# imports for array-handling and plotting

import numpy as np

import matplotlib

matplotlib.use('agg')

import matplotlib.pyplot as plt

# let's keep our keras backend tensorflow quiet

import os

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL']='3'

# for testing on CPU

#os.environ['CUDA\_VISIBLE\_DEVICES'] = ''

# keras imports for the dataset and building our neural network

from keras.datasets import mnist

from keras.models import Sequential, load\_model

from keras.layers.core import Dense, Dropout, Activation

from keras.utils import np\_utils

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Digit: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

fig

fig = plt.figure()

plt.subplot(2,1,1)

plt.imshow(X\_train[0], cmap='gray', interpolation='none')

plt.title("Digit: {}".format(y\_train[0]))

plt.xticks([])

plt.yticks([])

plt.subplot(2,1,2)

plt.hist(X\_train[0].reshape(784))

plt.title("Pixel Value Distribution")

fig

# let's print the shape before we reshape and normalize

print("X\_train shape", X\_train.shape)

print("y\_train shape", y\_train.shape)

print("X\_test shape", X\_test.shape)

print("y\_test shape", y\_test.shape)

# building the input vector from the 28x28 pixels

X\_train = X\_train.reshape(60000, 784)

X\_test = X\_test.reshape(10000, 784)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

# normalizing the data to help with the training

X\_train /= 255

X\_test /= 255

# print the final input shape ready for training

print("Train matrix shape", X\_train.shape)

print("Test matrix shape", X\_test.shape)

print(np.unique(y\_train, return\_counts=True))

# one-hot encoding using keras' numpy-related utilities

n\_classes = 10

print("Shape before one-hot encoding: ", y\_train.shape)

Y\_train = np\_utils.to\_categorical(y\_train, n\_classes)

Y\_test = np\_utils.to\_categorical(y\_test, n\_classes)

print("Shape after one-hot encoding: ", Y\_train.shape)

# building a linear stack of layers with the sequential model

model = Sequential()

model.add(Dense(512, input\_shape=(784,)))

model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(Dense(10))

model.add(Activation('softmax'))

# compiling the sequential model

model.compile(loss='categorical\_crossentropy', metrics=['accuracy'], optimizer='adam')

# training the model and saving metrics in history

history = model.fit(X\_train, Y\_train,

batch\_size=128, epochs=20,

verbose=2,

validation\_data=(X\_test, Y\_test))

# saving the model

save\_dir = "/results/"

model\_name = 'keras\_mnist.h5'

model\_path = os.path.join(save\_dir, model\_name)

model.save(model\_path)

print('Saved trained model at %s ' % model\_path)

# plotting the metrics

fig = plt.figure()

plt.subplot(2,1,1)

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper right')

plt.tight\_layout()

fig

mnist\_model = load\_model($$ref{{["~:output","03ba7143-0469-4ab8-8850-a2a8fa3cb299","keras\_mnist.h5"]}})

loss\_and\_metrics = mnist\_model.evaluate(X\_test, Y\_test, verbose=2)

print("Test Loss", loss\_and\_metrics[0])

print("Test Accuracy", loss\_and\_metrics[1])

# load the model and create predictions on the test set

mnist\_model = load\_model($$ref{{["~:output","03ba7143-0469-4ab8-8850-a2a8fa3cb299","keras\_mnist.h5"]}})

predicted\_classes = mnist\_model.predict\_classes(X\_test)

# see which we predicted correctly and which not

correct\_indices = np.nonzero(predicted\_classes == y\_test)[0]

incorrect\_indices = np.nonzero(predicted\_classes != y\_test)[0]

print()

print(len(correct\_indices)," classified correctly")

print(len(incorrect\_indices)," classified incorrectly")

# adapt figure size to accomodate 18 subplots

plt.rcParams['figure.figsize'] = (7,14)

figure\_evaluation = plt.figure()

# plot 9 correct predictions

for i, correct in enumerate(correct\_indices[:9]):

plt.subplot(6,3,i+1)

plt.imshow(X\_test[correct].reshape(28,28), cmap='gray', interpolation='none')

plt.title(

"Predicted: {}, Truth: {}".format(predicted\_classes[correct],

y\_test[correct]))

plt.xticks([])

plt.yticks([])

# plot 9 incorrect predictions

for i, incorrect in enumerate(incorrect\_indices[:9]):

plt.subplot(6,3,i+10)

plt.imshow(X\_test[incorrect].reshape(28,28), cmap='gray', interpolation='none')

plt.title(

"Predicted {}, Truth: {}".format(predicted\_classes[incorrect],

y\_test[incorrect]))

plt.xticks([])

plt.yticks([])

figure\_evaluation