Problem 5

Problem 5.1

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data.dataset import random_split
from torchvision import datasets
from sklearn.metrics import confusion_matrix
import PIL
```

In [ ]:
```
##Do Not Touch This Cell

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 8, 5)
        self.conv2 = nn.Conv2d(8, 16, 3)
        self.bn1 = nn.BatchNorm2d(8)
        self.bn2 = nn.BatchNorm2d(16)
        self.fc1 = nn.Linear(16*6*6, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.bn2(self.conv2(out)))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

In [ ]:
```
##Do Not Touch This Cell

device = 'cuda' if torch.cuda.is_available() else 'cpu'
net = Net().to(device)
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)
if device =='cuda':
    print("Train on GPU...")
else:
    print("Train on CPU...")
```

```
Train on GPU...
```

In [ ]:
```
##Do Not Touch This Cell
```

```
max_epochs = 50

random_seed = 671
torch.manual_seed(random_seed)
```

Out[ ]:
```
<torch._C.Generator at 0x7f395a580090>
```

In [ ]:
```
train_transform = transforms.Compose(
     [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))]

test_transform = transforms.Compose(
     [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))]

dataset = torchvision.datasets.CIFAR10(root='./data', train=True, download=Tru
##TODO: Split the set into 80% train, 20% validation (there are 50K total imag
train_num = int(40e3)
val_num = int(10e3)
train_set, val_set = random_split(dataset, [train_num, val_num])

train_loader = torch.utils.data.DataLoader(train_set, batch_size=128, shuffle=
val_loader = torch.utils.data.DataLoader(val_set, batch_size=128, shuffle=Fals

test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=T
test_loader = torch.utils.data.DataLoader(test_set, batch_size=1, shuffle=Fals

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'shi
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:
```
loss_list, acc_list = [], []
loss_list_val, acc_list_val = [], []
criterion = nn.CrossEntropyLoss()

for epoch in range(max_epochs):
    #TODO: set the net to train mode:
    net.train()

    epoch_loss = 0.0
    correct = 0
    for batch_idx, (data, labels) in enumerate(train_loader):
        data, labels = data.to(device), labels.to(device)

        optimizer.zero_grad()
        ##TODO: pass the data into the network and store the output
        outputs = net(data)

        ##TODO: Calculate the cross entropy loss between the output and target
        loss = criterion(outputs,labels)

        ##TODO: Perform backpropagation
        loss.backward()
        optimizer.step()

        ##TODO: Get the prediction from the output
        _,predicted = torch.max(outputs,1)

        ##TODO: Calculate the correct number and add the number to correct
```

```
            correct += (predicted == labels).sum().item()

            ##TODO: Add the loss to epoch_loss.
            epoch_loss += loss.item()

        ##TODO: calculate the average loss
        avg_loss = epoch_loss / len(train_loader)

        ##TODO: calculate the average accuracy
        avg_acc = correct / train_num


        ##TODO: append average epoch loss to loss list
        loss_list.append(avg_loss)

        ##TODO: append average accuracy to accuracy list
        acc_list.append(avg_acc)

        # validation
        ##TODO: set the model to eval mode
        net.eval()

        with torch.no_grad():
            loss_val = 0.0
            correct_val = 0
            for batch_idx, (data, labels) in enumerate(val_loader):
                data, labels = data.to(device), labels.to(device)
                ##TODO: pass the data into the network and store the output
                outputs = net(data)

                ##TODO: Calculate the cross entropy loss between the output and ta
                loss = criterion(outputs,labels)

                ##TODO: Get the prediction from the output
                _,predicted = torch.max(outputs,1)

                ##TODO: Calculate the correct number and add the number to correct
                correct_val += (predicted == labels).sum().item()

                ##TODO: Add the loss to loss_val
                loss_val = loss.item()

            ##TODO: calculate the average loss of validation
            avg_loss_val = loss_val/len(val_loader)

            ##TODO: calculate the average accuracy of validation
            avg_acc_val = correct_val/val_num

            ##TODO: append average epoch loss to loss list of validation
            loss_list_val.append(avg_loss_val)

