Modern CNN

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)에서 수상했던 **5개의 대표적인 CNN 모델**

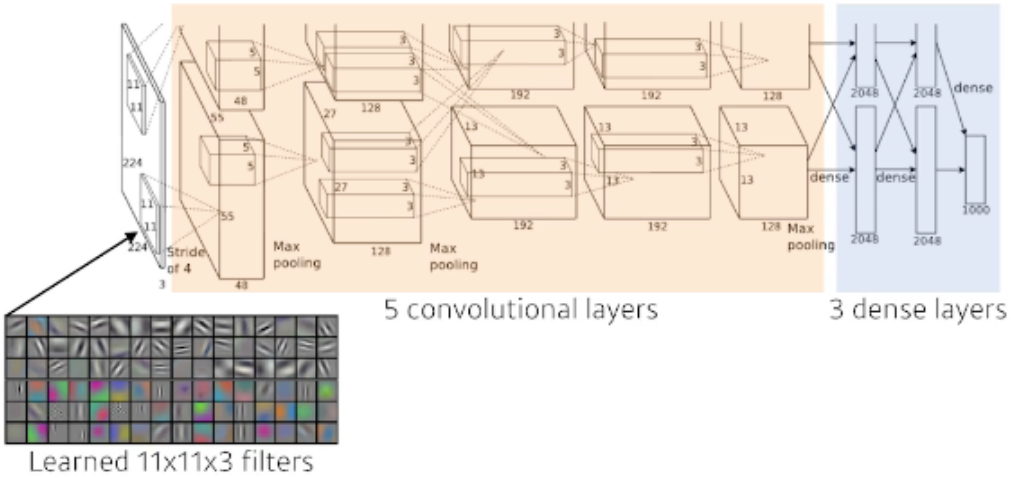
- **AlexNet** (2012)

  ILSVRC 최초로 DL 이용한 모델

  당시에는 흔하지 않은 방법들 (하나의 기준이 됨)

  최대 11×11 filter 사용 (파라미터 수 급증)

  5 convolutional layer, 3 dense layer



  - **Rectified Linear Unit(ReLU) activation**

    piecewise-linear

    But, 전체적으로 non-linear인 ReLU 함수를 통해 linear model 좋은 특성 유지

    입력값 그대로 전달 => gradient descent 최적화 쉬움

    sigmod/tanh의 gradient vanishing 문제 해결

  - **GPU implementation (2 GPUs)**

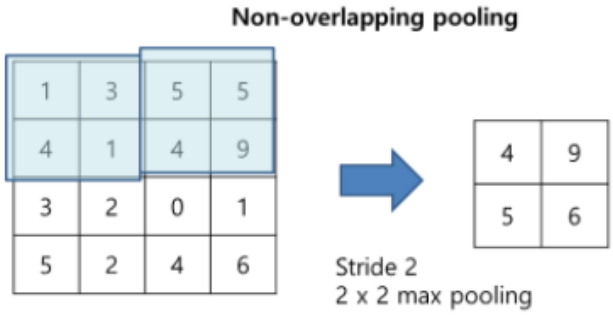
    당시, GPU VRAM 부족 => 2개의 GPU 이용해서 학습, 합침

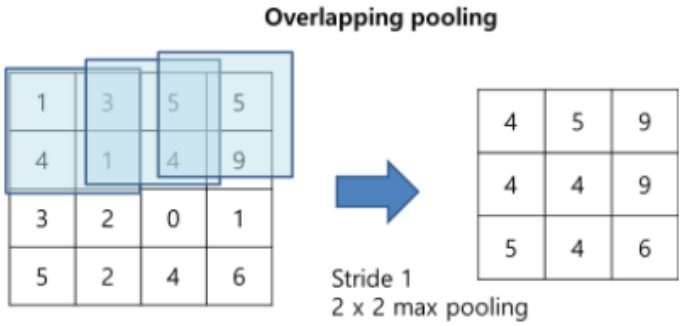
    - GPU-1: 주로 컬러와 상관없는 정보 추출하기 위한 커널 학습
    - GPU-2: 주로 컬러 관련된 정보 추출하기 위한 커널 학습

