## Dataset name: Traffic Sign Dataset - Classification

## Dataset link: [https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification](https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification)

This project's purpose is the classification of a dataset with images about traffic signs using Convolutional Neural Networks with the Keras module . The training images, the input data is 3818 photos of traffic signs . The testing data is 1994 .

## Importing the data

Data is in a folder formation for Training , with sub directories that has a class from 0 to 46 and Testing which has just the photos , no sub directories .

```
path="/content/drive/MyDrive/Colab Notebooks/nn/traffic_Data/DATA"
labelfile="/content/drive/MyDrive/Colab Notebooks/nn/labels.csv"
```

## Loading data

Two lists are created from the following code : Images , that represents the training images , and Classno , the number of classes , taken from the naming of the directory in drive .

```python
count=0
Images=[]
Classno=[]
mylist=os.listdir(path)
print("Total Classes Detected: ",len(mylist))
noofclasses=len(mylist)
print("Importing Classes .....")
for i in range(0,len(mylist)):
    mypics=os.listdir("/content/drive/MyDrive/Colab Notebooks/nn/traffic_Data/DATA/" + str(count))
    for y in mypics:
        current=cv2.imread(path+"/"+str(count)+"/"+ y)
        Images.append(current)
        Classno.append(count)
    print(str(count) + "/" + str(noofclasses))
    count=count+1
print(str(noofclasses)+("/")+str(noofclasses))
print(" ")
#print(Images)
#print(Classno)
Images=np.array(Images)
Classno=np.array(Classno)
```

```
Total Classes Detected:  47
Importing Classes .....
0/47
1/47
2/47
3/47
4/47
5/47
6/47
7/47
8/47
9/47
```

```
36/47
37/47
38/47
39/47
40/47
41/47
42/47
43/47
44/47
45/47
46/47
47/47
```

Shape of data

```
print(Images.shape)
print(Classno.shape)
```

```
(3818,)
(3818,)
```

Reading the labels file

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/nn/labels.csv')
```

Lowering the classes number

These rows had duplicates classes , they needed to be dropped .

```
df.drop([18],axis=0,inplace=True)
df.drop([19],axis=0,inplace=True)
df.drop([36],axis=0,inplace=True)
```

## Renaming classes due to text size

These classes should be : Dangerous curve to the Left and to the Right .

```
df['Name'][38] = 'Dang curve to L'
df['Name'][39] = 'Dang curve to R'
```

## Labels

| | 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) |
| 5 | 5 | Speed limit (60km/h) |
| 6 | 6 | Speed limit (70km/h) |
| 7 | 7 | speed limit (80km/h) |
| 8 | 8 | Dont Go straight or left |
| 9 | 9 | Dont Go straight or Right |
| 10 | 10 | Dont Go straight |
| 11 | 11 | Dont Go Left |

## Dropping rows that are Unknown

```
[ ]  #print(df['Name'])
     print(df.shape)
     for i in df.index:
       if "Unknown" in df['Name'][i]:
         print("DROPPED" + "   " + str(i))
         df.drop([i],axis=0,inplace=True)
     print(df.shape)
```

```
(55, 2)
DROPPED   40
DROPPED   41
DROPPED   42
DROPPED   45
DROPPED   49
DROPPED   52
DROPPED   56
DROPPED   57
(47, 2)
```

Reindexing the labels data

```
uu = np.arange(0,47)
```

```
print(uu)
print(df['ClassId'])
df['ClassId'] = uu
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46]
0      0
1      1
2      2
3      3
4      4
5      5
6      6
7      7
8      8
9      9
10     10
11     11
12     12
13     13
14     14
15     15
16     16
```

```
df['ClassId']
```

```
34    32
35    33
37    34
38    35
39    36
43    37
44    38
46    39
47    40
48    41
50    42
51    43
53    44
54    45
55    46
Name: ClassId, dtype: int64
```

```
df
```

| | | |
|---|---|---|
| **39** | 36 | Dang curve to R |
| **43** | 37 | Go right or straight |
| **44** | 38 | Go left or straight |
| **46** | 39 | ZigZag Curve |
| **47** | 40 | Train Crossing |
| **48** | 41 | Under Construction |
| **50** | 42 | Fences |
| **51** | 43 | Heavy Vehicle Accidents |
| **53** | 44 | Give Way |
| **54** | 45 | No stopping |
| **55** | 46 | No entry |

