CNN 세션
제 3강

주제: 심화모델 개요

목차

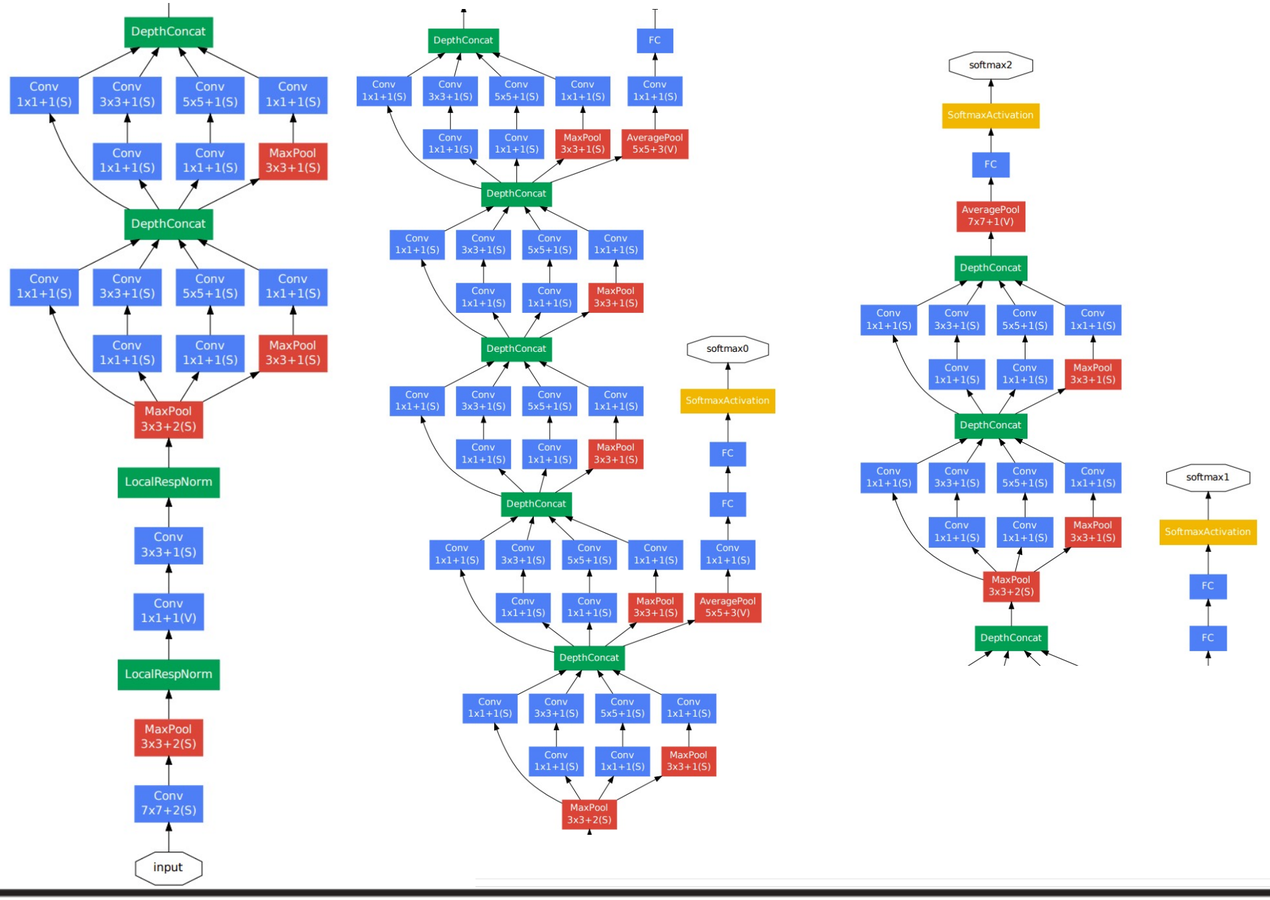
1. GoogleNet

논문출처

https://arxiv.org/pdf/1409.4842v1.pdf

2014년도에 가장 좋은 성적을 냈던 모델

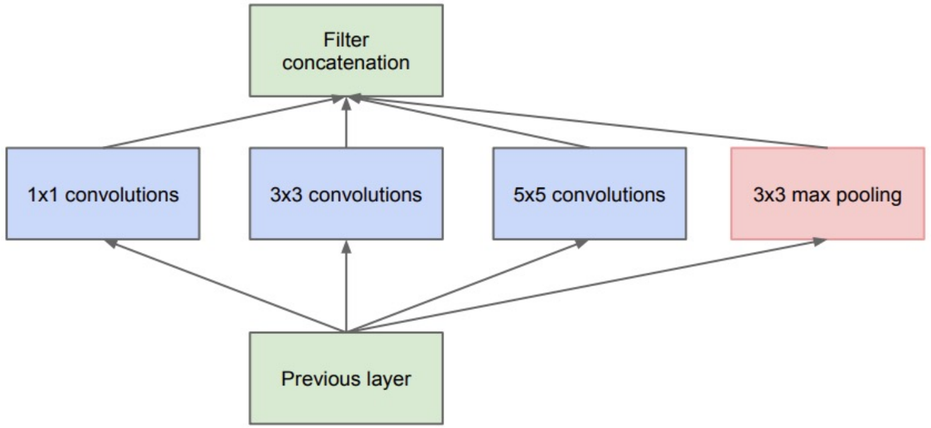
인셉션 모듈이라는 블록을 가지고 있어서 인셉션 네트워크라고 불리기도 합니다.



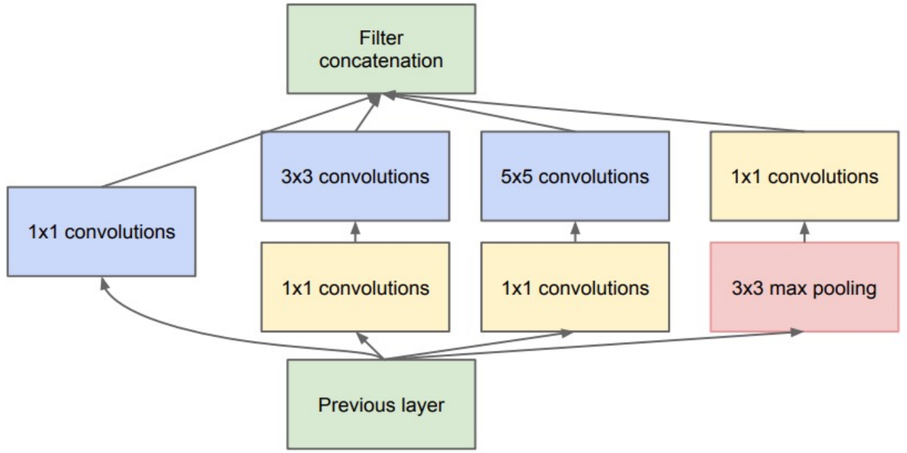
반복되는 부분

인셉션 모듈은 어떤 부분일까요?

1. GoogleNet



(a) Inception module, naïve version



