Identify the Digits

Automatic digit recognition is of popular interest today. Deep Learning techniques makes it possible for object recognition in image data . This practice problem is meant to give you a kick start in deep learning. As usual, we will not only provide you with the challenge and a solution checker, but also a set of tutorials to get you off the ground!

The data set used for this problem is from the populat MNIST data set. Developed by Yann LeCun, Corina Cortes and Christopher Burger for evaluating machine learning model on the handwritten digit classification problem.

```
get --header="Host: datahack-prod.s3.amazonaws.com" --header="User-Agent: Mozilla/5.0 (Win
```

```
--2020-10-29 18:08:34--  https://datahack-prod.s3.amazonaws.com/train_file/Train_UQcU
Resolving datahack-prod.s3.amazonaws.com (datahack-prod.s3.amazonaws.com)... 52.219.6
Connecting to datahack-prod.s3.amazonaws.com (datahack-prod.s3.amazonaws.com)|52.219
HTTP request sent, awaiting response... 200 OK
Length: 52075589 (50M) [application/zip]
Saving to: ‘Train_UQcUa52.zip’

Train_UQcUa52.zip   100%[===================>]  49.66M  10.4MB/s    in 5.4s

2020-10-29 18:08:40 (9.12 MB/s) - ‘Train_UQcUa52.zip’ saved [52075589/52075589]
```

```
!ls
```

```
sample_data  Train_UQcUa52.zip
```

```
!unzip Train_UQcUa52.zip
```

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import keras
from keras import Model
from keras.layers import Conv2D,Dense,MaxPooling2D,AveragePooling2D,BatchNormalization,Inp
from keras.optimizers import Adam
from keras.models import Sequential
from skimage.io import imread
from skimage.transform import resize
from tqdm import tqdm
import matplotlib.pyplot as plt
%matplotlib inline

# for creating validation set
from sklearn.model_selection import train_test_split
```

```
# for evaluating the model
from sklearn.metrics import accuracy_score
```

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334 # this function is used to update the plots for eac
def plt_dynamic(x, vy, ty, ax, colors=['b']):
  ax.plot(x, vy, 'b', label="Validation Loss")
  ax.plot(x, ty, 'r', label="Train Loss")
  plt.legend()
  plt.grid()
  fig.canvas.draw()
```

```
train=pd.read_csv('train.csv')
test=pd.read_csv('Test_fCbTej3_0j1gHmj.csv')
```

```
train.tail()
```

	filename	label
48995	48995.png	2
48996	48996.png	4
48997	48997.png	9
48998	48998.png	3
48999	48999.png	0

```
y_train=train.label
```

```
y_train
```

```
0        4
1        9
2        1
3        7
4        3
        ..
48995    2
48996    4
48997    9
48998    3
48999    0
Name: label, Length: 49000, dtype: int64
```

```
y_train=keras.utils.to_categorical(y_train,num_classes)
```

```
num_classes=train.label.nunique()
print(num_classes)
```

```
10
```

```
test.tail()
```

