```
1
```

Connecting to google drive

```
1  from google.colab import drive
2  drive.mount("/content/drive/", force_remount=False)
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mou
```

Importing modules

```
1  import os
2  import numpy as np
3  import pandas as pd
4  import cv2
5  import matplotlib.pyplot as plt
6  import random
7  from itertools import chain
8  from skimage import color
9  from keras.models import Sequential
10 from keras.layers import Dense,Dropout,Flatten
11 from keras.utils.np_utils import to_categorical
12 from keras.optimizers import Adam
13 from keras.layers.convolutional import Conv2D,MaxPooling2D
14 from sklearn.model_selection import train_test_split
15 from keras.preprocessing.image import ImageDataGenerator
```

Parametres

```
1  path = "/content/drive/My Drive/Colab Notebooks/traffic signs python
2  labels = "/content/drive/My Drive/Colab Notebooks/traffic signs pyth
3  batchSizeVal = 50 # how many to process together (blocks of 50)
4  stepsPerEpoch = 200
5  epochsVal = 60 # the larger epochs the more time to train
```

```
imageDimention = (32,32,3) # 3 for RGB colors
testRatio = 0.2 # for each 1000 images 200 images will be kept for t
validationRatio = 0.2 # the remaining 800 20% of'em will kept for va
batch_size_value = 10
steps_per_epoch_value = None
epochs_value = 15
```

Importing all images in one array

```
count = 0
images = []
classNumber = []
myList = os.listdir(path)
print(">> Total Classes Detected:",len(myList))
numberOfClasses=len(myList)
print("Importing Classes...")


for x in range (len(myList)):
    for y in os.listdir(path+"/"+str(count)):
        images.append(cv2.imread(path+"/"+str(count)+"/"+y))
        classNumber.append(count)
    print('>>> class ',count, end ="\n")
    count +=1
print(" ")
images = np.array(images)
classNumber = np.array(classNumber)
len(images)
np.save('drive/My Drive/Colab Notebooks/traffic signs python/traffic
print('\nimages successfully saved !')
```

```
 imageNumberInClass=[]
 count = 0
 numberOfClasses=len(os.listdir(path))
for i in range(numberOfClasses):
    for j in os.listdir(path+'/'+str(count)):
        imageNumberInClass.append(count)
    print(count,'|',end=' ')
    count+=1
print(' ')
```

```
10
11  imageNumberInClass=np.array(imageNumberInClass)
12  images = np.load('/content/drive/My Drive/Colab Notebooks/traffic si
13  assert(len(imageNumberInClass) == images.shape[0])
14  print('Done!')
```

```
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19
Done!
```

Splitting data

```
1  x_train,x_test,y_train,y_test = train_test_split(images,imageNumberI
2  x_train,x_validation,y_train,y_validation = train_test_split(x_train
```

Check if there is a mismatch in data

```
1  print('data shapes',end='\n')
2
3  print(f'x_train shape :{x_train.shape}',end='\n')
4  print(f'y_train shape :{y_train.shape}',end='\n\n')
5
6  print(f'x_test shape :{x_test.shape}',end='\n')
7  print(f'y_test shape :{y_test.shape}',end='\n\n')
8
9  print(f'x_validatio shape :{x_validation.shape}',end='\n')
10 print(f'y_valoidation shape :{y_validation.shape}',end='\n\n')
11
12 assert(x_train.shape[0] == y_train.shape[0])
13 assert(x_test.shape[0] == y_test.shape[0])
14 assert(x_validation.shape[0] == y_validation.shape[0])
15
16 assert(x_train.shape[1:] == (imageDimention))
17 assert(x_test.shape[1:] == (imageDimention))
18 assert(x_validation.shape[1:] == (imageDimention))
19 print('everything is correct ')
```

```
data shapes
x_train shape :(22271, 32, 32, 3)
```

```
y_train shape :(22271,)

x_test shape :(6960, 32, 32, 3)
y_test shape :(6960,)

x_validatio shape :(5568, 32, 32, 3)
y_valoidation shape :(5568,)

everything is correct
```