(b) Inception module with dimension reductions

필기

1. GoogleNet

1X1 합성곱의 특징

1x1 합성곱은 이미지의 가로 ,세로 기준으로 하나의 픽셀을 입력으로 받습니다. 하지만, 채널의 관점에서는 채널의 갯수만큼의 입력을 받아, 채널,높이,너비가 각각 1인 결과를 생성합니다. 이를 목표 채널 수(필터 갯수)만큼 반복합니다.

따라서 1x1 합성곱은 입력 채널과 결과 채널간의 완전연결 네트워크입니다.

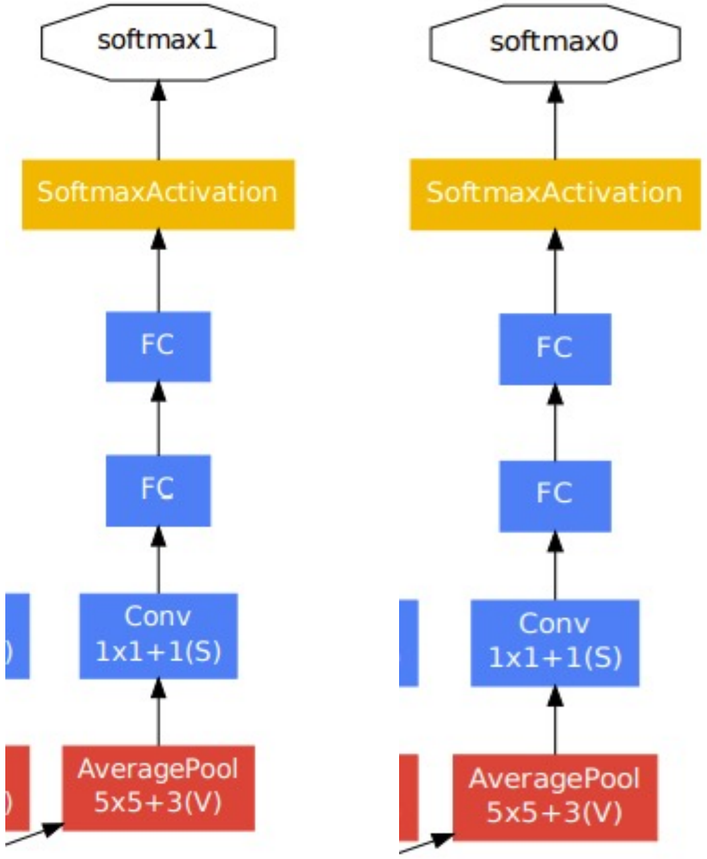
채널의 수가 줄어들게 설계. -> 입력 텐서를 채널 방향으로 압축!

