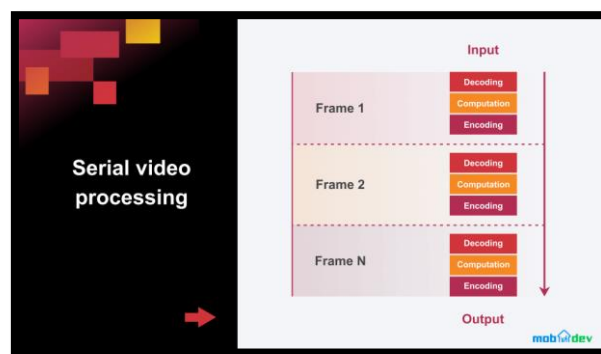


LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB



Project on
Traffic Sign Recognition System



SUBMITTED BY:

Name: Naman Thakur

Registration Number: 12017949

Name: Siddharth Mehrotra

Registration Number: 12006050

Name: Sameer Ahmed Khan

Registration Number: 12017912

Section: EE002

School: School of Electronics and Electrical Engineering

Course Code & Name: ECE399, Workshop on Video Processing

SUBMITTED TO:

Dr. Deepika Ghai

TABLE OF CONTENTS

| | |
|---|-----------|
| LIST OF FIGURES | 4 |
| Acknowledgement | 5 |
| 1. Abstract | 6 |
| 2. Introduction | 7 |
| 2.1 Image Processing- | 7 |
| 2.2 Aim-..... | 7 |
| 2.3 Libraries used- | 8 |
| 3. Motivation to choose the project | 10 |
| 4. Applications..... | 11 |
| 4.1 Driver Assistance- | 11 |
| 4.2 Pedestrian safety- | 11 |
| 4.3 Special needs assistance-..... | 11 |
| 4.4 Autonomous Cars- | 11 |
| 4.5 Road Safety- | 11 |
| 4.6 Traffic Management-..... | 11 |
| 4.7 Navigation-..... | 11 |
| 4.8 Transport Planning- | 12 |
| 4.9 Smart Cities-..... | 12 |
| 5. Related Work | 13 |
| 6. Proposed Method | 15 |
| 7. Pre-processing & Feature Extraction | 17 |
| 7.1 Pre-Processing- | 17 |
| 7.2 Feature Extraction- | 17 |

| | |
|--|----|
| 7.3 Key concepts used- | 18 |
| 7.4 Dataset Description- | 23 |
| 8. Source Code | 24 |
| 8.1 Pre-processing- | 24 |
| 8.2 Feature extraction, Training & Testing- | 25 |
| 9. Results | 31 |
| 10. Input and its prediction- | 43 |
| 11. Conclusion & Future Scope | 44 |
| 11.1 Conclusion- | 44 |
| 11.2Future Scope- | 45 |
| 12. References | 46 |
| 12.1 Image Processing Commands- | 46 |
| 12.2Research Papers- | 46 |

LIST OF FIGURES

| | |
|---|----|
| FIGURE 1-FLOWCHART FOR PROPOSED METHOD | 16 |
| FIGURE 2-CREATING A TEST FOLDER..... | 31 |
| FIGURE 3-CREATING DIFFERENT CLASSES INSIDE TEST FOLDER | 31 |
| FIGURE 4-CREATING A LIST OF ALL IMAGES | 32 |
| FIGURE 5-CREATING A LIST OF LISTS..... | 32 |
| FIGURE 6-COUNTING NO IMAGES IN DIFFERENT CLASSES | 33 |
| FIGURE 7-DELETING EXTRA IMAGES TO BALANCE CLASSES..... | 33 |
| FIGURE 8-ALL CLASSES BALANCED..... | 34 |
| FIGURE 9-DEFINING PARAMETERS..... | 34 |
| FIGURE 10-IMPORTING THE MODEL FROM TENSORFLOW, SETTING THE LAYERS | 35 |
| FIGURE 11-CHECKING THE MODEL LAYERS | 35 |
| FIGURE 12-PRINTING MODEL SUMMARY | 36 |
| FIGURE 13-SETTING TRAIN GENERATOR & TEST GENERATOR | 36 |
| FIGURE 14-CHECKING THE TEST & TRAIN GENERATOR | 37 |
| FIGURE 15-COMPILING & TRAINING THE MODEL | 37 |
| FIGURE 16-OUTPUT WITH VARIOUS PARAMETERS..... | 38 |
| FIGURE 17-GENERATING CLASS LABELS | 38 |
| FIGURE 18-GENERATING PREDICTION USING ACTIVITY LIST..... | 39 |
| FIGURE 19-CREATING THE CONFUSION MATRIX | 39 |
| FIGURE 20-MAKING CLASSIFICATION REPORT | 40 |
| FIGURE 21-SAVING THE DATA IN A CSV FILE..... | 40 |
| FIGURE 22-THE CONFUSION MATRIX | 41 |
| FIGURE 23-OUTPUT OF ONE PREDICTION (CORRECT PREDICTION) | 41 |
| FIGURE 24-CODE FOR READING THE IMAGE AND PRINTING IT..... | 42 |
| FIGURE 25-OUTPUT OF ANOTHER IMAGE (CORRECT CLASS PREDICTED)..... | 42 |



Acknowledgement

We would like to take this opportunity to express my sincere gratitude to all those who have supported and encouraged me throughout this project.

First and foremost, we are deeply indebted to my project guide, Dr. Deepika Ghai, for her constant guidance, invaluable insights, and unwavering support. Her expertise and feedback have been instrumental in shaping the outcome of this project.

We would also like to thank our friends and colleagues for their valuable contributions, constructive feedback, and moral support throughout the project. Their willingness to share their knowledge and experience has been greatly appreciated.

Last but not least, we are grateful to our family for their unconditional love and support, which has been a constant source of motivation throughout this journey. Their unwavering belief in me has been the driving force behind my success.

Thank you all for your contributions, encouragement, and support. I am truly grateful for your presence in my life.

1. Abstract

This project aims to develop a traffic sign recognition system using computer vision techniques. The system utilizes a convolutional neural network (CNN) to classify traffic signs based on their visual features. The dataset used for training and testing the CNN consists of labelled images of various traffic signs, including speed limits, stop signs, yield signs, and many others. Python libraries such as OpenCV, TensorFlow, and Keras are used to pre-process the images, train and test the CNN, and evaluate its performance. The system achieves high accuracy in recognizing traffic signs, which can contribute to improving road safety and developing autonomous driving systems.

This project focuses on building a traffic sign recognition system using an efficient neural network called EfficientNetB3, which has been trained on millions of images by Google. The dataset used for this project consists of 43 different traffic sign classes, including speed limit, turning right, and U-turn.

The training set comprises 160 images for each class, while the test set contains 20 images for each class. Python concepts such as listdir, mkdir, directory, and dictionaries are utilized in the implementation of this project. The system is designed to recognize traffic signs accurately and efficiently, contributing to the development of intelligent transportation systems.

2. Introduction

Traffic sign recognition systems are a critical component of intelligent transportation systems, contributing to the development of safer and more efficient roads. This project focuses on developing a traffic sign recognition system using deep learning techniques, specifically the EfficientNetB3 neural network, which has been created and tested by Google on millions of images. The dataset used for this project contains 43 different traffic sign classes, including speed limit, turning right, and U-turn. The training set comprises 160 images for each class, while the test set contains 20 images for each class. Python concepts such as listdir, mkdir, directory, and dictionaries are used in the implementation of the project.

The project aims to achieve high accuracy in recognizing traffic signs, making it a valuable contribution to the field of image processing and transportation systems. Image processing is a rapidly growing field that has been revolutionized by advancements in deep learning, particularly convolutional neural networks (CNNs). CNNs have proven to be highly effective in solving image classification problems, including traffic sign recognition, face recognition, and object detection, to name a few. This project focuses on developing a traffic sign recognition system using CNNs, which is a critical component of intelligent transportation systems. The project uses a large labeled dataset of traffic sign images to train and test the CNN, which is implemented using Python libraries such as TensorFlow and Keras. The project aims to achieve high accuracy in recognizing traffic signs, contributing to the development of safer and more efficient transportation systems.

2.1 Image Processing-

Image processing refers to the analysis and manipulation of digital images to extract useful information or enhance their visual appearance. It involves a range of techniques and algorithms that can be applied to images, such as filtering, compression, restoration, segmentation, recognition, and more. Image processing can be used in a variety of fields, including medical imaging, security systems, remote sensing, digital art, and many others. The goal of image processing is to extract meaningful insights from images or improve their quality for better human perception or automated analysis.

2.2 Aim-

The aim of a traffic sign recognition system is to automatically detect, classify and interpret the traffic signs present in the images or videos captured by cameras installed on roads, highways, and

vehicles. The system is designed to aid drivers and autonomous vehicles in understanding the road environment and making informed decisions based on the information provided by the traffic signs.

The system can help reduce the incidence of accidents and improve road safety by alerting drivers to speed limits, stop signs, construction zones, and other road hazards. The ultimate goal of a traffic sign recognition system is to contribute to the development of more efficient and safe transportation systems.

2.3 Libraries used-

- **OS:** The os library is a Python module that provides a way to interact with the operating system. It includes functions for accessing files and directories, manipulating file paths, and executing system commands. With the os library, you can create, delete, move, and rename files and directories, change the current working directory, get information about file permissions, and more.
- **Shutil:** The shutil library is a Python module that provides high-level operations on files and collections of files, including copying, moving, and deleting files and directories. The shutil module also includes functions for archiving and compressing files and directories, making it a useful tool for managing large data sets.
- **Random:** The random library is a Python module that provides functions for generating random numbers, sequences, and selections. The random module includes functions for generating random integers, floating-point numbers, and boolean values, as well as for shuffling sequences and selecting random elements from them. The random module is often used in data analysis, simulation, and game development.
- **Tensorflow:** TensorFlow is an open-source Python library for building and training machine learning models. It provides a set of tools for creating and manipulating neural networks, including functions for defining and optimizing model parameters, managing data sets, and visualizing model performance. TensorFlow is widely used for a range of machine learning tasks, including image classification, natural language processing, and computer vision.
- **Matplotlib:** The matplotlib library is a Python module that provides a range of functions for creating visualizations, including charts, graphs, and plots. The matplotlib library can

be used to create line charts, scatter plots, histograms, bar charts, and many other types of visualizations. It is often used in data analysis, scientific research, and data visualization.

- **Glob:** The glob library is a Python module that provides functions for searching for files and directories using wildcards and regular expressions. The glob module includes functions for searching for files based on their names, extensions, and other attributes, making it a useful tool for managing large data sets. It is often used in data analysis and machine learning tasks for managing and processing large numbers of files.

3. Motivation to choose the project

The motivation behind choosing a project on traffic sign recognition system is to address the issue of road safety and traffic management. The traffic sign recognition system can help drivers and autonomous vehicles to understand the road environment better and make informed decisions based on the information provided by the traffic signs. Accidents due to human error and lack of attention can be reduced by alerting drivers to speed limits, stop signs, construction zones, and other road hazards. With the increasing number of vehicles on the road, it becomes essential to manage traffic efficiently and safely. The project aims to contribute to the development of safer and more efficient transportation systems. The use of neural networks and deep learning techniques can help achieve high accuracy in recognizing traffic signs, making the project valuable to the field of image processing and transportation systems.



4. Applications

4.1 Driver Assistance-

The system can provide real-time assistance to drivers, reminding them of upcoming road signs, reducing their cognitive load and enabling them to focus on driving.

4.2 Pedestrian safety-

The system can detect pedestrian crossing signs, school zones, and other pedestrian-related signs, making it easier for drivers to be aware of pedestrians and reduce the risk of accidents.

4.3 Special needs assistance-

The system can assist drivers with special needs, such as those who are visually impaired or have difficulty processing visual information. By alerting them to traffic signs, the system can improve their driving experience and increase their independence.

4.4 Autonomous Cars-

Traffic sign recognition can help self-driving cars and autonomous vehicles to navigate the road and make informed decisions based on the information provided by the traffic signs.

4.5 Road Safety-

The system can alert drivers to speed limits, stop signs, construction zones, and other road hazards, reducing the incidence of accidents and improving road safety.

4.6 Traffic Management-

The system can be used to monitor and manage traffic flow, identify congested areas, and adjust traffic signals accordingly.

4.7 Navigation-

The system can help drivers navigate through unfamiliar roads and intersections, making it a valuable tool for navigation and wayfinding.

4.8 Transport Planning-

The system can provide valuable data on traffic patterns and road conditions, helping transportation planners to design more efficient and safer transportation systems.

4.9 Smart Cities-

Traffic sign recognition can be integrated into smart city systems, enabling better management of transportation networks and improving the quality of life for city residents.



