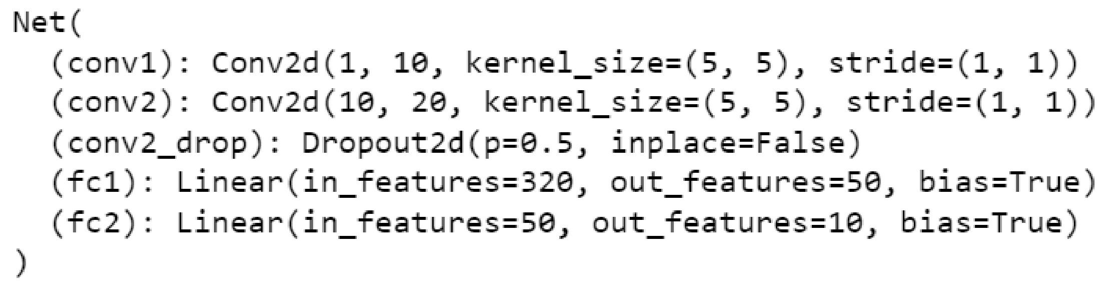
Samuel Ghebreyesus

Project Description

For this project, we were implementing deep learning networks for both visual and non-visual tasks. For example, in Problem 1, we used deep learning to recognize handwritten digits using the MNIST dataset, which consisted of 70,000 handwritten digits(60k of which were used for testing). Then, in Problem 2, we took the MNIST fashion dataset, which consists of 70k images of clothing items, and used deep learning to classify those images into one of ten classes. We tried modifying the parameters of the network along three dimensions(learning rate, batch size and dropout rate) to see how those modifications affected accuracy. Furthermore, in Problem 3, we applied the network used for MNIST to recognize three Greek letters: alpha, beta and gamma. Lastly, for Problem 4, we created two neural networks: one that consists of two fully connected linear layers and one that consists of two convolution layers and one linear layer and applied it to the Heart Disease Dataset from Project 3 trying to see if we can improve accuracy from the previous project.

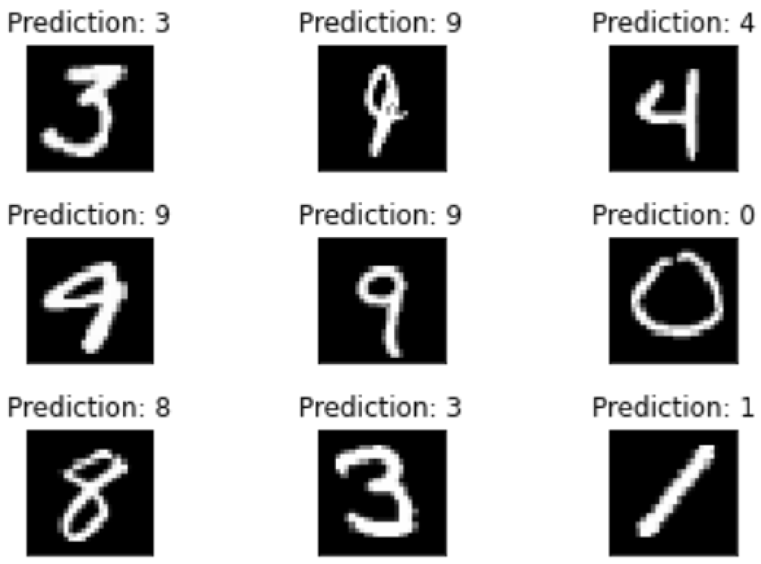
Problem 1:

The screenshot below shows a depiction of the network we used for the MNIST dataset. You can see that it consists of 2 convolutional layers with 5x5 filters, a dropout layer that has a dropout rate of 50% and 2 fully connected linear layers.

