Digit Recognizer Summary
Nicholas Bergeland
MSDS 422, Dr. Fulton

Summary:

This week for our assignment I built an algorithm trained to recognize images of numerical digits 1-10. The model I built tested at 99.175% recognition, which was good enough for 378 / 1806 (top 21%).

The model was built and trained with a combination of matplotlib, numpy, seaborne, tenorflow, SKlearn, and Keras. I first read in the testing and training datasets for the various models to work with. From this point, I began understanding the dataset with some basic EDA.

To perform the basic analysis, I used a graph within matplotlib to see the occurrence of different numbers. Is the dataset normally distributed? After observing the normality (shape) of the dataset, I got to work. The work involved some simple encoding, then creation of different layers of the CNN model.

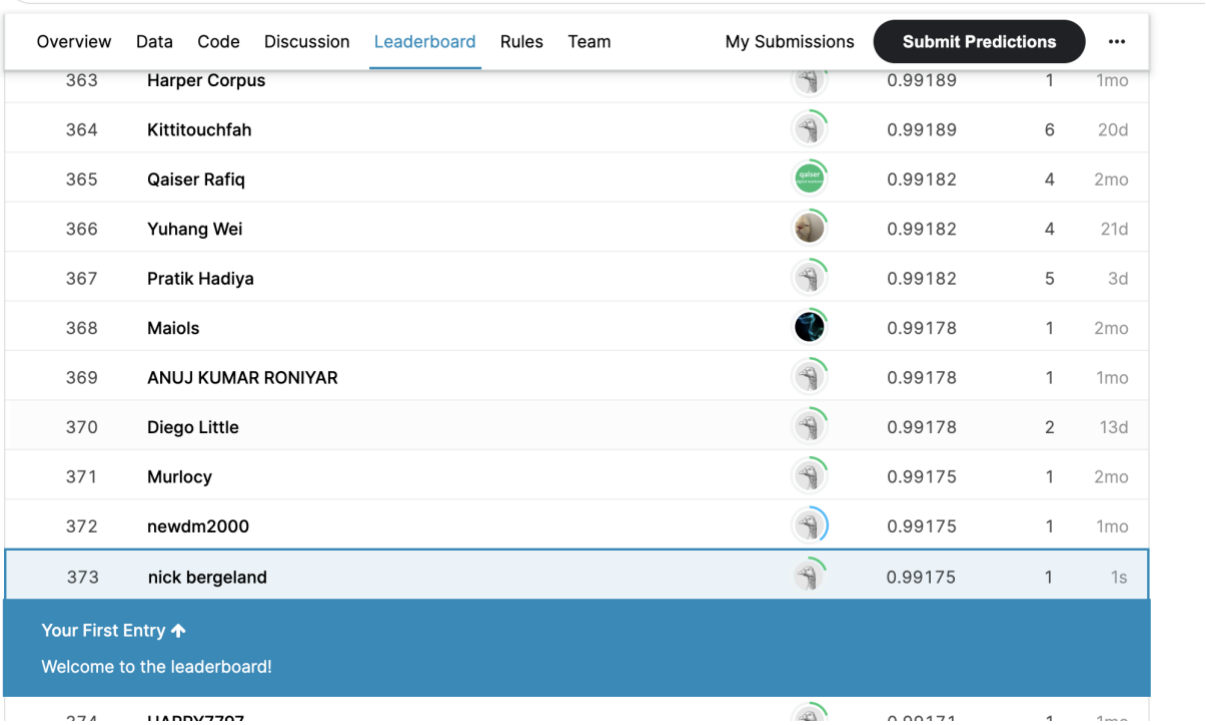
The thing I was not prepared for was how time intensive the model was. Each different visualization model took 45-60 minutes to run through all of the sequences of testing. I found it fascinating to watch the model simulate through “epochs” of data and continually improve itself with no guidance provided by me.

The result was just as impressive. With no structure provided by me, the model was recognizing nearly every number it was presented with. One could argue the 99 + % success rate may be below a human, which would likely get all of the numbers. However, with no prior

learning and in such a short period, it is truly shocking the success the model was able to develop given the timeframe.

Despite placing in the top 25% of this challenge, I will look to improve my model moving forward to next week. Having seen scores testing at 100% recognition, I will set my sights on this mark as I develop my models further.

