Republic of Tunisia

Ministry of Higher Education and Scientific Research

General Directorate of Technological Studies

Higher Institute of Technological Studies of Bizerte
Department of Electrical Engineering

Title of the project:

Traffic Signs Recognition

Prepared by:

YASSINE MATHLOUTHI

MASTER RAIA

2022/2023

Introduction:

In the first semester, we worked on developing an Auto Brake System. However, in this semester, we focused on developing Traffic Sign Recognition System in order to progress in our major project, "Autonomous Vehicle."

Traffic sign recognition is crucial for autonomous driving and driver assistance technologies. It allows vehicles to interpret and respond to the different signs they encounter on the road, ensuring they follow traffic rules and promoting safer driving. By accurately identifying and understanding traffic signs, our autonomous vehicle will have the necessary information to make smart decisions and navigate effectively in complex driving situations.

Our goal in this phase of the project is to develop an advanced traffic sign recognition system using computer vision techniques. We will utilize deep learning models and image processing algorithms to achieve high accuracy in traffic sign recognition.

In this report, we will explain how we developed the traffic sign recognition system, including collecting and preparing data. We will describe the deep learning model we chose and its different components. Additionally, we will discuss the creation of a user-friendly graphical interface using Tkinter, allowing easy interaction with the traffic sign recognition system. We will present the results of extensive testing and evaluation, assessing the system's accuracy, the classification report and the confusion matrix.

This report will provide insights into the design, implementation, and evaluation of our traffic sign recognition system, highlighting its significance within our autonomous vehicle project.

2- Data Collection, Visualization, and Preparation:

2-1 Data Collection:

We utilized the GTSRB (German Traffic Sign Recognition Benchmark) dataset, which is widely available on the Kaggle platform, for our traffic sign recognition project. This dataset is extensively used by the community working on traffic sign recognition.

The GTSRB dataset consists of thousands of real-world images capturing various classes of traffic signs, captured in Germany. It provides a comprehensive collection of traffic sign images, encompassing different shapes, colors, and symbols, making it a valuable resource for our project.

By utilizing this dataset, we gained access to a diverse range of real traffic signs, enabling us to develop a traffic sign recognition model capable of effectively processing and recognizing a wide array of commonly encountered signs on the road.

The quality and variety of data within the GTSRB dataset provided us with a robust foundation for training our model, ensuring its ability to accurately recognize traffic signs in real-world scenarios.

kaggle

2-2 Data Visualization:

Displaying Example Images of Traffic Signs:

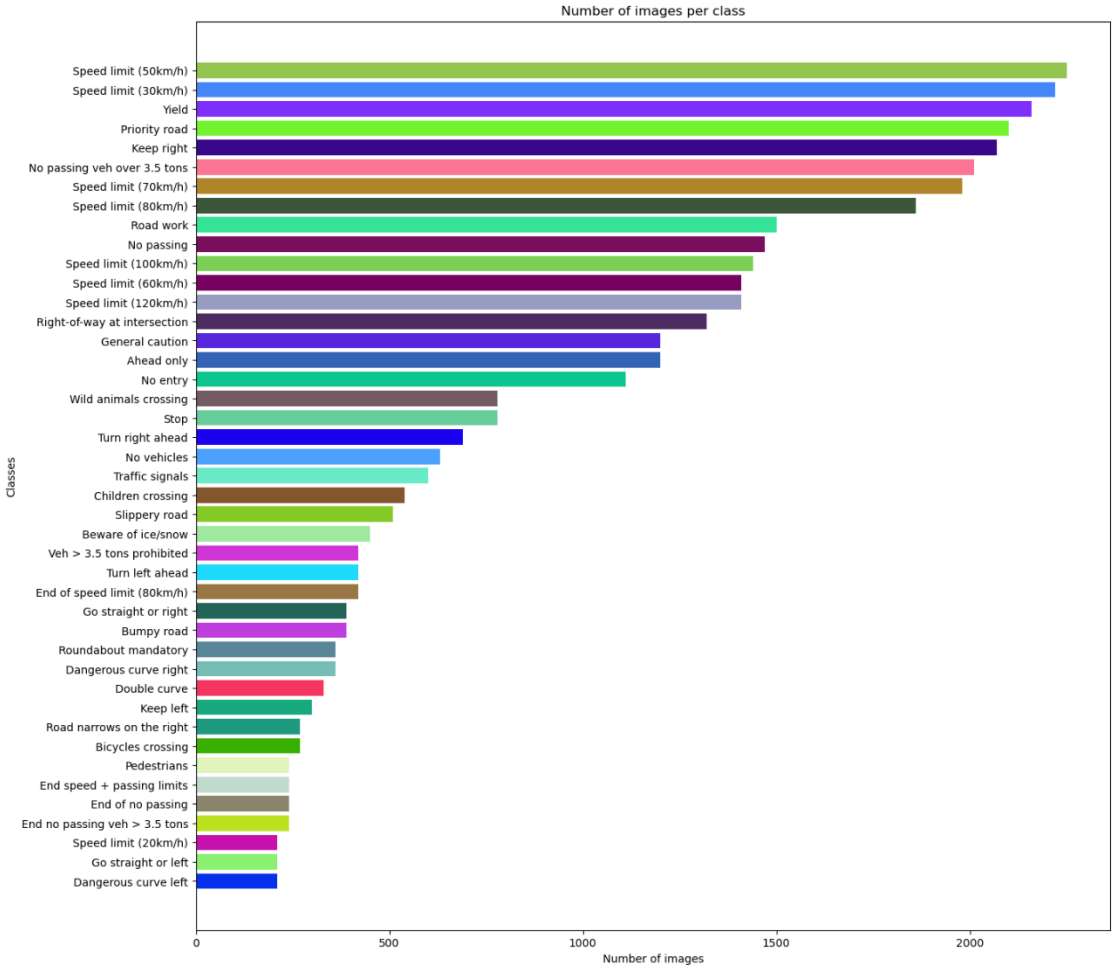
To gain a better understanding of our dataset, we performed data visualization by displaying a selection of example images of traffic signs from the GTSRB dataset. This visual representation allowed us to observe the variety of traffic sign shapes, colors, and symbols present in our dataset. By examining these examples, we could get a visual sense of the different types of traffic signs included in our dataset.

```
from matplotlib.image import imread
train = pd.read_csv(data_path + '/Train.csv')
imgs = train["Path"].values
plt.figure(figsize=(15,15))
plt.title("Some of Train images")
for i in range(1,16):
    plt.subplot(5,5,i)
    random_img_path = data_path + '/' + random.choice(imgs)
    rand_img = imread(random_img_path)
    plt.imshow(rand_img)
```

Distribution of Images per Traffic Sign Class:

In order to get an overview of the data distribution, we created a diagram illustrating the number of images per traffic sign class. This diagram provided us with a visual representation of the class distribution and helped ensure a balanced representation of the different traffic sign categories. Having a balanced distribution of images across classes is important to avoid bias in the model training. By examining this diagram, we could identify any data disparities and take necessary steps to balance the dataset if needed.

By conducting these visualizations, we gained insights into the diversity and distribution of our dataset. This allowed us to fine-tune our data preprocessing approach, select appropriate data augmentation techniques, and evaluate the performance of our model accurately.



2-3 Data Preparation:

2-3-a Extraction and organization of images by traffic sign classes:

After downloading the dataset, we extracted the images and organized them based on different traffic sign classes. This step involved grouping the images into distinct sets corresponding to each type of traffic sign. By organizing the data in this manner, we could handle the data more efficiently during subsequent steps.

2-3-b Data augmentation:

To enhance the diversity of our dataset and improve the model's ability to generalize, we applied data augmentation techniques. Specifically, we performed random rotation of the images within a range of -15 degrees to +15 degrees. Additionally, we applied random translation within a range of -10% to +10% to introduce additional variations in the position of the traffic signs. This

helped enrich our dataset with examples of traffic signs seen from different angles and positions.

```
import imgaug.augmenters as iaa

# Création de l'objet d'augmentation
augmenter = iaa.Sequential([
    iaa.Affine(rotate=(-15, 15)),  # Rotation aléatoire entre -10 et 10 degrés
    iaa.Affine(translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)})  # Translation aléatoire
])
```

We combined the data augmentation and extraction/organization of images by traffic sign classes into a single function. This function takes the initial dataset as input and performs both the data augmentation and grouping of images by their respective traffic sign classes.