	filename
20995	69995.png
20996	69996.png
20997	69997.png
20998	69998.png
20999	69999.png

```
train_path=os.path.join(os.getcwd(),'Images/train/')
print(train_path)
test_path=os.path.join(os.getcwd(),'Images/test/')
print(test_path)
```

```
/content/Images/train/
/content/Images/test/
```

```
train_img=[]
for img in tqdm(train['filename']):
  image=imread(train_path+img)
  image=image/255.
  image=resize(image,(28,28,1),mode='constant')
  image=image.astype('float')
  train_img.append(image)
```

```
100%|██████████| 49000/49000 [00:40<00:00, 1221.15it/s]
```

```
train_img=np.array(train_img)
train_img.shape
```

```
(49000, 28, 28, 1)
```

```
test_img=[]
for img in tqdm(test['filename']):
  image=imread(test_path+img)
  image=image/255.
  image=resize(image,(28,28,1),mode='constant')
  image=image.astype('float')
  test_img.append(image)
test_img=np.array(test_img)
test_img.shape
```

```
100%|██████████| 21000/21000 [00:16<00:00, 1250.30it/s]
```

```
(21000, 28, 28, 1)
```

```
np.save('train_img.npy',train_img)
np.save('test_img.npy',test_img)


X_train=np.load('./train_img.npy',allow_pickle=True)
X_test=np.load('./test_img.npy',allow_pickle=True)


# Network Architecture
# input -> conv -> conv -> pooling -> conv -> conv -> pooling ->dropout-> FC -> output
# 16 16 32 32 512

inp_shape=X_test.shape[1:]
model=Sequential()
model.add(Conv2D(16,kernel_size=(3,3),padding='same',activation='relu',input_shape=inp_sha
model.add(Conv2D(16,5,padding='same',activation='relu'))
model.add(MaxPooling2D(strides=2))
model.add(Conv2D(32,5,activation='relu',padding='same'))
model.add(Conv2D(32,5,activation='relu',padding='same'))
model.add(MaxPooling2D(strides=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_31 (Conv2D)           (None, 28, 28, 16)        160
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 28, 28, 16)        6416
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 14, 14, 32)        12832
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 14, 14, 32)        25632
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               803328
_________________________________________________________________
dense_11 (Dense)             (None, 10)                5130
=================================================================
Total params: 853,498
Trainable params: 853,498
Non-trainable params: 0
_________________________________________________________________
```

```
history=model.fit(X_train,y_train,epochs=15,batch_size=128,verbose=1,validation_split=0.2)
```

```
Epoch 1/15
307/307 [==============================] - 3s 9ms/step - loss: 0.2614 - accuracy: 0.9
Epoch 2/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0672 - accuracy: 0.9
Epoch 3/15
307/307 [==============================] - 2s 7ms/step - loss: 0.0420 - accuracy: 0.9
Epoch 4/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0333 - accuracy: 0.9
Epoch 5/15
307/307 [==============================] - 2s 7ms/step - loss: 0.0290 - accuracy: 0.9
Epoch 6/15
307/307 [==============================] - 2s 7ms/step - loss: 0.0222 - accuracy: 0.9
Epoch 7/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0195 - accuracy: 0.9
Epoch 8/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0182 - accuracy: 0.9
Epoch 9/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0178 - accuracy: 0.9
Epoch 10/15
307/307 [==============================] - 2s 7ms/step - loss: 0.0145 - accuracy: 0.9
Epoch 11/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0131 - accuracy: 0.9
Epoch 12/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0118 - accuracy: 0.9
Epoch 13/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0105 - accuracy: 0.9
Epoch 14/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0092 - accuracy: 0.9
Epoch 15/15
307/307 [==============================] - 2s 8ms/step - loss: 0.0096 - accuracy: 0.9
```

```
history2=model.fit(X_train,y_train,epochs=30,batch_size=256,verbose=1,validation_split=0.2
```

```
Epoch 1/30
144/144 [==============================] - 2s 15ms/step - loss: 0.3496 - accuracy:
Epoch 2/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0768 - accuracy:
Epoch 3/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0545 - accuracy:
Epoch 4/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0399 - accuracy:
Epoch 5/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0339 - accuracy:
Epoch 6/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0270 - accuracy:
Epoch 7/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0222 - accuracy:
Epoch 8/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0191 - accuracy:
Epoch 9/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0160 - accuracy:
Epoch 10/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0171 - accuracy:
Epoch 11/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0142 - accuracy:
Epoch 12/30
```

```
144/144 [==============================] - 2s 13ms/step - loss: 0.0116 - accuracy:
Epoch 13/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0106 - accuracy:
Epoch 14/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0103 - accuracy:
Epoch 15/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0107 - accuracy:
Epoch 16/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0089 - accuracy:
Epoch 17/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0099 - accuracy:
Epoch 18/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0079 - accuracy:
Epoch 19/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0084 - accuracy:
Epoch 20/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0064 - accuracy:
Epoch 21/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0077 - accuracy:
Epoch 22/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0071 - accuracy:
Epoch 23/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0075 - accuracy:
Epoch 24/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0045 - accuracy:
Epoch 25/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0051 - accuracy:
Epoch 26/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0055 - accuracy:
Epoch 27/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0052 - accuracy:
Epoch 28/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0064 - accuracy:
Epoch 29/30
144/144 [==============================] - 2s 13ms/step - loss: 0.0056 - accuracy:
```

```
X_test=np.load('./test_img.npy',allow_pickle=True)# Network Architecture
# input -> conv -> polling -> conv -> polling -> conv -> polling ->dropout-> FC -> output
# 8 32 128 64
model1 = Sequential()
model1.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=inp_shape))
model1.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariance
model1.add(Conv2D(64, (3,3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model1.add(Dropout(0.9))
model1.add(Flatten())
model1.add(Dense(64, activation='relu'))
model1.add(Dense(num_classes, activation='softmax'))
model1.compile(loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])

model1.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 3, 3, 128)         73856
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 101,578
Trainable params: 101,578
Non-trainable params: 0
_________________________________________________________________
```

```
history2=model1.fit(X_train,y_train,epochs=30,verbose=1,validation_split=0.20,batch_size=1
```

```
307/307 [==============================] - 2s 6ms/step - loss: 1.6325 - accuracy:
Epoch 2/30
307/307 [==============================] - 2s 5ms/step - loss: 1.1111 - accuracy:
Epoch 3/30
307/307 [==============================] - 2s 5ms/step - loss: 0.9568 - accuracy:
Epoch 4/30
307/307 [==============================] - 2s 5ms/step - loss: 0.8626 - accuracy:
Epoch 5/30
307/307 [==============================] - 2s 5ms/step - loss: 0.8060 - accuracy:
Epoch 6/30
307/307 [==============================] - 2s 5ms/step - loss: 0.7541 - accuracy:
Epoch 7/30
307/307 [==============================] - 2s 5ms/step - loss: 0.7045 - accuracy:
Epoch 8/30
307/307 [==============================] - 2s 5ms/step - loss: 0.6743 - accuracy:
Epoch 9/30
307/307 [==============================] - 2s 5ms/step - loss: 0.6422 - accuracy:
Epoch 10/30
307/307 [==============================] - 2s 5ms/step - loss: 0.6238 - accuracy:
Epoch 11/30
307/307 [==============================] - 2s 5ms/step - loss: 0.6158 - accuracy:
Epoch 12/30
307/307 [==============================] - 2s 5ms/step - loss: 0.5911 - accuracy:
Epoch 13/30
307/307 [==============================] - 2s 5ms/step - loss: 0.5629 - accuracy:
Epoch 14/30
307/307 [==============================] - 2s 5ms/step - loss: 0.5554 - accuracy:
Epoch 15/30
307/307 [==============================] - 2s 5ms/step - loss: 0.5320 - accuracy:
Epoch 16/30

307/307 [==============================] - 2s 5ms/step - loss: 0.5126 - accuracy:
```

```
Epoch 17/30
307/307 [==============================] - 2s 5ms/step - loss: 0.5026 - accuracy:
Epoch 18/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4908 - accuracy:
Epoch 19/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4806 - accuracy:
Epoch 20/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4684 - accuracy:
Epoch 21/30
307/307 [==============================] - 2s 6ms/step - loss: 0.4563 - accuracy:
Epoch 22/30
307/307 [==============================] - 2s 6ms/step - loss: 0.4448 - accuracy:
Epoch 23/30
307/307 [==============================] - 2s 6ms/step - loss: 0.4425 - accuracy:
Epoch 24/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4295 - accuracy:
Epoch 25/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4200 - accuracy:
Epoch 26/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4092 - accuracy:
Epoch 27/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4107 - accuracy:
Epoch 28/30
307/307 [==============================] - 2s 5ms/step - loss: 0.4029 - accuracy:
Epoch 29/30
307/307 [==============================] - 2s 5ms/step - loss: 0.3967 - accuracy:
Epoch 30/30
```

