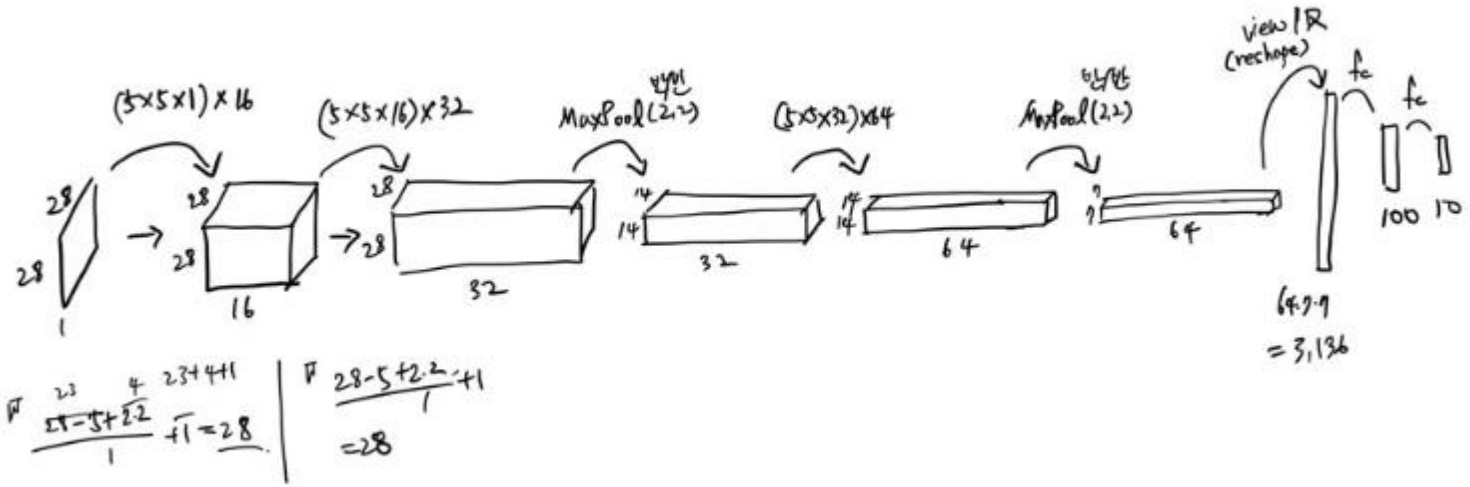
심층학습
[실습05] 합성곱 신경망(1)
나만의 CNN 으로 MNIST 예측하기

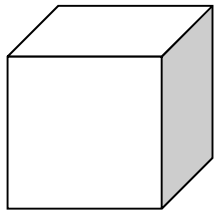
SW융합학부 양희경

GitHub 로 실습코드 관리하길 추천합니다

- AWS SageMaker 5GB 제약
- 포트폴리오 작성법 익힘(미래 나의 재산)
- 오픈소스에 기여

```
![alt text](./figs/mycnn.jpg)
```


(5×5×1)×16
(5×5×16)×32
MaxPool(2,2)
(5×5×32)×64
MaxPool(2,2)
view (R (reshape)
fc
fc
28
28
1
16
32
14
64
7
64
100
10
64·7·7
=3,136



Load packages

```
import torch
import torch.nn as nn
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt
%matplotlib inline
```

MNIST train, test dataset 가져오기

```
# "": 현재 폴더에 MNIST 있음
mnist_train=dset.MNIST("", train=True,transform=transforms.ToTensor(),    #train 용으로 쓰겠다.
                       target_transform=None, download=True)
mnist_test=dset.MNIST("", train=False,transform=transforms.ToTensor(),    #test 용으로 쓰겠다.
                       target_transform=None, download=True)
```

