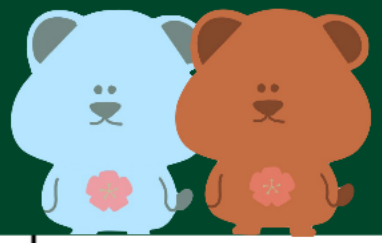
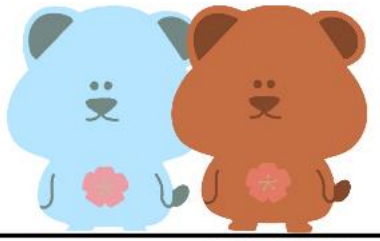


6-1장. 이미지 분류를 위한 신경망

이다현, 이서영, 손소현

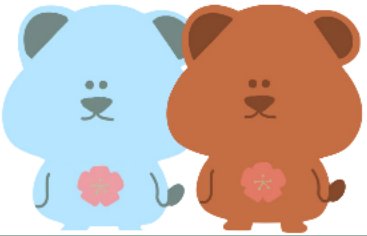
EWHA EURON

목차



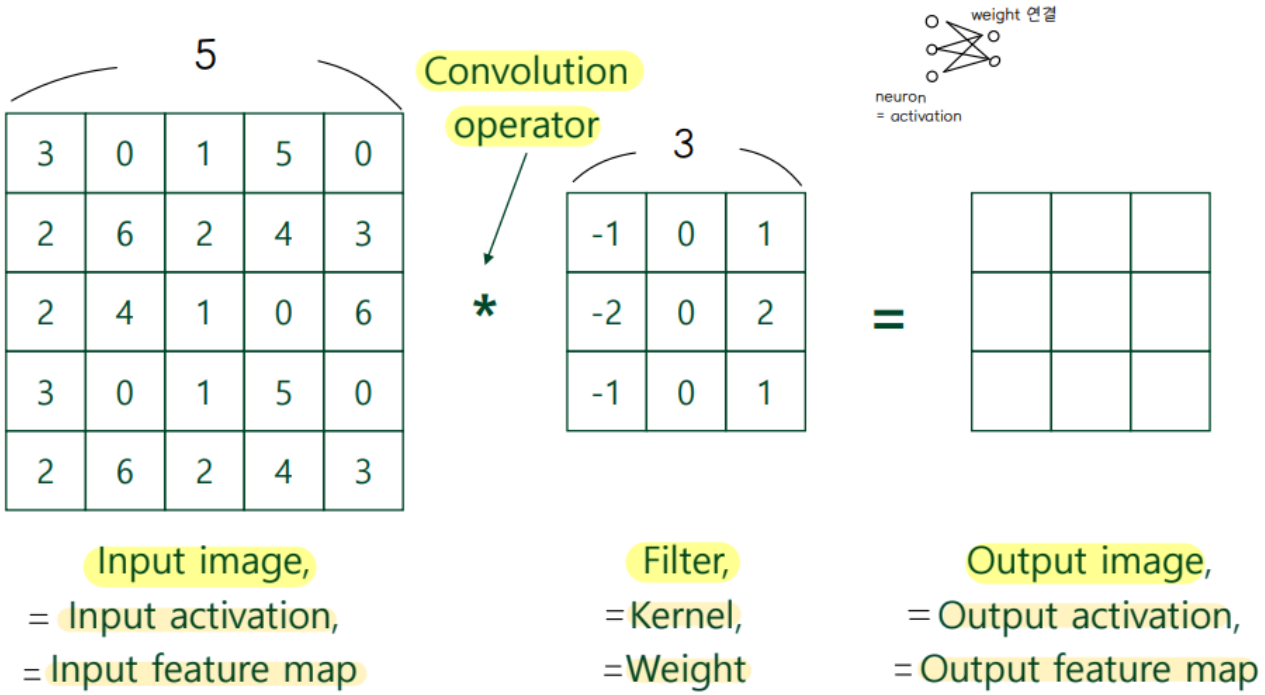
EWHA EURON

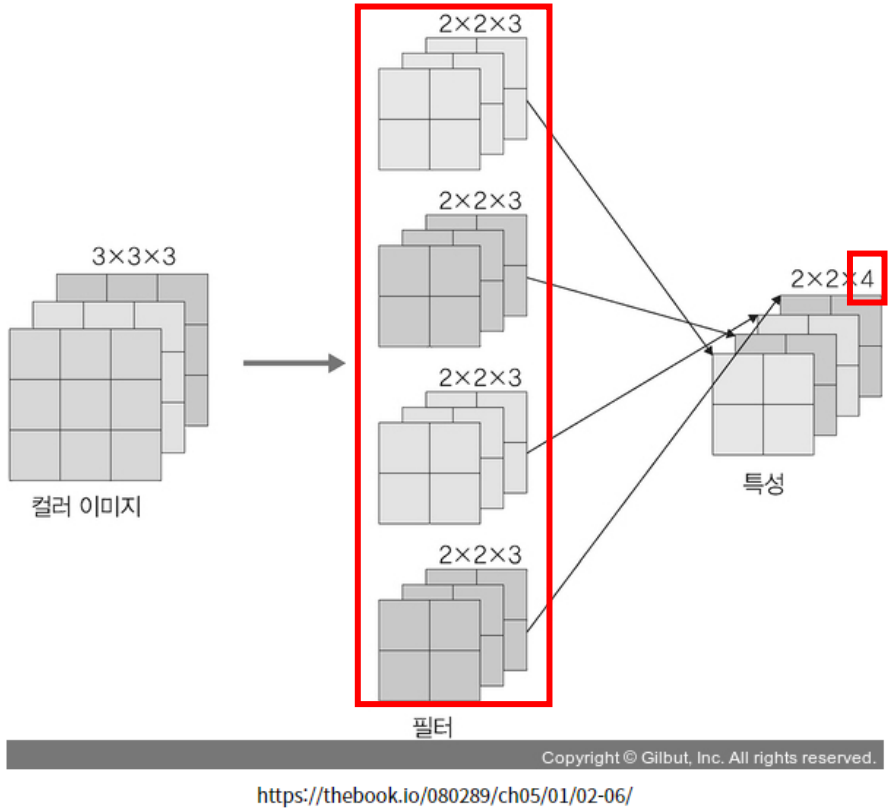
1. LeNet-5



EWHA EURON

채널과 필터의 개념 짚고가기





https://thebook.io/080289/ch05/01/02-06/

😳 필터는 이미지를 훑으면서 특성을 추출하는 역할을 하는 부분으로 학습을 통해 최적 값을 찾아야 할 가중치 부분이고, 채널은 gray –scale, RGB scale 처럼 이미지의 색상을 숫자로 표현하는 과정에서의 층층이 쌓여진 숫자 층이라고 생각하면 되지 않을까

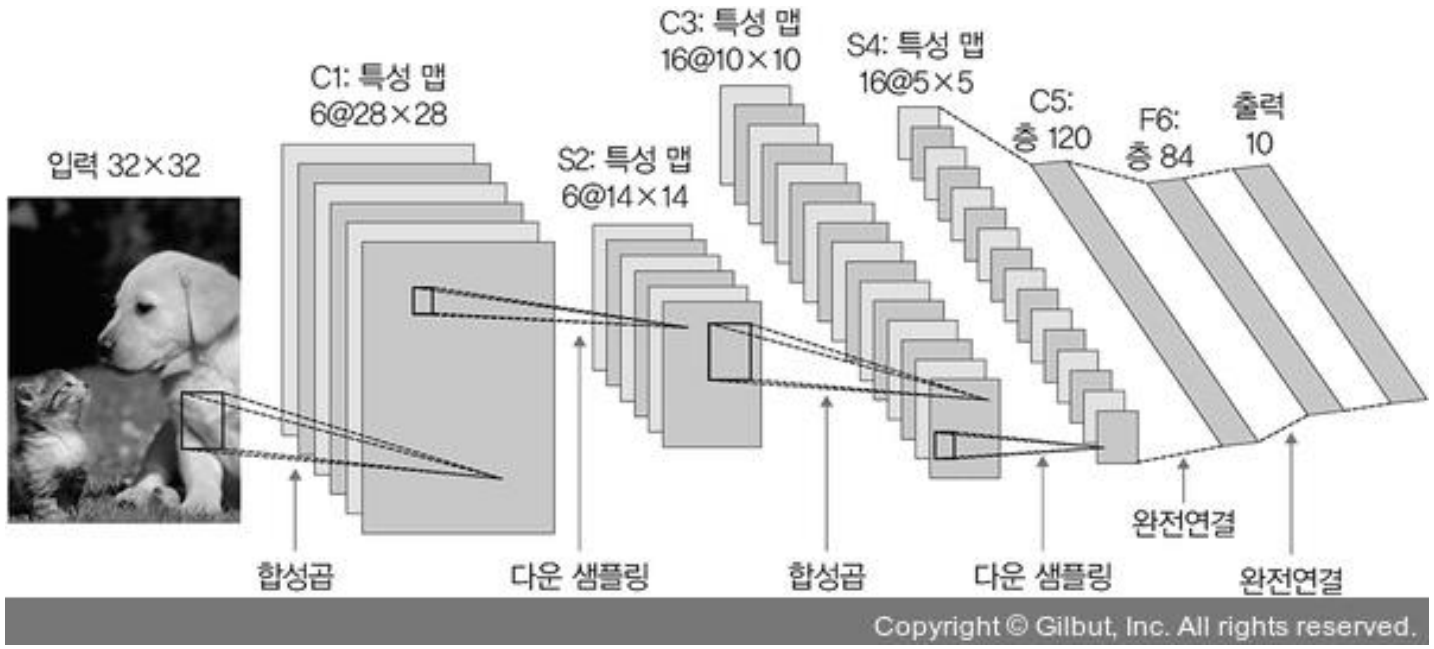
→ 필터 각각은 특성 추출 결과의 채널이 된다. (**4**개의 필터 👉 특성 2x2x**4**)

- 필터의 개수 → output feature map 의 채널 크기

1. LeNet-5

💡 합성곱 신경망이라는 개념을 최초로 개발한 구조로 CNN 의 초석이 된 알고리즘이다.

여러 버전들이 있으나 최종 버전은 LeNet-5 이다.



3개의 합성곱층

2개의 풀링층

1개의 완전 연결층

이미지 특성 추출 단계

분류 수행 단계

1. LeNet-5

Convolution layer 의 하이퍼파라미터

(1) **Filter 의 크기** : NxN → 출력 영상의 크기와 직결된다. Filter 의 형태는 논문마다 다른데, 매뉴얼이 있다기 보단, 학습 데이터에 따라 적절히 선택되어야 한다. 32x32 같은 작은 크기의 입력 영상에는 5x5 필터를 주로 사용하나 큰 이미지를 처리할 땐 11x11 같은 큰 크기의 필터를 사용하기도 한다. 때로는 필터 크기를 레이어마다 다르게 하기도 한다. 크기가 작은 필터를 여러 개 사용하면 좋다. 여러 개를 중첩해 사용함으로써 원하는 특징을 더 찾아내기 쉽고 연산량도 더 적다.

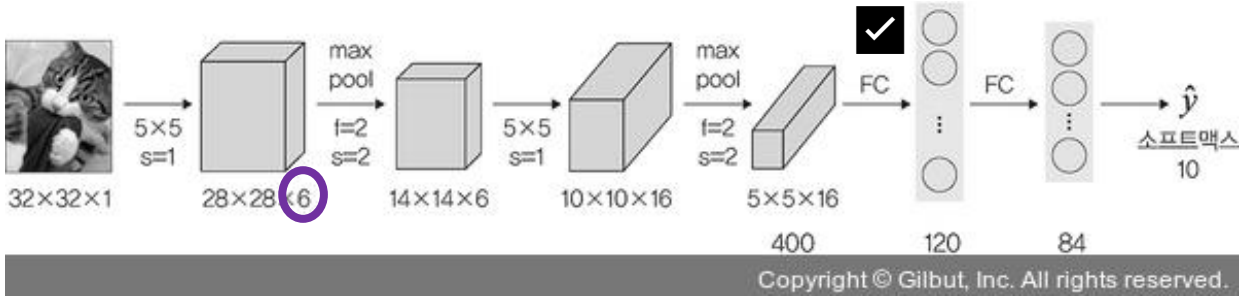
(2) **Filter 의 개수** : Feature map 의 크기 (출력 이미지 크기) 는 Conv 와 Pooling 을 반복하며 점점 작아지기 때문에 이를 방지하기 위해 filter 를 여러 개 써서 채널 수를 늘리는 것이 일반적이다.

