**



**Fig 4.2(c) GREYSCALED Image**

Grayscale as single channels of multichannel color images.

Color images are often built of several stacked color channels, each of them representing value levels of the given channel. For example, RGB images are composed of three independent channels for red, green and blue primary color components; CMYK images have four channels for cyan, magenta, yellow and black ink plates, etc. Above is an example of color channel splitting of a full RGB color image.

### 4.2.3 Pixel and Matrix Representation

Data in computers is stored and transmitted as a series of ones and zeros (also known as Binary). To store an image on a computer, the image is broken down into tiny elements called pixels. A pixel (short for picture element) represents one colour. Colour depth – how many bits represent each pixel. The word “pixel” means a picture element. Every photograph, in digital form, is made up of pixels. They are the smallest unit of information that makes up a picture. Usually round or square, they are typically arranged in a 2-dimensional grid. As you can see, the pixels approximate the actual image. In General, A 'pixel' (short for 'picture element') is a tiny square of colour. Lots of these pixels together can form a digital image. Each pixel has a specific number and this number tells the computer

what colour the pixel should be. The process of digitisation takes an image and turns it into a set of pixels.

The intensity of a pixel is expressed within a given range between a minimum and a maximum, inclusive. This range is represented in an abstract way as a range from 0 (or 0%) (total absence, black) and 1 (or 100%) (total presence, white), with any fractional values in between. This notation is used in academic papers, but this does not define what "black" or "white" is in terms of colorimetry. Sometimes the scale is reversed, as in printing where the numeric intensity denotes how much ink is employed in halftoning, with 0% representing the paper white (no ink) and 100% being a solid black (full ink).

In computing, although the grayscale can be computed through rational numbers, image pixels are usually quantized to store them as unsigned integers, to reduce the required storage and computation. Some early grayscale monitors can only display up to sixteen different shades, which would be stored in binary form using 4-bits. But today grayscale images (such as photographs) intended for visual display (both on screen and printed) are commonly stored with 8 bits per sampled pixel. This pixel depth allows 256 different intensities (i.e., shades of gray) to be recorded, and also simplifies computation as each pixel sample can be accessed individually as one full byte.

However, if these intensities were spaced equally in proportion to the amount of physical light they represent at that pixel (called a linear encoding or scale), the differences between adjacent dark shades could be quite noticeable as banding artifacts, while many of the lighter shades would be "wasted" by encoding a lot of perceptually-indistinguishable increments. Therefore, the shades are instead typically spread out evenly on a gamma-compressed nonlinear scale, which better approximates uniform perceptual increments for both dark and light shades, usually making these 256 shades enough (just barely) to avoid noticeable increments.

Technical uses (e.g. in medical imaging or remote sensing applications) often require more levels, to make full use of the sensor accuracy (typically 10 or 12 bits per sample) and to reduce rounding errors in computations. Sixteen bits per sample (65,536 levels) is often a convenient choice for such uses, as computers manage 16-bit words efficiently. The TIFF and PNG (among other) image file formats support 16-bit grayscale natively, although browsers and many imaging programs tend to ignore the low order 8 bits of each pixel.

Internally for computation and working storage, image processing software typically uses integer or floating-point numbers of size 16 or 32 bits. A digital grayscale image is presented in the computer by pixels matrix.

The numeric values in pixel presentation are uniformly changed from zero (black pixels) to 255 (white pixels). A digital grayscale image is presented in the computer by pixels matrix. The numeric values in pixel presentation are uniformly changed from zero to 255.

```
[[[ 50  34  96]
   [ 63  26 119]
   [ 59   4 108]
   ...
   [144 153 136]
   [150 162 142]
   [143 155 135]]]

[[[ 49  27  91]
   [ 59  21 106]
   [ 72  24 110]
   ...
   [109 114  92]
   [117 126  99]
   [111 120  93]]]

[[[ 58  27  95]
   [ 61  24  94]
   [110  70 130]
   ...
   [101 105  72]
   [ 99 103  68]
   [ 93  97  62]]]

...

[[[ 89  80  75]
   [ 89  80  75]
   [ 90  80  78]
   ...
   [110 115  92]
   [115 120  97]
   [115 120  97]]]

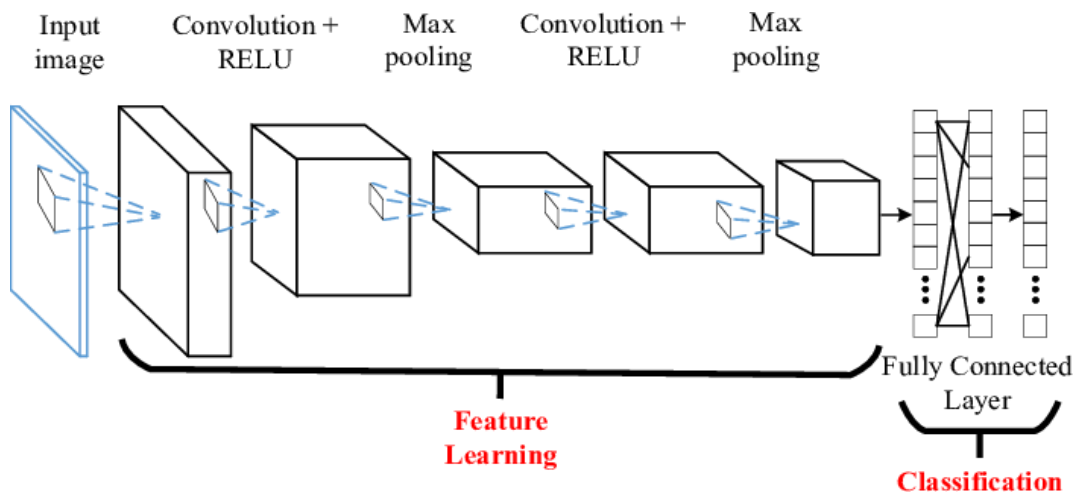
[[[ 90  82  79]
   [ 92  84  81]
   [ 94  86  84]
   ...
   [117 120  99]
   [119 122 101]
   [119 122 101]]]

[[[ 91  86  83]
   [ 94  89  86]
   [ 94  89  86]
   ...
   [116 117  99]
   [117 119  98]
   [117 119  98]]]
```

**Fig 4.2(d) Matrix form**

### 4.3 Network Structure

The Proposed model is based upon a Convolutional Neural Network structure. The network structure looks like:



**Fig 4.3(a) CNN Architecture**

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision. Artificial neural networks are one of the main tools used in machine learning. As the “neural” part of their name suggests, they are brain-inspired systems which are intended to replicate the way that we humans learn. Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. They are excellent tools for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize.

