```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
import os
Root = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification"
os.chdir(Root)
```

```python
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
np.random.seed(42)

from matplotlib import style
style.use('fivethirtyeight')
```

```python
data_dir = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification"
train_path = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification/Train/"
test_path = "/content/drive/MyDrive/Colab
Notebooks/TSF/Traffic_sign_classification/Test/"
```

```python
IMG_HEIGHT = 30
IMG_WIDTH = 30
channels = 3
```

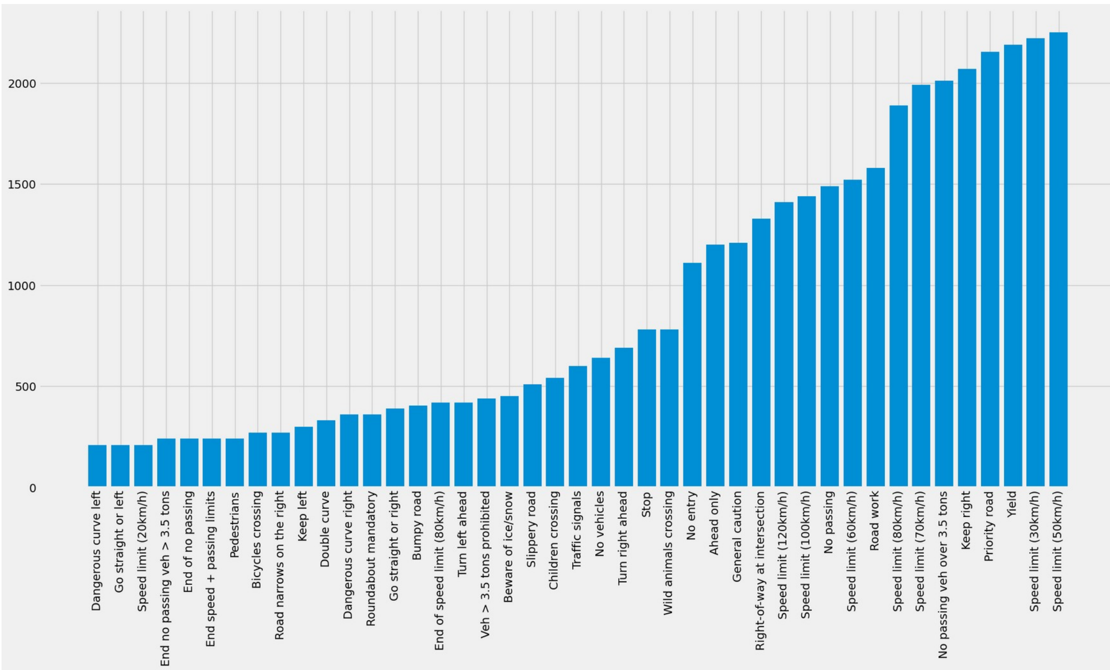
```python
NUM_CATEGORIES = len(os.listdir(train_path))
NUM_CATEGORIES
```

```
43
```

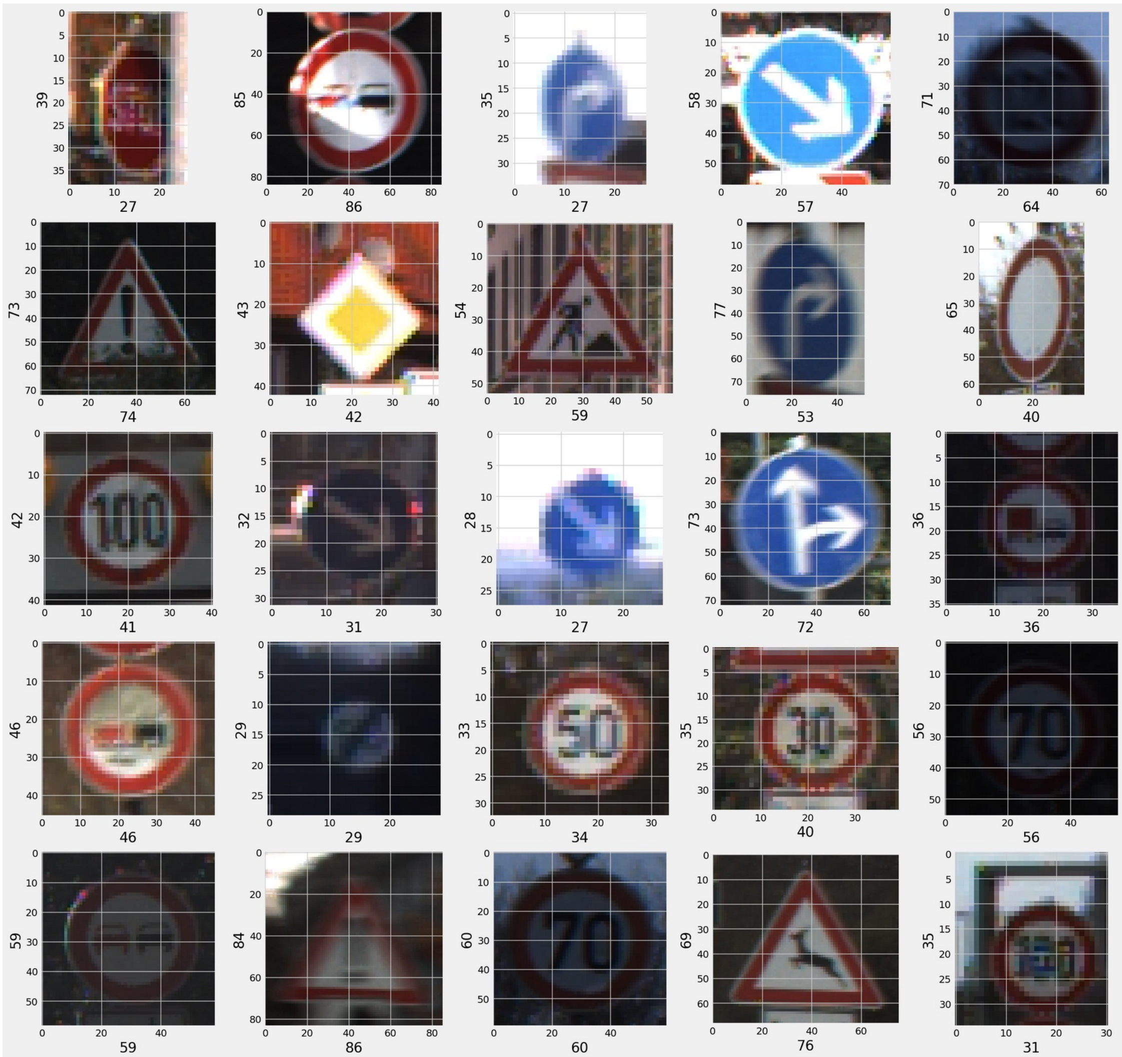
```python
classes = { 0:'Speed limit (20km/h)',
            1:'Speed limit (30km/h)',
            2:'Speed limit (50km/h)',
            3:'Speed limit (60km/h)',
            4:'Speed limit (70km/h)',
            5:'Speed limit (80km/h)',
            6:'End of speed limit (80km/h)',
            7:'Speed limit (100km/h)',
            8:'Speed limit (120km/h)',
```

```
            9:'No passing',
            10:'No passing veh over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield',
            14:'Stop',
            15:'No vehicles',
            16:'Veh > 3.5 tons prohibited',
            17:'No entry',
            18:'General caution',
            19:'Dangerous curve left',
            20:'Dangerous curve right',
            21:'Double curve',
            22:'Bumpy road',
            23:'Slippery road',
            24:'Road narrows on the right',
            25:'Road work',
            26:'Traffic signals',
            27:'Pedestrians',
            28:'Children crossing',
            29:'Bicycles crossing',
            30:'Beware of ice/snow',
            31:'Wild animals crossing',
            32:'End speed + passing limits',
            33:'Turn right ahead',
            34:'Turn left ahead',
            35:'Ahead only',
            36:'Go straight or right',
            37:'Go straight or left',
            38:'Keep right',
            39:'Keep left',
            40:'Roundabout mandatory',
            41:'End of no passing',
            42:'End no passing veh > 3.5 tons' }

folders = os.listdir(train_path)

train_number = []
class_num = []

for folder in folders:
    train_files = os.listdir(train_path + '/' + folder)
    train_number.append(len(train_files))
    class_num.append(classes[int(folder)])

# Sorting the dataset on the basis of number of images in each class
zipped_lists = zip(train_number, class_num)
sorted_pairs = sorted(zipped_lists)
tuples = zip(*sorted_pairs)
train_number, class_num = [ list(tuple) for tuple in  tuples]
```

```python
# Plotting the number of images in each class
plt.figure(figsize=(21,10))
