Mhammad Kazem Rajabi

The Image dataset consists of **43** classes (Unique traffic sign images).

Training Set has **34799** Images , Test set has **12630** images and the validation set has **4410** images.

Divide deta set to train set and test set:

import pickle

training\_file = "train.pickle"

testing\_file = "test.pickle"

validation\_file = "valid.pickle"

with open(training\_file, mode='rb') as file:

    train = pickle.load(file)

with open(testing\_file, mode='rb') as file:

    test = pickle.load(file)

with open(validation\_file, mode='rb') as file:

    valid = pickle.load(file)

X\_train, y\_train = train['features'], train['labels']

X\_test, y\_test = test['features'], test['labels']

X\_valid , y\_valid = valid['features'], valid['labels']

And the size of each data set is :

34799

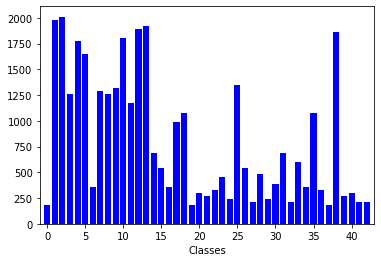
12630

4410

(32, 32, 3)

43

Number of data in each class:



Some part of data for example:



Data Augmentation(rotation and translation):

def augmentation(image):

    rows= image.shape[0]

    cols = image.shape[1]

    # rotation

    M\_rot = cv2.getRotationMatrix2D((cols/2,rows/2),10,1)

    # Translation

    M\_trans = np.float32([[1,0,3],[0,1,6]])

    img = cv2.warpAffine(image,M\_rot,(cols,rows))

    img = cv2.warpAffine(img,M\_trans,(cols,rows))

    img = cv2.bilateralFilter(img,9,75,75)

    return img

Incereasing data set in each class to 3000:

X\_after\_augmentation = []

Y\_after\_augmentation = []

for i in range(0,classes):

    class\_records = np.where(y\_train==i)[0].size

    max\_records = 3000

    if class\_records != max\_records:

        add\_number\_data = max\_records - class\_records

        samples = X\_train[np.where(y\_train==i)[0]]

        X\_aug = []

        Y\_aug = [i] \* add\_number\_data

        for x in range(add\_number\_data ):

            img = samples[x % class\_records]

            trans\_img = augmentation(img)

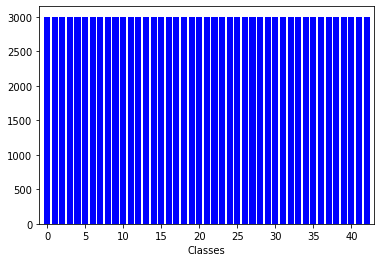
            X\_aug.append(trans\_img)

        X\_train\_final = np.concatenate((X\_train\_final, X\_aug), axis=0)

        y\_train\_final = np.concatenate((y\_train\_final, Y\_aug))

        Y\_after\_augmentation = Y\_after\_augmentation + Y\_aug

        X\_after\_augmentation = X\_after\_augmentation + X\_aug



After augmentation:

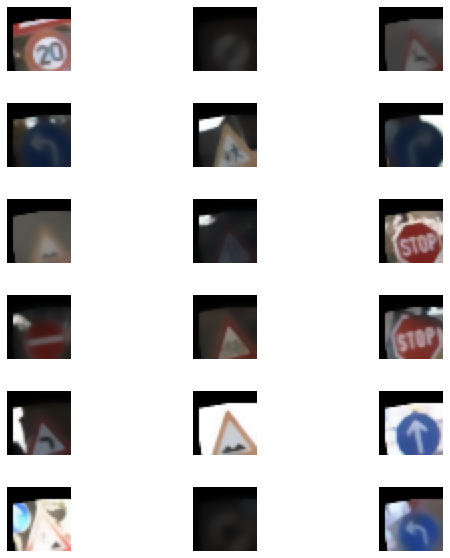
34799

129000

34799

129000

Some sample of new data-set:



Convertin new data-set to appropriate shape:

X\_train\_aug = X\_train\_final.reshape(len(X\_train\_final), 32,32,3).astype('float32')

X\_valid\_aug = X\_valid.reshape(len(X\_valid), 32,32,3).astype('float32')

y\_train\_aug = keras.utils.to\_categorical(y\_train\_final, classNumber)

y\_valid\_aug = keras.utils.to\_categorical(y\_valid, classNumber)

print(X\_train\_aug.shape)

print(y\_train\_aug.shape)

print(X\_valid\_aug.shape)

print(y\_valid\_aug.shape)

(129000, 32, 32, 3)

(129000, 43)

