파이토치 첫걸음

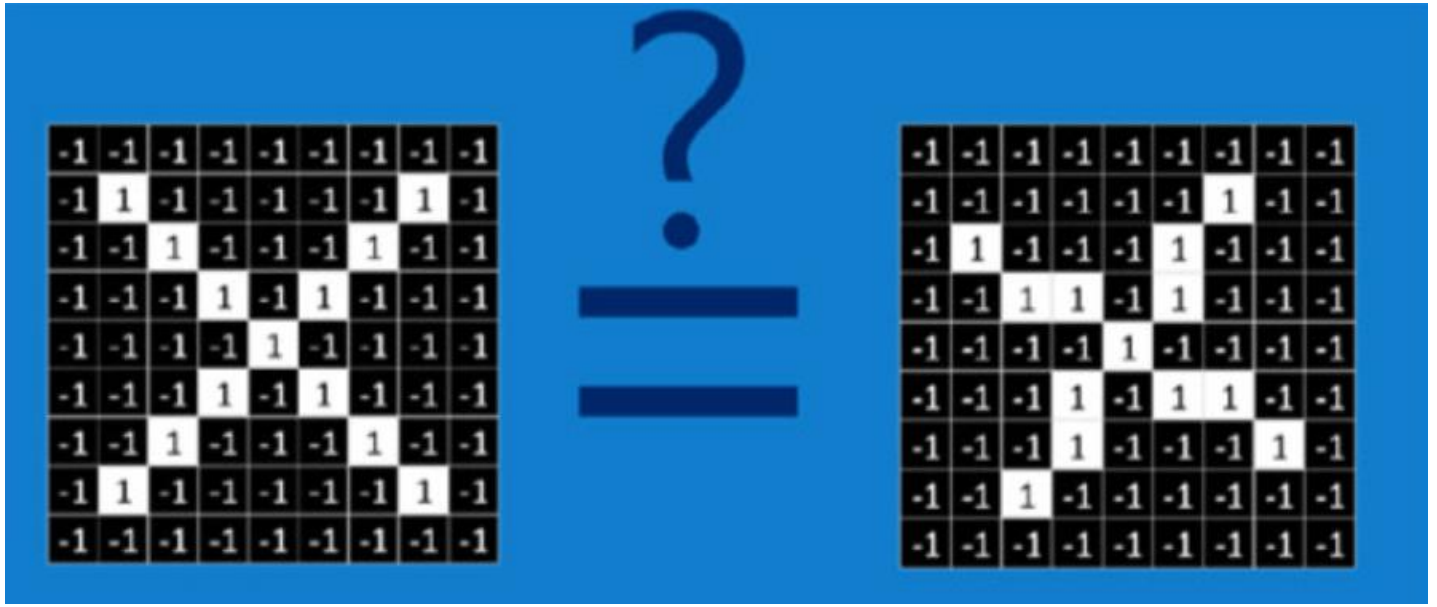
chapter 5. 합성곱 신경망

21.10.07

곽민지

합성곱 연산

합성곱이란

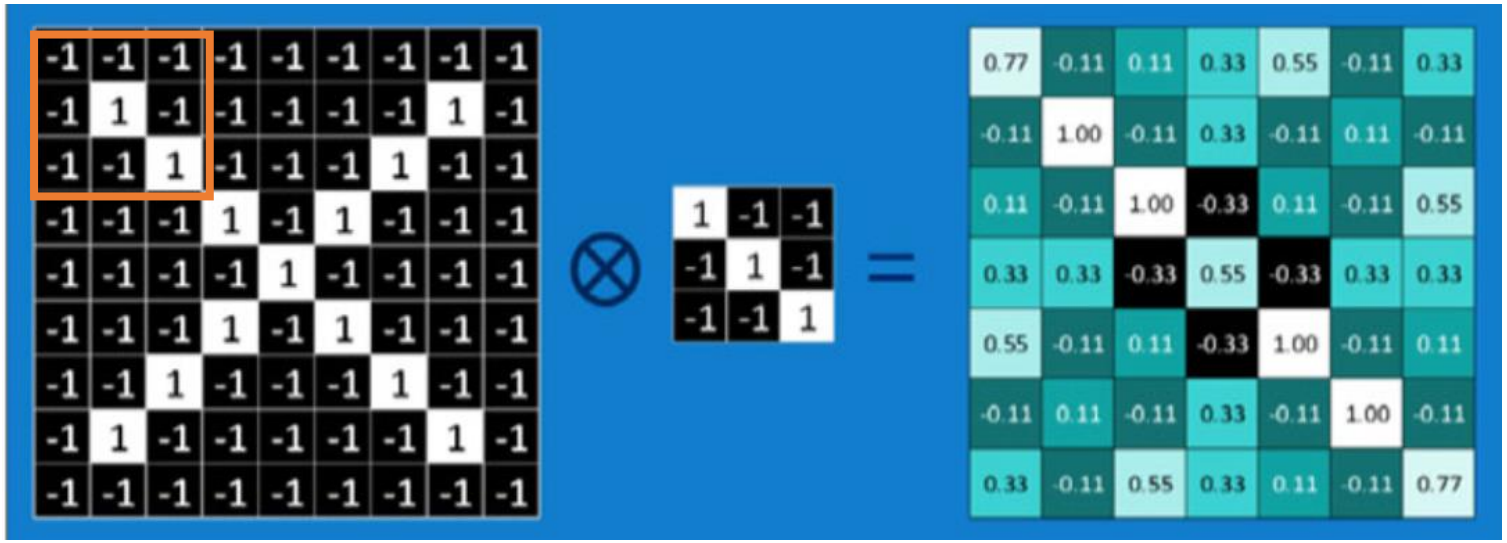


☑ 합성곱(CNN; Convolutional Neural Network)

- 포유류와 인간의 시각 체계를 관찰
  (국소적인 영역을 보고 단순한 패턴에 자극을 받는 단순 세포와 넓은 영역을 보고 복잡한 패턴에 자극을 받는 복잡세포의 계층)
- "하나의 함수가 다른 함수와 얼마나 일치하는가"에 의미가 있다.

