


## HW6 – Write Up


### Initial Accuracies:



```
Accuracy for class: plane is 41.0 %
Accuracy for class: car   is 41.5 %
Accuracy for class: bird  is 32.5 %
Accuracy for class: cat   is 27.6 %
Accuracy for class: deer  is 41.1 %
Accuracy for class: dog   is 34.5 %
Accuracy for class: frog  is 51.9 %
Accuracy for class: horse is 66.6 %
Accuracy for class: ship  is 75.7 %
Accuracy for class: truck is 87.3 %
```

Change 1: Increasing the number of loops from 2 to 8. It should increase the accuracy of the test sets.

Observations: The accuracies of the class improved after increasing the number of epochs. Increasing the number of epochs can improve the accuracy of the CNN by allowing the network to see more examples of the training data during the training process. During each epoch, the network updates its parameters based on the loss between its predicted outputs and the ground truth labels of the training examples. By running the training process for more epochs, the network has the opportunity to fine-tune its parameters to better capture the patterns in the data and improve its ability to generalize to new, unseen examples.



```
Accuracy for class: plane is 64.8 %
Accuracy for class: car   is 75.4 %
Accuracy for class: bird  is 46.9 %
Accuracy for class: cat   is 39.4 %
Accuracy for class: deer  is 35.9 %
Accuracy for class: dog   is 67.8 %
Accuracy for class: frog  is 74.1 %
Accuracy for class: horse is 65.6 %
Accuracy for class: ship  is 73.5 %
Accuracy for class: truck is 66.6 %
```

Change 2: Increase the learning rate from .001 to .01. I think increasing the rate of learning will improve the accuracy of the data. With a higher learning rate we can converge quicker on to a good set of parameters by taking larger steps in the direction of the steepest descent of the loss function.

Observations: The accuracies significantly went down. They are way worse than before. I believe I set it way too high. If the learning rate is set too high, the network may overshoot the optimal set of parameters and become unstable, resulting in poor performance. This is because a high learning rate can cause the network to "bounce" around the optimal solution and fail to converge to a good set of parameters.

```
↳ Accuracy for class: plane is 15.9 %  
Accuracy for class: car is 47.2 %  
Accuracy for class: bird is 43.1 %  
Accuracy for class: cat is 4.8 %  
Accuracy for class: deer is 13.3 %  
Accuracy for class: dog is 9.1 %  
Accuracy for class: frog is 0.1 %  
Accuracy for class: horse is 28.3 %  
Accuracy for class: ship is 3.5 %  
Accuracy for class: truck is 48.1 %
```

Change 3: Adding a linear layer by changing up the Net class. It might make the accuracy go down because it might cause instability which leads to poor performance because the values are too small or too large.

Observations: Some accuracies went up and some went down. The addition of a linear layer can provide an additional level of abstraction and enable the network to capture higher-level representations of the input data. This could improve the network's ability to generalize unseen examples and improve the accuracy. However, the addition of a linear layer can also introduce new sources of error and potentially decrease the accuracy of the network.

```

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 69)
        self.fc4 = nn.Linear(69, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.fc4(x)
        return x

```

## Accuracies

```

☞ Accuracy for class: plane is 53.6 %
Accuracy for class: car is 67.3 %
Accuracy for class: bird is 8.4 %
Accuracy for class: cat is 11.5 %
Accuracy for class: deer is 0.0 %
Accuracy for class: dog is 11.1 %
Accuracy for class: frog is 66.8 %
Accuracy for class: horse is 14.6 %
Accuracy for class: ship is 0.9 %
Accuracy for class: truck is 0.2 %

```

Change 4 : Changing up the sizes of the fully connected layers. I think increasing the size of the output of a linear layer can increase the capacity of the model to learn complex features and potentially improve the accuracy. This is because the increased output size allows for a higher-dimensional feature space in which to learn representations of the input data.

Observations:

Some accuracies decreased while some increased. Based off the output of the accuracies, the increased capacity can also heighten the risk of overfitting the training data. This can cause the accuracies to go down, particularly if the model is not able to generalize well to new, unseen examples.

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 250)
        self.fc2 = nn.Linear(250, 169)
        self.fc3 = nn.Linear(169, 100)
        #self.fc4 = nn.Linear(69, 10)
```

Accuracies:

```
➡ Accuracy for class: plane is 14.5 %
Accuracy for class: car is 11.9 %
Accuracy for class: bird is 2.0 %
Accuracy for class: cat is 19.5 %
Accuracy for class: deer is 6.3 %
Accuracy for class: dog is 12.0 %
Accuracy for class: frog is 63.4 %
Accuracy for class: horse is 49.5 %
Accuracy for class: ship is 57.9 %
Accuracy for class: truck is 38.7 %
```

Final Network:

Modified CNN

```

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 300)
        self.fc2 = nn.Linear(300, 150)
        self.fc3 = nn.Linear(150, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x

```

Set the learning rate to .0009

Set number of epochs to 12

```

▶ for epoch in range(12): # loop over the dataset multiple times

```

I think this will increase the accuracies because the slower learning rate will allow all the data to be more accurate, because it is taking smaller steps towards the most optimal solution.

Observations: The accuracies significantly improved from the previous run. By decreasing the learning rate, the model is less likely to overfit and more likely to learn generalizable features that lead to better performance on the test data.

## Accuracies:

☞ Accuracy for class: plane is 72.2 %  
Accuracy for class: car is 84.3 %  
Accuracy for class: bird is 44.0 %  
Accuracy for class: cat is 48.2 %  
Accuracy for class: deer is 50.3 %  
Accuracy for class: dog is 52.7 %  
Accuracy for class: frog is 67.1 %  
Accuracy for class: horse is 71.1 %  
Accuracy for class: ship is 72.7 %  
Accuracy for class: truck is 69.3 %