ECE 60146 - HW5

Bhavya Patel - pate1539@purdue.edu

02/18/25

1 Net 1

1.1 Code

```
#take in a parameter for which model to train and test
parser = argparse.ArgumentParser()
parser.add_argument("--model", type=int, required=True)
args = parser.parse_args()
#create the DL Studio parameters
dls = DLStudio(
            #dataroot = "/home/kak/ImageDatasets/CIFAR-10/",
            dataroot = "./data/CIFAR-10/",
            image_size = [32,32],
            path_saved_model = "./saved_model",
            momentum = 0.9,
            learning_rate = 1e-3,
            epochs = 5,
            batch_size = 5,
            classes = ('plane','car','bird','cat','deer','dog','frog','horse'
    ,'ship','truck'),
            use_gpu = True,
        )
#get nnew class object
custom_cifar = CustomExperimentsWithCIFAR(dl_studio = dls)
#get the CIFAR10 Dataset
custom_cifar.load_cifar_10_dataset()

#train and run Net 1
if args.model == 1:
    model1 = custom_cifar.Net()
    custom_cifar.run_code_for_training(model1, display_images=False)
    labels, predicted = custom_cifar.run_code_for_testing(model1,
    display_images=False)
                                    ...
                                    ...
                                    ...
#create a confusion matrix
class_names = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
     'ship', 'truck']
cm = confusion_matrix(labels, predicted)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
    yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'Confusion Matrix for Model {args.model}')
plt.savefig(f'confusion_matrix_{args.model}')
```

Listing 1: Net 1 Code

Note: The run_code_for_testing() method was modified to return labels and predictions to create the Confusion Matrix. The method otherwise was unmodified and gives the training loss graph.

1.2 Training Loss Graph and Confusion Matrix

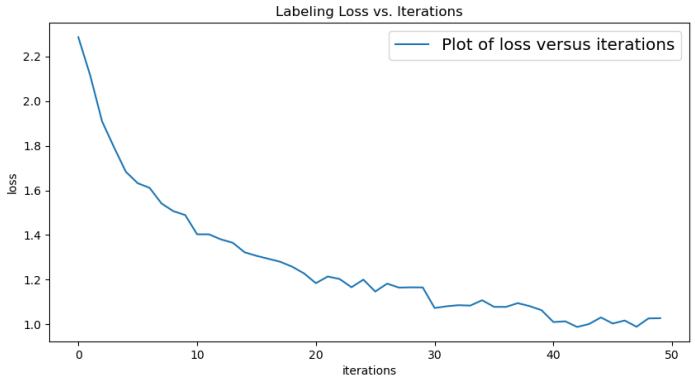

Figure 1: Training Loss

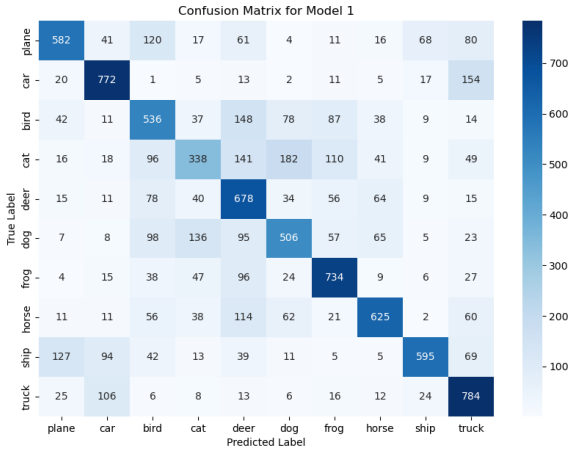

Figure 2: Confusion Matrix

2 Net 2

2.1 Code

```
#take in a parameter for which model to train and test
parser = argparse.ArgumentParser()
parser.add_argument("--model", type=int, required=True)
args = parser.parse_args()
#create the DL Studio parameters
dls = DLStudio(
            #dataroot = "/home/kak/ImageDatasets/CIFAR-10/",
            dataroot = "./data/CIFAR-10/",
            image_size = [32,32],
            path_saved_model = "./saved_model",
            momentum = 0.9,
            learning_rate = 1e-3,
            epochs = 5,
            batch_size = 5,
            classes = ('plane','car','bird','cat','deer','dog','frog','horse'
    ,'ship','truck'),
            use_gpu = True,
        )
#get nnew class object
custom_cifar = CustomExperimentsWithCIFAR(dl_studio = dls)
#get the CIFAR10 Dataset
custom_cifar.load_cifar_10_dataset()

                                    ...
elif args.model == 2:
    model2 = custom_cifar.Net2()
    custom_cifar.run_code_for_training(model2, display_images=False)
    labels, predicted = custom_cifar.run_code_for_testing(model2,
    display_images=False)
                                    ...
#create a confusion matrix
class_names = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
     'ship', 'truck']
cm = confusion_matrix(labels, predicted)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
    yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'Confusion Matrix for Model {args.model}')
plt.savefig(f'confusion_matrix_{args.model}')
```