            ##TODO: append average accuracy to accuracy list of validation
            acc_list_val.append(avg_acc_val)

        print('[epoch %d] loss: %.5f accuracy: %.4f val loss: %.5f val accuracy: %
    [epoch 1] loss: 0.69313 accuracy: 0.7575 val loss: 0.02030 val accuracy: 0.677
    1
    [epoch 2] loss: 0.67881 accuracy: 0.7624 val loss: 0.01914 val accuracy: 0.675
    8
    [epoch 3] loss: 0.66187 accuracy: 0.7671 val loss: 0.01557 val accuracy: 0.686
    6
    [epoch 4] loss: 0.64433 accuracy: 0.7743 val loss: 0.02032 val accuracy: 0.672
    0
    [epoch 5] loss: 0.62643 accuracy: 0.7791 val loss: 0.01711 val accuracy: 0.687
    1
    [epoch 6] loss: 0.61180 accuracy: 0.7855 val loss: 0.01588 val accuracy: 0.671
    3
    [epoch 7] loss: 0.59400 accuracy: 0.7917 val loss: 0.01722 val accuracy: 0.666
    8
    [epoch 8] loss: 0.58561 accuracy: 0.7946 val loss: 0.01814 val accuracy: 0.688
    3
    [epoch 9] loss: 0.56668 accuracy: 0.8007 val loss: 0.02446 val accuracy: 0.681
    6
    [epoch 10] loss: 0.55102 accuracy: 0.8059 val loss: 0.01401 val accuracy: 0.66
    66
    [epoch 11] loss: 0.53541 accuracy: 0.8118 val loss: 0.02055 val accuracy: 0.68
    19
    [epoch 12] loss: 0.52561 accuracy: 0.8144 val loss: 0.01947 val accuracy: 0.68
    69
    [epoch 13] loss: 0.51197 accuracy: 0.8209 val loss: 0.01788 val accuracy: 0.68
    69
    [epoch 14] loss: 0.49704 accuracy: 0.8268 val loss: 0.02116 val accuracy: 0.68
    38
    [epoch 15] loss: 0.48233 accuracy: 0.8314 val loss: 0.01711 val accuracy: 0.67
    92
    [epoch 16] loss: 0.46919 accuracy: 0.8335 val loss: 0.02153 val accuracy: 0.68
    08
    [epoch 17] loss: 0.45705 accuracy: 0.8383 val loss: 0.01591 val accuracy: 0.67
    63
    [epoch 18] loss: 0.44371 accuracy: 0.8461 val loss: 0.02421 val accuracy: 0.67
    53
    [epoch 19] loss: 0.42889 accuracy: 0.8526 val loss: 0.01690 val accuracy: 0.67
    70
    [epoch 20] loss: 0.41471 accuracy: 0.8556 val loss: 0.01876 val accuracy: 0.67
    75
    [epoch 21] loss: 0.40582 accuracy: 0.8571 val loss: 0.02280 val accuracy: 0.66
    68
    [epoch 22] loss: 0.39219 accuracy: 0.8633 val loss: 0.02153 val accuracy: 0.67
    72
    [epoch 23] loss: 0.38248 accuracy: 0.8666 val loss: 0.02586 val accuracy: 0.65
    63
    [epoch 24] loss: 0.36800 accuracy: 0.8702 val loss: 0.02170 val accuracy: 0.67
    83
    [epoch 25] loss: 0.36074 accuracy: 0.8737 val loss: 0.02186 val accuracy: 0.66
    53
    [epoch 26] loss: 0.35078 accuracy: 0.8778 val loss: 0.02234 val accuracy: 0.66
    82
    [epoch 27] loss: 0.33616 accuracy: 0.8822 val loss: 0.02359 val accuracy: 0.67
    37
    [epoch 28] loss: 0.32943 accuracy: 0.8840 val loss: 0.01999 val accuracy: 0.65
    82
    [epoch 29] loss: 0.31073 accuracy: 0.8918 val loss: 0.02598 val accuracy: 0.67
    17
    [epoch 30] loss: 0.30685 accuracy: 0.8927 val loss: 0.02987 val accuracy: 0.66
    47
```

