

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

```

In [40]: from PIL import Image
import os

path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset2\\traffic_Data\\DATA"
# path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset3\\Images"
# path = "C:\\Users\\Ishaan\\AI project model (3)\\Dataset5\\train"

i=0
# r=root, d=directories, f = files
for r, d, f in os.walk(path):
    for file in f:
        #         if file.endswith('.png'):
        #             pat=os.path.join(r, file)
        #             with Image.open(pat) as im:
        #                 if im.size!=(32, 32):
        #                     im=im.resize((32, 32),Image.LANCZOS)
        #                     im.save(pat.replace(".png", ".jpg"))
        #             os.remove(pat)
        #             i+=1
        #             print(i,end='\r')

        if file.endswith('.jpg'):
            pat=os.path.join(r, file)
            with Image.open(pat) as im:
                if im.size!=(32, 32):
                    im=im.resize((32, 32),Image.LANCZOS)
            #                 im.save(pat)
            im.save(pat.replace(".jpg", ".png"))
            os.remove(pat)
            i+=1
            print(i,end='\r')

        elif file.endswith('.ppm'):
            pat=os.path.join(r, file)
            with Image.open(pat) as im:
                im.save(pat.replace(".ppm", ".png"))
            os.remove(pat)
            i+=1
            print(i,end='\r')

        elif file.endswith('.csv'):
            pat=os.path.join(r, file)

```

```
os.remove(pat)
```

4170

```
In [2]: ##### Parameters #####

path = "Dataset2/traffic_Data/DATA" # folder with all the class folders
labelFile = 'Dataset2/labels.csv' # file with all names of classes
batch_size_val=50 # how many to process together
steps_per_epoch_val=2000
epochs_val=20
imageDimesions = (32,32,3)
testRatio = 0.2 # if 1000 images split will 200 for testing
validationRatio = 0.2 # if 1000 images 20% of remaining 800 will be 160 for validation
#####
```

```
In [3]: ##### Importing of the Images

count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses=len(myList)
print("Importing Classes.....")
for x in range (0,len(myList)):
    myPicList = os.listdir(path+"/"+str(count))
    for y in myPicList:
        curImg = cv2.imread(path+"/"+str(count)+"/"+y)
        images.append(curImg)
        classNo.append(count)
    print(count, end = " ")
    count +=1
print(" ")
images = np.array(images)
classNo = np.array(classNo)
```

Total Classes Detected: 58

Importing Classes.....

```
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 47 48 49 50 51 52 53 54 55 56 57
```

```
In [4]: ##### Split 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)

# X_train = ARRAY OF IMAGES TO TRAIN
# y_train = CORRESPONDING CLASS ID
```

```
In [5]: ##### TO CHECK IF NUMBER OF IMAGES MATCHES TO NUMBER OF LABELS FOR EACH DATA SET
print("Data Shapes")
print("Train",end = "");print(X_train.shape,y_train.shape)
print("Validation",end = "");print(X_validation.shape,y_validation.shape)
print("Test",end = "");print(X_test.shape,y_test.shape)
assert(X_train.shape[0]==y_train.shape[0]), "The number of images in not equal to the number of lables in traini
assert(X_validation.shape[0]==y_validation.shape[0]), "The number of images in not equal to the number of lables
assert(X_test.shape[0]==y_test.shape[0]), "The number of images in not equal to the number of lables in test set
assert(X_train.shape[1:]==(imageDimesions)), " The dimesions of the Training images are wrong "
assert(X_validation.shape[1:]==(imageDimesions)), " The dimesionas of the Validation images are wrong "
assert(X_test.shape[1:]==(imageDimesions)), " The dimesionas of the Test images are wrong"
```

Data Shapes

Train(2668, 32, 32, 3) (2668,)

Validation(668, 32, 32, 3) (668,)

Test(834, 32, 32, 3) (834,)