예를 들어 128개 채널에서 256개 채널로 늘어나는 연산을 , 128개 채널을 가진 input에 1x1 합성곱을 통해 32개로 줄이고 다시 다른 합성곱 연산을 통해 256개로 늘립니다.

합성곱 연산의 특성을 이용한 이 방법이 메모리 효율을 증가시킨다고 합니다.

필기

1. GoogleNet 보조 분류기



대부분의 분류 모델에는 분류기(Softmax)가 맨 마지막에만 있습니다. 그런데 구글 네트워크는 중간 중간에 보조 분류기를 두었습니다.

모델이 깊어지면서 마지막의 분류기에서 발생한 손실이 입력 부분까지 도달하지 않는 현상이 발생합니다. (기울기 소실) 이를 보완하기 위해 보조 분류기를 두는 것입니다.

1. GoogleNet

모델 사용하기

```
model = torchvision.models.googlenet(pretrained=True).to(device)
#model=CNN().to(device)
```

Pretrained란?

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape `(3 x H x W)`, where `H` and `W` are expected to be at least `224`. The images have to be loaded in to a range of `[0, 1]` and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`.

더 자세한 모델 코드를 원하시면 아래 링크를 참고해주세요.

https://github.com/pytorch/vision/blob/master/torchvision/models/googlenet.py

2. ResNet

논문 출처: https://arxiv.org/pdf/1512.03385.pdf

네트워크를 얼마나 깊게 쌓을 수 있을까요?

네트워크를 쌓다보니, 너무 깊은 모델의 성능이 얕은 모델의 성능보다 떨어진다는 점이 발견되었습니다.

이를 해결하기 위한 모델이 바로 **ResNet**입니다.

이 방법으로 ResNet이 2015년도 이미지넷 대회에서 우승했다고 합니다.

2. ResNet

ResNet의 핵심특징: 잔차학습(residual learning)

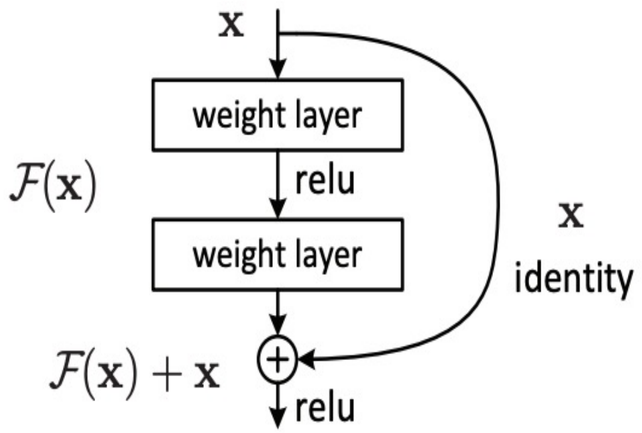


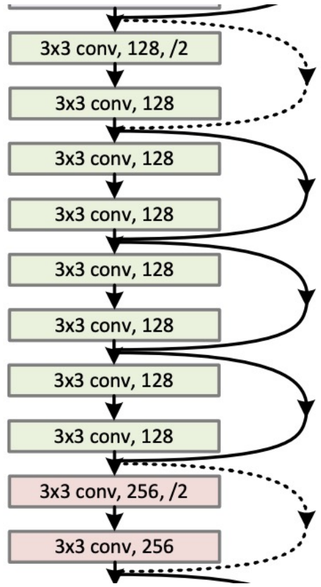
Figure 2. Residual learning: a building block.

