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 [17]:
```
################## Parameters ####################

path = "Dataset3/Images" # folder with all the class folders
labelFile = 'Dataset3/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 [18]:
```
################################ 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 [19]:
```
############################### 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 [20]:
```
############################### 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(8940, 32, 32, 3) (8940,)
Validation(2236, 32, 32, 3) (2236,)
Test(2795, 32, 32, 3) (2795,)
```

In [21]:
```
############################### 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 [22]:
```
############################### 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()
```

```
[137, 147, 127, 120, 114, 207, 134, 113, 375, 122, 130, 130, 126, 133, 123, 126, 114, 137, 135, 129, 401, 128,
124, 127, 136, 136, 130, 125, 112, 130, 134, 124, 138, 125, 121, 135, 131, 139, 132, 138, 136, 134, 119, 125,
112, 126, 99, 104, 37, 125, 382, 151, 399, 369, 405, 123, 127, 122]
```

In [23]:
```
############################### 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 PROP
```

In [24]:
```
############################### 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 [25]:
```
############################### AUGMENTATAION OF IMAGES: TO MAKEIT MORE GENERIC
dataGen= ImageDataGenerator(width_shift_range=0.1,   # 0.1 = 10%     IF MORE THAN 1 E.G 10 THEN IT REFFERS 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 GENRATOR 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 [26]:
```
############################### 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),activatic
    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_Nodes,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 [27]:
```
############################### 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_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 28, 28, 60)        1560

 conv2d_5 (Conv2D)            (None, 24, 24, 60)        90060

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 60)       0
 2D)

 conv2d_6 (Conv2D)            (None, 10, 10, 30)        16230

 conv2d_7 (Conv2D)            (None, 8, 8, 30)          8130

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 dropout_2 (Dropout)          (None, 4, 4, 30)          0