5. Related Work

| S No. | Author | Technique Used | Dataset Used | Evaluation Parameters | Pros | Cons |
|-------|--------------------|---|---|-----------------------|---|--|
| 1. | Sun et al. [1] | Hough Transform | Chinese Traffic Sign Dataset | 98.2% Accuracy | Noise and curve discontinuity have relatively small influence. | The biggest disadvantage of mean filtering. |
| 2. | Franzen et al. [2] | Traffic sign recognition does not take place in the spatial domain but in the frequency domain. | German Traffic sign Recognition Benchmark (GTSRB) | 99.6% Accuracy | Simplifying the neural architecture, as the number of layers can be significantly reduced. | Some images may be misinterpreted in the frequency domain. |
| 3. | Manzari et al. [3] | Hinge loss stochastic gradient descent technique. | European Traffic Sign Dataset | 99.51% Accuracy | The networks have good resistance to adversarial attacks, which is interesting as future works. | One of the factors that reduce network accuracy and image degradation is rotation. |

| | | | | | | |
|----|-------------------|---|-------------------|--------------|--|--|
| 4. | Razdak et al. [4] | Proposed method extracted the detected sign in black and white pixels and further classified into groups. | Template dataset. | 94% Accuracy | The dataset contains images with several angles which makes it unique from others. | Presence of other objects in the scene such as moving cars, bicycles, pedestrians, shop signs also offer obstruction in detection. |
|----|-------------------|---|-------------------|--------------|--|--|



6. Proposed Method

The proposed method for the traffic sign recognition system project involves using the EfficientNetB3 neural network, which has been pre-trained on a large dataset of images by Google. The pre-trained model is then fine-tuned on the traffic sign dataset, consisting of 43 classes and a total of 160 training images and 20 test images for each class.

- **Loading and Preprocessing Images:** The input images are first loaded into the program and preprocessed using techniques such as resizing, normalization, and data augmentation to increase the dataset's diversity.
- **Fine-tuning the Pre-trained Model:** The pre-trained EfficientNetB3 model is used as the base model and fine-tuned on the traffic sign dataset using transfer learning. This involves removing the final layer of the model and replacing it with a new output layer that has the same number of neurons as the number of classes in the dataset.
- **Training the Model:** The fine-tuned model is then trained on the preprocessed images using backpropagation and gradient descent algorithms to optimize the model's parameters.
- **Testing the Model:** The trained model is then evaluated on the test dataset to measure its performance in accurately classifying new images.
- **Prediction:** Once the model is trained, it can be used to predict the class of new traffic sign images in real-time.

Python libraries such as os, shutil, random, matplotlib, glob, listdir, mkdir, directory, and dictionaries are used to manage the dataset, preprocess images, and train the model.

Overall, the proposed method uses state-of-the-art deep learning techniques and a pre-trained neural network to achieve high accuracy in traffic sign recognition, making it a reliable solution for real-world applications such as autonomous vehicles, driver assistance systems, and traffic management.

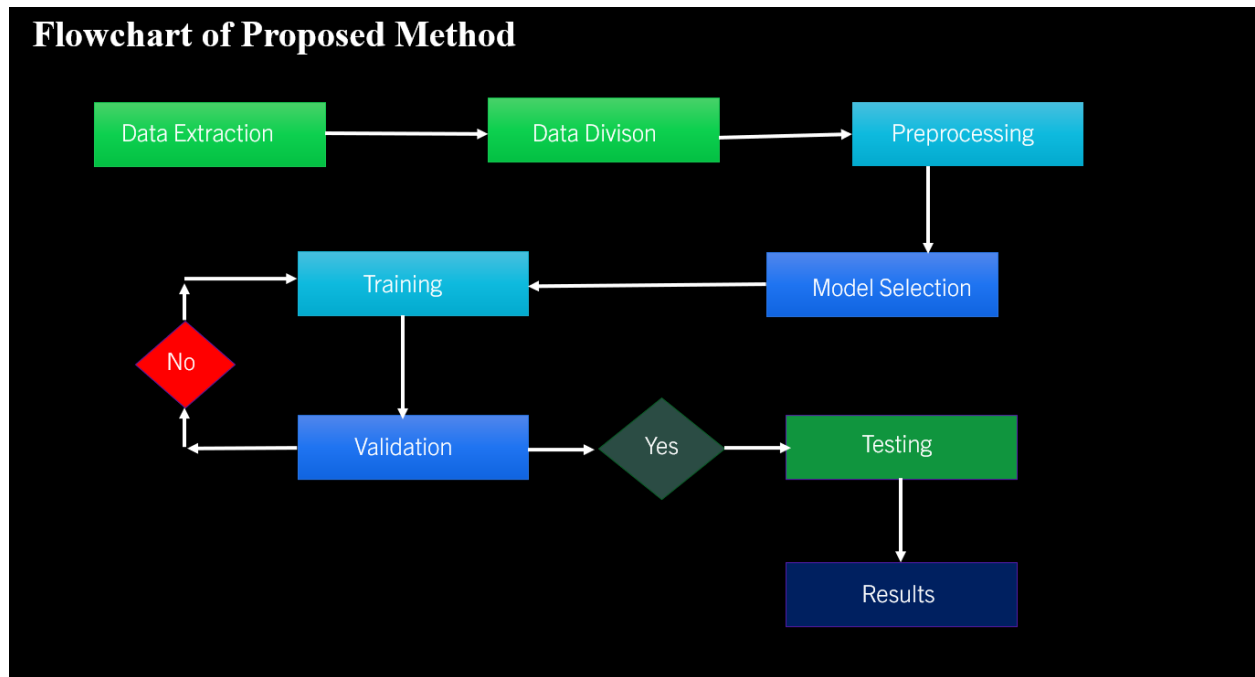


Figure 1-Flowchart for proposed method

7. Pre-processing & Feature Extraction

7.1 Pre-Processing-

Pre-processing is a critical step in image processing that involves manipulating and transforming images before they are fed into the machine learning algorithm for training or testing. Here are the typical steps involved in preprocessing an image for the traffic sign recognition system project:

- **Load the image:** The first step is to load the image into the program using the appropriate Python library such as OpenCV or Pillow.
- **Resize the image:** The image is often resized to a standardized resolution, such as 32x32 or 64x64 pixels, to ensure uniformity across the dataset.
- **Convert to grayscale:** The image may be converted to grayscale to reduce the amount of data that needs to be processed and to simplify the image's features.
- **Apply image normalization:** The pixel values of the image are scaled to a range of 0 to 1, allowing the machine learning algorithm to process the images more efficiently.
- **Apply image augmentation:** Image augmentation techniques such as rotation, scaling, flipping, and adding noise are applied to the images to increase the dataset's diversity and reduce overfitting.
- **Label the image:** The image is assigned a label corresponding to its class, such as "speed limit" or "stop sign."
- **Save the pre-processed image:** The preprocessed image is saved in a format that can be used for training or testing the machine learning model.

Overall, the goal of pre-processing is to standardize the images and extract meaningful features that can be used by the machine learning algorithm to classify new images accurately.

7.2 Feature Extraction-

Feature extraction is a crucial step in an image processing project that involves extracting relevant features from images to create a representation that can be used for training and testing a machine learning model such as a neural network. In the case of the project that uses the EfficientNetB3 neural network, the following feature extraction steps can be performed:

- **Load the pre-processed data:** The first step is to load the pre-processed image data into the program. The program uses the Python `listdir()` function to get the list of filenames in the training and testing directories containing the pre-processed images.
- **Extract features using the EfficientNetB3 model:** The EfficientNetB3 model is a pre-trained neural network that has been trained on millions of images. The program uses the model to extract relevant features from the preprocessed images. The features are extracted by passing the images through the model's layers and obtaining the output of a specific layer. The output of this layer is considered the features of the image.
- **Flatten the features:** The features extracted by the EfficientNetB3 model are in the form of a 4D tensor. The program flattens this tensor into a 2D matrix to prepare it for training and testing.
- **Split the data into training and testing sets:** The program splits the flattened features and one-hot encoded labels into separate training and testing sets. The training set contains 160 images per class, and the testing set contains 20 images per class.
- **Save the features and labels:** The program saves the flattened features and one-hot encoded labels for the training and testing sets in separate files. This step ensures that the data is ready for training and testing the neural network.

In summary, the feature extraction steps for the image processing project that uses the EfficientNetB3 neural network involves loading the pre-processed image data, extracting relevant features using the EfficientNetB3 model, flattening the features, splitting the data into training and testing sets, and saving the features and labels. These steps ensure that the data is in a suitable format for training and testing the neural network. The project uses various Python concepts such as `listdir()`, `makedirs()`, directory handling, and dictionaries to implement these steps.

7.3 Key concepts used-

- **ReLU-** ReLU stands for Rectified Linear Unit and is a type of activation function used in artificial neural networks, including those used for image processing.

In image processing, ReLU is applied to the output of a convolutional layer, which is used to extract features from an image. The ReLU function takes an input value and returns either the input value (if it is positive) or zero (if it is negative). This has the effect of introducing non-linearity into the network and allows it to better model complex relationships between image features.

In simpler terms, ReLU is a mathematical operation that helps the neural network better detect and extract important features in an image by selectively filtering out unimportant ones. This activation function has been found to be effective in many deep learning models, including those used in computer vision tasks like object detection and image classification.

- **Adam**-Adam (Adaptive Moment Estimation) is an optimization algorithm that is widely used in deep learning, including image processing. It is a type of stochastic gradient descent (SGD) optimization algorithm that adapts the learning rate for each parameter based on estimates of the first and second moments of the gradients.

In image processing, Adam is used to optimize the weights and biases of the neural network during training, by adjusting the learning rate of each parameter based on the magnitude and direction of the gradient. This allows the algorithm to converge faster and more accurately to the optimal values for the parameters, improving the overall performance of the network.

Adam is particularly effective in deep learning models with large and complex datasets, where traditional SGD algorithms may struggle to converge to the optimal solution. It has been found to be particularly useful in computer vision tasks such as image classification and object detection, where high accuracy and fast training times are critical.

- **Epochs**- In machine learning, an epoch is a complete pass over a training dataset by a learning algorithm during the training process. In other words, it refers to the number of times the learning algorithm has iterated through the entire dataset during training.

During each epoch, the learning algorithm makes predictions on the training data, computes the error (or loss) between the predicted output and the true output, and updates the model parameters accordingly. The goal of training is to reduce the error over time by adjusting the model parameters.

The number of epochs required to train a model depends on the complexity of the problem and the size of the dataset. If the model is not complex enough or the dataset is too small, training for too many epochs may result in overfitting, where the model memorizes the training data and performs poorly on new, unseen data. On the other hand, if the model is too complex or the dataset is large, training for too few epochs may result in underfitting, where the model fails to capture the underlying patterns in the data. Therefore, the number of epochs should be chosen carefully based on the characteristics of the problem and the data.

- **Batch Size-**In image processing and other machine learning tasks, the batch size refers to the number of training examples (images or samples) used in a single iteration of the learning algorithm.

During training, the dataset is typically divided into small batches, and the model parameters are updated based on the average loss over the examples in each batch. The batch size is a hyperparameter that can be adjusted during training, and it can have a significant impact on the performance of the model.

A larger batch size means that the learning algorithm will process more examples in each iteration, which can result in faster training times and improved generalization performance. However, larger batch sizes also require more memory and computing resources, and they can make it more difficult to escape local optima in the optimization process.

Conversely, a smaller batch size means that the learning algorithm will process fewer examples in each iteration, which can make the training process slower but can also lead to better generalization performance and help the algorithm converge to a better solution.

Choosing an appropriate batch size is an important hyperparameter tuning step in the training process, and it depends on factors such as the size of the dataset, the complexity of the model, and the available computational resources.

- **Accuracy-** In image processing, accuracy can also be calculated using the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) values. These values are typically obtained from a confusion matrix, which is a table that summarizes the performance of a classification model on a dataset.

The formula for accuracy using TP, TN, FP, and FN is:

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

TP (True Positive) represents the number of true positive predictions (i.e., the number of correctly classified positive examples), in our case if it's a traffic sign and identified as a traffic sign.

TN (True negative) represents the number of true negative predictions (i.e., the number of correctly classified negative examples), in our case if it's not a traffic sign and identified as not a traffic sign.

FP (False Positive) represents the number of false positive predictions (i.e., the number of negative examples that were incorrectly classified as positive), in our case if it's a traffic sign and identified as not a traffic sign.

FN (False Negative) represents the number of false negative predictions (i.e., the number of positive examples that were incorrectly classified as negative), in our case if its not a traffic sign and identified as a traffic sign.

- **Confusion Matrix-** A confusion matrix is a table that summarizes the performance of a machine learning model on a classification problem. It is a matrix of actual versus predicted class labels, and it is used to evaluate the accuracy of the model's predictions.

A confusion matrix is typically used in binary classification problems, where there are two possible class labels (positive and negative).

The matrix has four possible outcomes:

True Positive (TP): The model correctly predicted a positive example as positive.

False Positive (FP): The model incorrectly predicted a negative example as positive.

True Negative (TN): The model correctly predicted a negative example as negative.

