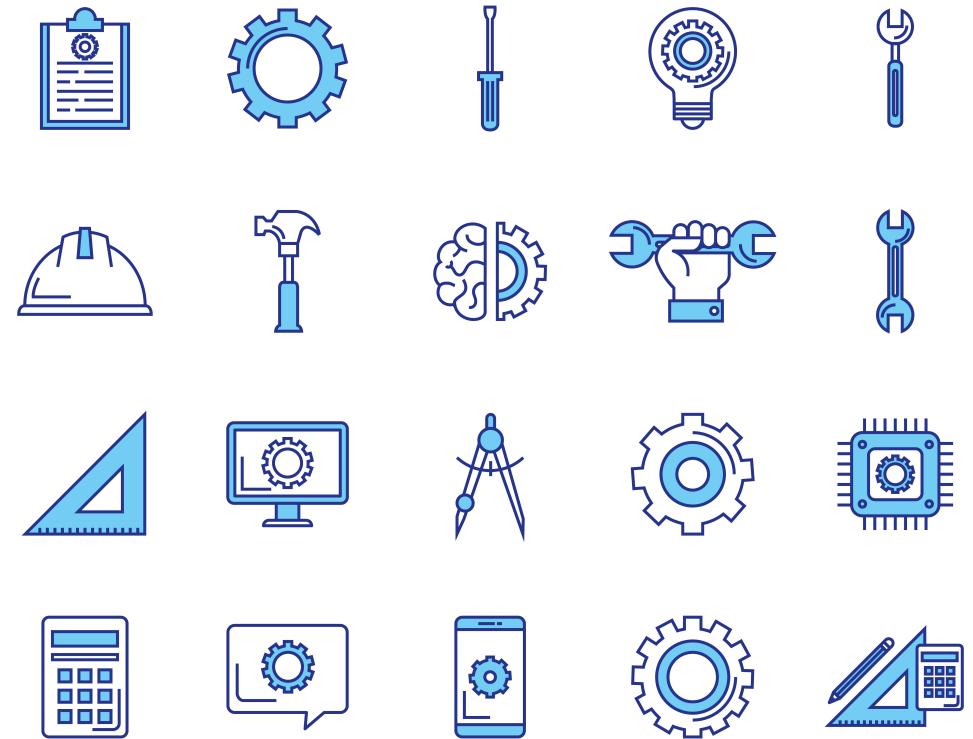


인슈어테크

파이썬 데이터 분석

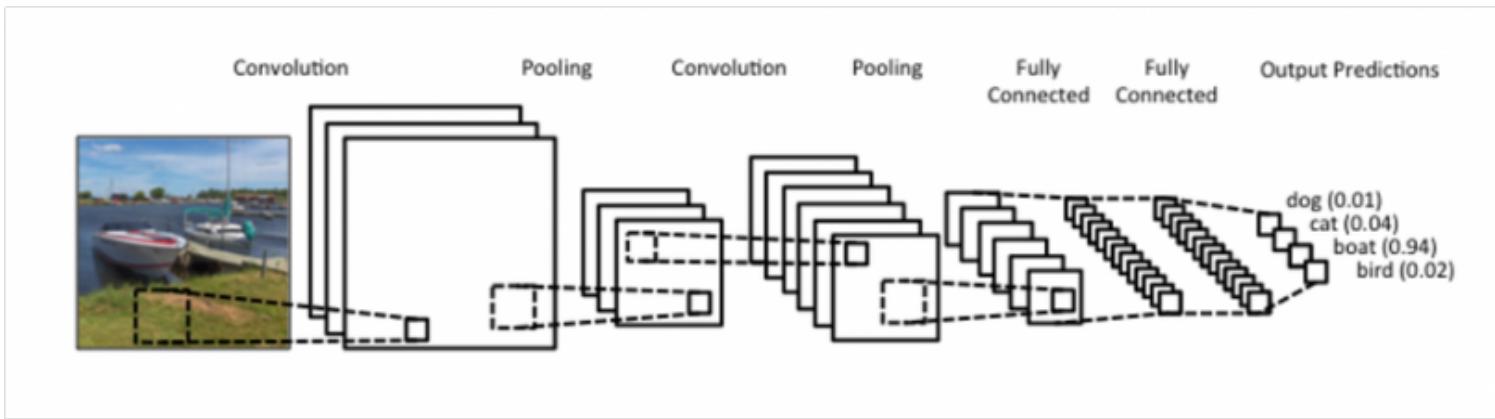


이태일 강사

CNN

- convolutional neural network의 약자
- 딥러닝 모델로 데이터를 학습하여 classification을 진행
- MNIST데이터 셋을 이용하여 학습 진행 예정

