FACULTY OF COMPUTING & INFORMATION TECHNOLOGY
KING ABDULAZIZ UNIVERSITY

FCIT
KAU

كلية الحاسبات وتقنية المعلومات
جامعة الملك عبدالعزيز

# Traffic Signs Classification

Artificial Intelligence (II)
CPCS-432

| **Team Members:** Group No.3 | |
|---|---|
| **Name** | **Section** |
| Sumayah Ahmed Alkathiri | EAR |
| Hadeel Saeed ALoufi | |
| Aya Mohamad Kazzaz | |
| Razan Mohammed Aljohani | |

***May 2022***

## Task Assignment

| Name | Task | Percentage% |
|---|---|---|
| Sumayah Ahmed Alkathiri | • Dataset analysis.<br>• Code implementation and analysis using python.<br>• Code evaluation.<br>• Report documentation. | 25% for each task. |
| Hadeel Saeed Aloufi | • Dataset analysis.<br>• Code implementation and analysis using python.<br>• Code evaluation.<br>• Report documentation. | 25% for each task. |
| Aya Mohamad Kazzaz | • Dataset analysis.<br>• Code implementation and analysis using python.<br>• Code evaluation.<br>• Report documentation. | 25% for each task. |
| Razan Mohammed Aljohani | • Dataset analysis.<br>• Code implementation and analysis using python.<br>• Code evaluation.<br>• Report documentation. | 25% for each task. |

## Summary

Traffic signs displayed on the roads play an important role in our lives while driving. They provide crucial information to road users. As a result, they must regulate their driving behavior to ensure that they strictly follow the road regulations currently enforced without causing any trouble to other drivers and pedestrians. Also, with the advent of autonomous vehicles, it has become important to have an automated system for traffic sign classification that helps these vehicles to recognize and understand traffic signs in order to prevent accidents and regulate the traffic on roads. Traffic Sign Classification is employed to classify traffic signs to inform and warn a driver or autonomous vehicles beforehand to avoid violation of rules. The proposed approach implements a traffic signs classification algorithm employing a convolutional neural network. The pre-trained model EfficientNetB0 is used in our context as a base model. After implementing and deploying the model, we achieved 96.61% a testing accuracy.

## Table of Contents

## List of Figures

## List of Tables

## 1. Introduction

Artificial intelligence techniques have proven their efficiency in various fields. The process of automatically recognizing traffic indicators along the road, such as speed limit signs, no car, no horn, and so on, is known as traffic sign classification. Traffic sign classification. It is an established fact that traffic sign classification is one of the most important aspects to the system of autonomous vehicles as they help prevent accidents and regulate the traffic on roads. Similarly, "driver alert" systems in cars must understand the road environment in order to assist and protect drivers. As soon as driver-less vehicles came into play a couple of decades ago, traffic sign classification has been a hot topic, with a lot of research going into establishing an optimal model. Initially, traditional computer vision and machine learning approaches were initially employed to implement the recognition model, but seeing as they did not have enough accuracy and efficiency, they were soon replaced by deep learning-based classification methods. In this project we will talk about traffic sign classification using deep learning-based classification (CNN) and transfer learning. Our proposed model achieved 98.37% accuracy for training and 96.61% for testing. In this project we will talk about research related to our work, our datasets, our model, experiments and results discussion.

## 2. Related work

Throughout our research in the field, we found that many studies utilized deep learning technique in traffic sign classification.