(3) **Stride** : 값이 클수록 feature map 크기가 줄어들기 때문에 입력단과 가까운 쪽에서만 1보다 크게 설정한다.

(4) **Zero Padding** : 이미지 크기가 작아지는 것을 피하기 + 경계면의 정보를 살리기 위해 입력의 경계면에 0을 추가

https://sonsnotation.blogspot.com/2020/11/9-2-cnn-best-architecture.html?m=1

1. LeNet-5



▼ 표 6-1 LeNet-5 예제 신경망 상세 Feature map 크기 필터 크기 Stride

계층 유형	특성 맵 (필터개수 / 채널 크기)	크기	커널 크기	스트라이드	활성화 함수
이미지	1	32×32	–	–	–
합성곱층	6 (6개의 필터, 6장의 채널)	28×28	5×5	1	렐루(ReLU)
최대 풀링층	6 (풀링 연산 - 유지)	14×14	2×2	2	–
합성곱층	16	10×10	5×5	1	렐루(ReLU)
최대 풀링층	16	5×5	2×2	2	–
완전연결층	–	120 (1x1), 120 개 특성맵	–	–	렐루(ReLU)
완전연결층	–	84	–	–	렐루(ReLU)
완전연결층	–	2	–	–	소프트맥스(softmax)

LeNet 신경망 구조

- (Conv + Pooling) x 2
- FC layer

Conv2d 계층에서의 출력 크기 구하는 공식

출력 크기 = (W-F+2P)/S+1

- W: 입력 데이터의 크기(input_volume_size)
- F: 커널 크기(kernel_size)
- P: 패딩 크기(padding_size)
- S: 스트라이드(strides)

- input channel = filter channel
- filter 의 개수 = output channel

- (32 − 5 + 0)/1 + 1 = 28
- (28 − 2 + 0)/2 + 1 = 14

☑ 16장의 5 x 5 특성맵을 **120개** 5 x 5 x 16 사이즈의 필터와 컨볼루션 : (5-5+0)/1 + 1 = 1

MaxPool2d 계층에서의 출력 크기 구하는 공식

출력 크기 = IF/F

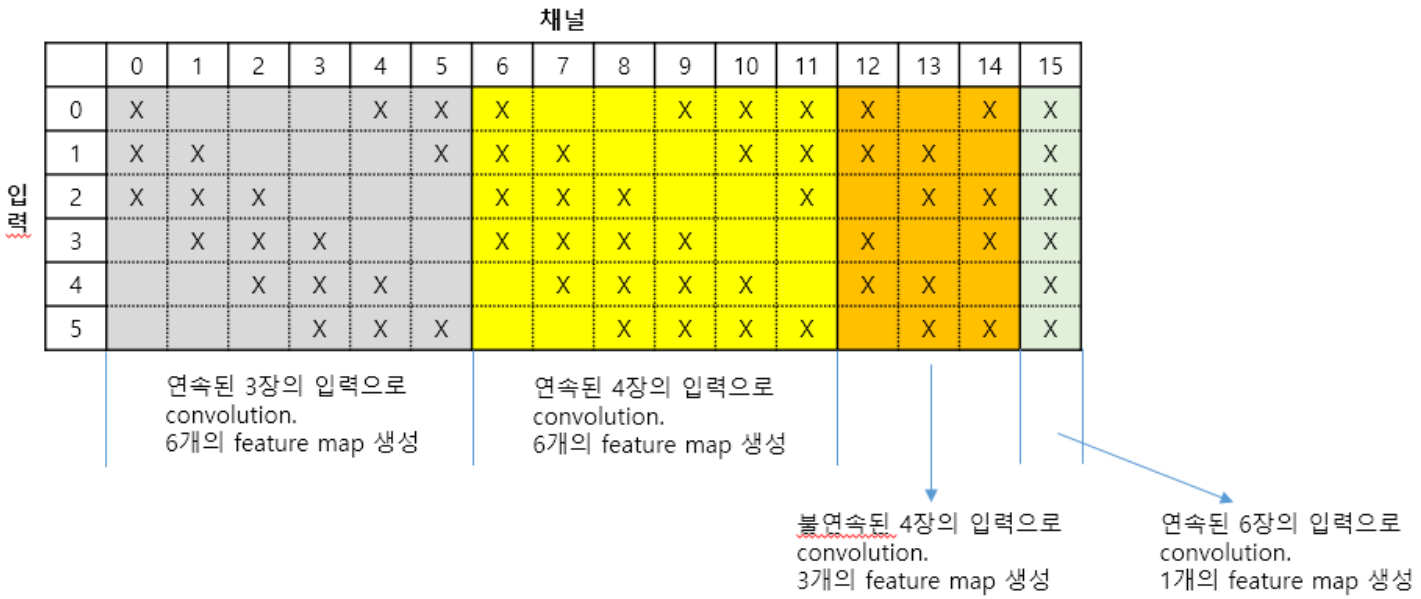
- IF: 입력 필터의 크기(input_filter_size, 또한 바로 앞의 Conv2d의 출력 크기이기도 합니다)
- F: 커널 크기(kernel_size)

▼ 표 6-1 LeNet-5 예제 신경망 상세

계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
이미지	1	32×32	–		
합성곱층	6	28×28	5×5		
최대 풀링층	6	14×14	2×2		
합성곱층	16	10×10	5×5		
최대 풀링층	16	5×5	2×2		
완전연결층	–	120	–		
완전연결층	–	84	–		
완전연결층	–	2	–		

어떻게 나오게 된 값인가

ftmax)



입력에서 연속된 3장, 4장이나 불연속된 4장의 hyper parameter값은 논문에서 작성자가 임의로 선택한 값이라고 하였다.

필터개수 (채널 크기)

😳 결국에 사용자가 지정한 하이퍼 파라미터

1. LeNet-5

① 이미지 데이터 전처리 , 클래스, 데이터 로더 정의 : transforms.Compose

↪ RandomResizedCrop, RandomHorizontalFlip : 이미지 자르기, 수평반전 등을 통해 Augmentation 수행

↪ ToTensor : 이미지 값의 범위나 차원을 자동으로 변경해주는 메서드

↪ Normalize : 사전 훈련된 모델을 사용할 시, 사전 훈련 때 사용한 이미지 데이터셋의 각 채널의 평균과 표준편차에 맞게, 새로이 사용할 이미지를 정규화 시켜주어야 한다. (OpenCV 사용 시 BGR 이미지 채널 순서 주의)

↪ DataLoader 정의 시, batch size 를 지정하여 나눠서 불러올 데이터 크기를 지정한다

② 모델 네트워크 생성 및 객체 생성

③ 옵티마이저와 손실함수 정의 + GPU/CPU 할당

↪ optim.SGD(momentum = 0.9 …) , nn.CrosEntropyLoss()

④ Train 함수 정의 + 훈련

↪ zero_grad( ) 기울기 초기화 – model() 훈련 – criterion(outputs,labels) 오차 계산 – backward() 역전파 – step() 파라미터 갱신

⑤ Test 함수 정의 + 테스트

↪ with torch.no_grad() ☆ 역전파 중 텐서들에 대한 변화도를 계산할 필요가 없음 (test 과정이라)

1. LeNet-5

② 모델 네트워크 생성 및 객체 생성

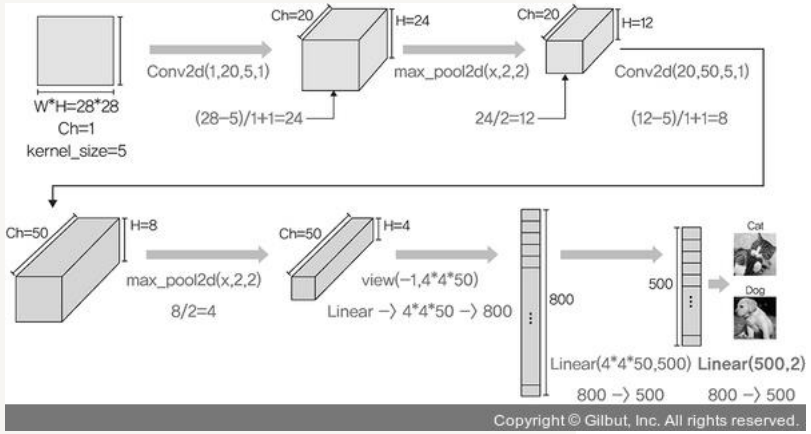
```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.cnn1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=1, padding=0)
# ------ 2D 합성곱층이 적용됩니다. 이때 입력 형태는 (3, 224, 224)가 되며 출력 형태는 (weight-kernel_size+1)/stride에 따라 (16, 220, 220)이 됩니다.

        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
# ------ 최대 풀링이 적용됩니다. 적용 이후 출력 형태는 220/2가 되어 (16, 110, 110)입니다.
        self.cnn2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=0)
#------ 또다시 2D 합성곱층이 적용되며 출력 형태는 (32, 106, 106)입니다.
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
#------ 최대 풀링이 적용되며 출력 형태는 (32, 53, 53)입니다.

        self.fc1 = nn.Linear(32*53*53, 512)
        self.relu5 = nn.ReLU()
        self.fc2 = nn.Linear(512, 2)
        self.output = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.cnn1(x)
        out = self.relu1(out)
        out = self.maxpool1(out)
        out = self.cnn2(out)
        out = self.relu2(out)
        out = self.maxpool2(out)
        out = out.view(out.size(0), -1) #----- 완전연결층에 데이터를 전달하기 위해 데이터 형태를 1차원으로 바꿉니다.
        out = self.fc1(out)
        out = self.fc2(out)
        out = self.output(out)
        return out
```