Kaggle Score:



Appendix:

Out[2]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import tensorflow as tf
```

In [2]:
```
train = pd.read_csv("train2.csv")
test = pd.read_csv("test2.csv")
train.head()
```

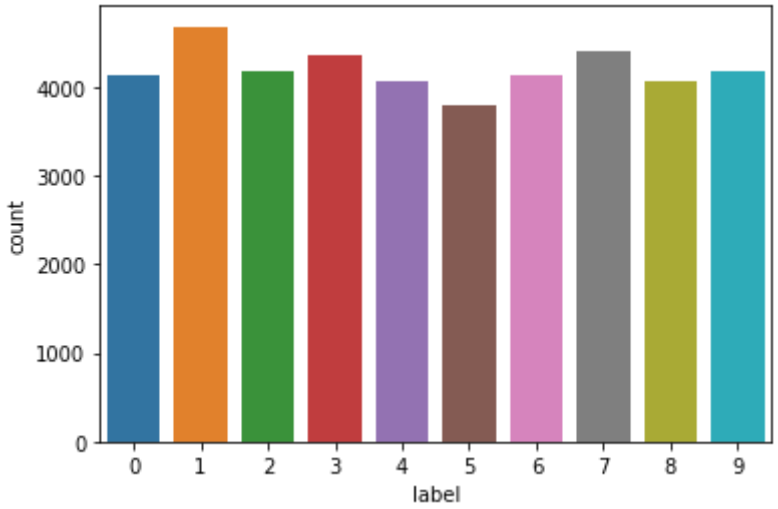
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775
0	1	0	0	0	0	0	0	0	0	0	...		
1	0	0	0	0	0	0	0	0	0	0	...		
2	1	0	0	0	0	0	0	0	0	0	...		
3	4	0	0	0	0	0	0	0	0	0	...		
4	0	0	0	0	0	0	0	0	0	0	...		

5 rows × 785 columns

In [3]:
```
y_train = train['label'].astype('float32')
X_train = train.drop(['label'], axis=1).astype('int32')
X_test = test.astype('float32')
X_train.shape, y_train.shape, X_test.shape
```

Out[3]: ((42000, 784), (42000,), (28000, 784))

In [4]:
```
sns . countplot ( x = 'label' , data = train ) ;
```



In [5]:
```
# Data normalization
X_train = X_train / 255
X_test = X_test / 255
```

In [6]:
```
X_train = X_train.values.reshape(-1,28,28,1)
X_test = X_test.values.reshape(-1,28,28,1)
X_train.shape, X_test.shape
```

Out[6]: ((42000, 28, 28, 1), (28000, 28, 28, 1))

In [7]:
```
# one-hot encoding from keras.utils.np_utils
import to_categorical y_train = to_categorical(y_train,
num_classes = 10) y_train.shape
```

Using TensorFlow backend.

Out[7]: (42000, 10)

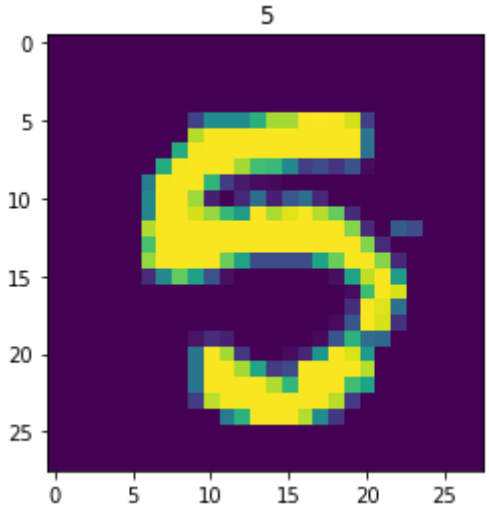
In [8]:
```
print ( train [ 'label' ] . head ())
y_train [ 0: 5,:]
```