1. **Sreya et al**. Proposed CNN based model for traffic sign classification is proposed to increase the accuracy of the classification model which consists of 4 convolutional layers and 2 max pooling layers. They did data pre-processing such as compress/interpolate the images to a single dimension [32*32*1]. They divided dataset into 60:20:20 ratio as training, validation, test dataset respectively. After that they fed the dataset to CNN model. Also, they used LeNet as base model. A popular dataset German traffic signs detection dataset (GTSRB) dataset is used, with 39209 images and 43 different classes. They achieved 90.07% accuracy. (Sreya, 2021)
2. **Chaudhari et a**l. Proposed an effective solution to detect traffic signs on road by first classifying the traffic sign images using Convolutional Neural Network (CNN) with

different dimensional filters like 3 × 3, 5 × 5, 9 × 9, 13 × 13, 15 × 15,19 × 19, 23 × 23, 25 × 25 and 31 ×31 from which the most efficient filter is chosen to classifying the image detected. The CNN model has one convolutional layer with 32 filters, ReLU activation function, one down sampling layer with 2 × 2 maximum factor, and hidden affine layer with 500 neurons followed by the output layer with 43 neurons as number of classes The detection involves detecting the traffic sign using YOLO v3-v4 and BLOB detection. A popular dataset German traffic signs detection dataset (GTSRB) dataset is used, with 39209 images and 43 different classes. They achieved 87% accuracy. (Manjiri Bichkar, Suyasha Bobhate, Prof. Sonal Chaudhari, 2021)

3. **Preeti Bailke et al**. They proposed system that uses Convolutional Neural Network (CNN) which consists of two convolutional layers, one pooling layer, dropout layer, flattening layer, dense layer, again a dropout layer and finally the dense layer. . GTSRB dataset used for both detection as well as classification. A popular dataset German traffic signs detection dataset (GTSRB) dataset is used and they divided it into training, testing and validation dataset. The train, test and validation split percentage are 65%, 25% and 10% respectively for the proposed system. Also, small dataset (consisting of 13 images) is built for testing the performance of the proposed model. They experimented with different hyper parameters such as (the number of epochs, the choice of activation function). The images are pre-processed by converting the RGB data set into gray scale, and histogram equalization is done to enhance the image contrast. The final accuracy is 93% and 69% on GTSRB and generated dataset respectively. (Preeti Bailke, Kunjal Agrawal, 2022)

## 3. Technical Description

Deep learning is a type of artificial intelligence (AI) approaches, that enables systems to learn from experience, it can extract unclear features from complicated data such as images and train the model effectively using multiple layers. It is extremely beneficial for collecting, analyzing, and predictive modeling big data. The Coevolutionary Neural Network (CNN) is an efficient deep learning technique that can extract attributes of a set of images and classify them. (Ian Goodfellow, 2016) (Ian Goodfellow, 2016)

## 3.1. Dataset

We used Traffic Sign Dataset - Classification from the Kaggle website (Hemateja, 2021) This dataset contains the different classes of traffic signs such as (No Uturn, No Car and Don't Go Right, etc.) and has around 49 classes and each class has around 120 images. Also, the (labels.csv file) has the respective description of the traffic sign class. We did some preprocessing technique in order to modify some features for all data, such as (rotation range, width shift range, height shift range, fill mode, shear range and zoom range). We applied customized partitioning when we divided the dataset into training, testing and validation. The reason of applying customized partitioning is that some labels contain very few image instances. If we apply random partitioning to training and validation, there is a chance that the training set will not reflect the entire general population. So, we split the training and validation data only for the part where the number of image instances is higher than SPLIT_MINIMUM_COUNT which is equal to 10.

