problem1

November 29, 2023

```
[ ]: # Import packages
     # DL Packages
     import tensorflow as tf
     import keras

     # Others
     import matplotlib.pyplot as plt
     import numpy as np
     import scipy as sp
     import sympy as sym
     import seaborn as sns

     from sklearn.metrics import confusion_matrix
```

1 Examine the Data:

Begin by downloading the data and looking at shapes:

```
[ ]: (Xtrain, Ytrain), (Xtest, Ytest) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

```
[ ]: print("Xtrain shape: ", Xtrain.shape)
     print("Xtrain min, max: ", Xtrain.min(), Xtrain.max())
     print("-----------------------")
     print("Ytrain sthape: ", Ytrain.shape)
     print("Ytrain classes: ", np.unique(Ytrain))
     print("-----------------------")
     print("Xtest.shape: ", Xtest.shape)
     print("Xtest min, max: ", Xtrain.min(), Xtrain.max())
     print("-----------------------")
     print("Ytest shape: ", Ytest.shape)
     print("Ytest classes: ", np.unique(Ytrain))
```

```
Xtrain shape:  (60000, 28, 28)
Xtrain min, max:  0 255
-----------------------
```

```
Ytrain sthape:  (60000,)
Ytrain classes:  [0 1 2 3 4 5 6 7 8 9]
-----------------------
Xtest.shape:  (10000, 28, 28)
Xtest min, max:  0 255
-----------------------
Ytest shape:  (10000,)
Ytest classes:  [0 1 2 3 4 5 6 7 8 9]
```

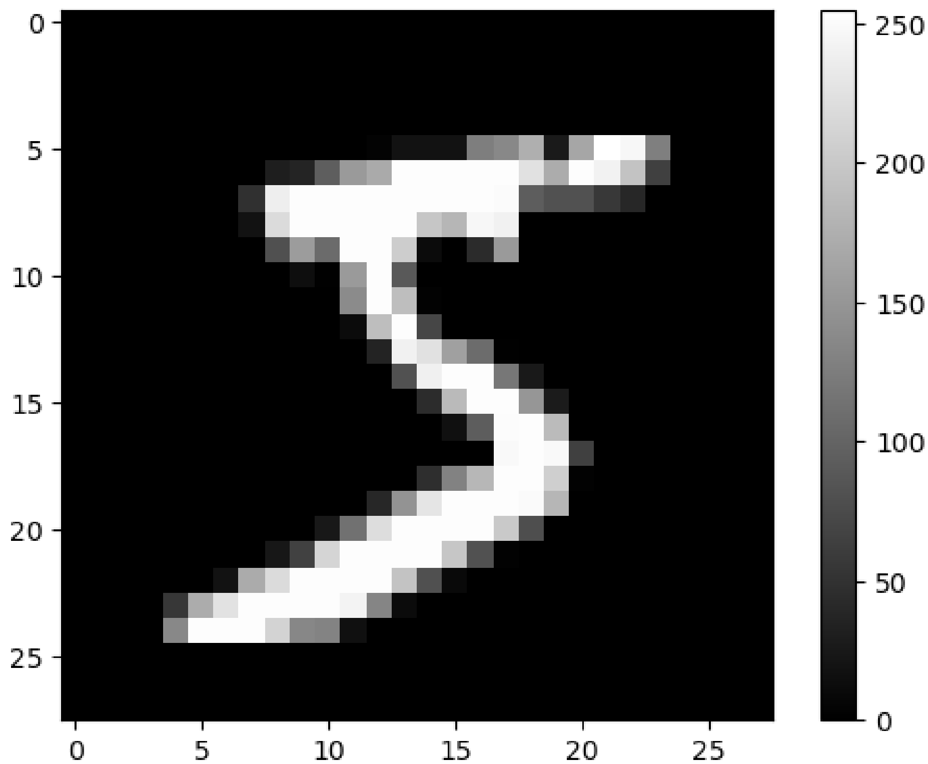
We have 60,000 training samples and 10,000 test samples. Let's look at some of the data:

[ ]:
```
plt.imshow(Xtrain[0], cmap="gray")
plt.colorbar()
Xtrain[0]
```

[ ]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
        253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
        253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
        253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
        205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
         90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
```

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
 253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
 241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
 253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
 253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
 195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
  11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```



As expected, it's a 28x28 integer array (0-255) that corresponds to a grayscale image of a handwritten digit!

