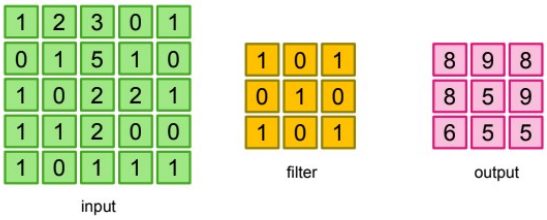
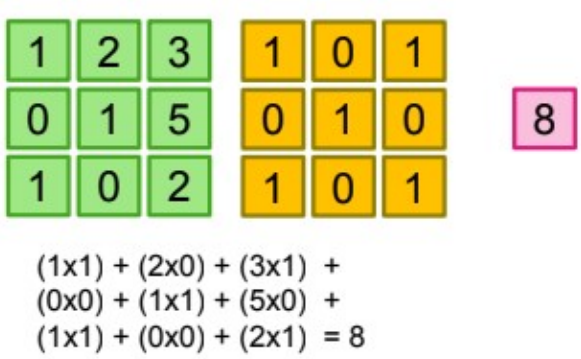
20.05.18

주제	Convolution / MNIST CNN

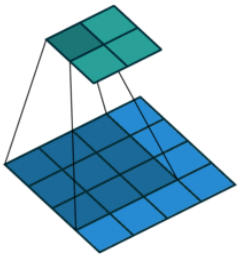
10-1 Convolution

Convolution의 개념





이미지 위에서 stride 값 만큼 filter를 이동시키며
겹쳐지는 부분의 각 원소의 값을 곱해 모두 더한 값을 출력으로 하는 연산



위 그림은 4×4의 input data에 3×3의 filter를 적용하여 2×2의 output을 구하는 과정을 보여준다. 이러한 방식으로 Filter를 사용하여 연산해주는 layer를 Convolution Layer라고 한다.

Conv2d

```
torch.nn.Conv2d(in_chanels,out_chanels,kernel_size,stride=1,padding=0)
```

input chanel : input 채널의 개수이다. 일반적으로 RGB 3개라고 함.

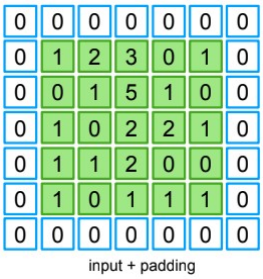
output chanel : 마찬가지로 output 채널의 개수

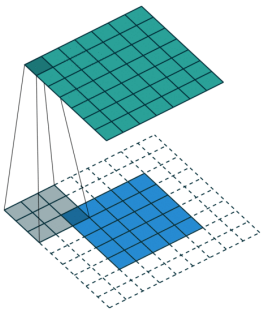
kernal size : 필터 사이즈. 3 을 입력하면 3×3으로 인식.
정사각형이 아니면 숫자 대신 (2,3)과 같이 써놓으면 된다.

stride : 한번에 필터를 몇칸씩 움직일 것인지. stride=1 이면 한칸씩 움직인다는 뜻.
당연하지만 커널 사이즈 보다 크면 안된다.(계산이 안되는 값들이 생김)

padding : 주위에 띠가 더 둘러지는 것.

ex) padding=0





zero padding 은 input의 size를 유지해주면서, edge의 정보를 잃지 않게 하기위하여 사용하는 방법이다. 위 예시와 같이 data의 edge 바깥 부분을 0으로 채워주는 방법을 zero padding이라고 한다. 위와 같이 0으로 두 겹을 쌓아줄 경우 padding=2 옵션이며, 따로 설정해주지 않을 경우에는 padding=0 이 default 옵션으로 사용된다.

잠깐 구글링 해보니 padding 에도 여러 형태가 있는 듯 함(추가 필요)

dilation : Spacing between kernel elements. Default: 1

필터 원소들 간의 간격인가? 써봐야 알 것 같다. (아직 뭔지 모르겠음)

grouds : Number of blocked connections from input channels to output channels.
Default: 1 (blocked connections??)