plt.bar(class_num, train_number)
plt.xticks(class_num, rotation='vertical')
plt.show()
```




```python
import random
from matplotlib.image import imread

test = pd.read_csv(data_dir + '/Test.csv')
imgs = test["Path"].values

plt.figure(figsize=(25,25))

for i in range(1,26):
    plt.subplot(5,5,i)
    random_img_path = data_dir + '/' + random.choice(imgs)
    rand_img = imread(random_img_path)
    plt.imshow(rand_img)

    plt.xlabel(rand_img.shape[1], fontsize = 20)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 20)#height of image
```




```python
image_data = []
image_labels = []

for i in range(NUM_CATEGORIES):
    path = data_dir + '/Train/' + str(i)
    images = os.listdir(path)

    for img in images:
        try:
            image = cv2.imread(path + '/' + img)
            image_fromarray = Image.fromarray(image, 'RGB')
            resize_image = image_fromarray.resize((IMG_HEIGHT,
IMG_WIDTH))
            image_data.append(np.array(resize_image))
            image_labels.append(i)
        except:
            print("Error in " + img)
```

```
# Changing the list to numpy array
image_data = np.array(image_data)
image_labels = np.array(image_labels)

print(image_data.shape, image_labels.shape)
```

```
(39605, 30, 30, 3) (39605,)
```

```
shuffle_indexes = np.arange(image_data.shape[0])
np.random.shuffle(shuffle_indexes)
image_data = image_data[shuffle_indexes]
image_labels = image_labels[shuffle_indexes]
```

```
X_train, X_val, y_train, y_val = train_test_split(image_data,
image_labels, test_size=0.3, random_state=42, shuffle=True)

X_train = X_train/255
X_val = X_val/255

print("X_train.shape", X_train.shape)
print("X_valid.shape", X_val.shape)
print("y_train.shape", y_train.shape)
print("y_valid.shape", y_val.shape)
```

```
X_train.shape (27723, 30, 30, 3)
X_valid.shape (11882, 30, 30, 3)
y_train.shape (27723,)
y_valid.shape (11882,)
```

```
y_train = keras.utils.to_categorical(y_train, NUM_CATEGORIES)
y_val = keras.utils.to_categorical(y_val, NUM_CATEGORIES)

print(y_train.shape)
print(y_val.shape)
```

```
(27723, 43)
(11882, 43)
```

```
from tensorflow.keras import layers, models, optimizers

model = models.Sequential()

# First convolutional block
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH,
channels)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
```

```python
model.add(layers.Dropout(rate=0.25))

# Second convolutional block
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

# Third convolutional block
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

# Flatten the output and feed it into a fully connected layer
model.add(layers.Flatten())
model.add(layers.Dense(units=512, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(rate=0.5))

# Output layer
model.add(layers.Dense(units=NUM_CATEGORIES, activation='softmax'))


from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

