Traffic\_Signs\_detection

# Importing Libraries[¶](#Importing-Libraries)

In [146]:

import os  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import random  
import glob  
from PIL import Image  
import tensorflow as tf  
from sklearn.model\_selection import train\_test\_split  
from keras.utils import to\_categorical  
from keras.models import Sequential  
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten,Dropout  
from keras.layers.normalization import BatchNormalization  
from sklearn.metrics import accuracy\_score  
import pandas as pd  
from sklearn.metrics import accuracy\_score

### Setting Data set Path[¶](#Setting-Data-set-Path)

In [2]:

Path = "D:\Dataset\Traffic Sign Detection"

### 2 list to store images and corresponding labels[¶](#X10ae5b42ea391dab566976a809216b1a82f2330)

In [29]:

X = []  
Y = []  
total\_class = 43  
cur\_directory = os.getcwd()

In [18]:

os.listdir(Path)

Out[18]:

['archive.zip', 'Meta', 'Meta.csv', 'Test', 'Test.csv', 'Train', 'Train.csv']

In [140]:

print(Path)

D:\Dataset\Traffic Sign Detection

### Extracting Images and labels, storring it into X,Y[¶](#X84f82b2586e2cac46dd61365b497bfc672ffac0)

In [30]:

#The dataset has folders from 0–42 i.e. 43 classes  
for index in range(total\_class):  
 img\_path = os.path.join(Path,'train',str(index))  
 #print(img\_path)  
 images = os.listdir(img\_path)  
 #print(images)  
#iterating on all the images of the index folder  
 for img in images:  
 try:  
 image = Image.open(img\_path + '\\' + img)  
 image = image.resize((30,30))  
 image = np.array(image)  
 X.append(image)  
 Y.append(index)  
 except:  
 print('Error loading image')

(39209, 30, 30, 3) (39209,)

### Size of X and Y[¶](#Size-of-X-and-Y)

In [143]:

X = np.array(X)  
Y = np.array(Y)  
print(X.shape, Y.shape)

(39209, 30, 30, 3) (39209,)

In [16]:

os.listdir(Path)

Out[16]:

['archive.zip', 'Meta', 'Meta.csv', 'Test', 'Test.csv', 'Train', 'Train.csv']

## Plotting random images from X[¶](#Plotting-random-images-from-X)

In [50]:

total\_images = 39209  
  
for i in range(5):  
 n = random.randint(0,39209)  
 image = X[n]  
 plt.imshow(image)  
 plt.xlabel("Class : " + str(Y[n]), fontcol)  
 plt.xticks([])  
 plt.yticks([])  
 plt.show()

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

In [ ]:

### Saving X,Y numpy array[¶](#Saving-X,Y-numpy-array)

In [10]:

#os.mkdir("Saved Weights")  
os.chdir("Saved Weights")  
np.save("X.npy",X)  
np.save("Y.npy",Y)

### Loading X,Y from Saved Weights folder[¶](#Loading-X,Y-from-Saved-Weights-folder)

In [144]:

print(os.getcwd())

C:\Users\sahib\OneDrive\Desktop\CB\_ML\Traffic Sign Detection project

In [9]:

X = np.load("X.npy")  
Y = np.load("Y.npy")

In [ ]:

### Dividing data into test and train split[¶](#Dividing-data-into-test-and-train-split)

In [20]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=42)  
print('Shape of x\_train: ', x\_train.shape, ' and y\_train: ',y\_train.shape)  
print('Shape of x\_test: ', x\_test.shape, ' and y\_test: ',y\_test.shape)  
#one hot encoding the labels  
y\_train = to\_categorical(y\_train, 43)  
y\_test = to\_categorical(y\_test, 43)

Shape of x\_train: (31367, 30, 30, 3) and y\_train: (31367,)  
Shape of x\_test: (7842, 30, 30, 3) and y\_test: (7842,)

In [21]:

print(y\_train[1]) #y\_train is now one hot ie its shape is 43,1 and the  
print(y\_train[1].shape)#class it belongs to will be 1 an rest of class will be 0

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
(43,)

In [ ]:

## Building Model[¶](#Building-Model)

In [24]:

model = Sequential()  
model.add(Conv2D(filters=32, kernel\_size=(5,5), activation='relu', input\_shape=x\_train.shape[1:]))  
model.add(Conv2D(filters=32, kernel\_size=(5,5), activation='relu'))  
model.add(MaxPool2D(pool\_size=(2, 2)))  
model.add(Dropout(rate=0.25))  
model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'))  
model.add(Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'))  
model.add(MaxPool2D(pool\_size=(2, 2)))  
model.add(Dropout(rate=0.25))  
model.add(Flatten())  
model.add(Dense(256, activation='relu'))  
model.add(Dropout(rate=0.5))  
model.add(Dense(43, activation='softmax'))  
model.compile(loss= 'categorical\_crossentropy', optimizer= 'adam' , metrics=['accuracy'])

In [25]:

model.summary()