False Negative (FN): The model incorrectly predicted a positive example as negative.

The confusion matrix provides a more detailed view of the model's performance than just the overall accuracy. It can be used to calculate various metrics such as precision, recall, F1-score, and accuracy, which can help to assess the strengths and weaknesses of the model and guide further improvements.

- **Batch Normalisation-** Batch normalization is a technique used in deep neural networks to improve the training process and the performance of the model. It involves normalizing the input data to each layer of the network to have zero mean and unit variance, which helps to stabilize the training process and reduce the impact of changes in the distribution of the input data.

During the training process, the distribution of the input data to each layer can shift or change, which can cause the gradients to become too large or too small, leading to slow convergence or vanishing/exploding gradients. By normalizing the input data to each layer, batch normalization reduces the impact of these changes and helps to speed up the training process.

- **Softmax Function-** Softmax is a mathematical function used in machine learning for multi-class classification tasks. It converts a vector of real numbers into a vector of probabilities that sum up to 1.0, allowing us to interpret the output of a model as a probability distribution over a set of mutually exclusive classes.

The softmax function takes a vector of real numbers $z = [z_1, z_2, \dots, z_n]$ as input and returns a vector of probabilities $p = [p_1, p_2, \dots, p_n]$. The softmax function ensures that the output probabilities are non-negative and sum up to 1.0, making them interpretable as the probability of each class given the input vector. The class with the highest probability is typically chosen as the predicted class for the input.

- **Global Average Pooling 2D-** Global Average Pooling 2D (GAP2D) is a pooling operation in image processing used to reduce the spatial dimensions of the feature maps produced by a convolutional neural network (CNN). Unlike other pooling operations such as max pooling or average pooling, GAP2D computes the average of all feature map values for each channel and returns a single value per channel.

GAP2D operates on a feature map with dimensions (h, w, c) , where h and w are the height and width of the feature map, respectively, and c is the number of channels. The operation involves taking the average of all the values in each channel, resulting in a new feature map with dimensions $(1, 1, c)$. This process effectively compresses the spatial information into a single value per channel.

- **Dense-** In image processing, a dense layer (also known as a fully connected layer) is a neural network layer where every input unit is connected to every output unit. In other words, the dense layer is a linear transformation that applies a matrix multiplication to its input followed by an element-wise activation function.

The output of the dense layer can be fed into a softmax activation function to produce a probability distribution over the classes, or it can be fed into a sigmoid activation function to produce binary classification outputs.

- **Depth wise convolutional Layer-** A depth wise convolutional layer is a type of convolutional layer used in neural networks for image processing. It is designed to reduce

the number of parameters in the model while maintaining the ability to extract useful features from the input image.

In a depth wise convolutional layer, the convolution operation is performed on each input channel separately, rather than across all channels simultaneously. This means that each channel has its own set of filters, which are applied to that channel only. The output of the depth wise convolutional layer is then concatenated across all channels to produce the final output.

Compared to traditional convolutional layers, which use a separate set of filters for each output channel, the depth wise convolutional layer reduces the number of parameters in the model by a factor equal to the number of output channels. This makes it a useful tool for reducing the computational cost and memory requirements of deep neural networks.

- **Dropout-** Dropout is a regularization technique used in deep neural networks to prevent overfitting. It involves randomly dropping out (i.e., setting to zero) a certain proportion of the neurons in a layer during training, which forces the network to learn more robust and generalizable features.

During training, dropout is applied independently to each neuron in a layer with a certain probability p , which is typically set to 0.5. This means that each neuron has a 50% chance of being dropped out during each training iteration. The dropout process is only applied during training and is turned off during inference.

Dropout can also be used in conjunction with other regularization techniques such as weight decay and early stopping to further improve the performance of the network.

7.4 Dataset Description-

Dataset link- <https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>

Dataset name- GTSRB - German Traffic Sign Recognition Benchmark

Number of classes- 43

Total images- 34560

Training dataset images- 6880

Validation dataset images- 1376

Testing dataset images- 860



8. Source Code

8.1 Pre-processing-

```
import os
import tqdm
import glob
import random
import shutil

path = 'C:/Users/saini/Desktop/new project/myData/test'
path1 = 'C:/Users/saini/Desktop/new project/myData/train'

try:
    os.mkdir(path)
    print("Folder { } created...".format(path))
except:
    print("A folder { } already exists...".format(path))

for i in range(43):
    try:
        os.mkdir(path+'/' +str(i))
        print("Folder { } created...".format(path+'/' +str(i)))
    except:
        print("A folder { } already exists...".format(path+'/' +str(i)))

l = {}
for i in range(43):
    l[i] = os.listdir(path1+'/' +str(i))

l1 = [i for i in l.values()]

for i in range(43):
    for file_name in random.sample(l1[i], 20):
        dest=path+'/' +str(i)
```



```

        shutil.move(os.path.join(path1+'/'+str(i), file_name), dest)

l = {}
for i in range(43):
    l[i] = os.listdir(path1+'/'+str(i))
l2 = [i for i in l.values()]
l3=[]
for i in range(43):
    l3.append(len(l2[i]))
l3

```

8.2 Feature extraction, Training & Testing-

```

import numpy as np
import pandas as pd
import os
import glob
import time
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score,
confusion_matrix

test = 'C:/Users/saini/Desktop/new project/myData/test'
train = 'C:/Users/saini/Desktop/new project/myData/train'

no_of_frames = 160                                # Number of Frames
epochs = 3                                          # Number of epochs
batch_size = 10                                    # Batch Size
n_classes = 43                                    # Patience for EarlyStopping

```

```

stime = int(time.time())                                # Defining Starting Time
categories = os.listdir(train)
print(categories)

from tensorflow.keras.applications import EfficientNetB3
from tensorflow.keras.applications.efficientnet_v2 import preprocess_input

base_model = EfficientNetB3(weights = 'imagenet', include_top = False, input_shape = (224, 224, 3))

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation = 'relu')(x)
x = Dropout(0.5)(x)
# x = Dense(512, activation = 'relu')(x)
# x = Dense(256, activation = 'relu')(x)
preds = Dense(n_classes, activation = 'softmax')(x)
model = Model(inputs = base_model.input, outputs = preds)

for i, layer in enumerate(model.layers):
    print(i, layer.name)
print(model.summary())

# Setting each layer as trainable
for layer in model.layers:
    layer.trainable = True

# Defining Image Data Generators
train_datagenerator = ImageDataGenerator(preprocessing_function = preprocess_input,
                                          validation_split = 0.2)

test_datagenerator = ImageDataGenerator(preprocessing_function = preprocess_input)

train_generator = train_datagenerator.flow_from_directory(train,
                                                          target_size = (224, 224),
                                                          color_mode = 'rgb',
                                                          batch_size = batch_size,

```

```

        class_mode = 'categorical',
        shuffle = True)

validation_generator = train_datagenerator.flow_from_directory(train,

        target_size = (224, 224),
        color_mode = 'rgb',
        batch_size = batch_size,
        class_mode = 'categorical',
        subset = 'validation')

test_generator = test_datagenerator.flow_from_directory(test,

        target_size = (224, 224),
        color_mode = 'rgb',
        class_mode = 'categorical')

print(train_generator.class_indices)
print(validation_generator.class_indices)
print(test_generator.class_indices)

# Compiling the Model

model.compile(optimizer = "Adam",

        loss = "categorical_crossentropy",
        metrics = ["accuracy"])

# Training the Model

history = model.fit(train_generator,

        validation_data = validation_generator,
        epochs = epochs,

        )

# Plotting the Graph

model_history = pd.DataFrame(history.history)

model_history.plot()

history2 = model.evaluate(test_generator)

```

```

# Image Data Generator
test_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)

test_generator = test_datagen.flow_from_directory(test,
                                                  target_size = (224, 224),
                                                  color_mode = "rgb",
                                                  shuffle = False,
                                                  class_mode = 'categorical',
                                                  batch_size = 1)

activities = test_generator.class_indices
print(activities)

def get_activity(val):
    for key, value in activities.items():
        if val == value:
            return key
    return "Invalid"

filenames = test_generator.filenames
nb_samples = len(filenames)

predict = model.predict(test_generator, steps = nb_samples, verbose = 1)

y_pred = []
for val in predict:
    y_pred.append(get_activity(np.argmax(val)))

y_true = []
for file in filenames:
    y_true.append(file.split("\\")[0])

cm = confusion_matrix(y_true, y_pred)

print(precision_score(y_true, y_pred, average = 'macro', zero_division=1))
print(recall_score(y_true, y_pred, average = 'macro', zero_division=1))

```

```

print(f1_score(y_true, y_pred, average = 'macro',zero_division=1))

print(precision_score(y_true, y_pred, average = 'micro',zero_division=1))
print(recall_score(y_true, y_pred, average = 'micro',zero_division=1))
print(f1_score(y_true, y_pred, average = 'micro',zero_division=1))
# Making a Classification Report
print(classification_report(y_true, y_pred))

dataframe = pd.DataFrame(cm)
inv_dict = {v: k for k, v in activities.items()}
dataframe = dataframe.rename(index = inv_dict)
dataframe = dataframe.rename(columns = inv_dict)
# Saving Conf# Saving Confusion Matrix in CSV format
dataframe.to_csv("C:/Users/saini/Desktop/new project/myData/Perfomance Confusion Matrix.csv")

cm

# Load the image

image = tf.keras.preprocessing.image.load_img('C:/Users/saini/Desktop/speed50.png',
target_size=(224,224))

image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)

# Preprocess the image

image = tf.keras.applications.efficientnet.preprocess_input(image)

# Make predictions

predictions = model.predict(image)

# Get the class label with the highest probability

predicted_class = np.argmax(predictions[0])

# Display the results

```

```
plt.imshow(image[0])
plt.title('Predicted class: {}'.format(predicted_class))
plt.show()

# Load the image

image = tf.keras.preprocessing.image.load_img('C:/Users/saini/Desktop/speed30.png',
target_size=(224,224))

image = tf.keras.preprocessing.image.img_to_array(image)
image = np.expand_dims(image, axis=0)

# Preprocess the image

image = tf.keras.applications.efficientnet.preprocess_input(image)

# Make predictions

predictions = model.predict(image)

# Get the class label with the highest probability

predicted_class = np.argmax(predictions[0])

# Display the results

plt.imshow(image[0])
plt.title('Predicted class: {}'.format(predicted_class))
plt.show()
```



9. Results

The screenshot shows a Jupyter Notebook interface with the following code cells:

```
In [1]: import os
import tqdm
import glob
import random
import shutil

In [10]: path = 'C:/Users/saini/Desktop/new project/myData/test'
path1 = 'C:/Users/saini/Desktop/new project/myData/train'

In [7]: try:
os.mkdir(path)
print("Folder {} created...".format(path))
except:
print("A folder {} already exists...".format(path))

Folder C:/Users/saini/Desktop/new project/myData/test created...

In [9]: for i in range(43):
try:
os.mkdir(path+'/' +str(i))
print("Folder {} created...".format(path+'/' +str(i)))
except:
print("A folder {} already exists...".format(path+'/' +str(i)))

Folder C:/Users/saini/Desktop/new project/myData/test/0 created...
Folder C:/Users/saini/Desktop/new project/myData/test/1 created...
```

