# ECE 601: Homework 5

Hazem Hanafy

Email: hhanafy@purdue.edu

Due date: 11:59 PM, Feb. 17, 2025

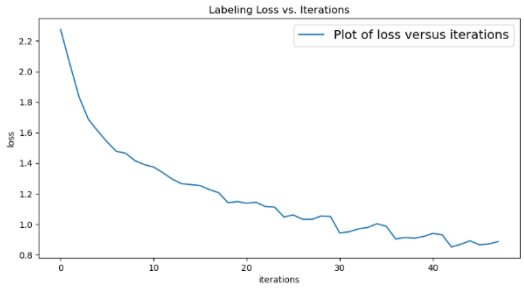
(Spring 2025)

## 2. Experimental Tasks and Analysis

### Task 1: Compare between Net, Net2, and Net3

Using Net and Net2 from ExperimentsWithCIFAR and comparing the results with the custom network Net3 which contains 8 convolution layers in terms of training vs loss curve, confusion matrix, and overall accuracy.

Net() network form ExperimentsWithCIFAR results:



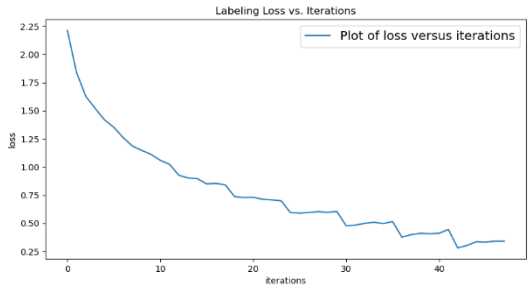

(a)

Displaying the confusion matrix:

| | plane | car | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| plane: | 79.60 | 1.00 | 2.90 | 1.80 | 1.00 | 0.60 | 1.70 | 0.70 | 5.70 | 5.00 |
| car: | 4.90 | 72.20 | 0.50 | 0.80 | 0.70 | 0.70 | 2.50 | 0.60 | 4.20 | 12.90 |
| bird: | 11.10 | 0.50 | 48.10 | 6.40 | 9.90 | 5.00 | 13.70 | 2.60 | 1.00 | 1.70 |
| cat: | 3.90 | 0.80 | 7.30 | 44.70 | 6.50 | 14.50 | 15.40 | 2.50 | 1.30 | 3.10 |
| deer: | 4.90 | 0.30 | 8.30 | 5.10 | 53.60 | 2.20 | 19.10 | 5.30 | 0.90 | 0.30 |
| dog: | 2.20 | 0.50 | 6.10 | 18.60 | 6.10 | 50.70 | 8.70 | 5.20 | 0.40 | 1.50 |
| frog: | 1.00 | 0.50 | 2.50 | 3.40 | 2.40 | 1.20 | 87.70 | 0.30 | 0.60 | 0.40 |
| horse: | 3.90 | 0.00 | 3.80 | 5.30 | 9.40 | 6.70 | 3.00 | 64.20 | 0.80 | 2.90 |
| ship: | 14.80 | 3.50 | 1.10 | 2.10 | 0.30 | 0.70 | 1.30 | 0.60 | 71.70 | 3.90 |
| truck: | 5.70 | 4.90 | 1.10 | 2.60 | 0.70 | 1.20 | 2.50 | 1.80 | 3.10 | 76.40 |

(b)

*Figure 1. (a) represent training vs loss curve for the Net() network, (b) Confusion matrix*

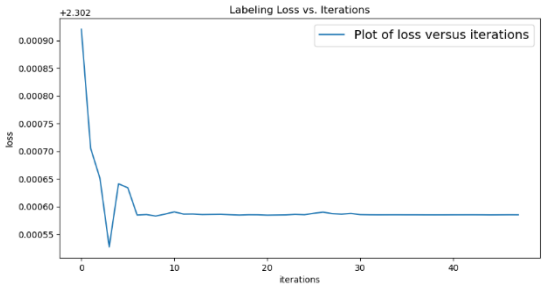



Displaying the confusion matrix:

| | plane | car | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| plane: | 83.10 | 1.70 | 4.60 | 1.60 | 1.70 | 0.30 | 1.30 | 1.20 | 2.50 | 2.00 |
| car: | 0.70 | 87.30 | 1.10 | 1.20 | 0.10 | 0.30 | 1.20 | 0.20 | 1.10 | 6.80 |
| bird: | 6.20 | 0.80 | 66.90 | 5.00 | 6.90 | 5.20 | 5.50 | 2.40 | 0.70 | 0.40 |
| cat: | 2.70 | 0.60 | 6.40 | 58.70 | 5.80 | 15.00 | 6.00 | 2.80 | 0.60 | 1.40 |
| deer: | 1.00 | 0.50 | 5.50 | 5.20 | 77.10 | 1.70 | 5.40 | 3.00 | 0.50 | 0.10 |
| dog: | 1.00 | 0.50 | 4.90 | 15.40 | 4.60 | 66.80 | 3.50 | 2.60 | 0.30 | 0.40 |
| frog: | 0.80 | 0.20 | 3.90 | 4.10 | 1.90 | 1.60 | 86.50 | 0.40 | 0.50 | 0.10 |
| horse: | 1.10 | 0.20 | 2.30 | 3.30 | 8.60 | 5.10 | 0.30 | 78.20 | 0.20 | 0.70 |
| ship: | 7.10 | 2.70 | 1.80 | 1.70 | 0.80 | 0.40 | 0.40 | 0.60 | 81.40 | 3.10 |
| truck: | 3.50 | 7.20 | 1.00 | 1.80 | 0.40 | 0.80 | 1.00 | 1.80 | 0.90 | 81.60 |

