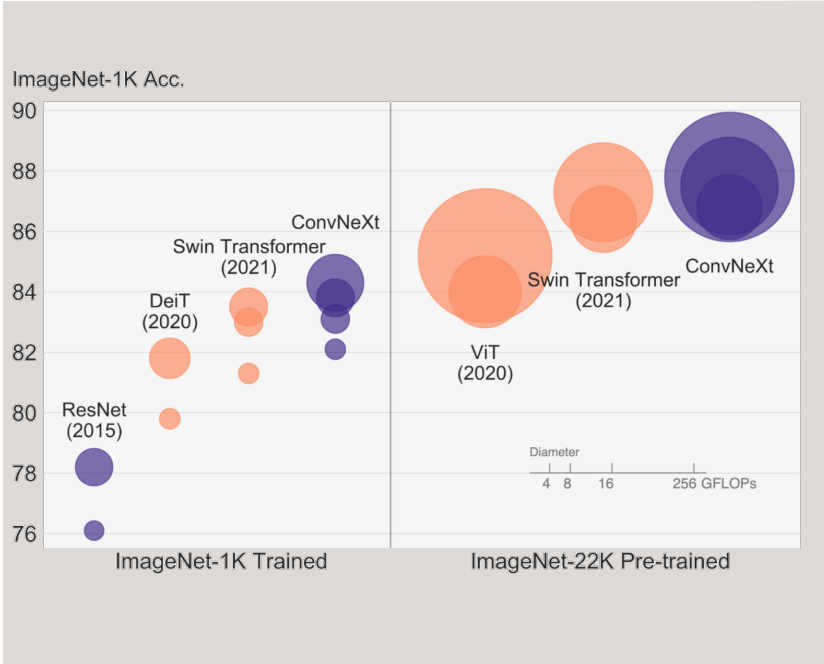
SOTA CONVOLUTIONAL NEURAL NETWORK

Chih-Chung Hsu (許志仲)
Institute of Data Science
National Cheng Kung University
https://cchsu.info



Last 2 lectures: Training neural networks

- One time setup
  - activation functions, preprocessing, weight initialization, regularization, gradient checking
- Training dynamics
  - babysitting the learning process,
  - parameter updates, hyperparameter optimization
- Evaluation
  - model ensembles, test-time augmentation

One more thing: Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

One more thing: Transfer Learning

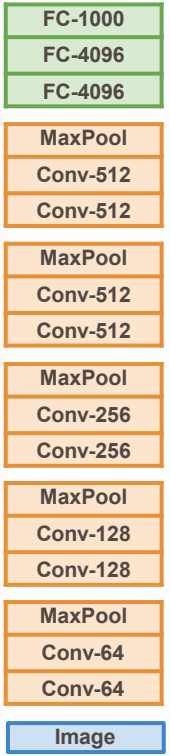
"You need a lot of a data if you want to train/use CNNs"

BUSTED

Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

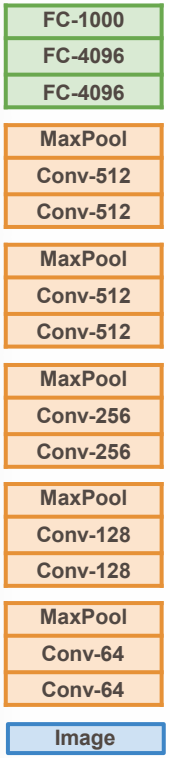
1. Train on Imagenet



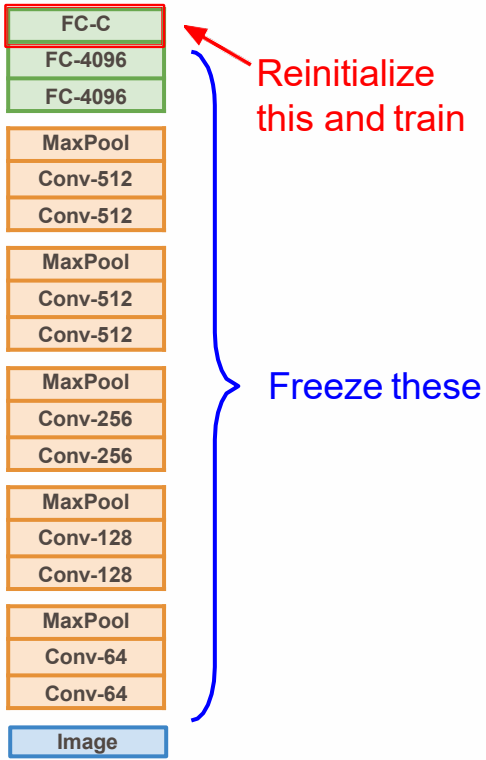
Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Train on Imagenet