참고
https://pytorch.org/docs/stable/nn.html?highlight=conv#torch.nn.Conv2d

만약 conv = nn.Conv2d(1,1,3) 와 같이 선언 되었다면,

입출력 채널이 1개인 커널사이즈 3짜리 필터를 쓰는 convolution 연산이라는 뜻.

또 위와 같이 선언되었을 때

out = conv(input) 과 같이 써서 출력을 표현하고 싶다면

input type 은 troch,Tensor
input shape 은 (N x C x H x W) (batch size, chanel, height, width)

와 같이 넣어주면 된다.

Output size

convolution 연산에서 output size를 구하는 공식은 다음과 같다.

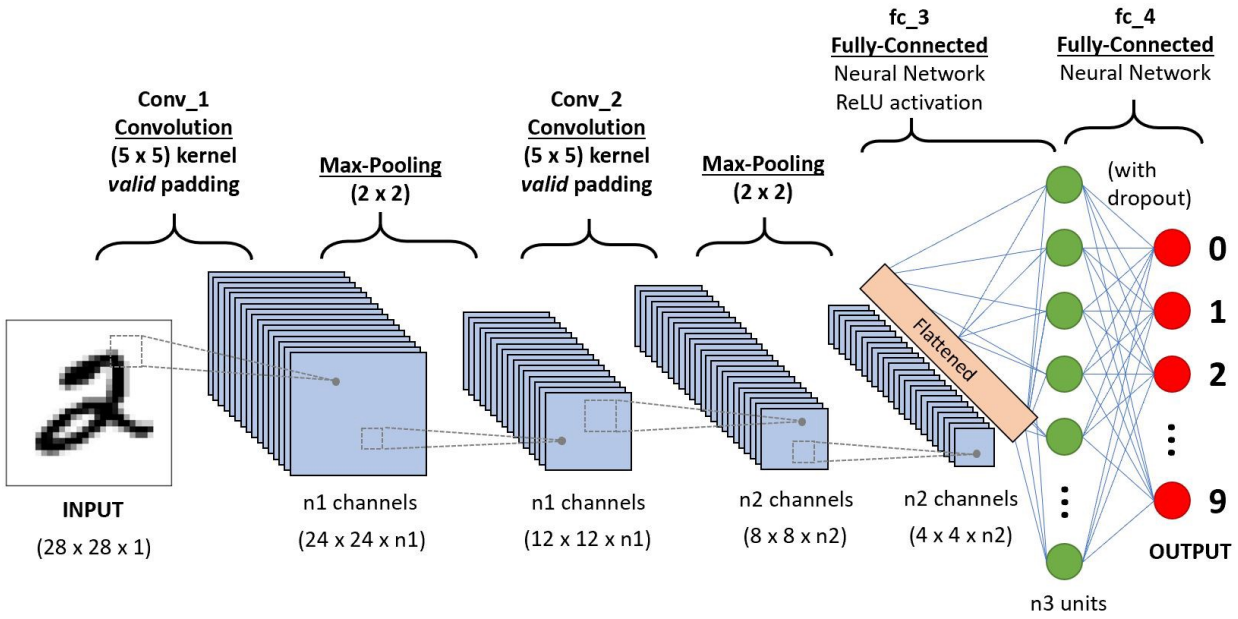
$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

서로 관계가 있는 변수들이므로 잘 유도해 보면 위와같은 식을 얻을 수 있을 것이다.

구글링 해보면 친절하게 유도해준 사람이 있긴 할텐데 굳이 유도를 해봐야 되나..?

