Automated Traffic Sign Recognition via CNN Deep Learning

A Project Report submitted in the partial fulfillment of the

Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted By

NARENDRA YENUGANTI (21471A05K9)

RAMU AVULA (21471A05E3)

ADI MADAM (21471A05I0)

Under the esteemed guidance of

K.V.Narasimha Reddy, M.Tech.

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Cycle -1 NIRF

rank in the band of 251-320 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2024-2025

NARASARAOPETA ENGINEERING COLLEGE

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "AUTOMATED TRAFFIC SIGN RECOGNITION via CNN DEEP LEARNING" is a bonafide work done by the team NARENDRA YENUGANTI(21471A05K9), RAMU AVULA(21471A05E3), ADI MADAM(21471A05I0), in partial fulfilment of the requirements for the award of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during 2024-2025.

PROJECT GUIDE

K.V.Narasimha Reddy, M.Tech..
Assistant Professor

PROJECT CO-ORDINATOR

Dr. M. Sireesha, M.Tech., Ph.D.
Associate Professor

HEAD OF THE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D.
Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled **"Automated Traffic Sign Recognition via CNN Deep Learning"** is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has not been submitted for any other degree or professional qualification except as specified.

By

NARENDRA.Y (21471A05K9)

ADI REDDY.M (21471A05I0)

RAMU.A (21471A05E3)

ACKNOWLEDGEMENT

We wish to express our thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman sri **M.V.Koteswara Rao,B.Sc.,** who took keen interest in us in every effort throughout this course. We owe out sincere gratitude to our beloved principal **Dr.S.Venkateswarlu, Ph.D.,** for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr.S.N.Tirumala Rao, M.Tech.,Ph.D.,** HOD of CSE department and also to our guide **K.V.Narasimha Reddy, M.Tech..** Assistant Professor of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Dr.M.Sireesha, M.Tech.,ph.D.,** Associate professor &Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech. degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying our doubts which had really helped us in successfully completing our project.

By

NARENDRA.Y (21471A05K9)

ADI REDDY.M (21471A05I0)

RAMU.A (21471A05E3)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community,

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

NARASARAOPETA
NEC ENGINEERING COLLEGE
(AUTONOMOUS)

Program Outcomes

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. Low level
2. Medium level
3. High level

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathering the requirements and defining the problem, plan to develop a Automated Traffic Sign Recognition via CNN Deep Learning	PO1, PO3
CC421.1	Each and every requirement is critically analyzed, the process model is identified and divided into five modules	PO2, PO3
CC421.2	Logical design is done by using the unified modelling language which involves individual team work	PO3, PO5, PO9
CC421.3	Each and every module is tested, integrated, and evaluated in our project	PO1, PO5
CC421.4	Documentation is done by all our three members in the form of a group	PO10
CC421.5	Each and every phase of the work in group is presented periodically	PO10, PO11
C2202.2,C2203.3, C1206.3	Implementation is done and the project will be handled by the users using this project to detect and classify traffic signs.	PO4, PO7
C32SC4.3	The physical design includes the website to classify the Traffic Signs from the input images.	PO5, PO6

ABSTRACT

The rapid development of the road traffic systems is a very important component of the nation's infrastructure, and that is reflected in growing importance of the traffic safety. Traffic Sign Recognition (TSR) is an important field of research because traffic offenses, particularly the disregard for traffic signs, are a major contributor to accidents. This paper gives a comprehensive review of the latest developments in the area of traffic sign detection and recognition techniques with attention to the applications of CNNs. This paper discusses the need for traffic signs to be detected under complex conditions and proposes an architecture based on modified CNN that helps reduce the processing time and increase accuracy. Improving recognition accuracy in realistic environments is the motivation behind the CNN model, which has been architected for real-time training as well as target identification. Experiments suggest that this solution outperforms present intelligent driving systems and the state-of the-art performance achieved by the image processing algorithms currently in use and traffic sign datasets.

INDEX

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The computer vision and Machine Learning problem of ”Traffic Sign Recognition” (TSR) aims to recognize and classification of the traffic signs from image or video streams. In essence, it is meant to develop models and algorithms that are capable of automatically recognizing and interpreting many types of traffic indicators, ranging from yield signs and stop signs to speed restrictions among others[1]. Due to this, TSR is essential in supporting drivers to follow traffic laws, hence it ought to be kept up to date with contemporary road safety. Given the nature of this technology, human errors on roads could be greatly minimized and incidences of road accidents could be avoided. These technologies also reduce vehicle emissions and fuel consumption by encouraging people to respect the set speed and other provisions of traffic law[2]. In cities, which often suffer from high pollution levels, this is particularly important. Using video feed coming from cameras mounted on vehicles, TSRs are made to recognize traffic signs promptly. For such systems, image processing techniques are used for sign recognition and classification according to form, color, and symbols[3]. The procedure followed for the classification of signs is comparison-based: In this, the system’s database forms a predefined set of templates against which the detected signs are compared. Traffic sign recognition technologies are fast becoming a critical necessity in light of ever-burgeoning traffic congestion and accidents. These devices identify and decode traffic signs, hence giving motorists information and guidelines on what actions to under take [4].Some of the major benefits of Traffic sign recognition include its ability to push forward road safety as it ensures that crashes are prevented and reductions in deaths occur with fast notifications of traffic signs. Alam et al. [5,6] had developed SURF, which in prior work could be referred to as a feature identification and description approach for traffic sign recognition. It is suggested in the paper [7] that the best strategy for traffic sign identification is to integrate grid search, SVM classifier, and HOG features. There are many techniques to develop a traffic sign recognition system, like Machine Learning algorithms on CNN [8,9], template matching [10], rule-based systems [11,12] etc.

2. LITERATURE SURVEY

Future technologies used to incorporate cloud computing, artificial intelligence, and Internet of Things will eventually help improve the intelligent driving technologies. TSR is therefore an essential element in intelligent transportation systems since the whole safety of driving depends on the understanding of traffic signs and the realization of appropriate avoidance and moving actions. An effective TSR system can ensure the safety of driver by fast and proper provision of real time traffic information to drivers to make proper decisions or let the vehicle drive on its own to avoid danger. It is in this concern that traffic sign detection technology is at present being emphasized on a research involving important traffic sign identification technologies. Through the evolution of traffic sign identification, we have categorized common detection techniques into four major groups, such as color based techniques, shape-based techniques, and Deep Learning based techniques

A. Detection based on color-feature: In general, there are three kinds of traffic signs: warning, indication, and prohibition. According to the categories, each type has quite distinct color characteristics. On the RGB color space, Akatsuka et al [5]. marked out the red, yellow, and blue colors with threshold segmentation algorithms to finish the traffic sign detection within a threshold range of distinct colors. Based on this, in the 1980s.A sign-detection system based on RGB color was developed. Zhangka et al. counted all color details of sign images, calculated the red, yellow, and blue thresholds of statistical distribution, and extracted the global color characteristics. Huang Zhiyong et al. used RGB mapping on a row of three-component differences, which supports derivational empirical thresholds, splitting of recognition indications, and avoiding multiplications fully in this technique.

B. Shape-feature-based detection: Anandhalli and Baligar [13] utilized a Harris corner detector in order to locate triangle and rectangle symbols in the ROI, while locating corners in a defined control region. Li [14] exploited edge information during the detection of occluded traffic signs during driving like circle, triangle, and rectangles were identified within the image and masked over 95% of all traffic signs by the use of the nonparametric shape detector with form properties of scale invariant edge turning angles .From the multi-feature fusion of the traffic sign recognition methods, Jin et al.

[15] proposed a two-module detector as follows: the first module uses the commonality of symbol borders to extract ROIs and the second module uses HOG and SVM to validate the effectiveness of the produced ROIs to detect the traffic signs of dataset. Zheng et al[16] had proposed sliding window detection(SWD). where this approach integrates channel feature classifier to search for the presence of traffic signals on different sizes. Gim et al. [16] has come up with a system that has two coarse filter modules specifically for the required and prohibited signs in the GTSDB; the first module is HOG and LDA-based. Both modules use an SVM classifier and operate with huge windows; the second uses a small sliding window. Although these attempts prove to be successful in traffic sign recognition with graphical approaches, they are not very good when it comes to complicated situations (like dim lighting, partly covered signs, etc.), and especially bad in recognizing signs with different orientations or perspectives.

C. Detection using Deep Learning: It utilizes learning and training to identify traffic signs and extract characteristics, which is extremely distinct from previous techniques. Relatively, the Deep Learning-based approach applies a better accuracy level and more robust generalization ability compared to the classical traffic sign detection approaches. Features acquisition through big data, strong feature expression capabilities, insensitivity to influences from outside factors such as illumination and occlusion, it is actually a fundamental approach in itself. The algorithm of target detection has high detection accuracy; its basis is Prospective Region extraction. Girshick et al [5].designed a target detection system based on a Prospective Region, which is also popularly referred to as RCNN (Regions with CNN characteristics). True identification of objects and separation of semantic content requires a deep feature hierarchy which used Convolutional Neural Network (CNN). This it categorizes item suggestions, achieving an accuracy of object identification. However, it consumes much computational time and space as it repeats the extraction of every qualifying region and stores it in its memory. Meanwhile, region stretching applied by RCNN pool together every Prospective Region into a 227 × 227 size thus reducing the accuracy for the detection and further reducing the quality of the features CNN extraction. FPN combines, depth feature ,shallow feature to produce predictions on the several scales of feature pyramid scale. Candidate areas are extracted by layering an RPN network over the feature pyramid to achieve this. Such a concept raises the semantic quality of the shallow feature map and amplifies the precision in detecting tiny targets. One is YOLO network, which

enforces a single view regarding detection-not as many classification or regression tasks but applies regression on the entire image for predicting the coordinates of the bounding boxes and class probabilities.

2.1 Deep Learning

Deep Learning (also known as deep structured learning) is part of a broader family of Machine Learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep Learning architectures such as deep neural networks, deep belief networks, graphneural networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analogue.

The adjective "Deep" in Deep Learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a nonpolynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In Deep Learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the"structured" part.

2.2 Some Deep Learning Models:

i. Feedforward neural network:

- This type of neural network is the very basic neural network where the flow control occurs from the input layer and goes towards the output layer.
- These kinds of networks are only having single layers or only 1 hidden layer.
- Since the data moves only in 1 direction there is no backpropagation technique in this network.

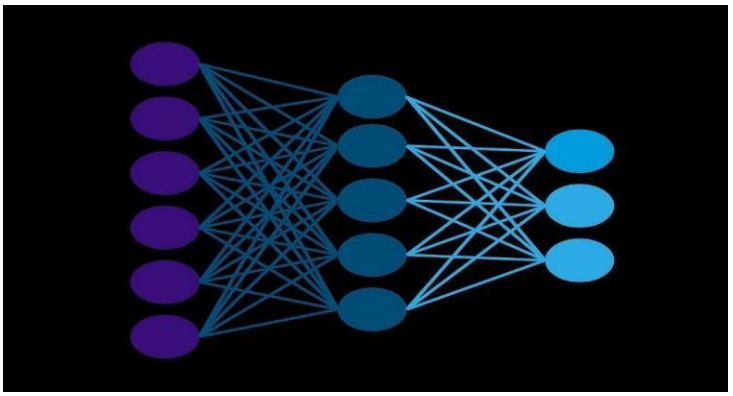

Fig: 2.1: Feed Forward Neural Network

- In this network Fig. 2.1, the sum of the weights present in the input is fed into the input layer.
- These kinds of networks are used in the facial recognition algorithm using computer vision.

ii. Radial basis function neural networks:

- This kind of neural networks have generally more than 1 layer preferably two layers
- In this kind of networks, the relative distance from any point to the center is calculated and the same is passed towards the next layer
- Radial basis networks are generally used in the power restoration systems to restore the power in the shortest span of time to avoid the blackouts.

iii. Multi-Layer perceptron:

- This type of network Fig2.2 are having more than 3 layers and its used to classify the data which is not linear
- These kinds of networks are fully connected with every node.
- These networks are extensively used for speech recognition and other Machine Learning technologies.

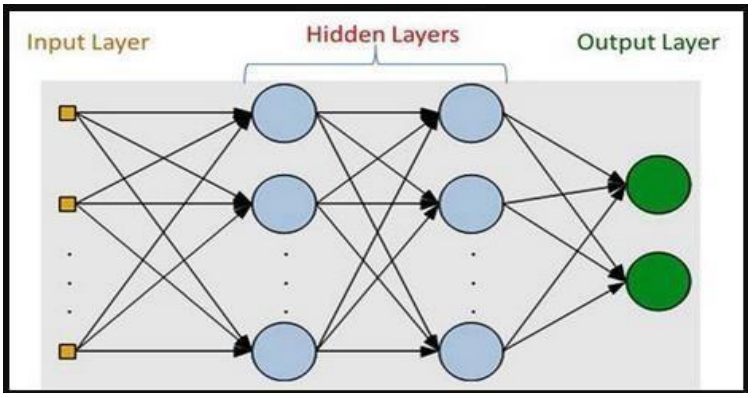

Fig: 2.2. Multi-Layer Perceptron

Convolution Neural Network:

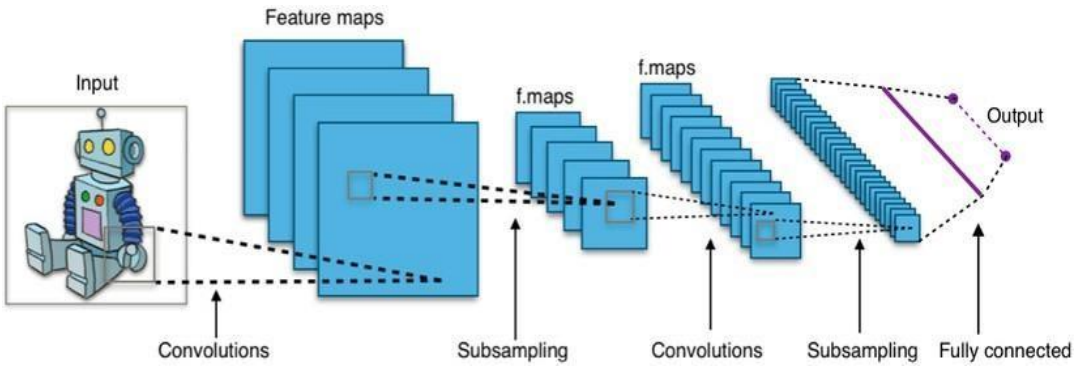

Fig: 2.3. Convolutional Neural Network

- CNN Fig.2.3 is one of the variations of the multilayer perceptron.
- CNN can contain more than 1 convolution layer and since it contains a convolution layer the network is very deep with fewer parameters.
- CNN is very effective for image recognition and identifying different image patterns.

iv. Recurrent Neural Network:

Recurrent Neural Networks (RNNs) Fig.2.4 work a bit different from regular neural networks. In neural network the information flows in one direction from input to output. However in RNN information is fed back into the system after each step. Think of it like reading a sentence, when you're trying to predict the next word you don't just look at the current word but also need to remember the words that came before to make accurate guess.

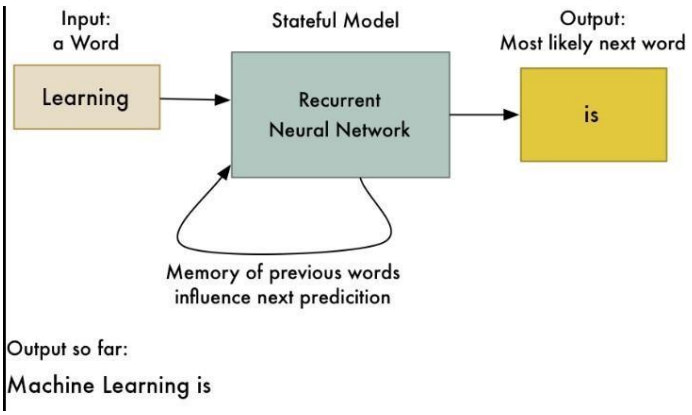

Fig: 2.4 Recurrent Neural Network

v. Modular Neural Network:

- This kind of network Fig.2.5 is not a single network but a combination of multiple small neural networks.
- All the sub-networks make a big neural network and all of them work independently to achieve a common target.

- These networks are very helpful in breaking the small-large problem into small pieces and then solving it.