```
print(Classno)
print(Classno.shape)
print(df.shape)
```

```
[ 0  0  0 ... 46 46 46]
(3818,)
(47, 2)
```

I have deleted the folders from the drive that has been dropped or been duplicates . And I renamed them for consistency .

## Changing type ( int to str ) of attribute due to further need of data representation

```
import numpy as np
Classno1 = Classno
Classno = np.array(Classno,dtype='U')
df['ClassId'] = np.array(df['ClassId'],dtype='U')
```

```
print(type(Classno[0]))
```

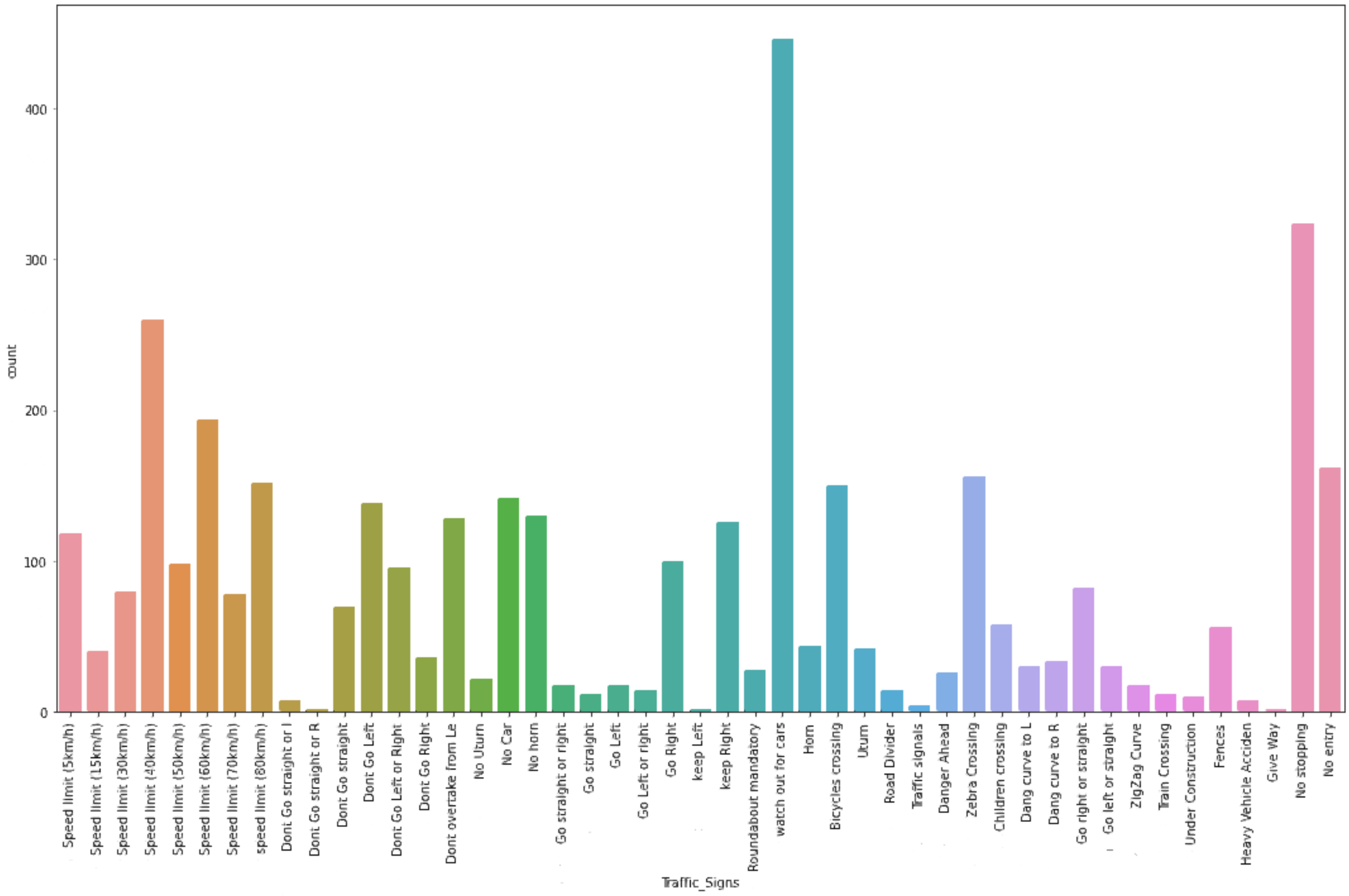
```
<class 'numpy.str_'>
```

