# 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)** | **Professor&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**

# LIST OF FIGURES

## 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.,**Professor & 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 DataScience 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

## 1.1 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.

## 1.2 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.

## 1.3 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.

## 1.4 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

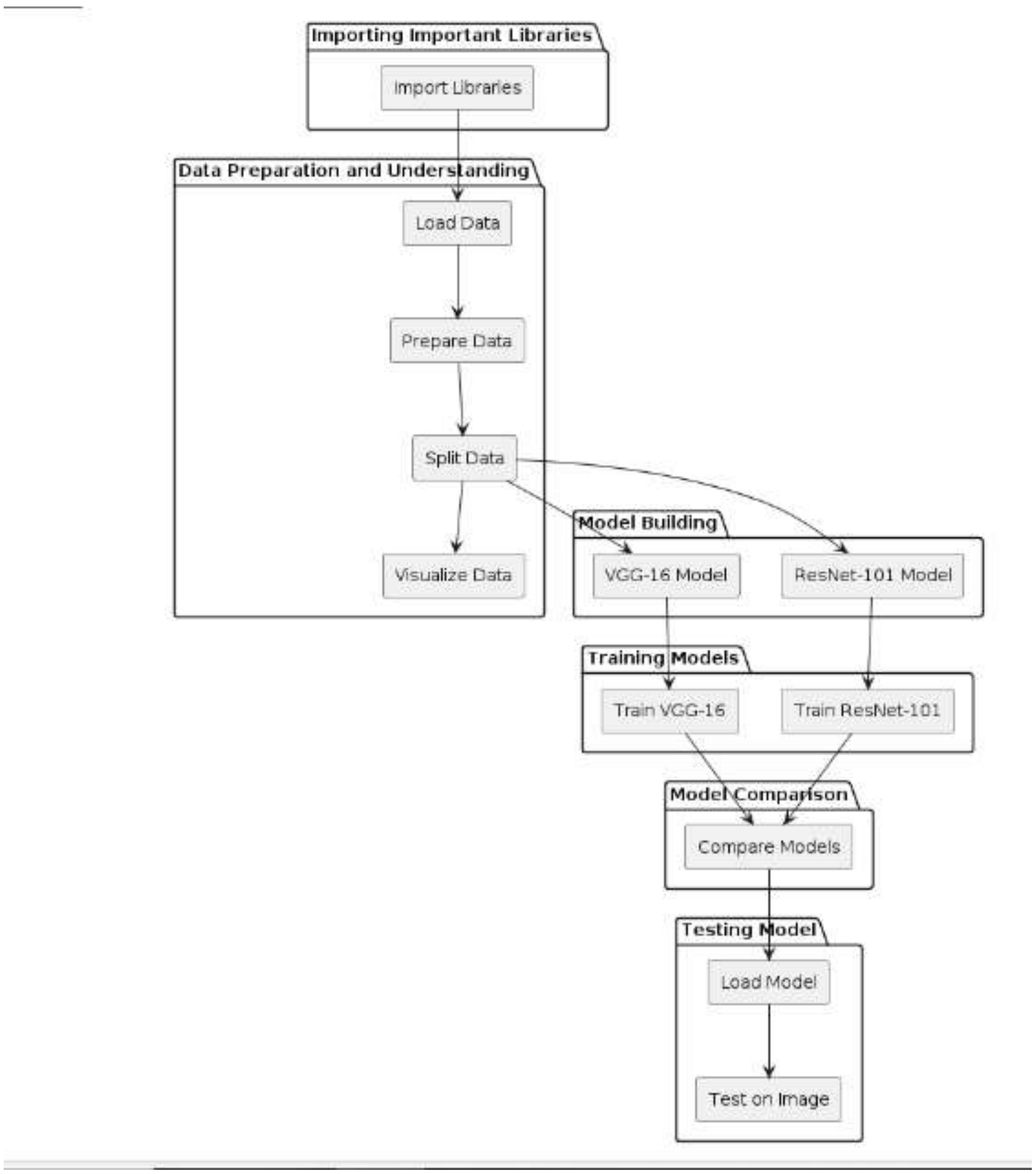


Figure 2.1.1 – Architecture of the model

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 (GTSRB) is a widely recognized dataset and benchmark for evaluating algorithms in the field of traffic sign recognition. It was created to address the need for standardized evaluation methods and benchmarks for traffic sign detection and classification systems. consists of over 50,000 images of traffic signs collected from real-world scenarios. These images cover a wide range of traffic sign types, including speed limits, stop signs, yield signs, and more. It consists of 10 classes and they were captured under various lighting conditions, weather conditions, and environmental factors, reflecting the diverse challenges faced by traffic sign recognition systems in real-world settings.

