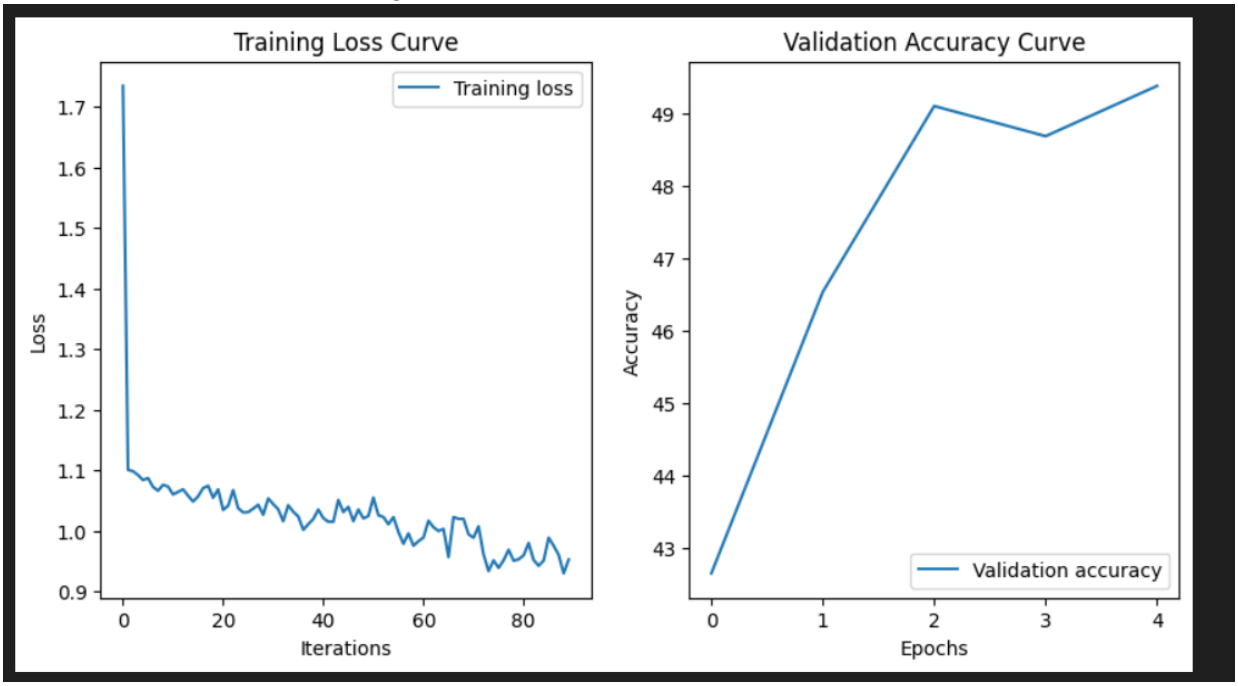
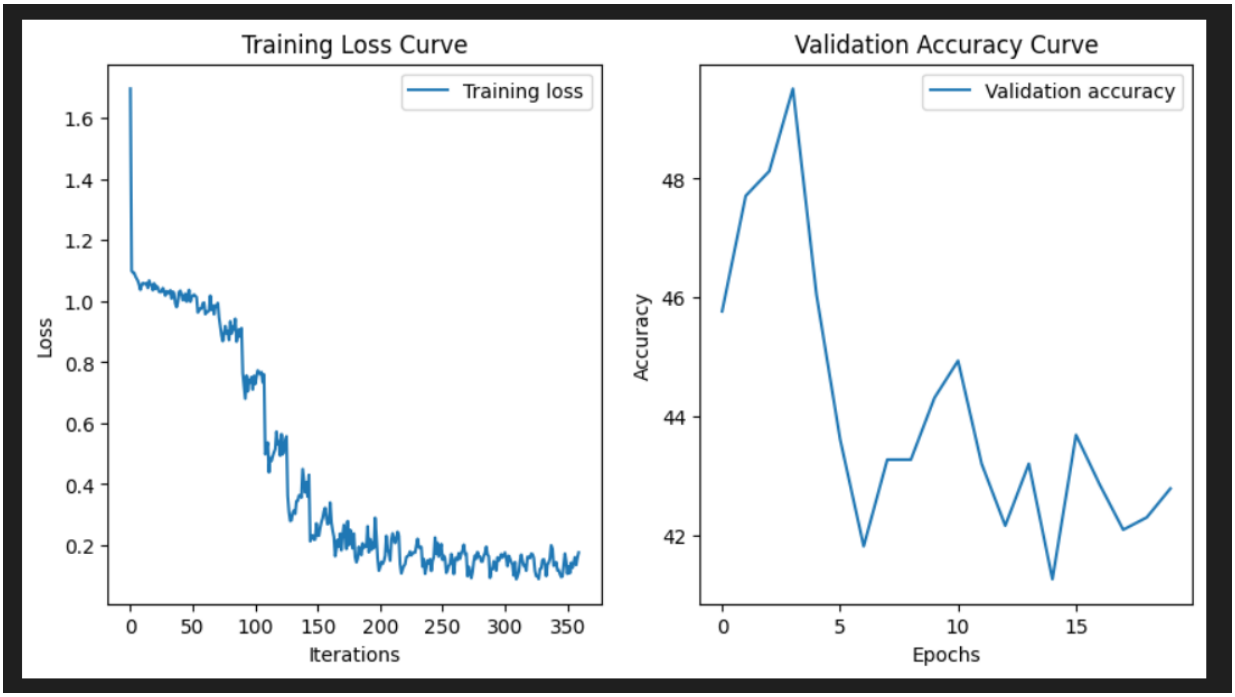
I'm training the model on an external GPU since it was taking too long to do on Google Colab. I will save the best model once I find it so I can just upload it and I will calculate other performance metrics for it as well. I'm trying to get validation above 70%.

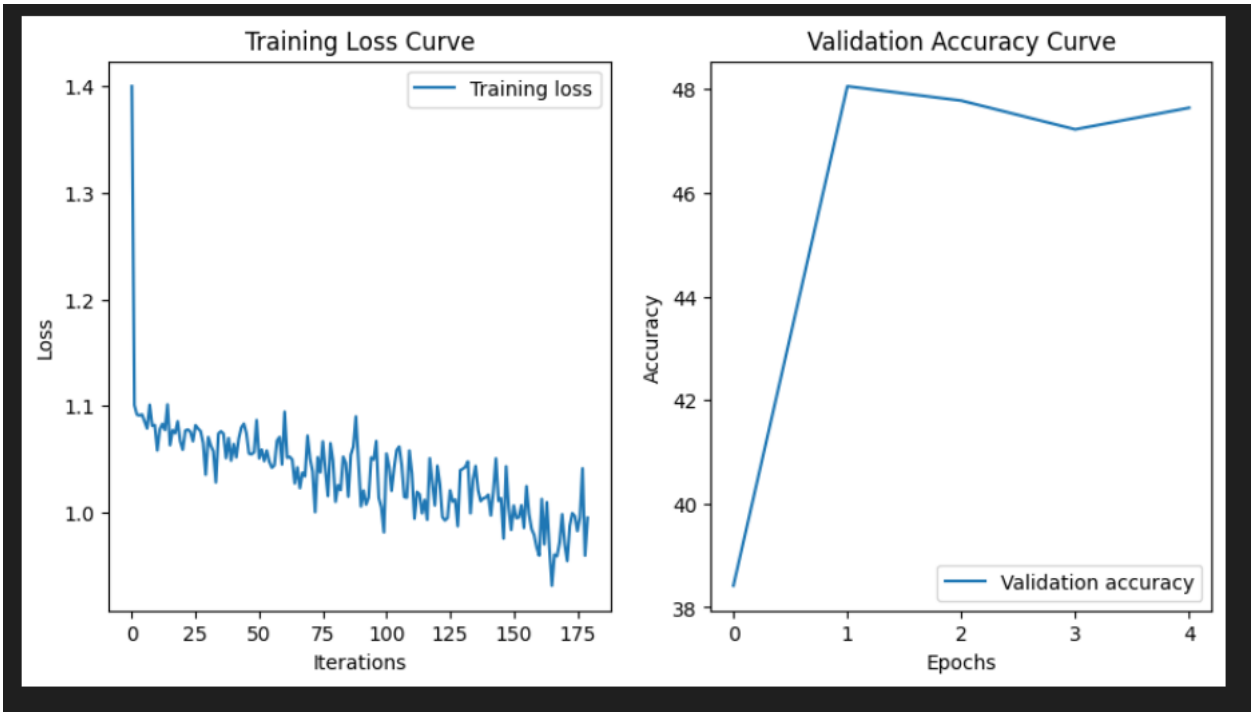
Batch size: 64 Learning rate: 0.001 Epochs: 5



