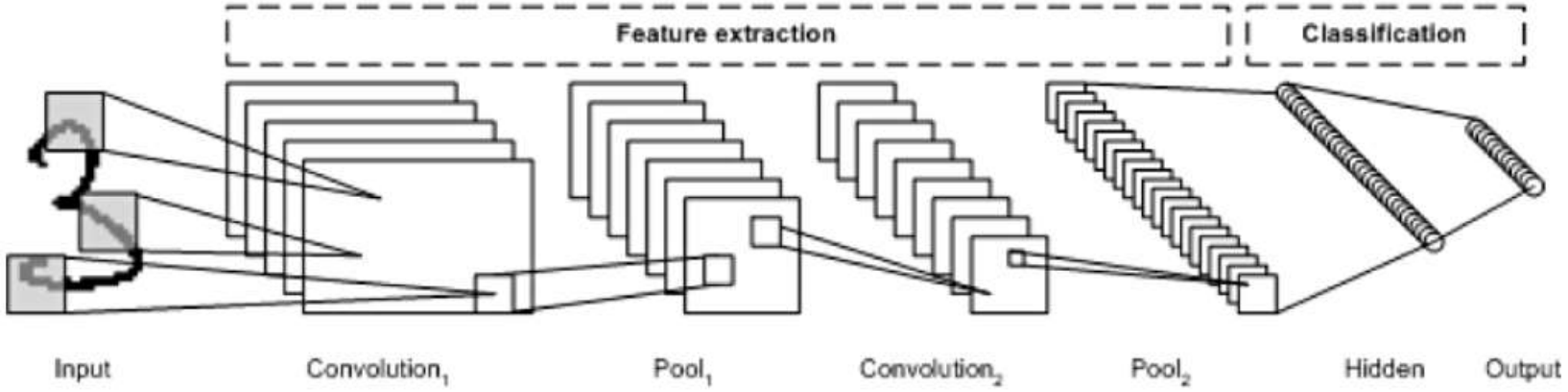
CNN

Eunseo, Lee

Index

- Brief introduction of CNN
- Convolution
- Pooling
- Mnist CNN
- Visdom()
- Image Folder

CNN



Fully connected Neural network VS Convolutional Neural network

Maintains the shape of input& output data on each layer
Effectively recognizes characteristics with adjacent images while maintaining spatial information in the image.
Extract and learn the characteristics of an image with multiple filters
Pooling Layer to Collect and Strengthen the Characteristics of Extracted Images
Since filters are used as shared parameters, there are very few learning parameters compared to normal artificial neural networks

CNN has repeatedly used Convolution and Pooling to find unchanging features and send them to the Fully-connected neural network as input data.

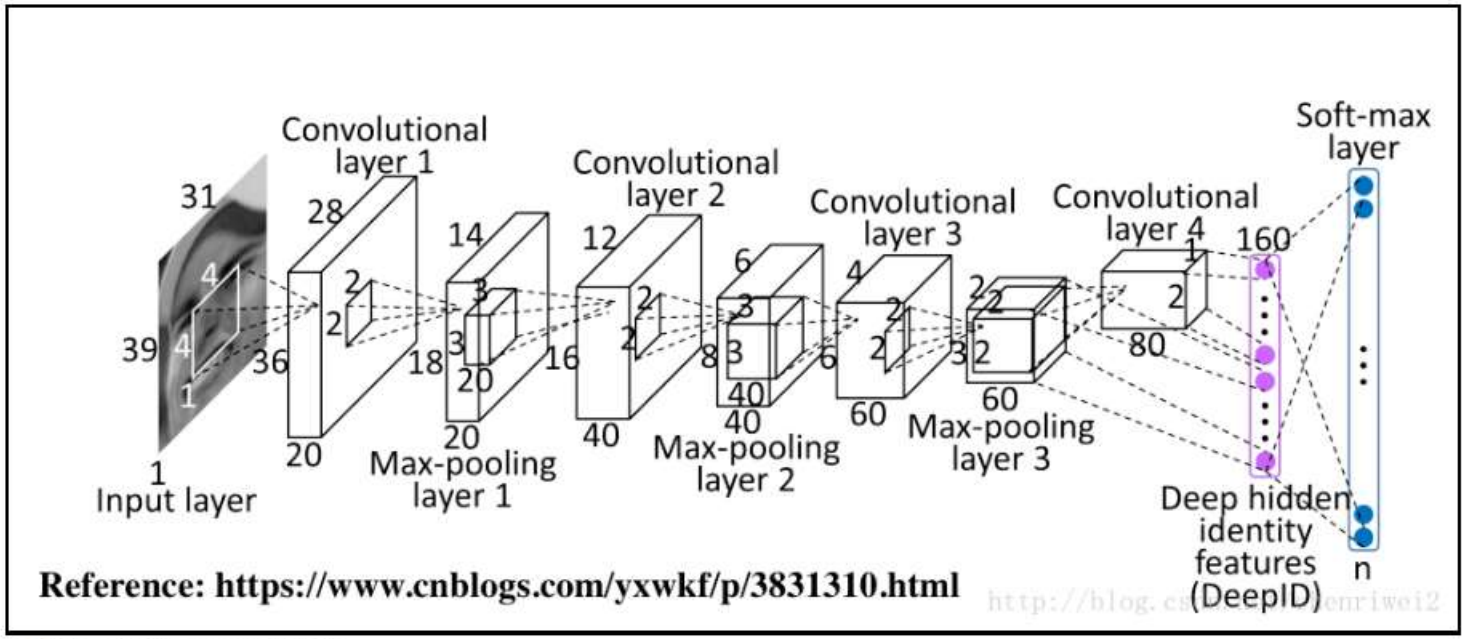
Then Perform the Classification.

http://taewan.kim/post/cnn/#fn:1

Words

- Convolution
- Channel
- Filter
- Kernel
- Strid
- Padding
- Feature Map
- Activation Map
- Pooling Layer
- Control size of output data: size of filter,stride,padding,pooling
- Output channel: number of filter

Convolution



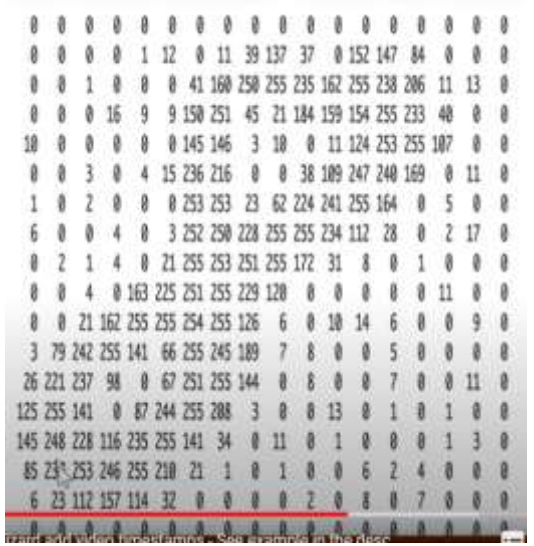
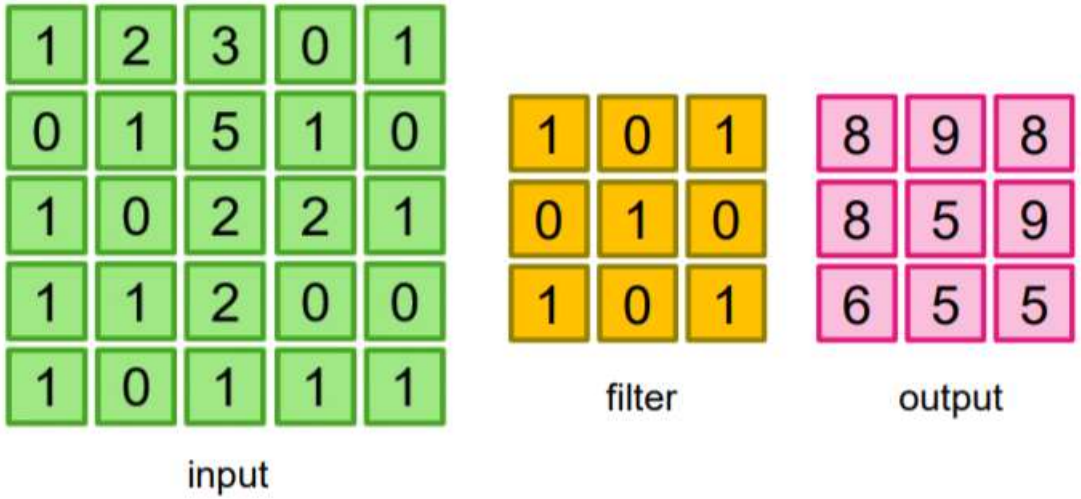


그림 9: 예제 CNN 이미지



Convolution의 output 크기

$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

예제 1)
input image size : 227 x 227
filter size = 11x11
stride = 4
padding = 0
output image size = ?

예제 2)
input image size : 64 x 64
filter size = 7x7
stride = 2
padding = 0
output image size = ?

예제 3)
input image size : 32 x 32
filter size = 5x5
stride = 1
padding = 2
output image size = ?

예제 4)
input image size : 32 x 64
filter size = 5x5

예제 5)
input image size : 64 x 32
filter size = 3x3

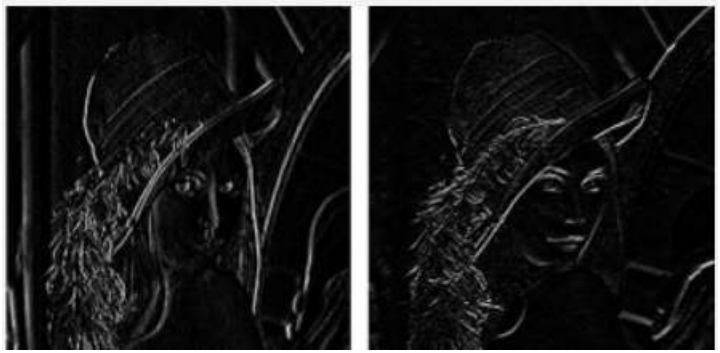
Filter=Kernel

-1	0	+1
-2	0	+2
-1	0	+1

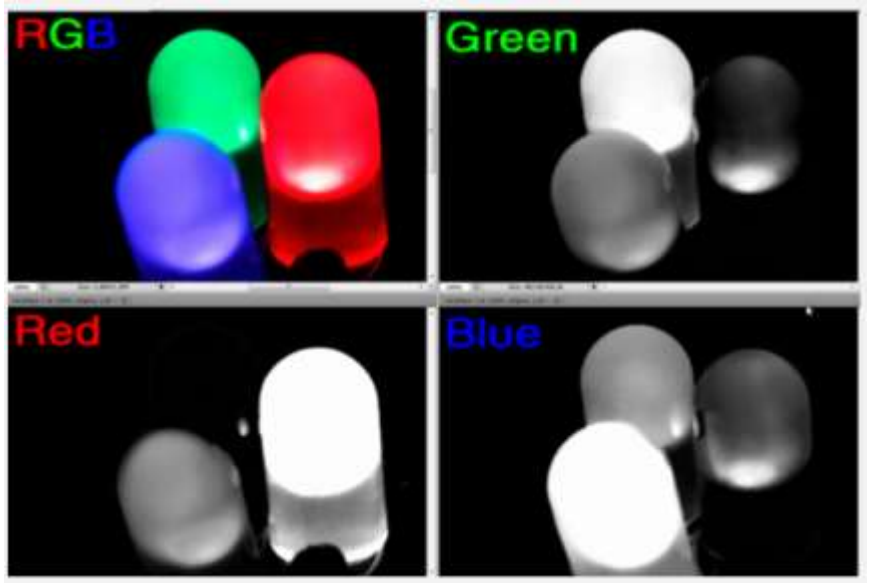
Sobel-X
(vertical)

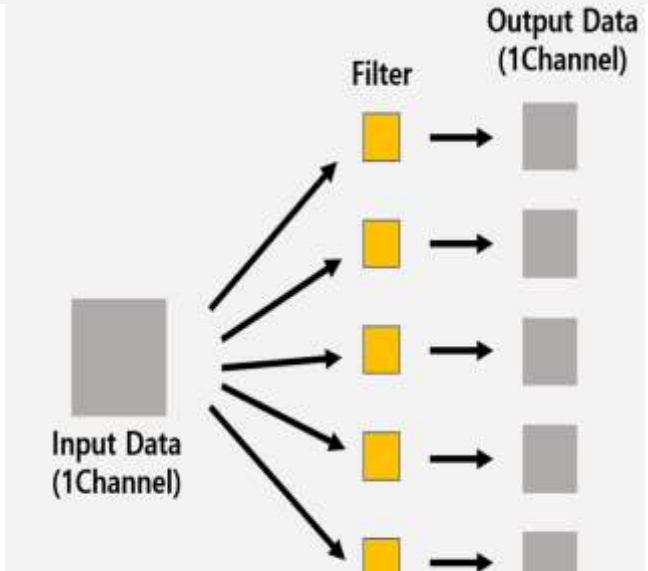
-1	0	+1
-2	0	+2
-1	0	+1

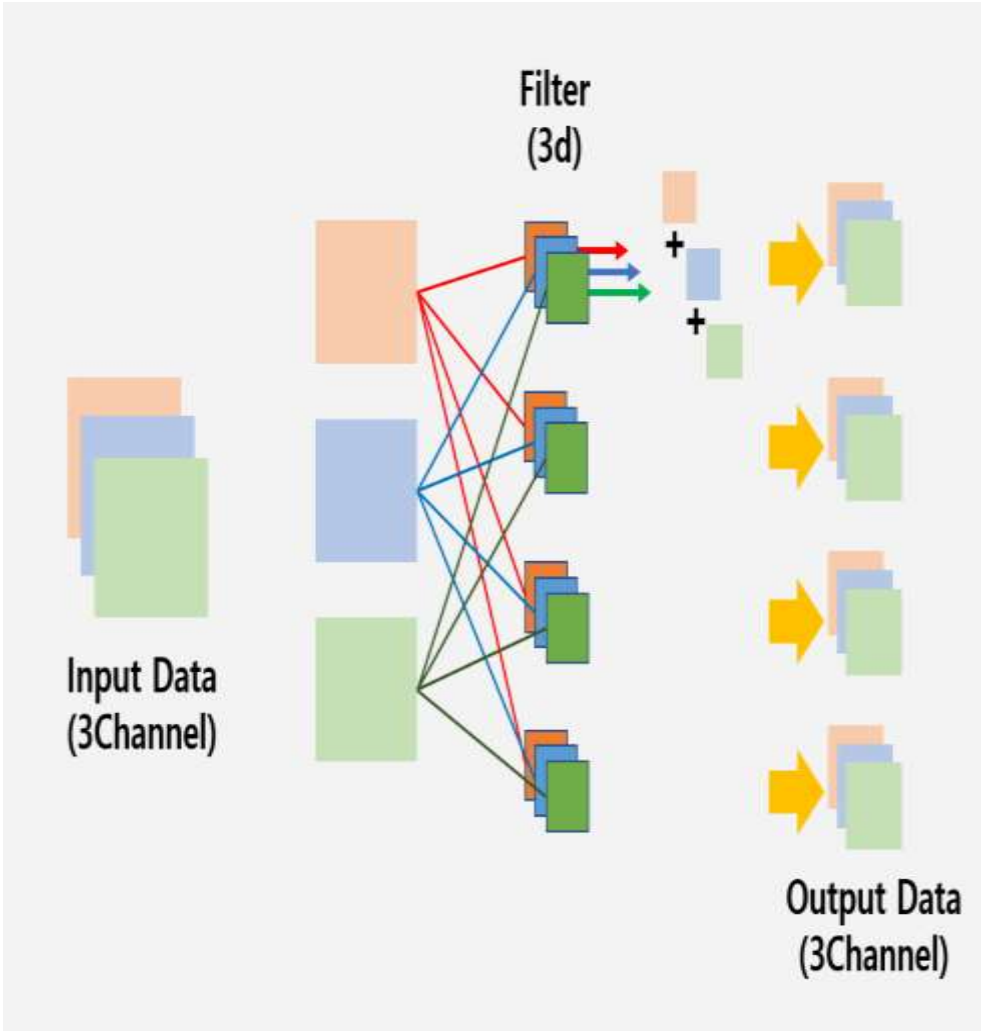
Sobel-Y
(horizontal)



Channel


RGB
Green
Red
Blue

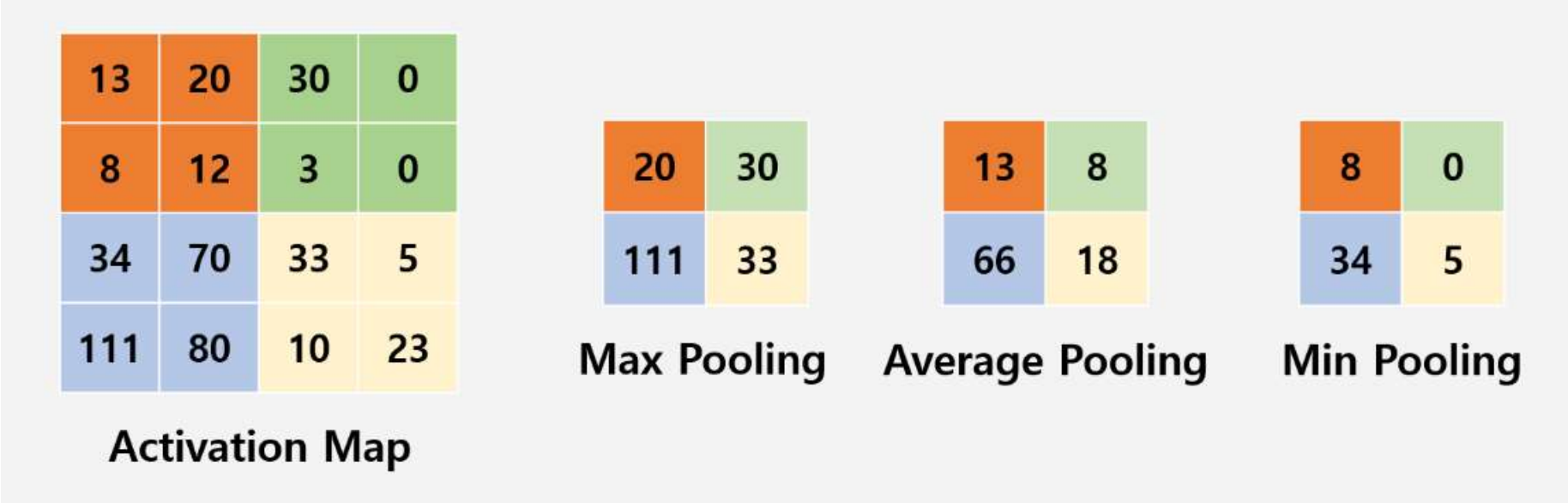

Output Data
(1Channel)
Filter
Input Data
(1Channel)


Filter
(3d)
Input Data
(3Channel)
Output Data
(3Channel)

Pooling

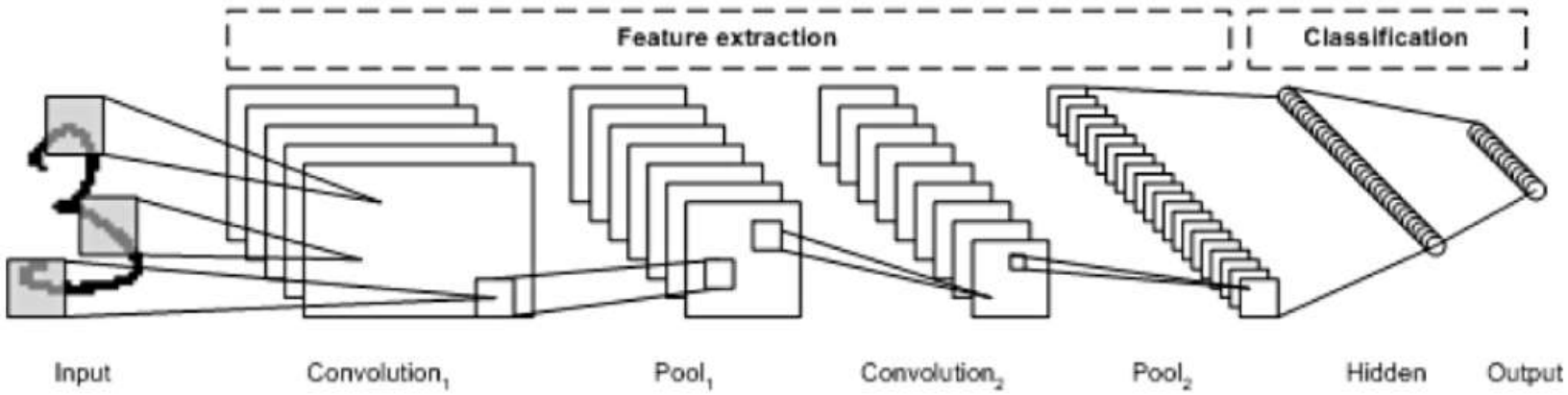
$$OutputRowSize = \frac{InputRowSize}{PoolingSize}$$

$$OutputColumnSize = \frac{InputColumnSize}{PoolingSize}$$



Activation map: feature map+ activation function

Process



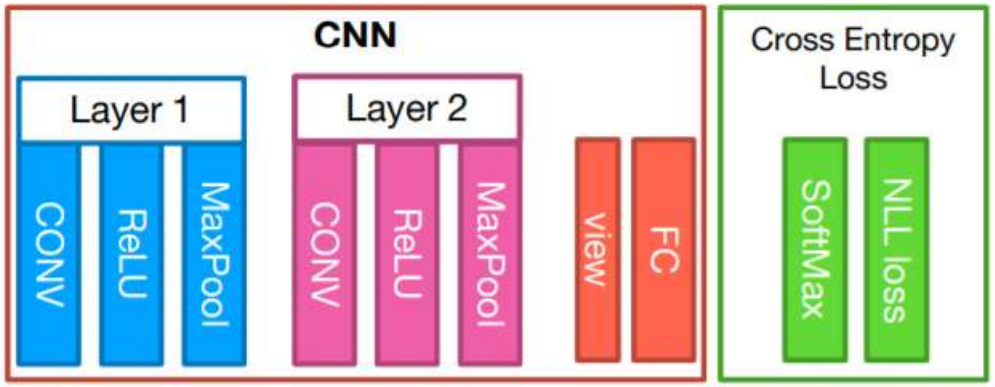
특징 추출 단계(Feature Extraction)
•Convolution Layer : extract feature through filter
•Pooling Layer : strengthen feature& reduce size of the image

이미지 분류 단계(Classification)
•Flatten Layer : change data type into Fcnetwork form. Change shape of input data.
•Softmax Layer : Classification
Output

https://seongkyun.github.io/study/2019/01/25/num_of_parameters/
http://taewan.kim/post/cnn/
:calculating number of parameters

Mnist CNN

1x28x28



(Layer 1) Convolution layer = (in_c=1, out_c=32,kernel_size =3, stride=1,padding=1)
(Layer 1) MaxPool layer = (kernel_size=2, stride =2)

(Layer 2) Convolution layer = (in_c=32, out_c=64, kernel_size =3, stride=1,padding=1)
(Layer 2) MaxPool layer = (kernel_size=2, stride =2)

view => (batch_size x [7,7,64] => batch_size x [3136])
Fully_Connect layer => (input=3136, output = 10)

```
# 1번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 1, out_channel = 32, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2))

# 2번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 32, out_channel = 64, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2))

# 3번 레이어 : 전결합층(Fully-Connected layer)
특성맵을 펼친다. # batch_size x 7 x 7 x 64 → batch_size x 3136
전결합층(뉴런 10개) + 활성화 함수 Softmax
```

Mnist CNN

```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn.init
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# 랜덤 시드 고정
torch.manual_seed(777)

# GPU 사용 가능일 경우 랜덤 시드 고정
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

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

```
mnist_train = dsets.MNIST(root='MNIST_data/', # 다운로드 경로 지정
                          train=True, # True를 지정하면 훈련 데이터로 다운로드
                          transform=transforms.ToTensor(), # 텐서로 변환
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/', # 다운로드 경로 지정
                         train=False, # False를 지정하면 테스트 데이터로 다운로드
                         transform=transforms.ToTensor(), # 텐서로 변환
                         download=True)
```

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

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

    def __init__(self):
        super(CNN, self).__init__()
        # 첫번째층
        # 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))

        # 두번째층
        # 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))

        # 전결합층 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)

        # 전결합층 한정으로 가중치 초기화
        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)   # 전결합층을 위해서 Flatten
        out = self.fc(out)
        return out
```

Out.view(out.size(0),-1)

`a` is a tensor that has 16 elements from 1 to 16(included). If you want to reshape this tensor to make it a `4 x 4` tensor then you can use

```
a = a.view(4, 4)
```

Now `a` will be a `4 x 4` tensor. *Note that after the reshape the total number of elements need to remain the same. Reshaping the tensor `a` to a `3 x 5` tensor would not be appropriate.*

What is the meaning of parameter -1?

