ECEN524 HW5
Aly Khater

Link to Notebook found here:
https://colab.research.google.com/drive/1F7wMRQXTgYMtsxdnaFuoOI9rMzZTSWl4?usp=sharing

1) Describe the model architecture and loss function you selected in order to improve the accuracy

The model utilizes two convolutional layers. The first layer transforms the 3 channel inputs to 32 features, which is followed by a batch normalization, activated by a ReLu, which is a rectified linear unit that practically assigns a value of 0 or 1 depending on the sign of the input. It is then followed by a max pooling of size 2x2, to reduce the map to a 16x16, then ended with a dropout to regularize the output. This process is repeated for the 2nd convolutional layer, except with the second block going up to 128 features. After the two convolutional layers, I flatten the dimensions to 512 neurons, which end up in an output layer of 10 classes.
The model utilizes the Adam optimizer (mainly because that's what I generally found the most success with in Deep Learning class and the hyperparameters I can test with). The optimizer has a learning rate of 0.001, betas of 0.9 and 0.999, and an epsilon of 1e-08. The loss function is Cross Entropy Loss, which was provided in the pytorch document, which applies a softmax activation to predict the image at the output.

2) Copy the code you implemented.

Credit to ChatGPT for providing the input/output sizes after each call

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class ImprovedNet(nn.Module):
    def __init__(self):
        super(ImprovedNet, self).__init__()
        # First convolutional block
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)  # 32x32
-> 32x32
        self.bn1   = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)  # 32x32
-> 32x32
        self.bn2   = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(2, 2)                           # 32x32
-> 16x16
        self.drop1 = nn.Dropout(0.25)

        # Second convolutional block
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1) # 16x16
-> 16x16
        self.bn3   = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1) # 16x16
-> 16x16
        self.bn4   = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(2, 2)                           # 16x16
-> 8x8
        self.drop2 = nn.Dropout(0.25)

        # Fully connected block
        self.fc1   = nn.Linear(128 * 8 * 8, 512)
        self.bn_fc = nn.BatchNorm1d(512)
        self.drop_fc = nn.Dropout(0.5)
        self.fc2   = nn.Linear(512, 10)  # 10 output classes

    def forward(self, x):
        # Block 1
        x = F.relu(self.bn1(self.conv1(x)))
```

```
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool1(x)
        x = self.drop1(x)

        # Block 2
        x = F.relu(self.bn3(self.conv3(x)))
        x = F.relu(self.bn4(self.conv4(x)))
        x = self.pool2(x)
        x = self.drop2(x)

        # Flatten and Fully Connected Layers
        x = torch.flatten(x, 1)  # flatten all dimensions except batch
        x = F.relu(self.bn_fc(self.fc1(x)))
        x = self.drop_fc(x)
        x = self.fc2(x)
        return x
```

3) Provide screenshots from your computer showing that you run the code, including showing the results of the improved accuracy for the network and the per-class accuracy.

In Figure 1 and 2, the output with the white background is from the pytorch document found here: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. The output with the grey background is the output of the improved network.

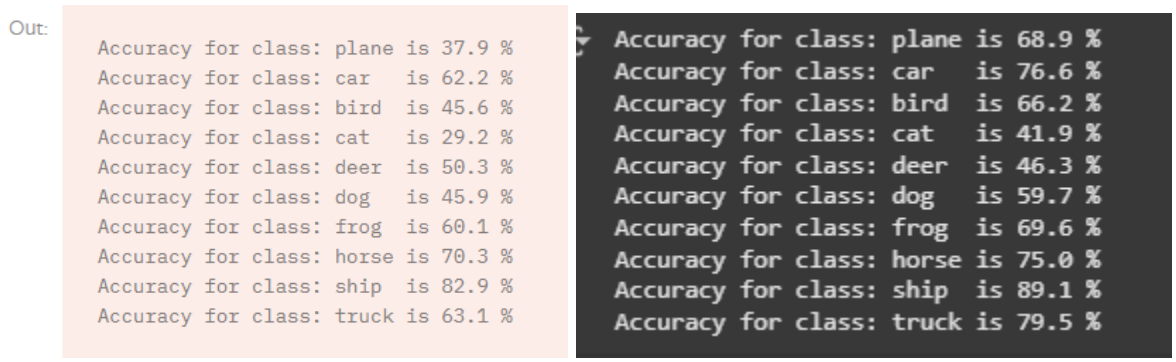

Figure 1: Accuracies per class

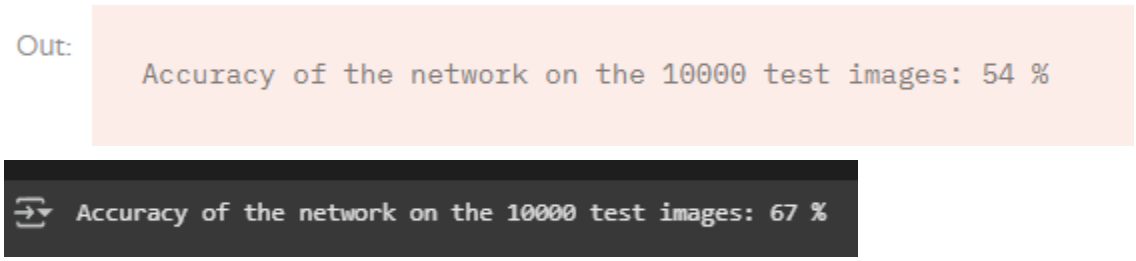

Figure 2: Total accuracy of the network