Model: "sequential\_3"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
conv2d\_4 (Conv2D) (None, 26, 26, 32) 2432   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_5 (Conv2D) (None, 22, 22, 32) 25632   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_2 (MaxPooling2 (None, 11, 11, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_3 (Dropout) (None, 11, 11, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_6 (Conv2D) (None, 9, 9, 64) 18496   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_7 (Conv2D) (None, 7, 7, 64) 36928   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_3 (MaxPooling2 (None, 3, 3, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_4 (Dropout) (None, 3, 3, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten\_1 (Flatten) (None, 576) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 256) 147712   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_5 (Dropout) (None, 256) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 43) 11051   
=================================================================  
Total params: 242,251  
Trainable params: 242,251  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [62]:

os.chdir("..")  
os.chdir("Saved Weights/")

In [63]:

os.getcwd()

Out[63]:

'C:\\Users\\sahib\\OneDrive\\Desktop\\CB\_ML\\Traffic Sign Detection project\\Saved Weights'

In [64]:

epochs = 25  
history = model.fit(x\_train, y\_train, batch\_size=64, epochs=epochs,validation\_data=(x\_test, y\_test))  
model.save('traffic\_recognition.h5')

Epoch 1/25  
491/491 [==============================] - 6s 13ms/step - loss: 2.6142 - accuracy: 0.3353 - val\_loss: 1.2111 - val\_accuracy: 0.6831  
Epoch 2/25  
491/491 [==============================] - 6s 11ms/step - loss: 1.2126 - accuracy: 0.6473 - val\_loss: 0.6901 - val\_accuracy: 0.7942  
Epoch 3/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.8287 - accuracy: 0.7467 - val\_loss: 0.3751 - val\_accuracy: 0.9063  
Epoch 4/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.6323 - accuracy: 0.8068 - val\_loss: 0.3246 - val\_accuracy: 0.9110  
Epoch 5/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.5041 - accuracy: 0.8468 - val\_loss: 0.1950 - val\_accuracy: 0.9471  
Epoch 6/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.4346 - accuracy: 0.8675 - val\_loss: 0.1713 - val\_accuracy: 0.9538  
Epoch 7/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.3851 - accuracy: 0.8833 - val\_loss: 0.1232 - val\_accuracy: 0.9662  
Epoch 8/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.3397 - accuracy: 0.8958 - val\_loss: 0.1051 - val\_accuracy: 0.9672  
Epoch 9/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.3329 - accuracy: 0.9012 - val\_loss: 0.1005 - val\_accuracy: 0.9707  
Epoch 10/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2868 - accuracy: 0.9143 - val\_loss: 0.1094 - val\_accuracy: 0.9691  
Epoch 11/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2747 - accuracy: 0.9181 - val\_loss: 0.1061 - val\_accuracy: 0.9717  
Epoch 12/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2504 - accuracy: 0.9278 - val\_loss: 0.0760 - val\_accuracy: 0.9793  
Epoch 13/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.2441 - accuracy: 0.9296 - val\_loss: 0.0869 - val\_accuracy: 0.9756  
Epoch 14/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.2233 - accuracy: 0.9347 - val\_loss: 0.0709 - val\_accuracy: 0.9792  
Epoch 15/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2192 - accuracy: 0.9356 - val\_loss: 0.0672 - val\_accuracy: 0.9815  
Epoch 16/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2181 - accuracy: 0.9370 - val\_loss: 0.0549 - val\_accuracy: 0.9850  
Epoch 17/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.2028 - accuracy: 0.9401 - val\_loss: 0.0654 - val\_accuracy: 0.9811  
Epoch 18/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.2012 - accuracy: 0.9432 - val\_loss: 0.0551 - val\_accuracy: 0.9832  
Epoch 19/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.1926 - accuracy: 0.9439 - val\_loss: 0.0602 - val\_accuracy: 0.9830  
Epoch 20/25  
491/491 [==============================] - 6s 12ms/step - loss: 0.2069 - accuracy: 0.9428 - val\_loss: 0.0586 - val\_accuracy: 0.9848  
Epoch 21/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.1866 - accuracy: 0.9475 - val\_loss: 0.0495 - val\_accuracy: 0.9871  
Epoch 22/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.1822 - accuracy: 0.9503 - val\_loss: 0.0635 - val\_accuracy: 0.9811  
Epoch 23/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.1874 - accuracy: 0.9498 - val\_loss: 0.0516 - val\_accuracy: 0.9860  
Epoch 24/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.1712 - accuracy: 0.9539 - val\_loss: 0.0408 - val\_accuracy: 0.9878  
Epoch 25/25  
491/491 [==============================] - 6s 11ms/step - loss: 0.1754 - accuracy: 0.9518 - val\_loss: 0.0532 - val\_accuracy: 0.9858

In [ ]:

### Plotting Accuracy & Loss[¶](#Plotting-Accuracy-&-Loss)

In [65]:

plt.figure(0)  
plt.plot(history.history['accuracy'], label='training accuracy')  
plt.plot(history.history['val\_accuracy'], label='val accuracy')  
plt.title('Accuracy')  
plt.xlabel('epochs')  
plt.ylabel('accuracy')  
plt.legend()  
plt.figure(1)  
plt.plot(history.history['loss'], label='training loss')  
plt.plot(history.history['val\_loss'], label='val loss')  
plt.title('Loss')  
plt.xlabel('epochs')  
plt.ylabel('loss')  
plt.legend()

