TRAFFIC SIGN RECOGNITION SYSTEM FOR AUTONOMOUS VEHICLE

Project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Saransh Rohilla (191023)

Sannidhya Yadav (191008)

UNDER THE GUIDANCE OF

Dr Emjee Puthooran

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

May 2023

TABLE OF CONTENTS

DECLARATION

We hereby declare that the work reported in the B.Tech Project Report entitled **"TRAFFIC SIGN RECOGNITION SYSTEM FOR AUTONOMOUS VEHICLE"** submitted at Jaypee University of Information Technology, Waknaghat, India is an authentic record of our work carried out under the supervision of Dr Emjee Puthooran, Associate Professor. We have not submitted this work elsewhere for any other degree or diploma.

Saransh Rohilla

191023

Saanidhya Yadav

191008

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

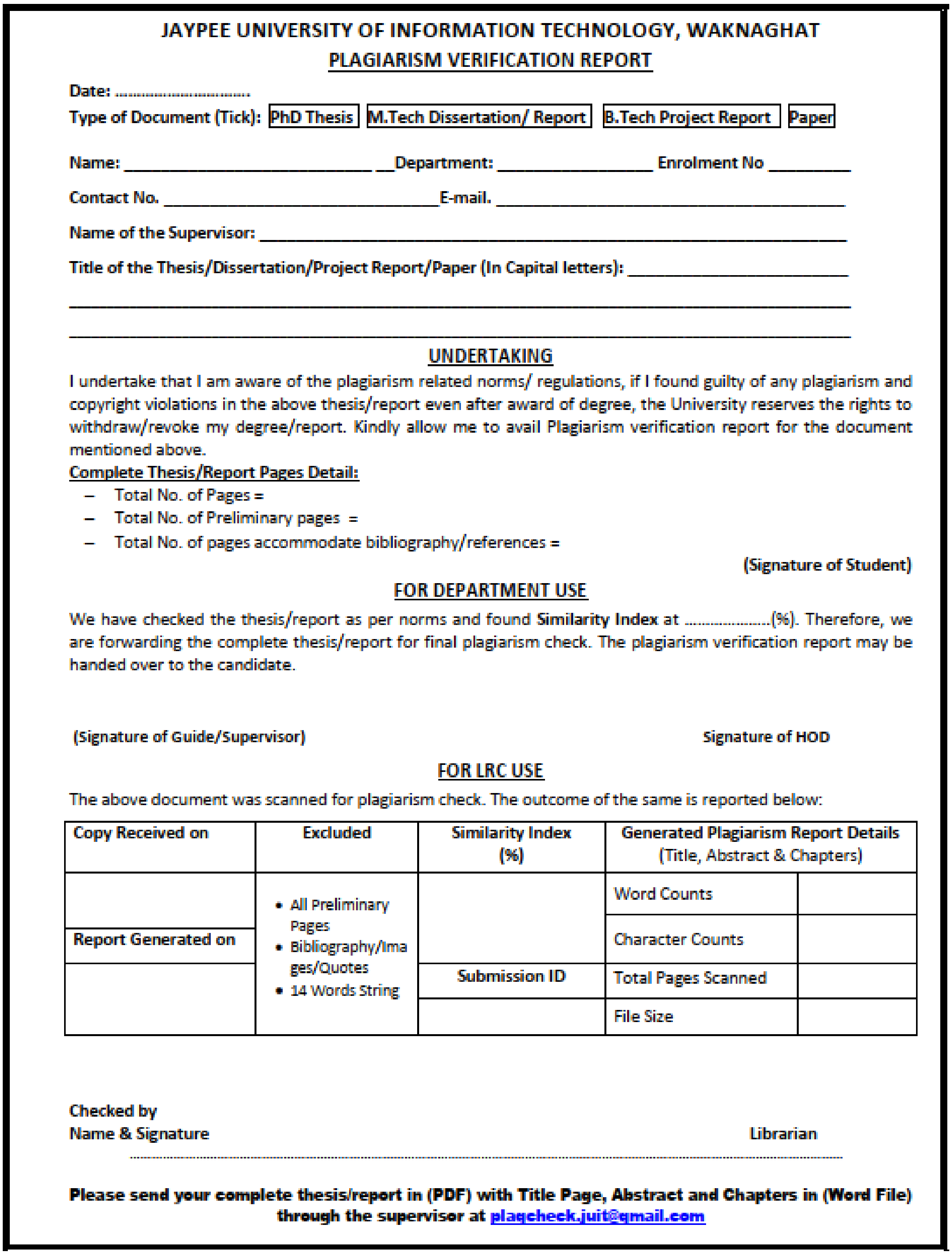
Dr Emjee Puthooran

Associate Professor

Jaypee University of Information Technology, Waknaghat

Date:

PLAGRISM CERTIFICATE



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ____________________ __Department: ______________ Enrolment No ________

Contact No. ______________________E-mail. _____________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): __________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

– Total No. of Pages =
– Total No. of Preliminary pages =
– Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature Librarian

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

I would want to convey my heartfelt gratitude to Dr Emjee Puthooran, my guide, for his invaluable advice and assistance in completing my project. He was there to assist me every step of the way, and his motivation is what enabled me to accomplish my task effectively. I would also like to thank all the other supporting personnel who assisted me by supplying the equipment that was essential and vital, without which I would not have been able to perform efficiently on this project.

I would also want to thank the Jaypee University of Information Technology for accepting my project in my desired field of expertise. I would also like to thank my friends and parents for their support and encouragement as I worked on this assignment.

LIST OF ACRONYMS AND ABBREVIATIONS

CNN	Convolutional Neural Network
ML	Machine Learning
TSR	Traffic Sign Recognition
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
HOG	Histogram of Oriented Gradients
LBP	Local Binary Patterns
DCT	Discrete Cosine Transform
ELM	Extreme Learning Machine
GTSRB	German Traffic Sign Recognition Benchmark
BTSC	Belgium Traffic Sign Classification
TSDR	Traffic sign detection and recognition
ReLU	Rectified Liner Unit

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Automated driving and driver assistance systems heavily rely on accurate traffic sign recognition. This involves two steps: detection and classification, which require sophisticated vision algorithms due to the diverse visual characteristics of traffic sign images. Researchers are actively working on developing novel methods to tackle this challenging problem. Traffic sign recognition is crucial for self-driving cars as it enables them to understand the traffic environment and make informed decisions based on road signs and markings.

The German Traffic Sign Recognition Benchmark (GTSRB) and the German Traffic Sign Detection Benchmark (GTSDB) have provided standardized datasets for evaluating and comparing traffic sign recognition and detection algorithms. These benchmarks have facilitated the development of more accurate and robust systems and have accelerated progress in the field through international competitions and challenges.

Deep learning, particularly convolutional neural networks (CNNs), has revolutionized traffic sign detection. CNNs automatically extract relevant features from input images, eliminating the need for manual feature extraction. The use of CNNs in traffic sign recognition has become a popular research focus, as they are effective in image classification tasks. CNNs consist of convolutional layers, pooling layers, and fully-connected layers, which collectively extract features, reduce spatial dimensions, and classify images.

The Le-Net model, modified for traffic sign recognition, achieved 95.08% accuracy, while the Modified Le-Net model achieved 97.86% accuracy. Different optimizers were used for training these models, and the choice of optimizer depended on the model's architecture. RMSProp performed best for the Le-Net model, preventing overshooting and providing smooth updates. Nadam performed best for the Modified Le-Net model, converging quickly to the optimal point. It is important to experiment with different optimizers and hyperparameters to find the best combination for a specific model architecture.

The learning rate is another critical aspect in training neural networks. A learning rate of 0.001 was found to provide better results than 0.01. A higher learning rate can cause overshooting and instability, while a lower learning rate enables more stable convergence and accurate predictions. The optimal learning rate depends on the problem and network architecture, and it is crucial to experiment with different values.

Regularization techniques, such as dropout, can improve the generalization performance of neural networks by reducing overfitting. The choice of dropout percentage should be determined through experimentation. Pooling methods like max pooling and average pooling have their strengths and weaknesses, with max pooling being effective for identifying important features and average pooling for identifying overall trends. The optimal pooling method depends on the specific requirements of the task and the characteristics of the input data.

CHAPTER – 1

INTRODUCTION

1.1 Introduction

Automated driving and driver assistance systems heavily rely on the ability to recognize traffic signs, which is a challenging task for computers due to the diverse visual characteristics of traffic sign images, such as partial opacity, varying angles, lighting conditions, and weather. To identify traffic signals in an image, the common approach involves two steps: detection and classification, both of which require sophisticated vision algorithms. Consequently, researchers are actively working on developing novel methods or modifying existing ones to tackle this demanding problem. Traffic sign recognition system is crucial for self-driving cars as it enables them to perceive and understand the traffic environment and make decisions accordingly. In the absence of human intervention, self-driving cars rely entirely on sensors and cameras to detect, classify, and respond to traffic signs, road markings, and other objects.

The ability to accurately recognize and interpret traffic signs is essential for self-driving cars to navigate safely and efficiently. For instance, self-driving cars need to identify and respond to speed limit signs, stop signs, no-entry signs, and other regulatory signs. They also need to detect warning signs, such as pedestrian crossings, school zones, and construction zones, and adjust their speed and behaviour accordingly. Moreover, the traffic sign recognition system can assist self-driving cars in making informed decisions based on the road network and traffic conditions. By analysing data collected from cameras and sensors, self-driving cars can optimize their route and speed, avoid congestion, and reduce travel time.

Autonomous vehicles, also known as self-driving cars, have become increasingly prevalent in recent times. They are designed to function without human intervention, relying on sensors, cameras, and software to perceive their surroundings and determine their route. The traffic sign recognition system is a vital component of this technology, enabling autonomous vehicles to detect and react to road signs.

Traffic sign recognition is a critical technology for self-driving cars. It enables these vehicles to understand the rules of the road and respond accordingly, making them safer and more efficient. With traffic sign recognition, self-driving cars can detect and interpret a wide range of traffic signs, including speed limits, stop signs, yield signs, and more.

Here are some of the key reasons why traffic sign recognition is so important for self-driving cars:

1. Safety: By recognizing traffic signs, self-driving cars can ensure that they are driving safely and within the law. For example, a self-driving car that can recognize a stop sign can slow down and come to a complete stop, preventing accidents, and improving overall safety on the road.

2. Efficiency: Traffic sign recognition can also help self-driving cars operate more efficiently. By understanding speed limits and other traffic regulations, self-driving cars can adjust their speed and driving behaviour to optimize fuel consumption and reduce travel times.

3. Compliance: Traffic sign recognition is also important for regulatory compliance. Self-driving cars must adhere to the same traffic laws as human drivers, and failure to recognize and obey traffic signs can result in legal issues and penalties.

While traffic sign recognition is a critical technology for self-driving cars, there are also several challenges that must be overcome in order to make it effective. Here are some of the key challenges in traffic sign recognition for self-driving cars:

1. Environmental Factors: Traffic signs can be affected by a variety of environmental factors, such as weather, lighting conditions, and even vandalism. Self-driving cars must be able to recognize signs even in difficult conditions, which can be a challenging task.

2. Variability: Traffic signs can also vary in appearance, depending on factors such as location, design, and age. Self-driving cars must be able to recognize a wide range of sign variations to operate effectively.

3. Speed: Traffic sign recognition must be fast and accurate in order to keep up with the pace of driving. Self-driving cars must be able to recognize and respond to signs quickly in order to ensure safety and compliance.

4. Integration: Finally, traffic sign recognition must be integrated with other autonomous vehicle technologies, such as object detection and path planning. This integration can be complex and requires careful coordination and engineering.

To overcome these challenges and enable effective traffic sign recognition for self-driving cars, a variety of technologies are used. These include:

1. Machine Learning: ML algorithms are used to train self-driving car systems to recognize and interpret traffic signs. These algorithms use large datasets of images of traffic signs to learn patterns and features that can be used to identify signs in the real world.

2. Computer Vision: Computer vision technologies, such as image processing and object detection, are used to analyse images of traffic signs and extract relevant features. These features can be used to identify the type of sign and its location in the environment.

3. Deep Learning: Deep learning algorithms, such as CNNs, are used to process images and extract features for traffic sign recognition. These algorithms are capable of learning complex patterns and features that can be used to identify signs in a wide range of conditions.

4. Sensor Fusion: Sensor fusion technologies, such as lidar and radar, are used in conjunction with cameras and other sensors to provide a more comprehensive view of the environment around the self-driving car. By combining data from multiple sensors, self-driving cars can better recognize and respond to traffic signs, as well as other objects and obstacles on the road.

5. Localization: Localization technologies, such as GPS and map data, are used to help self-driving cars understand their position and navigate the road. This information can be used in conjunction with traffic sign recognition to ensure that the self-driving car is complying with traffic laws and driving safely.

6. Artificial Intelligence (AI): AI technologies are used to enable self-driving cars to make decisions based on the data they receive from sensors and other technologies. This includes recognizing and responding to traffic signs, as well as identifying potential hazards on the road and making decisions about how to navigate around them.

Traffic sign recognition is a critical technology for self-driving cars, with a wide range of applications. Here are some of the key ways in which traffic sign recognition is used in autonomous vehicles:

1. Speed Limit Recognition: Self-driving cars can use traffic sign recognition to identify and respond to speed limit signs. By adjusting their speed to comply with speed limits, self-driving cars can improve safety and efficiency on the road.

2. Stop Sign Recognition: Self-driving cars can also use traffic sign recognition to identify and respond to stop signs. By coming to a complete stop at stop signs, self-driving cars can prevent accidents and ensure compliance with traffic laws.

3. Yield Sign Recognition: Yield signs indicate that a driver should give way to other vehicles or pedestrians. Self-driving cars can use traffic sign recognition to identify and respond to yield signs, ensuring safety and compliance on the road.

4. Lane Marking Recognition: Traffic sign recognition can also be used to identify and follow lane markings on the road. This can help self-driving cars navigate complex roadways and stay within the appropriate lanes.

5. Traffic Light Recognition: Traffic sign recognition can be used to identify and respond to traffic lights, enabling self-driving cars to stop at red lights and proceed at green lights. This can improve safety and efficiency at intersections and other traffic control points.

Traffic sign recognition is a critical technology for self-driving cars. By enabling autonomous vehicles to recognize and respond to traffic signs, self-driving cars can improve safety, efficiency, and compliance on the road. While there are several challenges associated with traffic sign recognition, a variety of technologies, including machine learning, computer vision, sensor fusion, localization, and AI, are being used to overcome these challenges and enable effective traffic sign recognition for self-driving cars. With further development and refinement of these technologies, self-driving cars have the potential to revolutionize transportation and make our roads safer and more efficient.