합성곱 연산

합성곱 연산 과정

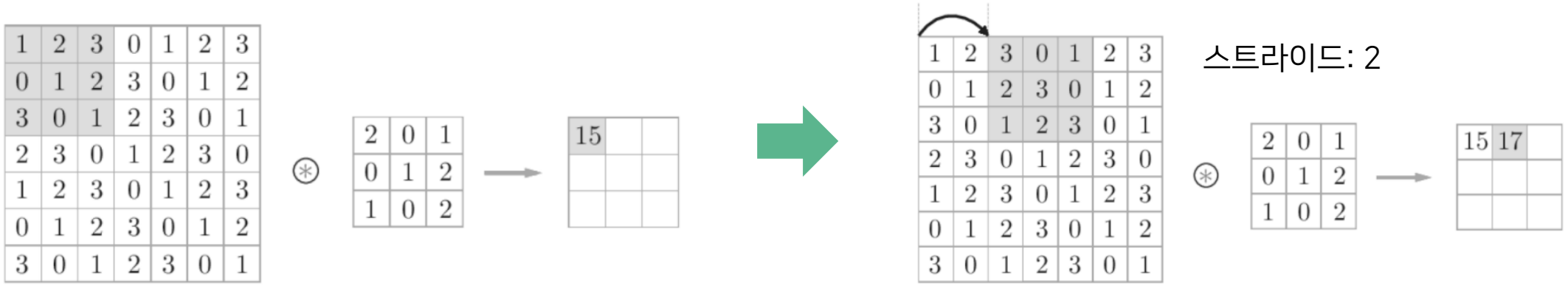


☑ 합성곱 연산 과정

- 하나의 필터(커널)에 대해, 이미지를 쭉 지나가면서 이미지의 부분 부분이 필터와 얼마나 일치하는지 계산
- 1.00 → 필터와 이미지가 완벽히 일치하는 부분
- 예시)
  1/9 x {(-1) x 1 + (-1) x (-1) + (-1) x (-1) + (-1) x (-1) + 1 x 1 + (-1) x (-1) + (-1) x (-1) + (-1) x (-1) + 1 x 1} = 0.77

합성곱 연산

합성곱 연산 과정



https://sean-parkk.github.io/study/DLscratch_CNN/

- 스트라이드(Stride): 필터의 이동 단위

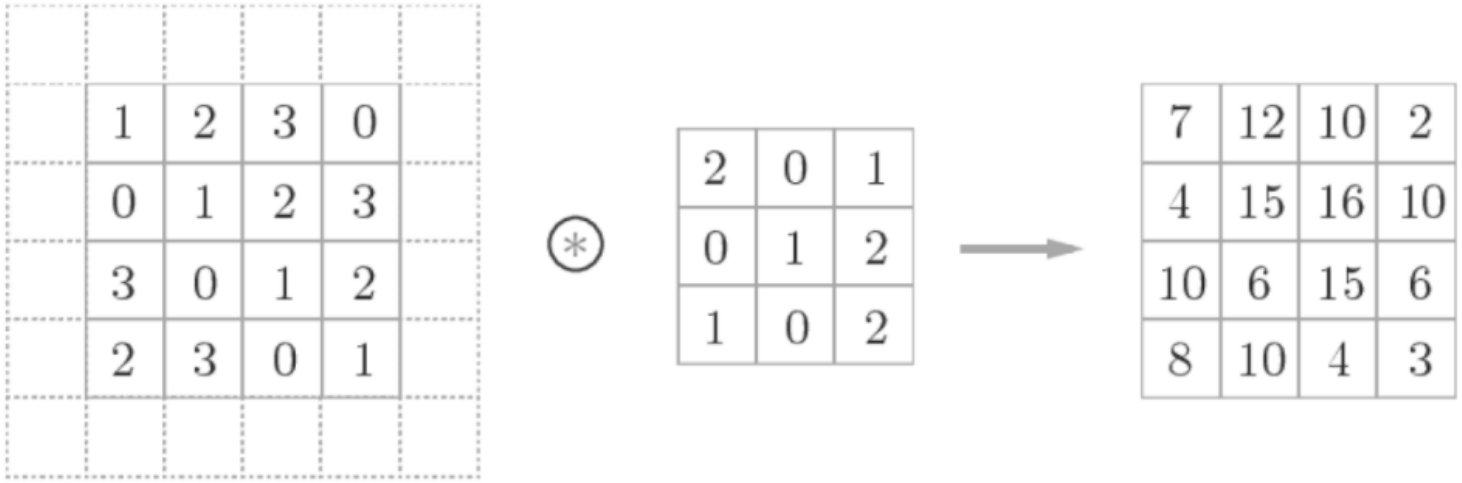
  → 필터의 크기와 스트라이드는 자유롭게 지정 가능

Q. 만약 스트라이드 1로 지정 후, 필터를 여러 번 반복하여 적용하면 어떻게 될 것인가?

합성곱 연산

합성곱 연산 과정

zero padding



https://sean-parkk.github.io/study/DLscratch_CNN/

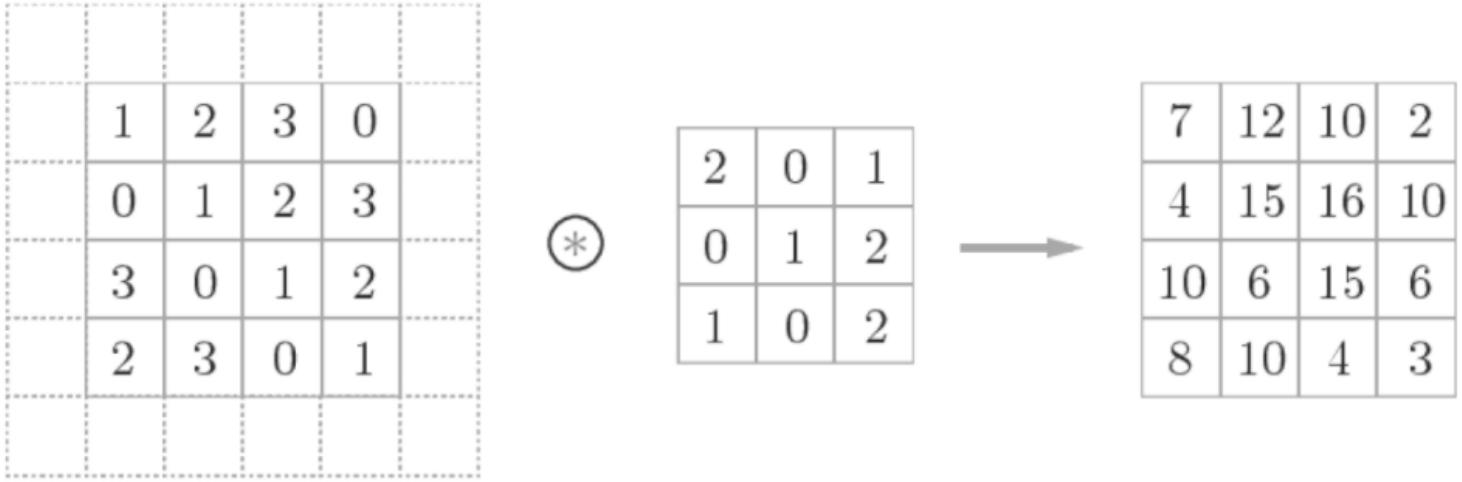
- 패딩(Padding): 일정한 크기의 층으로 이미지를 감싸는 것
- 특성 지도(feature map): 필터 하나당 입력 이미지 전체에 대한 필터의 일치 정도