하여간 좌변의 값을 알고 있으면 output size를 미리 계산해 볼 수 있다.

neuron 과 convolution

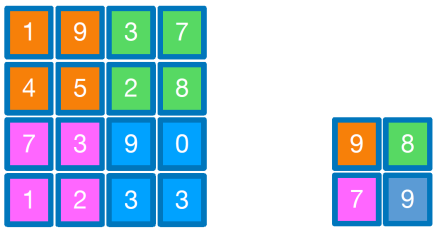


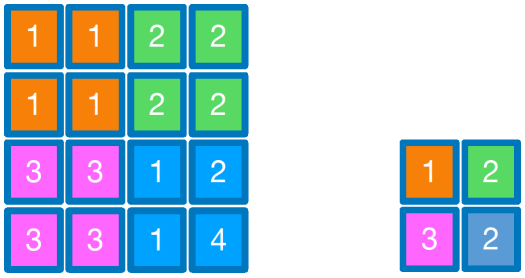
어차피 convolution 연산도 perceptron을 거치기 때문에 bias 가 개입 될 수 있다.

따라서 실제 출력값은 convolution 연산값 + bias 로 출력이 된다.

Pooling

CNN에서 사용되는 또 하나의 layer가 있는데, 바로 Pooling layer이다. Pooling layer는 Down sampling을 위해 사용되는데, 대표적인 예시로 Max pooling이나 Average Pooling이 있다.





Max pooling의 경우, filter 안에 들어오는 elements 중에서 가장 큰 값을 선택하여 output 행렬의 element로 사용하는 방식이다.

반면, Average pooling의 경우, filter 안에 들어오는 elements에 대하여 평균 값을 구하여 output 행렬의 element로 사용하는 방식이다.

torch.nn.MaxPool2d()

MaxPool2d

CLASS `torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)` [SOURCE]

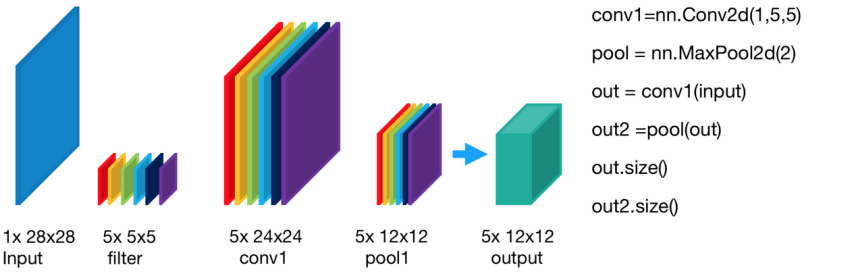
Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points. `dilation` controls the spacing between the kernel points. It is harder to describe, but this link has a nice visualization of what `dilation` does.

CNN implementation

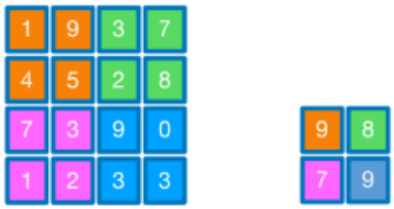



```
input = torch.Tensor(1,1,28,28)
conv1=nn.Conv2d(1,5,5)
pool = nn.MaxPool2d(2)
out = conv1(input)
out2 =pool(out)
out.size()
out2.size()
```

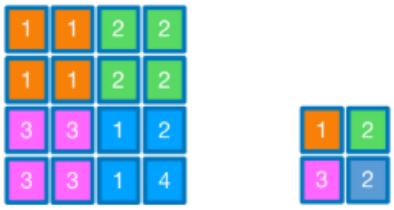
이미지를 여러 구획으로 나누어 특정 연산을 수행하게 하는 방식이다.

max pooling 은 구획 내에서 최댓값을 반환하며

average polling 은 구획의 평균값을 반환한다.



max pooling



Average pooling

max polling의 선언은 다음과 같이 해주면 된다.

```
torch.nn.MaxPool2d(kernel_size,stride=1,padding=0)
```

예를 들어 위 의미지에서의 max pooling은
torch.nn.MaxPool2d(2,stride=2,padding=0)

pooling을 선언 할 때 return_indices=False, ceil_mode=False 와 같은 요소들도 있다.

참고링크
https://pytorch.org/docs/stable/nn.html#pooling-layers

```
input = torch.Tensor(1,1,28,28)
conv1=nn.Conv2d(1,5,5)
pool = nn.MaxPool2d(2)
out = conv1(input)
out2 =pool(out)
out.size()
# torch.Size([1, 5, 24, 24])
out2.size()
# torch.Size([1, 5, 12, 12])
```

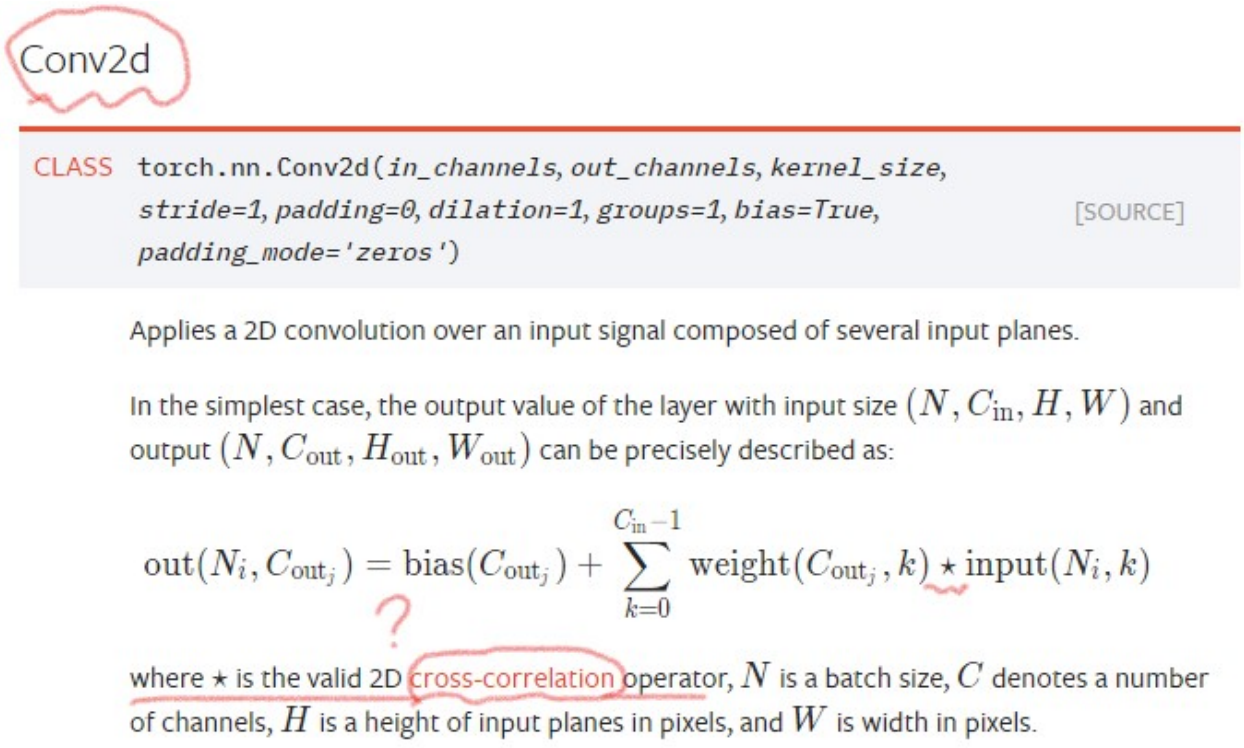
One more Thing!