잔차학습 블록은 이전 단계에서 뽑았던 특성들을 변형하지 않고 그대로 더해서 전달합니다.

때문에 입력 단에 가까운 곳에서 뽑은 단순한 특징과, 뒷부분에서 뽑은 복잡한 특징 모두 학습에 사용한다는 장점을 가지고 있습니다.

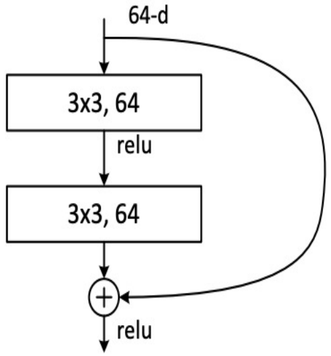
또한 identity를 그저 더하는 연산은 역전파 계산할 때 기울기가 1이기 때문에 손실이 줄어들지 않고, 모델의 앞부분까지 잘 전파됩니다. Googlenet에서 배운 보조분류기가 굳이 필요 없게 되는 것이지요.

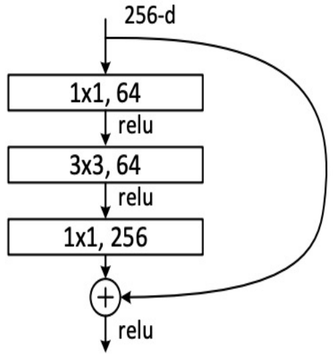
필기


3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256

2. ResNet

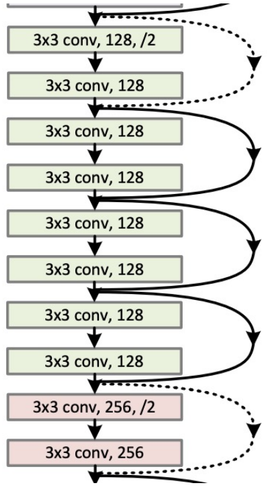
Bottlenet



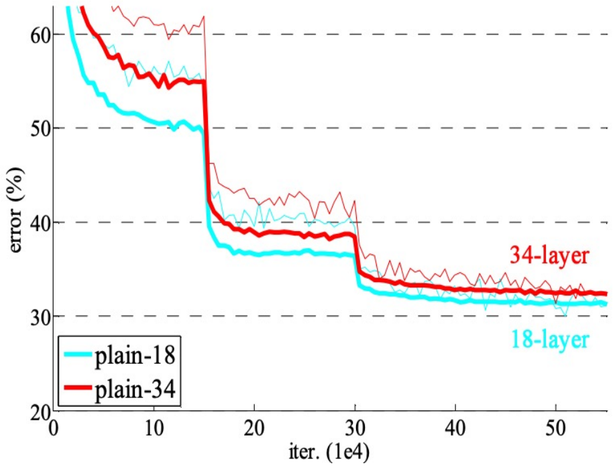


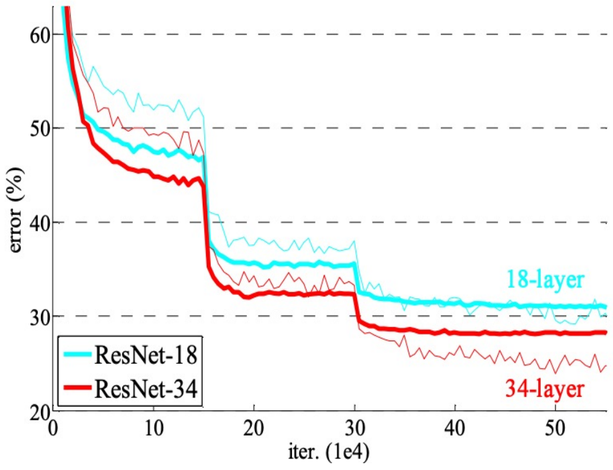
1x1합성곱으로 채널방향 압축. 압축된 상태에서 3X3 합성곱으로 추가 특성을 추출, 다시 1x1합성곱을 사용해 채널 수를 늘려줍니다.

이렇게 함으로써 변수 수를 줄이면서도 원하는 갯수의 특성을 뽑을 수 있다고 합니다.