CNN

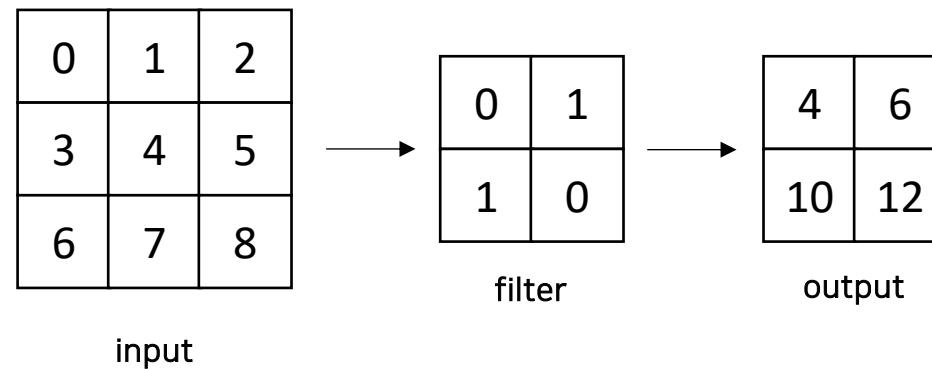


- 데이터를 부분적으로 분할하고 비교해 얼마나 유사한지를 판단

CNN

- convolution

- 이미지 위에서 특정한 값 만큼 필터를 이동시키면서 겹쳐지는 부분의 모든 원소를 곱하고 그 값을 더하는 연산
- convolution의 결과를 피쳐 맵이라고 함



CNN

- convolution

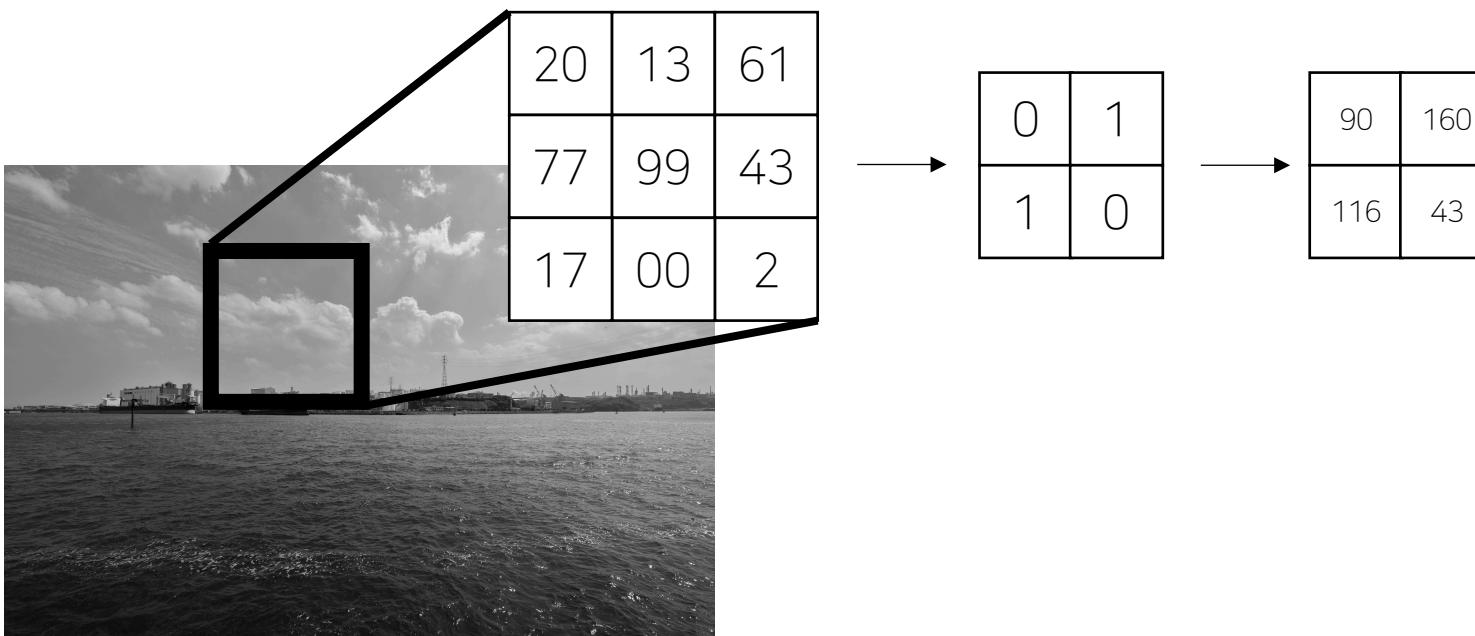
- 이미지 위에서 특정한 값 만큼 필터를 이동시키면서 겹쳐지는 부분의 모든 원소를 곱하고 그 값을 더하는 연산