# CHAPTER 3

# Working

## 3.1 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.

## 3.2 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

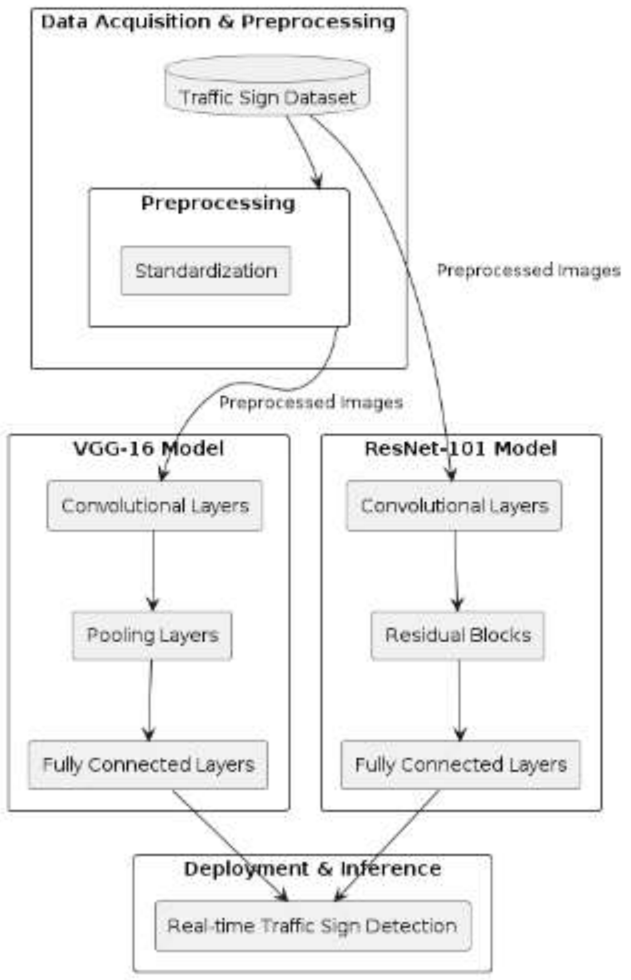


Figure 3.2.1 – Working of the model

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

## 4.1 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

### 5.1 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.

### 5.2 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.

### 5.3 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.

### 5.4 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 likeline, bar, scatter, histogram, etc.