점선: 이전 단계의 특성 지도의 크기가 가로 세로 방향으로 ½됩니다.





2. ResNet

모델 사용

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet101', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet152', pretrained=True)
model.eval()
```

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape `(3 x H x W)`, where `H` and `W` are expected to be at least `224`. The images have to be loaded in to a range of `[0, 1]` and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`.

https://pytorch.org/hub/pytorch_vision_resnet/

Model Description

Resnet models were proposed in “Deep Residual Learning for Image Recognition”. Here we have the 5 versions of resnet models, which contains 18, 34, 50, 101, 152 layers respectively. Detailed model architectures can be found in Table 1. Their 1-crop error rates on imagenet dataset with pretrained models are listed below.

Model structure	Top-1 error	Top-5 error
resnet18	30.24	10.92
resnet34	26.70	8.58
resnet50	23.85	7.13
resnet101	22.63	6.44
resnet152	21.69	5.94

```
model=torchvision.models.resnet50(pretrained=True)
```

간단한 복습

크고, 많은 데이터를 어떻게 하면 효율적으로 처리할 수 있을까?

나머지 두 개의 모델은 accuracy와 효율을 모두 잡은 모델입니다.

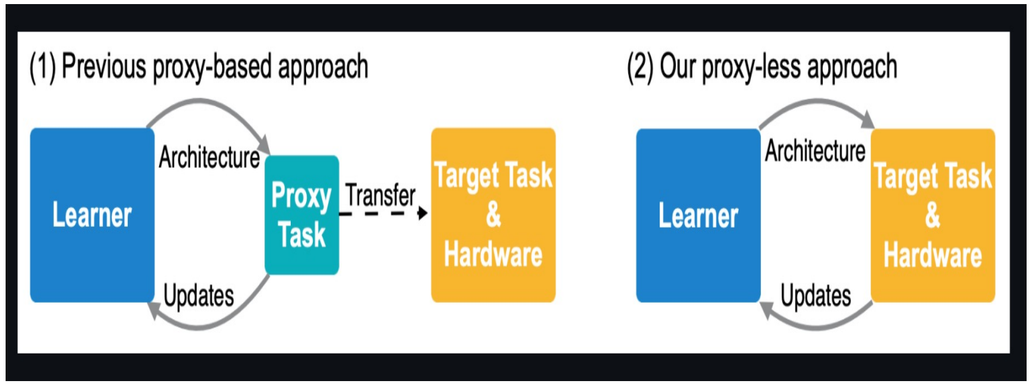
3. ProxylessNas

논문출처: https://arxiv.org/pdf/1812.00332.pdf

NAS: Neural Architecture Search: 강화학습 기반으로 최적의 네트워크를 찾는 방법

Proxy: 컴퓨터 네트워크에서 다른 서버 상의 지원을 찾는 클라이언트로부터 요청을 받아 중계하는 서버를 말한다.

Proxy서버는 실제 사용자의 로컬 컴퓨터에 상주할 수도 있고, 독립적인 서버로 존재할 수 있으며,인터넷 상에서 클라이언트와목적지 서버 사이 어느 곳이든 놓일 수 있다.

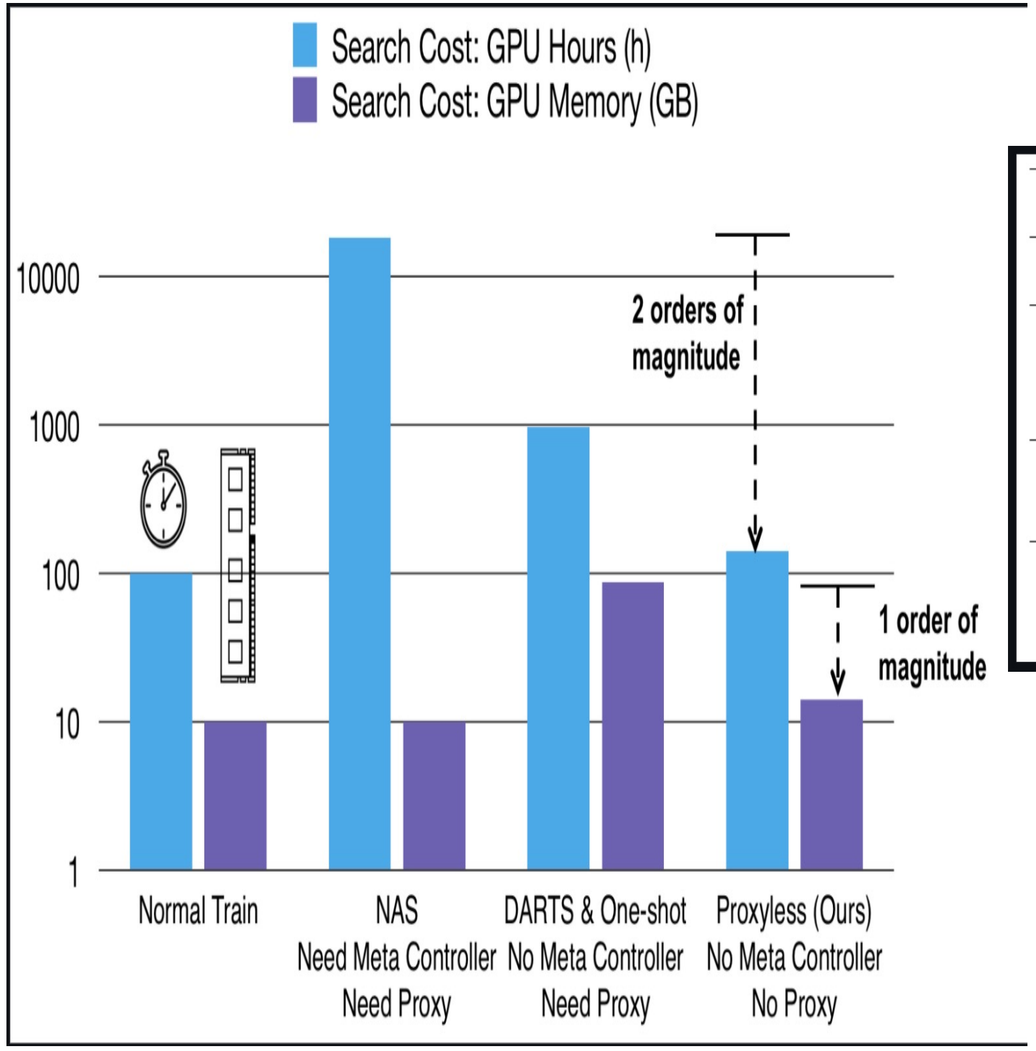


ProxylessNas는 proxy를 사용하지 않고, 모델 스스로 하드웨어와 Input_size에 따라 효율적인 학습 방식을 선택합니다.

Without any proxy, directly and efficiently search neural network architectures on your target **task** and **hardware**!

3. ProxylessNas

얼마나 효율적일까?



https://github.com/mit-han-lab/proxylessnas

Model	Top-1	Top-5	Mobile Latency	Hardware -aware	No Proxy	No Repeat	Search cost (GPU hours)
MobileNetV1 [16]	70.6	89.5	113ms	-	-	✗	Manual
MobileNetV2 [30]	72.0	91.0	75ms	-	-	✗	Manual
NASNet-A [38]	74.0	91.3	183ms	✗	✗	✗	48,000
AmoebaNet-A [29]	74.5	92.0	190ms	✗	✗	✗	75,600
MnasNet [31]	74.0	91.8	76ms	✓	✗	✗	40,000
MnasNet (our impl.)	74.0	91.8	79ms	✓	✗	✗	40,000
Proxyless-G (mobile)	71.8	90.3	83ms	✗	✓	✓	200
Proxyless-G + LL	74.2	91.7	79ms	✓	✓	✓	200
Proxyless-R (mobile)	**74.6**	**92.2**	78ms	✓	✓	✓	200

200x fewer

ProxylessNAS achieves state-of-the art accuracy (%) on ImageNet (under mobile latency constraint ≤ 80ms) with 200× less search cost in GPU hours.

3. ProxylessNas

모델 사용