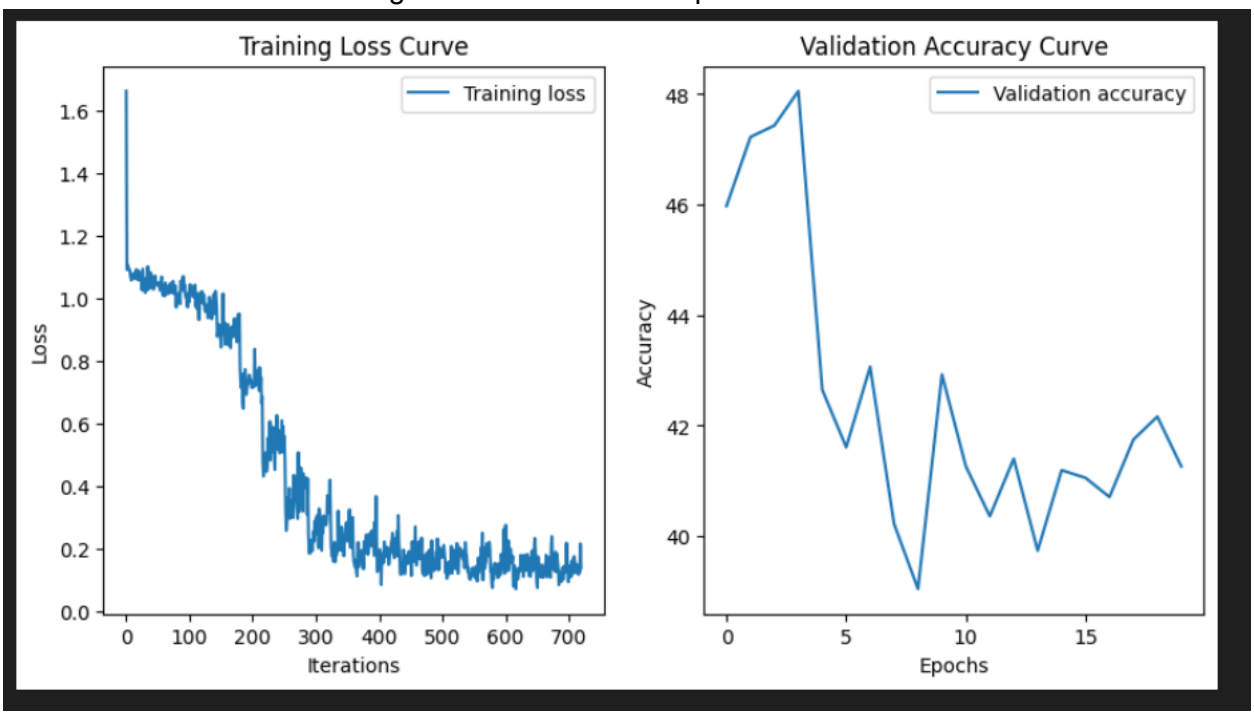
Batch size: 64 Learning rate: 0.001 Epochs: 20



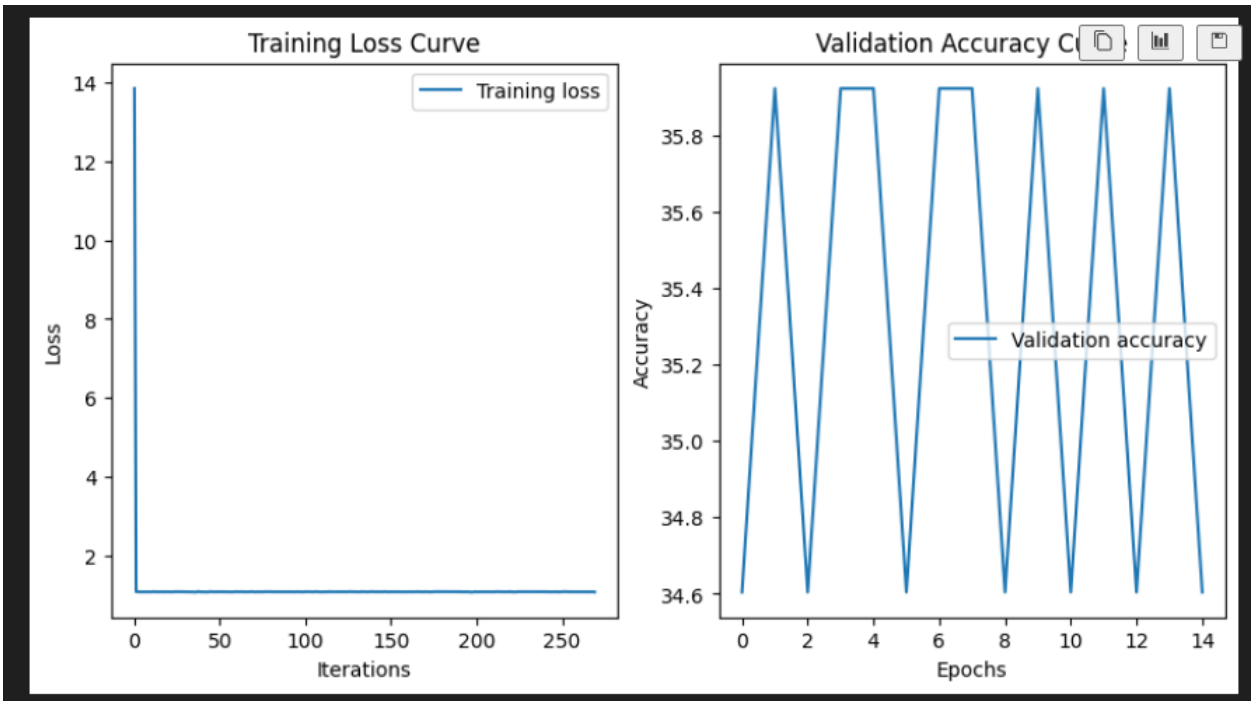
Batch size: 32 Learning rate: 0.001 Epochs: 5



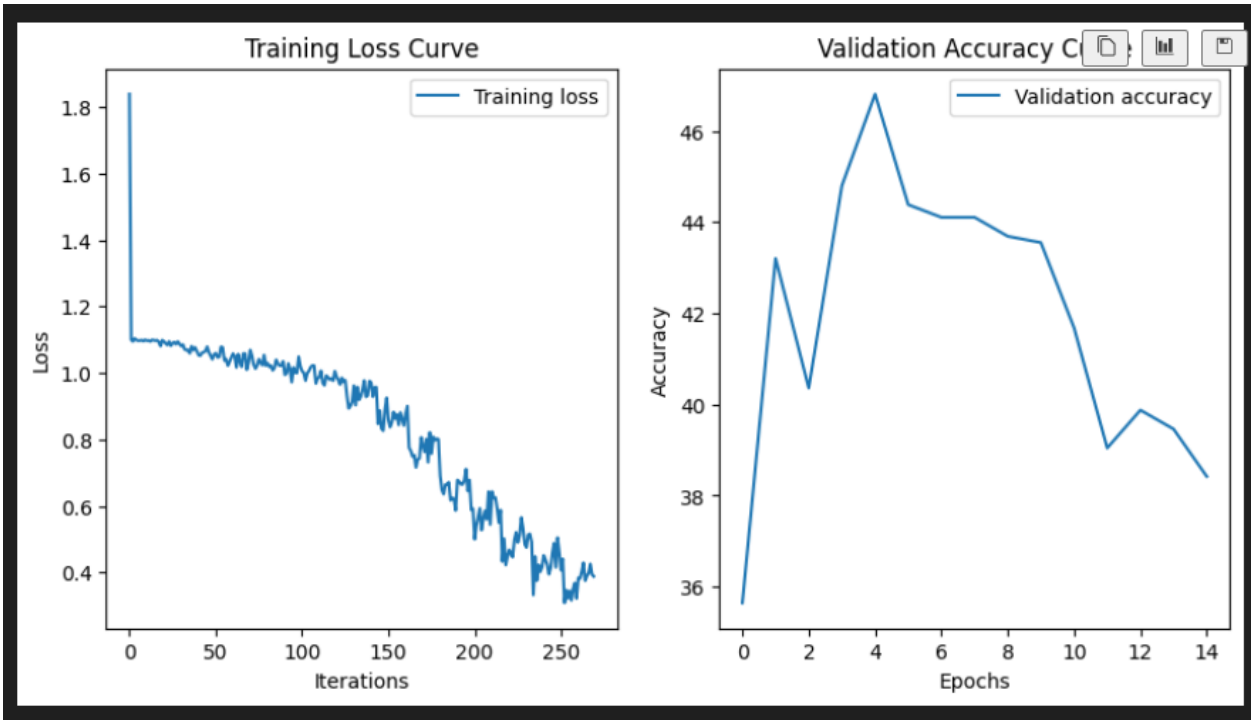
Batch size: 32 Learning rate: 0.001 Epochs: 20



