가천대 회화·조소과 AI 특강

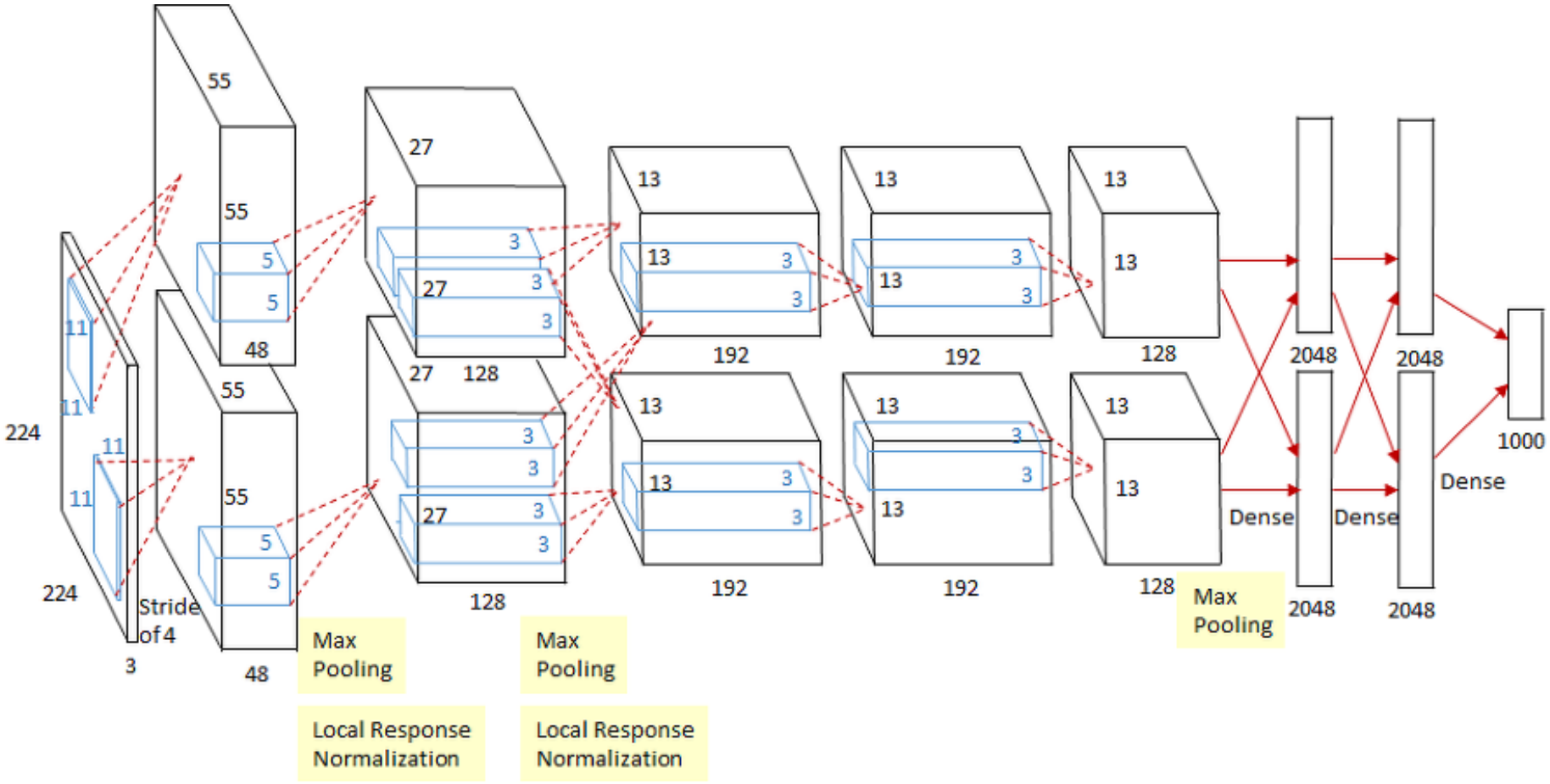
2021-06-19

조형래

CNN의 주요 모델

AlexNet
GoogleNet
incception
Resnet
DenseNet

AlexNet



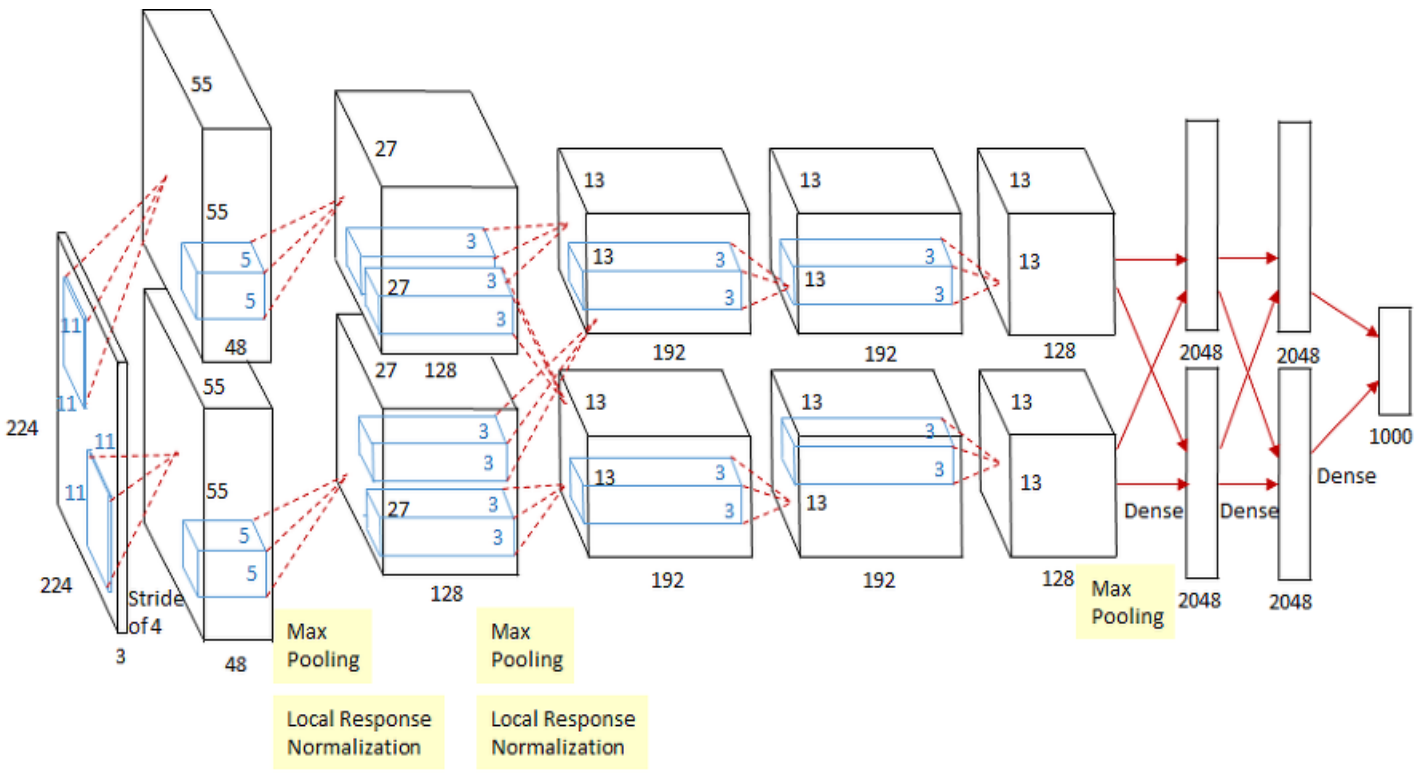
- conv layer, max-pooling layer, dropout layer 5개
- fully connected layer 3개
- nonlinearity function : ReLU
- batch stochastic gradient descent

