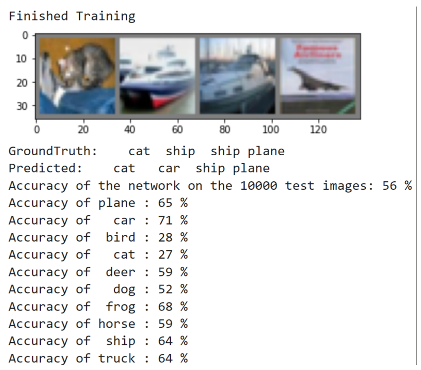
COMP 5212 HA3

Leung Pak Hei 20690382

March 2023

1 Introduction

Default Baseline performance: The default output with nn.Conv2d(3, 6, 5), nn.MaxPool2d(2, 2), nn.Conv2d(6, 16, 5), nn.Linear(16 * 5 * 5, 120), nn.Linear(120, 84), nn.Linear(84, 10)
The result of the default classifier is shown as follows:



```
Finished Training
GroundTruth:    cat  ship  ship plane
Predicted:    cat   car  ship plane
Accuracy of the network on the 10000 test images: 56 %
Accuracy of plane : 65 %
Accuracy of   car : 71 %
Accuracy of  bird : 28 %
Accuracy of   cat : 27 %
Accuracy of  deer : 59 %
Accuracy of   dog : 52 %
Accuracy of  frog : 68 %
Accuracy of horse : 59 %
Accuracy of  ship : 64 %
Accuracy of truck : 64 %
```

Figure 1: Benchmark accuracy

The baseline accuracy is 0.56.
In the following, different similar neural network structures will be compared using different parameters such as kernel size, learning rate, batch normalization, and optimizer.

2 Comparison using different numbers of hidden layers:

This section would explore the accuracy of the prediction using different numbers of hidden layers in the same epoch with a learning rate =0.01. The optimizer is SGD. For easy implementation, padding is added to some cases for fitting the kernel sizes. The baseline neural network structure is Linear(16 * 5 * 5, 120), Linear(120, 84), Linear(84, 64),Linear(64,10). The result is as follows

Comparison using different numbers of hidden layers (others hold equal)	
Neural Network Structure (different number of fully connected layers)	Accuracy
linear (Baseline structure)	0.55
Linear(16 * 5 * 5, 120), Linear(120, 84), Linear(84, 64),Linear(64, 32), Linear(32, 10)	0.53
Linear(16 * 5 * 5, 120), Linear(120, 84), Linear(84, 64),Linear(64, 32), Linear(32,16) , Linear(16, 10)	0.45
Linear(16 * 5 * 5, 120), Linear(120, 84), Linear(84, 64),Linear(64, 32), Linear(32,16) , Linear (16,8) Linear(8, 10)	0.42

From the above experiments, in general, simply adding more fully convolutional layers does not always lead to better performance. Increasing the depth of a neural network can also make it more difficult to train, especially if the network becomes too deep. This is because deeper networks can suffer from the vanishing gradient problem, where the gradients used to update the weights of earlier layers become very small, making it difficult to learn good representations.
Therefore, when increasing the number of hidden layers, it is important to monitor the training process carefully and adjust the architecture as needed to prevent overfitting or underfitting, and ensure that the network is able to converge to a good solution.
Additionally, increasing the number of hidden layers also comes at the cost of increased computational complexity and longer training times, so it is important to consider these trade-offs when designing a neural network architecture.

3 Comparison using Different numbers of filters

Comparison using different numbers of filter size (others hold equal)	
Neural Network Structure (Convolution layers) + same fully connected layers	Accuracy
Conv2d(3, 1, 5), MaxPool2d(2, 2), Conv2d(1, 3, 5)	0.36
Conv2d(3, 3, 5), MaxPool2d(2, 2), Conv2d(3, 6, 5)	0.45
Conv2d(3, 6, 5), MaxPool2d(2, 2), Conv2d(6, 12, 5) (Baseline structure)	0.56
Conv2d(3, 30, 5), MaxPool2d(2, 2), Conv2d(30, 65, 5)	0.64
Conv2d(3, 64, 5), MaxPool2d(2, 2), Conv2d(64, 128, 5)	0.66

From the above experiments, more numbers of filters tend to have better accuracy.

4 Comparison using Different numbers of learning rate

This section would explore the accuracy of the prediction using different learning rate in the same epoch. The default neural network is Conv2d(3, 6, 5), MaxPool2d(2, 2), Conv2d(6, 16, 5), FC(400, 120),FC(120, 84), FC(84, 10) (Baseline structure). The result is as follows:

Comparison using different learning rate (others hold equal)	
Learning rate + same hidden layers	Accuracy
0.001	0.54
0.005	0.48

The above results show that the accuracy tends to decrease with a larger learning rate.

Comparison using different learning rates (rate decay and cosine scheduling) (others hold equal)	
Learning rate + same hidden layers	Accuracy
rate decay (learning rate = 0.001, step size=10, gamma=0.1)	0.57
cosine scheduling(learning rate =0.001, T max=100))	0.51

The rate decay scheduler performs better in this task, with the above parameters.

5 Comparison using Different optimizers

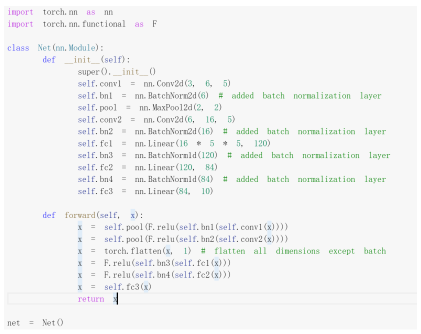
This section would explore the accuracy of the prediction using different optimizers in the same epoch with a learning rate =0.001. The default neural network is Conv2d(3, 6, 5), MaxPool2d(2, 2), Conv2d(6, 16, 5), FC(480, 120),FC(120, 84), FC(84, 10) (Baseline structure). The result is as follows:

Comparison using different optimizer (others hold equal)	
Optimizers	Accuracy
SGD (Default baseline)	0.41
Adagrad	0.35
Adam	0.58

From the above, it is shown that Adam performs the best in our case

6 Comparison using batch normalization

This section would explore the effect of batch normalization on the prediction accuracy versus the benchmark accuracy as the following neural network structure:



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

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.bn1 = nn.BatchNorm2d(6) # added batch normalization layer
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.bn2 = nn.BatchNorm2d(16) # added batch normalization layer
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.bn3 = nn.BatchNorm1d(120) # added batch normalization layer
        self.fc2 = nn.Linear(120, 84)
        self.bn4 = nn.BatchNorm1d(84) # added batch normalization layer
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.bn3(self.fc1(x)))
        x = F.relu(self.bn4(self.fc2(x)))
        x = self.fc3(x)
        return x

net = Net()
```

Figure 2: Batch Normalization

The accuracy is 0.48 which does not improve much from the baseline accuracy. This may be attributed to the fact that batch normalization may not be necessary or may not have a significant impact on the performance of the network. For example, in shallow networks or networks with few learnable parameters, batch normalization may not be needed. Additionally, batch normalization can add extra computational overhead, so it may not always be practical in resource-constrained environments. Of course, other batch normalization may improve using other parameters or layers.