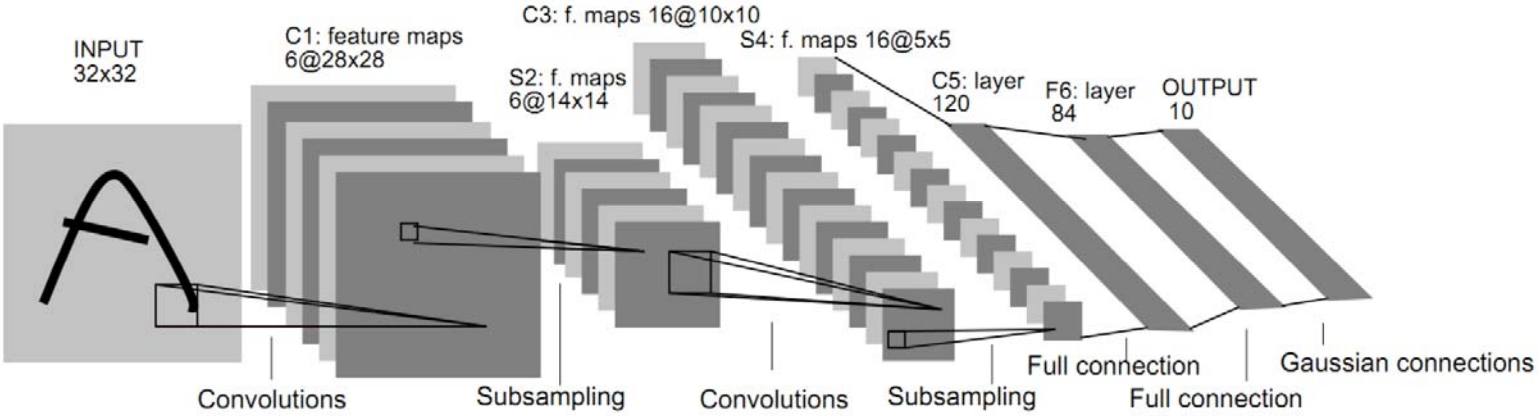
Pretrained Models

Prof. Hyunseok Oh

School of Mechanical Engineering
Gwangju Institute of Science and Technology

LeNet

- Y. LeCun, et al., “Gradient-based learning applied to document recognition, Proceedings of IEEE, 1998.
- CNN = Convolutional Neural Networks = ConvNet
- All are still the basic components of modern ConvNets.
  - The architecture is [Conv-Pool-Conv-Pool-FC-FC].



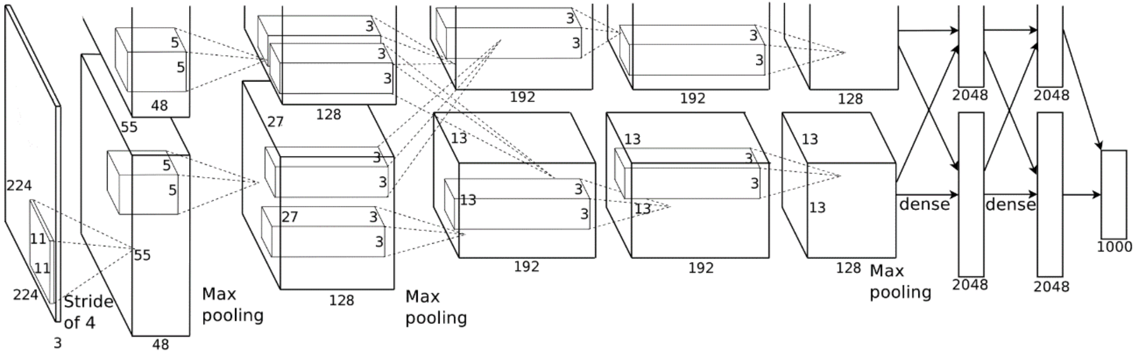
Issue and Breakthrough

- Too slow computers
  - Graphics processing unit (GPU)
- Wrong type of non-linearity
  - ReLU activation function: Glorot et al. “Deep sparse rectifier neural networks,” in Proceedings of 14^{th} AISTATS, 2011.
- Insufficient labeled datasets
  - Data augmentation
- Other key improvements
  - Dropout: N. Srivastava et al. “Dropout: A simple way to prevent neural networks from overfitting,” Journal of Machine Learning Research, 15:1929-1958, 2014.
  - Batch normalization: Loffe and Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” ICML, 2015.
  - Xavier initialization: Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks,” AISTATS, 2010.

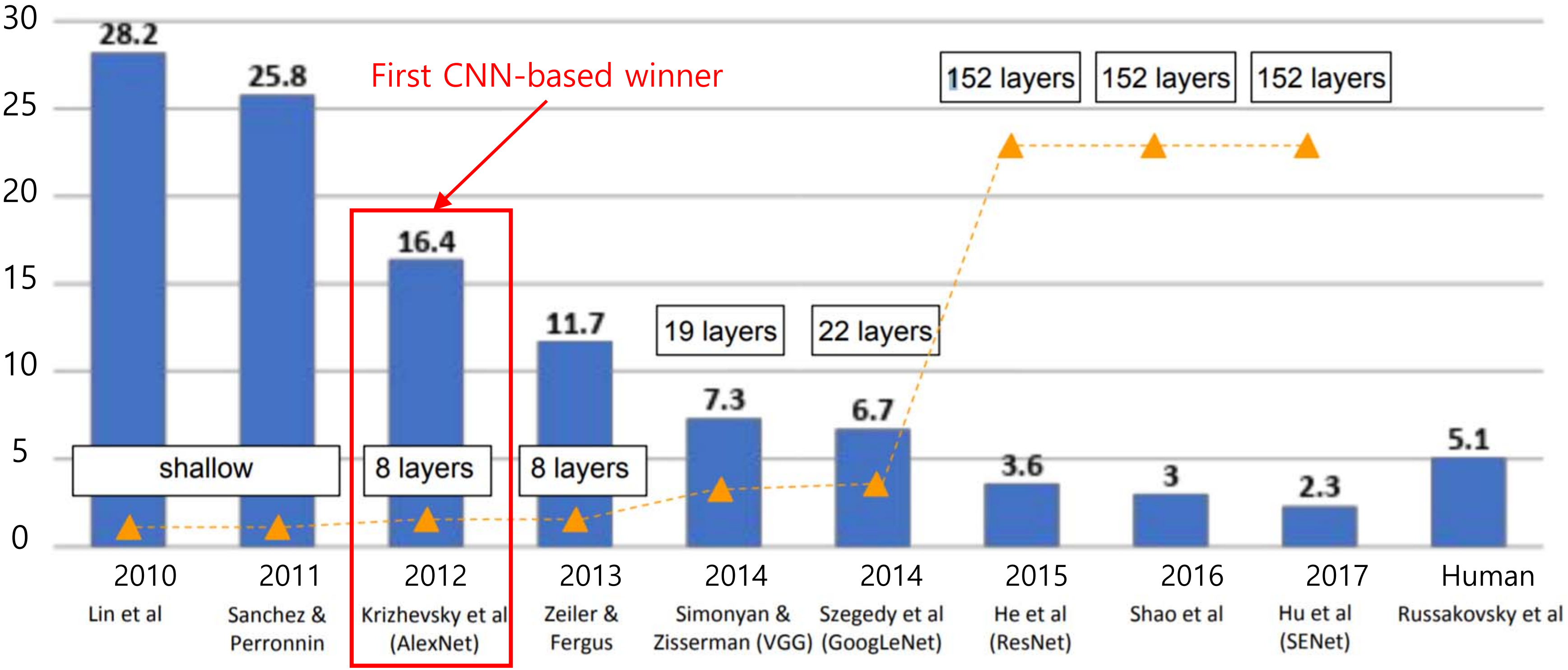
Overview of Modern CNNs

AlexNet

- A. Krizhevsky, et al., “Imagenet classification with deep convolutional neural networks”, NIPS, 2012.
- The architecture is: Conv1-Pool1-Conv2-Pool2-Conv3-Conv4-Conv5-Pool3-FC-FC-FC.
- Two of nVidia “GTX 580” GPU
- LeNet-style backbone, plus:
  - ReLU
    - “RevoLUtion of deep learning”
    - Accelerate training
  - Data augmentation
    - Label-preserving transformation
    - Reduce overfitting
  - Dropout
    - In-network ensemble
    - Reduce overfitting