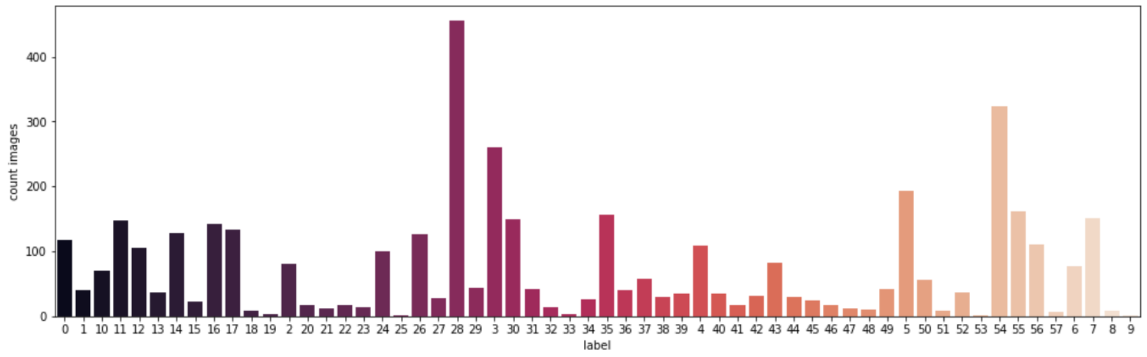


Figure 1: The Number of Images per Each Label

## 3.2. Methods

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with a grid-like topology. It is used to detect and classify objects in an image. It has multiple hidden layers that help extract information from an image. (Biswal, 2022) We chose CNN because, based on our research, this problem can be solved by CNN and get high accuracy. EfficientNet-b0 is a convolutional neural network trained on more than 1 million images from the ImageNet database. (EfficientNet-b0 convolutional neural network, 2022) (EfficientNet-b0 convolutional neural network, 2022) We used the EfficientNetB0 as the base model trained on the ImageNet dataset. We entered the data into the base model, and then added three layers: The first layer is Batch normalization. Batch normalization is a technique for training deep neural networks that standardizes the inputs to a layer for each mini-batch. It stabilizes the learning process and dramatically reduces the number of training epochs required to train deep networks. (Brownlee, 2019) The second layer is the dense layer containing 512 neurons and activation function (relu). The last layer is the output layer containing 49 neurons with the same number of classes and the activation function (softmax) used for multiclass classification.

## 4. Experimental Results and Results Discussion

This section will show the evaluation methods we used, explain them, and show the experiments we applied to the model. In the last, we will present the results in detail.

### 4.1. Evaluation Method

Evaluating the deep learning model plays a critical role in any project, and a selection of suitable evaluation metrics is an important key to determining the optimal classifier.  Accuracy is employed by our project as a metric measure to distinguish the optimal solution. It measures the ratio of correct predictions over the total number of instances evaluated. (Hossin, Sulaiman, 2015)

Figure 2 illustrates the accuracy graph of the training and validation accuracy, which shows the accuracy of the model in classifying the training dataset during the training, and the validation dataset during the validation. In the graph, the validation accuracy is slightly higher than training accuracy, so there's a high probability that the model is overfitted. The validation accuracy should always be less than the training accuracy; because the model is more familiar with the training dataset, whereas the validation dataset is a set of data new to the model. The training accuracy and validation accuracy are growing on each epoch and then begin to stabilize around 98%, which means that the model reaches a better level and does not need additional iterations.