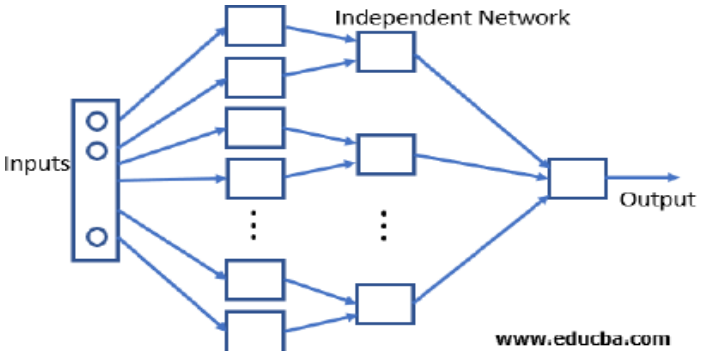

Fig: 2.5: Modular Neural Network

Sequence to sequence models:

- This type of network is generally a combination of two RNN networks.
- The network works on the encoding and decoding that is it consists of the encoder which is used to process the input and there is a decoder which processes the output
- Generally, this kind of network is used for text processing where the length of the inputted text is not as same as outputted text.

2.3 Applications of Deep Learning:

Applications of Deep Learning as shown in the Fig:2.6

Fig: 2.6: Applications of Deep Learning

- Self-Driving Cars
- News Aggregation and Fraud News Detection
- Natural Language Processing
- Virtual Assistants

- Entertainment
- Visual Recognition
- Fraud Detection
- Healthcare
- Personalisations
- Detecting Developmental Delay in Children
- Colourisation of Black and White images
- Adding sounds to silent movies
- Automatic Machine Translation
- Automatic Handwriting Generation
- Automatic Game Playing
- Language Translations
- Pixel Restoration
- Photo Descriptions
- Demographic and Election Predictions
- Deep Dreaming

Characteristics of Deep Learning

1. **Supervised, Semi-Supervised or Unsupervised**

   When the category labels are present while you train the data then it is Supervised learning. Algorithms like Linear regression. Logistic regression, decision trees use Supervised Learning. When category labels are not known while you train data then it is unsupervised learning. Algorithms like Cluster Analysis, K means clustering, Anomaly detection uses Unsupervised Learning. The data set consists of both labeled and un labelled data then we call it is Semi-Supervised learning. Graph-based models, Generative models, cluster assumption, continuity assumption use Semi-Supervised learning.

2. **Huge Amount of Resources**

   It needs advanced Graphical Processing Units for processing heavy workloads. A huge amount of data needs to be processed like Big data in the form of structured or unstructured data. Sometimes more time also required to process the data, it depends on the amount of data fed in.

3. **Large Amount of Layers in Model**

   A huge amount of layers like input, activation, the output will be required, sometimes the output of one layer can be input to another layer by making few small findings and then these findings are summed up finally in the soft-max layer to find out a broader classification for final output.

4. **Optimizing Hyper-parameters**

   Hyperparameters like no of epochs, Batch size, No of layers, Learning rate, needs to be tuned well for successful Model accuracy because it creates a link between layer predictions to final output prediction.

   Cost Function It says how well the model performance in prediction and accuracy. Foreach iteration in Deep Learning Model, the goal is to minimize the cost when compared to previous iterations. Mean absolute error, Mean Squared Error, Hinge loss, Cross entropy are different types according to different algorithms used.

Advantages of Deep Learning

- Solve Complex problems like Audio processing in Amazon echo, Image recognition, etc, reduce the need for feature extraction, automated tasks wherein predictions can be done in less time using Kera's and Tensor-flow.
- Parallel computing can be done thus reducing overheads.
- Models can be trained on a huge amount of data and the model gets better with more data.
- High-Quality Predictions when compared with humans by training tirelessly.
- Works well-unstructured data like video clips, documents, sensor data, webcam data, etc.

2.4 Deep Learning Algorithms

To create a Deep Learning model, one must write several algorithms, blend them together and create a net of neurons. Deep Learning has a high computational cost. To aid Deep Learning models, there are Deep Learning platforms like Tensor flow, Py-Torch, Chainer, Kera's, etc. In Deep Learning, we have tried to replicate the human neural network with an artificial neural network; the human neuron is called perceptron in the Deep Learning model. We connect these perceptron units together to create a neural network; it has 3 sections:

i. Input layer
ii. Hidden layers
iii. Output layer

A perceptron has input nodes (dendrites in the human brain), an actuation function to make a small decision and output nodes (axon in the human brain). We will see how one perceptron works; connecting them together will create a Deep Learning model. Input information (number of input variables/features) are assigned some weight and fed to the actuation function. The actuation function makes a decision and sends output. This perceptron's output will be input toother neurons. Once the batch is processed, with backpropagation error is calculated at each neuron, with the help of a cost function/ cross-entropy. In this way, input weights are reassigned, and the whole process continues until cross-entropy satisfies the condition. We have different actuation functions like Sigmoid functions, hyperbolic tangent function, Rectified Linear Unit (Re-LU) to take a small decision. A Deep Learning model needs a vast amount of data to build a good model. Generally, a model with more than 3 hidden layers is treated as a deep neural network. Basically, Deep Learning is a set of neurons with a number of parameters defined for each layer.

2.5 Architectural Methods for Deep Learning Algorithms

To build this architecture following algorithms are used:

i. Back Propagation

In this algorithm, we calculate partial derivatives. In general, the gradient descent method for optimization, derivatives (gradients) are calculated at each iteration. In Deep Learning, functions are not simple; they are the composition of different functions. In this case, it is hard to calculate gradients, so we use approximate differentiation to calculate derivatives. The more the number of parameters, the more expensive approximate differentiation will be.

ii. Stochastic Gradient Descent

In Gradient descent, the goal is to find global minima or optimum solution. But to get that, we have to consider local minima solutions (not desirable) also. If the objective function is a convex function, it is easy to find the global minima. The initial value for the function and learning rate are deciding parameters for finding global minima. This can easily be understood by considering a river from

the mountain top and searching for a foothill (global minima). Butin the way, there will be some ups and downs (local minima) which must be avoided. The river originating point and speed (initial value and learning rate in our case) are deciding factors to find global minima.

iii. Learning Rate

The learning rate is like the speed of the river; it can reduce training time and increase performance. In general, to learn any technique/sport, in the beginning, the learning rate is relatively high than at the end when one is to master it. After the intermediate stage, the learning will be slow; the focus will be on fine-tuning. The same is applied in Deep Learning; too large changes are tackled by a higher learning rate and by slowly decreasing the learning rate later for fine-tuning.

vi. Batch Normalization

In Deep Learning initial value of weight (randomly chosen) and learning rate is defined for a minibatch. In the beginning, there would be many outliers, and during backpropagation, these outliers must be compensated to compute the weights to get output. This compensation results in extra epochs. So to avoid it, we use batch normalization.

v. Drop Out

In Deep Learning, we generally encounter the problem of overfitting. Overfitting in large networks with several parameters makes it difficult to predict on test data. So, to avoid that, we use the dropout method, which drops random units during training by creating different 'thinned networks'. When testing these thinned networks' predictions are averaged, which helps to avoid overfitting.

vi. Bag of Words

We use a continuous bag of words to predict the next word. For e.g., we see in email writing the autosuggestion for completing the sentence is part of NLP. This is done by considering lots of sentences and for a specific word surrounding words that are captured. These specific words and surrounding words are fed to the neural network. After the training model, it can predict the specific word based

on the surrounding words.

vii. Long Short-Term Memory

LSTM is very useful in sequence prediction problems like language translation, predicting sales and finding the stock price. LSTM has the edge over other techniques because it is able to consider previous data. LSTM makes modification by cell states mechanism. It remembers to forget things. The 3 main aspects of LSTM make it stand out from other Deep Learning techniques. First is when the neuron should have input, second when to remember previous data and what to forget and third is when to pass output.

2.6 Libraries of Deep Learning

All the libraries which are generally used for Deep Learning are open source and few of them are as follows:

- TensorFlow
- deeplearning4j
- Torch
- Caffe
- Microsoft CNTK
- ML.NET
- Theano
- Deepmat
- Neon

i. TensorFlow

- TensorFlow is the Machine Learning and Deep Learning library developed by Google and it came into the market around the 2016 march.
- TensorFlow grew out of an in-house library of google brain known as DistBelief.
- Currently, TensorFlow is the leading and most used library in the market. Different types of deep nets can be developed and also the various packages available in this library are used to attain and address most of the tasks and problems in the field of Deep Learning.
- This library is completely written in python and so it’s easy to use for the

python programmers.

- Due to a flexible computational graphical structure of TensorFlow, this library is not only limited to Deep Learning operations it can be used for many different operations and applications.
- TensorFlow provides a layer or we can say more of a wrapper, known as Keras which is used to access the different packages and methods easily of TensorFlow.
- Google provides a very well documentation for this library where every small intricacies and usage are mentioned anyone can refer that and use the library.
- TensorFlow is a very fast-evolving library, this library can be used for educational purposes as well as huge large scale commercial application can also be built.
- Google has developed this library for Mobil platforms as well which is known as TensorFlow lite.
- TensorFlow is the only library which provides support for Python, Java, C++, javascript and swift programming language, for Javascript TensorFlow.js
- TensorFlow has also support for GPU and big data.

ii. Deeplearning4j

- Deeplearning4j is the open-source java library which only supports java programming language and this library is written in Java.
- This was developed by Adam Gibson to provide distributed multimode capabilities for deep neural networks.
- This library is very much use full for the application which is having build on top of big data.
- This library works with Scala and also provide inbuilt GPU support.

iii. Torch

- This open-source Deep Learning library was developed by Facebook and Twitter.
- This library is written in Lua programming language.

iv. Caffe

However PyTorch is the library which is widely used, and it's written in a

Python programming language.

- Caffe is an open-source Deep-Learning library written in C++/CUDA and developed by Yangqing Jia of Google.
- This library was first developed for computer vision tax.
- Caffe gives permission to the user to configure the hyperparameters for a deep net.
- The layer configuration is very robust and very much sophisticated.

v. **Theano**

- This is the open-source Deep Learning library written in Python and CUDA.
- This library is very similar to the TensorFlow library but the implementation and usage are not that simple as that of TensorFlow.
- This library is generally used for educational and research purposes.
- Theano is not that easy to use and many Deep Learning libraries extend the features of this library to help ease the life of the developer for coding the Deep Learning models.
- Theano is fastest amongst most of the libraries mentioned because it makes use of vectors and matrices for all the functions and the vectorized code runs faster since parallel processing for the multiple values makes the things faster.

vi. **Microsoft CNTK**

- This is a cognitive toolkit developed by Microsoft to venture in the field of Artificial intelligence.
- This library is written in python and its supports the other packages and libraries which python programming language supports, and it comes with Microsoft visual studio.
- CNTK is used to describe neural networks as a series of computational directed graphs.
- ML.NET is the open-source library which is also developed by Microsoft for the dotnet developers.

vii. **ML.NET**

- This library is written in C# and F# and it uses the Microsoft dot net platform
- With the library, it becomes easy to create desktop as well as large scale web applications which can bring the vast possibility of the Machine Learning

algorithm to the end-user.

viii. **Deep mat**

- This library is developed in MATLAB.
- With the use of this library, we can implement Deep Learning using MATLAB. With this library GSN, CNN, Restricted Boltzmann machine, Deep belief networks, multi-layer. Perceptron, and many more artificial neural networks.

ix. **Neon**

- Neon is a Deep Learning framework created by the Nervana systems to deliver industry-leading cutting-edge technologies.
- This framework has been depreciated as of 2018 and further research has been carried out by Intel corporation on the same.
- As per the Intel corporation website, alternative frameworks are asked to be used such as Intel optimization for tensor Flow, Intel optimization for Caffe, pytorch, etc.

2.7 Layers in Convolutional Neural Networks

Below are the Layers of convolutional neural networks:

a. **Image Input Layer:**

The input layer gives inputs (mostly images), and normalization is carried out. Input size has to be mentioned here.

b. **Convolutional Layer:**

Convolution is performed in this layer. The image is divided into perceptrons(algorithm); local fields are created, leading to the compression of perceptrons to feature maps as a matrix with size m x n.

c. **Non-Linearity Layer:**

Here feature maps are taken as input, and activation maps are given as output with the help of the activation function. The activation function is generally implemented as sigmoid or hyperbolic tangent functions.

d. **Rectification Layer:**

The crucial component of CNN, this layer does the training faster without reducing accuracy. It performs element-wise absolute value operation on activation maps.

e. **Rectified Linear Units(Re-LU):**

Re-LU combines non-linear and rectification layers on CNN. This does the threshold operation where negative values are converted to zero. However, Re-LU doesn't change the size of the input.

f. Pooling Layer:

The pooling layer is also called the down sampling layer, as this is responsible for reducing the size of activation maps. A filter and stride of the same length are applied to the input volume. This layer ignores less significant data; hence image recognition is done in a smaller representation. This layer reduces overfitting. Since the amount of parameters is reduced using the pooling layer, the cost is also reduced. The input is divided into rectangular pooling regions, and either maximum or average is calculated ,which returns maximum or average consequently. Max Pooling is a popular one.

g. Dropout Layer:

This layer randomly sets the input layer to zero with a given probability. More results in different elements are dropped after this operation. This layer also helps to reduce overfitting. It makes the network to be redundant.

h. Fully Connected Layer:

Activation maps, which are the output of previous layers, is turned into a class probability distribution in this layer. FC layer multiplies the input by a weight matrix and adds the bias vector.

i. Output Layer:

FC layer is followed by soft-max and classification layers. The soft-max function is applied to the input. The classification layer computes the cross- entropy and loss function for classification problems.

j. Regression Layer:

Half the mean squared error is computed in this layer. This layer should follow the FClayer.

Common steps for any TensorFlow based Algorithms:

The basic steps of TensorFlow algorithm are:

Step 1: Data is Imported/Generated:

TensorFlow Models depends heavily on the huge amount of Data. Either you can import your own dataset or TensorFlow also comes with the collection of Type this command to check out available datasets in TensorFlow.

```
import TensorFlow as tf
import TensorFlow datasets as tendata
#This command will generate a list of datasets available in the TensorFlow
print(tfds.list_builders())
```

Step 2: Data Normalization or Transformation:

If the data is not in the appropriate forum. The Batch Normalization is the command approach used to normalize data in the TensorFlow.

Step 3: Set the Parameters of the Algorithm:

For eg; the number of Iterations, Learning rate,etc.

Step 4: Set and initialize the variables and Placeholders:

Variables and Placeholders are two basic programming Elements of the TensorFlow. Variables hold the state of the graph and placeholders are used to feed the data in the graph at the later date.

Step 5: Create Model structure:

What operations will be performed on the data is defined.

Step 6: Define the Loss Function:

It calculates the difference between predicted values and actual values. It tells how well your model is trained basically used to evaluate the output.

Step 7: Train Model:

Initialize computational graph and create an Instance of a graph. Feed data into the model with the help of placeholders and let the TensorFlow do the rest of the processing for better predictions.

Step 8: Evaluate the performance:

Evaluate the model by checking with new data.

Step 9: Predict the Outcome:

Also checks your model on new and unseen data. To better visualize model TensorFlow provides Tensor board. It helps us to visualize any statistics of the neural network, debug and optimize them. You can check what happens in the code and will give you a detailed understanding of the inner working. You can fix problems very easily with the help of this tool. Tensor board provides five types of

Visualizations:

- Scalars
- Images
- Audio
- Histograms
- Graphs

3. EXISTING SYSTEM

The early evolution of image-based traffic sign detection primarily relied on conventional computer vision techniques and Random Forest Classifier (RFC). Input images underwent conversion into various color spaces, with features such as color, shape, edges, and gray-level intensity utilized for detection. Traffic signs convey crucial information essential for describing contemporary traffic scenarios, delineating right-of-way, enforcing regulations, warning of potential hazards, and more. In practical driving scenarios, road signs aid drivers in navigation and recognition, facilitated by computer vision technology. However, real-world road conditions pose significant challenges, making it arduous for researchers to develop efficient systems that operate with high accuracy. While existing systems have successfully detected and classified signs using standard computer vision methods, they often suffer from prolonged processing time.

DISADVANTAGES:

Here are some potential Disadvantages that can be addressed:

1. Adaptability to Diverse Environments.
2. Real-time Processing for Autonomous Vehicles.
3. Generalization Across Geographical Regions.
4. Handling Occlusions and Complex Traffic Scenarios.
5. Robustness to changes in sign appearance over time.

4. PROPOSED SYSTEM

This section presents the image-based traffic sign detection system on which the proposed detection methods will be executed Fig 4.1. The system will be described in term of the functional blocks and the physical hardware. Fig. 3 displays the functional blocks of the system including

i. Input image,
ii. Regions Preprocessing
iii. CNN based feature extraction.
iv. Classification.
v. Traffic sign labelling.

The physical hardware of the system consists of a CCD camera and image grabber card for the image acquisition unit and the computer system for the image processing and the display unit. Following subsections describe the image acquisition and the image processing unit in terms of function and hardware specifications.

