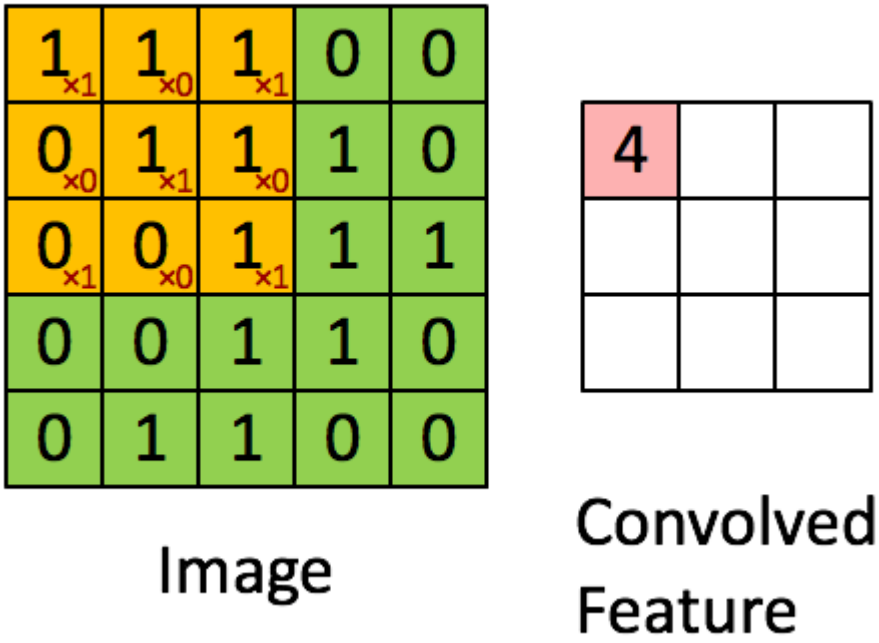
Convolutional Neural Network

ESC – 2조
김정현
신예진
이규민
최우현
최정욱
한인욱

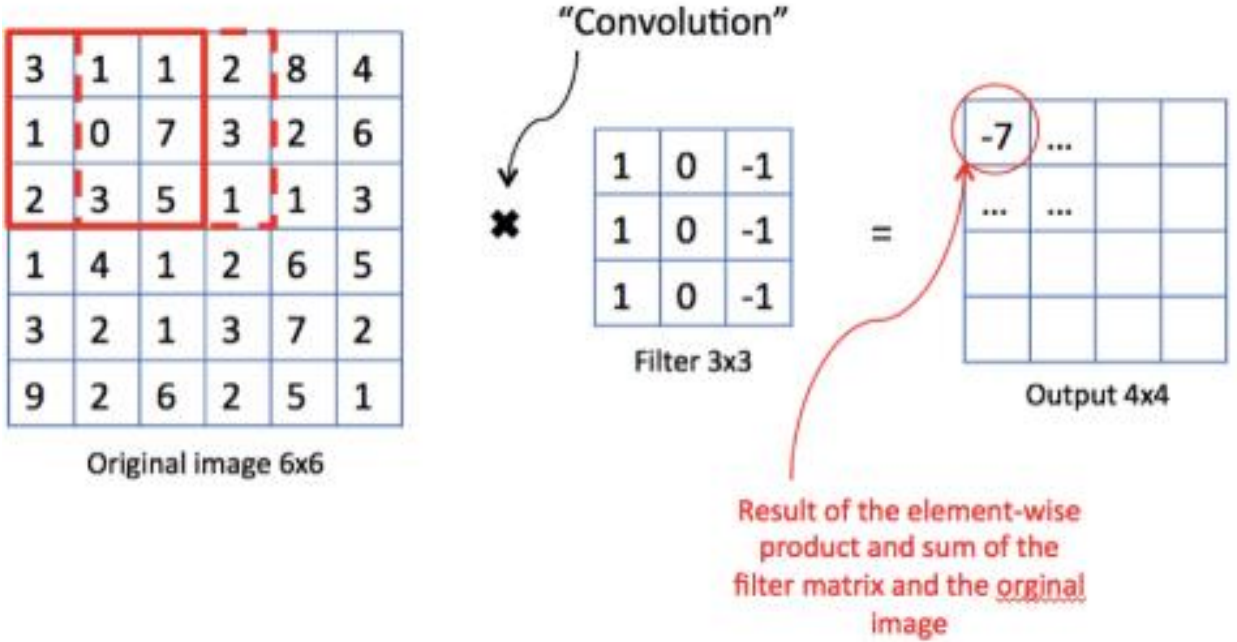
Convolution



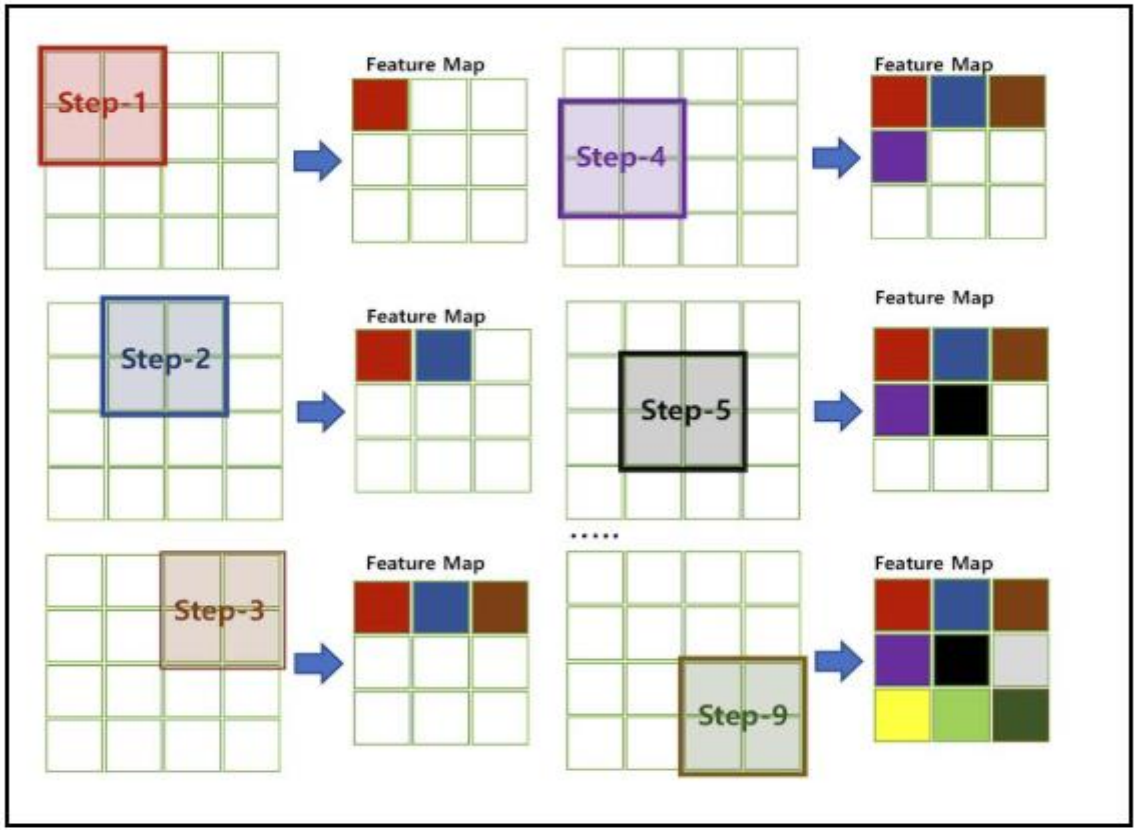
Channel

그림 2: 3개의 채널로 만들어진 컬러 사진

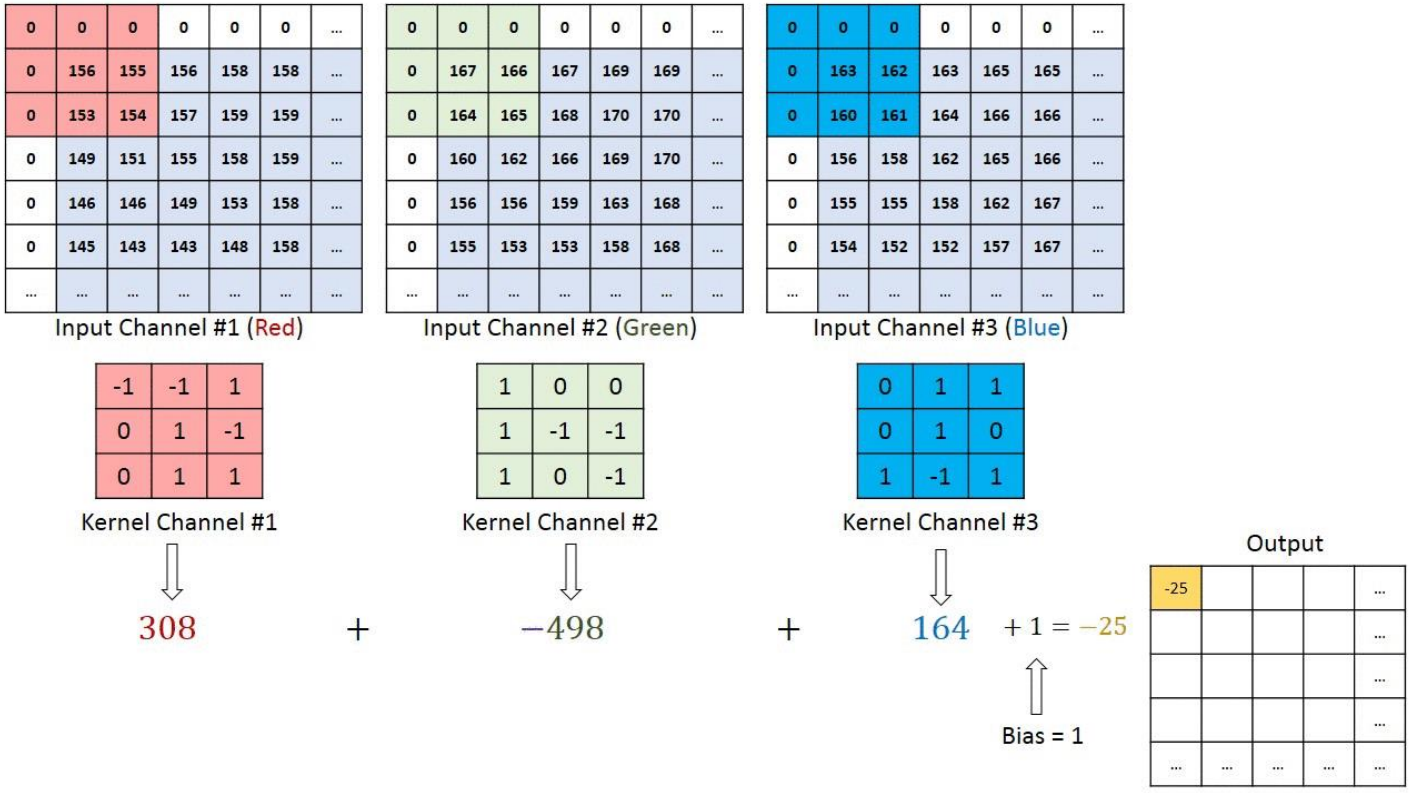
Filter


3 1 1 2 8 4
1 0 7 3 2 6
2 3 5 1 1 3
1 4 1 2 6 5
3 2 1 3 7 2
9 2 6 2 5 1
Original image 6x6
"Convolution"
1 0 -1
1 0 -1
1 0 -1
Filter 3x3
=
-7 ...
... ...
Output 4x4
Result of the element-wise product and sum of the filter matrix and the orginal image

