result-mnist-cnn

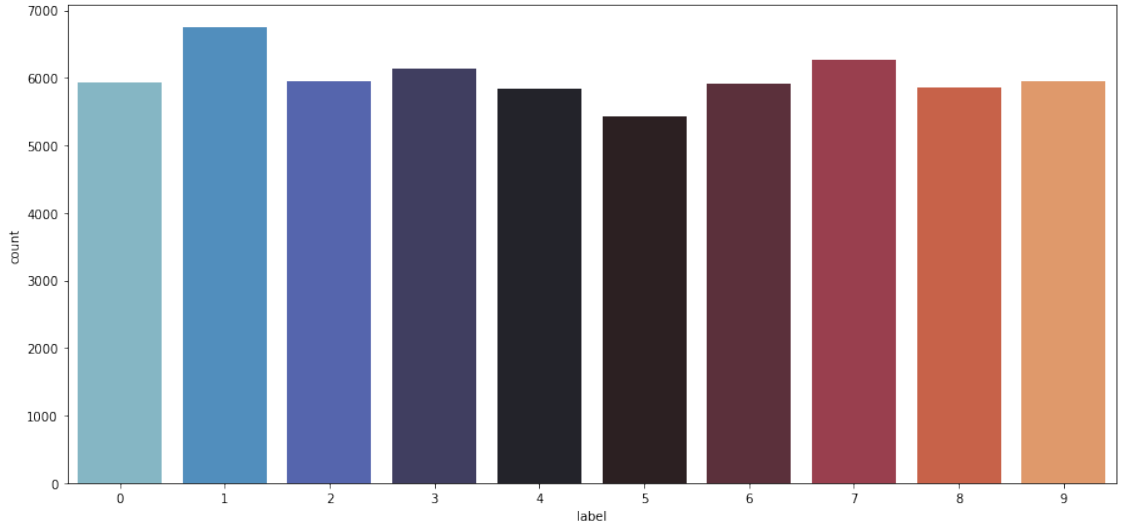
September 24, 2021

1 Mnist Visual Data Set Analysis

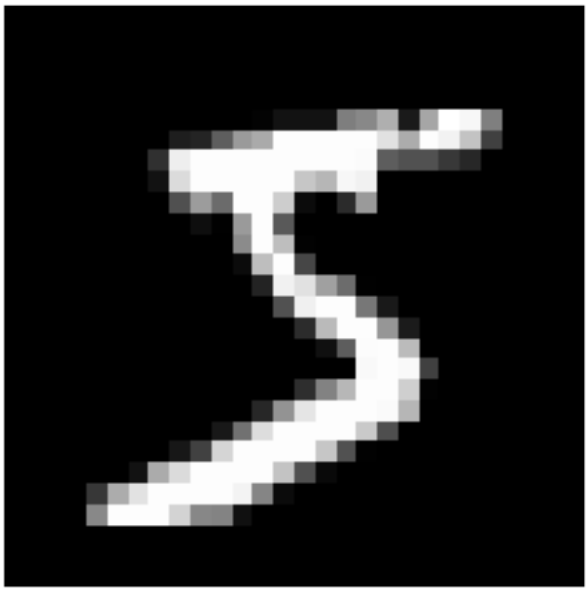
[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

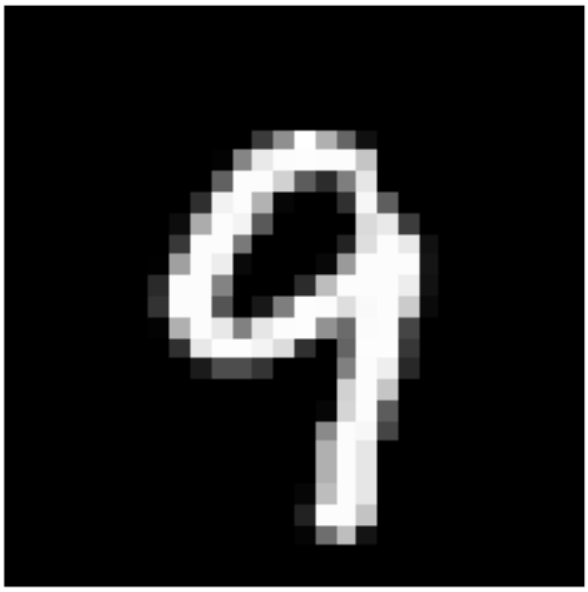
[2]:
```
trainSet = pd.read_csv("mnist_train.csv")
testSet = pd.read_csv("mnist_test.csv")

Y_train = trainSet["label"]
X_train = trainSet.drop(labels=["label"] , axis = 1)

testSetLabel = testSet["label"]
testSet = testSet.drop(labels=["label"] , axis = 1)
```