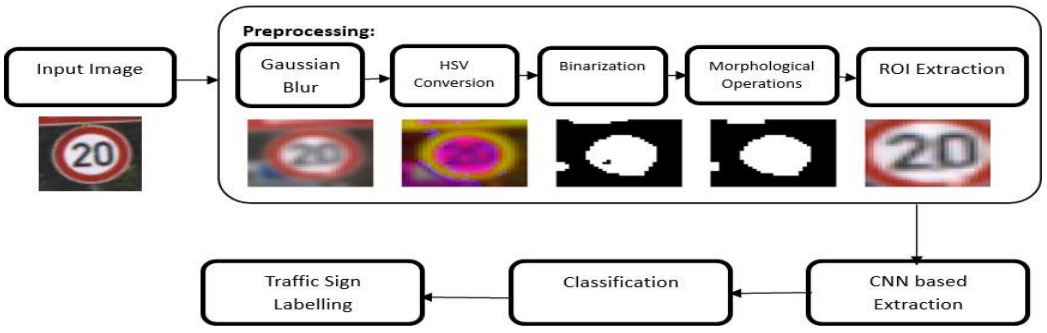

Fig: 4.1: Proposed System

A. Input Image:

- The raw traffic sign image is taken as input.
- This image can be captured from a camera (e.g., in autonomous vehicles) or loaded from a dataset.
- The image may contain noise, variations in lighting, and distortions, requiring preprocessing before classification.

B. Preprocessing:

Preprocessing enhances the image quality and extracts important features before classification. It includes:

i. Gaussian Blur:

- A smoothing filter is applied to reduce noise and minor details.
- Helps in removing unwanted high-frequency information (sharp edges, grainy textures).
- Prevents overfitting in Deep Learning models by removing unnecessary variations.

ii. HSV Conversion:

- The image is converted from RGB (Red-Green-Blue) to HSV (Hue-Saturation-Value) color space.
- HSV allows better segmentation of traffic signs because the hue component is less affected by lighting changes.
- Helps in distinguishing colors accurately, especially for signs with red, blue, or yellow elements.

iii. Binarization:

- Converts the HSV image into a binary (black-and-white) format using a thresholding technique.
- Segments the foreground (traffic sign) from the background.
- Makes it easier for the model to identify important regions.

iv. Morphological Operations:

- Uses operations like erosion, dilation, opening, and closing to refine the binarized image.
- Erosion: Removes small noise pixels.
- Dilation: Enhances the detected traffic sign by filling small gaps.
- Closing: Connects broken parts of the traffic sign's edges.
- This step ensures a clean and continuous shape of the traffic sign.

v. ROI (Region of Interest) Extraction:

- Extracts the region containing the traffic sign while removing unnecessary background.
- This step ensures that only the traffic sign is passed to the classification model.

- The extracted ROI is resized to match the input size required by the CNN model.

C. CNN-Based Feature Extraction:

- The Convolutional Neural Network (CNN) processes the extracted ROI to learn important patterns.
- CNN layers (Conv2D, Pooling, Dropout, etc.) extract spatial features such as:
  i. Shape and structure of the sign.
  ii. Color distribution.
  iii. Edge and texture patterns.
- CNN learns from labeled training data and makes predictions based on learned features.

D. Classification:

- The extracted features are passed to fully connected layers in the CNN for classification.
- The model assigns a probability to each traffic sign class.
- The sign is classified into one of 43 predefined categories based on the highest probability score.

E. Traffic Sign Labelling:

- The classified sign is labeled with its corresponding name (e.g., "Speed Limit 20", "Stop", "No Entry").
- The output label is displayed to the user or used in real-time applications (e.g., autonomous driving).

5. SYSTEM REQUIREMENTS

5.1 Hardware Requirements

CPU	: Intel Core i5 or above
Cache Memory	: 12MB
RAM	: 8 GB
Storage	: 512GB SSD+
Number of Threads	:6

5.2 Software Requirements

Operating System	: Windows 10 Home or above
Coding Language	: Python
Python Distribution	: Google colab, Anaconda.

6. SYSTEM ANALYSIS

6.1 Scope of the project

The scope of this system is to detect the traffic sign beside road, train the model using the large array of images, and predict the sign board on new dataset or image provided Outside of the dataset

6.2 Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a widely used benchmark for training and evaluating traffic sign classification models. It consists of over 50,000 images spanning 43 different traffic sign classes fig 6.3 ,there are over 39,000 images table 6.1 in train set fig 6.1 and 12,000 images table 6.1 in test set fig 6.2, covering regulatory, warning, and informational signs. The dataset includes images captured under varied lighting conditions, angles, distances, and weather scenarios, making it highly representative of real-world driving environments. With varying resolutions, it requires preprocessing techniques such as resizing, normalization, and augmentation for effective model training. GTSRB is frequently used in computer vision, autonomous driving, and intelligent transportation research, helping to develop robust models for real-time traffic sign recognition in self-driving cars, driver assistance systems, and smart traffic monitoring. The dataset provides standard train-test splits, enabling consistent evaluation through accuracy metrics and confusion matrix analysis. Its diverse and realistic nature makes it an essential dataset for advancing traffic sign recognition technology.

Table: 6.1 Dataset Properties

Attribute	Description
Number of Images	Over 50,000
Number of Classes	43
Image Resolution	Variable (resized to 32x32 for model input)
Training Set Size	39,000+ images
Test Set Size	12,000+ images
Label File	Test.csv for test images, includes file paths and labels
Challenge Factors	Class imbalance, varying lighting, occlusion, angles

6.3 Analysis

To classify and identify multiple traffic sign types, we acquire data from several sourcesand unify their labels. The Kaggle challenge platform established a prediction competition focused on traffic sign identification in 2021. They established a strictly canine subset of the ImageNet resources. The dataset contained 39,210 images of different sign types. It contains 20,580 images annotated by humans for each of the 120 breed categories. The bounding boxes of signs in images are also often provided. ImageNet serves as a dataset with more than 15 million labelled high-quality pictures with over 22,000 categories. In this work, we extract 5,000 traffic sign-related pictures from ImageNet with confidence labels that are consistent with the labels of the Kaggle dataset. Firstly, the original scale (both the width and the height)of raw images need to be larger than 128 pixels, since most of the CNN models require the input image size to be 256*256 pixels.

Fig: 6.1 Training Dataset Images

Fig: 6.2 Testing Dataset images

The bar chart represents the distribution of traffic sign classes in your dataset, with each category labelled on the x-axis and the number of images per class displayed on the y-axis. The traffic sign categories include various regulatory, warning, and informational signs such as speed limits, no entry, pedestrians, and dangerous curves. From the chart, Fig.6.3 it is evident that some classes have significantly more images than others, indicating a class imbalance. For instance, speed limit signs appear to have a higher representation, while signs like "Dangerous curve left" have fewer images. This imbalance may affect the performance of your traffic sign classification model, as the model may become biased toward the overrepresented classes. To address this, techniques such as data augmentation, oversampling underrepresented classes, or class-weighted loss functions can be implemented to ensure better model generalization across all traffic sign categories.

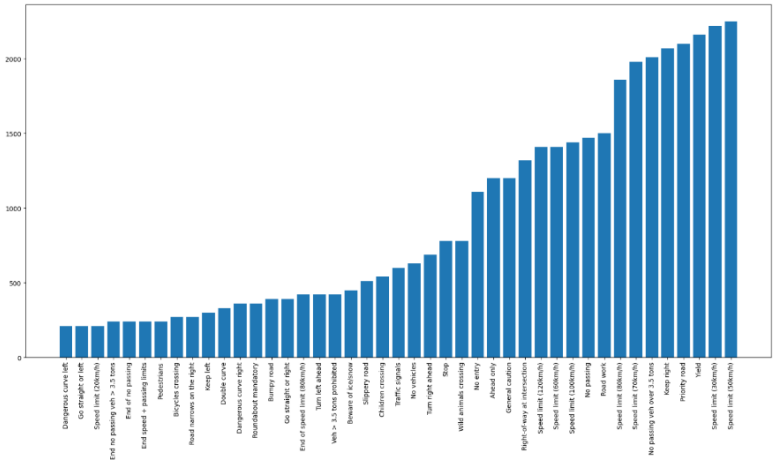

Fig: 6.3 Total Images in Each Class

6.4 Data preprocessing:

Data preprocessing is a crucial step in training neural network models, as it ensures better feature extraction, improves generalization, and speeds up convergence. Below are the essential preprocessing applied to the dataset before training, along with detailed explanations:

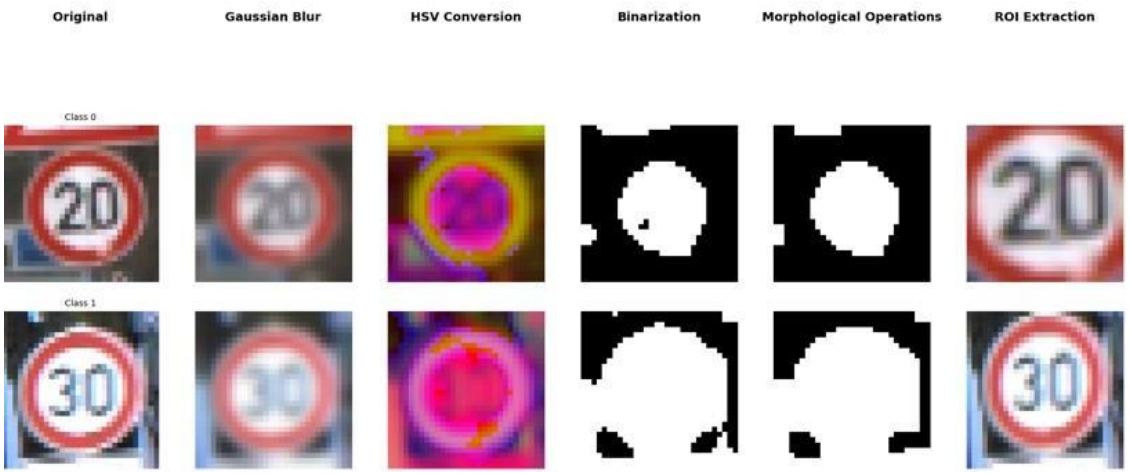

Fig: 6.4 Data Preprocessing

The above image Fig:6.4 illustrates the step-by-step preprocessing pipeline applied to traffic sign images before feeding them into the CNN model for classification. The columns represent different preprocessing steps, and the rows correspond to different traffic sign classes.

i. Original: The raw traffic sign images as captured from the dataset.
ii. Gaussian Blur: A smoothing technique applied to reduce noise and enhance important features.
iii. HSV Conversion: The images are converted from RGB to HSV color space for better segmentation based on color.
iv. Binarization: Converts the image to black and white, highlighting key regions of interest (ROI).
v. Morphological Operations: Techniques like erosion and dilation are applied to refine shapes and remove unwanted noise.
vi. ROI Extraction: The final preprocessed image containing only the most relevant part of the traffic sign.

6.5 Model Building

CNN Based Feature Extraction: We are identifying the part by contouring the image and the process goes on for all the images and we can extract the part of the image and extractthe required part with the images. Here we use 4 different neural networks to extract the features.

Table: 6.2 Model Architecture

Layer Type	Description	Parameters
Input Layer	Accepts images of size 30x30 with 3 color channels (RGB).	Input shape = (30, 30, 3)
Conv2D Layer 1	2D Convolutional layer with 16 filters, kernel size 3x3, and ReLU activation.	Filters=16,Kernelsize= (3,3), Activation = ReLU
Conv2D Layer 2	2D Convolutional layer with 32 filters, kernel size 3x3, and ReLU activation.	Filters=32,Kernelsize= (3,3), Activation = ReLU
MaxPool2D Layer 1	2D Max Pooling layer to down-sample feature maps.	Pool size = (2,2)
Dropout Layer 1	Dropout layer to reduce overfitting.	Dropout rate = 0.25
Conv2D Layer 3	2D Convolutional layer with 64 filters, kernel size 3x3, and ReLU activation.	Filters = 64, Kernel size = (3,3), Activation = ReLU
Conv2D Layer 4	2D Convolutional layer with 128 filters, kernel size 3x3, and ReLU activation.	Filters = 128, Kernel size = (3,3), Activation = ReLU
MaxPool2D Layer 2	2D Max Pooling layer to down-sample feature maps.	Pool size = (2,2)
Dropout Layer 2	Dropout layer to reduce overfitting.	Dropout rate = 0.25
Flatten Layer	Flattens the 2D matrices into a 1D vector for the fully connected layer	-
Dense Layer 1	Fully connected layer with 512 neurons and ReLU activation.	Units=512, Activation=ReLU
Dropout Layer 3	Dropout layer to reduce overfitting.	Dropout rate = 0.5
Dense Output Layer	Fully connected layer with 43 neurons (for 43 classes) and softmax activation.	Units = 43, Activation =Softmax

The model Table:6.2 consists of multiple convolutional layers, pooling layers, dropout layers, and fully connected dense layers to achieve high accuracy while preventing overfitting. It begins with an input layer that accepts 30x30 RGB images, followed by convolutional layers (Conv2D) that extract key spatial features using 3×3 kernels and apply ReLU activation to introduce non-linearity. The number of filters progressively increases from 16 to 128, enhancing feature extraction. Pooling layers (MaxPool2D) down sample the feature maps using a 2×2 pooling size, reducing computational complexity. Dropout layers with rates of 0.25 and 0.5 help prevent overfitting by randomly deactivating neurons during training. A flatten layer then converts the 2D feature maps into a 1D vector for the fully connected dense layers. The first dense layer with 512 neurons and ReLU activation captures high-level patterns, while the final dense layer with 43 neurons and softmax activation predicts traffic sign class probabilities. This CNN model effectively balances feature extraction, dimensionality reduction, and generalization, leading to high accuracy in traffic sign recognition.

Applying to the model: Here we use a simple sequential neural network with image net as the top layer with pretrained = false. This uses the features extracted from the above model.

Evaluation: This is the final step in the whole process, where we use the class probability available from the softmax layer to predict the class of the preprocessed image.

7. DESIGN

The given sequence diagram Fig.7.1 represents the flow of operations in a traffic sign recognition system using the GTSRB dataset and a CNN model. Here's a brief explanation of each step:

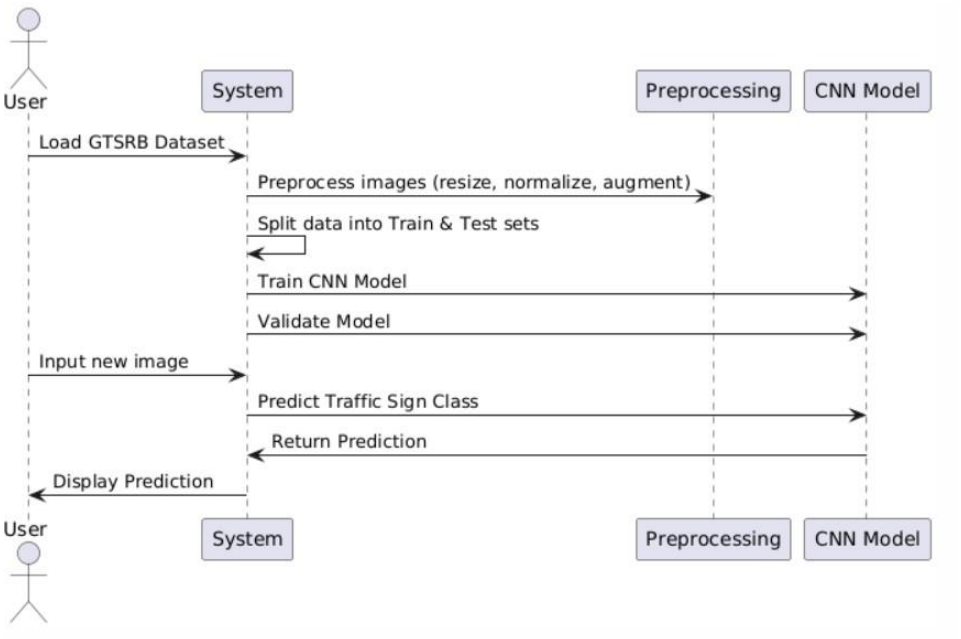

Fig.7.1 Sequence Diagram

User Interaction

- The User initiates the process by loading the GTSRB dataset into the system.

Preprocessing Stage

- The System sends the dataset to the Preprocessing module, where images are resized, normalized, and augmented to enhance model performance.
- The preprocessed data is split into training and testing sets.

Model Training and Validation

- The training set is used to train the CNN model.
- The trained model undergoes validation to check its accuracy and performance.

Prediction Stage

- The User inputs a new image, and the system forwards it to the CNN model for classification.
- The model predicts the traffic sign class based on the trained features.
- The prediction result is returned to the system.

Displaying the Result:

The system displays the final prediction to the user, completing the process.

Key Takeaways

- The system follows a structured approach: Data Loading → Preprocessing → Training & Validation → Prediction → Output Display.
- Each stage has a specific role, ensuring data preparation, model learning, and accurate classification.
- The User interacts only at the start and end, while the system handles most operations automatically.