Batch size: 64 Learning rate: 0.01 Epochs: 15



Batch size: 64 Learning rate: 0.002 Epochs: 15



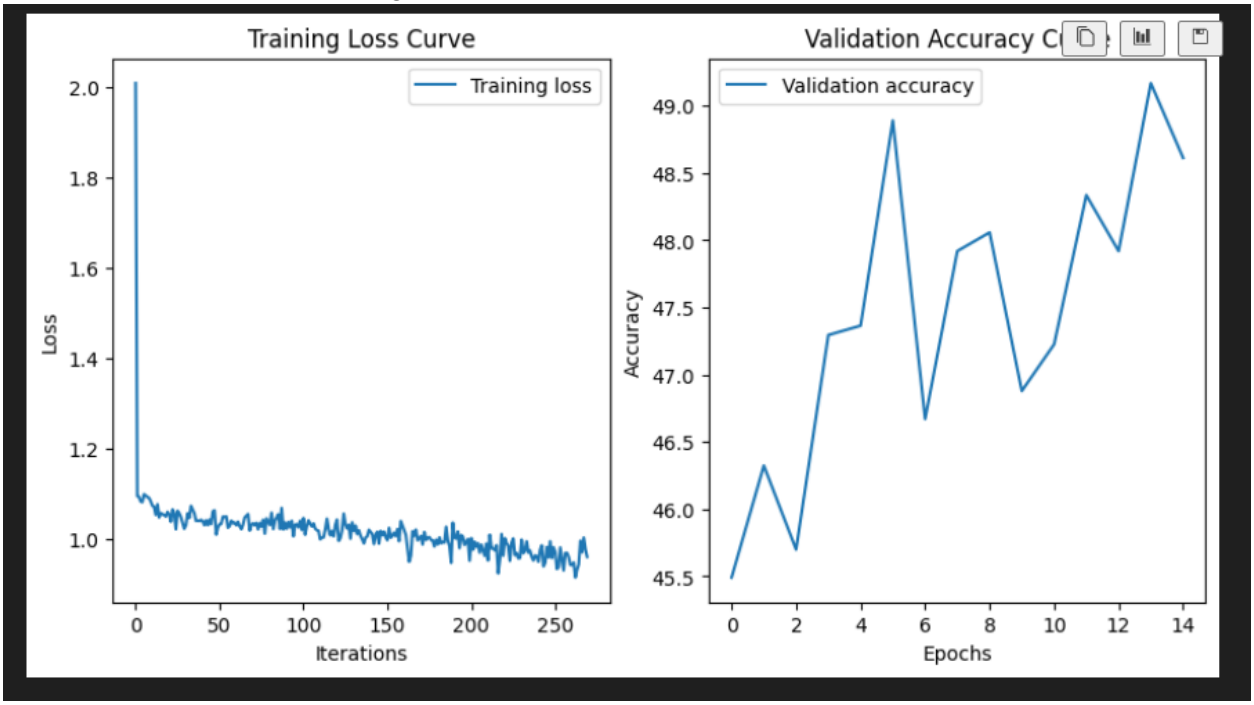
Validation seems to be capping at around 40-45% regardless of how I tune the hyperparameters so I will make some adjustments to the code.
I'll first try changing how we do the data augmentation. I changed the code to this:

```
from torchvision import transforms, datasets
```

```
# Define data transforms with augmentation
train_transform = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
hue=0.1),
    transforms.RandomResizedCrop(244, scale=(0.8, 1.0)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4373, 0.4091, 0.3735], std=[0.2990,
0.2863, 0.2986])
])

# Apply transforms to training dataset
dataset = datasets.ImageFolder(root='~/APS 360/Data preprocessing/Image
data', transform=train_transform)
```

Batch size: 64 Learning rate: 0.001 Epochs: 15

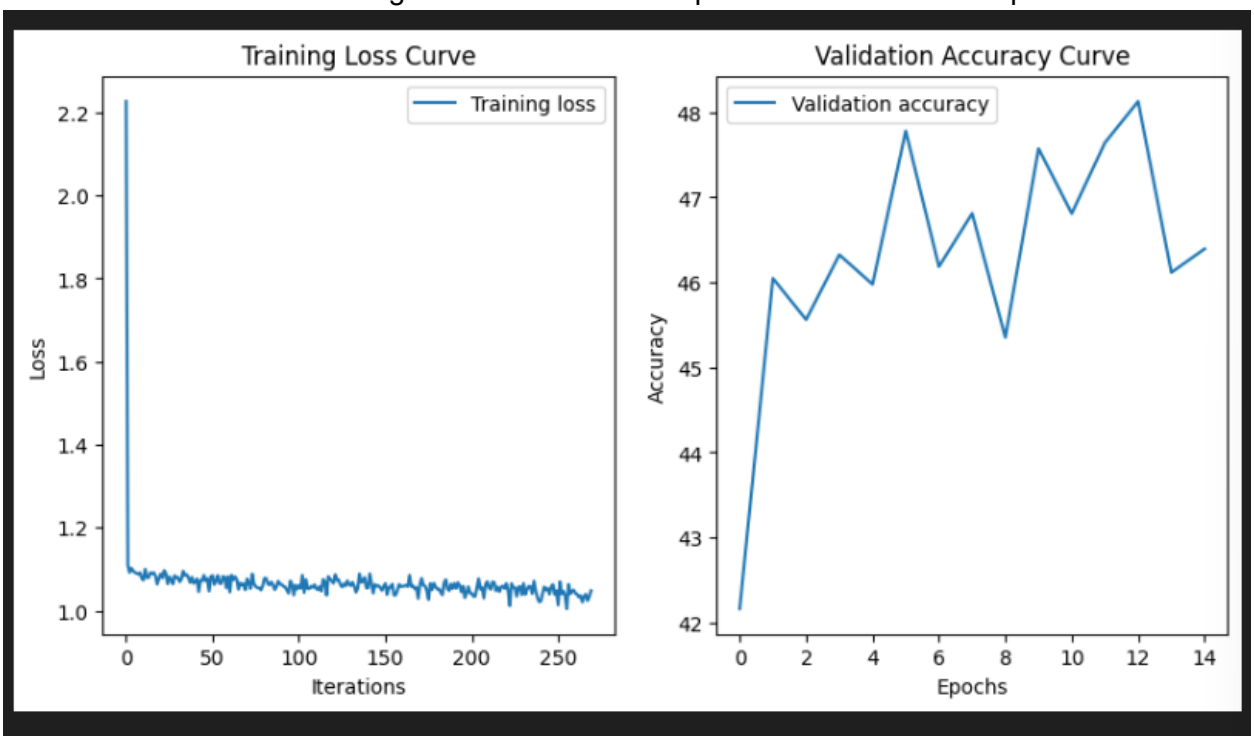


This had a slightly higher accuracy, around 48% but more loss

I'm now gonna try adding dropout layers to the optimizer to help improve generalization. This is how I modified the code:

```python
# Define the CNN model
class SentimentCNN(nn.Module):
    def __init__(self):
        super(SentimentCNN, self).__init__()
        .
        .
        .
        self.dropout = nn.Dropout(0.5)
        .
        .
        .
    def forward(self, x):
        .
        .
        .
        x = self.dropout(x)
        return x
```

Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.5



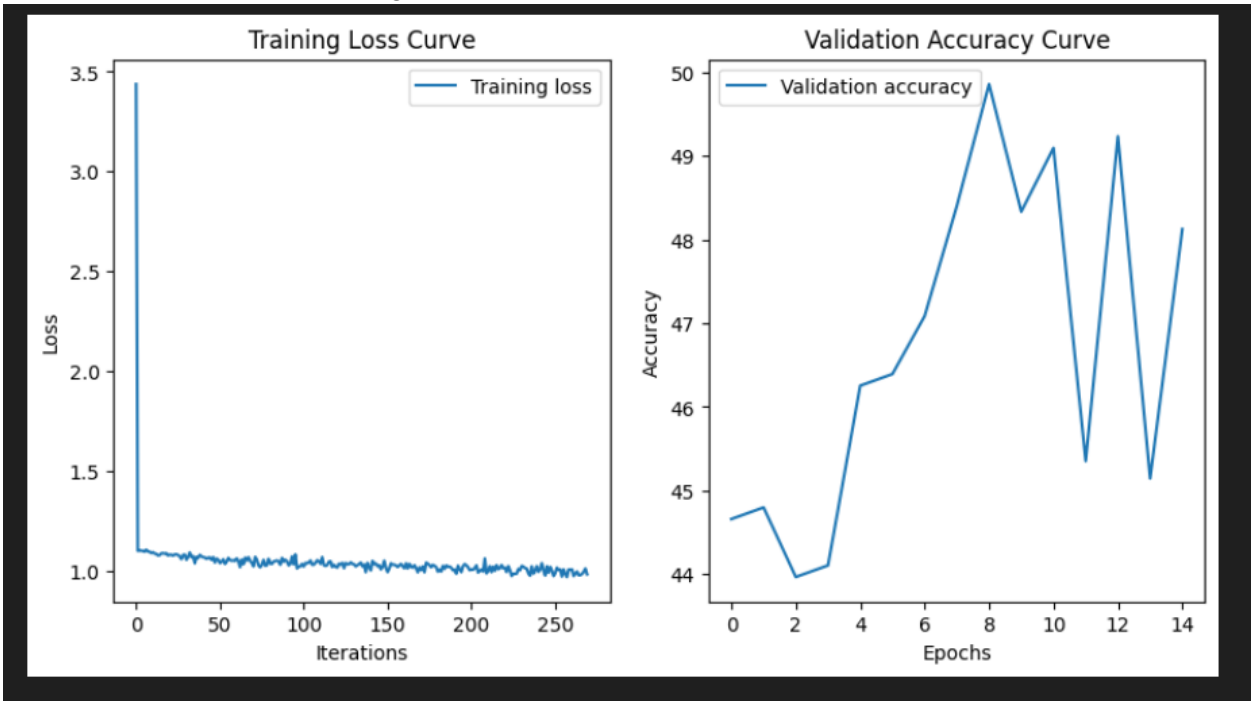
Similar to previous

Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.9

```
[3,    10] loss: 1.095
[3,    20] loss: 1.099
[3,    30] loss: 1.099
[3,    40] loss: 1.098
[3,    50] loss: 1.097
[3,    60] loss: 1.099
[3,    70] loss: 1.101
[3,    80] loss: 1.099
[3,    90] loss: 1.098
[3,   100] loss: 1.099
[3,   110] loss: 1.100
[3,   120] loss: 1.099
[3,   130] loss: 1.097
[3,   140] loss: 1.098
[3,   150] loss: 1.100
[3,   160] loss: 1.099
[3,   170] loss: 1.099
[3,   180] loss: 1.098
Validation Accuracy of the network on the validation images: 35 %
```