😳 CNN 네트워크를 직접 설계할 때, 훈련 데이터에 맞추어 입력 형태 및 출력형태를 계산하여 정의

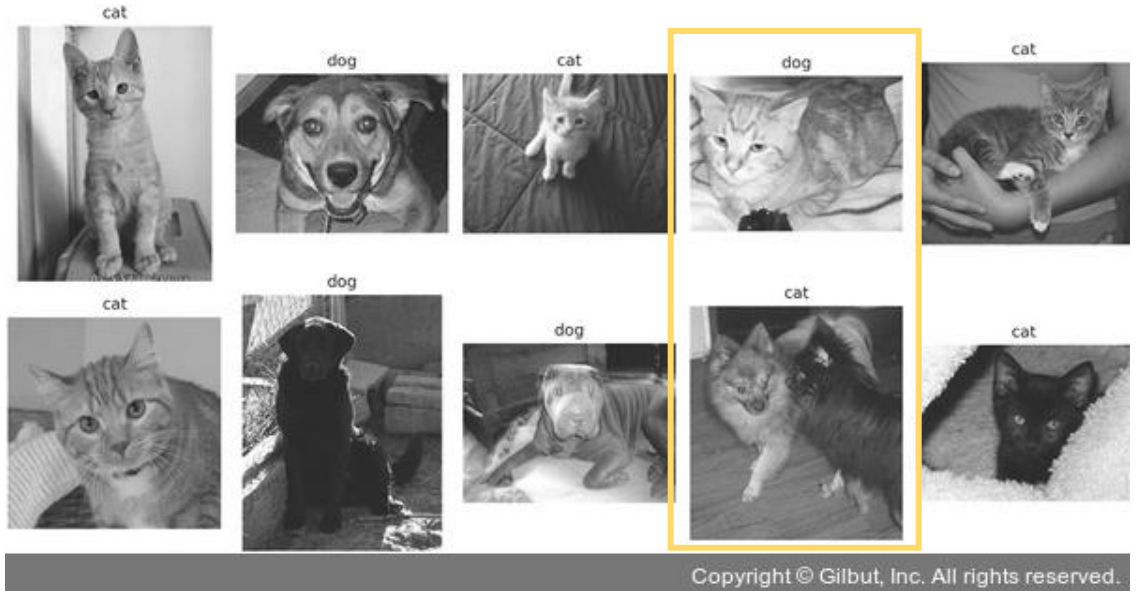


1. LeNet-5

👀 Torchsummary 라이브러리 사용해서 모델 구조 살펴보기

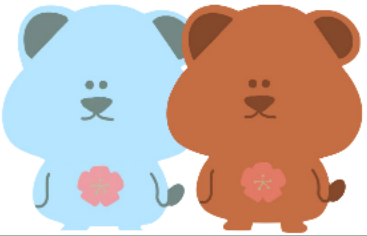
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 220, 220]           1,216
              ReLU-2         [-1, 16, 220, 220]               0
         MaxPool2d-3         [-1, 16, 110, 110]               0
            Conv2d-4         [-1, 32, 106, 106]          12,832
              ReLU-5         [-1, 32, 106, 106]               0
         MaxPool2d-6           [-1, 32, 53, 53]               0
            Linear-7                 [-1, 512]      46,023,168
            Linear-8                   [-1, 2]           1,026
           Softmax-9                   [-1, 2]               0
================================================================
Total params: 46,038,242
Trainable params: 46,038,242
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 19.47
Params size (MB): 175.62
Estimated Total Size (MB): 195.67
----------------------------------------------------------------
```

👀 테스트 결과



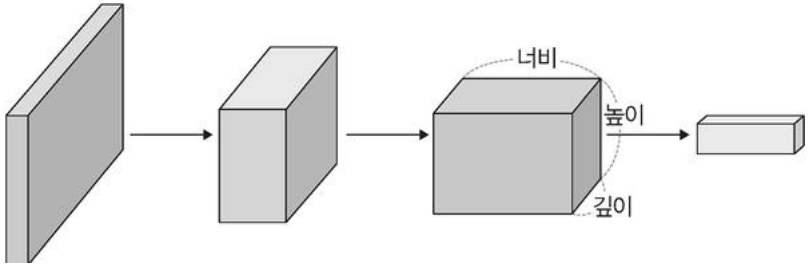
↪ 교재에서는 일부 데이터를 이용하여 모델 학습을 진행했기 때문에 예측력이 그닥 높진 않음 (참고)

2. AlexNet



2. AlexNet

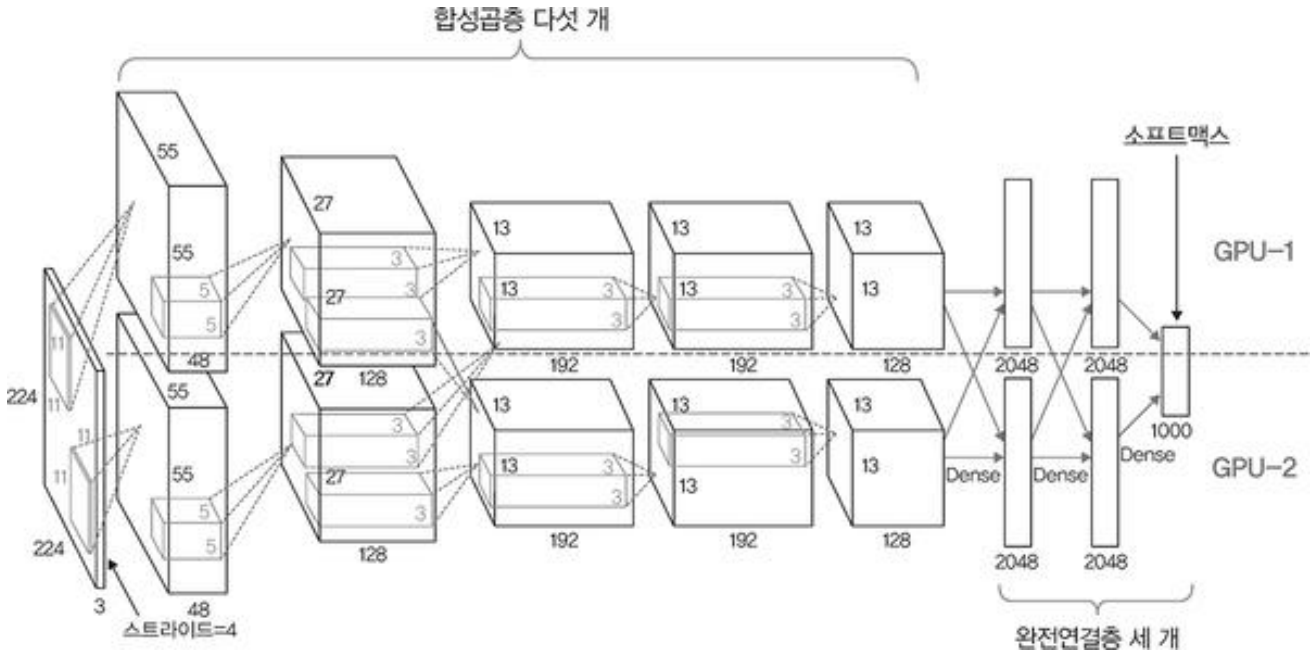
ImageNet 영상 DB 를 기반으로 한 대회에서 우승한 CNN 구조



Remind : CNN 은 (너비, 높이, 깊이) 의 3차원 구조를 갖는다.

색상이 많은 이미지는 RGB 성분 3개를 가지므로 시작은 3이지만,

합성곱을 거치면서 중간 이미지의 깊이가 달라진다.



5개의 합성곱층

3개의 풀링층

3개의 완전 연결층

병렬구조

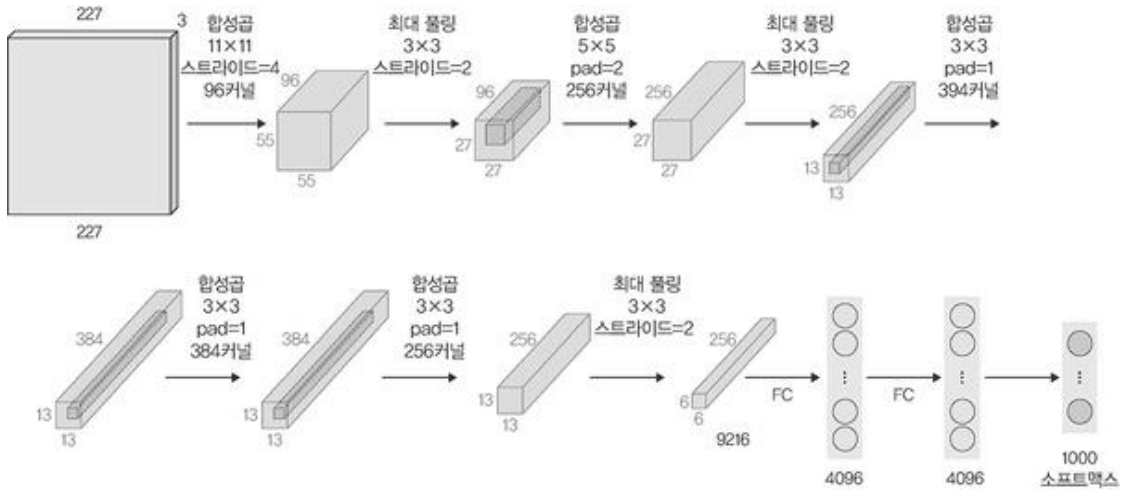
- **GPU-1 :** 컬러와 상관없는 정보 추출
- **GPU-2 :** 컬러와 관련된 정보 추출

2. AlexNet

▼ 표 6-2 AlexNet 구조 상세

계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
이미지	1	227×227	–	–	–
합성곱층	96	55×55	11×11	4	렐루(ReLU)
최대 풀링층	96	27×27	3×3	2	–
합성곱층	256	27×27	5×5	1	렐루(ReLU)
최대 풀링층	256	13×13	3×3	2	–
합성곱층	384	13×13	3×3	1	렐루(ReLU)
합성곱층	384	13×13	3×3	1	렐루(ReLU)
합성곱층	256	13×13	3×3	1	렐루(ReLU)
최대 풀링층	256	6×6	3×3	2	–
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	1000	–	–	소프트맥스(softmax)

- (227 − 11 + 0)/4 + 1 = 55



각 클래스에 해당하는
1000x1 확률 벡터를 출력

2. AlexNet

```python
class AlexNet(nn.Module):
    def __init__(self) -> None:
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256*6*6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 2),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

풀링

$$H_{out} = \left[\frac{H_{in} + 2 \times \text{padding}[0] - \text{kernel\_size}[0]}{\text{stride}[0]} + 1\right]$$

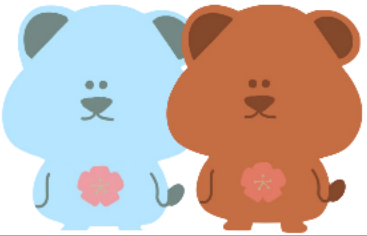
$$W_{out} = \left[\frac{W_{in} + 2 \times \text{padding}[1] - \text{kernel\_size}[1]}{\text{stride}[1]} + 1\right]$$

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 63, 63]	23,296
ReLU-2	[-1, 64, 63, 63]	0
MaxPool2d-3	[-1, 64, 31, 31]	0
Conv2d-4	[-1, 192, 31, 31]	307,392
ReLU-5	[-1, 192, 31, 31]	0
MaxPool2d-6	[-1, 192, 15, 15]	0
Conv2d-7	[-1, 384, 15, 15]	663,936
ReLU-8	[-1, 384, 15, 15]	0
Conv2d-9	[-1, 256, 15, 15]	884,992
ReLU-10	[-1, 256, 15, 15]	0
Conv2d-11	[-1, 256, 15, 15]	590,080
ReLU-12	[-1, 256, 15, 15]	0
MaxPool2d-13	[-1, 256, 7, 7]	0
AdaptiveAvgPool2d-14	[-1, 256, 6, 6]	0
Dropout-15	[-1, 9216]	0
Linear-16	[-1, 4096]	37,752,832
ReLU-17	[-1, 4096]	0
Dropout-18	[-1, 4096]	0
Linear-19	[-1, 512]	2,097,664
ReLU-20	[-1, 512]	0
Linear-21	[-1, 2]	1,026

Total params: 42,321,218
Trainable params: 42,321,218
Non-trainable params: 0

↪ 교재에서는 일부 데이터를 이용하여 모델 학습을 진행했기 때문에 예측력이 그닥 높진 않음 (참고)

03. VGGNet



EWHA EURON

#3.1 모델 개요

- 옥스포드 연구팀에서 합성곱층의 파라미터 수를 줄이고 훈련 시간을 개선하려고 제작.
- **네트워크를 깊게 만드는 것**이 성능에 어떤 영향을 미치는지 확인하고자 나온것이 VGG.
- VGG연구팀은 깊이의 영향만 최대한 확인하고자 합성곱층에서 사용하는 필터/커널의 크기를 **가장 작은 3X3으로 고정**
- 네트워크 계층의 총 개수에 따라 여러 유형의 VGGNet(VGG16, VGG19 등)이 있는데, 책에 소개된 VGG16와 VGG19에 초점을 맞추어 설명 (숫자는 층의 개수)
- ImageNet Challenge 2014 의 classification 분야에서 2등을 차지.
  (VGGNet은 사용하기 쉬운 구조와 좋은 성능 덕분에 그 대회에서 우승을 거둔 조금 더 복잡한 형태의 GoogLeNet보다 더 인기를 얻었다.)