  - **Local response normalization**

    같은 위치의 픽셀에 대한 다수의 feature map 간의 정규화 적용

  - **Overlapping pooling**





일반적으로 각각 중복되지 않는 영역에서 pooling

But, AlexNet에서는 **overlapping pooling** 이용

- **Data augmentation**

  over-fitting 막기 위함

  - 256 X 256 이미지에서 227 X 227 이미지 random crop
  - RGB 채널 값 변화

- **Dropout**

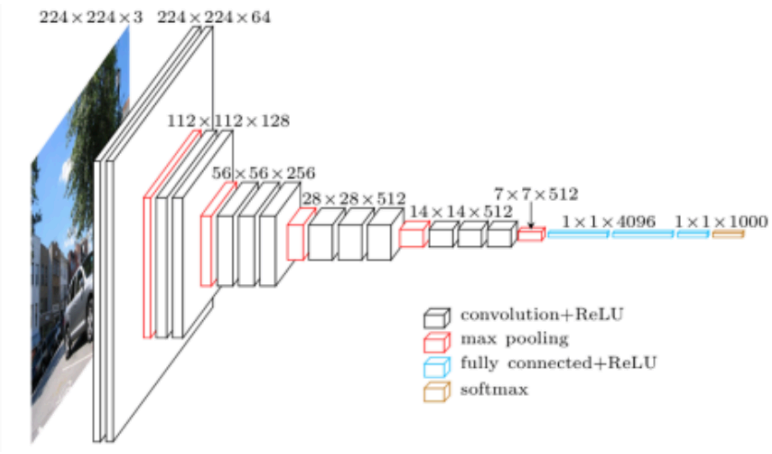
  over-fitting 막기 위함, 랜덤하게 일부 뉴런 생략하여 학습

- **VGGNet** (2014)

  GoogLeNet과 함께 주목, 간소한 차이로 2위

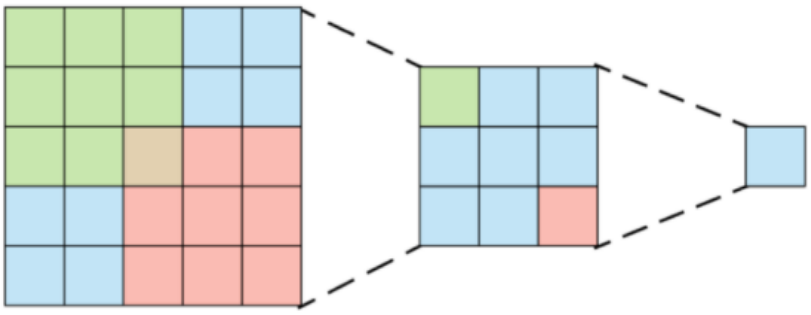
  network의 depth가 어떤 영향을 주는지 연구

  => **convolution filter를 하나의 크기로 고정, layer를 늘려나가는 방식으로 테스트 진행**

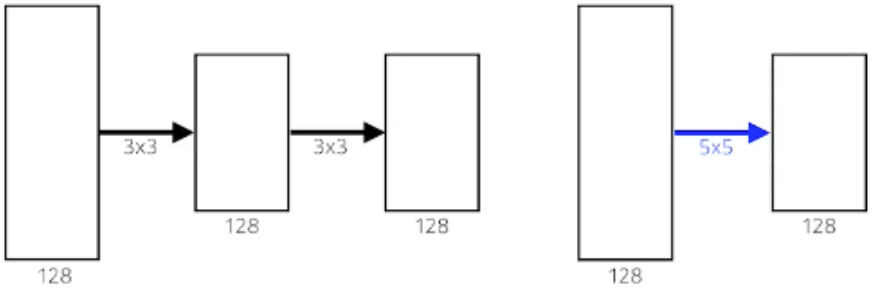


  - **Increasing depth with 3×3 convolution filters (with stride 1)**
    - **WHY?**

      receptive field 유지, parameter 수 줄이는 방법



5x5 receptive field 3x3 receptive field



Receptive field	5x5	5x5
# of params	3x3x128x128+3x3x128x128 = **294,912**	5*5*128*128 = **409,600**