AlexNet

처음 3채널의 224x224 이미지
11x11x3 필터 연산 96개가 존재, 55x55 Feature map이 총 96개가 출력
필터가 한번에 2개로 표시되었는데, 이는 병렬 연산을 위한 분할
stride는 4, 패딩은 0

특징 값에 ReLU를 하고,
3x3 윈도우로, Max pooling을 하고 나면 27 x 27 x 96라는 데이터가 남게 되는 것

다시 이를 가지고, 계속 위와 같은 특징 추출을 하다 보면, 13x13x256의 최종 특성이 남게 되는데,
이것에 풀링을 실행한 6 x 6 x 256이 최종으로 남고,
이를 serializing한 9216 벡터를 fully connected 레이어에 넣어줌으로써 객체 탐지를 하는 것
출력 층은 softmax로 각 클래스에 속할 확률을 구한다.

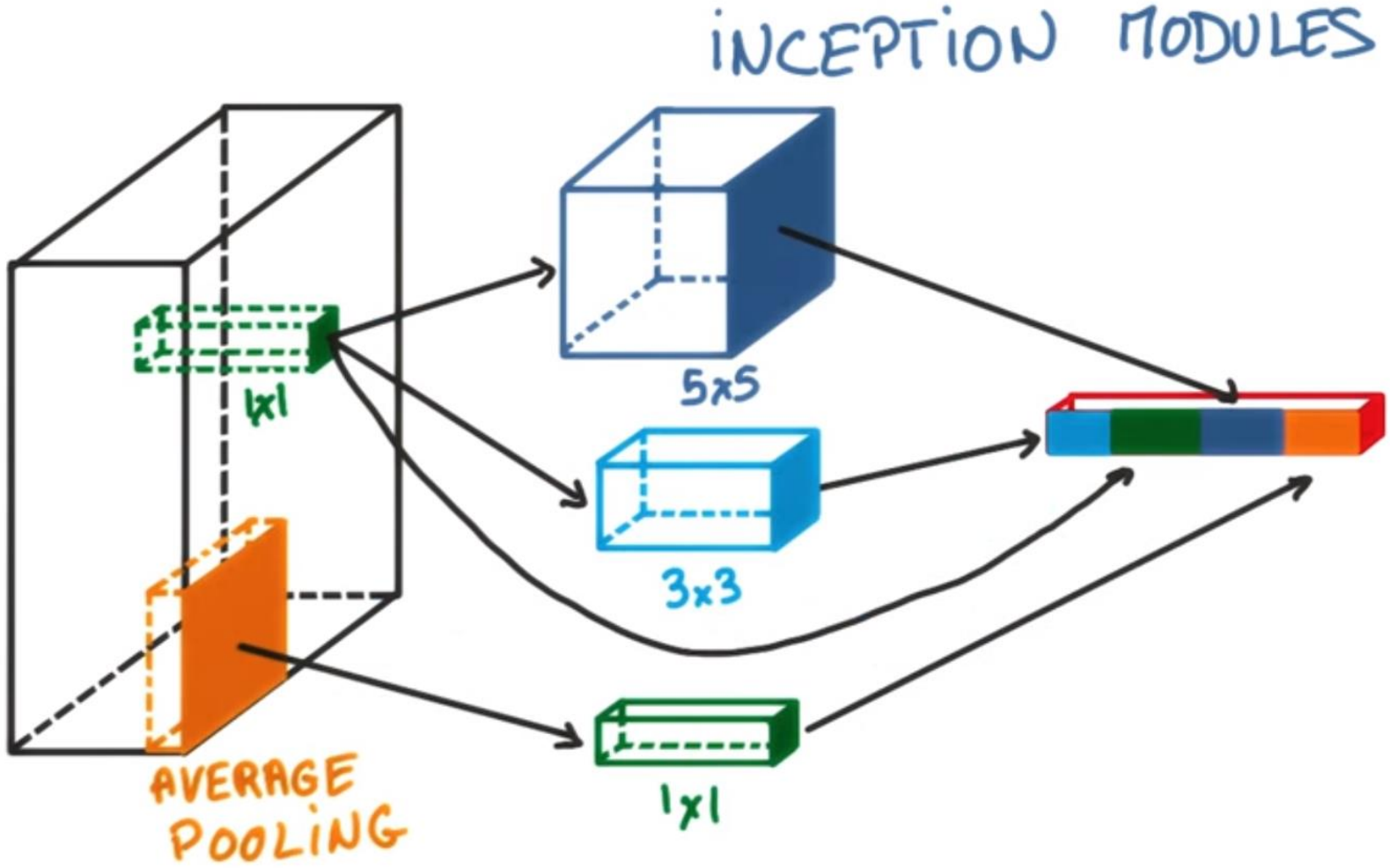


AlexNet

https://github.com/artjow/-AI-/blob/main/%EC%BD%94%EB%93%9C%EC%8B%A4%EC%8A%B5/PyTorch_on_Cloud_TPUs_Single_Core_Training_AlexNet_on_Fashion_MNIST.ipynb