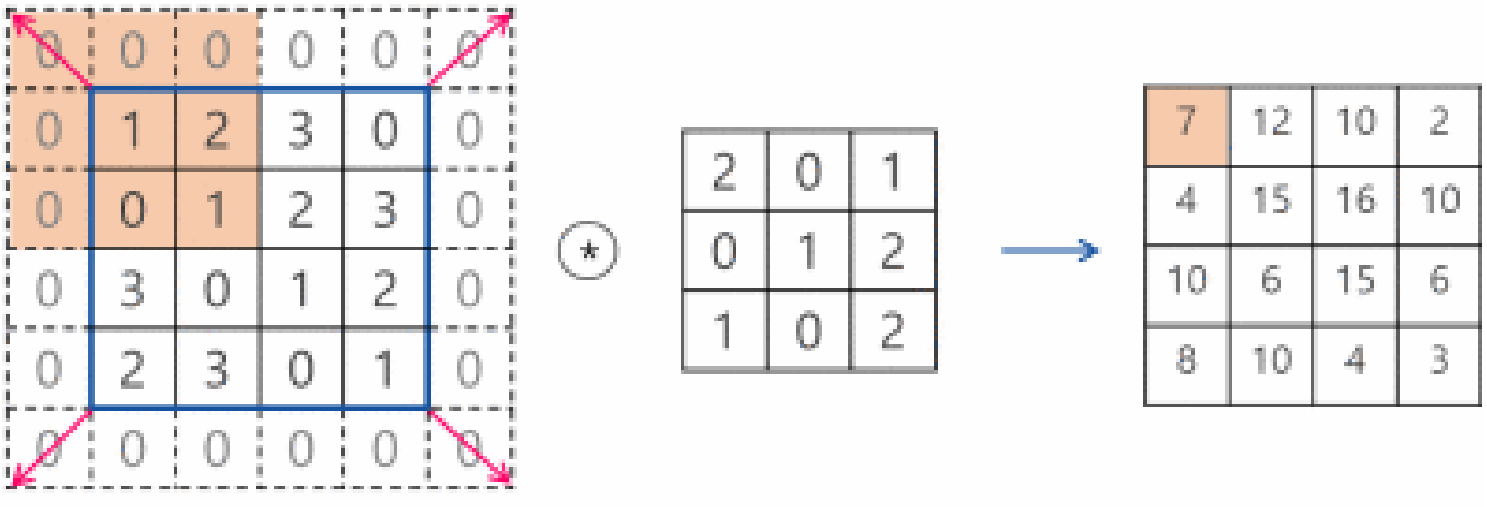
Stride


Step-1
Feature Map
Step-4
Feature Map
Step-2
Feature Map
Step-5
Feature Map
.....
Step-3
Feature Map
Step-9
Feature Map

연산 과정


0 0 0 0 0 0 ...
0 156 155 156 158 158 ...
0 153 154 157 159 159 ...
0 149 151 155 158 159 ...
0 146 146 149 153 158 ...
0 145 143 143 148 158 ...
... ...
Input Channel #1 (Red)
0 0 0 0 0 0 ...
0 167 166 167 169 169 ...
0 164 165 168 170 170 ...
0 160 162 166 169 170 ...
0 156 156 159 163 168 ...
0 155 153 153 158 168 ...
... ...
Input Channel #2 (Green)
0 0 0 0 0 0 ...
0 163 162 163 165 165 ...
0 160 161 164 166 166 ...
0 156 158 162 165 166 ...
0 155 155 158 162 167 ...
0 154 152 152 157 167 ...
... ...
Input Channel #3 (Blue)
-1 -1 1
0 1 -1
0 1 1
Kernel Channel #1
1 0 0
1 -1 -1
1 0 -1
Kernel Channel #2
0 1 1
0 1 0
1 -1 1
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25 ... ...
... ...
... ...
... ...
... ...

Padding


0 0 0 0 0 0
0 1 2 3 0 0
0 0 1 2 3 0
0 3 0 1 2 0
0 2 3 0 1 0
0 0 0 0 0 0
*
2 0 1
0 1 2
1 0 2
7 12 10 2
4 15 16 10
10 6 15 6
8 10 4 3

Output size

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 = ?

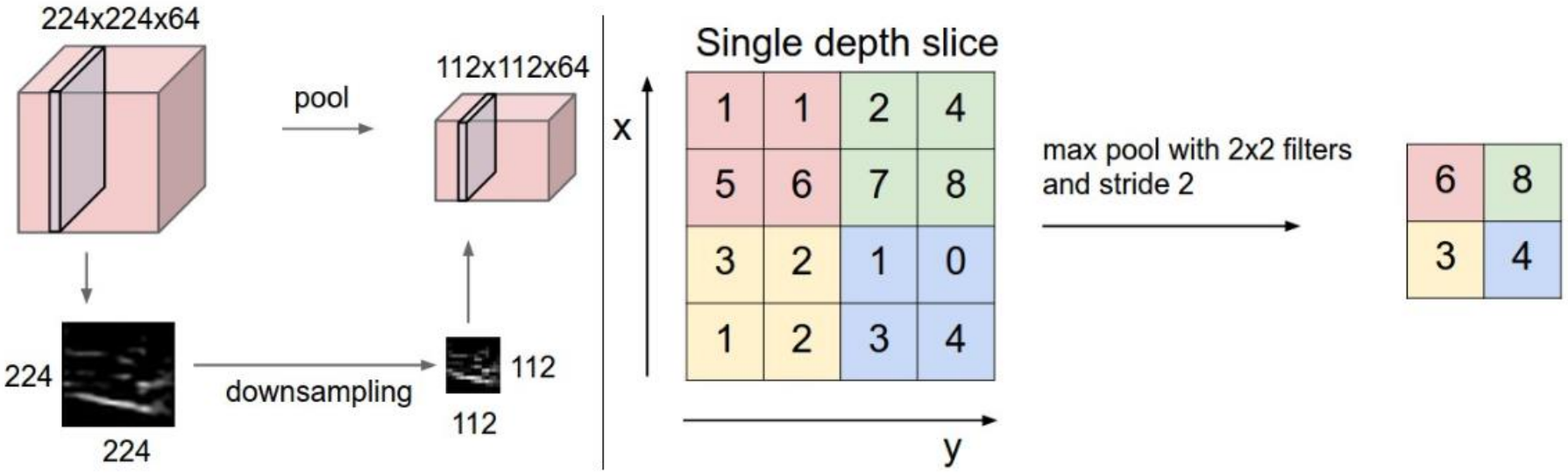
예제 2)
input image size : 64 x 64
filter size = 7x7
stride = 2
padding = 0
output image size = ?

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
stride = 1
padding = 0
output image size = ?

예제 5)
input image size : 64 x 32
filter size = 3x3
stride = 1
padding = 1
output image size = ?

Pooling Layer



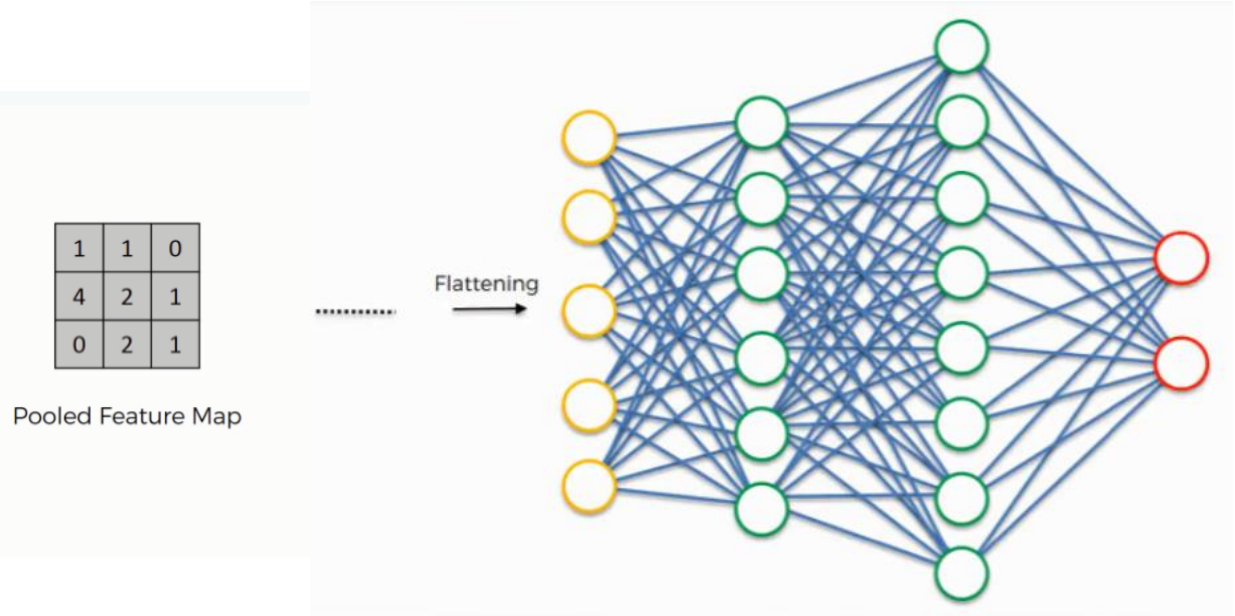
Pooling Layer

$$\text{Output row size} = \frac{input\ row\ size}{pooling\ layer\ size}$$

$$\text{Output column size} = \frac{input\ column\ size}{pooling\ layer\ size}$$

최신 모델: filter 작게, layer 깊게, pooling/ fully-connected layer 조금만!

Pooling 이후…

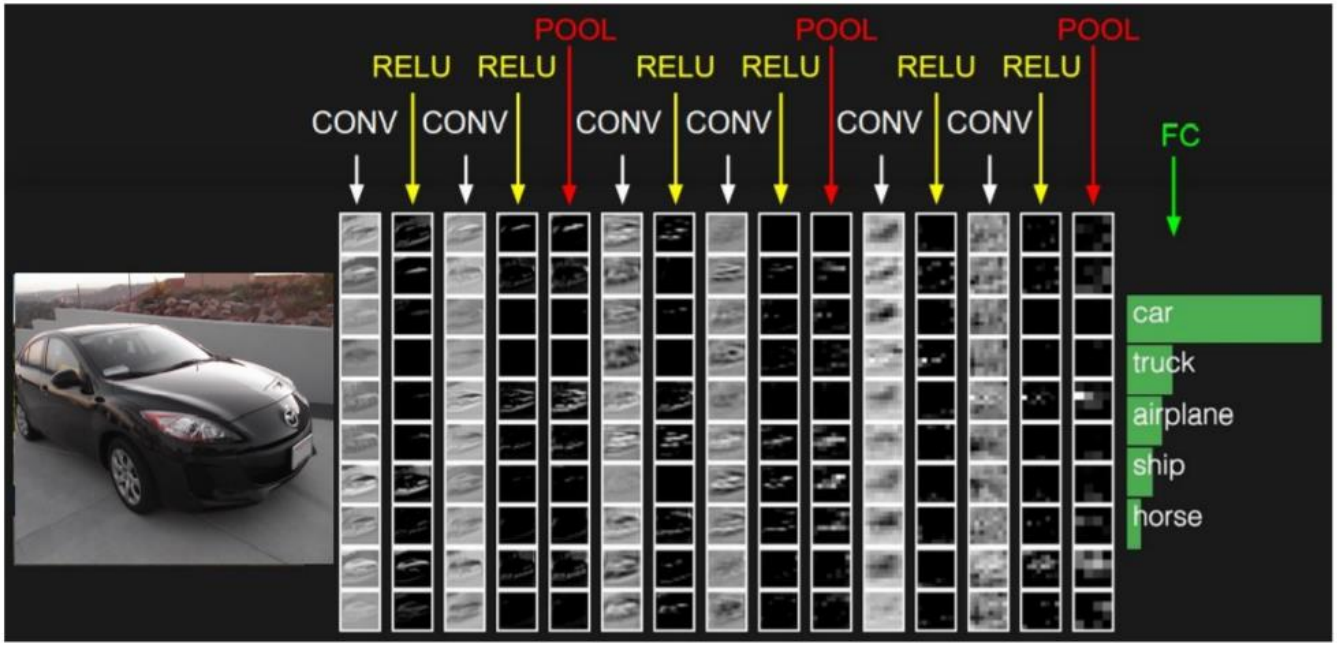


Column vector로 만든 후

앞선 강의에서 배운

fully connected neural network로!