- 1×1 **convolution for fully connected layers**
- **Droupout (p=0.5)**

  over-fitting 막기 위함, 랜덤하게 일부 뉴런 생략하여 학습
- **VGG16, VGG19**

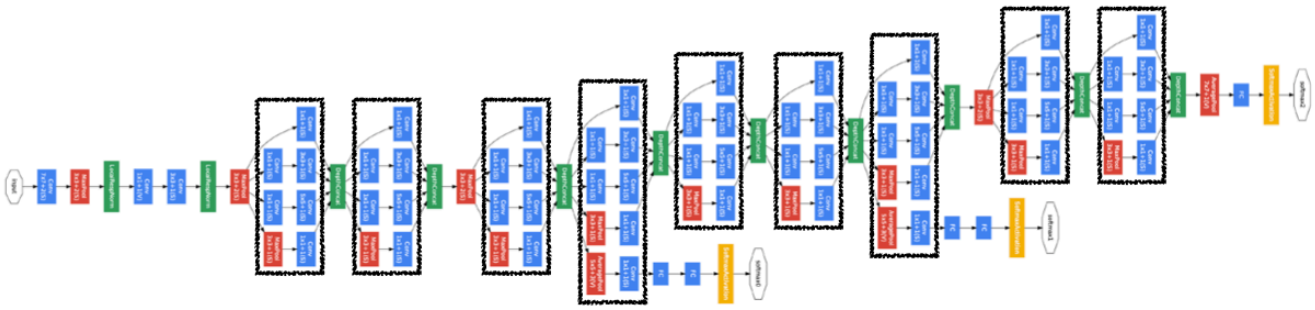
  16-layer와 비교했을 때, 19-layer은 error가 비슷하거나 더 안 좋은 결과 보여줌

- **GoogLeNet** (2014)

  ILSVRC에서 우승한 모델

  network-in-network(NIN) 개념 차용

  => inception block이라는 모듈의 적층 구조 가짐



22 layers

- **Network-in-Network(NIN)**

  NIN 개념 차용, Mlpconv layer와 같은 하나의 모듈로 완성되는 구조로 만듦
  - CNN의 convolutional layer: local receptive field에서 feature를 추출해내는 능력 우수

    but, filter의 특징이 linear

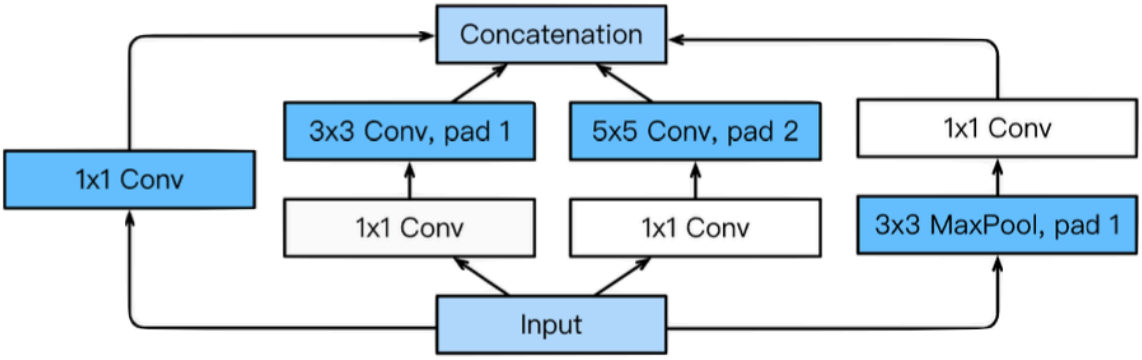
=> non-linear한 성질을 갖는 feature 추출하기에는 어려움

=> feature-map 개수를 늘려야 하는 문제, 필터 개수 늘리면 연산량 늘어남

=> Micro Neural network, convolution을 수행하기 위한 filter 대신 MLP(Multi-Layer Perceptron)를 사용, feature 추출

(a) Linear convolution layer (b) Mlpconv layer

- **Inception blocks**



$1 \times 1,\ 3 \times 3,\ 5 \times 5$의 필터 사용

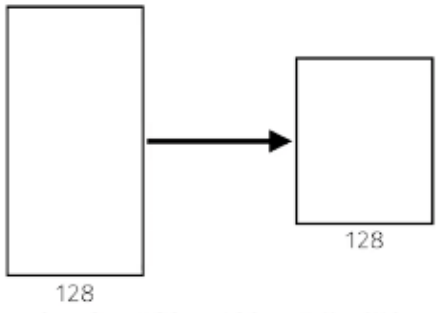
- 1×1: 입력데이터의 공간적 정보, 비교적 잘담게 만듦
- $3 \times 3,\ 5 \times 5$: 더 추상적이옥 퍼져있는 정보를 잘 담게 만듦