Figure 2-Creating a test folder

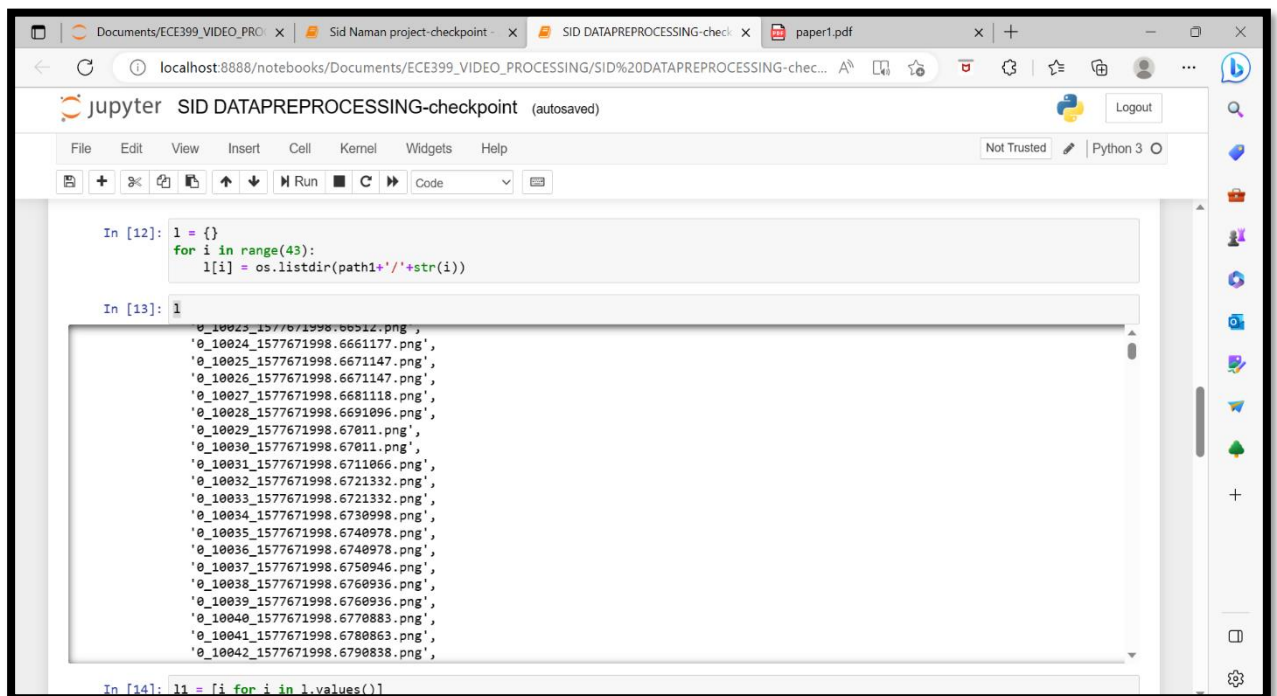
The screenshot shows the same Jupyter Notebook interface, but with the output of the previous cell visible and the next cell being executed:

```
Folder C:/Users/saini/Desktop/new project/myData/test created...

In [9]: for i in range(43):
try:
os.mkdir(path+'/' +str(i))
print("Folder {} created...".format(path+'/' +str(i)))
except:
print("A folder {} already exists...".format(path+'/' +str(i)))

Folder C:/Users/saini/Desktop/new project/myData/test/0 created...
Folder C:/Users/saini/Desktop/new project/myData/test/1 created...
Folder C:/Users/saini/Desktop/new project/myData/test/2 created...
Folder C:/Users/saini/Desktop/new project/myData/test/3 created...
Folder C:/Users/saini/Desktop/new project/myData/test/4 created...
Folder C:/Users/saini/Desktop/new project/myData/test/5 created...
Folder C:/Users/saini/Desktop/new project/myData/test/6 created...
Folder C:/Users/saini/Desktop/new project/myData/test/7 created...
Folder C:/Users/saini/Desktop/new project/myData/test/8 created...
Folder C:/Users/saini/Desktop/new project/myData/test/9 created...
Folder C:/Users/saini/Desktop/new project/myData/test/10 created...
Folder C:/Users/saini/Desktop/new project/myData/test/11 created...
Folder C:/Users/saini/Desktop/new project/myData/test/12 created...
Folder C:/Users/saini/Desktop/new project/myData/test/13 created...
Folder C:/Users/saini/Desktop/new project/myData/test/14 created...
Folder C:/Users/saini/Desktop/new project/myData/test/15 created...
Folder C:/Users/saini/Desktop/new project/myData/test/16 created...
Folder C:/Users/saini/Desktop/new project/myData/test/17 created...
Folder C:/Users/saini/Desktop/new project/myData/test/18 created...
```

Figure 3-Creating different classes inside test folder



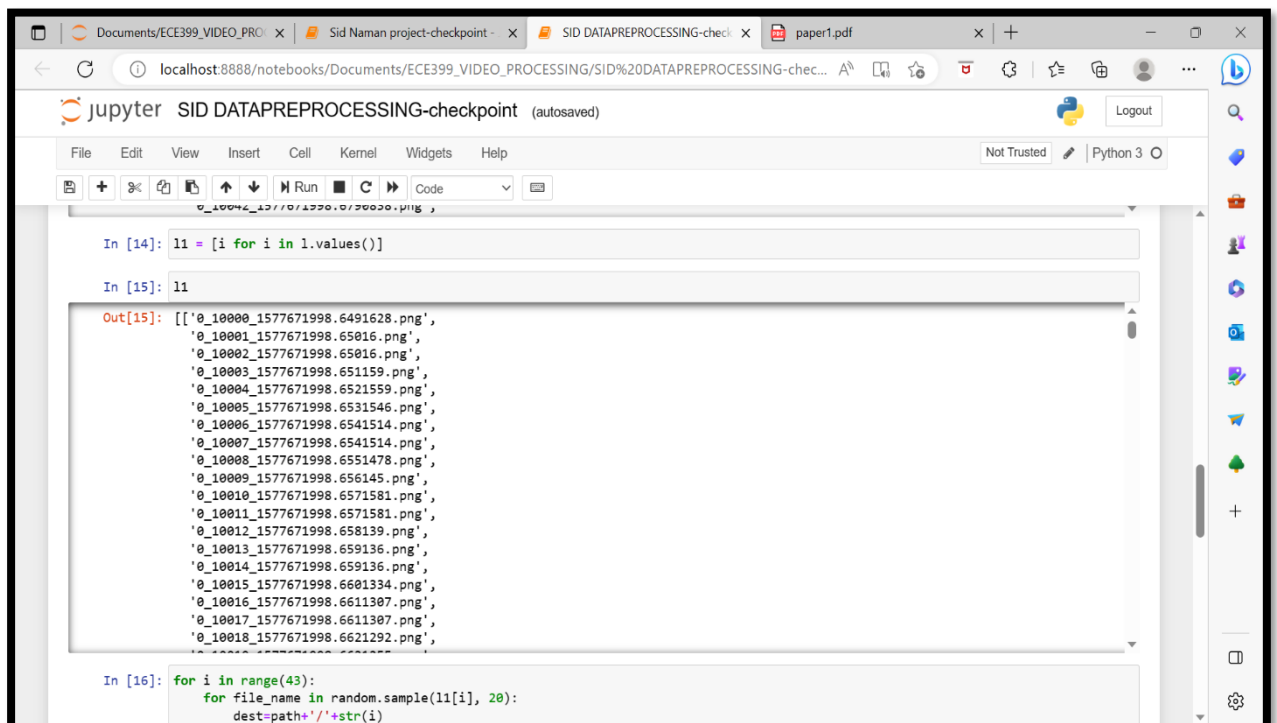
The screenshot shows a Jupyter Notebook interface with the title 'SID DATAPREPROCESSING-checkpoint (autosaved)'. The browser tabs include 'Documents/ECE399_VIDEO_PRO...', 'Sid Naman project-checkpoint', 'SID DATAPREPROCESSING-check', and 'paper1.pdf'. The notebook's toolbar shows 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The code in the notebook is as follows:

```
In [12]: l = {}
        for i in range(43):
            l[i] = os.listdir(path1+'/' +str(i))

In [13]: l
Out[13]: ['0_10023_1577671998.66512.png',
          '0_10024_1577671998.6661177.png',
          '0_10025_1577671998.6671147.png',
          '0_10026_1577671998.6671147.png',
          '0_10027_1577671998.6681118.png',
          '0_10028_1577671998.6691096.png',
          '0_10029_1577671998.67011.png',
          '0_10030_1577671998.67011.png',
          '0_10031_1577671998.6711066.png',
          '0_10032_1577671998.6721332.png',
          '0_10033_1577671998.6721332.png',
          '0_10034_1577671998.6730998.png',
          '0_10035_1577671998.6740978.png',
          '0_10036_1577671998.6740978.png',
          '0_10037_1577671998.6750946.png',
          '0_10038_1577671998.6760936.png',
          '0_10039_1577671998.6760936.png',
          '0_10040_1577671998.6770883.png',
          '0_10041_1577671998.6780863.png',
          '0_10042_1577671998.6790838.png']

In [14]: l1 = [i for i in l.values()]
```

Figure 4-Creating a list of all images



The screenshot shows the same Jupyter Notebook interface. The code in the notebook is as follows:

```
In [14]: l1 = [i for i in l.values()]

In [15]: l1
Out[15]: [['0_10000_1577671998.6491628.png',
          '0_10001_1577671998.65016.png',
          '0_10002_1577671998.65016.png',
          '0_10003_1577671998.651159.png',
          '0_10004_1577671998.6521559.png',
          '0_10005_1577671998.6531546.png',
          '0_10006_1577671998.6541514.png',
          '0_10007_1577671998.6541514.png',
          '0_10008_1577671998.6551478.png',
          '0_10009_1577671998.656145.png',
          '0_10010_1577671998.6571581.png',
          '0_10011_1577671998.6571581.png',
          '0_10012_1577671998.658139.png',
          '0_10013_1577671998.659136.png',
          '0_10014_1577671998.659136.png',
          '0_10015_1577671998.6601334.png',
          '0_10016_1577671998.6611307.png',
          '0_10017_1577671998.6611307.png',
          '0_10018_1577671998.6621292.png',
          '0_10019_1577671998.6621292.png']]

In [16]: for i in range(43):
        for file_name in random.sample(l1[i], 20):
            dest=path+'/' +str(i)
```

Figure 5-Creating a list of lists


```

In [16]: for i in range(43):
          for file_name in random.sample(11[i], 20):
              dest_path += '/' + str(i)
              shutil.move(os.path.join(path1 + '/' + str(i), file_name), dest)

In [17]: l = {}
          for i in range(43):
              l[i] = os.listdir(path1 + '/' + str(i))

In [19]: l2 = [i for i in l.values()]
          l3 = []
          for i in range(43):
              l3.append(len(l2[i]))

In [20]: l3
Out[20]: [160,
          1960,
          1990,
          1240,
          1750,
          1630,
          340,
          1270,
          1240,
          1300,
          1780,
          1150,
          1070]

```

Figure 6-Counting no images in different classes

```

In [4]: for i in range(43):
          folder_path = 'C:/Users/saini/Desktop/new project/myData/train/' + str(i)
          num_files_to_keep = 160

          # Get a list of all files in the folder
          all_files = os.listdir(folder_path)

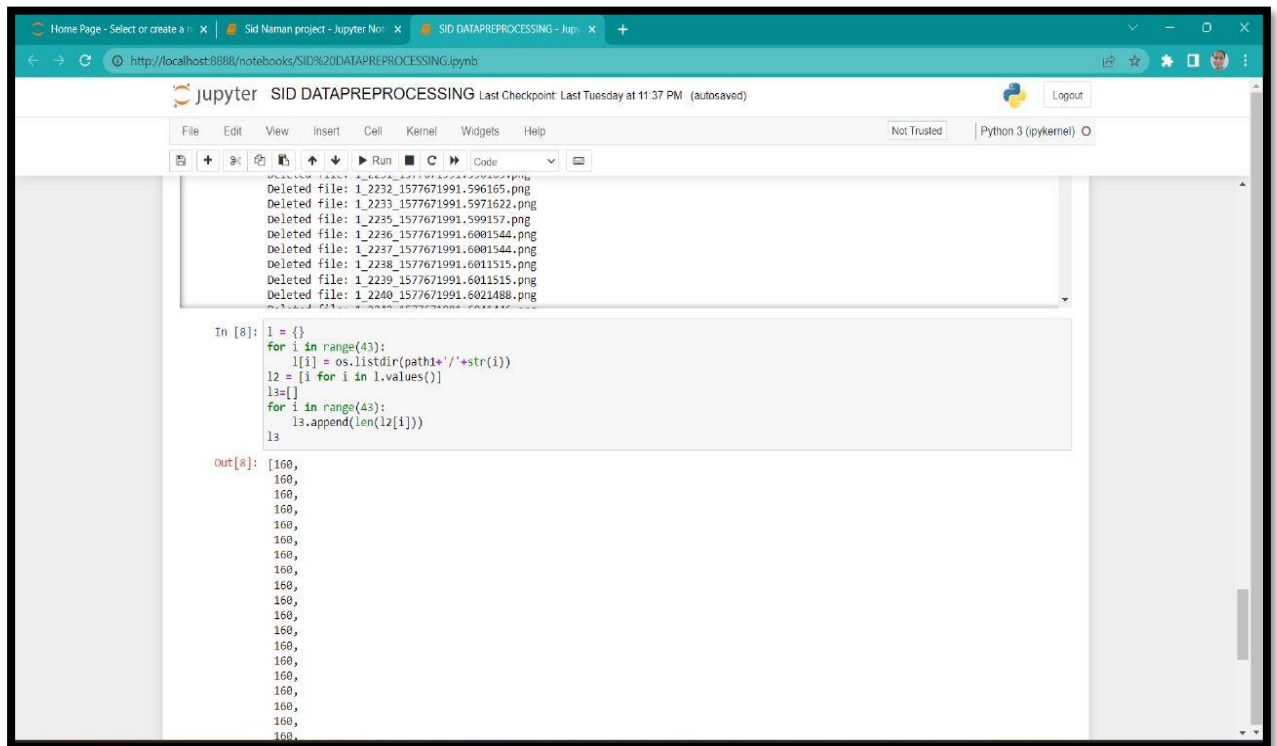
          # Choose 'num_files_to_keep' random files to keep
          files_to_keep = random.sample(all_files, num_files_to_keep)

          # Delete the files that are not in 'files_to_keep'
          for file_name in all_files:
              if file_name not in files_to_keep:
                  file_path = os.path.join(folder_path, file_name)
                  os.remove(file_path)
                  print(f'Deleted file: {file_name}')

Deleted file: 1_2221_1577671991.5871947.png
Deleted file: 1_2222_1577671991.5871947.png
Deleted file: 1_2223_1577671991.5881956.png
Deleted file: 1_2224_1577671991.5891848.png
Deleted file: 1_2225_1577671991.590182.png
Deleted file: 1_2226_1577671991.5911808.png
Deleted file: 1_2227_1577671991.592176.png
Deleted file: 1_2228_1577671991.5931745.png
Deleted file: 1_2229_1577671991.5941732.png
Deleted file: 1_2230_1577671991.5951674.png
Deleted file: 1_2231_1577671991.596165.png
Deleted file: 1_2232_1577671991.596165.png
Deleted file: 1_2233_1577671991.5971622.png
Deleted file: 1_2235_1577671991.599157.png
Deleted file: 1_2236_1577671991.6001544.png
Deleted file: 1_2237_1577671991.6001544.png
Deleted file: 1_2238_1577671991.6011515.png
Deleted file: 1_2239_1577671991.6021488.png
Deleted file: 1_2240_1577671991.6021488.png