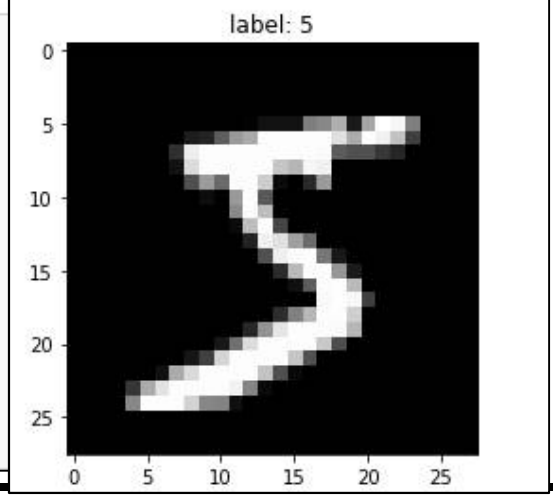
대략적인 데이터 형태

```
print "mnist_train 길이:", len(mnist_train)
print "mnist_test 길이:", len(mnist_test)

# 데이터 하나 형태
image, label = mnist_train.__getitem__(0)  # 0번째 데이터
print "image data 형태:", image.size()
print "label: ", label

# 그리기
img = image.numpy()  # image 타입을 numpy 로 변환 (1,28,28)
plt.title("label: %d" %label )
plt.imshow(img[0], cmap='gray')
plt.show()
```

```
mnist_train 길이: 60000
mnist_test 길이: 10000
image data 형태: torch.Size([1, 28, 28])
label:  5
```

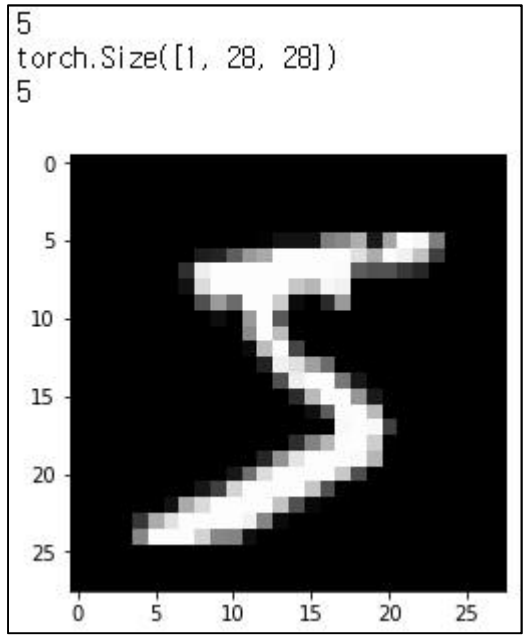


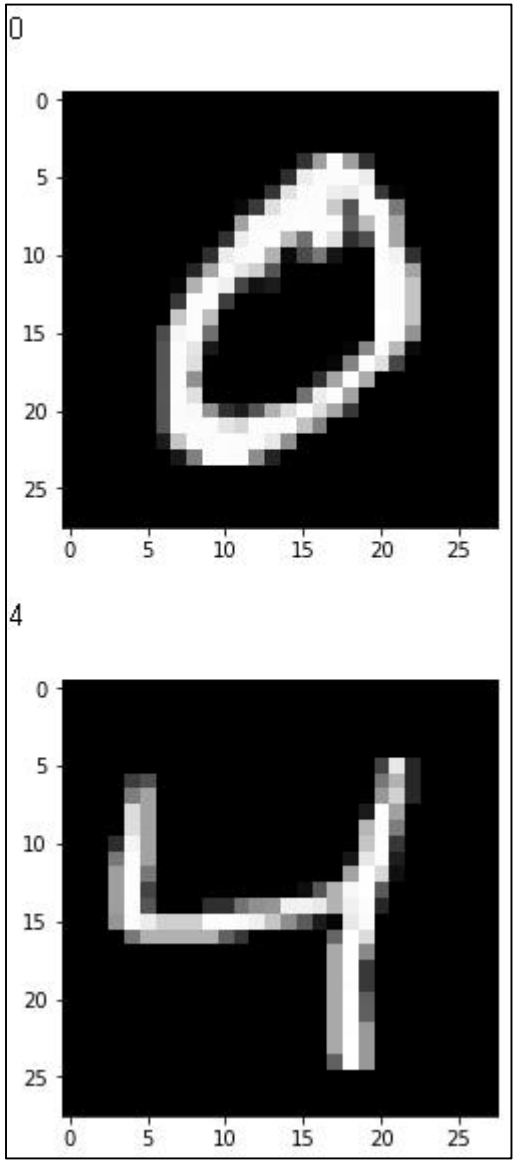
MNIST data 띄워보기

```
print(mnist_train[0][1])  # label
print(mnist_train[0][0].size())  # image

for i in range(3):
    img=mnist_train[i][0].numpy()
    print(mnist_train[i][1])
    plt.imshow(img[0],cmap='gray')
    plt.show()
```

```
5
torch.Size([1, 28, 28])
5
```