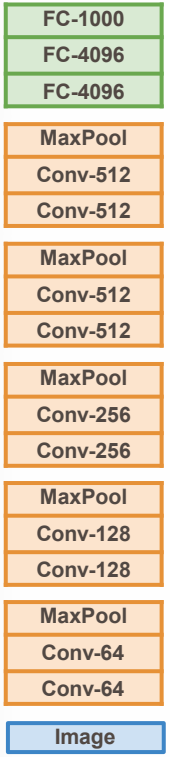
2. Small Dataset (C classes)



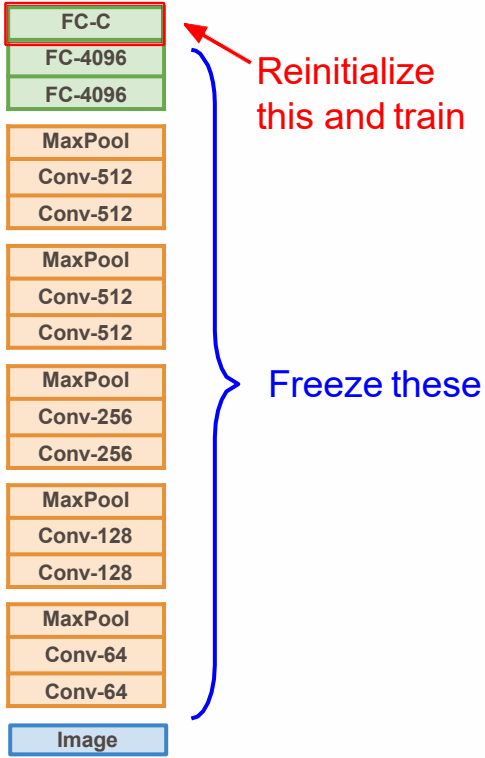
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