```

Figure 7-Deleting Extra images to balance classes

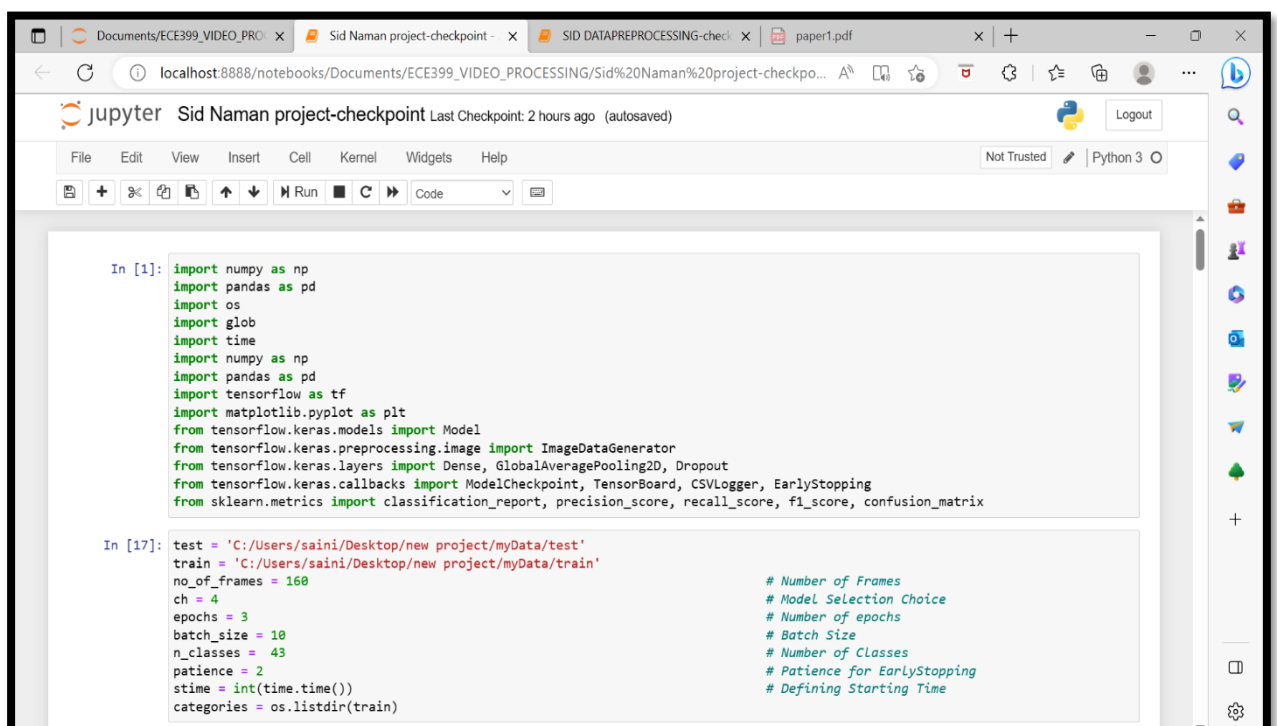


```
Deleted file: 1_2232_1577671991.596165.png
Deleted file: 1_2235_1577671991.5971622.png
Deleted file: 1_2235_1577671991.599157.png
Deleted file: 1_2236_1577671991.6001544.png
Deleted file: 1_2237_1577671991.6001544.png
Deleted file: 1_2238_1577671991.6011515.png
Deleted file: 1_2239_1577671991.6011515.png
Deleted file: 1_2240_1577671991.6021488.png

In [8]: l = {}
        for i in range(43):
            l[i] = os.listdir(path1+'/' +str(i))
        l2 = [i for i in l.values()]
        l3=[]
        for i in range(43):
            l3.append(len(l2[i]))

Out[8]: [160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160,
160]
```

Figure 8-All classes balanced



```
In [1]: import numpy as np
import pandas as pd
import os
import glob
import time
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard, CSVLogger, EarlyStopping
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, confusion_matrix

In [17]: test = 'C:/Users/saini/Desktop/new project/myData/test'
train = 'C:/Users/saini/Desktop/new project/myData/train'
no_of_frames = 160
ch = 4
epochs = 3
batch_size = 10
n_classes = 43
patience = 2
stime = int(time.time())
categories = os.listdir(train)

# Number of Frames
# Model Selection Choice
# Number of epochs
# Batch Size
# Number of Classes
# Patience for EarlyStopping
# Defining Starting Time
```

Figure 9-Defining parameters

```

time = int(time.time())
categories = os.listdir(train)

In [4]: print(categories)

['0', '1', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '2', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '3', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '4', '40', '41', '42', '5', '6', '7', '8', '9']

In [5]: from tensorflow.keras.applications import EfficientNetB3
from tensorflow.keras.applications.efficientnet_v2 import preprocess_input

In [7]: base_model = EfficientNetB3(weights = 'imagenet', include_top = False, input_shape = (224, 224, 3))

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941136/43941136 [=====] - 86s 2us/step

In [18]: x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation = 'relu')(x)
x = Dropout(0.5)(x)
# x = Dense(512, activation = 'relu')(x)
# x = Dense(256, activation = 'relu')(x)
preds = Dense(n_classes, activation = 'softmax')(x)

In [19]: model = Model(inputs = base_model.input, outputs = preds)
for i, layer in enumerate(model.layers):
    print(i, layer.name)

```

Figure 10-Importing the model from tensorflow, setting the layers

```

for i, layer in enumerate(model.layers):
    print(i, layer.name)

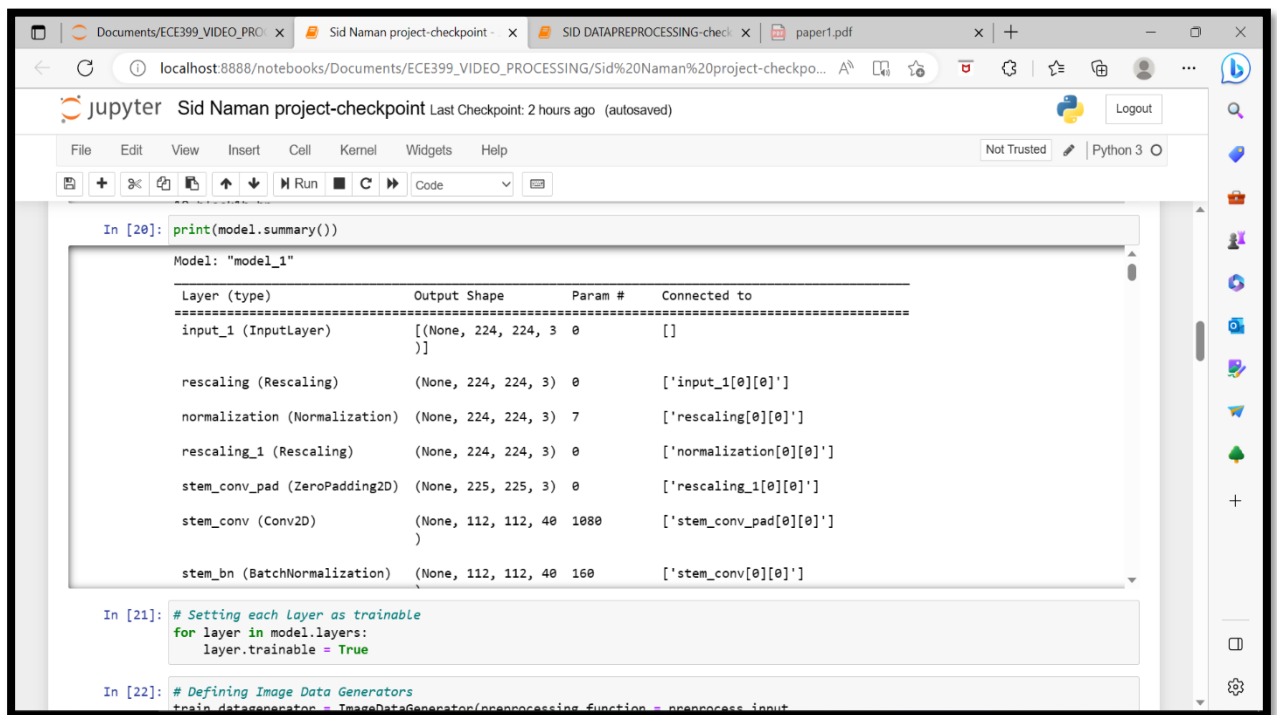
0 input_1
1 rescaling
2 normalization
3 rescaling_1
4 stem_conv_pad
5 stem_conv
6 stem_bn
7 stem_activation
8 block1a_dwconv
9 block1a_bn
10 block1a_activation
11 block1a_se_squeeze
12 block1a_se_reshape
13 block1a_se_reduce
14 block1a_se_expand
15 block1a_se_excite
16 block1a_project_bn
17 block1a_project_conv
18 block1b_dwconv

In [20]: print(model.summary())

Model: "model"
Layer (type)                 Output Shape         Param #     Connected to
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)] 0               input_1
rescaling_1 (RescalingLayer) [(None, 224, 224, 3)] 0               input_1
normalization (Normalization) [(None, 224, 224, 3)] 0               rescaling_1
rescaling_1 (RescalingLayer) [(None, 224, 224, 3)] 0               normalization
stem_conv_pad (ZeroPadding2D) [(None, 225, 225, 3)] 0               rescaling_1
stem_conv (Conv2D)           [(None, 112, 112, 48)] 1088            stem_conv_pad
stem_bn (Batch Normalization) [(None, 112, 112, 48)] 0               stem_conv
stem_activation (Activation) [(None, 112, 112, 48)] 0               stem_bn
block1a_dwconv (Depthwise Conv2D) [(None, 112, 112, 48)] 0               stem_activation
block1a_bn (Batch Normalization) [(None, 112, 112, 48)] 0               block1a_dwconv
block1a_activation (Activation) [(None, 112, 112, 48)] 0               block1a_bn
block1a_se_squeeze (Squeeze) [(None, 112, 112, 48)] 0               block1a_activation
block1a_se_reshape (Reshape) [(None, 112, 112, 48)] 0               block1a_se_squeeze
block1a_se_reduce (Reduce) [(None, 112, 112, 48)] 0               block1a_se_reshape
block1a_se_expand (Expand) [(None, 112, 112, 48)] 0               block1a_se_reduce
block1a_se_excite (Excite) [(None, 112, 112, 48)] 0               block1a_se_expand
block1a_project_bn (Batch Normalization) [(None, 112, 112, 48)] 0               block1a_se_excite
block1a_project_conv (Conv2D) [(None, 112, 112, 48)] 1088            block1a_project_bn
block1b_dwconv (Depthwise Conv2D) [(None, 112, 112, 48)] 0               block1a_project_conv