```
[epoch 31] loss: 0.29580 accuracy: 0.8951 val loss: 0.03046 val accuracy: 0.66
02
[epoch 32] loss: 0.28529 accuracy: 0.9003 val loss: 0.02866 val accuracy: 0.66
44
[epoch 33] loss: 0.27396 accuracy: 0.9045 val loss: 0.02699 val accuracy: 0.66
95
[epoch 34] loss: 0.26255 accuracy: 0.9104 val loss: 0.03071 val accuracy: 0.66
27
[epoch 35] loss: 0.25470 accuracy: 0.9097 val loss: 0.02862 val accuracy: 0.66
79
[epoch 36] loss: 0.24751 accuracy: 0.9143 val loss: 0.03297 val accuracy: 0.65
96
[epoch 37] loss: 0.24387 accuracy: 0.9144 val loss: 0.01981 val accuracy: 0.65
20
[epoch 38] loss: 0.23275 accuracy: 0.9197 val loss: 0.03164 val accuracy: 0.65
88
[epoch 39] loss: 0.22010 accuracy: 0.9238 val loss: 0.02972 val accuracy: 0.65
68
[epoch 40] loss: 0.21311 accuracy: 0.9273 val loss: 0.03777 val accuracy: 0.65
67
[epoch 41] loss: 0.20612 accuracy: 0.9294 val loss: 0.02484 val accuracy: 0.66
03
[epoch 42] loss: 0.19155 accuracy: 0.9342 val loss: 0.02975 val accuracy: 0.65
91
[epoch 43] loss: 0.18921 accuracy: 0.9349 val loss: 0.03089 val accuracy: 0.65
20
[epoch 44] loss: 0.18019 accuracy: 0.9376 val loss: 0.03232 val accuracy: 0.65
83
[epoch 45] loss: 0.16975 accuracy: 0.9429 val loss: 0.02817 val accuracy: 0.65
52
[epoch 46] loss: 0.16541 accuracy: 0.9443 val loss: 0.04068 val accuracy: 0.65
83
[epoch 47] loss: 0.16469 accuracy: 0.9432 val loss: 0.04316 val accuracy: 0.66
15
[epoch 48] loss: 0.16330 accuracy: 0.9426 val loss: 0.03715 val accuracy: 0.65
68
[epoch 49] loss: 0.15491 accuracy: 0.9474 val loss: 0.03950 val accuracy: 0.65
33
[epoch 50] loss: 0.14930 accuracy: 0.9485 val loss: 0.03915 val accuracy: 0.64
62
```

Problem 5.2

In [ ]:

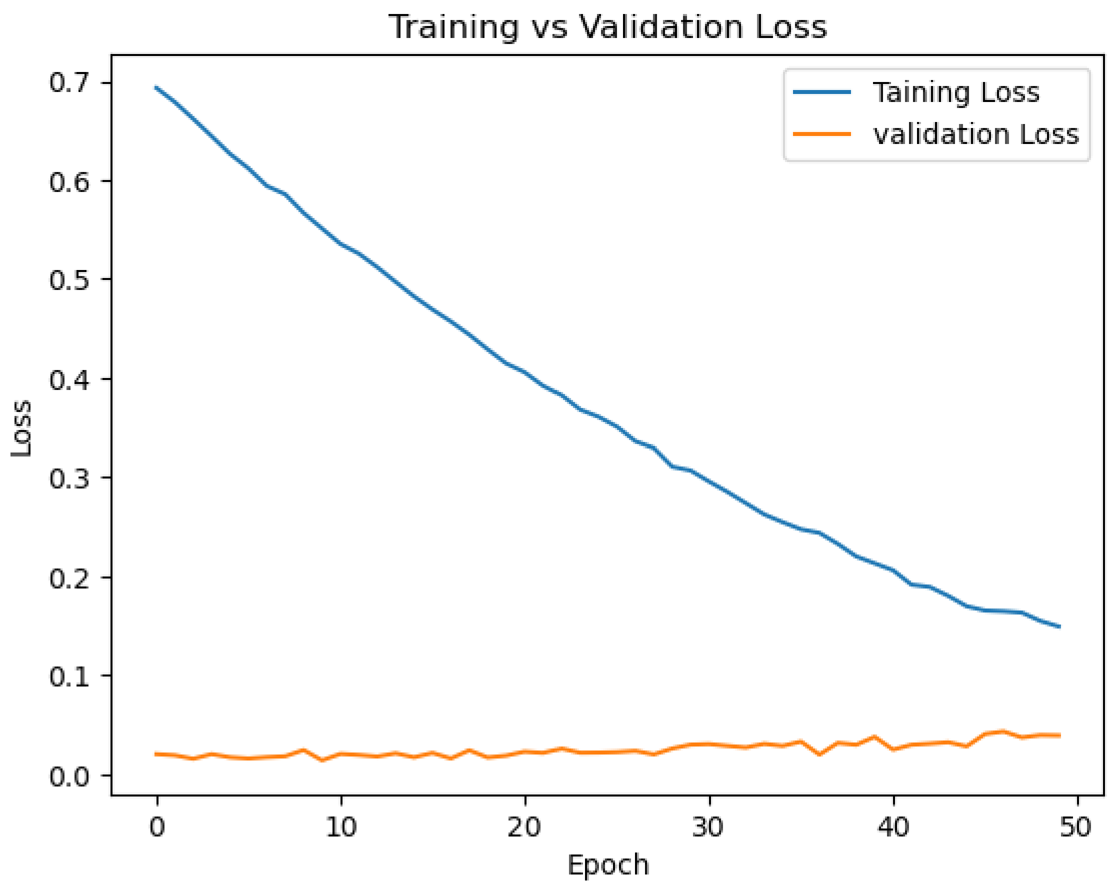
```
##TODO: Plot the training losses and validation losses
fig, ax = plt.subplots()