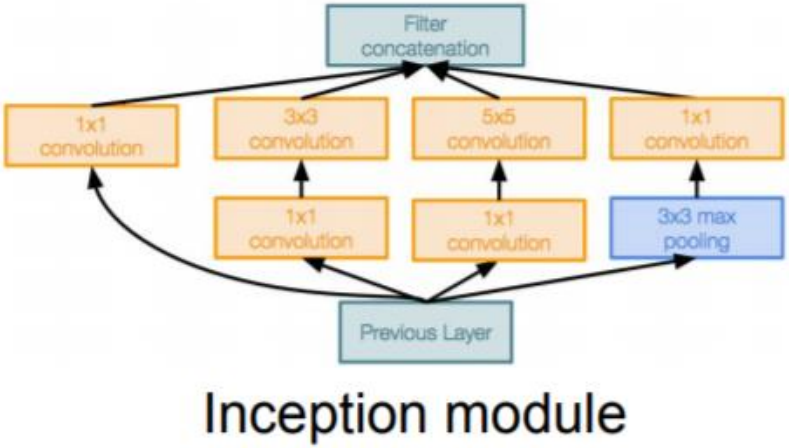
CNN 종합



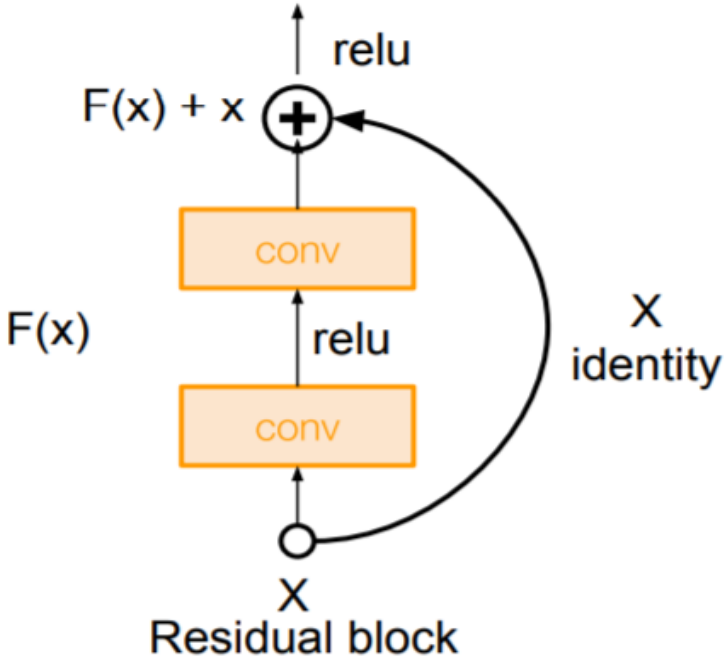
m은 1~5,
n은 보통 큰 수,
k는 0~2

일반적인 구조: ((CONV – RELU) * m – POOL) * n – (FC – RELU) * k

최신 모형

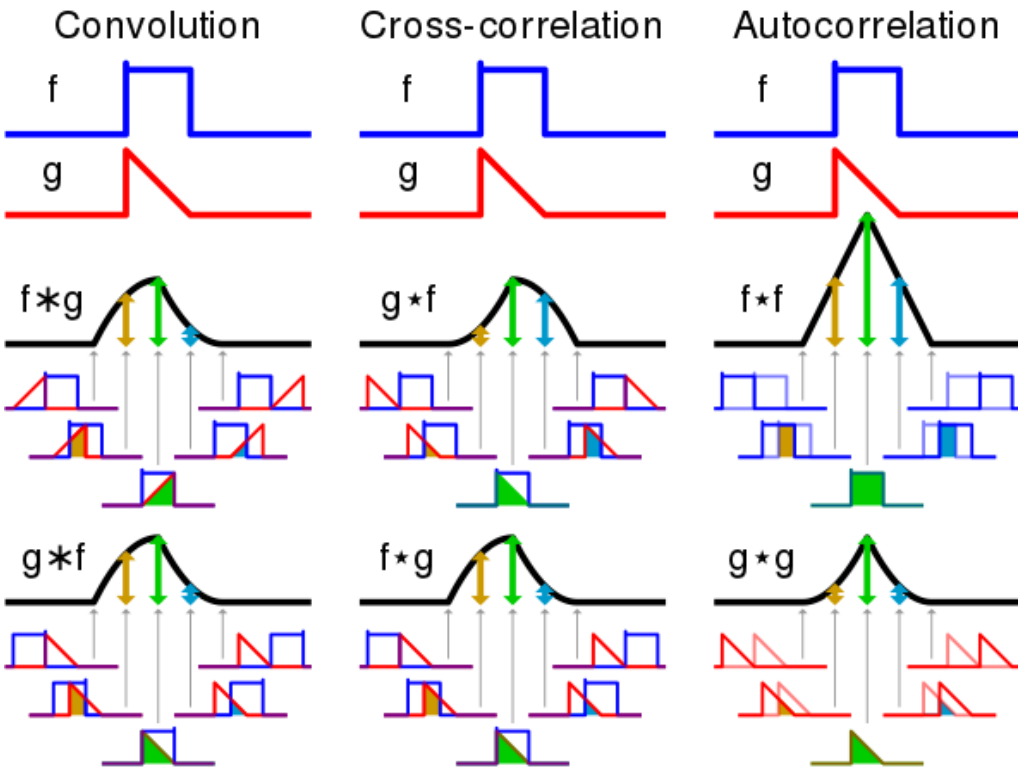


GooLeNet의 기본 모형



ResNet의 기본 모형

Convolution vs Cross-correlation



Convolution은 뒤집어서!

Cross-correlation은 안 뒤집고!

Auto correlation은 자기 자신과!

Convolution vs Cross-correlation

Convolution

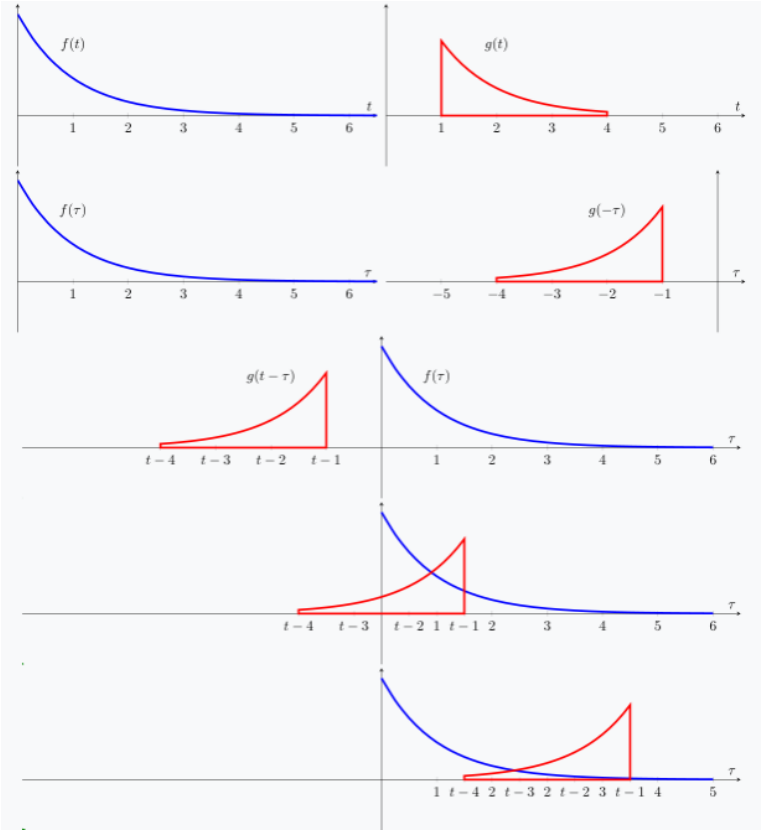
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m)$$

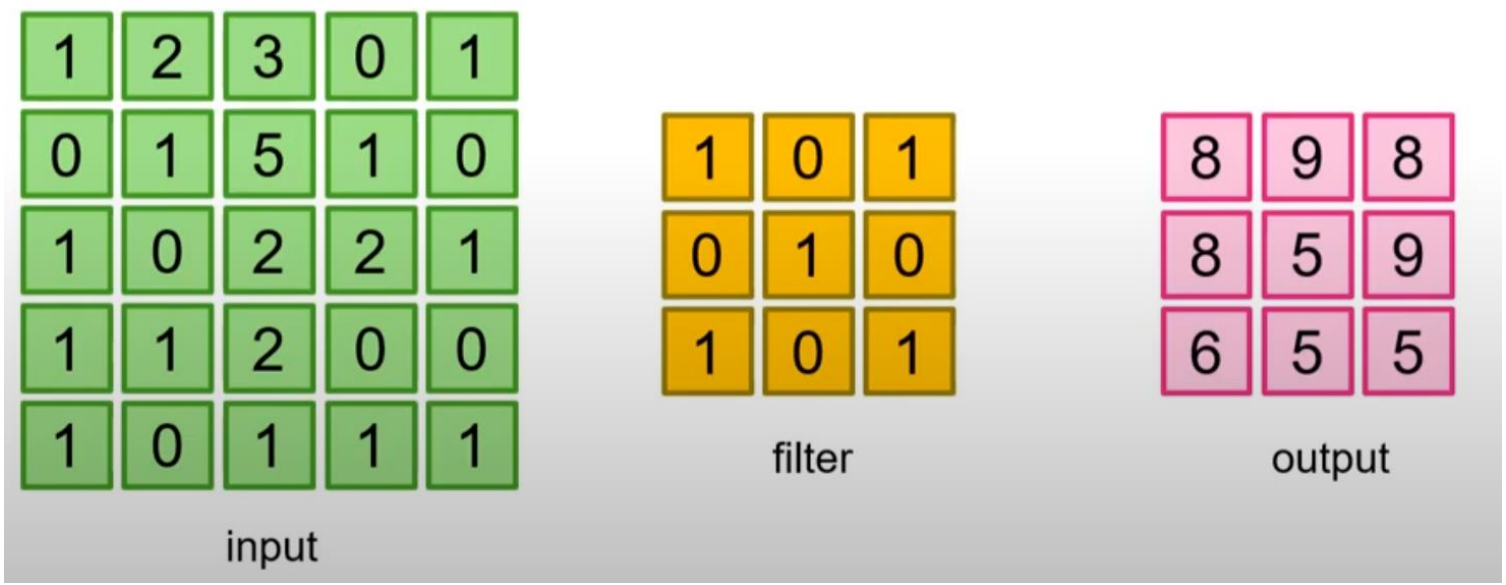
Cross-correlation

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t+\tau)d\tau$$

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n+m)$$



Convolution vs Cross-correlation



input $\approx f$

filter $\approx g$

output $\approx f * g$

숙제

1번

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

위 같은 흑백 이미지 파일이 있다

3 * 3 filter A

1	0	0
0	1	0
0	0	1

이미지 파일에

1 . zero padding 을 크기 1만큼 적용한 뒤 (즉 위 아래 양옆으로 0을 한겹만)

2 .stride 를 (1,1) 로 filter A 를 적용하고

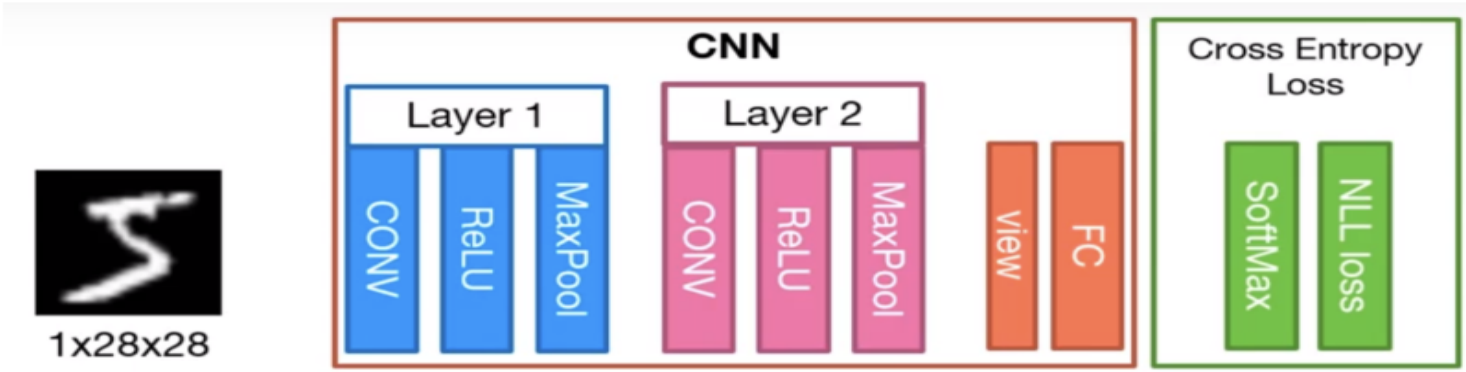
3. stride (2,2) 로 2*2 maxpooling 을 적용했을 때의 이미지 파일은 어떻게 생겼는지 예측해주세요!

실습해보자!

1 학습 단계(code 기준)

1. 라이브러리 가져오기
   - torch, torchvision, matplotlib 등등
2. GPU 사용 설정, seed 설정
   - cuda 사용 가능할 때 GPU연산, 불가할 때 CPU연산
   - 동일한 난수 생성을 위해 seed 설정
3. parameter 설정
   - learning_rate, training_epochs, batch_size
4. 데이터셋 가져오기, 데이터loader 만들기
5. CNN 학습 모델 만들기
6. Loss function(Criterion) 선택, 최적화 도구(Optimizer) 선택
7. 모델 학습 및 loss check (Criterion의 output)
8. 학습된 모델의 성능 확인

CNN 구조확인



- Layer 1
  Convolution Layer:(in_c=1, out_c=32, kernel_size=3, stride=1, padding=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)
  MaxPool Layer: (kernel_size=2, stride=2)
  view batch_size x [7,7,64] ⇒ batch_size x[3136]
- Fully_Connected Layer input=3136, output=10

코딩하기 이전에 MNIST dataset이 위 CNN을 통과하는 구조를 먼저 확인하고자 한다. Data가 CNN의 각 layer에 입력되어 통과하면서 input data size와 output data size가 달라진다. CNN 구조 확인은 output data size가 올바르게 나오는지 확인하는 것이다.

ReLU는 input data size와 output data size가 같기 때문에 CNN 구조 확인용 Layer에서는 제외되었다.

1. Input, Layer1, Layer 2 설정
2. CNN 통과 후 size check!

1.1 Input 설정

In [2]:
```
import torch
import torch.nn as nn
inputs=torch.Tensor(1,1,28,28)
print(inputs.shape)
```

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

- input에 들어가는 tensor는 (batch_size, channel, weight, width)이다

1.2 Layer1, Layer2 설정

In [3]:

```
#Layer1 설정
conv1=nn.Conv2d(1,32,3,padding=1)
pool=nn.MaxPool2d(2) #Layer2에도 사용

#Layer2 설정
conv2=nn.Conv2d(32, 64, 3, padding=1)
```

2.1 Layer1, Layer2 통과

In [4]:
```
Image(filename='Output_size.png')
```

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

$$Outputsize = \frac{28-3+(2*1)}{1} + 1 = 28$$

In [5]:
```
out=conv1(inputs)
out.shape
```

```
torch.Size([1, 32, 28, 28])
```

$$Outputsize = \frac{28-2+(2*0)}{2} + 1 = 14$$

In [6]:
```
out=pool(out)
out.shape
```

```
torch.Size([1, 32, 14, 14])
```

$$Outputsize = \frac{14-3+(2*1)}{1} + 1 = 14$$

In [7]:
```
out=conv2(out)
out.shape
```

```
torch.Size([1, 64, 14, 14])
```

$$Outputsize = \frac{14-2+(2*0)}{2} + 1 = 7$$

In [8]:
```
out=pool(out)
out.shape
```

```
torch.Size([1, 64, 7, 7])
```

2.2 view, FC 설정 후 size 확인!

In [9]:
```
out=out.view(out.size(0), -1)
out.shape
```

```
torch.Size([1, 3136])
```

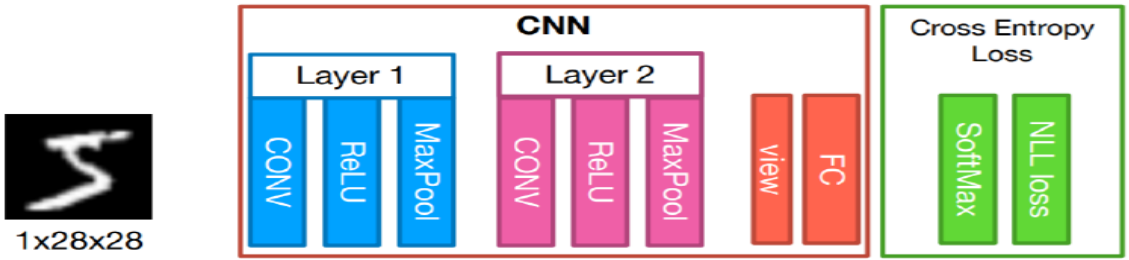
- out.size(0) out의 1번째 index출력, 즉 batch size(=1)에 해당한다.
- out.view(out.size(0), -1)은 batch size를 그대로 두고 나머지를 한 줄로 펼칠때 사용 (64x7x7=3136)

In [10]:
```
fc=nn.Linear(3136,10)
out=fc(out)
out
out.shape
```

```
torch.Size([1, 10])
```

본격적으로 모델링!

0. Layer 2개짜리 CNN 만들어보기!



- Layer 1

  Convolution Layer: (in_c=1, out_c=32, kernel_size=3, stride=1, padding=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)

  MaxPool Layer: (kernel_size=2, stride=2)
- view

  batch_size x [7,7,64] => batch_size x[3136]
- Fully_Connected Layer

  input=3136, output=10

###코드를 통해 더 자세히 살펴보자!

1) 라이브러리 가져오기

In [ ]:
```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn.init
```

2) GPU 사용 설정 + random value를 위한 seed설정

In [ ]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

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

3) 학습에 사용되는 parameters 설정

In [ ]:
```
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

4) 데이터셋 가져오기

In [ ]:
```
# 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)
```

4-1) (학습에 쓰기 편하게) loader 만들기

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

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

    def __init__(self):
        super(CNN, self).__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(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
        out = self.fc(out)
        return out
```

In [ ]:
```
# instantiate CNN model
model = CNN().to(device)
```

6) Loss function(Criterion)을 선택하고 최적화 도구 선택(optimizer)

In [ ]:
```
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

7) 모델 학습 및 loss check(Criterion의 output)

- Linear와 달리 data를 reshape할 필요가 없기 때문에 $X.view(-1, 28 * 28).to(device)$에서 $X.to(device)$로 변화

```
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!')
```

8) 학습된 모델의 성능을 확인한다.

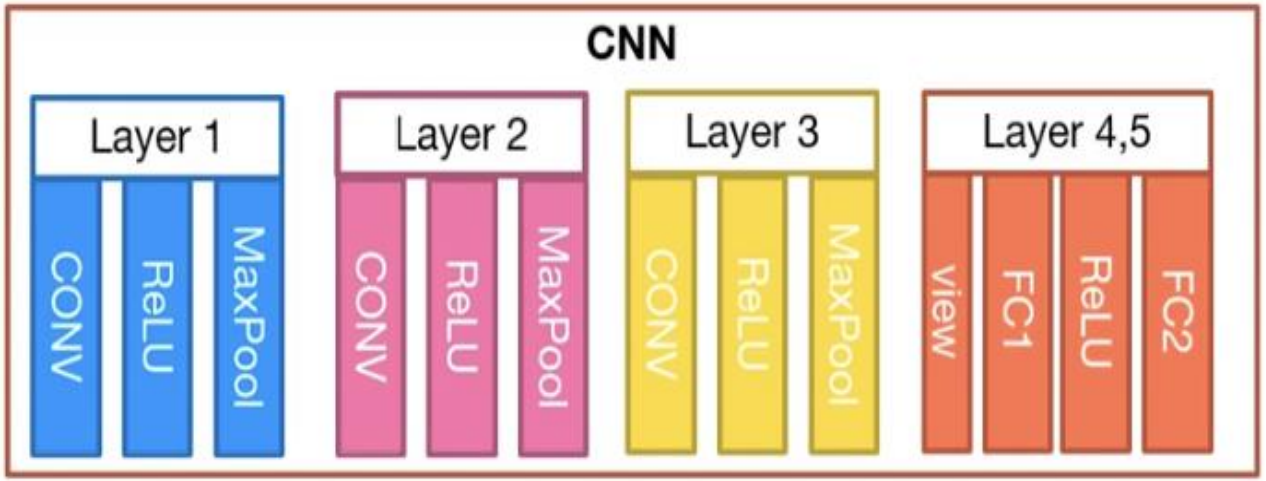
- Test data의 X 변수를 읽을 때 구조가 (data 수, channel수, 가로, 세로)의 순서로 변화

```
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())
```

```
/usr/local/lib/python3.6/dist-packages/torchvision/datasets/mnist.py:60: UserWarning: te
  warnings.warn("test_data has been renamed data")
/usr/local/lib/python3.6/dist-packages/torchvision/datasets/mnist.py:50: UserWarning: te
  warnings.warn("test_labels has been renamed targets")


Accuracy: 0.9855999946594238
```

레이어 더 쌓아보자!

1x28x28



4 CNN Layer 추가

- Layer 1

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

- **Layer 3**

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

- view

  batch_size x [4,4,128] ⇒ batch_size x[2048]

- **Layer 4**

  Fully Connected Layer:(in_c=4*4*128, out_c=625)

- **Layer 5**

  Fully Connected Layer:(in_c=625, out_c=10)

Layer 3의 Outputsize 계산:

- $Outputsize = \frac{7-3+(2*1)}{1} + 1 = 4$

class CNN(torch.nn.Module):

```
def __init__(self):
    super(CNN, self).__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))
```

추가되는 Layer

```
        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))

        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))

        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
```

결론

- **Layer 3**의 구조
  Image In: (?, 7, 7, 64)
  -> Conv ->(?, 7, 7, 128)
  -> Pool ->(?, 4, 4, 128)

- **Layer4**FC
  (?, 4, 4, 128) -> 625 outputs

- **Layer5**FC
  625 -> 10 outputs

Layer을 추가한 뒤 학습 결과 cost가 오히려 상승한다. 따라서 효율적인 방법으로 Layer를 쌓는 것이 중요하다!

숙제

2번(앞의 코드를 활용한 문제입니다!)

1. Layer1의 Kernel size를 4로 늘리고, Layer3의 stride를 2로 늘려라
2. Layer3의 Output size를 구하여라
3. Cost와 accuracy가 기존 CNN모델에 비해 어떻게 변할 것인가?
4. 코딩하여 cost와 accuracy를 확인해보자!

Visdom

1. 설치

   명령 프롬프트/ 터미널에

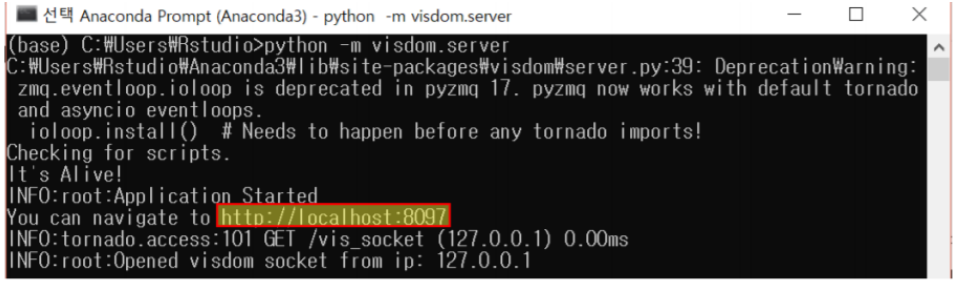
   pip install visdom 입력

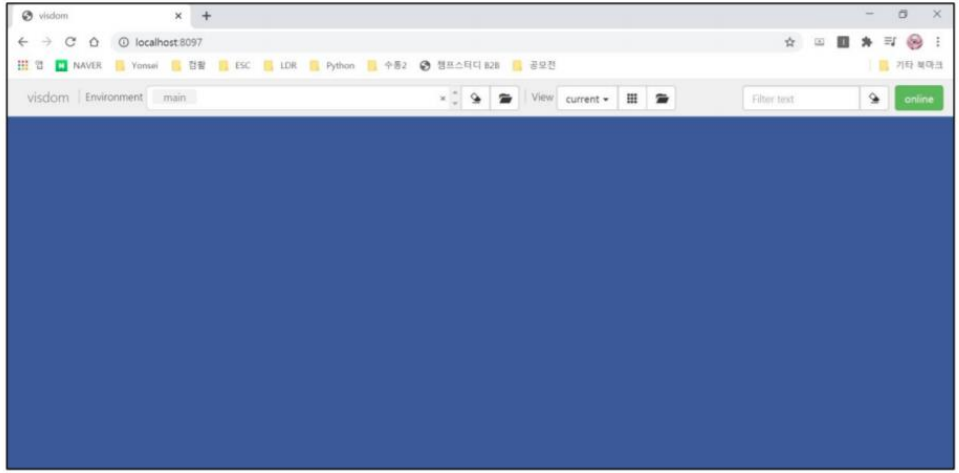
2. 서버 켜기

   명령 프롬프트(아나콘다 프롬프트 등 사용하시는 프롬프트 가능)에

   python –m visdom.server 입력

Visdom





Visdom 서버 주소(하이라이트된 부분)으로 가면 visdom 창이 켜진다

Visdom

3. 사용법

※ 반드시 2번의 서버 켜놓은 상태에서 진행하세요!

10-3-1 Visdom Example

```
import torch
import torch.nn as nn

import torchvision
import torchvision.datasets as dsets
```

import visdom

Jupyter Notebook > Terminal 를 새로 켜서 `python -m visdom.server` 를 입력하세요!

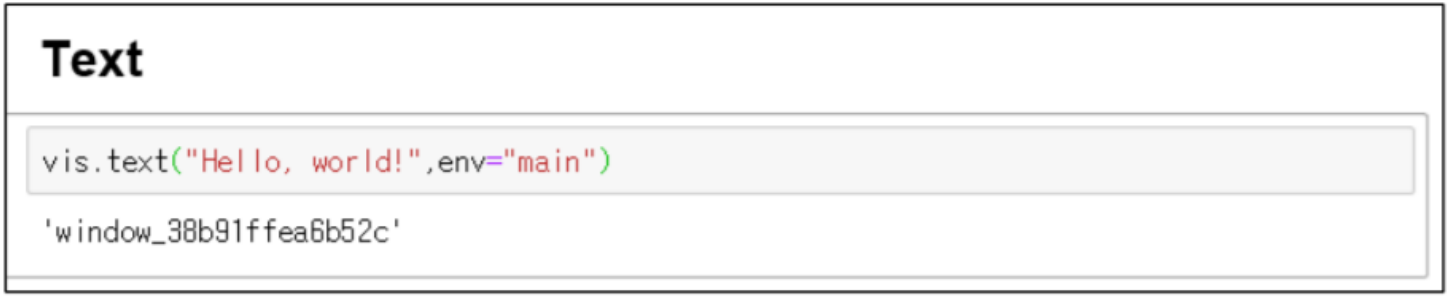
```
import visdom
vis = visdom.Visdom()
```

```
Setting up a new session...
```

Visdom

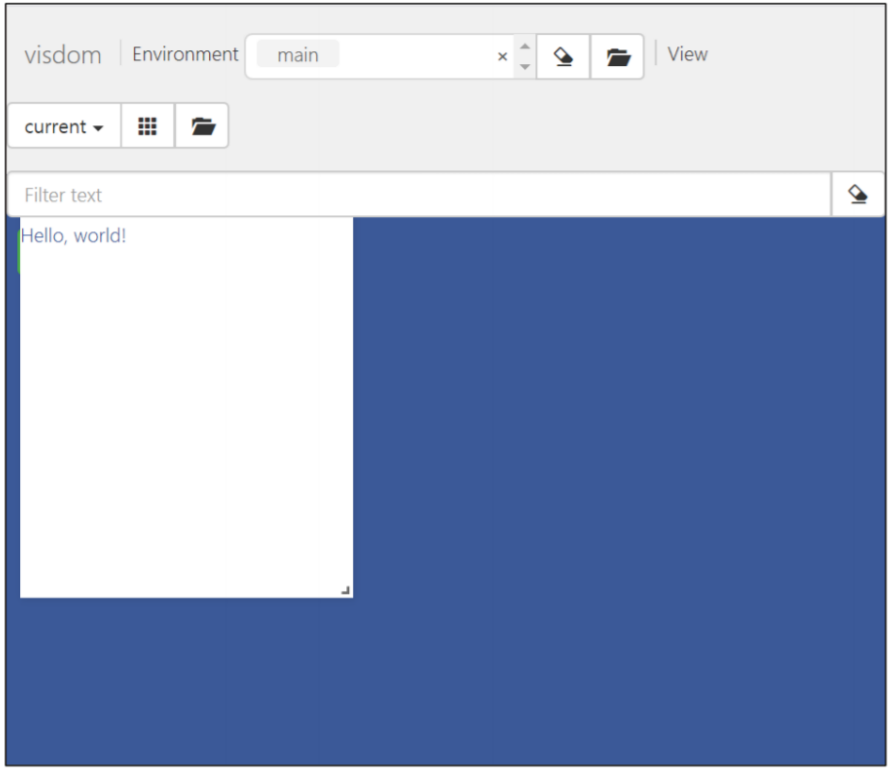
3-1. text

env = "main": 나중에 visdom 창을 닫을 때 main에 있는 모든 것을 한 번에 닫을 수 있도록 한다



Visdom

3-1. text



Visdom

3-2. image

a=torch.randn(3, 200, 200) : RGB 2인 200 × 200짜리 이미지 출력

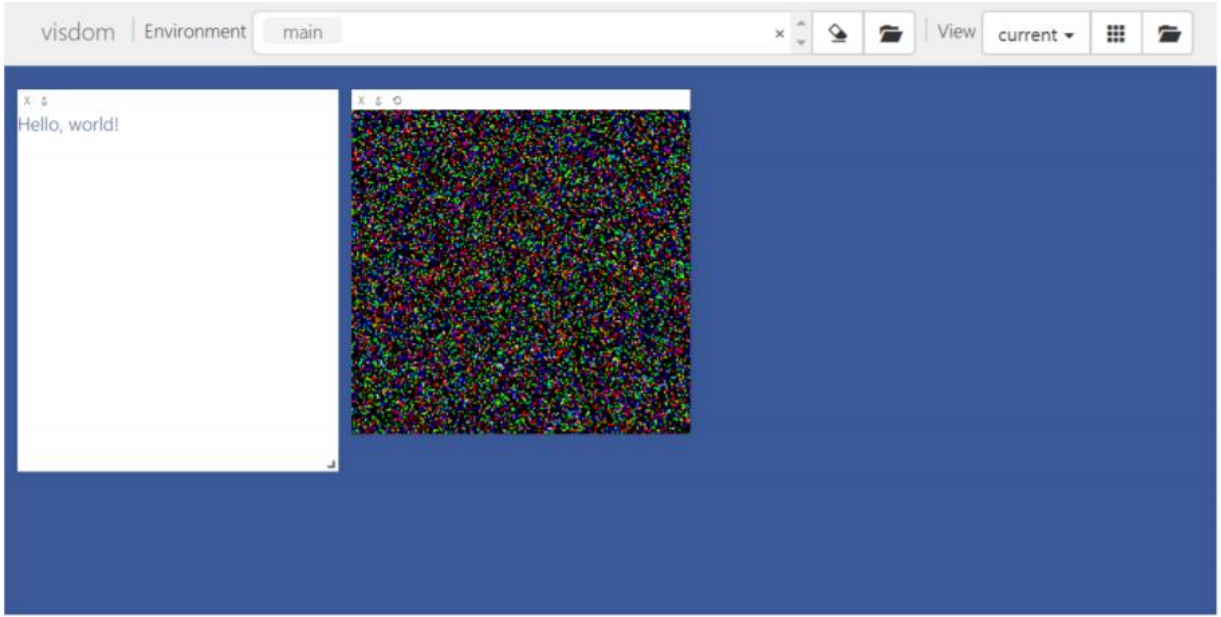
image

```
a=torch.randn(3,200,200)
vis.image(a)
```

```
'window_38b92341813538'
```

Visdom

3-2. image



이미지 윗부분의 창을 드래그해 **위치** 이동 가능

우측 하단에 마우스를 가져다 놓으면 화살표로 바뀌며 이미지 **크기** 조정 가능

Visdom

3-2. image

28 × 28 짜리 이미지 3개 출력

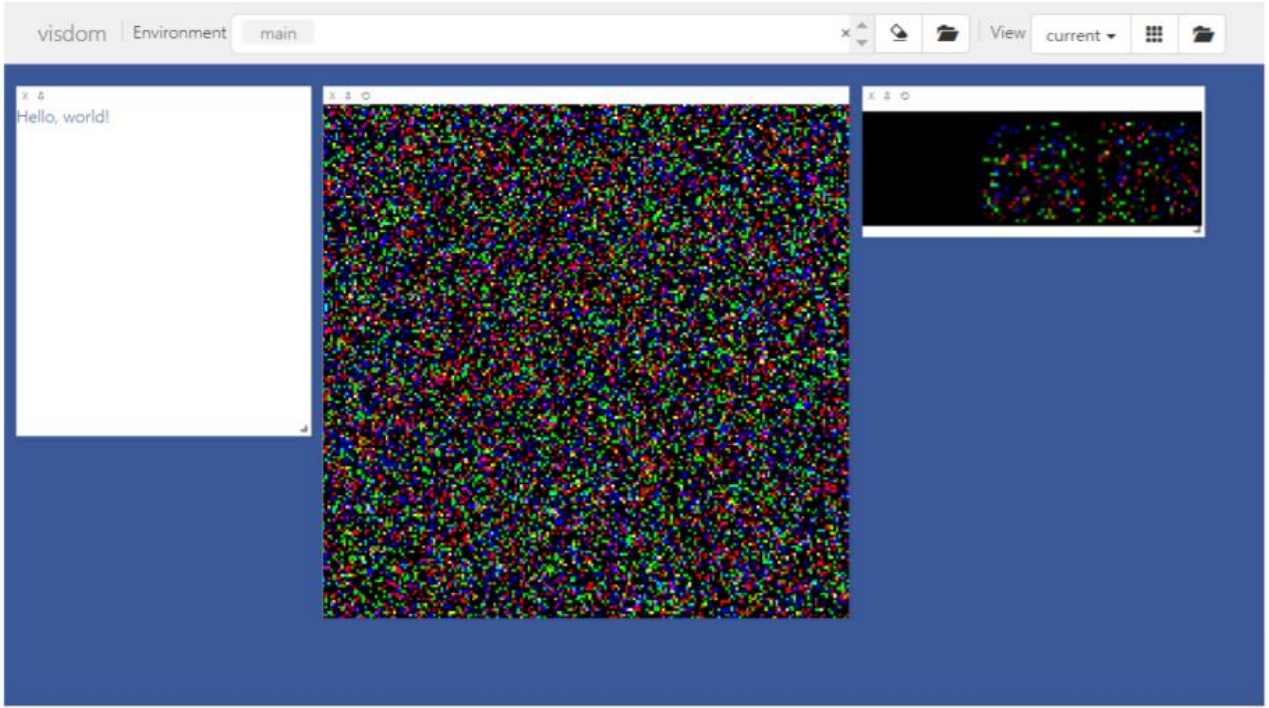
images

```
vis.images(torch.Tensor(3,3,28,28))
```

```
'window_38b92471480306'
```

Visdom

3-2. image



Visdom

3-2. image

데이터 다운로드 후

getitem으로

첫번째 item 가져오기

example (using MNIST and CIFAR10)

```
# 시간이 좀 걸립니다.
MNIST = dsets.MNIST(root="./MNIST_data",train = True,transform=torchvision.transforms.
cifar10 = dsets.CIFAR10(root="./cifar10",train = True, transform=torchvision.transform
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./cifar10₩cifa
r-10-python.tar.gz

170500096/? [01:20<00:00, 3271794.45it/s]

Extracting ./cifar10₩cifar-10-python.tar.gz to ./cifar10
```

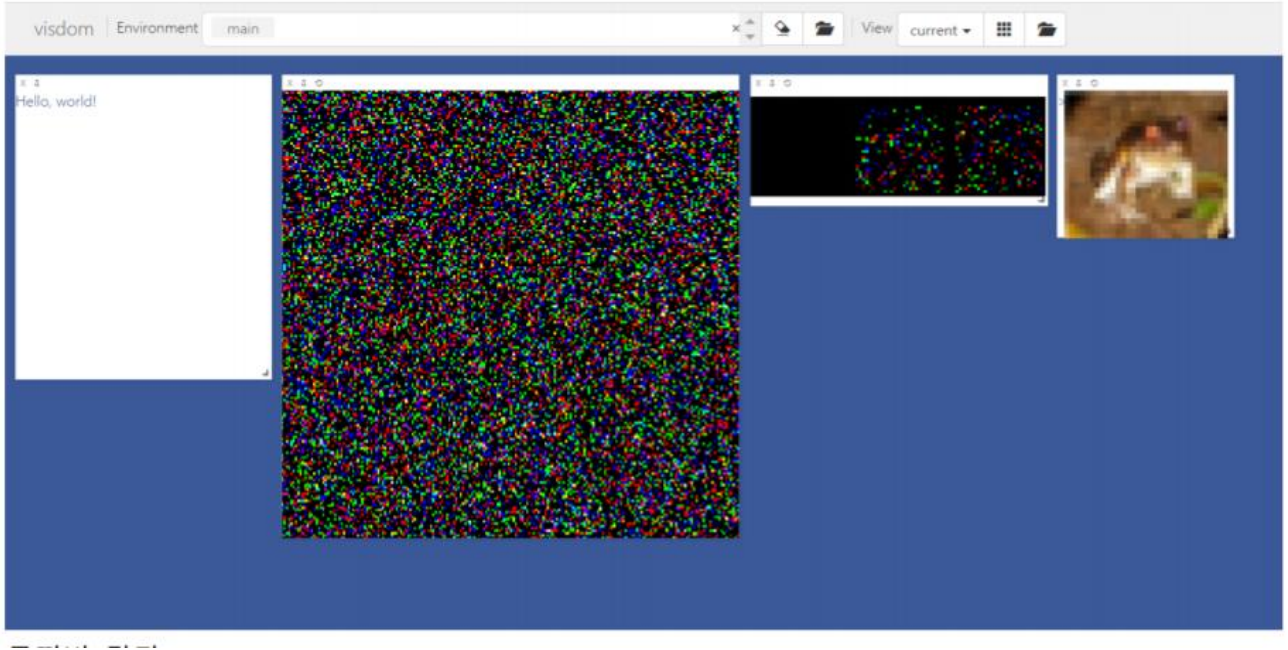
CIFAR10

```
data = cifar10.__getitem__(0)
# 가지고 온 cifar10에서 getitem function을 이용해 첫번째 item 가져오기
print(data[0].shape)
vis.images(data[0],env="main")
```

```
torch.Size([3, 32, 32])

'window_38b9250abce7b0'
```

Visdom

3-2. image



Visdom

3-2. image

MNIST example

MNIST

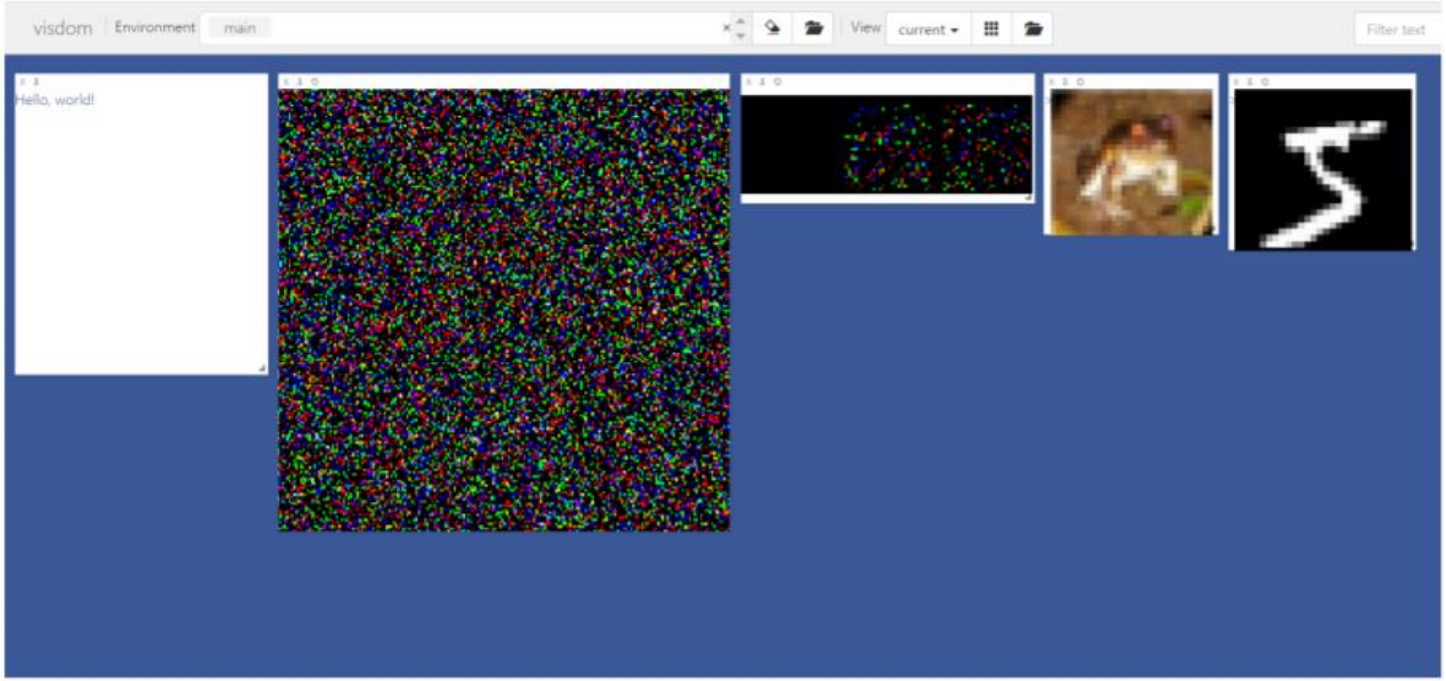
```
data = MNIST.__getitem__(0)
print(data[0].shape)
vis.images(data[0],env="main")
```

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

```
'window_38b925782fac2a'
```

Visdom

3-2. image



Visdom

3-2. image

MNIST 여러 개의 이미지 한 번에 불러오기

Check dataset

```
# data loader를 통해 batch_size에 입력한 수의 이미지를 한번에 볼 수 있다
data_loader = torch.utils.data.DataLoader(dataset = MNIST,
                                          batch_size = 32,
                                          shuffle = False)
```

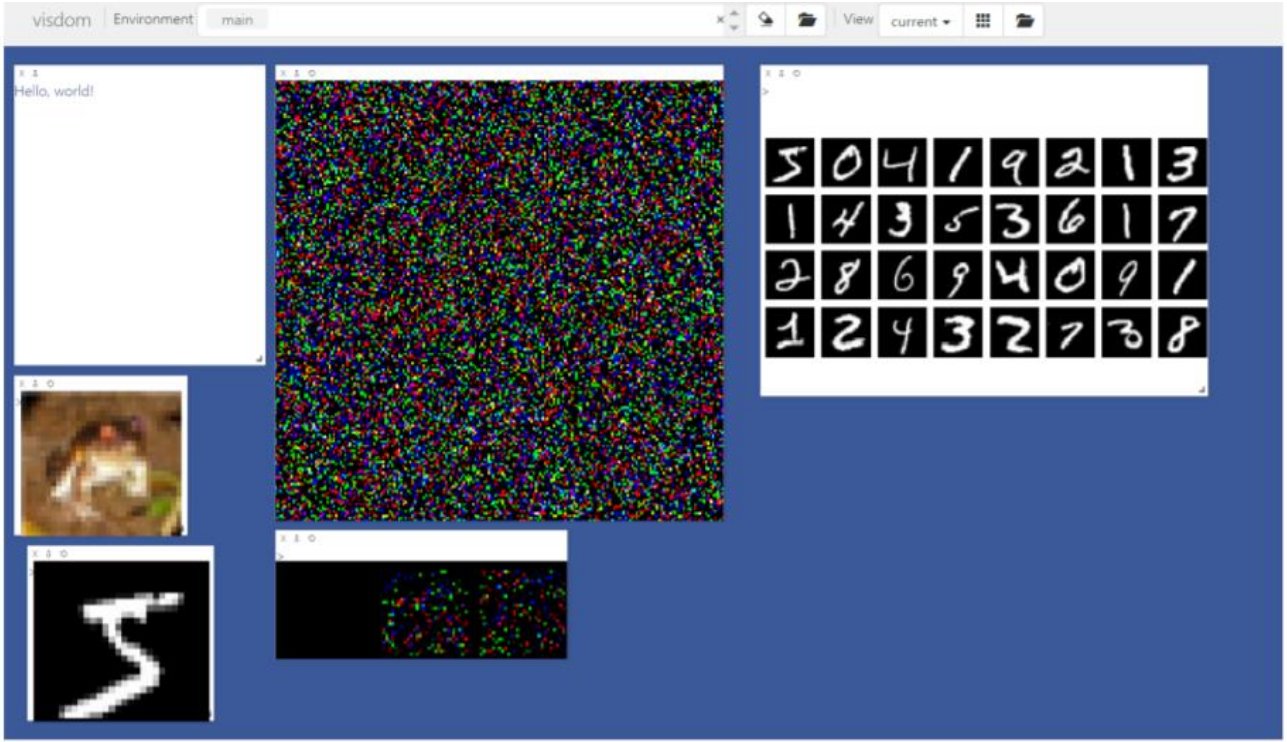
```
for num, value in enumerate(data_loader):
    value = value[0]
    print(value.shape)
    vis.images(value)
    break
```

```
torch.Size([32, 1, 28, 28])
```

```
vis.close(env="main")
```

Visdom

3-2. image



Visdom

3-3. 창 닫기

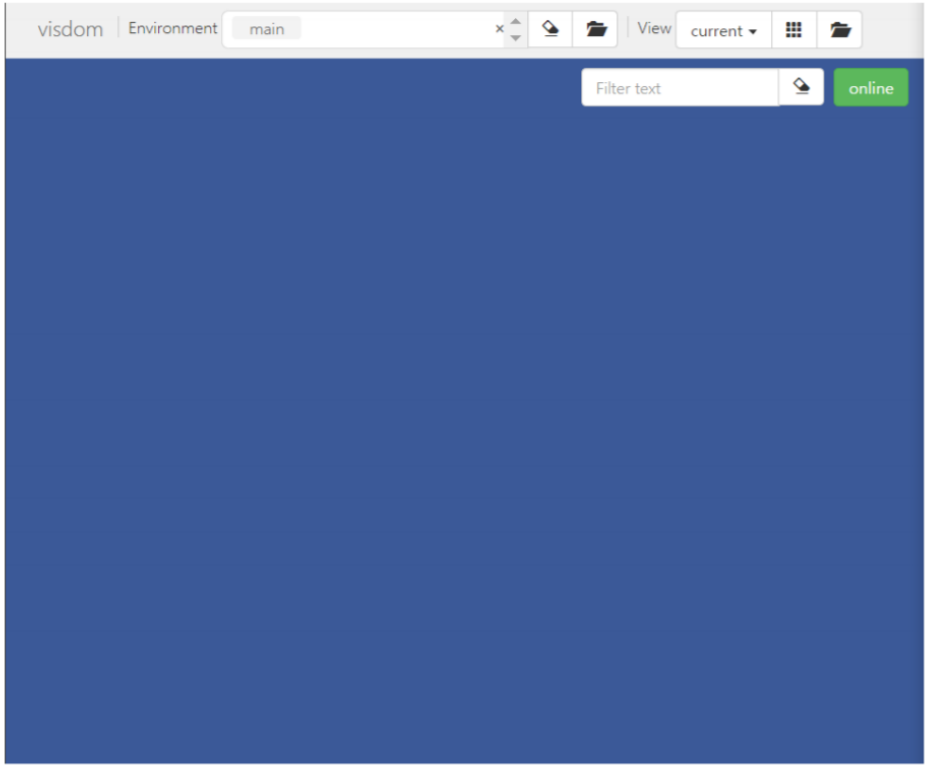
environment가 main인 창 모두 닫기