The function starts by extracting the images from the initial dataset and organizes them based on their traffic sign classes. Then, it applies data augmentation techniques, such as rotation and translation, to the images of each class. This enriches the dataset with additional variations for each type of traffic sign.

By combining these two steps into a single function, we simplify the data preparation process by reducing the need for multiple separate functions. It also ensures consistency and efficiency in data processing, ensuring that all images are properly augmented and grouped by class before being used for model training.

```
from PIL import Image


for i in range(len(folders)):
    path = os.path.join(data_path, 'train', str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30, 30))
            image = np.array(image)
            data.append(image)
            labels.append(i)

            # Appliquer la data augmentation à l'image
            augmented_image = augmenter(image=image)

            data.append(augmented_image)
            labels.append(i)

        except:
            print("Error loading image")
```

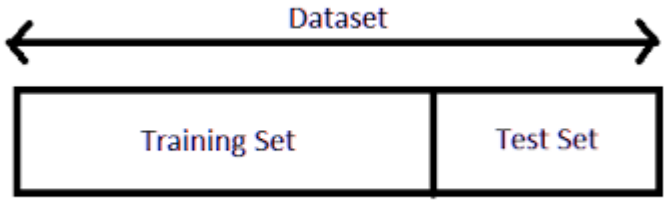
After this function we needed to convert the data and labels into NumPy arrays allows us to benefit from the array's efficient and optimized operations for numerical computations. It provides us with additional functionalities and ease of manipulation when working with the data.

By converting the data and labels into NumPy arrays, we can leverage various array-based operations and functions provided by the NumPy library. This

includes performing mathematical operations, reshaping the data, and accessing specific elements or subsets of the data with ease.

2-3-c Division of the dataset for training and testing:

In order to evaluate the performance of our model, we divided the dataset into distinct sets for training and testing, 80% for the training set. During this division, we ensured that each set contained a representative sample from each traffic sign class. This ensures that the model is trained and evaluated on the data distribution representing all traffic sign classes, thereby avoiding potential biases.



After splitting the data into training and testing sets, we need to encode the target variable into categorical format.

Encoding the target variable into categorical format is necessary when dealing with classification tasks, such as traffic sign recognition. Categorical encoding converts the target variable from its original form (class labels) into a binary matrix representation, where each class is represented by a binary vector.

By encoding the target variable in categorical format, we ensure that the model can appropriately interpret and classify the traffic sign classes during training and evaluation. This encoding step is crucial for accurate performance assessment and prediction in classification tasks.

2-3-d Choice of model:

For the traffic sign recognition project, several models can be considered, such as linear classifiers, decision tree-based models, support vector machines (SVM), and convolutional neural networks (CNN).

Among these options, CNN is generally regarded as one of the most powerful choices for computer vision tasks, including traffic sign recognition. Here are some arguments in favor of choosing a CNN:

1. Ability to learn hierarchical features: CNNs are specifically designed to automatically extract meaningful features from images. They can learn complex patterns and structures by capturing information at different levels of abstraction. This makes them well-suited for traffic sign recognition, which requires detecting diverse visual patterns.

2. Translation and deformation invariance: CNNs employ convolutional layers that allow feature detection irrespective of their precise position in the image. This makes CNN models robust to translations and deformations, which is essential for recognizing traffic signs that may appear at different positions and angles.

3. Ability to handle large volumes of data: CNN models are efficient at processing large amounts of data. In the case of traffic sign recognition, where the dataset can contain thousands of images, CNNs can handle these data volumes and effectively learn from them.

4. High classification accuracy: CNNs have demonstrated the ability to achieve high performance in image classification. They have been successfully used in various traffic sign recognition competitions, often achieving high levels of accuracy.

3- CNN (Convolutional Neural Network)

A Convolutional Neural Network (CNN) is a widely used deep learning model in the field of computer vision. It is specifically designed to process image data and has demonstrated excellent performance in tasks such as object recognition, object detection, and traffic sign recognition.