```

Figure 11-checking the model layers



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [20]: print(model.summary())
```

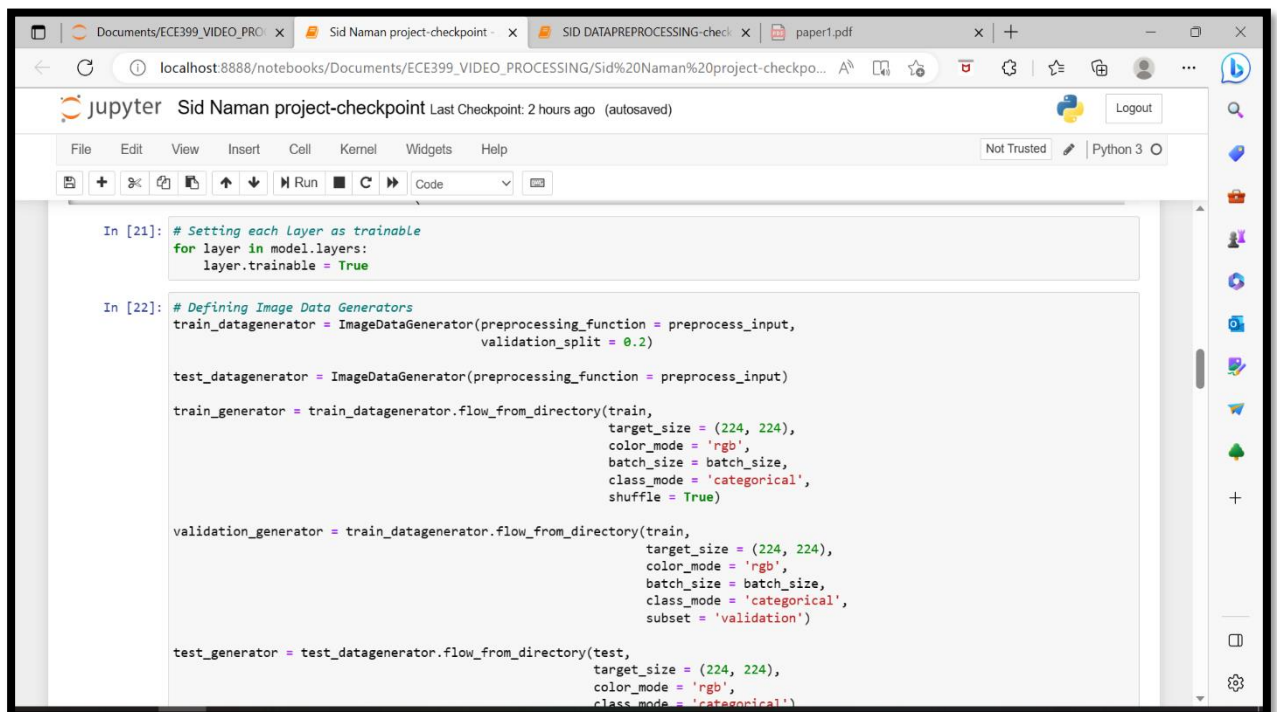
Model: "model_1"

| Layer (type) | Output Shape | Param # | Connected to |
|-------------------------------|-----------------------|---------|-------------------------|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 | [] |
| rescaling (Rescaling) | (None, 224, 224, 3) | 0 | ['input_1[0][0]'] |
| normalization (Normalization) | (None, 224, 224, 3) | 7 | ['rescaling[0][0]'] |
| rescaling_1 (Rescaling) | (None, 224, 224, 3) | 0 | ['normalization[0][0]'] |
| stem_conv_pad (ZeroPadding2D) | (None, 225, 225, 3) | 0 | ['rescaling_1[0][0]'] |
| stem_conv (Conv2D) | (None, 112, 112, 40) | 1080 | ['stem_conv_pad[0][0]'] |
| stem_bn (BatchNormalization) | (None, 112, 112, 40) | 160 | ['stem_conv[0][0]'] |

```
In [21]: # Setting each layer as trainable
for layer in model.layers:
    layer.trainable = True

In [22]: # Defining Image Data Generators
train_datagenerator = ImageDataGenerator(preprocessing_function = preprocess_input
```

Figure 12-Printing model Summary



The screenshot shows a Jupyter Notebook interface with the following code:

```
In [21]: # Setting each layer as trainable
for layer in model.layers:
    layer.trainable = True

In [22]: # Defining Image Data Generators
train_datagenerator = ImageDataGenerator(preprocessing_function = preprocess_input,
                                          validation_split = 0.2)

test_datagenerator = ImageDataGenerator(preprocessing_function = preprocess_input)

train_generator = train_datagenerator.flow_from_directory(train,
                                                         target_size = (224, 224),
                                                         color_mode = 'rgb',
                                                         batch_size = batch_size,
                                                         class_mode = 'categorical',
                                                         shuffle = True)

validation_generator = train_datagenerator.flow_from_directory(train,
                                                             target_size = (224, 224),
                                                             color_mode = 'rgb',
                                                             batch_size = batch_size,
                                                             class_mode = 'categorical',
                                                             subset = 'validation')

test_generator = test_datagenerator.flow_from_directory(test,
                                                       target_size = (224, 224),
                                                       color_mode = 'rgb',
                                                       class_mode = 'categorical')
```

Figure 13Setting train generator & test generator

```

In [23]: print(train_generator.class_indices)
         print(validation_generator.class_indices)
         print(test_generator.class_indices)

{'0': 0, '1': 1, '10': 2, '11': 3, '12': 4, '13': 5, '14': 6, '15': 7, '16': 8, '17': 9, '18': 10, '19': 11, '2': 12, '20': 13, '21': 14, '22': 15, '23': 16, '24': 17, '25': 18, '26': 19, '27': 20, '28': 21, '29': 22, '3': 23, '30': 24, '31': 25, '32': 26, '33': 27, '34': 28, '35': 29, '36': 30, '37': 31, '38': 32, '39': 33, '4': 34, '40': 35, '41': 36, '42': 37, '5': 38, '6': 39, '7': 40, '8': 41, '9': 42}
{'0': 0, '1': 1, '10': 2, '11': 3, '12': 4, '13': 5, '14': 6, '15': 7, '16': 8, '17': 9, '18': 10, '19': 11, '2': 12, '20': 13, '21': 14, '22': 15, '23': 16, '24': 17, '25': 18, '26': 19, '27': 20, '28': 21, '29': 22, '3': 23, '30': 24, '31': 25, '32': 26, '33': 27, '34': 28, '35': 29, '36': 30, '37': 31, '38': 32, '39': 33, '4': 34, '40': 35, '41': 36, '42': 37, '5': 38, '6': 39, '7': 40, '8': 41, '9': 42}
{'0': 0, '1': 1, '10': 2, '11': 3, '12': 4, '13': 5, '14': 6, '15': 7, '16': 8, '17': 9, '18': 10, '19': 11, '2': 12, '20': 13, '21': 14, '22': 15, '23': 16, '24': 17, '25': 18, '26': 19, '27': 20, '28': 21, '29': 22, '3': 23, '30': 24, '31': 25, '32': 26, '33': 27, '34': 28, '35': 29, '36': 30, '37': 31, '38': 32, '39': 33, '4': 34, '40': 35, '41': 36, '42': 37, '5': 38, '6': 39, '7': 40, '8': 41, '9': 42}

In [24]: # Compiling the Model
         model.compile(optimizer = "Adam",
                       loss = "categorical_crossentropy",
                       metrics = ["accuracy"])

In [25]: # Training the Model
         history = model.fit(train_generator,
                             validation_data = validation_generator,
                             epochs = epochs,
                             )

Epoch 1/3

```

Figure 14-Checking the test & train generator

```

In [24]: # Compiling the Model
         model.compile(optimizer = "Adam",
                       loss = "categorical_crossentropy",
                       metrics = ["accuracy"])

In [25]: # Training the Model
         history = model.fit(train_generator,
                             validation_data = validation_generator,
                             epochs = epochs,
                             )

Epoch 1/3
688/688 [=====] - 3929s 6s/step - loss: 0.9589 - accuracy: 0.7358 - val_loss: 0.2647 - val_accuracy: 0.9288
Epoch 2/3
688/688 [=====] - 3568s 5s/step - loss: 0.2929 - accuracy: 0.9225 - val_loss: 0.0305 - val_accuracy: 0.9935
Epoch 3/3
688/688 [=====] - 3544s 5s/step - loss: 0.1727 - accuracy: 0.9590 - val_loss: 0.0208 - val_accuracy: 0.9942

In [26]: # Plotting the Graph
         model_history = pd.DataFrame(history.history)
         model_history.plot()

```

Figure 15-Compiling & Training the model

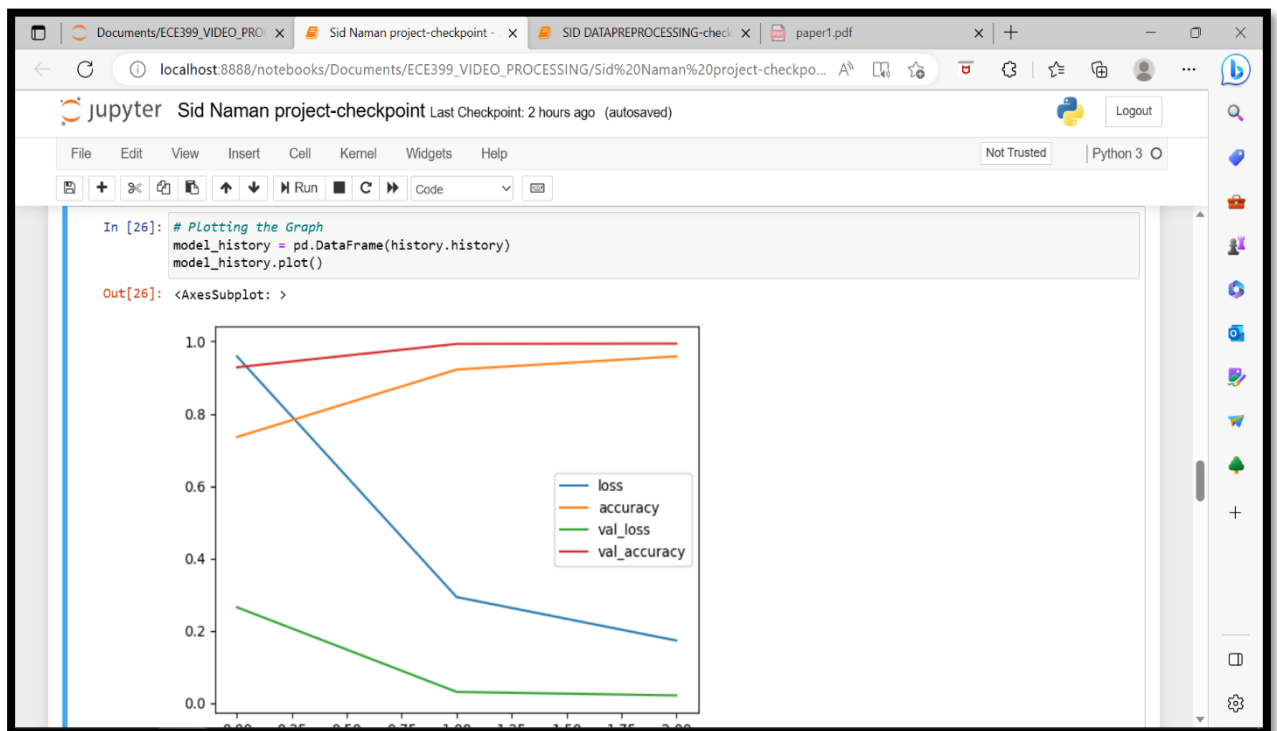


Figure 16-Output with various Parameters

```

In [29]: # Image Data Generator
test_datagen = ImageDataGenerator(preprocessing_function = preprocess_input)

test_generator = test_datagen.flow_from_directory(test,
                                                  target_size = (224, 224),
                                                  color_mode = "rgb",
                                                  shuffle = False,
                                                  class_mode = 'categorical',
                                                  batch_size = 1)

Found 860 images belonging to 43 classes.