GoogleNet

Inception module 적용;
입력데이터 이미지의 차원 수가 100×100×60이고, 1×1 conv filter를 20개 사용한다면 데이터의 차원 수는 100×100×20으로 줄어든다.
60개 채널(차원)에 달하는 하나의 픽셀이 20차원의 feature space로 선형변환, 차원 축소되는 것

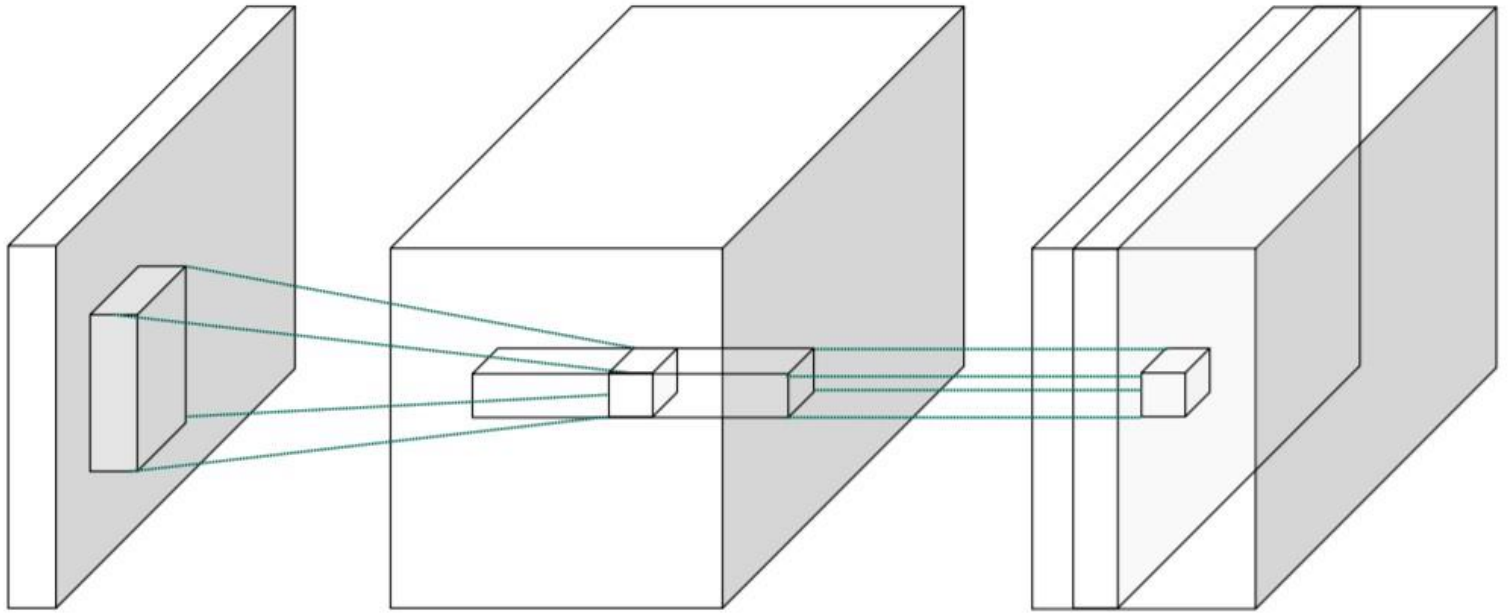


incception

1x1 Conv를 적절하게 사용하면 비선형적 함수를 더 잘 만들어낼 수 있게 되는 것

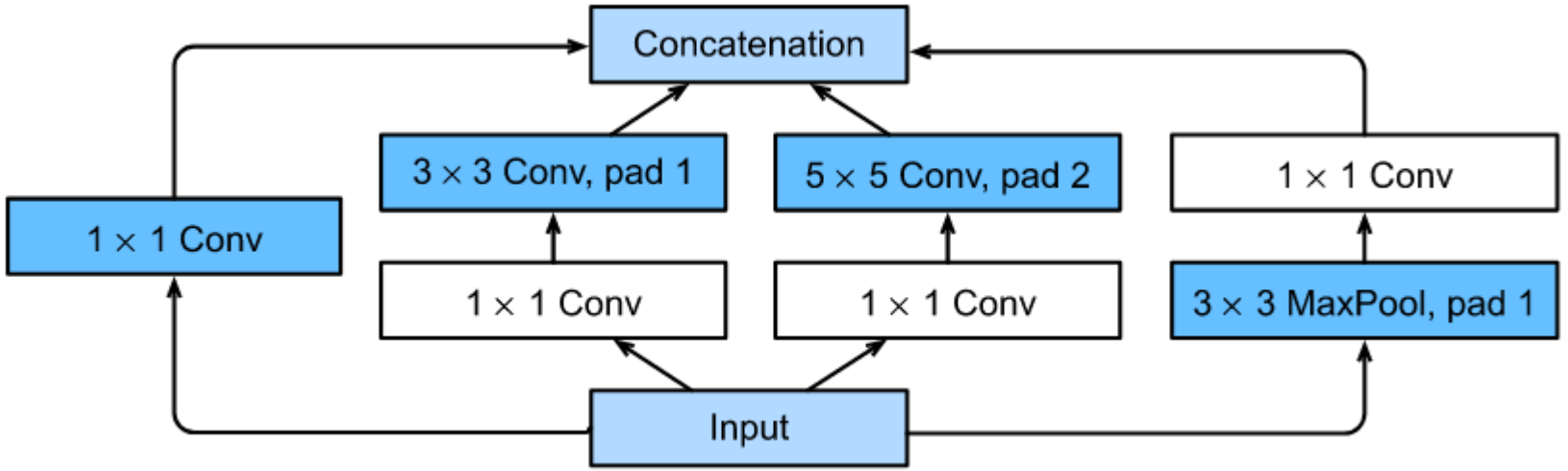
깊은 레이어를 1x1 Conv를 통해 차원 축소 함.

