CSE 532 Final Project

Submitted by: ahasanm

Initial Work:

In homework 6 I have implemented a network with below parameter with below convolution network and the accuracy I achieved was **77%**.

Parameter:

Total_layer	8
batch_size	128
epochs	100
learning_rate	0.01
optimizer	SGD
momentum	0.9
weight_decay	5.00E-04
Accuracy	77%

Convolution Network:

Layers	input_channel	output_channel	kernel/filter	stride	padding
Conv1	3	8	11	1	5
relu					
Conv2	8	16	7	1	3
relu					
Conv3	16	16	5	1	2
relu					
Conv4	16	16	5	1	0
relu					
maxpool			2		
Conv5	16	16	5	1	0
relu					
maxpool			2		
Conv6	16	160	5	1	0
relu					
Fc1	160	160			
Fc2	160	10			

Final Project Iteration #1

I have introduced multiple factor in the final project concerning HW6. The list is:

1. A modified convolution network with 3 convolution layer and 5 fully connected layer compared to 6 convolution layer and 2 fully connected layer in HW6.
2. Introduced batchnormalization and dropout after convolution layer and fully connected layer. The value of dropout layer is fixed to 0.2.
3. The batch size and epoch are kept the same as in HW6, but I have changed the learning rate 0.001 and changed the optimizer to "AdamW" compared to SGD in HW6.

The details parameter of Iteration #1 is, and the network architecture is below. However, the accuracy I achieved was **86.18%.**

Parameter:

Total_layer	8
batch_size	128
epochs	100
learning_rate	0.001
optimizer	AdamW
Accuracy	86.18%

Network Details:

Layers	input_channel	output_channel	kernel/filter	stride	padding	Value
Conv1	3	32	5	0	2	
bacthnormalization		32				
relu						
maxpool		2				
Conv2	32	64	5	0	2	
bacthnormalization		64				
relu						
maxpool		2				0.2
droupout						
Conv3	64	128	5	0	2	
bacthnormalization		128				
relu						
maxpool		2				
droupout						0.2
Fc1	2048	1000				
bacthnormalization		1000				
relu						
droupout						0.2
Fc2	1000	1000				
bacthnormalization		1000				
relu						
droupout						0.2
Fc3	1000	1000				
bacthnormalization		1000				
relu						
droupout						0.2
Fc4	1000	1000				
bacthnormalization		1000				
relu						
droupout						0.2
Fc5	1000	10				
bacthnormalization		10				

Final Project Iteration #2

In this iteration I have introduced a stable CNN architecture, different data transformation and those are:

Network:

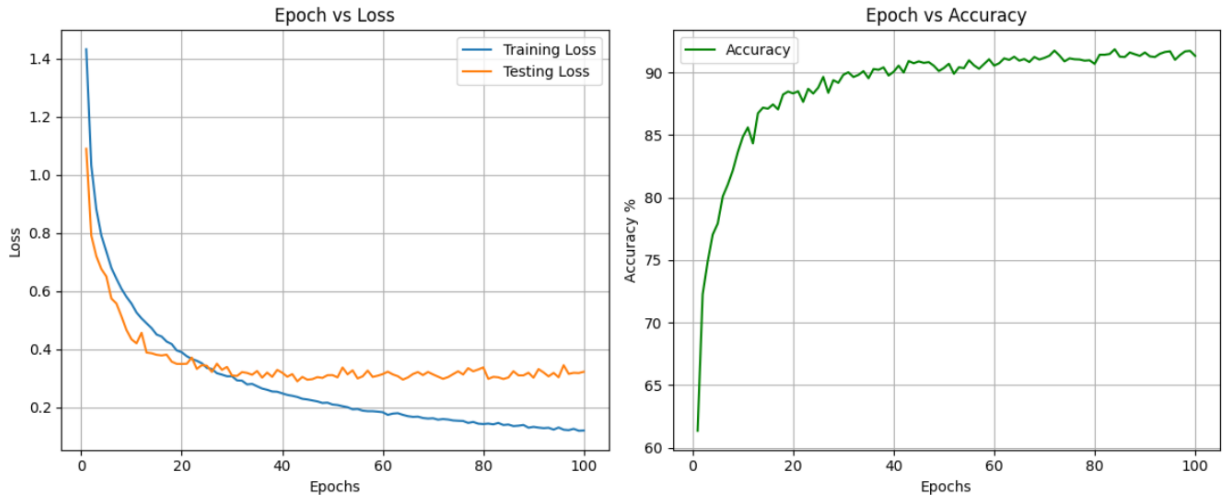
1. The are total 5 convolution layer and 4 fully connected layer making the total 9 layer
2. The network has five convolutional layers with varying numbers of filters (64, 64, 128, 128, 256). Each layer has a consistent kernel size of 3 and padding 1.
3. Introduced a better batch normalization. I have used batchnormalization2d after the convolution layers and batchnormalization1d after the fully connected layer except the last one.
4. To reduce image spatial dimension used the maxpooling after second, fourth and fifth convolution layer.
5. Increased the dropout rate to 0.5 and applied it after each fully connected layer except the last one.
6. In the fully connected layers, the dimension decreases from (2048,1024,512,10)
7. Though AdamW optimizer has a default weight decay, I fixed the weight decay to 1e-4.

Image Transformation:

1. Introduce RandomCrop that helps models to focus on different parts of the image. The cropped image size is 32x32 pixels with padding of 4 pixel.
2. Used RandomHorizentalFlip with default probability of 50%.This helps to recognize the image irrespective to its orientation.
3. Use RandomRotation with value 15 that rotates the images uniformly between -15 to 15. This increases the robustness of the images.
4. Used ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1) which randomly changes the brightness, saturation, contrast, and hue of the image. This augmentation helps the model to perform better when there is a variation of lighting.

Implementing the above I have finally achieved **91.32%** accuracy.

Graphical Presentation:



Parameter:

Total_layer	9
batch_size	128
epochs	100
learning_rate	0.001
optimizer	AdamW (1e-4)
Accuracy	91.32%

Network Details:

```
class LeNet(nn.Module):
    def __init__(self, number_of_classes=10):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(128)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.conv5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn5 = nn.BatchNorm2d(256)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(256 * 4 * 4, 2048)
        self.fc_bn1 = nn.BatchNorm1d(2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.fc_bn2 = nn.BatchNorm1d(1024)
        self.fc3 = nn.Linear(1024, 512)
        self.fc_bn3 = nn.BatchNorm1d(512)
        self.fc4 = nn.Linear(512, number_of_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool1(F.relu(self.bn2(self.conv2(x))))

        x = F.relu(self.bn3(self.conv3(x)))
        x = self.pool2(F.relu(self.bn4(self.conv4(x))))
        x = self.pool3(F.relu(self.bn5(self.conv5(x))))

        x = x.view(-1, 256 * 4 * 4)
```

```
        x = F.relu(self.fc_bn1(self.fc1(x)))
        x = self.dropout(x)
        x = F.relu(self.fc_bn2(self.fc2(x)))
        x = self.dropout(x)
        x = F.relu(self.fc_bn3(self.fc3(x)))
        x = self.fc4(x)

        return x
```