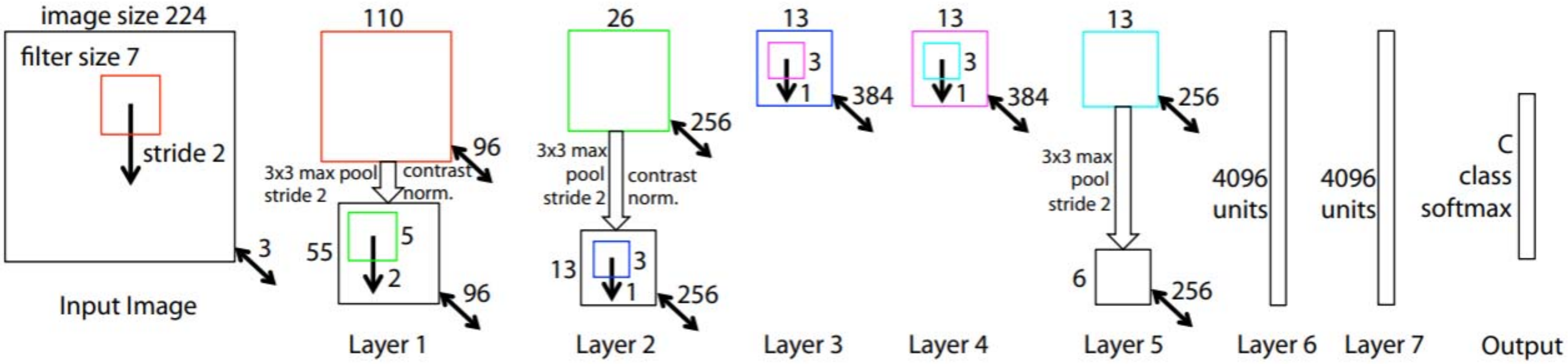


ImageNet Large Scale Visual Recognition Challenge (ILSVRC: 2010 - 2017)

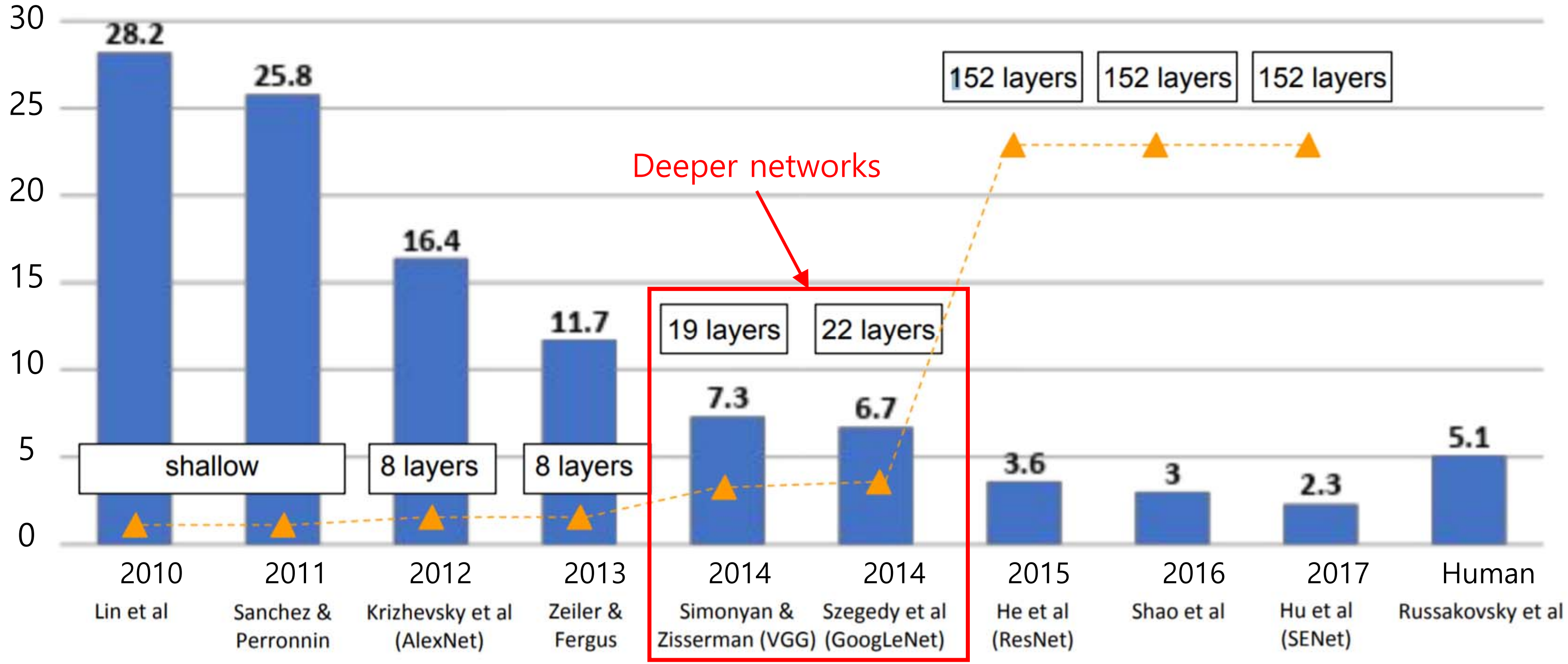


ZFNet

- M.D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV, 2013.
- Improved hyperparameters over AlexNet
- Conv1: changed from (11x11 stride 4) to (7x7 stride 2)
- Conv3, 4, 5: changed fro 384, 384, 256 filters to 512, 1024, 512

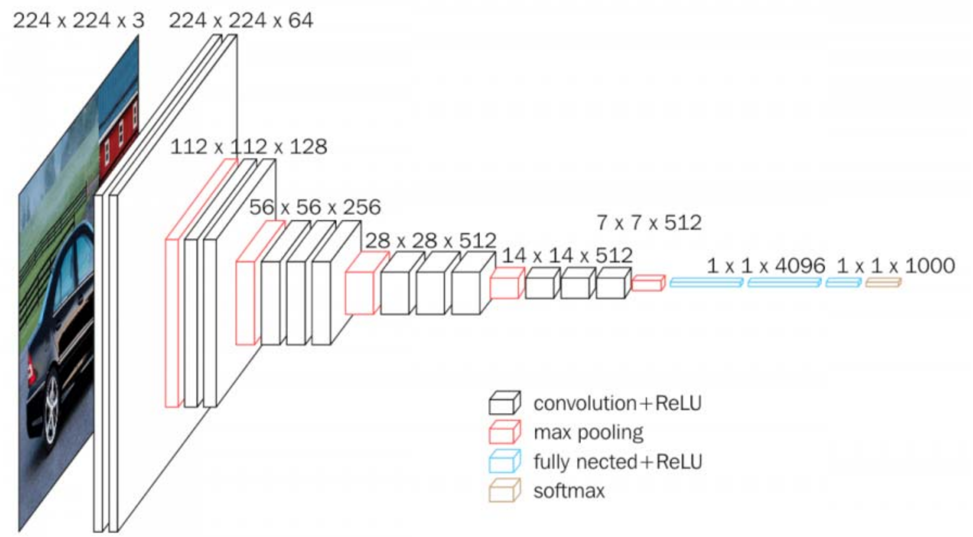


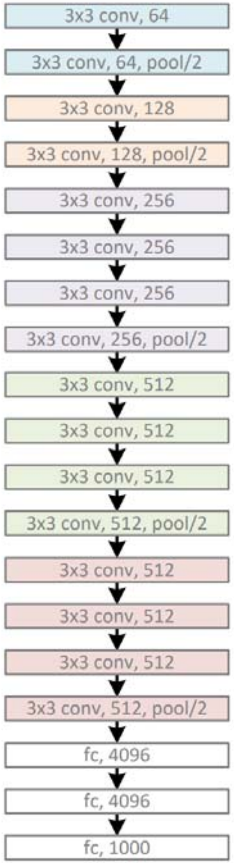
ImageNet Large Scale Visual Recognition Challenge (ILSVRC: 2010 - 2017)



VGG

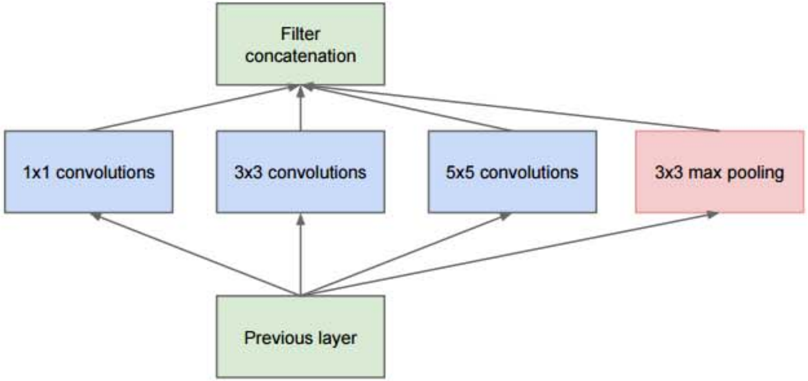
- K. Simonyan, et al., “Very deep convolutional networks for large-scale image recognition”, CVPR, 2014.
- Deeper networks: 8 layers (AlexNet) to 16 layers (VGG 16)
- Small filters: Only 3x3 Conv, Stride 1, Pad 1