```
a1 = Classno[0]
a2 = df['ClassId'][0]

print(a1)
print(a2)

print(type(a1))
print(type(a2))
```

```
0
0
<class 'numpy.str_'>
<class 'str'>
```

```python
df.index = np.arange(0,47)
for i in range(0,47):
  for y in range(0,3818):
    if Classno[y] == df['ClassId'][i]:
      Classno[y] = df['Name'][i]
      print(Classno[y])
```

Data distribution among classes




Structure of Labels after renaming

```
df
```

| | Traffic_Signs |
|---|---|
| 0 | Speed limit (5km/h) |
| 1 | Speed limit (5km/h) |
| 2 | Speed limit (5km/h) |
| 3 | Speed limit (5km/h) |
| 4 | Speed limit (5km/h) |
| ... | ... |
| 3813 | No entry |
| 3814 | No entry |
| 3815 | No entry |
| 3816 | No entry |
| 3817 | No entry |

3818 rows × 1 columns

Preview of the training data


## Machine learning model

I have remembered the labels in int type too , for the model to understand , the strings are just for representative purposes.

```
Classno = Classno1
```

## Splitting data

```
X_train, X_test, Y_train, Y_test = train_test_split(Images,Classno,test_size=testratio)
X_train, X_validation ,Y_train, Y_validation = train_test_split(X_train,Y_train,test_size=validationratio)
```

```
testratio=0.1
validationratio=0.1
```

Data shapes

```
DATA SHAPES
Train:
(3092,) (3092,)
Validation:
(344,) (344,)
Test:
(382,) (382,)
```

Preprocessing the images to gray level from BGR, and reshaping them to the “None,100,100,1” size

```
def grayscale(img):
    img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img,(100,100))
    return img
def equalize(img):
    img=cv2.equalizeHist(img)
    return img
def preprocessing(img):
    img=grayscale(img)
    img=equalize(img)
    img=img/255.0
    return img
```

```
X_train=np.array(list(map(preprocessing,X_train)))
X_validation=np.array(list(map(preprocessing,X_validation)))
X_test=np.array(list(map(preprocessing,X_test)))




X_train=X_train.reshape(X_train.shape[0],100,100,1)
X_validation=X_validation.reshape(X_validation.shape[0],100,100,1)
X_test=X_test.reshape(X_test.shape[0],100,100,1)
```

Defining the model

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_6 (Conv2D)           (None, 98, 98, 32)        320       
                                                                 
 batch_normalization_8 (Batc  (None, 98, 98, 32)       128       
 hNormalization)                                                 
                                                                 
 max_pooling2d_6 (MaxPooling  (None, 49, 49, 32)       0         
 2D)                                                             
                                                                 
 dropout_8 (Dropout)         (None, 49, 49, 32)        0         
                                                                 
 conv2d_7 (Conv2D)           (None, 47, 47, 64)        18496     
                                                                 
 batch_normalization_9 (Batc  (None, 47, 47, 64)       256       
 hNormalization)                                                 
                                                                 
 max_pooling2d_7 (MaxPooling  (None, 23, 23, 64)       0         
 2D)                                                             
                                                                 
 dropout_9 (Dropout)         (None, 23, 23, 64)        0         
                                                                 
 conv2d_8 (Conv2D)           (None, 21, 21, 128)       73856     
                                                                 
 batch_normalization_10 (Bat  (None, 21, 21, 128)      512       
 chNormalization)                                                
                                                                 
 max_pooling2d_8 (MaxPooling  (None, 10, 10, 128)      0         
 2D)                                                             
                                                                 
 dropout_10 (Dropout)        (None, 10, 10, 128)       0         
                                                                 
 flatten_2 (Flatten)         (None, 12800)             0         
                                                                 
 dense_4 (Dense)             (None, 512)               6554112   
                                                                 
 batch_normalization_11 (Bat  (None, 512)              2048      
 chNormalization)                                                
                                                                 
 dropout_11 (Dropout)        (None, 512)               0         
                                                                 
 dense_5 (Dense)             (None, 47)                24111     
                                                                 
=================================================================
Total params: 6,673,839
Trainable params: 6,672,367
Non-trainable params: 1,472
_________________________________________________________________
```

```python
def mymodel():
    nooffilters=32
    sizeoffilters=(3,3)
    sizeoffilters2=(3,3)
    sizeofpool=(2,2)
    noofnodes=500

    model=Sequential()
    model.add((Conv2D(32,(3,3),input_shape=(100,100,1),activation='relu')))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))


    model.add((Conv2D(64, (3, 3),activation='relu')))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=sizeofpool))
    model.add(Dropout(0.25))


    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512,activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(noofclasses,activation='softmax'))

    model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

- The first layer is a 2D convolutional layer with 32 filters of size (3,3) and ReLU activation function. Input shape is (100,100,1) .
- The second layers is batch normalization layer to normalize the activations of the previous layer.
- The model then applies a 2D max pooling layer to down sample the feature maps from the previous layer.
- At the end , a dropout layer with 0.25 is added to prevent overfitting by randomly dropping out 25% of the neurons.

- Two more convolutional layers are the same but , with 64 filters and 128 filters respectively.

- Finally , the models flattens the feature maps from the previous layer and passes them through two dense (fully connected) layers with 512 nodes and the number of classes , 47 in our case .
- The final dense layer uses the softmax activation function, for the multiclass classification problem .

- The model then is compiled using the RMSprop optimization algorithm, the categorical crossentropy loss function, and for metric is accuracy .

## Training the data