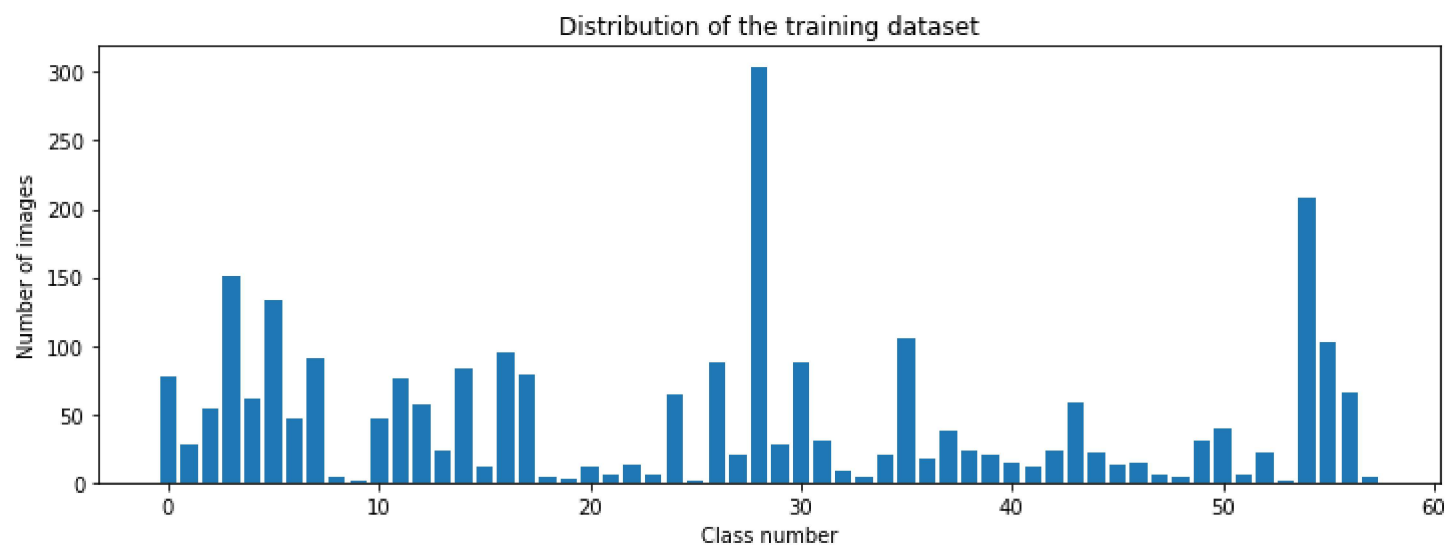
```
In [6]: ##### READ CSV FILE
data=pd.read_csv(labelFile)
print("data shape ",data.shape,type(data))

##### DISPLAY SOME SAMPLES IMAGES OF ALL THE CLASSES
num_of_samples = []
cols = 5
num_classes = noOfClasses
fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 300))
fig.tight_layout()
for i in range(cols):
    for j,row in data.iterrows():
        x_selected = X_train[y_train == j]
        axs[j][i].imshow(x_selected[random.randint(0, len(x_selected)- 1), :, :], cmap=plt.get_cmap("gray"))
        axs[j][i].axis("off")
        if i == 2:
            axs[j][i].set_title(str(j)+ "-" +row["Name"])
            num_of_samples.append(len(x_selected))
```

```
data shape (58, 2) <class 'pandas.core.frame.DataFrame'>
```

```
In [7]: ##### DISPLAY A BAR CHART SHOWING NO OF SAMPLES FOR EACH CATEGORY
print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()
```

```
[77, 28, 54, 151, 62, 133, 47, 91, 4, 1, 47, 76, 57, 23, 83, 12, 95, 79, 4, 3, 12, 6, 13, 6, 65, 1, 88, 20, 30,
4, 28, 88, 30, 8, 4, 20, 105, 17, 38, 23, 20, 15, 12, 23, 59, 22, 13, 14, 6, 4, 30, 40, 5, 21, 1, 208, 102, 6,
6, 4]
```



```
In [8]: ##### PREPROCESSING THE IMAGES

def grayscale(img):
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    return img
def equalize(img):
    img =cv2.equalizeHist(img)
    return img
def preprocessing(img):
    img = grayscale(img)      # CONVERT TO GRAYSCALE
    img = equalize(img)       # STANDARDIZE THE LIGHTING IN AN IMAGE
    img = img/255             # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD OF 0 TO 255
    return img

X_train=np.array(list(map(preprocessing,X_train))) # TO IRETATE AND PREPROCESS ALL IMAGES
X_validation=np.array(list(map(preprocessing,X_validation)))
X_test=np.array(list(map(preprocessing,X_test)))
cv2.imshow("GrayScale Images",X_train[random.randint(0,len(X_train)-1)]) # TO CHECK IF THE TRAINING IS DONE PROF
```

```
In [9]: ##### ADD A DEPTH OF 1
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_validation=X_validation.reshape(X_validation.shape[0],X_validation.shape[1],X_validation.shape[2],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)
```

```

In [10]: ##### AUGMENTATION OF IMAGES: TO MAKE IT MORE GENERIC
dataGen= ImageDataGenerator(width_shift_range=0.1, # 0.1 = 10% IF MORE THAN 1 E.G 10 THEN IT REFERS TO NO
                             height_shift_range=0.1,
                             zoom_range=0.2, # 0.2 MEANS CAN GO FROM 0.8 TO 1.2
                             shear_range=0.1, # MAGNITUDE OF SHEAR ANGLE
                             rotation_range=10) # DEGREES

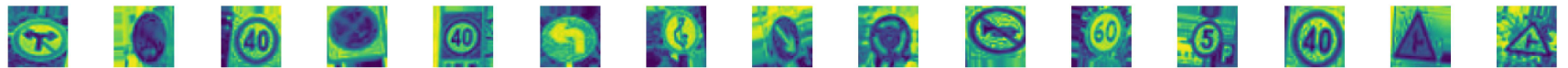
dataGen.fit(X_train)
batches= dataGen.flow(X_train,y_train,batch_size=20) # REQUESTING DATA GENERATOR TO GENERATE IMAGES BATCH SIZE
X_batch,y_batch = next(batches)

# TO SHOW AGMENTED IMAGE SAMPLES
fig,axs=plt.subplots(1,15,figsize=(20,5))
fig.tight_layout()

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(imageDimesions[0],imageDimesions[1]))
    axs[i].axis('off')
plt.show()

y_train = to_categorical(y_train,noOfClasses)
y_validation = to_categorical(y_validation,noOfClasses)
y_test = to_categorical(y_test,noOfClasses)