Prior to the release of the GTSRB in 2011 [1] and the GTSDB in 2013 [2], there was no publicly available dataset for evaluating traffic sign recognition and detection systems. Therefore, it was challenging to compare different methodologies. However, with the release of these benchmarks, researchers can now evaluate and compare their algorithms using the same standardized datasets. The availability of standardized datasets such as GTSRB and GTSDB has allowed researchers to assess the performance of their traffic sign recognition and detection algorithms using the same test data, providing a fair and reliable means of comparing different methodologies. Furthermore, these datasets have facilitated the development and testing of new approaches and algorithms, ultimately leading to more accurate and robust systems. The adoption of GTSRB and GTSDB benchmarks has been widespread in the research community and has been used as the foundation for several international competitions and challenges in traffic sign recognition and detection. These events have served to accelerate progress in the field and have fostered the creation of innovative and novel solutions.

The emergence and rapid development of deep learning have revolutionized the process of traffic sign detection. Deep neural networks have gained popularity among researchers, eliminating the need for manual feature extraction and annotation. Instead, the built-in network can automatically extract relevant features from input images, even for complex images. These features can then be used for target classification [3]. The advancements in deep learning have played a crucial role in accelerating the development of traffic sign recognition systems. For example, Bouti *et al.* [4] adjusted the Le-Net network to achieve good classification results on the GTSRB dataset. In another study, a neural network combined with a hinge loss function and a loss of function was used to recognize traffic signs [5].

CNN has become a popular research focus in traffic sign recognition as academics devote more attention to this area [6], [7], [8]. In computer vision, CNN has become one of the most frequently used models for image classification. A CNN typically comprises three fundamental components: a convolutional layer, a pooling layer, and a fully-connected layer.

The convolutional layer is an essential element of a convolutional neural network (CNN) [9]. This layer applies a convolution kernel to a section of the input image with a predetermined step size, resulting in a two-dimensional feature map. This process increases the image's dimensionality, enabling the extraction of high-dimensional features. The convolutional layer also provides translation invariance, allowing the network to detect features at various locations in the image. The pooling layer is another key component of the CNN, which reduces the size of the input image while preserving important information to accelerate network training. Various pooling techniques such as maximum pool, average pool, and random pool can be used for this purpose. Regardless of the technique, the primary goal of pooling is to decrease the spatial feature dimension, reduce the system's workload, and speed up the network training rate. Finally, the CNN's last layer typically consists of one or more fully-connected layers. Its role is to transform the high-dimensional feature information extracted into a one-dimensional feature map, which is then used to classify the image. The final fully connected layer outputs the classification results.

1.2 Problem Statement

Traffic sign recognition is an essential component of modern autonomous vehicles, where a vehicle is equipped with sensors and cameras that help it perceive its surroundings and make decisions based on that perception. The traffic sign recognition system is designed to detect, classify, and interpret traffic signs such as speed limits, stop signs, and warning signs, among others. However, there are several challenges and problems that can arise in traffic sign recognition systems, making it difficult to achieve accurate and reliable results. In this article, we will discuss some of the most common problems encountered in traffic sign recognition systems.

1. Variations in lighting conditions: One of the most significant challenges in traffic sign recognition is variations in lighting conditions. Traffic signs can be affected by different lighting conditions, such as glare, shadows, and low light conditions, which can make it difficult for the system to recognize them accurately. For example, a traffic sign that is well-lit during the day may appear significantly darker and harder to see at night, making it harder for the system to detect and recognize it. Similarly, a sign that is located in the shade may appear differently than a sign located in direct sunlight, making it difficult for the system to recognize it accurately.

2. Occlusions and obstructions: Another common problem in traffic sign recognition is occlusions and obstructions. Obstructions such as trees or other vehicles can obstruct the view of traffic signs, making them difficult to detect. Additionally, signs can be partially or fully occluded by other objects on the road, making it harder for the system to recognize them.

3. Variation in sign shape and size: Traffic signs can vary in shape and size depending on the location, which can make it difficult for the system to detect them accurately. For example, a stop sign in one country may be a different shape and color than a stop sign in another country, making it challenging for the system to recognize it accurately.

4. Changes in weather conditions: Weather conditions such as rain, snow, or fog can significantly impact the performance of traffic sign recognition systems. For example, snow can cover the traffic signs, making them difficult to detect, while fog can reduce visibility, making it harder for the system to recognize the signs accurately.

In recent years, there has been a rapid growth in the development of self-driving vehicles, which has led to advancements in technologies such as LiDAR sensors and cameras. To process the input data from these sensor fields, computer vision has shown significant growth. Deep neural networks have been implemented for object detection and recognition, but they require substantial computation power for learning and prediction. Researchers worldwide are striving to develop more efficient algorithms that can provide better accuracy while requiring less computation.

In the domain of traffic sign recognition, the most used model is the Le-Net model. Although this model is relatively simple, it suffers from problems such as over-fitting and difficulty in learning complex shapes due to the limited number of convolutional layers, which results in fewer feature maps. Our project aims to contribute to the development of more efficient algorithms that can provide better accuracy in traffic sign recognition while overcoming the limitations of current models.

1.3 Objective

- The project involves a thorough literature review.
- The aim is to develop a more efficient and accurate algorithm for traffic sign recognition.

- The project will address the limitations of current models, including over-fitting and difficulty in recognizing complex shapes due to a limited number of convolutional layers.
- All hyper-parameters of the CNN model will be studied in-depth.
- The project will contribute to the broader research efforts in computer vision, specifically in the area of object detection and recognition.
- Novel approaches and techniques will be explored to improve traffic sign recognition.

1.4 Methodology

The purpose of this experiment is to investigate the impact of different optimizers, pooling layers, and learning rates on the accuracy of traffic sign classification models. To achieve this, we will use the GTSRB dataset, which contains 43 different classes of traffic signs, with a total of 39,209 training images and 12,630 testing images.

We will begin by pre-processing the dataset, which will involve resizing the images to gray scale, normalizing the pixel values, and applying histogram equalization. This step will ensure that the data is in a suitable format for the models to learn from.

Next, we will use the Le-Net model, which is a well-known CNN architecture for image classification. We will experiment with four different optimizers - Adam, Nadam, RMSprop, and AdaMax, and two different pooling layers - Average and Max, on the Le-Net model. We will also use two different learning rates for the Le-Net model.

We will train and test each model configuration for 25 epochs and evaluate their performance based on their accuracy in predicting the class of a random image from the testing dataset. To ensure that the results are consistent, we will repeat each experiment five times and take the average accuracy as the result. Furthermore, we will also use a modified version of the Le-Net model, which will have one pooling layer and one learning rate. We will experiment with three different optimizers - Adam, RMSprop, and AdaMax, on the modified Le-Net model.

Finally, we will compare the results of all model configurations and determine which optimizer, pooling layer, and learning rate combination achieves the highest accuracy in classifying traffic signs.

Overall, this experimental design will provide insight into the impact of different model configurations on the accuracy of traffic sign classification models. The findings of this study could be useful in enhancing road safety by improving the accuracy of traffic sign classification models.

1.5 Organisation

The project is organized into several chapters, starting with the Introduction in Chapter 1, which provides a brief overview of the topic and its relevance. This chapter also covers the problem statement, methodology, and objectives of the study.

Chapter 2, titled Literature Survey, reviews the previous studies conducted in this field, with a focus on Machine Learning, Datasets, Le-Net models, and Neural networks.

Chapter 3, the Methodology, explains the project's methodology in detail, including the pre-processing phase, the training phase, and the testing phase. It also includes a thorough discussion of the dataset, experimentation, and the results of the project.

Chapter 4, Results and Discussion, presents the findings of the experiment and provides a detailed discussion of the results. This chapter also includes a comparison of the different model configurations and discusses the impact of optimizers, pooling layers, and learning rates on the accuracy of the models.

Finally, all the sources, including research studies, datasets, algorithms, and others, are referenced in the last section of the project.

CHAPTER - 2

LITERATURE REVIEW

2.1 Literature Review

There are two main approaches for solving the problem of recognizing road signs: color-based and Gray-scale based methods. Color-based recognition aims to reduce false positives by using color information [10]-[18], while Gray-scale recognition focuses on the object's shape [19]-[22]. Recent studies have combined both approaches to improve detection rates. For instance, in [23], a threshold is applied to a hue, saturation, and value representation of the image to identify regions likely to contain a traffic sign. However, this method may detect objects in the background that share similar colors with traffic signs, which is why size and aspect ratio heuristics are used to reduce false alarms. Once the regions are normalized to a standard size, a linear support vector machine (SVM) is used for coarse classification, followed by a fine classification using an SVM with Gaussian kernels. However, color information is subject to camera type, lighting conditions, and sign aging, which complicates the recognition process. To address this, de la Escalera et al. [24] apply an enhancement step before thresholding color values. After removing non-sign regions using size heuristics, the authors use a combination of color information, gradient, and distance images to eliminate regions with a low probability of containing a traffic sign. Final classification is performed using a neural network. Some recent studies have focused on improving the final classification step. For example, Paclik et al. [25] propose a road-sign data representation based on extending the normalized cross-correlation approach to a similarity based on individual matches in a set of local image regions.

Xie et al. [26] use Histogram of Oriented Gradient (HOG) descriptors trained with one-versus-all Support Vector Machines (SVMs) for each class. The Forest Error Correcting Output Code (FECOC) classifiers were found to have high performance rates in [27]. Multilayer Perceptron (MLP) models were also shown to produce high accuracy rates in [28] and [29]. Additionally, low false positive rates were achieved in [30] for recognizing characters in speed limit signs. In [31], the neural network performance was improved by preselecting color-shape features using Principal Component Analysis (PCA) and Fisher's Linear Discriminant.

The GTSRB dataset has been employed in numerous research studies to devise algorithms for traffic sign recognition and classification. For example, Persson S [32] utilized a pre-trained VGG16 network to categorize traffic signs. Another study by R. Rajesh, K. Rajeev, K. Suchithra, V.P. Lekhesh, V. Gopakumar, and N.K. Ragesh [33] proposed the Coherence Vector of Oriented Gradients (CVOG) and neural network to recognize and classify traffic signs. They conducted various experiments using CVOG characteristics and a combination of other features.

Dongfang et al. proposed a CNN based on GoogleLeNet to recognize and classify traffic signs, which prevents over-fitting by enhancing Inception Modules and adding Batch Normalization [34]. Sun et al. proposed a CNN model that utilizes the Hough transform to locate circular images after pre-processing to emphasize important information [35]. These studies demonstrate the efficacy of deep learning algorithms for traffic sign recognition and classification. The use of different models such as pre-trained VGG16 network, capsule network, and GoogleLeNet-based CNN provides diverse options for developing efficient algorithms. These models have proven to overcome issues such as over-fitting and difficulty in learning complex shapes due to limited convolutional layers in traditional models. Moreover, the GTSRB dataset is widely used in these studies as a standardized benchmark for evaluating algorithm performance.

To overcome the limitations of color-based detection techniques, shape-based detection methods have been developed that detect traffic signs based on their geometric properties. Common traffic sign shapes include triangles, rectangles, octagons, and circles. Several shape analysis algorithms have been proposed in the literature for traffic sign detection. For instance, a Hough transform is employed to detect circular and triangular traffic signs using the Canny edge detection method in [36]. To reduce computational costs, some contours are rejected based on their area and perimeter before applying the Hough transform. The authors achieved high detection rates of 97.2% and 94.3% for speed limit and warning signs, respectively, using this method.

In [37], a grayscale-based approach using the radial symmetry transform was proposed for detecting speed limit traffic signs. This method involves calculating the gradient of each pixel, creating two vote images with gradient orientation and magnitude information, and then combining them to form a radial symmetry image. Circles are detected through thresholding the radial symmetry image, but this method is only effective for regular polygons and not for signs with geometric distortion.

Another template-based approach for traffic sign detection is proposed in [38]. This method involves using two binary images, the feature image (I) and the feature template (T). The Distance Transform is used to measure the matching between T and I, with each pixel value in the DT image representing the distance of that pixel to its nearest edge. The template is matched against the DT image to identify the shape of interest, but this method can result in high computational costs.

Traffic sign detection is crucial in ensuring safety in autonomous driving systems. One of the main challenges in this task is the variation in ambient illumination, which affects color-based detection methods. To address this, shape-based detection methods have been developed, including algorithms for shape analysis.

In one study, a novel technique was proposed for detecting speed limit traffic signs. The radial symmetry transform was used, which involved converting the original image to grayscale and calculating the gradient of each pixel. From this, two vote images were created, one containing gradient orientation and the other containing magnitude information, and these were combined to produce a radial symmetry image. By applying a threshold to this image, circles were detected. However, this method was limited to regular polygons and could not be used for signs with geometric distortion.

Detecting traffic signs accurately is a critical task in developing advanced driver assistance systems. Since detecting traffic signs based on color can be challenging due to varying illumination conditions, shape-based detection methods have become more popular. Common traffic sign shapes include circles, rectangles, octagons, and triangles. In a previous study, the Hough transform was applied to detect circular and triangular signs using Canny edge detection to identify edges in the image. Rejecting some contours based on area and perimeter reduced computational costs, and the remaining contours were then processed using the Hough transform.

To improve the accuracy of traffic sign detection and reduce interferences, combining color and shape information has shown promising results. A recent study used RGB ratios for segmentation and the Douglas-Peucker (DP) algorithm for shape analysis. The DP algorithm approximates contours and detects traffic signs based on the number of object boundaries, allowing for detection even with geometric distortions. In another study, color segmentation was performed in the HSV color space, and bounding boxes were added to detected regions. The traffic sign was then identified using features such as mean color, size, and the number of pixels enclosed in the boundary box.

In a pioneering work [41], Maximally Stable Extremal Regions (MSERs) were employed to detect traffic signs. The RGB frame was first converted to grayscale and then binarized using various threshold levels to identify MSERs, which are known for their robustness to changes in illumination. Another approach [42] utilized a color probability model and Histogram of Oriented Gradient (HOG) features to detect traffic signs. The color probability model transformed the image into probability maps, enhancing the colors associated with traffic signs and suppressing other colors. The MSER region detector was then utilized to identify regions likely to contain traffic signs. SVM was trained using color HOG features computed on the probability maps. In [43], the HSI color space was adopted to improve segmentation performance. The segmentation process produced blobs, for which Distance to Border features were computed. SVM was then used to classify the blobs.

A pixel classification algorithm using SVM was introduced in [44], which trained the SVM using labelled samples of the target color and other colors from training images. The RGB color space was used for simplicity, and the SVM outperformed conventional color thresholding methods. However, the main drawback of SVM is its speed.