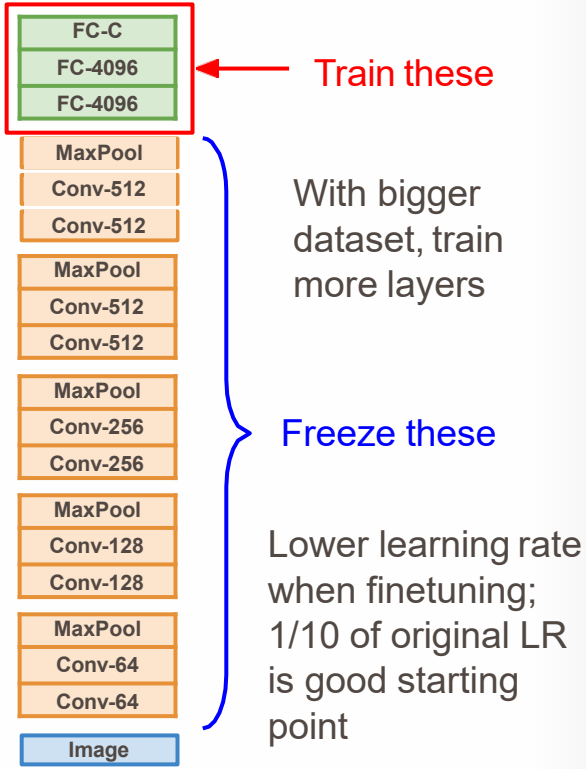
1. Train on Imagenet

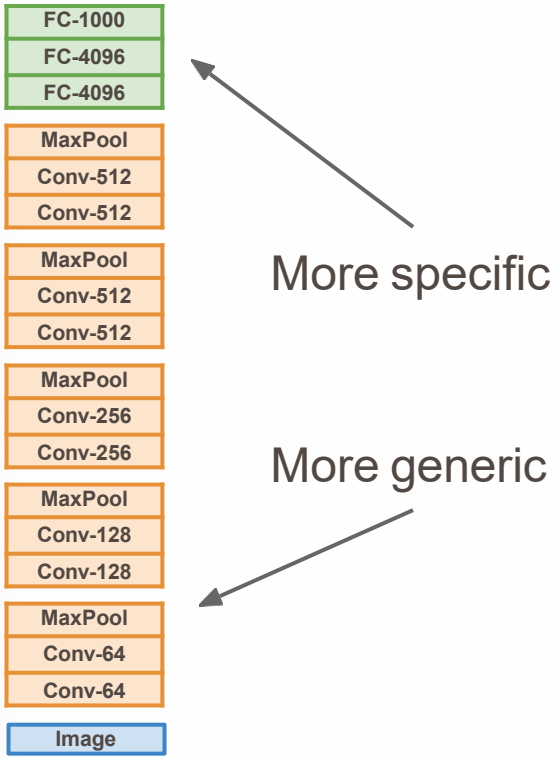


2. Small Dataset (C classes)

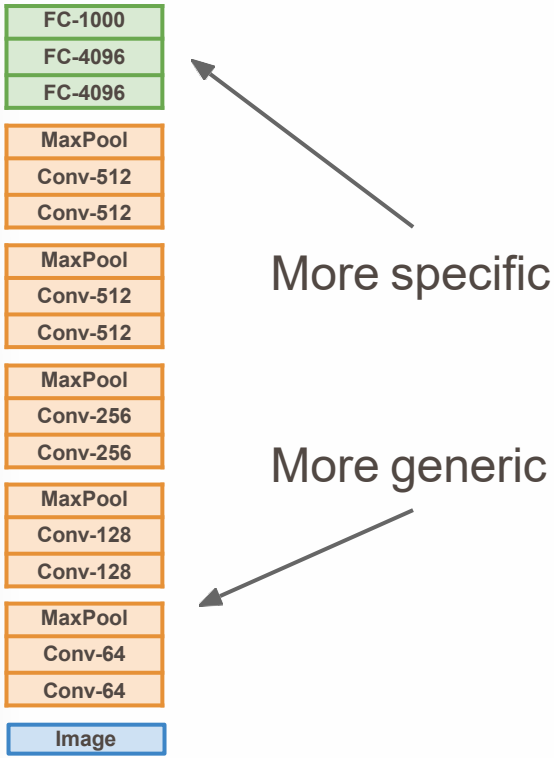


3. Bigger dataset

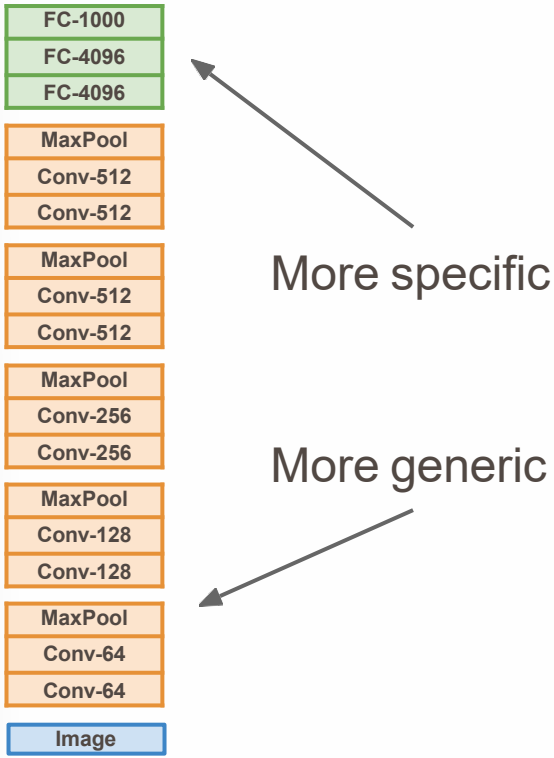




	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?



	very similar dataset	very different dataset
very little data	**Use Linear Classifier on top layer**	**?**
quite a lot of data	**Finetune a few layers**	**?**