```
Epoch 1/100
 99/100 [============================>.] - ETA: 0s - loss: 3.0941 - accuracy: 0.2812
Epoch 1: val_loss improved from inf to 8.62593, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.h5
100/100 [==============================] - 4s 27ms/step - loss: 3.0923 - accuracy: 0.2809 - val_loss: 8.6259 - val_accuracy: 0.0930 - lr: 0.0010
Epoch 2/100
 98/100 [============================>.] - ETA: 0s - loss: 1.9563 - accuracy: 0.4994
Epoch 2: val_loss did not improve from 8.62593
100/100 [==============================] - 2s 24ms/step - loss: 1.9489 - accuracy: 0.4987 - val_loss: 21.0015 - val_accuracy: 0.0000e+00 - lr: 0.0010
Epoch 3/100
 99/100 [============================>.] - ETA: 0s - loss: 1.5828 - accuracy: 0.5604
Epoch 3: val_loss did not improve from 8.62593

Epoch 3: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
100/100 [==============================] - 2s 23ms/step - loss: 1.5842 - accuracy: 0.5617 - val_loss: 19.7535 - val_accuracy: 0.0087 - lr: 0.0010
Epoch 4/100
 99/100 [============================>.] - ETA: 0s - loss: 1.1953 - accuracy: 0.6597
Epoch 4: val_loss did not improve from 8.62593
100/100 [==============================] - 2s 24ms/step - loss: 1.1923 - accuracy: 0.6612 - val_loss: 12.1505 - val_accuracy: 0.0145 - lr: 5.0000e-04
Epoch 5/100
100/100 [==============================] - ETA: 0s - loss: 1.0008 - accuracy: 0.7053
Epoch 5: val_loss improved from 8.62593 to 6.12694, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.h5
100/100 [==============================] - 3s 27ms/step - loss: 1.0008 - accuracy: 0.7053 - val_loss: 6.1269 - val_accuracy: 0.2791 - lr: 5.0000e-04
Epoch 6/100
100/100 [==============================] - ETA: 0s - loss: 0.8884 - accuracy: 0.7494
Epoch 6: val_loss improved from 6.12694 to 2.12607, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.h5
100/100 [==============================] - 3s 25ms/step - loss: 0.8884 - accuracy: 0.7494 - val_loss: 2.1261 - val_accuracy: 0.6047 - lr: 5.0000e-04
Epoch 7/100
 98/100 [============================>.] - ETA: 0s - loss: 0.7686 - accuracy: 0.7857
Epoch 7: val_loss improved from 2.12607 to 0.81505, saving model to /content/drive/MyDrive/Colab Notebooks/best_model.h5
100/100 [==============================] - 3s 26ms/step - loss: 0.7636 - accuracy: 0.7869 - val_loss: 0.8150 - val_accuracy: 0.8459 - lr: 5.0000e-04
Epoch 8/100
```

