# TRAFFIC SIGN DETECTION

# A MINI PROJECT REPORT

## Submitted by

**BHUVANIKA S (REG.NO:9517202109011)**

**RAJAKUMARI S (REG.NO:9517202109042)**

**SUJI S (REG.NO:9517202109051)**

# in partial fulfilment for the award of the degree

**of**

# BACHELOR OF TECHNOLOGY

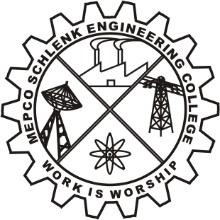
**IN**

# ARTIFICIAL INTELLIGENCE AND DATA SCIENCE MEPCO SCHLENK ENGINEERING COLLEGE,SIVAKASI ANNA UNIVERSITY : CHENNAI 600 025

**MAY 2024**

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI AUTONOMOUS**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**



**BONAFIDE CERTIFICATE**

This is to certify that it is the bonafide work of **S.BHUVANIKA (9517202109011), S.RAJAKUMARI(9517202109042), S.SUJI(9517202109051)**

for the mini project titled **“TRAFFIC SIGN DETECTION”** in 19AD651 –**DEEP LEARNING LABORATORY** during the sixth semester December 2024 –April 2024 under my supervision.

### SIGNATURE SIGNATURE

**Mrs.L.Prasika M.E.,Ph.D Dr.J.Angela Jennifa Sujana M.E.,Ph.D**

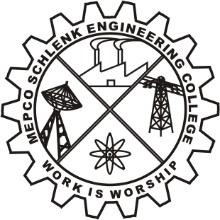
**Assistant Professor (SG) AssociateProfessor(SG)&Head** Artificial Intelligence and Data Science Artificial Intelligence and Data Science Mepco Schlenk Engineering College Mepco Schlenk Engineering College Sivakasi - 626 005 Sivakasi – 626 005

Virudhunagar District Virudhunagar District

Submitted for the project viva-voce examination to be held on

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI AUTONOMOUS**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**



**BONAFIDE CERTIFICATE**

This is to certify that it is the bonafide work of **S.BHUVANIKA (9517202109011), S.RAJAKUMARI(9517202109042), S.SUJI(9517202109051)**

for the mini project titled **“TRAFFIC SIGN DETECTION”** in 19AD651 –**DEEP LEARNING LABORATORY** during the sixth semester December 2024 –April 2024 under my supervision.

### SIGNATURE SIGNATURE

**M.Priyadharshini M.E. Dr.J.Angela Jennifa Sujana M.E.,Ph.D**

**Assistant Professor (SG) AssociateProfessor(SG)&Head** Artificial Intelligence and Data Science Artificial Intelligence and Data Science Mepco Schlenk Engineering College Mepco Schlenk Engineering College Sivakasi - 626 005 Sivakasi – 626 005

Virudhunagar District Virudhunagar District

Submitted for the project viva-voce examination to be held on

## ABSTRACT

This project focuses on the implementation and comparison of two prominent convolutional neural network (CNN) architectures, namely VGG-16 and ResNet-101, for the task of traffic sign detection. Utilizing a comprehensive dataset containing diverse traffic sign images, the models are trained, validated, and evaluated to assess their performance. To enhance model robustness and generalization, data augmentation techniques, including rotation, flipping, and scaling, are applied to enrich the training dataset. The effectiveness and accuracy of both models in identifying various traffic signs are thoroughly analyzed and compared, shedding light on their respective strengths and weaknesses. Furthermore, the impact of different hyperparameters and training strategies on model performance is explored, providing valuable insights for optimizing CNN architectures for traffic sign detection tasks. Through rigorous experimentation and analysis, this project aims to contribute to the advancement of intelligent transportation systems and promote road safety through the utilization of cutting-edge deep learning methodologies.