convolution 하나 씌워보기

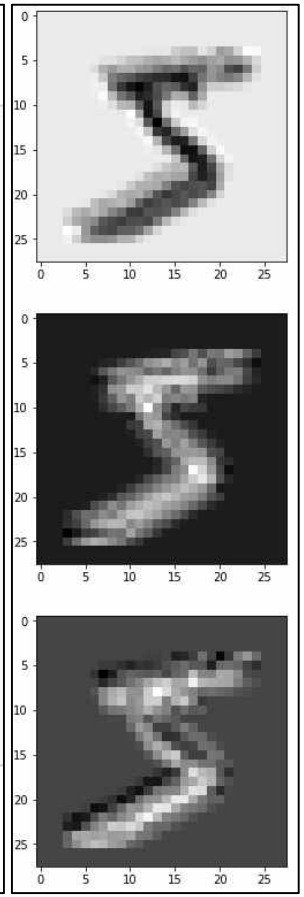
```
# mnist 의 첫 번째 이미지, 라벨 가져오기
image, label = mnist_train[0]
# view: tensor 의 사이즈 조절, -1: 해당 차원 차원 확장시켜라
# [1, 28, 28] -> [1, 1, 28, 28]
image=image.view(-1, image.size()[0], image.size()[1], image.size()[2])
print(image.size())

print label

# convolutional filter 정의
conv_layer=nn.Conv2d(in_channels=1,out_channels=3,kernel_size=3,padding=1)
# image 에 filter 적용
output=conv_layer(Variable(image))
print(output.size())

for i in range(3):
    plt.imshow(output[0,i,:,:].data.numpy(), cmap='gray')
    plt.show()
```

```
torch.Size([1, 1, 28, 28])
5
torch.Size([1, 3, 28, 28])
```



CNN 만들기

train, test data 가져오기

```
import numpy as np
import torch.optim as optim

batch_size = 16
learning_rate = 0.0002
num_epoch = 10  # 1000
```

```
# 추후 학습시킬 때 batch_size 단위로 학습시켜나감
train_loader = torch.utils.data.DataLoader(list(mnist_train)[:batch_size*100], batch_size=batch_size,     # mnist_train 를 트레인 시키
                                           shuffle=True, num_workers=2,
                                           drop_last=True)  # batch_size 만큼 나눌 때 나머지는 버려라
test_loader = torch.utils.data.DataLoader((mnist_test), batch_size=batch_size,
                                           shuffle=False, num_workers=2,
                                           drop_last=True)
```

CNN 클래스 만들기 (모델 만들기)

```
class CNN(nn.Module):  # nn.Module 상속받음
    def __init__(self):
        super(CNN, self).__init__()  # 28 x 28
        self.layer=nn.Sequential(
            nn.Conv2d(1, 16, 5, padding=2),
            nn.ReLU(),

            nn.Conv2d(16, 32, 5, padding=2),  # 28x28
            nn.ReLU(),
            nn.MaxPool2d(2,2), # 28x28 -> 14x14

            nn.Conv2d(32, 64, 5, padding=2),  # 14x14
            nn.ReLU(),
            nn.MaxPool2d(2,2) # 14x14 -> 7x7
        )
        self.fc_layer=nn.Sequential(
            nn.Linear(64*7*7, 100),
            nn.ReLU(),
            nn.Linear(100, 10)
        )

    def forward(self, x):
        out = self.layer(x)
        out = out.view(batch_size, -1)
        out = self.fc_layer(out)
        return out

model = CNN()  #.cuda()
```

```python
# 파라미터 체크하기
for parameter in model.parameters():
    #print(parameter)
    print(parameter.shape)
```

```
torch.Size([16, 1, 5, 5])
torch.Size([16])
torch.Size([32, 16, 5, 5])
torch.Size([32])
torch.Size([64, 32, 5, 5])
torch.Size([64])
torch.Size([100, 3136])
torch.Size([100])
torch.Size([10, 100])
torch.Size([10])
```

```
Parameter containing:
tensor([[[[-0.1887,  0.1129, -0.1489, -0.0068, -0.0159],
          [ 0.1696, -0.0784, -0.0015,  0.0723, -0.0772],
          [-0.1927, -0.0100, -0.0953,  0.0848,  0.0235],
          [-0.0105,  0.1037, -0.0410, -0.1080, -0.0772],
          [-0.0686,  0.0284,  0.0736, -0.1816,  0.1350]]],


        [[[ 0.1999,  0.0078,  0.0319, -0.1338,  0.1147],
          [-0.0109, -0.1797, -0.1692, -0.1761, -0.0948],
          [-0.0352,  0.0369, -0.0302, -0.0821, -0.1885],
          [-0.1129,  0.1022, -0.0583, -0.1412, -0.1346],
          [-0.0010,  0.1089, -0.1664, -0.1130,  0.0037]]],


        [[[ 0.0696, -0.0388,  0.0671,  0.0746,  0.1279],
          [ 0.1933, -0.0219, -0.0519, -0.1040,  0.0996],
          [-0.0172,  0.1272,  0.0765,  0.1681, -0.1148],
          [ 0.1217,  0.1516, -0.1965, -0.0714,  0.0500],
```

```
Parameter containing:
tensor([ 0.1849,  0.1389,  0.1066, -0.0438, -0.0866,  0.0270,  0.0975,  0.0471,
        -0.0411,  0.0839,  0.0837, -0.1520, -0.0953,  0.1173, -0.0452, -0.0727],
       device='cuda:0', requires_grad=True)
```

```
Parameter containing:
tensor([-0.0457,  0.0095,  0.0529, -0.0666, -0.0933, -0.0054,  0.0498,  0.0911,
         0.0459,  0.0169], device='cuda:0', requires_grad=True)
```