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

CNN

- convolution

- 이미지 위에서 특정한 값 만큼 필터를 이동시키면서 겹쳐지는 부분의 모든 원소를 곱하고 그 값을 더하는 연산
- convolution의 결과를 피쳐 맵이라고 함



CNN

- convolution
 - convolution 결과를 피쳐맵이라고 부르는 이유
 - convolution 필터는 그 특징이 데이터에 있는지 없는지를 검출해주는 함수



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Filter

= 0



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

CNN

- torch.nn.Conv2d()

```
import torch
import torch.nn as nn

conv = nn.Conv2d(1,1,2,stride=1)

t1 = torch.Tensor([[[[20., 13., 61.],
                    [77., 99., 43.],
                    [17., 0., 2.]]]]))

conv(t1)
```

```
t = torch.Tensor(1,3,128,128)
conv = torch.nn.Conv2d(3,64,kernel_size=3,stride=2,padding=2)
conv(t).shape

torch.Size([1, 64, 65, 65])
```

CNN

- **stride**
 - 필터를 데이터에서 얼마나 이동시킬 것인가
- **padding**
 - 0으로 된 패딩을 몇 개 만들 것인가?

0	0	0	0
0	90	160	0
0	116	43	0
0	0	0	0

padding = 1

0	0	0	0	0	0
0	0	0	0	0	0
0	0	90	160	0	0
0	0	116	43	0	0
0	0	0	0	0	0
0	0	0	0	0	0

padding = 2

CNN

- convolution의 input데이터
 - (배치사이즈-B, 채널-C, 높이-H, 넓이-W)로 구성된 텐서
- convolution output 텐서 size 공식

$$\text{output width} = \frac{\text{input width} - \text{kernel size} + (2 * \text{padding})}{\text{stride}} + 1$$

$$\text{output height} = \frac{\text{input height} - \text{kernel size} + (2 * \text{padding})}{\text{stride}} + 1$$

실습 문제

*input : (1*3*128*128)
output channel : 64
kernel size = 3*3
stride = 2
padding = 2*

*output W : (128-3+4)/2 + 1
output H : (128-3+64/2 + 1
output shape: (1*64*65*65)*

CNN

- pooling : `torch.nn.MaxPool2d()`
 - 필터에 겹쳐지는 부분 중 가장 큰 값을 추출(이미지의 사이즈를 줄이기 위한 방법)
 - stride 기본 값을 2로 설정

$$\text{output width} = \frac{\text{input width} - \text{kernel size} + (2 * \text{padding})}{\text{stride}} + 1$$

$$\text{output height} = \frac{\text{input height} - \text{kernel size} + (2 * \text{padding})}{\text{stride}} + 1$$