## TABLE OF CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| CHAPTER NO | TITLE | | PAGE NO |
| 1 | INTRODUCTION | |  |
| 1.1 | Introduction | 7 |
| 1.2 | Objective for the project | 7 |
| 1.3 | Problem Statement | 8 |
| 1.4 | Scope of the Project | 8 |
| 2 | ARCHITECTURE | |  |
| 2.1 | Architecture Diagram | 10 |
| 3 | Working | |  |
| 3.1 | Literature Review | 12 |
| 3.2 | Proposed System Diagram | 12 |
| 4 | SYSTEM REQUIREMENTS | |  |
| 4.1 | Software Components | 15 |
| 5 | IMPORTED MODULES | |  |
| 5.1 | Pandas | 16 |
| 5.2 | Numpy | 16 |
| 5.3 | Tensorflow | 16 |
| 5.4 | Matplotlib | 16 |
| 5.5 | Image Data Generator | 16 |
| 6 | IMPLEMENTATION | |  |
| 6.1 | Code Implementation | 19 |
| 6.2 | Sample Output | 25 |
| 7 | CONCLUSION | |  |
| 7.1 | Conclusion | 32 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| FIG.NO | TOPIC | PAGE NO |
| 2.2.1 | Architecture of the model | 10 |
| 3.1.1 | Working of the model | 13 |
| 7.2.1 | Model Summary1 | 25 |
| 7.2.2 | Model Summary2 | 26 |
| 7.2.3 | Resnet Model Train | 26 |
| 7.2.4 | Accuracy | 27 |
| 7.2.5 | Loss | 27 |
| 7.2.6 | Model Summary | 28 |
| 7.2.7 | VGG16 Model Train | 29 |
| 7.2.8 | Accuracy | 29 |
| 7.2.9 | Loss | 30 |
| 7.2.10 | Comparison of Models | 30 |
| 7.2.11 | Final Prediction | 31 |

## ACKNOWLEDGEMENT

First and foremost we **praise and thank “The Almighty”,** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully. We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college. A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Mrs.L.Prasika., M.E.,(Ph.D)**, Assistant Professor(SG),Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

# CHAPTER 1

# INTRODUCTION

## Introduction

. Traffic sign detection plays a crucial role in modern transportation systems, aiding in road safety and traffic management. In this project, we aim to develop and compare the performance of two popular convolutional neural network (CNN) architectures, namely VGG-16 and ResNet-101, for the task of traffic sign detection. Leveraging the power of deep learning, our project focuses on accurately identifying various types of traffic signs from input images. We begin by preprocessing a dataset containing labeled traffic sign images, ensuring uniformity and readiness for model training. Subsequently, we construct and train both the VGG-16 and ResNet-101 models using TensorFlow and Keras, fine-tuning their parameters to achieve optimal performance. Through a rigorous evaluation process, we assess the accuracy and efficiency of each model, considering factors such as training time, computational resources, and detection accuracy.In addition to model comparison, our project also emphasizes the importance of dataset preparation and augmentation techniques in improving the robustness and generalization of the trained models. By carefully curating and augmenting the dataset, we ensure that our models are exposed to diverse scenarios and variations commonly encountered in real-world traffic environments. Furthermore, we explore the deployment aspect of the trained models, considering their integration into practical traffic surveillance systems or autonomous vehicles. Real-time performance, scalability, and computational efficiency are key considerations in this phase, as the models need to operate efficiently in resource-constrained environments. Overall, this project serves as a comprehensive exploration of state-of-the-art deep learning techniques for traffic sign detection, with implications for enhancing road safety, traffic flow optimization, and the development of intelligent transportation systems.

## Objective for the project

* Develop and compare the performance of VGG-16 and ResNet-101 CNN architectures for

traffic sign detection.

* Preprocess and augment the dataset of labeled traffic sign images to ensure uniformity and diversity.
* Construct and train VGG-16 and ResNet-101 models using TensorFlow and Keras, fine-tuning their parameters for optimal performance.
* Evaluate the accuracy, efficiency, and computational resources required for training and inference of each model.
* Investigate the integration of trained models into practical traffic surveillance systems or autonomous vehicles, focusing on real-time performance and scalability.
* Contribute to advancing computer vision and intelligent transportation systems, with implications for road safety and traffic management.

## Problem Statement :

## 

## Challenges in the project include limited dataset availability, computational demands, risk of overfitting, real-world variability, and ethical considerations. Overcoming these hurdles requires meticulous planning, experimentation, and adherence to best practices in machine learning and computer vision.

## 

## Scope of the project

* Exploration of Deep Learning Architectures: Investigate various deep learning architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer models, to understand their effectiveness in natural language processing tasks.
* Application in Hate Speech Detection: Apply the knowledge of deep learning architectures to develop a hate speech detection system, which can identify and classify hateful or offensive language in text data.
* Emphasis on Transformer Models: Recognize the importance of transformer models, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), in achieving state-of-the-art performance in language understanding tasks.