```
100/100 [==============================] - 2s 23ms/step - loss: 0.2207 - accuracy: 0.9395 - val_loss: 0.1960 - val_accuracy: 0.9506 - lr: 1.0000e-06
Epoch 97/100
100/100 [==============================] - ETA: 0s - loss: 0.2168 - accuracy: 0.9351
Epoch 97: val_loss did not improve from 0.11142
100/100 [==============================] - 2s 23ms/step - loss: 0.2168 - accuracy: 0.9351 - val_loss: 0.2500 - val_accuracy: 0.9535 - lr: 1.0000e-06
Epoch 98/100
100/100 [==============================] - ETA: 0s - loss: 0.2232 - accuracy: 0.9389
Epoch 98: val_loss did not improve from 0.11142
100/100 [==============================] - 2s 23ms/step - loss: 0.2232 - accuracy: 0.9389 - val_loss: 0.2664 - val_accuracy: 0.9564 - lr: 1.0000e-06
Epoch 99/100
 99/100 [============================>.] - ETA: 0s - loss: 0.2520 - accuracy: 0.9313
Epoch 99: val_loss did not improve from 0.11142
100/100 [==============================] - 2s 22ms/step - loss: 0.2506 - accuracy: 0.9320 - val_loss: 0.1739 - val_accuracy: 0.9651 - lr: 1.0000e-06
Epoch 100/100
100/100 [==============================] - ETA: 0s - loss: 0.2094 - accuracy: 0.9496
Epoch 100: val_loss did not improve from 0.11142
100/100 [==============================] - 2s 23ms/step - loss: 0.2094 - accuracy: 0.9496 - val_loss: 0.2159 - val_accuracy: 0.9564 - lr: 1.0000e-06
Test Score:  0.19060850143432617
Test Accuracy:  0.9685863852500916
```

## Importing the model in .h5 format

```
from keras.models import load_model
model = load_model('/content/drive/MyDrive/Colab Notebooks/best_model.h5')
model.summary()
```

## Loss and Accuracy values over epochs


Testing the model on new data

```
path="/content/drive/MyDrive/Colab Notebooks/nn/traffic_Data/TEST/"
```

```
test_filenames = os.listdir(path)
test_data = pd.DataFrame({
    'filename': test_filenames
})
nb_samples = test_data.shape[0]
```

```
test_data.shape
```

```
(1994, 1)
```

```
test_data.head()
```

| | filename |
|---|---|
| 0 | 026_0014_j.png |
| 1 | 026_0023_j.png |
| 2 | 024_1_0013.png |
| 3 | 026_0005_j.png |
| 4 | 026_0052_j.png |

Bringing the testing data

```
count=0
Images_Test=[]
#Classno_Test=[]
test_photos=os.listdir("/content/drive/MyDrive/Colab Notebooks/nn/traffic_Data/TEST/")
print("Total Images Detected: ",len(mylist))
#noofclasses=len(mylist)
print("Importing Images .....")
for y in test_photos:
    current=cv2.imread(path + "/" + y)
    Images_Test.append(current)
        #Classno.append(count)
    #print(str(count) + "/" + str(noofclasses))
    count=count+1
    print(count)
#print(str(noofclasses)+("/")+str(noofclasses))
print(" ")
#print(Images)
#print(Classno)
Images_Test=np.array(Images_Test)
#Classno=np.array(Classno)
```

```
Images_Test.shape
```

```
(1994,)
```

Plotting the new data


Shape of test images after changing the color channels number

```
Images_Test=np.array(list(map(preprocessing,Images_Test)))
Images_Test = Images_Test.reshape(Images_Test.shape[0],100,100,1)
```

