1. **Data preparation, exploration, visualization**

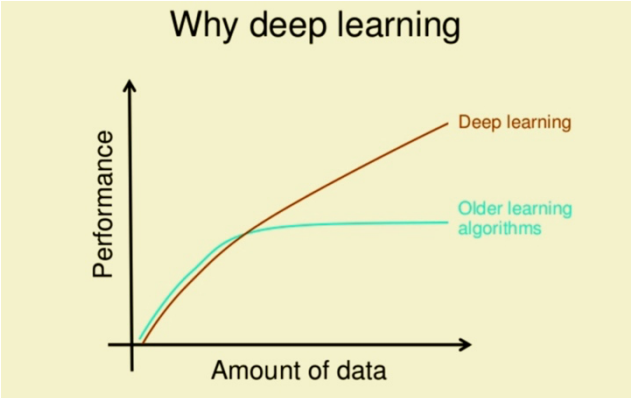
   In this assignment, not much needed to be done in terms of preparing the data as much of the data was image data sets. In this assignment, the only data that I need to obtain were the values in both the test and train sets. To get the values in the training set, I used the iloc method in order to get the values in all of the columns except the label column since label is what is being predicted. To get the values in the test set, I simply used the values method in order to get the values in the test set. To get the labels, I again used the iloc method to get all of the values in the label column of the training set. I then normalized the data to the scale 0-1 by dividing the values in the training and test set by 255. Finally, I created a pair of validation and training sets within the training sets that were created before.

2. **Review research design and modeling methods**

   This assignment entailed creating Neural Networks using TensorFlow. In this assignment, I created four such neural networks, a neural network containing two hidden layers with ten neurons each, a neural network containing two hidden layers with twenty neurons each, a neural network containing five hidden layers with ten neurons each, and a neural network containing five hidden layers with twenty neurons each. Creating each of the four neural networks involved initializing the neural network using tf.keras.models.Sequential and then passing the number of hidden layers, the number of neurons, and the shape of the data as parameters. I then used the compile method to configure the neural network for training and passed parameters such as optimizer, loss, and metrics. Finally, I fitted the neural network to both the training and validation data, and it was then used to predict on the test data. At this point the predictions generated by the neural network represent the likelihood of a particular label corresponding to the respective number and I was able to figure out which labels correspond to which numbers using the argmax method.

3. **Review results, evaluate models**

   Compared to the results generated by the methods that were used in last week's assignment such as, PCA, Random Forest Classifiers, Train-Test split and clustering, the results generated by the neural networks method were much better. In addition, the performance of the neural networks was considerably better as well. I noticed that the depth of the neural network is proportional to the score that I received when submitted to Kaggle. In neural networks, the depth is represented by the number of layers and therefore, the neural networks that had more layers such as the 5x10 and 5x20 had better scores than the 2x10 and 2x20 which had fewer number of layers. However, a deeper and wider neural network has more information capacity and is therefore prone to overfitting, thereby resulting in a poorer score. In the future, I would certainly investigate the impact that adding more layers has on overfitting the data. When submitted to Kaggle, the 2x10, 2x20, 5x10, and 5x20 neural networks resulted in a score of 0.93332, 0.94921, 0.95339, 0.95339 respectively whereas in last week's assignment, the train test split resulted in a score of 0.89780, principal components analysis resulted in a score of 0.85920, and random forests resulted in a score of 0.83760. Moreover, increasing the depth of the neural network by adding more layers seems to improve the performance mainly because more data is being captured. One possible explanation for why the neural networks performed better than the methods used in last week's assignment is because compared to other machine learning methods such as random forests and principal components, the performance of a deep learning method such as neural networks increases in proportion to the amount of the data that the model is training on whereas in traditional machine learning methods, the performance remains more or less constant because feature engineering becomes more difficult as more data is used.



Source – https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

4. **Kaggle Submission Relative to Peers**

Username – anaswarj

Neural Networks

Number of layers	Nodes per layer	Processing time	Kaggle Score
2	10	47 seconds	0.93332
2	20	46 seconds	0.94921
5	10	55 seconds	0.95339
5	20	51 seconds	0.95339

Traditional Machine Learning Models

Model	Processing time	F1 Score	Kaggle Score
Random Forests	7 mins 30 seconds (fitting) 3 seconds (prediction)	0.9193333333333333	0.83760
Principal Components	12 min 47 seconds (fitting) 3.34 seconds (prediction)	0.9502666666666667	0.85920

5. **Exposition, problem description and management recommendations**

From the benchmark study, I can conclude that neural networks are definitely better than other machine learning methods such as random forests and principal components. I would recommend a neural network that is wide and deep as it can not only capture more information but capture abstract features as well. That being said, while this type of neural network is certainly advantageous, it is also prone to overfitting due to the increased information capacity of the network and therefore, the accuracy and performance will be affected. I am not quite sure which hyperparameter settings I would recommend as being the most trustworthy; however, some techniques for finding the most trustworthy hyperparameters include cross validation through grid search and random search. I was not able to implement these techniques to find the most trustworthy hyperparameters, however in the future I would certainly implement the cross validation technique.

MSDS_422_Assignment_6

May 9, 2021

1 Assignment 6: Neural Networks

You will continue work on the Digit Recognition problem in Kaggle.com this week. As in Assignment 5, we will assess classification performance accuracy and processing time. Python TensorFlow should be used for Assignment 6. (If you experience difficulty installing TensorFlow, Python scikit-learn may be used as an alternative for Assignment 6.)

The Benchmark Experiment Tested neural network structures should be explored within a benchmark experiment, a factorial design with at least two levels on each of two experimental factors (at least a 2x2 completely crossed design). But due to the time required to fit each neural network, we will observe only one trial for each cell in the design. You will build your models on train.csv and submit your forecasts for test.csv to Kaggle.com, providing your name and user ID for each experimental trial..

An example experiment could include two values for the number of nodes per inner layer and two values for the number of inner layers. Various machine learning hyperparameter settings may be used.

Students are encouraged to work in study teams on this assignment, with the understanding that each student must run the code himself/herself and write an independent report of the experimental results.

In summary, this assignment asks you to fit a number of neural networks, comparing processing time and performance across experimental treatments. Processing time will be recorded for the fitting on the train.csv. Kaggle.com accuracy scores will be reported for all benchmarks.

Management Question Suppose you are a financial institution evaluating machine learning technologies for optical character recognition. Initial testing is on the MNIST digits. What can you conclude from your benchmark study? Which neural network typology and hyperparameter settings would you recommend as being the most trustworthy?

2 Data preprocessing

[ ]:
```
import tensorflow as tf
import tensorflow_datasets as tfds
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

[ ]:
```
!ls drive/MyDrive/
```

```
'Colab Notebooks'             original_mnist_test.csv
 five_by_ten_labels.csv       original_mnist_train.csv
 five_by_twenty_labels.csv    two_by_ten_labels.csv
 IMG_0711.jpg                 two_by_twenty_labels.csv
```

[ ]:
```
import pandas as pd

ds_train = pd.read_csv('/content/drive/MyDrive/original_mnist_train.csv')
ds_test = pd.read_csv('/content/drive/MyDrive/original_mnist_test.csv')
```

[ ]:
```
ds_train.shape
```

[ ]:
```
(42000, 785)
```

[ ]:
```
ds_test.shape
```

[ ]:
```
(28000, 784)
```

[ ]:
```
ds_train.head()
```

[ ]:

	label	pixel0	pixel1	pixel2	…	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	…	0	0	0	0
1	0	0	0	0	…	0	0	0	0
2	1	0	0	0	…	0	0	0	0
3	4	0	0	0	…	0	0	0	0
4	0	0	0	0	…	0	0	0	0

[5 rows x 785 columns]

[ ]:
```
ds_test.head()
```

[ ]:

	pixel0	pixel1	pixel2	pixel3	…	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0	…	0	0	0	0
1	0	0	0	0	…	0	0	0	0
2	0	0	0	0	…	0	0	0	0
3	0	0	0	0	…	0	0	0	0
4	0	0	0	0	…	0	0	0	0

[5 rows x 784 columns]

[ ]:
```
X = ds_train.iloc[:, 1:].values
X
```

[ ]:
```
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]])
```

[ ]:
```
X_test = ds_test.values
X_test
```

[ ]:
```
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]])
```

[ ]:
```
y = ds_train.iloc[:, 0].values
y
```

[ ]:
```
array([1, 0, 1, …, 7, 6, 9])
```

[ ]:
```
X.shape
```

[ ]:
```
(42000, 784)
```

[ ]:
```
X_test.shape
```

[ ]:
```
(28000, 784)
```

[ ]:
```
y.shape
```

[ ]:
```
(42000,)
```

[ ]:
```
import numpy as np
```

[ ]:
```
X = X.astype(np.float32) / 255.
```

[ ]:
```
X.astype(np.float32)
```

[ ]:
```
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.]], dtype=float32)
```

[ ]:
```
X_test = X_test.astype(np.float32) / 255.
```

[ ]:
```
X_test.astype(np.float32)
```

[ ]:
```
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.]], dtype=float32)
```

[ ]:
```
#np.random.shuffle(X)
```

[ ]:
```
X_val, X_train = X[:5120], X[5120:]
y_val, y_train = y[:5120], y[5120:]
```

[ ]:
```
X_val
```

[ ]:
```
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.]], dtype=float32)
```

[ ]:
```
X_train
```

[ ]:
```
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.]], dtype=float32)
```

[ ]:
```
y_val
```

[ ]:
```
array([1, 0, 1, …, 7, 9, 2])
```

[ ]:
```
y_train
```

[ ]:
```
array([9, 9, 8, …, 7, 6, 9])
```

[ ]:
```
X_val.shape
```

[ ]:
```
(5120, 784)
```

[ ]:
```
X_train.shape
```

[ ]: (36880, 784)

[ ]:
```
y_val.shape
```

[ ]: (5120,)

[ ]:
```
y_train.shape
```

[ ]: (36880,)

3 Neural Network - 2 hidden layers each with 10 neurons

[64]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Input(shape=(784,)),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    x = X_train, y = y_train,
    epochs=25,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/25
1153/1153 [==============================] - 2s 2ms/step - loss: 1.1596 -
sparse_categorical_accuracy: 0.6327 - val_loss: 0.3859 -
val_sparse_categorical_accuracy: 0.8869
Epoch 2/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.3623 -
sparse_categorical_accuracy: 0.8946 - val_loss: 0.3058 -
val_sparse_categorical_accuracy: 0.9127
Epoch 3/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.3046 -
sparse_categorical_accuracy: 0.9102 - val_loss: 0.2975 -
val_sparse_categorical_accuracy: 0.9172
Epoch 4/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2706 -
sparse_categorical_accuracy: 0.9230 - val_loss: 0.2604 -
val_sparse_categorical_accuracy: 0.9273
Epoch 5/25
```

```
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2485 -
sparse_categorical_accuracy: 0.9262 - val_loss: 0.2639 -
val_sparse_categorical_accuracy: 0.9219
Epoch 6/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2369 -
sparse_categorical_accuracy: 0.9308 - val_loss: 0.2438 -
val_sparse_categorical_accuracy: 0.9279
Epoch 7/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2270 -
sparse_categorical_accuracy: 0.9322 - val_loss: 0.2393 -
val_sparse_categorical_accuracy: 0.9330
Epoch 8/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2290 -
sparse_categorical_accuracy: 0.9318 - val_loss: 0.2556 -
val_sparse_categorical_accuracy: 0.9277
Epoch 9/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2069 -
sparse_categorical_accuracy: 0.9383 - val_loss: 0.2329 -
val_sparse_categorical_accuracy: 0.9297
Epoch 10/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1976 -
sparse_categorical_accuracy: 0.9397 - val_loss: 0.2268 -
val_sparse_categorical_accuracy: 0.9336
Epoch 11/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2002 -
sparse_categorical_accuracy: 0.9404 - val_loss: 0.2393 -
val_sparse_categorical_accuracy: 0.9270
Epoch 12/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1921 -
sparse_categorical_accuracy: 0.9425 - val_loss: 0.2227 -
val_sparse_categorical_accuracy: 0.9348
Epoch 13/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1912 -
sparse_categorical_accuracy: 0.9434 - val_loss: 0.2363 -
val_sparse_categorical_accuracy: 0.9301
Epoch 14/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1877 -
sparse_categorical_accuracy: 0.9436 - val_loss: 0.2360 -
val_sparse_categorical_accuracy: 0.9301
Epoch 15/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1809 -
sparse_categorical_accuracy: 0.9456 - val_loss: 0.2246 -
val_sparse_categorical_accuracy: 0.9340
Epoch 16/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1778 -
sparse_categorical_accuracy: 0.9476 - val_loss: 0.2285 -
val_sparse_categorical_accuracy: 0.9299
Epoch 17/25
```

```
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1794 -
sparse_categorical_accuracy: 0.9466 - val_loss: 0.2377 -
val_sparse_categorical_accuracy: 0.9326
Epoch 18/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1733 -
sparse_categorical_accuracy: 0.9485 - val_loss: 0.2277 -
val_sparse_categorical_accuracy: 0.9336
Epoch 19/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1745 -
sparse_categorical_accuracy: 0.9458 - val_loss: 0.2260 -
val_sparse_categorical_accuracy: 0.9328
Epoch 20/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1666 -
sparse_categorical_accuracy: 0.9492 - val_loss: 0.2297 -
val_sparse_categorical_accuracy: 0.9330
Epoch 21/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1659 -
sparse_categorical_accuracy: 0.9502 - val_loss: 0.2339 -
val_sparse_categorical_accuracy: 0.9289
Epoch 22/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1602 -
sparse_categorical_accuracy: 0.9510 - val_loss: 0.2575 -
val_sparse_categorical_accuracy: 0.9262
Epoch 23/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1636 -
sparse_categorical_accuracy: 0.9493 - val_loss: 0.2258 -
val_sparse_categorical_accuracy: 0.9344
Epoch 24/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1615 -
sparse_categorical_accuracy: 0.9499 - val_loss: 0.2192 -
val_sparse_categorical_accuracy: 0.9334
Epoch 25/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1652 -
sparse_categorical_accuracy: 0.9510 - val_loss: 0.2201 -
val_sparse_categorical_accuracy: 0.9348
```

[64]: <tensorflow.python.keras.callbacks.History at 0x7f55cf3b9690>

[ ]:
```
y_test = model.predict(X_test)
```

[ ]:
```
y_test.shape
```

[ ]: (28000, 10)

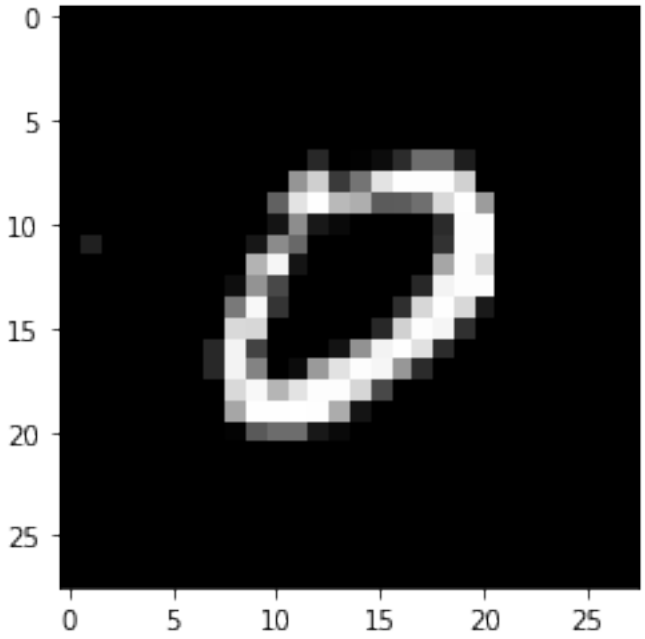
[ ]:
```
y_test[4]
```

[ ]:
```
array([-1042.2966 ,  -962.84344,   257.18417,  1459.2311 , -2862.0784 ,
        -455.13547, -1969.5304 , -2153.4077 ,  -273.3878 , -2122.5847 ],
```

```
      dtype=float32)
```

[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
plot_num = X_test[3].reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.show()
```



[ ]:
```
y_label = np.argmax(y_test, axis = 1)
y_label
```

[ ]:
```
array([2, 0, 8, …, 3, 9, 2])
```

[ ]:
```
y_label.shape
```

[ ]:
```
(28000,)
```

[ ]:
```
y_label[0:5]
```

[ ]:
```
array([2, 0, 8, 4, 3])
```

[ ]:
```
image_id = np.arange(len(y_test)) + 1
submission_data = np.c_[image_id, y_label]
submission_df = pd.DataFrame(data = submission_data, columns =␣
 ↪["ImageId","Label"])
submission_df['ImageId'] = submission_df['ImageId'].astype('int64')
```

```
submission_df['Label'] = submission_df['Label'].astype('int64')
submission_df
```

[ ]:

	ImageId	Label
0	1	2
1	2	0
2	3	8
3	4	4
4	5	3
…	…	…
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

[28000 rows x 2 columns]

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/two_by_ten_labels.csv", index =␣
↪False)
```

4 Neural Network - 2 hidden layers each with 20 neurons

[65]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Input(shape=(784,)),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    x = X_train, y = y_train,
    epochs=25,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.8409 -
sparse_categorical_accuracy: 0.7393 - val_loss: 0.2521 -
val_sparse_categorical_accuracy: 0.9246
Epoch 2/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2495 -
```

```
sparse_categorical_accuracy: 0.9254 - val_loss: 0.2278 -
val_sparse_categorical_accuracy: 0.9311
Epoch 3/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2058 -
sparse_categorical_accuracy: 0.9401 - val_loss: 0.2035 -
val_sparse_categorical_accuracy: 0.9406
Epoch 4/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1731 -
sparse_categorical_accuracy: 0.9479 - val_loss: 0.1881 -
val_sparse_categorical_accuracy: 0.9432
Epoch 5/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1586 -
sparse_categorical_accuracy: 0.9523 - val_loss: 0.1788 -
val_sparse_categorical_accuracy: 0.9480
Epoch 6/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1445 -
sparse_categorical_accuracy: 0.9555 - val_loss: 0.2104 -
val_sparse_categorical_accuracy: 0.9385
Epoch 7/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1330 -
sparse_categorical_accuracy: 0.9579 - val_loss: 0.1713 -
val_sparse_categorical_accuracy: 0.9482
Epoch 8/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1241 -
sparse_categorical_accuracy: 0.9617 - val_loss: 0.1662 -
val_sparse_categorical_accuracy: 0.9516
Epoch 9/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1152 -
sparse_categorical_accuracy: 0.9644 - val_loss: 0.1659 -
val_sparse_categorical_accuracy: 0.9531
Epoch 10/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1115 -
sparse_categorical_accuracy: 0.9654 - val_loss: 0.1598 -
val_sparse_categorical_accuracy: 0.9545
Epoch 11/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1014 -
sparse_categorical_accuracy: 0.9698 - val_loss: 0.1609 -
val_sparse_categorical_accuracy: 0.9545
Epoch 12/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0959 -
sparse_categorical_accuracy: 0.9699 - val_loss: 0.1614 -
val_sparse_categorical_accuracy: 0.9529
Epoch 13/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0913 -
sparse_categorical_accuracy: 0.9717 - val_loss: 0.1515 -
val_sparse_categorical_accuracy: 0.9576
Epoch 14/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0859 -
```

```
sparse_categorical_accuracy: 0.9731 - val_loss: 0.1648 -
val_sparse_categorical_accuracy: 0.9563
Epoch 15/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0828 -
sparse_categorical_accuracy: 0.9749 - val_loss: 0.1563 -
val_sparse_categorical_accuracy: 0.9559
Epoch 16/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0761 -
sparse_categorical_accuracy: 0.9764 - val_loss: 0.1668 -
val_sparse_categorical_accuracy: 0.9541
Epoch 17/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0767 -
sparse_categorical_accuracy: 0.9759 - val_loss: 0.1589 -
val_sparse_categorical_accuracy: 0.9563
Epoch 18/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0713 -
sparse_categorical_accuracy: 0.9776 - val_loss: 0.1694 -
val_sparse_categorical_accuracy: 0.9533
Epoch 19/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0672 -
sparse_categorical_accuracy: 0.9805 - val_loss: 0.1906 -
val_sparse_categorical_accuracy: 0.9512
Epoch 20/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0669 -
sparse_categorical_accuracy: 0.9797 - val_loss: 0.1669 -
val_sparse_categorical_accuracy: 0.9553
Epoch 21/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0634 -
sparse_categorical_accuracy: 0.9798 - val_loss: 0.1711 -
val_sparse_categorical_accuracy: 0.9584
Epoch 22/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0572 -
sparse_categorical_accuracy: 0.9822 - val_loss: 0.1700 -
val_sparse_categorical_accuracy: 0.9588
Epoch 23/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0598 -
sparse_categorical_accuracy: 0.9815 - val_loss: 0.1701 -
val_sparse_categorical_accuracy: 0.9580
Epoch 24/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0561 -
sparse_categorical_accuracy: 0.9809 - val_loss: 0.1776 -
val_sparse_categorical_accuracy: 0.9547
Epoch 25/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0536 -
sparse_categorical_accuracy: 0.9827 - val_loss: 0.1856 -
val_sparse_categorical_accuracy: 0.9555
```

[65]: <tensorflow.python.keras.callbacks.History at 0x7f55cc1ef210>

[ ]:
```
y_test = model.predict(X_test)
```

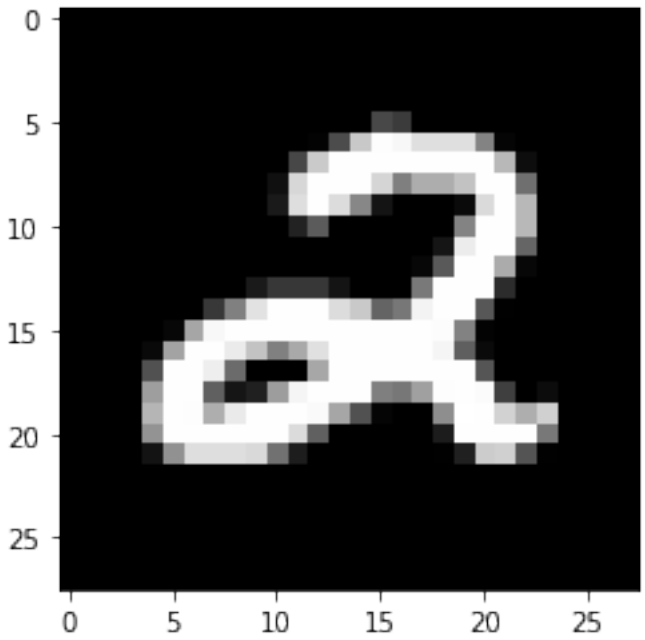
[ ]:
```
y_test.shape
```

[ ]: (28000, 10)

[ ]:
```
y_test[4]
```

[ ]:
```
array([-12.4616585 ,  -7.2582393 ,   4.0828543 ,   8.999565  ,
       -18.671888  ,  -3.4049277 , -11.948742  , -10.700039  ,
         0.56999344,  -8.263292  ], dtype=float32)
```

[ ]:
```
plot_num = X_test[27999].reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.show()
```



[ ]:
```
y_label = np.argmax(y_test, axis = 1)
```

[ ]:
```
y_label.shape
```

[ ]: (28000,)

[ ]:
```
y_label[0:5]
```

[ ]:
```
array([2, 0, 9, 2, 3])
```

[ ]:
```
image_id = np.arange(len(y_test)) + 1
submission_data = np.c_[image_id, y_label]
submission_df = pd.DataFrame(data = submission_data, columns =␣
↪["ImageId","Label"])
submission_df['ImageId'] = submission_df['ImageId'].astype('int64')
submission_df['Label'] = submission_df['Label'].astype('int64')
submission_df
```

[ ]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	2
4	5	3
…	…	…
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

[28000 rows x 2 columns]

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/two_by_twenty_labels.csv", index =␣
↪False)
```

4.1 Neural Network - 5 hidden layers each with 10 neurons

[66]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Input(shape=(784,)),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10,activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    x = X_train, y = y_train,
```

```
    epochs=25,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/25
1153/1153 [==============================] - 3s 2ms/step - loss: 1.4462 -
sparse_categorical_accuracy: 0.4693 - val_loss: 0.5809 -
val_sparse_categorical_accuracy: 0.8346
Epoch 2/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.5254 -
sparse_categorical_accuracy: 0.8475 - val_loss: 0.4403 -
val_sparse_categorical_accuracy: 0.8762
Epoch 3/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.4142 -
sparse_categorical_accuracy: 0.8834 - val_loss: 0.3713 -
val_sparse_categorical_accuracy: 0.8988
Epoch 4/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.3447 -
sparse_categorical_accuracy: 0.9033 - val_loss: 0.3358 -
val_sparse_categorical_accuracy: 0.9090
Epoch 5/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.3009 -
sparse_categorical_accuracy: 0.9125 - val_loss: 0.3301 -
val_sparse_categorical_accuracy: 0.9127
Epoch 6/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2840 -
sparse_categorical_accuracy: 0.9172 - val_loss: 0.2919 -
val_sparse_categorical_accuracy: 0.9219
Epoch 7/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2688 -
sparse_categorical_accuracy: 0.9228 - val_loss: 0.3020 -
val_sparse_categorical_accuracy: 0.9189
Epoch 8/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2564 -
sparse_categorical_accuracy: 0.9290 - val_loss: 0.2885 -
val_sparse_categorical_accuracy: 0.9219
Epoch 9/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2392 -
sparse_categorical_accuracy: 0.9312 - val_loss: 0.2838 -
val_sparse_categorical_accuracy: 0.9223
Epoch 10/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2328 -
sparse_categorical_accuracy: 0.9346 - val_loss: 0.2744 -
val_sparse_categorical_accuracy: 0.9266
Epoch 11/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2307 -
sparse_categorical_accuracy: 0.9340 - val_loss: 0.2757 -
val_sparse_categorical_accuracy: 0.9217
```

```
Epoch 12/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2190 -
sparse_categorical_accuracy: 0.9371 - val_loss: 0.2750 -
val_sparse_categorical_accuracy: 0.9256
Epoch 13/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2197 -
sparse_categorical_accuracy: 0.9373 - val_loss: 0.2752 -
val_sparse_categorical_accuracy: 0.9242
Epoch 14/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2098 -
sparse_categorical_accuracy: 0.9401 - val_loss: 0.2677 -
val_sparse_categorical_accuracy: 0.9254
Epoch 15/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2100 -
sparse_categorical_accuracy: 0.9407 - val_loss: 0.2586 -
val_sparse_categorical_accuracy: 0.9283
Epoch 16/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2021 -
sparse_categorical_accuracy: 0.9403 - val_loss: 0.2739 -
val_sparse_categorical_accuracy: 0.9246
Epoch 17/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1974 -
sparse_categorical_accuracy: 0.9416 - val_loss: 0.2608 -
val_sparse_categorical_accuracy: 0.9309
Epoch 18/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1981 -
sparse_categorical_accuracy: 0.9436 - val_loss: 0.2598 -
val_sparse_categorical_accuracy: 0.9279
Epoch 19/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1968 -
sparse_categorical_accuracy: 0.9438 - val_loss: 0.2659 -
val_sparse_categorical_accuracy: 0.9266
Epoch 20/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1926 -
sparse_categorical_accuracy: 0.9446 - val_loss: 0.2499 -
val_sparse_categorical_accuracy: 0.9316
Epoch 21/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1883 -
sparse_categorical_accuracy: 0.9451 - val_loss: 0.2530 -
val_sparse_categorical_accuracy: 0.9309
Epoch 22/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1831 -
sparse_categorical_accuracy: 0.9465 - val_loss: 0.2543 -
val_sparse_categorical_accuracy: 0.9307
Epoch 23/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1824 -
sparse_categorical_accuracy: 0.9464 - val_loss: 0.2619 -
val_sparse_categorical_accuracy: 0.9297
```

```
Epoch 24/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1786 -
sparse_categorical_accuracy: 0.9490 - val_loss: 0.2602 -
val_sparse_categorical_accuracy: 0.9303
Epoch 25/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1762 -
sparse_categorical_accuracy: 0.9483 - val_loss: 0.2560 -
val_sparse_categorical_accuracy: 0.9285
```

[66]:
```
<tensorflow.python.keras.callbacks.History at 0x7f55cc1e2650>
```

[ ]:
```
y_test = model.predict(X_test)
```

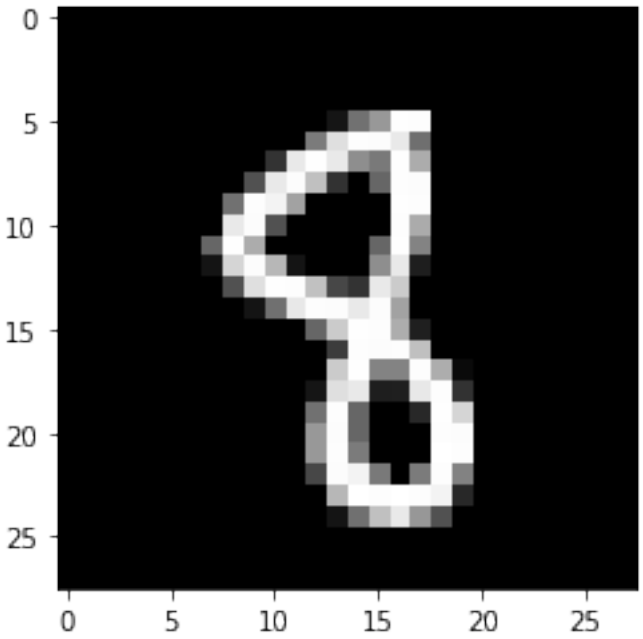
[ ]:
```
y_test.shape
```

[ ]:
```
(28000, 10)
```

[ ]:
```
y_test[10]
```

[ ]:
```
array([ -6.271258 ,  -9.083488 , -11.561647 ,   5.236908 , -26.266588 ,
        10.8661585,  -7.436074 , -18.832329 ,   1.7752887,  -1.6562455],
      dtype=float32)
```

[ ]:
```
plot_num = X_test[2000].reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.show()
```



[ ]:
```
y_label = np.argmax(y_test, axis = 1)
```

[ ]:
```
y_label.shape
```

[ ]:
```
(28000,)
```

[ ]:
```
y_label[0:5]
```

[ ]:
```
array([2, 0, 8, 7, 3])
```

[ ]:
```
image_id = np.arange(len(y_test)) + 1
submission_data = np.c_[image_id, y_label]
submission_df = pd.DataFrame(data = submission_data, columns =␣
↪["ImageId","Label"])
submission_df['ImageId'] = submission_df['ImageId'].astype('int64')
submission_df['Label'] = submission_df['Label'].astype('int64')
submission_df
```

[ ]:

	ImageId	Label
0	1	2
1	2	0
2	3	8
3	4	7
4	5	3
…	…	…
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

[28000 rows x 2 columns]

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/five_by_ten_labels.csv", index =␣
↪False)
```

5 Neural Network - 5 hidden layers each with 20 neurons

[67]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Input(shape=(784,)),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(20,activation='relu'),
  tf.keras.layers.Dense(10)
])
```

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(x = X_train,
          y = y_train,
          epochs=25,
          validation_data=(X_val, y_val))
```

```
Epoch 1/25
1153/1153 [==============================] - 3s 2ms/step - loss: 0.9818 -
sparse_categorical_accuracy: 0.6657 - val_loss: 0.2540 -
val_sparse_categorical_accuracy: 0.9268
Epoch 2/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.2477 -
sparse_categorical_accuracy: 0.9263 - val_loss: 0.2212 -
val_sparse_categorical_accuracy: 0.9377
Epoch 3/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1946 -
sparse_categorical_accuracy: 0.9417 - val_loss: 0.1943 -
val_sparse_categorical_accuracy: 0.9424
Epoch 4/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1598 -
sparse_categorical_accuracy: 0.9545 - val_loss: 0.1873 -
val_sparse_categorical_accuracy: 0.9424
Epoch 5/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1482 -
sparse_categorical_accuracy: 0.9561 - val_loss: 0.1749 -
val_sparse_categorical_accuracy: 0.9520
Epoch 6/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1310 -
sparse_categorical_accuracy: 0.9618 - val_loss: 0.1610 -
val_sparse_categorical_accuracy: 0.9520
Epoch 7/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1212 -
sparse_categorical_accuracy: 0.9632 - val_loss: 0.1715 -
val_sparse_categorical_accuracy: 0.9490
Epoch 8/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1101 -
sparse_categorical_accuracy: 0.9660 - val_loss: 0.1756 -
val_sparse_categorical_accuracy: 0.9459
Epoch 9/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.1022 -
sparse_categorical_accuracy: 0.9684 - val_loss: 0.1439 -
val_sparse_categorical_accuracy: 0.9566
Epoch 10/25
```

```
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0997 -
sparse_categorical_accuracy: 0.9691 - val_loss: 0.1553 -
val_sparse_categorical_accuracy: 0.9563
Epoch 11/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0939 -
sparse_categorical_accuracy: 0.9714 - val_loss: 0.1544 -
val_sparse_categorical_accuracy: 0.9564
Epoch 12/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0915 -
sparse_categorical_accuracy: 0.9725 - val_loss: 0.1449 -
val_sparse_categorical_accuracy: 0.9557
Epoch 13/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0832 -
sparse_categorical_accuracy: 0.9744 - val_loss: 0.1521 -
val_sparse_categorical_accuracy: 0.9551
Epoch 14/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0775 -
sparse_categorical_accuracy: 0.9756 - val_loss: 0.1675 -
val_sparse_categorical_accuracy: 0.9541
Epoch 15/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0759 -
sparse_categorical_accuracy: 0.9768 - val_loss: 0.1582 -
val_sparse_categorical_accuracy: 0.9543
Epoch 16/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0769 -
sparse_categorical_accuracy: 0.9760 - val_loss: 0.1624 -
val_sparse_categorical_accuracy: 0.9547
Epoch 17/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0649 -
sparse_categorical_accuracy: 0.9795 - val_loss: 0.1629 -
val_sparse_categorical_accuracy: 0.9578
Epoch 18/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0657 -
sparse_categorical_accuracy: 0.9797 - val_loss: 0.1570 -
val_sparse_categorical_accuracy: 0.9566
Epoch 19/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0627 -
sparse_categorical_accuracy: 0.9804 - val_loss: 0.1734 -
val_sparse_categorical_accuracy: 0.9551
Epoch 20/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0616 -
sparse_categorical_accuracy: 0.9802 - val_loss: 0.1577 -
val_sparse_categorical_accuracy: 0.9588
Epoch 21/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0574 -
sparse_categorical_accuracy: 0.9826 - val_loss: 0.1727 -
val_sparse_categorical_accuracy: 0.9531
Epoch 22/25
```

```
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0582 -
sparse_categorical_accuracy: 0.9825 - val_loss: 0.1808 -
val_sparse_categorical_accuracy: 0.9578
Epoch 23/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0581 -
sparse_categorical_accuracy: 0.9815 - val_loss: 0.1761 -
val_sparse_categorical_accuracy: 0.9570
Epoch 24/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0536 -
sparse_categorical_accuracy: 0.9833 - val_loss: 0.1722 -
val_sparse_categorical_accuracy: 0.9590
Epoch 25/25
1153/1153 [==============================] - 2s 2ms/step - loss: 0.0510 -
sparse_categorical_accuracy: 0.9839 - val_loss: 0.2209 -
val_sparse_categorical_accuracy: 0.9490
```

[67]:
```
<tensorflow.python.keras.callbacks.History at 0x7f55cf4ea8d0>
```

[ ]:
```
y_test = model.predict(X_test)
```

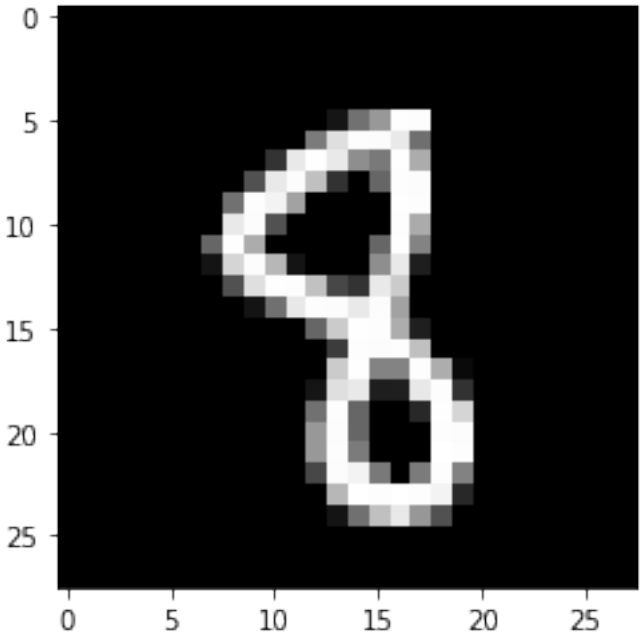
[ ]:
```
y_test.shape
```

[ ]:
```
(28000, 10)
```

[ ]:
```
y_test[10]
```

[ ]:
```
array([-20.537151  , -27.05243   ,  -3.876841  ,  -1.3491281 ,
       -14.759333  ,  16.857197  ,  -7.4521537 , -20.507378  ,
         0.2424821 ,   0.71157336], dtype=float32)
```

[ ]:
```
plot_num = X_test[2000].reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.show()
```



[ ]:
```
y_label = np.argmax(y_test, axis = 1)
```

[ ]:
```
y_label.shape
```

[ ]:
```
(28000,)
```

[ ]:
```
y_label[0:5]
```

[ ]:
```
array([2, 0, 9, 9, 3])
```

[ ]:
```
image_id = np.arange(len(y_test)) + 1
submission_data = np.c_[image_id, y_label]
submission_df = pd.DataFrame(data = submission_data, columns =␣
↪["ImageId","Label"])
submission_df['ImageId'] = submission_df['ImageId'].astype('int64')
submission_df['Label'] = submission_df['Label'].astype('int64')
submission_df
```

[ ]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3
…	…	…
27995	27996	9

```
27996    27997      7
27997    27998      3
27998    27999      9
27999    28000      2

[28000 rows x 2 columns]
```

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/five_by_twenty_labels.csv", index
↪= False)
```

[ ]:
```

```