In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

np.random.seed(2)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
import cv2
from PIL import Image
import numpy as np
import sys
import os
import csv
import numpy as np
import pandas as pd
import math
import scipy.ndimage

sns.set(style='white', context='notebook', palette='deep')
```

In [ ]:

```
# Load the data
train = pd.read_csv("../input/digit/digit/train.csv")  #loading training dataset of mnist
test = pd.read_csv("../input/digit/digit/test.csv")      #loading training dataset of mnis
t
train.head(1)
```

NEXT BLOCK TAKES INPUT IMAGES OF ''+,'-'operators converts it into MNIST format and makes a dataframe out of it and defines separate classes also...TO CONVERT IT INTO FORMAT OF TRAIN

In [ ]:

```
df = pd.DataFrame()
cla = 10
for i in range(1,3251):
    gray = scipy.ndimage.imread("../input/digit/digit/+/sum " +"("+ str(i)+")" + ".jpg",
flatten = False)
    #gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (28, 28),interpolation=cv2.INTER_AREA)
    (thresh, gray) = cv2.threshold(255-gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTS
U)
    #plt.imshow(gray)
    while np.sum(gray[0]) == 0:
        gray = gray[1:]
    while np.sum(gray[:,0]) == 0:
        gray = np.delete(gray,0,1)
    while np.sum(gray[-1]) == 0:
        gray = gray[:-1]
    while np.sum(gray[:,-1]) == 0:
        gray = np.delete(gray,-1,1)
    rows,cols = gray.shape
    if rows > cols:
```

```
        factor = 20.0/rows
        rows = 20
        cols = int(round(cols*factor))
        gray = cv2.resize(gray, (cols,rows))
    else:
        factor = 20.0/cols
        cols = 20
        rows = int(round(rows*factor))
        gray = cv2.resize(gray, (cols, rows))
    colsPadding = (int(math.ceil((28-cols)/2.0)),int(math.floor((28-cols)/2.0)))
    rowsPadding = (int(math.ceil((28-rows)/2.0)),int(math.floor((28-rows)/2.0)))
    gray = np.lib.pad(gray,(rowsPadding,colsPadding),'constant')
    def getBestShift(img):
        cy,cx = scipy.ndimage.measurements.center_of_mass(img)
        rows,cols = img.shape
        shiftx = np.round(cols/2.0-cx).astype(int)
        shifty = np.round(rows/2.0-cy).astype(int)
        return shiftx,shifty
    def shift(img,sx,sy):
        rows,cols = img.shape
        M = np.float32([[1,0,sx],[0,1,sy]])
        shifted = cv2.warpAffine(img,M,(cols,rows))
        return shifted
    shiftx,shifty = getBestShift(gray)
    shifted = shift(gray,shiftx,shifty)
    gray = shifted
    #gray = gray/255.0
    ########################
    imCrop = Image.fromarray(gray)
    width, height = imCrop.size
    value = np.asarray(imCrop.getdata(), dtype=np.float32).reshape((imCrop.size[1], imCr
op.size[0]))
    value = value.flatten()
    df2 = pd.DataFrame({'Label': [cla]})
    for i in range(1, 785):
        df2['pixel'+str(i-1)] = [value[i-1]]
    df = df.append(df2, ignore_index=True)
    print(1)
########################################33333
cla = 11
for i in range(1,3251):
    gray = scipy.ndimage.imread("../input/digit/digit/-/minus "+"("+ str(i) +")"+ ".jpg"
,flatten = False)
    #gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (28, 28),interpolation=cv2.INTER_AREA)
    (thresh, gray) = cv2.threshold(255-gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTS
U)
    #plt.imshow(gray)
    while np.sum(gray[0]) == 0:
        gray = gray[1:]
    while np.sum(gray[:,0]) == 0:
        gray = np.delete(gray,0,1)
    while np.sum(gray[-1]) == 0:
        gray = gray[:-1]
    while np.sum(gray[:,-1]) == 0:
        gray = np.delete(gray,-1,1)
    rows,cols = gray.shape
    if rows > cols:
        factor = 20.0/rows
        rows = 20
        cols = int(round(cols*factor))
        gray = cv2.resize(gray, (cols,rows))
    else:
        factor = 20.0/cols
        cols = 20
        rows = int(round(rows*factor))
        gray = cv2.resize(gray, (cols, rows))
    colsPadding = (int(math.ceil((28-cols)/2.0)),int(math.floor((28-cols)/2.0)))
    rowsPadding = (int(math.ceil((28-rows)/2.0)),int(math.floor((28-rows)/2.0)))
    gray = np.lib.pad(gray,(rowsPadding,colsPadding),'constant')
    def getBestShift(img):
        cy,cx = scipy.ndimage.measurements.center_of_mass(img)
```