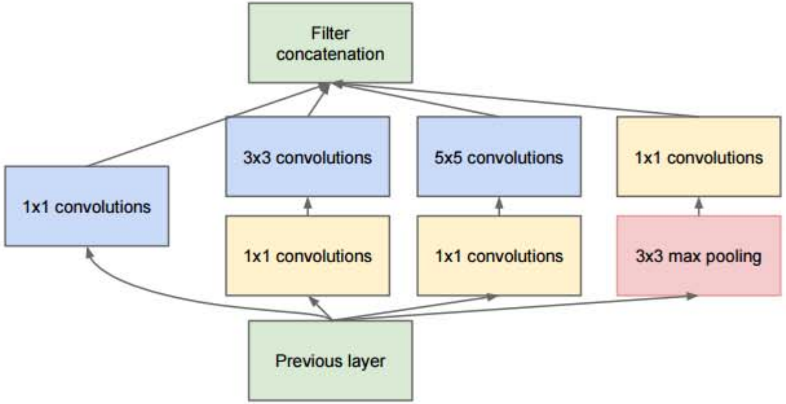


GoogleNet/Inception

- Multiple branches: 1x1, 3x3, 5x5, Pool
- Shortcuts: Stand-alone 1x1, merged by concatenation
- Bottleneck: Reduce dim by 1x1 before expensive 3x3 or 5x5 Conv



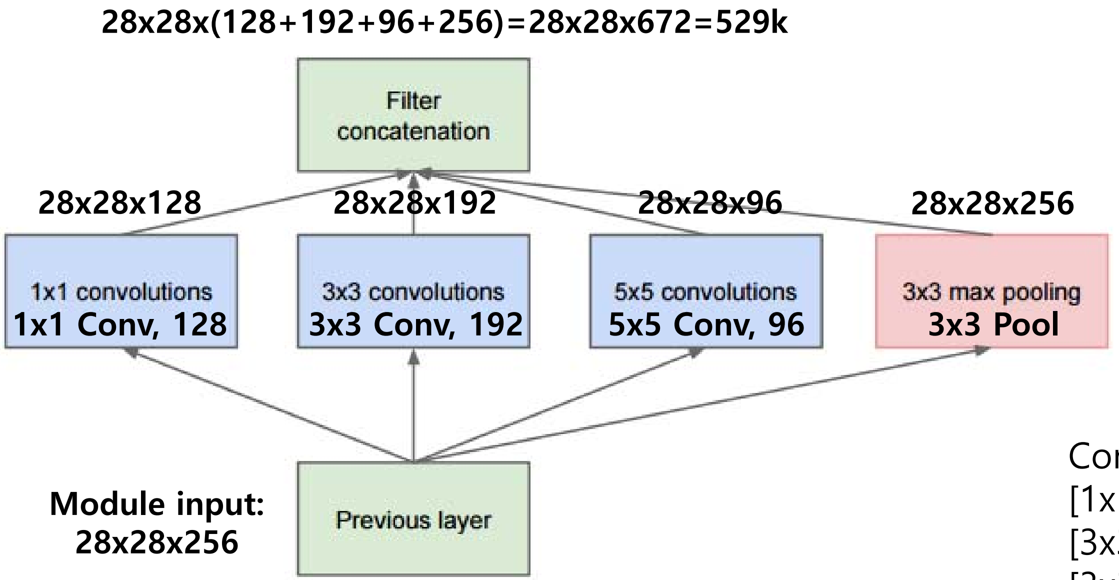
Naive inception module



Inception module

GoogleNet/Inception

- Operation cost is too expensive using the naive inception module.
- The cost can be reduced by using bottleneck layers of 1x1 convolution.

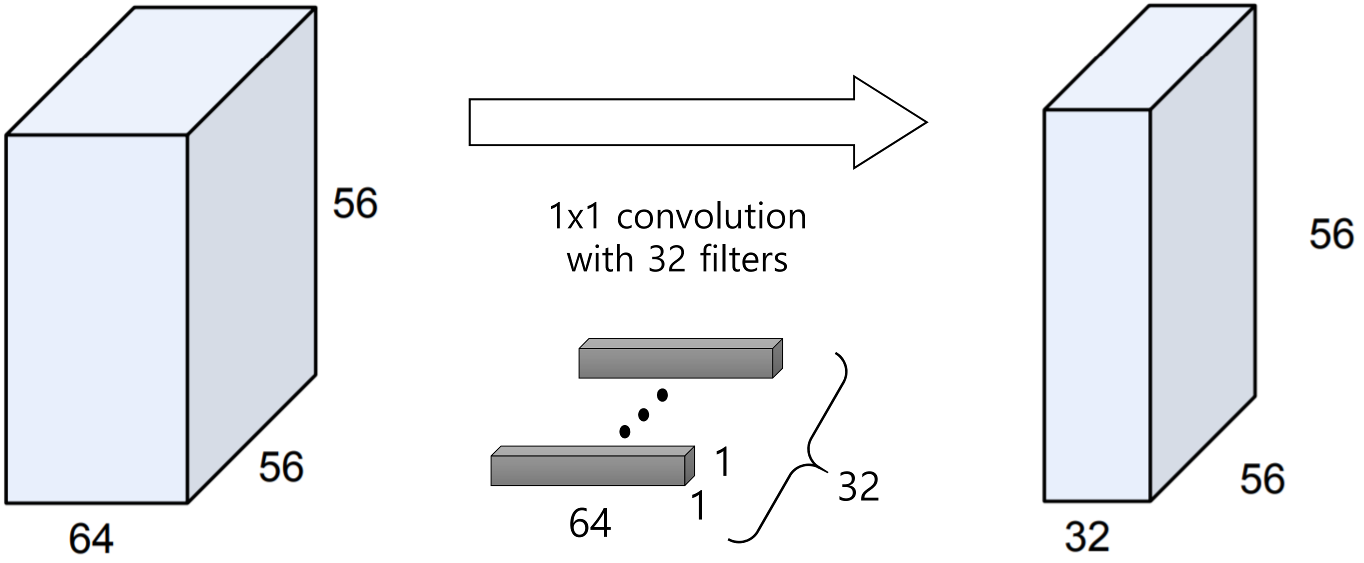


Naive inception module

Conv operations
[1x1 Conv, 128] 28x28x128x1x1x256
[3x3 Conv, 192] 28x28x192x3x3x256
[3x3 Conv, 196] 28x28x96x5x5x256
= 854 ops in total

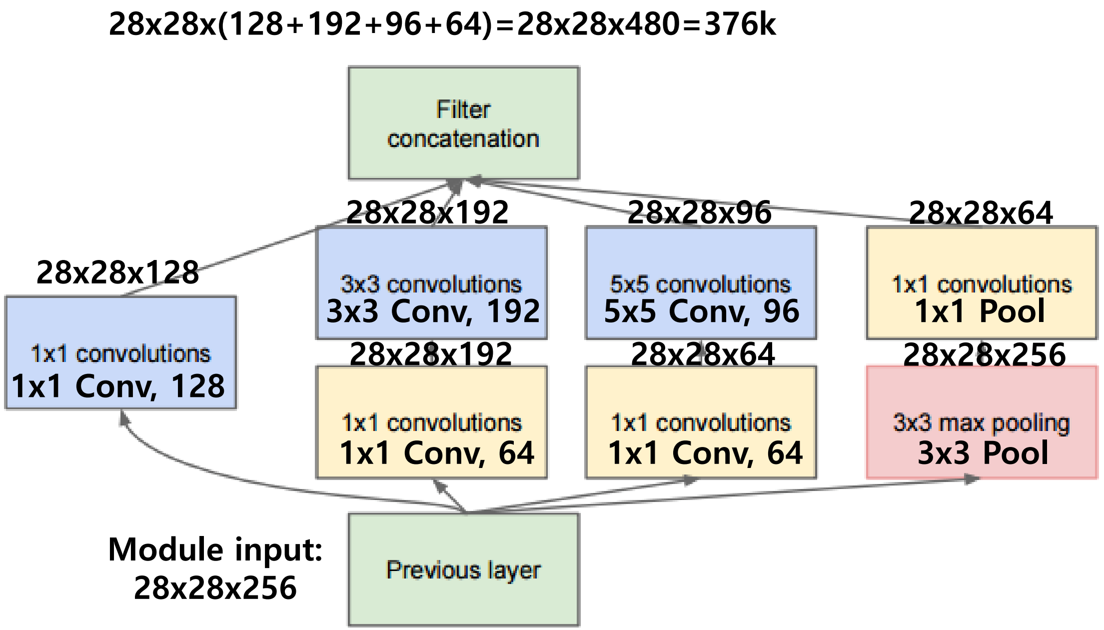
GoogleNet/Inception

- 1x1 convolution preserves spatial dimension while reducing depth.



GoogleNet/Inception

- Operation cost is reduced by using bottleneck layers of 1x1 Convolution.