Out[65]:

<matplotlib.legend.Legend at 0x1a3748f3848>

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

In [ ]:

### Testing the model[¶](#Testing-the-model)

In [145]:

y\_test = pd.read\_csv(Path + '\\Test.csv')  
labels = y\_test['ClassId'].values  
img\_paths = y\_test['Path'].values  
test\_data=[]  
for img\_path in img\_paths:  
 img\_path = os.path.join(Path,img\_path)  
 image = Image.open(img\_path)  
 image = image.resize((30,30))  
 test\_data.append(np.array(image))  
test\_data = np.array(test\_data)  
pred = model.predict\_classes(test\_data)  
#Accuracy with the test data  
  
print(accuracy\_score(labels, pred))

0.956215360253365

### Model has 95% Accuracy on Test cases[¶](#Model-has-95%-Accuracy-on-Test-cases)

In [ ]:

### Load the trained model to classify sign[¶](#Load-the-trained-model-to-classify-sign)

In [26]:

model.load\_weights('traffic\_recognition.h5')

### Dictionary to label all traffic signs class[¶](#Xf5cb4cb587ab92e5bcb3d46027e703885add32f)

In [85]:

classes = { 1:'Speed limit (20km/h)',  
 2:'Speed limit (30km/h)',  
 3:'Speed limit (50km/h)',  
 4:'Speed limit (60km/h)',  
 5:'Speed limit (70km/h)',  
 6:'Speed limit (80km/h)',  
 7:'End of speed limit (80km/h)',  
 8:'Speed limit (100km/h)',  
 9:'Speed limit (120km/h)',  
 10:'No passing',  
 11:'No passing veh over 3.5 tons',  
 12:'Right-of-way at intersection',  
 13:'Priority road',  
 14:'Yield',  
 15:'Stop',  
 16:'No vehicles',  
 17:'Veh > 3.5 tons prohibited',  
 18:'No entry',  
 19:'General caution',  
 20:'Dangerous curve left',  
 21:'Dangerous curve right',  
 22:'Double curve',  
 23:'Bumpy road',  
 24:'Slippery road',  
 25:'Road narrows on the right',  
 26:'Road work',  
 27:'Traffic signals',  
 28:'Pedestrians',  
 29:'Children crossing',  
 30:'Bicycles crossing',  
 31:'Beware of ice/snow',  
 32:'Wild animals crossing',  
 33:'End speed + passing limits',  
 34:'Turn right ahead',  
 35:'Turn left ahead',  
 36:'Ahead only',  
 37:'Go straight or right',  
 38:'Go straight or left',  
 39:'Keep right',  
 40:'Keep left',  
 41:'Roundabout mandatory',  
 42:'End of no passing',  
 43:'End no passing veh > 3.5 tons' }

### Function To plot Image[¶](#Function-To-plot-Image)

In [34]:

def plot\_img(img,sign):  
 plt.imshow(img.reshape(-1,30,3))  
 plt.xlabel(sign)  
 plt.show()

### Testing model on test X\_test[¶](#Testing-model-on-test-X_test)

In [35]:

for i in range(6):  
 num = random.randint(0,x\_test.shape[0])  
   
 img = x\_test[num].reshape(-1,30,30,3)  
 #print(img.shape)  
 pred = model.predict\_classes(img)[0]  
 sign = classes[pred+1]  
 plot\_img(img,sign)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

![](data:image/png;base64;base64,)

## Testing Model on Traffic Signs from google[¶](#X3626840a23f5d4bd1e4f27f874156b79880be9a)

In [128]:

def classify(file\_path):  
 image = Image.open(file\_path)  
 image = image.convert("RGB")  
 image = image.resize((30,30))  
 image = np.expand\_dims(image, axis=0)  
 image = np.array(image)  
 pred = model.predict\_classes([image])[0]  
 sign = classes[pred+1]  
 plot\_img(image,sign)

In [139]:

for el in glob.glob('./my\_sign images/\*.jpg') + glob.glob('./my\_sign images/\*.png'):  
 print(el)  
 classify(el)  
   
#plt.savefig('new\_images.png',bbox\_inches='tight')

./my\_sign images\001.jpg

![](data:image/png;base64;base64,)

./my\_sign images\002.jpg

![](data:image/png;base64;base64,)

./my\_sign images\003.jpg

![](data:image/png;base64;base64,)

./my\_sign images\004.jpg

![](data:image/png;base64;base64,)

./my\_sign images\005.jpg

![](data:image/png;base64;base64,)

./my\_sign images\sign10.png

![](data:image/png;base64;base64,)

./my\_sign images\sign4.png

![](data:image/png;base64;base64,)

./my\_sign images\sign7.png

![](data:image/png;base64;base64,)

./my\_sign images\sign8.png

![](data:image/png;base64;base64,)

./my\_sign images\sign9.png

![](data:image/png;base64;base64,)