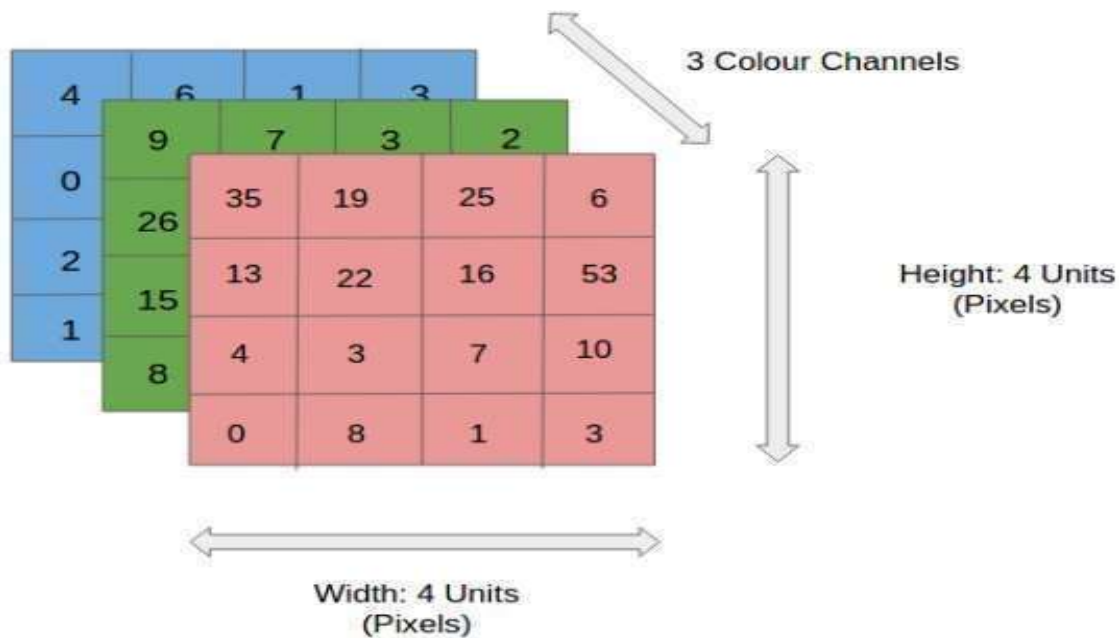
ANN is rarely used for predictive modelling. The reason being that Artificial Neural Networks (ANN) usually tries to over-fit the relationship. ANN is generally used in cases where what has happened in past is repeated almost exactly in same way. For example, say we are playing the game of Black Jack against a computer. An intelligent opponent based on ANN would be a very good opponent in this case (assuming they can manage to keep the computation time low). With time ANN will train itself for all possible cases of card flow. And given that we are not shuffling cards with a dealer, ANN will be able to memorize every single call. Hence, it is a kind of machine learning technique which has enormous memory. But it does not work well in case where scoring population is significantly different compared to training sample. For instance, if I plan to target customer for a campaign using their past response by an ANN. I will probably be using a wrong technique as it might have over-fitted the relationship between the response and other predictors.

The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm - a Convolutional Neural Network.

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist Grayscale, RGB, HSV, etc. You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.



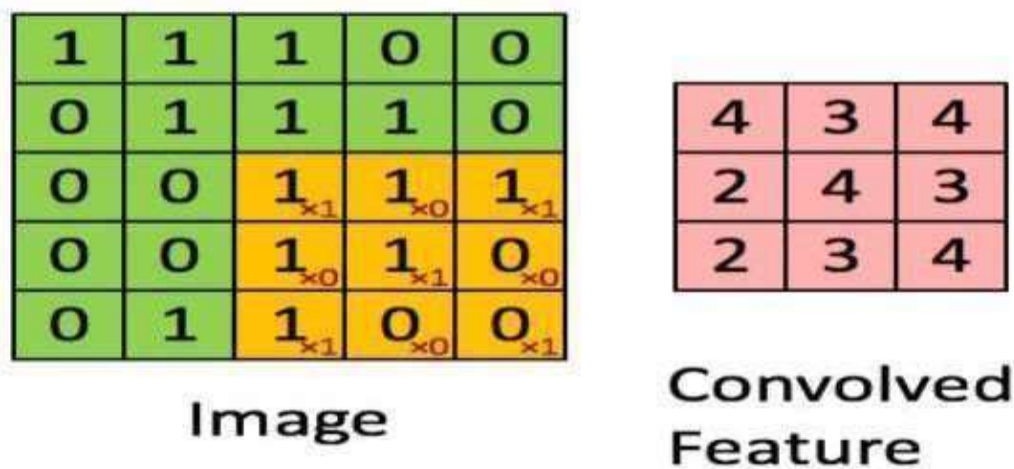
**Fig 4.3(b) 4x4x3 RGB Image**

### 4.3.1 Convolution Layer -The Kernel

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image. The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

A 3 convolution layers with 2\*2 max pooling was used for training the model. 3d matrix form of single convolution layer is as shown. Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB). In the above demonstration, the green section resembles our 5x5x1 input image. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix. The given input filter passes over the input image and forms dot product with the pixel values of the input image and forms a convolved feature map as shown in the figure.

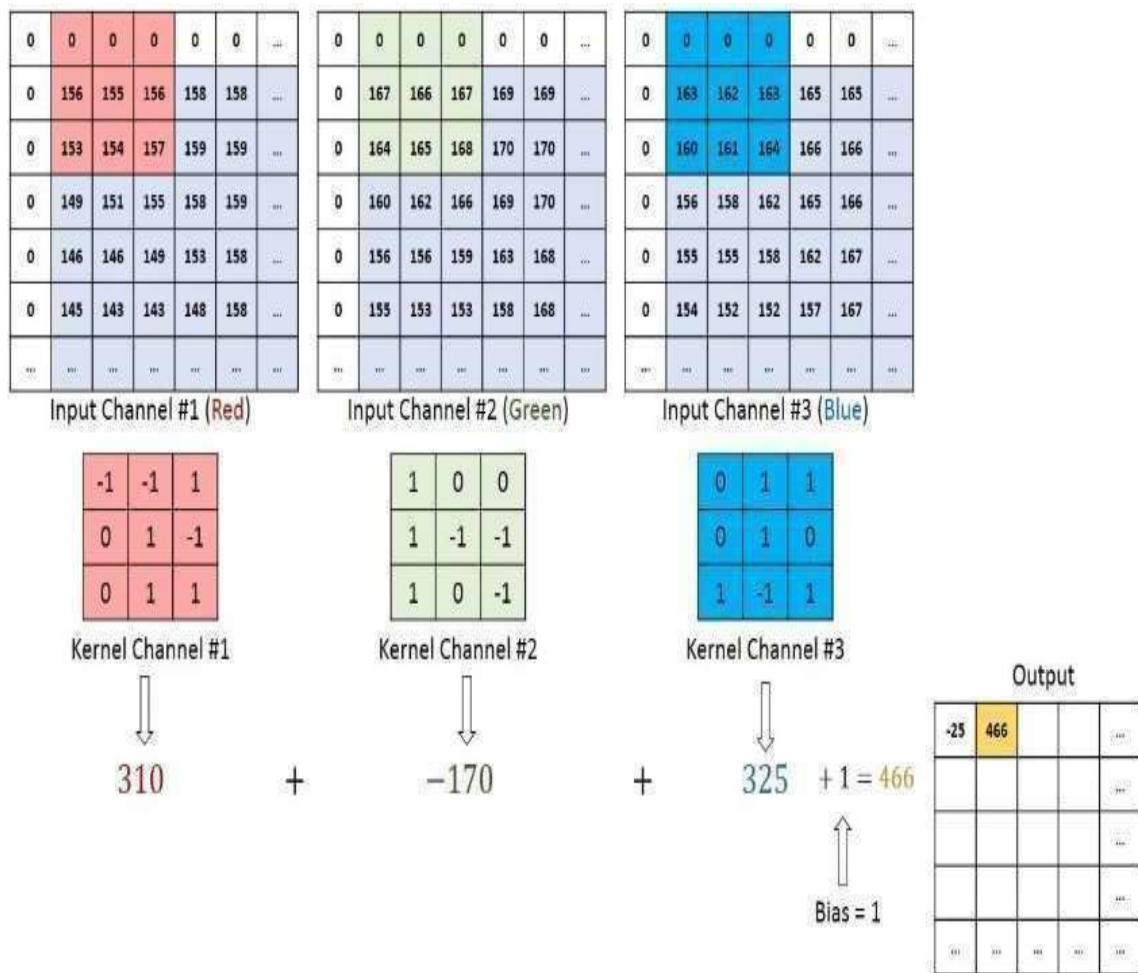




**Fig 4.3(c) Convoluting 5x5x1 image with a 3x3x1 kernel to get 3x3x1 convolved feature**

The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is the element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the “scalar product”. Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.