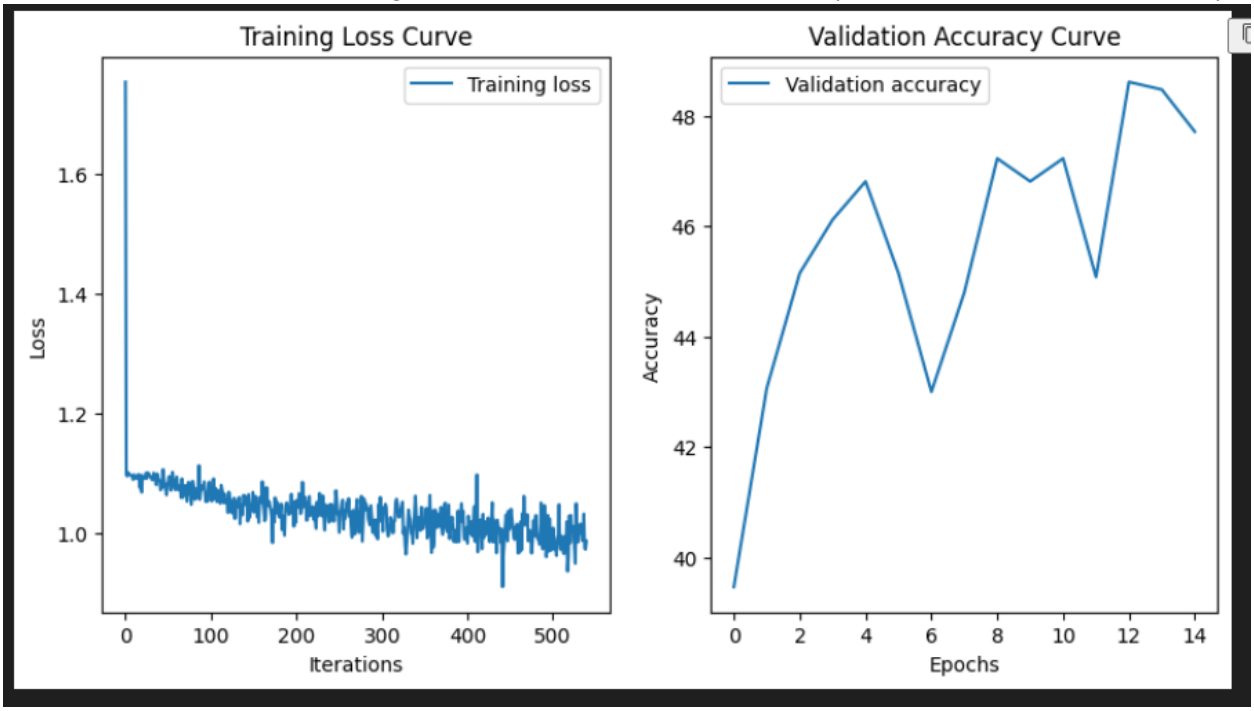
I stopped after 3 interactions since the validation was around 35%

Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.1



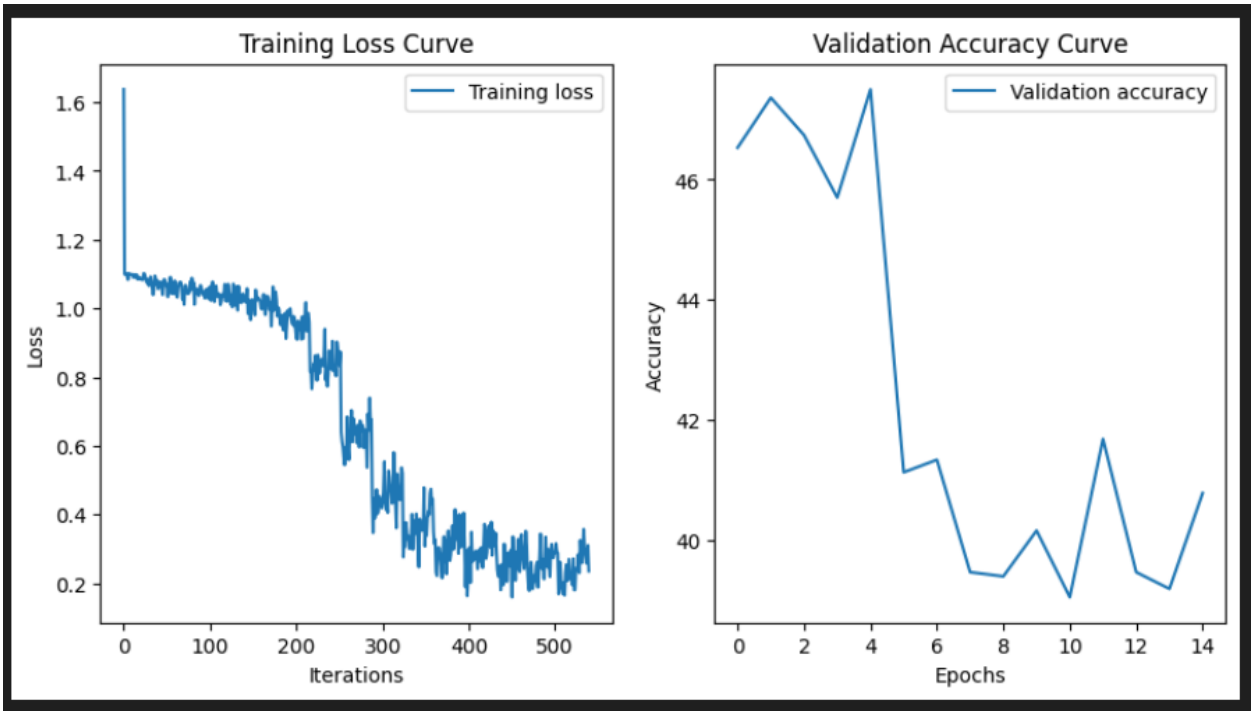
Doesn't seem to be improving the validation, and the loss is high. I'm gonna remove this for now.

Batch size: 32 Learning rate: 0.001 Epochs: 20 (updated transformed dataset)

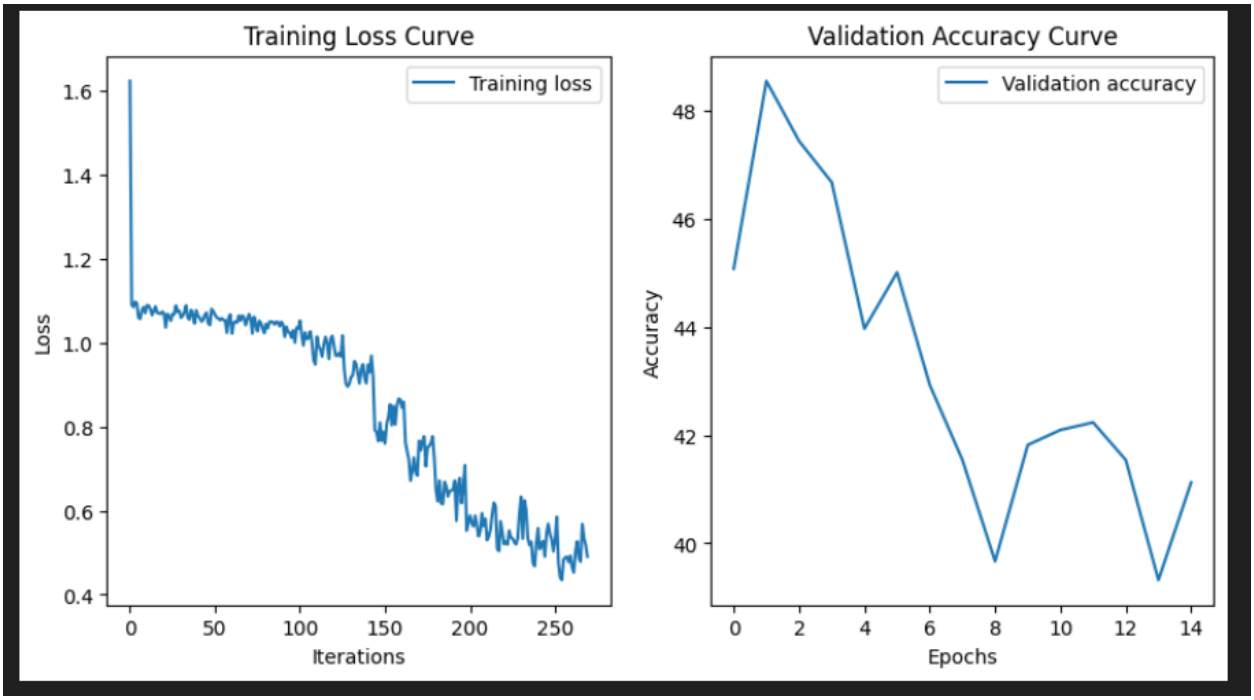


Training loss is still around 1, I was getting around 0.1-0.2 at best earlier so I'm going to revert the updated transformation to the original and add the regularization back in.

