Introduction to Deep Learning
Assignment 1: Classification

Sunayana Gupta
1222389090
sgupt279@asu.edu

In this assignment, we had to vary the number of dense hidden layers and the number of neurons per hidden layer to see and understand why changes are occurring when we changed these numbers in the TensorFlow neural network given to us.

So first I created a logic that iterated with 1 hidden layer and neurons in that hidden layer varying from 2 to 16384 in multiple of 2 that is 2, 4, 8, etc in below fashion:

```
neurons = 1
for n in range(14):
    neurons = neurons*2
    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(neurons, activation='relu'),
        tf.keras.layers.Dense(10)
    ])

    model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])

    model.fit(train_images, train_labels, epochs=10)

    test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

    print('\nTest accuracy:', test_acc, 'neurons ' , neurons)
    df = df.append({'neurons' : neurons, 'test_accuracy' : test_acc},
        ignore_index = True)
df.to_csv('layer1.csv', encoding='utf-8', index=False)
```

Got results like:

neurons	test_accuracy
4096	0.98229998
16384	0.98199999
512	0.98159999
1024	0.98110002
2048	0.97970003
8192	0.9795
256	0.97939998
128	0.97649998
64	0.97530001
32	0.96810001
16	0.95300001
8	0.9271
4	0.87080002
2	0.6347

(The images show a part of the results and code, full code and results csv are uploaded to Github and link is in the bottom of the report)

In this, we could see that the accuracy continuously increased till 98.11% the number of neurons reached 1024. After that, it dropped to 2048 neurons and subsequently increased and decreased in the layers further. Also, there was a rapid increase in the accuracy from 2 neurons in the hidden layer to 512 neurons. For 512 neurons we reached an accuracy of 98.15. Post that there was not much increase in the accuracy till 16384 neurons. This is because the model was training and getting underfitted till a certain number that is 512 in our case has reached. Once a certain accuracy is reached when we continued to increase the number of neurons the accuracy didn't change much. This is where the training accuracy was still increasing but the test accuracy was stagnant hence, we can say that the model has reached overfitting post 1024 neurons in the one hidden layer.

In the second experiment, I changed the number of hidden layers to 2 and tried to observe the model behavior. For this analysis, I kept two for loops which checked for all possible permutations and combinations for the hidden layers for a multiple of 2 till 4096 for both the hidden layers like below:

Got results like:

A	B	C
neu	neurons	test_accuracy
4096	2048	0.98100002
512	64	0.98060002
2048	256	0.97829999
256	512	0.97119999
1024	128	0.97109997
2048	512	0.9709
1024	4096	0.97030002
2048	1024	0.97030002
512	4096	0.97009999
512	512	0.97000003
2048	64	0.9698
4096	512	0.96960002
256	64	0.96920002
2048	2048	0.9691
4096	256	0.9691
512	2048	0.96899998
1024	2048	0.96890003

Even after increasing the number of hidden layers to 2 the max accuracy, I got was 98.1% with 4096 neurons in 1st layer and 2048 neurons in the 2nd layer. From this analysis, I observed that if the neurons in 2nd layer is very less like 2, 4 or 8 the accuracy doesn't increase much even if the neurons in 1st layer are increased to a very large number like 4096. This is because such a model could not do the feature extractions properly as the neurons shrank from a very large number to a very small number and with less feature extraction the model resulted in poor accuracy.

But if we see the opposite case where 1st layer has a very small number of neurons like 4, 8, and 16 and 2nd layer has a large number of neurons like 512 or 1048 then it results in a fairly good model with an accuracy of around 90%. This is because here the feature extraction is possible as the neurons are increasing in the direction where we want the result. This implies that the 2nd layer has to have a sufficient number of neurons in the 2nd layer for good accuracy.

From further results where neurons were (1024, 512) I got an accuracy of 96.53% which is more than (1024, 1024) 96.17%. Hence this is where the overfitting starts to happen and (1024, 512) were just enough for the classification as compared to (1024,1024). We got the maximum accuracy of 98.1% with (4096, 2048) neurons in the case of 2 hidden layers. So according to this if we increase the number of hidden layers and decrease the neurons per layer then it would result in almost the same accuracy.