  ➡ 만약 이미지에 필터를 3개 사용하면, feature map은 3개가 생성

합성곱 연산

합성곱 연산 과정

zero padding



https://sean-parkk.github.io/study/DLscratch_CNN/

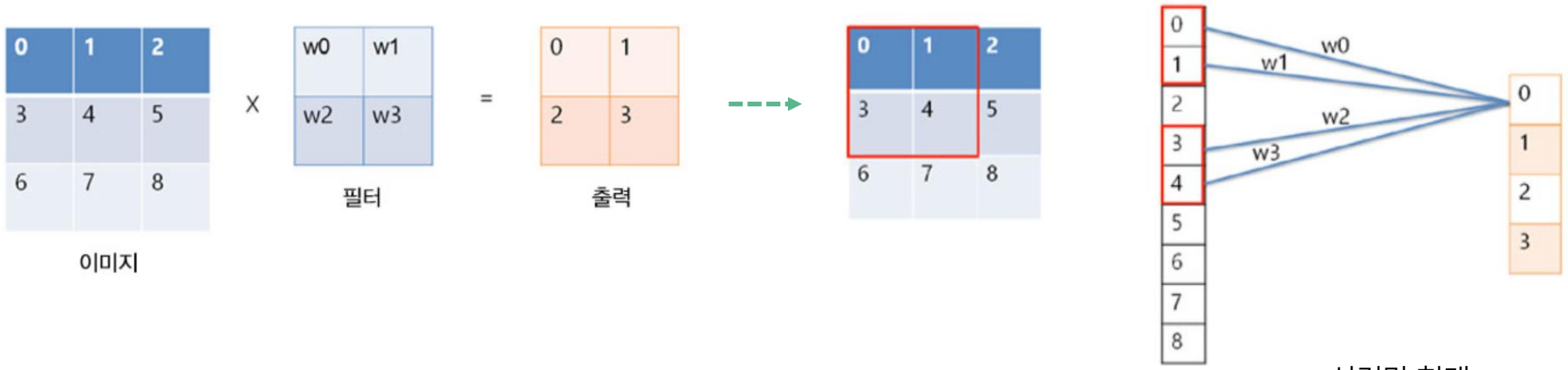
☑ feature map의 크기

$$\left\lfloor \frac{I - K + 2P}{S} + 1 \right\rfloor$$

(I: 이미지 크기, K: 필터의 크기, S: 스트라이드 크기, P: 패딩의 크기)

합성곱 연산

합성곱 연산 과정



→ 신경망 형태

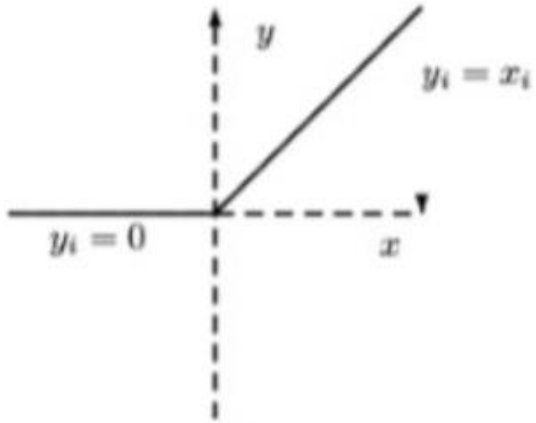
- 합성곱 연산도 인공 신경망의 일종

<차이점>

- 하나의 결과값이 생성될 때, 입력값 전체가 들어가지 않고 필터가 지나가는 부분만 연산에 포함
- 하나의 이미지에 같은 필터를 연달아 적용하기 때문에 가중치가 공유되어 학습 대상이 되는 변수가 적음

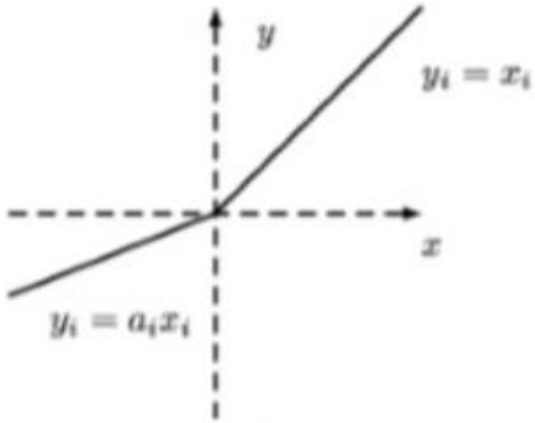
합성곱 연산

합성곱 연산 과정



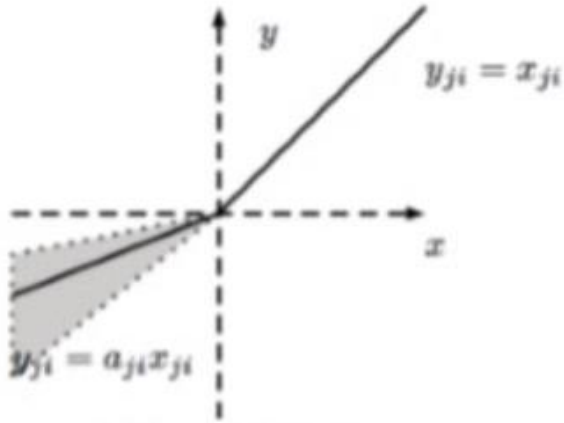
렐루

$f(x)=\max(0,x)$



리키 렐루

$f(x)=\max(ax,x)$



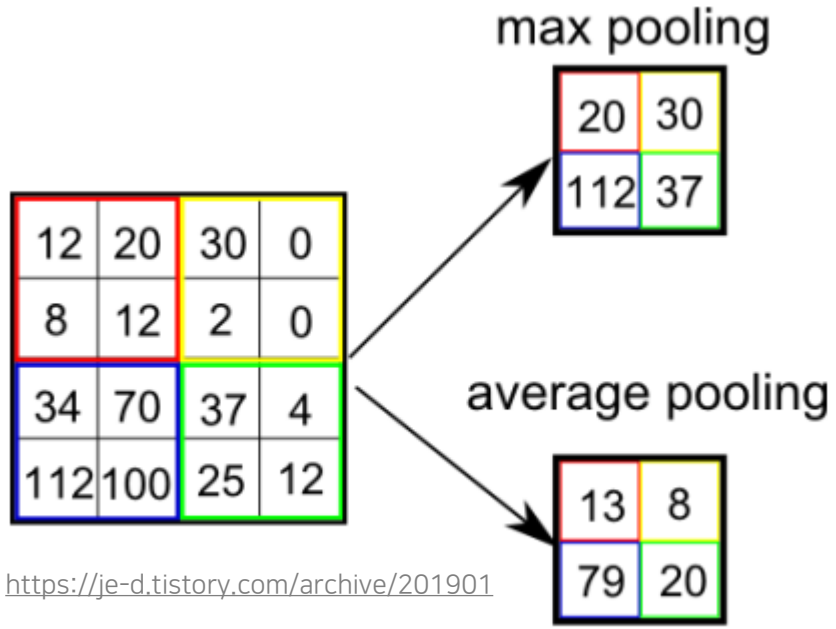
랜덤 리키 렐루

$f(x)=\max(ax,x)$

- 즉, 입력과 가중치의 조합으로 이루어진 연산이기 때문에 비선형성을 추가하기 위해 활성화 함수 필요
- 보통 렐루를 사용한다.