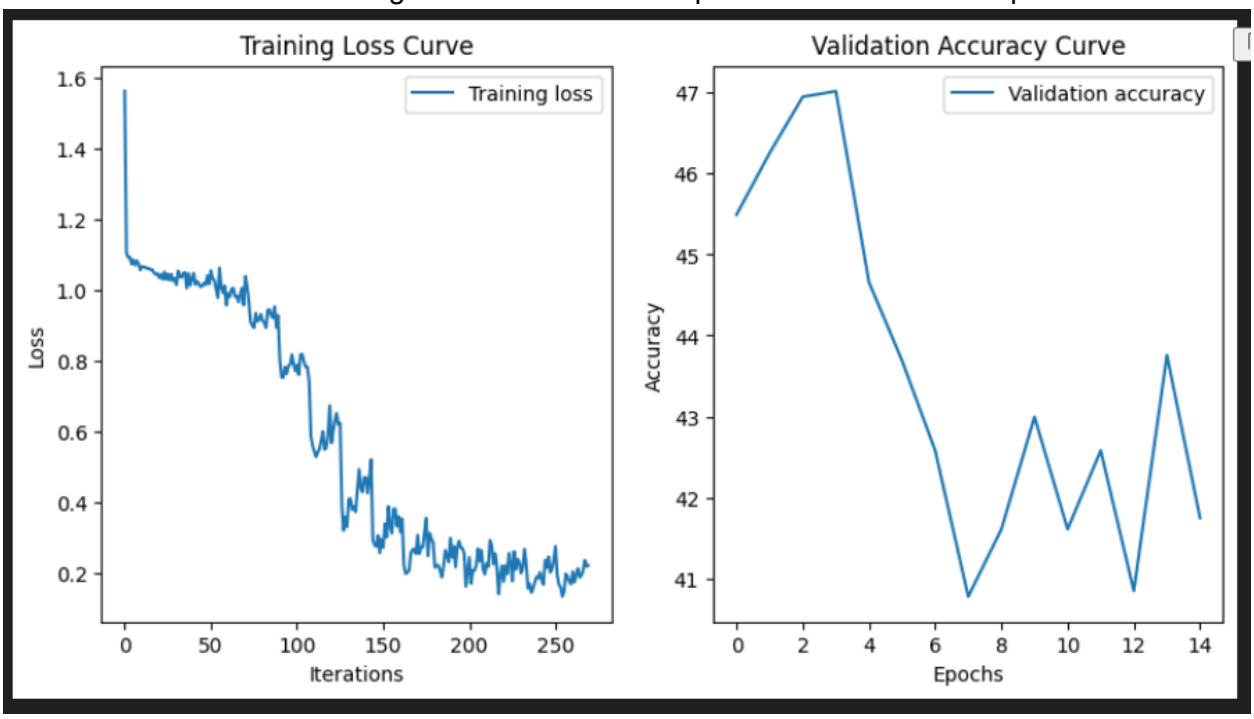
Batch size: 32 Learning rate: 0.001 Epochs: 15 Dropout: 0.2



Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.5



Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.1

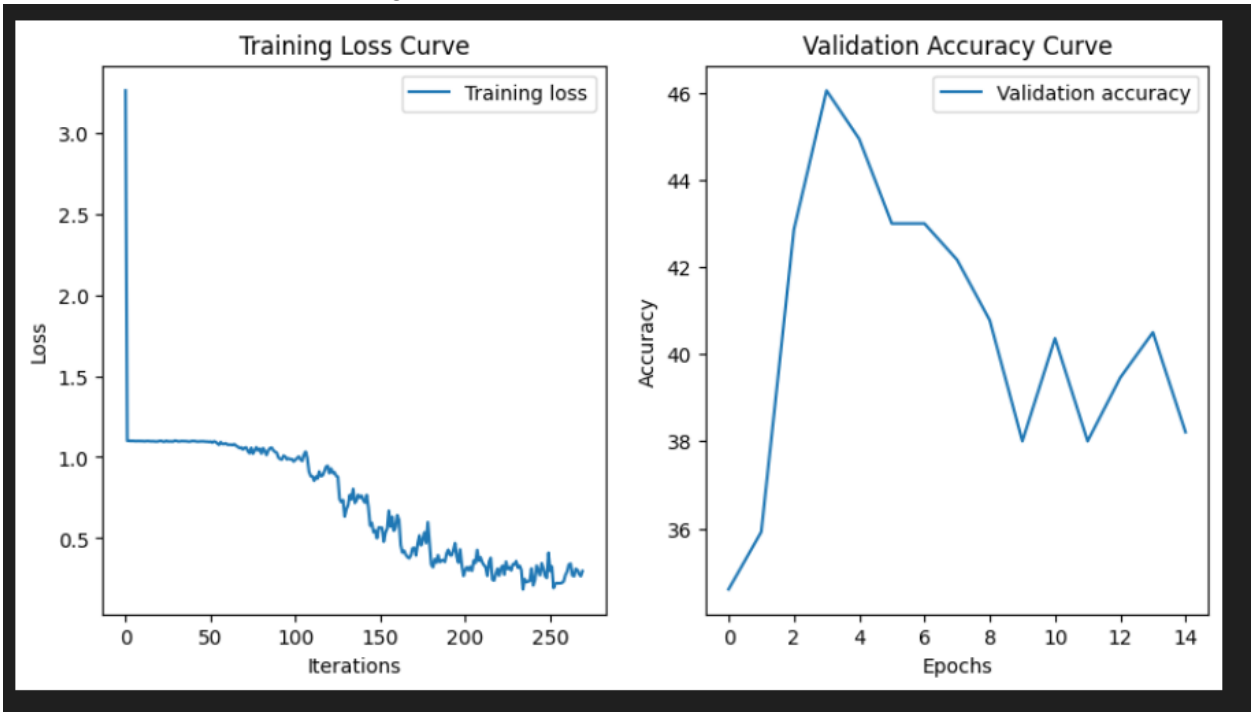


I’m gonna try adding batch normalization. This is how I updated the code:

```
# Define the CNN model
```

```python
class SentimentCNN(nn.Module):
    def __init__(self):
        super(SentimentCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.conv1_bn = nn.BatchNorm2d(16)
        .
        .
        .
    def forward(self, x):
        x = self.pool(F.relu(self.conv1_bn(self.conv1(x))))
        .
        .
        .
        return x
```

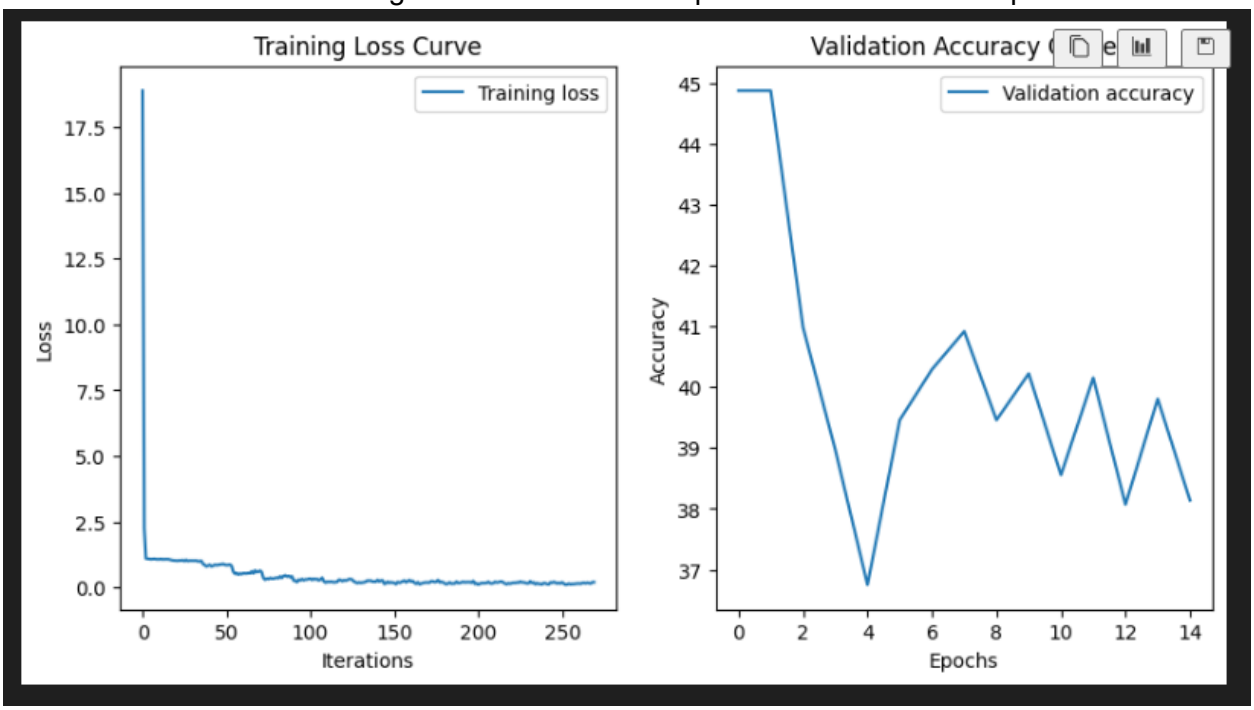
Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.1