필터를 뒤집지 않은 Convolution을 Cross-Correalation이라고 한다

cross-correlation

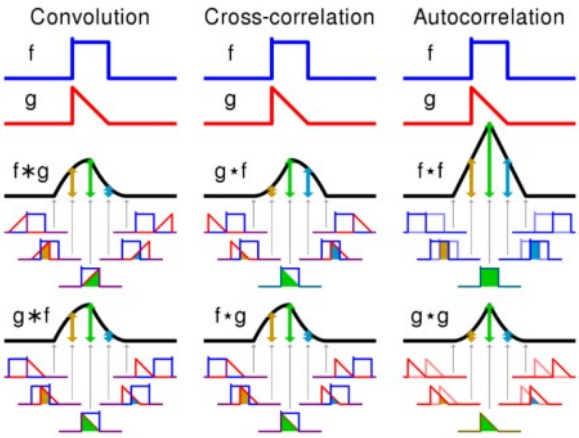
https://pytorch.org/docs/stable/nn.html?highlight=conv#torch.nn.Conv2d

파이토치측의 Conv2d 설명을 보면

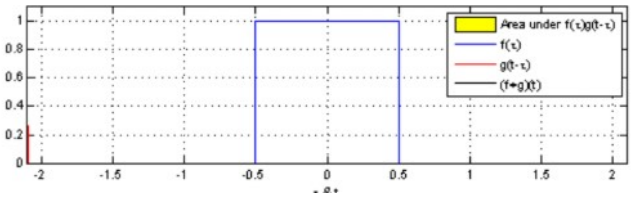


Conv2d 를 설명하면서 수식에 cross-correlation 연산자를 사용했다고 하는데 링크를 클릭하면

잔악무도한 영단어가 난무하는 영어 위키 페이지로 우리를 인도한다.



$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.$$



두 형태의 output을 얻을 수 있어서 사용하는 것 같다.

10-2 MNIST CNN

MNIST에 CNN 적용해보기

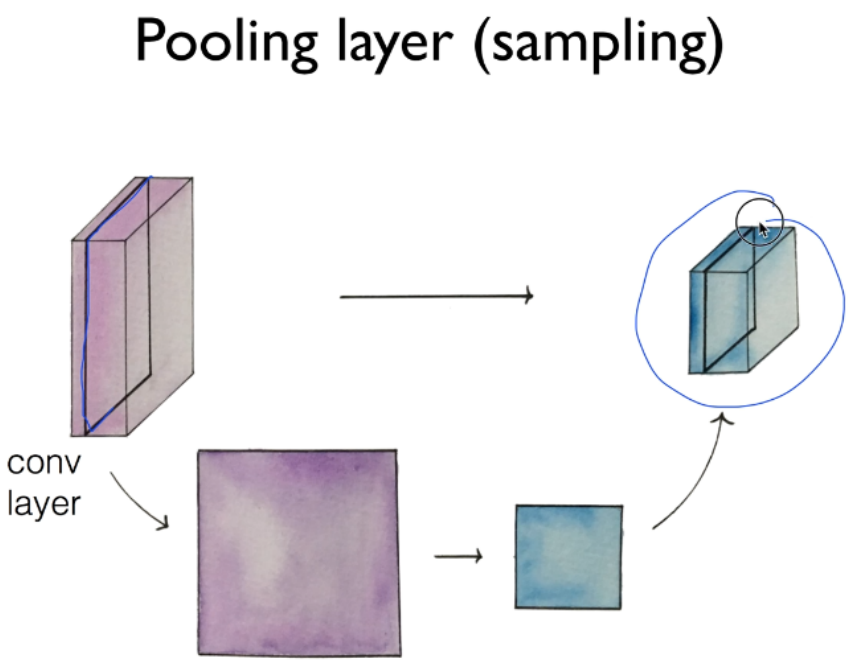
1. 딥러닝 학습 단계
2. 우리가 만들 CNN 구조 확인
3. MNIST에 CNN 적용 코드 작성

결론: 단순히 레이어를 깊게 구성한다고 성능이 향상되는 것은 아니다.

학습 단계

1. 라이브러리 가져오고 (torch, torchvision, matplotlib 같은것들)
2. GPU 사용 설정 하고 random value를 위한 seed 설정!
3. 학습에 사용되는 parameter 설정!(learning_rate, training_epochs, batch_size, etc)
4. 데이터셋을 가져오고 (학습에 쓰기 편하게) loader 만들기
5. 학습 모델 만들기(class CNN(torch.nn.Module))
6. Loss function (Criterion)을 선택하고 최적화 도구 선택(optimizer)
7. 모델 학습 및 loss check(Criterion의 output)
8. 학습된 모델의 성능을 확인한다.