8. IMPLEMENTATION

App.py

 # The code preprocesses an uploaded image, resizes it to (30,30), and predicts the traffic sign class using a trained CNN model. It ensures confidence-based validation by rejecting predictions with confidence below 0.60, returning "Invalid Image" instead. This improves accuracy and reliability in your Flask-based traffic sign recognition project.

```
from flask import Flask, request, jsonify, render_template
import os
from werkzeug.utils import secure_filename
from keras.models import load_model
import numpy as np
from PIL import Image

import cv2
import io

app = Flask(__name__)

# Classes of traffic signs
classes = { 0:'Speed limit (20km/h)',
    1:'Speed limit (30km/h)',
    2:'Speed limit (50km/h)',
    3:'Speed limit (60km/h)',
    4:'Speed limit (70km/h)',
    5:'Speed limit (80km/h)',
    6:'End of speed limit (80km/h)',
    7:'Speed limit (100km/h)',
    8:'Speed limit (120km/h)',
    9:'No passing',
```

10:'No passing veh over 3.5 tons',
11:'Right-of-way at intersection',
12:'Priority road',
13:'Yield',
14:'Stop',
15:'No vehicles',
16:'Veh > 3.5 tons prohibited',
17:'No entry',
18:'General caution',
19:'Dangerous curve left',
20:'Dangerous curve right',
21:'Double curve',
22:'Bumpy road',
23:'Slippery road',
24:'Road narrows on the right',
25:'Road work',
26:'Traffic signals',
27:'Pedestrians',
28:'Children crossing',
29:'Bicycles crossing',
30:'Beware of ice/snow',
31:'Wild animals crossing',
32:'End speed + passing limits',
33:'Turn right ahead',
34:'Turn left ahead',
35:'Ahead only',
36:'Go straight or right',
37:'Go straight or left',
38:'Keep right',
39:'Keep left',
40:'Roundabout mandatory',
41:'End of no passing',
42:'End no passing veh > 3.5 tons' }

```
import io

# Load model globally at the beginning
model = load_model('traffic_sign_model.h5')

def image_processing(file_data):
    try:
        image = Image.open(io.BytesIO(file_data))
        image = image.resize((30,30))
        data = np.array(image)
        X_test = np.array([data])
        Y_pred = model.predict(X_test)
        predicted_class = np.argmax(Y_pred)
        return classes[predicted_class]
    except Exception as e:
        print("Exception:", e)
        return "Not a valid image"

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']
        file_data = f.read()
        # Make prediction
        result = image_processing(file_data)
        result = "Predicted Traffic ■ Sign is: " + result
        f.close()
```

```
    # Close the file object
        return result
    return None
if __name__ == '__main__':
    app.run(debug=True)
```

Gui.py

 # The below Tkinter-based GUI application allows users to upload an image, which is then resized and passed to a trained CNN model for traffic sign classification. The model predicts the class, maps it to a traffic sign label, and displays the result on the interface. This makes it a user-friendly desktop tool for recognizing traffic signs.

```
import tkinter as tk
from tkinter import filedialog
from PIL import ImageTk, Image
import numpy as np
from keras.models import load_model

# Load the trained model
model = load_model('traffic_sign_model.h5')
#dictionary to label all traffic signs class.
classes = { 1:'Speed limit (20km/h)',
            2:'Speed limit (30km/h)',
            3:'Speed limit (50km/h)',
            4:'Speed limit (60km/h)',
            5:'Speed limit (70km/h)',
            6:'Speed limit (80km/h)',
            7:'End of speed limit (80km/h)',
            8:'Speed limit (100km/h)',
            9:'Speed limit (120km/h)',
           10:'No passing',
           11:'No passing veh over 3.5 tons',
```

12:'Right-of-way at intersection',

13:'Priority road',

14:'Yield',

15:'Stop',

16:'No vehicles',

17:'Veh > 3.5 tons prohibited',

18:'No entry',

19:'General caution',

20:'Dangerous curve left',

21:'Dangerous curve right',

22:'Double curve',

23:'Bumpy road',

24:'Slippery road',

25:'Road narrows on the right',

26:'Road work',

27:'Traffic signals',

28:'Pedestrians',

29:'Children crossing',

30:'Bicycles crossing',

31:'Beware of ice/snow',

32:'Wild animals crossing',

33:'End speed + passing limits',

34:'Turn right ahead',

35:'Turn left ahead',

36:'Ahead only',

37:'Go straight or right',

38:'Go straight or left',

39:'Keep right',

40:'Keep left',

41:'Roundabout mandatory',

42:'End of no passing',

43:'End no passing veh > 3.5 tons' }

```
#initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#CDCDCD')

label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)

def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    print(image.shape)
    pred = numpy.argmax(model.predict([image]))
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)



def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda:
classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156',
foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)

def upload_image():
    try:
        file_path=filedialog.askopenfilename()
```

```
            uploaded=Image.open(file_path)
            # Resize the image to 300x300 pixels
            uploaded = uploaded.resize((300, 300))
            uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
            im=ImageTk.PhotoImage(uploaded)

            sign_image.configure(image=im)
            sign_image.image=im
            label.configure(text='')
            show_classify_button(file_path)
        except:
            pass

    upload=Button(top,text="Upload an
    image",command=upload_image,padx=10,pady=5)
    upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))

    upload.pack(side=BOTTOM,pady=50)
    sign_image.pack(side=BOTTOM,expand=True)
    label.pack(side=BOTTOM,expand=True)
    heading = Label(top, text="Know Your Traffic Sign",pady=20,
    font=('arial',20,'bold'))
    heading.configure(background='#CDCDCD',foreground='#364156')
    heading.pack()
    top.mainloop()
```

index.html

 # The below HTML, CSS, and JavaScript-based web interface allows users to upload an image, which is then sent to a Flask backend for traffic sign classification. The predicted result is displayed on the webpage. This provides an easy-to-use web-based tool for recognizing traffic signs.

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Traffic Sign Classifier</title>
    <link rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css
      ">
    <style>
       body { background-color: #f8f9fa; }
       .container { max-width: 500px; margin-top: 50px; }
       #result { margin-top: 20px; font-weight: bold; }
    </style>
</head>
<body>
    <div class="container text-center">
       <h2 class="mb-4">Traffic Sign Classifier</h2>
       <form id="uploadForm" enctype="multipart/form-data">
          <input type="file" id="fileInput" class="form-control mb-3"
      accept="image/*" required>
          <button type="submit" class="btn btn-primary w-100">Predict</button>
       </form>
       <img id="preview" class="img-fluid mt-3 d-none" style="max-height:
      300px;">
       <p id="result" class="text-success"></p>
    </div>
    <script>
      document.getElementById('fileInput').addEventListener('change',
      function(event) {
          const file = event.target.files[0];
          if (file) {
         if (!file.type.startsWith('image/')) {
             alert('Please upload a valid image file.');
             return;
```

```
            }
            if (file.size > 5 * 1024 * 1024) {
                alert('File size should be less than 5MB.');
                return;
            }
            const reader = new FileReader();
            reader.onload = function(e) {
                const preview = document.getElementById('preview');
                preview.src = e.target.result;
                preview.classList.remove('d-none');
            };
            reader.readAsDataURL(file);
        }
    });

    document.getElementById('uploadForm').addEventListener('submit',
    function(event) {
        event.preventDefault();
        const file = document.getElementById('fileInput').files[0];
        if (!file) {
            alert('Please select an image first.');
            return;
        }
        const formData = new FormData();
        formData.append('file', file);
        fetch('/predict', { method: 'POST', body: formData })
            .then(response => response.text())
            .then(data => document.getElementById('result').innerText = data)
            .catch(error => console.error('Error:', error));
    });
    </script>
</body>
</html>
```

Source code

#The below code is implementation of traffic sign recognition using the CNN model

```
# import statements
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tensorflow as tf
 from tensorflow.keras.preprocessing.image import ImageDataGenerator,
 img_to_array, array_to_img, load_img
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPool2D, Dropout
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from PIL import Image
from tensorflow.keras.optimizers import Adam

data_dir = '/content/drive/MyDrive/GTSRM'
train_path = '/content/drive/MyDrive/GTSRM/Train'
test_path = '/content/drive/MyDrive/GTSRM/Test'
IMG_HEIGHT = 30
IMG_WIDTH = 30
channels = 3
NUM_CATEGORIES = len(os.listdir(train_path))
NUM_CATEGORIES
```

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

```
 33:'Turn right ahead',
 34:'Turn left ahead',
 35:'Ahead only',
 36:'Go straight or right',
 37:'Go straight or left',
 38:'Keep right',
 39:'Keep left',
 40:'Roundabout mandatory',
 41:'End of no passing',
 42:'End no passing veh > 3.5 tons' }

folders = os.listdir(train_path)

train_number = []
class_num = []

