Final Project
Convolutional Neural Network for Image Classification

Zhenyu Bu[1*] and Yijing Liu[2*]

[1] Department of Biomedical Engineering, The Ohio State University
[2] Department of Electrical and Computer Engineering, The Ohio State University

{bu.123, liu.11713}@osu.edu

1 Question 1

We have explored various methods to enhance accuracy, including data augmentation and experimenting with different networks. In this section, we will demonstrate how the modifications incrementally improved accuracy.

1.1 Network

The proposed multi-layer network is overly simplistic and insufficient for effective image classification. Running the original network directly yields only a 61.88% testing accuracy after 50 epochs.

To address this, our proposed network (Figure 1) builds upon a residual design inspired by ResNet, integrating four sequential layers of convolutional blocks to efficiently extract hierarchical features. Each block incorporates two 3x3 convolutional layers with Batch Normalization and ReLU activation, alongside a shortcut path to preserve input information and enhance gradient flow. The network progressively reduces spatial dimensions through strided convolutions in the second, third, and fourth layers, facilitating effective downsampling. A global adaptive average pooling layer further reduces the spatial dimensions to 1x1, enabling a fully connected layer to map the extracted features to the final output classes (10 classes). This design ensures a balance between computational efficiency and model performance, making it well-suited for classification tasks on our CIFAR10 dataset.

After 50 epochs training, the final testing accuracy is 76.34%.

1.2 Data augmentation

Random horizontal flipping and random cropping are utilized for data augmentation. The implementation details are as follows:

[*] Equal contribution.

```python
class Block(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(Block, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(out_channels)
        )

    def forward(self, x):
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x) # resnet design
        out = self.relu(out)
        return out

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layer1 = Block(3, 64, stride=1)
        self.layer2 = Block(64, 128, stride=2)
        self.layer3 = Block(128, 256, stride=2)
        self.layer4 = Block(256, 512, stride=2)

        self.pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, 10)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

Fig. 1. Our proposed network.

```
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(norm_mean, norm_std)
])
```

The testing accuracy increased significantly from 76.34% to 83.48% after incorporating the data agmentation technology, demonstrating its effectiveness in enhancing the model's performance and ability in testing data.

2 Question 2

2.1 Learning rate and momentum

We carried out extensive testing using different learning rates, such as 0.1, 0.01, and 0.001, alongside momentum values of 0.8 and 0.9.

Momentum demonstrated its effectiveness in speeding up convergence by utilizing past gradients. From our experiments, we determined that a momentum value of 0.9 and a learning rate of 0.001 yielded optimal results. Ultimately, the model achieved its best performance with a learning rate of 0.001 and momentum set to 0.9.

Learning rate	Accuracy ↑
0.1	82.11%
0.01	83.35%
0.001	83.48%
Momentum	**Accuracy ↑**
0.8	83.27%
0.9	84.25%

2.2 L2 Regularization

After incorporating a momentum value of 0.9, we proceed to add L2 regularization to our model. In PyTorch, the 'weight_decay' parameter represents L2 regularization. We select a 'weight_decay' of 5e-4. The implementation is as follows:

```
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum,
weight_decay=weight_decay)
```

After adding the L2 regularization, the final testing accuracy is 86.85%.

3 Question 3: Results

In this section, we tune more parameters (such as more epochs for training, bigger batchsize and MultiStepLR strategies) in the code and make the improvement. Here, we use wandb to monitor the training process. Our finally average accuracy for the 10 classes is 88.55%.

Also, in this section, we show the confusion matrix, the visualization results for the highest and the lowest scores and the comparative results.

The following table presents the accuracy for each class, along with a comparison between our final results and those from Project 4. As illustrated in the table, our new model outperforms the previous one across all classes. Notably, we achieved the highest accuracy of 94% for the car class. Additionally, the model attained an accuracy of 92% for both ship and truck classes.

Class	Final (Acc ↑)	Project4 (Acc ↑)	Improvement (Acc ↑)
plane	91%	38%	53%
car	94%	41%	53%
bird	81%	21%	60%
cat	78%	26%	52%
deer	89%	28%	61%
dog	83%	26%	57%
frog	91%	41%	50%
horse	91%	34%	57%
ship	92%	57%	35%
truck	92%	44%	48%

Next, we present the confusion matrix. Each class includes a total of 1,000 test images. The diagonal elements of the matrix indicate the number of correctly classified samples out of the 1,000 images. Additionally, darker colors correspond to higher accuracy. In Figure 2, the color block in the second row (car) is the darkest, signifying the highest accuracy. Specifically, the correct count for this class is 941, indicating that 941 out of 1,000 test samples were correctly classified.

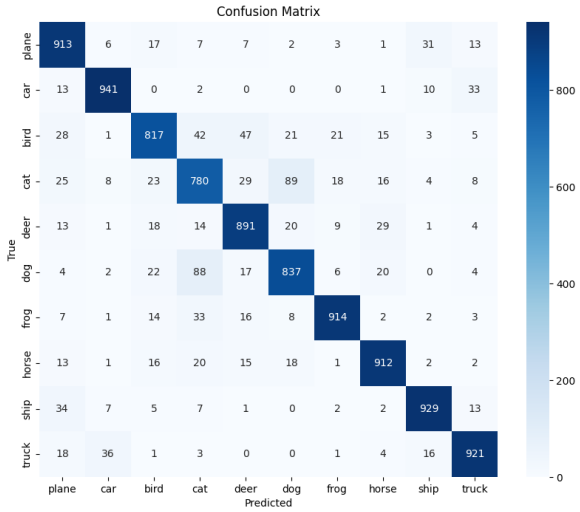

Fig. 2. The confusion matrix for the final classification results

Similar to the visualization in Project 4, we also perform visualization for the highest and lowest scores in each class. However, due to the poor performance of the model in Project 4, the visualization results in its figure were entirely incorrect. In contrast, for our final project, the model is well-trained, clearly

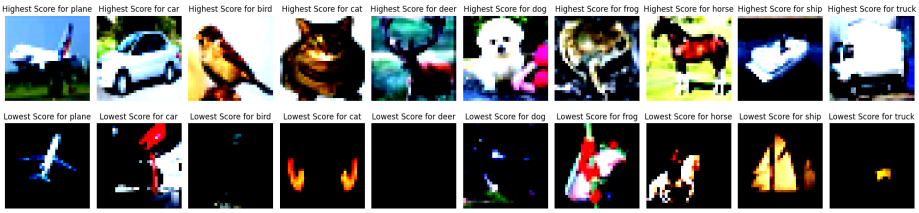

Fig. 3. The visualization results for the highest and the lowest scores.

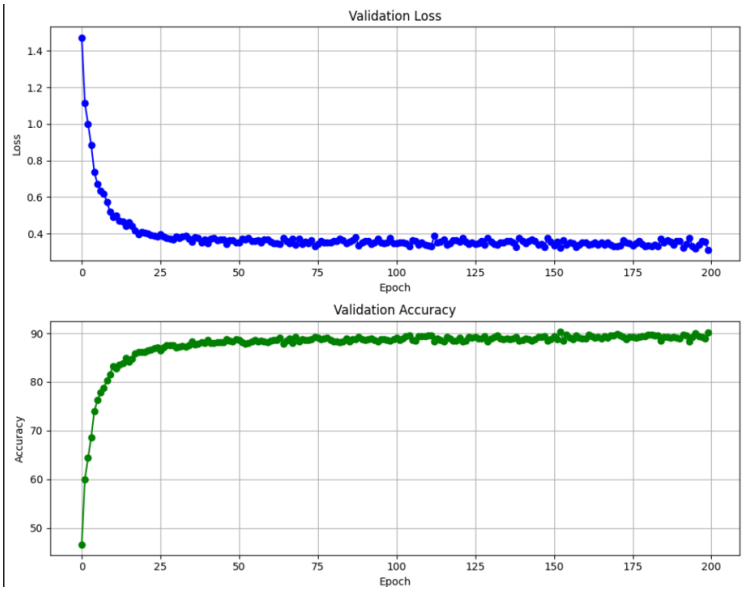

Fig. 4. The validation loss and accuracy.

demonstrating that the images with the highest scores accurately correspond to their respective classes.

In Figure 4, we present both the validation loss and accuracy to demonstrate that our model is well-trained and capable of achieving optimal performance in this context.

For further details, please refer to our code in ece5460FinalProject.ipynb.