MAX-Pooling

20	13	61
77	99	43
17	00	2

99	99
99	99

CNN

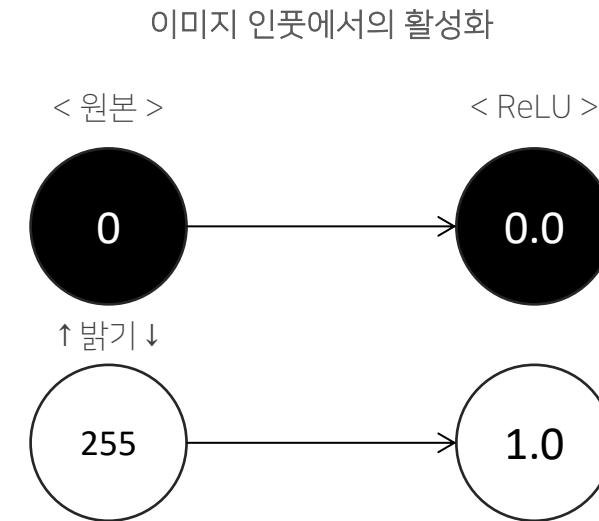
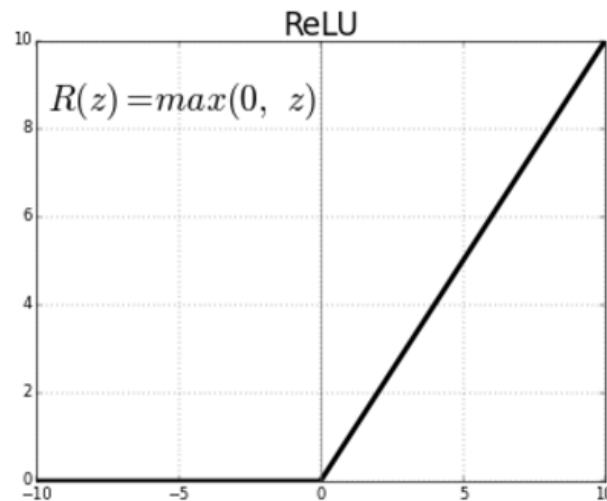
- convolution-pooling 사용예제

```
import torch
import torch.nn as nn

conv = nn.Conv2d(1,1,2,stride=1)
t1 = torch.Tensor(1,1,3,3)
pool = nn.MaxPool2d(1)
out = pool(conv(t1))
```