1x1 Conv를 채널 단위에서 Pooling을 하여 입력의 채널보다 작게 하는 dimension reduction(차원 축소)의 효과



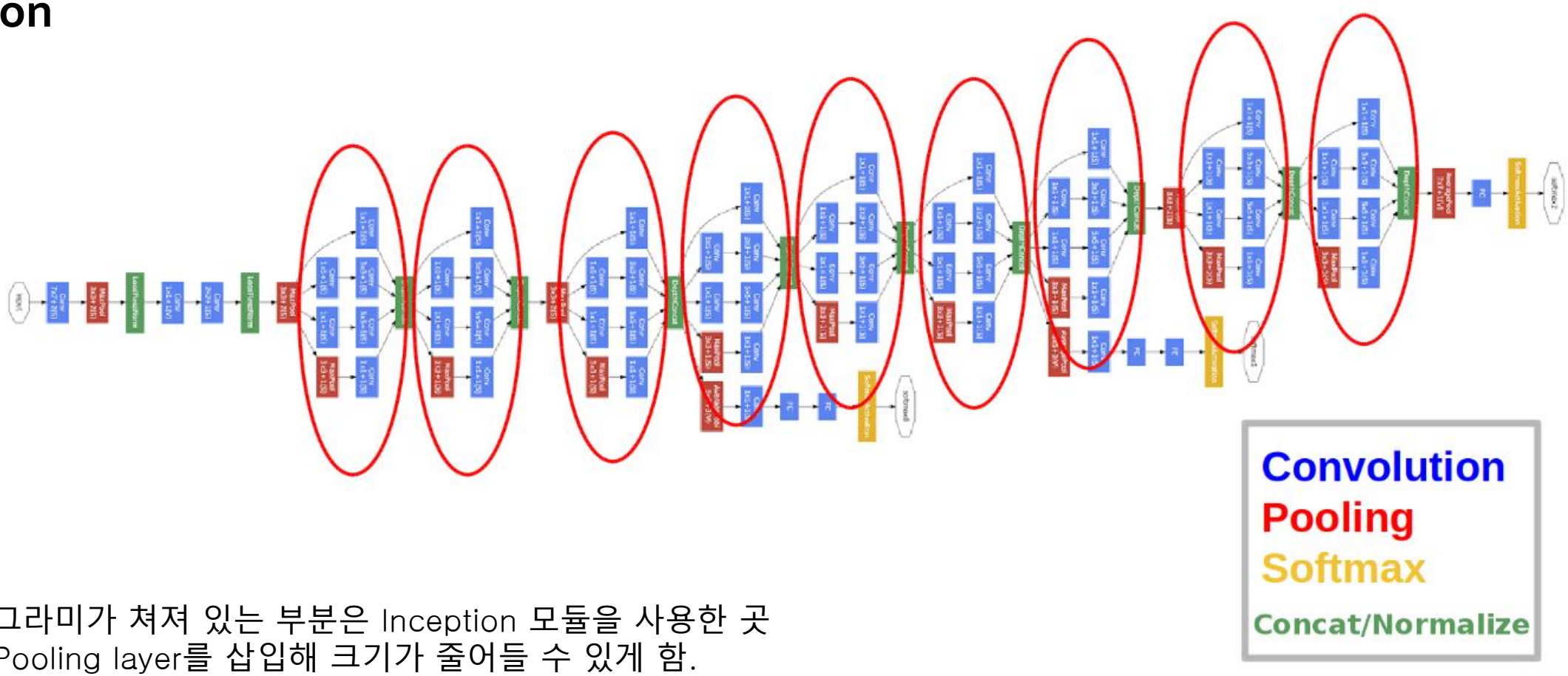
1x1 conv layer

incception



이전의 입력을 1x1 Conv에 넣어 channel을 줄였다가, 3x3나 5x5 Conv를 거치게 해 다시 확장
이렇게 되면 필요한 연산의 양이 확 줄어들게 된다. 또 Pooling의 경우 1x1 Conv를 뒤에 붙였는데 이는 Pooling연산의 결과 채널의 수가 이전의 입력과 동일하므로 이를 줄여주기 위함이다.
이렇게 sparse하게 각 연산을 거친 다음, dense한 outpu을 만들어내는데 H와 W는 모두 동일, 즉 Concat연산을 channel에 적용

incception



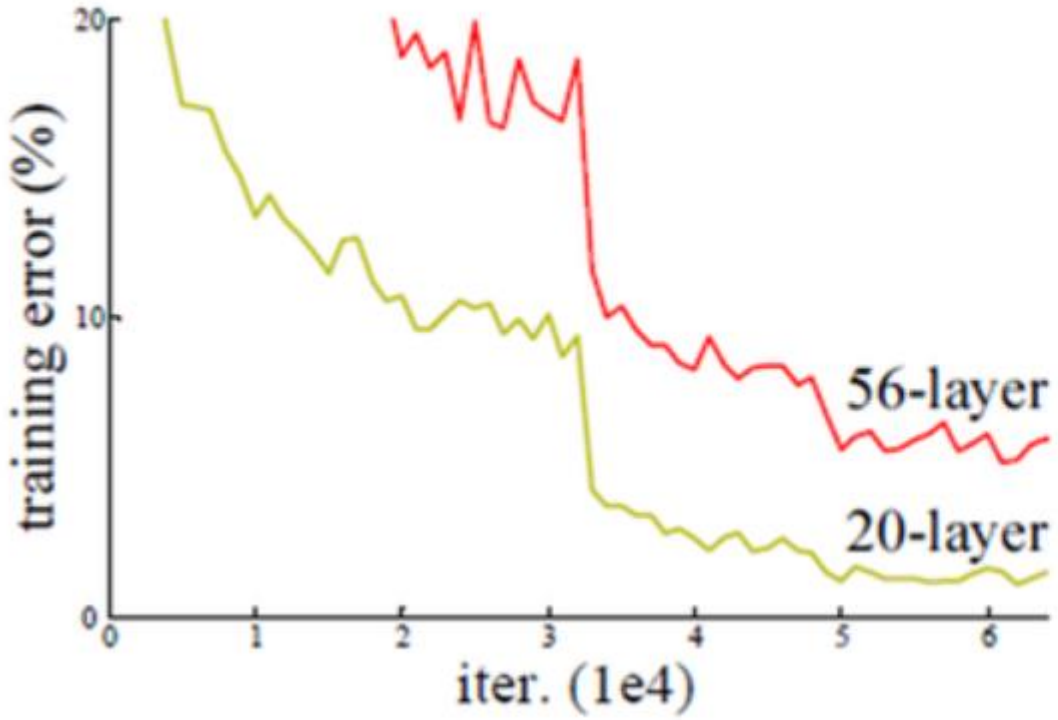
- 빨간색 동그라미가 쳐져 있는 부분은 Inception 모듈을 사용한 곳
- 중간중간 Pooling layer를 삽입해 크기가 줄어들 수 있게 함.
- 입력과 가까운 부분에는 Inception 모듈을 사용하지 않았다는 것(Inception의 효과가 없기때문에)
- 일반적인 CNN과 동일하게 Conv와 Pooling 연산을 수행
- softmax를 통해 결과를 뽑아내는 부분이 맨 끝에만 있는 것이 아니라, 중간 중간에 있는데 이것을 auxiliary classifier라고 한다(Vanishing Gradient 문제를 해결하기 위함, Loss를 맨 끝 뿐만 아니라 중간 중간에서 구하기 때문에 gradient가 적절하게 역전파 되게됨, auxiliary classifier의 loss는 0.3을 곱했음) 하지만 실제로 테스트하는 과정에서는 auxiliary classifier를 제거하고 맨 끝, 제일 마지막의 softmax만을 사용