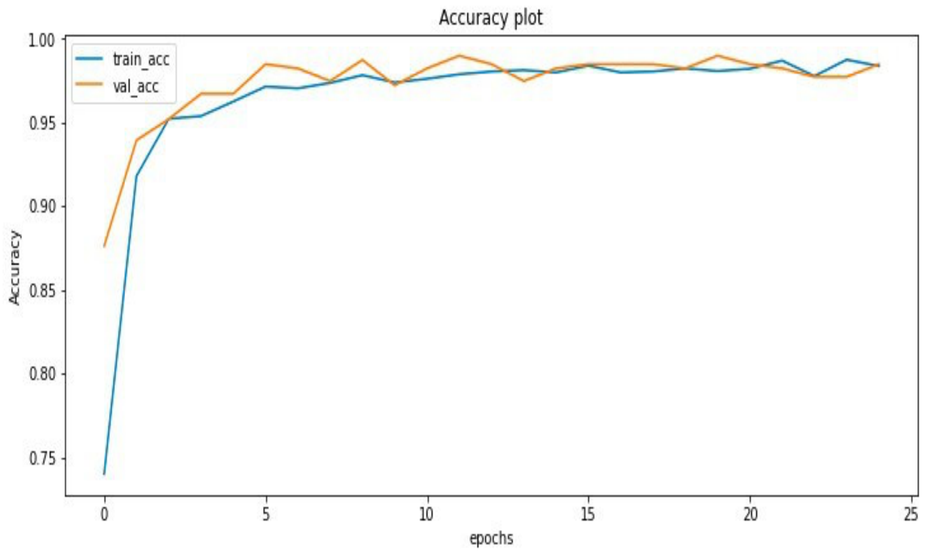


Figure 2: Accuracy Plot

On another side, the loss decreases for each iteration as shown in Figure 3, which indicates that the model works correctly with better performance.

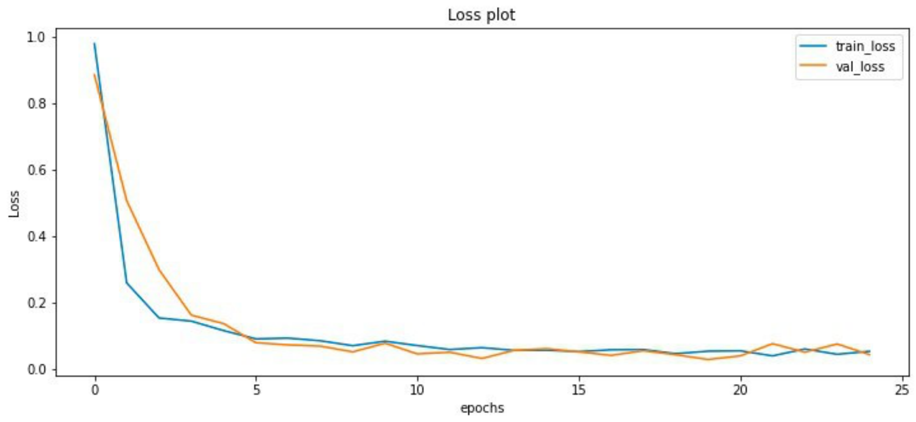


Figure 3: Loss Plot

Our model achieved a high test accuracy equal to 96.61% which indicates it gives accurate results. However, the model can be improved by reducing the bias and the variance. (Murphy, 2012)

## 4.2. Experiments

In this section, we will perform some experiments on the model to make sure that it serves its purpose to the fullest and solves the problem without any shortcomings. We will perform four experiments on the model, which are: compare the model with the previously published work, execute the model on a different number of epochs, test the model on different splitting of the dataset, and execute the model on different optimizer types.

### 4.2.1. Comparison with Previously Published Work

In comparing with the previse work that was clarified in section 2, many researchers tried to establish an optimal model for the traffic sign classification, and they reached solutions that get the benefit of a CNN algorithm, and they achieved an acceptable performance with good accuracy but that was not enough. However, our project utilizes the advantages of the CNN algorithm and trained the model with the big dataset to get an improvement in the model accuracy as clarified in the following points:

1. It achieved 98.37% accuracy for training and 96.61% for testing.
2. It used EfficientNet-b0, a convolutional neural network trained on more than 1 million images from the ImageNet database.
3. It applies three layers, batch normalization layer that reduces the number of training epochs required to train deep networks. And the dense layer that activates the Relu function, as well as the output layer that activates the Softmax function used for multiclass classification.

So, our model achieved a positive difference in accuracy and better performance because of effectively using A convolutional neural network.

### 4.2.2. Different Number of Epochs

Epoch in machine learning refer to the number of times the training dataset passes to the machine learning model. (Bell, 2020) So, one epoch indicates one cycle of training the dataset on the model, so a forward and a backward pass will be considered as one pass. (Epoch in Neural Networks, 2021)

In this experiment, we implemented the model on different numbers of epochs and noticed the difference in the accuracy of the results.
Table 1 shows the result of the experiment. When the number of epochs increases, the model accuracy increases as well, but in the last three epochs (10, 12, and 14), the accuracy didn't increase that much, which indicates that the suitable number of epochs is in the range from 10 to 14.