In the 3rd experiment, I kept 3 hidden layers and iterated the model with all combinations of number of neurons in them in multiples of 2 to 8192 like below:

```
ne=4096
for b in range(11):
    ne = ne*2
    neu=1
    for a in range(11):
        neu = neu*2
        neurons = 1
        for n in range(11):
          neurons = neurons*2
          model = tf.keras.Sequential([
              tf.keras.layers.Flatten(input_shape=(28, 28)),
              tf.keras.layers.Dense(ne, activation='relu'),
              tf.keras.layers.Dense(neu, activation='relu'),
              tf.keras.layers.Dense(neurons, activation='relu'),
              tf.keras.layers.Dense(10)
          ])

          model.compile(optimizer='adam',
                        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                        metrics=['accuracy'])

          model.fit(train_images, train_labels, epochs=10, verbose=0)

          test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=0)

          print('\nTest accuracy:', test_acc, 'ne' , ne, 'neu', neu , 'neurons ' , neurons)
          df2 = df2[0:0]
          df2 = df2.append({'ne': ne, 'neu': neu, 'neurons' : neurons, 'test_accuracy' : test_acc},
              ignore_index = True)
          df2.to_csv('layer3two.csv', mode='a', index=False, encoding='utf-8', header =False)
```

Got results like:

Neurons1st layer	Neurons 2nd layer	Neurons 3rd layer	Accuracy
2	4	2	0.1135
2	128	4	0.1135
2	512	2	0.1135
2	1024	2	0.1135
4	256	2	0.1135
4	2048	2	0.1135
16	128	2	0.1135
16	2048	4	0.1135
32	2	2	0.1135
128	2	2	0.1135
128	2048	4	0.1135
512	256	2	0.1135
512	2048	2	0.1135
1024	2	8	0.1135
1024	16	2	0.1135
1024	32	2	0.1135
2048	128	2	0.1135
2048	256	2	0.1135
2048	1024	2	0.1135

In 3 hidden layer composition, the accuracy is pretty low when the last layer has very few neurons as the feature extraction isn't proper, but I got good accuracy where the number of neurons was large like (2048 or 1024) in the first layer little less like (512, 256 or 128) in the 2nd layer and (128, 64, 32) in the last layer. Combinations like these resulted in an accuracy of around 98.2%. And got maximum accuracy of 98.47% with 16384 neurons in 1st layer, 64 in 2nd layer, and 258 in the last layer. The second last was (2048, 512, 128) 98.43%. Which is quite similar to (1024, 32, 64) 98.2%. Therefore after (1024, 32, 64) there was not much increase as the model shifted towards overfitting post this combination if we increase the number of neurons.

Overall, after completing the whole experiment, I would say the computation increases when we increase the number of hidden layers but it helps in better feature extraction and hence better classification accuracy. But if we keep on increasing the number of neurons in any of the layers then after a point overfitting starts and the point where the accuracy starts becoming stagnant is where we should stop further training and save the model as that is possibly the best that model can do with the given data. The other observation was if we keep a high number of neurons in initial layers and decrease them in subsequent layers by a multiple of 2 then the resulting model has a good accuracy compared to the one where the initial layer has very few neurons and subsequent layers have more. This is because such a combination helps in good feature extraction.

Code:

I have uploaded the code in the below public repository. Ran this in google colab.
https://github.com/sunayana17/CSE598IDL/blob/master/classification/classification1.ipynb

Results:
I have uploaded the results in github repo:
Results for 1 hidden layer:
https://github.com/sunayana17/CSE598IDL/blob/master/classification/1HiddenLayer.csv

Results for 2 hidden layer:
https://github.com/sunayana17/CSE598IDL/blob/master/classification/2HiddenLayers.csv

Results for 3 hidden layer:
https://github.com/sunayana17/CSE598IDL/blob/master/classification/3hiddenLayers.csv