[4]:
```
#Visualize Some Data
import warnings
warnings.filterwarnings('ignore')

plt.figure(figsize=(15,7))
valueList = Y_train.value_counts()
sns.countplot(Y_train , palette="icefire")
plt.show()
```



[5]:
```
#Visualize Examples
img1 = X_train.loc[[0]].to_numpy()
img1 = img1.reshape((28,28))
plt.imshow(img1,cmap='gray')
plt.axis("off")
plt.show()

img2 = X_train.loc[[550]].to_numpy()
img2 = img2.reshape((28,28))
plt.imshow(img2,cmap='gray')
plt.axis("off")
plt.show()
```





[6]:
```
#Normalization
X_train = X_train / 255.0
testSet = testSet / 255.0

#Reshape
X_train = X_train.values.reshape(-1,28,28,1)
```

```
testSetData = testSet.values.reshape((-1,28,28,1))

#OneHotEncoding
from keras.utils.np_utils import to_categorical
Y_train = to_categorical(Y_train , num_classes=10)

#Train , Test
X_train , X_val , Y_train , Y_val = train_test_split(X_train , Y_train ,␣
 ↪test_size = 0.05 , random_state = 25)
```

[7]:
```
#Building a Model
from keras.models import Sequential
from keras.layers import Dense , Conv2D , MaxPooling2D , Dropout ,Flatten
from keras.preprocessing.image import ImageDataGenerator

#CNN - Step 1
model = Sequential()
model.add(Conv2D(filters = 16 , kernel_size = (5,5) , padding = "same" ,
                 activation='relu', input_shape=(28,28,1) , name =␣
 ↪"Conv-Input"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

#Step 2
model.add(Conv2D(filters = 16 , kernel_size = (3,3) , padding = "same" ,␣
 ↪activation='relu', name = "Conv-Hidden-1"))
model.add(MaxPooling2D(pool_size=(2,2) , strides=(2,2)))
model.add(Dropout(0.25))

#Step 3
model.add(Conv2D(filters = 16 , kernel_size = (3,3) , padding = "same" ,␣
 ↪activation='relu', name = "Conv-Hidden-2"))
model.add(MaxPooling2D(pool_size=(2,2) , strides=(2,2)))
model.add(Dropout(0.25))

#DL
model.add(Flatten())
model.add(Dense(256, activation = "relu" , name = "AI-Neural-Input"))
model.add(Dropout(0.25))
model.add(Dense(10 , activation= "softmax" , name = "AI-Neural-Output"))


model.compile(optimizer = "adam" , loss="categorical_crossentropy" ,␣
 ↪metrics=["accuracy"])
```

[8]:
```
model.compile(optimizer = "adam" , loss="categorical_crossentropy" ,␣
 ↪metrics=["accuracy"])
```