© 정관장 CF 중, 2020.1

```
# loss function, optimizer 선언
loss_func = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

optimization

```
for i in range(num_epoch):
    for j, [image, label] in enumerate(train_loader):  # batch_size 만큼
        x = Variable(image)   #.cuda()
        y_ = Variable(label)  #.cuda()

        optimizer.zero_grad()   # optimizer 안에서 이전 gradient 들을 초기화.
        output=model.forward(x)
        loss = loss_func(output, y_)
        loss.backward()  # gradient 계산
        optimizer.step() # parameter 업데이트

        if j%50==0:
            print(loss, j, i)
```

```
(tensor(2.2991, grad_fn=<NllLossBackward>), 0, 0)
(tensor(1.8544, grad_fn=<NllLossBackward>), 50, 0)
(tensor(0.6138, grad_fn=<NllLossBackward>), 0, 1)
(tensor(0.6471, grad_fn=<NllLossBackward>), 50, 1)
(tensor(0.1409, grad_fn=<NllLossBackward>), 0, 2)
(tensor(0.1365, grad_fn=<NllLossBackward>), 50, 2)
(tensor(0.0531, grad_fn=<NllLossBackward>), 0, 3)
(tensor(0.2963, grad_fn=<NllLossBackward>), 50, 3)
(tensor(0.1931, grad_fn=<NllLossBackward>), 0, 4)
(tensor(0.2960, grad_fn=<NllLossBackward>), 50, 4)
(tensor(0.1474, grad_fn=<NllLossBackward>), 0, 5)
(tensor(0.0814, grad_fn=<NllLossBackward>), 50, 5)
```

```
# 모델 저장시키기
torch.save(model, 'nets/mycnn_model_%d.pkl'%(num_epoch))
```

```
/usr/local/lib/python2.7/dist-packages/torch/serialization.py:256: Use
t won't be checked for correctness upon loading.
  "type " + obj.__name__ + ". It won't be checked "
```

```
try:
    # 미리 학습시킨 네트워크의 파라미터 집합 [피클]이라 발음함.
    model=torch.load('nets/mycnn_model_10.pkl')
    print("model restored")
except:
    print("model not restored")
```

```
model restored
```

```python
def ComputeAccr(dloader, imodel):
    correct = 0
    total = 0

    for j, [imgs, labels] in enumerate(dloader):  # batch_size 만큼
        img = Variable(imgs)  #.cuda()  # x
        label = Variable(labels)  # y
        #label = Variable(labels).cuda()
        # .cuda() : GPU 에 로드되기 위함. 만약 CPU로 설정되어 있다면 에러남


        output = imodel.forward(img)  # forward prop.
        _, output_index = torch.max(output, 1)

        total += label.size(0)
        correct += (output_index == label).sum().float()
    print("Accuracy of Test Data: {}".format(100*correct/total))
```

```python
ComputeAccr(test_loader, model)
```

```
Accuracy of Test Data: 94.1399993896
```

오늘의 과제

- '[실습05] 합성곱 신경망(1)' 을 실습한다.
  - P.3 ~ 13
- ipynb, HTML 파일을 다운받아 e-campus 에 제출한다.
- 마감: e-campus 과제 마감일 확인