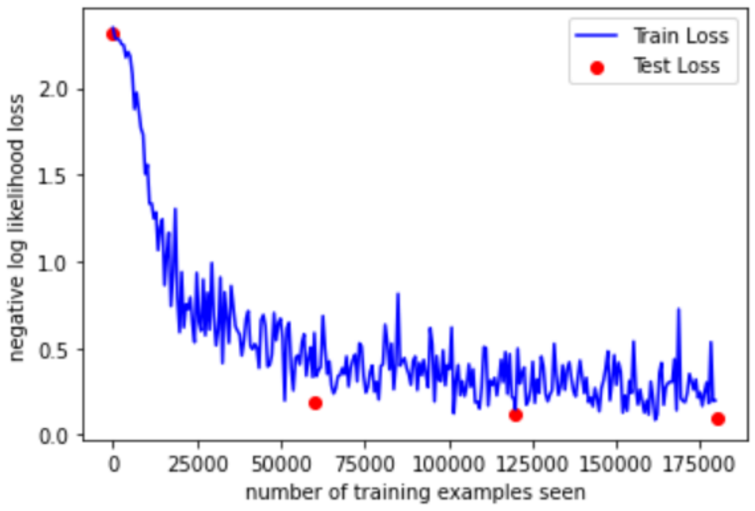


```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
```

The screenshot below shows a plot of nine example outputs and their labels as generated by the network. We can see that it got all 9 of these example outputs correct.



The following is a plot of the training and testing error. We can see that the more training examples seen, the negative log likelihood loss decreases generally.



Problem 2:

Part A: For this problem, the 3 dimensions along which I wanted to evaluate were learning rate, batch size and dropout rate. I chose to evaluate using the Fashion MNIST dataset. Each of the dimensions had 4 possible options for their values. For learning rate, the original network had a learning rate of 0.1. For my experimentation, I chose two values lower than 0.1: 0.001 and 0.0001 and two values higher: 0.15, 0.2. The four values for batch_size were 100, 250, 500 and 1000 (all higher than the original 64). The four values for dropout_rate were 0.4, 0.6, 0.7 and 0.8 (original network had default dropout rate of 0.5). I kept the same number of epochs at 3. I wanted to do something similar to a gridsearch to find the best parameters, so I used a RunBuilder and RunManager class to create a dataframe that shows for all 4*4*4*3 combinations of epochs, learning rates, batch sizes and dropout rates, the loss, accuracy and epoch duration.

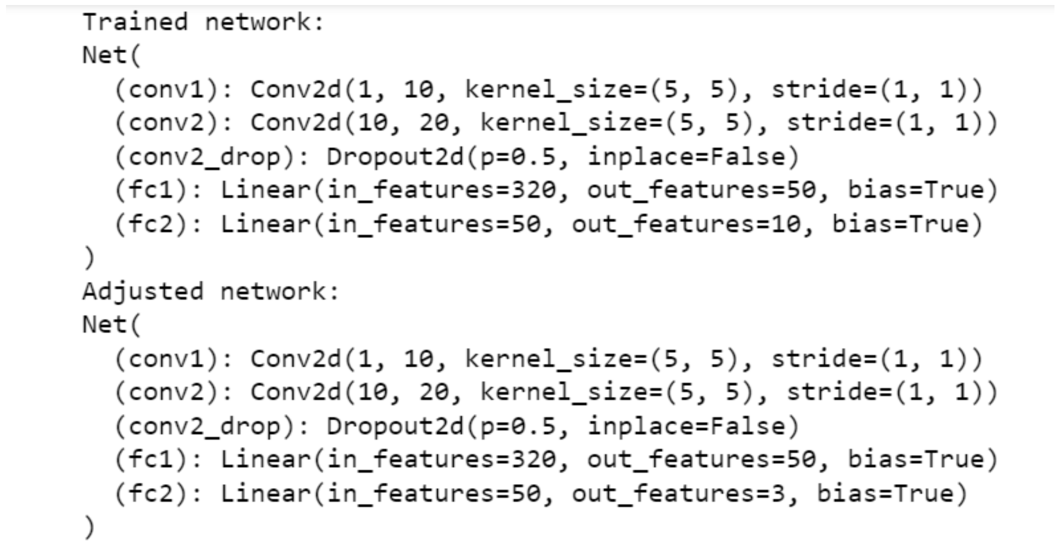
Part B: My prediction is that increasing the values of each of the three dimensions will lead to an increase in accuracy. I think this is the case because for both learning rate and dropout rate, increasing those values will lead to a reduction in overfitting. Additionally, for batch size, increasing it will lead to lesser noise in the gradients leading to better estimates. After evaluating, I saw that increasing the learning rate to 0.15 and 0.2 led to big increases in accuracy in comparison to 0.001 and 0.0001. However, increasing the batch size and dropout rate had minimal impacts on accuracy. Thus, the results only somewhat supported my hypotheses.