If there is any situation that you don't know how many rows you want but are sure of the number of columns, then you can specify this with a -1. (*Note that you can extend this to tensors with more dimensions. Only one of the axis value can be -1*). This is a way of telling the library: "give me a tensor that has these many columns and you compute the appropriate number of rows that is necessary to make this happen".

MNIST CNN

```
# CNN 모델 정의
model = CNN().to(device)
```

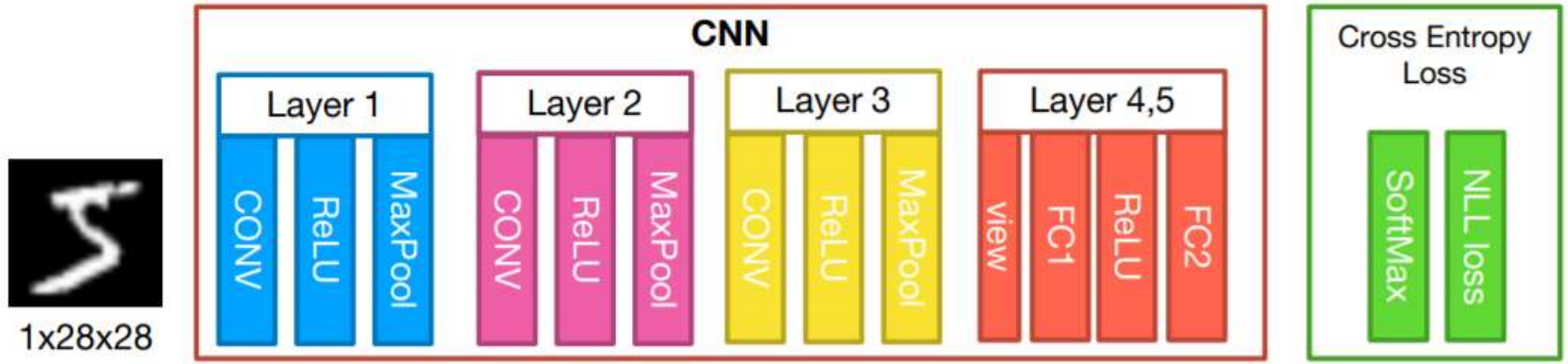
```
criterion = torch.nn.CrossEntropyLoss().to(device)    # 비용 함수에 소프트맥스 함수 포함되어져 있음.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
total_batch = len(data_loader)
print('총 배치의 수 : {}'.format(total_batch))
```

총 배치의 수 : 600

```
for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader: # 미니 배치 단위로 꺼내온다. X는 미니 배치, Y는 레이블.
        # 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))
```

```
# 학습을 진행하지 않을 것이므로 torch.no_grad()
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())
```



(Layer 1) Convolution layer = (in_c=1, out_c=32,kernel_size =3, stride=1,padding=1)
(Layer 1) MaxPool layer = (kernel_size=2, stride =2)

(Layer 2) Convolution layer = (in_c=32, out_c=64,kernel_size =3, stride=1,padding=1)
(Layer 2) MaxPool layer = (kernel_size=2, stride =2)

(Layer 3) Convolution layer = (in_c=64, out_c=128,kernel_size =3, stride=1,padding=1)
(Layer 3) MaxPool layer = (kernel_size=2, stride =2)