```
vis.close(env="main")
```

```
''
```

Visdom

3-3. 창 닫기



Visdom

4. line plot

학습을 진행하며 나타나는 loss의 변화를 그래프로 표현

X값을 따로 주지 않고 Y값만 설정

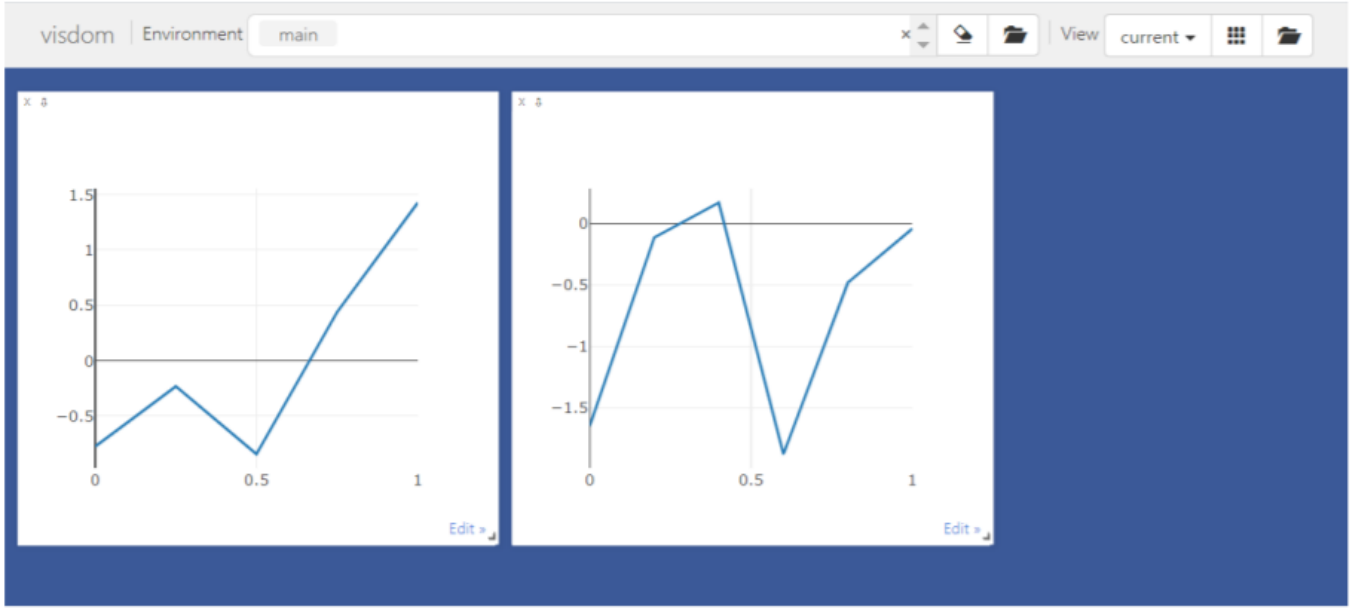
이런 식으로 X를 따로 지정하지 않으면 X는 무조건 0과 1 사이의 값을 갖는다

```
Y_data = torch.randn(5)    # y data 선언: random으로 5개
plt = vis.line (Y=Y_data)  # 그래프 반환
```

```
Y_data = torch.randn(6)
plt = vis.line (Y=Y_data)
```

Visdom

4. line plot

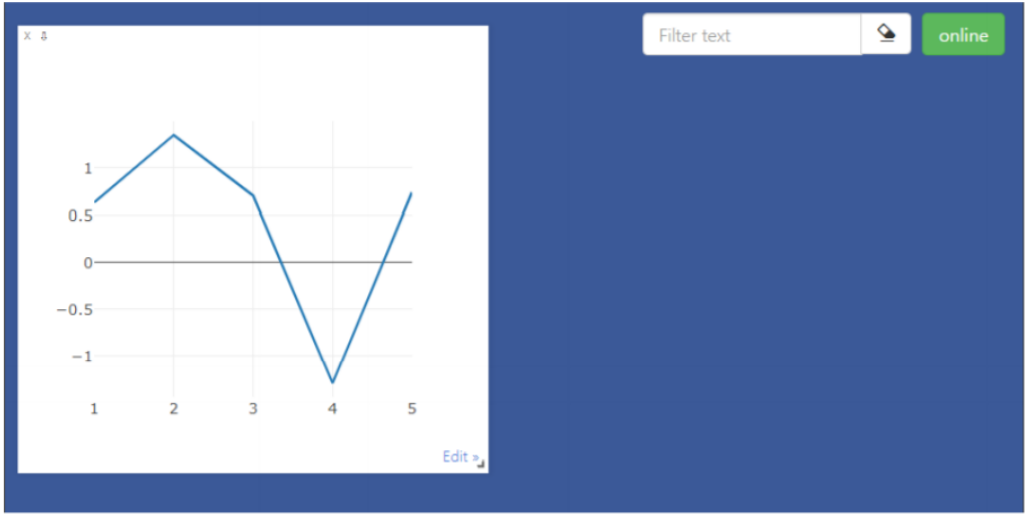


Visdom

4. line plot

X와 Y값 모두 주기

```
Y_data = torch.randn(5)
X_data = torch.Tensor([1,2,3,4,5])
plt = vis.line(Y=Y_data, X=X_data)
```



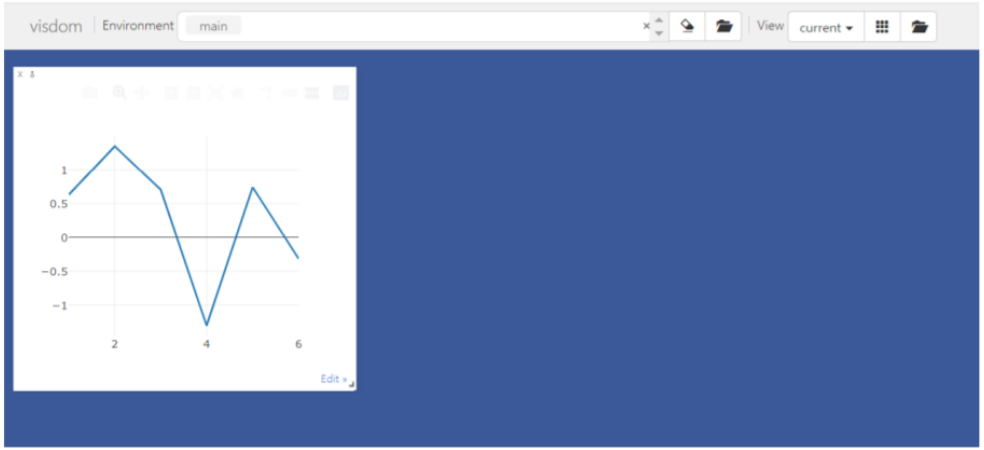
Visdom

4. line plot

Line Update

```
Y_append = torch.randn(1)    # 들어가야 할 loss 값 정의
X_append = torch.Tensor([6]) # 들어가야 할 X 값 정의

vis.line(Y=Y_append, X=X_append, win=plt, update='append')
# win: update할 window의 이름
# update = 'append' : 확장하는 방식으로 update
```

```
'window_38b927c05fef4a'
```



끝 부분에 값이 추가된 것을 확인할 수 있다

Visdom

4. line plot

하나의 line plot 창에서 두 개의 line을 그리는 방법

```
num = torch.Tensor(list(range(0,10)))
print(num.shape)
num = num.view(-1,1)
print(num.shape)
num = torch.cat((num,num),dim=1)
print(num.shape)

plt = vis.line(Y=torch.randn(10,2), X = num)
```

```
torch.Size([10])
torch.Size([10, 1])
torch.Size([10, 2])
```

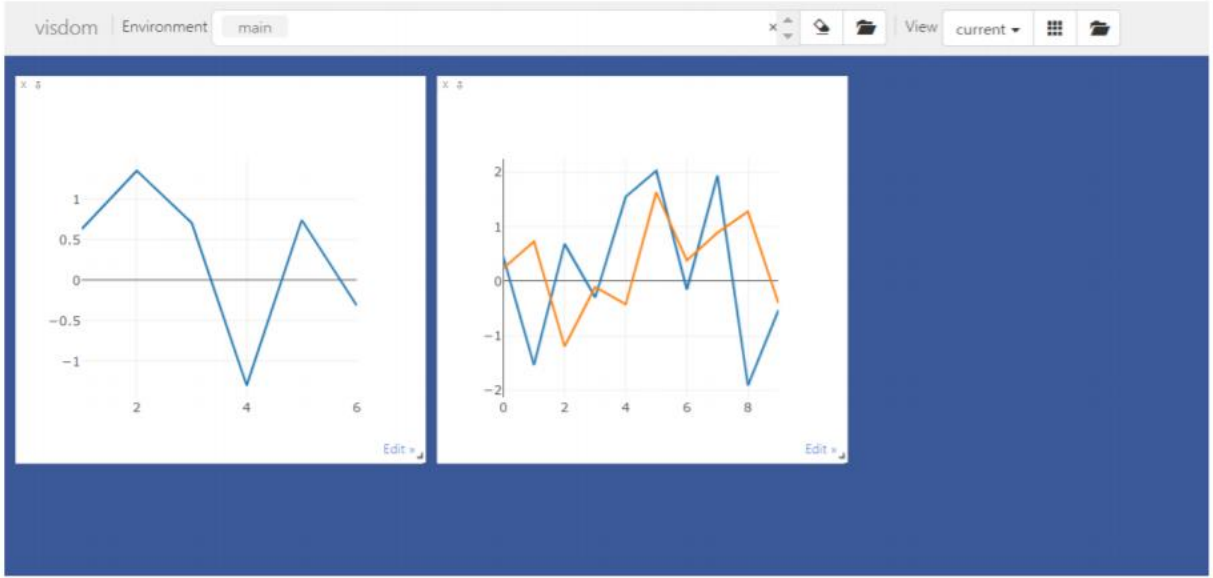
tensor의 reshpae을 위해 .view 이용

두 tensor의 결합을 위해 torch.cat 이용

line을 두 개를 그리려면 Y와 같은 shpae을 가지고 있는 index값을 넣어주어야 한다

Visdom

4. line plot

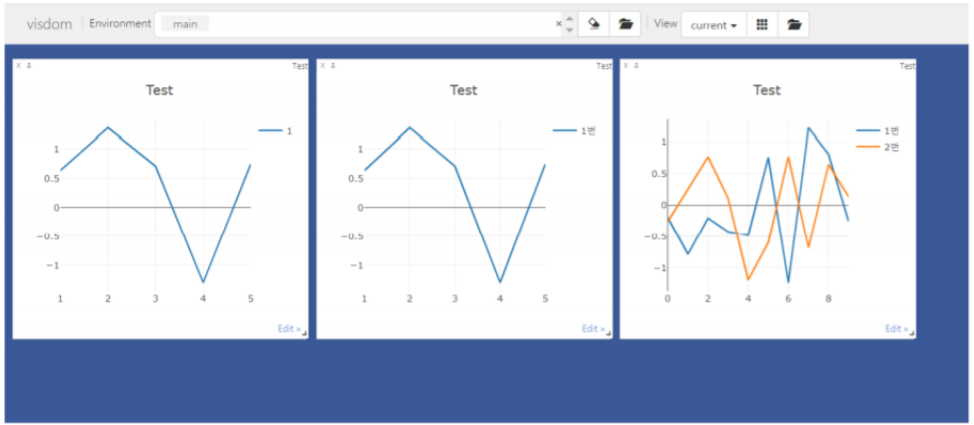


Visdom

4. line plot

Line info

```
# 그래프에 제목, legend 넣기
plt = vis.line(Y=Y_data, X=X_data, opts = dict(title='Test', showlegend=True))

# legend에 축 이름 넣기
plt = vis.line(Y=Y_data, X=X_data, opts = dict(title='Test', legend = ['1번'],showlegend=True))

# line 여러 개인 plot에 lengend 축 이름 넣기
plt = vis.line(Y=torch.randn(10,2), X = num, opts=dict(title='Test', legend=['1번','2번'],showlegend=True))
```