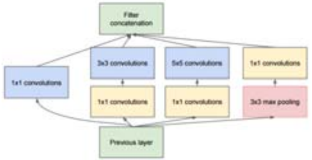
Inception module

Conv operations
[1x1 Conv, 64] 28x28x64x1x1x256
[1x1 Conv, 64] 28x28x64x1x1x256
[1x1 Conv, 128] 28x28x128x1x1x256
[3x3 Conv, 192] 28x28x192x3x3x64
[5x5 Conv, 96] 28x28x96x5x5x64
[1x1 Conv, 64] 28x28x64x1x1x256
= 358 ops in total

GoogleNet/Inception

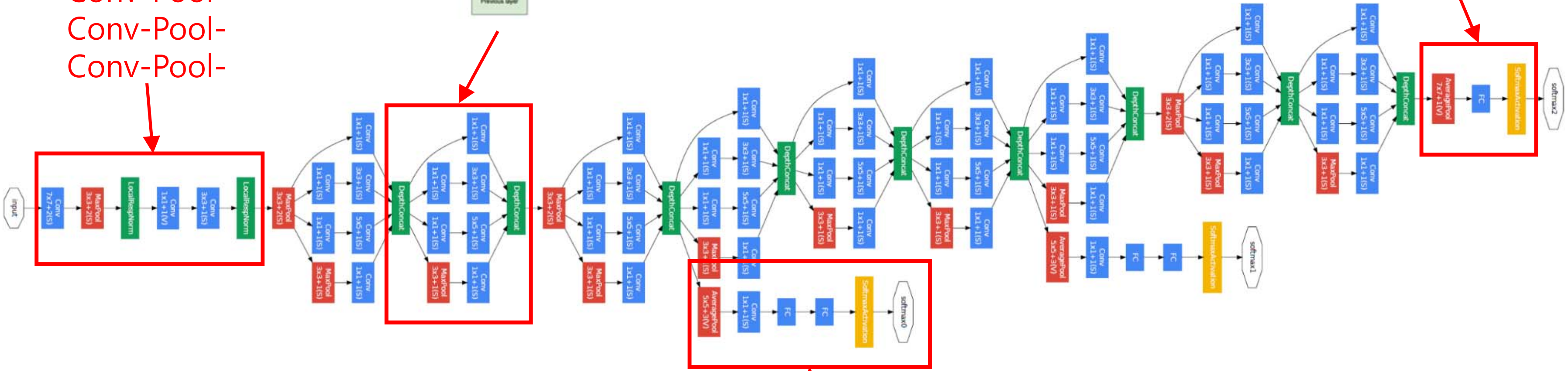
- Full google architecture : [Conv + Pool] + stacked inception module + GAP

Inception module



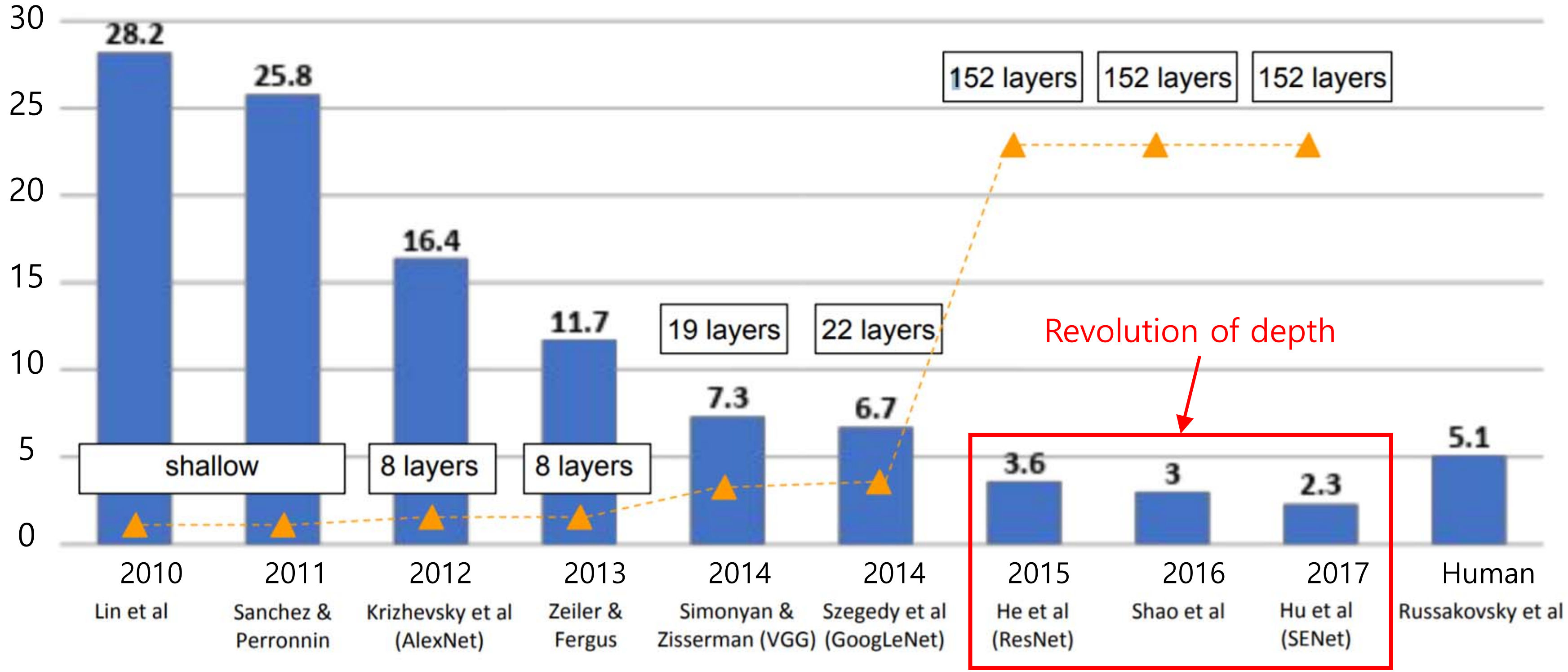
Conv-Pool-
Conv-Pool-
Conv-Pool-

Classifier using global average pooling (GAP)



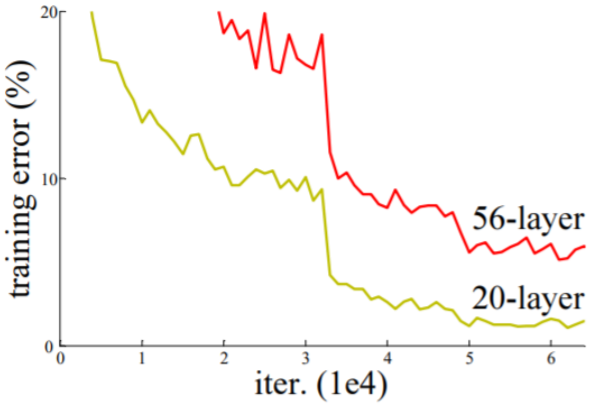
Auxiliary classification output to provide additional gradient: AvgPool–Conv–FC-FC-Softmax

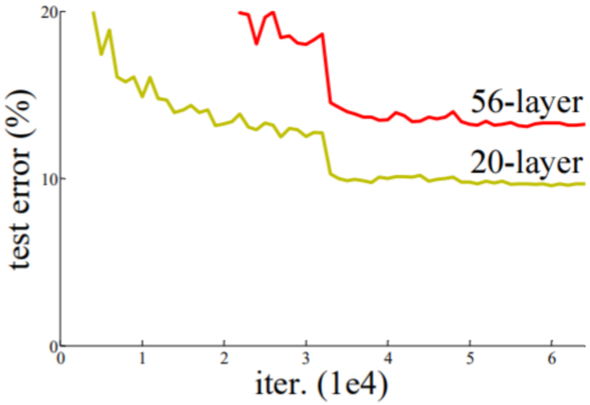
ImageNet Large Scale Visual Recognition Challenge (ILSVRC: 2010 - 2017)



Deep Model by Stacking Additional Layers

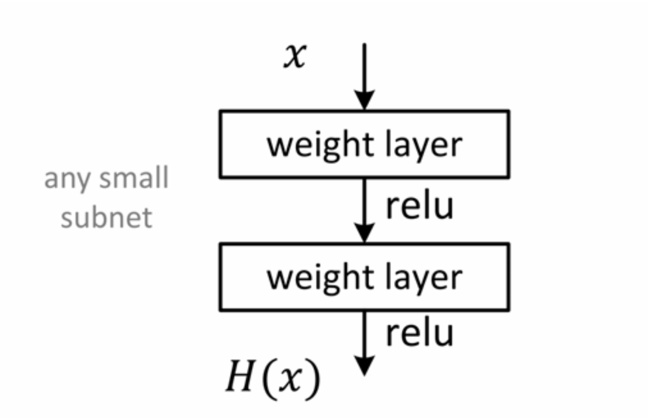
- What happens if a large number of layers are stacked on a plain network?
  - Unexpectedly, it leads to high training and test errors.
- This observation is attributed to an optimization problem. Deeper models are difficult to optimize. The search space is too large to handle.
- Hypothesis for possible solution: The deeper model is able to perform at least as well as the shallower model.