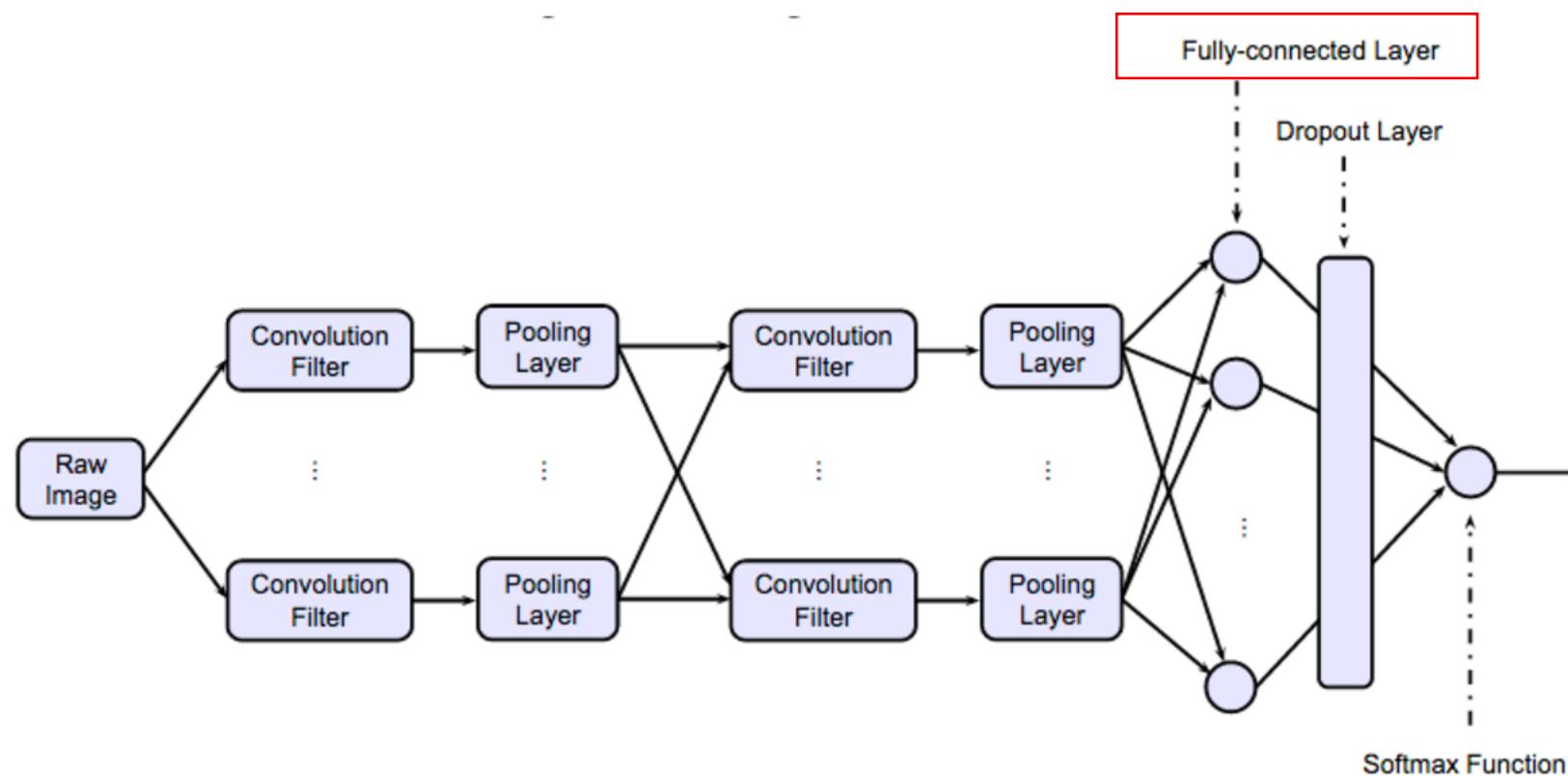
CNN

- ReLU 함수
 - 정량적인 값을 True, False의 비선형적 데이터로 치환하는 과정 (활성함수)
* 참에 가까우면 0.5~1사이, 거짓에 가까우면 0~0.5 사이의 값으로 리턴