### 5.5 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

### 6.1 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.

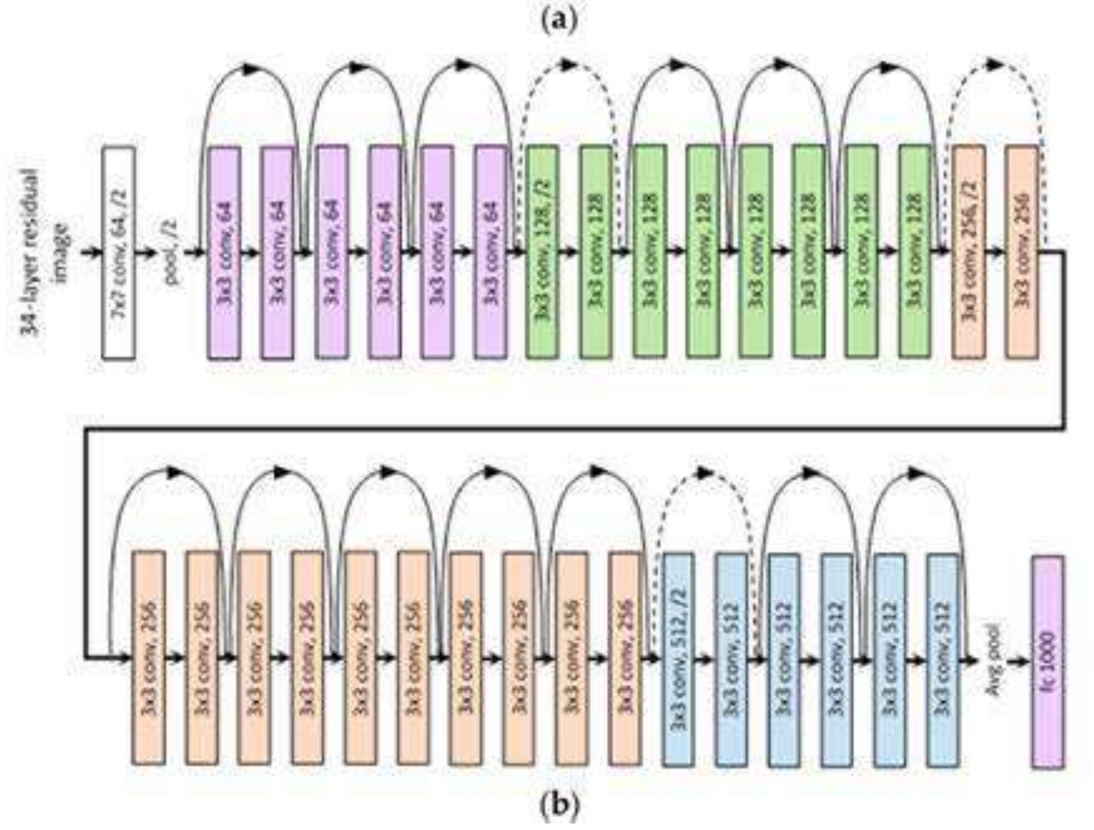


Figure 6.1.1

## 6.2 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..

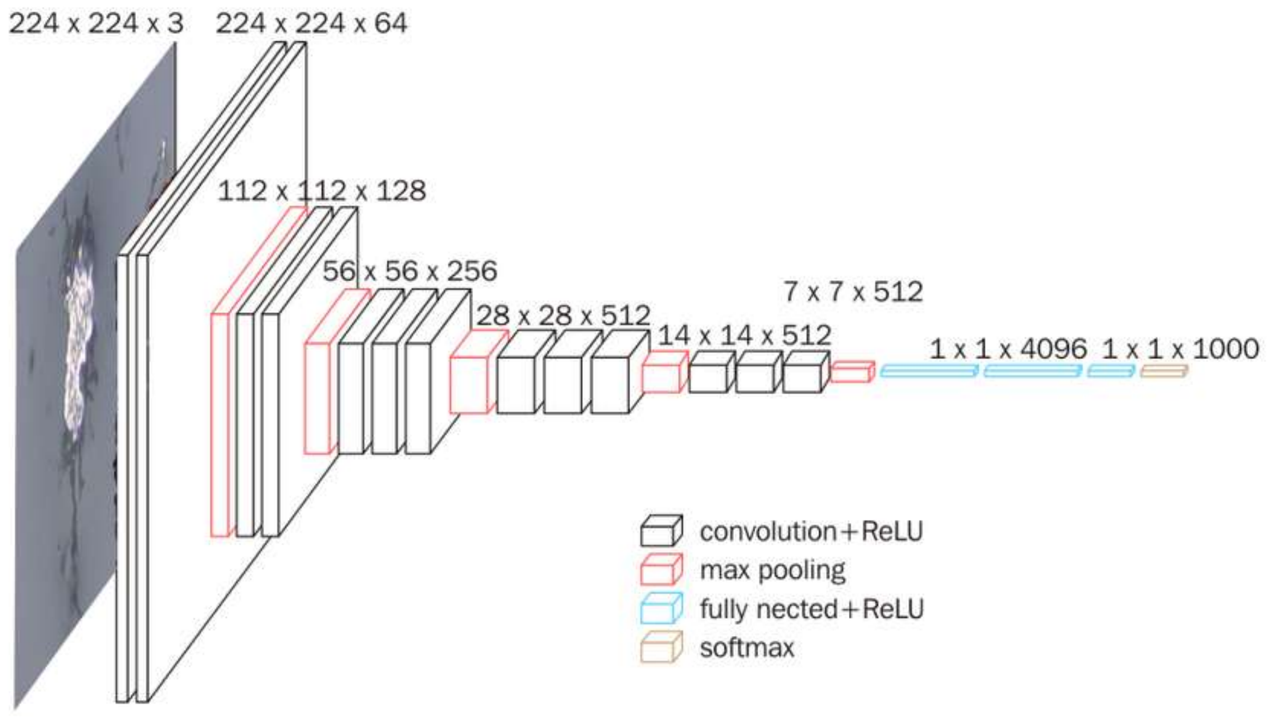


Figure 6.2.1

# 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/tra
ffic"
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

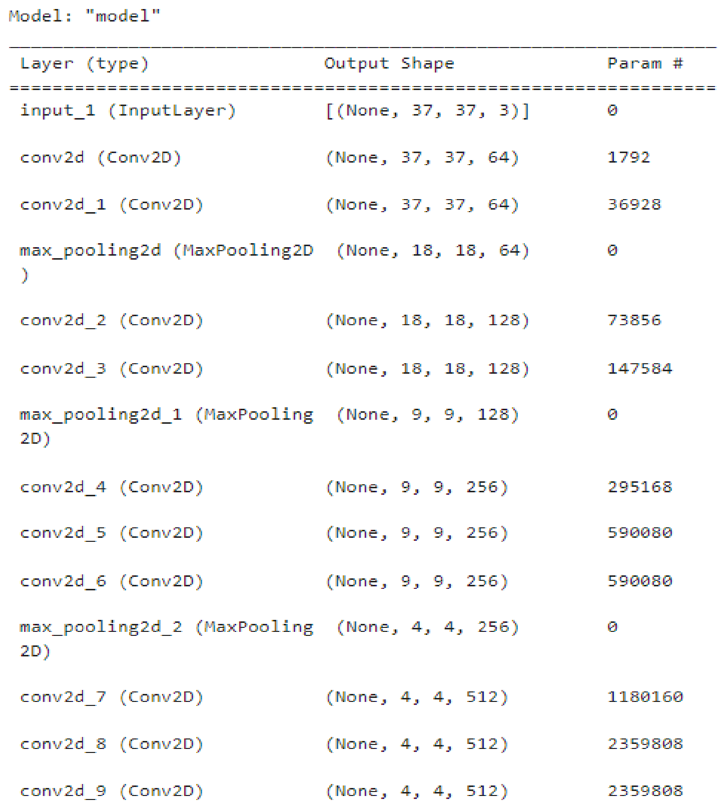
```
epochs=30
history = vgg16_model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 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_to
p=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']
```