1×1 convolution이 각 필터에 적용되어 있음

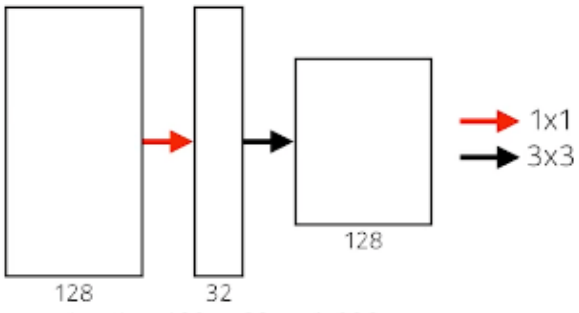
- **1X1 convolution**

가장 큰 이점: **차원감소 효과**

1×1 convolution 사용 => 파라미터 수 줄임, 깊은 망 구성 (연산량 조절 가능)



$$3 \times 3 \times 128 \times 128 = 147,456$$



$$1 \times 1 \times 128 \times 32 = 4,096$$
$$3 \times 3 \times 32 \times 128 = 36,864$$
$$4,096 + 36,864 = 40,960$$

- **Auxiliary Classifiers**

층이 깊어질수록 gradient vanishing 문제 발생

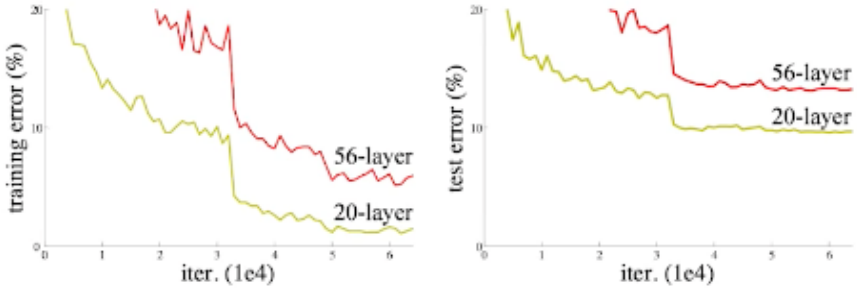
=> auxiliary classifiers로, 중간 단계에서 gradient 주입, 하위 레이어에도 잘 전달될 수 있도록 함

- 학습 시에만 사용, inference time에서는 제거

- **ResNet** (2015)

  152-layer 가짐

  깊이가 깊어지면 무조건 성능이 좋아지는가에 대한 의문에서부터 시작

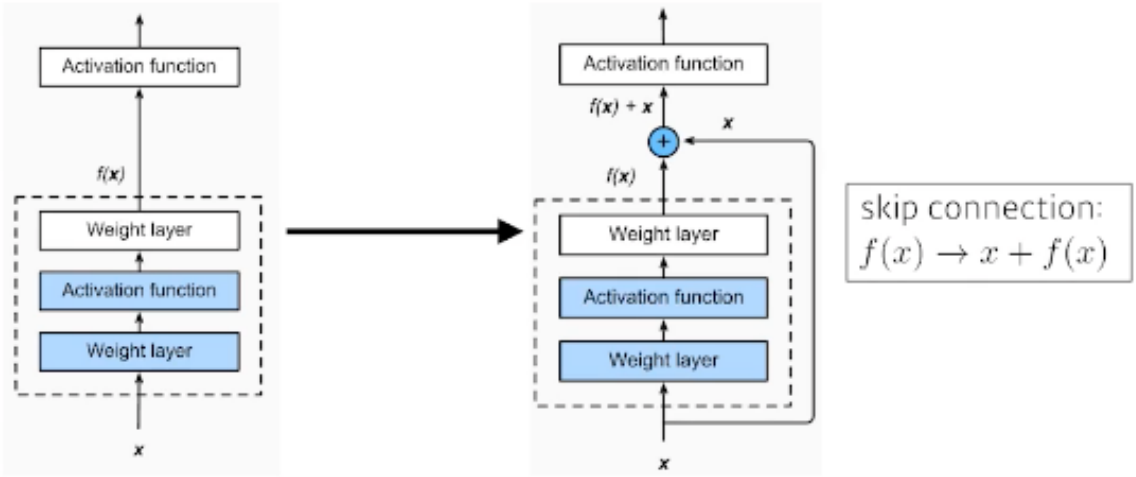


  => gradient vanishing & exploding로 인한 degradation 문제로 56-layer가 20-layer보다 더 나쁜 성능 보임

