In this project, we will use Convolutional Neural Network to build train and test a traffic sign classification model. We will build this model using tensorflow and keras. It is a multiclass classification problem. This model can be used to make smarter cars.

```
# Run this cell and select the kaggle.json file downloaded from the Kaggle account settings page.
from google.colab import files
files.upload()
```

Choose Files kaggle (1).json

- **kaggle (1).json**(application/json) - 76 bytes, last modified: 7/12/2024 - 100% done

```
Saving kaggle (1).json to kaggle (1).json
{'kaggle (1).json':
b'{"username":"matheusdavingiovanni","key":"27078588bcdc2c4f122a69a122b337bd"}'}
```

We will start by connecting to Kaggle using Kaggle API which can be downloaded from your Kaggle account's settings and uploading it here(upload box).

```
# Next, install the Kaggle API client.
!pip install -q kaggle
```

Installing kaggle api using pip

```
# The Kaggle API client expects this file to be in ~/.kaggle, so move it there.
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

Setting up Kaggle using Kaggle API.

```
# Creating directory and changing the current working directory
!mkdir traffic_sign_dataset
%cd traffic_sign_dataset
```

```
/content/traffic_sign_dataset/traffic_sign_dataset
```

To store the data we will create a new directory and make it as current working directory.

```
# Searching for dataset
!kaggle datasets list -s gtsrb-german-traffic-sign
```

```
401 - Unauthorized - Unauthenticated
```

Searching Kaggle for the required dataset using search option(-s) with title 'dogbreedidfromcomp'. We can also use different search options like searching competitions, notebooks, kernels, datasets, etc.

```
# Downloading dataset and coming out of directory
!kaggle datasets download meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
%cd ..
```

```
Dataset URL: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
License(s): CC0-1.0
Downloading gtsrb-german-traffic-sign.zip to /content/traffic_sign_dataset/traffic_sign_dataset
 98% 602M/612M [00:09<00:00, 83.2MB/s]
100% 612M/612M [00:09<00:00, 64.5MB/s]
/content/traffic_sign_dataset
```

After searching the data next step would be downloading the data into collab notebook using references found in search option.

```
# Unzipping downloaded file and removing unusable file
!unzip traffic_sign_dataset/gtsrb-german-traffic-sign.zip -d traffic_sign_dataset
!rm traffic_sign_dataset/gtsrb-german-traffic-sign.zip
!rm -rf traffic_sign_dataset/Meta
!rm -rf traffic_sign_dataset/meta
!rm -rf traffic_sign_dataset/test
!rm -rf traffic_sign_dataset/train
!rm traffic_sign_dataset/Meta.csv
```

```
  inflating: traffic_sign_dataset/train/9/00009_00047_00004.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00005.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00006.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00007.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00008.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00009.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00010.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00011.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00012.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00013.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00014.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00015.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00016.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00017.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00018.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00019.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00020.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00021.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00022.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00023.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00024.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00025.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00026.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00027.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00028.png
  inflating: traffic_sign_dataset/train/9/00009_00047_00029.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00000.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00001.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00002.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00003.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00004.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00005.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00006.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00007.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00008.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00009.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00010.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00011.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00012.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00013.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00014.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00015.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00016.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00017.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00018.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00019.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00020.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00021.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00022.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00023.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00024.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00025.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00026.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00027.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00028.png
  inflating: traffic_sign_dataset/train/9/00009_00048_00029.png
```

We will unzip the data which is downloaded and remove the irrelevant files.

```python
# Importing libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
import seaborn as sns
import random
from PIL import Image
from sklearn.model_selection import  train_test_split
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPool2D
```

Importing required libraries.

```python
# Plotting 12 images to check dataset
plt.figure(figsize=(12,12))
path = "traffic_sign_dataset/Test"
for i in range(1,17):
    plt.subplot(4,4,i)
    plt.tight_layout()
    rand_img = imread(path +'/'+ random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xlabel(rand_img.shape[1], fontsize = 10)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 10)#height of image
```

Visualizing some images of traffic sign from the test dataset. we can see here that the dimension of images are uneven.