This model has not performed well. we will use first model

```
pred =np.array(model.predict(X_test))
```

```
pred
```

```
array([[6.4498579e-25, 4.0007149e-18, 8.9591040e-20, ..., 1.5699406e-16,
        1.9459938e-20, 1.4998182e-19],
       [1.0000000e+00, 7.2433094e-14, 3.1988835e-11, ..., 4.7379042e-16,
        1.0028973e-12, 1.0242691e-11],
       [1.7701690e-05, 2.5618079e-11, 5.4091409e-08, ..., 7.0005754e-08,
        2.2611146e-06, 9.9997640e-01],
       ...,
       [2.2670352e-13, 2.8482740e-18, 1.3518777e-15, ..., 9.9225124e-25,
        9.8218500e-10, 1.6196993e-19],
       [3.9621896e-11, 1.9836280e-14, 7.8765992e-15, ..., 5.2805887e-19,
        9.8923714e-10, 3.5995552e-17],
       [8.6779484e-16, 3.7193874e-18, 1.0000000e+00, ..., 4.3823165e-22,
        2.4916688e-22, 2.5258925e-21]], dtype=float32)
```

```
predictions=[]
for i in pred:
  predictions.append(np.argmax(i))
```

```
predictions
```

```
[4,
 0,
 9,
 7,
 9,
 6,
 6,
 7,
 0,
 4,
 2,
 8,
 4,
 6,
 1,
 2,
 9,
 6,
 1,
 4,
 0,
 8,
 4,
 3,
 7,
 7,
 5,
 1,
 6,
 4,
 1,
 1,
 2,
 7,
 1,
 8,
 0,
 3,
 2,
 4,
 3,
 1,
 8,
 7,
 7,
 7,
 3,
 5,
 0,
 0,
 2,
 5,
 6,
 5,
 1,
 7,
 2,
 6,
 7,
```

```
sub=test['filename']
```

```
sub.head()
```

```
0    49000.png
1    49001.png
2    49002.png
3    49003.png
4    49004.png
Name: filename, dtype: object
```

```
sub['label']=predictions
```

```
predict = pd.DataFrame(data=predictions ,columns=["label"])
```

```
sub = test['filename']
DT = pd.merge(sub , predict, on=None, left_index= True,
    right_index=True)
```

```
DT.to_csv('brahm_submssion_mnist.csv',index=False)
```