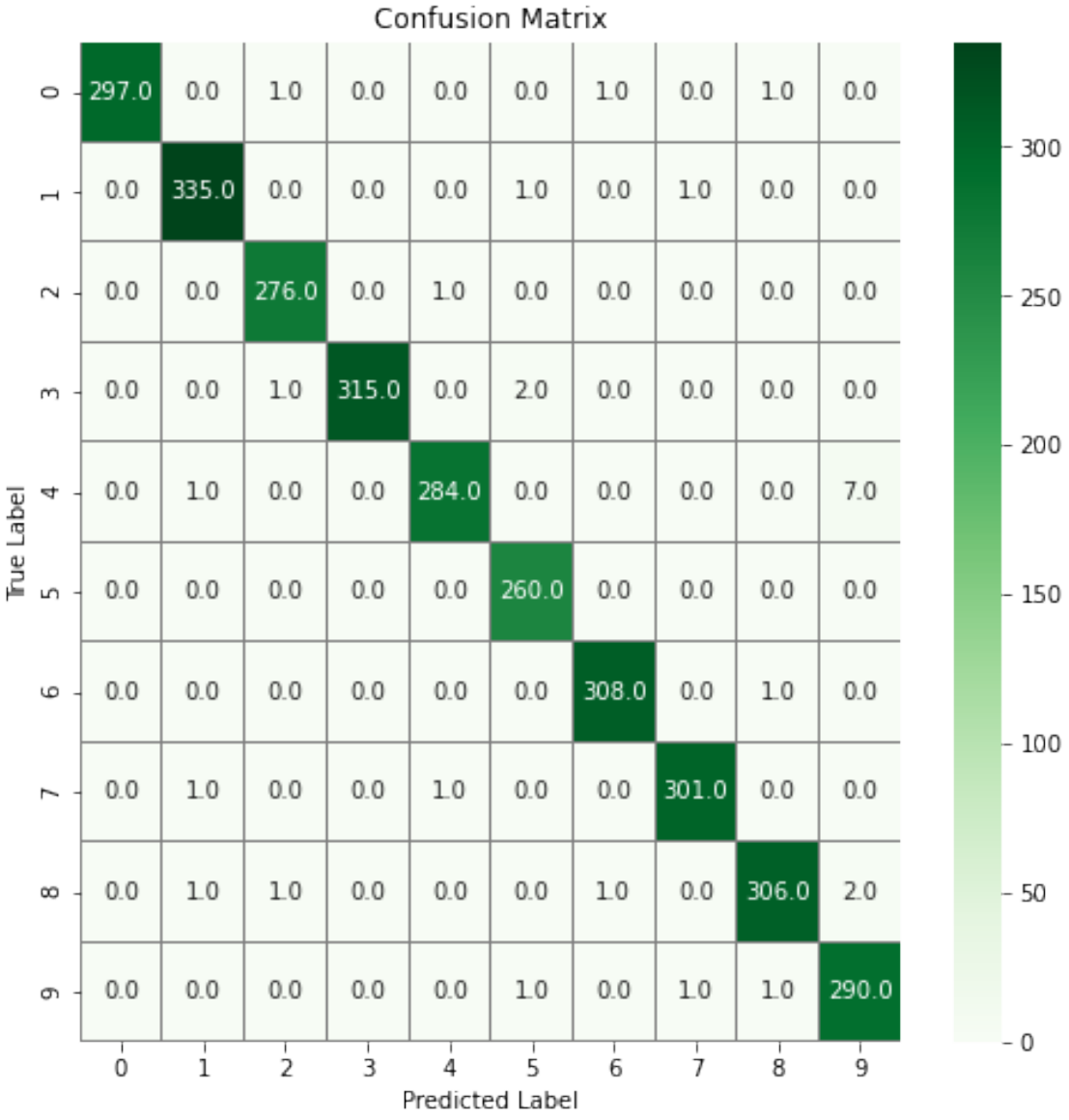
```
EPOCHS = 20
BATCH_SIZE = 125

imgDataGenerator = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=5,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)


imgDataGenerator.fit(X_train)

hist = model.fit_generator(imgDataGenerator.flow(X_train , Y_train ,␣
↪batch_size=BATCH_SIZE) ,
                           validation_data = (X_val,Y_val) ,epochs=EPOCHS,␣
↪steps_per_epoch=X_train.shape[0] // BATCH_SIZE)
```

```
WARNING:tensorflow:From <ipython-input-8-3d6f1fb68b2c>:22: Model.fit_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/20
456/456 [==============================] - 24s 53ms/step - loss: 0.8908 -
accuracy: 0.6972 - val_loss: 0.1192 - val_accuracy: 0.9687
Epoch 2/20
456/456 [==============================] - 24s 53ms/step - loss: 0.3473 -
accuracy: 0.8911 - val_loss: 0.0776 - val_accuracy: 0.9790
Epoch 3/20
456/456 [==============================] - 24s 52ms/step - loss: 0.2613 -
accuracy: 0.9191 - val_loss: 0.0591 - val_accuracy: 0.9840
Epoch 4/20
456/456 [==============================] - 24s 53ms/step - loss: 0.2206 -
accuracy: 0.9302 - val_loss: 0.0529 - val_accuracy: 0.9853
Epoch 5/20
456/456 [==============================] - 17s 38ms/step - loss: 0.1903 -
accuracy: 0.9404 - val_loss: 0.0499 - val_accuracy: 0.9843
Epoch 6/20
456/456 [==============================] - 14s 32ms/step - loss: 0.1727 -
accuracy: 0.9460 - val_loss: 0.0428 - val_accuracy: 0.9857
```