Part C: Below are the results of the evaluation, a data frame containing for each possible combination, the loss and accuracy values among other things.

	run	epoch	loss	accuracy	epoch duration	run duration	lr	batch_size	dropout_rate
0	1	1	2.304900	0.097633	25.604221	26.027066	0.0001	100	0.4
1	1	2	2.303068	0.102017	23.707237	49.832035	0.0001	100	0.4
2	1	3	2.301980	0.105133	22.424883	72.334706	0.0001	100	0.4
3	2	1	2.307537	0.099867	22.488711	22.664230	0.0001	100	0.6
4	2	2	2.306243	0.104183	23.085083	45.827103	0.0001	100	0.6
...	...	...	...	...	...	...	...	...	...
187	63	2	0.891003	0.664300	25.584711	48.856976	0.2000	1000	0.7
188	63	3	0.771774	0.713083	23.373298	72.311053	0.2000	1000	0.7
189	64	1	1.563306	0.413050	22.838142	24.016929	0.2000	1000	0.8
190	64	2	0.888818	0.662833	22.308603	46.412295	0.2000	1000	0.8
191	64	3	0.775733	0.706883	24.378559	70.882605	0.2000	1000	0.8

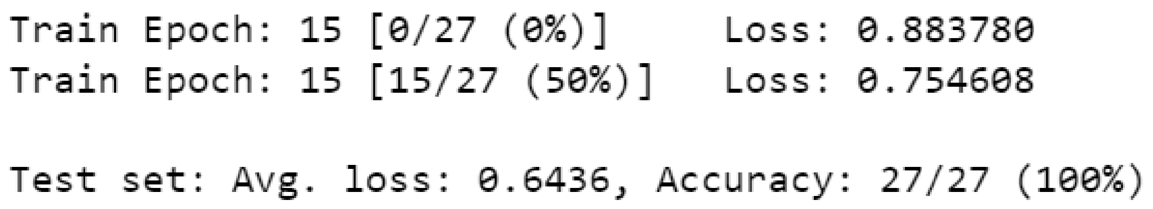
192 rows × 9 columns

Problem 3:
Below is a printout of the original MNIST network and the adjusted network. The main thing that changed is that in the second fully connected layer the output features went from 10 to 3, representing that we are trying to classify three Greek letters: alpha, beta and gamma.



```
Trained network:
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
Adjusted network:
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=3, bias=True)
)
```

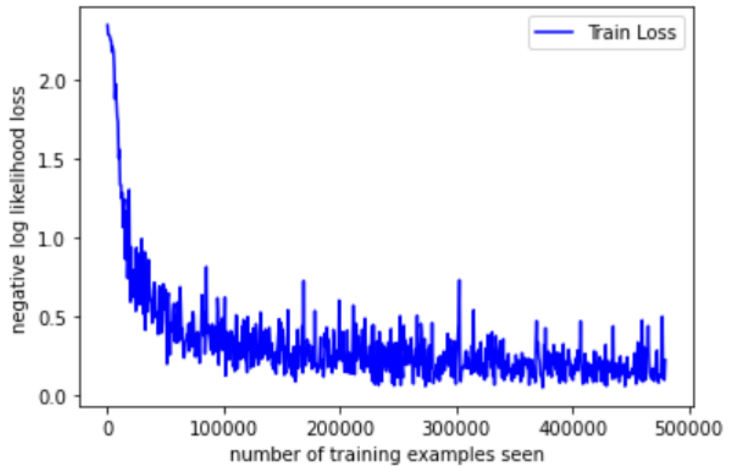
It took about 15 epochs to classify the 27 examples perfectly.



