College code:7311

College name: Government College of Engineering, Erode

Department: Computer Science and Engineering

TRAFFIC MANAGEMENT

TEAM MEMBERS:

Poornima @ Buvana R [21CSE57L]

Sivaranjani S [21CSE58L]

PHASE 3:

DEVELOPMENT PART 1

This Traffic Management System will not only optimize traffic flow but also contribute to a safer and more sustainable urban environment. By harnessing the power of Python and its extensive libraries, we aim to create a cutting-edge solution that addresses the challenges of modern urban transportation.

Obtain the Dataset:

You can find datasets for traffic management from various sources, including government agencies, research institutions, or online data repositories. Make sure the dataset is relevant to your specific problem (e.g., traffic prediction, congestion analysis, etc.).

Read and Understand the Data:

Load the dataset into a suitable data structure (e.g., pandas DataFrame in Python). Understand the features (columns) and the meaning of each variable.

Data Cleaning:

Handle missing values: Depending on the dataset, you may encounter missing data. Decide whether to impute missing values or remove the corresponding entries.
Handle duplicates: Check for and remove duplicate records if they exist.
Handle outliers: Identify and decide whether to remove or transform outliers.
Data Exploration and Visualization:

Understand the distribution of variables, correlations, and any interesting patterns in the data. Visualization tools like matplotlib or seaborn in Python can be helpful.

Feature Engineering:

Create new features if necessary. For example, you might extract time-related information (hour, day of the week, etc.) from timestamps, or derive additional features from existing ones.

Data Transformation:
Convert categorical variables into numerical format (e.g., one-hot encoding or label encoding).
Standardize or normalize numerical features if necessary.

Splitting the Data:
Divide the dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used for hyperparameter tuning, and the test set is used for evaluating the final model.

Scaling:
Depending on the algorithm you plan to use, you might need to scale the features (e.g., using StandardScaler in scikit-learn) to ensure they have similar magnitudes.

Save Preprocessed Data:

Once you've completed the preprocessing steps, it's a good practice to save the preprocessed data so you can easily load it for future analyses without repeating the entire process.

Model Building:

Use the preprocessed data to train a model for your specific traffic management task. This could be a regression model for traffic prediction, a classification model for congestion analysis, or any other relevant machine learning approach.

Importing Libraries

- Pandas – Use to load the data frame in a 2D array format.
- NumPy – NumPy arrays are very fast and can perform large computations in a very short time.
- Matplotlib – This library is used to draw visualizations.
- Sklearn – It contains multiple libraries having pre-implemented functions of model development and evaluation.
- OpenCV – This library mainly focused on image processing and handling.
- Tensorflow – It provides a range of functions to achieve complex functionalities with single lines of code.

PYTHON CODE:

```
import matplotlib.image as mpimg  import os

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing import image_dataset_from_directory  from
tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from keras.utils.np_utils import to_categorical
from tensorflow.keras.utils import image_dataset_from_directory  from
tensorflow.keras.optimizers import Adam  from tensorflow.keras.layers import
Conv2D, MaxPooling2D  from tensorflow.keras.layers import Activation,
Dropout, Flatten, Dense  from tensorflow.keras.models import Sequential  from
keras import layers  from tensorflow import keras
from tensorflow.keras.layers.experimental.preprocessing import Rescaling  from
sklearn.model_selection import train_test_split
```

```
import matplotlib.pyplot as plt
import tensorflow as tf  import
pandas as pd  import numpy as np
from glob import glob  import cv2
import warnings
warnings.filterwarnings('ignore')
```

Loading and Extracting the Dataset:

The dataset has 58 classes of Traffic Signs and a label.csv file. The folder is in zip format. To unzip the dataset, we will run the code below.

PYTHON CODE:

```
 # Extracting the compressed dataset.  from zipfile import ZipFile data_path
   = '/content/traffic-sign-dataset-classification.zip' with
      ZipFile(data_path, 'r') as zip
    : zip.extractall()
```

Data Visualization

Data visualization means the visualization of data to understand the key features of the dataset.