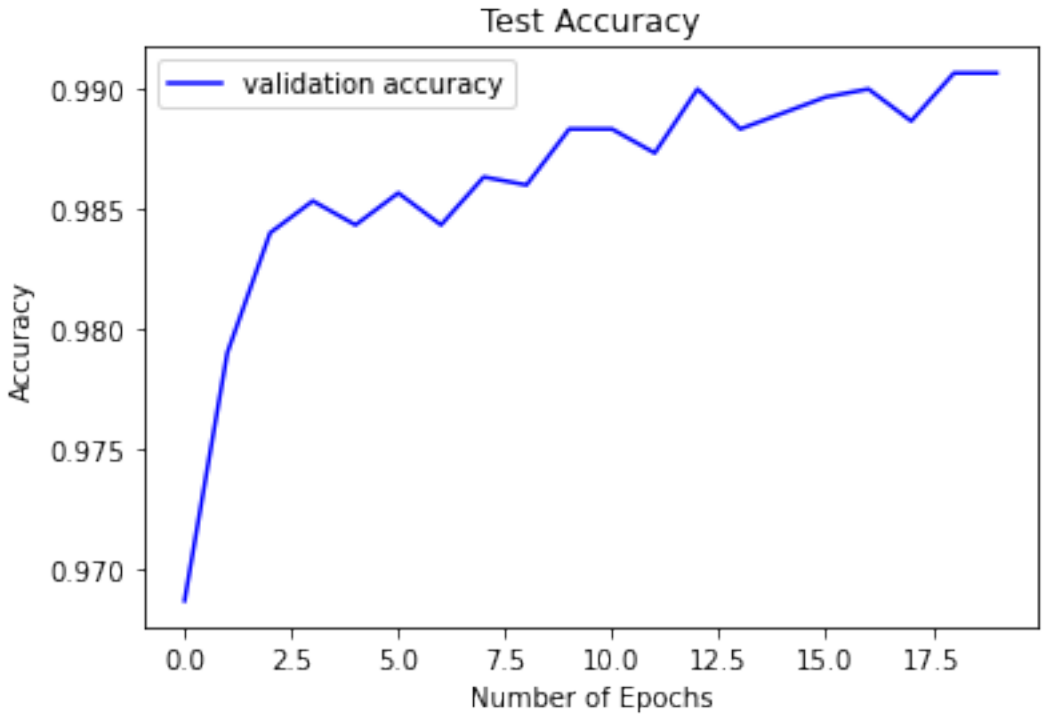
```
Epoch 7/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1542 -
accuracy: 0.9521 - val_loss: 0.0417 - val_accuracy: 0.9843
Epoch 8/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1480 -
accuracy: 0.9537 - val_loss: 0.0414 - val_accuracy: 0.9863
Epoch 9/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1372 -
accuracy: 0.9574 - val_loss: 0.0418 - val_accuracy: 0.9860
Epoch 10/20
456/456 [==============================] - 14s 32ms/step - loss: 0.1310 -
accuracy: 0.9587 - val_loss: 0.0361 - val_accuracy: 0.9883
Epoch 11/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1255 -
accuracy: 0.9617 - val_loss: 0.0348 - val_accuracy: 0.9883
Epoch 12/20
456/456 [==============================] - 14s 32ms/step - loss: 0.1285 -
accuracy: 0.9598 - val_loss: 0.0364 - val_accuracy: 0.9873
Epoch 13/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1218 -
accuracy: 0.9613 - val_loss: 0.0319 - val_accuracy: 0.9900
Epoch 14/20
456/456 [==============================] - 14s 32ms/step - loss: 0.1165 -
accuracy: 0.9632 - val_loss: 0.0344 - val_accuracy: 0.9883
Epoch 15/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1171 -
accuracy: 0.9641 - val_loss: 0.0318 - val_accuracy: 0.9890
Epoch 16/20
456/456 [==============================] - 14s 30ms/step - loss: 0.1116 -
accuracy: 0.9654 - val_loss: 0.0314 - val_accuracy: 0.9897
Epoch 17/20
456/456 [==============================] - 14s 30ms/step - loss: 0.1111 -
accuracy: 0.9648 - val_loss: 0.0284 - val_accuracy: 0.9900
Epoch 18/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1078 -
accuracy: 0.9666 - val_loss: 0.0343 - val_accuracy: 0.9887
Epoch 19/20
456/456 [==============================] - 14s 31ms/step - loss: 0.1041 -
accuracy: 0.9679 - val_loss: 0.0257 - val_accuracy: 0.9907
Epoch 20/20
456/456 [==============================] - 14s 32ms/step - loss: 0.1064 -
accuracy: 0.9673 - val_loss: 0.0282 - val_accuracy: 0.9907
```