```
# As size of images are different we have to make them equal so we will take mean of dimanesions
dim1 = []
dim2 = []

for i in range(0,43):
    labels = 'traffic_sign_dataset/Train' + '/{0}'.format(i)
    image_path = os.listdir(labels)
    for x in image_path:
        img = imread(labels + '/' + x)
        dim1.append(img.shape[0])
        dim2.append(img.shape[1])
```

For further processing we will require the images of same dimension. So, we will start storing the dimension of all the images from training dataset from all 43 classes.

```
#Printing mean dimension of images
print("Dimension 1 Mean : ",np.mean(dim1), " Dimension 2 Mean : ",np.mean(dim2))
```

```
Dimension 1 Mean :  50.328929582493814  Dimension 2 Mean :  50.83587951745773
```

Now we will find out the mean value of both the dimensions and analyse them. Here, we can see that (50,50) is the average shape for all the images.

```
# Now we will reshape the images to (50,50)
images = []
label_id = []

for i in range(43):
    labels = 'traffic_sign_dataset/Train' + '/{0}'.format(i)
    image_path = os.listdir(labels)
    for x in image_path:
        img = Image.open(labels + '/' + x)
        img = img.resize((50,50))
        img = np.array(img)
        images.append(img)
        label_id.append(i)
```

Now we will reshape the images into (50,50) and also store their label ids.

```
#Converting images into numpy array
images = np.array(images)
#The pixel value of each image ranges between 0 and 255
#Dividing each image by 255 will scale the values between 0 and 1. This is also known as normalization.
images = images/255
```

Now we will convert all the images into numpy array and normalize them.

```
label_id = np.array(label_id)
label_id.shape
```

```
(39209,)
```

Storing the label ids into numpy array and printing the shape. Here we can observe that their are 39209 label ids.

```
images.shape
```

```
(39209, 50, 50, 3)
```

Checking the shape of the images. Here we can see that their are 39209 images with a shape of (50,50,3.)

```
# Visualize the number of classes count
label_counts = pd.DataFrame(label_id).value_counts()
label_counts.head()
```

```
2     2250
1     2220
13    2160
12    2100
38    2070
Name: count, dtype: int64
```

Now we will observe images per class for checking whether the data is balanced or not. From the result we can say that data is balanced.

```
#Splitting the data
x_train, x_val, y_train, y_val = train_test_split(images, label_id , test_size = 0.2, random_state = 42)
```

The next step would be to split the data into training and validation with 80% of training data and 20% of validation data.

```
#keras has a built-in function for one-hot encoding.
y_train_cat = to_categorical(y_train)
y_val_cat = to_categorical(y_val)
```

Converting the classes column into categorical using to_categorical() function.

```
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (3,3), input_shape = x_train.shape[1:], activation = 'relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Conv2D(filters = 64, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(43, activation = 'softmax'))
```

Defining the model architecture. In this we will define all the layers with their input shape kernel size, activation, etc.

```
model.compile(loss = 'sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 64)        1792

 max_pooling2d (MaxPooling2  (None, 25, 25, 64)        0
 D)

 dropout (Dropout)           (None, 25, 25, 64)        0

 conv2d_1 (Conv2D)           (None, 23, 23, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 11, 11, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 11, 11, 64)        0

 flatten (Flatten)           (None, 7744)              0

 dense (Dense)               (None, 128)               991360

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 43)                5547

=================================================================
Total params: 1035627 (3.95 MB)
Trainable params: 1035627 (3.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the model using metrics, optimizer and loss as required and printing out the summary of the model.