POOLING

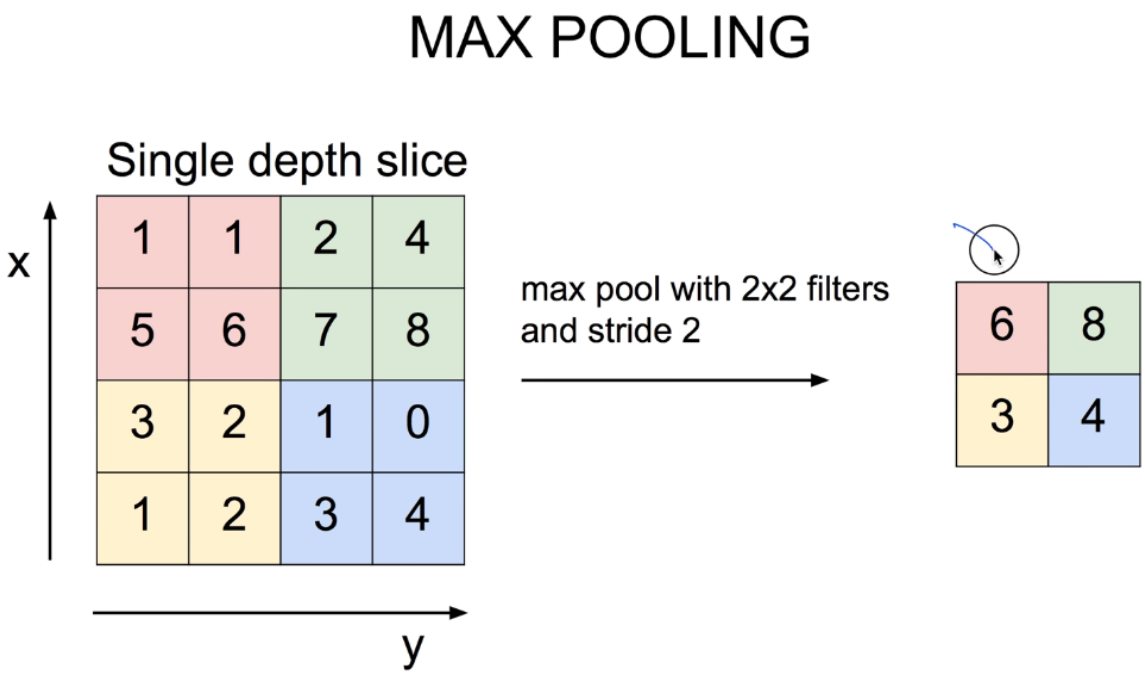


- 하나의 레이어를 sampling(작아짐)한 후, 다시 층을 쌓는다.

풀링 레이어는 컨볼류션 레이어의 출력 데이터를 입력으로 받아서 출력 데이터(Activation Map)의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용됩니다. - 출처

Fully Connected Layer (FC layer)

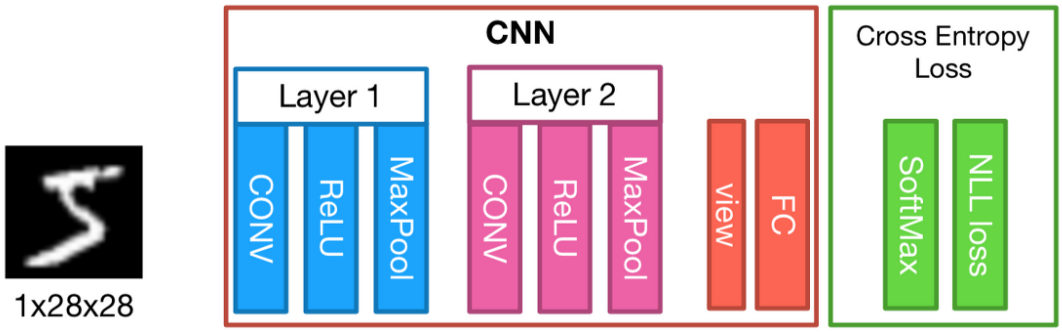
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