```




```

In [11]: ##### CONVOLUTION NEURAL NETWORK MODEL
def myModel():
    no_Of_Filters=60
    size_of_Filter=(5,5) # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE TO GET THE FEATURES.
                        # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32 IMAGE
    size_of_Filter2=(3,3)
    size_of_pool=(2,2) # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE, TO REDUCE OVERFITTING
    no_Of_Nodes = 500 # NO. OF NODES IN HIDDEN LAYERS
    model= Sequential()
    model.add((Conv2D(no_Of_Filters,size_of_Filter,input_shape=(imageDimesions[0],imageDimesions[1],1),activation='relu'))
    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu'))
    model.add(MaxPooling2D(pool_size=size_of_pool)) # DOES NOT EFFECT THE DEPTH/NO OF FILTERS

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

    model.add(Flatten())
    model.add(Dense(no_Of_Filters,activation='relu'))
    model.add(Dropout(0.5)) # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL 0 NONE
    model.add(Dense(noOfClasses,activation='softmax')) # OUTPUT LAYER
    # COMPILE MODEL
    model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
    return model

```

```
In [12]: ##### TRAIN
model = myModel()
print(model.summary())
history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val),steps_per_epoch=len(X_train))
# history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val),steps_per_epoch=steps_per_
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 60)	1560
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 (MaxPooling2D)	(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, 58)	29058
=====		
Total params: 385,538		
Trainable params: 385,538		
Non-trainable params: 0		

