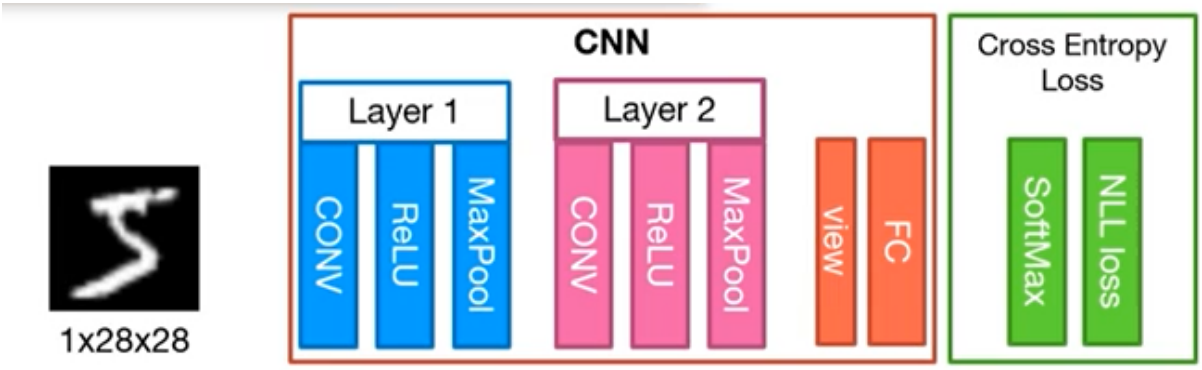
MNIST CNN

상태	CNN
담당자	

딥러닝 학습 단계

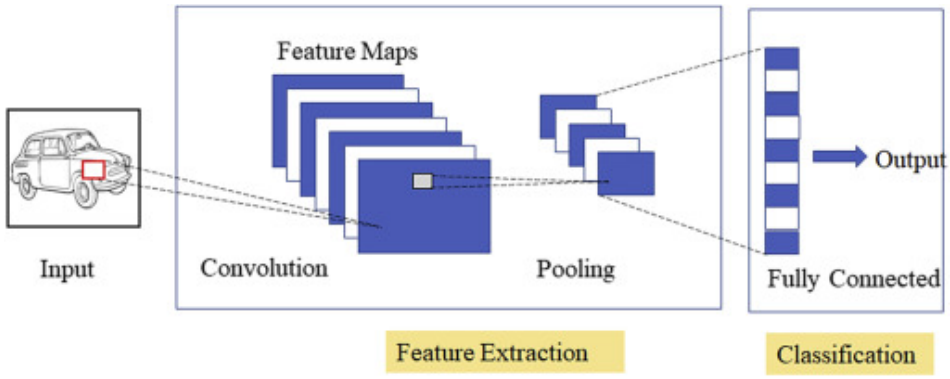
만들고자 하는 네트워크 구조 확인



1채널의 28x28 MNIST 데이터셋을 가져와 conv1 layer, conv2 layer, fully connected layer로 구성된 CNN 네트워크를 만들고, cross entropy loss를 이용하여 학습해 보자.

- conv1 layer
  - convolution layer : `input_channel=1`, `output_channel=32`, `kernel_size=3`, `stride=1`, `padding=1`
  - ReLU : activation function
  - MaxPool : `kernel_size=2`, `stride=2`
- conv1 layer
  - convolution layer : `input_channel=32`, `output_channel=64`, `kernel_size=3`, `stride=1`, `padding=1`
  - ReLU : activation function
  - MaxPool : `kernel_size=2`, `stride=2`
- View : batch_size x [3136]
- FC : `input=3136`, `output=10`

conv1 layer를 통해 32개의 feature map을 추출하고, conv2 layer를 통해 64개의 feature map을 추출하게 된다.



input에서 channel이 R, G, B의 채널을 의미하는 반면, conv layer에서 channel은 추출해내는 feature map의 개수를 의미하는 것임에 유의하자.

Pooling은 channel을 변화시키지 않고 이미지의 크기만 변화시킨다.

▼ `view` vs `.reshape`

기본적으로 거의 같은 기능을 하지만, `reshape`의 경우 데이터를 copy해 온다.

```
[79] a = torch.zeros(3,2)
     b = a.t().reshape(6)
     a.fill_(1)

     tensor([[1., 1.],
             [1., 1.],
             [1., 1.]])

[80] b

     tensor([0., 0., 0., 0., 0., 0.])
```

라이브러리 가져오기

```
# Lab 11 MNIST and Convolutional Neural Network
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn.init
```

GPU와 random seed 설정

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

하이퍼파라미터 설정

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

데이터셋 가져오기, DataLoader 만들기

```
# MNIST dataset
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)
```

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

모델 만들기

```
# CNN Model (2 conv layers)
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32) - pooling은 채널 수 보존
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform_(self.fc.weight)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)   # Flatten them for FC, batch size만큼 펼쳐 주고 나머지는 한 줄로
        out = self.fc(out)
        return out
```

```
# instantiate CNN model
model = CNN().to(device)
```