Still validation around 40% :/

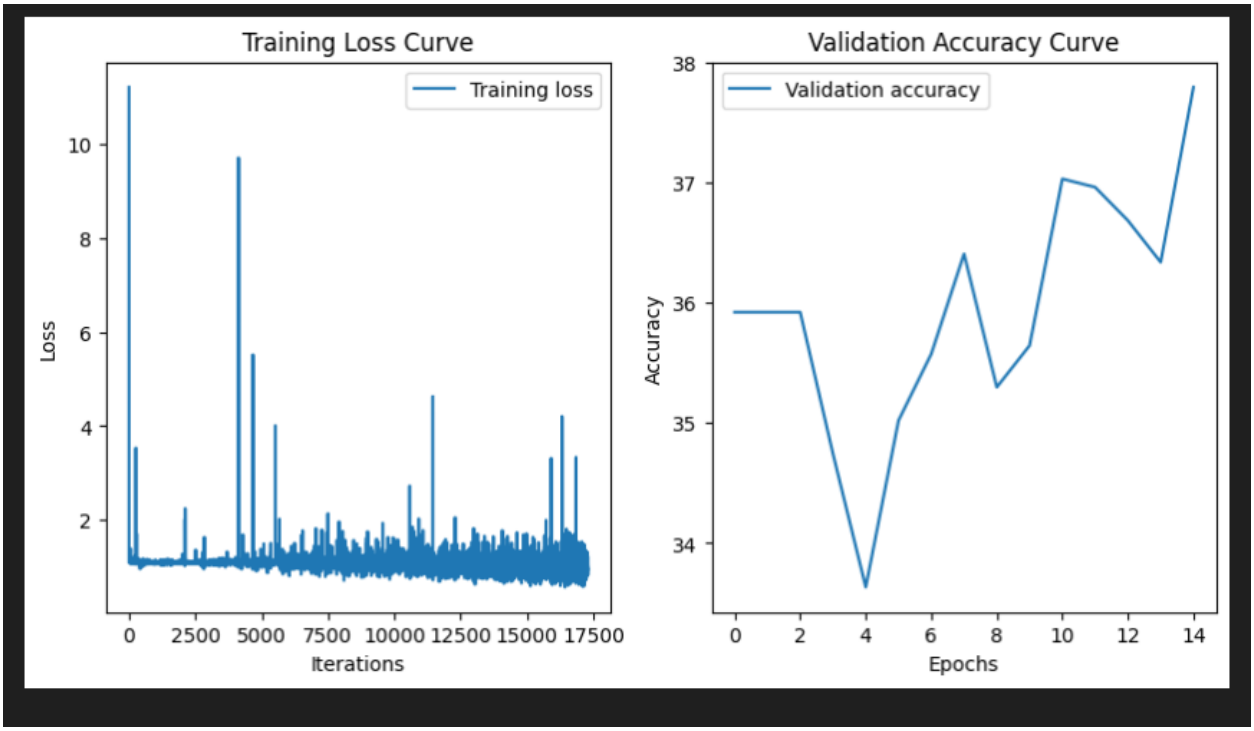
I will try removing a layer so going from 3 layers to 2 layers