3-1 CNN Architecture:

A Convolutional Neural Network (CNN) is composed of multiple layers that work together to extract and classify visual information from images. Here is a general description of the CNN architecture:

3-1-1 Input Layer:

The input layer receives the images as input data for the network.

Each image is represented as a multidimensional tensor (typically 3D for color images or grayscale).

3-1-2 Convolutional Layers:

Convolutional layers are the essential building blocks of a CNN.

Each convolutional layer applies filters (kernels) to local parts of the image to extract specific features.

Each filter moves across the entire image, calculating dot products with the corresponding pixels to produce a feature map.

3-1-3 Activation Functions:

Activation functions introduce non-linearity into the network.

They are applied after each convolution operation to introduce thresholds and enable the model to capture non-linear relationships between features.

3-1-4 Pooling Layers:

Pooling layers reduce the spatial dimension of the feature maps.

They aggregate information by keeping only the most important values (e.g., maximum value in a region) to reduce data size and computational complexity.

3-1-5 Fully Connected Layers:

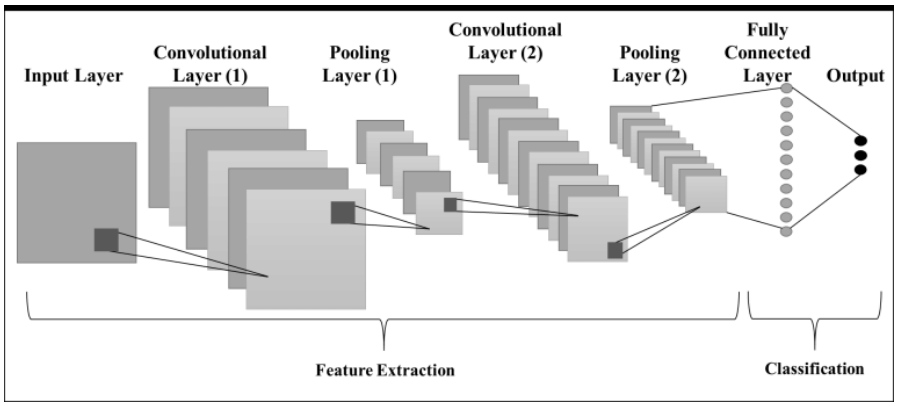
Fully connected layers transform the extracted features into a representation suitable for the classification task.

Each neuron in a fully connected layer is connected to all neurons in the previous layer, allowing for complex combinations of features.

3-1-6 Loss Function and Optimization:

The loss function measures the discrepancy between the model's predictions and the ground truth labels.

The optimizer adjusts the network's weights to minimize the loss function, using algorithms such as stochastic gradient descent (SGD) or Adam.



3-2 Our CNN model:

Our CNN model consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Let's break down the different layers and discuss the hyperparameters chosen for the model:

Convolutional Layers:

The first convolutional layer has 32 filters with a kernel size of (3,3) and uses the Leaky ReLU activation function.

The second convolutional layer has 64 filters with a kernel size of (3,3) and also uses the Leaky ReLU activation function.

The third convolutional layer has 128 filters with a kernel size of (3,3) and uses the Leaky ReLU activation function.

The fourth convolutional layer has 256 filters with a kernel size of (3,3) and also uses the Leaky ReLU activation function.

The fifth convolutional layer has 512 filters with a kernel size of (3,3) and uses the Leaky ReLU activation function.

Pooling Layers:

MaxPooling2D layers with a pool size of (2,2) are used after the second and fourth convolutional layers. These layers downsample the feature maps, reducing their spatial dimensions.

Dropout Layers:

Dropout layers are inserted after the first, second and third pooling layers, with a dropout rate of 0.15, 0.2 and 0.3, respectively. Dropout helps prevent overfitting by randomly deactivating neurons during training.

Flatten Layer:

The Flatten layer is used to convert the 2D feature maps into a 1D vector, preparing the data for the fully connected layers.

Fully Connected Layers:

Two fully connected dense layers with 512 and 256 units, respectively, are added after the Flatten layer. They use the leaky ReLU activation function and apply L2 regularization with a strength of 0.1.

Dropout layers with a dropout rate of 0.5 are added after each fully connected layer to further prevent overfitting.

Output Layer:

The final dense layer consists of 43 units, representing the 43 different classes of traffic signs in the dataset. It uses the softmax activation function to produce class probabilities.