Loss function과 optimizer 선택

```
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

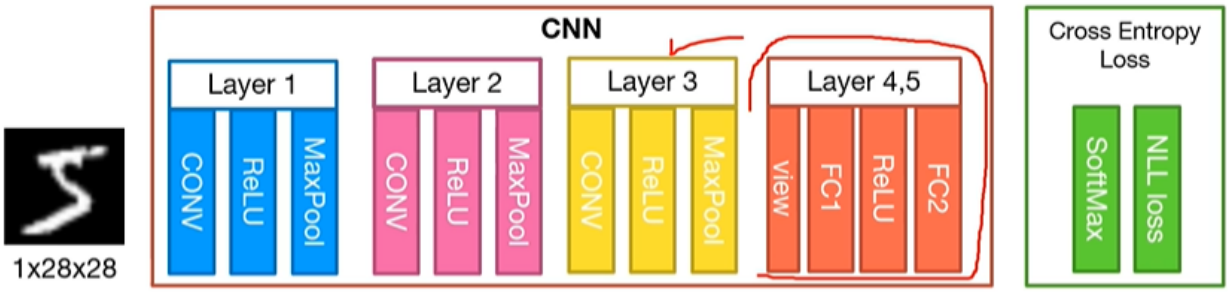
모델 학습과 loss 확인

```
# train my model
total_batch = len(data_loader)
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader:
        # image is already size of (28x28), no reshape
        # label is not one-hot encoded
        X = X.to(device)
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print('[Epoch: {:>4}] cost = {:>.9}'.format(epoch + 1, avg_cost))

print('Learning Finished!')
```

모델 성능 확인

```
# Test model and check accuracy
with torch.no_grad():
    X_test = mnist_test.test_data.view(len(mnist_test), 1, 28, 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

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

더 좋은 모델을 위해?



레이어를 더 깊게 쌓으면 정확도가 더 높아지지 않을까? 위와 같은 레이어 3, 4, 5를 추가해 보자.

```
# CNN Model
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        self.keep_prob = 0.5
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L3 ImgIn shape=(?, 7, 7, 64)
        #    Conv      ->(?, 7, 7, 128)
        #    Pool      ->(?, 4, 4, 128)
        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2, padding=1))

        # L4 FC 4x4x128 inputs -> 625 outputs
        self.fc1 = torch.nn.Linear(4 * 4 * 128, 625, bias=True)
        torch.nn.init.xavier_uniform_(self.fc1.weight)
        self.layer4 = torch.nn.Sequential(
            self.fc1,
            torch.nn.ReLU(),
            torch.nn.Dropout(p=1 - self.keep_prob))
        # L5 Final FC 625 inputs -> 10 outputs
        self.fc2 = torch.nn.Linear(625, 10, bias=True)
        torch.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)   # Flatten them for FC
        out = self.layer4(out)
        out = self.fc2(out)
        return out
```

`self.layer3` 의 pooling 결과 64x7x7 → 64x4x4 가 되는데, 자투리가 남았을 때 알아서 모자란 부분을 무시하지 않고 계산하기 때문에 4x4가 될 수 있다. 이는 `nn.MaxPool2d`의 기본 옵션 `ceil_mode=False` 때문이다.

> MaxPool2d - PyTorch 1.10 documentation
>
> Join the PyTorch developer community to contribute, learn, and get your questions answered.
>
> https://pytorch.org/docs/1.9.1/generated/torch.nn.MaxPool2d.html

모델을 깊게 쌓는다고 항상 좋은 것만은 아니다! 효율적인 모델을 구성하는 것이 더 중요하다.

Tips?

사이즈를 항상 손으로 계산할 필요가 없다. 우선 임의의 숫자를 넣어서 모델을 구성한 다음, 파이썬이 내뱉는 에러를 보면서 사이즈를 맞춰줘도 된다.

예를 들어서 `self.fc`의 첫번째 파라미터(인풋의 크기)를 `1234`처럼 임의의 숫자로 우선 넣어본다.

```
class CNN_test(torch.nn.Module):

    def __init__(self):
        super().__init__()
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))

        self.fc = torch.nn.Linear(1234, 10, bias=True)
        torch.nn.init.xavier_uniform_(self.fc.weight)

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

```
        out = self.fc(out)
        return out
```

이 모델을 학습시키려 하면 다음과 같은 에러가 발생한다.

`RuntimeError: size mismatch, m1: [100 x 3136], m2: [1234 x 10] at /opt/conda/conda-bld/pytorch-cpu_1549626403278/work/aten/src/TH/generic/THTensorMath.cpp:940`

3136을 받을 것으로 예상했지만 1234를 받아 에러가 발생했다. 따라서 위의 `1234`를 `3136`으로 수정해 주면 간편하다.