lr = 0.001
epochs = 30

opt = Adam(lr=lr, decay=lr / (epochs * 0.5))
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])

aug = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip=False,
    vertical_flip=False,
```

```
    fill_mode="nearest")

history = model.fit(aug.flow(X_train, y_train, batch_size=32),
epochs=epochs, validation_data=(X_val, y_val))
```

```
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/
adam.py:117: UserWarning: The `lr` argument is deprecated, use
`learning_rate` instead.
  super().__init__(name, **kwargs)

Epoch 1/30
867/867 [==============================] - 182s 207ms/step - loss:
1.9217 - accuracy: 0.4796 - val_loss: 0.5357 - val_accuracy: 0.8249
Epoch 2/30
867/867 [==============================] - 172s 198ms/step - loss:
0.4141 - accuracy: 0.8671 - val_loss: 0.0822 - val_accuracy: 0.9756
Epoch 3/30
867/867 [==============================] - 179s 206ms/step - loss:
0.1979 - accuracy: 0.9376 - val_loss: 0.0462 - val_accuracy: 0.9859
Epoch 4/30
867/867 [==============================] - 179s 207ms/step - loss:
0.1395 - accuracy: 0.9557 - val_loss: 0.0233 - val_accuracy: 0.9937
Epoch 5/30
867/867 [==============================] - 180s 207ms/step - loss:
0.1093 - accuracy: 0.9637 - val_loss: 0.0169 - val_accuracy: 0.9955
Epoch 6/30
867/867 [==============================] - 173s 199ms/step - loss:
0.0980 - accuracy: 0.9695 - val_loss: 0.0176 - val_accuracy: 0.9949
Epoch 7/30
867/867 [==============================] - 174s 200ms/step - loss:
0.0730 - accuracy: 0.9766 - val_loss: 0.0080 - val_accuracy: 0.9976
Epoch 8/30
867/867 [==============================] - 179s 207ms/step - loss:
0.0716 - accuracy: 0.9774 - val_loss: 0.0103 - val_accuracy: 0.9972
Epoch 9/30
867/867 [==============================] - 171s 198ms/step - loss:
0.0619 - accuracy: 0.9811 - val_loss: 0.0096 - val_accuracy: 0.9967
Epoch 10/30
867/867 [==============================] - 180s 207ms/step - loss:
0.0552 - accuracy: 0.9826 - val_loss: 0.0104 - val_accuracy: 0.9971
Epoch 11/30
867/867 [==============================] - 177s 204ms/step - loss:
0.0505 - accuracy: 0.9828 - val_loss: 0.0088 - val_accuracy: 0.9979
Epoch 12/30
867/867 [==============================] - 179s 206ms/step - loss:
0.0514 - accuracy: 0.9848 - val_loss: 0.0060 - val_accuracy: 0.9982
Epoch 13/30
867/867 [==============================] - 181s 208ms/step - loss:
0.0370 - accuracy: 0.9880 - val_loss: 0.0101 - val_accuracy: 0.9965
Epoch 14/30
```

```
867/867 [==============================] - 173s 200ms/step - loss:
0.0377 - accuracy: 0.9891 - val_loss: 0.0060 - val_accuracy: 0.9981
Epoch 15/30
867/867 [==============================] - 174s 200ms/step - loss:
0.0354 - accuracy: 0.9885 - val_loss: 0.0080 - val_accuracy: 0.9978
Epoch 16/30
867/867 [==============================] - 173s 200ms/step - loss:
0.0285 - accuracy: 0.9915 - val_loss: 0.0064 - val_accuracy: 0.9980
Epoch 17/30
867/867 [==============================] - 178s 206ms/step - loss:
0.0320 - accuracy: 0.9903 - val_loss: 0.0035 - val_accuracy: 0.9987
Epoch 18/30
867/867 [==============================] - 180s 207ms/step - loss:
0.0267 - accuracy: 0.9916 - val_loss: 0.0098 - val_accuracy: 0.9972
Epoch 19/30
867/867 [==============================] - 180s 208ms/step - loss:
0.0256 - accuracy: 0.9915 - val_loss: 0.0031 - val_accuracy: 0.9992
Epoch 20/30
867/867 [==============================] - 180s 208ms/step - loss:
0.0225 - accuracy: 0.9930 - val_loss: 0.0046 - val_accuracy: 0.9985
Epoch 21/30
867/867 [==============================] - 180s 208ms/step - loss:
0.0234 - accuracy: 0.9926 - val_loss: 0.0033 - val_accuracy: 0.9991
Epoch 22/30
867/867 [==============================] - 179s 207ms/step - loss:
0.0196 - accuracy: 0.9938 - val_loss: 0.0028 - val_accuracy: 0.9992
Epoch 23/30
867/867 [==============================] - 173s 200ms/step - loss:
0.0186 - accuracy: 0.9939 - val_loss: 0.0051 - val_accuracy: 0.9985
Epoch 24/30
867/867 [==============================] - 176s 203ms/step - loss:
0.0204 - accuracy: 0.9939 - val_loss: 0.0042 - val_accuracy: 0.9988
Epoch 25/30
867/867 [==============================] - 178s 205ms/step - loss:
0.0192 - accuracy: 0.9939 - val_loss: 0.0049 - val_accuracy: 0.9987
Epoch 26/30
867/867 [==============================] - 178s 206ms/step - loss:
0.0205 - accuracy: 0.9933 - val_loss: 0.0048 - val_accuracy: 0.9989
Epoch 27/30
867/867 [==============================] - 173s 200ms/step - loss:
0.0177 - accuracy: 0.9943 - val_loss: 0.0033 - val_accuracy: 0.9993
Epoch 28/30
867/867 [==============================] - 172s 199ms/step - loss:
0.0144 - accuracy: 0.9950 - val_loss: 0.0037 - val_accuracy: 0.9992
Epoch 29/30
867/867 [==============================] - 173s 200ms/step - loss:
0.0133 - accuracy: 0.9956 - val_loss: 0.0032 - val_accuracy: 0.9993
Epoch 30/30
867/867 [==============================] - 178s 205ms/step - loss:
0.0163 - accuracy: 0.9950 - val_loss: 0.0029 - val_accuracy: 0.9992
```

```python
model.save("vinit.h5")

test = pd.read_csv(data_dir + '/Test.csv')

labels = test["ClassId"].values
imgs = test["Path"].values

data = []

for img in imgs:
    try:
        image = cv2.imread(data_dir + '/' + img)
        image_fromarray = Image.fromarray(image, 'RGB')
        resize_image = image_fromarray.resize((IMG_HEIGHT, IMG_WIDTH))
        data.append(np.array(resize_image))
    except:
        print("Error in " + img)

X_test = np.array(data)
X_test = X_test / 255

pred = model.predict(X_test)
pred_classes = np.argmax(pred, axis=1)

#Accuracy with the test data
print('Test Data accuracy: ', accuracy_score(labels, pred_classes) *
100)
```

```
395/395 [==============================] - 18s 44ms/step
Test Data accuracy:  98.83610451306414
```