**Fig 4.3(d) Convolution operation on a  $M \times N \times 3$  image matrix with a  $3 \times 3 \times 3$  Kernel**

### 4.3.2 Normalization

We normalize the input layer by adjusting and scaling the activations. For example, when we have features from 0 to 1 and some from 1 to 1000, we should normalize them to speed up learning. If the input layer is benefiting from it, why not do the same thing also for the values in the hidden layers, that are changing all the time, and get 10 times or more improvement in the training speed.

Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). To explain covariance shift, let's have a deep network on cat detection. We train our data on only black cats' images. So, if we now try to apply this network to data with colored cats, it is obvious; we're not going to do well. The training set and the prediction

set are both cats' images but they differ a little bit. In other words, if an algorithm learned some X to Y mapping, and if the distribution of X changes, then we might need to retrain the learning algorithm by trying to align the distribution of X with the distribution of Y. Also, batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.

We can use higher learning rates because batch normalization makes sure that there's no activation that's gone really high or really low. And by that, things that previously couldn't get to train, it will start to train. It reduces overfitting because it has a slight regularization effects. Similar to dropout, it adds some noise to each hidden layer's activations. Therefore, if we use batch normalization, we will use less dropout, which is a good thing because we are not going to lose a lot of information. However, we should not depend only on batch normalization for regularization; we should better use it together with dropout.

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. However, after this shift/scale of activation outputs by some randomly initialized parameters, the weights in the next layer are no longer optimal. SGD ( Stochastic gradient descent) undoes this normalization if it's a way for it to minimize the loss function.

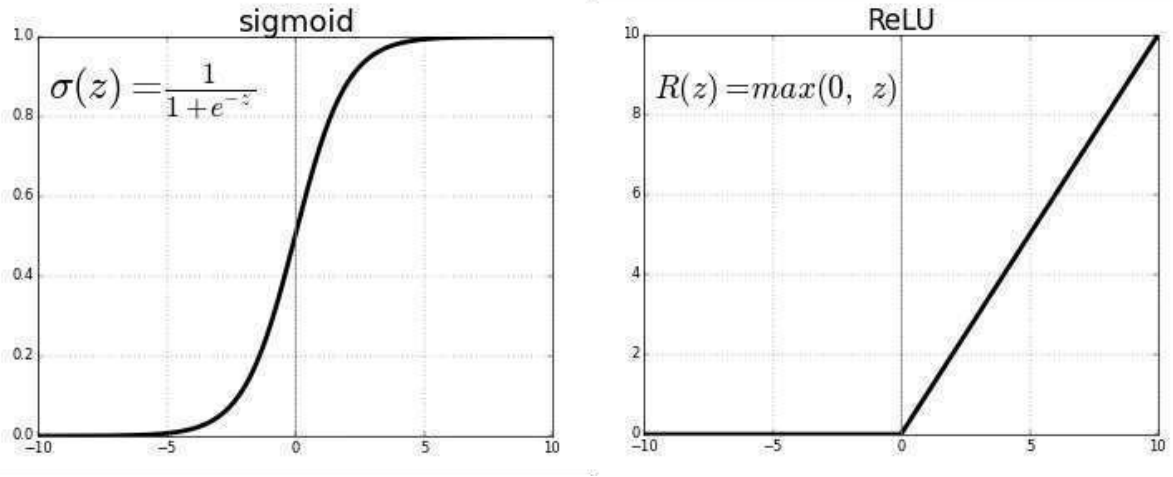
Consequently, batch normalization adds two trainable parameters to each layer, so the normalized output is multiplied by a "standard deviation" parameter (gamma) and add a "mean" parameter (beta). In other words, batch normalization lets SGD do the denormalization by changing only these two weights for each activation, instead of losing the stability of the network by changing all the weights.

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. RELU layer will apply an elementwise activation function, such as the  $\max(0, x)$  thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).

As you can see, the ReLU is half rectified (from bottom).  $f(z)$  is zero when  $z$  is less than zero and  $f(z)$  is equal to  $z$  when  $z$  is above or equal to zero.

Range: [ 0 to infinity)

The function and its derivative both are monotonic. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.



**Fig 4.3(e) ReLU v/s Logistic Sigmoid**

### 4.3.3 POOLING LAYER

Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input. Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g. lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image. A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task.

Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image. A more robust and common approach is to use a pooling layer. A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g. ReLU) has been applied to the feature maps output by a convolutional layer. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting.

The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps. Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always  $2 \times 2$  pixels applied with a stride of 2 pixels.

This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. For example, a pooling layer applied to a feature map of  $6 \times 6$  (36 pixels) will result in an output pooled feature map of  $3 \times 3$  (9 pixels). The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

Average Pooling: Calculate the average value for each patch on the feature map.  
Maximum Pooling (or Max Pooling): Calculate the maximum value for each patch of the feature map.

The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. This capability added by pooling is called the model's invariance to local translation.

### 4.3.3.1 Average pooling

On two-dimensional feature maps, pooling is typically applied in  $2 \times 2$  patches of the feature map with a stride of (2,2). Average pooling involves calculating the average for each patch of the feature map. This means that each  $2 \times 2$  square of the feature map is down sampled to the average value in the square. Because the down sampling operation halves each dimension, we will expect the output of pooling applied to the  $6 \times 6$  feature map to be a new  $3 \times 3$  feature map. Given the horizontal symmetry of the feature map input, we would expect each row to have the same average pooling values. Therefore, we would expect the resulting average pooling of the detected line feature map. Average pooling works well, although it is more common to use max pooling.

### 4.3.3.2 Maximum Pooling

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map.

The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling. This has been found to work better in practice than average pooling for computer vision tasks like image classification. We can make the max pooling operation concrete by again applying it to the output feature map of the line detector convolutional operation and manually calculate the first row of the pooled feature map. Max pooling is a technique used to reduce the dimensions of an image by considering the maximum amount of pixel value in the grid. Max pooling helps to reduce over fitting and makes the model to look more generic.

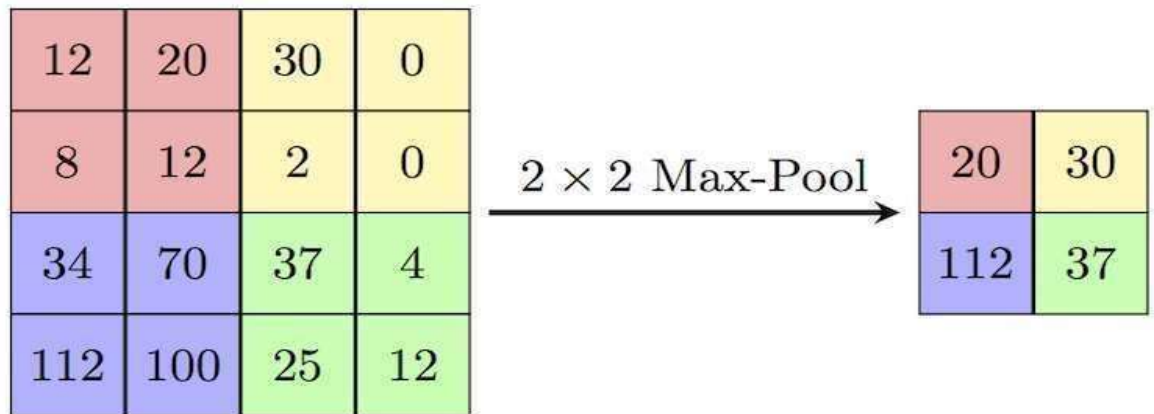


Fig 4.3(f) Down sampling

#### 4.3.4 Output layer

After multiple layers of convolution and padding, we would need the output in the form of a class. The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original images. However, to generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need. It becomes tough to reach that number with just the convolution layers. Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class. The output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction.