Reading csv file and make some plots

```
1
2  imageNumberInClass = [len(os.listdir(path+'/'+str(i))) for i in rang
3  data = pd.read_csv(labels)
4  print(data.head(5))
5  plt.figure(figsize=(15,8))
6  plt.bar(range(len(imageNumberInClass)),imageNumberInClass,width=.8)
7  imageNumberInClass=np.array(imageNumberInClass)
8  plt.xlabel('class index')
9  plt.ylabel('number of samples')
10 plt.show()
```

```
   ClassId                  Name
0        0  Speed limit (20km/h)
1        1  Speed limit (30km/h)
2        2  Speed limit (50km/h)
3        3  Speed limit (60km/h)
4        4  Speed limit (70km/h)
```

Preprocessing images to work with

```
# run it just one time to avoid getting erro
def grayscale(image):       # convert the image to grayscale
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    return image
def equalize(image):       # standarization of lightning in the imag
    image = cv2.equalizeHist(image)
    return image
def preprocessing(image):
    image = grayscale(image)
    image = equalize(image)
    image = image/255          # standarization of the pixels each i
    return image

x_train = np.array(list(map(preprocessing,x_train)))
x_test = np.array(list(map(preprocessing,x_test)))
x_validation = np.array(list(map(preprocessing,x_validation)))
```

add a depth of 1

```
x_train = x_train.reshape(x_train.shape[0],x_train.shape[1],x_train.
x_validation = x_validation.reshape(x_validation.shape[0],x_validati
#y_validation = y_validation.reshape(y_validation.shape[0],y_validat
x_test = x_test.reshape(x_test.shape[0],x_test.shape[1],x_test.shape
```

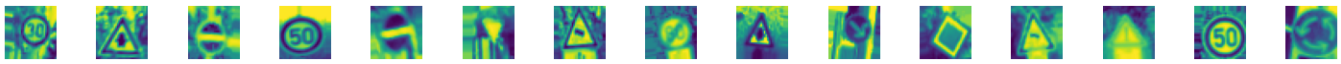
Images augmentation (make them more generic)

```
dataGen= ImageDataGenerator(width_shift_range=0.1,   # 10%
```

```
                                height_shift_range=0.1,
                                zoom_range=0.2,  # 0.2 means zoom goes f
                                shear_range=0.1,  # MAGNITUDE OF SHEAR A
                                rotation_range=10)  # rotation of 10 deg

dataGen.fit(x_train)
batches= dataGen.flow(x_train,y_train,batch_size=20)  # requiesting
x_batch,y_batch = next(batches)
```

Plotting some augmented images

```
fig,axs=plt.subplots(1,15,figsize=(20,5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(x_batch[i].reshape(imageDimention[0],imageDimentio
    axs[i].axis('off')
plt.show()

y_train = to_categorical(y_train,43)
y_validation = to_categorical(y_validation,43)
y_test = to_categorical(y_test,43)
```



Convolution neural network function

```
def myModel():
    no_Of_Filters=60
    size_of_Filter=(5,5) # THIS IS THE KERNEL THAT MOVE AROUND THE I
                         # THIS WOULD REMOVE 2 PIXELS FROM EACH BORD
    size_of_Filter2=(3,3)
    size_of_pool=(2,2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MO
    no_Of_Nodes = 500   # NO. OF NODES IN HIDDEN LAYERS
    model= Sequential()
    model.add((Conv2D(no_Of_Filters,size_of_Filter,input_shape=(imag
    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='rel
```

```
    model.add(MaxPooling2D(pool_size=size_of_pool)) # DOES NOT EFFEC

    model.add((Conv2D(no_Of_Filters//2, size_of_Filter2,activation='
    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activatio
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(no_Of_Nodes,activation='relu'))
    model.add(Dropout(0.5)) # INPUTS NODES TO DROP WITH EACH UPDATE
    model.add(Dense(numberOfClasses,activation='softmax')) # OUTPUT
    # COMPILE MODEL
    model.compile(Adam(lr=0.001),loss='categorical_crossentropy',met
    print('model compiled successfully')
    return model
```