ResNet

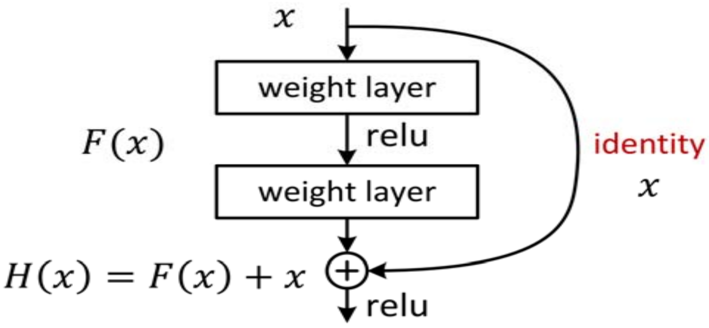
- K. He, et al. "Deep residual learning for image recognition," CVPR, 2016.



$H(x)$ is any desired mapping,
hope the small subnet fit $H(x)$.

ResNet

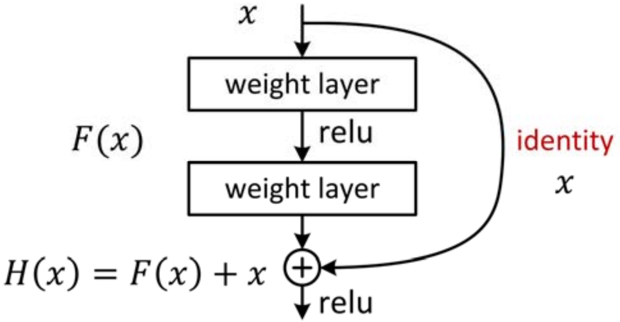
- K. He, et al. "Deep residual learning for image recognition," CVPR, 2016.
- Residual net
- Skip connection
  - A direct connection between two non-consecutive layers
  - No gradient vanishing

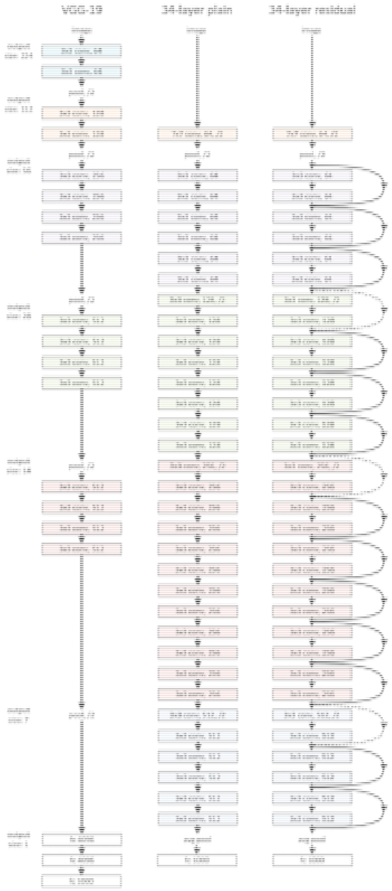


$H(x)$ is any desired mapping,
~~hope the small subnet fit $H(x)$~~
hope the small subnet fit $F(x)$
Let $H(x) = F(x) + x$

ResNet

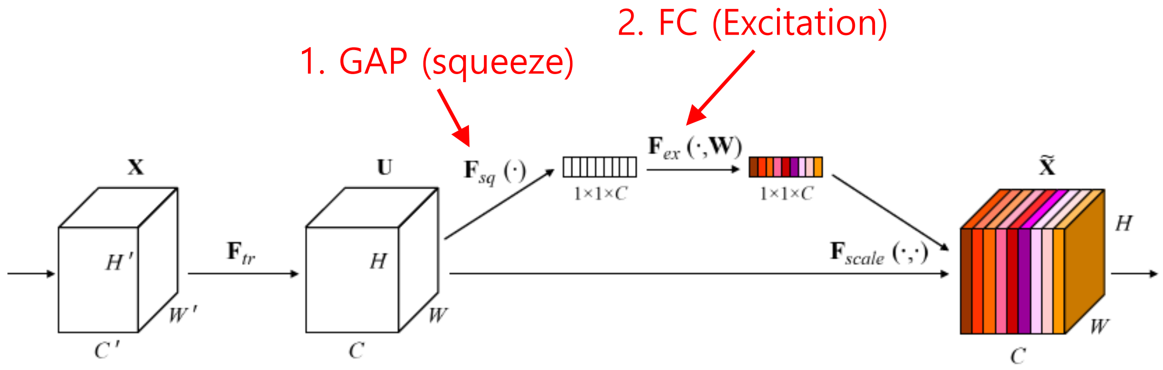
- Parameters are optimized to learn a residual, that is the difference between the value before the block and the one needed after.
- *F*(*x*) is a residual mapping w.r.t. identity
  - If identity were optimal, easy to set weights as 0.
  - If optimal mapping is closer to identity, easier to find small fluctuations.

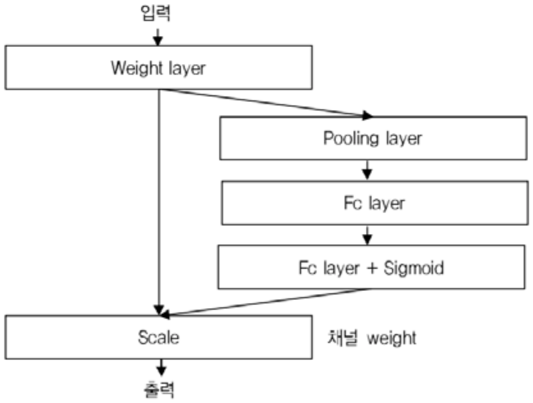




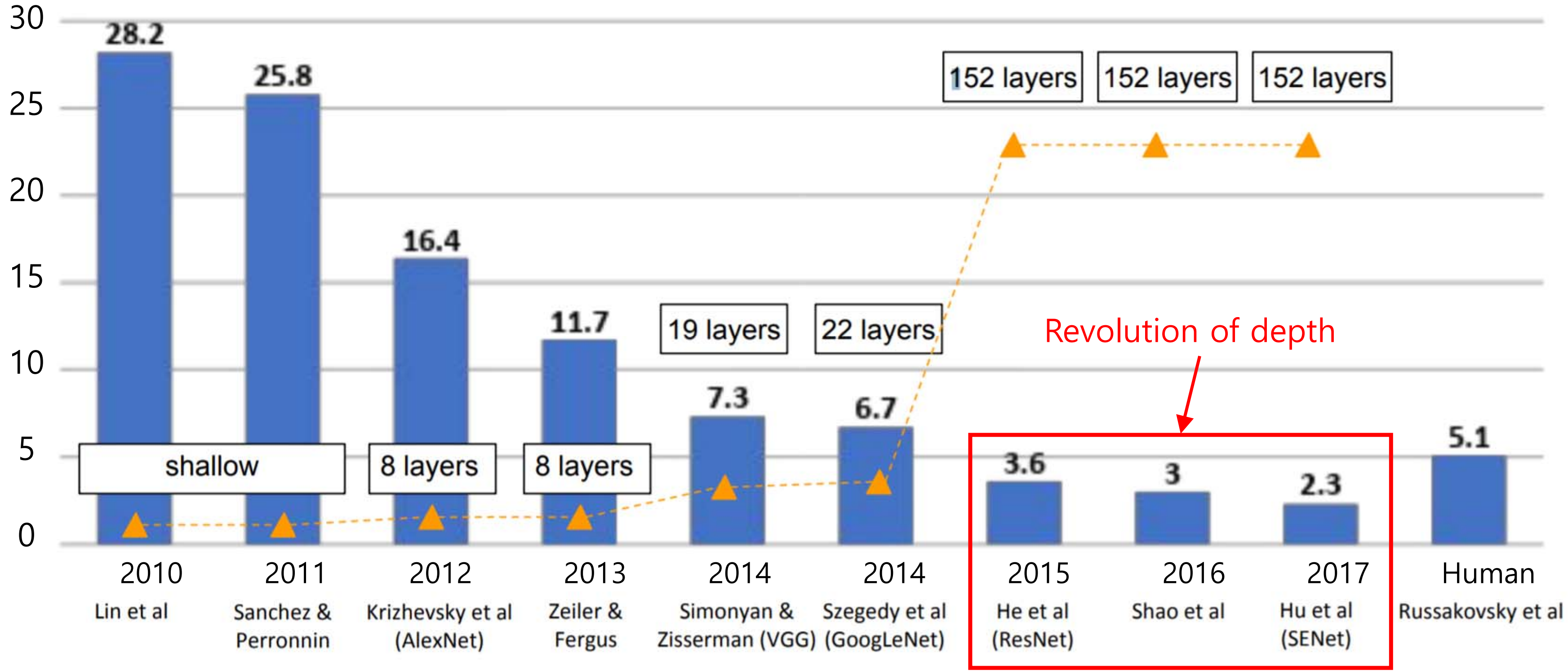
SENet

- SE block: 특징 채널 간의 상호작용에 가중치를 부여하여 성능(분류 정확도) 증가. 그럼에도 불구하고 연산량 증가는 크지 않음.
  - GAP를 통해 각 채널을 1차원으로 압축(Squeeze)
  - FC 층을 연결하여 각 채널의 상대적 중요도(가중치) 판단(Excitation)
- SE block은 기존 CNN 모델에 결합 가능.





ImageNet Large Scale Visual Recognition Challenge (ILSVRC: 2010 - 2017)

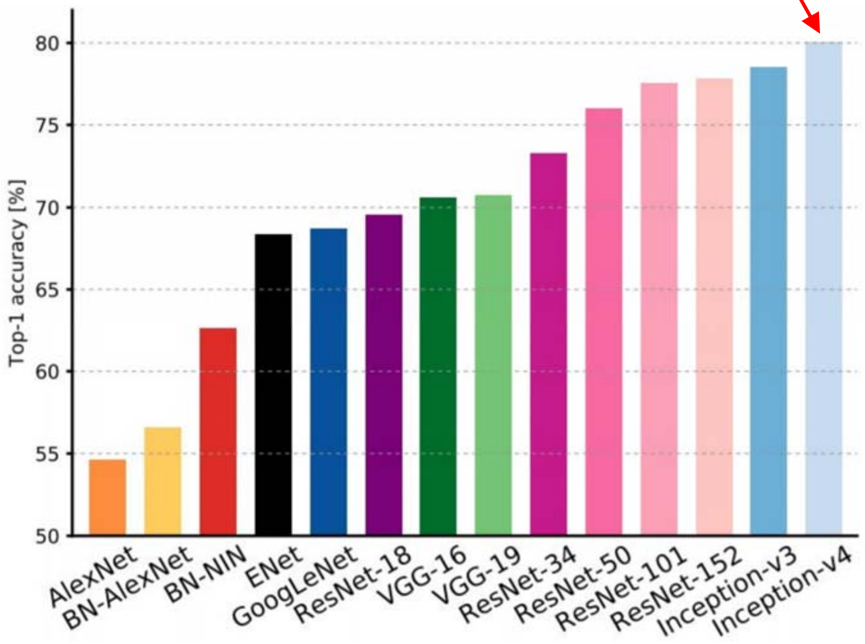


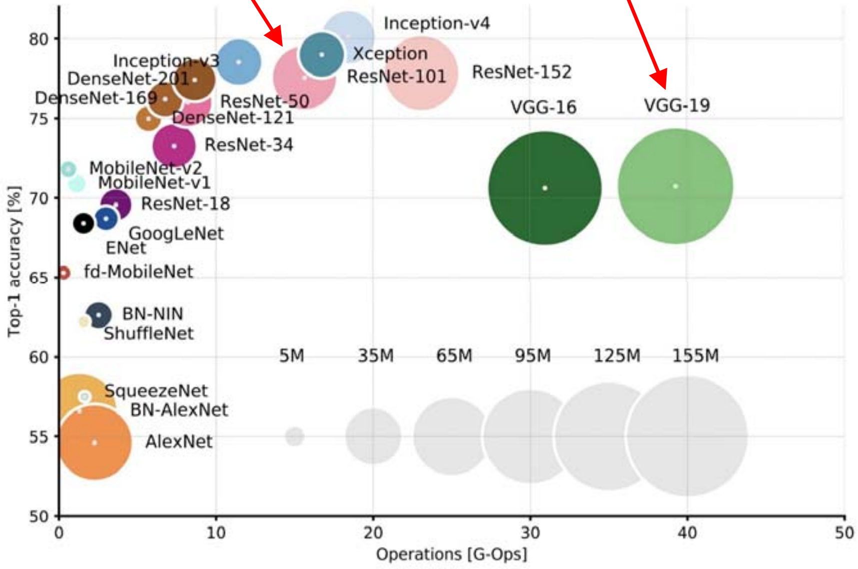
Comparison of Modern CNNs (As of 2017)

Inception v4:
ResNet +Inception

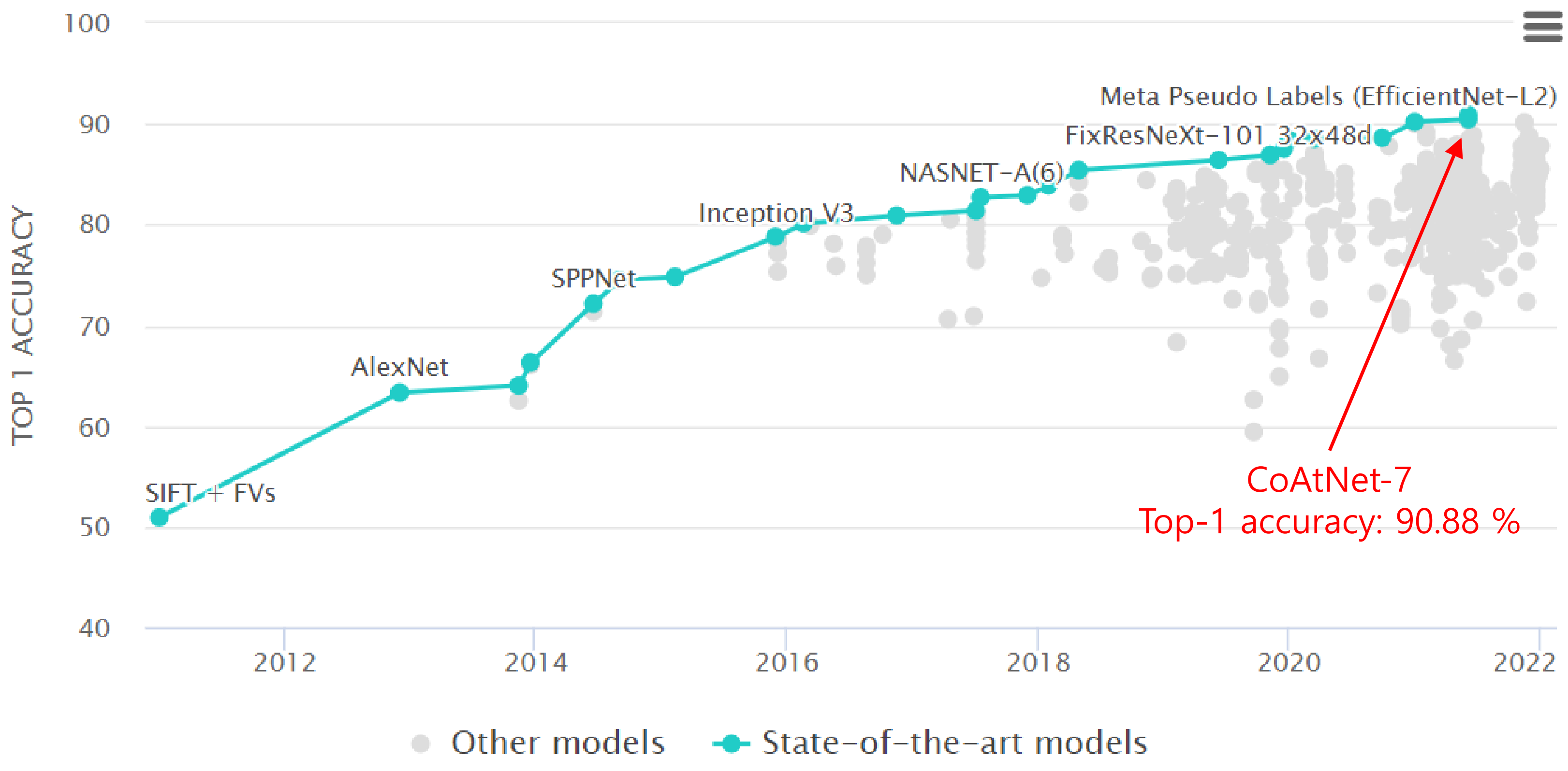
ResNet-101:
Moderate efficiency,
High accuracy

VGG19: Highest memory,
most operation





State of the Art Model for ILSVRC (As of 2022)



https://paperswithcode.com/sota/image-classification-on-imagenet

Representative TensorFlow API Models

Available Pretrained Models

- VGG
- Inception (GoogleNet)
- ResNet

- MobileNet
- DenseNet
- NasNet
- EfficientNet

`densenet` module: Public API for tf.keras.applications.densenet namespace.

`efficientnet` module: Public API for tf.keras.applications.efficientnet namespace.

`imagenet_utils` module: Public API for tf.keras.applications.imagenet_utils namespace.

`inception_resnet_v2` module: Public API for tf.keras.applications.inception_resnet_v2 namespace.

`inception_v3` module: Public API for tf.keras.applications.inception_v3 namespace.

`mobilenet` module: Public API for tf.keras.applications.mobilenet namespace.

`mobilenet_v2` module: Public API for tf.keras.applications.mobilenet_v2 namespace.

`mobilenet_v3` module: Public API for tf.keras.applications.mobilenet_v3 namespace.

`nasnet` module: Public API for tf.keras.applications.nasnet namespace.

`resnet` module: Public API for tf.keras.applications.resnet namespace.