  => 무조건 깊게 X, 새로운 방법 찾아야 함


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, 256, /2
3x3 conv, 512, /2
avg pool
fc 1000

- Plain Network: 단순히 convolution 연산, 단순히 쌓음
- ResNet: Block단위로 파라미터를 전달하기 전에 이전의 값을 더하는 방식
- **Residual Block**

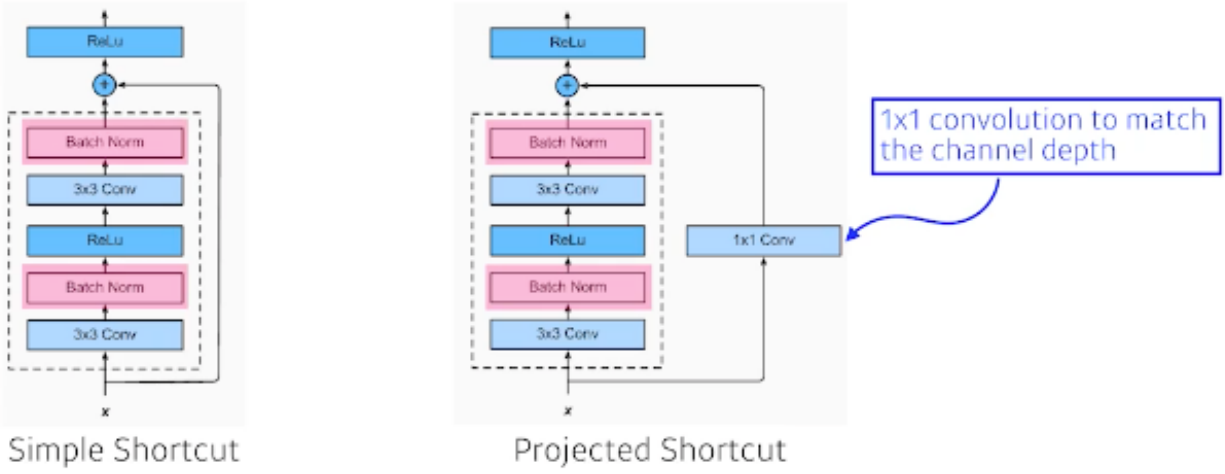


  - **Residual Mapping**: weight layer들을 통과한 $f(x)$와 weight layer들을 통과하지 않은 x의 합

    간단, over-fitting 해결

    상위 레이어의 어떤 gradient라고 해도 그 값이 변하지 않고 그대로 전파 => **Vanishing gradient 문제 해결**

    => 성능 향상
  - **Residual Block**: 그림의 구조
  - **Residual Network(ResNet)**: residual block이 쌓인 것
- **Simple Shortcut / Projected Shortcut**