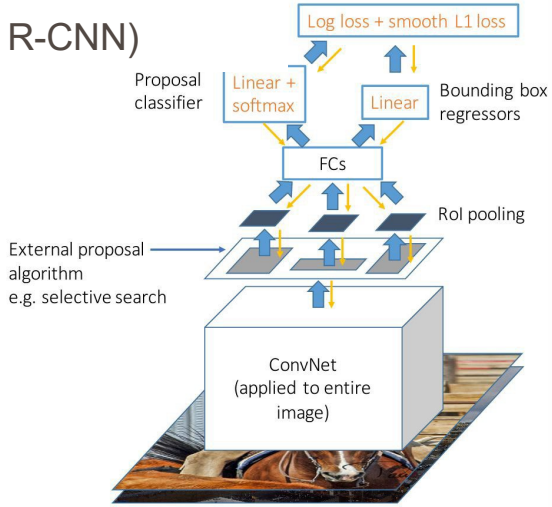


	very similar dataset	very different dataset
very little data	**Use Linear Classifier on top layer**	**You're in trouble… Try linear classifier from different stages**
quite a lot of data	**Finetune a few layers**	**Finetune a larger number of layers**

Transfer learning with CNNs is pervasive…

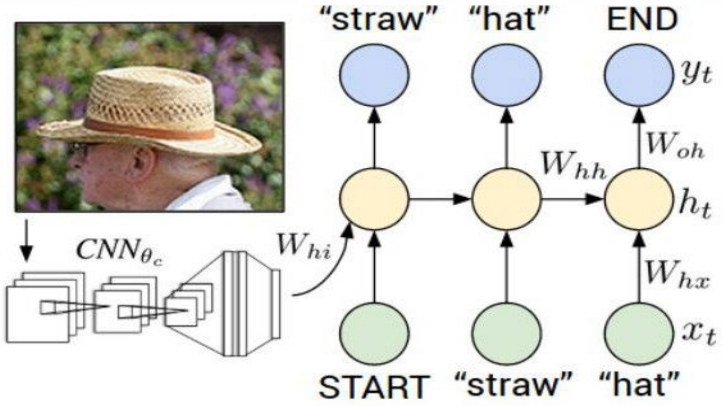
(it's the norm, not an exception)
Object Detection

(Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Image Captioning: CNN + RNN



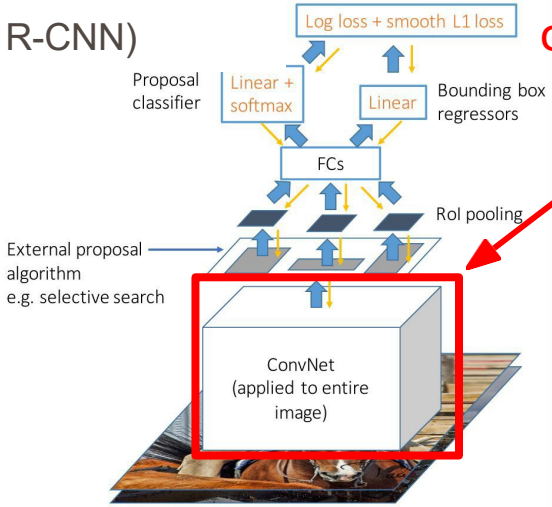
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Transfer learning with CNNs is pervasive…

(it's the norm, not an exception)

