Homework4 Part2

Incorporating a convolutional neural network (CNN) into the neural network framework amplifies its capacity to comprehend and scrutinize images.

Within our model, CNN layers are introduced through an initial application of filters to input images, aiding in the detection of crucial patterns.

Subsequently, pooling layers condense feature sizes while retaining their fundamental information.

The flattened output then progresses to conventional fully connected layers for classification.

This sequential arrangement enables CNN to proficiently grasp and discern patterns within images, thereby enhancing the model's precision in tasks such as image classification. The ensuing observations were as follows:

Initial

Fully connected layers: 2

CNN: 0

Activation functions: ReLu, Softmax

Epochs: 1

Test Accuracy: 94 %

Average loss: 0.1955

```
Train Epoch: 1 [57280/60000 (95%)]      Loss: 0.097395
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.118450
Train Epoch: 1 [57920/60000 (97%)]      Loss: 0.390881
Train Epoch: 1 [58240/60000 (97%)]      Loss: 0.175111
Train Epoch: 1 [58560/60000 (98%)]      Loss: 0.344278
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.318049
Train Epoch: 1 [59200/60000 (99%)]      Loss: 0.399491
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.482808
Train Epoch: 1 [59840/60000 (100%)]     Loss: 0.500538
/usr/local/lib/python3.10/dist-packages/torch/nn/_reduction.py:42: UserWar
  warnings.warn(warning.format(ret))

Test set: Average loss: 0.1955, Accuracy: 9441/10000 (94%)
```

With added convolutional layers

Fully connected layers: 2

CNN and max pooling: 2

Activation functions: ReLu, Softmax

Learning rate: 0.01

Epochs: 5

Test Accuracy: 99 %

Average loss: 0.0259

```
def __init__(self):
    super(Net, self).__init__()
    # First Convolutional Layer
    self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
    # Second Convolutional Layer
    self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
    # Fully connected layers
    self.fc1 = nn.Linear(64 * 7 * 7, 256)
    self.fc2 = nn.Linear(256, 10)
```

```
args['epochs'] = 5

model = Net()
if args['cuda']:
    model.cuda()

optimizer = optim.SGD(model.parameters(), lr=args['lr'], momentum=args['momentum'])

for epoch in range(1, args['epochs'] + 1):
    train(epoch)
    test()
```

```
Train Epoch: 5 [58240/60000 (97%)]      Loss: 0.000243
Train Epoch: 5 [58560/60000 (98%)]      Loss: 0.016527
Train Epoch: 5 [58880/60000 (98%)]      Loss: 0.017304
Train Epoch: 5 [59200/60000 (99%)]      Loss: 0.054130
Train Epoch: 5 [59520/60000 (99%)]      Loss: 0.017126
Train Epoch: 5 [59840/60000 (100%)]     Loss: 0.007116

Test set: Average loss: 0.0259, Accuracy: 9903/10000 (99%)
```

Parameter	Value
Fully connected layers	2
CNN and max pooling	2
Activation functions	ReLu, Softmax
Learning rate	0.01
Epochs	5
Test Accuracy	99%
Average loss	0.0259

Parameter	Value
Fully connected layers	2
CNN	0
Activation functions	ReLu, Softmax
Epochs	1
Test Accuracy	94%
Average loss	0.1955

Batch Size	Accuracy
32	0.98
32	0.98

64	0.99
64	0.98

The introduction of a convolutional neural network (CNN) evidently boosted the model's efficacy. In the absence of CNN, the accuracy on the test set stood at 94%, whereas with CNN, accuracy surged to 99%. This showcases CNN’s effectiveness in extracting and assimilating features from image data, thus fostering improved classification accuracy. The reduced average loss with CNN suggests that the model's predictions align more closely with the ground truth labels, reinforcing the premise that CNNs augment the model's capacity to comprehend and assess image data.