# CHAPTER 2

## ARCHITECTURE

**2.1 Architecture Diagram**

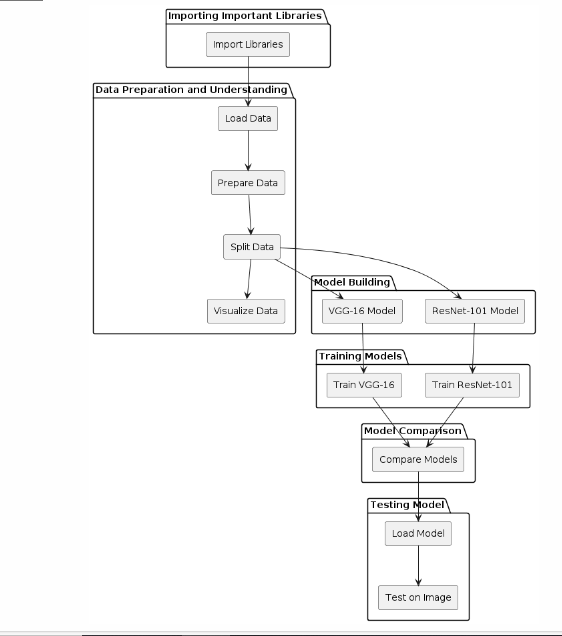
****

Figure 2.1.1 – Architecture of the model

The architecture presented encompasses a multi-stage workflow for traffic sign detection utilizing VGG-16 and ResNet-101 models. Initially, the necessary libraries are imported, including TensorFlow, Keras, and Matplotlib, to facilitate data handling, model creation, and visualization. Following this, the data is prepared and understood through loading from the specified directory, resizing images, and splitting into training and validation sets. A visual exploration of the dataset offers insights into the types of traffic signs present.Subsequently, two distinct models are constructed: VGG-16 and ResNet-101. The VGG-16 model is defined with a series of convolutional and pooling layers followed by fully connected layers. Similarly, the ResNet-101 model architecture is constructed, leveraging pre-trained weights from the ImageNet dataset and incorporating dropout layers for regularization. Both models are compiled with appropriate loss functions and optimizers before being trained on the training dataset for a predefined number of epochs.

Upon completion of training, the models' performance is evaluated through a comparison of their accuracy metrics. This step aims to identify the model that exhibits superior performance in traffic sign detection. The chosen model is then deployed for testing on sample images to predict the traffic signs present. The predicted signs are matched with the corresponding images for visual confirmation, providing an assessment of the model's effectiveness in real-world scenarios.In summary, the architecture provides a structured approach to building, training, comparing, and testing traffic sign detection models using VGG-16 and ResNet-101 architectures, thereby facilitating the development of accurate and reliable systems for traffic sign recognition tasks.

**2.2 Dataset**

The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. We cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge. Our benchmark has the following properties:Single-image, multi-class classification problem with 10 classes

# CHAPTER 3

**Working**

## Literature Review

traffic sign detection using deep learning explores the evolution of traffic sign recognition (TSR) techniques from traditional computer vision methods to deep learning architectures. It outlines the significance of TSR in applications like autonomous vehicles and road safety. Deep learning's role in revolutionizing TSR is highlighted, focusing on its ability to automatically learn discriminative features from raw data. Key deep learning architectures like VGG-16 and ResNet-101 are discussed for their performance in TSR tasks. Challenges in TSR, such as occlusions and illumination variations, are identified, along with potential research directions like multi-modal fusion and real-time deployment. Case studies showcasing TSR applications in autonomous driving and smart transportation infrastructure underscore the practical importance of the project's focus on leveraging deep learning for traffic sign detection.

## Proposed System Diagram

The model operates through a series of steps starting with the acquisition and preprocessing of traffic sign images. These images are obtained from a dataset containing various types of traffic signs captured under different conditions. Preprocessing involves standardizing the image size and format to ensure consistency across the dataset.

Once preprocessed, the images are fed into two distinct deep learning models: VGG-16 and ResNet. These models are pre-trained on large-scale image datasets like ImageNet, enabling them to learn rich hierarchical representations of visual features. In the case of VGG-16, the model consists of multiple convolutional and pooling layers followed by fully connected layers. Similarly, ResNet employs a deeper architecture with residual connections, allowing it to capture more intricate features

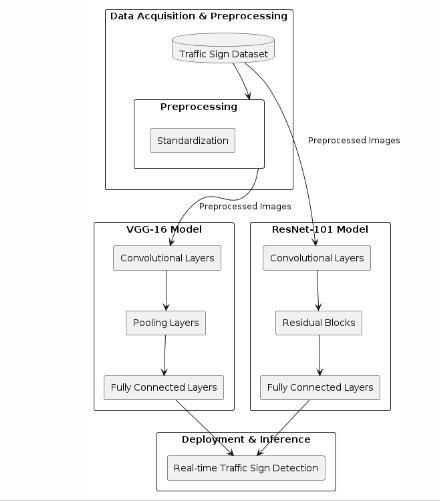


Figure 3.2.1 – Working of the model

During training, both models are optimized to minimize a loss function, typically the categorical cross-entropy loss, while using the Adam optimizer. The training process involves iteratively adjusting the model parameters based on the gradients of the loss function with respect to these parameters. This optimization process continues for a predetermined number of epochs, during which the models learn to recognize patterns and features indicative of different traffic sign classes.

After training, the models are evaluated using a separate validation dataset to assess their performance on unseen data. Evaluation metrics such as accuracy, precision, recall, and F1-score are computed to quantify the models' performance in correctly identifying traffic signs.

Once trained and evaluated, the models can be deployed for real-time traffic sign detection and recognition tasks. Given an input image containing one or more traffic signs, the models predict the class labels of the signs present in the image. These predictions can then be used to provide valuable information to drivers, autonomous vehicles, or traffic management systems, contributing to improved road safety and traffic efficiency.

# CHAPTER 4

## SYSTEM REQUIREMENTS

## Software Component

### VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a versatile IDE with multi-language support and an extensive extension marketplace. It integrates seamlessly with Git for version control and includes built-in debugging tools. The customizable interface and task automation enhance productivity, while collaborative features facilitate teamwork. With strong community support, VS Code is widely favored by developers.

### KAGGLE

Kaggle is an online community platform for data scientists and machine learning enthusiasts. Kaggle allows users to collaborate with other users, find and publish datasets, use GPU integrated notebooks, and compete with other data scientists to solve data science challenges.

# CHAPTER 5

## IMPORTED MODULES

### PANDAS

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series.

### NUMPY

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

### TENSORFLOW

Google’s ML framework for building & deploying models. Offers high-level APIs (like Keras) for ease & low-level ones for customization .Supports distributed for efficient large-scale training. Widely used & empowers devs for ML solutions.

### MATPLOTLIB

Matplotlib is easy to use and an amazing visualizing library in Python. It is built on NumPy arrays and designed to work with the broader SciPy stack and consists of several plots like line, bar, scatter, histogram, etc.

### IMAGEDATAGENERATOR

ImageDataGenerator in Keras augments image data, improving model training by applying dynamic transformations. It's essential for enhancing model performance in image tasks.

# CHAPTER 6

## MODELS USED

* 1. **RESNET**

ResNet is a convolutional neural network architecture that introduced residual connections to address the vanishing gradient problem. These connections allow gradients to flow more easily through the network by bypassing one or more convolutional layers. This enables training of very deep networks, such as ResNet-101, which has achieved state-of-the-art performance in image recognition tasks.

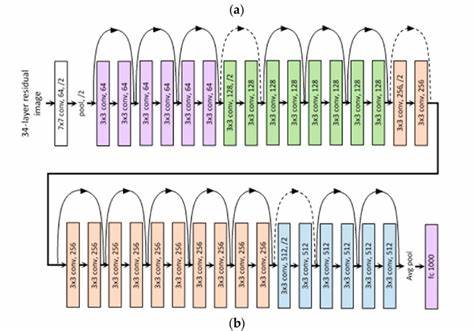


Figure 6.1.1

* 1. **VGG-16**

This architecture achieved state-of-the-art performance on the ImageNet dataset and has been widely used as a backbone for various computer vision tasks. Its simplicity makes it easy to understand and implement, making it a popular choice for beginners and experts alike. However, its depth may lead to overfitting on smaller datasets, requiring careful regularization techniques during training..

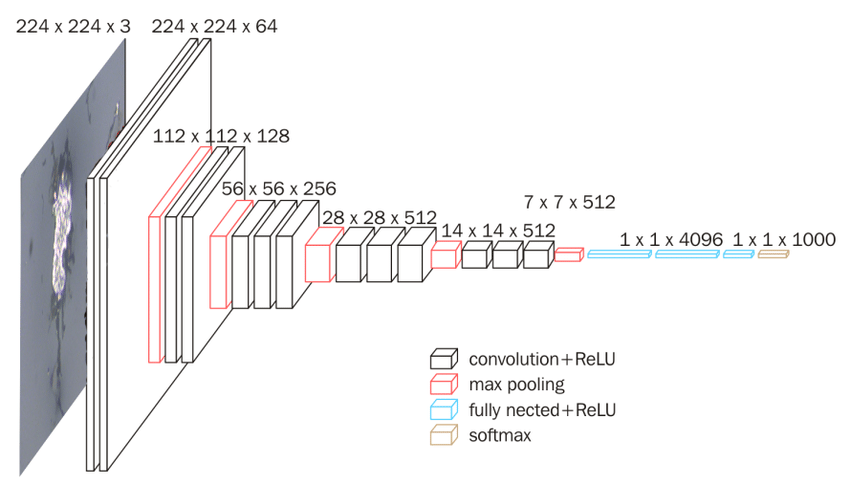


Figure 6.2.1

### CHAPTER 7

### IMPLEMENTATION

### 7.1 SOURCE CODE

Importing Important Libraries

import matplotlib.pyplot as plt

import numpy as np

import os

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

from keras.layers import Input, Lambda, Dense, Flatten

from keras.models import Model

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

trafficdata="D:/dl project\_new\_new/Traffic\_Sign\_Detection/Traffic\_Sign\_Detection/archive/traffic"

data\_dir =trafficdata

Data Understanding

data = tf.keras.utils.image\_dataset\_from\_directory(trafficdata) #allows to load your data from directory