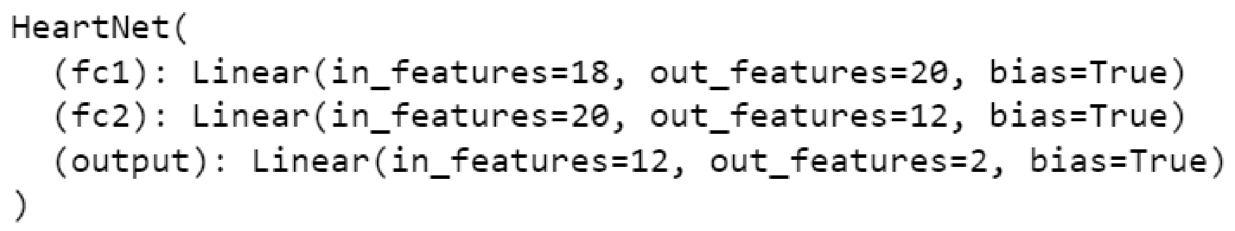
```
Train Epoch: 15 [0/27 (0%)]      Loss: 0.883780
Train Epoch: 15 [15/27 (50%)]    Loss: 0.754608

Test set: Avg. loss: 0.6436, Accuracy: 27/27 (100%)
```

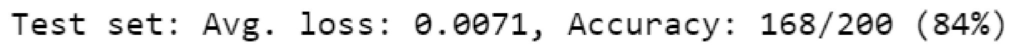
Below is a plot of the train error, showing how the negative log likelihood loss decreases as the number of training examples increases.



Problem 4:
For my first neural network for the Heart Disease Dataset, I used three linear layers: 2 that were fully connected and one that represented an output node. The first fully connected layer consisted of 18 input features and 20 output features while the second consisted of 20 input features and 12 output features. The third output layer consisted of 12 input features and 2 output features representing the binary classification (0/1 for heart disease). The performance of this network on the test set was an average loss of 0.0071 and an accuracy of 84%. I used 100 epochs, a learning rate of 0.1, cross entropy loss as the criterion, sgd optimizer among other parameters. In project 3, using Random Forest Classifier, my best results produced an accuracy of 88% so the neural network did not perform as well.

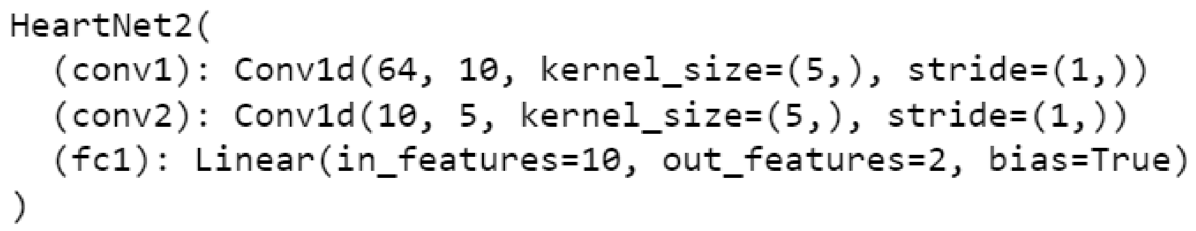


```
HeartNet(
  (fc1): Linear(in_features=18, out_features=20, bias=True)
  (fc2): Linear(in_features=20, out_features=12, bias=True)
  (output): Linear(in_features=12, out_features=2, bias=True)
)
```

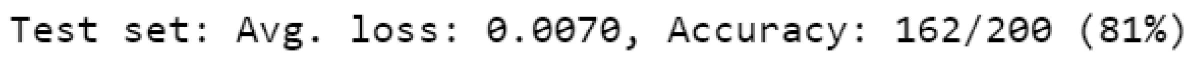


```
Test set: Avg. loss: 0.0071, Accuracy: 168/200 (84%)
```

For my second neural network, I used two convolutional layers with 5x5 filters and strides of 1. The first convolutional layer was 1D and had input channels 64 and output channels 10. The second convolutional layer was also 1D and had input channels 10 and output channels 5. After those two convolutional layers, a fully connected linear layer was implemented that had input features 10 and output features 2 for the binary 0/1 classification. I used the same hyperparameters as the first neural network described above (100 epochs, 0.1 learning rate, etc.) and my performance saw an average loss on the test set of 0.007 and accuracy of 81%, which was worse than the 88% I got in Project 3 for the random forest classifier with hyperparameter tuning.