```
        rows,cols = img.shape
        shiftx = np.round(cols/2.0-cx).astype(int)
        shifty = np.round(rows/2.0-cy).astype(int)
        return shiftx,shifty
    def shift(img,sx,sy):
        rows,cols = img.shape
        M = np.float32([[1,0,sx],[0,1,sy]])
        shifted = cv2.warpAffine(img,M,(cols,rows))
        return shifted
    shiftx,shifty = getBestShift(gray)
    shifted = shift(gray,shiftx,shifty)
    gray = shifted
    print(2)
    #gray = gray/255.0
    ########################
    imCrop = Image.fromarray(gray)
    width, height = imCrop.size
    value = np.asarray(imCrop.getdata(), dtype=np.float32).reshape((imCrop.size[1], imCr
op.size[0]))
    value = value.flatten()
    df2 = pd.DataFrame({'Label': [cla]})
    for i in range(1, 785):
        df2['pixel'+str(i-1)] = [value[i-1]]
    df = df.append(df2, ignore_index=True)
#print(df.head())
################################
cla = 12
for i in range(1,3251):
    gray = scipy.ndimage.imread("../input/digit/digit/times/multiply " +"("+ str(i)+")"+
".jpg",flatten = False)
    #gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (28, 28),interpolation=cv2.INTER_AREA)
    (thresh, gray) = cv2.threshold(255-gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTS
U)
    while np.sum(gray[0]) == 0:
        gray = gray[1:]
    while np.sum(gray[:,0]) == 0:
        gray = np.delete(gray,0,1)
    while np.sum(gray[-1]) == 0:
        gray = gray[:-1]
    while np.sum(gray[:,-1]) == 0:
        gray = np.delete(gray,-1,1)
    rows,cols = gray.shape
    if rows > cols:
        factor = 20.0/rows
        rows = 20
        cols = int(round(cols*factor))
        gray = cv2.resize(gray, (cols,rows))
    else:
        factor = 20.0/cols
        cols = 20
        rows = int(round(rows*factor))
        gray = cv2.resize(gray, (cols, rows))
    colsPadding = (int(math.ceil((28-cols)/2.0)),int(math.floor((28-cols)/2.0)))
    rowsPadding = (int(math.ceil((28-rows)/2.0)),int(math.floor((28-rows)/2.0)))
    gray = np.lib.pad(gray,(rowsPadding,colsPadding),'constant')
    def getBestShift(img):
        cy,cx = scipy.ndimage.measurements.center_of_mass(img)
        rows,cols = img.shape
        shiftx = np.round(cols/2.0-cx).astype(int)
        shifty = np.round(rows/2.0-cy).astype(int)
        return shiftx,shifty
    def shift(img,sx,sy):
        rows,cols = img.shape
        M = np.float32([[1,0,sx],[0,1,sy]])
        shifted = cv2.warpAffine(img,M,(cols,rows))
        return shifted
    shiftx,shifty = getBestShift(gray)
    shifted = shift(gray,shiftx,shifty)
    gray = shifted
    #gray = gray/255.0
    print(3)
```

```
    ########################
    imCrop = Image.fromarray(gray)
    width, height = imCrop.size
    value = np.asarray(imCrop.getdata(), dtype=np.float32).reshape((imCrop.size[1], imCr
op.size[0]))
    value = value.flatten()
    df2 = pd.DataFrame({'Label': [cla]})
    for i in range(1, 785):
        df2['pixel'+str(i-1)] = [value[i-1]]
    df = df.append(df2, ignore_index=True)
```

JUMBLES THE DATAFRAME

In [ ]:

```
df1 = df
df = df.sample(frac=1).reset_index(drop=True)
df.head()
```

renames the label column

In [ ]:

```
df.rename(columns={'Label':'label'}, inplace=True)
```

In [ ]:

```
train = train.append(df,ignore_index=True) # add the dataframe made to train
train.to_csv('../inputtraining.csv', encoding='utf-8', index=False)
#train = train.append(test,ignore_index=True)
Y_train = train["label"]

# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

# free some space
del train

g = sns.countplot(Y_train)

Y_train.value_counts()
```

In [ ]:

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
```

In [ ]:

```
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

In [ ]:

```
# Encode labels to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0,0,0,0,0])
def to_categorical(y, nb_classes=None):
    '''Convert class vector (integers from 0 to nb_classes)
    to binary class matrix, for use with categorical_crossentropy
    '''
    y = np.asarray(y, dtype='int32')
    if not nb_classes:
        nb_classes = np.max(y)+1
    Y = np.zeros((len(y), nb_classes))
    for i in range(len(y)):
        Y[i, y[i]] = 1.
    return Y
d_train = to_categorical(Y_train,nb_classes=13)
```

In [ ]:

```
d_train.shape
Y_train = d_train
```

In [ ]:

```
# Set the random seed
random_seed = 2
```

In [ ]:

```
# Split the train and the validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, ran
dom_state=random_seed)
```

In [ ]:

```
# Some examples
g = plt.imshow(X_train[0][:,:,0])
```

In [ ]:

```
from keras.models import Sequential # to create a cnn model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,SeparableConv2D,Leak
yReLU,UpSampling2D,DepthwiseConv2D,BatchNormalization,LocallyConnected2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
model = Sequential()
model.add(Conv2D(filters = 128, kernel_size = (5,5),padding = 'Same', input_shape = (28,
28,1)))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))

model.add(Conv2D(filters = 128, kernel_size = (5,5),padding = 'Same'))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same'))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.1))

model.add(Conv2D(filters = 256, kernel_size = (3,3),padding = 'Same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
model.add(LeakyReLU(alpha=0.1))

model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = 'Same'))
model.add(LeakyReLU(alpha=0.1))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(13, activation = "softmax"))
```

In [ ]:

```
# Define the optimizer
optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

In [ ]:

```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accur
```

```
acy"])
```

In [ ]:

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

In [ ]:

```
epochs = 30 # Turn epochs to 30 to get 0.9967 accuracy
batch_size = 86
```

In [ ]:

```
# With data augmentation to prevent overfitting (accuracy 0.99286)

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=5,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total he
ight)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

In [ ]:

```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_siz
e
                              , callbacks=[learning_rate_reduction])
```

In [ ]:

```
model.save('layer_kaggle.h5')
```