ASSIGNMENT -2

LAMBTON COLLEGE

CBD 3335_2 DATA MINING AND ANALYSIS

TRAFFIC SIGN CLASSIFICATION

Team Members

SAI VARUN KOLLIPARA – C0828403

BHANU PRAKASH MAHADEVUNI – C0850515

DEEKSHA NAIKAP – C0835440

PRAMOD REDDY GURRALA – C0850493

Instructor: Ali Nouhi

Date: 11-08-2022

Problem Statement:

As the population increases rapidly, it is becoming difficult to monitor every moment that is happening in daily activities. But some human activities are to be observed as they will raise the risk. One of these activities is driving. Since there is a more extensive scale of automobile usage, it is becoming difficult for humans. Such advanced technologies as automated cameras will be helpful. In this case, we have built a CNN-based Image Classification Model that can vary 43 different types of traffic signs, for UI Tkinter front-end service from python is implemented.

Dataset Utilized:

This case study uses GTSRB - German Traffic Sign Recognition Benchmark image data. It is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.

- 43 Classes
- more than 50K Images
- real-time data
- contains testing and training image data and image information in CSV files.

Steps Involved:

1. Loading the Packages
2. Loading the Dataset
3. Data Pre-processing
    a. Check for errors while loading the images
    b. Resizing all the photos into similar dimensions
    c. Load the pictures into labels for further processing
4. Data Splitting
5. Model Building:
    a. CNN Model Architecture
6. Data Visualization
7. Model Evaluation
8. UI Building:
    a. UI Designing
    b. UI Functions

Prerequisites:

- Jupyter Notebook
- Numpy and Pandas - Loading and Preprocessing functions
- Sklearn - Feature Evaluation and Data Splitting

- Keras and TensorFlow - For functions of the CNN Model
- Tkinter - For building a User Interface
- Matplotlib - Visualizing the Results from Model Performance

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os

from PIL import Image
```

```
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

Image 1: Loading Packages

Data Pre-processing:

- In General, this phase prepares the data for machine learning or deep learning models. Pre-processing means all the invalid, incorrect, and outlier data will be removed from the dataset.
- Since this is image-based data, we check all the directories for loading all the pictures and check for any errors. Next, we are changing the dimensions of all the images into a standard format to load the model.
- Next, we assign and create some labels to handle the data.

```
# Open the image
image = Image.open(path + '/' + a)
# Resizes the image to 30x30
image = image.resize((30, 30))
# Turns the image into an array
image = np.array(image)
# Append the image to "data" list
data.append(image)
# Append the label to "labels" list
labels.append(i)
```

Image 2: Data Pre-Processing

```
print('Data Shape: ', data.shape)
print('Labels Shape', labels.shape)
```

```
Data Shape:  (39209, 30, 30, 3)
Labels Shape (39209,)
```

Image 3: Variables to handle the dataset

Data Splitting:

- Data Splitting is a stage where the data is decided and assigned to multiple variables that will be used in the future for training and testing the model. (This is a primary step for every machine/deep learning model building)
- The data can be split into training, Testing, and Validating.
- For our study, we use 80% for training and 20% for Testing.

```
print(X_train.shape,'|', X_test.shape,'|',y_train.shape,'|',y_test.shape)
```

```
(31367, 30, 30, 3) | (7842, 30, 30, 3) | (31367,) | (7842,)
```

Image 4: Splitting the Data and displaying the results

Model Building:

The foremost step for every machine learning / deep learning is to build a model based on the data. The data type varies the algorithm, functions, and attributes to be implemented. For most deep learning projects, we can use Neural Networks like CNN, ANN, and RNN. Since this project, we have been using image data. It is advisable to use CNN(Convolution Neural Networks).

CNN Model Architecture:

- For any Neural network, there will be three layers Input Layer, Hidden Layer, and the Output Layer. For any NN multiple layers are added to the hidden layer to train the model better. Since we are using images, CNN works best. For the architecture we have built, we used:
    - **conv2d layer** - Even though the images have three-dimensional data, we use Convolution 2D because it is used for Output purposes only. We sometimes use multiple convolutional layers to learn hierarchical representations of your input data. For example, the first layers will learn edges, and the second layers will combine these edges to form more abstract representations.
    - **MaxPool2D** - Max pooling is an operation that is added to CNN's following individual convolutional layers. When added to a model, max pooling reduces the

dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer.

- Two reasons for applying Max Pooling :
  - **Downscaling Image** by extracting an essential feature.
  - **Removing Invariances** like shift, rotational, and scale.