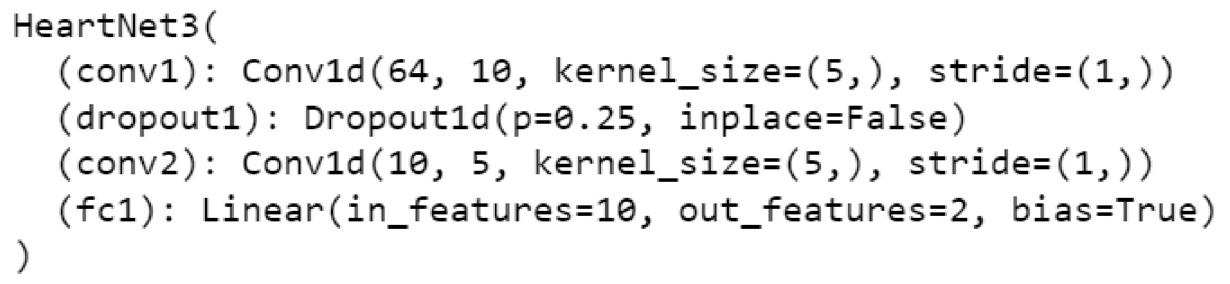


```
HeartNet2(
  (conv1): Conv1d(64, 10, kernel_size=(5,), stride=(1,))
  (conv2): Conv1d(10, 5, kernel_size=(5,), stride=(1,))
  (fc1): Linear(in_features=10, out_features=2, bias=True)
)
```

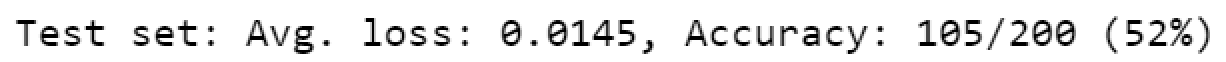


```
Test set: Avg. loss: 0.0070, Accuracy: 162/200 (81%)
```

Extension:
For my extension, I tested two additional architectures to my Heart Disease Dataset. In the third architecture, I added a dropout layer with a dropout rate of 25% after the first convolutional layer. In the fourth architecture, I added two max pooling layers with size of 2 to each of the convolutional layers. Neither of these additional architectures improved the performance of the classifier in terms of accuracy (in fact, accuracy decreased significantly down to 52% on the test set).



```
HeartNet3(
  (conv1): Conv1d(64, 10, kernel_size=(5,), stride=(1,))
  (dropout1): Dropout1d(p=0.25, inplace=False)
  (conv2): Conv1d(10, 5, kernel_size=(5,), stride=(1,))
  (fc1): Linear(in_features=10, out_features=2, bias=True)
)
```



```
Test set: Avg. loss: 0.0145, Accuracy: 105/200 (52%)
```

Reflection:
Overall, through doing this project, I learned how to set up neural networks to do various different types of tasks. I also learned how to create train/test functions to run these networks and how to plot the train/test errors and how to interpret the results. Furthermore, I learned how to do hyperparameter tuning to experiment with different variations in the architecture in order to get the most optimal results. I also learned how to use networks I already created and apply them to new problems (transfer learning- greek letters example). While the first 3 problems involved image data, the last problem taught me how to use neural networks when the data is numeric data in a dataframe, which presented a lot of difficult challenges that I eventually overcame to successfully run a network.

Acknowledgements
Neha Singh office hours, Professor Maxwell’s lecture notes, towardsdatascience.com, machinelearningmastery.com, pytorch documentation