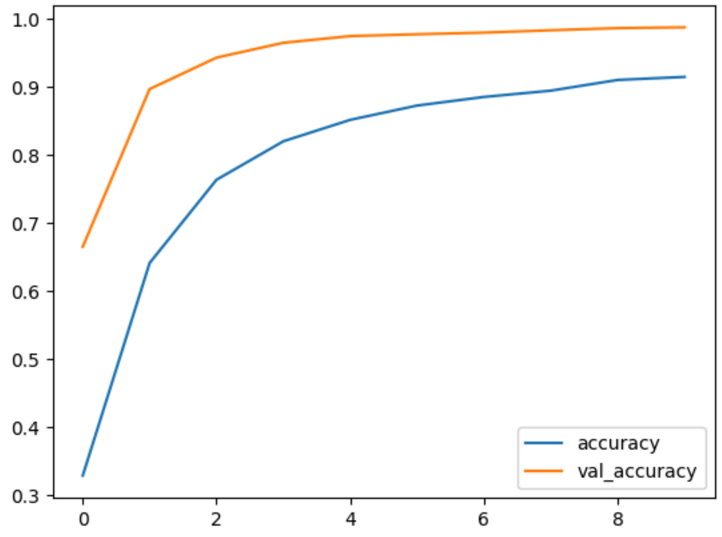
```
model.fit(x_train, y_train, epochs = 10, batch_size = 128, validation_data = (x_val, y_val), verbose = 2)
```

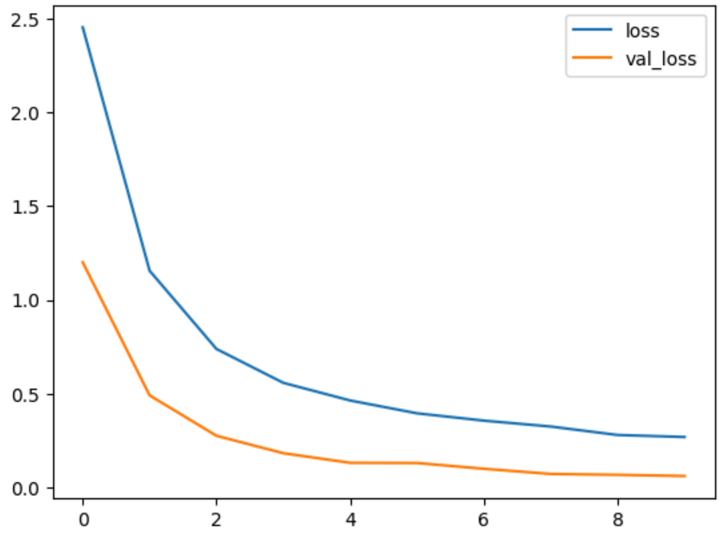
```
Epoch 1/10
246/246 - 235s - loss: 2.4534 - accuracy: 0.3290 - val_loss: 1.2003 - val_accuracy: 0.6649 - 235s/epoch - 953ms/step
Epoch 2/10
246/246 - 240s - loss: 1.1557 - accuracy: 0.6411 - val_loss: 0.4911 - val_accuracy: 0.8963 - 240s/epoch - 974ms/step
Epoch 3/10
246/246 - 237s - loss: 0.7384 - accuracy: 0.7632 - val_loss: 0.2756 - val_accuracy: 0.9425 - 237s/epoch - 965ms/step
Epoch 4/10
246/246 - 235s - loss: 0.5579 - accuracy: 0.8197 - val_loss: 0.1827 - val_accuracy: 0.9643 - 235s/epoch - 954ms/step
Epoch 5/10
246/246 - 231s - loss: 0.4637 - accuracy: 0.8512 - val_loss: 0.1314 - val_accuracy: 0.9741 - 231s/epoch - 939ms/step
Epoch 6/10
246/246 - 228s - loss: 0.3954 - accuracy: 0.8722 - val_loss: 0.1305 - val_accuracy: 0.9768 - 228s/epoch - 925ms/step
Epoch 7/10
246/246 - 233s - loss: 0.3563 - accuracy: 0.8848 - val_loss: 0.0996 - val_accuracy: 0.9792 - 233s/epoch - 949ms/step
Epoch 8/10
246/246 - 235s - loss: 0.3249 - accuracy: 0.8940 - val_loss: 0.0720 - val_accuracy: 0.9827 - 235s/epoch - 955ms/step
Epoch 9/10
246/246 - 230s - loss: 0.2798 - accuracy: 0.9097 - val_loss: 0.0674 - val_accuracy: 0.9858 - 230s/epoch - 936ms/step
Epoch 10/10
246/246 - 234s - loss: 0.2691 - accuracy: 0.9142 - val_loss: 0.0610 - val_accuracy: 0.9870 - 234s/epoch - 951ms/step
<keras.src.callbacks.History at 0x7f85b342d570>
```