(a) (b)

*Figure 2. (a) represent training vs loss curve for the Net2() network, (b) Confusion matrix*




Displaying the confusion matrix:

| | plane | car | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| plane: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| car: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| bird: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| cat: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| deer: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| dog: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| frog: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| horse: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| ship: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| truck: | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

(a) (b)

*Figure 3. (a) represent training vs loss curve for the Net3() network, (b) Confusion matrix*

Based on the results from figures 1-3 increasing the number of layers does not always give better results. As noticed when the number of layers increased from Net() to Net2() the loss value at the end is lower also the classification results are better based on the confusion matrix. But for Net3() which contain 8 layers, I got really good results till the 7$^{th}$ layer, but when I added the 8$^{th}$ layer the network always overfitted and gives only one dominant class as the result.

## Overall accuracy for each network

| Network | Overall accuracy |
|---|---|
| **Net()** | 64% |
| **Net2()** | 76% |
| **Net3()** | 10% |

## Per class accuracy

| Class | Net accuracy (%) | Net2 accuracy (%) | Net3 accuracy (%) |
|---|---|---|---|
| **Plane** | 79 | 83 | 100 |
| **Car** | 72 | 87 | 0 |
| **Bird** | 48 | 66 | 0 |
| **Cat** | 44 | 58 | 0 |
| **Deer** | 53 | 77 | 0 |
| **Dog** | 50 | 66 | 0 |
| **Frog** | 87 | 86 | 0 |
| **Horse** | 64 | 78 | 0 |
| **Ship** | 71 | 81 | 0 |
| **Truck** | 76 | 81 | 0 |

## Discussion

The accuracy differences among the three networks stem from their architectural choices. **Net()** achieves 64% accuracy with a simple design of two convolutional layers, effective max pooling, and fully connected layers, providing a balance between depth and feature extraction. **Net2()**, at 76% accuracy, improves upon this by increasing the number of filters, using smaller kernel sizes for better feature extraction. In contrast, **Net3()** drastically underperforms at 10% due to excessive convolutional layers without sufficient pooling, leading to large feature maps that are difficult to process, vanishing gradients, and poor weight updates. Additionally, aggressive dropout may contribute to information loss, preventing the network from learning effectively. To improve **Net3()**, adding more pooling layers, reducing dropout, and incorporating batch normalization could help stabilize training and enhance accuracy.

Code used:

```
# %%
class CustomExperimentwithCIFAR(DLStudio.ExperimentsWithCIFAR):
    def __init__(self, dl_studio):
        super().__init__(dl_studio)

    def Net3(self):
        class Net3(nn.Module):
            def __init__(self):
                super().__init__()

                # Convolutional Layers
                self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
                self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)

                self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)
                self.conv4 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)

                self.conv5 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
                self.conv6 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)

                self.conv7 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
                self.conv8 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)

                # Pooling and Dropout Layers
                self.pool = nn.MaxPool2d(2, 2)  # Reduce size after key layers
                self.dropout = nn.Dropout(0.2)  # Regularization

                # Fully Connected Layers
                self.fc1 = nn.Linear(512*2*2, 256)
                self.fc2 = nn.Linear(256, 10)


            def forward(self, x):
                x = self.pool(F.relu(self.conv1(x)))  # 1st conv + pool
```

```
                x = F.relu(self.conv2(x))

                x = self.pool(F.relu(self.conv3(x)))
                x = F.relu(self.conv4(x))

                x = self.pool(F.relu(self.conv5(x)))
                x = F.relu(self.conv6(x))

                x = self.pool(F.relu(self.conv7(x)))
                x = F.relu(self.conv8(x))
                # print(x.shape)  # Add this before x.view()
                x = x.view(-1, 512*2*2)

                x = F.relu(self.fc1(x))
                x = self.dropout(x)  # Dropout applied after first FC layer
                x = F.relu(self.fc2(x))

                return x


        return Net3()
# exp_cifar = DLStudio.ExperimentsWithCIFAR( dl_studio = dls )
exp_cifar = CustomExperimentwithCIFAR( dl_studio = dls )

#exp_cifar.load_cifar_10_dataset_with_augmentation()
exp_cifar.load_cifar_10_dataset()

# model = exp_cifar.Net()
# model = exp_cifar.Net2()            ## <<< Try this also but first comment out
                                      ##     the above line.
model = exp_cifar.Net3()

## display network properties
number_of_learnable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the model: %d" % number_of_learnable_params)

exp_cifar.run_code_for_training(model, display_images=False)

exp_cifar.run_code_for_testing(model, display_images=False)

```