합성곱 연산

합성곱 연산 과정



https://je-d.tistory.com/archive/201901

- 풀링(Pooling): downsampling / subsampling의 일종.

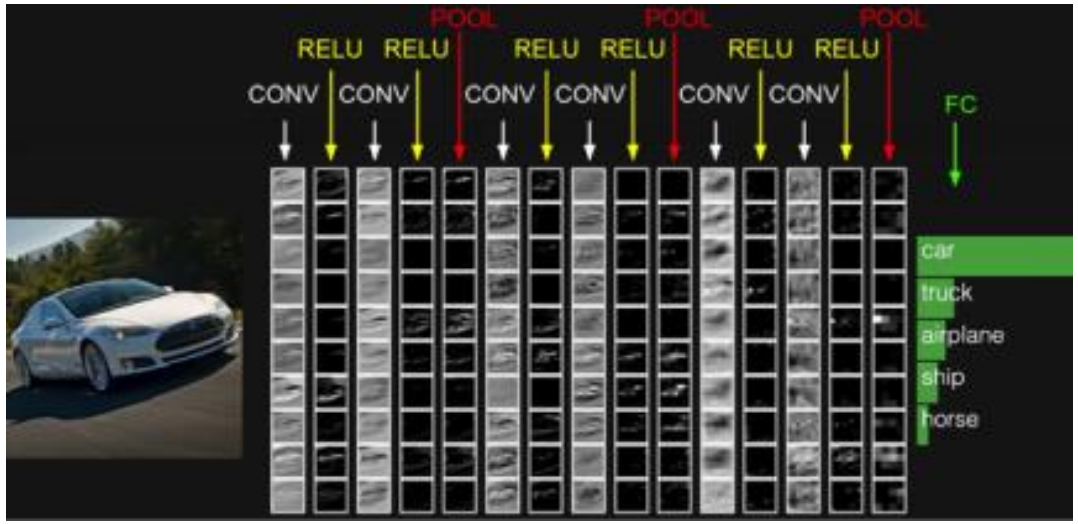
  → 매우 높은 화질이 필요하지 않을 때, 넓게 봐야 파악할 수 있는 특성이 존재할 때 사용

  ① max pooling: 일정 크기의 구간 내에서 가장 큰 값 만을 전달하고 다른 정보는 버리는 방법

  ② average pooling: 일정 크기의 구간 내의 값들의 평균을 전달하는 방법

합성곱 연산

합성곱 연산 과정



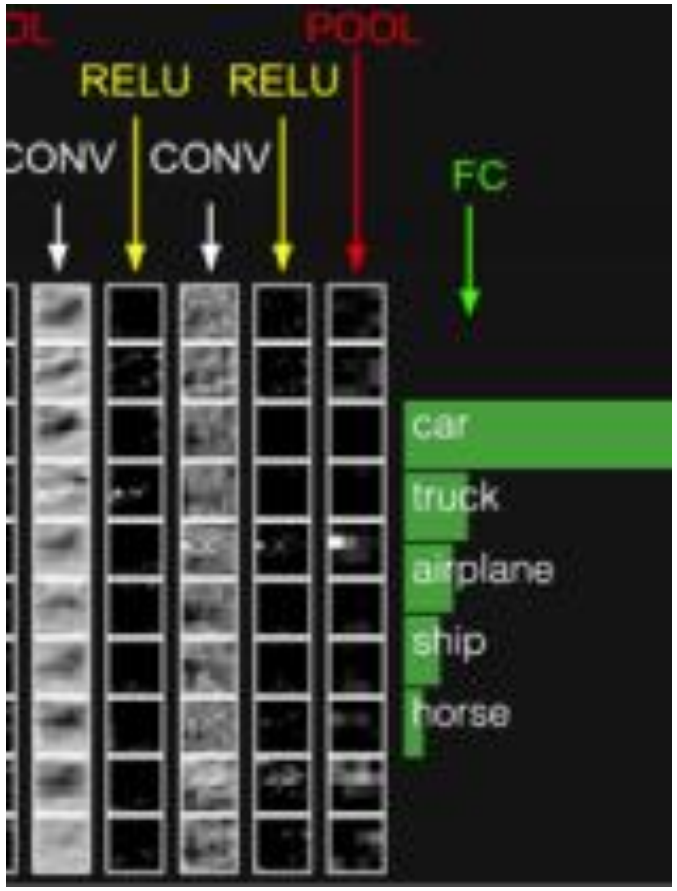
http://cs231n/github.io/convolutional-networks

☑ summary

1. 입력값에 대해 몇 번의 합성곱 연산을 활성화 함수와 함께 적용
2. pooling으로 전체 크기를 줄여주는 과정 반복
3. 어느 정도 특성을 다 뽑은 이후에는, 특성들을 입력으로 받는 인공 신경망을 붙여서 분류 or 회귀 문제 해결

합성곱 신경망

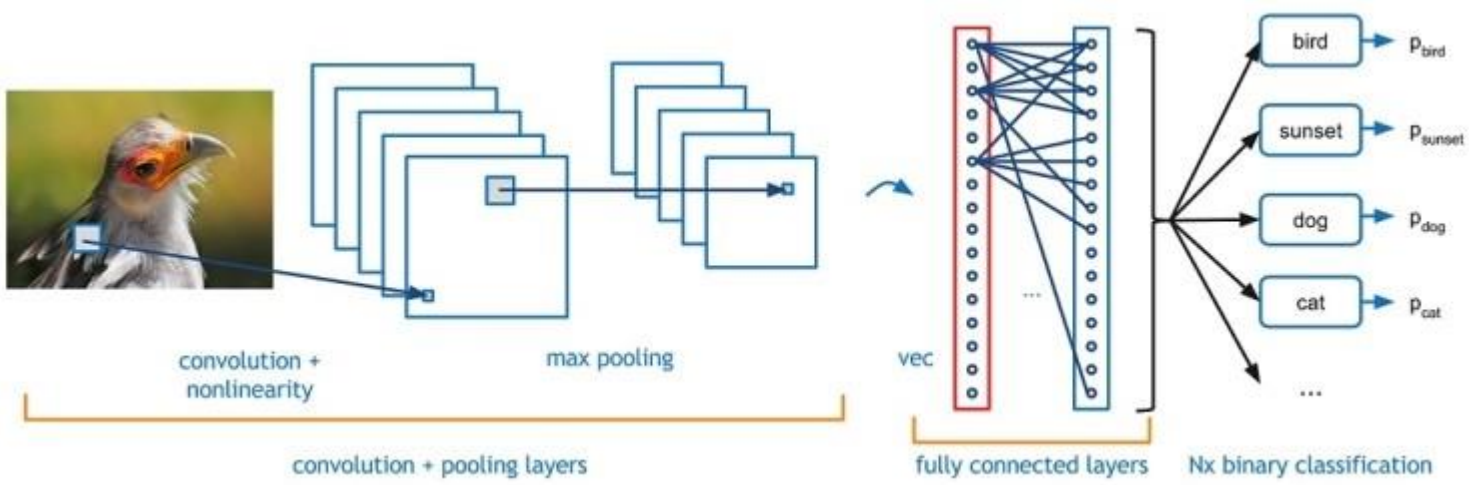
분류 문제



특성들을 충분히 뽑은 이후에는 fully connected layer 적용

→ 가로 x 세로 x 채널이었던 텐서를 한 줄로 쭉 핀다.

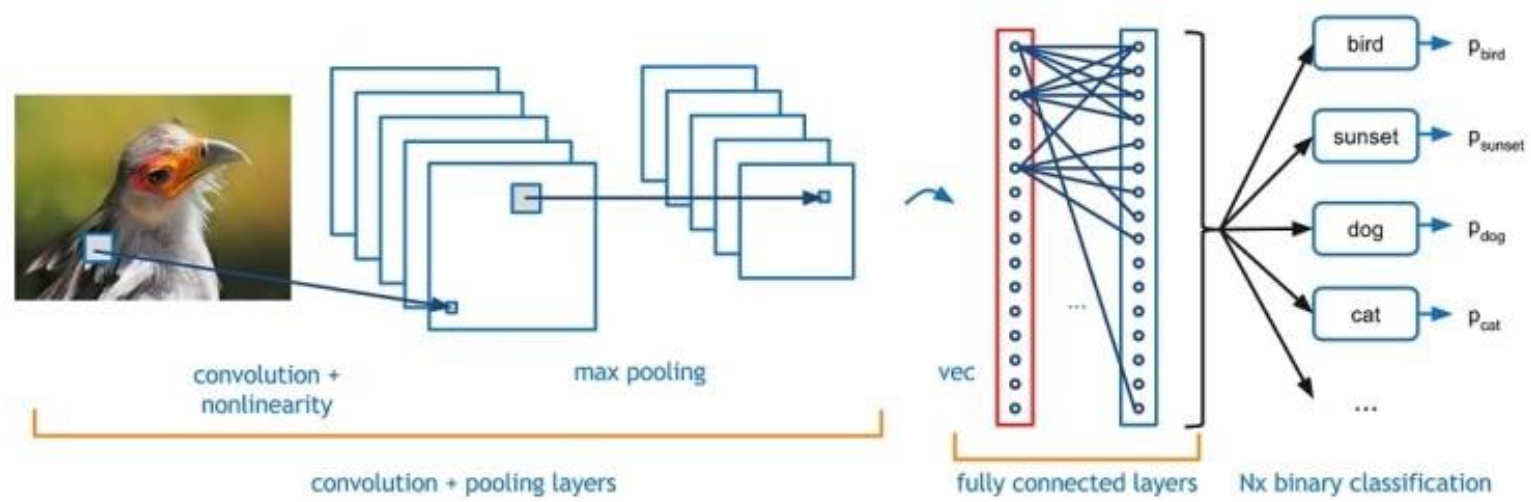
→ 가로 x 세로 x 채널 길이를 가지는 하나의 벡터를 생성



https://m.blog.naver.com/msnayana/220776380373

합성곱 신경망

분류 문제



https://m.blog.naver.com/msnayana/220776380373

인공신경망의 결과는 특정 수치를 결과 값으로 출력

오류 계산 및 역전파를 통한 모델 학습 방법

- 정답을 구분하려는 클래스의 개수에 맞게 원-핫 인코딩 진행 (ex. 고양이: [0, 1] / 강아지: [1, 0])
- 신경망의 결과값을 확률로 바꾸기 위해 소프트맥스(softmax) 함수 사용

합성곱 신경망

softmax

$$softmax(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_i)}$$

- 고양이: [0, 1] / 강아지: [1, 0]
- 신경망의 결과값: [0.37, 1.58]
- softmax: [0.2296, 0,7704]

Q. 손실 측정 방법?

“교차 엔트로피 손실 함수”

합성곱 신경망

교차 엔트로피 함수

☑ 엔트로피(정보량의 기댓값)

$$H(p) = -\sum_{x} p(x) \log p(x)$$

$p(x)$가 작은 값을 가지면 $-logp(x)$값은 커진다.
일어날 확률이 작을수록 가지고 있는 정보가 크고,
일어날 확률이 클수록 가지고 있는 정보가 작은 것

☑ 교차 엔트로피

$$H(p,q) = -\sum_{x} p(x) \log q(x)$$

$$= \mathrm{H(p)} + \sum_{x} p(x) \log \frac{p(x)}{q(x)}$$

목표로 하는 최적의 확률분포 p와 이를 근사하려는
확률분포 q가 얼마나 다른지 측정하는 방법

합성곱 신경망

교차 엔트로피 함수

고양이: [0, 1] / 강아지: [1, 0]

softmax	교차 엔트로피 손실	L1 손실
[0.1, 0.9]	0.1053	0.1
[0.7, 0.3]	2.3025	0.7

교차 엔트로피 값은 예측이 잘못될수록 L1 손실보다 더 크게 증가
그만큼 penalty가 크고 손실 값이 크기 때문에, 학습 면에서도 장점이 존재한다.