Now we will fit the model and observe how our is getting trained on each epoch.

```
evaluation = pd.DataFrame(model.history.history)
evaluation[['accuracy', 'val_accuracy']].plot()
evaluation[['loss', 'val_loss']].plot()
```

```
<Axes: >
```





Next we will visualize the accuracy and loss per epoch. For this we will store the model history in the pandas dataframe and plot them.

```
test_path = 'traffic_sign_dataset/Test'
!rm traffic_sign_dataset/Test/GT-final_test.csv
```

Creating the variable which has path of test dataset. As we downloded the dataset we found out that their is a GT-final_test.csv file in the test images folder which cannot be processed. So, we will remove that file.

```
#defining a function that will scale images
from PIL import Image

def scaling(test_images, test_path):
    images = []

    image_path = test_images

    for x in image_path:
        img = Image.open(test_path + '/' + x)
        img = img.resize((50,50))
        img = np.array(img)
        images.append(img)

    #Converting images into numpy array
    images = np.array(images)
    #The pixel value of each image ranges between 0 and 255
    #Dividing each image by 255 will scale the values between 0 and 1. This is also known as normalization.
    images = images/255

    return images
```

Next step would we creating a function to resize the test images converting them into a numpy array and normalize them.

```
test_images = scaling(sorted(os.listdir(test_path)),test_path)
```

Calling the above created function on test images.

```
test = pd.read_csv('traffic_sign_dataset/Test.csv')
y_test = test['ClassId'].values
y_test
```

```
array([16,  1, 38, ...,  6,  7, 10])
```

Next we will read label ids from Test.csv and store the values of the class id in y_test variable.

```
import numpy as np
y_pred_prob = model.predict(test_images)
y_pred = np.argmax(y_pred_prob, axis=-1)
```

```
395/395 [==============================] - 27s 67ms/step
```

Now we will use the model to make predictions on our test images and save them in y_pred.

```
# Storing all lables
all_lables = ['Speed limit (20km/h)','Speed limit (30km/h)','Speed limit (50km/h)','Speed limit (60km/h)',
              'Speed limit (70km/h)','Speed limit (80km/h)','End of speed limit (80km/h)','Speed limit (100km/h)',
              'Speed limit (120km/h)','No passing','No passing for vechiles over 3.5 metric tons',
              'Right-of-way at the next intersection','Priority road','Yield','Stop','No vechiles',
              'Vechiles over 3.5 metric tons prohibited','No entry','General caution','Dangerous curve to the left',
              'Dangerous curve to the right','Double curve','Bumpy road','Slippery road','Road narrows on the right',
              'Road work','Traffic signals','Pedestrians','Children crossing','Bicycles crossing','Beware of ice/snow',
              'Wild animals crossing','End of all speed and passing limits','Turn right ahead','Turn left ahead',
              'Ahead only','Go straight or right','Go straight or left','Keep right','Keep left','Roundabout mandatory',
              'End of no passing','End of no passing by vechiles over 3.5 metric']
```

Storing the labels according to the image classes.

```
# Visualize test image
img = Image.open(test_path + '/00001.png')
img
```

Let's visualize test image

```
# Original label
print("Original label : ",all_lables[y_test[1]])
```

```
Original label :  Speed limit (30km/h)
```

Finding out original label for the image above.

```
# Predicted label
print("Predicted label : ",all_lables[y_pred[1]])
```

```
Predicted label :  Speed limit (30km/h)
```

Finding out the predicted label for the image above.

Conclusion:

We started with downloading the dataset, preprocessing it, created the model and found out the predictions using the model. During preprocessing we found that this dataset has 43 classes. Model reached an accuracy of 95%+ in just 50 epochs, we can further optimize the model using hyper parameter tuning and reach a higher accuracy.

Scope:

This model can be used in self driving cars which will enable them to automatically recognize traffic signs similarly the driver alert system inside cars will help and protect drivers by understanding the traffic signs around them.

```
Start coding or generate with AI.
```