PYTHON CODE:

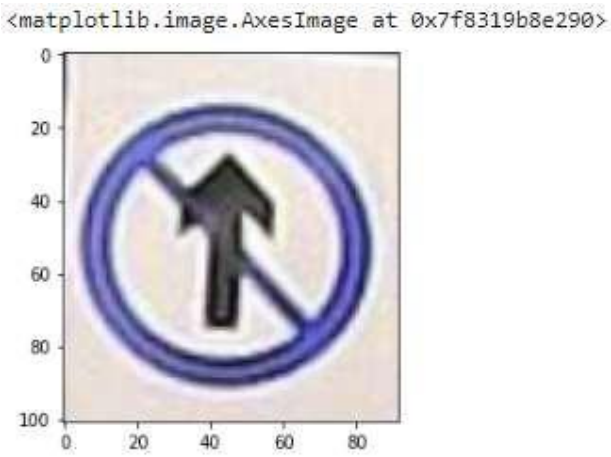
```
# path to the folder containing our dataset  dataset =
'../content/traffic_Data/DATA'

# path of label file  labelfile =
pd.read_csv('labels.csv')
```

Once we load the dataset, now let's visualize some random images from the different folders. For that, we will use plt.imshow() command.

PYTHON CODE:

```
# Visualize some images from the dataset
img = cv2.imread("/content/traffic_Data/DATA/10/010_0011.png")  plt.imshow(img)
```

Output:



PYTHON CODE:

img = cv2.imread("/content/traffic_Data/DATA/23/023_0001.png") plt.imshow(img)

Output:

Label File

This file includes the 58 rows and 2 columns. The columns contain the class id and the name of the symbol. And the rows depict the 58 different classes id and names.

PYTHON CODE:

labelfile.head()

Output:

	ClassId	Name
0	0	Speed limit (5km/h)
1	1	Speed limit (15km/h)
2	2	Speed limit (30km/h)
3	3	Speed limit (40km/h)
4	4	Speed limit (50km/h)

Data Preparation for Training

In this section, we will split the dataset into train and val set. Train set will be used to train the model and val set will be use to evaluate the performance of our model.

PYTHON CODE:

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset,
validation_split=0.2, subset='tr aining',
image size= (224, 224), seed=123, batch_size=32)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(dataset,
validation_split=0.2, subset='vali dation',
image_size = (224, 224), seed=123, batch_size=32)
```

Output:

Found 4170 files belonging to 58 classes.

Using 3336 files for training.

Found 4170 files belonging to 58 classes.

Using 834 files for validation.

Once we split the dataset, Let’s create the list of the class names and print few images along with their class names.

PYTHON CODE:

```
class_numbers = train_ds.class_names
class_names = []  for i in class_numbers:
class_names.append(labelfile['Name'][int(i)])
```

Let's visualize the train dataset and print 25 images from the dataset.

PYTHON CODE:

```
plt.figure(figsize=(10, 10))  for images, labels in
train_ds.take(1):      for i in range(25):
      ax = plt.subplot(5, 5, i + 1)
plt.imshow(images[i].numpy().astype("uint8"))
plt.title(class_names[labels[i]])          plt.axis("off")
  plt.show()
```

Output:

Data Augmentation:

Sometimes the data is limited and we the model is not performing well with limited data. For this method, we use Data Augmentation. It is the method to increase the amount and diversity in data. We do not collect new data, rather we transform the already present data.

PYTHON CODE:

```
data_augmentation = tf.keras.Sequential ( [
tf.keras.layers.experimental.preprocessing.RandomFlip(              "horizontal",
input_shape=(224, 224, 3)),
      tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
tf.keras.layers.experimental.preprocessing.RandomZoom(0.2),
tf.keras.layers.experimental.preprocessing.RandomFlip(
mode="horizontal_and_vertical")])
```

Model Architecture:

The model will contain the following Layers:

- Four Convolutional Layers followed by MaxPooling Layers.
- The Flatten layer to flatten the output of the convolutional layer.
- Then we will have three fully connected Dense layers followed by the output of the of Softmax activation function.

PYTHON CODE:

```
model = Sequential()  model.add(data_augmentation)
model.add(Rescaling(1./255))  model.add(Conv2D(128, (3, 3),
activation='relu'))  model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))  model.add(Conv2D(128, (3, 3),
activation='relu'))  model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))  model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))  model.add(Dense(128,
activation='relu'))
model.add(Dense(len(labelfile), activation='softmax'))
```

PYTHON CODE:

```
model.summary()
```

Output:

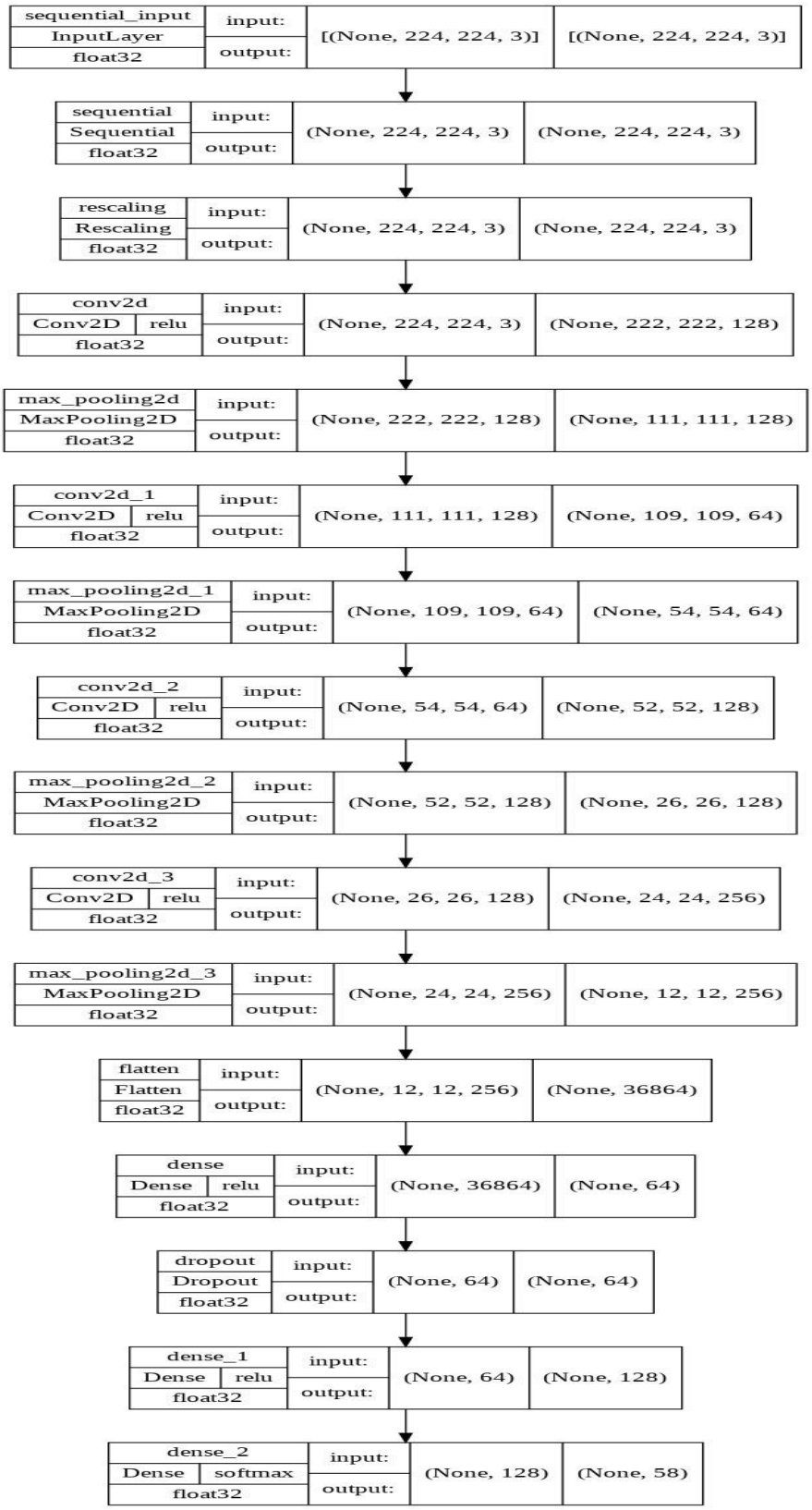
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_2 (Sequential)    (None, 224, 224, 3)       0

rescaling_1 (Rescaling)      (None, 224, 224, 3)       0

conv2d_14 (Conv2D)           (None, 222, 222, 128)     3584

max_pooling2d_12 (MaxPoolin  (None, 111, 111, 128)     0
g2D)

conv2d_15 (Conv2D)           (None, 109, 109, 64)      73792

max_pooling2d_13 (MaxPoolin  (None, 54, 54, 64)        0
g2D)

conv2d_16 (Conv2D)           (None, 52, 52, 128)       73856

max_pooling2d_14 (MaxPoolin  (None, 26, 26, 128)       0
g2D)

conv2d_17 (Conv2D)           (None, 24, 24, 256)       295168

max_pooling2d_15 (MaxPoolin  (None, 12, 12, 256)       0
g2D)

flatten_3 (Flatten)          (None, 36864)             0

dense_9 (Dense)              (None, 64)                2359360

dropout_5 (Dropout)          (None, 64)                0

dense_10 (Dense)             (None, 128)               8320

dense_11 (Dense)             (None, 58)                7482

=================================================================
Total params: 2,821,562
Trainable params: 2,821,562
Non-trainable params: 0
_________________________________________________________________
```