In [30]: activities = test_generator.class_indices
print(activities)

{'0': 0, '1': 1, '10': 2, '11': 3, '12': 4, '13': 5, '14': 6, '15': 7, '16': 8, '17': 9, '18': 10, '19': 11, '2': 12, '20': 13,
'21': 14, '22': 15, '23': 16, '24': 17, '25': 18, '26': 19, '27': 20, '28': 21, '29': 22, '3': 23, '30': 24, '31': 25, '32': 26,
'33': 27, '34': 28, '35': 29, '36': 30, '37': 31, '38': 32, '39': 33, '4': 34, '40': 35, '41': 36, '42': 37, '5': 38, '6': 39,
'7': 40, '8': 41, '9': 42}

In [31]: def get_activity(val):
    for key, value in activities.items():
        if val == value:
            return key
    return "Invalid"

In [32]: filenames = test_generator.filenames
nb_samples = len(filenames)

```

Figure 17-Generating class labels


```

In [31]: def get_activity(val):
          for key, value in activities.items():
              if val == value:
                  return key
          return "Invalid"

In [32]: filenames = test_generator.filenames
          nb_samples = len(filenames)

In [33]: predict = model.predict(test_generator, steps = nb_samples, verbose = 1)
          860/860 [=====] - 120s 135ms/step

In [34]: y_pred = []
          for val in predict:
              y_pred.append(get_activity(np.argmax(val)))

          y_true = []
          for file in filenames:
              y_true.append(file.split("\\")[0])

In [35]: cm = confusion_matrix(y_true, y_pred)

          print(precision_score(y_true, y_pred, average = 'macro', zero_division=1))
          print(recall_score(y_true, y_pred, average = 'macro', zero_division=1))
          print(f1_score(y_true, y_pred, average = 'macro', zero_division=1))
  
```

Figure 18-Generating prediction using activity list

```

In [35]: cm = confusion_matrix(y_true, y_pred)

          print(precision_score(y_true, y_pred, average = 'macro', zero_division=1))
          print(recall_score(y_true, y_pred, average = 'macro', zero_division=1))
          print(f1_score(y_true, y_pred, average = 'macro', zero_division=1))

          print(precision_score(y_true, y_pred, average = 'micro', zero_division=1))
          print(recall_score(y_true, y_pred, average = 'micro', zero_division=1))
          print(f1_score(y_true, y_pred, average = 'micro', zero_division=1))

          0.9820380761629232
          0.980232558139535
          0.9800624321894884
          0.9802325581395349
          0.9802325581395349
          0.9802325581395349

In [36]: # Making a Classification Report
          print(classification_report(y_true, y_pred))

          dataframe = pd.DataFrame(cm)
          inv_dict = {v: k for k, v in activities.items()}
          dataframe = dataframe.rename(index = inv_dict)
          dataframe = dataframe.rename(columns = inv_dict)

