CAP5415
Programming Assignment 2

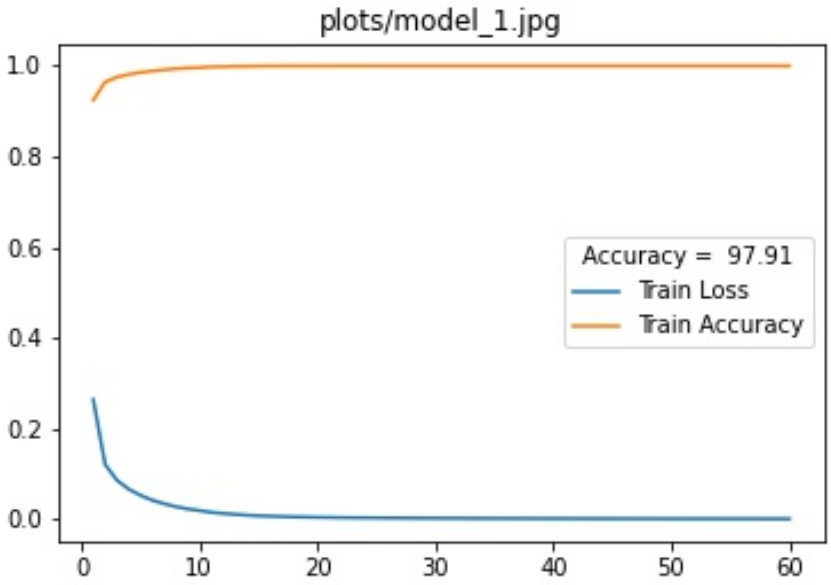
Name: Prudvi Kamtam
UCF ID: 5498416

Code: https://github.com/greysou1/ConvolutionalNN

Model 1

A fully connected (FC) hidden layer (with 100 neurons)

```
criterion = CrossEntropy
optimizer = Stochastic Gradient Descent
learning_rate = 0.1
batch_size = 10
num_epochs = 60
```



Training accuracy: 100%
Testing accuracy: 97.91%

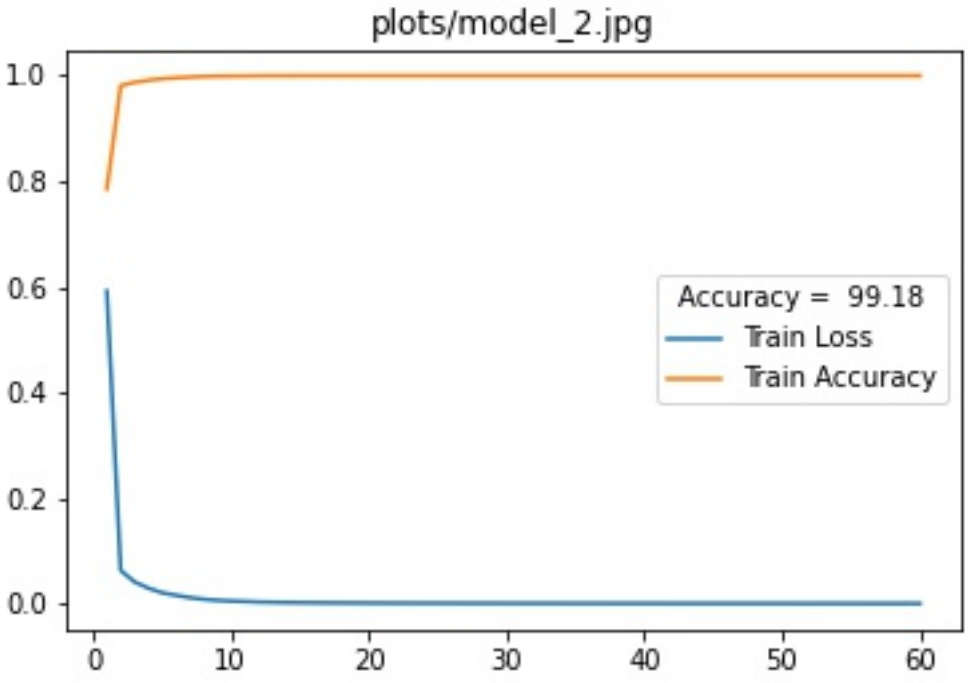
- This model is the simplest of all the models as it has only the fully connected layers in it. Hence, it runs the fastest.
- The loss starts high at first but quickly declines for the first couple of epoch.
- Over the next 10-15 epochs the loss slowly reduces.
- After this point the loss is at it's lowest and completely flattens for most of the epochs.
- The Accuracy starts off pretty good at around 90% and climbs to 98-99 just after a few epochs and then completely flattens for the rest of the epochs.