```
# `opts.linecolor` : line colors
# `opts.dash` : line dash type (`np.array`; default = None)
```

Visdom

4. line plot

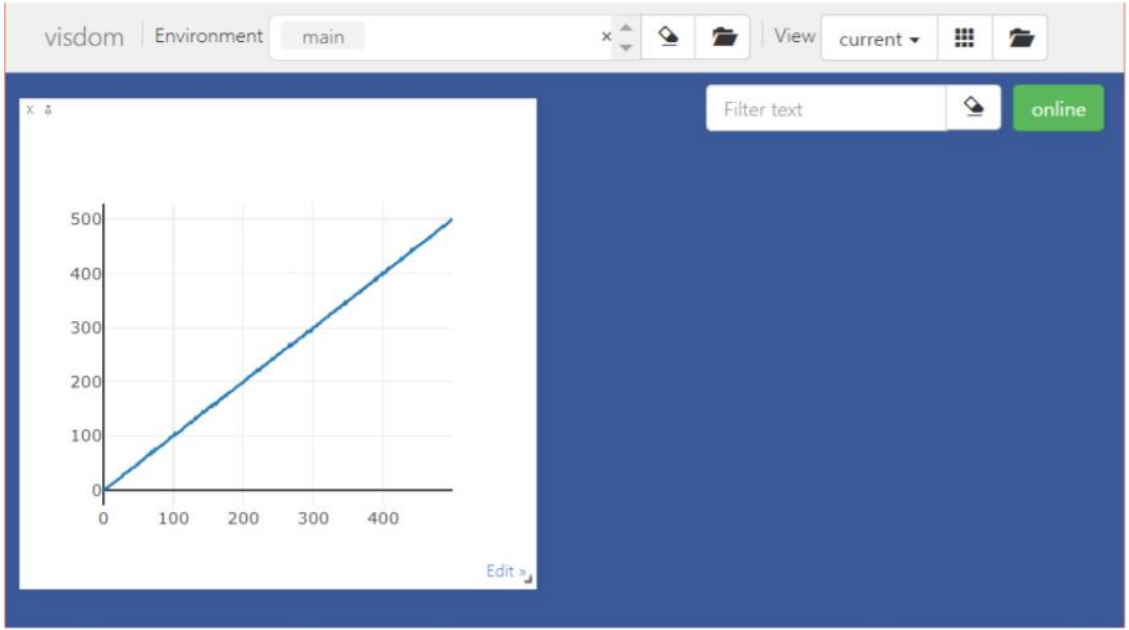
loss value가 update되는 function

```
def loss_tracker(loss_plot, loss_value, num):
    # loss_plot: 업데이트할 plot의 이름
    # num: X값의 index
    '''num, loss_value, are Tensor'''
    vis.line(X=num,
             Y=loss_value,
             win = loss_plot,
             update='append'
             )
```

```
plt = vis.line(Y=torch.Tensor(1).zero_())
# plt라는 plot에 loss값 update

for i in range(500):
    loss = torch.randn(1) + i
    loss_tracker(plt, loss, torch.Tensor([i]))
```

Visdom

4. line plot



두번째 block의 코드가 돌아가는 동안 visdom 상을 보면 update 되며 그래프가 변화하는 과정을 실시간으로 지켜볼 수 있다!

Visdom

5. MNIST-CNN with Visdom

MNIST-CNN에 visdom function을 집어넣어 loss값을 추적하는 그래프 그리기

train with loss_tracker

```
#training
total_batch = len(data_loader)

for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader:
        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:{}] cost = {}'.format(epoch+1, avg_cost))
    loss_tracker(loss_plt, torch.Tensor([avg_cost]), torch.Tensor([epoch]))
    # 한 epoch이 끝날 때마다 loss값의 평균을 넣는다
print('Learning Finished!')
```

Image Folder

Image Folder는 아래와 같이 계층적 폴더구조를 가진 이미지 데이터를 불러올 때 사용할 수 있다.

```
Origin_data
├── grey
│   ├── 1.png
│   ├── 2.png
│   ├── 3.png
│   ├── 4.png
│   ├── 5.png
│   └── 6.png
└── red
    ├── 1.png
    ├── 2.png
    ├── 3.png
    └── 4.png

2 directories(or more), 10 files(or more)
```

source: https://ndb796.tistory.com/373

모두딥 강의는 ImageFolder를 생성하고 dataset을 변환하여 저장하는 전반부와 변환된 dataset을 이용하여 CNN 학습을 진행하는 후반부로 양분됩니다.

Image Folder

우선 Grey, red 의자 같이 구분이 쉬운(색깔) 두 가지 카테고리별 사진을 준비하여 각각 폴더에 저장한다.

1. 라이브러리 가져오기

```
import torchvision

from torchvision import transforms ## img 파일을 pytorch에 맞게 변환 (ToTensor), Resize 등

from torch.utils.data import DataLoader ## dataset 불러오는 함수
```

Image Folder

2. 사진이 담긴 파일 경로 지정

```
trans = transforms.Compose([ transforms.Resize((64,128)),
                            ]) # 여기선 하나지만 여러 transforms를 Compose로
                            묶는 것.
```

보통의 사진 파일은 이전에 다루었던 MNIST(28*28)와 달리 1922*2560 정도로 크기가 크고 파일별로 크기가 상이한 경우가 많습니다(1440*1080, 가로 세로 방향에 따라 1080*1440 등등) resize를 통해 조금 더 작은 크기로 통일시킨다고 생각하면 될 것 같아요

Image Folder

2. 사진이 담긴 파일 경로 지정

```
train_data = torchvision.datasets.ImageFolder(root= './origin _data',
transform=trans)
```

#MNIST dataset을 불러올 때 코드와 유사하다. MNIST/CIFAR10자리에 ImageFolder를 적어주기만 하면 된다. 즉 우리가 보통 dsets로 import하는 torchvision.datasets에 ImageFolder도 있다.

Image Folder

2. 사진이 담긴 파일 경로 지정

```
for num, value in enumerate(train_data):
    data, label = value
    print(num, data, label)
    if(label == 0):   ## 같은 폴더 내의 파일에 같은 label이 부여됨
        data.save('origin_data/train/grey/%d_%d.jpeg'%(num, label))
    else:
        data.save('origin_data/train/red/%d_%d.jpeg'%(num, label))
```

resize된 새로운 이미지 파일들이 label에 따라 괄호 안 경로에 새롭게 저장됨

Image Folder

1. 라이브러리 가져오기

```
import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms
```

2. Random seed

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

torch.manual_seed(777)

if device =='cuda':

    torch.cuda.manual_seed_all(777)
```

Image Folder

2. Dataset 가져오고 dataloader 설정

```
trans = transforms.Compose([
    transforms.ToTensor()
]) ## 아까 resize한 데이터를 ToTensor를 이용해 pytorch에 맞게 다시 변환.
```

```
train_data = torchvision.datasets.ImageFolder(root='./origin_data/train', transform=trans)

data_loader = torch.utils.data.DataLoader(dataset = train_data,
                                          batch_size = 8,
                                          shuffle = True,
                                          num_workers=2) # multi-process data loading
```

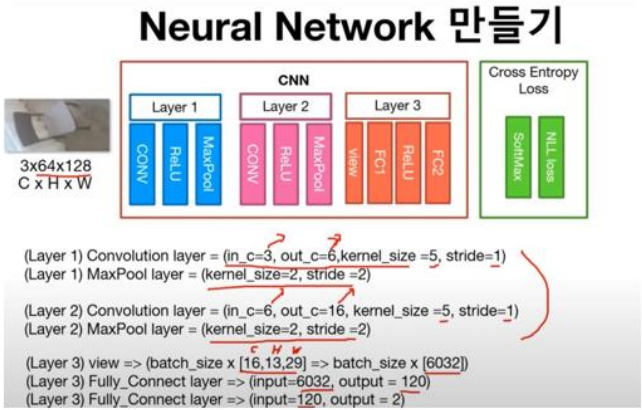
num_workers = 학습 도중 CPU 의 작업을 몇 개의 코어를 사용해서 진행할지에 대한 설정 파라미터. 해당 환경에서 사용 가능한 코어의 개수를 확인해보고 학습 외의 작업에 영향을 주지 않을 정도의 숫자로 설정해주시는 것이 좋다고 합니다.

Source: https://m.blog.naver.com/qbxlvnf11/221728476511

Image Folder

4. 모델

아래 그림과 같은 구조로 모델 설정.



source: https://www.youtube.com/watch?v=NqwUmCZbjNA&list=PLQ28Nx3M4JrhkqBVIXg-i5_CVVoS1UzAv&index=23

Image Folder

4. 모델

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()    ##kernel_size = 5, input channel = 3(RGB)
        self.layer1 = nn.Sequential(nn.Conv2d(3,6,5), nn.ReLU(), nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(nn.Conv2d(6,16,5),nn.ReLU(),nn.MaxPool2d(2))
        self.layer3 = nn.Sequential(nn.Linear(16*13*29, 120),nn.ReLU(),nn.Linear(120,2))
        ## 13*29는 resize에서 지정한 64*128이 CNN과 Maxpooling 과정을 거쳐서 나온 값이고
16은 input channel 3(RGB)의 최종 output_channel. 세 수치를 곱한 6032가 FC의 input이 됨
```

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.shape[0], -1) ##flattening for FC
        out = self.layer3(out)
        return out
```

5) Optimizer/ criterion

```
optimizer = torch.optim.Adam(net.parameters(), lr=0.0005)
```

Image Folder

5. optimizer/ criterion

```
optimizer = torch.optim.Adam(net.parameters(), lr=0.0005)
loss_func = nn.CrossEntropyLoss().to(device)
```

6. 학습

```
total_batch = len(data_loader)
epochs = 7
```

Image Folder

6. 학습

```
for epoch in range(epochs):
    avg_cost = 0.0
    for num, data in enumerate(data_loader):   ##training dataset
        imgs, labels = data
        imgs = imgs.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        out = net(imgs)
        loss = loss_func(out, labels)
        loss.backward()
        optimizer.step()

        avg_cost += loss / total_batch

    print('[Epoch:{}] cost = {}'.format(epoch+1, avg_cost))
print('Learning Finished!')
```

Image Folder

7. 모델 저장

```
torch.save(net.state_dict(), "./model/model.pth") #model 폴더에 model명으로 모델 저장


new_net = CNN().to(device) # 불러올 때 모형은 똑같지만 net이 아니라 새롭게 네트워크 지정

new_net.load_state_dict(torch.load('./model/model.pth')) #저장된 모델 불러오기
```

Image Folder

8. 성능 평가

```
trans=torchvision.transforms.Compose([
    transforms.Resize((64,128)),
    transforms.ToTensor()
])  ## 아까 train set에는 변환 두 가지를 나누어 했지만 test set에는 한번에 적용!
test_data = torchvision.datasets.ImageFolder(root='./origin_data/test_data', transform=trans)
test_set = torch.utils.data.DataLoader(dataset = test_data, batch_size = len(test_data))
```

여기서 주의할 것이 path가 origin_data 폴더의 test_data 폴더인데, test_data의 하위폴더에 이미지 파일이 있어야 합니다. 그냥 test_data에 폴더가 아닌 이미지 파일들로 저장이 되어있으면 이미지 파일이 없다고 뜹니다. 폴더 단위로 인식한다고 생각하시면 좋을 것 같아요!

Image Folder

8. 성능 평가

```
with torch.no_grad():
    for num, data in enumerate(test_set):
        imgs, label = data
        imgs = imgs.to(device)
        label = label.to(device)

        prediction = net(imgs)
        correct_prediction = torch.argmax(prediction, 1) == label

        accuracy = correct_prediction.float().mean()
        print('Accuracy:', accuracy.item())
```

숙제

1번

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

위 같은 흑백 이미지 파일이 있다

3 * 3 filter A

1	0	0
0	1	0
0	0	1

이미지 파일에

1 . zero padding 을 크기 1만큼 적용한 뒤 (즉 위 아래 양옆으로 0을 한겹만)

2 .stride 를 (1,1) 로 filter A 를 적용하고

3. stride (2,2) 로 2*2 maxpooling 을 적용했을 때의 이미지 파일은 어떻게 생겼는지 예측해주세요!

숙제

2번(앞의 코드를 활용한 문제입니다!)

1. Layer1의 Kernel size를 4로 늘리고, Layer3의 stride를 2로 늘려라
2. Layer3의 Output size를 구하여라
3. Cost와 accuracy가 기존 CNN모델에 비해 어떻게 변할 것인가?
4. 코딩하여 cost와 accuracy를 확인해보자!