CNN

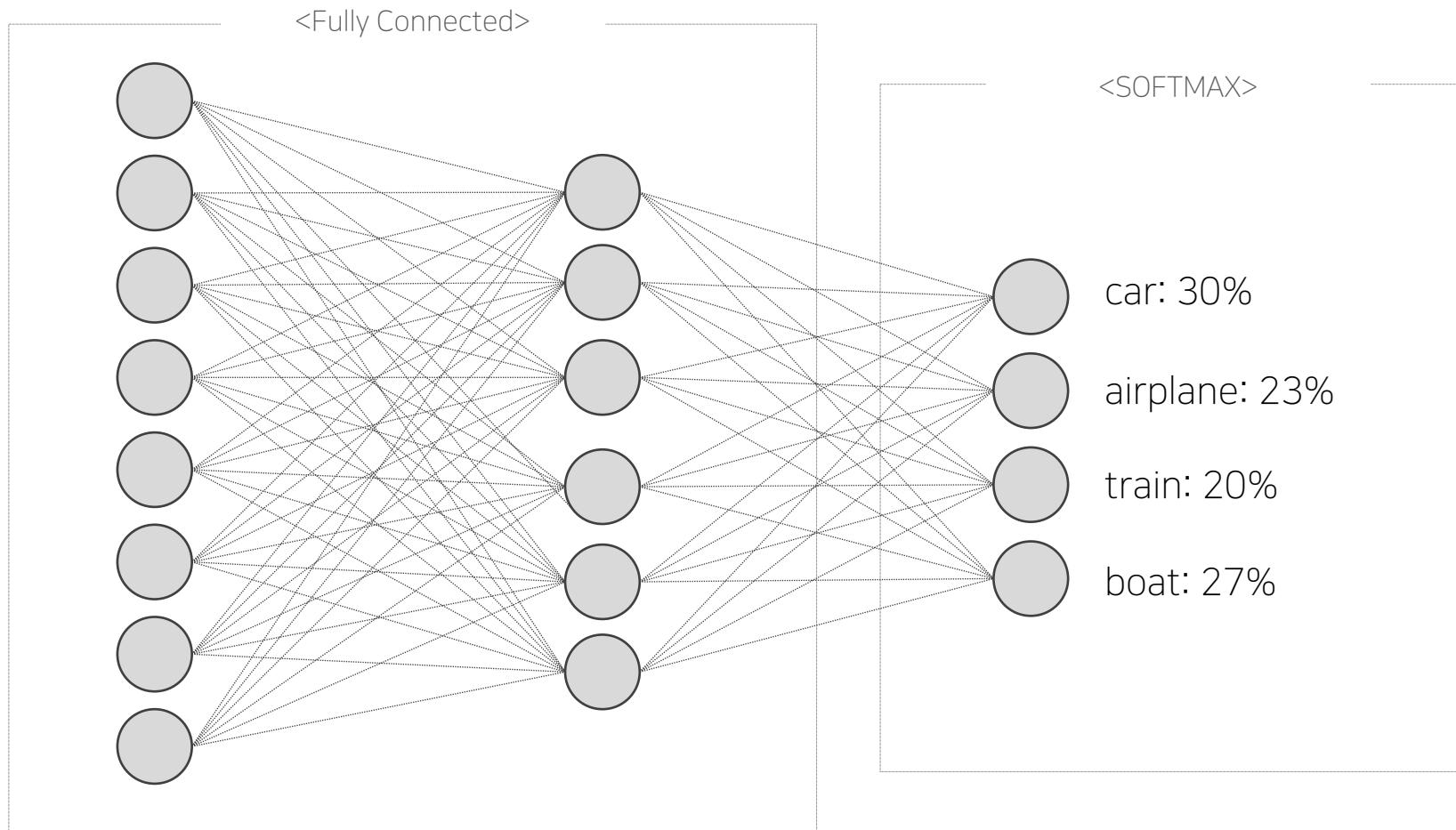
- Fully-Connected 연산



CNN

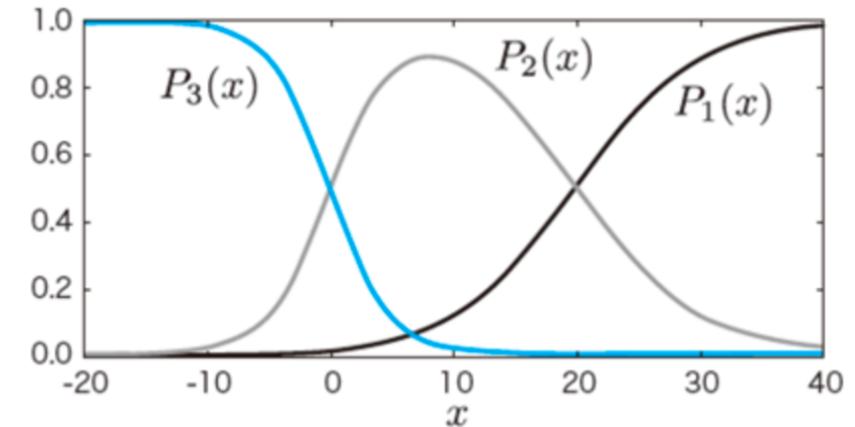
- Fully-Connected 연산

- 학습 데이터를 1열로 펼쳐서 뉴런간 선형 결합(channel x width x height) -> 1 dimension vector



CNN

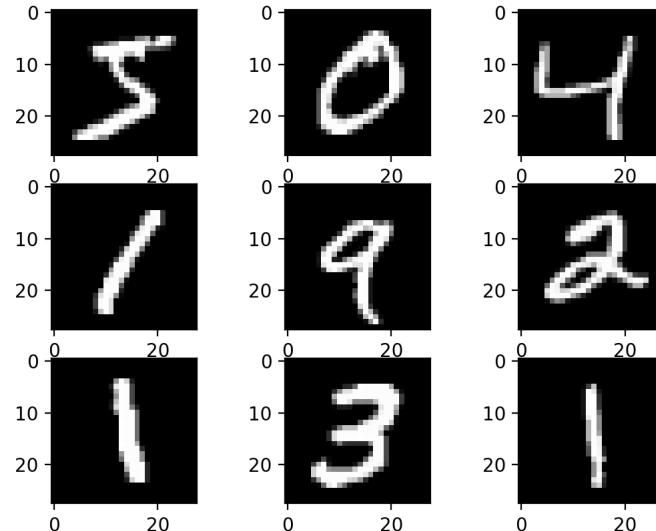
- softmax
 - ReLU와 같은 활성함수의 일종
 - 타 활성함수가 이산 분류인 반면, softmax는 여러개의 분류를 가짐
 - $p_3(x)$ 는 x (feature)가 특정 값일 때 x 가 p_3 일 확률을 의미
- CNN 프로세스
 - o 특징 추출(convolution layer)
 - 물체 사진을 입력 값으로 받는다.
 - 사진은 데이터화 되어 Convolution layer를 통과한다.
 - o 분류(fully connected layer)
 - fully connected layer를 통과하면서 각 클래스일 확률을 계산한다.
 - 사진을 가장 높은 확률을 보인 클래스로 분류한다.