- **Dropout layer** - Dropout is simply a way to reduce dependencies in the Neural Network structure. It encourages each neuron to form its representation of the input data.
- **Flatten layer** - Flattening is the last step performed in a Convolution Neural Network. It involves taking the pooled feature map generated in the pooling step and transforming it into a one-dimensional vector. Flattening is done so you can feed them as inputs to the dense layer.
- **Dense layer** - The dense layer is fully connected, so all the neurons in a layer are connected to those in the next layer.
- **Epoch** means the number of iterations the entire data trains into the model.
- **Batch size**: Since the entire data cannot be fed simultaneously, we will divide the data into batches. If the batch size is 200 and there are 10,000 photos in the data, then an epoch should run 50 iterations (10,000 divided by 50).

- **2** Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- **2** Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer (256 nodes, activation="relu")
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation="softmax")

Image 5: CNN Architecture

```
history = model.fit(X_train, y_train, batch_size=64, epochs=15, validation_data=(X_test, y_test))
```

```
Epoch 1/15
491/491 [==============================] - 30s 61ms/step - loss: 2.5440 - accuracy: 0.3905 - val_loss: 0.9465 - val_accuracy:
0.8166
Epoch 2/15
491/491 [==============================] - 33s 67ms/step - loss: 1.1097 - accuracy: 0.6879 - val_loss: 0.4938 - val_accuracy:
0.8965
Epoch 3/15
491/491 [==============================] - 32s 64ms/step - loss: 0.8417 - accuracy: 0.7580 - val_loss: 0.3823 - val_accuracy:
0.9110
Epoch 4/15
491/491 [==============================] - 34s 68ms/step - loss: 0.6986 - accuracy: 0.7946 - val_loss: 0.2857 - val_accuracy:
0.9288
Epoch 5/15
491/491 [==============================] - 33s 67ms/step - loss: 0.5464 - accuracy: 0.8395 - val_loss: 0.2033 - val_accuracy:
0.9522
```

Image 6: CNN Running

Data Visualization:

- Visualizing the results will help in understanding the trends quickly. So, we always use this step for understanding or representing the data. For this case, we are using the line graphs using matplotlib. Here the plotting for accuracy and loss is done.

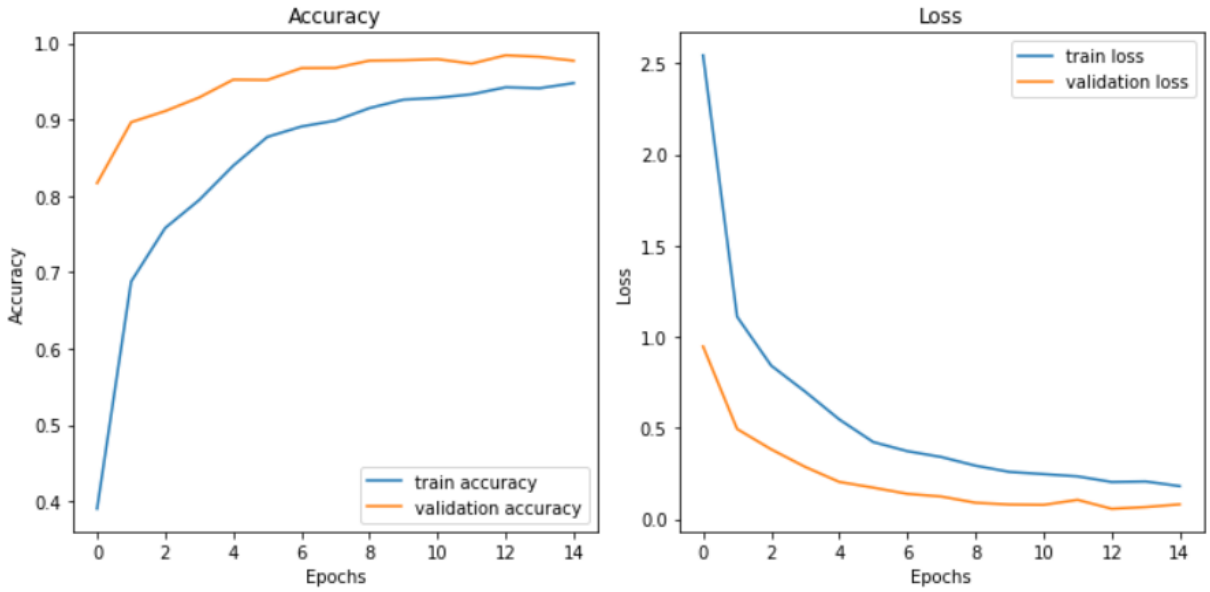

Image 7: Visualizing the Results

Model Evaluation:

- After building a machine learning / deep learning model, we can evaluate them using metrics like accuracy, loss, precision, and so on. Based on the type of algorithm, the metrics vary, but in this case, we will use accuracy for evaluating the performance of the model.
- We are using the images from the testing folder to calculate this accuracy. We will use the accuracy metrics from sklearn to identify the performance with some new pictures. For our model, we for the accuracy of 93.563%.

```
# Make predictions
predict_x = model.predict(X_test)
classes_x = np.argmax(predict_x,axis=1)
```

```
# Evaluate model
print('ACCURACY: {} %'.format(round(accuracy_score(labels, classes_x) * 100, 3)))
```

```
ACCURACY: 93.563 %
```