(4410, 32, 32, 3)

(4410, 43)

Build cnn:

model\_conv = Sequential()

model\_conv.add(Conv2D(32, kernel\_size=(3, 3),activation='relu', input\_shape=(32, 32, 3)))

model\_conv.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv.add(MaxPooling2D(pool\_size=(2, 2)))

model\_conv.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv.add(MaxPooling2D(pool\_size=(2, 2)))

model\_conv.add(Dropout(0.5))

model\_conv.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv.add(MaxPooling2D(pool\_size=(2, 2)))

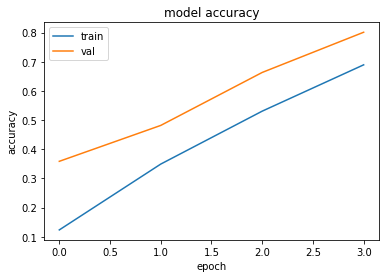
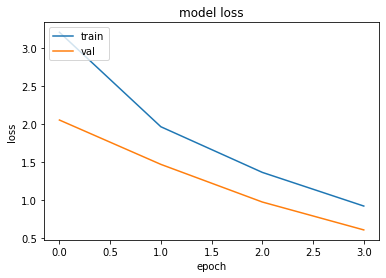
model\_conv.add(Dropout(0.5))

model\_conv.add(Flatten())

model\_conv.add(Dense(128, activation='relu'))

model\_conv.add(Dropout(0.5))

model\_conv.add(Dense(classNumber, activation='softmax'))

loss- 0.41

accuracy- 0.90

adding gray scale to data-set to achive better result:

def gray\_scale(image):

    return cv2.cvtColor(image, cv2.COLOR\_RGB2GRAY)

gray\_images\_data = list(map(gray\_scale, X\_train\_final))

print(local\_histo.shape)

%matplotlib inline

f\_show, a\_show = plt.subplots(6,3, figsize=(8, 8))

f\_show.subplots\_adjust(hspace = 1, wspace=.001)

a\_show = a\_show.ravel()

for i in range(18):

    index = np.random.randint(0, len(gray\_images\_data))

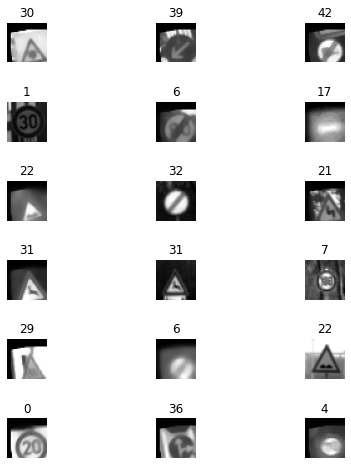
    image = gray\_images\_data[index]

    a\_show[i].axis('off')

    a\_show[i].imshow(image,cmap='gray')

    a\_show[i].set\_title(y\_train\_final[index])

sample of data after adding gray scale:



In the next step I add Histogram Equalization to data-set:

def histo\_equalize(image):

    kernel = morp.disk(30)

    img\_local = rank.equalize(image, selem=kernel)

    return img\_local

local\_histo = np.array(list(map(histo\_equalize, gray\_images\_data)))

print(local\_histo.shape)

%matplotlib inline

f\_show, a\_show = plt.subplots(6,3, figsize=(8, 8))

f\_show.subplots\_adjust(hspace =1, wspace=.001)

a\_show = a\_show.ravel()

for i in range(18):

    index = np.random.randint(0, len(local\_histo))

    image = local\_histo[index]

    a\_show[i].axis('off')

    a\_show[i].imshow(image,cmap = 'gray')

The shape of out data-set:

(129000, 1024)

(4410, 1024)

(129000, 43)

(4410, 43)

CNN model: model\_conv2 = Sequential()

model\_conv2.add(Conv2D(32, kernel\_size=(3, 3),activation='relu', input\_shape=(32, 32, 1)))

model\_conv2.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv2.add(MaxPooling2D(pool\_size=(2, 2)))

model\_conv2.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv2.add(MaxPooling2D(pool\_size=(2, 2)))

model\_conv2.add(Dropout(0.25))

model\_conv2.add(Conv2D(128, kernel\_size=(3, 3), activation='relu'))

model\_conv2.add(MaxPooling2D(pool\_size=(2, 2)))

model\_conv2.add(Dropout(0.5))

model\_conv2.add(Flatten())

model\_conv2.add(Dense(128, activation='relu'))

model\_conv2.add(Dropout(0.5))

model\_conv2.add(Dense(classNumber, activation='softmax'))

so with these changes I get better results:

loss- 0.21

accuracy- 0.94