Table 1: The Accuracy on Different Number of The Epochs

| No. of Epochs | Accuracy |
|---|---|
| 2 | 81.02% |
| 4 | 83.33% |
| 6 | 86.98% |
| 8 | 93.13% |
| 10 | 97.16% |
| 12 | 98.87% |
| 14 | 98.89% |

### 4.2.3. Different Splitting of the Dataset

Sometimes splitting the dataset with different rations could lead to different results, and that depends on the dataset content.
Since our dataset is unbalanced, in this experiment we will try different splitting of the data, to see if that will affect the accuracy of the model or not. (Kumar, 2020) Different splitting will lead to different content of the data in the train and test sets.
Table 2 shows that the differences in the accuracy are not huge with the changing the contents of the data in each split, which indicates that the accuracy is correct for the model even with different splitting. Also, the average of the accuracy on all the trials is 97.33% which is close to all the accuracies on different trials.

Table 2: The Accuracy of Different Numbers of Splitting Trials

| No. of Trials | Accuracy |
|---|---|
| 1 | 97.76% |
| 2 | 96.08% |
| 3 | 97.36% |
| 4 | 98.11% |

#### 4.2.4. Different Optimizer Types

The optimizer in machine learning indicates the function that leads to modify the neural network attributes (e.g., learning rate and weight). So, it is helpful in reducing the loss and improving the accuracy. (Gupta, 2021)

The choosing of the right optimizer is based on the purpose of the machine learning model. So, in the last experiment, we will be trying different optimizers to see the suitable one for our model.

Table 3 shows that Adam is the best optimizer for our model based on the accuracy. Also, Adam is considered as the best optimization algorithm; because it requires fewer tunning parameters and is faster in the computation. (Gupta, 2021)

Table 3: The Accuracy of Different Optimizers

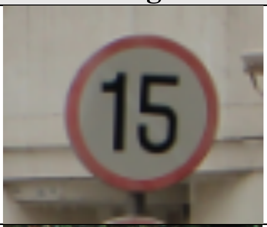
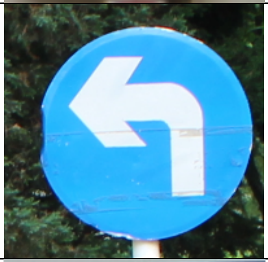
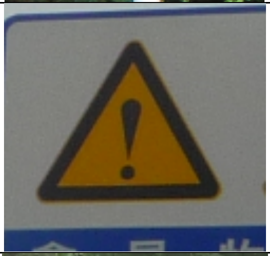
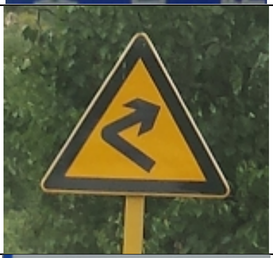
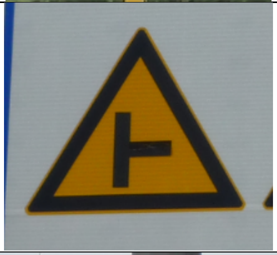
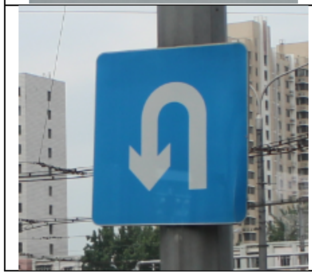
| Optimizer | Accuracy |
|---|---|
| Adam | 97.76% |
| Gradient Descent | 82.23% |
| AdaDelta | 78.89% |

### 4.3. Results and Results Discussion

In this section we will show the results of our model by predicting the class of the traffic sign images of the testing dataset, explain the accuracy plot, and compare the results with the previous work.