```
# pytorch
from proxyless_nas import proxyless_cpu, proxyless_gpu, proxyless_mobile, proxyless_mobile_14, pr
net = proxyless_cpu(pretrained=True) # Yes, we provide pre-trained models!
```

All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape `(3 x H x W)`, where `H` and `W` are expected to be at least `224`. The images have to be loaded in to a range of `[0, 1]` and then normalized using `mean = [0.485, 0.456, 0.406]` and `std = [0.229, 0.224, 0.225]`.

4. EfficientNet

논문 출처 https://arxiv.org/pdf/1905.11946.pdf

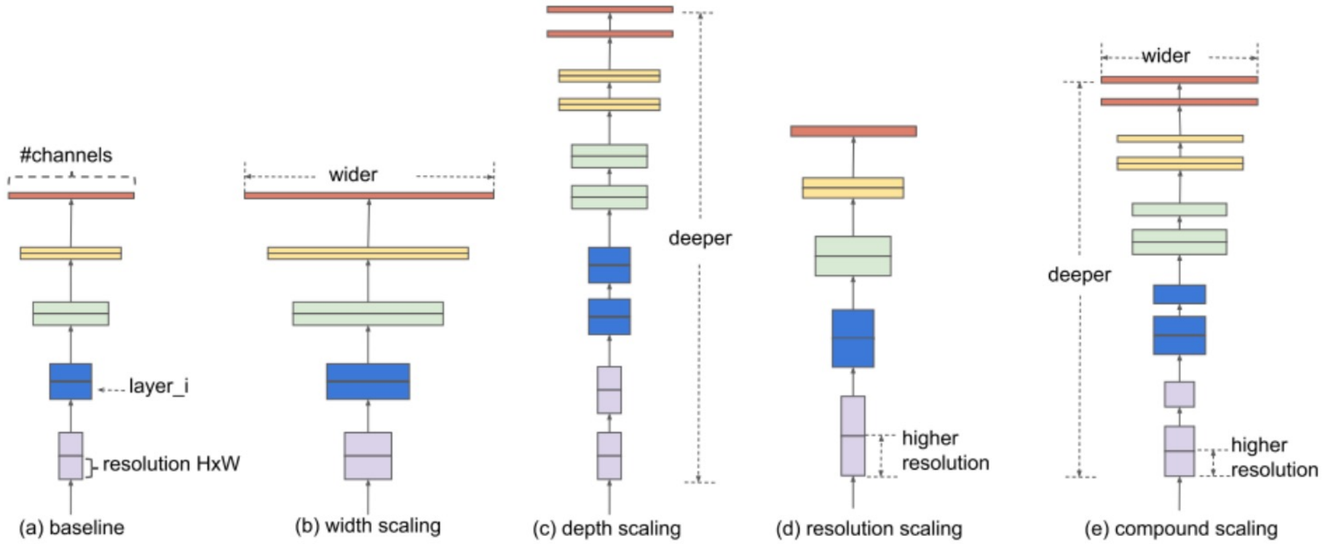


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

[Model Scaling 기법 예시]

모델을 Scaling할 때 고려하는 세 가지 요소:
1.필터크기, 2.layer의 깊이, 3.input_image의 해상도

각각
1. **width scaling**
2. **depth scaling**
3. **resolution scaling과 대응**

Depth scaling을 해준 대표적인 예가 Resnet이고, Width scaling해준 대표적인 예가 Mobilnet입니다.

Model scaling에 의한 성능향상은 baseline network에 매우 의존적이어서, baseline network를 설정하는 데에는 NAS를 사용합니다.

Efficientnet의 핵심 아이디어는 이 세 가지 스케일링을 동시에 고려하는 것!입니다.

4. EfficientNet

Compound Scaling

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

황금비율

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

베타와 감마는 왜 제곱일까요

[Compung Scaling 방법에 사용되는 notation]

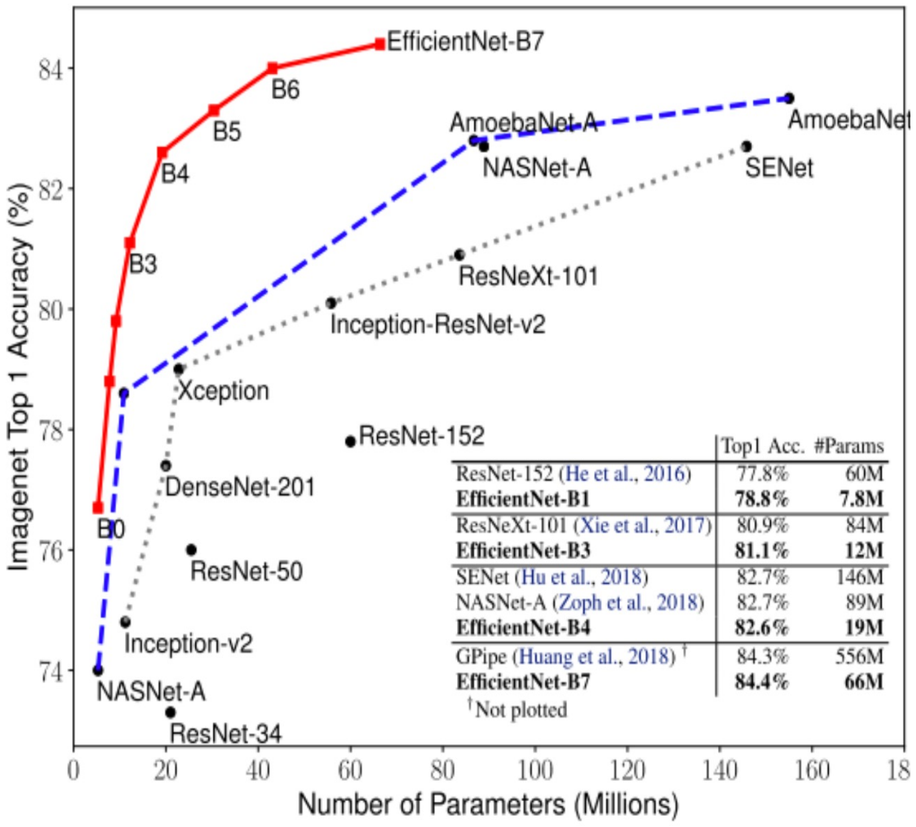
Model	Top-1 Acc.	Top-5 Acc.	#Params
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M
Xception (Chollet, 2017)	79.0%	94.5%	23M