학습 과정 시각화

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

직접 실습할 CNN 구조



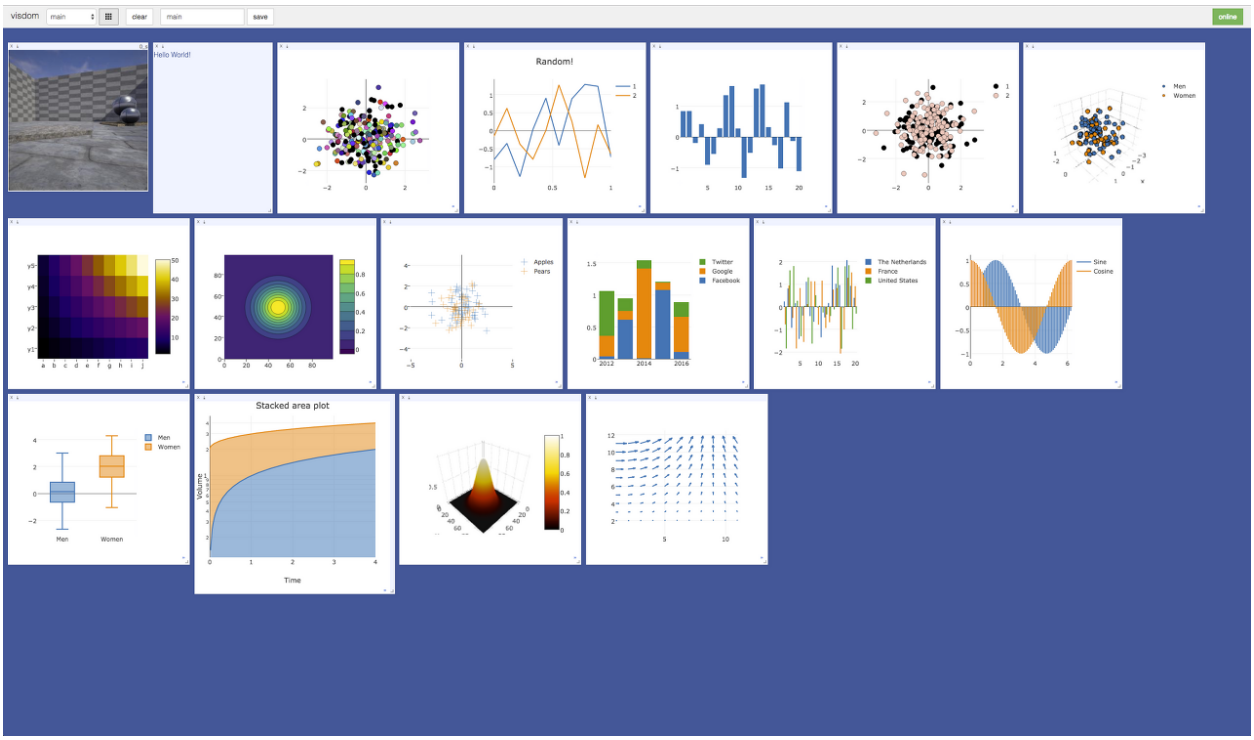
(Layer 1) Convolution layer = (in_c=1, out_c=32,kernel_size =3, stride=1,padding=1)
(Layer 1) MaxPool layer = (kernel_size=2, stride =2)

(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)

Lab-10-3 visdom

Visdom



Visdom은 라이브 데이터를 풍부하게 시각화 해주는 시각화 도구이다. 연구원과 개발자가 원격 서버에서 과학 실험을 지속적으로 할 수 있도록 도와주며, 브라우저에서 실행되며 다른 사람과 쉽게 공유할 수 있다.