```
0    1
1    0
2    1
3    4
4    0
Name: label, dtype: int64
```

Out[8]:
```
array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train , X_cv , y_train , y_cv = train_test_split ( X_train , y_train , test_s
ize = 0.1 , random_state =42 )
```

In [10]:
```
plt . imshow ( X_train [ 1][:,: , 0])
plt . title ( y_train [ 1] . argmax ()) ;
```



In [11]:
```
from keras.layers import Input,InputLayer, Dense, Activation, ZeroPaddin g2D,
BatchNormalization, Flatten, Conv2D
from keras.layers import AveragePooling2D, MaxPooling2D, Dropout
from keras.models import Sequential,Model from keras.optimizers
import SGD
from keras.callbacks import ModelCheckpoint,LearningRateScheduler import
keras
from keras import backend as K
```

In [12]:
```
# Building a CNN model
input_shape = (28,28,1) X_input =
Input(input_shape)

# layer 1
x = Conv2D(64,(3,3),strides=(1,1),name='layer_conv1',padding='same')(X_i nput)
x = BatchNormalization()(x) x =
Activation('relu')(x)
x = MaxPooling2D((2,2),name='maxPool1')(x)
# layer 2
x = Conv2D(32,(3,3),strides=(1,1),name='layer_conv2',padding='same')(x) x =
BatchNormalization()(x) x = Activation('relu')(x)
x = MaxPooling2D((2,2),name='maxPool2')(x)
# layer 3
x = Conv2D(32,(3,3),strides=(1,1),name='conv3',padding='same')(x) x =
BatchNormalization()(x) x = Activation('relu')(x)
x = MaxPooling2D((2,2), name='maxPool3')(x)
# fc x =
Flatten()(x)
x = Dense(64,activation ='relu',name='fc0')(x) x =
Dropout(0.25)(x)
x = Dense(32,activation ='relu',name='fc1')(x) x =
Dropout(0.25)(x)
x = Dense(10,activation ='softmax',name='fc2')(x)

conv_model = Model(inputs=X_input, outputs=x, name='Predict') conv_model.summary()
```

```
Model: "Predict"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
===== input_1 (InputLayer)        (None, 28, 28, 1)         0
_________________________________________________________________
_____ layer_conv1 (Conv2D)        (None, 28, 28, 64)        640
_________________________________________________________________
_____ batch_normalization_1 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
_____ activation_1 (Activation)    (None, 28, 28, 64)        0
_________________________________________________________________
_____ maxPool1 (MaxPooling2D)      (None, 14, 14, 64)        0
_________________________________________________________________
_____ layer_conv2 (Conv2D)        (None, 14, 14, 32)        18464
_________________________________________________________________
_____ batch_normalization_2 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
_____ activation_2 (Activation)    (None, 14, 14, 32)        0
```

```
_____ maxPool2 (MaxPooling2D)      (None, 7, 7, 32)         0
_____ conv3 (Conv2D)               (None, 7, 7, 32)         9248
_____ batch_normalization_3 (Batch (None, 7, 7, 32)         128
_____ activation_3 (Activation)    (None, 7, 7, 32)         0
_____ maxPool3 (MaxPooling2D)      (None, 3, 3, 32)         0
_____ flatten_1 (Flatten)          (None, 288)              0
_____ fc0 (Dense)                  (None, 64)               18496
_____ dropout_1 (Dropout)          (None, 64)               0
_____ fc1 (Dense)                  (None, 32)               2080
_____ dropout_2 (Dropout)          (None, 32)               0
_____ fc2 (Dense)                  (None, 10)               330
=================================================================
=====
Total params: 49,770
Trainable params: 49,514
Non-trainable params: 256
```

In [13]:
```
# Adam optimizer
conv_model.compile(optimizer='adam',loss='categorical_crossentropy',metr ics=['accuracy'])
conv_model.fit(X_train, y_train, epochs=10, batch_size=100, validation_d ata=(X_cv,y_cv))
```

```
Train on 37800 samples, validate on 4200 samples
Epoch 1/10
37800/37800 [==============================] - 144s 4ms/step - loss: 0.
4887 - accuracy: 0.8468 - val_loss: 0.6940 - val_accuracy: 0.8188
Epoch 2/10
37800/37800 [==============================] - 138s 4ms/step - loss: 0.
1349 - accuracy: 0.9610 - val_loss: 0.0763 - val_accuracy: 0.9776
Epoch 3/10
37800/37800 [==============================] - 138s 4ms/step - loss: 0.
0911 - accuracy: 0.9744 - val_loss: 0.0541 - val_accuracy: 0.9857
Epoch 4/10
37800/37800 [==============================] - 138s 4ms/step - loss: 0.
```