Listing 2: Net 2 Code

2.2 Training Loss Graph and Confusion Matrix

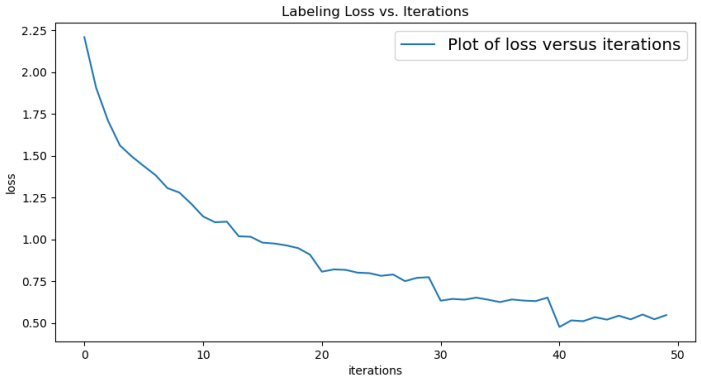

Figure 3: Training Loss

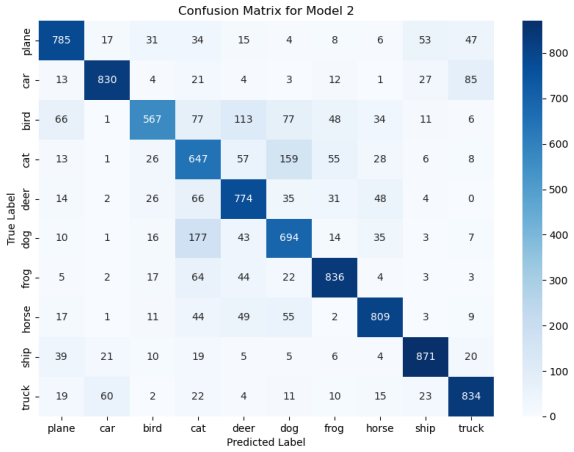

Figure 4: Confusion Matrix

3 Net 3

3.1 Code

```
class CustomExperimentsWithCIFAR(DLStudio.ExperimentsWithCIFAR):
    # pass dl_studio class to constructor
    def __init__(self, dl_studio):
        super().__init__(dl_studio)
    #create a new Network class
    class Net3(nn.Module):
        def __init__(self):
            super().__init__()
            #conv layers
            self.conv_block1 = nn.Sequential(nn.Conv2d(3, 32, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(32))
            self.conv_block2 = nn.Sequential(nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(64))
            self.conv_block3 = nn.Sequential(nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(128), nn.MaxPool2d(kernel_size=2, stride=2))
            self.conv_block4 = nn.Sequential(nn.Conv2d(128, 128, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(128))
            self.conv_block5 = nn.Sequential(nn.Conv2d(128, 256, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(256))
            self.conv_block6 = nn.Sequential(nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(256), nn.MaxPool2d(kernel_size=2, stride=2))
            self.conv_block7 = nn.Sequential(nn.Conv2d(256, 512, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(512))
            self.conv_block8 = nn.Sequential(nn.Conv2d(512, 512, kernel_size=3, padding=1), nn.ReLU(), nn.BatchNorm2d(512))
            #fully connected
            self.fc1 = nn.Linear(512*8*8, 512)
            self.fc2 = nn.Linear(512, 10)

        def forward(self, x):
            #conv layers
            x1 = self.conv_block1(x)
            x2 = self.conv_block2(x1)
            x3 = self.conv_block3(x2)
            x4 = self.conv_block4(x3)
            x5 = self.conv_block5(x4)
            x6 = self.conv_block6(x5)
            x7 = self.conv_block7(x6)
            x8 = self.conv_block8(x7)

            #fully connected
            x8 = x8.view(x8.size(0), -1)
            x8 = F.relu(self.fc1(x8))
            x8 = self.fc2(x8)
            return x8
```

Listing 3: Net 3 Architecture

```
#take in a parameter for which model to train and test
parser = argparse.ArgumentParser()
parser.add_argument("--model", type=int, required=True)
args = parser.parse_args()
#create the DL Studio parameters
dls = DLStudio(
            #dataroot = "/home/kak/ImageDatasets/CIFAR-10/",
            dataroot = "./data/CIFAR-10/",
            image_size = [32,32],
            path_saved_model = "./saved_model",
            momentum = 0.9,
            learning_rate = 1e-3,
            epochs = 5,
            batch_size = 5,
            classes = ('plane','car','bird','cat','deer','dog','frog','horse'
    ,'ship','truck'),
            use_gpu = True,
        )
#get nnew class object
custom_cifar = CustomExperimentsWithCIFAR(dl_studio = dls)
#get the CIFAR10 Dataset
custom_cifar.load_cifar_10_dataset()

                                    ...
elif args.model == 3:
    model3 = custom_cifar.Net3()
    custom_cifar.run_code_for_training(model3, display_images=False)
    labels, predicted = custom_cifar.run_code_for_testing(model3,
     display_images=False)
                                    ...
#create a confusion matrix
class_names = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
      'ship', 'truck']
cm = confusion_matrix(labels, predicted)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names,
     yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title(f'Confusion Matrix for Model {args.model}')
plt.savefig(f'confusion_matrix_{args.model}')
```