VGG19 기반으로 한 Plain-model에 shortcut 추가

shortcut 적용할 때, $F(x) + x$에서 $F(x)$는 convolution filter 통과

=> filter 수 증가, 차원수 높아짐 => x의 차원 높여줘야 함

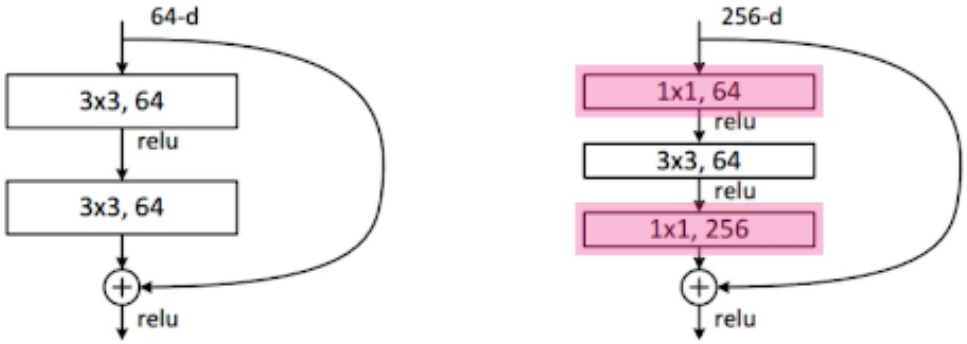
  - simple shortcut

    $F(x)$차원이 증가하지 않은 경우, identity shortcut

    차원이 증가한 경우, zero padding shortcut
  - projected shortcut

    1×1 convolution 통해 증가

- **Batch normalization after convolutions**
- **Bottleneck architecture**



도로의 병목 현상과 비슷하다고 해서 bootleneck 구조라고 부름

학습시간 줄이기 위해 1×1 convolution 이용한 bottleneck 구조 적용

```
# standard
class Standard(nn.Module):
    def __init__(self, in_dim=256, mid_dim=64, out_dim=64):
        super(BuildingBlock, self).__init__()
        self.building_block = nn.Sequential(
            nn.Conv2d(in_channels=in_dim, out_channels=mid_dim, kernel_size=3,
padding=1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=out_dim, kernel_size=3,
padding=1, bias=False),
        )
        self.relu = nn.ReLU()

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

# BottleNeck
class BottleNeck(nn.Module):
    def __init__(self, in_dim=256, mid_dim=64, out_dim=256):
        super(BottleNeck, self).__init__()
        self.bottleneck = nn.Sequential(
            nn.Conv2d(in_channels=in_dim, out_channels=mid_dim, kernel_size=1,
bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=mid_dim, kernel_size=3,
padding=1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=in_dim, kernel_size=1,
bias=False),
        )
```

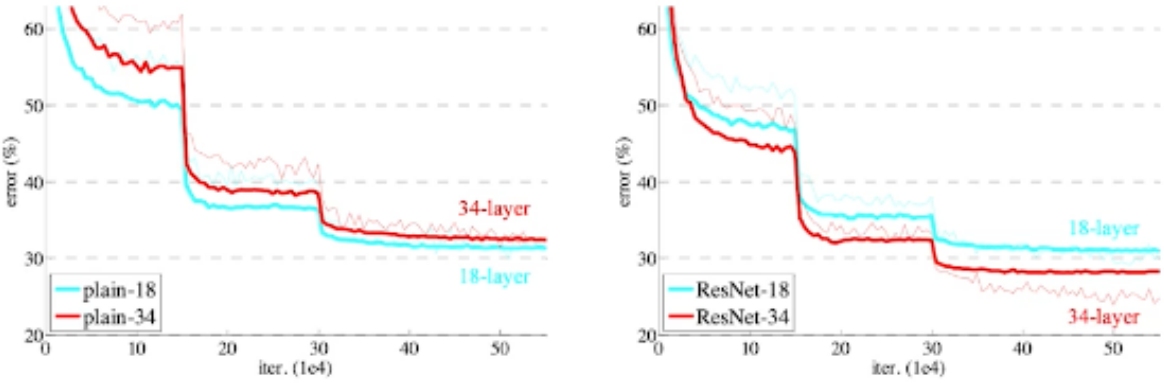
```
        self.relu = nn.ReLU()

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

- **He initialization**

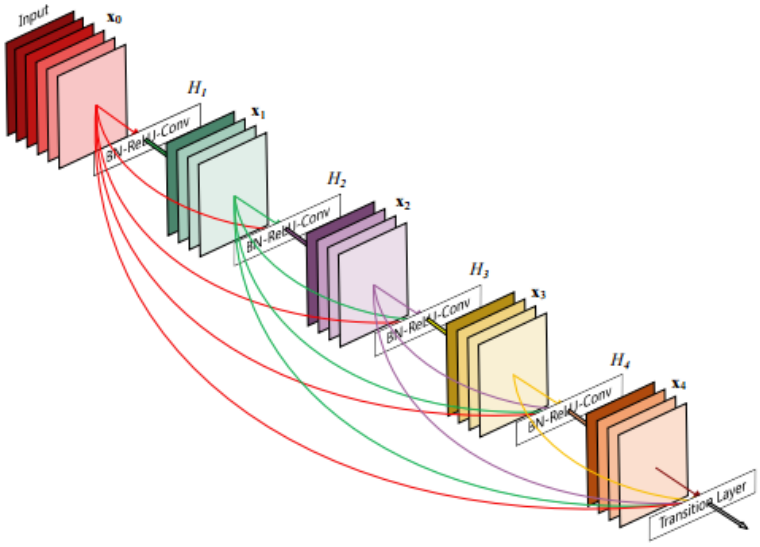
  indentity mapping으로 이전 입력값을 그대로 전달

  => 가중치 초기값 설정, 학습과정에 큰 영향 끼침

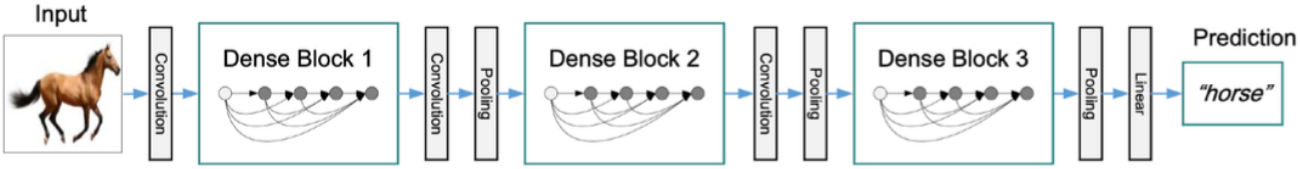
  => 적합한 초기화 방법으로 He initialization 사용



- **DenseNet** (2017)



Conv 1 input channel : 6, 2:6+4, ..., Transition Layer input channel: 6+4+4+4+4

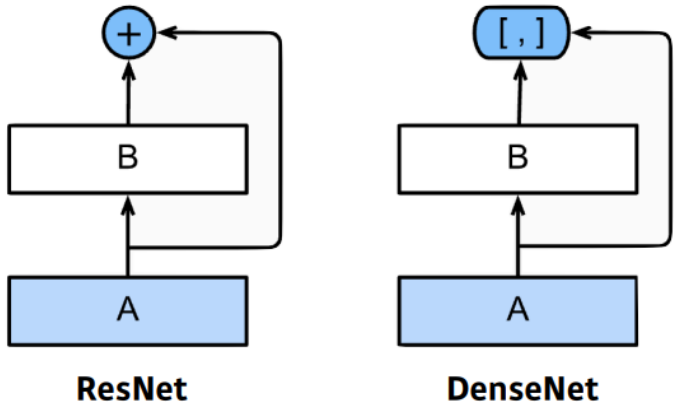


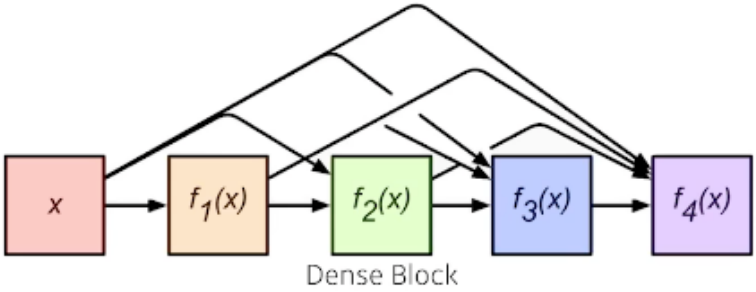
  - concatenation instead of addition

- **Dense Block**
  - ResNet

    feature map끼리 더해주는 방식
  - DenseNet

    순차적으로 앞단의 모든 layer의 결과를 그 다음 layer에 dense하게 concatencate





$$\mathbf{x} \mapsto [\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f(\mathbf{x})), f_3(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x})), f_4(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x}), f_3(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x}))))]$$

- **Transition Block**

  dense block으로 채널 커짐, 파라미터 수 급격히 증가
  - Batch normalization = 1×1 conv => 2×2 AvgPooling

  => transition block으로 차원 수 줄임

네트워크은 깊어지고, 파라미터는 줄어들고, 성능은 향상

CNN	Points
VGG	repeated 3×3 blocks
GoogLeNet	1×1 convolution
ResNet	skip-connection
DenseNet	concatenation