```
Images_Test.shape
```

```
(1994, 100, 100, 1)
```

Prediction

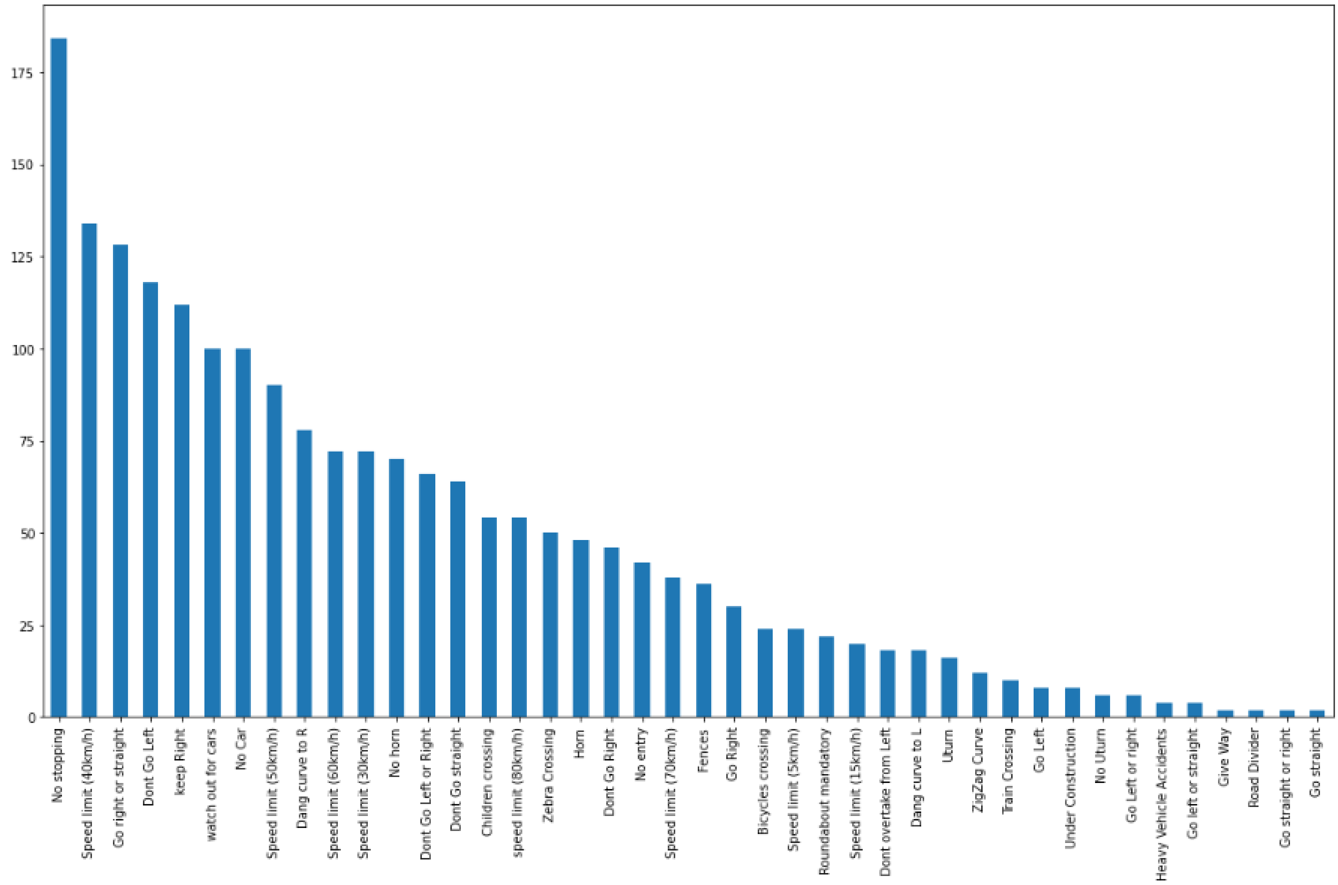
```
predict = model.predict(Images_Test)
```

Making a labeling for the predicted classes

```
test_data['category'] = np.argmax(predict, axis=-1)
label_map = {
 0   :"Speed limit (5km/h)",
1 :"Speed limit (15km/h)",
2 :"Speed limit (30km/h)",
3 :"Speed limit (40km/h)",
4 :"Speed limit (50km/h)",
5 :"Speed limit (60km/h)",
6 :"Speed limit (70km/h)",
7 :"speed limit (80km/h)",
8 :"Dont Go straight or left",
9 :"Dont Go straight or Right",
10  :"Dont Go straight",
11  :"Dont Go Left",
12  :"Dont Go Left or Right",
13  :"Dont Go Right",
14  :"Dont overtake from Left",
15  :"No Uturn",
```

```
38  :"Go left or straight",
39  :"ZigZag Curve",
40  :"Train Crossing",
41  :"Under Construction",
42  :"Fences",
43  :"Heavy Vehicle Accidents",
44  :"Give Way",
45  :"No stopping",
46  :"No entry"
}
test_data['category'] = test_data['category'].replace
test_data
```

Plotting the new data values count




Plotting predicted images


026_0014_j.png(Dont Go Left)
026_0023_j.png(keep Right)
024_1_0013.png(Speed limit (30km/h))
026_0005_j.png(keep Right)
026_0052_j.png(keep Right)
026_0059_j.png(keep Right)
026_0030_j.png(keep Right)
北区出口
024_1_0011.png(Go Right)
026_0006_j.png(keep Right)
026_1_0015_1_j.png(watch out for cars)
026_1_0055_1_j.png(keep Right)
026_0066_j.png(keep Right)
026_1_0056_1_j.png(keep Right)
026_1_0047_1_j.png(keep Right)
026_1_0034_1_j.png(No stopping)
026_1_0041_1_j.png(keep Right)
026_1_0045_1_j.png(keep Right)
026_1_0040_1_j.png(keep Right)
026_1_0038_1_j.png(keep Right)
026_1_0011_1_j.png(keep Right)