Image 8: Calculating the Accuracy

Model Saving:

- As we build an end-to-end application for the traffic sign classification, there will be a UI to represent the results. For building out UI, we need to save the model in h5 format and load it while building the UI.

```
model.save('traffic_classifier.h5')
```

Image 9: Model Saving

UI Designing:

- For our UI, we have selected using Tkinter Library. As per the requirement, we have added the required buttons and dimensions of the window. The steps involved are:
    - Loading the packages
    - Loading the CNN Model
    - Labeling the classes of the data
    - Set the Dimensions of the Window, Title, Font, and Button Placement.
    - Building the Functions for loading and classifying the Images.

```
heading.pack()
top.mainloop()
```

Image 10: For starting the UI Application

Image 11: UI Design

UI Functions:

- **Upload Image Function**: This function opens a window to select a picture from the local machine and saves it into variables that will be used for classify function.
- **Classify Function**: This function uses the model to make predictions from any picture. The model which is loaded previously will be utilized at this phase.

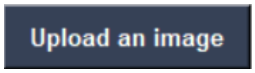

Image 12: Upload Button

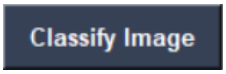

Image 13 Classify Button

Results:

Here is the output and Results from the model.

Image 14: Sample Result 1

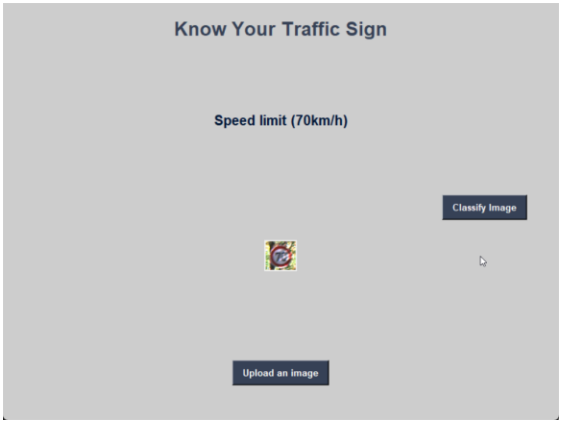

Image 15: Sample Result 2

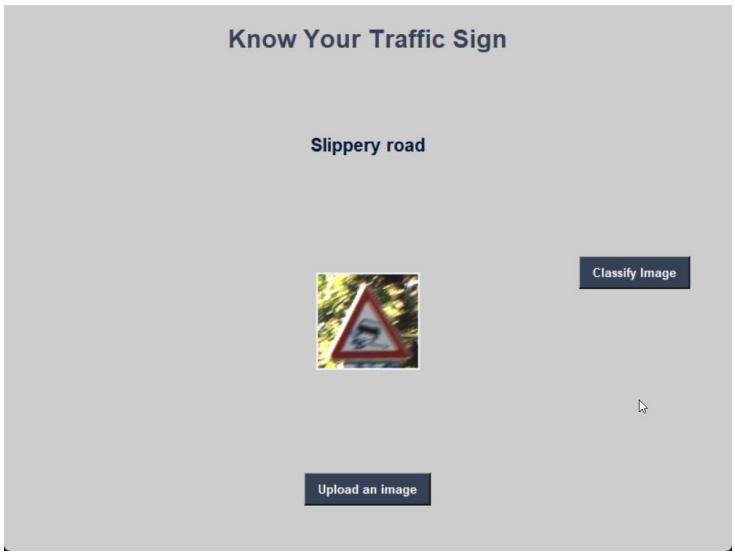

Image 16: Sample Result 3

Image 17: Sample Result 4

Conclusion:

- From this project, we conclude that CNN is suitable for handling the image type data. We also identified how to build the architecture of a CNN model to increase productivity. Coming to UI, we learned how to integrate the front-end with the back-end model.

Future Developments:

Some of the applications that can be built by using this as a base are:

- Sign Detection in Self-Driving Cars
- Automobiles that are not following the signs
- Notifying the driver regarding the upcoming signs

References:

1. https://www.quora.com/What-is-the-purpose-of-using-more-than-one-convolutional-layer-in-a-convolutional-neural-network
2. https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks
3. https://www.quora.com/What-does-a-dropout-in-neural-networks-mean
4. https://www.quora.com/What-is-the-meaning-of-flattening-step-in-a-convolutional-neural-network
5. https://www.quora.com/In-Keras-what-is-a-dense-and-a-dropout-layer
6. https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?datasetId=82373&sortBy=voteCount
7. https://data-flair.training/blogs/python-project-traffic-signs-recognition/
8. https://mksaad.wordpress.com/2020/02/07/the-difference-between-epoch-batch-and-iteration-in-deep-learning/