Batch size: 64 Learning rate: 0.001 Epochs: 15 Dropout: 0.1

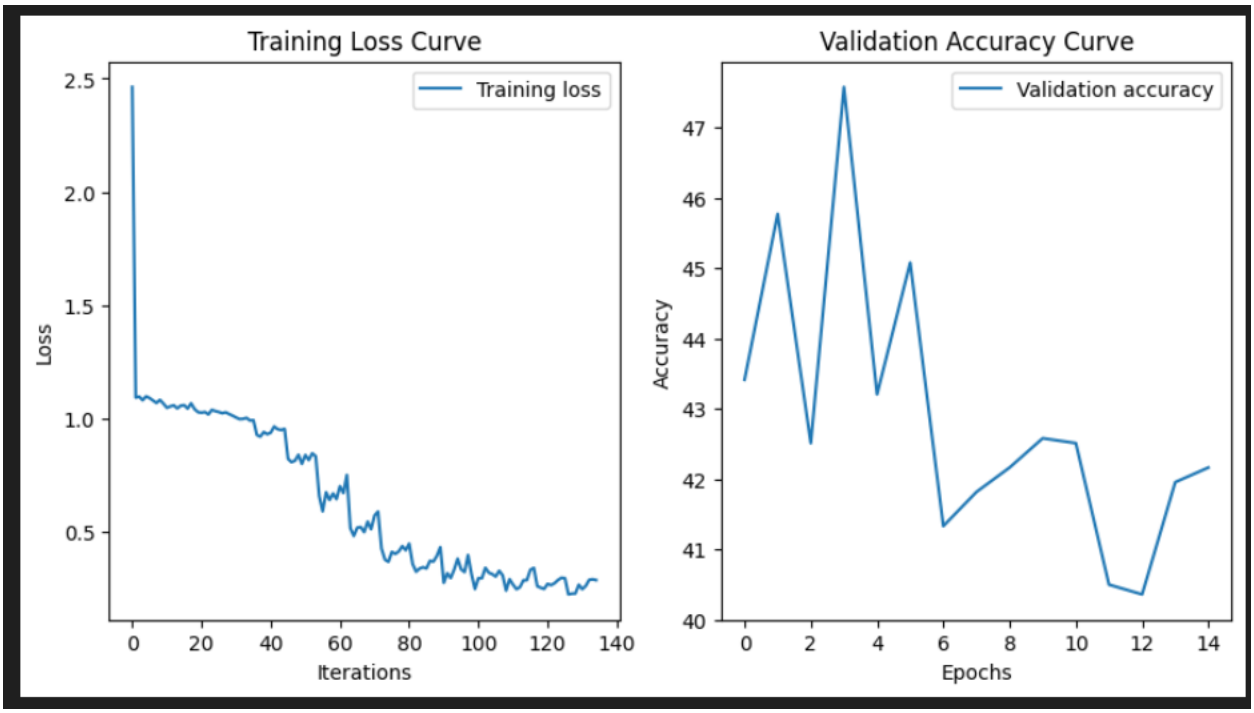


Didn’t make much of a change. I added back the third layer

Batch size: 1 Learning rate: 0.001 Epochs: 15 Dropout: 0.1

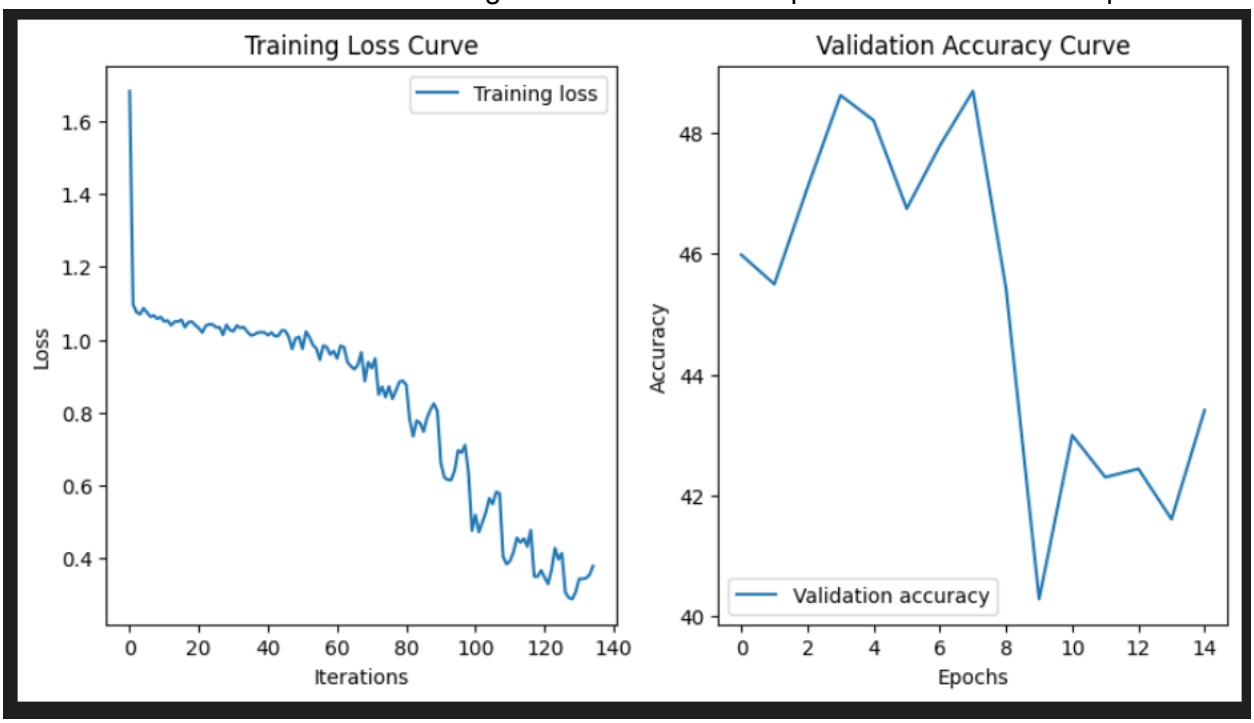


Batch size: 128 Learning rate: 0.001 Epochs: 15 Dropout: 0.2

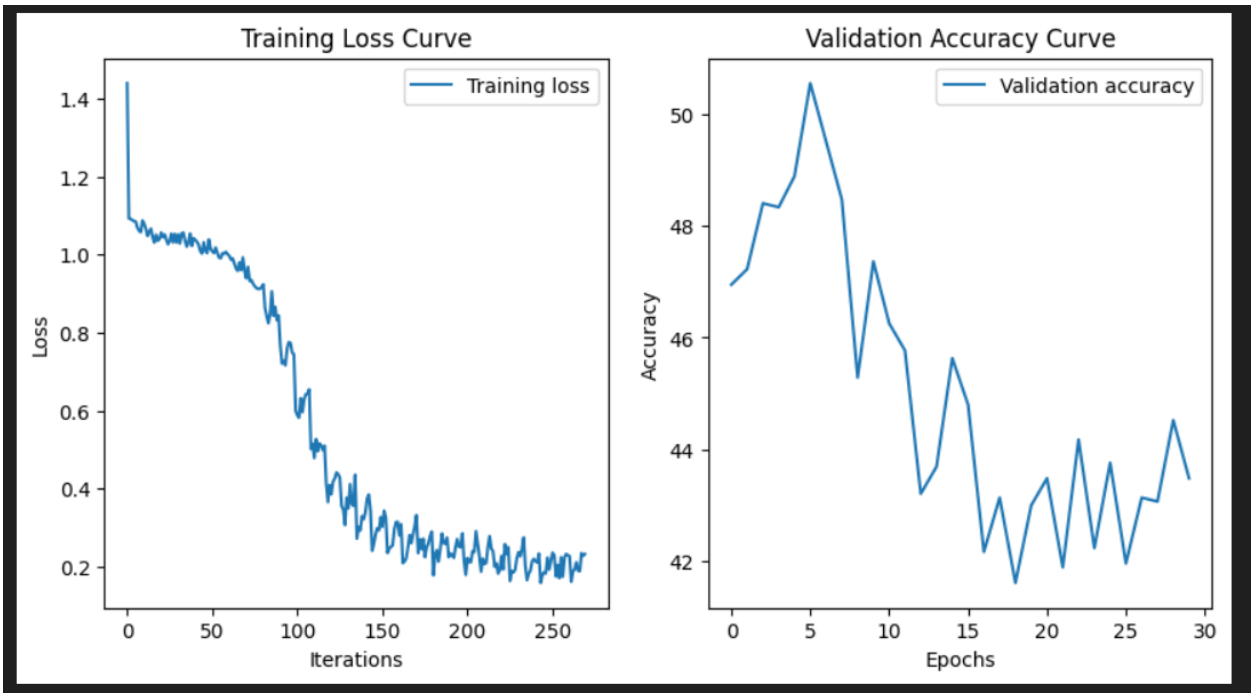


I will try adding a fourth layer

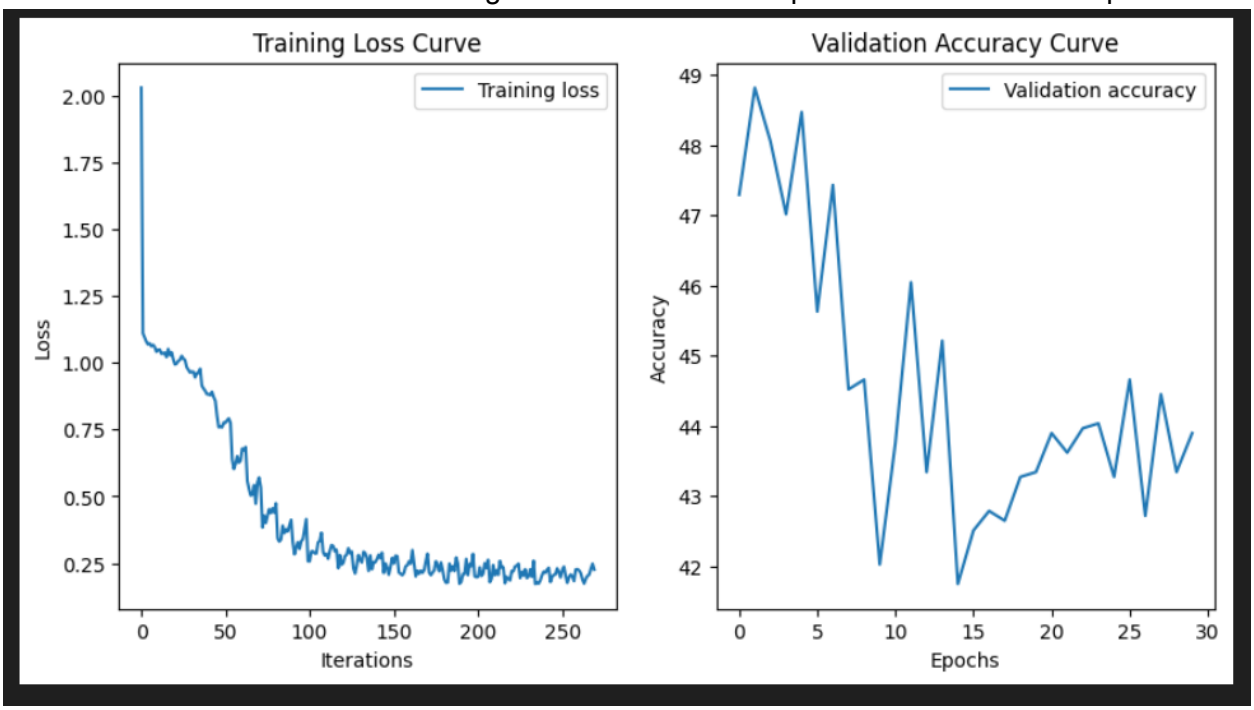
Batch size: 128 Learning rate: 0.001 Epochs: 15 Dropout: 0.2



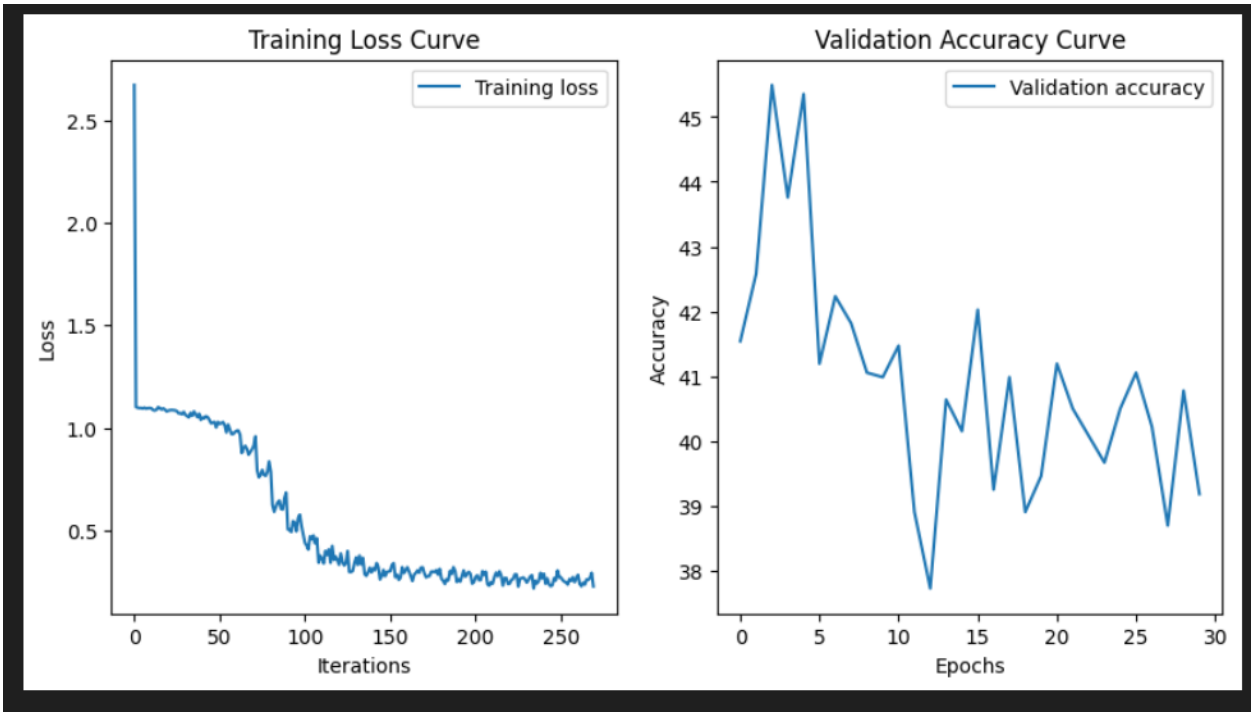
Batch size: 128 Learning rate: 0.001 Epochs: 30 Dropout: 0.2



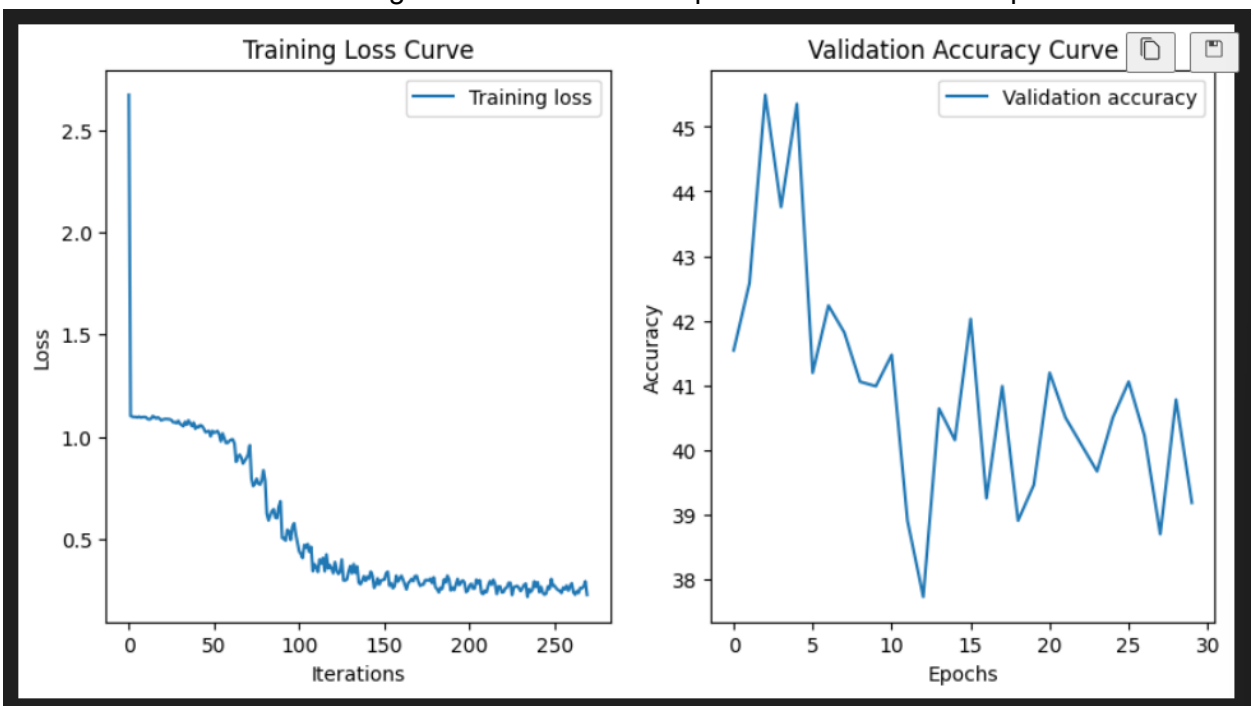
Batch size: 128 Learning rate: 0.0005 Epochs: 30 Dropout: 0.2