The output layer in a CNN as mentioned previously is a fully connected layer, where the input from the other layers is flattened and sent so as to transform the output into the number of classes as desired by the network. The output is then generated through the output layer and is compared to the output layer for error generation. A loss function is defined in the fully connected output layer to compute the mean square loss. The gradient of error is then calculated. The error is then backpropagated to update the filter(weights) and bias values. One training cycle is completed in a single forward and backward pass.

The objective of a fully connected layer is to take the results of the convolution/pooling process and use them to classify the image into a label (in a simple classification example). The output of convolution/pooling is flattened into a single vector of values, each representing a probability that a certain feature belongs to a label. The fully connected part

of the CNN network goes through its own backpropagation process to determine the most accurate weights. Each neuron receives weights that prioritize the most appropriate label. Finally, the neurons “vote” on each of the labels, and the winner of that vote is the classification decision.

A Softmax function is a type of squashing function. Squashing functions limit the output of the function into the range 0 to 1. This allows the output to be interpreted directly as a probability. Since the outputs of a softmax function can be interpreted as a probability (i.e. they must sum to 1), a softmax layer is typically the final layer used in neural network functions. We use softmax as the output function of the last layer in neural networks (if the network has  $n$  layers, the  $n$ -th layer is the softmax function). This fact is important because the purpose of the last layer is to turn the score produced by the neural network into values that can be interpreted by humans. To understand the softmax function, we must look at the output of the  $(n-1)$ th layer. In this layer, the values get multiplied by some weights, passed through an activation function and aggregated into a vector which contains one value per every class of the model.

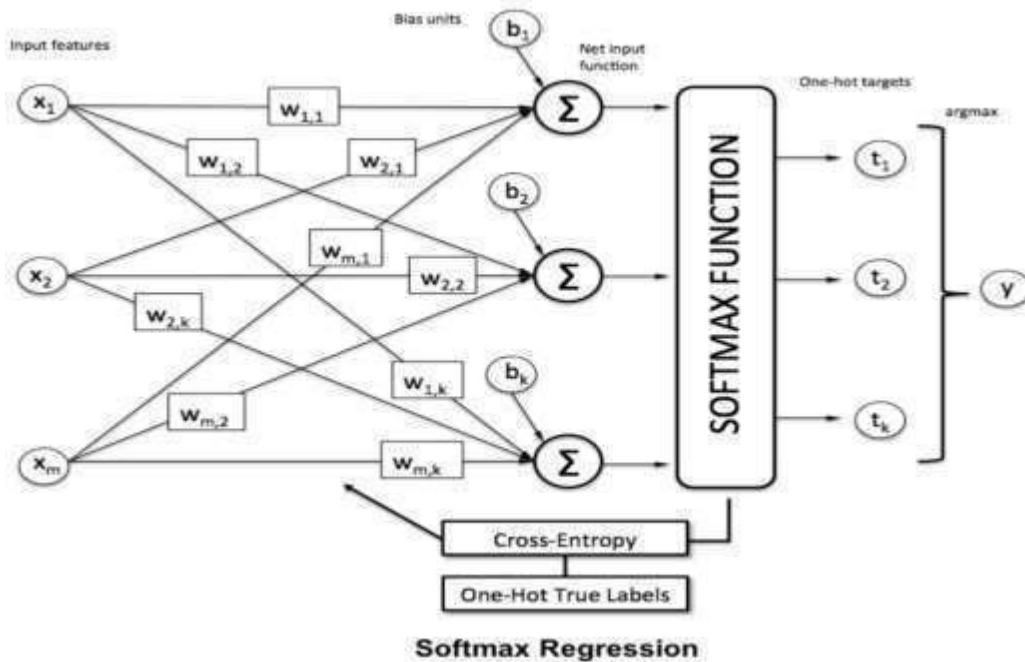


Fig 4.3(g) Softmax Regression in Output layer



## 4.4 Testing and Recognition

Several adjustments were made for training the data. The paper uses a batch size of 32 since the images were around from 2000-3000. The training set uses epoch of 40. After iteration of 20,000 times, the precision accuracy was found to be 96%. Batch size of 32 and epoch of 20 /50 are used for testing set. The model is capable of recognizing the images with an accuracy of 97% for an epoch value of 50. For increase in epoch value can produce accuracy rate of 98%, but time complexity of algorithm increases. The model is tested over several classes (up to 43) for some classes the training and testing pattern can be shown in the below tabular column.

class	output label	# patterns		
		training	testing	total
background	0 00000000	2489	415	2904
30 km/h max	1 10000000	77	13	90
50 km/h max	1 01000000	120	20	140
60 km/h max	1 00100000	62	10	72
70 km/h max	1 00010000	129	21	150
80 km/h max	1 00001000	75	13	88
90 km/h max	1 00000100	79	13	92
100 km/h max	1 00000010	69	12	81
		3100	517	3617

**Table 4.4(a) Dataset configuration for Desired Classes**



**Fig 4.4(a) Testing of the Trained Model**

## 4.5 Experimental Analysis

After the experiment, the data were as follows: in the process of the driver when the vehicle speed is guaranteed in 60 km/h, when the intersection angle is  $45^\circ$ , the average length of time the driver turned to 0.773 s, observe the traffic marker is 0.385 s, after the return is the average length of 0.747 s. When the intersection angle is  $75^\circ$ , the average turning time of the driver is 0.257 s, while the traffic sign is 0.365 s while the average time to go back is 0.286 s. It can be seen that at the intersection of  $t_1 = t_4$  at the same corner, the time of T1 and T4 will change with the angle of the angle when the drivers' driving speed is ensured.

On comparison with the existing system, the proposed model was able to overcome the drawbacks of existing system with an prediction accuracy rate of 98.9 % which is a satisfactory value for recognition of traffic signs.

	<b>Existing System</b>	<b>Proposed System</b>
<b>Algorithm</b>	Multitask Neural Network and SVM	Conv NET
<b>Accuracy rate (Prediction)</b>	97.1	98.9
<b>Dataset</b>	MASTIF	GTSRB

**Table 4.5(a) Comparison of Existing system Vs Proposed System**

## 4.6 Algorithm based on Artificial Neural Network

Artificial neural networks (ANNs) are old and used to the system know the assignment of property values that can be used to categorise hidden and new unusual activities. ANN is influenced by the pattern or behaviour of the users or people brain and contain of interconnected nodes and balanced connections. ANN nodes are called neurons similar to biological neurons . Various network architectures are developed on the basis of real actions. More commonly used hierarchy contains of three levels: the input level, the hidden level, and the output level. The flow of information is represented by a line between one node. The nodes in the input hierarchy take one value in the input data and duplicate

several output values without changing the data. On the contrary, Output levels actively change information.

These easy functional block consists of a network that can study classification problems after studying enough data. ANN is useful for detecting and predicting user activity. The perceptron multilayer teaching method was chosen to identify consumer activity. The ANN algorithmic law is characterised by its potentiality to with efficiency grant and divine user activity, to perform and check efficiency for computation instead the big volume of information set. Different categorization techniques like (HMM) Hidden Andrei Markov Model, Naive Thomas Bayes, and C4.5 face provocation in the form of the interleaving events and the run time. but on the other hand, the ANN algorithmic rule is wide used for temporary relationship identification and abstraction. Thus, as we have a tends to aforesaid antecedently, along with the ANN algorithmic rule, the J48 call tree is used to beat the irrelevancy and redundancy of options which will considerably increase the machine quality and categorization errors of the algorithms, particularly ANN.

In ANN implementations, the "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. ANNs have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games, medical diagnosis and even in activities that have traditionally been considered as reserved to humans, like painting

ANNs began as an attempt to exploit the architecture of the human brain to perform tasks that conventional algorithms had had little success. They soon reoriented towards improving empirical results, mostly abandoning attempts to remain true to their biological precursors. Neurons are connected to each other in various patterns, to allow the output of some neurons to become the input of others. The network forms a directed, weighted graph.