`resnet50` module: Public API for tf.keras.applications.resnet50 namespace.

`resnet_v2` module: Public API for tf.keras.applications.resnet_v2 namespace.

`vgg16` module: Public API for tf.keras.applications.vgg16 namespace.

`vgg19` module: Public API for tf.keras.applications.vgg19 namespace.

`xception` module: Public API for tf.keras.applications.xception namespace.

https://www.tensorflow.org/api_docs/python/tf/keras/applications?hl=ko

MobileNet (딥러닝 모델 경량화)

- MobileNet (2017)
  - 딥러닝 모델 경량화
  - 스마트폰 혹은 임베디드 시스템을 위한 저용량 메모리 환경에서 딥러닝 적용 가능
  - ImageNet Top-5 에러율 10.5 %
- Depthwise separable convolution
  - Xception과 반대로 Depthwise conv 수행 이후 합산된 모든 채널에 대해 Pointwise conv 수행

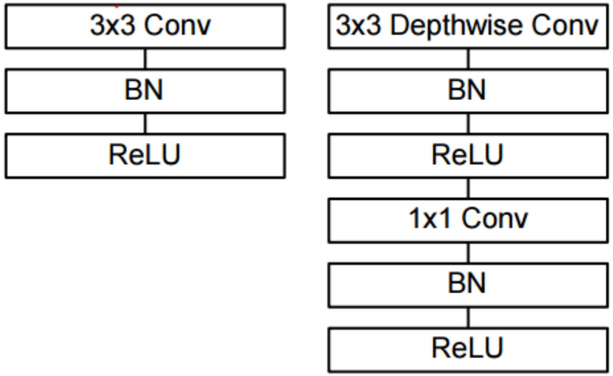
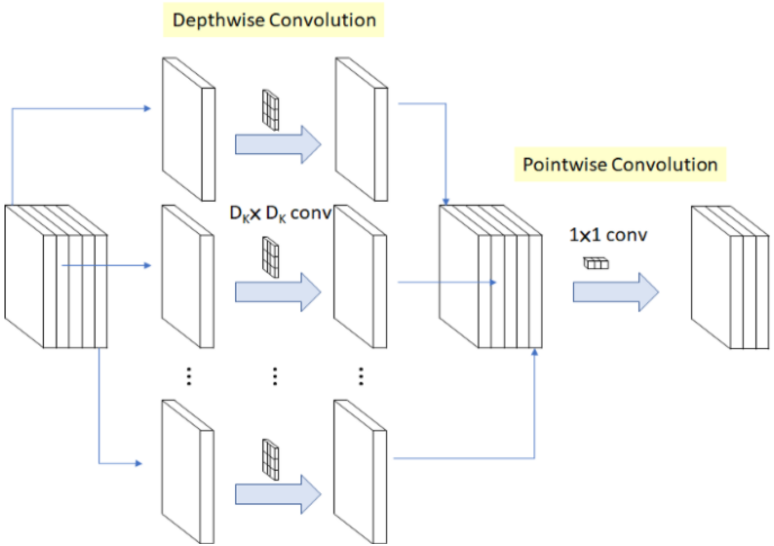
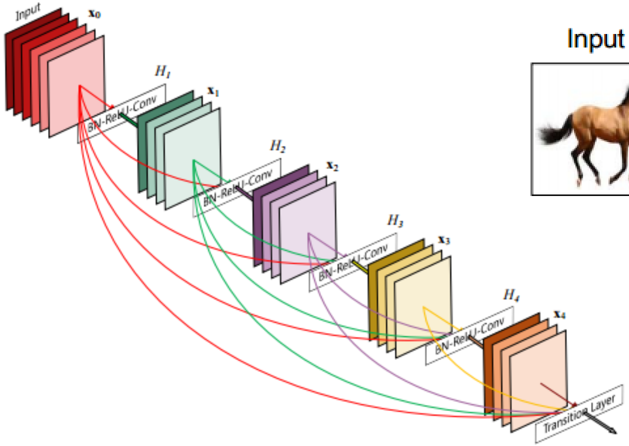


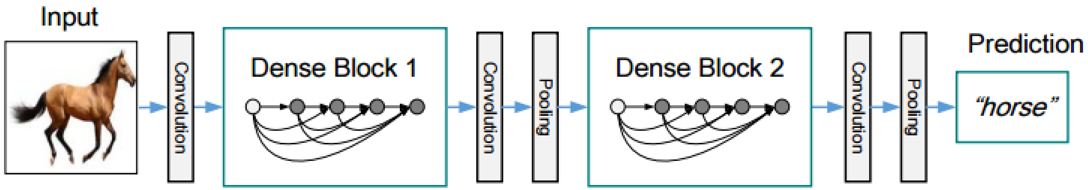
Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.



DenseNet (모든 특징 맵 연결)

- DenseNet (2017)
  - 네트워크 레이어에서 얻어지는 최대한의 정보 흐름을 이용
  - ImageNet Top-5 에러율 6.4 % (DenseNet 201 기준)
- Dense Block
  - 이전 레이어에서 얻어지는 특징 맵 (Feature map)을 그 이후의 모든 레이어의 특징 맵에 연결 (Concatenate)
  - 연결 시, 모든 특징 맵의 크기가 같으며 과도한 채널 수를 방지하기 위해 적은 채널 수 사용

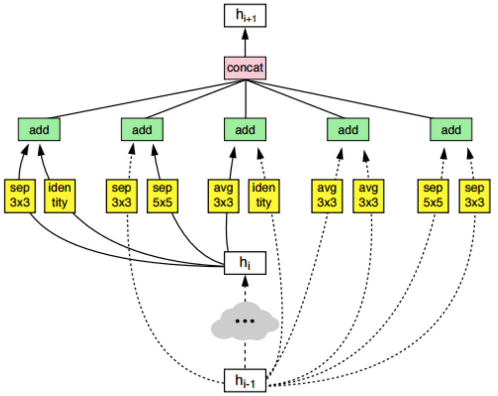




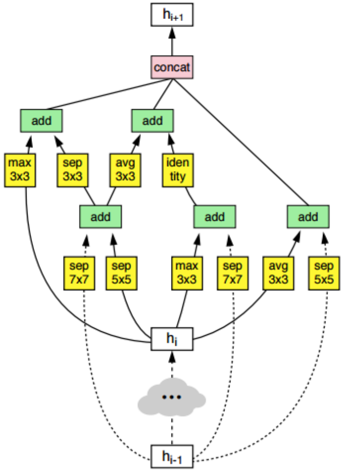
Dense block

NasNet (강화학습 기반 아키텍쳐 탐색)

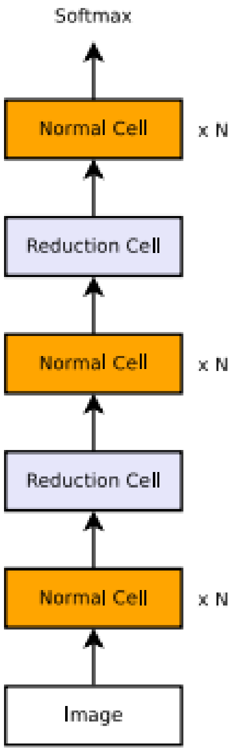
- NasNet (Neural architecture search Network) (2018)
  - Conv 레이어의 stride, 필터 크기 등을 RNN과 강화학습을 활용해 설계
  - ImageNet Top-5 에러율 4.0 % (NasNetLarge 기준)
- NasNet 세부 사항
  - 상대적으로 작은 데이터셋인 CIFAR10으로 최적의 모델 탐색
  - ImageNet에 적용



Normal Cell



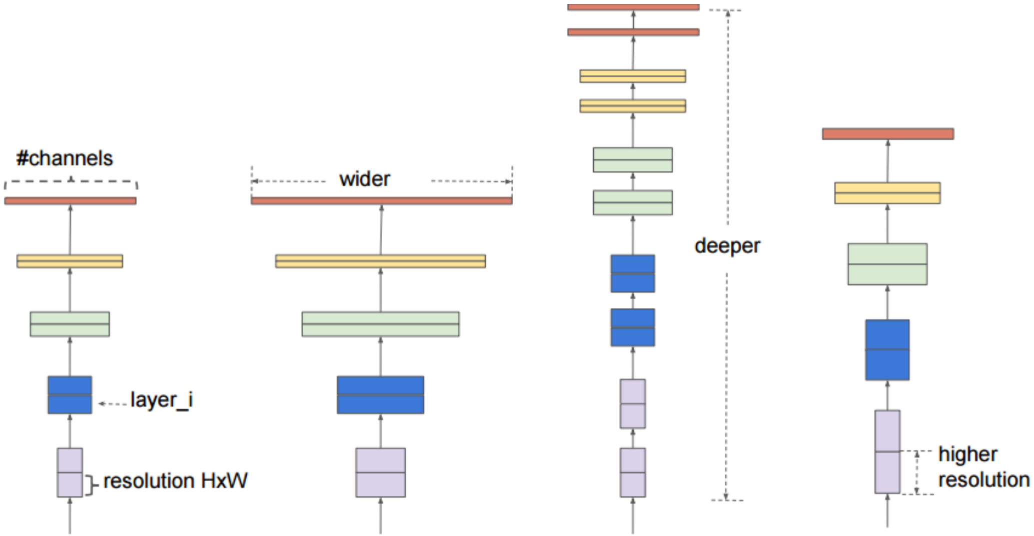
Reduction Cell



CIFAR10 Architecture

EfficientNet (이론 모델 기반 Scaling Up)