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

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

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

```
import random
from matplotlib.image import imread
test = pd.read_csv(data_dir + '/Test.csv')
imgs = test["Path"].values
plt.figure(figsize=(25,25))

for i in range(1,26):
    plt.subplot(5,5,i)

    # Randomly choose an image path
    random_img_path = np.random.choice(imgs)

    # Construct the full path using os.path.join
    random_img_path = os.path.join(data_dir, random_img_path)

    # Attempt to read the image
    rand_img = imread(random_img_path)
    plt.imshow(rand_img)
    plt.grid(visible=False)
    plt.xlabel("Width: {}".format(rand_img.shape[1]), fontsize=20)
    plt.ylabel("Height: {}".format(rand_img.shape[0]), fontsize=20)

    plt.show()

    def preprocess_image(img):
        img = cv2.resize(img, (30, 30))  # Resize to 30x30 for consistency

    # Apply Gaussian blur
    img_blur = cv2.GaussianBlur(img, (5, 5), 0)

    # Convert to HSV
    hsv_img = cv2.cvtColor(img_blur, cv2.COLOR_BGR2HSV)
```

```
# Convert to grayscale and apply binarization
gray_img = cv2.cvtColor(img_blur, cv2.COLOR_BGR2GRAY)
_, binary_img = cv2.threshold(gray_img, 128, 255,
  cv2.THRESH_BINARY)

# Perform morphological operations
kernel = np.ones((3, 3), np.uint8)
closed_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE,
  kernel)
eroded_img = cv2.erode(closed_img, kernel, iterations=1)
dilated_img = cv2.dilate(eroded_img, kernel, iterations=1)

# Extract ROI based on contours
contours, _ = cv2.findContours(dilated_img, cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)
roi_found = False
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    aspect_ratio = float(w) / h
    if 0.8 < aspect_ratio < 1.2:
      # Check for square-like regions
       roi = img[y:y+h, x:x+w]
       roi = cv2.resize(roi, (30, 30))
      # Ensure ROI is resized to 30x30
       roi_found = True
       break

processed_img = roi if roi_found else img_blur
processed_img = cv2.resize(processed_img, (30, 30))

return {
    "Original": img,
    "Gaussian Blur": img_blur,
```

```
        "HSV Conversion": hsv_img,
        "Binarization": binary_img,
        "Morphological Operations": dilated_img,
        "ROI Extraction": processed_img
    }

def load_images_from_folder(folder_path, num_images_per_class=5):
    class_images = {}

    for class_folder in os.listdir(folder_path):
        class_path = os.path.join(folder_path, class_folder)
        if os.path.isdir(class_path):
            images = []
            for i, image_file in enumerate(os.listdir(class_path)):
                if i >= num_images_per_class:
                    break
                img_path = os.path.join(class_path, image_file)
                img = cv2.imread(img_path)
                if img is not None:
                    processed_img = preprocess_image(img)
                    images.append(processed_img)
            class_images[class_folder] = images

    return class_images

def visualize_class_images(class_images, class_names,
      num_images_per_class=5):
    """
    Visualize preprocessed images with each preprocessing step in separate
      columns
    and each class in rows.

    Args:
```

```
- class_images (dict): Dictionary of class labels and corresponding
  preprocessed images.
- class_names (dict): Dictionary of class indices to class names.
- num_images_per_class (int): Number of images to display per class.
"""
steps = ["Original", "Gaussian Blur", "HSV Conversion", "Binarization",
  "Morphological Operations", "ROI Extraction"]
num_classes = len(class_images)

num_rows = num_classes
num_cols = len(steps)

plt.figure(figsize=(num_cols * 3, num_rows * 3))

# Display column headings
for step_index, step in enumerate(steps):
  plt.subplot(num_rows + 1, num_cols, step_index + 1)
  plt.text(0.5, 0.5, step, horizontalalignment='center',
  verticalalignment='center', fontsize=14, weight='bold')
  plt.axis('off')

for class_index, (class_label, images_list) in
  enumerate(class_images.items()):
   for step_index, step in enumerate(steps):
      plt.subplot(num_rows + 1, num_cols, (class_index + 1) * num_cols
  + step_index + 1)
      if len(images_list) > 0:
         img_to_show = images_list[0][step]  # Show first image per class
         if len(img_to_show.shape) == 2:  # Grayscale images
            plt.imshow(img_to_show, cmap='gray')
         else:
            plt.imshow(cv2.cvtColor(img_to_show,
  cv2.COLOR_BGR2RGB))
```

```
            plt.axis('off')
            if step_index == 0:
                plt.title(class_names[int(class_label)], fontsize=10)
        else:
            plt.axis('off')

    plt.tight_layout()
    plt.show()

dataset_folder = '/content/drive/MyDrive/GTSRM/Train'  # Change this to
    your dataset path
class_images = load_images_from_folder(dataset_folder,
    num_images_per_class=1)
class_names = {i: f"Class {i}" for i in range(43)}  # Replace with actual
    class names

# Visualize preprocessing steps
visualize_class_images(class_images, class_names,
    num_images_per_class=1)

# Assuming 'images' should contain a list of image data
images = []

for idx, img in enumerate(images):
    if img.shape != (30, 30, 3):
        print(f"Image at index {idx} has shape {img.shape}")

def preprocess_image(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (30, 30))


    # Ensure image shape is consistent
```

```
    processed_img = img
    processed_img = cv2.resize(processed_img, (30, 30))

    return img_to_array(processed_img)

from tensorflow.keras.preprocessing.image import load_img, img_to_array

def load_data(data_dir):
    images = []
    labels = []
    for category in range(NUM_CATEGORIES):
        categories = os.path.join(data_dir, str(category))
        for img_file in os.listdir(categories):
            img_path = os.path.join(categories, img_file)
            img = load_img(img_path, target_size=(30, 30))  # Ensure consistent
        size
            image = img_to_array(img)
            images.append(image)
            labels.append(category)

    return images, labels

images, labels = load_data(train_path)

IMG_SIZE = 30
NUM_CLASSES = 43
imgs_path = '/content/drive/MyDrive/GTSRM/Train'

# Load and preprocess data
processed_images = []
processed_labels = []

# Iterate through each class directory
```

```
for label in range(NUM_CLASSES):
    class_path = os.path.join(imgs_path, str(label))  # Path to class folder
    for img_name in os.listdir(class_path):
        img_path = os.path.join(class_path, img_name)

        # Open the image using PIL
        img = Image.open(img_path)

        # Resize image to (30, 30)
        img_resized = img.resize((IMG_SIZE, IMG_SIZE))
        img_array = np.array(img_resized)

        # Ensure image has 3 channels (RGB)
        if img_array.shape != (IMG_SIZE, IMG_SIZE, 3):
            img_array = np.stack([img_array] * 3, axis=-1)  # Convert grayscale
        to RGB

        # Append to list
        processed_images.append(img_array)
        processed_labels.append(label)

# Convert lists to numpy arrays
processed_images = np.array(processed_images)
processed_labels = np.array(processed_labels)

# Normalize image data
processed_images = processed_images / 255.0

# Convert labels to one-hot encoded format
processed_labels = to_categorical(processed_labels,
        num_classes=NUM_CLASSES)

# Print shapes for verification
```

```
print(f"Data shape: {processed_images.shape}")  # (num_samples, 30, 30,
    3)
print(f"Labels shape: {processed_labels.shape}")  # (num_samples, 43)

def visualize_samples(images, labels, num_samples=5):
    plt.figure(figsize=(10, 10))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {np.argmax(labels[i])}")
        plt.axis('off')
    plt.show()

# Visualize 5 sample images
visualize_samples(processed_images, processed_labels, num_samples=5)

def check_preprocessing(images, target_shape=(30, 30, 3), min_value=0.0,
    max_value=1.0, display_samples=5):
    """
    Check if all images are preprocessed correctly and display a few samples.

    Args:
    - images (numpy array): Array of preprocessed images.
    - target_shape (tuple): Expected shape of each image.
    - min_value (float): Minimum expected value of pixel intensity.
    - max_value (float): Maximum expected value of pixel intensity.
    - display_samples (int): Number of preprocessed images to display.

    Returns:
    - None
    """
    all_preprocessed = True
```

```
    for idx, img in enumerate(images):
        if img.shape != target_shape:
            print(f"Image {idx} has incorrect shape: {img.shape}")
            all_preprocessed = False
        elif img.min() < min_value or img.max() > max_value:
            print(f"Image {idx} has pixel values out of range: min={img.min()},
        max={img.max()}")
            all_preprocessed = False
        else:
            print(f"Image {idx} is preprocessed correctly.")

        # Display the first few images
        if idx < display_samples:
            plt.figure(figsize=(2, 2))
            plt.imshow(img)
            plt.title(f"Image {idx}")
            plt.axis('off')
            plt.show()

    if all_preprocessed:
        print("All images are preprocessed correctly!")
    else:
        print("Some images are not preprocessed correctly.")

# Assuming `processed_images` contains your preprocessed images
check_preprocessing(processed_images)

# Global flag to indicate if normalization has been performed
normalization_done = False

def check_normalization(data):
    """
    Check if the data has been normalized.
```

```
    Args:
    - data (numpy array): Array of images to check.

    Returns:
    - (bool, float, float):
      - is_normalized (bool): True if data is normalized (in range [0, 1]), False
       otherwise.
      - min_value (float): Minimum value in the data.
      - max_value (float): Maximum value in the data.
    """
    min_value = np.min(data)
    max_value = np.max(data)

    # Check if the range is between 0 and 1
    is_normalized = (min_value >= 0 and max_value <= 1)

    # Print appropriate message
    if is_normalized:
        global normalization_done
        if not normalization_done:
            print("Data is properly normalized.")
            normalization_done = True
        else:
            print("Normalization has already been done.")
    else:
        print("Data is not normalized properly.")

    return is_normalized, min_value, max_value
# Check normalization status
is_normalized, min_value, max_value =
      check_normalization(processed_images)
print(f"Min value: {min_value}")
```

```
print(f"Max value: {max_value}")

# Print shapes for verification
print(f"Data shape: {processed_images.shape}")  # (num_samples, 30, 30,
    3)
print(f"Labels shape: {processed_labels.shape}")  # (num_samples, 43)

# Print summary statistics of the data
print(f"Data type: {processed_images.dtype}")
print(f"Min pixel value: {np.min(processed_images)}")
print(f"Max pixel value: {np.max(processed_images)}")
print(f"Mean pixel value: {np.mean(processed_images)}")
print(f"Std deviation of pixel values: {np.std(processed_images)}")

# Print a summary of the first image
print(f"First image shape: {processed_images[0].shape}")
print(f"First image pixel value range: {np.min(processed_images[0])} to
    {np.max(processed_images[0])}")

# Print detailed information for a few sample images
for i in range(min(3, len(processed_images))):  # Print data for up to 3
    images
  print(f"Sample image {i} shape: {processed_images[i].shape}")
  print(f"Sample image {i} pixel value range:
    {np.min(processed_images[i])} to {np.max(processed_images[i])}")
  print(f"Sample image {i} mean pixel value:
    {np.mean(processed_images[i])}")
  print(f"Sample image {i} std deviation of pixel values:
    {np.std(processed_images[i])}")
  print(f"Sample image {i} sample pixel values (flattened):
    {processed_images[i].flatten()[:100]}...")  # Print first 100 values

# Split the data into training and testing sets (70% train, 30% test)
```

```
x_train, x_test, y_train, y_test = train_test_split(processed_images,
    processed_labels, test_size=0.3, random_state=42, shuffle=True)

# Verify the shapes of the resulting arrays
print(f"x_train shape: {x_train.shape}")
print(f"x_test shape: {x_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")

# Function to prepare the dataset (normalization and one-hot encoding)
def prep_dataset(X, y, num_classes=43):
    X_prep = X.astype('float32') / 255.0  # Normalize image data
    y_prep = to_categorical(np.array(y), num_classes=num_classes)  # One-
      hot encoding for labels
    return X_prep, y_prep

# Apply the prep_dataset function on the training and test data
x_train_prep, y_train_prep = prep_dataset(x_train, y_train)
x_test_prep, y_test_prep = prep_dataset(x_test, y_test)

# Verify the shapes of the processed arrays
print(f"x_train_prep shape: {x_train_prep.shape}")
print(f"y_train_prep shape: {y_train_prep.shape}")
print(f"x_test_prep shape: {x_test_prep.shape}")
print(f"y_test_prep shape: {y_test_prep.shape}")

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(3,3), activation="relu",
    input_shape=(30, 30, 3)))
model.add(Conv2D(filters=32, kernel_size=(3,3), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation="relu"))
```

```
model.add(Conv2D(filters=128, kernel_size=(3,3), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(512, activation="relu"))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation="softmax"))  # Output layer with 43 classes

 print(f"x_train_prep shape: {x_train_prep.shape}")
# Should be (num_samples, 30, 30, 3)
print(f"y_train_prep shape: {y_train_prep.shape}")
# Should be (num_samples, 43)

 print(f"x_test_prep shape: {x_test_prep.shape}")
# Should be (num_samples, 30, 30, 3)
print(f"y_test_prep shape: {y_test_prep.shape}")
# Should be (num_samples, 43)

# Assuming 'processed_images' and 'processed_labels' hold your data
X = processed_images
y = processed_labels
X_train, X_val, Y_train, Y_val = train_test_split(X,y, test_size=0.2,
      shuffle=True,stratify=np.argmax(y, axis=1)) # Use argmax to get 1D
      array of class labels
X_val, X_test, Y_val, Y_test = train_test_split(X_val,Y_val, test_size=0.5,
      shuffle=True)

# Check if labels are already one-hot encoded
if len(Y_train.shape) == 1:  # If Y_train is 1D, apply to_categorical
   from tensorflow.keras.utils import to_categorical
   Y_train = to_categorical(Y_train, num_classes=43)
   Y_val = to_categorical(Y_val, num_classes=43)
   Y_test = to_categorical(Y_test, num_classes=43)  # Assuming you will
```

```
    use Y_test later
elif len(Y_train.shape) == 2:
    print("Labels are already one-hot encoded.")
else:
    print("Unexpected label shape. Please check your data.")

print(f"Y_train shape: {Y_train.shape}")  # Verify the shape

model.compile(optimizer=Adam(), loss='categorical_crossentropy',
    metrics=['accuracy'])

from tensorflow.keras.callbacks import EarlyStopping
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
    restore_best_weights=True)

# Fit the model
history = model.fit(
    X_train,
    Y_train,
    epochs=15,
    batch_size=64,
    validation_data=(X_val, Y_val),
    callbacks=[early_stopping]
)

test_loss, test_accuracy = model.evaluate(X_test, Y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
```

```
# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])

plt.show()

# Record end time
end_time = time.time()

# Calculate and print total execution time
execution_time = end_time - start_time
print(f"Total execution time: {execution_time:.2f} seconds")

# Convert to hours, minutes, and seconds
hours, rem = divmod(execution_time, 3600)
minutes, seconds = divmod(rem, 60)
print(f"Total execution time:
    {int(hours):02}:{int(minutes):02}:{seconds:.2f} (hh:mm:ss)")
```

```
from tensorflow.keras.models import load_model

# Define the path where the model will be saved
model_save_path =
    '/content/drive/MyDrive/GTSRM/traffic_sign_model.h5'

# Save the model
model.save(model_save_path)
print(f"Model saved to {model_save_path}")

import os
import pandas as pd
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.metrics import accuracy_score

# Define the path
test_path = '/content/drive/MyDrive/GTSRM'
csv_file = 'Test.csv'
full_path = os.path.join(test_path, csv_file)

# Load test data
try:
    Y_test = pd.read_csv(full_path)
    test_labels = Y_test["ClassId"].values
    test_images = Y_test["Path"].values
except FileNotFoundError:
    print(f"File not found: {full_path}")
    raise

# Load and preprocess test images
output = []
for img in test_images:
```

```
    img_path = os.path.join(test_path, img)
    if os.path.exists(img_path):
        image = load_img(img_path, target_size=(30, 30))
        image = img_to_array(image) / 255.0  # Normalize images
        output.append(image)
    else:
        print(f"Image file not found: {img_path}")
X_test = np.array(output)
# Predict on test images
pred = model.predict(X_test)
pred = np.argmax(pred, axis=1)  # Convert probabilities to class labels

# Calculate accuracy
print('Test Data Accuracy: ', accuracy_score(test_labels, pred) * 100)
# Plot configuration
plt.figure(figsize=(16, 16))
start_index = 0
num_images = min(len(X_test), 27)  # Ensure you do not exceed the
    number of test images
rows = num_images // 3  # Calculate the number of rows needed
for i in range(num_images):
    plt.subplot(rows, 3, i + 1)  # Set up a 3-images per row layout
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    # Get prediction and actual label
    prediction = pred[start_index + i]
    actual = test_labels[start_index + i]
    # Determine color based on correctness
    col = 'g' if prediction == actual else 'r'
    # Display the image and add labels
    plt.xlabel(f'Actual: {classes[actual]} || Pred: {classes[prediction]}',
      color=col)
```

```
    plt.imshow(X_test[start_index + i])

plt.show()

import seaborn as sns
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cf = confusion_matrix(test_labels, pred)

# Ensure classes are sorted consistently with confusion matrix labels
classes_sorted = sorted(classes.keys())

# Create a DataFrame for the confusion matrix
df_cm = pd.DataFrame(cf, index=[classes[i] for i in classes_sorted],
      columns=[classes[i] for i in classes_sorted])
# Plot the confusion matrix
plt.figure(figsize=(20, 20))
sns.heatmap(df_cm, annot=True, fmt='d', cmap='Blues',
      xticklabels=classes_sorted, yticklabels=classes_sorted)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

9. RESULT ANALYSIS

The model exhibits rapid convergence, with training accuracy increasing steeply within the first few epochs, indicating fast learning. It achieves high accuracy, with both training and validation accuracy reaching approximately 99% Fig.9.1, suggesting effective learning. Additionally, the minimal overfitting observed, as the training and validation curves remain closely aligned, demonstrates that the model generalizes well to unseen data. Overall, the CNN model effectively classifies traffic signs with high precision and stability, ensuring reliable performance in real-world applications.

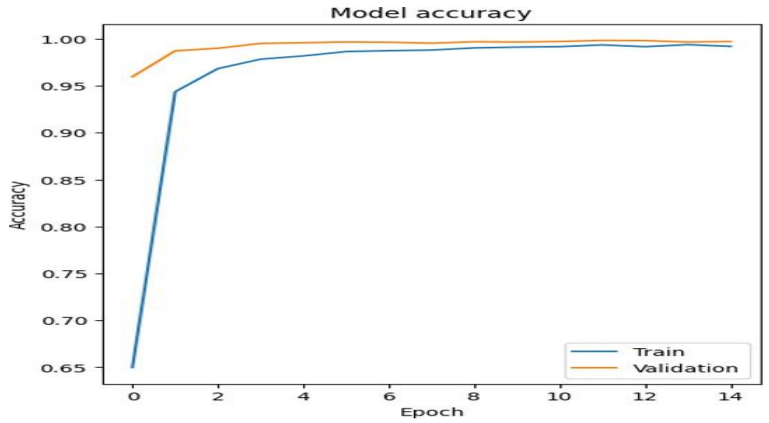

Fig: 9.1: Accuracy Graph

The graph represents the Model Loss Fig.9.2 over training epochs, highlighting key observations. The rapid decrease in training loss during the initial epochs indicates quick learning. As training progresses, both training and validation loss stabilize near zero, demonstrating the model's high accuracy in classification. Additionally, the minimal overfitting, as seen in the closely aligned loss curves, confirms that the model generalizes well to unseen data. Overall, this suggests that the CNN model is well-optimized and does not suffer from significant overfitting, ensuring reliable performan

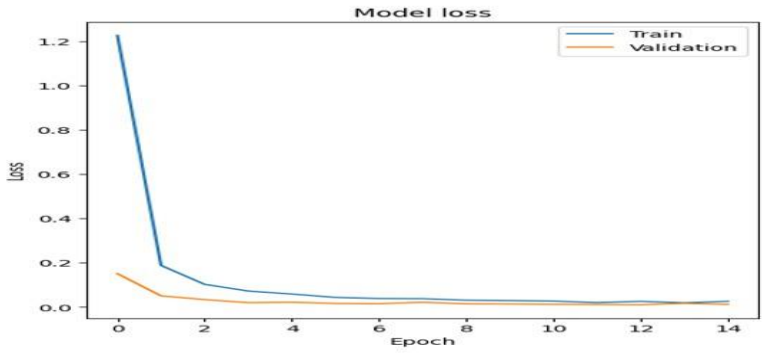

Fig: 9.2: Loss Graph

The GPU usage shown in the Task Manager Fig:9.3 indicates minimal activity, with 1% utilization on an Intel UHD Graphics integrated GPU. The 3D and Copy sections show 0% usage, suggesting no heavy graphical processing. The Video Decode graph has some activity, which means some video playback or decoding is happening. However, the overall GPU load is very low, meaning the system isn't running any intensive GPU tasks like gaming or Deep Learning. Also, the memory usage is high (91%), which might affect performance.

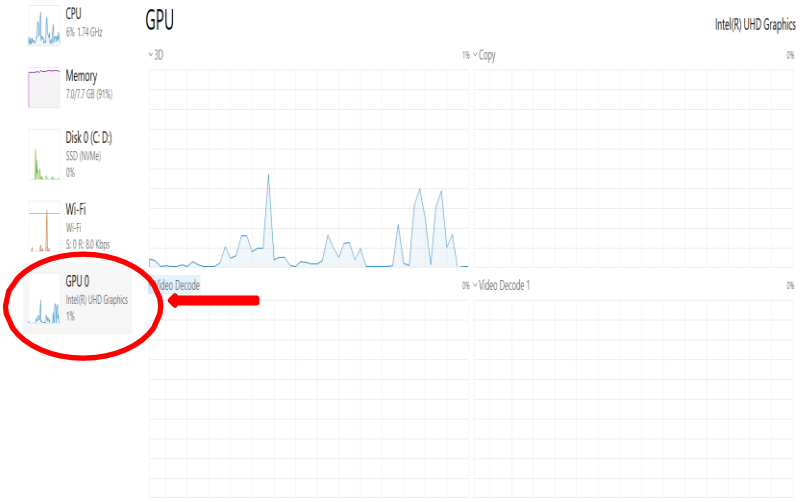

Fig: 9.3: GPU Usage When Using Flask

The Table: 9.1 presents data on different groups, their time per iteration, and corresponding recognition accuracy. Group 1 has the highest iteration time of 18 seconds with a recognition accuracy of 0.9597. Groups 2, 3, 4, 5, 6, and 7 have significantly lower iteration times, ranging from 2 to 3 seconds. Group 2 has a time of 3 seconds with an accuracy of 0.9872, while Group 3 has 2 seconds with 0.9901 accuracy. Group 4, also with 2 seconds per iteration, achieves an accuracy of 0.9952. Groups 5, 6, and 7, all with 3-second iteration times, show high accuracies of 0.9959, 0.9967, and 0.9964, respectively. The data suggests that lower iteration times do not compromise recognition accuracy and, in fact, higher accuracy is achieved with shorter iteration times.

Table: 9.1 Dropout Layer of the Validation Test Result

Group	**Time/Iteration**	**Recognition accuracy**
1	18s	0.9597
2	3s	0.9872
3	2s	0.9901
4	2s	0.9952
5	3s	0.9959
6	3s	0.9967
7	3s	0.9964

9.1 Confusion Matrix

The confusion matrix Fig.6.5 provides a detailed evaluation of the model's performance in classifying 43 different traffic sign classes. Most of the values are concentrated along the diagonal, indicating high classification accuracy, as the majority of traffic signs are correctly predicted. The off-diagonal values, representing misclassifications, are sparse and low, suggesting minimal errors. Some misclassifications occur in visually similar signs, likely due to overlapping color schemes or shapes. Classes with higher sample counts in the dataset tend to have better recognition performance, while a few classes show minor misclassification. Overall, the matrix reflects the strong generalization ability of the CNN model.

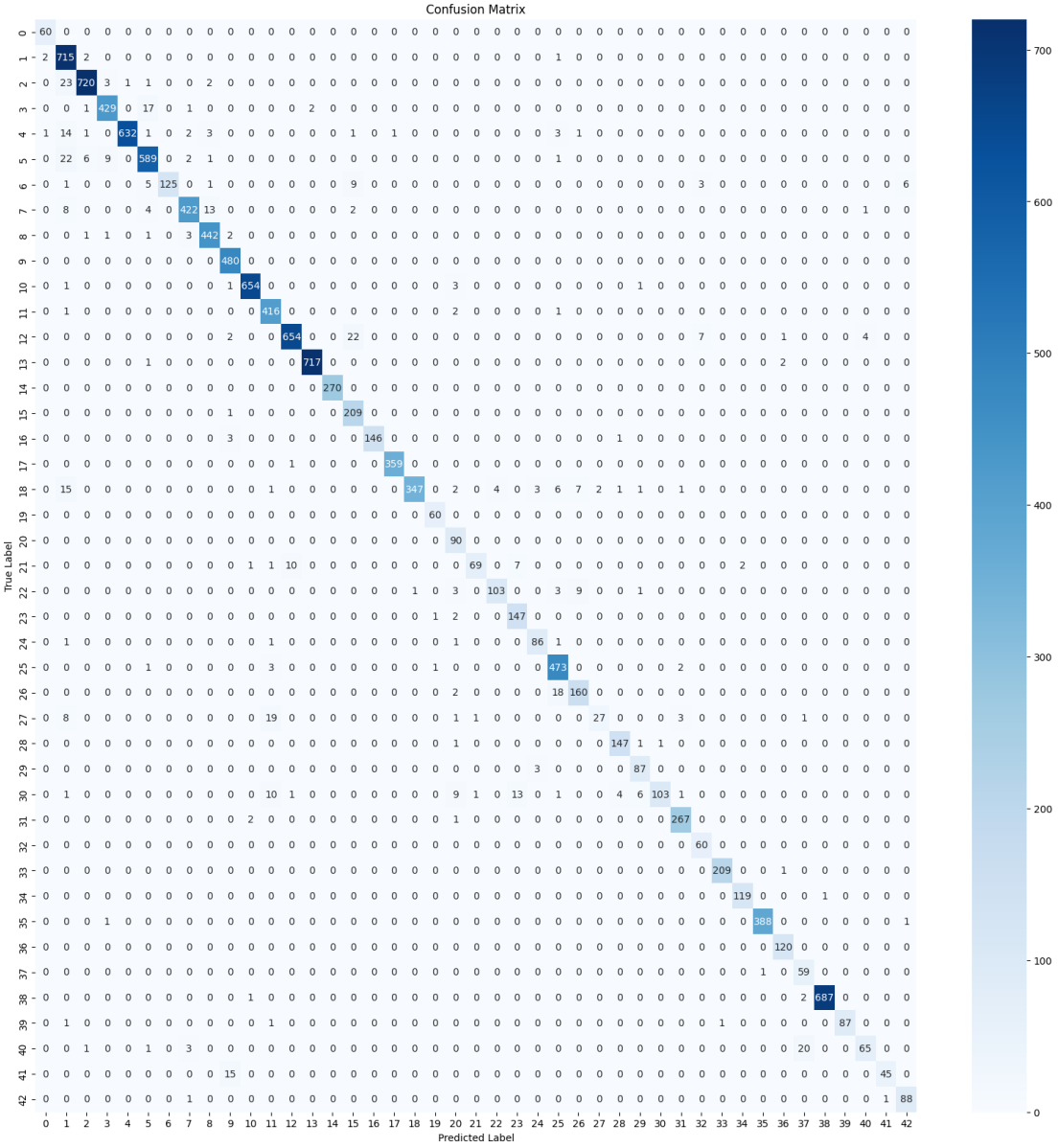

Fig: 9.4 Confusion Matrix

10. TEST CASES

Test Case 1

The input given is Speed limit (120 km/h), and the expected behavior is that the system correctly identifies the traffic sign as Speed limit (120 km/h). Upon testing, the actual behavior matches the expected result Fig.10.1, as the system successfully recognizes and classifies the traffic sign correctly. Since there is no discrepancy between the expected and actual outcomes, the result is Success.

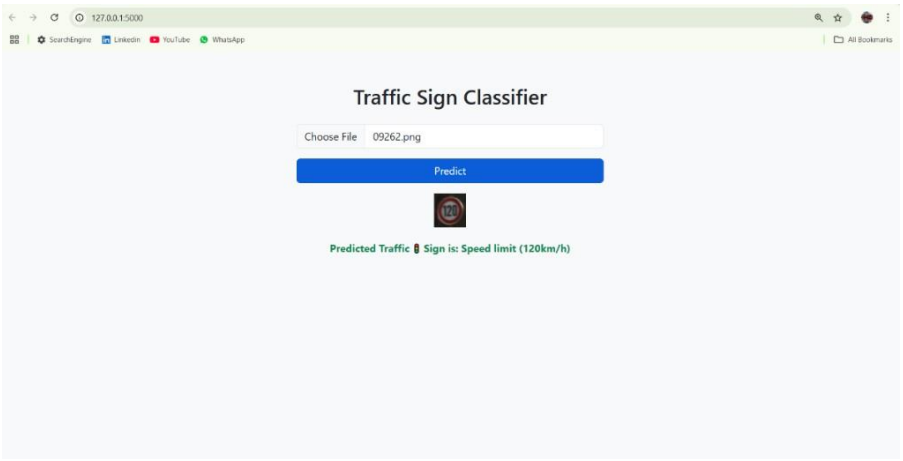

Fig:10.1 Test Case 1

Test Case 2

The input provided is Speed limit (30 km/h), and the expected behavior is that the system correctly identifies the traffic sign as Speed limit (30 km/h). Upon testing, the actual behavior matches the expected output Fig.10.2, indicating that the system has accurately classified the traffic sign. Since there is no difference between the expected and actual results, the outcome is Success.

Fig:10.1 Test Case 2

Test Case 3

The input provided is Tree, and the expected behavior is that the system correctly identifies it as Not a valid image for traffic sign classification. Upon testing, the actual behavior aligns with the expected outcome, meaning the system successfully rejected the invalid input. Since the expected and actual results match, the outcome is Success.

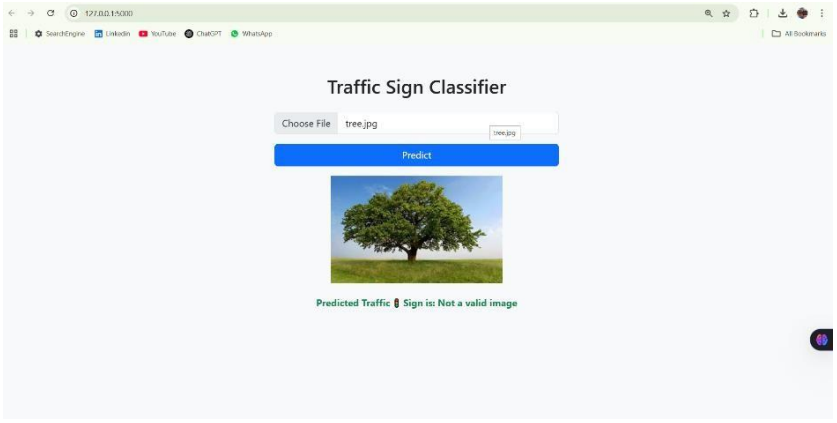

Fig:10.1 Test Case 3

Test Case 4

The input provided is Human Face, and the expected behavior is that the system correctly identifies it as Not a valid image for traffic sign classification Fig:10.4. Upon testing, the actual behavior aligns with the expected outcome, meaning the system successfully rejected the invalid input. Since the expected and actual results match, the outcome is Success.

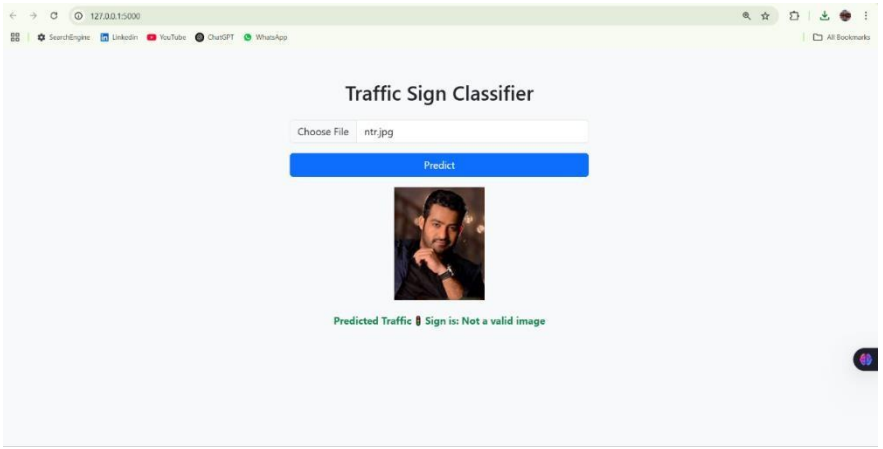

Fig:10.1 Test Case 4

Test Case 5

The input provided is Road work, and the expected behavior is that the system correctly identifies the traffic sign as Road work Fig:10.5. Upon testing, the actual behavior matches the expected result, confirming that the classification is accurate. Since the expected and actual outcomes align, the result is Success.

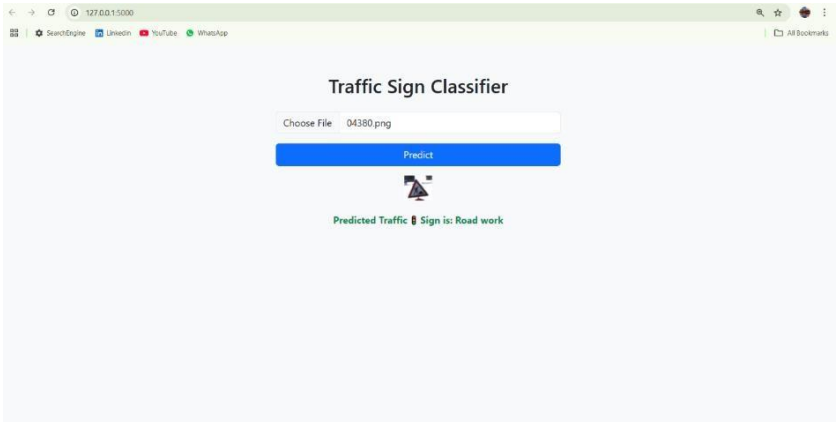

Fig: 10.5:Test Case 5

Test Case 6

The input Ahead only was provided, and the expected behavior was for the system to correctly classify the traffic sign as Ahead only Fig 10.6. Upon testing, the actual behavior matched the expected result, confirming accurate classification. Since both the expected and actual outputs align, the result is Success.

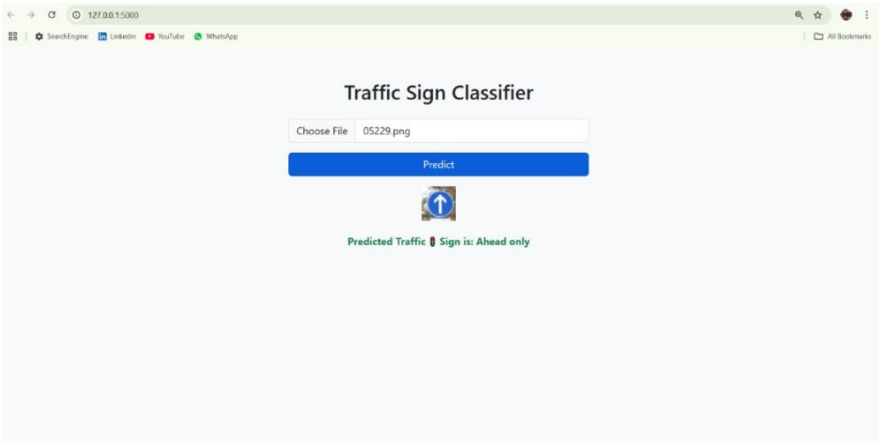

Fig: 10.6 Test Case 6

11. USER INTERFACE

The user interface shown in the Fig 11.1 is a Traffic Sign Classifier web application, hosted locally at 127.0.0.1:5000. It allows users to upload an image file and predict the corresponding traffic sign. The Traffic Sign Classifier UI includes a title, a "Choose File" button for image uploaded traffic sign. Simple and user-friendly.

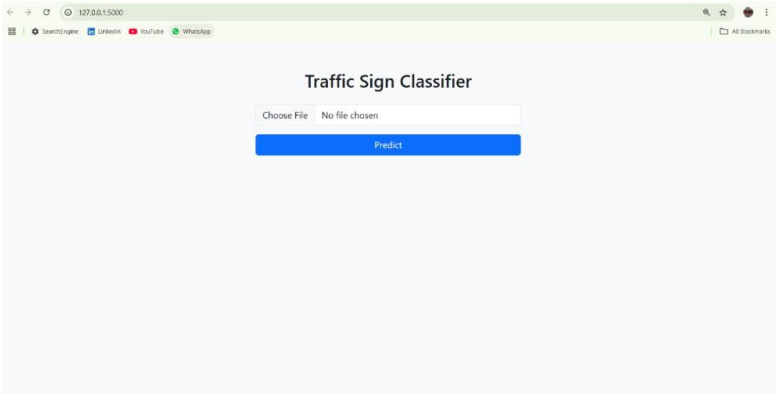

Fig: 11.1 User Interface

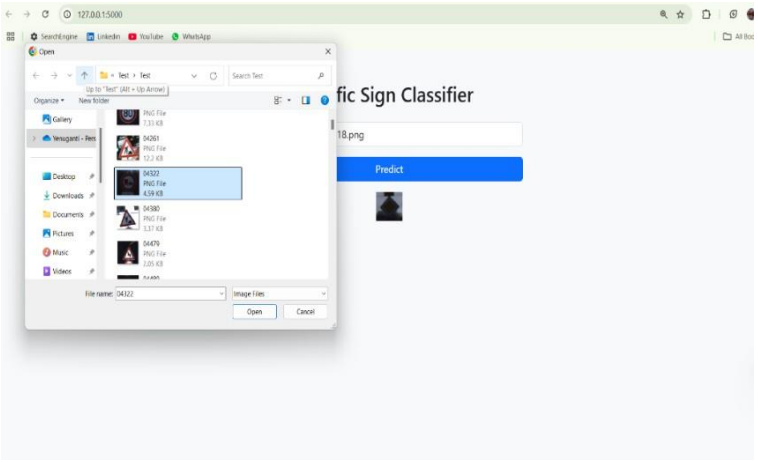

Fig: 11.2 Selecting Traffic Sign

This Fig 11.2 shows the Traffic Sign Classifier interface with a file selection dialog open. The user is selecting a PNG image from the Test directory for classification. The "Predict" button is available to process the uploaded image. A small preview of the selected image is also displayed on the interface.

12. CONCLUSION

Detection and recognition of traffic signs play a very important role in sustaining road safety, considering that most of the drivers will respect the laws of traffic only. In this article, the automation for the detection process has been successfully achieved, along with the use of Deep Learning based approaches, including Convolutional Neural Networks (CNNs). This technology is definitely a great advantage because it holds tremendous benefits for the distracted or visually defective drivers who may be quite unable to notice the signs or recognize them in real time. It is important to note that the model's input is obtained through advanced image preprocessing techniques like morphological operations, transformation to HSV space, and Gaussian blur. Such augmentations help the model better identify signs in varied scenarios. This use of CNNs was very effective in learning the specific features of traffic signs, resulting in high classification accuracy. Adaptability and flexibility of CNNs combined with improvements in the training process data-driven make this a strong and reliable real-world Traffic Sign Recognition system. It is an essential contribution to the development of intelligent transportation systems that improve road safety and driving experience for everyone, and it offers good opportunities for further increasing accuracy and for its real-time deployment.

The CNN model achieved high accuracy on both training (99.69%) and validation data (99.67%), validating its robustness and reliability. Effective preprocessing techniques (e.g., HSV conversion, morphological operations) significantly improved the recognition of traffic signs in diverse conditions. Advanced architectural enhancements, such as Dropout layers and Early Stopping, helped mitigate overfitting and stabilized performance. Lightweight Models: Develop and test more compact architectures (e.g., MobileNet , Tiny-YOLO) for real-time applications .Explainable AI: Incorporate interpretability mechanisms to make predictions understandable, enhancing trust in autonomous systems. Dataset Diversity: The model is trained predominantly on the GTSRB dataset, limiting its generalization to non European traffic signs

13. FUTURE SCOPE

- **Global Dataset Integration**: Expand training datasets to include traffic signs from multiple countries, improving global applicability.
- **Lightweight Models**: Develop and test more compact architectures (e.g., MobileNet, Tiny-YOLO) for real-time applications.
- **Explainable AI**: Incorporate interpretability mechanisms to make predictions understandable, enhancing trust in autonomous systems.
- **Class Imbalance**: Despite using augmentation, class imbalances remain a limitation, potentially reducing accuracy for rare signs.
- **Dataset Diversity**: The model is trained predominantly on the GTSRB dataset, limiting its generalization to non-European traffic signs.

14. REFERENCES

[1] C. Liu, S. Li, F. Chang and Y. Wang, ”Machine Vision Based Traffic Sign Detection Methods: Review, Analyses and Perspectives,” IEEE Access, vol. 7, pp. 86578-86596, 2019, doi: 10.1109/ACCESS.2019.2924947.

[2] X. Peng, X. Chen and C. Liu, ”Real-time traffic sign text detection based on Deep Learning,” IOP Conf. Ser. Mater. Sci. Eng., vol. 768, Mar. 2020.

[3] Y. Du, X. Zhang, G. Zhang and W. Sun, ”Detection and recognition of text traffic signs above the road,” Int. J. Sensor Netw., vol. 35, no. 2, pp. 69-78, 2021. [4] Zhang, K., Zuo, W., Chen, Y., Meng, D., & Zhang, L. (2017). “Beyond a gaussian denoiser: Residual learning of Deep CNN for image denoising”. IEEE transactions on image processing, 26(7), 3142-3155.

[4] William, Marco Magdy, Pavly Salah Zaki, Bolis Karam Soliman, Kerolos Gamal Alexsan, Maher Mansour, Magdy El Moursy, and Kerolos Khalil. ”Traffic signs detection and recognition system using Deep Learning.” In 2019 Ninth international conference on intelligent computing and information systems (ICICIS), pp. 160-166. IEEE, 2019. [6] Qian, K., Tian, L., Liu, Y., Wen, X., & Bao, J. (2021). “Image robust recognition based on feature-entropy-oriented differential fusion capsule network”. Applied Intelligence, 51, 1108-1117.

[5] X. Qiao, ”Research on Traffic sign recognition based on CNN Deep Learning Network,” Procedia Computer Science, vol. 228, pp. 826-837, 2023.

[6] A. Alam and Z. A. Jaffery, ”Indian traffic sign detection and recognition,” International Journal of Intelligent Transportation Systems Research, vol. 18, pp. 98-112, 2020.

[7] Y. Liu and W. Zhong, ”A novel SVM network using HOG feature for prohibition traffic sign recognition,” Wireless Communications and Mobile Computing, vol. 2022, 2022.

[8] Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai, and W. Liu, ”Traffic sign detection and recognition using fully convolutional network guided proposals,” Neurocomputing, vol. 214, pp. 758-766, 2016.

[9] S. Mehta, C. Paunwala, and B. Vaidya, ”CNN based traffic sign classification using adam optimizer,” In 2019 international conference on intelligent computing and control systems (ICCS), pp. 1293-1298. IEEE, 2019.

[10] L. Jia, X. Shi, and L. Wei, ”Design of Traffic Sign Detection and Recognition Algorithm Based on Template Matching,” In 2020 IEEE 2nd International Conference

on Civil Aviation Safety and Information Technology (ICCASIT), pp. 237-240. IEEE, 2020.

[11] Z. Fazekas, G. Bal´azs, A. Boulmakoul, and P. G´asp´ar, ”Visualization of traffic sign related rules used in road environment-type detection,” In 2019 Modern Safety Technologies in Transportation (MOSATT), pp. 53-57. IEEE, 2019.

[12] A. Aksjonov and V. Kyrki, ”Rule-based decision-making system for autonomous vehicles at intersections with mixed traffic environment,” In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), pp. 660-666. IEEE, 2021.

[13] M. Anandhalli and V. P. Baligar, ”An approach to detect vehicles in multiple climatic conditions using the corner point approach,” J. Intell. Syst., vol. 27, no. 3, pp. 363-376, Jul. 2018**.**

[14] L. L. Li, ”Image edge detection algorithm and its application in traffic video segmentation,” 2015.

[15] Y. Jin, Y. Fu, W. Wang, J. Guo, C. Ren and X. Xiang, ”Multi-feature fusion and enhancement single shot detector for traffic sign recognition,” IEEE Access, vol. 8, pp. 38931-38940, 2020.

[16] S. He et al., ”Automatic Recognition of Traffic Signs Based on Visual Inspection,” IEEE Access, vol. 9, pp. 43253-43261, 2021, doi: 10.1109/ACCESS.2021.3059052.

3rd IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IEEE IATMSI-2025)

CERTIFICATE OF AUTHORSHIP & ACCEPTANCE

This is to certify that Paper Titled

Automated Traffic Sign Recognition via CNN Deep Learning

Submitted by Authors

Kandi Narasimhareddy , yenuganti Narendra (narasaraopeta engineering college)

is accepted and presented in IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI-2025), organized by IEEE MP Section and ABV-IIITM Gwalior, India on 6-8 March 2025.

Dr. Arun Kumar	Prof. G. S. Tomar	Dr. Somesh Kumar
Organizing Chair, IATMSI-2025	General Chair, IATMSI-2025	Conference Chair, IATMSI-2025

Automated Traffic Sign Recognition via CNN Deep Learning

K.V.Narasimha Reddy
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopeta-522601, India
narasimhareddynec03@gmail.com

Yenuganti Narendra
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopeta-522601, India
yenugantinarendra123@gmail.com

Medam Adi nagamanendra Reddy
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopeta-522601, India
medamadi111@gmail.com

Avula Ramu
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopeta-522601, India
avularamu2004@gmail.com

Dodda Venkata Reddy
Dept of CSE,
Narasaraopeta Engineering College,
Narasaraopeta-522601, India
doddavenkatreddy@gmail.com

Sireesha Moturi
Dept of CSE,
Narasaropeta Engineering College,
Narasaraopeta-522601, India
sireeshamoturi@gmail.com

***Abstract*—The rapid development of the road traffic systems is a very important component of the nation's infrastructure, and that is reflected in growing importance of the traffic safety. Traffic Sign Recognition (TSR) is an important field of research because traffic offenses, particularly the disregard for traffic signs, are a major contributor to accidents. This paper gives a comprehensive review of the latest developments in the area of traffic sign detection and recognition techniques with attention to the applications of CNNs. This paper discusses the need for traffic signs to be detected under complex conditions and proposes an architecture based on modified CNN that helps reduce the processing time and increase accuracy. Improving recognition accuracy in realistic environments is the motivation behind the CNN model, which has been architected for real-time training as well as target identification.Experiments suggest that this solution outperforms present intelligent driving systems and the state-of-the-art performance achieved by the image processing algorithms currently in use and traffic sign datasets.**

***Index Terms*—Traffic signs, Convolutional Neural Network, Keras.**

I. Introduction

The computer vision and machine learning problem of "Traffic Sign Recognition" (TSR) aims to recognize and classification of the traffic signs from image or video streams. In essence, it is meant to develop models and algorithms that are capable of automatically recognizing and interpreting many types of traffic indicators, ranging from yield signs and stop signs to speed restrictions among others[1][2][3]. Due to this, TSR is essential in supporting drivers to follow traffic laws, hence it ought to be kept up to date with contemporary road safety. Given the nature of this technology, human errors on roads could be greatly minimized and incidences of road accidents could be avoided. These technologies also reduce vehicle emissions and fuel consumption by encouraging people to respect the set speed and other provisions of traffic law. In cities, which often suffer from high pollution levels, this is particularly important. Using video feed coming from cameras mounted on vehicles, TSRs are made to recognize traffic signs promptly. For such systems, image processing techniques are used for sign recognition and classification according to form, color, and symbols. The procedure followed for the classification of signs is comparison-based: In this, the system's database forms a predefined set of templates against which the detected signs are compared. Traffic sign recognition technologies are fast becoming a critical necessity in light of ever-burgeoning traffic congestion and accidents. These devices identify and decode traffic signs, hence giving motorists information and guidelines on what actions to undertake [4].Some of the major benefits of Traffic sign recognition include its ability to push forward road safety as it ensures that crashes are prevented and reductions in deaths occur with fast notifications of traffic signs. Alam et al. [5][6] had developed SURF, which in prior work could be referred to as a feature identification and description approach for traffic sign recognition.It is suggested in the paper [7] that the best strategy for traffic sign identification is to integrate grid search, SVM classifier, and HOG features. There are many techniques to develop a traffic sign recognition system, like machine learning algorithms on CNN [8] [9], template matching [10], rule-based systems [11] [12] etc.

II. Literature review

Future technologies used to incorporate cloud computing, artificial intelligence, and Internet of Things will eventually help improve the intelligent driving technologies. TSR is therefore an essential element in intelligent transportation systems since the whole safety of driving depends on the understanding of traffic signs and the realization of appropriate avoidance and moving actions. An effective TSR system can ensure the safety of driver by fast and proper provision of real-time traffic information to drivers to make proper decisions or let the vehicle drive on its own to avoid danger. It is in this concern that traffic sign detection technology is at present being emphasized on a research involving important

traffic sign identification technologies. Through the evolution of traffic sign identification, we have categorized common detection techniques into four major groups, such as color-based techniques, shape-based techniques, and deep learning-based techniques.

A. Detection based on color-feature

In general, there are three kinds of traffic signs: warning, indication, and prohibition. According to the categories, each type has quite distinct color characteristics. On the RGB color space, Akatsuka et al [5]. marked out the red, yellow, and blue colors with threshold segmentation algorithms to finish the traffic sign detection within a threshold range of distinct colors. Based on this, in the 1980s.A sign-detection system based on RGB color was developed. Zhangka et al. counted all color details of sign images, calculated the red, yellow, and blue thresholds of statistical distribution, and extracted the global color characteristics. Huang Zhiyong et al. used RGB mapping on a row of three-component differences, which supports derivational empirical thresholds, splitting of recognition indications, and avoiding multiplications fully in this technique. B. Using form and feature for detection.

B. Shape-feature-based detection

Anandhalli and Baligar [13] utilized a Harris corner detector in order to locate triangle and rectangle symbols in the ROI, while locating corners in a defined control region. Li [14] exploited edge information during the detection of occluded traffic signs during driving like circle, triangle, and rectangles were identified within the image and masked over 95% of all traffic signs by the use of the nonparametric shape detector with form properties of scale invariant edge turning angles.From the multi-feature fusion of the traffic sign recognition methods, Jin et al. [15] proposed a two-module detector as follows: the first module uses the commonality of symbol borders to extract ROIs and the second module uses HOG and SVM to validate the effectiveness of the produced ROIs to detect the traffic signs of dataset. Zheng et al[16] had proposed sliding window detection(SWD). where this approach integrates channel feature classifier to search for the presence of traffic signals on different sizes. Gim et al. [16] has come up with a system that has two coarse filter modules specifically for the required and prohibited signs in the GTSDB; the first module is HOG and LDA-based. Both modules use an SVM classifier and operate with huge windows; the second uses a small sliding window. Although these attempts prove to be successful in traffic sign recognition with graphical approaches, they are not very good when it comes to complicated situations (like dim lighting, partly covered signs, etc.), and especially bad in recognizing signs with different orientations or perspectives.

C. Detection using deep learning

It utilizes learning and training to identify traffic signs and extract characteristics, which is extremely distinct from previous techniques. Relatively, the deep learning-based approach applies a better accuracy level and more robust generalization ability compared to the classical traffic sign detection approaches. Features acquisition through big data, strong feature-expression capabilities, insensitivity to influences from outside factors such as illumination and occlusion, it is actually a fundamental approach in itself. The algorithm of target detection has high detection accuracy; its basis is Prospective Region extraction. Girshick et al [5].designed a target detection system based on a Prospective Region, which is also popularly referred to as RCNN (Regions with CNN characteristics). True identification of objects and separation of semantic content requires a deep feature hierarchy which used Convolutional Neural Network (CNN). This it categorizes item suggestions, achieving an accuracy of object identification. However, it consumes much computational time and space as it repeats the extraction of every qualifying region and stores it in its memory. Meanwhile, region stretching applied by RCNN pool together every Prospective Region into a 227 × 227 size thus reducing the accuracy for the detection and further reducing the quality of the features CNN extraction. FPN combines, depth feature ,shallow feature to produce predictions on the several scales of feature pyramid scale. Candidate areas are extracted by layering an RPN network over the feature pyra- mid to achieve this. Such a concept raises the semantic quality of the shallow feature map and amplifies the precision in detecting tiny targets. One is YOLO network, which enforces a single view regarding detection-not as many classification or regression tasks but applies regression on the entire image for predicting the coordinates of the bounding boxes and class probabilities.

III. METHODOLOGY

A. Traffic Sign Dataset

Over fifty thousand photos were used from the GTSRB set specially designed for the traffic sign classification. There are forty-three classes into which the various traffic signs fall,like stop sign or no entrance signs, or speed restriction. For the reasons that the photos were taken in a variety of real-world scenarios, involving various lighting, shadows, occlusion, and distances, the classification process is harder and more demanding compared to real case-scenarios. Convolutional Neural Networks are frequently applied for designing machine learning models in order to identify traffic signs. The dataset consists a training set and a test set. Over 39,000 images comprise the training set, broken down into 43 subdirectories, each of which corresponds to a unique type of traffic sign. Thus,the samples in classes are imbalanced. The test set consists of more than 12,000 images and the other file named Test.csv that carries the path for each picture along with its corresponding label for it. Scaling, normalization, as well as other preprocessing techniques can enhance the model, addressing the irregular sizes in the dataset, irregular illumination and orientation. Two of the challenges of the dataset are class imbalance, where some classes contain a lot more images than others, and heterogeneity in image environments, such as varying lighting, occlusion, or weather

conditions. Due to these characteristics, GTSRB constitutes an excellent benchmark to build and test advanced models for traffic surveillance and driverless cars. The following table gives an overview of some prominent features of the dataset:

TABLE I
PROPERTIES OF THE DATASET USED FOR TRAINING AND TESTING

Attribute	Description
Number of Images	Over 50,000
Number of Classes	43
Image Resolution	Variable (resized to 32x32 for model input)
Training Set Size	39,000+ images
Test Set Size	12,000+ images
Label File	Test.csv for test images, includes file paths and labels
Challenge Factors	Class imbalance, varying lighting, occlusion, angles
Real-world Conditions	Captured in various environmental conditions

Fig. 1. All types of Images in Dataset

Fig. 2. Visualization of sample traffic sign images from the dataset representing various classes

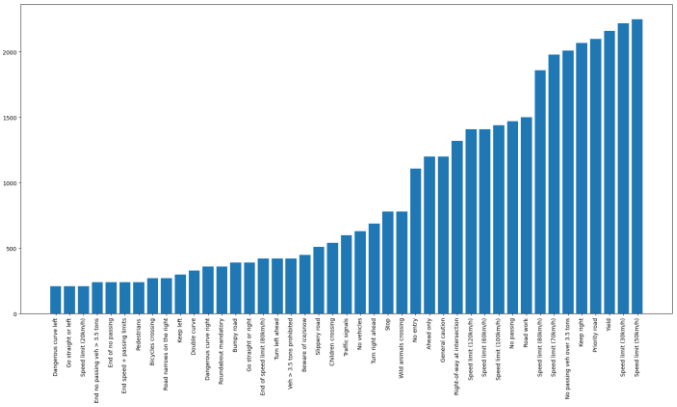

Fig. 3. Number Of Images in Each Class

B. *Preprocessing Techniques*

Gaussian Blur: Gaussian blur is one of the picture processing techniques that reduce noise of an image and detail. The picture is processed with a Gaussian function to reduce the high-frequency noise and keep the key edge and structure. This is very useful for traffic sign identification as it filters out all unwanted sounds leaving unique features of the traffic sign itself. It becomes soft and distributes itself in all possible directions from the central point such that extraneous features which may confuse the classification algorithm are minimized.

HSV Conversion: Another alternative to the RGB paradigm is the HSV, which can be particularly helpful in jobs involving picture analysis. Value gives the lightness of the color, while saturation is how intense the color is, and hue represents how it is represented. Conversion of the image into HSV helps recognize traffic signs more efficiently as this algorithm can focus more effectively on the signals' characteristics of color rather than getting diverted by brightness changes. This conversion makes color-based segmentation more robust to variations in different lighting conditions.

Binarization: This produces a binary image through binarization-a process that takes grayscale photographs and transforms them into black-and-white images with each pixel having pure black or white color. This results in the removal of in-between shades of gray, and it makes the method differentiate the significant features or characters from the signs, making the system quite helpful in the detecting the traffic signs. TSR uses binarization to make images even simpler for analysis to be made through emphasizing edges and unique features of signs. This allows the differentiation between the sign forms and the rest of the noise, much more so in complex environments.

Morphological operations:These binary images are then processed by morphological techniques, such as dilation, erosion, and closure, in order to further narrow down the shapes by eliminating all sorts of noise. Erosion reduces the borders of an object so that small, insignificant artifacts lose existence, whereas dilation amplifies them and makes features such as edges of a traffic sign much more visible. Finally, closing ensures that characteristics preserved of a traffic sign are actually prepared for analysis by filling gaps and reducing edges. These changes also make the forms of traffic signs much more recognizable and uniform.

ROI (Region of Interest) Extraction: This would reduce the background information with the isolation of the part of the image containing the traffic sign. It would pay more attention to the area of interest and block the other distracting areas of information within the image; hence, the network will learn to categorize the indicators more effectively. The procedure improves the accuracy of classification and gives better redictions for different crowded and heterogeneous conditions because it helps the model to focus only on the road traffic sign, which is most important part of image. only on the road traffic sign, which is the most important part of the image.

Fig. 4. Sample Images After Preprocessing from each Class

C. Model Building

The architecture used for traffic sign categorization uses convolutional neural networks (CNNs). The second layer from the top is a 2D convolutional layer named Conv2D that applies 16 filters and then ReLU activation, followed by another one of 32 filters. Dropout layer and MaxPooling are applied to avoid overfitting and reduce spatial dimensions. For further regularization, the architecture consists of deeper layers of 64 and 128 filters. In addition, there is a MaxPooling layer and Dropout. The feature maps are flattened into the 1D vector. A Dropout layer follows, along with dense layer containing 512 units,the ReLU activation. The output layer classifies the 43 types of traffic sign types, with softmax activation.

- Creating numpy arrays: After processing, numpy arrays are created for processed images as well as processed labels. This is an important step because the data have to be in the numerical array format so that deep learning.Frameworks like TensorFlow or PyTorch can process the input data efficiently.
- Normalization: The values of the pixel of the images are divided by 255.0 for normalization. This normalization brings the values within the range [0, 1] because pixel values normally vary between 0 and 255. Since normalization ensures that the input values are minimal and within a regular range, this in turn helps stabilise the CNN while training and leads to convergence.

$$\text{processed\_images} = \frac{\text{processed\_images}}{255.0}$$

- Label Encoding to_categorical: We need to convert the labels to one-hot encoded vectors with To_categorical. A label of 6, for instance, would translate into a vector of size 43, something like [0, 0, 0, 0, 0, 0, 1, 0,.]. This is necessary because one-hot encoding provides the model with an obvious way of contrasting a predicted output with the true class during training, and classification models such as CNNs return a list of probabilities for each class.

TABLE II
CNN MODEL ARCHITECTURE USED FOR TRAFFIC SIGN CLASSIFICATION

Layer Type	Description	Parameters
Input Layer	Accepts images of size 30x30 with 3 color channels (RGB).	Input shape = (30, 30, 3)
Conv2D Layer 1	2D Convolutional layer with 16 filters, kernel size 3x3, and ReLU activation.	Filters = 16, Kernel size = (3,3), Activation = ReLU
Conv2D Layer 2	Conv2D with 32 filters, kernel size 3x3, and ReLU	Filters = 32, Kernel size = (3,3), Activation = ReLU
MaxPool2D Layer 1	MaxPool2D layer to down sample feature maps.	Pool size = (2, 2)
Dropout Layer 1	Dropout layer to reduce overfitting.	Dropout rate = 0.25
Conv2D Layer 3	Conv2D layer having 64 filters, kernel size 3x3, and ReLU activation.	Filters = 64, Kernel size = (3,3), Activation = ReLU
Conv2D Layer 4	Conv2D layer having 128 filters, kernel size 3x3, and ReLU activation.	Filters = 128, Kernel size = (3,3), Activation = ReLU
MaxPool2D Layer 2	MaxPool2D layer to down sample feature maps.	Pool size = (2, 2)
Dropout Layer 2	Dropout layer to reduce overfitting.	Dropout rate = 0.25
Flatten Layer	Flattens the 2D matrices into a 1D vector for the fully connected layer.	-
Dense Layer 1	Fully connected layer with 512 neurons and ReLU activation.	Units = 512, Activation = ReLU
Dropout Layer 3	Dropout layer to reduce overfitting.	Dropout rate = 0.5
Dense Output Layer	Fully connected layer with 43 neurons (for 43 classes) and softmax activation.	Units = 43, Activation = Softmax

D. Model Enhancement

- Hyperparameter tuning: To tune hyperparameters, grid search or random search is applied in order to reduce the amount of calculation time and therefore improve accuracy.

- Early Stopping: This is a kind of regularizer against the overfitting problem; training will stop when the model's performance on the validation set begins to degrade.
- Dropout: This is another technique applied in layers. At each training step, it randomly removes units so that the model becomes more broadly applicable and less sensitive to any given single neuron.

IV. Result

The CNN's first neural network model The data from the train set is expressed in a set of graphs shows the accuracy of the train set, which was 70% used to train model, and that of test set, which was 30%. Apart from demonstrating the accuracy of the performance of the model, some test were carried out on the test datasets to confirm the correctness of the model. The accuracy of the model was 0.9969, with the test loss of 0.0102 and this is a fair assessment.

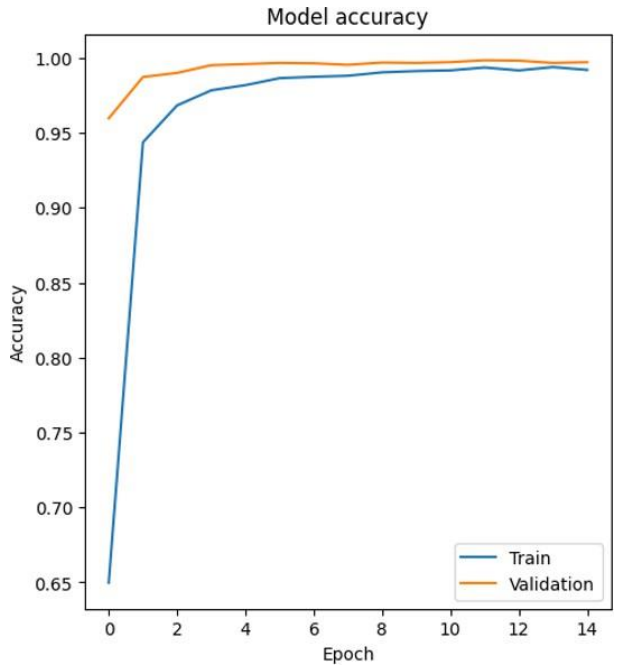

Fig. 5. Model Accuracy

V. Execution Time

The computation time spent during testing and training of the traffic sign recognition model was supported by the time that had to complete the project, which includes importing data, scaling and standardizing photos, training the model with the CNN architecture, and performing assessment processes. However, it would still depend on many factors such as size of dataset, epochs, the depth of the CNN model, and the amount of preprocessing done on images before entry into the CNN. For example, in my experiment, I have used morphological operations, HSV conversion, and a Gaussian blur on all images before feeding them into the CNN. Optimization of these operations and judicious use of the GPU resources directly

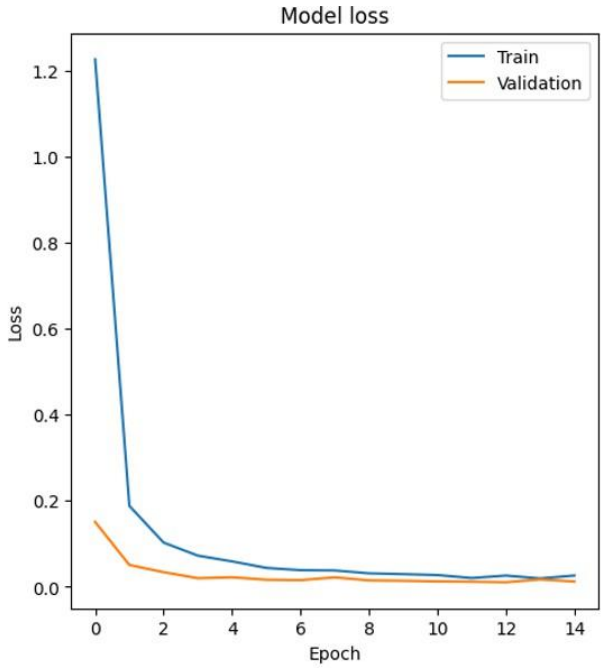

Fig. 6. Model Loss

reflects increased speed with relatively reduced execution times.

Table below displays the testing results of a model with a dropout layer in validation test. It indicates that the model performs equally well for seven groups, and evaluation of each group is performed in terms of time required in each iteration, recognition accuracy, and the validation loss. The time usage in iterations ranges between 2 and 18 seconds, where identification accuracy ranges from 95.97% to 99.64% with an increase in the number of iterations. The validation loss drops drastically, from 0.1509 to 0.0156, showing gener- alization improvement and less overfitting for each group.

TABLE III
VALIDATION TEST RESULTS WITH DROPOUT LAYERS

Group	Time/Iteration	Recognition accuracy	Validation Loss
1	18s	0.9597	0.1509
2	3s	0.9872	0.0508
3	2s	0.9901	0.0341
4	2s	0.9952	0.0200
5	3s	0.9959	0.0221
6	3s	0.9967	0.0165
7	3s	0.9964	0.0156

VI. Conclusion

Detection and recognition of traffic signs play a very important role in sustaining road safety, considering that most of the drivers will respect the laws of traffic only. In this article, the automation for the detection process has been

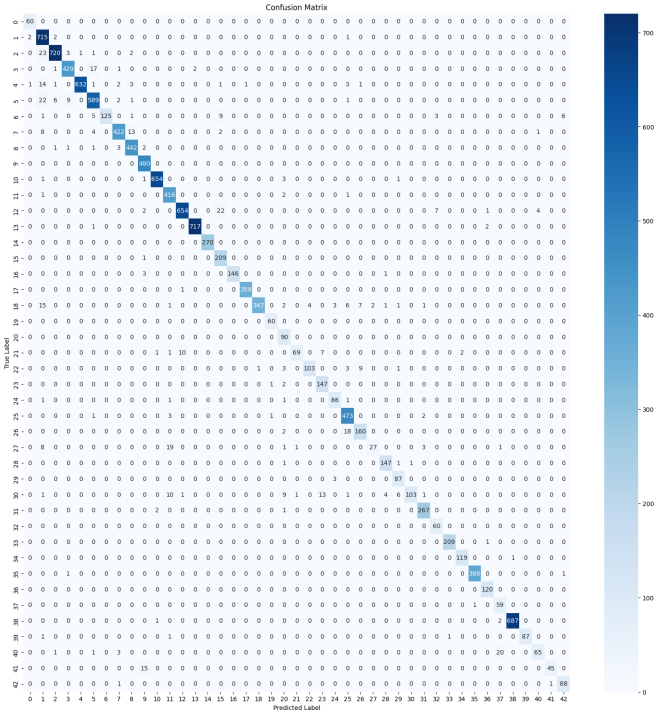

Fig. 7. Confusion Matrix

Fig. 8. Random image predictions from the traffic sign classification model

successfully achieved, along with the use of deep learning-based approaches, including Convolutional Neural Networks (CNNs). This technology is definitely a great advantage because it holds tremendous benefits for the distracted or visually defective drivers who may be quite unable to notice the signs or recognize them in realtime. It is important to note that the model's input is obtained through advanced image preprocessing techniques like morphological operations, transformation to HSV space, and Gaussian blur. Such augmentations help the model better identify signs in varied scenarios. This use of CNNs was very effective in learning the specific features of traffic signs,resulting in high classification accuracy. Adaptability and flexibility of CNNs combined with improvements in the training process data-driven make this a strong and reliable real-world Traffic Sign Recognition system. It is an essential contribution to the development of intelligent transportation systems that improve road safety and driving experience for everyone, and it offers good opportunities for further increasing accuracy and for its real-time deployment.

REFERENCES

[1] C. Liu, S. Li, F. Chang and Y. Wang, "Machine Vision Based Traffic Sign Detection Methods: Review, Analyses and Perspectives," *IEEE Access*, vol. 7, pp. 86578-86596, 2019, doi: 10.1109/ACCESS.2019.2924947.

[2] X. Peng, X. Chen and C. Liu, "Real-time traffic sign text detection based on deep learning," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 768, Mar. 2020.

[3] Y. Du, X. Zhang, G. Zhang and W. Sun, "Detection and recognition of text traffic signs above the road," *Int. J. Sensor Netw.*, vol. 35, no. 2, pp. 69-78, 2021.

[4] William, Marco Magdy, Pavly Salah Zaki, Bolis Karam Soliman, Kerolos Gamal Alexsan, Maher Mansour, Magdy El-Moursy, and Kerolos Khalil. "Traffic signs detection and recognition system using deep learning." In *2019 Ninth international conference on intelligent computing and information systems (ICICIS)*, pp. 160-166. IEEE, 2019.

[5] X. Qiao, "Research on Traffic sign recognition based on CNN Deep Learning Network," *Procedia Computer Science*, vol. 228, pp. 826-837, 2023.

[6] A. Alam and Z. A. Jaffery, "Indian traffic sign detection and recognition," *International Journal of Intelligent Transportation Systems Research*, vol. 18, pp. 98-112, 2020.

[7] Y. Liu and W. Zhong, "A novel SVM network using HOG feature for prohibition traffic sign recognition," *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[8] Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai, and W. Liu, "Traffic sign detection and recognition using fully convolutional network guided proposals," *Neurocomputing*, vol. 214, pp. 758-766, 2016.

[9] S. Mehta, C. Paunwala, and B. Vaidya, "CNN based traffic sign classification using adam optimizer," In *2019 international conference on intelligent computing and control systems (ICCS)*, pp. 1293-1298. IEEE, 2019.

[10] L. Jia, X. Shi, and L. Wei, "Design of Traffic Sign Detection and Recognition Algorithm Based on Template Matching," In *2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, pp. 237-240. IEEE, 2020.

[11] Z. Fazekas, G. Bala´zs, A. Boulmakoul, and P. Ga´spa´r, "Visualization of traffic sign related rules used in road environment-type detection," In *2019 Modern Safety Technologies in Transportation (MOSATT)*, pp. 53-57. IEEE, 2019.

[12] A. Aksjonov and V. Kyrki, "Rule-based decision-making system for autonomous vehicles at intersections with mixed traffic environment," In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pp. 660-666. IEEE, 2021.

[13] M. Anandhalli and V. P. Baligar, "An approach to detect vehicles in multiple climatic conditions using the corner point approach," *J. Intell. Syst.*, vol. 27, no. 3, pp. 363-376, Jul. 2018.

[14] L. L. Li, "Image edge detection algorithm and its application in traffic video segmentation," 2015.

[15] Y. Jin, Y. Fu, W. Wang, J. Guo, C. Ren and X. Xiang, "Multi-feature fusion and enhancement single shot detector for traffic sign recognition," *IEEE Access*, vol. 8, pp. 38931-38940, 2020.

[16] S. He et al., "Automatic Recognition of Traffic Signs Based on Visual Inspection," *IEEE Access*, vol. 9, pp. 43253-43261, 2021, doi: 10.1109/ACCESS.2021.3059052.

IEEE_Conference_Template _2_.pdf.1

ORIGINALITY REPORT

9%	4%	6%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Sim University Student Paper	1%
2	Xiaoxuan Qiao. "Research on Traffic sign recognition based on CNN Deep Learning Network", Procedia Computer Science, 2023 Publication	1%
3	Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023 Publication	1%
4	Shouhui He, Lei Chen, Shaoyun Zhang, Zhuangxian Guo, Pengjie Sun, Hong Liu, Hongda Liu. "Automatic Recognition of Traffic Signs Based on Visual Inspection", IEEE Access, 2021 Publication	1%
5	www.ijert.org Internet Source	1%
6	assets-eu.researchsquare.com Internet Source	<1%
7	jurnal.polsri.ac.id Internet Source	

		<1%
8	Subhash Chand Agrawal, Rajesh Kumar Tripathi, Ashutosh Gupta, Manish Sahu. "Traffic Sign Recognition System using CNN", 2023 World Conference on Communication & Computing (WCONF), 2023 Publication	<1%
9	Submitted to University of Alabama at Birmingham Student Paper	<1%
10	pt.scribd.com Internet Source	<1%
11	file.techscience.com Internet Source	<1%
12	viraai.com Internet Source	<1%
13	Submitted to AUT University Student Paper	<1%
14	Deepak Kumar Jain, A. KalyanapuSrinivas, B. SirasanagondlaVenkata Naga Srinivasu, R. Manikandan. "Machine learning based monitoring system with IoT using wearable sensors and Pre-convoluted Fast Recurrent Neural Networks (P-FRNN)", IEEE Sensors Journal, 2021 Publication	<1%

15	openaccess.thecvf.com Internet Source	<1%
16	Kathiresan Kandasamy, Yuvaraj Natarajan, K. R. Sri Preethaa, Ahmed Abdi Yusuf Ali. "A Robust TrafficSignNet Algorithm for Enhanced Traffic Sign Recognition in Autonomous Vehicles Under Varying Light Conditions", Neural Processing Letters, 2024 Publication	<1%
17	dokumen.pub Internet Source	<1%
18	www.isteonline.in Internet Source	<1%
19	www.techscience.com Internet Source	<1%
20	T. Surekha, R. Siva Rama Prasad. "LDT-MRF: Log decision tree and map reduce framework to clinical big data classification", International Journal of Engineering & Technology, 2017 Publication	<1%
21	Ton Duc Thang University Publication	<1%
22	"Inventive Communication and Computational Technologies", Springer Science and Business Media LLC, 2021	<1%

Publication

23	"Proceedings of International Conference on Communication and Computational Technologies", Springer Science and Business Media LLC, 2024 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		