Optimizer, loss function and metrics:

The optimizer chosen is Adam with a learning rate of 0.001, specified by optimizer = Adam(lr=0.001). Adam is a popular optimizer that combines the benefits of both the RMSprop and AdaGrad algorithms.

The model is compiled using the categorical cross-entropy loss function, which is suitable for multi-class classification tasks. This is specified by loss='categorical_crossentropy'. Additionally, the metrics used for evaluation during training are accuracy, specified by metrics=['accuracy'].

With the optimizer and loss function configured, the model is ready for training. The training process would involve passing the training dataset, along with the corresponding labels, to the model's fit() function. The number of epochs, batch size, and other training parameters can be defined based on the specific requirements of the project.

By compiling the model with the appropriate optimizer, loss function, and metrics, you have set the foundation for training the CNN model for traffic sign recognition.

3-2-1 Model summary:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, 30, 30, 3)        12
 ormalization)

 batch_normalization_1 (Batc  (None, 30, 30, 3)        12
 hNormalization)

 conv2d (Conv2D)             (None, 28, 28, 32)        896

 batch_normalization_2 (Batc  (None, 28, 28, 32)       128
 hNormalization)

 conv2d_1 (Conv2D)           (None, 26, 26, 64)        18496

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 dropout (Dropout)           (None, 13, 13, 64)        0

 conv2d_2 (Conv2D)           (None, 11, 11, 128)       73856

 batch_normalization_3 (Batc  (None, 11, 11, 128)      512
 hNormalization)

 conv2d_3 (Conv2D)           (None, 9, 9, 256)         295168

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 256)         0

 conv2d_4 (Conv2D)           (None, 2, 2, 512)         1180160

 batch_normalization_4 (Batc  (None, 2, 2, 512)        2048
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 1, 1, 512)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 512)               262656

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_4 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 43)                11051