합성곱 신경망

모델 구현

import

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.init as init
# torchvision: 유명한 영상처리용 데이터셋, 모델, 이미지 변환기가 들어있는 패키지
import torchvision.datasets as dset #데이터를 읽어오는 역할
import torchvision.transforms as transforms # 불러온 이미지를 필요에 따라 변환
from torch.utils.data import DataLoader # 데이터 전달 방법 정의

batch_size = 256
learning_rate = 0.0002
num_epoch = 10
```

합성곱 신경망

모델 구현

data load

```
# torchvision.datasets.___ => 해당 자리에 원하는 데이터 명시
# "./"의 경로 지정시 데이터가 없는 상태에서 다운로드하는 상황이라면 데이터 저장 위치 지정
mnist_train = dset.MNIST("./", train=True,
                         transform=transforms.ToTensor(), # transform: 데이터에 대한 변형
                         target_transform=None,
                         download=True) # target_transform: 라벨에 대한 변형
mnist_test = dset.MNIST("./", train=False,
                        transform=transforms.ToTensor(),
                        target_transform=None,
                        download=True)  # 데이터 다운 여부

train_loader = torch.utils.data.DataLoader(mnist_train,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=2, # 데이터 묶을 때 사용하는 프로세스 개수
                                           drop_last=True) # 묶고 남은 데이터를 버릴지에 대한 여부
test_loader = torch.utils.data.DataLoader(mnist_test,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          num_workers=2,
                                          drop_last=True)
```

합성곱 신경망

모델 구현

model 클래스 생성

```
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.layer = nn.Sequential(
        nn.Conv2d(1, 16, 5), # conv2d(in_channels, out_channels, kernel_size, ...)
        nn.ReLU(),
        nn.Conv2d(16, 32, 5),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),
        nn.Conv2d(32, 64, 5),
        nn.ReLU(),
        nn.MaxPool2d(2, 2)
    )
    self.fc_layer = nn.Sequential(
        nn.Linear(64*3*3, 100), # 64채널 (3 x 3 이미지)
        nn.ReLU(),
        nn.Linear(100, 10)
    )

  def forward(self, x):
    out = self.layer(x)
    out = out.view(batch_size, -1) # numpy의 reshape와 비슷
    out = self.fc_layer(out)
    return out
```

합성곱 신경망

모델 구현

모델 초기화 / 손실함수 지정 / 학습 진행

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
loss_func = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
loss_arr = []
for i in range(num_epoch):
  for j, [image,label] in enumerate(train_loader):
    x = image.to(device)
    y_ = label.to(device)

    optimizer.zero_grad()
    output = model.forward(x)
    loss = loss_func(output, y_)
    loss.backward()
    optimizer.step()

    if j % 1000 == 0:
      print(loss)
      loss_arr.append(loss.cpu().detach().numpy())
```

합성곱 신경망

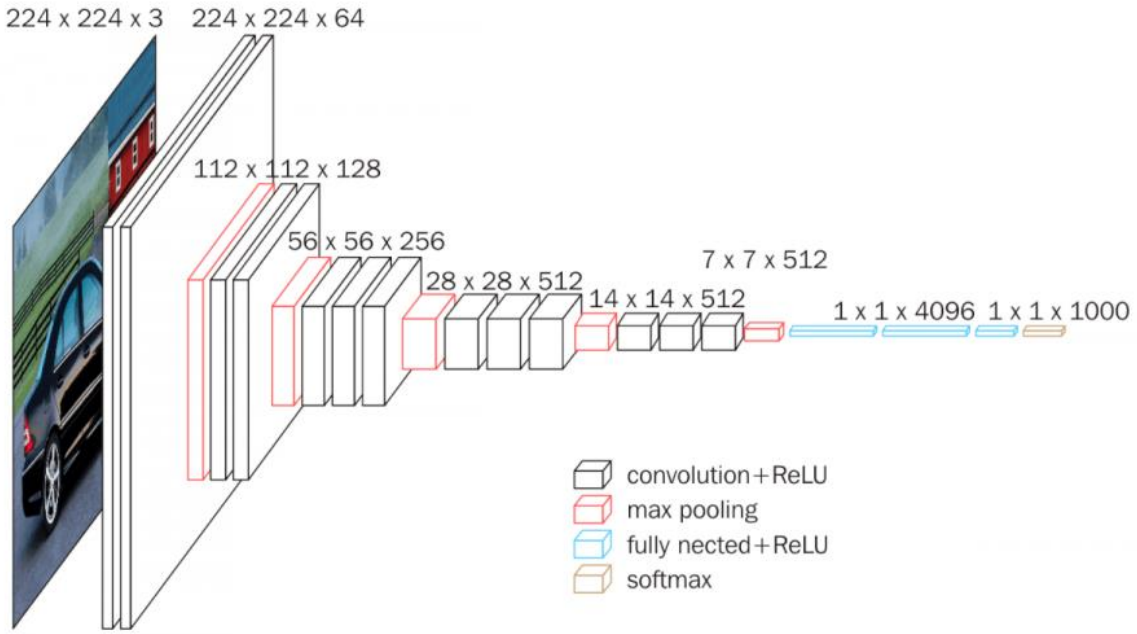
모델 구현

테스트 데이터 검증

```
correct = 0
total = 0

with torch.no_grad():
  for image, label in test_loader:
    x = image.to(device)
    y_ = label.to(device)

    output = model.forward(x)
    _, output_index = torch.max(output, 1)
    # torch.max => return (values, indices)

    total += label.size(0)
    correct += (output_index == y_).sum().float()

  print("Accuracy of Test Data: {}".format(100*correct/total))
```

```
Accuracy of Test Data: 98.65785217285156
```

모델 | VGGNet

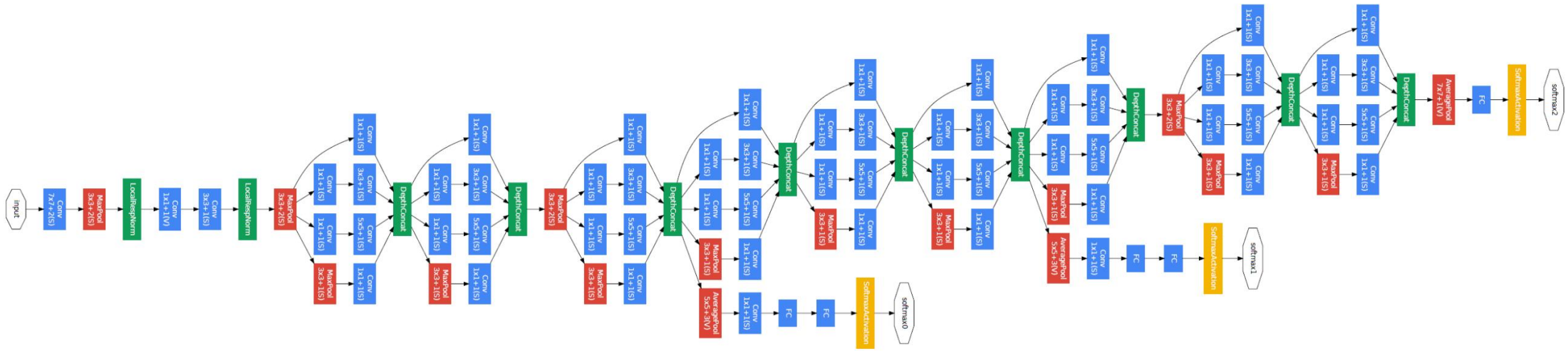


☑ summary

1. 신경망의 깊이가 모델의 성능에 미치는 영향을 조사하기 위해 시작한 연구
2. 3 x 3 convolution, max pooling, fully connected layer 연산만을 사용
3. 11개의 레이어를 가진 모델부터 19개의 레이어를 가진 모델까지 다양한 모델에 대한 실험을 진행