To understand the huge number of parameters and complexity of the model which helps us to achieve a high-performance model let's see the plot_model.

PYTHON CODE:

```
keras.utils.plot_model( model,
show_shapes=True,
show_dtype=True,
show_layer_activations=True  )
```

Output:


sequential_input
InputLayer
float32
input:
output:
[(None, 224, 224, 3)]
[(None, 224, 224, 3)]
sequential
Sequential
float32
input:
output:
(None, 224, 224, 3)
(None, 224, 224, 3)
rescaling
Rescaling
float32
input:
output:
(None, 224, 224, 3)
(None, 224, 224, 3)
conv2d
Conv2D relu
float32
input:
output:
(None, 224, 224, 3)
(None, 222, 222, 128)
max_pooling2d
MaxPooling2D
float32
input:
output:
(None, 222, 222, 128)
(None, 111, 111, 128)
conv2d_1
Conv2D relu
float32
input:
output:
(None, 111, 111, 128)
(None, 109, 109, 64)
max_pooling2d_1
MaxPooling2D
float32
input:
output:
(None, 109, 109, 64)
(None, 54, 54, 64)
conv2d_2
Conv2D relu
float32
input:
output:
(None, 54, 54, 64)
(None, 52, 52, 128)
max_pooling2d_2
MaxPooling2D
float32
input:
output:
(None, 52, 52, 128)
(None, 26, 26, 128)
conv2d_3
Conv2D relu
float32
input:
output:
(None, 26, 26, 128)
(None, 24, 24, 256)
max_pooling2d_3
MaxPooling2D
float32
input:
output:
(None, 24, 24, 256)
(None, 12, 12, 256)
flatten
Flatten
float32
input:
output:
(None, 12, 12, 256)
(None, 36864)
dense
Dense relu
float32
input:
output:
(None, 36864)
(None, 64)
dropout
Dropout
float32
input:
output:
(None, 64)
(None, 64)
dense_1
Dense relu
float32
input:
output:
(None, 64)
(None, 128)
dense_2
Dense softmax
float32
input:
output:
(None, 128)
(None, 58)

PYTHON CODE:

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
optimizer='adam', metrics=['accuracy'])
```

Model Training

Now we will train our model, the model is working fine on epochs = 50, but you can perform hyperparameter tuning for better results.

We can also use callback function for early stopping the model training.

PYTHON CODE:

```
# Set callback functions to early stop training
mycallbacks = [EarlyStopping(monitor='val_loss', patience=5)]
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=50,
                    callbacks=mycallbacks)
```

Output:

```
Epoch 34/50
105/105 [==============================] - 16s 149ms/step - loss: 0.4358 - accuracy: 0.8558 - val_loss: 0.2030 - val_accuracy: 0.9365
Epoch 35/50
105/105 [==============================] - 16s 150ms/step - loss: 0.4203 - accuracy: 0.8582 - val_loss: 0.2974 - val_accuracy: 0.9053
Epoch 36/50
105/105 [==============================] - 16s 148ms/step - loss: 0.4097 - accuracy: 0.8594 - val_loss: 0.1873 - val_accuracy: 0.9388
Epoch 37/50
105/105 [==============================] - 16s 148ms/step - loss: 0.4145 - accuracy: 0.8636 - val_loss: 0.1513 - val_accuracy: 0.9472
Epoch 38/50
105/105 [==============================] - 16s 148ms/step - loss: 0.3831 - accuracy: 0.8690 - val_loss: 0.1681 - val_accuracy: 0.9448
Epoch 39/50
105/105 [==============================] - 16s 149ms/step - loss: 0.3658 - accuracy: 0.8723 - val_loss: 0.1826 - val_accuracy: 0.9424
Epoch 40/50
105/105 [==============================] - 16s 149ms/step - loss: 0.4035 - accuracy: 0.8636 - val_loss: 0.1965 - val_accuracy: 0.9448
Epoch 41/50
105/105 [==============================] - 16s 148ms/step - loss: 0.3976 - accuracy: 0.8675 - val_loss: 0.1687 - val_accuracy: 0.9556
Epoch 42/50
105/105 [==============================] - 16s 149ms/step - loss: 0.3395 - accuracy: 0.8807 - val_loss: 0.1791 - val_accuracy: 0.9376
```

The above image shows the last epochs of the model.

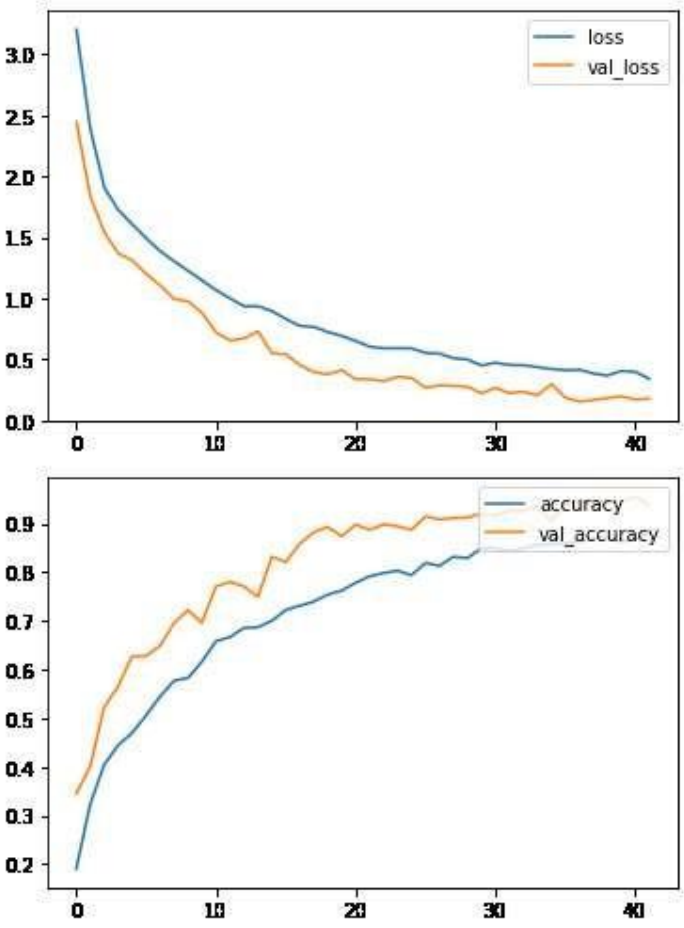
Model Evaluation:

Let’s visualize the training and validation accuracy with each epoch.

PYTHON CODE:

```
# Loss
plt.plot(history.history['loss'])  plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'], loc='upper right')
# Accuracy
plt.plot(hist.history['accuracy'])  plt.plot(hist.history['val_accuracy'])
plt.legend(['accuracy', 'val_accuracy'], loc='upper right')
```

Output:



Conclusion:

CNN model is performing very well with these layers. Further we can include more traffic and danger sign to warn the drivers.