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M
EfficientNet-B3	**81.6%**	**95.7%**	**12M**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M
EfficientNet-B4	**82.9%**	**96.4%**	**19M**
SENet (Hu et al., 2018)	82.7%	96.2%	146M
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M
EfficientNet-B5	**83.6%**	**96.7%**	**30M**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M
EfficientNet-B6	**84.0%**	**96.8%**	**43M**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained o

B0 :가장 가벼운 모델.

성능을 높이고 싶으면 B4~B7사용!

4. EfficientNet 결과



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

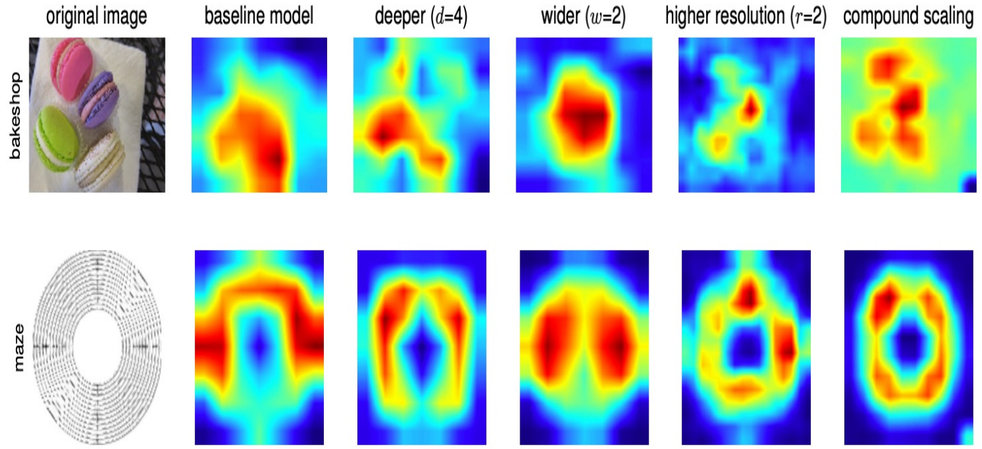


Figure 7. **Class Activation Map (CAM) (Zhou et al., 2016) for Models with different scaling methods-** Our compound scaling method allows the scaled model (last column) to focus on more relevant regions with more object details. Model details are in Table 7.

4. EfficientNet 모델 사용

https://github.com/lukemelas/EfficientNet-PyTorch

Upgrade the pip package with `pip install --upgrade efficientnet-pytorch`

pip -> !pip

```
from efficientnet_pytorch import EfficientNet
model = EfficientNet.from_pretrained('efficientnet-b7')
```

It is also now incredibly simple to load a pretrained model with a new number of classes for transfer learning:

```
model = EfficientNet.from_pretrained('efficientnet-b1', num_classes=23)
```

-끝-