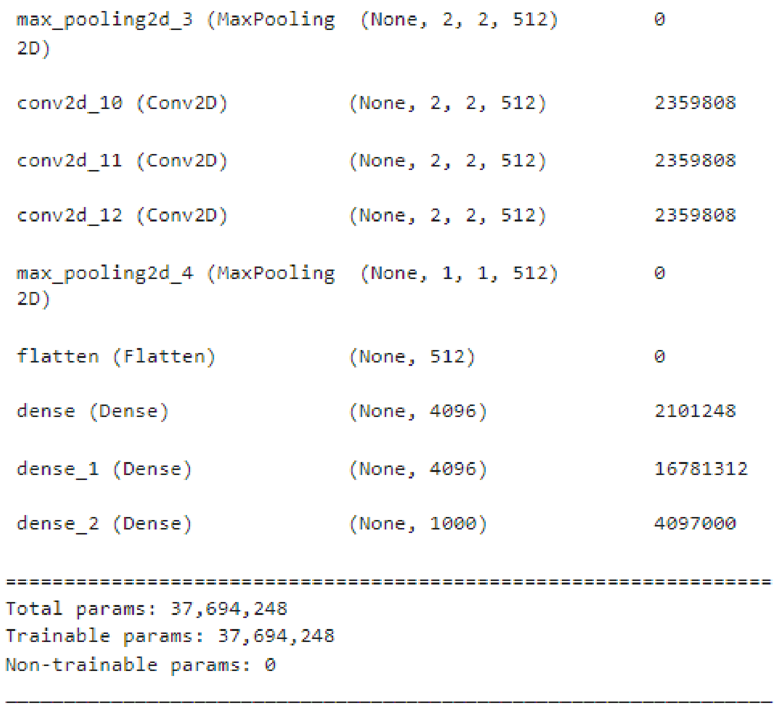
```
# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.8, color = ['green', 'green'])

# 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_Sig
ns_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\ttr
raaiinn\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



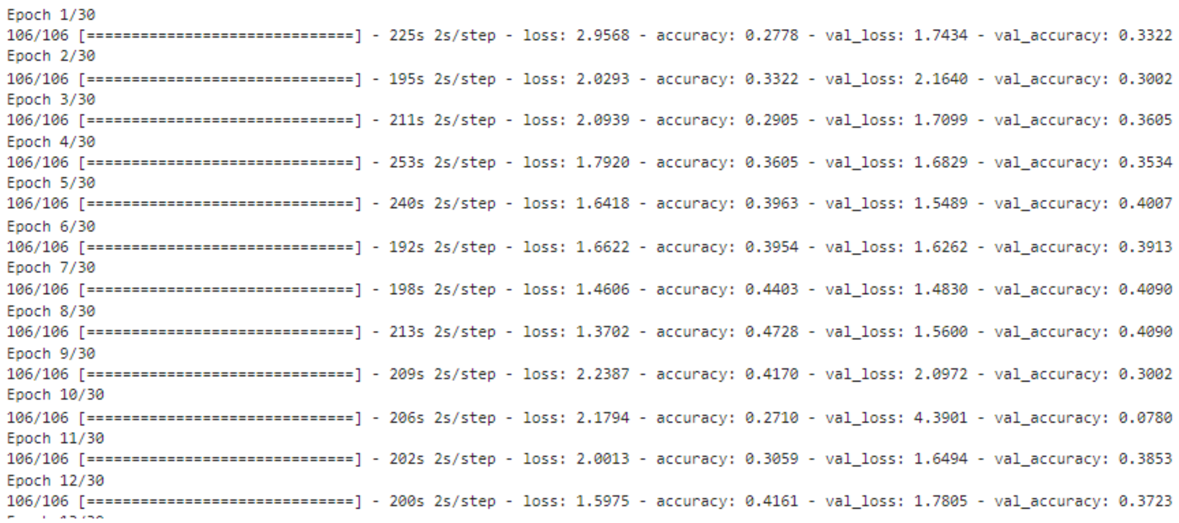
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 37, 37, 3)]       0

 conv2d (Conv2D)             (None, 37, 37, 64)        1792

 conv2d_1 (Conv2D)           (None, 37, 37, 64)        36928

 max_pooling2d (MaxPooling2D  (None, 18, 18, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 18, 18, 128)       73856

 conv2d_3 (Conv2D)           (None, 18, 18, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 9, 9, 128)        0
 2D)

 conv2d_4 (Conv2D)           (None, 9, 9, 256)         295168

 conv2d_5 (Conv2D)           (None, 9, 9, 256)         590080

 conv2d_6 (Conv2D)           (None, 9, 9, 256)         590080

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 conv2d_7 (Conv2D)           (None, 4, 4, 512)         1180160

 conv2d_8 (Conv2D)           (None, 4, 4, 512)         2359808

 conv2d_9 (Conv2D)           (None, 4, 4, 512)         2359808
```

Figure 7.2.1-Model Summary1



```
 max_pooling2d_3 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 conv2d_10 (Conv2D)          (None, 2, 2, 512)         2359808

 conv2d_11 (Conv2D)          (None, 2, 2, 512)         2359808

 conv2d_12 (Conv2D)          (None, 2, 2, 512)         2359808

 max_pooling2d_4 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 4096)              2101248

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 1000)              4097000

=================================================================
Total params: 37,694,248
Trainable params: 37,694,248
Non-trainable params: 0
_________________________________________________________________
```