< softmax 그래프 >

CNN

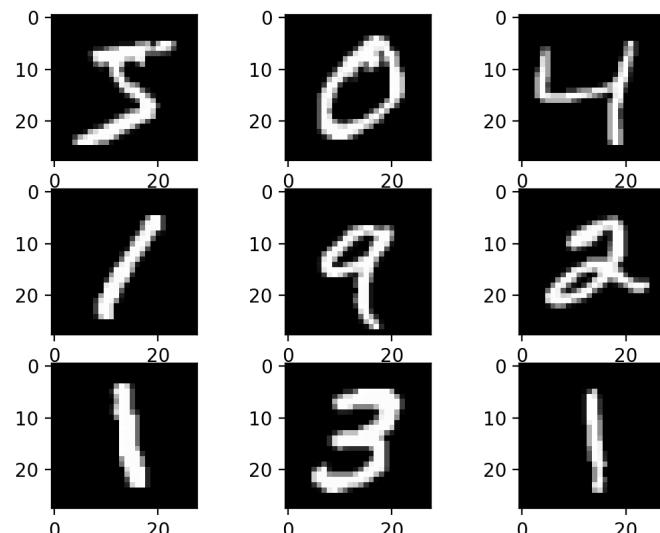
- MNIST 데이터셋



- 다양한 사람의 0~9 까지의 숫자 손글씨를 기록한 데이터셋
- 각각 1*28*28 사이즈(흑백 1채널)로 구성되어 있음

CNN

- 우리가 만드려는 모델



1*28*28



layer1 conv: in=1, out=32, filter size=3, padding=1
layer1 Pool: kernel_size=2, stride=2

layer2 conv: in=32, out=64, filter size=3, padding=1
layer2 Pool: kernel_size=2, stride=2

CNN

```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn.init

# gpu사용
device = "cuda" if torch.cuda.is_available() else "cpu"
torch.manual_seed(1000)
if device == "cuda":
    torch.cuda.manual_seed(1000)
```

라이브러리 임포트 및 gpu 사용

CNN

```
batch_size=100
learning_rate=0.001
epochs=15
```

받아올 데이터의 배치사이즈, 러닝 레이트, 학습 횟수 설정

CNN

```
mnist_train = datasets.MNIST(root="/content/drive/My Drive/insuretech/MNIST_CNN/",
                             train=True,
                             download=True,
                             transform=transforms.ToTensor())

mnist_test = datasets.MNIST(root="/content/drive/My Drive/insuretech/MNIST_CNN/",
                            train=False,
                            download=True,
                            transform=transforms.ToTensor())
```

학습, 테스트 데이터 다운로드 및 텐서로 변경

CNN

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

데이터 불러오기

CNN

```
# CNN Model (2-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)
        out = self.fc(out)
        return out
```

레이어 만들기

7*7*64= layer2를 거친 input의 크기

레이어 초기화

데이터를 1열로 펼치기

CNN

```
# CNN모델이 gpu를 사용하도록 만들기
model = CNN().to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
```

CrossEntropyLoss는 다른 종류의 cost 함수

Adam optimizer를 사용해서 최적의 cost를 도출

CNN

```
total_batch = len(data_loader) ← 데이터의 개수

for epoch in range(15): ← 학습횟수 만큼 반복

    # loss 값을 담을 변수 0으로 초기화
    avg_cost = 0

    # img는 input, label은 정답지
    for img,label in data_loader:
        img = img.to(device)
        label = label.to(device) ← 데이터 1개씩 순환

        # 선형회귀에서 했던 코드
        optimizer.zero_grad()
        hypothesis = model(img)
        cost = criterion(hypothesis, label) ← 비용함수 계산을 통한 학습

        cost.backward()
        optimizer.step()

        avg_cost += cost/total_batch ← 계산된 cost값과 전체 데이터개수를 나눠서
                                    평균을 구함

    print("Epoch:{} cost={}".format(epoch+1, avg_cost))
```

CNN

```
# 정확도 테스트
with torch.no_grad():

    # 테스트 데이터 불러와서 1렬 뉴런 형태로 불러오기
    img_test = mnist_test.test_data.view(len(mnist_test), 1, 28, 28).float().to(device)
    label_test = mnist_test.test_labels.to(device)

    # 학습된 CNN모델에 삽입
    prediction = model(img_test)
    # 테스트 라벨과 prediction의 라벨을 비교해서 값이 True인것만 추출
    correct_prediction = torch.argmax(prediction, 1) == label_test

    # 모든 데이터의 평균을 내서 정확도를 추출
    accuracy = correct_prediction.float().mean()

    # 정확도 프린트
    print('Accuracy:', accuracy.item())
```

Q&A