Traffic sign recognition (TSR) is crucial in computer vision as it enhances road safety. Various methods have been developed for TSR, including SIFT and SURF [44], HOG [45][46], DCT [47], LBP [48], and Gabor features [45]. These methods have achieved high accuracy in traffic sign recognition tasks, with HOG being particularly successful in capturing the object's shape, which represents the orientation of the gradient image.

LBP-based techniques have gained popularity for their superior performance in TSR [48][49], although they have some limitations, such as ignoring global shape distribution and only considering the sign of the difference between two Gray values in the rotational invariant version [45]. For classification, SVM is widely used [50], and a novel learning algorithm called ELM has also shown promising results with HOG-based features [51][52]. The ELM algorithm requires additional computing cost for more hidden neurons but has demonstrated better performance than traditional methods.

However, these state-of-the-art methods are computationally expensive, and achieving real-time performance is a challenge. To overcome this, a fusion of descriptors has been proposed in this paper. Specifically, three descriptors, namely Circular Local Binary Patterns (CLBP), HOG, and Gabor, are fused at the feature level in conjunction with ELM as a basic classifier. CLBP captures both the local texture and the shape of the object, Gabor captures the local frequency and orientation information, and HOG captures the shape of the object. The fusion of these descriptors aims to overcome the insufficiency of a single descriptor in the visual classification task, due to the high level of intra-class variability coupled with low interclass distance.

The proposed method was extensively evaluated on two benchmark datasets, including the GTSRB and the BTSC datasets. The results showed the effectiveness of the proposed method, achieving high accuracy with reduced computational cost.

2.2 Issues to be addressed in Dataset

TSDR systems are an essential component of intelligent transportation systems. However, developing an efficient TSDR system is not a trivial task, as there are various challenges and issues that need to be considered. Major issues that need to be addressed while developing a TSDR system.

1. Variable lighting conditions [53]: The TSDR system's accuracy can be negatively affected by variations in lighting conditions, which can alter the distinct color of a traffic sign. Hence, it is crucial to develop a system that can adapt to changes in lighting caused by various weather conditions and times of day.

2. Fading and blurring effect [54]: Fading and blurring effects caused by sunlight and rain can result in inaccurate detection and decreased efficiency of the TSDR system. To address this issue, techniques such as adaptive thresholding and Hough transformation can be used, which are resilient to varying illumination and lighting conditions.

3. Affected visibility: Visibility of traffic signs can be impacted by several factors such as shadows, glare from headlights, precipitation, and atmospheric conditions. These elements

can have a significant negative impact on the precision of the TSDR system, particularly in real-time scenarios. Therefore, developing a system that can withstand changes in visibility conditions is necessary for consistent and accurate results.

4. Multiple appearances of signs [55]: Overlapping traffic signs or man-made objects with similar shapes can result in inaccurate detection, especially due to factors like rotation, translation, scaling, and partial occlusion. To mitigate these issues, techniques such as HSI transform and fuzzy shape recognizer can be used as they are robust and not impacted by these challenges.

5. Motion artifacts [56]: Motion artifacts caused by the movement of the vehicle or using low-resolution cameras can result in blurred and noisy images. Using techniques like Haar-like features, support vector machines (SVM), and PWP3D algorithm can address these issues by reducing the effect of motion blur, rotation, scaling, and other related problems.

6. Chaotic background and viewing angle problem: The chaotic nature of the background and foreground scenery, as well as the continuous changes in the viewing angle while driving, can make the detection process challenging. Techniques like adaptive thresholding, adaptive shape analysis, and self-organizing maps can be employed to address these issues and improve the accuracy of detections.

7. Damaged and partially obscured signs [57]: Damaged or partially obscured signs can create challenges for detection and recognition, particularly when the system includes a shape recognizer. To mitigate these issues, 3D reconstruction methods can be utilized in real-time environments to detect damaged signs.

8. Unavailability of public database: The lack of free and well-organized publicly available databases is a significant challenge in developing a TSDR system. This limitation limits the ability to train and test models, leading to compromised system performance. Currently, only a few recognized publicly available datasets such as GTSRB, KUL, and STS are available.

CHAPTER - 3

METHODOLOGY

3.1 Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a widely used dataset for training and evaluating computer vision models for traffic sign recognition. The dataset was introduced in 2011 and consists of more than 50,000 images of 43 different traffic sign classes, with each class containing between 180 and 2000 images. The images were captured under different weather and lighting conditions and from different camera angles, simulating real-world scenarios. The dataset is split into a training set of 34,799 images, a test set of 12,630 images, and a validation set of 4410 images. The distribution of images in each class is represented in Figure 1. Table 1 illustrates the 43 different traffic sign classes with the corresponding Class ID.

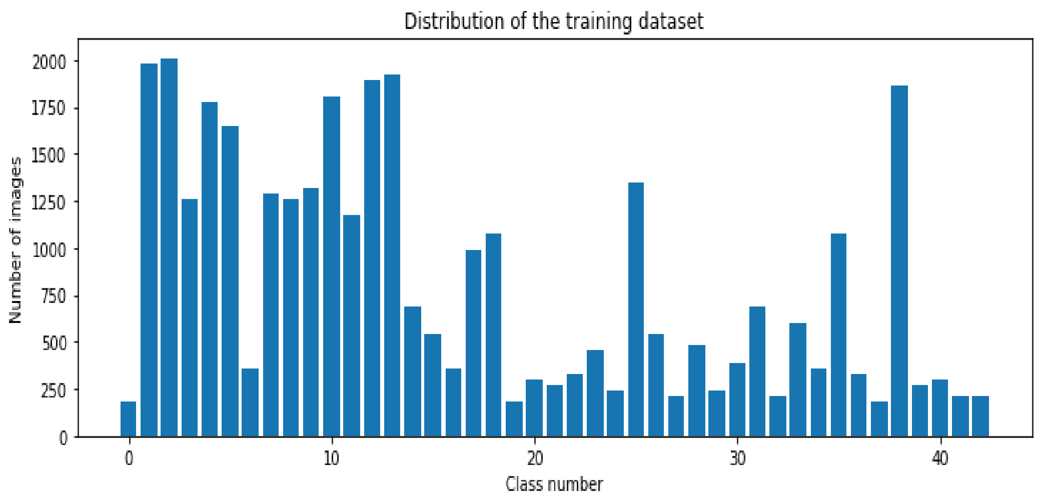


Figure 1: Distribution of images in 43 different traffic sign classes.

Table 1: 43 Traffic Sign Classes with the corresponding Class ID

Class ID	Traffic Sign Name
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)

5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric

3.2 Brief Description of Parameters

CNNs are a popular type of deep learning model that is commonly used for image and video processing tasks. These models typically have many parameters, which can greatly impact their performance. The process of choosing the right set of parameters for a CNN model is crucial to ensure that it performs well and can be generalized to new data.

In this article, we will discuss the importance of choosing parameters in a CNN model, including the various types of parameters and how to tune them effectively.

There are several types of parameters in a CNN model that need to be chosen carefully to ensure optimal performance. These include:

1. Filter Size: The filter size is the size of the kernel that is used to convolve the input image. The filter size determines the receptive field of each neuron in the convolutional layer, which affects the size and complexity of the learned features. Choosing the right filter size is crucial to ensure that the model can capture the relevant features in the input data.

2. Number of Filters: The number of filters utilized in each convolutional layer influences the model's capacity to learn features. While increasing the number of filters can aid the model in capturing complex features, it may lead to overfitting.

3. Stride Size: The stride size defines the distance between the application of each filter to the input image. A larger stride size can decrease the size of the output feature map and the number of parameters in the model, but it may also result in information loss.

4. Padding: Padding involves adding extra pixels around the edge of the input image to ensure that the output feature map is the same size as the input. Selecting the appropriate padding can prevent information loss at the edges of the input image.

5. Pooling: Pooling is the process of down-sampling the output feature map from the convolutional layer. This helps in reducing the size of the feature map and preventing overfitting. The size of the pooling window is a crucial parameter that needs to be selected carefully.

6. Learning Rate: The learning rate is a hyperparameter that controls the step size taken during gradient descent optimization. A higher learning rate can result in faster convergence, but it can also result in instability and overshooting. A lower learning rate can result in more stable convergence, but it can also result in slower convergence.

7. Dropout: Dropout is a regularization technique that randomly removes neurons during training to avoid overfitting. The dropout rate is a parameter that needs to be selected carefully to prevent excessive regularization.

To ensure that a CNN model performs optimally, it is important to tune the parameters effectively. The following techniques can be used to achieve this:

1. Grid Search: Grid Search involves systematically searching a range of parameter values to find the optimal combination. It may take some time, but it can effectively identify the optimal set of parameters.

2. Random Search: Random Search involves randomly sampling the parameter space to find the optimal combination. It can be more efficient than Grid Search as it doesn't require searching the entire parameter space.

3. Bayesian Optimization: Bayesian Optimization involves building a probabilistic model of the objective function and using it to guide the search for the optimal set of parameters. It's more efficient than Grid Search or Random Search as it considers the uncertainty in the objective function.

4. Early Stopping: Early Stopping involves stopping the training process when the validation loss starts to increase. This technique helps to prevent overfitting by avoiding overtraining the model, which can lead to memorizing the training data instead of generalizing to new data. Early Stopping is especially useful for large datasets as it saves time and computational resources.

5. Data Augmentation: Data Augmentation involves generating new training examples from existing ones by applying transformations like rotations, translations, and scaling. It can help

to increase the size of the training dataset and prevent overfitting by exposing the model to a wider range of variations in the input data.

6. Transfer Learning: Transfer Learning involves using a pre-trained CNN model as a starting point for a new task. This technique can be useful when working with limited data, as it allows the model to leverage the knowledge gained from a larger dataset.

7. Regularization: Regularization involves adding a penalty term to the objective function during training to prevent overfitting. It encourages the model to learn simpler representations that generalize better to new data.

Choosing the proper set of parameters is critical for a CNN model's performance. To provide optimal performance, several factors like as filter size, number of filters, stride size, padding, pooling, learning rate, and dropout must be properly set. Grid search, random search, Bayesian optimisation, early halting, data augmentation, transfer learning, and regularisation are all techniques that can be used to do this. We can ensure that a CNN model performs well and can be generalised to new data by carefully selecting and modifying its parameters.

3.2.1 Optimizers

Optimizers are essential components of ML algorithms that are used to optimise model learning. An optimizer's main goal is to minimise the loss function by modifying the model's weights and biases. The loss function quantifies how well the model performs on a given set of data. The optimisation process is iterative, with the optimizer updating the model's weights and biases based on the gradients of the loss function.

Optimizers are useful because they can dramatically increase the performance of machine learning models. The optimizer used can influence the model's convergence rate, stability, and accuracy. The optimizer employed can have a significant impact on the performance of a model, so selecting the correct optimizer for the task at hand is critical.

There are various sorts of optimizers available, each with their own set of advantages and disadvantages. Stochastic gradient descent (SGD), Adam, Adagrad, and RMSProp are some of the most commonly used optimizers.

The optimizer used can have a substantial impact on model performance. Different optimizers may be better suited to different sorts of datasets or designs in specific circumstances. Adagrad, for example, has been demonstrated to perform well on sparse datasets, whereas Adam has been demonstrated to perform well on a wide range of deep learning tasks.

One of the most important advantages of optimizers is that they can help to prevent overfitting. When a model performs well on training data but poorly on test data, this is referred to as overfitting. This can occur when a model becomes overly sophisticated and begins memorising the training data rather than understanding the underlying patterns. Optimizers can help to prevent overfitting by modifying the model's weights and biases to reduce training error while raising test error.

Another advantage of optimizers is that they can aid in the speeding up of the learning process. Optimizers can help to reduce the number of iterations required to obtain convergence by more efficiently modifying the model's weights and biases. This is especially crucial for deep learning models, which might take a long time to converge.

Optimizers can also help to increase a model's stability. Optimizers can help to lessen the likelihood of the model becoming trapped in a local minimum by more efficiently modifying the weights and biases. This can help to improve the model's robustness and lower the likelihood of it being locked in inferior answers. Furthermore, certain optimizers, such as Adam, incorporate a momentum term, which can assist the optimizer in moving through local minima and converge to a superior solution.

Furthermore, optimizers can be used to improve the interpretability of a model. By selecting an appropriate optimizer and tuning its hyperparameters, it is possible to adjust the trade-off between model complexity and accuracy. This can help to ensure that the model is not overfitting the data and that the underlying patterns are being learned correctly. This can be particularly important in applications where interpretability is critical, such as in the medical or financial industries.

Optimizers are also necessary for deep neural network training. Deep neural networks are challenging to train since they are highly nonlinear and contain numerous parameters. Because some optimizers may be more effective at dealing with the high-dimensional, non-linear nature of deep neural networks, the optimizer used can have a substantial impact on the training process. Furthermore, some optimizers, such as Adam, can do adaptive learning rate scheduling, which can help to speed up the training process and enhance model accuracy.

Finally, optimizers are an important component of ML algorithms that can have a substantial impact on a model's performance, stability, and interpretability. The optimizer selected should be based on the task at hand as well as the features of the dataset and architecture being used. It is possible to improve the accuracy, speed, and stability of an ML model by selecting an appropriate optimizer and tweaking its hyperparameters. The importance of optimizers will only grow as machine learning is employed in a growing number of sectors.

3.2.1.1 Adam

Adam is a machine learning optimisation technique that is specifically developed to train deep neural networks. It is a learning rate adaptable algorithm that combines the features of two previous optimisation algorithms: Adagrad and RMSprop.

Adam computes distinct adaptive learning rates for each parameter based on estimates of the gradient's first and second moments. The programme can then alter the learning rates of the parameters during the training process to improve model optimisation. Because of its adaptability, Adam is an efficient and effective algorithm for training deep neural networks.

The advantages of Adam are:

- Efficiency: Adam is an efficient optimization algorithm that can converge faster than other optimization algorithms such as Adagrad and SGD.

- Robustness: Adam is robust to noisy gradients and is capable of handling sparse gradients, which can be useful for large-scale deep learning problems.

- Adaptive learning rates: Adam adaptively tunes the learning rates of the parameters during training, making it more effective at optimizing the model.

However, there are also some disadvantages to using Adam:

- Sensitivity to hyperparameters: Adam requires the tuning of several hyperparameters, such as the learning rate and the decay rates, which can be time-consuming and require careful experimentation.

- Memory requirements: Adam requires storing additional parameters for each parameter in the model, which can be memory-intensive for large models.