Figure 7.2.2- Model Summary2



```
Epoch 1/30
106/106 [==============================] - 225s 2s/step - loss: 2.9568 - accuracy: 0.2778 - val_loss: 1.7434 - val_accuracy: 0.3322
Epoch 2/30
106/106 [==============================] - 195s 2s/step - loss: 2.0293 - accuracy: 0.3322 - val_loss: 2.1640 - val_accuracy: 0.3002
Epoch 3/30
106/106 [==============================] - 211s 2s/step - loss: 2.0939 - accuracy: 0.2905 - val_loss: 1.7099 - val_accuracy: 0.3605
Epoch 4/30
106/106 [==============================] - 253s 2s/step - loss: 1.7920 - accuracy: 0.3605 - val_loss: 1.6829 - val_accuracy: 0.3534
Epoch 5/30
106/106 [==============================] - 240s 2s/step - loss: 1.6418 - accuracy: 0.3963 - val_loss: 1.5489 - val_accuracy: 0.4007
Epoch 6/30
106/106 [==============================] - 192s 2s/step - loss: 1.6622 - accuracy: 0.3954 - val_loss: 1.6262 - val_accuracy: 0.3913
Epoch 7/30
106/106 [==============================] - 198s 2s/step - loss: 1.4606 - accuracy: 0.4403 - val_loss: 1.4830 - val_accuracy: 0.4090
Epoch 8/30
106/106 [==============================] - 213s 2s/step - loss: 1.3702 - accuracy: 0.4728 - val_loss: 1.5600 - val_accuracy: 0.4090
Epoch 9/30
106/106 [==============================] - 209s 2s/step - loss: 2.2387 - accuracy: 0.4170 - val_loss: 2.0972 - val_accuracy: 0.3002
Epoch 10/30
106/106 [==============================] - 206s 2s/step - loss: 2.1794 - accuracy: 0.2710 - val_loss: 4.3901 - val_accuracy: 0.0780
Epoch 11/30
106/106 [==============================] - 202s 2s/step - loss: 2.0013 - accuracy: 0.3059 - val_loss: 1.6494 - val_accuracy: 0.3853
Epoch 12/30
106/106 [==============================] - 200s 2s/step - loss: 1.5975 - accuracy: 0.4161 - val_loss: 1.7805 - val_accuracy: 0.3723
```