ax.plot(loss_list,label = "Taining Loss")
ax.plot(loss_list_val,label = "validation Loss")

ax.set_xlabel("Epoch")
ax.set_ylabel("Loss")
ax.set_title("Training vs Validation Loss")
ax.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7f38918caac0>
```

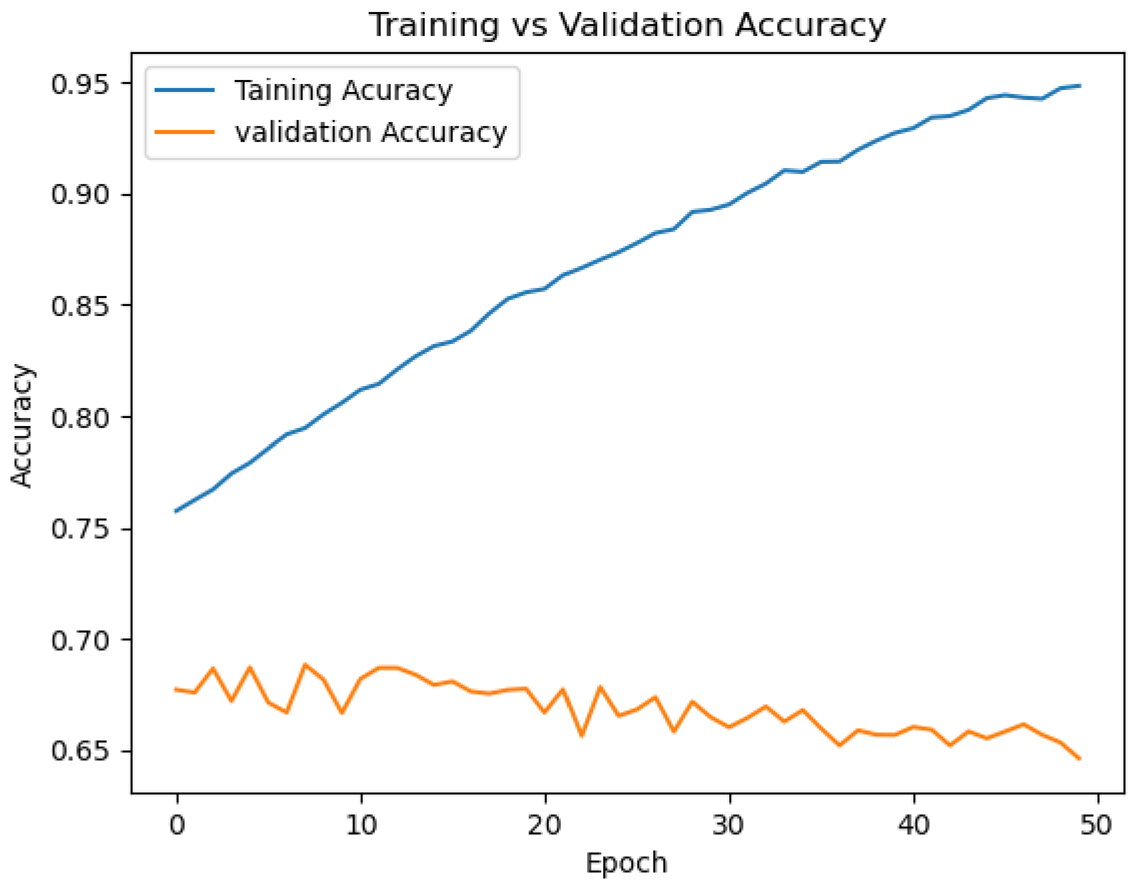


In [ ]:
```
##TODO: Plot the training accuracies and validation accuracies
fig, ax = plt.subplots()

ax.plot(acc_list,label = "Taining Acuracy")
ax.plot(acc_list_val,label = "validation Accuracy")

ax.set_xlabel("Epoch")
ax.set_ylabel("Accuracy")
ax.set_title("Training vs Validation Accuracy")
ax.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f38918a33a0>
```



Based on the plots, it is clear that the model is over fitting. It appears that the validation accuracy is remaining constant while the accuracy is improving. As such, it must be over fitting

5.3

In [ ]:

```
#Test
true_labels = []
predictions = []
correct_test = 0
net.eval()
with torch.no_grad():
    for batch_idx, (data, label) in enumerate(test_loader):
        data, label = data.to(device), label.to(device)
        ##TODO: pass the data into the network and store the output
        outputs = net(data)

        ##TODO: Get the prediction from the output
        _,predicted = torch.max(outputs,1)
        predicted_class = classes[predicted.item()]

        ##TODO: Calculate the correct number and add the number to correct_tes
        true_label = classes[label.item()]
        correct_test += (predicted == label).sum().item()

        ##TODO: update predictions list and true label list
        predictions.append(predicted_class)
        true_labels.append(true_label)
        ##We can directly append the value because here batch_size=1
```

```
print('Accuracy on the 10000 test images: %.2f %%' % (100 * correct_test / len
```

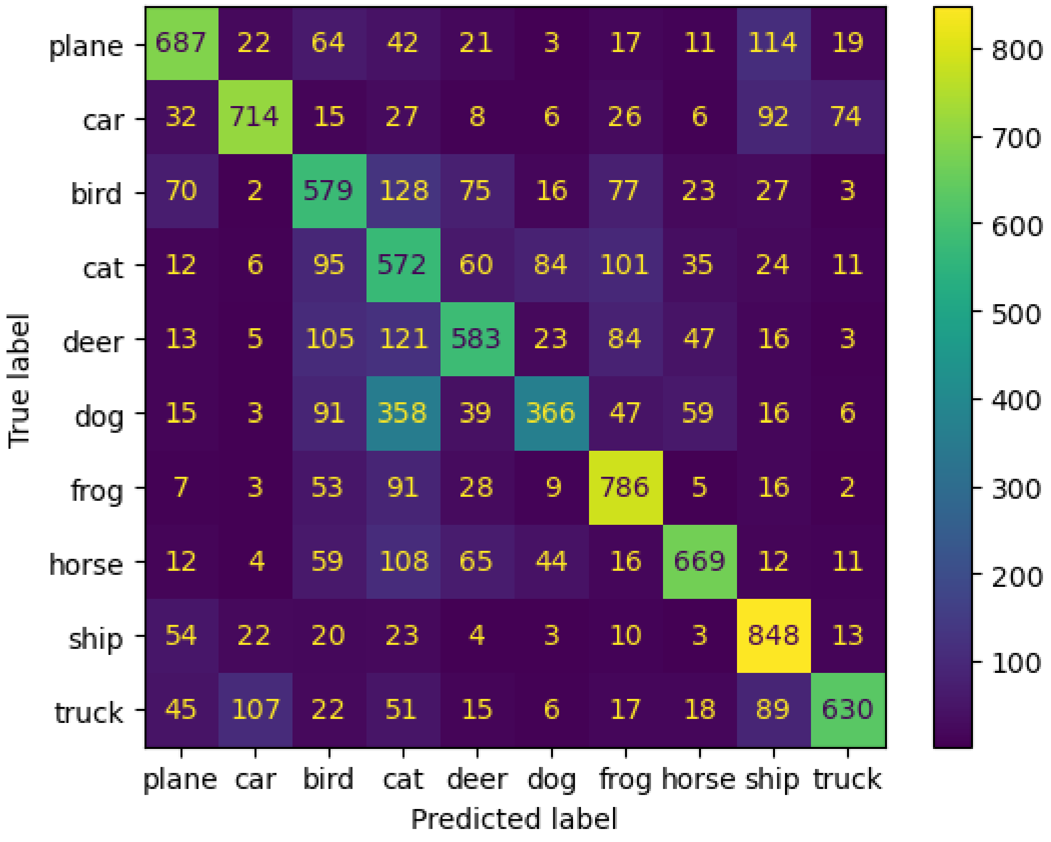
```
Accuracy on the 10000 test images: 64.34 %
```

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay
##TODO: print the confusion matrix of test set
##You can use sklearn.metrics.confusion_matrix

#confusion_matrix(true_labels,predictions,labels=classes)

ConfusionMatrixDisplay.from_predictions(true_labels,predictions,labels=classes
plt.show()
```



It appears that cat and dog were the two labels that were most confused. This makes sense as they look the most similar compared to all of the other classes