Mathematically, the update rule for Adam is defined by Equation(9-12),

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{8}$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \tag{9}$$

$$m_{hat_t} = \frac{m_t}{(1 - \beta_{1^t})} \tag{10}$$

$$v_{hat_t} = \frac{v_t}{(1 - \beta_{2^t})} \tag{11}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha * m_{hat_t}}{\left(\sqrt{v_{hat_t}} + \varepsilon\right)} \tag{12}$$

where θ is the parameter vector, g is the gradient, m and v are the first and second moments of the gradient respectively, m_{hat} and v_{hat} are bias-corrected estimates of the moments, α is the learning rate, $\beta_1$and β_2 are the decay rates for the first and second moments respectively, t is the current time step, and ε is a small constant added for numerical stability.

3.2.1.2 RMSProp

RMSProp is an optimisation approach that is often used for neural network training. It is a gradient descent version that adjusts the learning rate during training. The moving average of the squared gradients for each weight in the network is calculated by RMSProp and used to change the learning rate for that weight. This allows the method to efficiently decrease the learning rate for weights with big gradients while increasing it for weights with small gradients, hence boosting the optimisation process.

The advantages of RMSProp are:

- Adaptive learning rates: RMSProp adaptively tunes the learning rates of the parameters during training, making it more effective at optimizing the model.

- Efficient: RMSProp is an efficient optimization algorithm that can converge faster than other optimization algorithms such as SGD.

- Robust: RMSProp is robust to noisy gradients and can handle sparse gradients, which can be useful for large-scale deep learning problems.

However, there are also some disadvantages to using RMSProp:

- Sensitivity to hyperparameters: RMSProp requires the tuning of several hyperparameters, such as the learning rate and the decay rate, which can be time-consuming and require careful experimentation.

- Local minima: RMSProp may get stuck in local minima, especially if the learning rate is not set appropriately.

Mathematically, the update rule for RMSProp is defined by Equation(13-14),

$$y_t = \beta * v_{t+1} + (1 - \beta) * g_t^2 \tag{13}$$

$$\theta_{t+1} = \frac{\theta_t - \alpha * g_t}{\sqrt{(y_t + \varepsilon)}} \tag{14}$$

where θ is the parameter vector, g is the gradient, v is the moving average of the squared gradients, α is the learning rate, β is the decay rate, t is the current time step, and ε is a small constant added for numerical stability.

In this update rule, the squared gradients are accumulated over time with a decay rate of β, which allows the algorithm to adaptively adjust the learning rate based on the history of the gradients. The learning rate is then scaled by the square root of the moving average of the squared gradients, which effectively divides the learning rate by a larger value for weights with large gradients and by a smaller value for weights with small gradients.

3.2.1.3 Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) is an optimization algorithm that combines the benefits of Nesterov accelerated gradient (NAG) and Adam optimization algorithms. Nadam is designed to efficiently optimize the loss function in deep neural networks with large datasets and numerous parameters.

The advantages of Nadam are:

- Fast convergence: Nadam utilizes both Nesterov momentum and Adam optimization to accelerate convergence and reduce the number of iterations required for training.

- Robust: Nadam is robust to noisy gradients and can handle sparse gradients, which makes it useful for large-scale deep learning problems.

- Adaptive learning rates: Nadam adjusts the learning rates adaptively, which helps to optimize the model effectively.

However, there are also some disadvantages to using Nadam:

- Hyperparameters: Nadam requires the tuning of several hyperparameters, such as the learning rate and decay rate, which can be time-consuming and require careful experimentation.

- Complexity: Nadam is a complex algorithm that may not be easy to implement for beginners.

The mathematical explanation of Nadam is as follows:

First, the algorithm computes the gradient of the loss function w.r.t the parameters. The gradient is then used to update the exponentially weighted moving average of the first moment (m_t) and second moment (v_t) of the gradients, which are used to compute the adaptive learning rates for each parameter.

The update rule for the first moment is like the Adam algorithm, and is defined by Equation 15.

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{15}$$

where g_t is the gradient at time step t, and β_1 is the exponential decay rate of the first moment.

The update rule for the second moment is also like the Adam algorithm, and is defined by Equation 16.

$$y_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \tag{16}$$

where β_2 is the exponential decay rate of the second moment.

Finally, the parameters are updated using the Nesterov accelerated gradient descent algorithm, which combines the current gradient and the previously computed gradient to estimate the gradient at the next time step is defined by Equation 17.

$$\theta_{t+1} = \theta_t - \alpha * \frac{(\beta_1 * m_t + (1 - \beta_1) * g_t)}{\left(\sqrt{(y_t)} + \varepsilon\right)} \tag{17}$$

where θ is the parameter vector, α is the learning rate, and ε is a small constant added for numerical stability.

In summary, Nadam is an optimization algorithm that combines the benefits of Nesterov accelerated gradient (NAG) and Adam optimization algorithms. It is a robust and efficient algorithm that adjusts the learning rates adaptively, but it requires the tuning of several hyperparameters and may be complex to implement.

3.2.1.4 AdaMax

AdaMax is an optimization algorithm that extends the Adam optimization algorithm by using the L-infinity norm instead of the L2 norm for computing the second moment estimates. AdaMax is designed to provide better convergence and stability when training deep neural networks.

The advantages of AdaMax are:

- Better convergence: AdaMax is designed to provide better convergence than Adam by using the L-infinity norm, which can provide more accurate estimates of the gradients and result in faster convergence.

- Stability: AdaMax is designed to be more stable than Adam, which can prevent the algorithm from diverging during training.

- Robustness: AdaMax can handle sparse gradients and noisy data, which makes it useful for deep learning problems with large datasets.

However, there are also some disadvantages to using AdaMax:

- Hyperparameters: AdaMax requires tuning several hyperparameters, such as the learning rate and the β1 and β2 decay rates, which can be time-consuming and require careful experimentation.

- Complexity: AdaMax is a complex algorithm that may not be easy to implement for beginners.

The mathematical explanation of AdaMax is as follows:

The algorithm uses the first moment estimate (m_t) and the L-infinity norm of the second moment estimate (u_t) to update the parameters. The update rule for the first moment is the same as that used in Adam is defined by Equation 18,

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{18}$$

where g_t is the gradient at time step t, and β_1 is the exponential decay rate of the first moment.

The update rule for the second moment estimate is defined by Equation 19.

$$u_t = max(\beta_2 * u_{t-1}, |g_t|) \tag{19}$$

where β_2 is the exponential decay rate of the second moment and $|g_t|$ is the L-infinity norm of the gradient at time step t.

Finally, the parameters are updated using the following rule is defined by Equation 20.

$$\theta_{t+1} = \frac{\theta_t - \alpha * m_t}{(u_t + \varepsilon)} \tag{20}$$

where θ is the parameter vector, α is the learning rate, and ε is a small constant added for numerical stability.

In summary, AdaMax is an optimization algorithm that extends the Adam optimization algorithm by using the L-infinity norm instead of the L2 norm for computing the second moment estimates. AdaMax provides better convergence and stability than Adam, but it requires tuning several hyperparameters and may be complex to implement.

3.2.2 Pooling Layers

CNNs, which are commonly used for image identification, object detection, and other computer vision tasks, rely heavily on pooling layers. The use of pooling layers is critical in increasing the performance of these models. The following are the primary benefits of pooling layers in CNN models:

- Feature reduction: Pooling layers reduce the dimensionality of the feature maps generated by the convolutional layers. This helps in reducing the number of parameters in the model, which can lead to better generalization and faster training. The pooling operation summarizes the local features in each feature map by taking the maximum, average, or other aggregation function, which creates a smaller and more abstract representation of the input.

- Translation invariance: Pooling layers provide translation invariance to the model, which means that the model can recognize patterns even if they appear in different locations in the input. This is because the pooling operation aggregates the features in a local neighbourhood, which makes the output of the pooling layer less sensitive to small translations of the input.

- Robustness to noise: Pooling layers can improve the robustness of the model to noise and small perturbations in the input. The pooling operation averages the local features, which can help to smooth out noisy or spurious features in the input.

- Computational efficiency: Pooling layers can reduce the computational cost of the model by reducing the size of the feature maps. This makes it possible to use deeper and more complex models, which can achieve higher accuracy on challenging tasks.

Pooling layers are classified into three types: maximum pooling, average pooling, and global pooling. The most common type of pooling layer is max pooling, which has been found to be useful in capturing the most prominent aspects of the input. Average pooling can be effective in situations when geographical information is less critical, such as time-series data jobs. Global pooling is a technique for reducing the spatial dimension of feature maps to a single value that may be utilised as a feature vector in classification or regression applications.

3.2.2.1 Max Pooling

The max pooling procedure is a necessary component of CNN models that are used to downsample input feature maps. The method entails splitting the feature map into non-overlapping rectangular parts and picking the maximum value inside each sector as the max pooling layer's output. The spatial resolution of this downsampled output has been lowered by a factor of the pooling size.

In CNN models, max pooling has various advantages. It decreases the spatial dimension of the feature maps, lowering the number of parameters and lowering the danger of overfitting. Furthermore, it improves the model's translation invariance by minimising its sensitivity to tiny changes in the input, making it more robust to fluctuations in feature location.

To build a hierarchical representation of the input, max pooling is frequently used with convolutional layers. Convolutional layers capture local patterns, whereas max pooling layers summarise the most prevalent features and reduce the dimensionality of the representation. This allows the model to discover and classify complicated and abstract properties by exploiting the object's overall structure rather than simply individual pixels. Figure 2 depicts an illustration of the Max-Pooling operation.

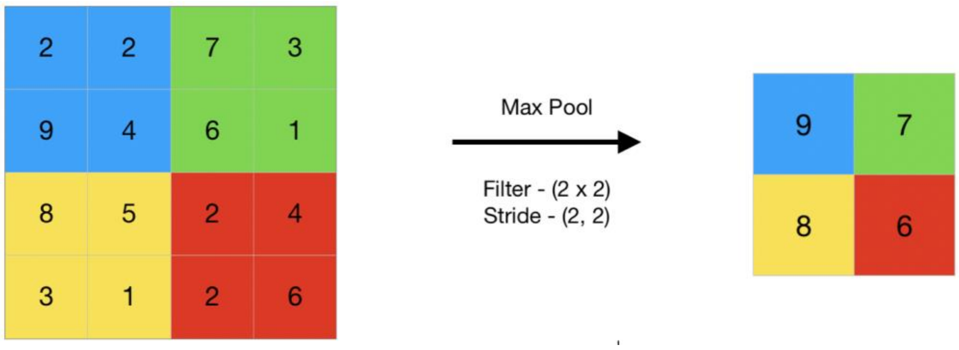


Figure 2: Max-Pooling operation on a matrix

3.2.2.2 Average pooling

The average value of the local features in a rectangular neighbourhood is used to aggregate them. Average pooling, like max pooling, is used to minimise the spatial dimension of feature maps while capturing the most important features of the input.

Average pooling has the advantage of smoothing out the local features in the input, making the model more resilient to noise and tiny fluctuations. This is because averaging decreases the impact of outlier features and makes the pooling layer's output more stable.

However, when compared to maximum pooling, average pooling has several drawbacks. For starters, it can result in a loss of spatial resolution because the pooling layer's output is a down-sampled version of the input feature map. This can impair the model's capacity to accurately localise the characteristics in the input. Second, because it does not select the most prominent characteristics in each local region, average pooling does not provide as much translation invariance as max pooling.

Despite these drawbacks, average pooling can be a beneficial alternative to max pooling in some instances, particularly where spatial information is less relevant or the input contains a high level of noise or fluctuation. In general, the pooling layer selected is determined by the specific properties of the input and the task requirements. Figure 3 depicts an example of an Average-Pooling procedure.

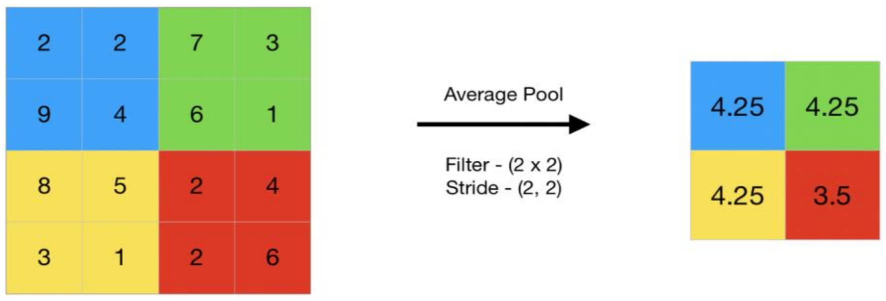


Figure 3: Average Pooling operation on a Matrix

3.2.3 Activation Function

Activation functions are critical components of CNN models because they enable the model to learn non-linear correlations between input and output. Without these functions, the model could only learn linear correlations, which may not be enough for complex tasks like object recognition.

CNN models can use a variety of activation functions, including sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU) functions. The precise activation function chosen is determined by the task at hand and the model design.

The significance of activation functions stems from their capacity to create nonlinearity, which aids in the capture of complicated patterns in the input. This is especially important for applications such as object identification, which require the model to detect complicated properties and relationships in the input.

Furthermore, activation functions have a role in preventing vanishing gradients in deep CNN models. The model can learn and update its parameters more efficiently if gradients are not too tiny. Finally, activation functions can help to normalize the output of the model, which promotes stable training and prevents numerical errors that could otherwise occur.

3.2.3.1 ReLU

The Rectified Linear Unit (ReLU) is a commonly used activation function in CNNs. Its functionality involves returning the input if it is positive and zero otherwise, making it a non-linear function.

Other activation functions, such as the sigmoid or hyperbolic tangent functions, have certain advantages over the ReLU activation function. For starters, because it is a basic function that only requires a comparison to zero, it is computationally efficient. This feature makes it suited for large-scale CNN models with many parameters.

Second, the ReLU function mitigates the vanishing gradient problem, which occurs when gradients get extremely small, causing the model to stop learning. This is because the ReLU function's derivative is either 0 or 1, which means it is always straightforward to compute and does not become extremely small.

Finally, the ReLU function has been demonstrated to enhance the performance of CNN models, particularly in object recognition tasks. This is because the function can capture the non-linearities and intricate patterns in the input, thus enabling the model to learn more discriminative features. The graphical representation of the ReLU activation function is illustrated in Figure 4.