Training the model

```
print(x_validation.shape)
print(y_validation.shape)
model = myModel()
print(model.summary())
print(x_validation.shape,'\n',y_validation.shape)

data_generator = dataGen.flow(x_train,y_train,batch_size=batch_size_
history=model.fit( data_generator,
                           steps_per_epoch=steps_per_epoch_value,
                           epochs=epochs_value,
                           shuffle=True,
                           validation_data=(x_validation,y_validat
                           )
print('Done')
```

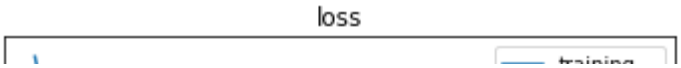
```
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
```

```
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 30)          0
_________________________________________________________________
flatten (Flatten)            (None, 480)               0
_________________________________________________________________
dense (Dense)                (None, 500)               240500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
(5568, 32, 32, 1)
 (5568, 43)
Epoch 1/15
2228/2228 [==============================] - 223s 100ms/step - loss: 2.1495 - accuracy
Epoch 2/15
2228/2228 [==============================] - 224s 100ms/step - loss: 1.0100 - accuracy
Epoch 3/15
2228/2228 [==============================] - 222s 100ms/step - loss: 0.6962 - accuracy
Epoch 4/15
2228/2228 [==============================] - 226s 101ms/step - loss: 0.5739 - accuracy
Epoch 5/15
2228/2228 [==============================] - 222s 99ms/step - loss: 0.4914 - accuracy
Epoch 6/15
2228/2228 [==============================] - 221s 99ms/step - loss: 0.4543 - accuracy
Epoch 7/15
2228/2228 [==============================] - 224s 101ms/step - loss: 0.4237 - accuracy
Epoch 8/15
2228/2228 [==============================] - 220s 99ms/step - loss: 0.4034 - accuracy
Epoch 9/15
2228/2228 [==============================] - 224s 100ms/step - loss: 0.3778 - accuracy
Epoch 10/15
2228/2228 [==============================] - 221s 99ms/step - loss: 0.3617 - accuracy
Epoch 11/15
2228/2228 [==============================] - 219s 98ms/step - loss: 0.3453 - accuracy
Epoch 12/15
2228/2228 [==============================] - 226s 102ms/step - loss: 0.3373 - accuracy
Epoch 13/15
2228/2228 [==============================] - 218s 98ms/step - loss: 0.3318 - accuracy
Epoch 14/15
2228/2228 [==============================] - 220s 99ms/step - loss: 0.3240 - accuracy
Epoch 15/15
```

Making plots of loss & accuracy

```
1  plt.figure(1)
```

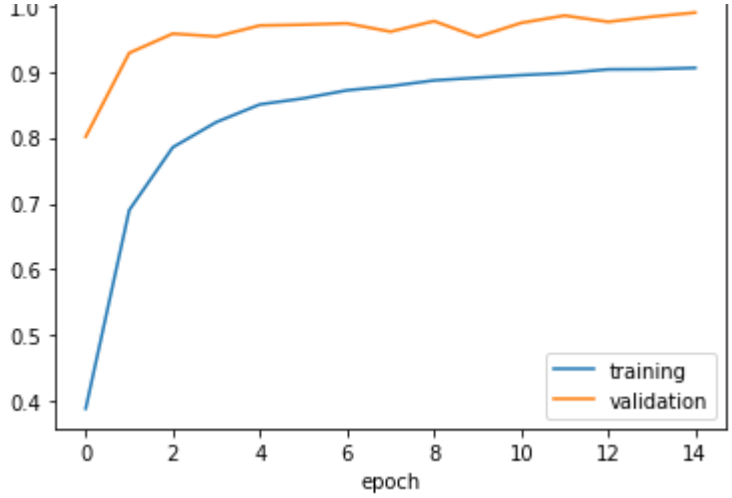
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')

plt.figure(2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title('accuracy')
plt.xlabel('epoch')

plt.show()
score = model.evaluate(x_test,y_test,verbose=0)
print('test score >> ',score[0])
print('test accuracy >> ',score[1])
```



Save the model

```
file_path = 'drive/My Drive/Colab Notebooks/traffic signs python/tra
model.save(filepath=file_path,overwrite=True,include_optimizer=True,
print('model saved !')
```

```
model saved !
```




```
test score >>  0.03999212011694908
test accuracy >>  0.991235613822937
```