1. 1) Zero padding

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	2	0	2	1	3	2	0
0	0	2	0	2	2	2	0
0	1	0	1	3	1	1	0
0	0	0	1	1	1	0	0
0	0	1	3	4	1	0	0
0	0	1	0	0	5	2	0
0	0	0	0	0	0	0	0

2) Filter A

4	**0**	**4**	**3**	**5**	**2**
0	5	3	5	4	5
1	1	4	4	3	3
1	4	5	3	4	1
1	1	3	10	4	1
0	1	1	3	9	3

3) Maxpooling

5	**5**	**5**
4	5	4
1	10	9

2. Before modifying.

```
total_batch = len(data_loader)

for epoch in range(training_epochs):
    avg_cost = 0
    for X, Y in data_loader:
        optimizer.zero_grad()
```

```
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()
        avg_cost += cost/total_batch
    print('[Epoch:{}] cost = {}'.format(epoch+1,avg_cost))
print('Learning Finished!')
```

```
[Epoch:1] cost = 0.05995052307844162
[Epoch:2] cost = 0.04381060600280762
[Epoch:3] cost = 0.03494998812675476
[Epoch:4] cost = 0.02915232814848423
[Epoch:5] cost = 0.02359003946185112
[Epoch:6] cost = 0.019573427736759186
[Epoch:7] cost = 0.016565781086683273
[Epoch:8] cost = 0.014849044382572174
[Epoch:9] cost = 0.01228309702128172
[Epoch:10] cost = 0.01083854865282774
[Epoch:11] cost = 0.008132913149893284
[Epoch:12] cost = 0.007936287671327591
[Epoch:13] cost = 0.005904927384108305
[Epoch:14] cost = 0.006556585896760225
[Epoch:15] cost = 0.005575613118708134
Learning Finished!
```

```
with torch.no_grad():
    X_test = mnist_test.test_data.view(len(mnist_test),1,28,28).float()
    Y_test = mnist_test.test_labels

    prediction = model(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.9857000112533569
```

After modifying

```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.init
```

```
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

```
mnist_train = dsets.MNIST(root = 'MNIST_data/',train=True, transform =
transforms.ToTensor(),download = True)
mnist_test = dsets.MNIST(root = 'MNIST_data/',train=False, transform =
transforms.ToTensor(),download = True)
```

```
data_loader = torch.utils.data.DataLoader(dataset = mnist_train, batch_size=
batch_size, shuffle = True, drop_last = True)
```

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(1,32,kernel_size=4, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(32,64,kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.layer3 = nn.Sequential(
            nn.Conv2d(64,128,kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc1 = nn.Linear(1*1*128, 625, bias = True)
        nn.init.xavier_uniform_(self.fc1.weight)

        self.keep_prob = 0.5

        self.layer4 = nn.Sequential(
            self.fc1,
            nn.ReLU(),
            nn.Dropout(p= 1 - self.keep_prob))

        self.fc2 = nn.Linear(625, 10, bias = True)
        nn.init.xavier_uniform_(self.fc2.weight)

    def forward(self,x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0),-1)
        out = self.layer4(out)
        out = self.fc2(out)
        return out
```

```
model = CNN()
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr= learning_rate)
```

```
total_batch = len(data_loader)

for epoch in range(training_epochs):
    avg_cost = 0
    for X, Y in data_loader:
        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()
        avg_cost += cost/total_batch
    print('[Epoch:{}] cost = {}'.format(epoch+1,avg_cost))
print('Learning Finished!')
```

```
[Epoch:1] cost = 0.3313051164150238
[Epoch:2] cost = 0.11052048206329346
[Epoch:3] cost = 0.08103954046964645
[Epoch:4] cost = 0.0661330297589302
[Epoch:5] cost = 0.05656668543815613
[Epoch:6] cost = 0.047789834439754486
[Epoch:7] cost = 0.04182928428053856
[Epoch:8] cost = 0.03749814257025719
[Epoch:9] cost = 0.03215562924742699
[Epoch:10] cost = 0.025916356593370438
[Epoch:11] cost = 0.02725824899971485
[Epoch:12] cost = 0.021699480712413788
[Epoch:13] cost = 0.02057006023824215
[Epoch:14] cost = 0.016710886731743813
[Epoch:15] cost = 0.020079145208001137
Learning Finished!
```

```
with torch.no_grad():
    X_test = mnist_test.test_data.view(len(mnist_test),1,28,28).float()
    Y_test = mnist_test.test_labels

    prediction = model(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())
```

```
C:\anaconda\lib\site-packages\torchvision\datasets\mnist.py:60: UserWarning:
test_data has been renamed data
  warnings.warn("test_data has been renamed data")
C:\anaconda\lib\site-packages\torchvision\datasets\mnist.py:50: UserWarning:
test_labels has been renamed targets
  warnings.warn("test_labels has been renamed targets")
```

```
Accuracy: 0.9193999767303467
```

더 많은 layer를 쌓고 나서 오히려 accuracy가 줄고 cost가 원래보다 덜 줄었다.