 flatten_1 (Flatten)          (None, 480)               0

 dense_2 (Dense)              (None, 500)               240500

 dropout_3 (Dropout)          (None, 500)               0

 dense_3 (Dense)              (None, 58)                29058

=================================================================
Total params: 385,538
Trainable params: 385,538
Non-trainable params: 0
_________________________________________________________________
```

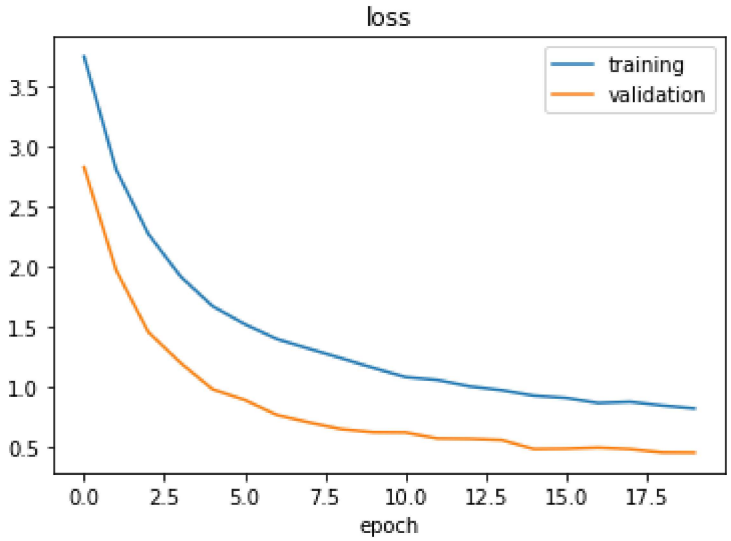
```
C:\Users\Ishaan\anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr
` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
C:\Users\Ishaan\AppData\Local\Temp\ipykernel_7824\37330348.py:4: UserWarning: `Model.fit_generator` 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
178/178 [==============================] - 49s 261ms/step - loss: 3.7439 - accuracy: 0.0919 - val_loss: 2.8235
- val_accuracy: 0.2496
Epoch 2/20
178/178 [==============================] - 45s 251ms/step - loss: 2.8034 - accuracy: 0.2442 - val_loss: 1.9680
- val_accuracy: 0.4718
Epoch 3/20
178/178 [==============================] - 44s 244ms/step - loss: 2.2701 - accuracy: 0.3625 - val_loss: 1.4547
- val_accuracy: 0.5997
Epoch 4/20
178/178 [==============================] - 44s 246ms/step - loss: 1.9176 - accuracy: 0.4472 - val_loss: 1.1995
- val_accuracy: 0.6740
Epoch 5/20
178/178 [==============================] - 44s 245ms/step - loss: 1.6697 - accuracy: 0.5067 - val_loss: 0.9789
- val_accuracy: 0.7267
Epoch 6/20
178/178 [==============================] - 44s 249ms/step - loss: 1.5204 - accuracy: 0.5414 - val_loss: 0.8910
- val_accuracy: 0.7464
Epoch 7/20
178/178 [==============================] - 44s 245ms/step - loss: 1.3967 - accuracy: 0.5744 - val_loss: 0.7655
- val_accuracy: 0.7773
Epoch 8/20
178/178 [==============================] - 44s 247ms/step - loss: 1.3162 - accuracy: 0.5996 - val_loss: 0.7035
- val_accuracy: 0.7858
Epoch 9/20
178/178 [==============================] - 44s 248ms/step - loss: 1.2378 - accuracy: 0.6153 - val_loss: 0.6480
- val_accuracy: 0.8157
Epoch 10/20
178/178 [==============================] - 46s 261ms/step - loss: 1.1573 - accuracy: 0.6415 - val_loss: 0.6215
- val_accuracy: 0.8059
Epoch 11/20
178/178 [==============================] - 49s 273ms/step - loss: 1.0810 - accuracy: 0.6639 - val_loss: 0.6199
- val_accuracy: 0.8086
Epoch 12/20
```

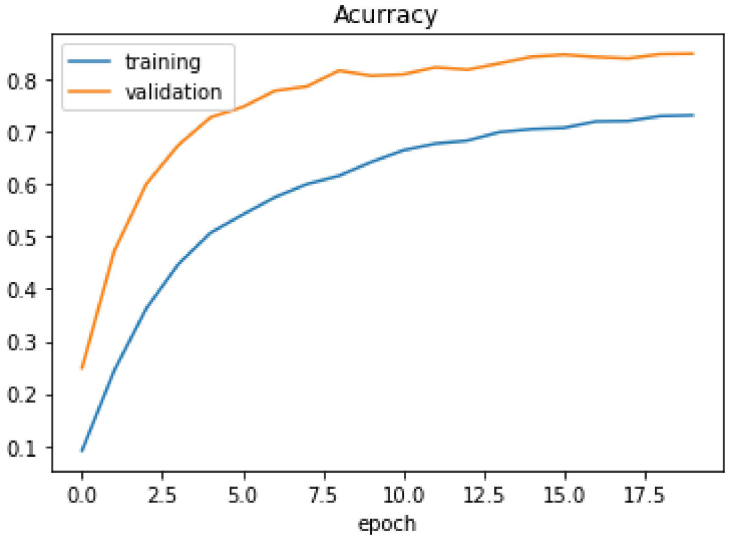
```
178/178 [==============================] - 29s 163ms/step - loss: 1.0564 - accuracy: 0.6766 - val_loss: 0.5699
- val_accuracy: 0.8220
Epoch 13/20
178/178 [==============================] - 29s 164ms/step - loss: 1.0034 - accuracy: 0.6823 - val_loss: 0.5671
- val_accuracy: 0.8175
Epoch 14/20
178/178 [==============================] - 37s 206ms/step - loss: 0.9712 - accuracy: 0.6987 - val_loss: 0.5572
- val_accuracy: 0.8296
Epoch 15/20
178/178 [==============================] - 38s 215ms/step - loss: 0.9263 - accuracy: 0.7044 - val_loss: 0.4819
- val_accuracy: 0.8421
Epoch 16/20
178/178 [==============================] - 38s 212ms/step - loss: 0.9073 - accuracy: 0.7067 - val_loss: 0.4846
- val_accuracy: 0.8462
Epoch 17/20
178/178 [==============================] - 38s 214ms/step - loss: 0.8655 - accuracy: 0.7189 - val_loss: 0.4938
- val_accuracy: 0.8417
Epoch 18/20
178/178 [==============================] - 38s 211ms/step - loss: 0.8750 - accuracy: 0.7198 - val_loss: 0.4814
- val_accuracy: 0.8390
Epoch 19/20
178/178 [==============================] - 37s 210ms/step - loss: 0.8438 - accuracy: 0.7291 - val_loss: 0.4539
- val_accuracy: 0.8470
Epoch 20/20
178/178 [==============================] - 32s 180ms/step - loss: 0.8193 - accuracy: 0.7306 - val_loss: 0.4522
- val accuracy: 0.8479
```

In [28]:
```
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')
```

```
88/88 [==============================] - 1s 12ms/step

C:\Users\Ishaan\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1370: UserWarning: Note that po
s_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:1370: UserWarning: Note that po
s_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 [29]:
```
############################## 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: 0.4494571387767792
Test Accuracy: 0.8436493873596191

Precision:  0.8062750963712157
Recall:  0.8030708732854444
```

In [30]:
```
# 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_d3_3.h5')
```

In [ ]:
```

```