Figure 7.2.3-Resnet Model Train

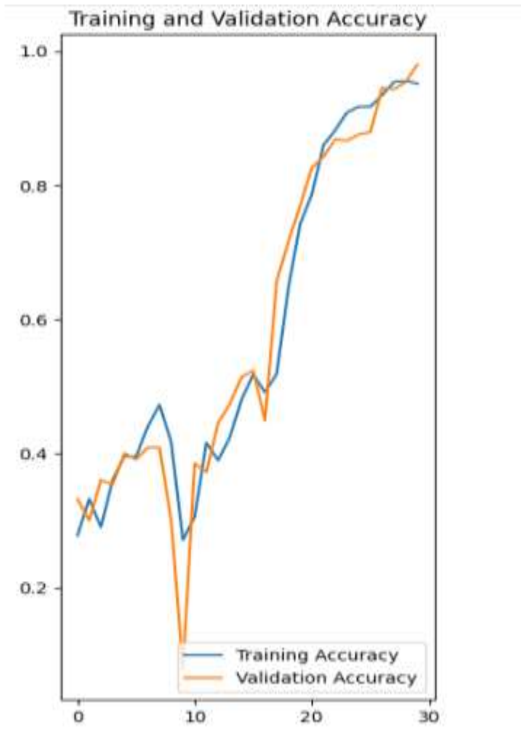


Figure 7.2.4-Accuracy

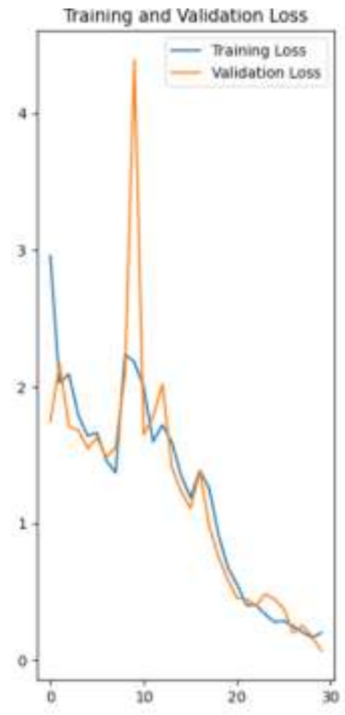
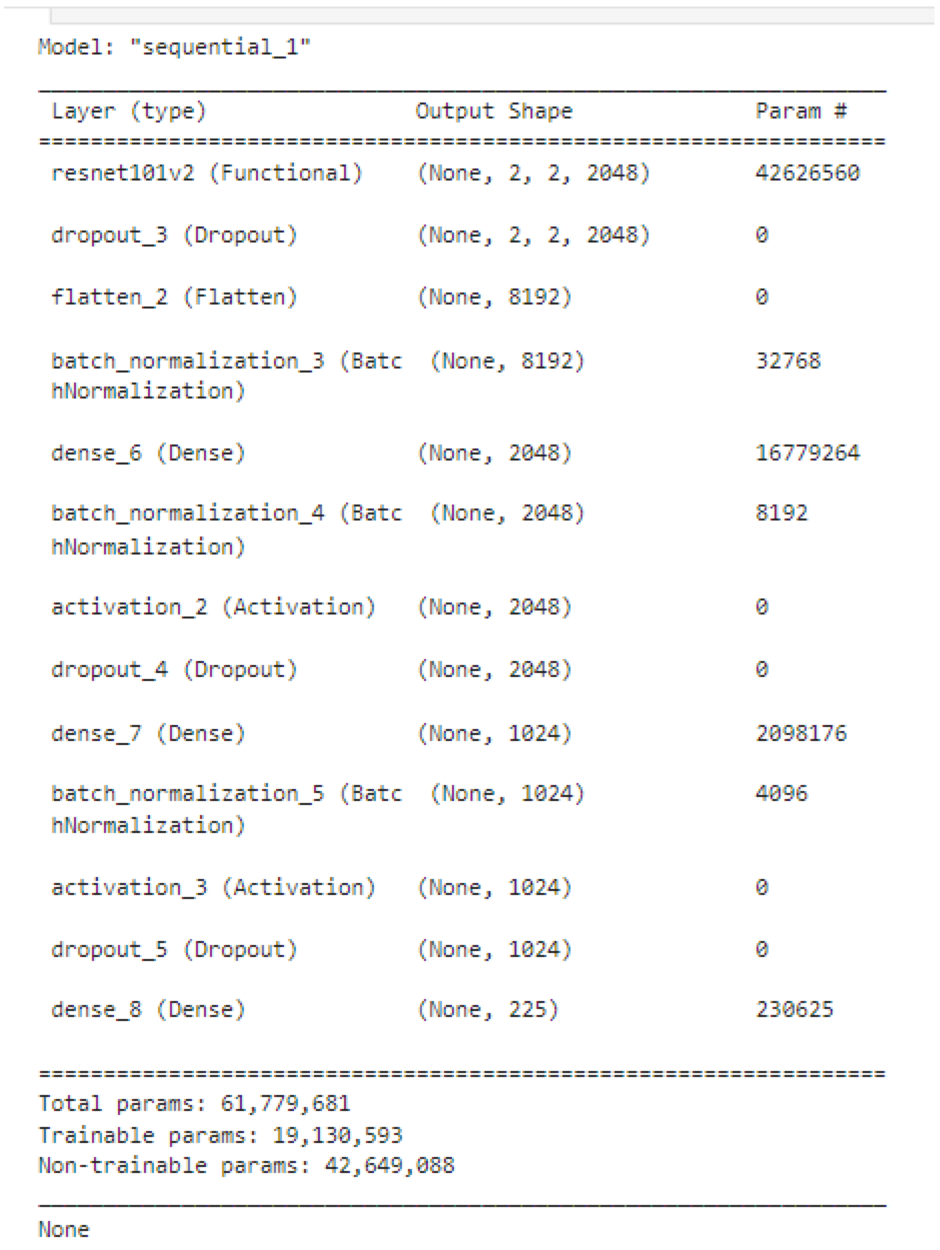
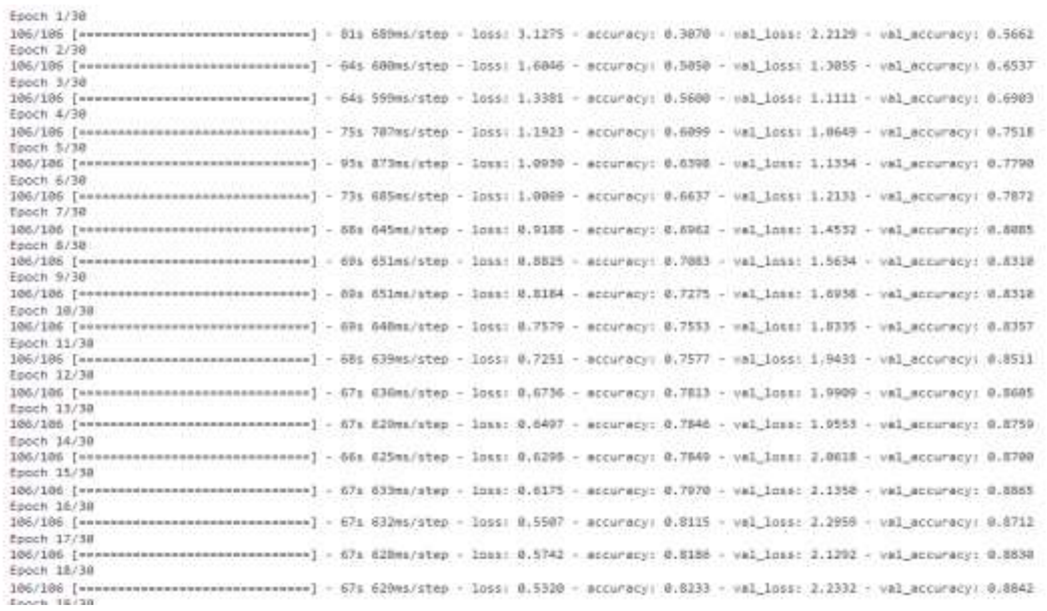


Figure 7.2.5-Loss



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet101v2 (Functional)    (None, 2, 2, 2048)        42626560

 dropout_3 (Dropout)         (None, 2, 2, 2048)        0

 flatten_2 (Flatten)         (None, 8192)              0

 batch_normalization_3 (Batc  (None, 8192)             32768
 hNormalization)

 dense_6 (Dense)             (None, 2048)              16779264

 batch_normalization_4 (Batc  (None, 2048)             8192
 hNormalization)

 activation_2 (Activation)   (None, 2048)              0

 dropout_4 (Dropout)         (None, 2048)              0

 dense_7 (Dense)             (None, 1024)              2098176

 batch_normalization_5 (Batc  (None, 1024)             4096
 hNormalization)

 activation_3 (Activation)   (None, 1024)              0

 dropout_5 (Dropout)         (None, 1024)              0

 dense_8 (Dense)             (None, 225)               230625

=================================================================
Total params: 61,779,681
Trainable params: 19,130,593
Non-trainable params: 42,649,088
_________________________________________________________________
None
```

Figure 7.2.6-Model Summary