=================================================================
Total params: 1,976,323
Trainable params: 1,974,967
Non-trainable params: 1,356
```

3-3 Model training:

The History <<HISTORY = model.fit(X_train, y_train, batch_size=32, epochs=25, validation_data=(X_test, y_test))>> contains the model's training history. This history records the metrics calculated during model training in each epoch.

the parameters used during model training:

- X_train and y_train: The training data, where X_train contains the features of the training samples and y_train contains the corresponding labels.
- batch_size: The batch size used during model training. It specifies the number of samples that are propagated through the network before updating the model weights.
- epochs: The number of epochs, i.e., the total number of times the entire data set is passed through the model during training.
- validation_data: The validation data used to evaluate the model's performance after each epoch. It consists of X_test and y_test, where X_test contains the features of the validation samples and y_test contains the corresponding labels.

Using these parameters, the model was trained for a certain number of epochs (25) with 32 samples in each batch. The training history records the loss and accuracy metrics for the training set and the validation set at each epoch. This allows tracking the model's performance over time.

3-4 Model Evaluation:

To evaluate the performance of the traffic sign classification model, we can use the test dataset. This dataset contains images that the model hasn't seen during training, allowing us to assess its generalization ability.

3-4-1 Accuracy:

The accuracy metric measures the percentage of correctly classified traffic signs from the total number of test samples. It provides an overall performance measure of the model.

The accuracy of our model is 98.2%, This high accuracy indicates that the model is performing well in distinguishing and recognizing different traffic sign classes.

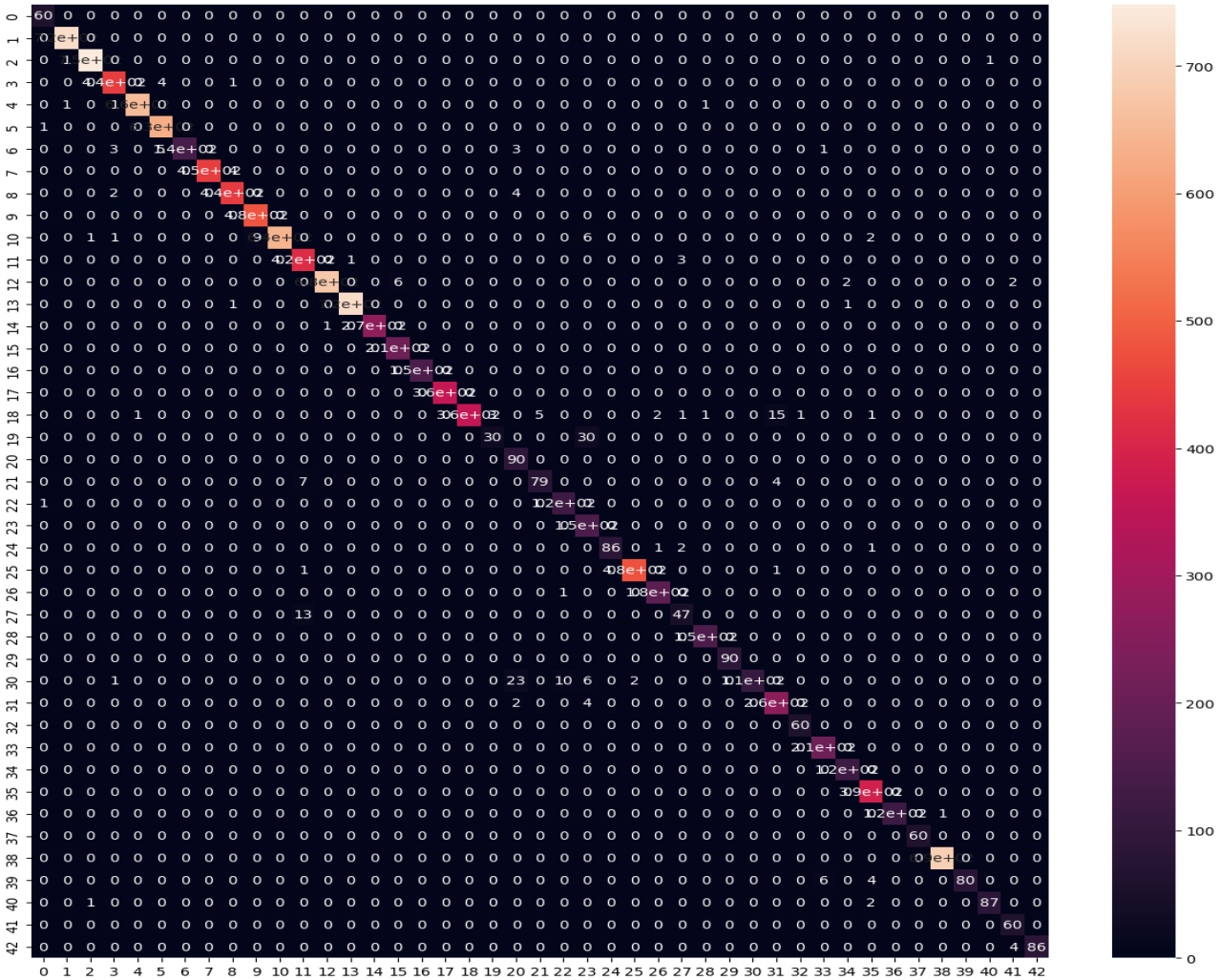
the model demonstrates a high level of precision in its predictions. This means that it is able to correctly classify the majority of the traffic signs it encounters. The model's ability to achieve such a high accuracy suggests that it has learned meaningful features and patterns from the training data, enabling it to make accurate predictions on unseen test images.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(label, y_pred))
```

```
0.9820269200316706
```

3-4-2 Confusion Matrix:

A confusion matrix is a tabular representation that shows the predicted labels against the actual labels. It provides insights into the model's performance for each class, highlighting which classes are frequently misclassified.



3-4-3 Classification Report:

A classification report provides precision, recall, F1-score, and support for each class. It gives a more detailed evaluation of the model's performance for individual classes:

- Accuracy: The accuracy of 0.98 indicates that the model correctly classified 98% of the traffic sign images in the test dataset.

- Macro average: The macro average represents the average performance across all classes, giving equal weight to each class. In this case, the macro average precision, recall, and F1-score are all around 0.96. This indicates that, on average, the model has achieved a high level of precision, recall, and overall performance across the different traffic sign classes.