```
C:\Users\Ishaan\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr`
` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
C:\Users\Ishaan\AppData\Local\Temp\ipykernel_7824\37330348.py:4: UserWarning: `Model.fit_generator` is depr
ecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val),steps_per_epoch=len(X
_train)//batch_size_val,epochs=epochs_val,validation_data=(X_validation,y_validation),shuffle=1)
```

None

Epoch 1/20

53/53 [=====] - 17s 262ms/step - loss: 3.7336 - accuracy: 0.0844 - val_loss: 3.6011 - val_accuracy: 0.0943

Epoch 2/20

53/53 [=====] - 13s 243ms/step - loss: 3.4246 - accuracy: 0.1379 - val_loss: 3.0164 - val_accuracy: 0.2111

Epoch 3/20

53/53 [=====] - 13s 249ms/step - loss: 2.8180 - accuracy: 0.2739 - val_loss: 2.0804 - val_accuracy: 0.4865

Epoch 4/20

53/53 [=====] - 13s 240ms/step - loss: 2.3270 - accuracy: 0.3728 - val_loss: 1.6393 - val_accuracy: 0.5793

Epoch 5/20

53/53 [=====] - 13s 239ms/step - loss: 1.9851 - accuracy: 0.4526 - val_loss: 1.3385 - val_accuracy: 0.6362

Epoch 6/20

53/53 [=====] - 13s 237ms/step - loss: 1.7372 - accuracy: 0.5118 - val_loss: 1.2869 - val_accuracy: 0.6602

Epoch 7/20

53/53 [=====] - 13s 240ms/step - loss: 1.6331 - accuracy: 0.5382 - val_loss: 0.8778 - val_accuracy: 0.7485

Epoch 8/20

53/53 [=====] - 13s 243ms/step - loss: 1.4228 - accuracy: 0.5894 - val_loss: 0.7897 - val_accuracy: 0.7710

Epoch 9/20

53/53 [=====] - 12s 234ms/step - loss: 1.3310 - accuracy: 0.6020 - val_loss: 0.6953 - val_accuracy: 0.7814

Epoch 10/20

53/53 [=====] - 13s 237ms/step - loss: 1.2271 - accuracy: 0.6375 - val_loss: 0.6873 - val_accuracy: 0.7799

Epoch 11/20

53/53 [=====] - 13s 240ms/step - loss: 1.1509 - accuracy: 0.6623 - val_loss: 0.6470 - val_accuracy: 0.8278

Epoch 12/20

```
53/53 [=====] - 13s 237ms/step - loss: 1.0577 - accuracy: 0.6833 - val_loss: 0.4857 -  
val_accuracy: 0.8533  
Epoch 13/20  
53/53 [=====] - 13s 236ms/step - loss: 0.9757 - accuracy: 0.7193 - val_loss: 0.4679 -  
val_accuracy: 0.8743  
Epoch 14/20  
53/53 [=====] - 13s 237ms/step - loss: 0.9299 - accuracy: 0.7223 - val_loss: 0.3892 -  
val_accuracy: 0.8743  
Epoch 15/20  
53/53 [=====] - 13s 238ms/step - loss: 0.9103 - accuracy: 0.7273 - val_loss: 0.3513 -  
val_accuracy: 0.9057  
Epoch 16/20  
53/53 [=====] - 13s 243ms/step - loss: 0.8448 - accuracy: 0.7441 - val_loss: 0.3339 -  
val_accuracy: 0.8982  
Epoch 17/20  
53/53 [=====] - 13s 248ms/step - loss: 0.8356 - accuracy: 0.7506 - val_loss: 0.3598 -  
val_accuracy: 0.8967  
Epoch 18/20  
53/53 [=====] - 13s 245ms/step - loss: 0.7600 - accuracy: 0.7712 - val_loss: 0.2945 -  
val_accuracy: 0.9207  
Epoch 19/20  
53/53 [=====] - 13s 249ms/step - loss: 0.7408 - accuracy: 0.7823 - val_loss: 0.2610 -  
val_accuracy: 0.9431  
Epoch 20/20  
53/53 [=====] - 13s 249ms/step - loss: 0.7170 - accuracy: 0.7892 - val_loss: 0.2507 -  
val accuracy: 0.9401
```