```
0760 - accuracy: 0.9791 - val_loss: 0.0640 - val_accuracy: 0.9840
Epoch 5/10
37800/37800 [==============================] - 138s 4ms/step - loss: 0.
0629 - accuracy: 0.9824 - val_loss: 0.0658 - val_accuracy: 0.9795
Epoch 6/10
37800/37800 [==============================] - 142s 4ms/step - loss: 0.
0566 - accuracy: 0.9845 - val_loss: 0.0549 - val_accuracy: 0.9838
Epoch 7/10
37800/37800 [==============================] - 139s 4ms/step - loss: 0.
0471 - accuracy: 0.9872 - val_loss: 0.0378 - val_accuracy: 0.9902
Epoch 8/10
37800/37800 [==============================] - 139s 4ms/step - loss: 0.
0460 - accuracy: 0.9872 - val_loss: 0.0558 - val_accuracy: 0.9864
Epoch 9/10
37800/37800 [==============================] - 139s 4ms/step - loss: 0.
0363 - accuracy: 0.9893 - val_loss: 0.0427 - val_accuracy: 0.9888
Epoch 10/10
37800/37800 [==============================] - 140s 4ms/step - loss: 0.
0366 - accuracy: 0.9899 - val_loss: 0.0680 - val_accuracy: 0.9850
```

Out[13]: <keras.callbacks.callbacks.History at 0x7f8bee1d2850>

In [14]:
```
# SGD optimizer sgd = SGD(lr=0.0005, momentum=0.5, decay=0.0, nesterov=False)
conv_model.compile(optimizer=sgd,loss='categorical_crossentropy',metrics
=['accuracy'])
                                conv_model.fit(X_train, y_train, epochs=30, validation_data=(X_cv, y_cv
))
```

Train on 37800 samples, validate on 4200 samples
Epoch 1/30
37800/37800 [==============================] - 173s 5ms/step - loss: 0.
0288 - accuracy: 0.9922 - val_loss: 0.0345 - val_accuracy: 0.9929
Epoch 2/30
37800/37800 [==============================] - 170s 5ms/step - loss: 0.
0237 - accuracy: 0.9937 - val_loss: 0.0337 - val_accuracy: 0.9929
Epoch 3/30
37800/37800 [==============================] - 172s 5ms/step - loss: 0.
0208 - accuracy: 0.9943 - val_loss: 0.0327 - val_accuracy: 0.9929
Epoch 4/30
37800/37800 [==============================] - 1599s 42ms/step - loss:
0.0188 - accuracy: 0.9949 - val_loss: 0.0327 - val_accuracy: 0.9938
Epoch 5/30
37800/37800 [==============================] - 1359s 36ms/step - loss:
0.0192 - accuracy: 0.9947 - val_loss: 0.0330 - val_accuracy: 0.9926
Epoch 6/30
37800/37800 [==============================] - 322s 9ms/step - loss: 0.
0178 - accuracy: 0.9951 - val_loss: 0.0327 - val_accuracy: 0.9929
Epoch 7/30
37800/37800 [==============================] - 1389s 37ms/step - loss:
0.0183 - accuracy: 0.9947 - val_loss: 0.0323 - val_accuracy: 0.9933
Epoch 8/30
37800/37800 [==============================] - 2006s 53ms/step - loss:
0.0186 - accuracy: 0.9948 - val_loss: 0.0320 - val_accuracy: 0.9933
Epoch 9/30
37800/37800 [==============================] - 1214s 32ms/step - loss:
0.0162 - accuracy: 0.9960 - val_loss: 0.0321 - val_accuracy: 0.9938
Epoch 10/30
37800/37800 [==============================] - 177s 5ms/step - loss: 0.
0172 - accuracy: 0.9952 - val_loss: 0.0324 - val_accuracy: 0.9938
Epoch 11/30
37800/37800 [==============================] - 174s 5ms/step - loss: 0.
0166 - accuracy: 0.9957 - val_loss: 0.0329 - val_accuracy: 0.9940
Epoch 12/30
37800/37800 [==============================] - 172s 5ms/step - loss: 0.
0170 - accuracy: 0.9952 - val_loss: 0.0325 - val_accuracy: 0.9940
Epoch 13/30
37800/37800 [==============================] - 568s 15ms/step - loss:
0.0169 - accuracy: 0.9953 - val_loss: 0.0319 - val_accuracy: 0.9943
Epoch 14/30
37800/37800 [==============================] - 180s 5ms/step - loss: 0.
0158 - accuracy: 0.9958 - val_loss: 0.0321 - val_accuracy: 0.9938
Epoch 15/30

37800/37800 [==============================] - 179s 5ms/step - loss: 0.
0156 - accuracy: 0.9959 - val_loss: 0.0321 - val_accuracy: 0.9938
Epoch 16/30
37800/37800 [==============================] - 179s 5ms/step - loss: 0.
0148 - accuracy: 0.9963 - val_loss: 0.0324 - val_accuracy: 0.9936
Epoch 17/30
37800/37800 [==============================] - 181s 5ms/step - loss: 0.
0163 - accuracy: 0.9953 - val_loss: 0.0328 - val_accuracy: 0.9938
Epoch 18/30
37800/37800 [==============================] - 180s 5ms/step - loss: 0.
0153 - accuracy: 0.9958 - val_loss: 0.0328 - val_accuracy: 0.9936
Epoch 19/30
37800/37800 [==============================] - 184s 5ms/step - loss: 0. 0159
- accuracy: 0.9958 - val_loss: 0.0331 - val_accuracy: 0.9938
Epoch 20/30
37800/37800 [==============================] - 181s 5ms/step - loss: 0.
0137 - accuracy: 0.9963 - val_loss: 0.0329 - val_accuracy: 0.9938
Epoch 21/30
37800/37800 [==============================] - 180s 5ms/step - loss: 0.
0153 - accuracy: 0.9959 - val_loss: 0.0325 - val_accuracy: 0.9936
Epoch 22/30
37800/37800 [==============================] - 180s 5ms/step - loss: 0.
0154 - accuracy: 0.9958 - val_loss: 0.0325 - val_accuracy: 0.9933
Epoch 23/30
37800/37800 [==============================] - 182s 5ms/step - loss: 0.
0151 - accuracy: 0.9957 - val_loss: 0.0324 - val_accuracy: 0.9936
Epoch 24/30
37800/37800 [==============================] - 182s 5ms/step - loss: 0.
0152 - accuracy: 0.9958 - val_loss: 0.0328 - val_accuracy: 0.9938
Epoch 25/30
37800/37800 [==============================] - 183s 5ms/step - loss: 0.
0134 - accuracy: 0.9966 - val_loss: 0.0323 - val_accuracy: 0.9940
Epoch 26/30
37800/37800 [==============================] - 182s 5ms/step - loss: 0.
0145 - accuracy: 0.9959 - val_loss: 0.0323 - val_accuracy: 0.9943
Epoch 27/30
37800/37800 [==============================] - 187s 5ms/step - loss: 0.
0130 - accuracy: 0.9965 - val_loss: 0.0326 - val_accuracy: 0.9938
Epoch 28/30
37800/37800 [==============================] - 182s 5ms/step - loss: 0.
0137 - accuracy: 0.9960 - val_loss: 0.0324 - val_accuracy: 0.9940
Epoch 29/30
37800/37800 [==============================] - 183s 5ms/step - loss: 0.
0139 - accuracy: 0.9961 - val_loss: 0.0329 - val_accuracy: 0.9936

```
Epoch 30/30
37800/37800 [==============================] - 184s 5ms/step - loss: 0.
0151 - accuracy: 0.9958 - val_loss: 0.0326 - val_accuracy: 0.9933
```

Out[14]: <keras.callbacks.callbacks.History at 0x7f8bf7b7c350>

In [16]:
```
y_pred = conv_model.predict(X_test) y_pred =
np.argmax(y_pred,axis=1)
my_submission = pd.DataFrame({'ImageId': list(range(1, len(y_pred)+1)),
'Label': y_pred}) my_submission.to_csv('dig_submission.csv', index=False)
```

In [ ]:

Works Cited:

sriram2397. "Digit-Recognizer-Kaggle/digit_recognizer.Ipynb at Master · SRIRAM2397/Digit-Recognizer-Kaggle." GitHub. Accessed February 6, 2022. https://github.com/sriram2397/digit-recognizer-kaggle/blob/master/Digit_
Recognizer.ipynb.