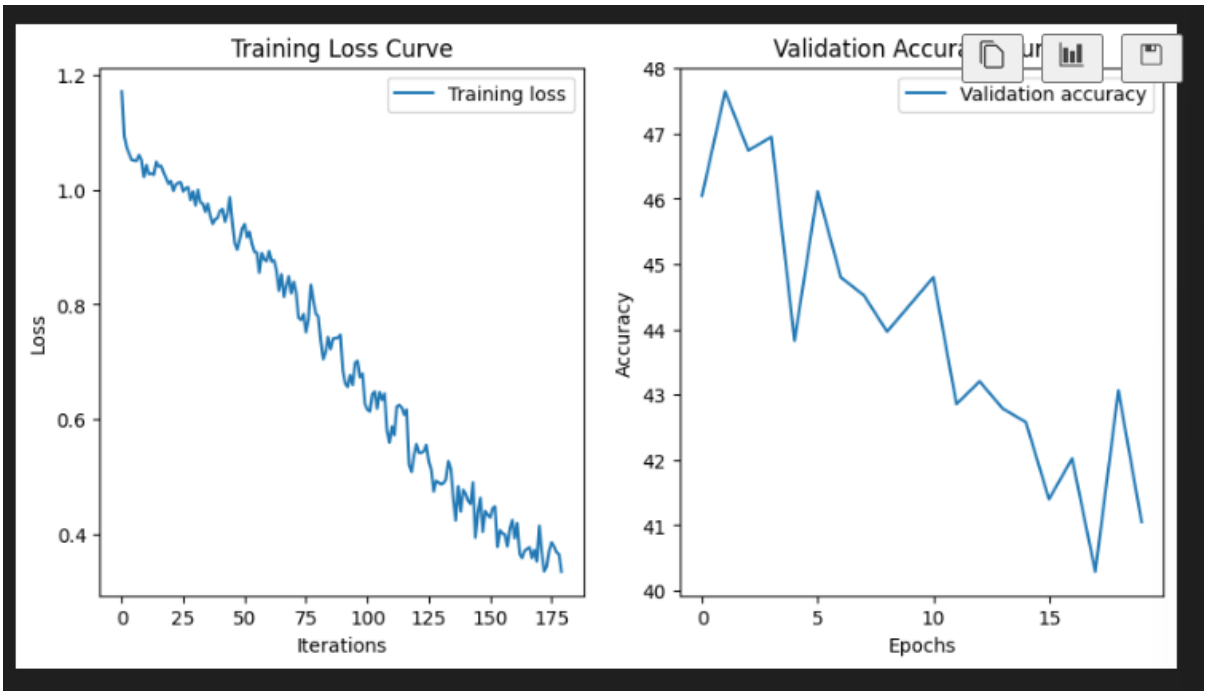
Batch size: 128 Learning rate: 0.001 Epochs: 30 Dropout: 0.3



Batch size: 32 Learning rate: 0.001 Epochs: 30 Dropout: 0.3



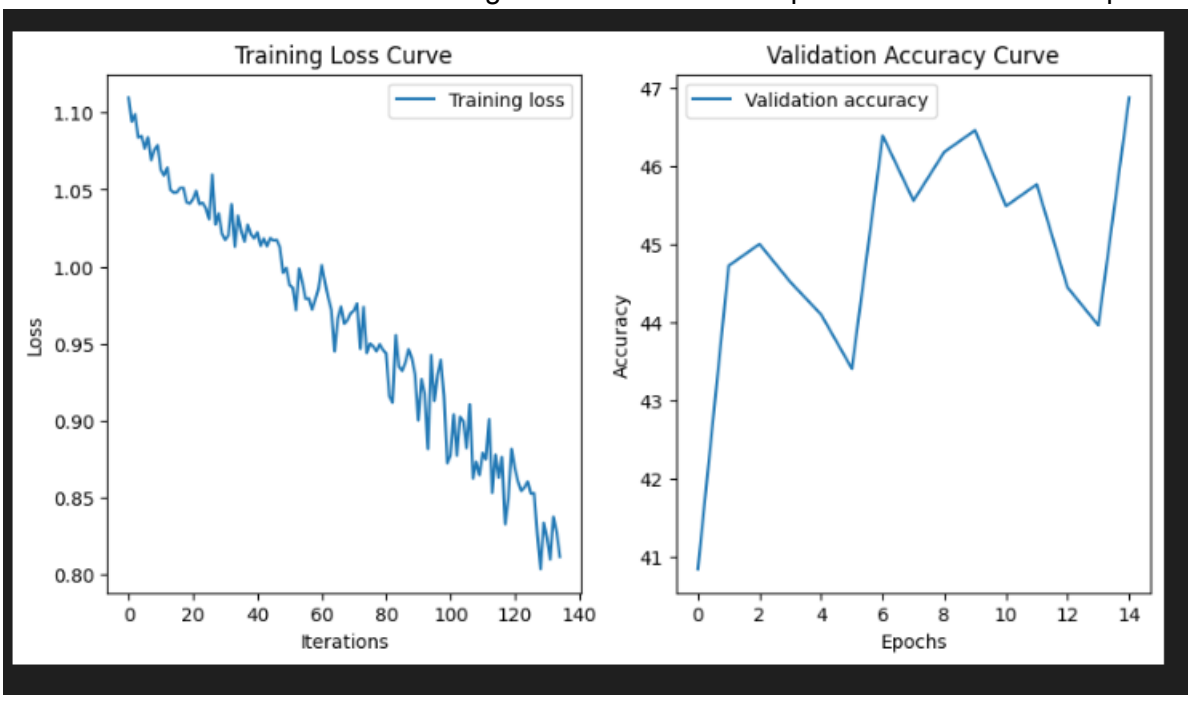
Batch size: 128 Learning rate: 1e-4 Epochs: 20 Dropout: 0.2



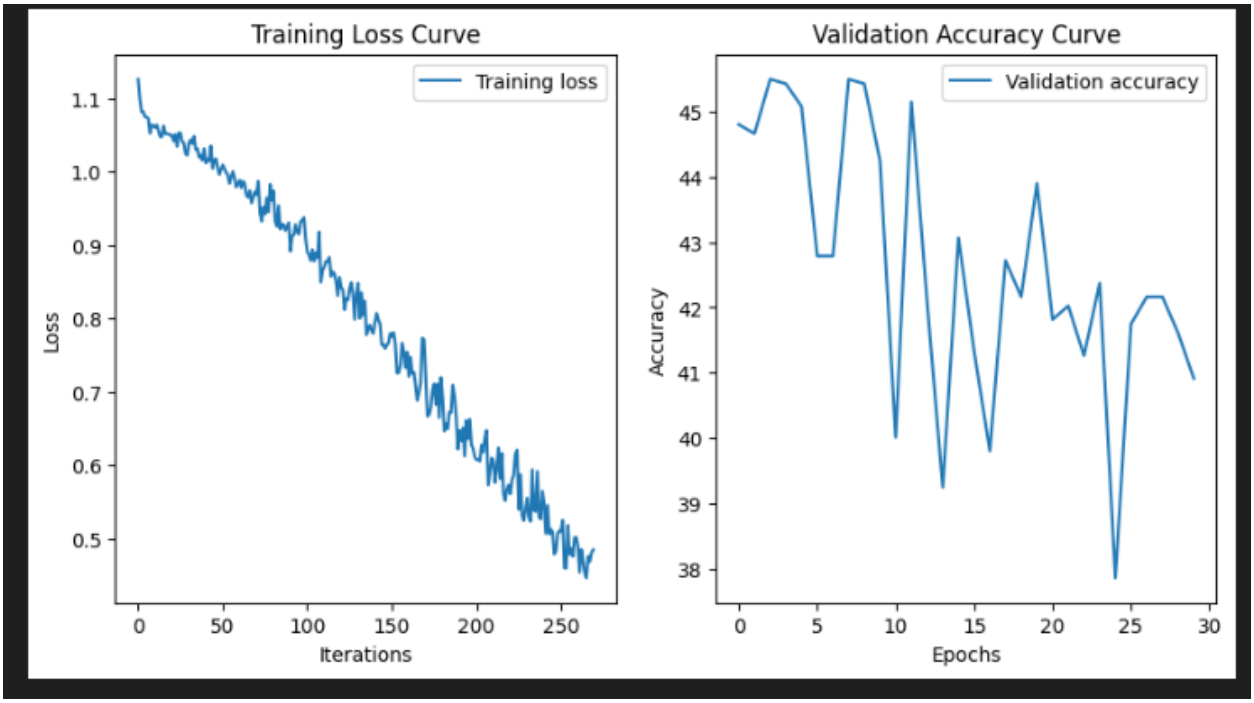
Changed

```
        self.conv1 = nn.Conv2d(3, 4, kernel_size=3, padding=1)
        self.conv1_bn = nn.BatchNorm2d(4)
        self.conv2 = nn.Conv2d(4, 8, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(8, 16, kernel_size=3, padding=1)
```

Batch size: 128 Learning rate: 1e-4 Epochs: 15 Dropout: 0.2



Batch size: 128 Learning rate: 1e-4 Epochs: 30 Dropout: 0.2



Batch size: 32 Learning rate: 1e-4 Epochs: 15 Dropout: 0.2