Homework 6 Writeup

Name: Syed Husain **Date:** 4/24/24

First Run:

For this run, there were no modifications.

Results:

```
Accuracy of the network on the 10000 test images: 50 %
Accuracy for class: plane is 54.9 %
Accuracy for class: car   is 49.5 %
Accuracy for class: bird  is 35.4 %
Accuracy for class: cat   is 63.7 %
Accuracy for class: deer  is 31.6 %
Accuracy for class: dog   is 19.8 %
Accuracy for class: frog  is 71.9 %
Accuracy for class: horse is 43.7 %
Accuracy for class: ship  is 64.8 %
Accuracy for class: truck is 73.9 %
```

Second Run - Change 1:

For this run, I decided to increase the number of filters in the convolutional layers. By increasing the number of filters, the network can capture more diverse features from the input images which will potentially improve its ability to discriminate between different classes.

Code:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5)  #increased from 6 to 12 filters
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 32, 5)  #increased from 16 to 32 filters
        self.fc1 = nn.Linear(32 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 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)
        return x
```

Results:

```
Accuracy of the network on the 10000 test images: 58 %
Accuracy for class: plane is 58.6 %
Accuracy for class: car   is 80.1 %
Accuracy for class: bird  is 50.6 %
Accuracy for class: cat   is 32.8 %
Accuracy for class: deer  is 59.8 %
Accuracy for class: dog   is 37.7 %
Accuracy for class: frog  is 53.5 %
Accuracy for class: horse is 73.3 %
Accuracy for class: ship  is 81.5 %
Accuracy for class: truck is 60.8 %
```

The overall accuracy of the network on the test images increased from 50% to 58%. There's a noticeable improvement in accuracy for several classes, such as car, deer, horse, and ship. This improvement can be attributed to the increased capacity of the network to capture more diverse features from the input images due to the higher number of filters in the convolutional layers.

Third Run - Change 2:

For this run, I decided to add a dropout layer after the second convolutional layer. By adding a dropout layer after the second convolutional layer, I aimed to improve the generalization ability of the network as dropout could prevent overfitting by randomly dropping a certain proportion of neurons during training

Code:

```python
class Net(nn.Module):
   def __init__(self):
       super().__init__()
       self.conv1 = nn.Conv2d(3, 12, 5)
       self.pool = nn.MaxPool2d(2, 2)
       self.conv2 = nn.Conv2d(12, 32, 5)
       self.dropout = nn.Dropout2d(p=0.5)  # add dropout layer with p=0.5
       self.fc1 = nn.Linear(32 * 5 * 5, 120)
       self.fc2 = nn.Linear(120, 84)
       self.fc3 = nn.Linear(84, 10)

   def forward(self, x):
       x = self.pool(F.relu(self.conv1(x)))
       x = self.pool(F.relu(self.conv2(x)))
       x = self.dropout(x)  # apply dropout
       x = torch.flatten(x, 1)
       x = F.relu(self.fc1(x))
```

```
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Results:

```
Accuracy of the network on the 10000 test images: 47 %
Accuracy for class: plane is 54.7 %
Accuracy for class: car   is 49.6 %
Accuracy for class: bird  is 25.2 %
Accuracy for class: cat   is 21.0 %
Accuracy for class: deer  is 49.3 %
Accuracy for class: dog   is 45.0 %
Accuracy for class: frog  is 59.8 %
Accuracy for class: horse is 58.6 %
Accuracy for class: ship  is 59.7 %
Accuracy for class: truck is 54.4 %
```

Contrary to expectations, the addition of the dropout layer resulted in a decrease in accuracy across most classes. Dropout aims at preventing overfitting by randomly dropping neurons during training. However, in this case, it seems that the dropout rate of 0.5 might have been too high which led to excessive information loss during training and resulted in a decrease in accuracy.

Fourth Run - Change 3:

For this run, I decided to increase the size of the fully connected layers since increasing the size of the fully connected layers can provide more capacity for the network to learn complex patterns in the data which may potentially improve its performance.

Code:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 32, 5)
        self.fc1 = nn.Linear(32 * 5 * 5, 240)  # Increased from 120 to 240
        self.fc2 = nn.Linear(240, 168)        # Increased from 84 to 168
        self.fc3 = nn.Linear(168, 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)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Results:

```
Accuracy of the network on the 10000 test images: 59 %
Accuracy for class: plane is 56.8 %
Accuracy for class: car   is 82.4 %
Accuracy for class: bird  is 34.5 %
Accuracy for class: cat   is 47.7 %
Accuracy for class: deer  is 63.5 %
Accuracy for class: dog   is 51.3 %
Accuracy for class: frog  is 58.7 %
Accuracy for class: horse is 59.4 %
Accuracy for class: ship  is 73.9 %
Accuracy for class: truck is 68.7 %
```

This modification led to an improvement in accuracy for some classes, including car, deer, frog, horse, and ship. By increasing the size of the fully connected layers, the network gains more capacity to learn complex patterns in the data which may have allowed it to better discriminate between different classes.