Model 2

Two convolutional layers + A fully connected (FC) hidden layer (with 100 neurons)

```
criterion = CrossEntropy
optimizer = Stochastic Gradient Descent
learning_rate = 0.1
batch_size = 10
num_epochs = 60,

num_of_filters = 40
pool_kernel_size = 2x2
conv_kernel_size = 5x5
Stride = 1
```



Training accuracy: 100%
Testing accuracy: 99.18%

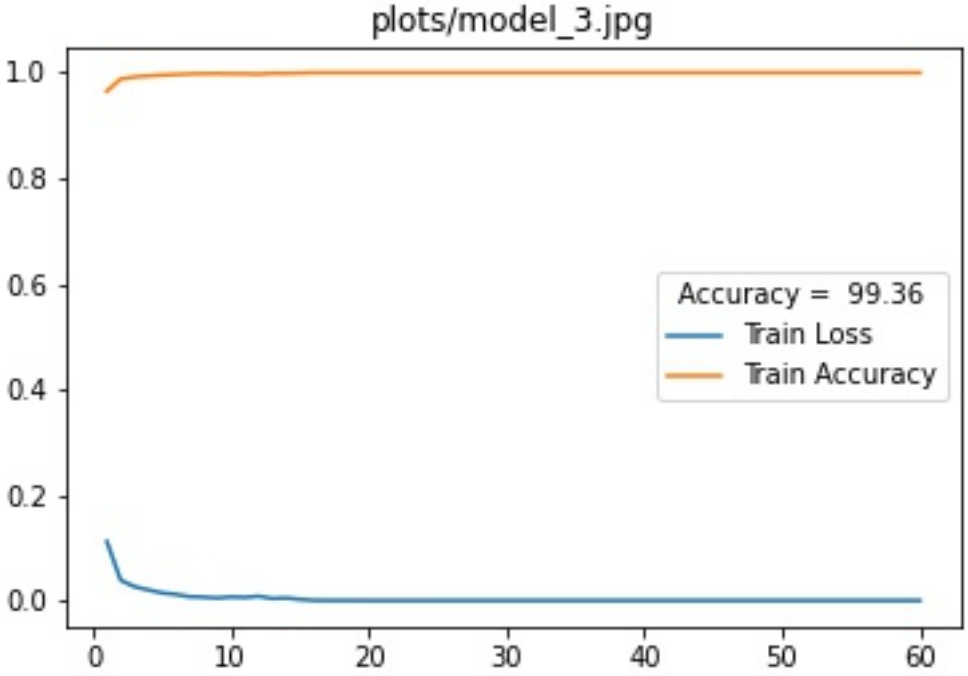
- This model adds 2 convolutional layers to the existing fully connected model 1 thereby adding more parameters that needs to be trained, hence this model take a bit longer to complete 60 epochs than model 1.
- The loss starts pretty high at first but just after 2-3 epoch the loss sees a steep decline.
- This is because the kernels in the convolutional layers are randomly initialised and quickly gets learned after 2-3 epochs but since the learning rate is comparatively high it quickly converges.
- The same can be seen in the accuracy, starts off low around 80% but then quickly climbs to almost 98-99 and then flattens at 100% (train accuracy)
- This model has a better test accuracy than the model 1 because of the convolutional layer.

Model 3

Replace Sigmoid with ReLU in model 2, and update the learning rate (=0.03).

```
criterion = CrossEntropy
optimizer = Stochastic Gradient Descent
learning_rate = 0.03
batch_size = 10
num_epochs = 60

num_of_filters = 40
pool_kernel_size = 2x2
conv_kernel_size = 5x5
Stride = 1
```



Training accuracy: 100%
Testing accuracy: 99.36%

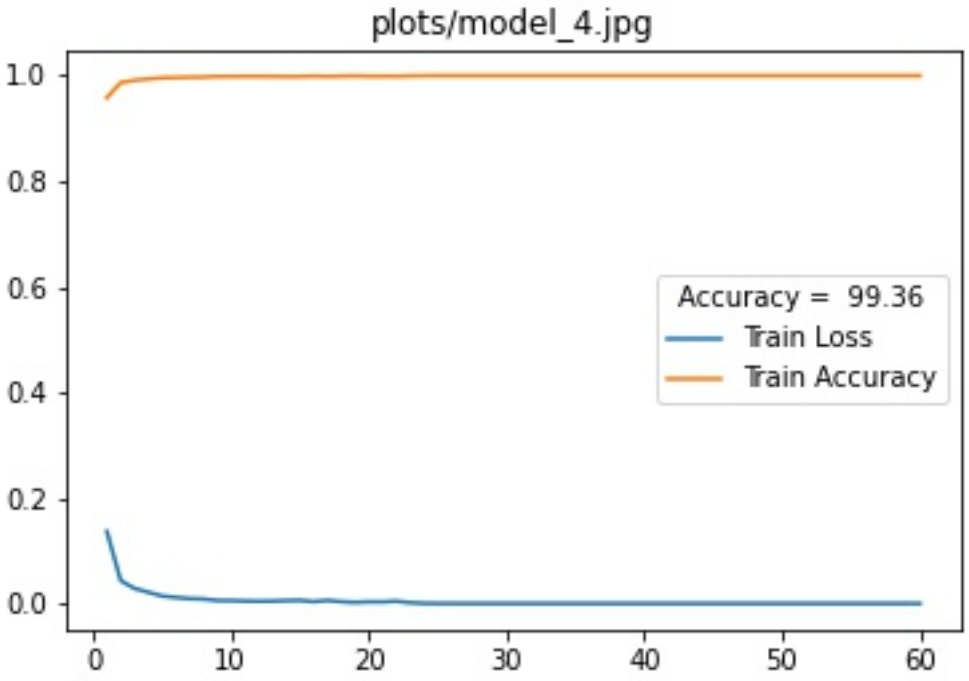
- This model is exactly the same as the previous model with the change in activation function and learning rate.
- There isn't significant different between model 2 and model 3's accuracy except, model 3 finished training a tad bit earlier than model 2 (~1 min difference). This could be due to the fact that ReLu is faster to compute than Sigmoid.
- The loss starts off pretty low compared to the model 2 but it takes longer to reach the lowest loss compared to the model 2 training because of the comparatively low learning rate.
- The accuracy starts off pretty high and easily flattens at 100%
- The model's slightly better accuracy can be attributed to ReLu activation function.