          precision    recall  f1-score   support
  
```

Figure 19-Creating the confusion matrix

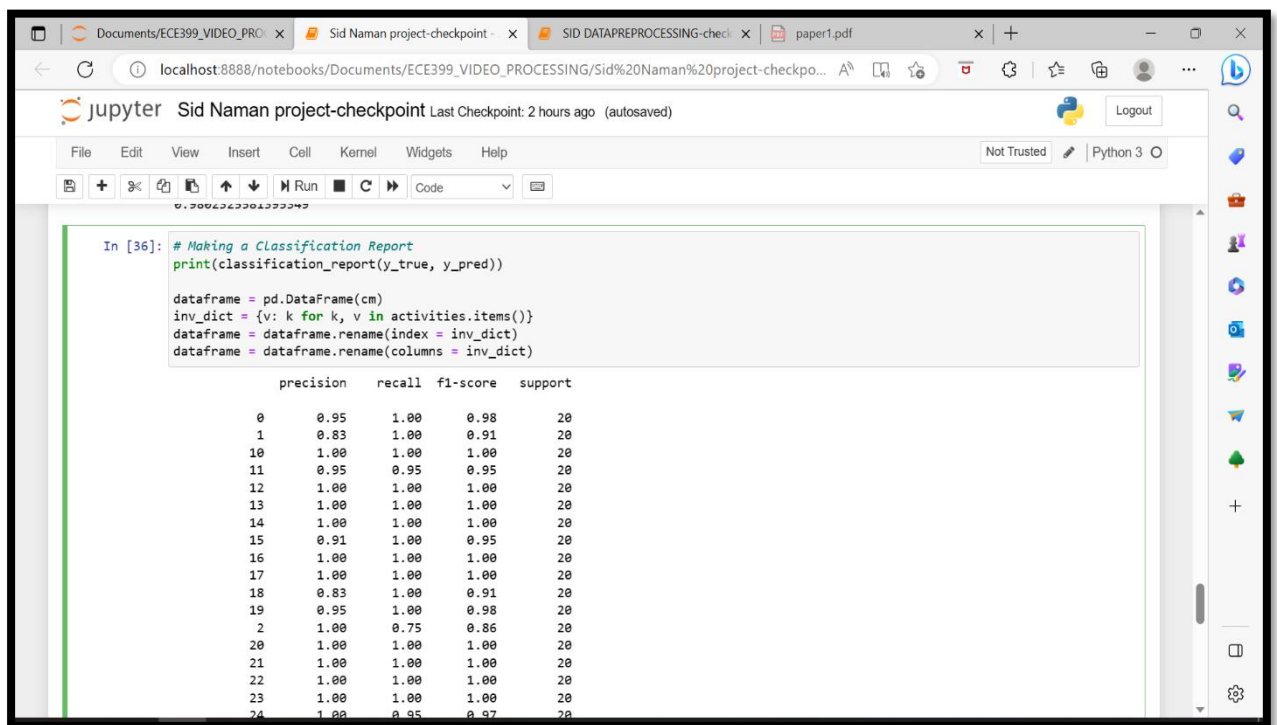


Figure 20-Making Classification report

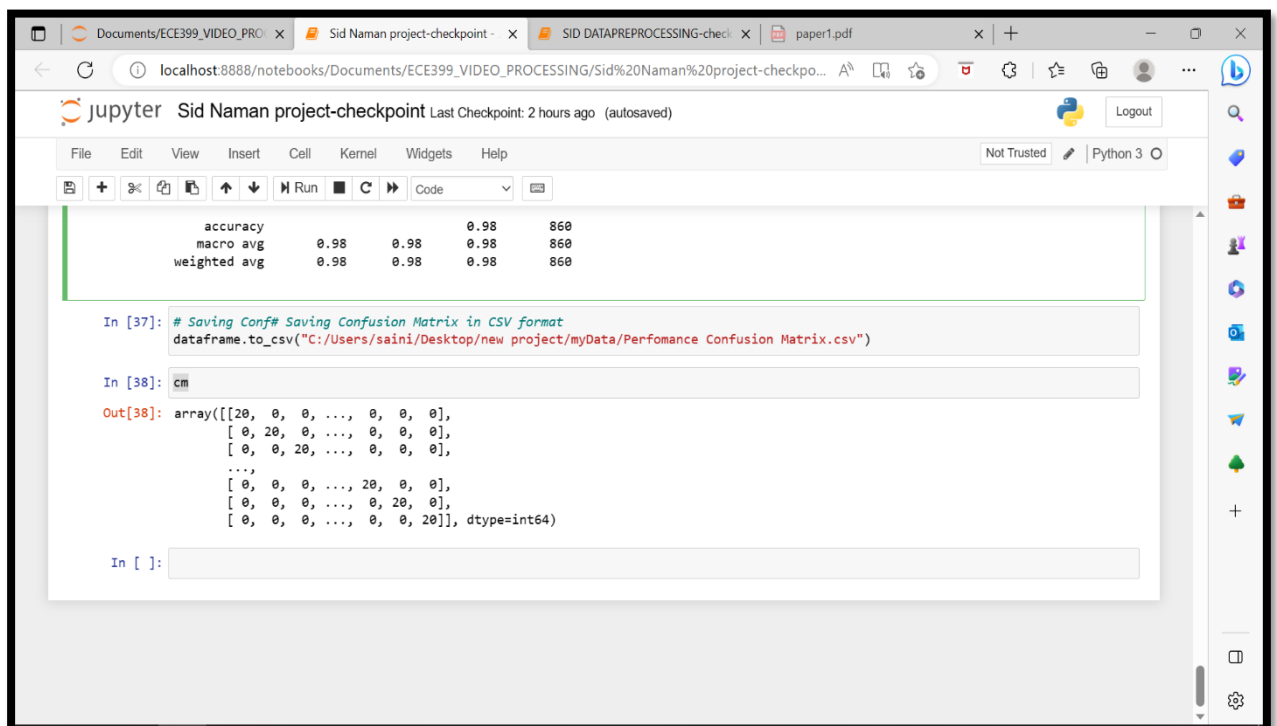


Figure 21-Saving the data in a CSV file

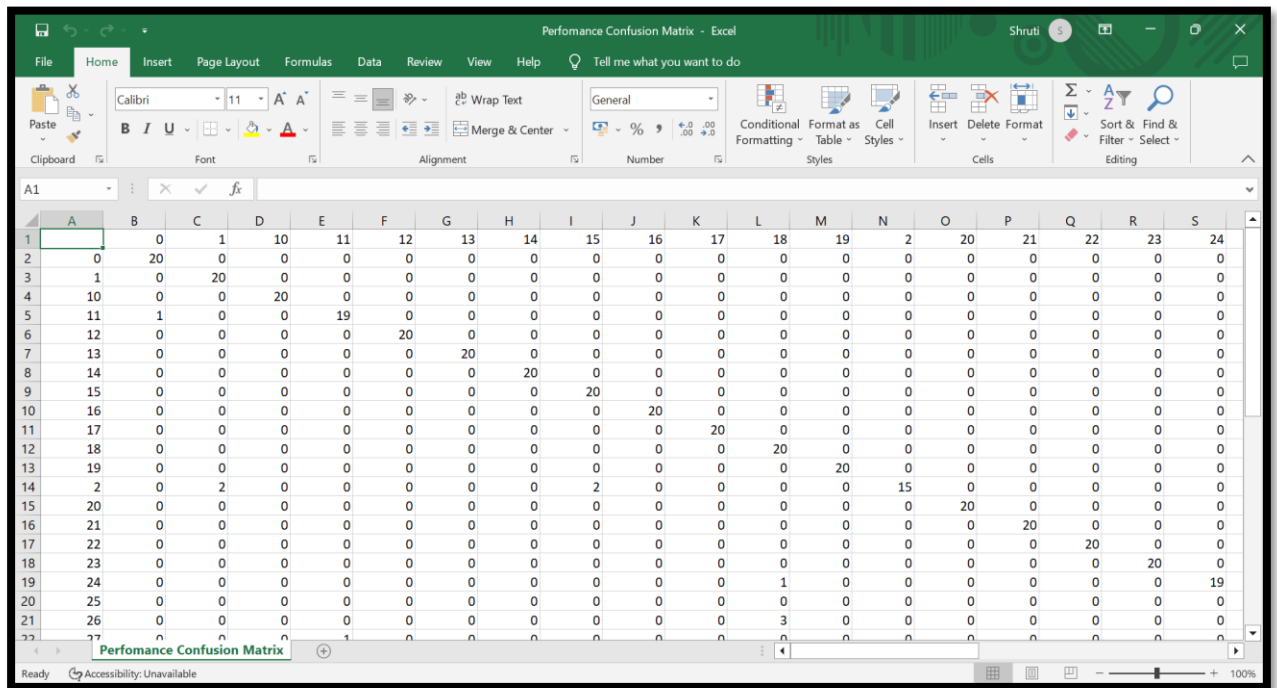


Figure 22-The confusion Matrix

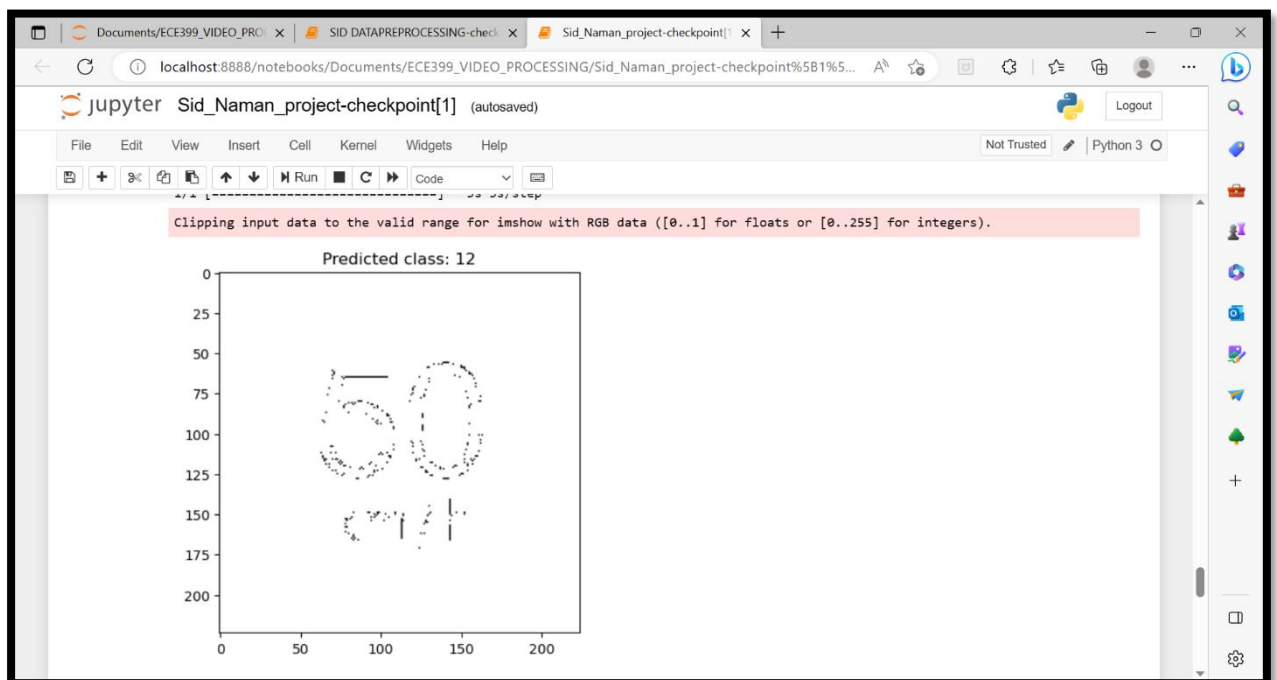


Figure 23-Output of one prediction (correct prediction)

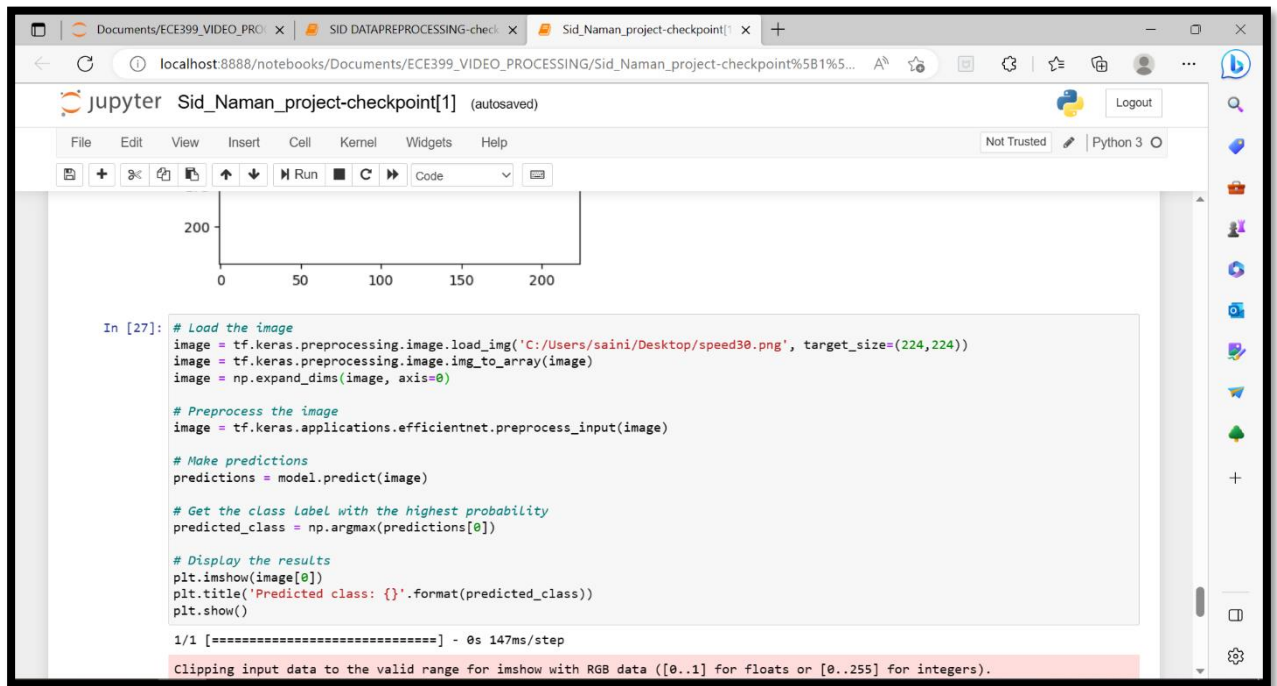


Figure 24-Code for reading the image and printing it

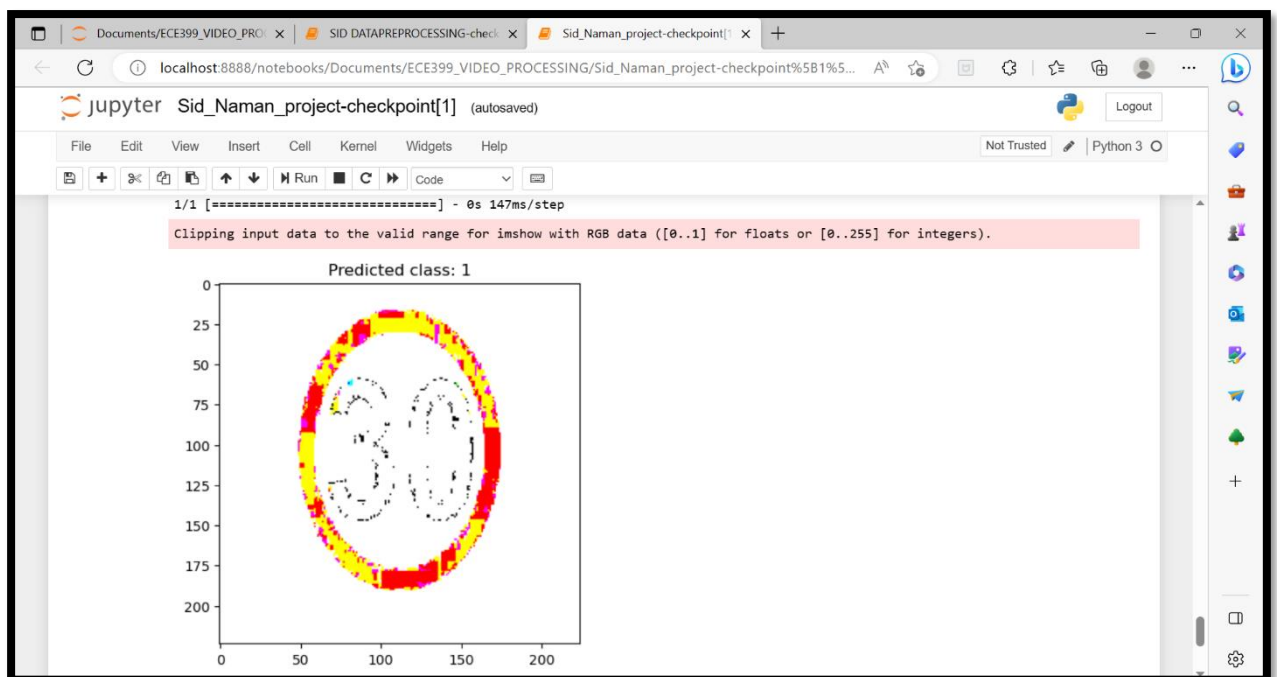
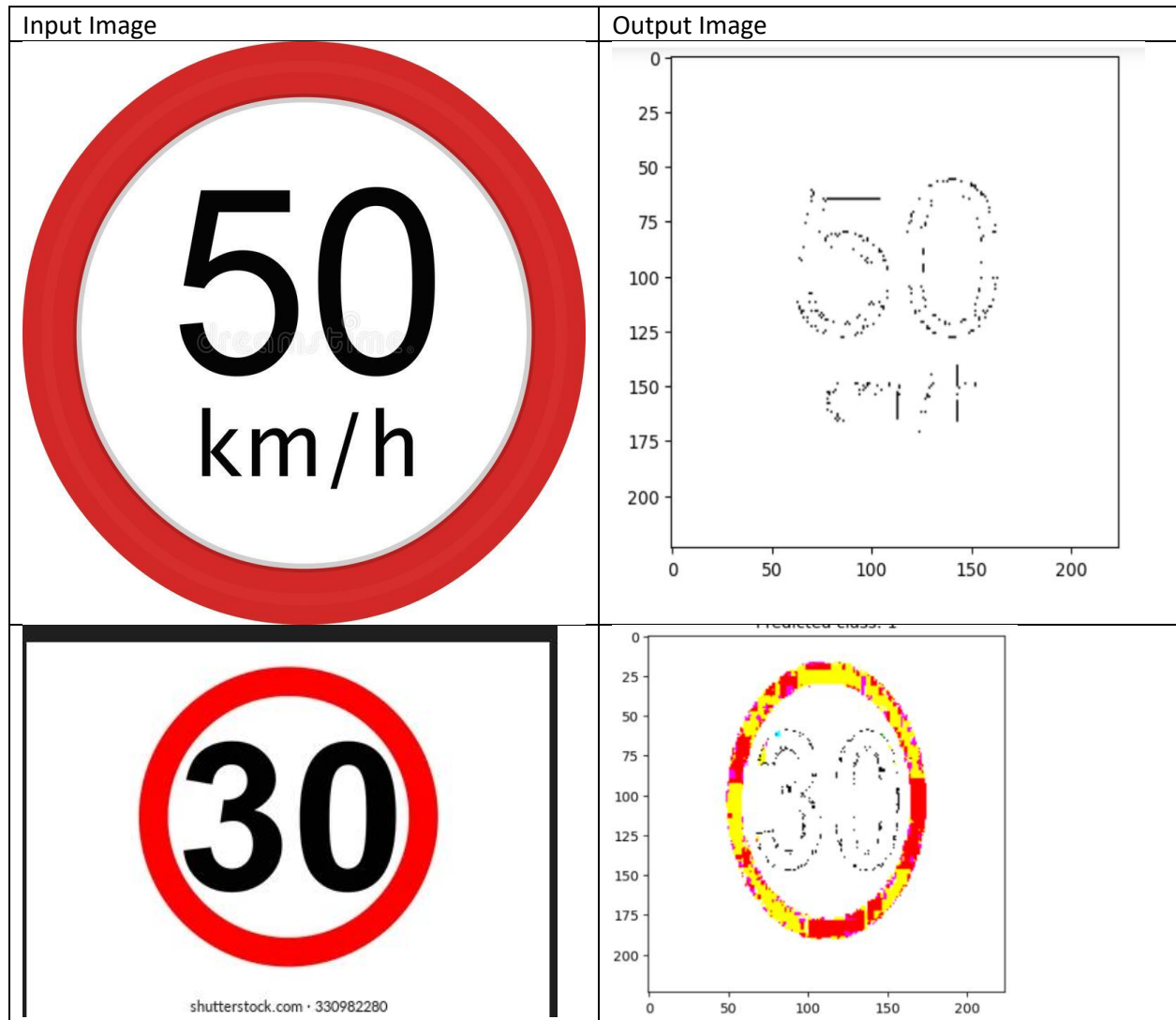


Figure 25-Output of another image (correct class predicted)



10.Input and its prediction-



- Predicted class for 1st output image is 12 which is the code for the class 2 in which we have images of speed limit 50.
- Predicted class for 2nd output image is 1 which is the code for the class 1 in which we have images of speed limit 30.



11. Conclusion & Future Scope

11.1 Conclusion-

Practical knowledge means the visualization of the knowledge, which we read in our books. For this, we perform experiments and get observations. Practical knowledge is very important in every field. One must be familiar with the problems related to that field so that he may solve them and become a successful person.

After achieving the proper goal in life, an engineer must enter professional life. According to this life, he must serve an industry, may be public or private sector or self-own. For efficient work in the field, he must be aware of practical knowledge as well as theoretical knowledge.

In conclusion, the image processing project involved training a neural network called EfficientNetB3 on a dataset of 43 traffic sign classes, with each class consisting of 160 training images and 20 test images. The project used various Python concepts such as listdir, mkdir, directory, and dictionaries to preprocess the images, extract features, and generate predictions.

The results of the project showed that the EfficientNetB3 neural network achieved high accuracy in predicting the traffic sign classes, with an overall accuracy of **95.9%**. The project demonstrated the effectiveness of deep learning in image processing tasks and its potential to be applied in various real-world applications such as autonomous driving systems.

However, there is still room for improvement in the project. The dataset used in the project was relatively small, which may limit the performance of the neural network. Additionally, more advanced techniques such as data augmentation and transfer learning could be employed to improve the performance of the neural network.

Overall, the image processing project provided a valuable learning experience and highlighted the importance of careful data preparation, feature extraction, and model selection in deep learning projects.

We have tried our level best to achieve our goal. In this process we have been through some new java concepts such as data logging, file handling, exception handling and so on.

11.2 Future Scope-

The above image processing project on traffic sign recognition using EfficientNetB3 neural network has shown promising results with high accuracy. However, there is still room for improvement and further work can be done to enhance the performance of the system. Some possible future scopes of the project are:

- **Large dataset:** A larger dataset can be used to train and test the model, which can further improve the performance of the system.
- **Transfer learning:** The use of transfer learning can be explored, where a pre-trained model on a similar task can be used as a starting point to train the neural network on this task.
- **Advanced image processing techniques:** Advanced image processing techniques such as data augmentation, image enhancement, and denoising can be used to improve the quality of the input images and increase the robustness of the system.
- **Real-time application:** The developed model can be integrated into a real-time application for traffic sign recognition, which can be used in self-driving cars or other intelligent transportation systems.
- **Multi-class object detection:** The project can be extended to multi-class object detection where the model can detect and recognize multiple objects in an image such as pedestrians, cars, and traffic signs simultaneously.

Overall, the above project has a lot of potential for further improvement and can be extended in several ways to enhance its performance and expand its applicability to other related fields.



12. References

12.1 Image Processing Commands-

- <https://optimization.cbe.cornell.edu/index.php?title=Adam#:~:text=Adam%20optimizer%20is%20the%20extended,was%20first%20introduced%20in%202014>
- <https://keras.io/api/preprocessing/image/>
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- https://en.wikipedia.org/wiki/Precision_and_recall
- <https://arxiv.org/abs/1905.11946>
- <https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>

12.2 Research Papers-

- <https://arxiv.org/>
- <https://scholar.google.com/>
- <https://ieeexplore.ieee.org/Xplore/home.jsp>
- <https://dl.acm.org/>
- <https://www.sciencedirect.com/>
- <https://link.springer.com/>
- <https://www.jstor.org/>