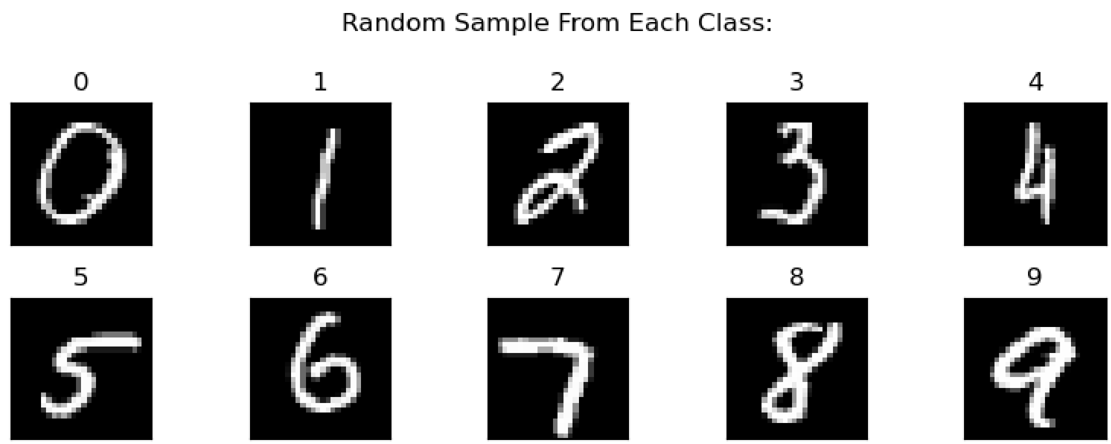
Now let's get a sense of the different classes present by plotting one of each:

[ ]:
```
label_vals = np.unique(Ytrain)
label_vals
```

[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

[ ]:
```
ncols = 5
nrows = 2
f, ax = plt.subplots(nrows=nrows, ncols=ncols)
f.set_size_inches(8,3)
plt.suptitle("Random Sample From Each Class:")
for i in range(nrows):
    for j in range(ncols):
        n = label_vals[i*ncols + j]
```

```
        is_n = np.nonzero(Ytrain==n)[0]
        random_i = np.random.choice(is_n)
        ax[i,j].imshow(Xtrain[random_i], origin="upper", cmap="gray")
        ax[i,j].set_aspect(1)
        ax[i,j].get_xaxis().set_visible(False)
        ax[i,j].get_yaxis().set_visible(False)
        ax[i,j].set_title(n)
plt.tight_layout()
```



2 Pre-Process Data:

First we will normalize the data to be between 0-1, instead of 0-255. We will also use one hot encoding to represent the data labels. To do the one hot encoding, use the code that I made for assignment 2:

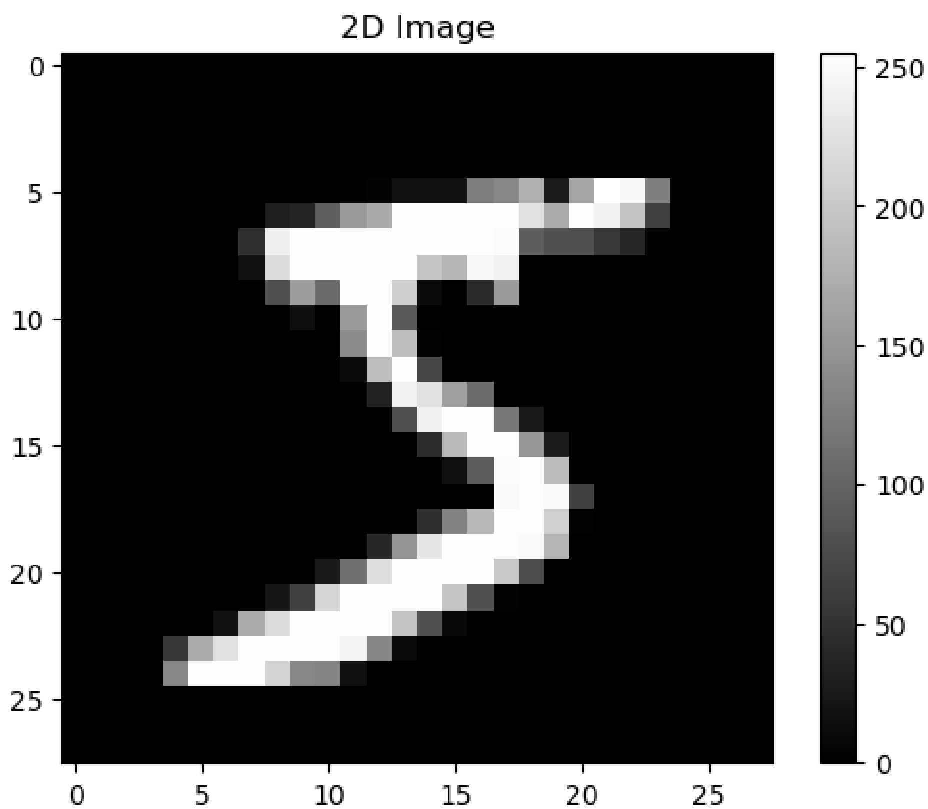
[ ]:
```
Xtrain_norm = Xtrain.astype(float)/np.max(Xtrain)
Xtest_norm = Xtest.astype(float)/np.max(Xtest)
def OHE(labels):

    Y = np.zeros((labels.size, 1), dtype=int)
    unique_vals = np.unique(labels)
    label_map = {}
    for i, val in enumerate(unique_vals):
        label_map[val] = i
    count = 0
    for i, label in enumerate(labels):
        # Assign label
        Y[i] = label_map[label]

    Y_OHE = np.zeros((Y.shape[0], len(label_map)), dtype=int)
    for i in range(Y.shape[0]):
```

```
        Y_OHE[i, Y[i]] = 1
    Y = Y_OHE

    return Y_OHE

Ytrain_OHE = OHE(Ytrain)
Ytest_OHE = OHE(Ytest)
```

Now look at the data, labels:

[ ]:
```
print("Ytrain[0]:", Ytrain[0], " Ytrain_OHE[0]:", Ytrain_OHE[0])
plt.imshow(Xtrain[0], cmap="gray")
plt.title("2D Image")
plt.colorbar();
```

```
Ytrain[0]: 5  Ytrain_OHE[0]: [0 0 0 0 0 1 0 0 0 0]
```



Looks good! Image is normalized to between 0-1 and 5 is encoded as a 1 in the [5] position. The flattened data seems right, as it's all zeros in the beginning and end of the sequence which corresponds to the top/bottom of the image.

3 Make/Train a Network:

[ ]:
```
input_size = Xtrain[0].shape
flat_size = input_size[0]*input_size[1]
output_size = 10 # one hot encoded label vals
model = keras.models.Sequential([
    keras.Input(shape=input_size),
    keras.layers.Flatten(),
    keras.layers.Dropout(0.1),
    keras.layers.Dense(flat_size, activation="relu"),
    keras.layers.Dense(flat_size/4, activation="relu"),
    keras.layers.Dense(output_size, activation="softmax")
], name="mnist_dense")
model.build(input_size)
model.compile(optimizer="adam", loss="categorical_crossentropy",
              metrics=[keras.metrics.CategoricalAccuracy()])
model.summary()
```

```
Model: "mnist_dense"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dropout (Dropout)           (None, 784)               0

 dense (Dense)               (None, 784)               615440

 dense_1 (Dense)             (None, 196)               153860

 dense_2 (Dense)             (None, 10)                1970

=================================================================
Total params: 771270 (2.94 MB)
Trainable params: 771270 (2.94 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
history = model.fit(Xtrain_norm, Ytrain_OHE, batch_size=1000, epochs=100,␣
 ↪validation_data=(Xtest_norm, Ytest_OHE))
```

```
Epoch 1/100
60/60 [==============================] - 1s 14ms/step - loss: 0.4508 -
categorical_accuracy: 0.8770 - val_loss: 0.1885 - val_categorical_accuracy:
0.9443
Epoch 2/100
60/60 [==============================] - 1s 13ms/step - loss: 0.1727 -
categorical_accuracy: 0.9485 - val_loss: 0.1260 - val_categorical_accuracy:
```

```
0.9623
Epoch 3/100
60/60 [==============================] - 1s 13ms/step - loss: 0.1176 -
categorical_accuracy: 0.9658 - val_loss: 0.0976 - val_categorical_accuracy:
0.9710
Epoch 4/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0843 -
categorical_accuracy: 0.9747 - val_loss: 0.0852 - val_categorical_accuracy:
0.9739
Epoch 5/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0695 -
categorical_accuracy: 0.9791 - val_loss: 0.0752 - val_categorical_accuracy:
0.9757
Epoch 6/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0537 -
categorical_accuracy: 0.9843 - val_loss: 0.0696 - val_categorical_accuracy:
0.9777
Epoch 7/100
60/60 [==============================] - 1s 12ms/step - loss: 0.0449 -
categorical_accuracy: 0.9866 - val_loss: 0.0645 - val_categorical_accuracy:
0.9800
Epoch 8/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0399 -
categorical_accuracy: 0.9877 - val_loss: 0.0648 - val_categorical_accuracy:
0.9802
Epoch 9/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0320 -
categorical_accuracy: 0.9905 - val_loss: 0.0610 - val_categorical_accuracy:
0.9802
Epoch 10/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0277 -
categorical_accuracy: 0.9914 - val_loss: 0.0680 - val_categorical_accuracy:
0.9792
Epoch 11/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0220 -
categorical_accuracy: 0.9938 - val_loss: 0.0552 - val_categorical_accuracy:
0.9841
Epoch 12/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0202 -
categorical_accuracy: 0.9939 - val_loss: 0.0583 - val_categorical_accuracy:
0.9821
Epoch 13/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0180 -
categorical_accuracy: 0.9947 - val_loss: 0.0565 - val_categorical_accuracy:
0.9829
Epoch 14/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0154 -
categorical_accuracy: 0.9956 - val_loss: 0.0550 - val_categorical_accuracy:
```

```
0.9828
Epoch 15/100
60/60 [==============================] - 1s 13ms/step - loss: 0.0139 -
categorical_accuracy: 0.9959 - val_loss: 0.0588 - val_categorical_accuracy:
0.9821
Epoch 16/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0139 -
categorical_accuracy: 0.9959 - val_loss: 0.0548 - val_categorical_accuracy:
0.9827
Epoch 17/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0116 -
categorical_accuracy: 0.9965 - val_loss: 0.0568 - val_categorical_accuracy:
0.9831
Epoch 18/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0093 -
categorical_accuracy: 0.9974 - val_loss: 0.0553 - val_categorical_accuracy:
0.9842
Epoch 19/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0108 -
categorical_accuracy: 0.9968 - val_loss: 0.0595 - val_categorical_accuracy:
0.9833
Epoch 20/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0100 -
categorical_accuracy: 0.9969 - val_loss: 0.0627 - val_categorical_accuracy:
0.9830
Epoch 21/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0080 -
categorical_accuracy: 0.9974 - val_loss: 0.0625 - val_categorical_accuracy:
0.9822
Epoch 22/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0077 -
categorical_accuracy: 0.9978 - val_loss: 0.0588 - val_categorical_accuracy:
0.9841
Epoch 23/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0077 -
categorical_accuracy: 0.9976 - val_loss: 0.0594 - val_categorical_accuracy:
0.9829
Epoch 24/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0076 -
categorical_accuracy: 0.9977 - val_loss: 0.0607 - val_categorical_accuracy:
0.9839
Epoch 25/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0062 -
categorical_accuracy: 0.9982 - val_loss: 0.0605 - val_categorical_accuracy:
0.9836
Epoch 26/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0061 -
categorical_accuracy: 0.9984 - val_loss: 0.0607 - val_categorical_accuracy:
```

```
0.9840
Epoch 27/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0069 -
categorical_accuracy: 0.9979 - val_loss: 0.0638 - val_categorical_accuracy:
0.9839
Epoch 28/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0074 -
categorical_accuracy: 0.9975 - val_loss: 0.0632 - val_categorical_accuracy:
0.9831
Epoch 29/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0049 -
categorical_accuracy: 0.9987 - val_loss: 0.0619 - val_categorical_accuracy:
0.9836
Epoch 30/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0041 -
categorical_accuracy: 0.9989 - val_loss: 0.0591 - val_categorical_accuracy:
0.9848
Epoch 31/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0046 -
categorical_accuracy: 0.9985 - val_loss: 0.0634 - val_categorical_accuracy:
0.9850
Epoch 32/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0052 -
categorical_accuracy: 0.9983 - val_loss: 0.0641 - val_categorical_accuracy:
0.9836
Epoch 33/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0060 -
categorical_accuracy: 0.9980 - val_loss: 0.0698 - val_categorical_accuracy:
0.9817
Epoch 34/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0059 -
categorical_accuracy: 0.9983 - val_loss: 0.0649 - val_categorical_accuracy:
0.9841
Epoch 35/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0060 -
categorical_accuracy: 0.9983 - val_loss: 0.0703 - val_categorical_accuracy:
0.9826
Epoch 36/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0057 -
categorical_accuracy: 0.9983 - val_loss: 0.0709 - val_categorical_accuracy:
0.9843
Epoch 37/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0069 -
categorical_accuracy: 0.9976 - val_loss: 0.0638 - val_categorical_accuracy:
0.9837
Epoch 38/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0054 -
categorical_accuracy: 0.9981 - val_loss: 0.0743 - val_categorical_accuracy:
```

```
0.9816
Epoch 39/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0052 -
categorical_accuracy: 0.9984 - val_loss: 0.0669 - val_categorical_accuracy:
0.9827
Epoch 40/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0068 -
categorical_accuracy: 0.9979 - val_loss: 0.0691 - val_categorical_accuracy:
0.9839
Epoch 41/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0056 -
categorical_accuracy: 0.9982 - val_loss: 0.0723 - val_categorical_accuracy:
0.9831
Epoch 42/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0046 -
categorical_accuracy: 0.9984 - val_loss: 0.0669 - val_categorical_accuracy:
0.9842
Epoch 43/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0040 -
categorical_accuracy: 0.9987 - val_loss: 0.0784 - val_categorical_accuracy:
0.9822
Epoch 44/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0058 -
categorical_accuracy: 0.9981 - val_loss: 0.0687 - val_categorical_accuracy:
0.9832
Epoch 45/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0041 -
categorical_accuracy: 0.9987 - val_loss: 0.0678 - val_categorical_accuracy:
0.9835
Epoch 46/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0037 -
categorical_accuracy: 0.9989 - val_loss: 0.0709 - val_categorical_accuracy:
0.9838
Epoch 47/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0042 -
categorical_accuracy: 0.9986 - val_loss: 0.0744 - val_categorical_accuracy:
0.9829
Epoch 48/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0052 -
categorical_accuracy: 0.9983 - val_loss: 0.0651 - val_categorical_accuracy:
0.9832
Epoch 49/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0059 -
categorical_accuracy: 0.9981 - val_loss: 0.0765 - val_categorical_accuracy:
0.9825
Epoch 50/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0049 -
categorical_accuracy: 0.9984 - val_loss: 0.0735 - val_categorical_accuracy:
```

```
0.9842
Epoch 51/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0048 -
categorical_accuracy: 0.9985 - val_loss: 0.0770 - val_categorical_accuracy:
0.9838
Epoch 52/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0046 -
categorical_accuracy: 0.9985 - val_loss: 0.0665 - val_categorical_accuracy:
0.9847
Epoch 53/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0032 -
categorical_accuracy: 0.9990 - val_loss: 0.0750 - val_categorical_accuracy:
0.9843
Epoch 54/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0043 -
categorical_accuracy: 0.9984 - val_loss: 0.0702 - val_categorical_accuracy:
0.9843
Epoch 55/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0028 -
categorical_accuracy: 0.9991 - val_loss: 0.0652 - val_categorical_accuracy:
0.9852
Epoch 56/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0026 -
categorical_accuracy: 0.9993 - val_loss: 0.0706 - val_categorical_accuracy:
0.9838
Epoch 57/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0041 -
categorical_accuracy: 0.9987 - val_loss: 0.0650 - val_categorical_accuracy:
0.9865
Epoch 58/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0031 -
categorical_accuracy: 0.9990 - val_loss: 0.0771 - val_categorical_accuracy:
0.9847
Epoch 59/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0028 -
categorical_accuracy: 0.9991 - val_loss: 0.0729 - val_categorical_accuracy:
0.9852
Epoch 60/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0034 -
categorical_accuracy: 0.9987 - val_loss: 0.0999 - val_categorical_accuracy:
0.9825
Epoch 61/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0065 -
categorical_accuracy: 0.9980 - val_loss: 0.0839 - val_categorical_accuracy:
0.9840
Epoch 62/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0062 -
categorical_accuracy: 0.9978 - val_loss: 0.0740 - val_categorical_accuracy:
```

```
0.9839
Epoch 63/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0038 -
categorical_accuracy: 0.9987 - val_loss: 0.0837 - val_categorical_accuracy:
0.9837
Epoch 64/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0046 -
categorical_accuracy: 0.9983 - val_loss: 0.0855 - val_categorical_accuracy:
0.9837
Epoch 65/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0040 -
categorical_accuracy: 0.9988 - val_loss: 0.0753 - val_categorical_accuracy:
0.9849
Epoch 66/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0024 -
categorical_accuracy: 0.9991 - val_loss: 0.0780 - val_categorical_accuracy:
0.9846
Epoch 67/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0022 -
categorical_accuracy: 0.9993 - val_loss: 0.0743 - val_categorical_accuracy:
0.9847
Epoch 68/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0039 -
categorical_accuracy: 0.9987 - val_loss: 0.0834 - val_categorical_accuracy:
0.9830
Epoch 69/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0035 -
categorical_accuracy: 0.9988 - val_loss: 0.0776 - val_categorical_accuracy:
0.9835
Epoch 70/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0034 -
categorical_accuracy: 0.9988 - val_loss: 0.0766 - val_categorical_accuracy:
0.9857
Epoch 71/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0021 -
categorical_accuracy: 0.9994 - val_loss: 0.0807 - val_categorical_accuracy:
0.9839
Epoch 72/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0027 -
categorical_accuracy: 0.9990 - val_loss: 0.0840 - val_categorical_accuracy:
0.9832
Epoch 73/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0040 -
categorical_accuracy: 0.9988 - val_loss: 0.0719 - val_categorical_accuracy:
0.9846
Epoch 74/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0034 -
categorical_accuracy: 0.9990 - val_loss: 0.0740 - val_categorical_accuracy:
```

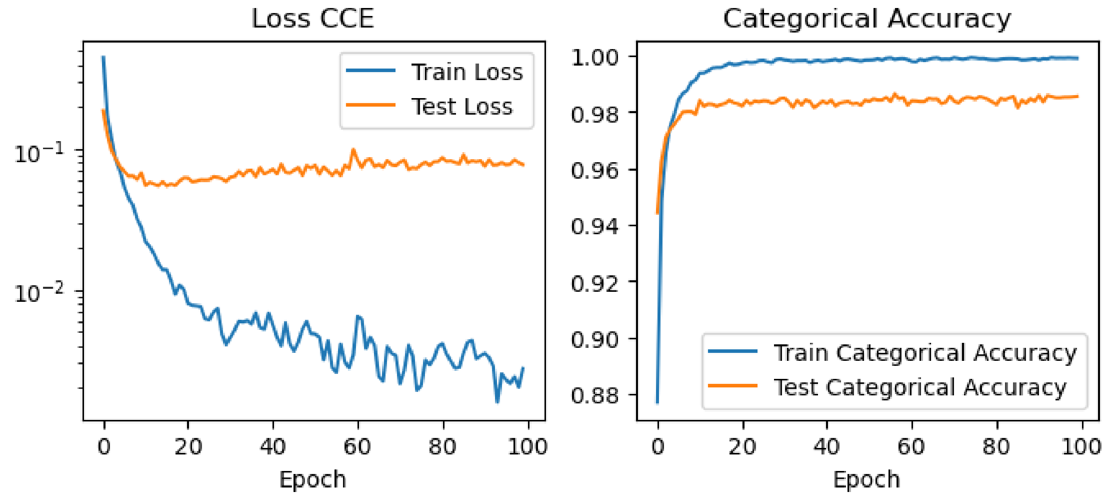
```
0.9853
Epoch 75/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0019 -
categorical_accuracy: 0.9994 - val_loss: 0.0730 - val_categorical_accuracy:
0.9851
Epoch 76/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0021 -
categorical_accuracy: 0.9992 - val_loss: 0.0778 - val_categorical_accuracy:
0.9846
Epoch 77/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0032 -
categorical_accuracy: 0.9990 - val_loss: 0.0809 - val_categorical_accuracy:
0.9855
Epoch 78/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0029 -
categorical_accuracy: 0.9991 - val_loss: 0.0769 - val_categorical_accuracy:
0.9851
Epoch 79/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0033 -
categorical_accuracy: 0.9987 - val_loss: 0.0815 - val_categorical_accuracy:
0.9845
Epoch 80/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0039 -
categorical_accuracy: 0.9986 - val_loss: 0.0819 - val_categorical_accuracy:
0.9836
Epoch 81/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0041 -
categorical_accuracy: 0.9985 - val_loss: 0.0870 - val_categorical_accuracy:
0.9827
Epoch 82/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0035 -
categorical_accuracy: 0.9988 - val_loss: 0.0821 - val_categorical_accuracy:
0.9849
Epoch 83/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0031 -
categorical_accuracy: 0.9990 - val_loss: 0.0828 - val_categorical_accuracy:
0.9844
Epoch 84/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0028 -
categorical_accuracy: 0.9990 - val_loss: 0.0804 - val_categorical_accuracy:
0.9850
Epoch 85/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0028 -
categorical_accuracy: 0.9990 - val_loss: 0.0790 - val_categorical_accuracy:
0.9856
Epoch 86/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0035 -
categorical_accuracy: 0.9989 - val_loss: 0.0917 - val_categorical_accuracy:
```

```
0.9815
Epoch 87/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0042 -
categorical_accuracy: 0.9985 - val_loss: 0.0800 - val_categorical_accuracy:
0.9842
Epoch 88/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0043 -
categorical_accuracy: 0.9985 - val_loss: 0.0830 - val_categorical_accuracy:
0.9831
Epoch 89/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0032 -
categorical_accuracy: 0.9988 - val_loss: 0.0816 - val_categorical_accuracy:
0.9848
Epoch 90/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0034 -
categorical_accuracy: 0.9990 - val_loss: 0.0832 - val_categorical_accuracy:
0.9839
Epoch 91/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0035 -
categorical_accuracy: 0.9985 - val_loss: 0.0758 - val_categorical_accuracy:
0.9859
Epoch 92/100
60/60 [==============================] - 1s 16ms/step - loss: 0.0033 -
categorical_accuracy: 0.9990 - val_loss: 0.0841 - val_categorical_accuracy:
0.9836
Epoch 93/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0029 -
categorical_accuracy: 0.9989 - val_loss: 0.0772 - val_categorical_accuracy:
0.9859
Epoch 94/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0016 -
categorical_accuracy: 0.9994 - val_loss: 0.0777 - val_categorical_accuracy:
0.9855
Epoch 95/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0025 -
categorical_accuracy: 0.9992 - val_loss: 0.0804 - val_categorical_accuracy:
0.9851
Epoch 96/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0023 -
categorical_accuracy: 0.9992 - val_loss: 0.0776 - val_categorical_accuracy:
0.9851
Epoch 97/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0021 -
categorical_accuracy: 0.9992 - val_loss: 0.0784 - val_categorical_accuracy:
0.9853
Epoch 98/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0024 -
categorical_accuracy: 0.9993 - val_loss: 0.0844 - val_categorical_accuracy:
```

```
0.9853
Epoch 99/100
60/60 [==============================] - 1s 14ms/step - loss: 0.0020 -
categorical_accuracy: 0.9992 - val_loss: 0.0803 - val_categorical_accuracy:
0.9854
Epoch 100/100
60/60 [==============================] - 1s 15ms/step - loss: 0.0027 -
categorical_accuracy: 0.9991 - val_loss: 0.0778 - val_categorical_accuracy:
0.9856
```

4 Evalute Results:

[ ]:
```
history.history.keys()
f, ax = plt.subplots(ncols=2)
f.set_size_inches(8,3)
ax[0].plot(history.history["loss"], label="Train Loss")
ax[0].set_title("Loss CCE")
ax[0].plot(history.history["val_loss"], label="Test Loss")
ax[0].set_yscale("log")
ax[0].set_xlabel("Epoch")
ax[0].legend()
ax[1].plot(history.history["categorical_accuracy"], label="Train Categorical␣
↪Accuracy")
ax[1].set_xlabel("Epoch")
ax[1].set_title("Categorical Accuracy")
ax[1].plot(history.history["val_categorical_accuracy"], label="Test Categorical␣
↪Accuracy")
ax[1].legend();
```



We see that even though the training loss continues to decrease, we saturate our test loss/accuravey after only a few epochs. With a higher dropout rate we might be able to avoid this.

To plot confusion matrix, again take code from assignment 2:

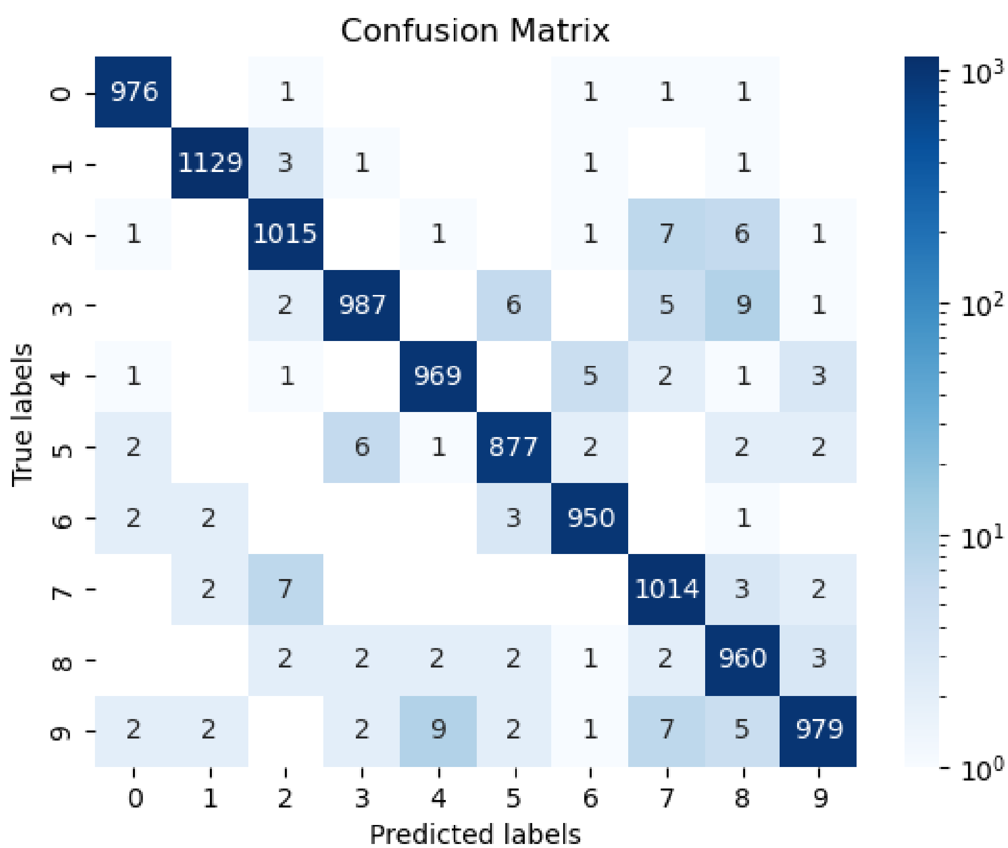
```
[ ]: def plot_confusion_matrix(Y: np.array, pred: np.array, labels=[], savename="",␣
     ↪logscale=False):
         """
         Convenience function for generating a confusion Matrix

         Args:
             Y (np.array): Actual labels for the dataset (n rows, 1 column)
             pred (np.array): Predicted labels for the data (n rows, 1 column)
             labels (list of str): class labels
             savename (str, optional): File to save plot to. If none is given shows␣
     ↪figure.
                                         Defaults to "".

         Returns:
             confusion matrix
         """
         # Figure out predicted class -- infer from Y and pred the number of classes
         if Y.shape[1] > 1:
             Y_labels = np.zeros(Y.shape[0], dtype=int)
             pred_labels = np.zeros_like(Y_labels)
             for i in range(Y.shape[0]):
                 Y_labels[i] = np.argmax(Y[i])
                 pred_labels[i] = np.argmax(pred[i])
         else:
             Y_labels = Y
             pred_labels = (Y >= 0.5).astype(int)
         cm = confusion_matrix(Y_labels, pred_labels)
         f, ax = plt.subplots()
         if logscale:
             from matplotlib.colors import LogNorm, Normalize
             sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap='Blues',␣
     ↪norm=LogNorm())
         else:
             sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap='Blues')
         # labels, title and ticks
         ax.set_xlabel("Predicted labels")
         ax.set_ylabel("True labels")
         ax.set_title("Confusion Matrix")
         if not labels:
             labels = np.arange(max(Y.shape[1], 2))
         ax.xaxis.set_ticklabels(labels)
         ax.yaxis.set_ticklabels(labels)

         if savename != "":
```

```
        plt.savefig(savename)
        plt.close(f)
    else:
        plt.show()

    return cm
```

Plot the confusion matrix with a log colorbar so we can see the nonzero off diagonal entries more easily. Note anything missing is 0.

[ ]:
```
Ypred = model.predict(Xtest_norm)
plot_confusion_matrix(Ytest_OHE, Ypred, logscale=True)
```

```
313/313 [==============================] - 0s 806us/step
```



[ ]:
```
array([[ 976,    0,    1,    0,    0,    0,    1,    1,    1,    0],
       [   0, 1129,    3,    1,    0,    0,    1,    0,    1,    0],
       [   1,    0, 1015,    0,    1,    0,    1,    7,    6,    1],
       [   0,    0,    2,  987,    0,    6,    0,    5,    9,    1],
       [   1,    0,    1,    0,  969,    0,    5,    2,    1,    3],
       [   2,    0,    0,    6,    1,  877,    2,    0,    2,    2],
```

```
[   2,    2,    0,    0,    0,    3,  950,    0,    1,    0],
[   0,    2,    7,    0,    0,    0,    0, 1014,    3,    2],
[   0,    0,    2,    2,    2,    2,    1,    2,  960,    3],
[   2,    2,    0,    2,    9,    2,    1,    7,    5,  979]])
```

The most confused digits seem to be 4 & 9, 9 & 7, and 3 & 8, which makes some sense!