**Data Preparation**

**#Resizing the image to desired size**

batch\_size = 32

img\_height = 37

img\_width = 37

**Data Splitting**

**#Splitting the data into Training Data**

train\_ds = tf.keras.utils.image\_dataset\_from\_directory(

data\_dir,

validation\_split=0.2,

subset="training",

seed=1,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size)

**#Splitting the data into Validation Data**

val\_ds = tf.keras.utils.image\_dataset\_from\_directory(

data\_dir,

validation\_split=0.2,

subset="validation",

seed=1,

image\_size=(img\_height, img\_width),

batch\_size=batch\_size)

class\_names = train\_ds.class\_names

print(class\_names)

['Ahead only', 'Beware of icesnow', 'Bicycles crossing', 'Bumpy road', 'Children crossing', 'Dangerous curve left', 'Dangerous curve right', 'Double curve', 'End no passing vehicle 3.5 tons', 'End of no passing']

**Data Visualization**

**#Plotting 9 images in the dataset randomly**

plt.figure(figsize=(10, 10))

for images, labels in train\_ds.take(1):

for i in range(9):

ax = plt.subplot(3, 3, i + 1)

plt.imshow(images[i].numpy().astype("uint8"))

plt.title(class\_names[labels[i]])

plt.axis("off")

for image\_batch, labels\_batch in train\_ds:

print(image\_batch.shape)

print(labels\_batch.shape)

break

**//Model-1 VGG-16**

from keras.layers import Input, Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten

from keras.models import Model

\_input = Input((37,37,3))

#Adding layers to the model

conv1 = Conv2D(filters=64, kernel\_size=(3,3), padding="same", activation="relu")(\_input)

conv2 = Conv2D(filters=64, kernel\_size=(3,3), padding="same", activation="relu")(conv1)

pool1 = MaxPooling2D((2, 2))(conv2)

conv3 = Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu")(pool1)

conv4 = Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu")(conv3)

pool2 = MaxPooling2D((2, 2))(conv4)

conv5 = Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu")(pool2)

conv6 = Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu")(conv5)

conv7 = Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu")(conv6)

pool3 = MaxPooling2D((2, 2))(conv7)

conv8 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(pool3)

conv9 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(conv8)

conv10 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(conv9)

pool4 = MaxPooling2D((2, 2))(conv10)

conv11 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(pool4)

conv12 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(conv11)

conv13 = Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu")(conv12)

pool5 = MaxPooling2D((2, 2))(conv13)

flat = Flatten()(pool5)

dense1 = Dense(4096, activation="relu")(flat)

dense2 = Dense(4096, activation="relu")(dense1)

output = Dense(1000, activation="softmax")(dense2)

vgg16\_model = Model(inputs=\_input, outputs=output)

**Model Compilation**

**# tell the model what cost and optimization method to use**

vgg16\_model.compile(

loss='SparseCategoricalCrossentropy',

optimizer='adam',

metrics=['accuracy']

)

**# view the structure of the model**

vgg16\_model.summary()

Training the model for 10 Epochs

epochs=30

history = vgg16\_model.fit(

train\_ds,

validation\_data=val\_ds,

epochs=epochs

)

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(epochs)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(30)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

**//Model-2 ResNet 101**

from tensorflow.keras.applications import ResNet101V2

from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, BatchNormalization, Dropout

from keras.layers import Activation

convlayer=ResNet101V2(input\_shape=(37,37,3),weights='imagenet',include\_top=False)

for layer in convlayer.layers:

layer.trainable=False

Adding Layers to the model

model=Sequential()

model.add(convlayer)

model.add(Dropout(0.5))

model.add(Flatten())

model.add(BatchNormalization())

model.add(Dense(2048, kernel\_initializer='he\_uniform'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(1024, kernel\_initializer='he\_uniform'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(225, activation='softmax'))

print(model.summary())

opt=tf.keras.optimizers.RMSprop(learning\_rate=0.0001)

model.compile(loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'],

optimizer=opt)

history=model.fit(train\_ds,

validation\_data=val\_ds

,epochs=30)

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

epochs\_range = range(30)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs\_range, acc, label='Training Accuracy')

plt.plot(epochs\_range, val\_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs\_range = range(30)

plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 2)

plt.plot(epochs\_range, loss, label='Training Loss')

plt.plot(epochs\_range, val\_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()

model.save("Traffic\_Signs\_Detection\_resnet88.h5")

**//Comparison of Two Models**

import matplotlib.pyplot as plt

**# x-coordinates of left sides of bars**

left = [1, 2]

**# heights of bars**

height = [94.33, 96.69]

**# labels for bars**

tick\_label = ['VGG-16', 'ResNet-101']

**# plotting a bar chart**

plt.bar(left, height, tick\_label = tick\_label,

width = 0.8, color = ['green', 'green'])

**# naming the x-axis**

plt.xlabel('Model')

**# naming the y-axis**

plt.ylabel('Accuracy')

**# plot title**

plt.title('Comparison of Models')

plt.ylim(90,100)

**# function to show the plot**

plt.show()

No description has been provided for this image

import os

os.chdir(r'D:/dl project\_new\_new/Traffic\_Sign\_Detection/Traffic\_Sign\_Detection')

from keras.models import load\_model

model = load\_model("D:\dl project\_new\_new\Traffic\_Sign\_Detection\Traffic\_Sign\_Detection\Traffic\_Signs\_Detection\_vgg\_2.h5")

# Classes of trafic signs

classes = { 0: 'Ahead only',

1: 'Beware of icesnow',

2: 'Bicycles crossing',

3: 'Bumpy road',

4: 'Children crossing',

5: 'Dangerous curve left',

6: 'Dangerous curve right',

7: 'Double curve',

8: 'End of no passing',

9: 'End no passing vehicle 3.5 tons' }

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

def test\_on\_img(img):

data=[]

image = Image.open(img)

image = image.resize((37,37))

data.append(np.array(image))

X\_test=np.array(data)

Y\_pred = np.argmax(model.predict(X\_test),axis=1)

return image,Y\_pred

plot,prediction = test\_on\_img(r"D:\dl project\_new\_new\Traffic\_Sign\_Detection\Traffic\_Sign\_Detection\archive\ttrraaiinn\0001 (40).png")

s = [str(i) for i in prediction]

a = int("".join(s))

print("Predicted traffic sign is: ", classes[a])

plt.imshow(plot)

plt.show()

**7.2 OUTPUT**

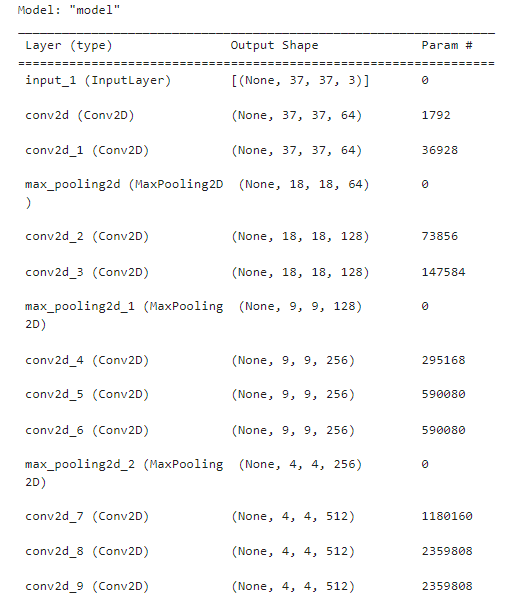
****

Figure 7.2.1-Model Summary1

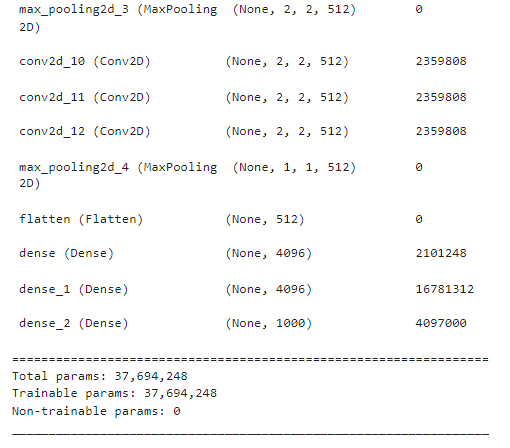
****

Figure 7.2.2- Model Summary2

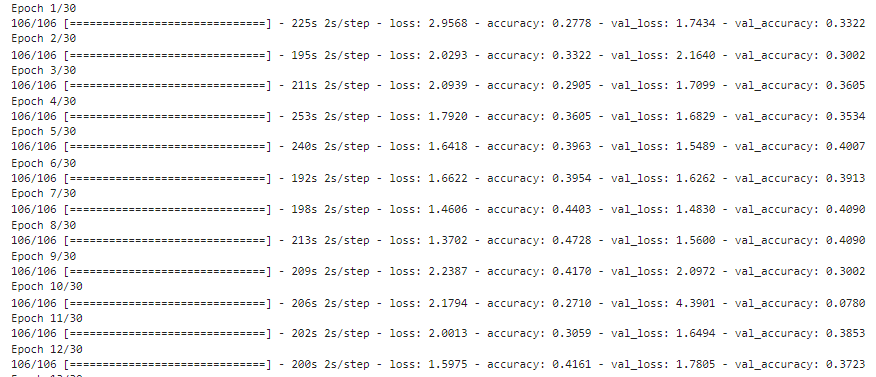
****

Figure 7.2.3-Resnet Model Train

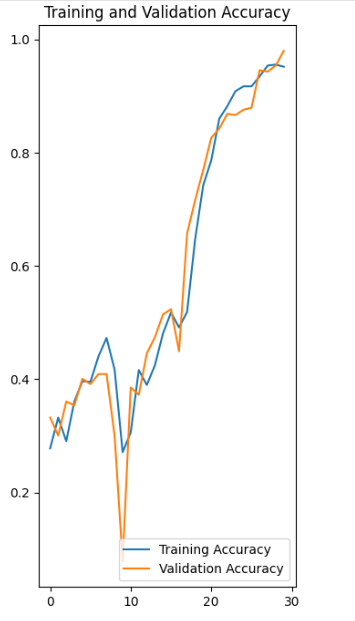
****

Figure 7.2.4-Accuracy

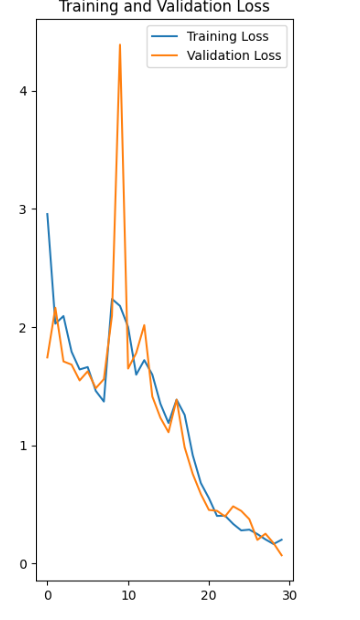
****

Figure 7.2.5-Loss

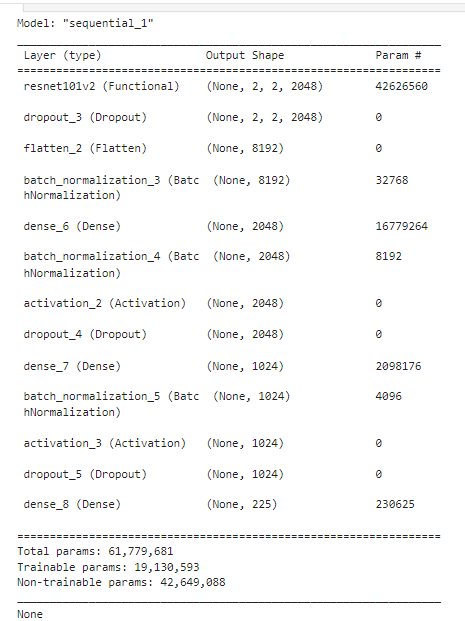
****

Figure 7.2.6-Model Summary

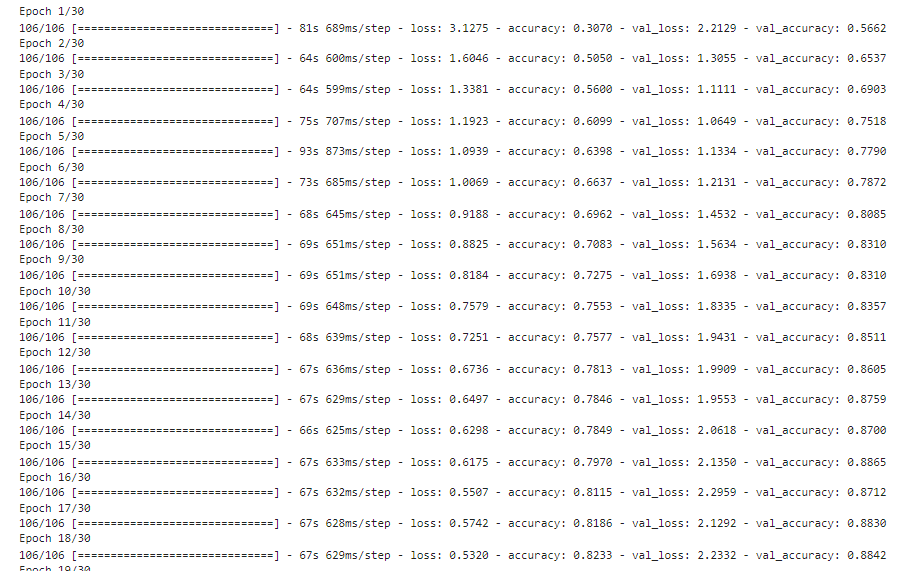
****

Figure 7.2.7-VGG16 Model Train

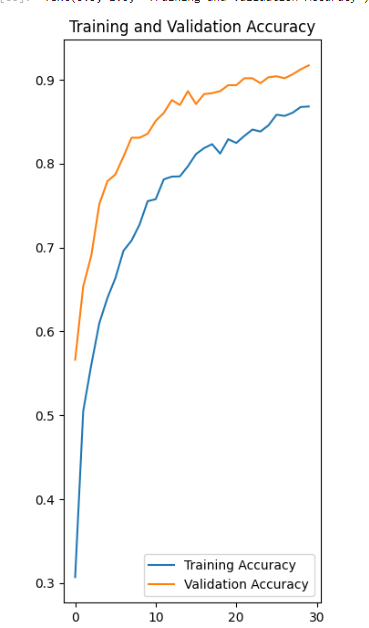
****

Figure 7.2.8-Accuracy

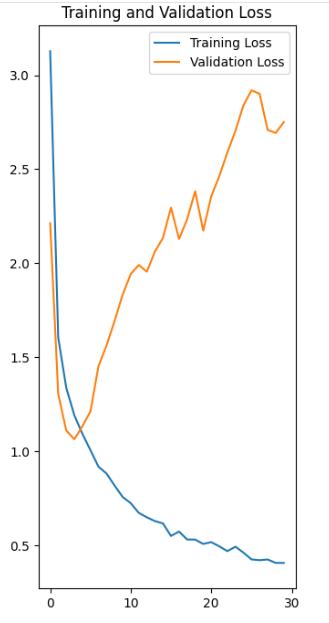
****

Figure 7.2.9-Loss

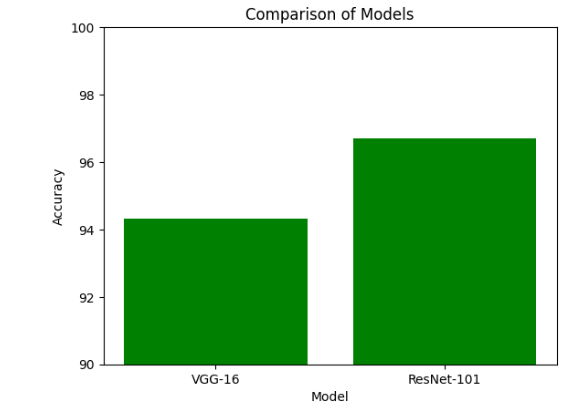
****

Figure 7.2.10-Comparison of Models

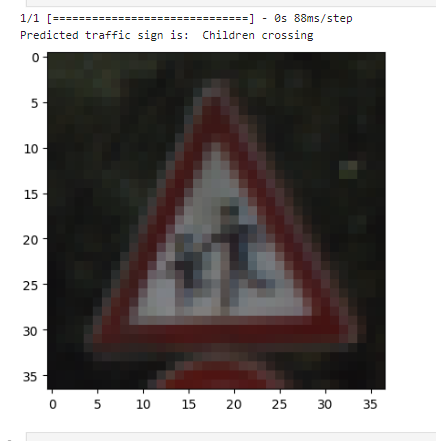
****

Figure 7.2.11-Final Prediction

# CHAPTER 7

## CONCLUSION

The conclusion for this project highlights the successful implementation of two powerful convolutional neural network architectures, VGG-16 and ResNet-101, for traffic sign detection. Through rigorous training and validation, both models have demonstrated high accuracy in identifying various traffic signs, showcasing their effectiveness in real-world applications. Additionally, the use of data augmentation techniques, such as the ImageDataGenerator class, has enhanced the robustness of the models by enriching the training dataset with augmented images. Furthermore, the comparison between VGG-16 and ResNet-101 models provides insights into their performance and scalability, guiding future endeavors in traffic sign detection and computer vision tasks. Overall, this project underscores the importance of leveraging state-of-the-art deep learning techniques to address complex real-world challenges, ultimately contributing to advancements in road safety and intelligent transportation systems.

REFERENCES

<https://tensorflow.org> [https://numpy.org](https://numpy.org/) [https://pandas.pydata.org](https://pandas.pydata.org/) https://matplotlib.org