- Weighted average: The weighted average takes into account the class imbalance in the dataset by weighting each class's performance based on the number of samples in that class. The weighted average precision, recall, and F1-score are all around 0.98. This suggests that the model has performed consistently well across all classes, considering the varying number of samples in each class.

- Overall, the classification report confirms that the model has achieved a high level of accuracy and performance in classifying the traffic sign images. The consistent high precision, recall, and F1-score values indicate that the model is robust and effective in recognizing and categorizing various traffic sign classes.

```
from sklearn.metrics import classification_report
print(classification_report(label, y_pred))
```

	precision	recall	f1-score	support
0	0.97	1.00	0.98	60
1	1.00	1.00	1.00	720
2	1.00	1.00	1.00	750
3	0.98	0.99	0.99	450
4	1.00	1.00	1.00	660
5	0.99	1.00	0.99	630
6	1.00	0.92	0.96	150
7	1.00	0.99	1.00	450
8	0.99	0.99	0.99	450
9	0.98	1.00	0.99	480
10	1.00	0.97	0.99	660
11	0.95	0.99	0.97	420
12	1.00	0.99	0.99	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	270
15	0.97	1.00	0.99	210
16	1.00	1.00	1.00	150
17	1.00	1.00	1.00	360
18	1.00	0.92	0.96	390
19	0.91	0.50	0.65	60
20	0.74	1.00	0.85	90
21	0.94	0.88	0.91	90
22	0.92	0.99	0.95	120
23	0.77	1.00	0.87	150
24	1.00	0.96	0.98	90
25	1.00	1.00	1.00	480
26	0.98	0.99	0.99	180
27	0.89	0.78	0.83	60
28	0.99	1.00	0.99	150
29	1.00	1.00	1.00	90
30	1.00	0.72	0.84	150
31	0.93	0.98	0.95	270
32	0.98	1.00	0.99	60
33	0.97	1.00	0.98	210
34	0.98	1.00	0.99	120
35	0.97	1.00	0.99	390
36	1.00	0.99	1.00	120
37	1.00	1.00	1.00	60
38	1.00	1.00	1.00	690
39	1.00	0.89	0.94	90
40	0.99	0.97	0.98	90
41	0.91	1.00	0.95	60
42	1.00	0.96	0.98	90
accuracy			0.98	12630
macro avg	0.97	0.96	0.96	12630
weighted avg	0.98	0.98	0.98	12630

4- Traffic Signs Recognition GUI:

When the user clicks the "Upload an image" button, a file dialog opens for selecting an image. Once the image is selected, it is resized and displayed in the GUI window.

Next, the "Classify Image" button is shown. When the user clicks this button, the classify() function is called with the file path of the selected image. In the classify() function, the image is preprocessed by resizing it, adding an extra dimension, and converting it to a NumPy array.