We test the model by entering different images of the traffic signs, and the model should predict the classes of the signs. Table 4 shows the entered image and the prediction of each one. The green color indicates correct prediction, whereas the red color indicates the wrong prediction.

Table 4: The Results of the Model Prediction

| Image | Prediction |
|---|---|
|  | `tf.Tensor([1], shape=(1,), dtype=int64)`<br>Class: **1 (Speed limit (15km/h))** |
|  | `tf.Tensor([22], shape=(1,), dtype=int64)`<br>Class: **22 (Go Left)** |
|  | `tf.Tensor([34], shape=(1,), dtype=int64)`<br>Class: **34 (Danger Ahead)** |
|  | `tf.Tensor([38], shape=(1,), dtype=int64)`<br>Class: **38 (Dangerous curve to the left)** |
|  | `tf.Tensor([43], shape=(1,), dtype=int64)`<br>Class: **43 (Go right or straight)** |
|  | `tf.Tensor([46], shape=(1,), dtype=int64)`<br>Class: **46 (ZigZag Curve)**<br>Actual class: Uturn |

| | |
|---|---|
| | `tf.Tensor([48], shape=(1,), dtype=int64)` <br> Class: **48 (Under Construction)** |

## 5. Conclusion

In conclusion, the classification of traffic sign is an important problem especially with the advent of autonomous vehicles. The proposed system is simple and does the classification quite accurately it uses Convolutional Neural Network (CNN) and EfficientNetB0 Pre-trained model. The images are pre-processed using different technique. The final accuracy on the test dataset is 96.61%. One of the things we learned while working on the project was dealing with new Python libraries and improving computer vision skills for classifying images. One of the challenges we faced is the unbalanced data, so some classes have smaller number of images than other. Our future work is to get more data in order to get an equal number for all classes and obtain higher accuracy.

## 6. References

Bell, D. D. (2020, May 27). *Epoch (machine learning)*. Retrieved from Radiopaedia: https://radiopaedia.org/articles/epoch-machine-learning#:~:text=An%20epoch%20is%20a%20term,of%20data%20is%20very%20large).

Biswal, A. (2022, February 21). *Convolutional Neural Network Tutorial*. Retrieved from Simpli Learn: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

Brownlee, J. (2019, January 16). *A Gentle Introduction to Batch Normalization for Deep Neural Networks*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/

*EfficientNet-b0 convolutional neural network*. (2022). Retrieved from MathWorks: https://www.mathworks.com/help/deeplearning/ref/efficientnetb0.html

*Epoch in Neural Networks*. (2021, February 27). Retrieved from Baeldung: https://www.baeldung.com/cs/epoch-neural-networks

Gupta, A. (2021, October 7). *A Comprehensive Guide on Deep Learning Optimizers*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.

Hemateja, A. V. (2021, December 21). *Traffic Sign Dataset - Classification*. Retrieved from Kaggle: https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification

Hossin, Sulaiman. (2015, March). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process, 5*(2), 1-11. doi:10.5121/ijdkp.2015.5201

Ian Goodfellow, Y. B. (2016). *Deep Learning.* MIT Press.

Kumar, S. (2020, May 1). *Data splitting technique to fit any Machine Learning Model*. Retrieved from Towards Data Science: https://towardsdatascience.com/data-splitting-technique-to-fit-any-machine-learning-model-c0d7f3f1c790

Manjiri Bichkar, Suyasha Bobhate, Prof. Sonal Chaudhari. (2021, May 18). Traffic Sign Classification and Detection of Indian Traffic Signs using Deep. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7*(3), 215-220. doi:https://doi.org/10.32628/CSEIT217325

Manmayi. (2020, June 8). *Choose optimal number of epochs*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/choose-optimal-number-of-epochs-to-train-a-neural-network-in-keras/

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* Cambridge: MIT Press.