- EfficientNet (2019)
  - 모델의 깊이(Depth), 너비(Width), 입력 이미지 해상도(Resolution)를 결정하도록 학습
- 기존에는 ConvNet 성능 향상을 위해 사용자가 임의로 모델 깊이, 폭 결정했음.
- 기존과 다르게 이론 모델에 근거하여 Scaling up 제안

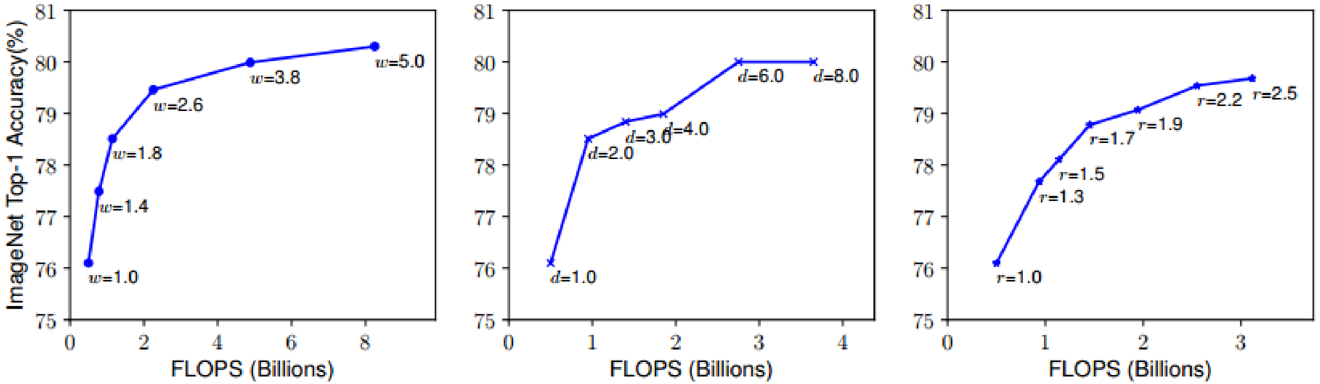


(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling

EfficientNet (이론 모델 기반 Scaling Up)

- Compound scaling
  - ϕ : 리소스 양에 따른 사용자 정의 상수
  - α, β, γ : Small grid search로 결정되는 변수
  - 총 FLOPS는 대략 2^{ϕ} 배 만큼 증가

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

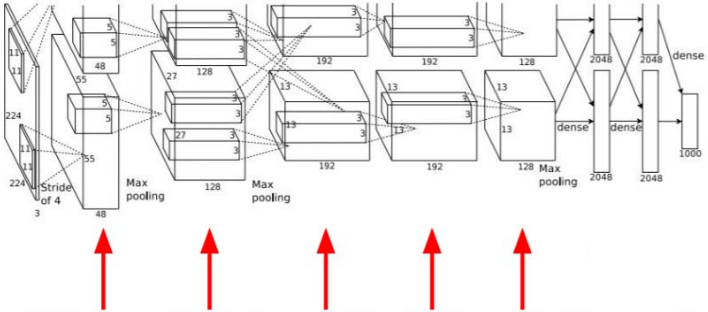


Demo

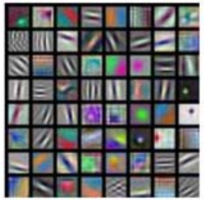
Pretrained Model as a Feature Extractor

- The pretrained models can be used as a generic feature extractor.
  - Pretrained models can extract general features that can help identify edges, textures, shapes, and object composition.
  - Better than handcrafted feature extraction on natural images.

Input Image:
3 x 224 x 224



What are the intermediate features looking for?



Alexnet:
64×11×11×3

ResNet-18:
64×7×7×3

DenseNet-121:
64×7×7×3

Pretrained Model (1/4)

- 모델 선택

```
# model_type = tf.keras.applications.densenet
# model_type = tf.keras.applications.inception_resnet_v2
# model_type = tf.keras.applications.inception_v3
model_type = tf.keras.applications.mobilenet
# model_type = tf.keras.applications.mobilenet_v2
# model_type = tf.keras.applications.nasnet
# model_type = tf.keras.applications.resnet50
# model_type = tf.keras.applications.vgg16
# model_type = tf.keras.applications.vgg19
```

- 모델 선언 및 세부 사항 확인

```
model = model_type.MobileNet() # Change Model (hint : use capital name)

model.summary()
```

Pretrained Model (2/4)

- 모델 세부 사항

Model: "mobilenet_1.00_224"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv1_bn (BatchNormalization	(None, 112, 112, 32)	128
conv1_relu (ReLU)	(None, 112, 112, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_dw_1_bn (BatchNormaliza	(None, 112, 112, 32)	128
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)	0
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_pw_1_bn (BatchNormaliza	(None, 112, 112, 64)	256
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_dw_2_bn (BatchNormaliza	(None, 56, 56, 64)	256
conv_dw_2_relu (ReLU)	(None, 56, 56, 64)	0
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_pw_2_bn (BatchNormaliza	(None, 56, 56, 128)	512
conv_pw_2_relu (ReLU)	(None, 56, 56, 128)	0

Pretrained Model (3/4)

- 입력 이미지 조절

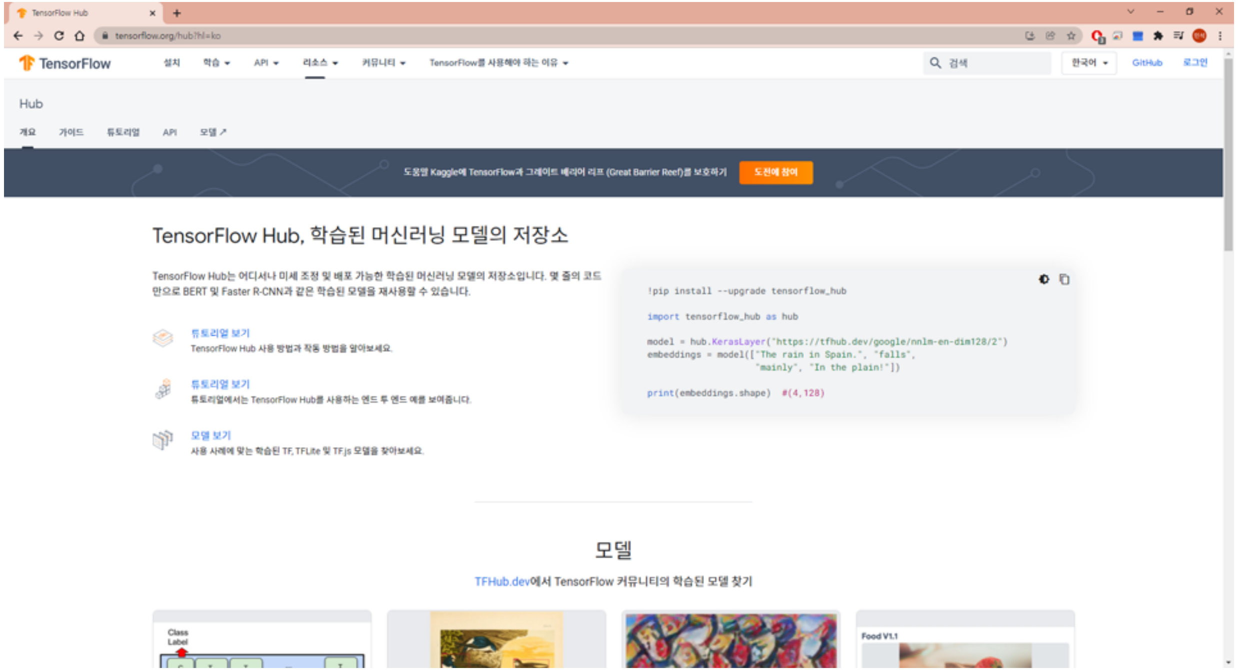
- 모델 예측

```
input_img = model_type.preprocess_input(resized_img)

pred = model.predict(input_img)
label = model_type.decode_predictions(pred)[0]
```

```
soccer_ball (92.07%)

knee_pad (2.68%)

football_helmet (2.44%)

ballplayer (1.17%)

tennis_ball (0.49%)
```

Pretrained Model (4/4)

- TF Hub
  - 재사용 가능한 머신러닝을 위한 개발형 리포지토리 및 라이브러리
  - 최근 연구된 모델을 간단한 코드로 사용 가능



https://www.tensorflow.org/hub?hl=ko