Listing 4: Net 3 Main Function Code

3.2 Training Loss Graph and Confusion Matrix

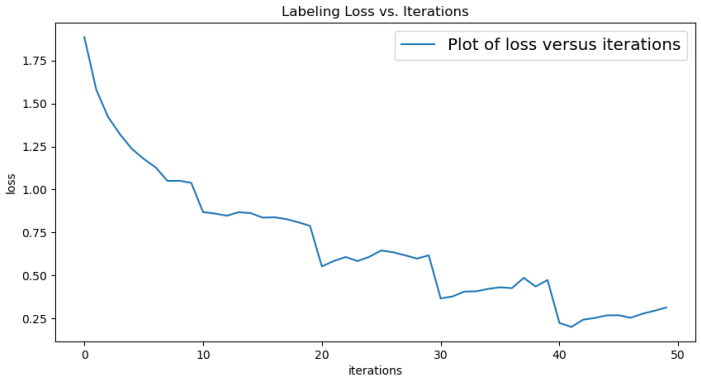

Figure 5: Training Loss

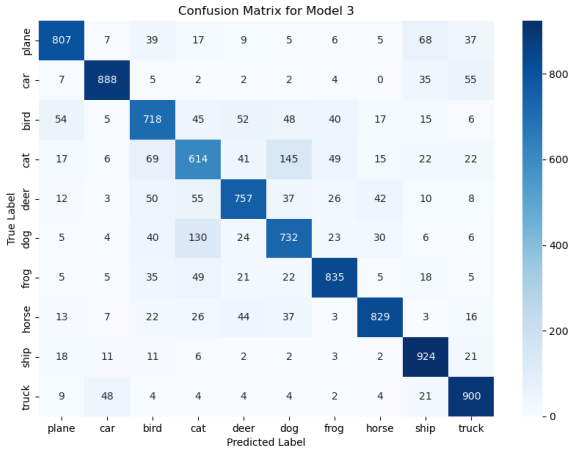

Figure 6: Confusion Matrix

4 Results

4.1 Overall Accuracy Table

Network	Overall Accuracy (%)
Net1 1	61
Net2	76
Net3	80

Table 1: Overall classification accuracy of each network.

4.2 Per-Class Accuracy Table

Class	Network 1 (%)	Network 2 (%)	Network 3 (%)
Plane	58	78	80
Car	77	83	88
Bird	53	56	71
Cat	33	64	61
Deer	67	77	75
Dog	50	69	73
Frog	73	83	83
Horse	62	80	82
Ship	59	87	92
Truck	78	83	90

Table 2: Per-class classification accuracy for each network.

4.3 Discussion

The classification accuracy of the three neural networks show the trend that as the architecture becomes deeper the performance improves. Network 1, the simplest model, achieves an accuracy of 61%. It consists of only two convolutional layers followed by three fully connected layers. Then Network 2 introduces an additional convolutional layer and increases the number of channels in each layer, which leads to a accuracy jump to 76%, which is a 15% improvement over Network 1. This shows the benefits of added depth and increased channel capacity, which is it enhances feature extraction.

Network 3, the most complex architecture, achieves the highest accuracy of 80%. It has eight convolutional layers, each followed by batch normalization and then there is a max-pooling twice in the entire architecture. The deeper architecture allows more feature learning which is crucial for object recognition. A key factor in its success is batch normalization, which stabilizes training, limits the vanishing gradient problem, and accelerates convergence. Without batch normalization Netowrk 3 struggled to learn and its performance was worse than Network 1 and Network 2.

Some things to note about each networks graphs and confusion matrix is they all tend to struggle with the same object classifications. For example when looking at the confusion matrices for all the networks we can see that it always struggles with differentiating dog and cat. We can also see that with a deeper network that its learns much quicker than the shallower network by using the training loss graphs.

The overall trend indicates that increasing network depth enhances accuracy. However, deeper networks require greater computational resources and are more prone to overfitting. The 15% accuracy boost from Network 1 to Network 2 shows gains from increasing model capacity, while the smaller 4% gain from Network 2 to Network 3 suggests diminishing returns beyond a certain depth. Adding layers can only provide only so many benefits. In conclusion, while deeper networks generally improve classification performance, techniques like batch normalization are essential for ensuring stable and efficient learning.