<b>Input:</b> Binary image $I(x, y)$	
<b>Output:</b> Features of the region $F(\ell)$ , where $\ell$ is the region label	
1.	raster scan (first by column, then by row) the input image $I$
2.	<b>if</b> current pixel $I(x, y)$ is 1 <b>then</b>
2.1	check neighbor labels in the row buffer
2.2	<b>if</b> all neighbor labels are 0 <b>then</b>
2.2.1	assign a new label $\ell$ to the row buffer $R(x)$
2.2.2	initialize the region feature $F(\ell)$
2.3	<b>else if</b> only a neighbor label is not 0, others are 0 <b>then</b>
2.3.1	assign the label $\ell$ of the neighbor label to the row buffer $R(x)$
2.3.2	update the region feature $F(\ell)$ with current pixel $I(x, y)$
2.4	<b>else</b> multiple neighbor labels are not 0 <b>then</b>
2.4.1	assign the smallest label $\ell$ of neighbor labels to the row buffer $R(x)$
2.4.2	merge the region feature $F(\ell)$ with the neighbor region features
2.4.3	record the equivalence between different neighbor labels
	<b>end if</b>
3	<b>else</b> current pixel $I(x, y)$ is 0 <b>then</b>
3.1	assign 0 to the row buffer $R(x)$
	<b>end if</b>

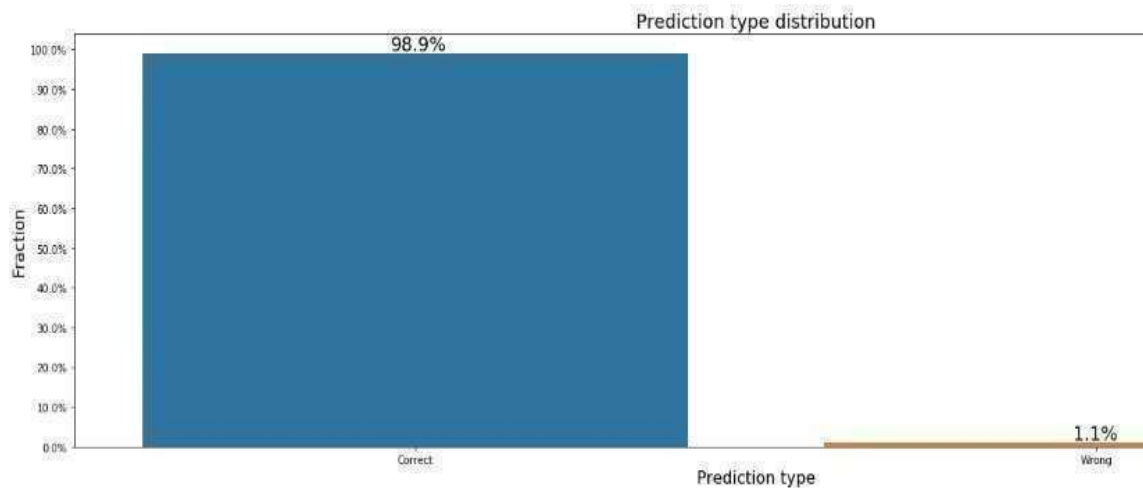
**Table 4.6(a) Algorithm of Single Neural Network**

## 4.7 Results

As the paper emphasizes on an end-to-end solution to real-time traffic sign localization and recognition, it is necessary to evaluate preprocessing, localization and classification performance. Paper shows an effective implementation of localization with preprocessing algorithms that executes in 20 ms. To evaluate the classification execution

time, we used the GPUs Nvidia GeForce GTX 1050 and Nvidia GeForce GT 1650, and CPU Intel Core i7 8500u.

One of the most efficient methods for TSR using GTSDB and GTSRB is the method using ConvNet for both localizing and classifying traffic signs. The method described in this paper shows slightly better results in both precision and performance. The developed algorithm was also tested on the video frames obtained in the streets using an Android device Nvidia Shield Tablet built in to a car. The accuracy of all methods shown in the table was obtained using the dataset GTSRB. Since, after training the model over 50,000 iterations with a epoch of 100 the prediction accuracy was found to be 98.9% which is a satisfactory value for recognition of traffic signs.



**Fig 4.7 Probability of Prediction rate**

## **CHAPTER 5**

### **CONCLUSION**

In this work, we propose a traffic sign recognition method and its resource efficient hardware implementation that processes in real-time. The proposed TSR method achieves 97.61% detection rate and 98.9% recognition rate while testing with GTSRB dataset. Our hardware implementation reduces the storage of CCL, and simplifies the HOG computation. Focussing on the need of driver's safety and safe journey to happen. The paper concludes a model which aims in recognition of road traffic signs in the real world supporting for the development of the driverless cars to reduce the no. of accidents happening around the world. The network model was developed based on artificial neural networks.

Comparisons of each stage by training and validating the dataset makes our model look good fit and helps in the output prediction. Thus by concluding that real time accuracy of 98% can be achieved for any situation in the real world scenario With image size of  $1360 \times 800$ , the processing speed is up to 135 fps. Therefore, this work is resource efficient and achieves real-time requirement.

## REFERENCES

- [1] Armingol. J, M. Mata and A. de la Escalera, vol.21, Traffic sign recognition and analysis for intelligence vehicles based on Image Vision pp. 247–258, 2003.
- [2] Aleksej Avramovic, Domen Tabernik, and Danijel Skocaj. Real time Large Scale Traffic Detection, Published on April 2018.
- [3] David Geronimo, Joan serrat, AntonioM Lopez and Raman Baldrich Traffic Sign Recognition for Computer Vision Project Based Learning. Published on 2013.
- [4] Girshick. R, Sun.J, Ren.S and Hen. K, Faster R-CNN: Towards Real Time Object Detection with Region Proposal Networks, Published on 2015.
- [5] Gil-Jimenez.P, Gomez Moreno. H, Maldonado Bascon.S, Lafuente-Arroyoand.S and Lopez-Ferreras.F ,Road-sign detection and recognition based on machine learning by support vector machines IEEE Trans. Intell. Transport., Jun. 2007.
- [6] Hasan.K, Performance analysis and improvement of optical camera communication, Applied sources, Dec 2018.
- [7] Khan. J, Adhami. K and Bhuiyan. S, Image segmentation and shape analysis for road-sign Detection, IEEE Transactions, Mar. 2011.
- [8] MohithBhairav Mahatme, Sonia Kuwelkar, Detection and Recognition of Traffic Signs based on RGB to RED Conversion and Published on May ,2017.
- [9] Mohd Khalid Hasan, Md. Shahjahal, Mostafa Zaman Chowdary, Nam Tuan Le and Yeong Min Jang Simultaneous Traffic Sign Detection and Real time communication using Dual camera in ITS. Dept of Electronics Engineering ,Kookmin University ,Seoul,South Korea. 2019.
- [10] Mostakim. M, Fleyeh. H, Biswas.R, Detection and classification of speed limit traffic signs, Proc. Computer Applications and Information Systems(WCCAIS), Jan. 2014.
- [11] Prabhu.V, Abdu Rahiman, Kiran C.G and Rajeev K Traffic Sign Detection and Recognition by SVM. published in 2009.

[12] Safat B. Wali , Majid A. Abdullah , Mahammad A. Hannan,Aini Hussain, Salina A,Samad, Pin J. Ker and Muhamad Bin Manson Vision-Based Traffic Sign Detection and Recognition Systems: Current Trends and Challenges.

[13] Wang Cong, Ma Weixin and Xiong Changzhen, Traffic Sign Recognition Algorithm Based on Deep Convolutional Neural Network, published in 2016.



## APPENDIX

### Source Code:

```
# In[1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv) import tensorflow as tf
import matplotlib.pyplot as plt import sklearn
import os import plotly
import plotly.graph_objs as go import time
import itertools import cv2
import seaborn as sns import warnings import tqdm
import math
warnings.simplefilter(action='ignore', category=FutureWarning)
get_ipython().run_line_magic('matplotlib', 'inline') plotly.offline.init_notebook_mode(True)
dataset_dir = '../'
meta_info = os.path.join(dataset_dir, 'Meta.csv') train_csv_path = os.path.join(dataset_dir,
'Train.csv') test_csv_path = os.path.join(dataset_dir, 'Test.csv')

# In[2]:
labels = ['20 km/h', '30 km/h', '50 km/h', '60 km/h', '70 km/h', '80 km/h', '80 km/h end', '100 km/h',
'120 km/h', 'No overtaking', 'No overtaking for tracks', 'Crossroad with secondary way', 'Main
road', 'Give way', 'Stop', 'Road up', 'Road up for track', 'Brock', 'Other dangerous', 'Turn left', 'Turn
right', 'Winding road', 'Hollow road', 'Slippery road', 'Narrowing road', 'Roadwork', 'Traffic light',
'Pedestrian', 'Children', 'Bike', 'Snow', 'Deer', 'End of the limits', 'Only right', 'Only left', 'Only
straight', 'Only straight and right', 'Only straight and left', 'Take right', 'Take left', 'Circle
crossroad', 'End of overtaking limit', 'End of overtaking limit for track']

# In[3]:
train_data_color = '#0f7b8e'
test_data_color = '#630f8e'

trainDf = pd.read_csv(train_csv_path)
testDf = pd.read_csv(test_csv_path)
metaDf = pd.read_csv(meta_info)
```

```

trainDf['Path'] = list(map(lambda x: os.path.join(dataset_dir,x.lower()), trainDf['Path']))
testDf['Path'] = list(map(lambda x: os.path.join(dataset_dir,x.lower()), testDf['Path']))
metaDf['Path'] = list(map(lambda x: os.path.join(dataset_dir,x.lower()), metaDf['Path']))
trainDf.sample(3)

```

# In[4]:

```

fig, axs = plt.subplots(1, 2, sharex=True, sharey=True, figsize=(25, 6))
axs[0].set_title('Train classes distribution')

axs[0].set_xlabel('Class')
axs[0].set_ylabel('Count')
axs[1].set_title('Test classes distribution')

axs[1].set_xlabel('Class')
axs[1].set_ylabel('Count')
sns.countplot(trainDf.ClassId, ax=axs[0])
sns.countplot(testDf.ClassId, ax=axs[1])
axs[0].set_xlabel('Class ID');
axs[1].set_xlabel('Class ID');

```

# In[5]:

```

trainDfDpiSubset = trainDf[(trainDf.Width < 80) & (trainDf.Height < 80)];
testDfDpiSubset = testDf[(testDf.Width < 80) & (testDf.Height < 80)];
g = sns.JointGrid(x="Width", y="Height", data=trainDfDpiSubset)
sns.kdeplot(trainDfDpiSubset.Width, trainDfDpiSubset.Height, cmap="Reds",
shade=False, shade_lowest=False, ax=g.ax_joint)
sns.kdeplot(testDfDpiSubset.Width, testDfDpiSubset.Height, cmap="Blues",
shade=False, shade_lowest=False, ax=g.ax_joint)
sns.distplot(trainDfDpiSubset.Width, kde=True, hist=False, color="r", ax=g.ax_marg_x,
label='Train distribution')

sns.distplot(testDfDpiSubset.Width, kde=True, hist=False, color="b", ax=g.ax_marg_x,
label='Test distribution')

sns.distplot(trainDfDpiSubset.Width, kde=True, hist=False, color="r", ax=g.ax_marg_y,
vertical=True)

```

```

sns.distplot(testDfDpiSubset.Height, kde=True, hist=False, color="b", ax=g.ax_marg_y,
vertical=True)

g.fig.set_figwidth(25)
g.fig.set_figheight(8)
plt.show();

# In[6]:

sns.set_style()
rows = 6
cols = 8
fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 12))
plt.subplots_adjust(left=None, bottom=None, right=None, top=0.9, wspace=None,
hspace=None)
metaDf = metaDf.sort_values(by=['ClassId']) idx = 0
for i in range(rows):
    for j in range(cols):
        if idx > 42:
            break
        img = cv2.imread(metaDf["Path"].tolist()[idx], cv2.IMREAD_UNCHANGED)
        img[np.where(img[:, :, 3]==0)] = [255,255,255,255]
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (60,60))
        axs[i,j].imshow(img)
        axs[i,j].set_facecolor('xkcd:salmon')
        axs[i,j].set_facecolor((1.0, 0.47, 0.42))
        axs[i,j].set_title(labels[int(metaDf["ClassId"].tolist()[idx])])
        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)
        idx += 1

```

```

# In[7]:
rows = 6
cols = 8+4
fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 12))
plt.subplots_adjust(left=None, bottom=None, right=None, top=0.9, wspace=None,
hspace=None)

visualize = trainDf.sample(rows*cols)
idx = 0
for i in range(rows):
    for j in range(cols):
        img = cv2.imread(visualize["Path"].tolist()[idx])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (60,60)) axs[i,j].imshow(img)
        axs[i,j].set_title(labels[int(visualize["ClassId"].tolist()[idx])])
        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)
        idx += 1

# In[8]:

img_load_size = (60,60)

zero_img = np.zeros([12,img_load_size[0], img_load_size[1], 3])
zero_label = np.zeros([12,1])

def parse_function(filename, label):
    image_string = tf.read_file(filename)

    image = tf.image.decode_jpeg(image_string, channels=3)
    # image = tf.py_func(eq, [image], image.dtype)

```

```

image.set_shape([None, None, 3])
return filename, image, label

def train_preprocess(filename, image, label):
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize_images(image, img_load_size)
return filename, image, label

def augmentate(filename, image, label):
grad = tf.random.uniform(shape=[], minval=-0.3, maxval=0.3)
dx = tf.random.uniform(shape=[], minval=-15, maxval=15, dtype=tf.int32)
dy = tf.random.uniform(shape=[], minval=-15, maxval=15, dtype=tf.int32)
image = tf.contrib.image.rotate(image, grad)
image = tf.contrib.image.translate(image, translations=[dx, dy])
return filename, image, label

def eq(img: np.ndarray):
res = img.copy()
res[:, :, 0] = cv2.equalizeHist(img[:, :, 0])
res[:, :, 1] = cv2.equalizeHist(img[:, :, 1])
res[:, :, 2] = cv2.equalizeHist(img[:, :, 2])

return res

# In[9]:
tf.reset_default_graph()
epochs = 100
batch_size = 12
prefetch_count = 1
samples_train = len(trainDf)
samples_test = len(testDf)
dataset_train = tf.data.Dataset.from_tensor_slices((trainDf['Path'], trainDf['ClassId']))
dataset_train = dataset_train.shuffle(len(trainDf['Path']))
dataset_train = dataset_train.repeat(epochs)

```

```

dataset_train = dataset_train.map(parse_function, num_parallel_calls=4)
dataset_train = dataset_train.map(train_preprocess, num_parallel_calls=4)
dataset_train = dataset_train.map(augmentate, num_parallel_calls=4)
dataset_train = dataset_train.batch(batch_size)
dataset_train = dataset_train.prefetch(prefetch_count)

dataset_iterator = tf.data.Iterator.from_structure(dataset_train.output_types,
dataset_train.output_shapes)
dataset_test = tf.data.Dataset.from_tensor_slices((testDf['Path'], testDf['ClassId']))
dataset_test = dataset_test.shuffle(len(testDf['Path']))
dataset_test = dataset_test.repeat(epochs+1)

dataset_test = dataset_test.map(parse_function, num_parallel_calls=4)
dataset_test = dataset_test.map(train_preprocess, num_parallel_calls=4)
dataset_test = dataset_test.batch(batch_size)
dataset_test = dataset_test.prefetch(prefetch_count)
train_init_op = dataset_iterator.make_initializer(dataset_train)
test_init_op = dataset_iterator.make_initializer(dataset_test)
load_filename, load_img, load_label = dataset_iterator.get_next()

# In[10]:

fig, ax = plt.subplots(ncols=8, nrows=1, figsize=(15, 6))
with tf.Session() as sess:
    sess.run(train_init_op)
    for j in range(8):
        i, l = sess.run([load_img, load_label])
        i = (i[0]*255).astype(np.uint8)
        ax[j].imshow(i)
        ax[j].set_title(labels[l[0]]

```

```

# In[11]:

dp_rate = tf.placeholder(dtype=tf.float32, shape=[], name='dp_rate')

img_placeholder = tf.placeholder(shape=[None, 60,60,3], dtype=tf.float32,
name='img_placeholder')

label_placeholder = tf.placeholder(shape=[None, 1], dtype=tf.int64,
name='label_placeholder')

manual_load = tf.placeholder(dtype=tf.bool, shape=[], name='manual_load_placeholder')

# inp = net = tf.cond(pred=manual_load, true_fn=lambda : img_placeholder,
false_fn=lambda : load_img, name='network_start')

# label = tf.cond(pred=manual_load, true_fn=lambda : label_placeholder,
false_fn=lambda : load_label, name='label')

inp = net = tf.cond(manual_load, lambda: img_placeholder, lambda: load_img)
label = load_label

conv1 = net = tf.layers.conv2d(inputs=net, filters=16, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

conv2 = net = tf.layers.conv2d(inputs=net, filters=32, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

conv3 = net = tf.layers.conv2d(inputs=net, filters=32, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

```

```

conv4 = net = tf.layers.conv2d(inputs=net, filters=64, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)net = tf.layers.max_pooling2d(inputs=net,
pool_size=(2,2), strides=(2,2))

conv5 = net = tf.layers.conv2d(inputs=net, filters=64, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

conv6 = net = tf.layers.conv2d(inputs=net, filters=128, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

conv5 = net = tf.layers.conv2d(inputs=net, filters=256, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)

conv6 = net = tf.layers.conv2d(inputs=net, filters=400, kernel_size=(3,3), strides=(1,1),
activation=tf.nn.leaky_relu)

net = tf.layers.batch_normalization(inputs=net)
flatten1 = net = tf.layers.flatten(inputs=net)

dp1 = net = tf.layers.dropout(inputs=net, rate=dp_rate)
dense1 = net = tf.layers.dense(inputs=net, units=256)
logits = tf.layers.dense(inputs=net, units=43)
pred_classes = tf.argmax(logits, axis=1)

pred_probab = tf.nn.softmax(logits)

acc, acc_op = tf.metrics.accuracy(labels=label, predictions=pred_classes)
end_loss = tf.losses.sparse_softmax_cross_entropy(logits=logits, labels=label)
loss = end_loss

label_transpose = tf.transpose(label)

```



```

correct_prediction = tf.equal(pred_classes, label_transpose)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

confusion_matrix_op = tf.confusion_matrix(labels=label, predictions=pred_classes,
num_classes=43)

opt = tf.train.AdamOptimizer(learning_rate=0.0001).minimize(loss)

# In[12]:

config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.4
sess = tf.Session(config=config)

saver = tf.train.Saver()
sess.run(tf.global_variables_initializer())

# irr.load_weights('inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5')
train_history = {'loss':[], 'acc':[], 'val_loss':[], 'val_acc':[]}

best_acc = 0

for e in range(epochs):

    epoch_history = {'loss':[], 'acc':[], 'val_loss':[], 'val_acc':[]}

    sess.run(train_init_op)

    for i in tqdm.tqdm_notebook(range(samples_train//batch_size), ascii=True, desc='Train
epoch { }'.format(e)):

        _, _loss, _acc, mn = sess.run([opt, loss, accuracy, inp], feed_dict={dp_rate: 0.3,
manual_load: False, img_placeholder: zero_img, label_placeholder: zero_label})

        #print(np.mean(mn))
        epoch_history['loss'].append(_loss)
        epoch_history['acc'].append(_acc)

    sess.run(test_init_op)

```

```

for i in tqdm.tqdm_notebook(range(samples_test//batch_size), ascii=True, desc='Test
epoch {}'.format(e)):

    _loss, _acc = sess.run([loss, accuracy], feed_dict={dp_rate: 0, manual_load: False,
img_placeholder: zero_img, label_placeholder: zero_label})

    epoch_history['val_loss'].append(_loss)
    epoch_history['val_acc'].append(_acc)
    train_history['loss'].append(np.mean(epoch_history['loss']))
    train_history['acc'].append(np.mean(epoch_history['acc']))
    train_history['val_loss'].append(np.mean(epoch_history['val_loss']))
    train_history['val_acc'].append(np.mean(epoch_history['val_acc']))

    print("***EPOCH SUMMARY*** Loss: {} Acc: {} | Test Loss: {} Test Acc
    {}".format(train_history['loss'][-1], train_history['acc'][-1],
    train_history['val_loss'][-1], train_history['val_acc'][-1])) if
    train_history['val_acc'][-1] > best_acc:

    best_acc = train_history['val_acc'][-1] save_path = saver.save(sess, "./model.ckpt") print("Model
    saved in path: %s" % save_path)

```

```

# In[13]:

titlefont = dict(family='Courier New, monospace', size=18, color='#7f7f7f')

layout = go.Layout(title='Traing & Test loss', xaxis=dict(title='Epoch', titlefont=titlefont),
yaxis=dict(title='Loss', titlefont=titlefont))

fig = go.Figure(data=[go.Scatter(y=train_history['loss'], name='Train loss'),
go.Scatter(y=train_history['val_loss'], name='Test loss')], layout=layout)

plotly.offline.iplot(fig)

layout = go.Layout(title='Traing & Test accuracy', xaxis=dict(title='Epoch',
titlefont=titlefont),

yaxis=dict(title='Accuracy', titlefont=titlefont))

fig = go.Figure(data=[go.Scatter(y=train_history['acc'], name='Train accuracy'),
go.Scatter(y=train_history['val_acc'], name='Test accuracy')], layout=layout)

plotly.offline.iplot(fig)

```

```

# In[14]:

saver.restore(sess, "./model.ckpt")
sess.run(test_init_op)
confusion_matrix = np.zeros([43,43])
test_analys = trainDf.copy()
predictions = []

probabilities = []
analys = []

for i in tqdm.tqdm_notebook(range(samples_test//batch_size), ascii=True, desc='Test best
model'):

    _files, _predictions, _probas, _gts, _cm = sess.run([load_filename, pred_classes,
pred_probas, load_label, confusion_matrix_op], feed_dict={dp_rate: 0, manual_load:
False, img_placeholder: zero_img, label_placeholder: zero_label})

    confusion_matrix += _cm
    for i in range(batch_size):

        sample_info = {'image': _files[i].decode(), 'prediction': int(_predictions[i]), 'gt':
int(_gts[i]), 'gt_probas': _probas[i][_gts[i]],

        'prediction_probas': _probas[i][_predictions[i]], 'prediction_type': 'Correct' if _gts[i] ==
_predictions[i] else 'Wrong'}

        for cls_id, j in enumerate(_probas[i]):
            sample_info['prob_{}'.format(cls_id)] = j

        analys.append(sample_info)

    analys_df = pd.DataFrame(analys)