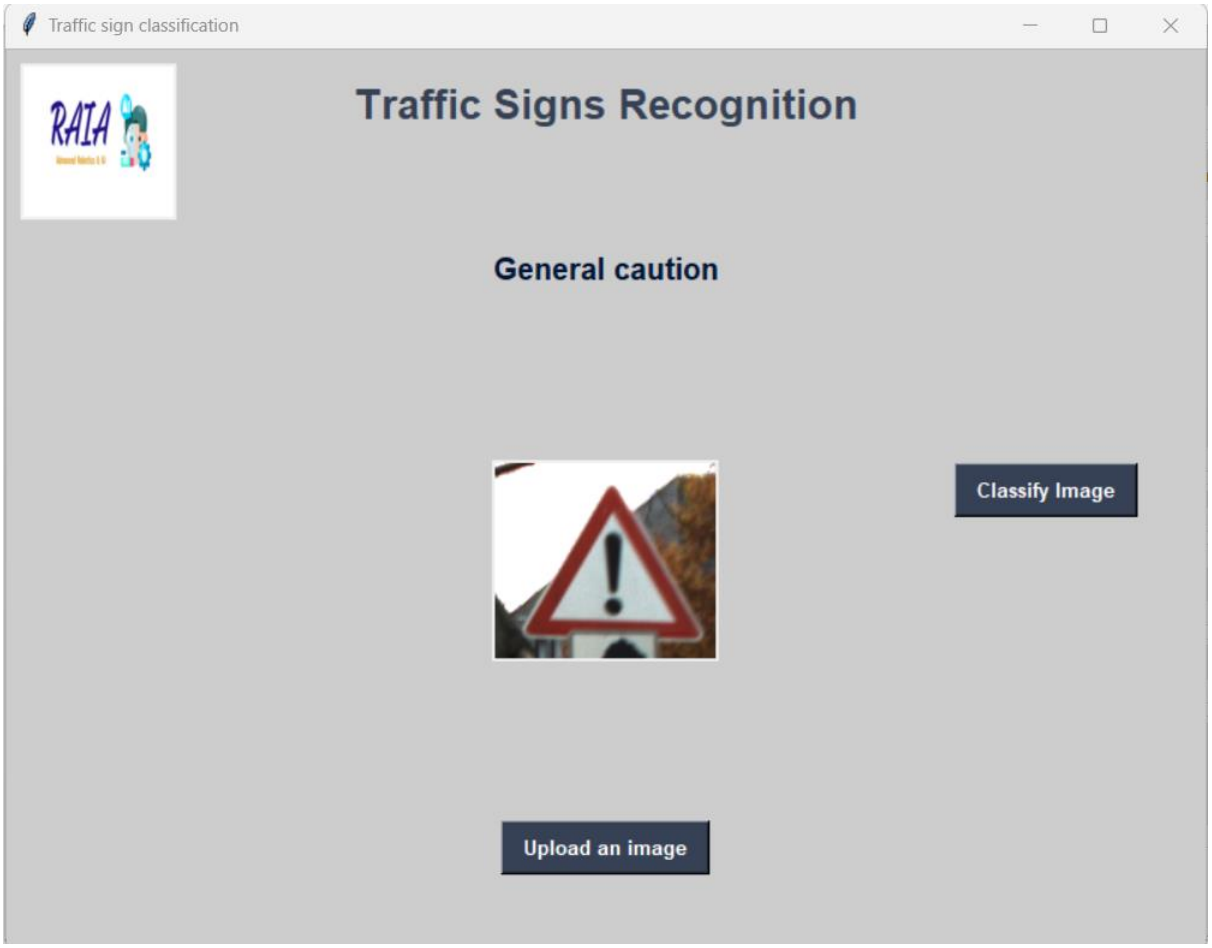
Then, the model is used to predict the class of the traffic sign in the image. The predicted class is obtained by taking the index of the maximum value in the model's prediction and adjusting it by 1 to match the class numbering in the

classes dictionary. The corresponding name for the predicted class is then extracted from the classes dictionary.

Finally, the text of the predicted class is displayed in the GUI using the label label.

The GUI also includes a heading "Traffic Signs Recognition" and a logo "logo raia.png" displayed in the top-left corner.

The user can upload a new image by clicking the "Upload an image" button and obtain the classification of the traffic sign by clicking the "Classify Image" button.



5- Conlusion:

In conclusion, this project focused on developing a traffic sign recognition system using deep learning techniques. The project involved various stages, including data collection, data preprocessing, model training, and evaluation.

The dataset used for training and testing was the GTSRB dataset, which contains real-world images of traffic signs captured in Germany. The data was prepared by organizing the images into different classes and applying data augmentation techniques such as rotation and translation to increase the diversity of the training data.

For the model architecture, a Convolutional Neural Network (CNN) was chosen. The CNN model consisted of multiple convolutional layers, pooling layers, and fully connected layers. Hyperparameters such as kernel size, number of filters, activation functions, and regularization techniques were carefully selected to optimize the model's performance.

The trained model achieved an impressive accuracy of 98.2% on the test dataset, indicating its ability to accurately classify traffic sign images. The classification report further confirmed the model's high precision, recall, and F1-score across different traffic sign classes, demonstrating its effectiveness in recognizing various signs.

Overall, this project successfully demonstrated the feasibility and effectiveness of using deep learning techniques, specifically CNNs, for traffic sign recognition. The developed model shows great potential for implementation in autonomous driving systems or driver assistance systems, where accurate and real-time recognition of traffic signs is crucial for ensuring road safety. Further improvements and optimizations can be explored to enhance the model's performance and expand its capabilities in handling more complex traffic scenarios.