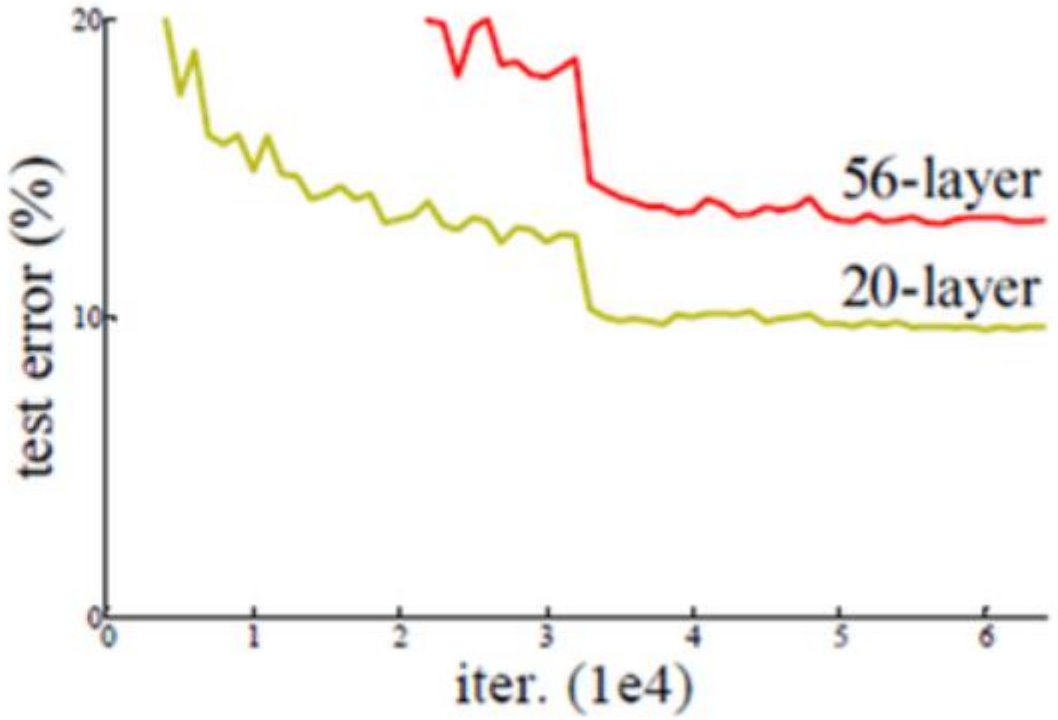
incception

https://github.com/artjow/-AI-/blob/main/chapter_convolutional-modern/googlenet.ipynb

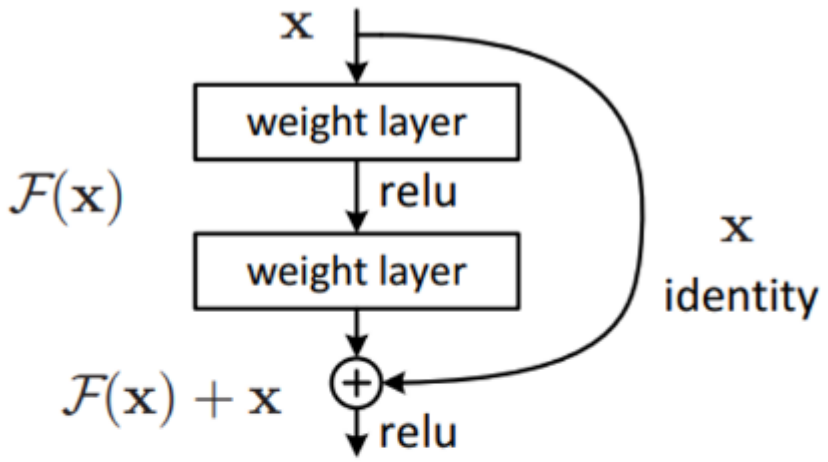
Resnet

네트워크가 더 깊어질 수록 Optimize(Train)하는 것이 더 어렵다.
따라서 Deep 네트워크는 Shallow 네트워크 만큼의 퍼포먼스를 보이지 않는다고 볼 수 있다.





Resnet



Residual *Block*

Residual *Mapping* $= f(x) + x$

Residual Block

F(x) : weight layer => relu => weight layer
x : identity

Residual Mapping)
weight layer들을 통과한 F(x)와 weight layer들을 통과하지 않은 x의 합

Residual Block)
Residual 블락 구조

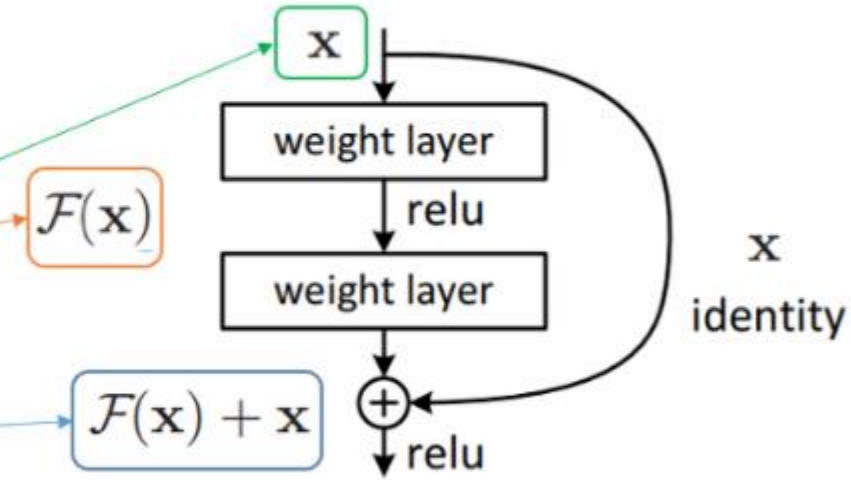
Residual Network(ResNet))
Residual Block이 쌓인 것

x가 그대로 보존되므로 **기존에 학습한 정보를 보존하고, 거기에 추가적으로 학습한 정보 f(x)**를 더한다.

Resnet

Residual Block

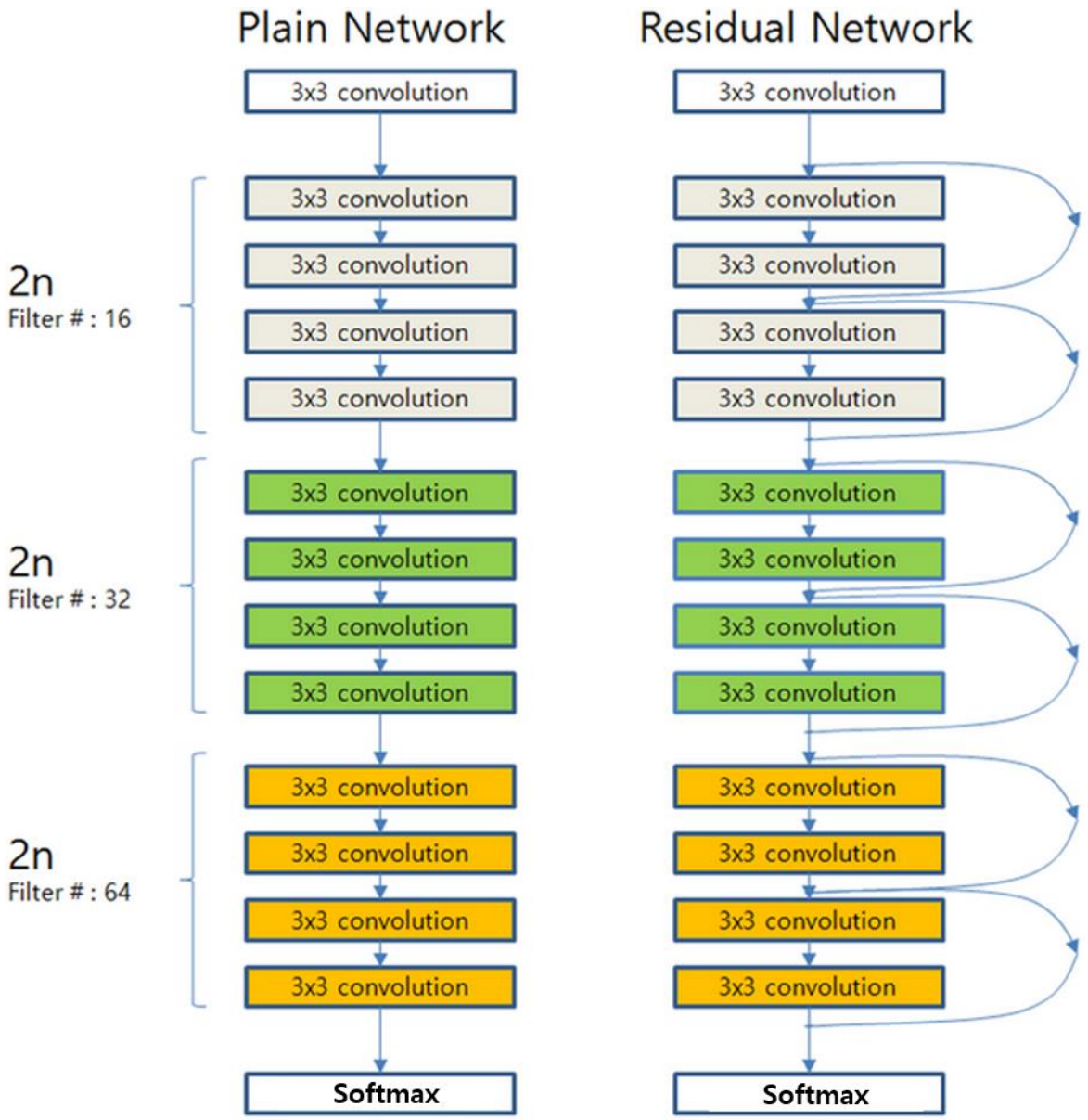
```
class Residual_Block(nn.Module):
  def __init__(self, in_dim, mid_dim, out_dim):
    super(Residual_Block,self).__init__()
    # Residual Block
    self.residual_block = nn.Sequential(
        nn.Conv2d(in_dim, mid_dim, kernel_size=3, padding=1),
        nn.ReLU,
        nn.Conv2d(mid_dim, out_dim, kernel_size=3, padding=1),
      )
    self.relu = nn.ReLU()

  def forward(self, x):
    out = self. residual_block(x)  # F(x)
    out = out + x  # F(x) + x
    out = self.relu(out)
    return out
```



Residual Mapping

Residual Block

Resnet


Plain Network
Residual Network
3x3 convolution
2n
Filter # : 16
2n
Filter # : 32
2n
Filter # : 64
Softmax
Softmax

Bottle Neck

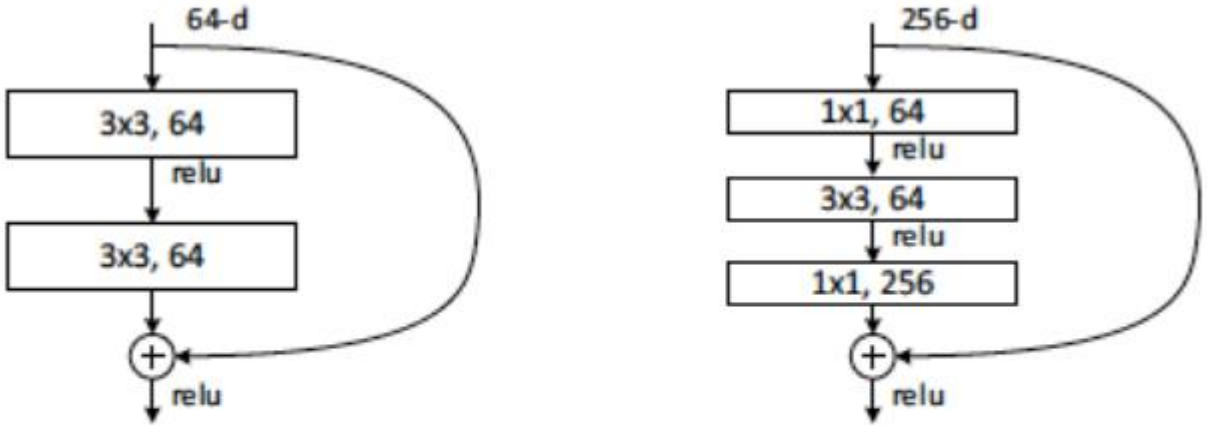


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

왼쪽은 Bottle Neck을 사용하지 않은 Residual Block이고 오른쪽은 Bottle Neck을 사용한 Residual Block

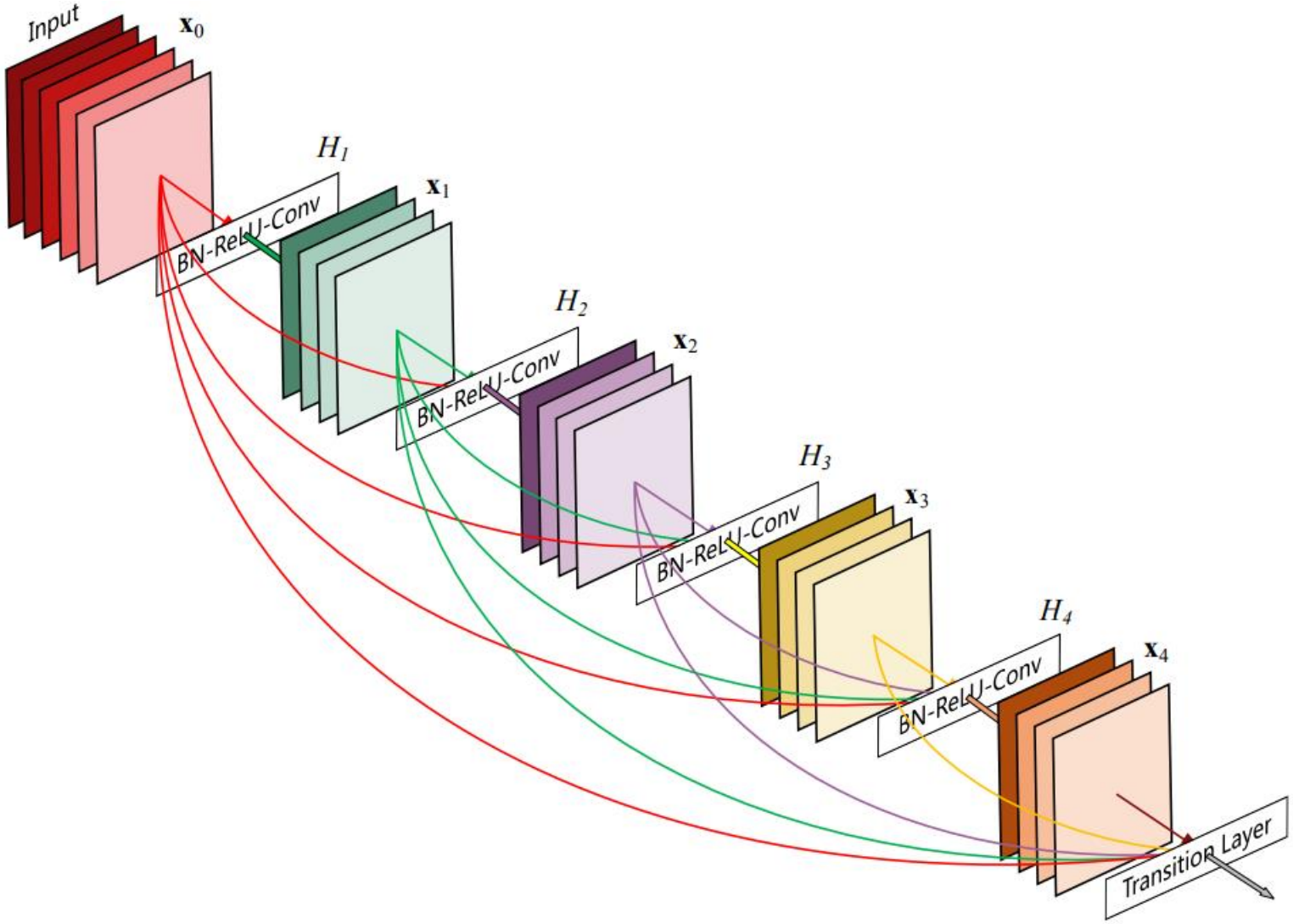
(ResNet50 이상에서는 Bottle Neck을 사용한 Block을 사용했다)

즉, Input x의 Depth를 64로 축소한 후에 f(x) 를 연산하고 마지막에 다시 256으로 높여준다.

Resnet

https://github.com/artjow/-AI-/blob/main/%EC%BD%94%EB%93%9C%EC%8B%A4%EC%8A%B5/resnet_(1).ipynb

DenseNet


Input
$\mathbf{x}_0$
H_1
BN-ReLU-Conv
$\mathbf{x}_1$
H_2
BN-ReLU-Conv
$\mathbf{x}_2$
H_3
BN-ReLU-Conv
$\mathbf{x}_3$
H_4
BN-ReLU-Conv
$\mathbf{x}_4$
Transition Layer

DenseNet

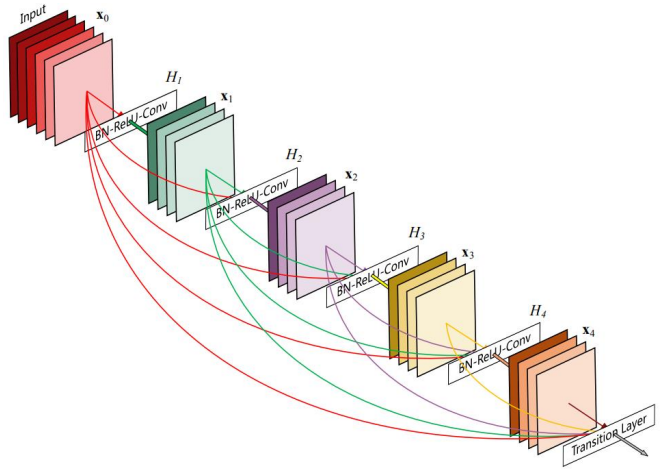
모든 레이어의 피쳐맵을 연결
이전 레이어의 피쳐맵을 그 이후의 모든 레이어의 피쳐맵에 연결
연결할 때는 ResNet과 다르게 덧셈이 아니라 concatenate(연결)을 수행한다. 따라서 연결할 때는, 피쳐맵 크기가 동일해야 한다. 피쳐맵을 계속해서 연결하면 채널 수가 많아질 수 있기 때문에, 각 레이어의 피쳐맵 채널 수는 굉장히 작은 값을 사용한다.