[9]:
```
#Confussion Matrix
predictions = model.predict(X_val)
predictions_classes = np.argmax(predictions,axis = 1)
truePred = np.argmax(Y_val,axis = 1)
```

```
confusion_mtx = confusion_matrix(truePred , predictions_classes)

f,ax = plt.subplots(figsize=(8, 8))

sns.heatmap(confusion_mtx, annot=True, linewidths=0.
 ↪01,cmap="Greens",linecolor="gray", fmt= '.1f',ax=ax)

plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

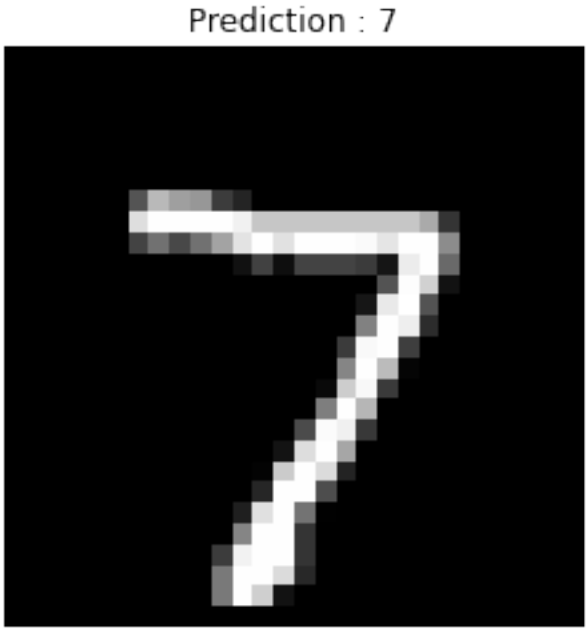


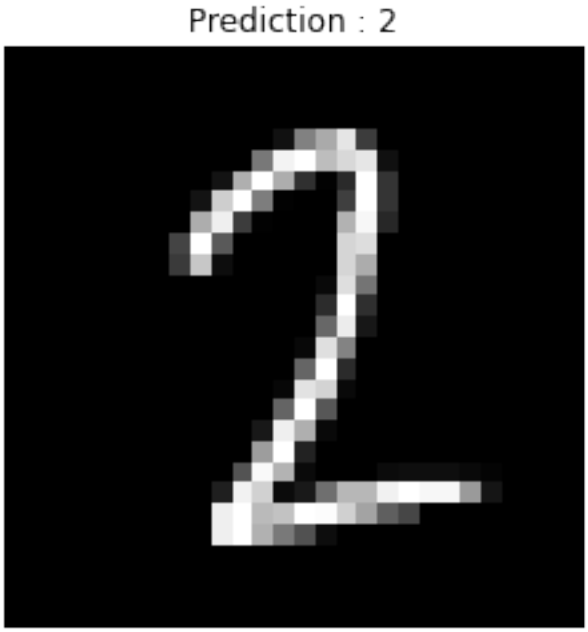
[10]:
```
plt.plot(hist.history['val_accuracy'], color='b', label="validation accuracy")
plt.title("Test Accuracy")
plt.xlabel("Number of Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[14]:
```
#Predict Test Data Set
predictions2 = model.predict(testSetData)
predictions2_classes = np.argmax(predictions2 , axis = 1)
predictions2List = np.argmax(predictions2 , axis = 1).tolist()

#Check Result
img1 = testSet.loc[[0]].to_numpy()
img1 = img1.reshape((28,28))
plt.imshow(img1,cmap='gray')
plt.title('Prediction : {}'.format(predictions2List[0]))
plt.axis("off")
plt.show()

img2 = testSet.loc[[456]].to_numpy()
img2 = img2.reshape((28,28))
plt.imshow(img2,cmap='gray')
```

```
plt.title('Prediction : {}'.format(predictions2List[456]))
plt.axis("off")
plt.show()
```





[ ]: