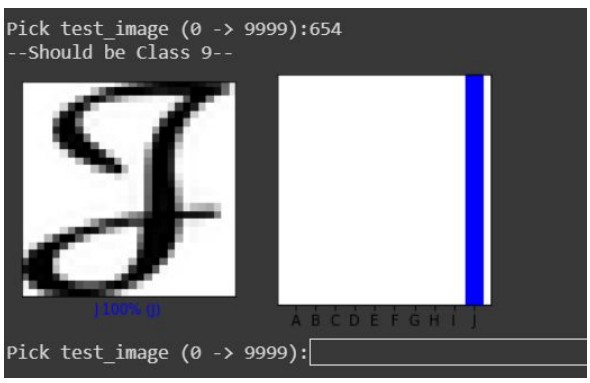
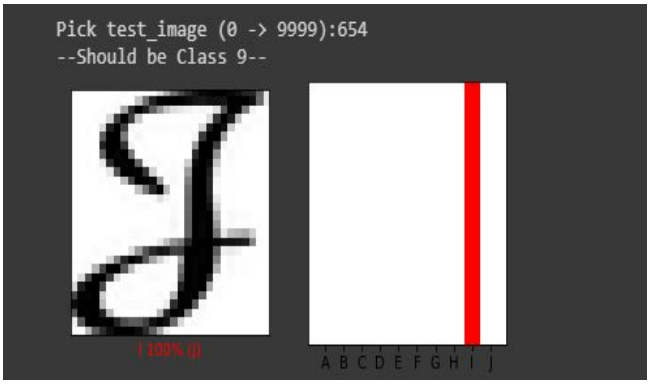
Part 1:

To run the code, simply open the MNIST.Ipynb on Google Colab then run the cell. I was able to get the training accuracy to 99.8% while the model accuracy is 98.1%. For the optimizer, I switched from SGD to Adam. According to Adam: A Method for Stochastic Optimization. Kingma et al, the method is “computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters”. In addition, I added many dense layers as a dense layer is akin to having layers of neurons which helps improvement of the ai. However too many neurons may cause overfit or underfit when it is the opposite. Due to trial and error I decided to use 150. 3 rgb, 1 black white for input__ (x, y, z). Since we are using many layers, sigmoid is not a good activation due to its nature of a hyperbolic tangent activation which cannot be used for many neuron networks. And due to trial and error, softmax and relu were the best activations to use. To export the file as h5, use the bottom code to export the file. When running the cell, click on the hyperlink in which it will lead you to a link where you will have to give permission for google colab. After that paste the security code inside the cell and the cell will produce a saved version of the h5 file inside your google drive.

Part 2:

To run the code: open notMNIST.ipynb as a notebook on Google Colab. Then use the upload cell to load the notMNIST.npz or you could use the google colab upload button on the left. After that run the main code then run the download cell. When running the cell, click on the hyperlink in which it will lead you to a link where you will have to give permission for google colab. After that paste the security code inside the cell and the cell will produce a saved version of the h5 file inside your google drive. Download the h5 file then open it on the predict_test or predict notebooks using the upload cell or upload button. After that run the check args() function and run the main func. When I was testing my model on predict test. I came across several images where the ai was not able to predict it. For instance image 654 is a J but the model was not able to predict it. So in order to improve it, I created more dense layers to help increase the model accuracy. The model accuracy was 93% and the newly made model is 94.1%. I tried using convolution1d to increase the performance, however I received errors about floating numbers when testing on predict.py In terms of prediction code, I used numerous sources on google such as: https://www.tensorflow.org/hub/tutorials/text_classification_with_tf_hub.





Part 3:

For the last part, I mainly copied this code from this website. https://www.tensorflow.org/tutorials/load_data/csv#top_of_page . There are sections where I added lines that were necessary such as importing the files and prints but mainly was borrowed from the website. To run the code, make sure you import the file as a notebook on google colab. Run each line procedurally then import the heart_train and heart_test files. Continue running each cell until the program is done. From the run, these were the test results which were mainly accurate.

```
Epoch 1/20
60000/60000 - 11s - loss: 0.5454 - acc: 0.8378
Epoch 2/20
60000/60000 - 8s - loss: 0.4048 - acc: 0.8770
Epoch 3/20
60000/60000 - 8s - loss: 0.3545 - acc: 0.8907
Epoch 4/20
60000/60000 - 8s - loss: 0.3213 - acc: 0.9017
Epoch 5/20
60000/60000 - 8s - loss: 0.2914 - acc: 0.9100
Epoch 6/20
60000/60000 - 8s - loss: 0.2670 - acc: 0.9165
Epoch 7/20
60000/60000 - 8s - loss: 0.2433 - acc: 0.9233
Epoch 8/20
60000/60000 - 8s - loss: 0.2216 - acc: 0.9304
Epoch 9/20
60000/60000 - 8s - loss: 0.2064 - acc: 0.9340
Epoch 10/20
60000/60000 - 8s - loss: 0.1889 - acc: 0.9395
Epoch 11/20
60000/60000 - 8s - loss: 0.1765 - acc: 0.9441
Epoch 12/20
60000/60000 - 8s - loss: 0.1629 - acc: 0.9481
Epoch 13/20
60000/60000 - 8s - loss: 0.1525 - acc: 0.9511
Epoch 14/20
60000/60000 - 8s - loss: 0.1472 - acc: 0.9544
Epoch 15/20
60000/60000 - 8s - loss: 0.1377 - acc: 0.9564
Epoch 16/20
60000/60000 - 8s - loss: 0.1285 - acc: 0.9601
Epoch 17/20
60000/60000 - 8s - loss: 0.1231 - acc: 0.9622
Epoch 18/20
60000/60000 - 8s - loss: 0.1143 - acc: 0.9645
Epoch 19/20
60000/60000 - 8s - loss: 0.1125 - acc: 0.9659
Epoch 20/20
60000/60000 - 8s - loss: 0.1043 - acc: 0.9666
```

The bottom are some of the results of the actuality and predictions. I played around the dense layers. Since we had a huge dataset it would make sense for me to use neurons of 512 to process the data.

```
Prediction of CGH: 5.60% Actual Outcome:  Negative
Prediction of CGH: 6.63% Actual Outcome:  Postive
Prediction of CGH: 40.28% Actual Outcome:  Postive
Prediction of CGH: 29.44% Actual Outcome:  Negative
Prediction of CGH: 22.93% Actual Outcome:  Negative
Prediction of CGH: 70.20% Actual Outcome:  Negative
Prediction of CGH: 69.15% Actual Outcome:  Negative
Prediction of CGH: 28.10% Actual Outcome:  Negative
Prediction of CGH: 19.77% Actual Outcome:  Postive
Prediction of CGH: 69.15% Actual Outcome:  Negative
Prediction of CGH: 44.87% Actual Outcome:  Negative
Prediction of CGH: 0.19% Actual Outcome:  Negative
Prediction of CGH: 76.22% Actual Outcome:  Postive
Prediction of CGH: 35.09% Actual Outcome:  Postive
Prediction of CGH: 31.66% Actual Outcome:  Negative
Prediction of CGH: 41.19% Actual Outcome:  Negative
Prediction of CGH: 35.58% Actual Outcome:  Negative
Prediction of CGH: 84.82% Actual Outcome:  Postive
Prediction of CGH: 66.67% Actual Outcome:  Negative
Prediction of CGH: 71.26% Actual Outcome:  Postive
Prediction of CGH: 90.20% Actual Outcome:  Negative
Prediction of CGH: 40.02% Actual Outcome:  Negative
Prediction of CGH: 23.80% Actual Outcome:  Negative
Prediction of CGH: 65.69% Actual Outcome:  Negative
Prediction of CGH: 92.57% Actual Outcome:  Negative
Prediction of CGH: 48.24% Actual Outcome:  Negative
Prediction of CGH: 40.80% Actual Outcome:  Negative
Prediction of CGH: 60.00% Actual Outcome:  Negative
Prediction of CGH: 53.12% Actual Outcome:  Negative
Prediction of CGH: 30.96% Actual Outcome:  Negative
```