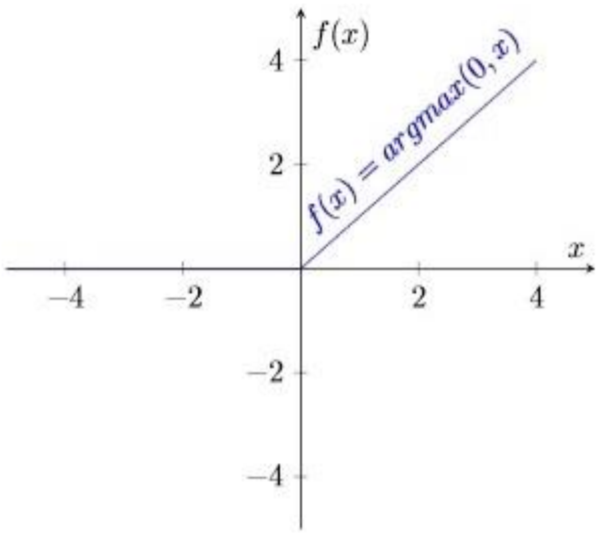


Figure 4: Graphical Representation of ReLU activation function.

Advantages of ReLU Activation Function:

- Efficient Computation: ReLU is a simple function that requires only a comparison to zero, making it computationally efficient. This makes it suitable for large-scale CNN models with many parameters.

- Prevents Vanishing Gradient Problem: The ReLU function helps to prevent the vanishing gradient problem, which can occur when the gradients become very small and cause the model to stop learning. This is because the derivative of the ReLU function is either 0 or 1, which means that it is always easy to compute and does not become very small.

- Improves Model Performance: The ReLU function has been shown to improve the performance of CNN models, especially in object recognition tasks. This is because the function can capture the non-linearities and complex patterns in the input, allowing the model to learn more discriminative features.

Disadvantages of ReLU Activation Function:

- Dead Neurons: The ReLU function can lead to dead neurons, where the output of the neuron is always zero and the gradient cannot be updated during training. This can occur when the input to the neuron is negative, as the ReLU function returns a zero output for negative inputs.

- Unbounded Output: The ReLU function has an unbounded output, which means that it can lead to exploding gradients. This can cause the model to become unstable during training, making it difficult to optimize the parameters.

- Non-centred Output: The ReLU function has a non-centred output, which means that the output is always non-negative. This can be a problem when the input data has negative values, as the ReLU function will always return a zero output for negative inputs, which can cause information loss.

3.2.3.2 Softmax

The Softmax is a mathematical function frequently utilized in machine learning, particularly in classification tasks. It is an extension of the logistic function and can transform a vector of real values into a probability distribution where the values lie between 0 and 1 and sum up to 1. Softmax is usually employed as the final layer in a neural network for classification problems to output a probability distribution over the classes.

Advantages:

- Softmax provides a smooth and continuous probability distribution over classes that can be easily interpreted.

- The outputs of the softmax function can be used directly for calculating the cross-entropy loss, which is commonly used as the loss function for classification tasks.

- Softmax is differentiable, which allows it to be used in backpropagation-based optimization algorithms like gradient descent.

Disadvantages:

- Softmax requires the input vector to be of fixed size, which can be a limitation in some applications.

- The output of the softmax function is sensitive to outliers in the input vector, which can result in misclassification of the samples.

- Softmax does not account for the relationships between the classes and assumes that they are independent, which may not be true in some applications.

3.2.4 Dropout Layer

The Dropout layer is an important component in convolutional neural networks since it acts as a regularisation approach to reduce overfitting and improve the model's generalisation capabilities. The Dropout layer works by omitting some neurons in the network at random during the training phase by setting their values to zero. The dropout probability hyperparameter can be tuned to change the degree of regularisation. The Dropout layer's significance in CNN models can be summarised as follows:

- Prevents Overfitting: One of the main advantages of Dropout is that it prevents overfitting, which can occur when the model is trained too well on the training data and cannot generalize well to unseen data. By randomly dropping out neurons, the model is forced to learn more robust features that are not dependent on the specific input data

- Improves Generalization: Dropout improves the generalization performance of the model by reducing the sensitivity of the model to specific features of the input data. This means that the model can perform better on unseen data and is less likely to make errors due to noise or irrelevant features.

- Reduces Complexity: Dropout can be seen to simplify the model by reducing the number of neurons in the network during training. This can make the model easier to optimize and reduce the risk of overfitting.

- Speeds up Training: Dropout can speed up the training process by reducing the number of iterations required to converge to a good solution. This is because the model is trained on a smaller subset of neurons, which reduces the computation required for each iteration.

Figure 5 represents two neural network one with dropout layer and other without dropout layers.

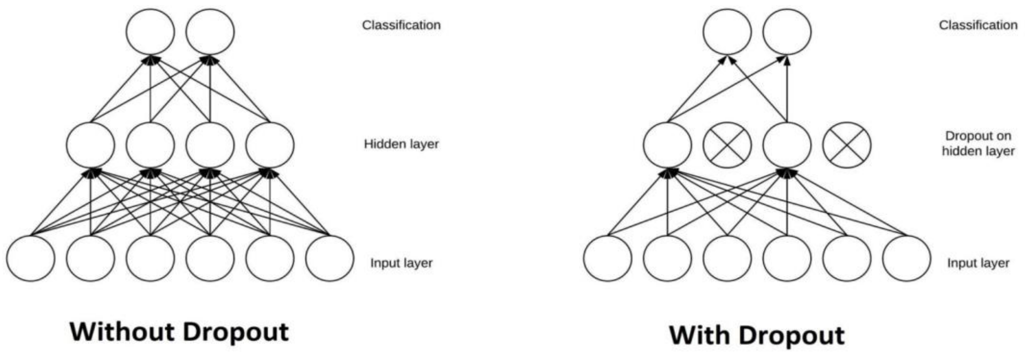


Figure 5: Comparison of neural network with and without dropout layer

3.3 Pre-processing

The importance of pre-processing in CNN can be attributed to the fact that the quality of the input data is directly related to the accuracy and performance of the model. Pre-processing techniques can help extract important features from the input data, reduce the impact of noise, and simplify the data to make it easier for the model to learn.

Pre-processing also helps to reduce overfitting by creating more diverse training data via data augmentation. This allows the model to generalise better to new, previously unknown data. Furthermore, normalisation of the input data can help to stabilise the learning process, making it more consistent and increasing the model's convergence speed. Another key advantage of pre-processing is that it can aid in model optimisation by lowering the complexity of the incoming data. Reduced data dimensionality improves the computational efficiency of the model and allows it to handle vast amounts of data more effectively.

In summary, pre-processing is a critical step in CNN that can help improve the quality of the input data, optimize the performance of the model, and reduce the likelihood of overfitting. By applying various pre-processing techniques, we can prepare the input data in a way that makes it easier for the model to learn and make accurate predictions.

3.3.1 Resizing

Resizing is an important pre-processing step in CNN that involves changing the size of the input images to a fixed size. The importance of resizing in CNN can be summarized as follows:

- Standardization: Resizing ensures that all the input images have the same size, regardless of their original size. This makes it easier to process the images and reduces the need for complex resizing algorithms during training. Standardizing the input size also enables us to compare the performance of different models accurately.

- Reducing Computational Complexity: Resizing can help reduce the computational complexity of the model by reducing the size of the input images. Smaller images require fewer computations and parameters, making the model faster and more efficient. This is especially important when working with large datasets or limited computing resources.

- Improving Model Performance: Resizing can also improve the performance of the model by reducing the noise and irrelevant features in the input images. This can help the model learn more important features and improve its accuracy on the test data.

- Enabling Transfer Learning: Resizing can also make it easier to use pre-trained models or transfer learning. Pre-trained models are typically trained on images of a specific size, and resizing the input images to match this size is necessary for the model to work properly. This makes it easier to use pre-trained models and speeds up the training process.

3.3.2 RGB to Gray Scale

Converting RGB (Red, Green, Blue) images to grayscale is an important pre-processing step in CNN that involves converting each color image into a grayscale image that has only one channel (Gray). The importance of converting RGB to grayscale in CNN can be summarized as follows:

- Simplifying the input data: By converting RGB images to grayscale, we simplify the input data for the model, reducing the complexity and computational requirements of the model. This is because grayscale images only have one channel, whereas color images have three channels.

- Reducing Overfitting: Grayscale images contain fewer features than color images, which can help reduce overfitting of the model to the training data. This is because the model is forced to focus on the most important features of the image and ignores less relevant information.

- Enhancing Contrast: Converting RGB to grayscale can enhance the contrast of the image, which can improve the ability of the model to distinguish between different features of the image.

- Preserving Spatial Information: Converting RGB to grayscale preserves the spatial information of the image, which can be important in tasks such as image segmentation, where spatial information is crucial.

- Saving Computational Resources: Since grayscale images have only one channel, they require less computational resources than color images, which can reduce the computational requirements of the model.

3.3.3 Histogram equalization

Histogram equalization is a method that redistributes the intensity values of an image in order to adjust its contrast. This technique is frequently used as a pre-processing step in CNN models due to its significance in the following ways:

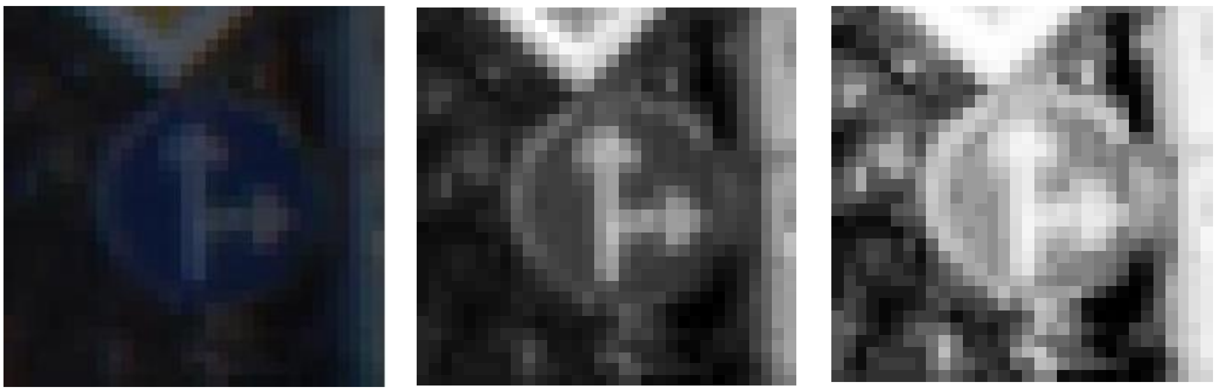
- Improving Image Contrast: Histogram equalization enhances the contrast of the image by distributing the intensity values uniformly across the entire range. This improves the visibility of details in the image and makes it easier for the model to detect features.

- Removing Image Bias: In some cases, images can have a bias towards specific intensity values, which can affect the performance of the model. Histogram equalization can help remove this bias and ensure that the model is not biased towards any specific intensity value.

- Normalizing Image Intensity: Histogram equalization can normalize the intensity values of the image, which can make it easier for the model to learn and generalize features across different images.

- Enhancing Features: Histogram equalization can also enhance the features of an image, making it easier for the model to detect important details and classify the image accurately.

The reduction of dimensions in an image is depicted in Figure 6(a), which is then transformed into an 8-bit grayscale image. Figure 6(b) represents the resulting grayscale image. To enhance the contrast of the image, histogram equalization is performed on all images. The outcome of histogram equalization on the grayscale image is demonstrated in Figure 6(c).



(a) (b) (c)

Figure 6: Input image at different steps of pre-processing (a) Image resized in 32x32 (b) RGB to grayscale (c) Histogram equalization

3.3.4 Data Augmentation

Data augmentation is a technique used in ML and computer vision to increase the size of a dataset by generating new data from existing data, without collecting new data from the real world. In the case of image data, data augmentation involves applying a variety of transformations to existing images to create new images that are variations of the original images. The augmented images are then used to train ML models. It is a crucial technique in computer vision, as it allows deep learning models to be trained on a larger and more diverse dataset, which can improve their accuracy and generalization performance. In addition, data augmentation can help to mitigate overfitting, as it introduces randomness and variability to the training data. Figure 7 represents the results of transformations on a traffic sign image.

Transformations we used in this model are:-

1. Height and Width Shift
2. Zoom
3. Shear
4. Rotation
5. Translation

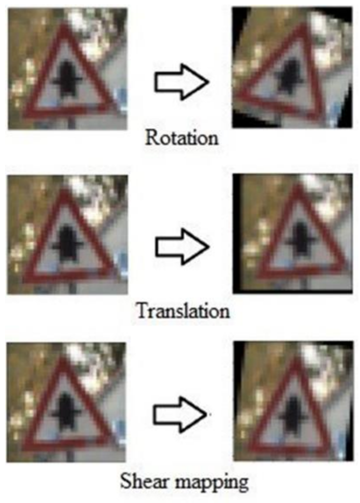


Figure 7: Example of Rotation, Translation and Shear mapping

3.4 Le-Net Model

The Le-Net is a popular CNN architecture that's frequently used for image recognition tasks. It comprises multiple layers, including convolutional layers, pooling layers, and fully connected layers.

This implementation of the Le-Net model consists of a total of 7 layers, with 2 convolutional layers, 2 pooling layers, 1 flatten layer, and 2 fully connected layers. The first convolutional layer contains 60 filters with a 5 x 5 kernel size and uses the ReLU activation function. The input shape of this layer is (32, 32, 1), indicating that grayscale images of size 32 x 32 pixels are used as inputs. The resulting feature maps are then passed through a max pooling layer with a pool size of (2, 2), which decreases their spatial dimensions.

The second convolutional layer comprises 30 filters with a kernel size of 3 x 3 and uses the ReLU activation function. This is followed by another max pooling layer with the same pool size as the previous layer. The output of the second pooling layer is then flattened into a 1-dimensional array and passed through a fully connected layer with 500 neurons and the ReLU activation function. To reduce overfitting, a dropout layer is added. Finally, the output layer has the same number of neurons as the number of classes and uses the softmax activation function. The architecture of the Le-Net model is depicted in Figure 8.

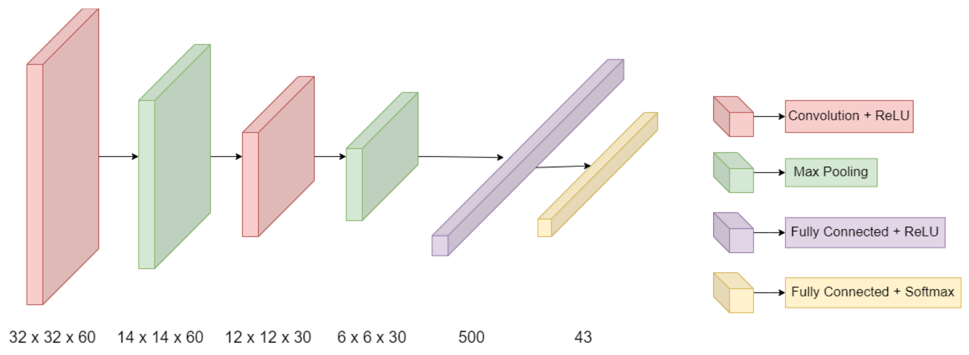


Figure 8: Architecture of Le-Net Model

The summary of the Le-Net Model is mentioned in Table 2.

Table 2: Summary of the Le-Net Model.

Layer (type)	Output Shape	Parameters
Convolution 2D	(28, 28, 60)	1560
Max-Pooling 2D	(14, 14, 60)	0
Convolution 2D	(12, 12, 30)	16230
Max-Pooling 2D	(6, 6, 30)	0
Flatten	(1080)	0

Dense	(500)	540500
Dropout	(500)	0
Dense	(43)	21543
Total parameters		5,79,833
Trainable parameters		5,79,833
Non-trainable parameters		0

3.5 Modified Le-Net

The Modified Le-Net model function defines a CNN model based on the Le-Net architecture for image classification tasks. The model takes as input grayscale images of size 32x32 pixels. Figure 9 represents the Modified Le-Net Model. Table 3 represents the summary of the model.

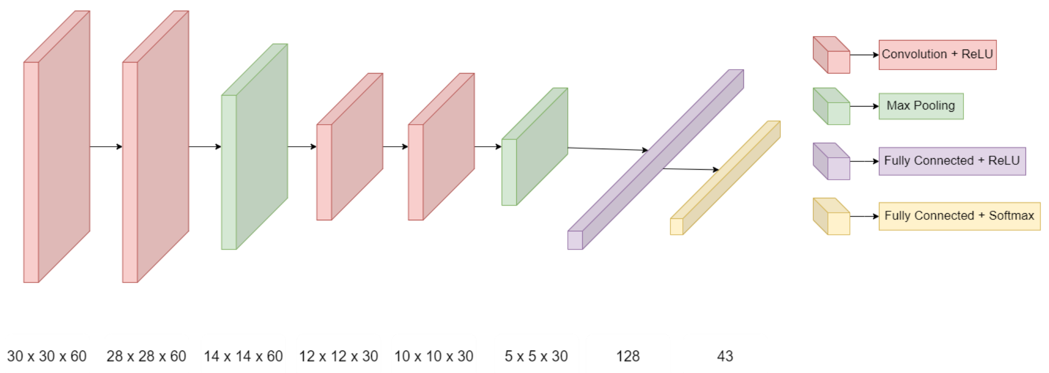


Figure 9: Architecture of Modified Le-Net Model

The model is composed of three blocks of layers:

- The first block consists of two Conv2D layers with 60 filters each, followed by a MaxPooling2D layer with a 2x2 pooling size. The activation function used is LeakyReLU, which helps to prevent the vanishing gradient problem.

- The second block consists of two Conv2D layers with 30 filters each, followed by a MaxPooling2D layer with a 2x2 pooling size. The activation function used is also LeakyReLU.

- The final block consists of a Flatten layer, which flattens the output of the previous block into a 1D vector, followed by a Dense layer with 128 units and a LeakyReLU activation function. A Dropout layer with a rate of 0.5 is then applied to reduce overfitting.

- The output layer is a Dense layer with the number of units equal to the number of classes in the dataset, and a softmax activation function to produce class probabilities.

Table 3: Summary of the Modified Le-Net Model

Layer (type)	Output Shape	Parameters
Convolution 2D	(30, 30, 60)	1560
Convolution 2D	(28, 28, 60)	90060
Max-Pooling 2D	(14, 14, 60)	0
Convolution 2D	(12, 12, 30)	16230
Convolution 2D	(10, 10, 30)	1560
Max-Pooling 2D	(5, 5, 30)	0
Flatten	(750)	0
Dense	(128)	540500
Dropout	(128)	0
Dense	(43)	21543
Total parameters		1,59,095
Trainable parameters		1,59,095
Non-trainable parameters		0

3.6 Evaluation Parameters

3.6.1 Accuracy

Accuracy is a commonly used evaluation metric in deep learning that measures the percentage of correctly classified instances over the total number of instances in a dataset. It is a simple and intuitive metric that gives a quick understanding of the overall performance of a model. It is defined in Equation (21).

$$Accuracy = \frac{True_{Positive} + True_{Negative}}{True_{Positive} + True_{Negative} + False_{Positive} + False_{Negative}} \quad (21)$$

3.6.2 Precision

Precision is a measure of a classifier's exactness. It is the ratio of true positives to the total number of positive predictions made by the classifier. It is defined in Equation 22.

$$Precision = \frac{True_{Poisitive}}{True_{Positive} + False_{Positive}} \tag{22}$$

3.6.3 Recall

Recall is also known as sensitivity or the true positive rate (TPR). It is calculated as the ratio of true positive (TP) instances to the sum of true positives and false negatives (FN). In other words, recall tells us the percentage of actual positive cases that are correctly identified by the model. It is defined in Equation 23.

$$Recall = \frac{True_{Positive}}{True_{Positive} + False_{Negative}} \tag{23}$$

3.6.4 F1-Score

F1 score (also known as F-measure, or balanced F-score) is a widely used metric in deep learning for evaluating the performance of classification models. It is a harmonic mean of precision and recall, which makes it a balanced measure of a model's accuracy. It is defined in Equation 24.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{24}$$

CHAPTER - 4

RESULTS & DISCUSSIONS

4.1 Results

This project aimed to test the performance of different optimizers, pooling layers, activation functions, and dropout layers on the Le-Net model. In total, four optimizers, including Nadam, Adam, RMSProp, and AdaMax, were evaluated with two learning rates of 0.01 and 0.001, and two pooling layers, namely Average and Max, were considered. Additionally, three activation functions, including ReLU and Softmax, were used along with four different dropout layers, namely No Dropout, 30%, 50%, and 70% dropout percentages in each layer.

Table 4 and Table 5 demonstrate the performance of the Le-Net model with different parameters and dropout layers, respectively. Based on the results, the top three performing parameters were selected for testing on the Modified Le-Net model.

Table 4: Accuracy achieved using different parameters on the Le-Net model

Learning Rate	Pooling Layers	Optimizers			
		Nadam	**Adam**	**RMSProp**	**AdaMax**
0.01	**Average Pooling**	89.59%	87.90%	89.48%	90.06%
	Max Pooling	88.96%	82.29%	91.40%	92.17%
0.001	**Average Pooling**	92.20%	91.22%	93.34%	88.58%
	Max Pooling	**93.58%**	**93.49%**	**94.17%**	91.33%

Table 5: Accuracy achieved using different dropout layers

S. No.	Dropout Percentage	Accuracy
1	No Dropout	92.92%
2	30%	93.20%
3	50%	94.17%
4	70%	**95.08%**

The experiment involved testing different parameters for the Le-Net model, including four optimizers, two pooling layers, three activation functions, and four dropout layers. The results were then analysed, and the top three parameters were selected to test on the Modified Le-Net model. The top three optimizers were Nadam, Adam, and RMSProp, and they were tested on the Modified Le-Net model. The results of these tests were then presented in Table 6 and Table 7, which showed the performance of the Modified Le-Net model with different parameters and dropout layers respectively.

Table 6: Accuracy achieved using different parameters on the modified Le-Net model

Learning Rate	Polling Layers	Optimizers		
		Nadam	**Adam**	**RMSProp**
0.001	**Average Pooling**	**97.68%**	97.43%	96.73%
	Max Pooling	97.07%	96.88%	96.06%

Table 7: Accuracy achieved using different dropout layers

S. No.	**Dropout Percentage**	**Accuracy**
1	No Dropout	94.36
2	30%	96.49
3	50%	**97.86**
4	70%	96.58

The classification report of the model is illustrated in Table 8. Which denotes the Precision, Recall and F1-Score of individual classes.

Table 8: Classification report of all 43 classes

Class ID	Precision	Recall	F1-score
0	1.00	0.92	0.96
1	0.98	0.99	0.99
2	0.99	0.98	0.99
3	0.98	0.98	0.98
4	0.98	1.00	0.99
5	0.96	0.98	0.97

6	0.93	1.00	0.97
7	0.99	0.98	0.99
8	0.98	0.99	0.98
9	1.00	1.00	1.00
10	0.99	1.00	0.99
11	0.98	0.94	0.96
12	0.99	0.94	0.96
13	1.00	0.99	1.00
14	1.00	0.92	0.96
15	0.98	0.93	0.95
16	1.00	1.00	1.00
17	0.94	1.00	0.97
18	0.85	0.98	0.91
19	1.00	0.78	0.88
20	0.99	0.99	0.99
21	0.67	0.98	0.79
22	0.98	0.98	0.98
23	1.00	0.95	0.97
24	0.93	0.97	0.95
25	0.94	0.99	0.97
26	0.97	0.86	0.91
27	0.62	0.43	0.51
28	1.00	0.98	0.99
29	1.00	0.97	0.98
30	0.85	0.95	0.90
31	0.97	0.99	0.98
32	0.98	1.00	0.99
33	0.99	0.99	0.99
34	1.00	0.99	1.00
35	0.99	0.99	0.99
36	1.00	0.98	0.99
37	1.00	0.95	0.98
38	0.96	0.98	0.97
39	0.99	0.95	0.97
40	0.90	0.98	0.94
41	0.98	0.98	0.98
42	0.99	0.99	0.99

4.2 Discussions

The optimizer used can have a big impact on the performance of a deep learning model. The Le-Net model was examined with four distinct optimizers in this case: Nadam, Adam, RMSProp, and AdaMax. The model performed best with the RMSProp optimizer, according to the results. However, the Modified Le-Net model fared best with the Nadam optimizer since it has more convolutional layers.

RMSProp is an optimizer that normalises the gradient by taking a moving average of the squared gradients. This prevents the learning rate from becoming too high and beyond the ideal point in the cost function. RMSProp also includes a momentum term, which smooth out the updates and allows the optimizer to move faster in the direction of the optimal point. As a result, it may be a viable choice for models with less convolutional layers, such as Le-Net.

In contrast, Adam combines the benefits of RMSProp and momentum optimisation. Adam is well-known for his ability to perform well with a wide range of deep learning models, including those with larger datasets and more sophisticated architectures. However, it did not outperform RMSProp for the Le-Net model in this example.

AdaMax is another Adam optimizer variation that is intended to address disappearing and exploding gradient concerns. It is an Adam optimizer extension that computes the gradient moving average using the Lp norm rather than the L2 norm. However, AdaMax did not outperform RMSProp in this scenario for the Le-Net model.

Nesterov accelerated gradient (NAG) and Adam optimizer are combined in Nadam. It computes the gradient using the NAG update algorithm rather than the conventional gradient update rule. The NAG update rule enables the optimizer to take the momentum of the previous step into account and alter the current gradient accordingly. Nadam is a suitable choice for models with more convolutional layers since it allows the optimizer to quickly converge to the optimal point. This is most likely why Nadam did the best for the Modified Le-Net model.

The optimizer utilised is determined by the model's specific properties. Because of its capacity to prevent overshooting the ideal point and smooth the updates, RMSProp performed best for the Le-Net model in this scenario. However, Nadam performed better for the Modified Le-Net model with additional convolutional layers due to its capacity to fast converge to the optimal point. Experiment with multiple optimizers and modify the hyperparameters to discover the optimal optimizer for a specific model.

The learning rate is a crucial hyperparameter in the training process of neural networks since it governs the size of the steps the optimizer takes during backpropagation. A high learning rate can accelerate convergence, but it can also cause overshooting and instability. In contrast, a low learning rate may ensure more stable training but slower convergence. Here, we will discuss why a learning rate of 0.001 outperforms 0.01.

When we employ a learning rate of 0.01, the optimisation process backpropagates in huge increments in the direction of the gradient. These huge steps may cause the optimizer to overshoot the minimum and fluctuate about it, making convergence to the optimal solution problematic. As a result, convergence is slower and forecasts are less accurate. Furthermore, a high learning rate can cause the optimisation process to diverge, resulting in a substantial increase in loss function and unstable training.

On the other hand, when we used learning rate equal to 0.001, the optimization algorithm takes smaller steps in the direction of the gradient. These smaller steps are less likely to overshoot the minimum and lead to more stable convergence. Smaller learning rates also help to avoid the issue of diverging during training. As a result, a learning rate of 0.001 allows the optimization algorithm to converge more quickly and accurately to the optimal solution.

In some cases, a higher learning rate may be more appropriate, especially for shallow networks. However, for deeper networks, a lower learning rate is generally preferred due to the increased risk of overshooting and instability.

In summary, a learning rate of 0.001 yields better results than 0.01 because it permits the optimisation process to take fewer, more steady steps in the gradient's direction during backpropagation. As a result, the convergence to the best solution is more accurate and faster. To get the optimum performance, the ideal learning rate must be carefully selected based on the unique problem and neural network architecture.

Dropout is a regularisation approach that is commonly used to prevent overfitting in deep learning models. It is a way of randomly removing some units during training, preventing the units from over-co-adapting. Dropout has been demonstrated to improve neural network generalisation performance, making them less prone to overfit the training data. The dropout percentage is the proportion of neurons in the network that are dropped out at random during training.

In this project, the Le-Net model was tested with four different dropout percentages: No Dropout, 30%, 50%, and 70%, to evaluate their impact on the model's performance. It was observed that both the Le-Net and Modified Le-Net models perform better with 50% and 70% dropout percentages, as compared to No Dropout or 30% dropout percentage. There are several possible reasons for this.

Firstly, Dropout can be seen as a form of ensemble learning, where different sub-networks are trained on different subsets of the input data. The dropout technique reduces the co-adaptation between the neurons, which forces the network to learn more robust features that are useful for classification. By randomly dropping out some of the neurons, the network becomes less sensitive to the noise in the input data, which can help it to generalize better to new examples.

Secondly, Dropout reduces the dependence of the network on a small subset of the input features. When the dropout percentage is high, the network is forced to learn more independent features, which can help it to generalize better. This can also reduce the risk of overfitting, as the network is less likely to rely on a small subset of the input features to make its predictions.

Thirdly, Dropout acts as a form of regularization by adding noise to the network during training. This can help to prevent overfitting, as it forces the network to learn more general features that are useful for the entire dataset, rather than just memorizing the training examples.

Finally, in this project, both the Le-Net and Modified Le-Net models have a relatively small number of parameters. This means that the models are more prone to overfitting, and therefore, dropout can be particularly effective in preventing overfitting and improving the model's performance.

Dropout is a powerful regularisation approach that can increase neural network generalisation performance. The dropout percentage defines how many neurons are randomly dropped out during training and has a substantial impact on the model's performance. It was discovered in this experiment that the Le-Net and Modified Le-Net models perform better with 50% and 70% dropout percentages, respectively, than with No Dropout or 30% dropout rate. Dropout minimises co-adaptation between neurons, lessens the network's reliance on a limited subset of input features, adds noise to the network during training, and works as a type of regularisation.

The choice between max pooling and average pooling depends on the specific problem and data at hand. In general, max pooling is more effective when the task involves identifying the most important features in the input data. This is because max pooling retains only the maximum value in each pool, discarding the less important features. This can be particularly useful in tasks such as object recognition, where identifying the most relevant features in an image is crucial for accurate classification.

On the other hand, average pooling is more effective when the task involves identifying the overall trends in the input data. This is because average pooling computes the average value in each pool, providing a smoother and more generalized representation of the input. This can be particularly useful in tasks such as sentiment analysis, where the overall sentiment of a text is more important than specific words or phrases.

It is worth noting that there are some cases where neither max pooling nor average pooling is the best choice. For example, if the input data contains a lot of noise or irrelevant features, both pooling methods may struggle to produce accurate results. In these cases, other pooling methods such as Lp pooling or stochastic pooling may be more effective.

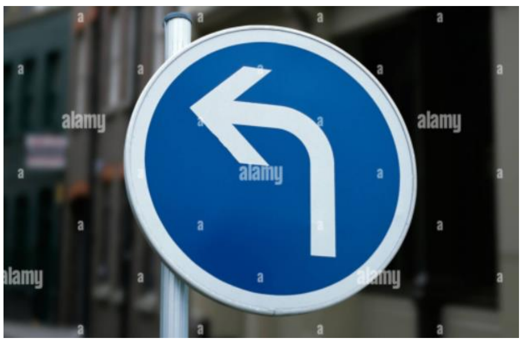
Ultimately, the choice between max pooling and average pooling (or other pooling methods) should be based on the specific requirements of the task and the characteristics of the input data. Experimentation and testing different pooling methods can help determine which approach works best for a particular problem. Certainly, In our model, both max pooling and average pooling were tested on the Le-Net model, and it was found that they worked better in different scenarios.

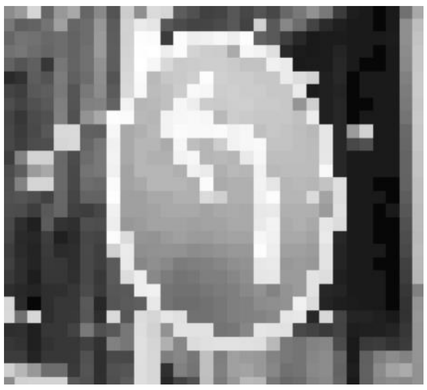
Max pooling is generally preferred in scenarios where the exact spatial location of features is not as important as their presence within a region. This is because max pooling retains the most prominent features within a given region and discards the rest. It is also known to preserve the spatial variance of the features to a greater extent than average pooling. Therefore, in cases where there is a lot of variation in the input data, max pooling may work better.

Average pooling, on the other hand, is better suited for instances where the precise geographical position of features is critical. This is since it takes into account all of the features within a region and averages them out, which can assist retain the general structure of the input data. It may also perform better in circumstances where the input data is less variable.

Image prediction is a critical task in computer vision, and it involves analysing and interpreting the content of an image to make accurate predictions. In recent years, there has been significant progress in the development of deep learning models for image prediction, which has led to the development of highly accurate and efficient image prediction systems.

To begin the process of image prediction, an image is first taken and pre-processed to prepare it for input into the prediction model. Pre-processing may involve tasks such as resizing the image, converting it to grayscale or another color space, and normalizing the pixel values to improve model performance. Figure 10(a) represents a random image and Figure 10(b) represent the result of the image after being pre-processed.



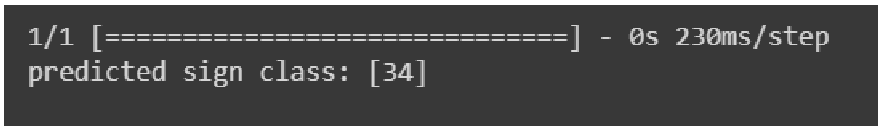


(a) (b)

Figure 10: (a) Random image of a traffic sign (b) Pre-processed image of the traffic sign

Once the image is pre-processed, it can be input into the image prediction model. The model analyses the image using various techniques such as CNNs, recurrent neural networks (RNNs), or transformers. The output of the model is a set of predicted labels or probabilities that represent the model's confidence in each possible prediction.

It is essential to evaluate the accuracy of the image prediction model to ensure that it is making accurate predictions. This can be done by comparing the predicted labels with the ground truth labels for a set of validation images. If the model's predictions match the ground truth labels, it is considered to have predicted the image correctly. Figure 11 represents the results of prediction of the model. The model predicts that the image belongs to the Class Id 34 which is correct according to the Table 1.



```
1/1 [==============================] - 0s 230ms/step
predicted sign class: [34]
```

Figure 11: Results of the prediction of the model

Appendix 1 and Appendix 2 consist of the code for Le-Net and the Modified Le-Net model respectively.

CHAPTER - 5

CONCLUSION & FUTURE WORK

5.1 Conclusion

To summarise, the optimizer used in deep learning model training is an important factor that can have a major impact on the model's performance. The Le-Net and Modified Le-Net models were trained using four different optimizers in this example, and the findings revealed that the ideal optimizer differed based on the model's design. The optimal optimizer for the Le-Net model, which has less convolutional layers, was found to be RMSProp. The ability of RMSProp to prevent overshooting and smooth out updates made it an excellent choice for this model. However, Nadam was discovered to be the best optimizer for the Modified Le-Net model, which contains more convolutional layers. Nadam's ability to quickly converge to the optimal point was critical for this model's performance.

It is critical to experiment with different optimizers and hyperparameters until the ideal combination for a certain model architecture is determined in order to obtain optimal performance in training neural networks. The best optimizer may differ depending on the model's properties, such as the number of convolutional layers, dataset size, and architecture complexity.

However, the model's performance is not only determined by the optimizer, but also by other aspects like as the quality of the dataset, the architecture, and the hyperparameters. As a result, all these factors must be evaluated and optimised in order for the deep learning model to function optimally.

Choosing the right learning rate is critical in achieving optimal performance when training neural networks. While a higher learning rate can lead to faster convergence, it increases the risk of overshooting the minimum and instability. Conversely, a lower learning rate may result in slower convergence but more stable training. We have discussed why a learning rate of 0.001 provides better results than 0.01. A learning rate of 0.01 can cause the optimization algorithm to overshoot the minimum, leading to oscillations and slow convergence. In contrast, a learning rate of 0.001 allows the optimization algorithm to take smaller steps in the direction of the gradient during backpropagation, resulting in more stable convergence and accurate predictions. Smaller learning rates also help to prevent the issue of divergence during training.

It is vital to remember that the best learning rate is determined by the individual problem being handled as well as the neural network architecture. A higher learning rate may be more acceptable for shallow networks, whereas a lower learning rate is often preferable for deeper networks due to the increased danger of overshooting and instability. To achieve the optimum performance, it is critical to carefully select the learning rate and experiment with different values.

In conclusion, choosing the correct learning rate is critical for optimising neural network training performance. A learning rate of 0.001 produces better outcomes than 0.01 because it produces more steady convergence and more accurate predictions. However, the appropriate learning rate is dependent on the specific problem and neural network architecture, and experimenting with different numbers is crucial to attain the best results.

In conclusion, regularization techniques such as dropout can significantly improve the generalization performance of neural networks by reducing overfitting. The choice of dropout percentage can have a significant impact on the performance of the model, and it is important to experiment with different values to find the optimal value for a specific problem and neural network architecture. By reducing co-adaptation between neurons and adding noise to the network during training, dropout acts as a form of regularization that can lead to better generalization performance. Regarding pooling methods, both max pooling and average pooling have their own strengths and weaknesses, and the choice between them should be based on the specific requirements of the task and the characteristics of the input data. Max pooling is generally preferred when identifying the most important features in the input data is crucial, while average pooling is more effective in identifying the overall trends in the input data. However, other pooling methods such as Lp pooling or stochastic pooling may be more effective in some scenarios.

Both max pooling and average pooling were found to operate better in distinct conditions in the context of the Le-Net model. Max pooling performed better in instances where the precise spatial position of features was less crucial than their presence within a region. Average pooling, on the other hand, was more effective in circumstances where recognising overall trends in the input data was more significant. As a result, it is critical to experiment with various pooling approaches in order to determine the best solution for a certain problem and neural network architecture. Finally, selecting the best pooling approach can assist increase the model's accuracy and generalisation performance.

5.2 Future Work

There are several possible directions for future work on traffic sign recognition systems:

- Using deep learning techniques: While current traffic sign recognition systems have attained excellent accuracy, they suffer from performance degradation in different environmental situations such as poor lighting, rain, and fog. Deep learning techniques such as convolutional neural networks (CNNs) have the potential to increase recognition accuracy and robustness in a variety of environments.

- Real-time performance optimisation: For traffic sign recognition systems to ensure driver safety, real-time performance is critical. As a result, future work might concentrate on creating optimisation strategies to minimise the system's processing cost while maintaining high accuracy.

- Multi-language support: Current traffic sign recognition systems are built primarily for a single language, limiting their usability in multilingual situations. Creating a system that recognises traffic signals in many languages will considerably expand their application in other places.

- Pedestrian recognition: Future work could focus on establishing a system that can recognise pedestrians and their behaviour in addition to recognising traffic signs. This would improve safety in regions where automobiles and people share the road.

- Data crowdsourcing: Gathering data for training traffic sign recognition systems can be time-consuming and expensive. Future research could look towards using crowdsourcing to collect massive amounts of data from various countries, languages, and driving circumstances, which could then be used to train and improve these systems' performance.

- • Integration with automated driving: As autonomous driving becomes more prevalent, traffic sign recognition systems will become increasingly more important in guaranteeing road safety. Future research could concentrate on combining these systems with autonomous driving technology in order to deliver real-time information to autonomous vehicles, thereby boosting their capacity to navigate highways safely.

REFRENCES

[1] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “The German traffic sign recognition benchmark: A multi-class classification competition,” in Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2011, pp. 1453–1460.

[2] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, “Detection of traffic signs in real-world images: The German traffic sign detection benchmark,” in Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN), Aug. 2013, pp. 1–8.

[3] Aghdam, H.H., Heravi, E.J. and Puig, D., 2016. A practical approach for detection and classification of traffic signs using convolutional neural networks. Robotics and autonomous systems, 84, pp.97-112.

[4] Bouti, A., Mahraz, M.A., Riffi, J. and Tairi, H., 2019. A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network. Soft Computing, pp.1-13.

[5] Jin, J., Fu, K. and Zhang, C., 2014. Traffic sign recognition with hinge loss trained convolutional neural networks. IEEE Transactions on Intelligent Transportation Systems, 15(5), pp.1991-2000.

[6] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[7] Zhang, J., Jin, X., Sun, J., Wang, J. and Li, K., 2019. Dual model learning combined with multiple feature selection for accurate visual tracking. IEEE Access, 7, pp.43956-43969.

[8] Zhou, Y., Shi, J., Yang, X., Wang, C., Wei, S. and Zhang, X., 2019. Rotational objects recognition and angle estimation via kernelmapping CNN. IEEE Access, 7, pp.116505-116518.

[9] Luo, W., Li, Y., Urtasun, R. and Zemel, R., 2016. Understanding the effective receptive field in deep convolutional neural networks. In Advances in neural information processing systems (pp. 4898-4906).

[10] H. Akatsuka and S. Imai, “Road signposts recognition system,” in Proc. SAE Veh. Highway Infrastructure—Safety Compatibility, 1987, pp. 189–196.

[11] D. Kellmeyer and H. Zwahlen, “Detection of highway warning signs in natural video images using color image processing and neural networks,” in Proc. IEEE Int. Conf. Neural Netw., 1994, vol. 7, pp. 4226–4231.

[12] M. de Saint Blancard, “Road sign recognition: A study of vision-based decision making for road environment recognition,” in Vision-Based Vehicle Guidance. New York: Springer-Verlag, 1992, pp. 162–172.

[13] A. D. L. Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, “Road traffic sign detection and classification,” IEEE Trans. Ind. Electron., vol. 44, no. 6, pp. 848–859, Dec. 1997.

[14] D. Ghica, S. W. Lu, and X. Yuan, “Recognition of traffic signs using a multilayer neural network,” in Proc. Can. Conf. Elect. Comput. Eng., Halifax, NS, Canada, 1994, vol. 44, pp. 848–859.

[15] A. de la Escalera, J. M. Armingol, and M. Salichs, “Recognition of traffic signs using a multilayer neural network,” in Proc. 3rd Int. Conf. Field Service Robot., Espoo, Finland, Jun. 2001, pp. 833–834.

[16] J. Miura, T. Kanda, and Y. Shirail, “An active vision system for real-time traffic sign recognition,” in Proc. IEEE Int. Conf. ITSC, 2000, pp. 52–57.

[17] D. Shaposhnikov, L. Podladchikova, A. Golovan, N. Shevtsova, K. Hong, and X. Gao, Road Sign Recognition by Single Positioning of Space-Variant Sensor, 2002. [Online]. Available: citeseer.csail.mit.edu/ 657126.html

[18] S. Hsu and C. Huang, "Road sign detection and recognition using matching pursuit method," Image Vis. Comput., vol. 19, no. 3, pp. 119–129, Feb. 2001.

[19] G. Piccioli, E. D. Micheli, and M. Campani, "A robust method for road sign detection and recognition," in Proc. ECCV, 1996, vol. 1, pp. 495– 500. [Online]. Available: citeseer.csail.mit.edu/piccioli96robust.html

[20] G. Piccioli, E. D. Michelli, P. Parodi, and M. Campani, "Robust road sign detection and recognition from image sequences," in Proc. Intell. Veh., 1994, pp. 278–283.

[21] S. Escalera and P. Radeva, "Fast gray scale road sign model matching and recognition," in Recent Advances in Artificial Intelligence Research and Development. Amsterdam, The Netherlands: IOS, Oct. 2004.

[22] G. Loy and A. Zelinsky, "Fast radial symmetry for detecting points of interest," IEEE Trans. Pattern Anal. Mach. Intell., vol. 25, no. 8, pp. 959–973, Aug. 2003.

[23] S. Maldonado-Bascon, S. Lafuente-Arroyo, P. Gil-Jimenez, H. GomezMoreno, and F. Lopez-Ferreras, "Road-sign detection and recognition based on support vector machines," IEEE Trans. Intell. Transp. Syst., vol. 8, no. 2, pp. 264–278, Jun. 2007.

[24] A. de la Escalera, J. Armingol, J. Pastor, and F. Rodriguez, "Visual sign information extraction and identification by deformable models for intelligent vehicles," IEEE Trans. Intell. Transp. Syst., vol. 5, no. 2, pp. 57–68, Jun. 2004.

[25] P. Paclik, J. Novovicova, and R. Duin, "Building road-sign classifiers using a trainable similarity measure," IEEE Trans. Intell. Transp. Syst., vol. 7, no. 3, pp. 309–321, Sep. 2006.

[26] Y. Xie, L. Liu, C. Li, and Y. Qu, "Unifying visual saliency with HOG feature learning for traffic sign detection," in Proc. IEEE Intell. Veh. Symp., 2009, pp. 24–29.

[27] X. Baró, S. Escalera, J. Vitriá, O. Pujol, and P. Radeva, "Traffic sign recognition using evolutionary Adaboost detection and forest-ECOC classification," IEEE Trans. Intell. Transp. Syst., vol. 10, no. 1, pp. 113–126, Mar. 2009.

[28] B. Hoferlin and K. Zimmermann, "Towards reliable traffic sign recognition," in Proc. IEEE Intell. Veh. Symp., 2009, pp. 324–329.

[29] Y.-Y. Nguwi and A. Kouzani, "Detection and classification of road signs in natural environments," Neural Comput. Appl., vol. 17, no. 3, pp. 265– 289, Apr. 2008.

[30] A. Bargeton, F. Moutarde, F. Nashashibi, and B. Bradai, "Improving pan-European speed-limit signs recognition with a new global number segmentation before digit recognition," in Proc. IEEE Intell. Veh. Symp., 2008, pp. 349–354.

[31] K. Lim, Jr., K. Seng, Jr., and L. Ang, Jr., "Intra color-shape classification for traffic sign recognition," in Proc. ICS, 2010, pp. 642–647.

[32] Persson, Siri. "Application of the German Traffic Sign Recognition Benchmark on the VGG16 network using transfer learning and bottleneck features in Keras," 2018.

[33] Rajesh, Reghunadhan, K. Rajeev, K. Suchithra, V. P. Lekhesh, V. Gopakumar, and N. K. Ragesh. "Coherence vector of oriented gradients for traffic sign recognition using neural networks," In The 2011 International Joint Conference on Neural Networks, pp. 907-910, IEEE, 2011.

[34] Dongfang, Zhao, Kang Wenjing, Li Tao, and Liu Gongliang. "Traffic sign classification network using inception module," In 2019 14th IEEE International Conference on Electronic Measurement & Instruments (ICEMI), pp. 1881-1890, IEEE, 2019.

[35] Sun, Ying, Pingshu Ge, and Dequan Liu. "Traffic Sign Detection and Recognition Based on Convolutional Neural Network," In 2019 Chinese Automation Congress (CAC), pp. 2851-2854, IEEE, 2019.

[36] Gracia-Garrido, M., Sotelo, M., & Martin-Gorostiza, E. (2006). Fast traffic sign detection and recognition under challenging lighting conditions. 2006 IEEE Intelligent Transportation Systems Conference, 811-816.

[37] Barnes, N., & Zelinsky, A. (2004). Real-time radial symmetry for speed sign detection. IEEE Intelligent Vehicles Symposium, 2004, 566-571.

[38] Garvila, D. M. (1999). Traffic sign recognition revisited. Informatik aktuell Mustererkennung 1999, 86-93.

[39] Zheng, Z., Zhang, H., Wang, B., & Gao, Z. (2012). Robust traffic sign recognition and tracking for Advanced Driver Assistance Systems. 2012 15th International IEEE Conference on Intelligent Transportation Systems, 704-709.

[40] Oruklu, E., Pesty, D., Neveux, J., & Guebey, J. (2012). Real-time traffic sign detection and recognition for in-car driver assistance systems. 2012 IEEE 55th International Midwest Symposium on Circuits and Systema (MWSCAS), 976-979.

[41] Greenhalgh, J., & Mirmehdi, M. (2012). Real-time detection and recognition of road traffic signs. IEEE Transactions on Intelligent Transportation Systems, 13(4), 1498-1506.

[42] Yang, Y., Luo, H., Xu, H., & Wu, F., (2014). Towards realtime traffic sign detection and classification. 17th International IEEE Conference on Intelligent Transportation Systems, 17(7), 2022-2031.

[43] C.g., K., Prabhu, L.V., A. R., & K., R. (2009). Traffic sign detection and pattern recognition using support vector machine. 2009 Seventh International Conference on Advances in Pattern Recognition, 87-90

[44] Gomez-Moreno, H., Maldonado-Bascon, S., Gil-Jimenez, P., & Lafuente-Arroyo, S. (2010). Goal evaluation of segmentation algorithms for traffic sign recognition. IEEE Transactions on Intelligent Transportation Systems, 11(4), 917-930.

[45] Berkaya, S. K., Gunduz, H., Ozsen, O., Akinlar, C., &Gunal, S. (2016) "On circular traffic sign detection and recognition." Expert Systems with Applications, 48, 67-75.

[46] Chen Li, Cheng Yang, (2016) "The research on traffic sign recognition based on deep learning.", International Symposium on Communications and Information Technologies (ISCIT), 156-161.

[47] El Margae, S., Kerroum, M. A., & Fakhri, Y. (2015) "Fusion of local and global feature extraction based on uniform LBP and DCT for traffic sign recognition." International Review on Computers and Software (IRECOS), 10(1), 52-60.

[48] T. Ojala, M. Pietikäinen, and D. Harwood, (1996) "A comparative study of texture measures with classification based on featured distributions." Pattern Recognition, 29(1), 51-59.

[49] W. Liu, J. Lv, H. Gao, B. Duan, H. Yuan, and H. Zhao, (2011) "An efficient real-time speed limit signs recognition based on rotation invariant feature." in IEEE Intelligent Vehicles Symposium (IV), 1000-1005.

[50] Wang, W., Sun, S., Jiang, M., Yan, Y., & Chen, X. (2017) "Traffic lights detection and recognition based on multi-feature fusion." Multimedia Tools and Applications, 76(13), 14829-14846.

[51] Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006) "Extreme learning machine: theory and applications." Neurocomputing, 70(1), 489-501.

[52] Huang, Z., Yu, Y., Gu, J., & Liu, H., (2017). "An efficient method for traffic sign recognition based on extreme learning machine." IEEE transactions on cybernetics, 47(4), 920-933.

[53] Wali, Safat, Mohammad A Hannan, Shahrum abdullah, Aini Hussain S.A.S., Shape Matching and Color Segmentation Based Traffic Sign Detection System, PrzeglD Elektrotechniczny, 1 (2015) 38–42.

[54] Li Y., Pankanti S., Guan W., Real-Time Traffic Sign Detection: An Evaluation Study. 2010 20th International Conference on Pattern Recognition, (2010), 3033–6.

[55] Li L., Li J., Sun J., Robust traffic sign detection using fuzzy shape recognizer. In: Ding M, Bhanu B, Wahl FM, Roberts J, editors, in proc. Pattern Recognition and Computer vision, 7496 (2009), 74960Z – 74960Z – 8.

[56] Pacl P., Novoviˇ J., Vitabile S., Gentile A., Sorbello F., Torresen J., et al., Real-Time Detection and Recognition of Road Traffic Signs, Robotics and Autonomous Systems, 13 (2012) 86–93.

[57] Soheilian B., Paparoditis N., Vallet B., Detection and 3D reconstruction of traffic signs from multiple view color images, ISPRS Journal of Photogrammetry and Remote Sensing, 77 (2013) 1–20.

APPENDIX A

A.1 Pre-processing

```python
def grayscale(img):
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  return img
```

```python
def equalize(img):
  img = cv2.equalizeHist(img)
  return img
```

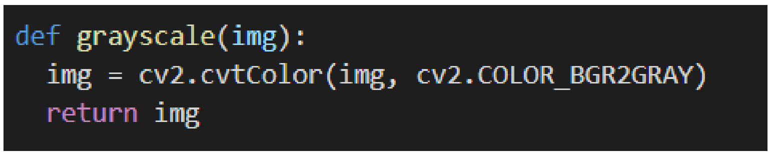
```python
def preprocessing(img):
  img = grayscale(img)
  img = equalize(img)
  img = img/255
  return img
```

A2. Architecture of Le-Net model

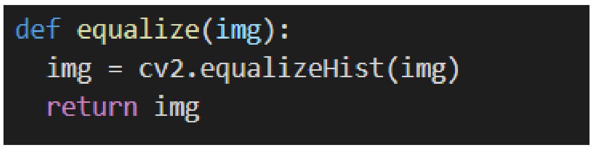
```python
def leNet_model():
  model = Sequential()
  model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
  model.add(MaxPooling2D(pool_size = (2, 2)))
  model.add(Conv2D(30, (3, 3), activation = "relu"))
  model.add(MaxPooling2D(pool_size = (2, 2)))

  model.add(Flatten())
  model.add(Dense(500, activation = 'relu'))
  model.add(Dropout(0.5))
  model.add(Dense(num_classes, activation = 'softmax'))
  model.compile(Adam(lr = 0.01), loss = 'categorical_crossentropy', metrics = ['accuracy'])
  return model
```

APPENDIX B

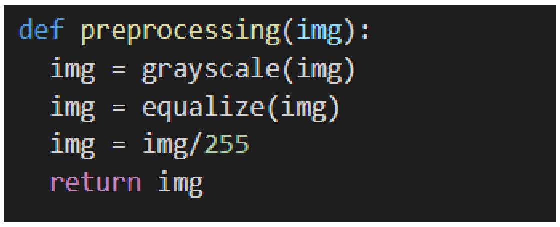
A1. Pre-processing



```
def grayscale(img):
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  return img
```

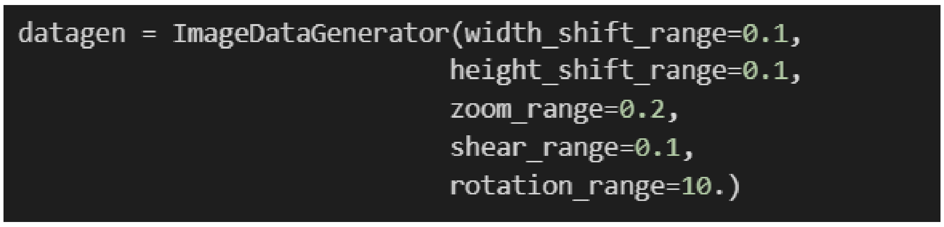


```
def equalize(img):
  img = cv2.equalizeHist(img)
  return img
```



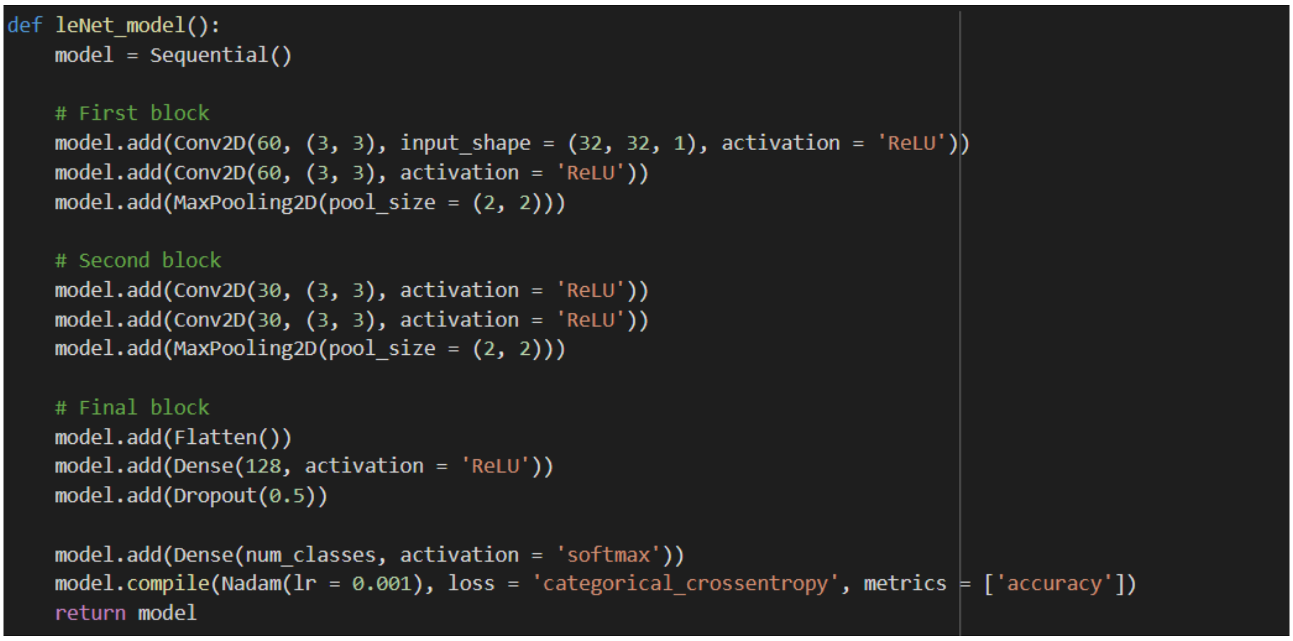
```
def preprocessing(img):
  img = grayscale(img)
  img = equalize(img)
  img = img/255
  return img
```

A2. Data Augmentation



```
datagen = ImageDataGenerator(width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10.)
```

A3. Architecture of Modified Le-Net model



```
def leNet_model():
    model = Sequential()

    # First block
    model.add(Conv2D(60, (3, 3), input_shape = (32, 32, 1), activation = 'ReLU'))
    model.add(Conv2D(60, (3, 3), activation = 'ReLU'))
    model.add(MaxPooling2D(pool_size = (2, 2)))

    # Second block
    model.add(Conv2D(30, (3, 3), activation = 'ReLU'))
    model.add(Conv2D(30, (3, 3), activation = 'ReLU'))
    model.add(MaxPooling2D(pool_size = (2, 2)))

    # Final block
    model.add(Flatten())
    model.add(Dense(128, activation = 'ReLU'))
    model.add(Dropout(0.5))

    model.add(Dense(num_classes, activation = 'softmax'))
    model.compile(Nadam(lr = 0.001), loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
```

PLAGRISM REPORT

Project Report

ORIGINALITY REPORT

18%	12%	13%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Limerick Student Paper	1%
2	M Swathi, K. V. Suresh. "Automatic traffic sign detection and recognition: A review", 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), 2017 Publication	1%
3	iq.opengenus.org Internet Source	1%
4	Submitted to Middle East College of Information Technology Student Paper	1%
5	easychair.org Internet Source	1%
6	www.cvc.uab.es Internet Source	1%
7	Submitted to CSU, San Jose State University Student Paper	<1%