ResNet connectivity는 덧셈연산으로 이전레이어와 현재 레이어를 연결하므로 신경망에서 정보 흐름(information flow)이 지연되는 단점이 있다.

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}.$$

Dense connectivity는
이전 레이어를 모든 다음 레이어에 직접적으로 연결하므로 정보 흐름(information flow)이 향상된다.

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]),$$



DenseNet

연결(concatenation) 연산을 수행하기 위해서는 피쳐맵의 크기가 동일해야 한다.
하지만 피쳐맵 크기를 감소시키는 pooling 연산을 위해 Dense Block 개념을 도입.
Dense Block은 여러 레이어로 구성되어 있는데 Dense Block 사이에 pooling 연산을 수행
pooling 연산은 BN, 1x1conv, 2x2 avg_pool 순서로 수행한다.
transition layer이라고 부른다.
transition layer는 피쳐 맵의 크기와 채널 수를 감소시킨다.

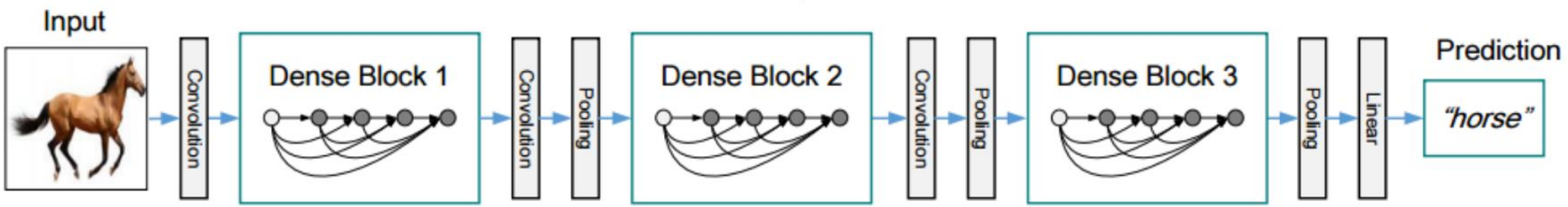
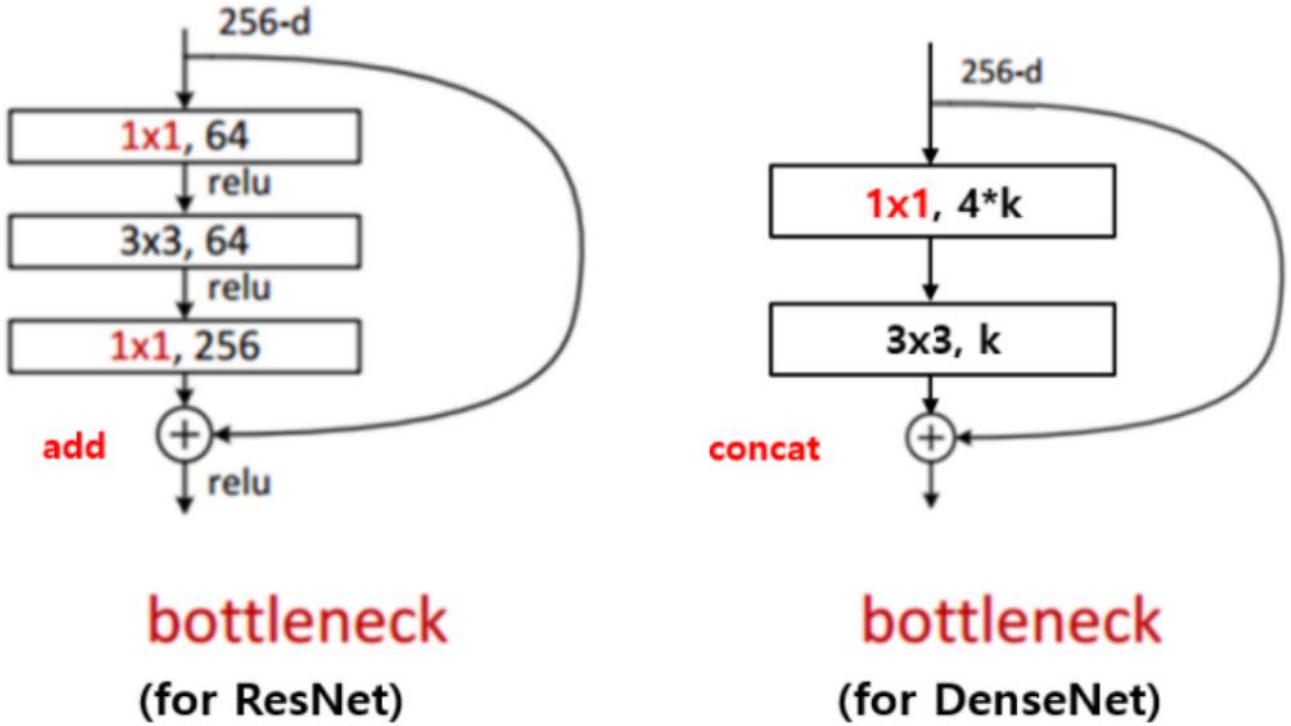


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

DenseNet은 3개의 Dense Block과 2개의 transition layer로 이루어져 있다.

Bottleneck layers

DenseNet은 Bottleneck layer를 사용
보틀넥 레이어는 3x3 conv의 입력값 채널을 조절하여 연산량에 이점을 얻기 위해 사용
1x1 conv는 3x3 conv의 입력값을 4k로 조절한다. 그리고 3x3 conv는 k개의 피쳐맵을 생성하고 이전 레이어와 연결된다.



DenseNet

https://github.com/artjow/-AI-/blob/main/%EC%BD%94%EB%93%9C%EC%8B%A4%EC%8A%B5/Simple_Implementation_of_Densely_Connected_Neural_Networks.ipynb