Fifth Run - Change 4:

For this run, I decided to use a different nonlinearity instead of ReLU. ReLU can have some limitations where neurons can become inactive during training. Therefore I tried using a different nonlinearity such as Leaky ReLU, to see if it improves the network's performance.

Code:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 32, 5)
        self.fc1 = nn.Linear(32 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.leaky_relu = nn.LeakyReLU(0.1) # use Leaky ReLU with negative slope of 0.1
```

```
def forward(self, x):
    x = self.pool(self.leaky_relu(self.conv1(x)))
    x = self.pool(self.leaky_relu(self.conv2(x)))
    x = torch.flatten(x, 1)
    x = self.leaky_relu(self.fc1(x))
    x = self.leaky_relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Results:

```
Accuracy of the network on the 10000 test images: 59 %
Accuracy for class: plane is 78.6 %
Accuracy for class: car   is 78.3 %
Accuracy for class: bird  is 57.6 %
Accuracy for class: cat   is 41.7 %
Accuracy for class: deer  is 26.3 %
Accuracy for class: dog   is 43.6 %
Accuracy for class: frog  is 73.9 %
Accuracy for class: horse is 62.7 %
Accuracy for class: ship  is 60.7 %
Accuracy for class: truck is 69.2 %
```

While there were improvements in accuracy for certain classes such as plane, car, frog, horse, ship, and truck others experienced decreases such as bird, cat, deer, and dog. Leaky ReLU introduces a small slope for negative input values, which might help prevent neurons from becoming inactive during training. However, the impact may have varied across different classes which resulted in mixed outcomes.

Sixth Run - Final Network:

For the final network, I decided to incorporate the modifications that yielded the most significant improvements in the previous experiments. Based on the results, I increased the number of filters in the convolutional layers as this modification improved the accuracy of several classes.
Next, I increased the size of the fully connected layers as this change also contributed to a notable increase in accuracy. Furthermore, I used a different nonlinearity (Leaky ReLU) instead of ReLU as this modification had the highest network accuracy. Finally, I also increased the number of epochs from 2 to 8 so that the training process goes through the entire dataset for training a total of 8 times.

```python
class FinalNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 12, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(12, 32, 5)
        self.fc1 = nn.Linear(32 * 5 * 5, 240)  # Increased from 120 to 240
        self.fc2 = nn.Linear(240, 168)        # Increased from 84 to 168
        self.fc3 = nn.Linear(168, 10)
        self.leaky_relu = nn.LeakyReLU(0.1) # Use Leaky ReLU with negative slope of 0.1

    def forward(self, x):
        x = self.pool(self.leaky_relu(self.conv1(x)))
        x = self.pool(self.leaky_relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = self.leaky_relu(self.fc1(x))
        x = self.leaky_relu(self.fc2(x))
        x = self.fc3(x)
        return x
for epoch in range(8):  # loop over the dataset multiple times
```

Results:

```
Accuracy of the network on the 10000 test images: 69 %
Accuracy for class: plane is 78.8 %
Accuracy for class: car   is 84.5 %
Accuracy for class: bird  is 63.0 %
Accuracy for class: cat   is 46.1 %
Accuracy for class: deer  is 63.6 %
Accuracy for class: dog   is 50.9 %
Accuracy for class: frog  is 80.9 %
Accuracy for class: horse is 76.6 %
Accuracy for class: ship  is 73.7 %
Accuracy for class: truck is 75.6 %
```

By incorporating the modifications that yielded the most significant improvements, the final network achieved an overall accuracy of 69% on the test images after training for 8 epochs. The final network benefits from the increased capacity to capture diverse features from the input images due to increased filters, enhanced capability to learn complex patterns of larger fully connected layers, and the use of Leaky ReLU. This ultimately led all classes to have an accuracy of close to 50% or greater.

In summary, each modification had a different impact on the network's performance, with some leading to improvements while others resulted in decreases in accuracy. The final network combines the most effective modifications to achieve the highest overall accuracy. Additionally, training the model for 8 epochs instead of 2 allowed for more thorough learning from the dataset, which resulted in further improvements in accuracy across various classes.