#3.2 Architecture

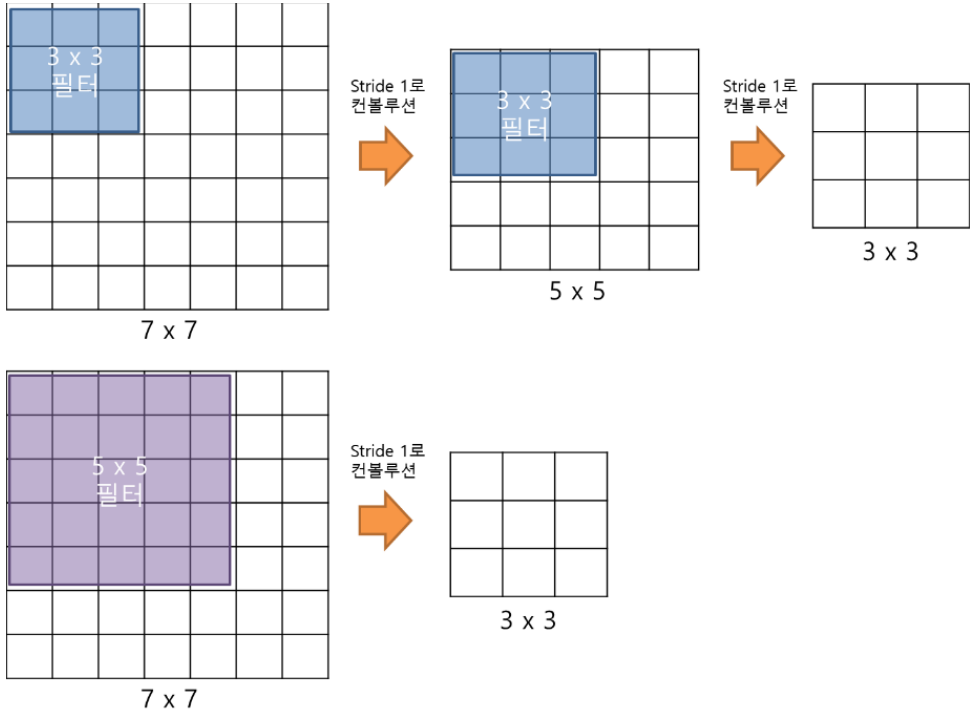
•입력값은 고정된 크기의 224x224 RGB 이미지

•입력 이미지는 3x3 filter로 고정된 필터가 적용된 ConvNet에 전달되고, 또한 비선형성을 위해 1x1 convolutional filters도 적용.

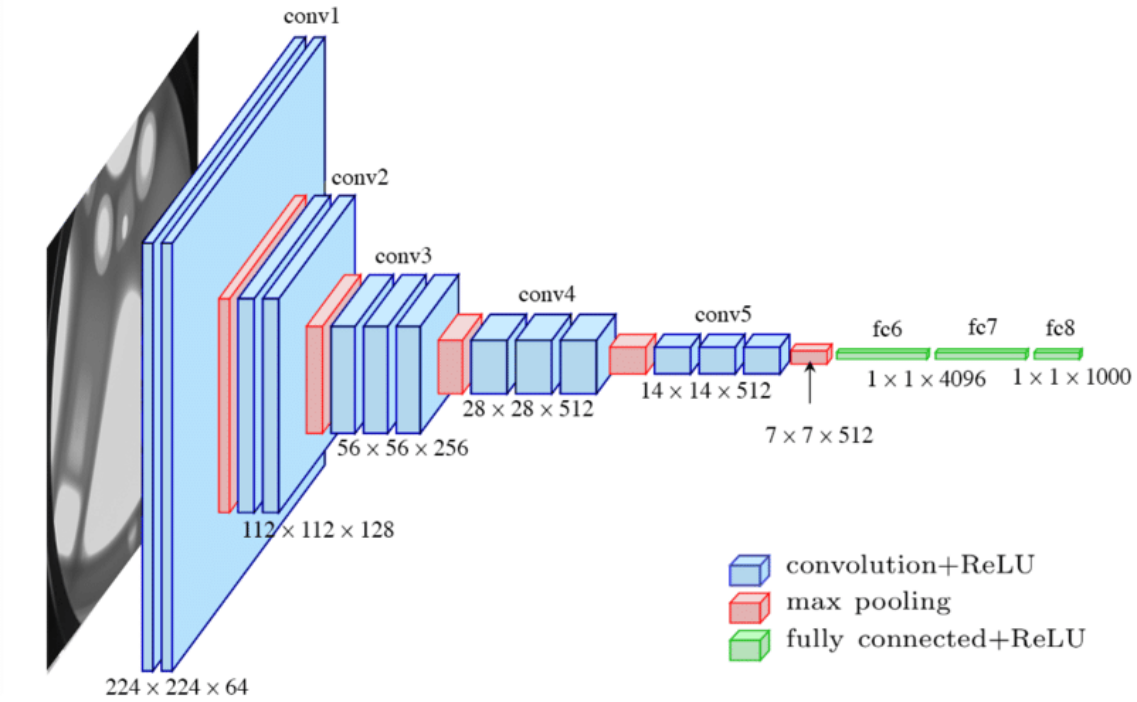
•stride=10이 적용되었고 공간 해상도를 보존하기 위해 padding을 적용합니다. 일부 conv layer에는 max-pooling(size=2x2, stride=2) layer를 적용

#3.2 Architecture

- **3x3 filter로 필터를 고정한 이유?**
  - 먼저 필터의 사이즈를 작게 함으로서 얻는 장점 :
    3 x 3 필터로 두 차례 컨볼루션을 하는 것과 5 x 5 필터로 한 번 컨볼루션을 하는 것이 결과적으로 동일한 사이즈의 특성맵을 산출한다는 것(그림 참고).
  - **3 x 3 필터로 세 차례 컨볼루션 하는 것은 7 x 7 필터로 한 번 컨볼루션 하는 것과 대응!!**
  - 또한 더 많은 relu함수를 사용가능하기도 함.



#3.2 Architecture



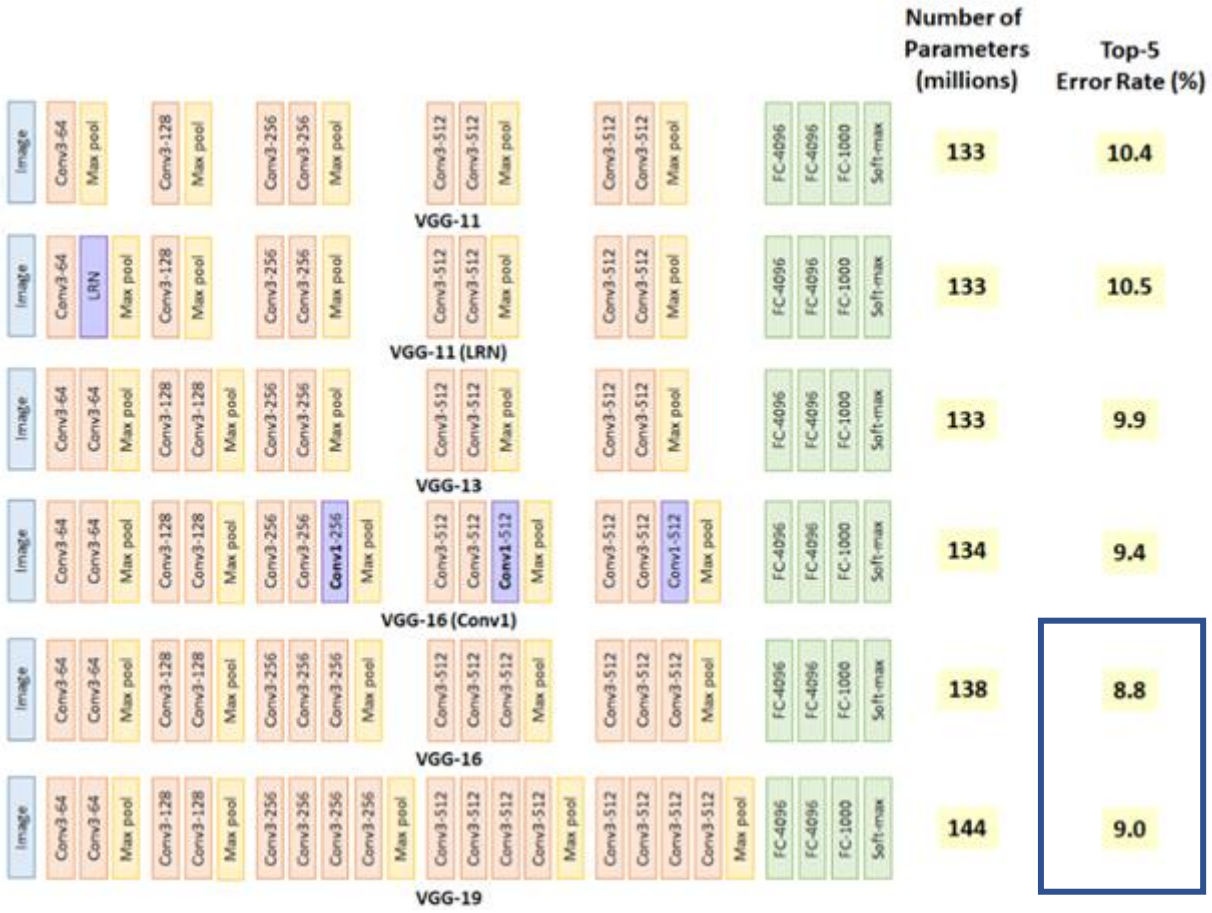
- 3개의 Fully-Connected(FC) layer가 있고, 첫 번째와 두 번째 FC는 4096 channel, 세 번째 FC는 1000 channels를 갖고 있는 soft-max layer.
- 모든 hidden layer에는 활성화 함수로 **ReLU**를 이용했으며, AlexNet에 적용된 LRN(Local Response Normalization)는 VGG 모델의 성능에 영향이 없기 때문에 적용하지 않았음
- 책에서는 11이 나오지만, 16과 19를 위주로 설명할 것임

#3.2 Architecture

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- VGGNet의 원본 논문에서는 네트워크의 깊이를 깊게 조절하는 것이 CNN의 성능에 어떤 영향을 주는지 확인하고자 했고, 총 6개의 구조(A, A-LRN, B, C, D, E)를 만들어 깊이에 따른 성능을 비교.
- VGGNet의 A~E까지 각각의 다른 모델이 아니라 학습의 단계부터 알 수 있듯, 업그레이드된 모델임.
- C와 D(이게 VGG16)는 마지막 maxpool에서만 다르고 동일

#3.2 Architecture



- VGG16, VGG19의 error rate가 적음을 확인할 수 있음

#3.3 특징 및 장단점

•학습 과정은 입력 이미지 crop 방법을 제외하고 AlexNet과 동일하게 진행
•논문에서는 AlexNet과 비교하여 이 모델이 깊고 더 많은 parameter를 갖고 있음에도 불구하고,
3x3 filter size의 여러 겹 이용한 것과 특정 layers에서 **pre-initialisation** 덕분에 수렴하기 위한 **적은 epoch를 요구한다고 주장**

- **pre-initialisation**란? bad initialisation은 gradine의 불안정함 때문에 학습을 지연. 이를 해결하기 위해 pre-initialisation을 이용. 가장 얇은 구조인 A를 학습시킨 이후에 학습된 첫 번째, 네 번째 Conv layer와 3개의 FC layer의 가중치를 이용하여 다른 깊은 모델을 학습시킴. 미리 가중치가 설정되어 수렴하기 까지의 적은 epoch가 필요함. 그리고 가중치 초기화 값은 평균 0 분산 0.01인 정규 분포에서 무작위로 추출.

#3.4 분류실험

#1 Single Scale evaluation : test set의 size가 고정된 단일 규모 평가

(1) AlexNet에서 이용되었던 LRN(local response normalisation)이 효과가 없었음

(2) 깊이가 깊어지 수록 error가 감소한다는 것을 관찰

(3) 다양한 scale[256~512]로 resize한 것이 고정된 scale의 training image보다 성능이 좋았음

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

#3.4 분류실험

#2 Multi-scale evaluation : test set의 다양한 scale에 대한 실험
test 이미지를 다양한 scale로 resize했을 때, 단일 size보다 더 나은 성능

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

#3.4 분류실험

#3 Multi-crop evalutation

test 이미지를 다양하게 crop을 해주어 더 나은 성능을 얻음

*crop란 ? 사진의 부분을 자르기

ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

#3.4 분류실험

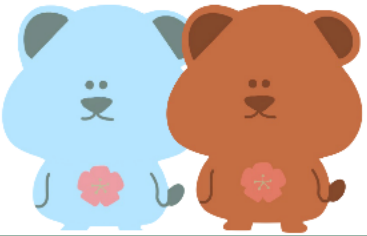
#4 ConvNet fusion

앙상블. 모델 7개를 앙상블한 ILSVRC 제출물은 test set top-5 error가 7.5% 나왔고, 추후에 모델 2개를 앙상블하여 test set top-5 error를 6.8% 까지 낮추었음.

Table 6: **Multiple ConvNet fusion results.**

Combined ConvNet models	Error		
	top-1 val	top-5 val	top-5 test
ILSVRC submission			
(D/256/224,256,288), (D/384/352,384,416), (D/[256;512]/256,384,512) (C/256/224,256,288), (C/384/352,384,416) (E/256/224,256,288), (E/384/352,384,416)	24.7	7.5	7.3
post-submission			
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), dense eval.	24.0	7.1	7.0
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop	23.9	7.2	-
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval.	**23.7**	**6.8**	**6.8**

04. GoogLeNet



EWHA EURON

#4.1 모델 개요 및 장점

- 주어진 하드웨어 자원을 최대한 효율적으로 이용하면서 학습능력은 극대화 할 수 있는 **깊고 넓은 신경망.**
- GoogLeNet은 AlexNet보다 파라미터가 **12배나 더 적음에도 불구하고 훨씬 정확**
- **인셉션 블록이라는 개념을 도입하여, 인셉션 네트워크락**

→ **그러나 앞에서 언급한 것 처럼 복잡하기 때문에 VGG보다 덜 활용되고 있음.**

•딥러닝 네트워크의 성능을 높이는 가장 간단한 방법은 네트워크를 깊게 하거나(depth 증가) 많은 채널수를 사용(width 증가)하는 등 크게 설계하는 것. 하지만 단순히 네트워크가 커지면 다음 과 같은 단점이 있음

- 학습 데이터에 과하게 학습되는 Overfitting이 발생하기 쉽다.
- 파라미터 수가 증가하여 많은 연산자원을 필요로 하게 된다.

GoogLeNet에 논문 저자들은 위 두가지 문제를 해결하기 위해선 dense한 Fully-connected 구조 대신 sparsely connected 한 구조로 만들어서 최적의 네트워크를 만들수 있다고 주장! 그렇게 만든 Clustering하여 submatrix를 만들어 좋은 성능 도출

#4.1 모델 개요 및 장점

- 요약하자면!
  1. 딥러닝 네트워크 성능을 높이기 위해서는 네트워크의 크기, 즉 Depth(layer 수)와 width(channel 수)를 늘려야한다.
  2. 기존의 **Dense한 구조**(weight들의 대부분이 0이 아닌 값을 갖는 형태)에서 크기를 키우면 **overfitting, 연산량 증가 등 문제가 발생**한다.
  3. GoogLeNet에서는 깊은 망을 설계하되 파라미터 수를 줄이기 위해 **dense한 구조 대신 sparse(weight의 대부분이 0인 형태)하면서 크기가 큰 구조로** 만든다.
  4. 그런데 Sparse한 구조는 컴퓨터 연산구조상 효율이 떨어진다.
  5. **Sparse한 구조들 중 상관도가 높은 것들을 clustering**하여 유사 dense한 형태를 만든다(sparse한 구조들을 묶어 desne한 형태로 만들기)

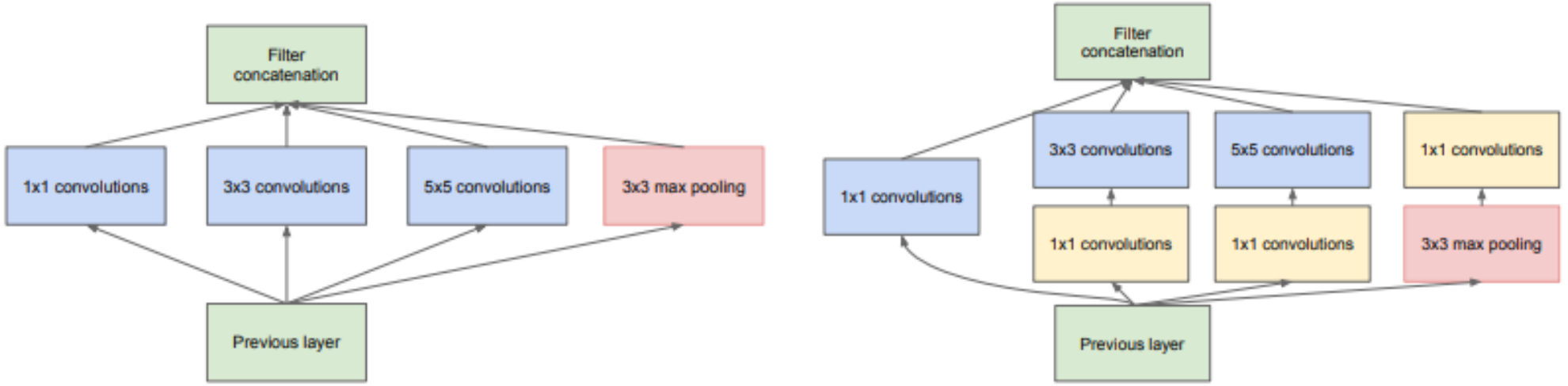
#4.2 특징

특징1 여러 스케일의 Conv layer의 병렬 사용 -> CNN 계산 용량 최적화

- **Inception** module은 1x1, 3x3, 5x5 세 개의 Conv layer와 1개의 Max-pooling을 사용합니다. 여러 스케일의 Convolution 연산을 활용해 다양한 스케일에서 효율적으로 **특징을 뽑아낸 뒤 ReLU함수를 사용합니다**. 각각의 결과를 연결해(concat) 하나의 output을 생성합니다.
- 인셉션 의미 ? GoogLeNet의 코드네임인 Inception이란 이름은 **Network in Network(NIN)라는 논문에서 유래하였으며,** 더 정확하게는 인셉션 영화의 대사인 "we need to go deeper"에서 착안하였다.

#4.2 특징

특징1 여러 스케일의 Conv layer의 병렬 사용 -> CNN 계산 용량 최적화
왼쪽이 기본적인 인셉션 모델 -> 연산 감소
오른쪽은 1*1 conv를 추가한 모델 -> 차원축소



(a) Inception module, naïve version (b) Inception module with dimension reductions

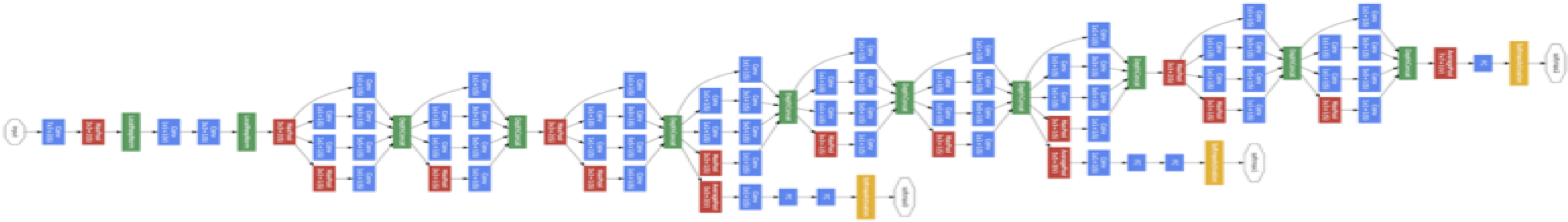
Figure 2: Inception module

#4.2 특징

특징2 1*1 conv layer 사용

- 왼쪽 모델에서 1*1 conv layer를 추가한 것임
- **추가된 1x1 Conv layer는 차원을 축소하여 연산량을 줄여주는 역할**
- 네트워크에서 더 뒤쪽 layer로 갈 수록 추상적인 특징을 가지고 있으며, 공간 집중도가 낮아집니다. 따라서 뒤쪽 layer로 갈수록 3x3, 5x5 Conv layer이 증가해야 함을 의미
- 큰 필터크기를 갖는 Conv layer는 많은 연산량을 필요로 하기 때문에, Inception module (b)구조에서는 **1x1 Conv layer를 사용해 먼저 차원의 수를 줄인 뒤 3x3, 5x5 Conv layer 연산을 수행하는 방식으로 연산량을 조절**
(즉, 병목현상을 제거하기 위한 **차원 축소**)

#4.3 Architecture

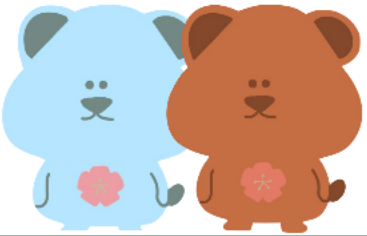


- GoogLeNet은 메모리 효율을 고려하여 초반에는 일반적인 Conv layer를 사용하고, 뒤쪽에는 inception module을 사용
- 이러한 구조의 장점은 다음 layer의 input-filter 수를 조절하여 연산량 증가 없이 유닛 수를 증가시킬 수 있고, 여러 스케일에서의 feature를 동시에 뽑을 수 있음
- 연산효율의 증가는 네트워크의 Depth(layer 수)와 Width(각 layer의 채널 수)를 증가시킬 수 있게 만듦

Convolution
Pooling
Softmax
Other

* GoogLeNet-V2와 V3도 있는데, V1에서 어떤 점이 개선되었는지 뒤에서 설명 예정

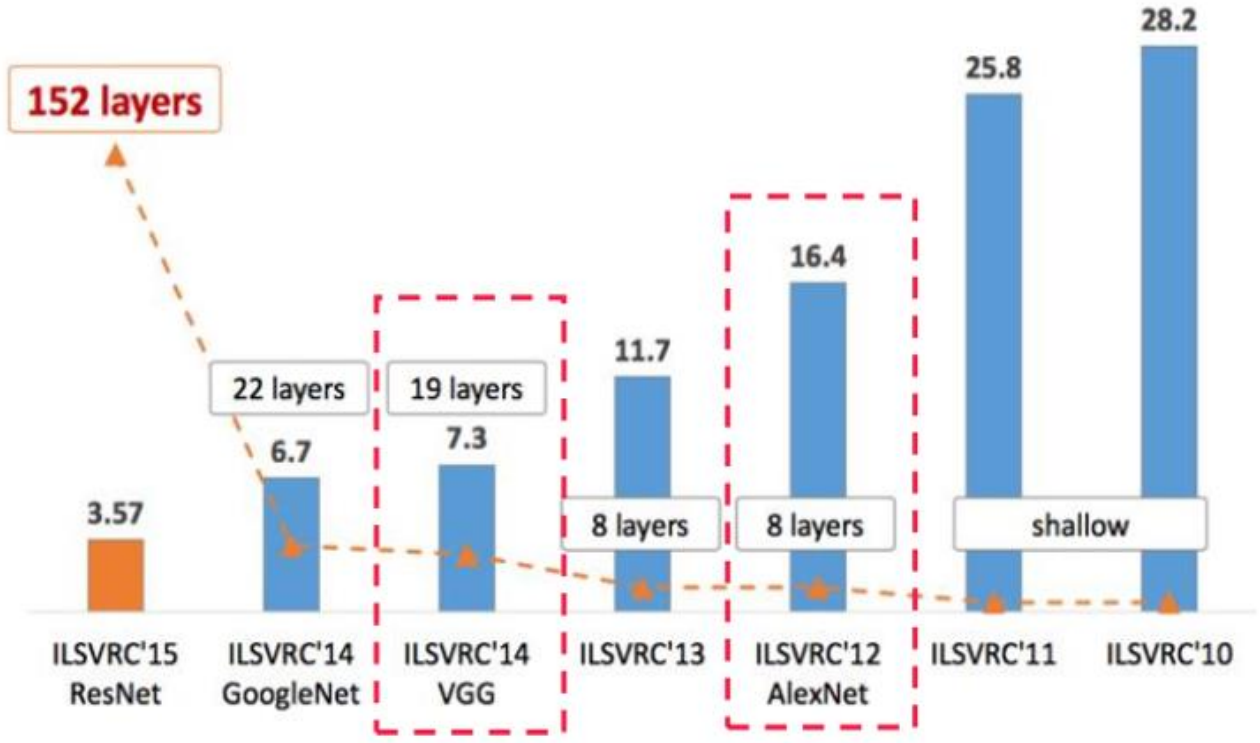
5. ResNet



EWHA EURON

#5.1 ResNet

- 마이크로소프트에서 “Deep Residual Learning for Image Recognition” 논문을 통해 발표한 이미지 모델
- 2015년 ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 우승 모델



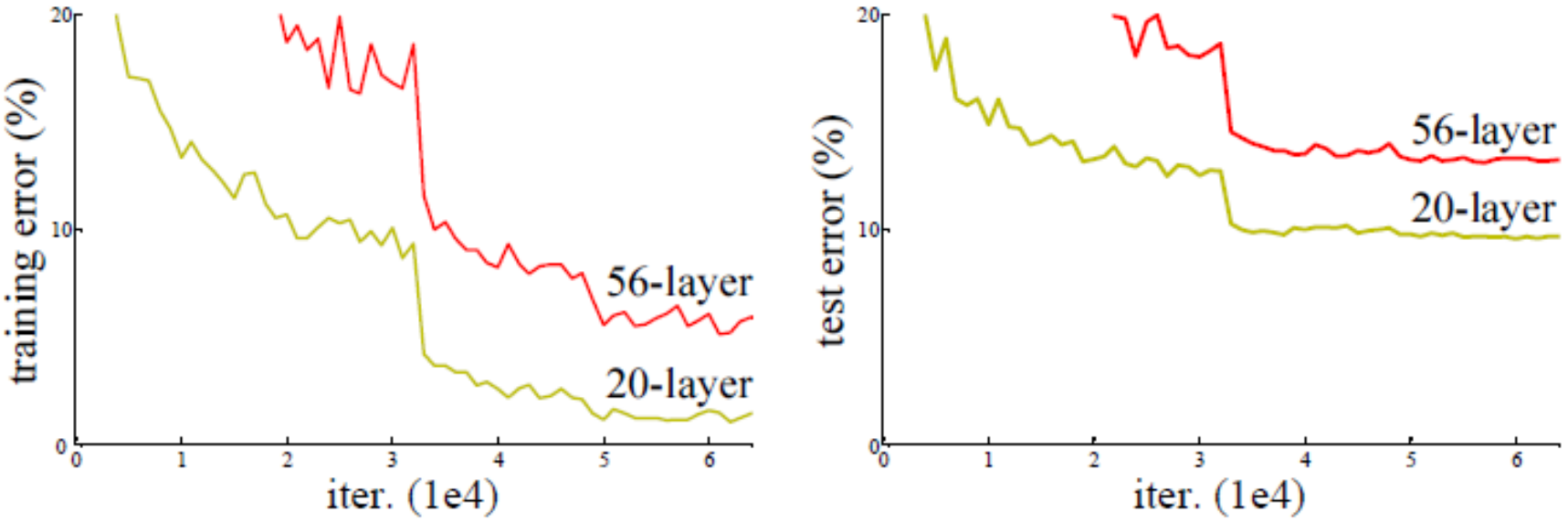
역대 ILSVRC 우승 알고리즘의 에러율

파란막대 위에 있는 숫자가 에러율이고 파란막대 아래에 있는 것이 ILSVRC 계최연도이다. (참고로 사람의 평균 에러율은 5이다.)

14년 우승 모델인 GoogLeNet 22층으로 이루어져 있는 것에 비해 ResNet은 152층으로 7배 넘는 레이어를 가지고 있음.

네트워크가 깊을수록 성능이 좋을까??

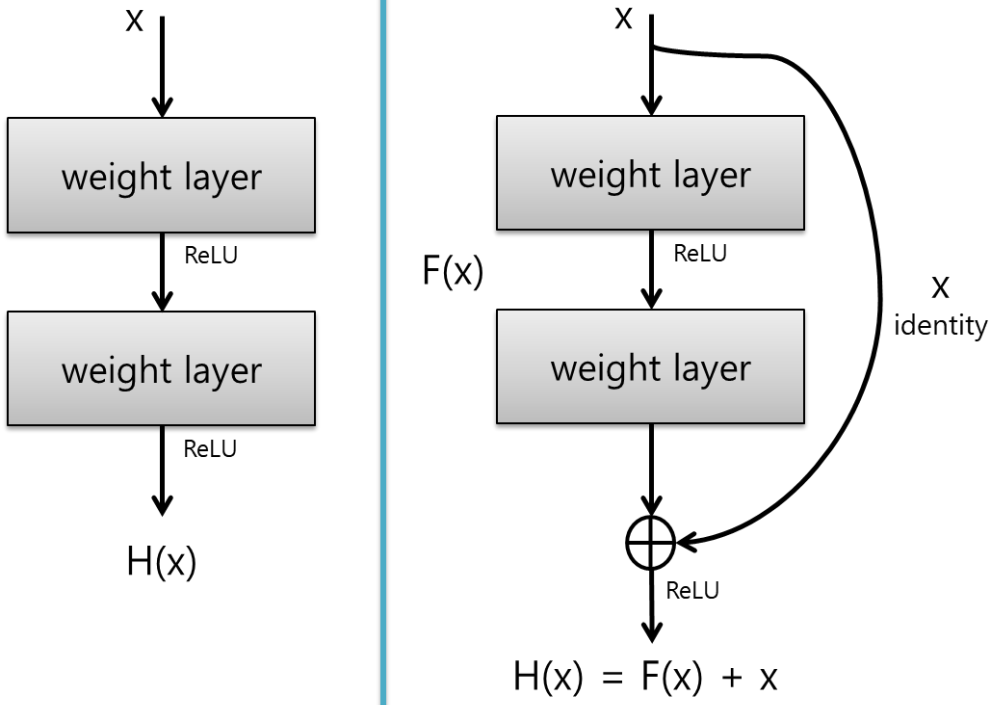
#5.2 Residual block



논문의 저자들이 20층과 56층의 네트워크를 만들어 실험해본 결과, 오히려 깊은 56층 네트워크의 성능이 더 나쁨을 확인
degradation problem : 네트워크가 깊어질수록 train/test accuracy 모두 감소하는 문제
-> 레이어가 깊어질수록 최적화가 힘들어지기 때문
** train accuracy가 증가할 때, test accuracy는 감소하는 오버피팅과는 다른 문제!

신경망이 깊어질수록 좋은 성능을 낼 수 있도록 residual block을 제안함

#5.2 Residual block



기존 방식

Residual block

Plain layer	Residual block
이전 층에서 학습한 정보를 보존하지 않고 변형시켜 새롭게 생성. 층이 깊어질수록 한 번에 학습해야 할 매핑 증가	아웃풋에서 이전 층에서 학습한 정보를 연결함으로써 추가적으로 학습해야 할 정보만 매핑(최적화) => skip connection

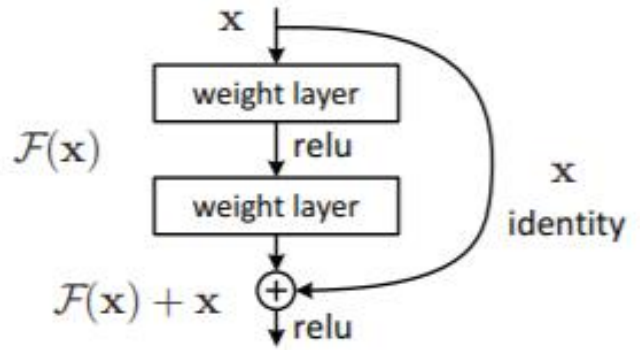


Figure 2. Residual learning: a building block.

- 왼쪽의 plain layer와 비교해보면 오른쪽은 입력값 x를 출력값에 더하도록 지름길을 만든 형태
- identity mapping으로 shortcut connection이 되게 하면서 하나 이상의 레이어를 skip하게 됨
- 네트워크가 깊어질수록 x는 출력값 H(x)에 근접하게 되므로 추가 학습량 F(x)는 0에 가까워짐

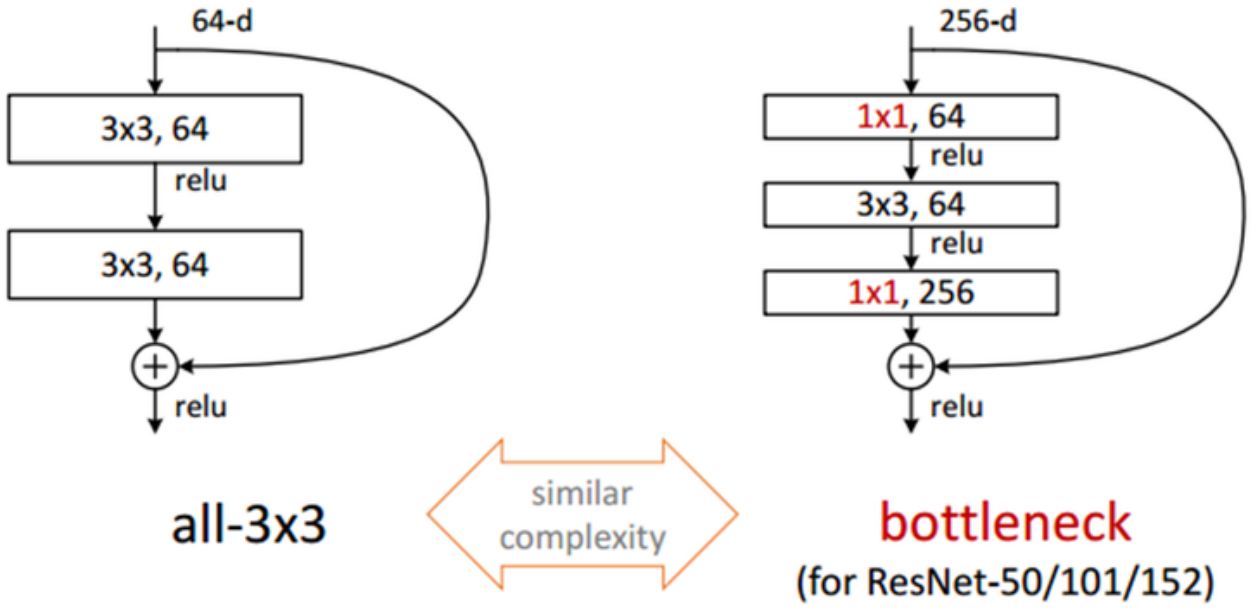
- H(x) −x를 잔차 즉 residual이라고 함. (ResNet인 이유) 다시 말해, 잔차를 최소화하는 방향으로 학습이 진행된다!
- 입력에서 출력으로 연결되는 숏컷만 연결했기 때문에 파라미터 수 영향이 없으며, 복잡한 곱셈 연산도 필요 없음(연산량 증가x)

https://velog.io/@lighthouse97/ResNet%EC%9D%98-%EC%9D%B4%ED%95%B4#1-%EB%8F%99%EA%B8%B0motivation

EWHA EURON

#5.3 Bottleneck

계층이 깊어질수록 파라미터는 무한으로 커지기 때문에 bottleneck block 제안



- 3층을 쌓을 때, (1 x 1) -> (3 x 3) -> (1 x 1) 순서로 병목 모양으로 쌓는 방식으로 차원 개수를 조절하면서 파라미터 수를 줄일 수 있음.
Ex) 차원을 256에서 64로 줄인 후 conv 연산을 한 후, 기존과 같은 사이즈를 만들기 위해 차원을 증가시키는 방식
- 연산량 절감 효과가 있으며, 기존 구조와 비슷한 복잡성을 가지면서 인풋과 아웃풋 차원을 줄일 수 있기 때문에 사용