```

```

In[15]:

analys_df.sample(4)
# In[16]:

rows = 3

cols = 4

fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 8))
visualize = trainDf.sample(rows*cols)

analys_df_copy = analys_df.copy()
analys_df_copy = analys_df_copy.sample(frac=1)
idx = 0

for i in range(rows):

    for j in range(cols):

        img = cv2.imread(analys_df_copy.iloc[idx]['image'])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (100, 100))

        gt = analys_df_copy.iloc[idx]['gt']

        pred = analys_df_copy.iloc[idx]['prediction']
        axs[i,j].imshow(img)

        axs[i,j].set_title('Predicted: {} \n Ground truth {}'.format(labels[pred], labels[gt]),
        fontsize=14)

        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)
        idx += 1

fig.suptitle("Random prediction", fontsize=30, y=2.1, x=0.515);

```

```

plt.subplots_adjust(left=None, bottom=None, right=0.9, top=1.9, wspace=None,
hspace=None)

# In[17]:

rows = 3

cols = 4

fig, axs = plt.subplots(rows, cols, sharex=True, sharey=True, figsize=(25, 8))
visualize = trainDf.sample(rows*cols)

analys_df_copy = analys_df[analys_df['prediction_type'] == 'Wrong'].copy()
analys_df_copy = analys_df_copy.sample(frac=1)

idx = 0

for i in range(rows):

    for j in range(cols):

        img = cv2.imread(analys_df_copy.iloc[idx]['image'])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (100, 100))

        gt = analys_df_copy.iloc[idx]['gt']

        pred = analys_df_copy.iloc[idx]['prediction']

        axs[i,j].imshow(img)

        axs[i,j].set_title('Predicted: { }\nGround truth { }'.format(labels[pred], labels[gt]),
        fontsize=14)

        axs[i,j].get_xaxis().set_visible(False)
        axs[i,j].get_yaxis().set_visible(False)

```

```

idx += 1

fig.suptitle("Wrong prediction", fontsize=30, y=2.1, x=0.515);

plt.subplots_adjust(left=None, bottom=None, right=0.9, top=1.9, wspace=None,
hspace=None)

# In[18]:

fig, axs = plt.subplots(1, 1, sharex=False, sharey=True, figsize=(25, 7))
px = sns.countplot(x='prediction_type', data=analys_df, ax=axs)
axs.set_title('Prediction type distribution', fontsize=18)
axs.set_xlabel('Prediction type', fontsize=16)

axs.set_ylabel('Fraction', fontsize=16);
total = analys_df.shape[0]

for idx, p in enumerate(px.patches):

px.annotate('{:.1f}%'.format(p.get_height()/total*100), (p.get_x()+0.365,
p.get_height()+100), fontsize=18)




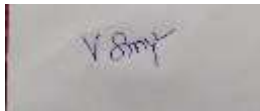
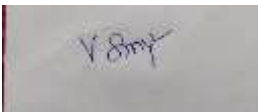
px.yaxis.set_ticks(np.linspace(0, total, 11))
px.set_yticklabels(map('{:.1f}%'.format, 100*px.yaxis.get_majorticklocs()/total));

```

## Format – I

<b>SRM INSTITUTE OF SCIENCE AND TECHNOLOGY</b> (Deemed to be University u / s 3 of UGC Act, 1956)		
<b>Office of Controller of Examinations</b>		
REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION / PROJECT REPORT FOR UG / PG PROGRAMMES (To be attached in the dissertation / project report)		
1	Name of the Candidate (IN BLOCK LETTERS)	<b>K THIRUPATHI, V KRANTHI, R PRANAV RAJU</b>
2	Address of Candidate	Plot No. 5-2-86/302, Ambica Apartments, Tirumala Nagar, Moula-ali, Hyderabad – 500040.  <b>Mobile Number:</b> 8317550785, 9677197443, 9182786989
3	Registration Number	RA1711003020456, RA1711003020467, RA1711003020473
4	Date of Birth	14/05/1999, 31/07/1999, 19/05/1999
5	Department	Computer Science Engineering
6	Faculty	Faculty of Engineering and Technology
7	Title of the Dissertation / Project	TRAFFIC SIGN RECOGNITION USING DEEP LEARNING BY NEURAL NETWORKS
8	Whether the above project / dissertation is done by	<b>Group</b>  <b>a)</b> If the project / dissertation is done in group, then how many students together completed the project : <b>3</b>  <b>b)</b> Mention the Name & Register number of other candidates : Thirupathi K [RA1711003020456] Kranthi V [RA1711003020467] Pranav R [RA1711003020473]
9	Name and address of the Supervisor / Guide	<b>Ms.V.Surya</b> Assistant Professor, SRM Institute of Science and Technology, Ramapuram Campus, Chennai – 89. <b>Mail ID :</b> suryav@srmist.edu.in <b>Mobile Number :</b> 7358586700
	Name and address of the Co-Supervisor /	<b>N/A</b>



10	Software Used	Turnitin		
11	Date of Verification	25/05/2021		
12	<b>Plagiarism Details : (to attach the final report from the software)</b>			
<b>Chapter</b>	<b>Title of the Report</b>	<b>Percentage of similarity index (including self citation)</b>	<b>Percentage of similarity index (Excluding self citation)</b>	<b>% of plagiarism after excluding Quotes, Bibliography, etc.,</b>
<b>1</b>	TRAFFIC SIGN RECOGNITION USING DEEP LEARNING BY NEURAL NETWORKS	2%	<2%	<2%
<b>Appendices</b>				
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
 <b>V. KRANTHI</b>  <b>R. PRANAV</b>  <b>K. THIRUPATHI</b> <b>Signature of the Candidate</b>		 <b>Ms.V.Surya</b> <b>Assistant Professor(O.G)</b> <b>Department of Computer Science and Engineering</b>		
 <b>Ms.V.Surya</b> <b>Assistant Professor</b> <b>Department of Computer Science and Engineering</b>		<b>Name &amp; Signature of the Co-Supervisor / Co-Guide</b>		
<p style="text-align: center;">Dr. K. RAJA, M.E., Ph.D.  Professor and Head  Computer Science and Engineering  SRM Institute of Science and Technology  Chennai</p>				

<div> <div>1</div> <div>ORIGINALITY REPORT</div> </div>			
2%	1%	1%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
PRIMARY SOURCES			
1	medium.com Internet Source	<1%	
2	www.mdpi.com Internet Source	<1%	
3	becominghuman.ai Internet Source	<1%	
4	Xiong Changzhen, Wang Cong, Ma Weixin, Shan Yanmei. "A traffic sign detection algorithm based on deep convolutional neural network", 2016 IEEE International Conference on Signal and Image Processing (ICSIP), 2016 Publication	<1%	
5	towardsdatascience.com Internet Source	<1%	
6	roohit.com Internet Source	<1%	
7	Luis Rodriguez-Benitez, Carlos Córdoba Ruiz, Luis Cabañero Gómez, Ramón Hervás, Luis Jimenez-Linares. "An Internet of Agents Architecture for Training and Deployment of	<1%	
	Deep Convolutional Models", Journal of Signal Processing Systems, 2020 Publication		
8	"Table of contents", 2017 International Conference on Computing Methodologies and Communication (ICCMC), 2017 Publication	<1%	
9	Aleksei Avramovic, Domen Tabernik, Danijel Skocaj. "Real-time Large Scale Traffic Sign Detection", 2018 14th Symposium on Neural Networks and Applications (NEUREL), 2018 Publication	<1%	
10	Alexander Shustanov, Pavel Yakimov. "CNN Design for Real-Time Traffic Sign Recognition", Procedia Engineering, 2017 Publication	<1%	
11	Submitted to Bannari Amman Institute of Technology Student Paper	<1%	
12	Submitted to October University for Modern Sciences and Arts (MSA) Student Paper	<1%	
13	www.ijcaonline.org Internet Source	<1%	
<div> <div>Exclude quotes</div> <div>On</div> <div>Exclude matches &lt; 10 words</div> <div>Exclude bibliography</div> <div>On</div> </div>			