(Layer 4) Fully Connected layer =(input=4*4*128, output = 625)
(Layer 5) Fully Connected layer =(input=625, output = 10)

```
# 1번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 1, out_channel = 32, kernel_size=3,
맥스풀링(kernel_size=2, stride=2))

# 2번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 32, out_channel = 64, kernel_size=3
맥스풀링(kernel_size=2, stride=2))

# 3번 레이어 : 합성곱층(Convolutional layer)
합성곱(in_channel = 64, out_channel = 128, kernel_size=3, stride=1, padding=1) + 활성화 함수 ReLU
맥스풀링(kernel_size=2, stride=2, padding=1))

# 4번 레이어 : 전결합층(Fully-Connected layer)
특성맵을 펼친다. # batch_size × 4 × 4 × 128 → batch_size × 2048
전결합층(뉴런 625개) + 활성화 함수 ReLU

# 5번 레이어 : 전결합층(Fully-Connected layer)
전결합층(뉴런 10개) + 활성화 함수 Softmax
```

```
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
#모델을 정의합니다.

# CNN 모델 정의
model = CNN().to(device)
#비용 함수와 옵티마이저를 정의합니다.

criterion = torch.nn.CrossEntropyLoss().to(device)    # 비용 함수에 소프트맥스 함수 포함되어져 있음.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

nn.init.Xavier_uniform_: initiate weight
https://eda-ai-lab.tistory.com/404
nn.Dropout: prevent overfitting/generalize
https://eda-ai-lab.tistory.com/405?category=764743

Visdom

- https://ai.facebook.com/tools/visdom/
- https://keep-steady.tistory.com/12

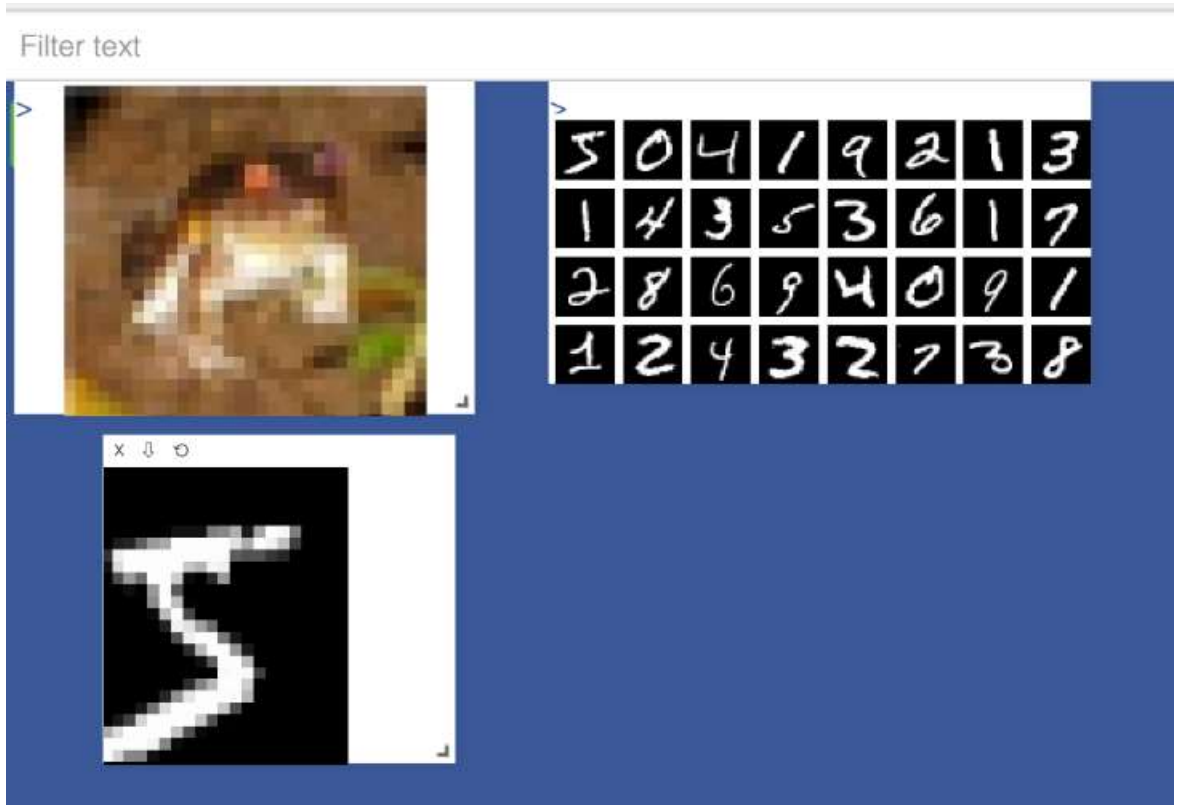


Image folder

- https://data-panic.tistory.com/13

Workflow

1. Dataset 만들기
2. transform을 사용해 image data compose하기 (예. image사이즈 조정, normalize, tensor로 바꾸기 등등)
3. ImageFolder를 이용해 dataload하기
4. DataLoader로 dataset을 (batch 이미지) 불러오기
5. Neural Network 만들기
6. NeuralNet의 Input과 Output Channel입력
7. Optimizer, loss_func 정의하기
8. DataLoader를 epoch횟수로 훈련하기.
9. Test하기