Preeti Bailke, Kunjal Agrawal. (2022). Traffic Sign Classification Using CNN. *International Journal for Research in Applied Science & Engineering Technology (IJRASET), 10*(II), 198-206.

Sreya, K. V. (2021, August). Traffic Sign Classification Using CNN. *International Journal for Research in Applied Science & Engineering Technology (IJRASET), 9*(VIII), 1952-1956. doi:https://doi.org/10.22214/ijraset.2021.37700

## 7. Appendix

Source Code:

```
#Import the libraries
import os
import glob
from pathlib import Path

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

import tensorflow as tf

TEST_SIZE=0.1
SEED=0
BATCH_SIZE=64

#Initialize the random number generator; the random number generator needs a number
#to start with (a seed value), to be able to generate a random number
tf.random.set_seed(SEED)
```

```
#Mount the Google Drive to Google Colab
from google.colab import drive
drive.mount('/content/drive/')
```

```
#Upload the dataset label
labels_df = pd.read_csv('/content/drive/MyDrive/archive/labels.csv')
print(labels_df.sample(10))

#Create a label map
label_map = dict(labels_df.values)
print(label_map)
```

```
#Upload the dataset
image_list = list(Path('/content/drive/MyDrive/archive/traffic_Data/DATA').glob(r'**/*.png'))
labels = list(map(lambda path: os.path.split(os.path.split(path)[0])[1], image_list))

#Create dataframe with path of images and labels
image_series = pd.Series(image_list).astype(str)
labels_series = pd.Series(labels).astype(str)
frame = {'image':image_series, 'label':labels_series}
image_df = pd.DataFrame(frame)
image_df.info()
print(image_df.sample(5))
```

```
#Plot to show the number of images for each label
count_labels = image_df.groupby(['label']).size()
plt.figure(figsize=(17,5))
plt.ylabel('count images')
sns.barplot(x=count_labels.index, y=count_labels, palette="rocket")
```

```
SPLIT_MINIMUM_COUNT = 10
```

```
#Splitting the dataset

#Allocate a  dataset that has at least SPLIT_MINIMUM_COUNT_IMAGES of images
#split_df: dataframe for train
#train1_df: dataframe for drop
def split_dataset(df, rate=SPLIT_MINIMUM_COUNT):

  count_labels = df.groupby(['label']).size()
  count_labels_df = count_labels.to_frame(name='count_images').reset_index()

  drop_label_list = list(
      count_labels_df['label'].\
      loc[count_labels_df['count_images']<SPLIT_MINIMUM_COUNT]
  )
```

```python
  drop_df = df.copy()
  split_df = df.copy()

  for index, row in df.iterrows():
    if str(row.label) in drop_label_list:
      split_df = split_df.drop(index)
    else:
      drop_df = drop_df.drop(index)

  return split_df, drop_df
```

```python
#Train test split where test_df has minimum 1 image in all labels in random split.
def custom_train_test_split(df):

    labels = df.label.unique()
    print(labels)
    test_df = pd.DataFrame()

    for label in labels:
      label_samples = df.loc[df.label==label]
      test_df = test_df.append(label_samples.sample(len(label_samples)//10+1,
                                                    random_state=SEED))

    train_df = df.drop(list(test_df.index), axis=0)
    test_df = test_df.sample(frac=1, random_state=SEED)
    train_df = train_df.sample(frac=1, random_state=SEED)

    return train_df, test_df
```

```python
#Splitting the dataset
split_df, _ = split_dataset(image_df)
train_df, test_df = custom_train_test_split(split_df)
train, val = custom_train_test_split(train_df)
```

```python
#Count the number of the classes
train_labels = train_df.groupby(['label']).size()
NUM_CLASSES = len(train_labels)
```

```python
#Plot samples from the train dataset
fig, axes = plt.subplots(2,4, figsize=(16, 7))
for idx, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(train_df.image.iloc[idx]))
    ax.set_title(train_df.label.iloc[idx])
plt.tight_layout()
plt.show()
```

```python
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.efficientnet.preprocess_input,
    rotation_range = 10,