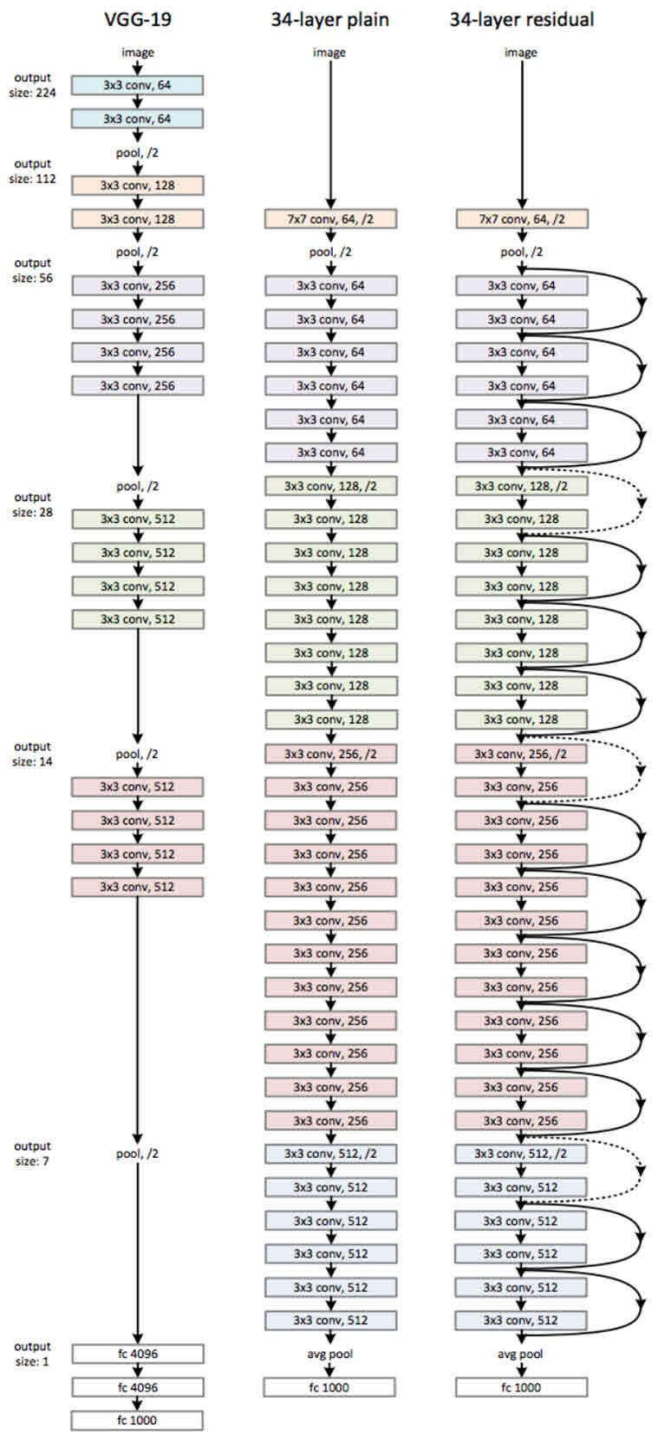
#5.4 ResNet Architecture

VGG19 구조 뼈대로 하며, 합성곱층을 추가하여 깊게 만든 후 숏컷들을 추가

1) Plain NW
- 동일한 피처맵 사이즈를 갖게 하기 위해 동일한 수의 필터를 가지게 함.
- 필터맵 사이즈가 절반이 되면 필터 수는 2배가 됨.
- conv : 3*3 filter, stride 2로 downsampling, global avg pooling layer 통과(연산량 줄이기 위해), 최종 레이어 34개로 VGG에 비해 필터 수와 복잡도가 작고, 연산량도 VGG의 18%에 불과함.

2) Residual NW
- plain net 기본으로 하나, shortcut conn 개념 도입.
- identity shortcut은 인풋과 아웃풋 차원을 같게 만들어야 하므로 차원이 증가하면 두 가지 방법 사용
  1) 차원을 늘리기 위해 0을 넣어 패딩(추가 파라미터 없음)
  2) linear projection (같은 차원으로 만들어주기 위해…)

[Implementation]
image resized 224 * 224/Batch normalization BN 사용/Initialize Weights/SGD, mini batch 256/Learning rate 0.1/Iteration 60 * 10^4/weight decay 0.0001, momentum 0.9/No dropout



#5.5 Experiments

Table 4. Comparisons with state-of-the-art methods on CIFAR-10 and CIFAR-100 using "*moderate data augmentation*" (flip/translation), except for ELU [12] with no augmentation. Better results of [13,14] have been reported using stronger data augmentation and ensembling. For the ResNets we also report the number of parameters. Our results are the median of 5 runs with mean±std in the brackets. All ResNets results are obtained with a mini-batch size of 128 except † with a mini-batch size of 64 (code available at https://github.com/KaimingHe/resnet-1k-layers).

CIFAR-10	error (%)
NIN [15]	8.81
DSN [16]	8.22
FitNet [17]	8.39
Highway [7]	7.72
All-CNN [14]	7.25
ELU [12]	6.55
FitResNet, LSUV [18]	5.84
ResNet-110 [1] (1.7M)	6.61
ResNet-1202 [1] (19.4M)	7.93
ResNet-164 [ours] (1.7M)	5.46
ResNet-1001 [ours] (10.2M)	4.92 (4.89±0.14)
ResNet-1001 [ours] (10.2M)†	**4.62** (4.69±0.20)

CIFAR-100	error (%)
NIN [15]	35.68
DSN [16]	34.57
FitNet [17]	35.04
Highway [7]	32.39
All-CNN [14]	33.71
ELU [12]	24.28
FitNet, LSUV [18]	27.66
ResNet-164 [1] (1.7M)	25.16
ResNet-1001 [1] (10.2M)	27.82
ResNet-164 [ours] (1.7M)	24.33
ResNet-1001 [ours] (10.2M)	**22.71** (22.68±0.22)

많은 성능 평가 내용 중 중요한 일부만 보자면!

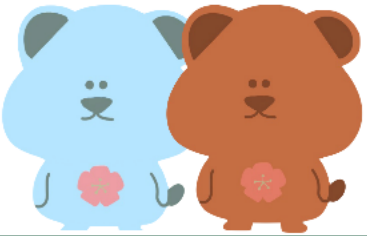
1) 18-layer ResNet 보다 34-layer ResNet이 성능 좋음

2) Layer의 깊이가 증가할 때도 degradation problem 잘 조절함

3) Cifar-10/100 데이터셋에서 다른 최신 (state-of-the-art) 모델들을 비교한 결과, 가장 좋은 성능 보임

4) 사람의 정확도인 95%를 넘은 첫 모델

6. Inception-v2,3

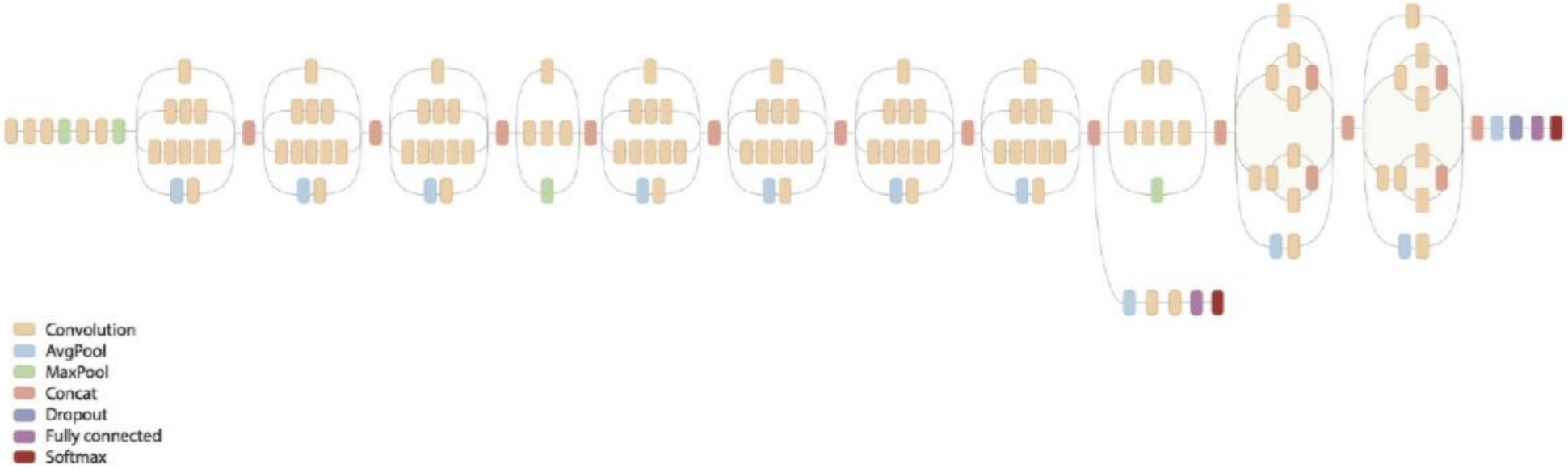


EWHA EURON

#6.1 Inception-v2,3

- GoogLeNet의 아버지이자 Batch Normalization을 고안한 Christian Szegedy가 "Rethinking the Inception Architecture for Computer Vision" 논문을 통해 발표한 이미지 모델

- 모델 크기를 증가시키면 연산량이 증가하는데 모바일이나 제한된 메모리를 사용해야하는 환경에서는 단점임을 지적
- 저자는 convolution 분해를 활용해서 연산량이 최소화 되는 방향으로 모델의 크기를 키우는데 집중



#6.2.1 합성곱 분해

- GoogLeNet은 1x1 convolution로 차원을 축소시키고 3x3 convolution을 여러 개 활용하여 파라미터 수를 감소시켰는데, 이처럼 합성곱을 분해하는 것은 파라미터 수를 감소시키고 이는 연산량을 줄여 빠른 학습이 가능하게 함
- VGG에서 5x5/7x7 convolution을 3x3 convolution으로 분해하면 연산량과 파라미터가 감소함을 확인

- 3x3 convolution을 2번 사용하여 5x5 convolution 분해 시
activation 함수를 1) linear와 relu 한번씩 사용하는 것과 2) 둘 다 relu를 사용하는 것 중 후자에서 정확도가 높았음
- 추가적으로 배치 정규화(Batch normalization)을 사용하면 정확도가 높아짐
⇒ Inception module에서 5x5 convolution 부분을 두 개의 3x3 convolution으로 대체함

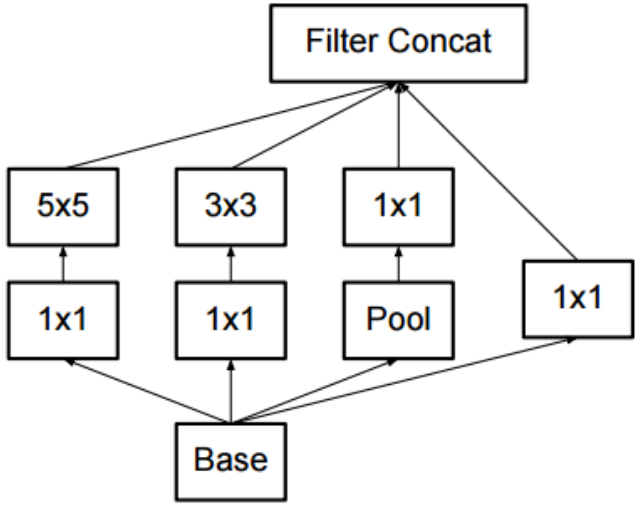


Figure 4. Original Inception module as described in [20].