모델

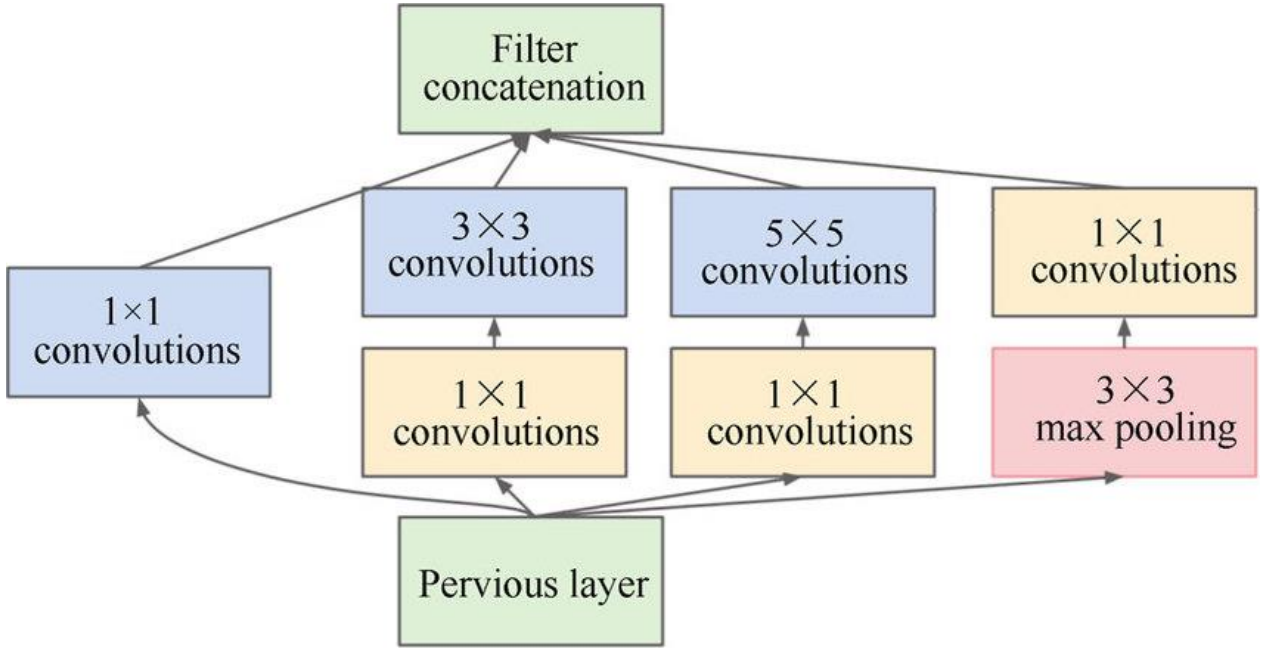
GoogLeNet



☑ summary

1. 인셉션 모듈을 가지고 있고 보다 복잡한 구조를 가지고 있다.
2. 마지막 부분에만 보조 분류기가 있는 보통의 모델과 다르게, 중간 중간에 보조 분류기를 사용
   이는 모델이 깊어지면서 마지막 단의 분류 네트워크에서 발생한 손실이 모델의 입력 부분가지 전달이 안 되는 현상을 극복하기 위함 (학습 보조)