Model 4

Add another fully connected (FC) layer (with 100 neurons) to model 3.

```
criterion = CrossEntropy
optimizer = Stochastic Gradient Descent
learning_rate = 0.03
batch_size = 10
num_epochs = 60

num_of_filters = 40
pool_kernel_size = 2x2
conv_kernel_size = 5x5
Stride = 1
```



Training accuracy: 100%
Testing accuracy: 99.36%

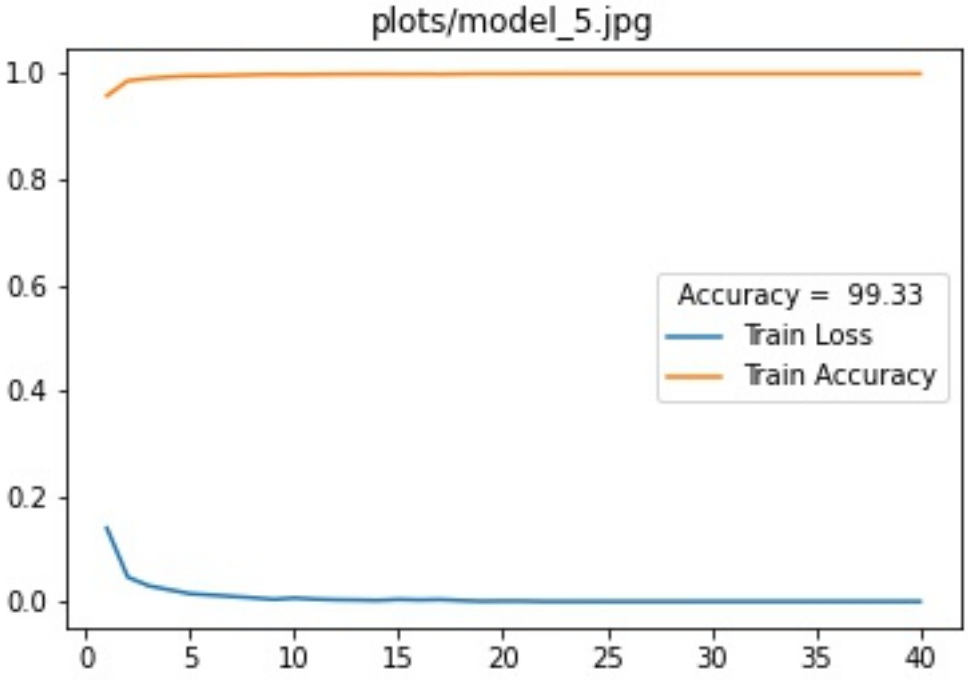
- The only change in this model is the addition of a new hidden layer.
- We can see even after adding a new layer to the existing model the accuracy is the exact same as the previous layer.
- The loss and accuracy has the same pattern as model 3
- The accuracy starts off pretty high and easily flattens at 100%
- This model took a tad bit longer to train than model 3 (20 seconds).

Model 5

Change the neurons numbers in FC layers into 1000. For Regularization, use Dropout (with a rate of 0.5). Train the whole system using 40 epochs.

```
criterion = CrossEntropy
optimizer = Stochastic Gradient Descent
learning_rate = 0.03
batch_size = 10
num_epochs = 40

num_of_filters = 40
pool_kernel_size = 2x2
conv_kernel_size = 5x5
Stride = 1
Dropout_rate = 0.5
```



Training accuracy: 100%
Testing accuracy: 99.33%

- This model's main difference is that the neurons in the layers are changed to 1000.
- We can see that even after changing the number of nodes in the FC layers in the existing model the accuracy is the almost the same as the previous layer.
- The loss and accuracy has the same pattern as model 3 and model 4
- The accuracy starts off pretty high and easily flattens at 100%
- Dropout is added to reduce the effects of overfitting but the accuracy (which is already high) is slightly lower than model 4. This could be because the dropout has been added right before the last layer so the effects of dropout couldn't be realised to the fullest
- Or it could just be for the fact that the model is already showing pretty good accuracy (99.36% in the last model) and that 0.03% difference could be from any other parameter or just at random.