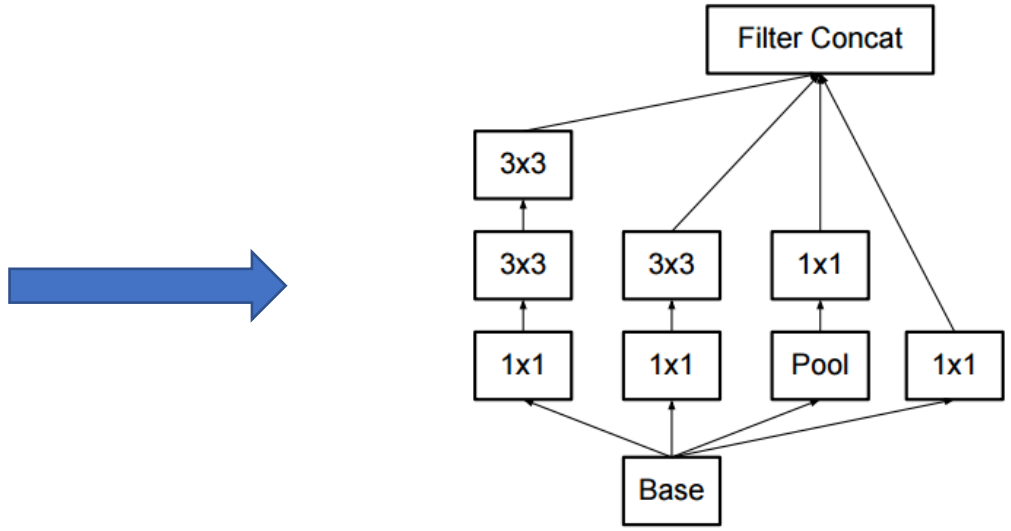


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

#6.2.2 비대칭 합성곱 분해

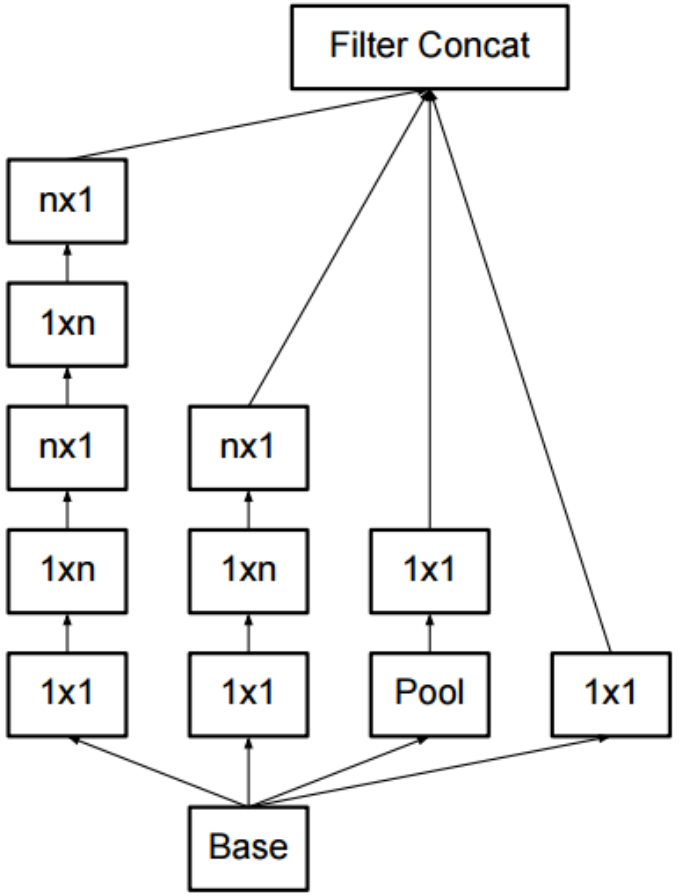


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ fo the 17×17 grid. (The filter sizes are picked using principle 3)

그렇다면 3x3 convolution를 더 작은 convolution로 분해가능할까??

- 실험 결과, 2x2 convolution보다 nx1 비대칭 convolution 분해가 더 성능이 좋았음
  - 2x2 conv로 분해하면 11%의 연산량 감소 효과가 있는 반면, 3x3을 1x3, 3x1로 분해하면 33%의 연산량 절감 가능

  - feature map 사이즈가 12~20 사이일 때 효과가 좋았음
  - Inception v2는 feature map 사이즈가 17이 되는 구간에서 위와 같은 inception module을 사용

#6.3 Auxiliary Classifiers & Efficient Grid Size Reduction

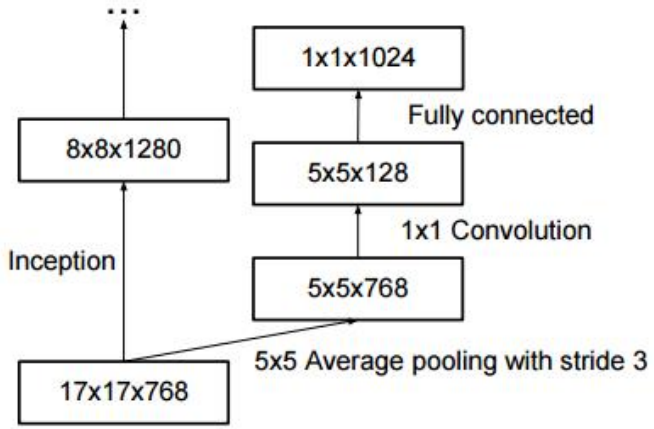


Figure 8. Auxiliary classifier on top of the last 17×17 layer. Batch normalization[7] of the layers in the side head results in a 0.4% absolute gain in top-1 accuracy. The lower axis shows the number of itertions performed, each with batch size 32.

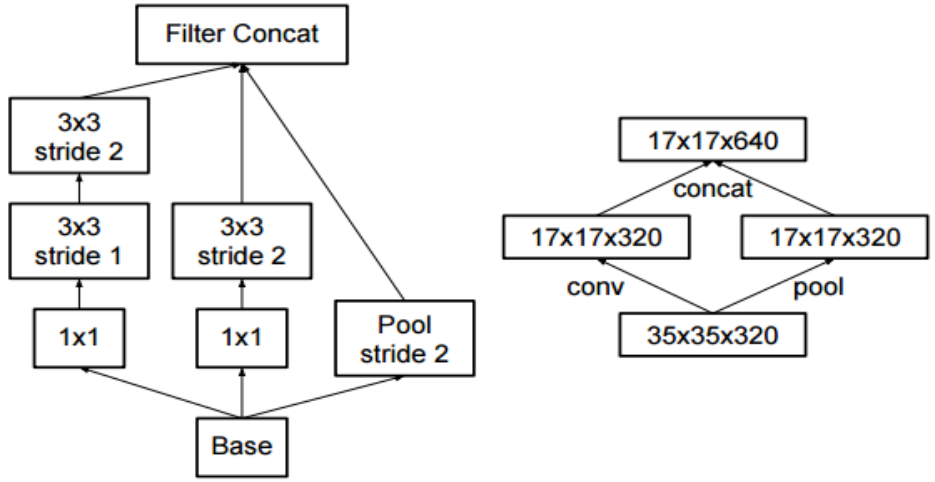


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

- GoogLeNet 논문에서 Auxiliary Classifiers를 활용하면 신경망이 수렴하는데 도움을 준다고 소개했지만, 실험 결과 별다른 효과가 없다고 언급함
- Drop out이나 Batch Normalization을 적용했을 때, main classifiers의 성능이 향상된 것으로 보아, 성능 향상의 효과보다 정규화 효과가 있을 것이라고 추측

- 일반적인 CNN에서는 피처맵 사이즈를 줄이기 위해 풀링 연산을 사용하고, representational bottlenet을 피하기 위해 필터 수를 증가시킴
-> 이는 연산량과 더불어 신경망의 표현력을 감소시킴
- stride 2를 지닌 pooling layer와 conv layer를 병렬로 연결하여 표현력을 감소시키지 않고 연산량만 감소시킴

EWHA EURON

#6.4 Inception-v2 Architecture

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

- Inception-v2는 42층의 신경망이지만 연산량은 GoogLeNet보다 2.5배 많고, VGGnet과 비슷
- 각 inception module에서 conv 연산은 0-padding을 적용하였고, 그 이외의 conv layer에는 0-padding을 적용하지 않음
- 3개의 인셉션 모듈 사용

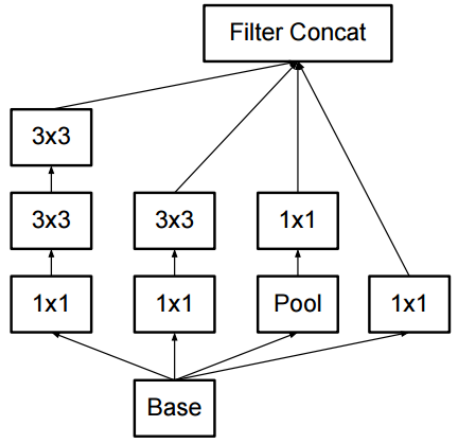


Figure 5. Inception modules where each 5 × 5 convolution is replaced by two 3 × 3 convolution, as suggested by principle 3 of Section 2.

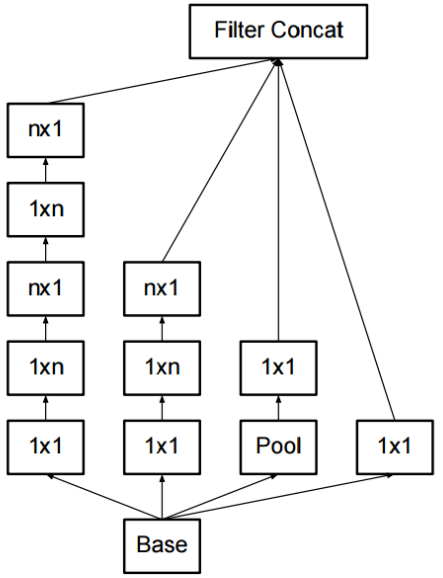


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

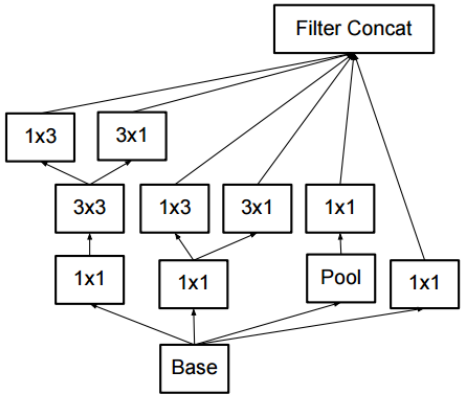


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8 × 8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1 × 1 convolutions) is increased compared to the spatial aggregation.

[Implementation]
Batch size 32/Epochs 100/Optimizer RMSProp with decay of 0.9,/epsilon = 1.0/Learning rate 0.045 decayed every 2 epoch using an exponential rate of 0.94.

#6.5 Label Smoothing

- Szegedy는 2015년 초에 더욱 빠른 학습을 가능하게 하는 Batch Normalization을 고안했고, 이 논문에서는 또 다른 정규화 방법인 Label Smoothing을 제안
- 정답에 대한 확신을 감소시켜 일반화된 성능을 얻을 수 있게 함
- 오버피팅을 피할 수 있도록 정답과 오답을 0과 1이 아닌 좀 더 스무스하게 점수를 주자!

Ex) [1,0,0,0] -> [0.75, 0.25, 0.25, 0.25]

- 스무딩 파라미터 입실론에 따라, label 분포를 뜻하는 u(k)=1/K를 사용했기 때문에 아래의 식을 가진다.
- K=1000, ϵ=0.1을 사용하여 이미지넷 2012 validation set에 대해 0.2% 정도의 성능 향상을 확인함

$$q'(k) = (1-\epsilon)\delta_{k,y} + \frac{\epsilon}{K}$$

#6.6 Inception-v3

Inception-v3 : Inception-v2 구조에서 위에서 설명한 기법들을 모두 적용하여 최고 성능을 나타내는 모델
=> BN-auxiliary + RMSProp + Label Smoothing + Factorized 7x7 (Factorized 7 x 7은 GoogLeNet의 도입부인 7 x 7 Conv를 3 x 3 세 개로 나눈 것)

Network	Crops Evaluated	Top-5 Error	Top-1 Error
GoogLeNet [20]	10	-	9.15%
GoogLeNet [20]	144	-	7.89%
VGG [18]	-	24.4%	6.8%
BN-Inception [7]	144	22%	5.82%
PReLU [6]	10	24.27%	7.38%
PReLU [6]	-	21.59%	5.71%
Inception-v3	12	19.47%	4.48%
Inception-v3	144	**18.77%**	**4.2%**

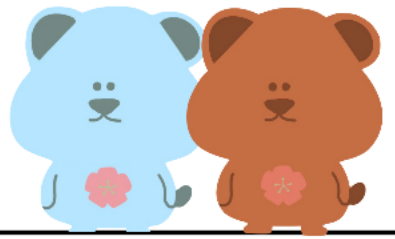
Table 4. Single-model, multi-crop experimental results comparing the cumulative effects on the various contributing factors. We compare our numbers with the best published single-model inference results on the ILSVRC 2012 classification benchmark.

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Table 5. Ensemble evaluation results comparing multi-model, multi-crop reported results. Our numbers are compared with the best published ensemble inference results on the ILSVRC 2012 classification benchmark. *All results, but the top-5 ensemble result reported are on the validation set. The ensemble yielded 3.46% top-5 error on the validation set.

1) Inception-v3 모델은 적은 파라미터를 가진 42-layer의 깊은 신경망으로 VGGNet와 비슷한 연산량을 보임
2) 앙상블 기법을 활용하여 ILSVR 2012 dataset으로 top-1 17.2% error, top-5 3.58% error를 달성

THANK YOU



EWHA EURON