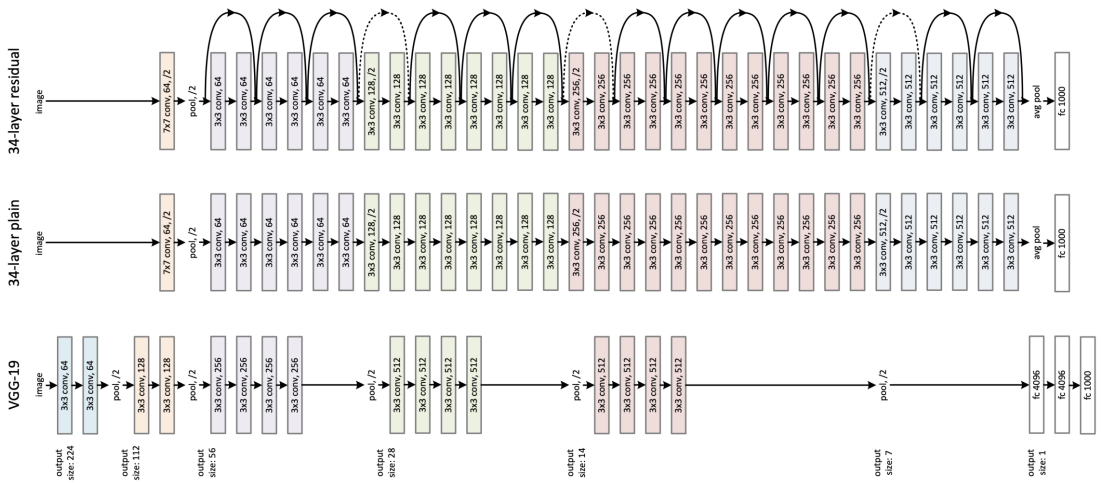
모델 | GoogLeNet



☑ Inception module

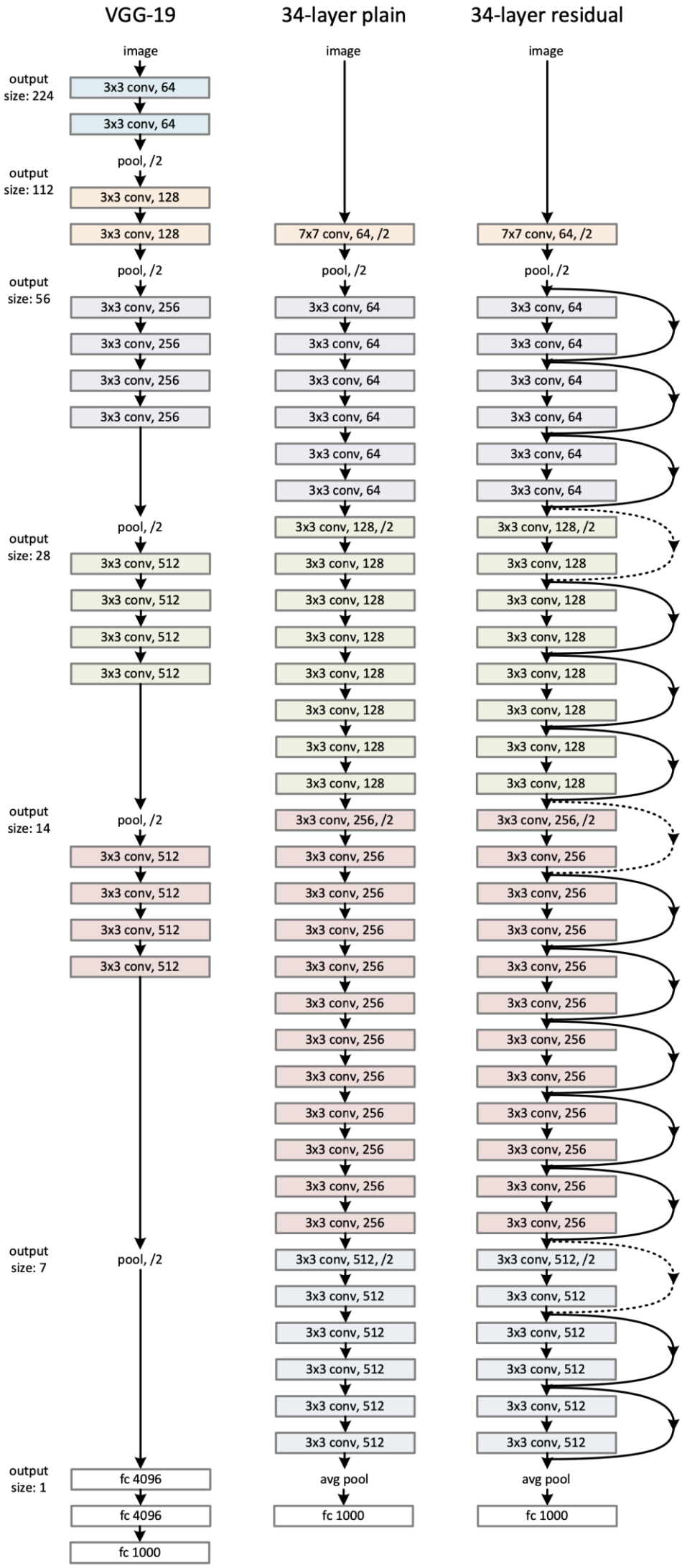
- 이전 단계의 feature map에 다양한 필터 크기로 합성곱 연산을 적용
- 각각 다른 연산 범위를 가지는데, 이는 특징을 찾는 관점 및 범위를 달리하기 위함
- 1 x 1 합성곱은 채널 수 조절 / 연산량 감소(파라미터 줄어듬) / 비선형성 추가의 기능이 있다.

모델 | ResNet

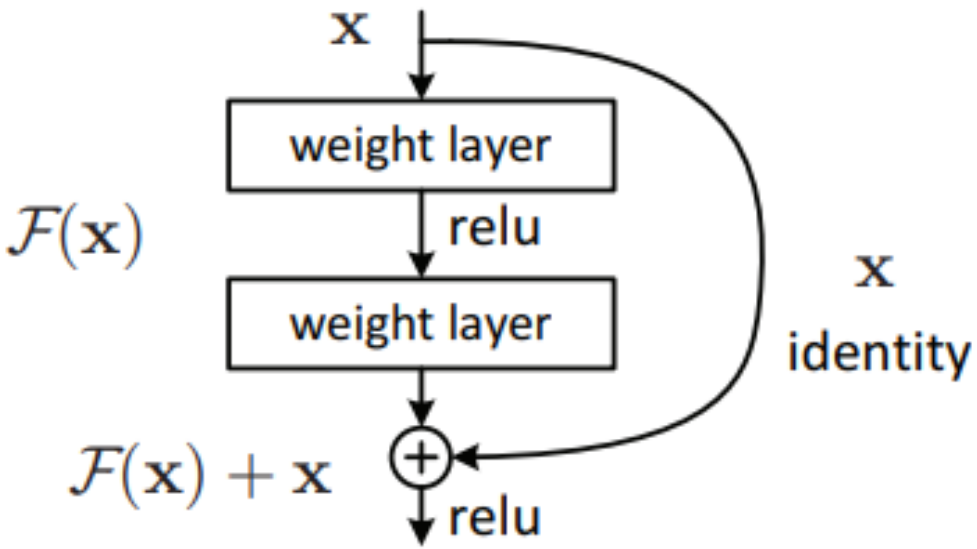


☑ summary

1. 네트워크를 얼마나 깊이 쌓을 수 있을까에 대한 의문에서 시작
2. 일정 수준 이상의 깊이가 되면 오히려 얕은 모델보다 깊은 모델의 성능이 떨어짐을 발견
3. 이를 해결하기 위해 잔차 학습 제안
   → 특정 위치에서 입력이 들어왔을 때 합성곱 연산을 통과한 결과와 입력으로 들어온 결과 두 가지를 더해서 다음 레이어에 전달


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

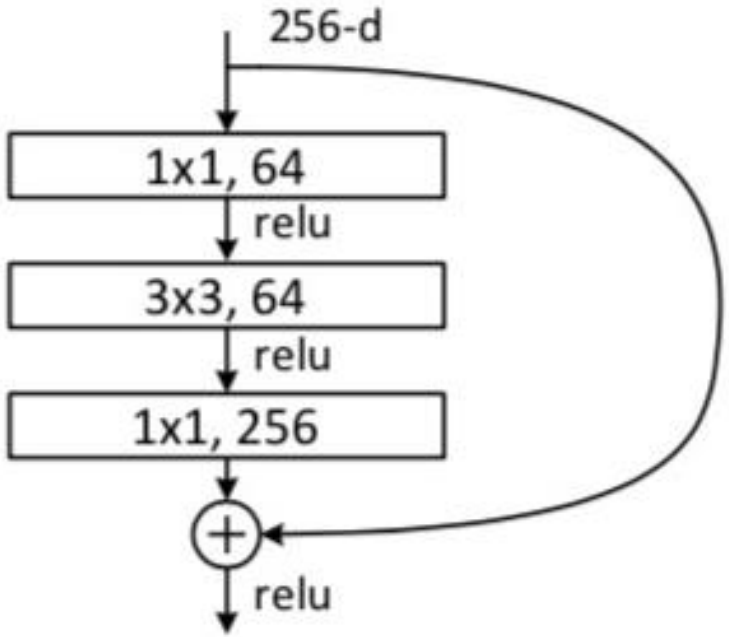
모델 | ResNet



☑ **잔차 학습 블록**

- 이전 단계에서 뽑았던 특성들을 변형시키지 않고 그대로 더해서 전달
- 입력 단에 가까운 곳에서 뽑았던 단순한 특성과 뒤에서 뽑은 복잡한 특성 모두를 사용한다는 장점 존재

모델 | ResNet



☑ bottleneck block

- 깊이가 깊어지고, 모델의 크기가 커지면서 사용하게 된 방법

1. 1 x 1 합성곱으로 채널 방향 압축
2. 압축된 상태에서 3 x 3 합성곱으로 추가 특성을 뽑아내고 다시 1 x 1 합성곱을 사용해 채널의 수를 늘림
3. 결론적으로 변수 수를 줄이면서도 원하는 개수의 특성을 뽑을 수 있음