    width_shift_range=0.2,
    height_shift_range=0.2,
    fill_mode='constant',
    shear_range=0.1,
    zoom_range=0.2,
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.efficientnet.preprocess_input,
)
```

```
#Define the dataframe for each set

train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='image',
    y_col='label',
    color_mode='rgb',
    class_mode='categorical',
    target_size=(128, 128),
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=SEED,
)

val_images = test_generator.flow_from_dataframe(
    dataframe=val,
    x_col='image',
    y_col='label',
    color_mode='rgb',
    class_mode='categorical',
    target_size=(128, 128),
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=SEED,
)
```

```
test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='image',
    y_col='label',
    color_mode='rgb',
    class_mode='categorical',
    target_size=(128, 128),
    batch_size=BATCH_SIZE,
)
```

```
#Create the model
def create_model(input_shape=(128,128,3)):

#Load EfficientNet without last layer and add Dense and ouput Dense with NUM_CLASSES units

  inputs = tf.keras.layers.Input(input_shape)

  base_model = tf.keras.applications.EfficientNetB0(
      include_top=False,
      weights='imagenet',
      pooling='avg'
  )
  base_model.trainable = False
  x = base_model(inputs)
  x = tf.keras.layers.BatchNormalization()(x)
  x = tf.keras.layers.Dense(512, activation='relu')(x)
  #x = tf.keras.layers.Dropout(0.2)(x)
  #x = tf.keras.layers.Dense(256, activation='relu')(x)
  outputs = tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')(x)

  return tf.keras.models.Model(inputs, outputs)
```

```
model = create_model()
```

```
#Compile the mdoel to define the loss function, the optimizer and the metrics
model.compile(
    optimizer='Adam',
    loss='categorical_crossentropy',
    metrics=['acc'],
)
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)

#Fitting the model on the dataset
history = model.fit(
    train_images,
    epochs=40,
    validation_data=val_images,
    callbacks=[callback]
)
```

```
#Plotting the train and the validation accuracy per each epoch
plt.figure(figsize=(12,5))
plt.plot(history.history['acc'], label='train_acc')
plt.plot(history.history['val_acc'], label='val_acc')
plt.title('Accuracy plot')
plt.xlabel('epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
#Plotting the train and the validation loss per each epoch
plt.figure(figsize=(12,5))
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Loss plot')
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
#Plotting the train validation accuracy per each epoch
plt.figure(figsize=(12,5))
plt.plot(history.history['val_acc'], label='val_acc')
plt.title('Validation accuracy')
plt.xlabel('epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
#Plotting the train validation loss per each epoch
plt.figure(figsize=(12,5))
plt.plot(history.history['val_loss'], label='loss_acc')
plt.title('Validation loss')
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
from pandas._libs.tslibs.conversion import precision_from_unit
#Score and accuracy of the testing set
score, accuracy = model.evaluate(test_images)
print(f'Test score: {round(score,4)}, Test accuracy: {round(accuracy,4)}')
```

```
#Test dataframe
test_df
```

```
#Traffic signs prediction
im = Image.open("/content/drive/MyDrive/archive/traffic_Data/DATA/22/022_1_0001.png")
im = im.resize((128,128))
im_array = np.array(im)
im_array = np.expand_dims(im_array, axis = 0)
pred = model.predict(im_array)
print(pred)
pred_labels = tf.argmax(pred, axis = -1)
print(pred_labels)
```

```
#The predicted label of the image
pred_labels = tf.argmax(pred, axis = -1)
print(pred_labels)
```

```
#Predict the label of the testing set
pred=model.predict(test_images)
print(pred)
pred_labels = tf.argmax(pred, axis = -1)
print(pred_labels)
```

```
#The label of the testing dataset
test_labels= test_df.label
print(test_labels)
```