```
In [14]: pred = model.predict(X_test)
pred=np.argmax(pred, axis=1)

y_test1=np.argmax(y_test, axis=1)

precision = precision_score(y_test1, pred, pos_label='positive', average='macro')
recall = recall_score(y_test1, pred, pos_label='positive', average='macro')
```

27/27 [=====] - 1s 44ms/step

C:\Users\Ishaan\anaconda3\lib\site-packages\sklearn\metrics_classification.py:1370: UserWarning: Note that pos_label (set to 'positive') is ignored when average != 'binary' (got 'macro'). You may use labels=[pos_label] to specify a single positive class.

warnings.warn(

C:\Users\Ishaan\anaconda3\lib\site-packages\sklearn\metrics_classification.py:1318: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

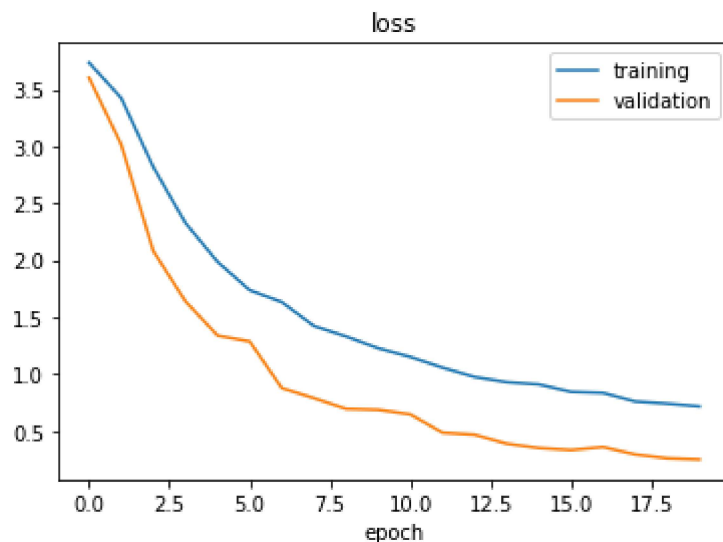
_warn_prf(average, modifier, msg_start, len(result))

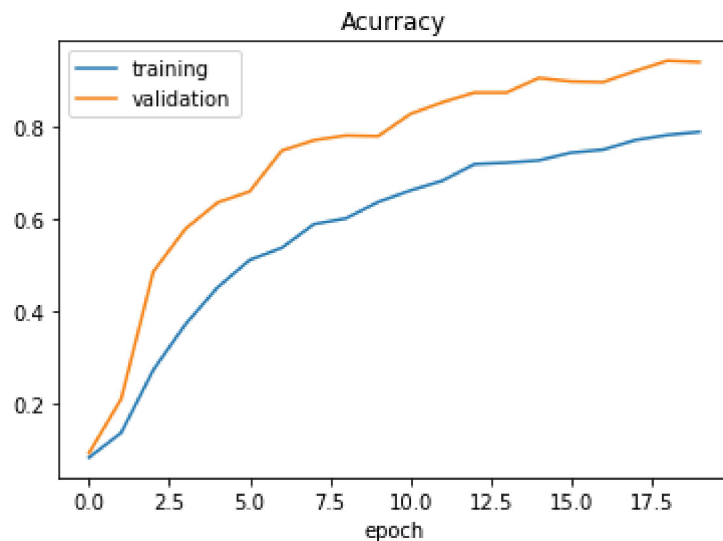
C:\Users\Ishaan\anaconda3\lib\site-packages\sklearn\metrics_classification.py:1370: UserWarning: Note that pos_label (set to 'positive') is ignored when average != 'binary' (got 'macro'). You may use labels=[pos_label] to specify a single positive class.

warnings.warn(

```
In [15]: ##### PLOT
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('Acurracy')
plt.xlabel('epoch')
plt.show()
score =model.evaluate(X_test,y_test,verbose=0)
print('Test Score:',score[0])
print('Test Accuracy:',score[1])

print('\nPrecision: ',precision)
print('Recall: ',recall)
```





Test Score: 0.27054160833358765
Test Accuracy: 0.9340527653694153

Precision: 0.8200461998917377
Recall: 0.7752237489044583

```
In [16]: # STORE THE MODEL AS A PICKLE OBJECT
# pickle_out= open("model_trained.sav","wb") # wb = WRITE BYTE
# pickle.dump(model,pickle_out)
# pickle_out.close()
# import joblib
# # save the model to disk
# filename = './finalized_model.sav'
# joblib.dump(model, filename)

from keras.models import load_model
model.save('model_d2_3.h5')
```

In []:

