

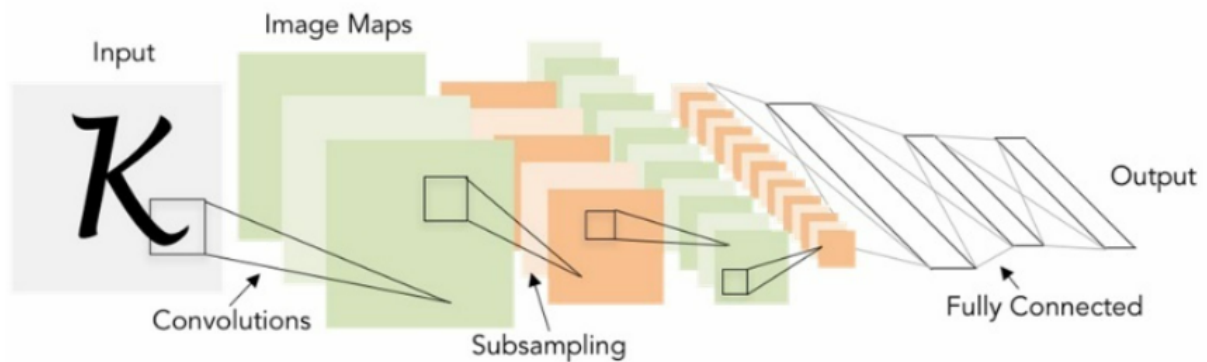
Week11 9강 CNN Architecture

ImageNet에서 우승을 한 AlexNet, VGGNet, GoogLeNet, ResNet과 역사적으로 의미있는 LeNet, 최신의 CNN모델을 살펴본다.

1. LeNet

Review: LeNet-5

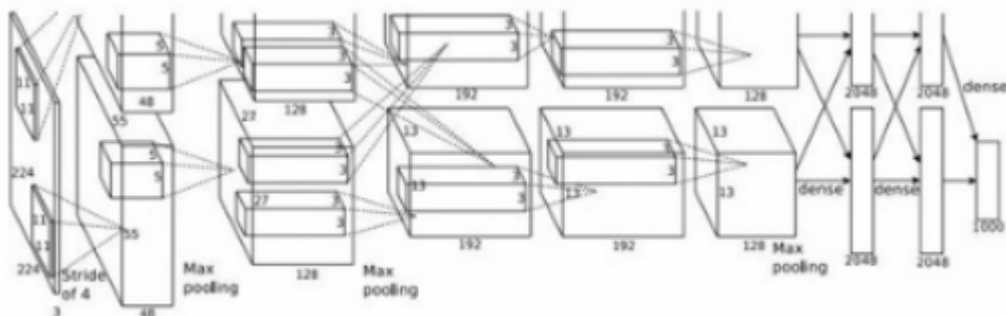
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

- 가장 초창기의 ConvNet
- 숫자 인식에서 성공을 거둠

2. AlexNet



- 2012년 ImageNet Challenge에 등장
- 최초의 Large scale CNN

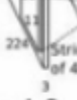
AlexNet Architecture

1. Convolution layer → Max pooling → Normalization 구조가 두 번 반복된다.
2. CONV3, 4, 5로 conv layer가 붙고 pooling layer (Max POOL3)
3. 마지막에 FC-layer(FC6, FC7, FC8)

⇒ 5개의 Conv Layer와 2개의 FC-Layer로 구성

AlexNet Output size와 parameter 개수

Full (simplified) AlexNet architecture:



```

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2 [27x27x96]
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2 [13x13x256]
[13x13x256] NORM2: Normalization layer [13x13x384]
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1 [13x13x384]
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1 [13x13x256]
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1 [6x6x256]
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)
  
```

- input : 227 x 227 x 3 (ImageNet)
1. CONV1 (11 x 11, stride 4)

Output size = 55 x 55 x 96 $((227 - 11) / 4 + 1 = 55)$

Parameter = 11 x 11 x 96 * 3 = 35K
 2. POOL2 (3 x 3, stride = 2)

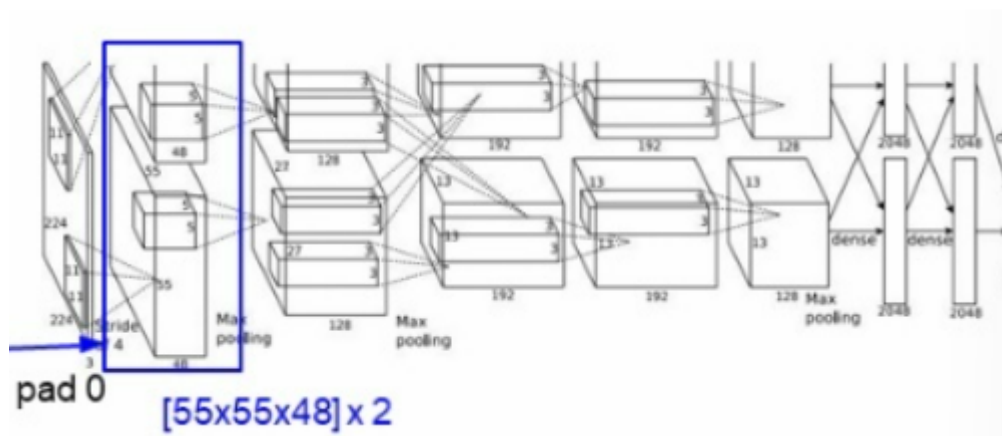
Output size = 27 x 27 x 96 (depth 유지)

Parameter = 없음 (Pooling layer는 가중치가 없고 특정 지역에서 큰 값 뽑아내는 역할)

끝부분 : FC6, FC7, FC8 \Rightarrow Softmax 통과 \Rightarrow 1000 ImageNet classes

AlexNet 특징

- ReLU 처음 사용
- Data augmentation을 많이 사용
- dropout사용
- Batch 사용 (Batch size = 128)
- SGD momentum 사용
- 초기 learning rate = $1e-2$, val accuracy가 올라가지 않는 시점에서 $1e-10$ 으로 줄임
- Weight decay 사용(regularization)
- 앙상블 사용



AlexNet에서 모델이 두개로 나뉘어져서 교차하는데 당시 GTX850은 3GB여서 전체 레이어를 GPU에 넣을 수 없었음

\Rightarrow Feature map을 절반씩 가짐

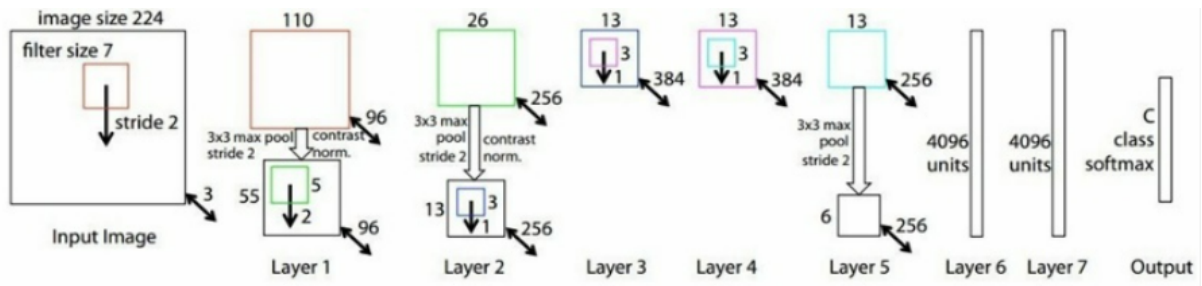
\Rightarrow CONV1, CONV2, CONV4, CONV5에서는 각각 GPU의 feature map만 본다.

(48 features)

\Rightarrow CONV3, FC6, FC7, FC8에서는 전체 feature map 연결됨

2. ZFNet

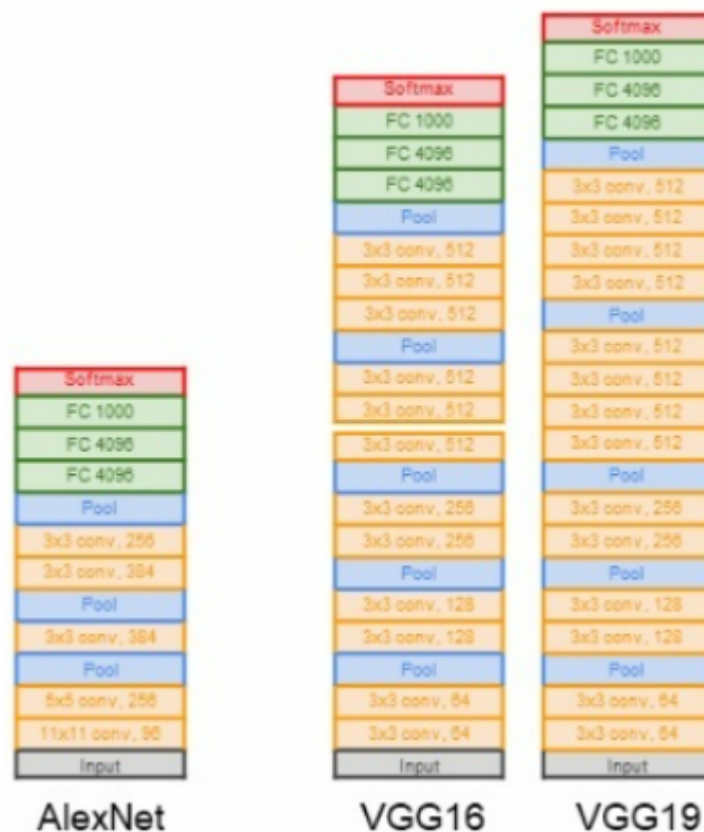
- AlexNet 기반 (2013년 우승)



- AlexNet에서 stride size, 필터 수 등의 하이퍼 파라미터 조절
- AlexNet과 레이어 수, 기본 구조 같다.

3. VGGNet

- AlexNet보다 더 깊은 layer(16-19), 더 작은 filter
- (3 x 3만 사용, stride = 1, pad = 1), (2 x 2 max pool, stride = 2)



3 x 3 filter로 작은 filter 사용한 이유

⇒ 3개의 3 x 3 conv사용하면 7 x 7과 같은 효과

(더 깊으면서 더 적은 파라미터를 가져 효율적임)

Output과 Parameter (VGG16기준)

```
INPUT: [224x224x3]      memory: 224*224*3=150K  params: 0      (not counting biases)
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*64)*64 = 36,864
POOL2: [112x112x64]     memory: 112*112*64=800K  params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M  params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M  params: (3*3*128)*128 = 147,456
POOL2: [56x56x128]      memory: 56*56*128=400K  params: 0
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*128)*256 = 294,912 C
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824 CO
POOL2: [28x28x256]      memory: 28*28*256=200K  params: 0
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*256)*512 = 1,179,648 C
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296 CO
POOL2: [14x14x512]      memory: 14*14*512=100K  params: 0
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296 C
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296 CO
POOL2: [7x7x512]        memory: 7*7*512=25K  params: 0
FC: [1x1x4096]          memory: 4096  params: 7*7*512*4096 = 102,760,448 F
C: [1x1x4096]           memory: 4096  params: 4096*4096 = 16,777,216
FC: [1x1x1000]          memory: 1000  params: 4096*1000 = 4,096,000
```

TOTALmemory: 24M * 4 bytes ~ 96MB / image (only forward! ~*2 for bwd)
TOTALparams: 138M parameters

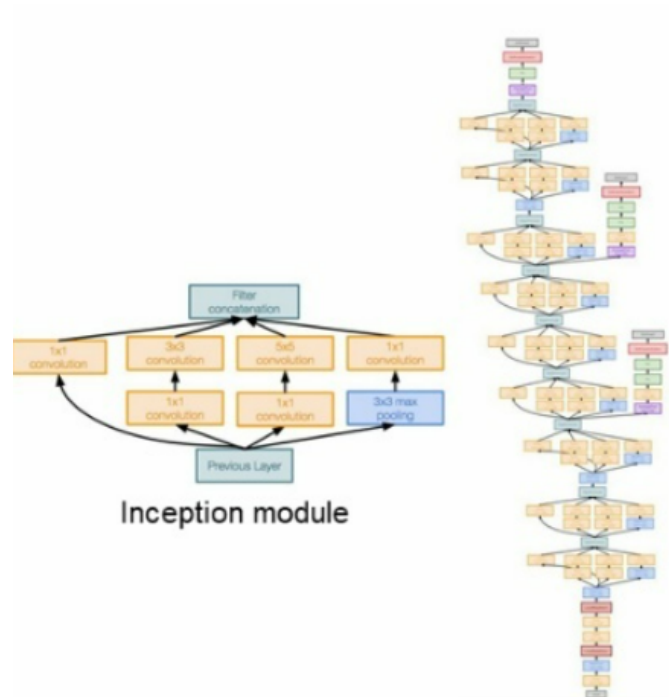
VGG는 메모리 사용량이 많다.

VGGNet 특징

- ImageNet에서 2등, Localization에서 1등
- Local response normalization사용 안함(AlexNet에서도 효과 없어서 사용 안함)
- 앙상블 기법 사용
- VGG마지막의 FC7 : 4096 size 레이어, 좋은 feature representation

4. GoogLeNet

- 2014년 Classification Challenge 우승
- 22layer \Rightarrow 근데 5M parameter 밖에 안됨 (AlexNet보다 12배 적음)
- Inception module 사용



Inception module

- Inception module 내부에 동일한 입력을 받는 서로 다른 다양한 필터들이 병렬로 존재한다.
- 출력들을 모두 Depth 방향으로 합친다.
- 하나의 tensor가 다음 layer로 전달됨.

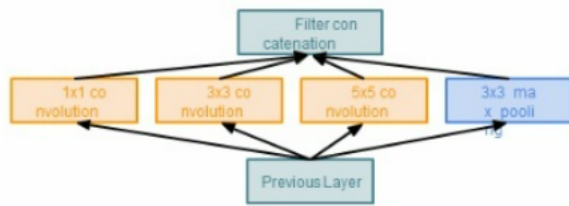
\Rightarrow 이건 naive한 방식(연산량이 너무 많다.)

ex) 그림 예시에서 input size = $28 \times 28 \times 256$ 에서 output $28 \times 28 \times 672$ 가 됨.

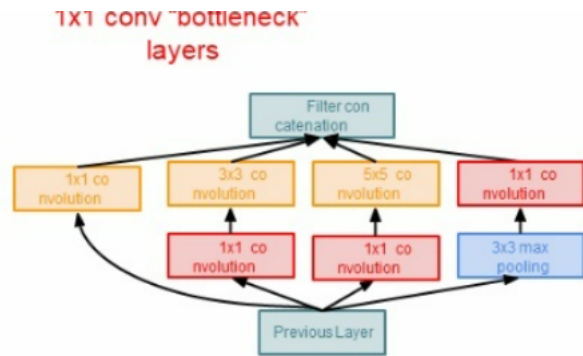
spatial dimension은 유지되고, depth가 증가함

\Rightarrow depth를 줄이는 방법 필요

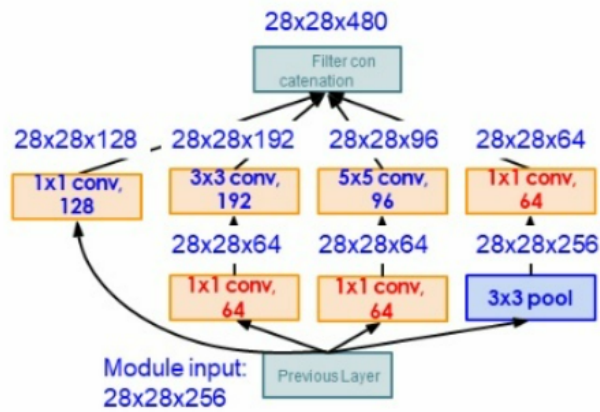
Bottleneck layer



Naive Inception module



Inception module with dimension reduction



Inception module with dimension reduction

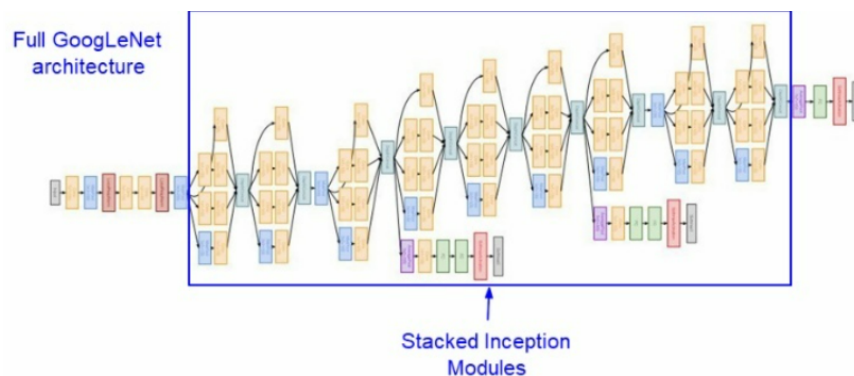
Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256 [1x1 conv, 64] 28x28x64x1x1x256 [1x1 conv, 128] 28x28x128x1x1x256 [3x3 conv, 192] 28x28x192x3x3x64 [5x5 conv, 96] 28x28x96x5x5x64 [1x1 conv, 64] 28x28x64x1x1x256 **Total: 358M ops**

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

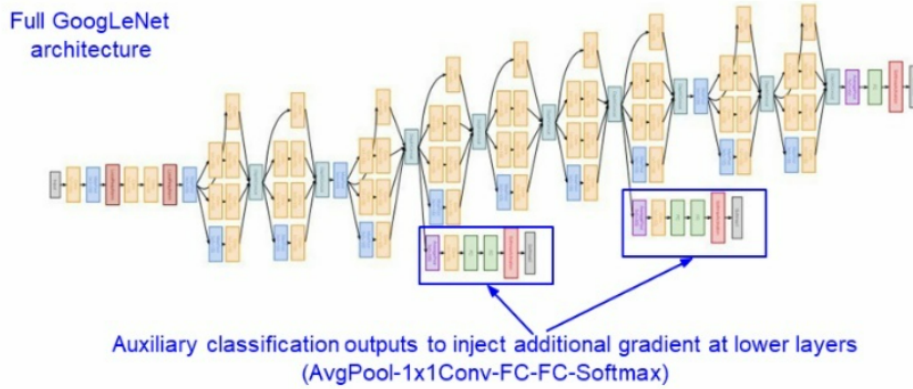
연산량이 줄어듦.

GoogLeNet Architecture



앞부분은 일반적인 네트워크 구조이고 이후 Inception module이 쌓인다.

뒷부분은 classifier



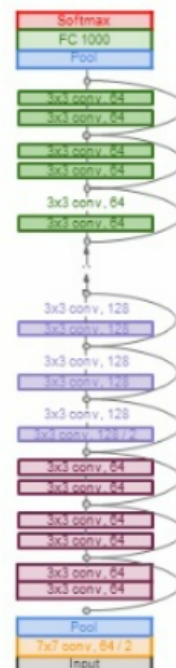
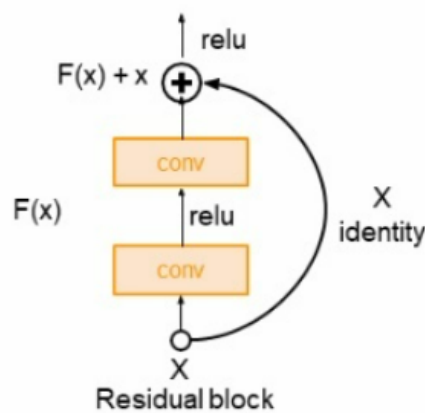
보조 분류기(auxiliary classifier) : Average pooling, fc, softmax로 classification, loss도 구한다.

네트워크가 너무 깊어서 gradient를 잃어버릴 수 있어 추가적인 gradient를 얻기 위한 것.

GoogLeNet 특징

- 22layer
- Inception module이 있고 FC layer를 들어냄
- AlexNet보다 12배 작은 parameter

5. ResNet



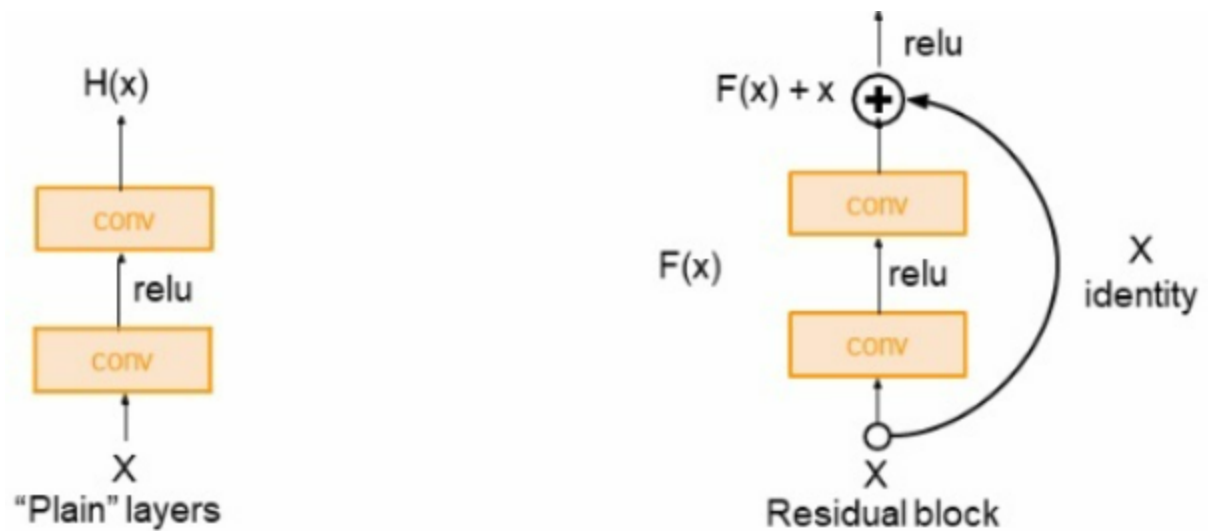
- 152 layer
- Residual connection을 사용

Residual connection

layer가 깊어지면 성능이 좋아질꺼라 생각했는데 실제로 그렇지 않았음.

(overfitting이랑 다른 방식으로 성능이 안좋아짐)

⇒ 깊은 모델 학습 시 optimization에 문제가 생긴다. (모델이 깊어질수록 최적화하기 어렵다.)

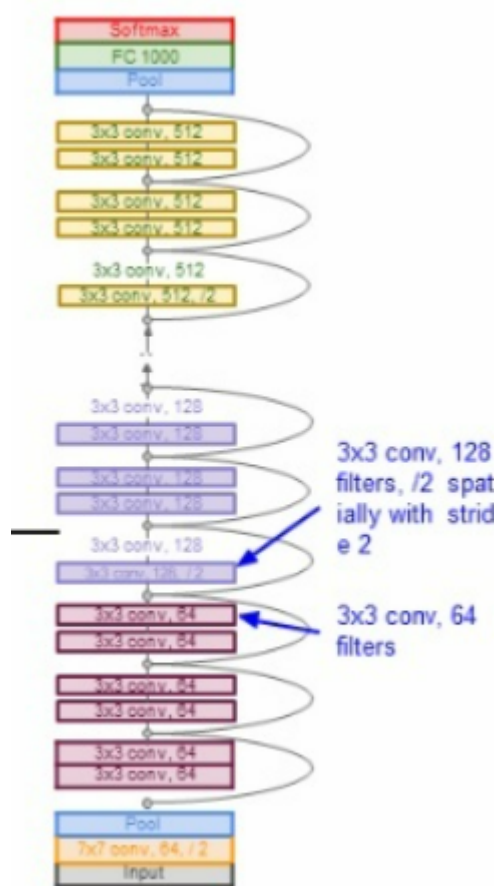


$h(x) = f(x) + x$ 를 $f(x) = h(x) - x$ 로 구함

앞부분의 얇은 모델의 가중치를 그대로 가져옴

⇒ 실제 레이어는 변화량만 학습하면 된다.

특징



주기적으로 필터를 2배씩 늘리고 stride를 2로 두어 downsampling 진행
bottleneck 사용 (layer가 50개를 넘어가면 보통 사용)

BatchNormalization

Xavier / 2를 사용해 초기화