```
Epoch 1/30
106/106 [==============================] - 81s 689ms/step - loss: 3.1275 - accuracy: 0.3070 - val_loss: 2.2129 - val_accuracy: 0.5662
Epoch 2/30
106/106 [==============================] - 64s 600ms/step - loss: 1.6046 - accuracy: 0.5050 - val_loss: 1.3055 - val_accuracy: 0.6537
Epoch 3/30
106/106 [==============================] - 64s 599ms/step - loss: 1.3381 - accuracy: 0.5600 - val_loss: 1.1111 - val_accuracy: 0.6903
Epoch 4/30
106/106 [==============================] - 75s 707ms/step - loss: 1.1923 - accuracy: 0.6099 - val_loss: 1.0649 - val_accuracy: 0.7518
Epoch 5/30
106/106 [==============================] - 93s 873ms/step - loss: 1.0930 - accuracy: 0.6398 - val_loss: 1.1334 - val_accuracy: 0.7790
Epoch 6/30
106/106 [==============================] - 73s 685ms/step - loss: 1.0009 - accuracy: 0.6637 - val_loss: 1.2131 - val_accuracy: 0.7872
Epoch 7/30
106/106 [==============================] - 68s 645ms/step - loss: 0.9188 - accuracy: 0.6962 - val_loss: 1.4532 - val_accuracy: 0.8085
Epoch 8/30
106/106 [==============================] - 69s 651ms/step - loss: 0.8825 - accuracy: 0.7083 - val_loss: 1.5634 - val_accuracy: 0.8310
Epoch 9/30
106/106 [==============================] - 69s 651ms/step - loss: 0.8184 - accuracy: 0.7275 - val_loss: 1.6936 - val_accuracy: 0.8310
Epoch 10/30
106/106 [==============================] - 69s 640ms/step - loss: 0.7579 - accuracy: 0.7553 - val_loss: 1.8335 - val_accuracy: 0.8357
Epoch 11/30
106/106 [==============================] - 68s 639ms/step - loss: 0.7251 - accuracy: 0.7577 - val_loss: 1.9431 - val_accuracy: 0.8511
Epoch 12/30
106/106 [==============================] - 67s 630ms/step - loss: 0.6736 - accuracy: 0.7813 - val_loss: 1.9909 - val_accuracy: 0.8605
Epoch 13/30
106/106 [==============================] - 67s 620ms/step - loss: 0.6497 - accuracy: 0.7846 - val_loss: 1.9553 - val_accuracy: 0.8759
Epoch 14/30
106/106 [==============================] - 66s 625ms/step - loss: 0.6298 - accuracy: 0.7849 - val_loss: 2.0618 - val_accuracy: 0.8700
Epoch 15/30
106/106 [==============================] - 67s 633ms/step - loss: 0.6175 - accuracy: 0.7970 - val_loss: 2.1350 - val_accuracy: 0.8865
Epoch 16/30
106/106 [==============================] - 67s 632ms/step - loss: 0.5507 - accuracy: 0.8115 - val_loss: 2.2958 - val_accuracy: 0.8712
Epoch 17/30
106/106 [==============================] - 67s 628ms/step - loss: 0.5742 - accuracy: 0.8186 - val_loss: 2.1292 - val_accuracy: 0.8830
Epoch 18/30
106/106 [==============================] - 67s 620ms/step - loss: 0.5320 - accuracy: 0.8233 - val_loss: 2.2332 - val_accuracy: 0.8842
```

Figure 7.2.7-VGG16 Model Train

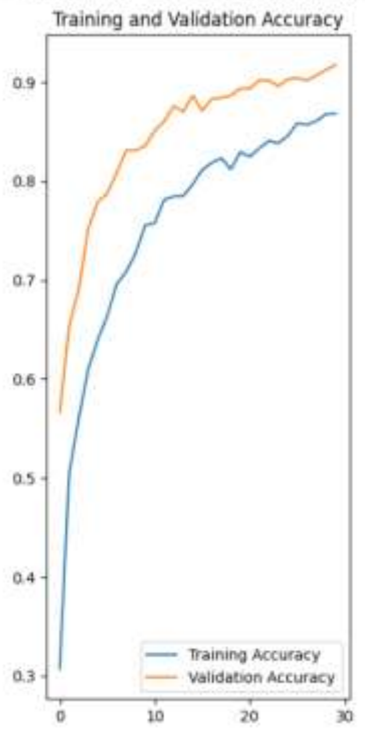


Figure 7.2.8-Accuracy

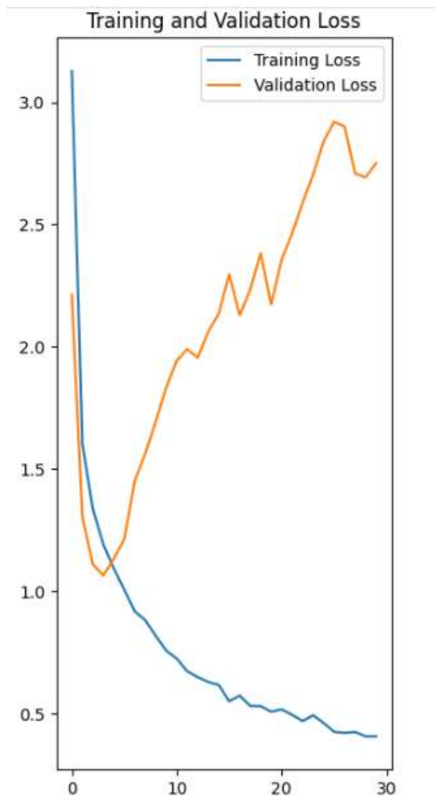


Figure 7.2.9-Loss

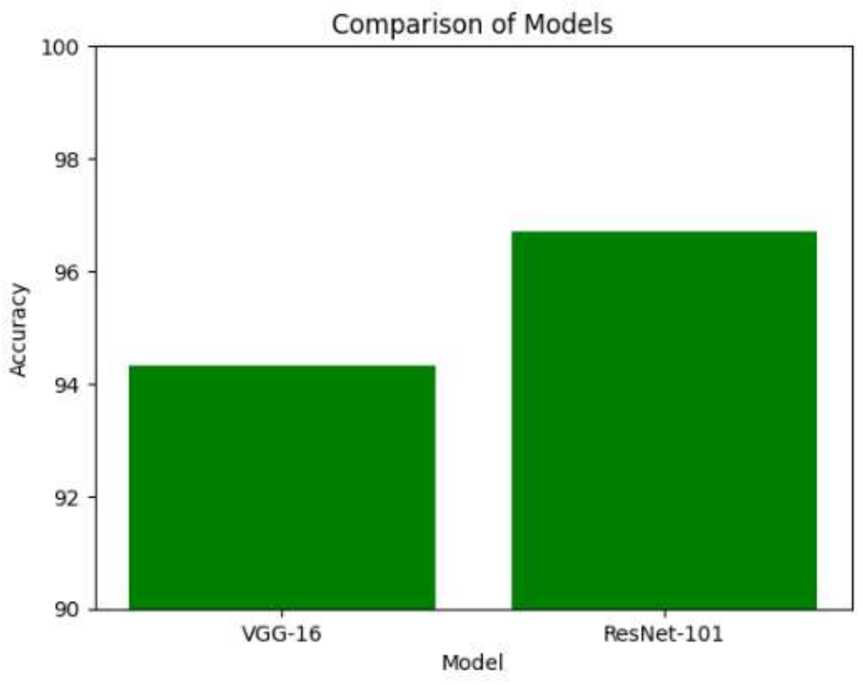
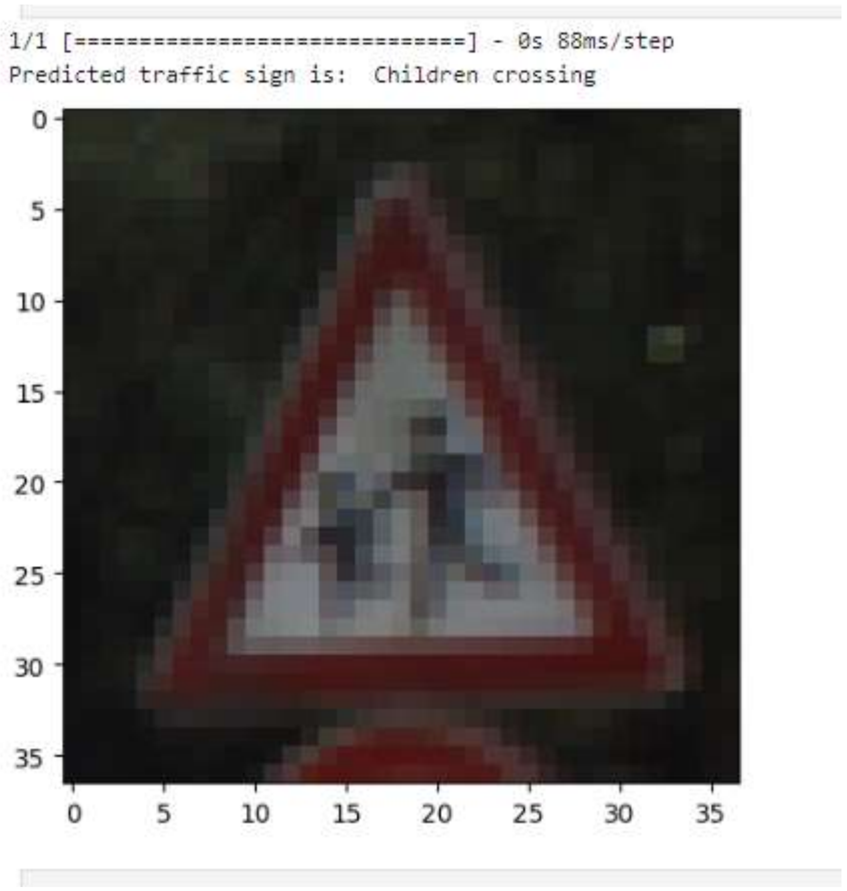


Figure 7.2.10-Comparison of Models



```
1/1 [==============================] - 0s 88ms/step
Predicted traffic sign is:  Children crossing
```

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://pandas.pydata.org

https://matplotlib.org