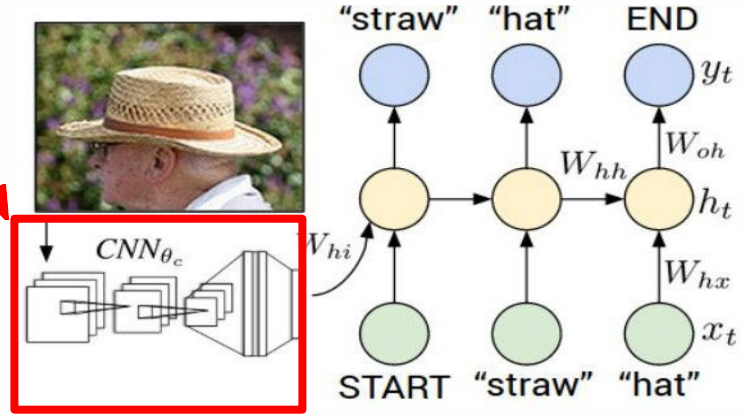
Object Detection

(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

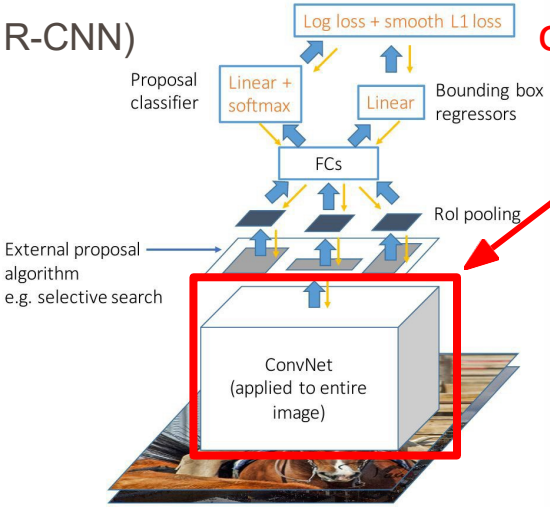
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Transfer learning with CNNs is pervasive…

(it's the norm, not an exception)

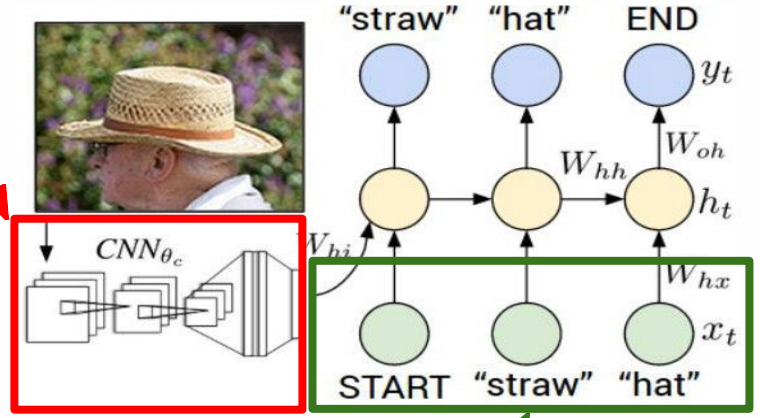
Object Detection

(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN

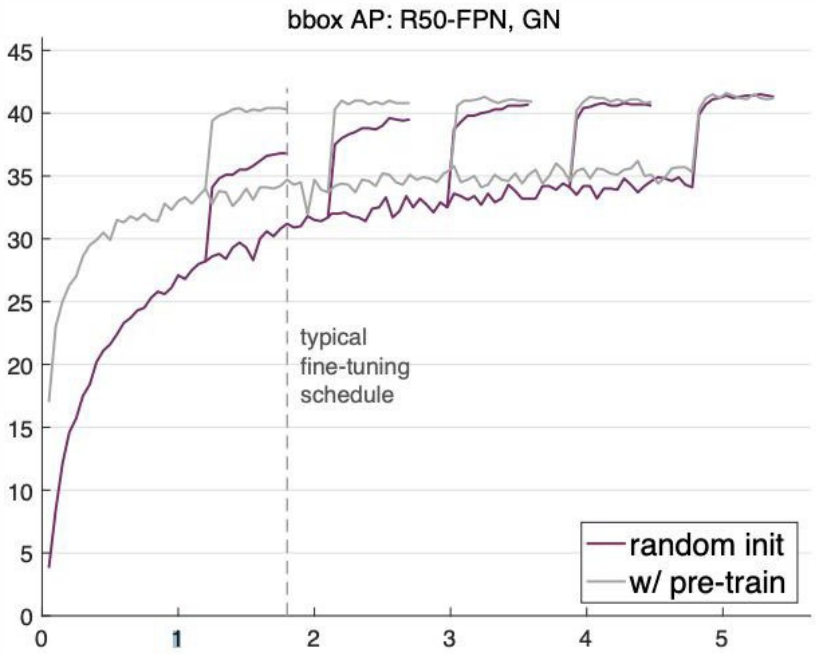


Word vectors pretrained with word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Transfer learning with CNNs is pervasive…
But recent results show it might not always be necessary!



He et al, “Rethinking ImageNet Pre-training”, arXiv 2018

Takeaway for your projects and beyond:

- Have some dataset of interest but it has < ~1M images?
- Find a very large dataset that has similar data, train a big ConvNet there
- Transfer learn to your dataset
- Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

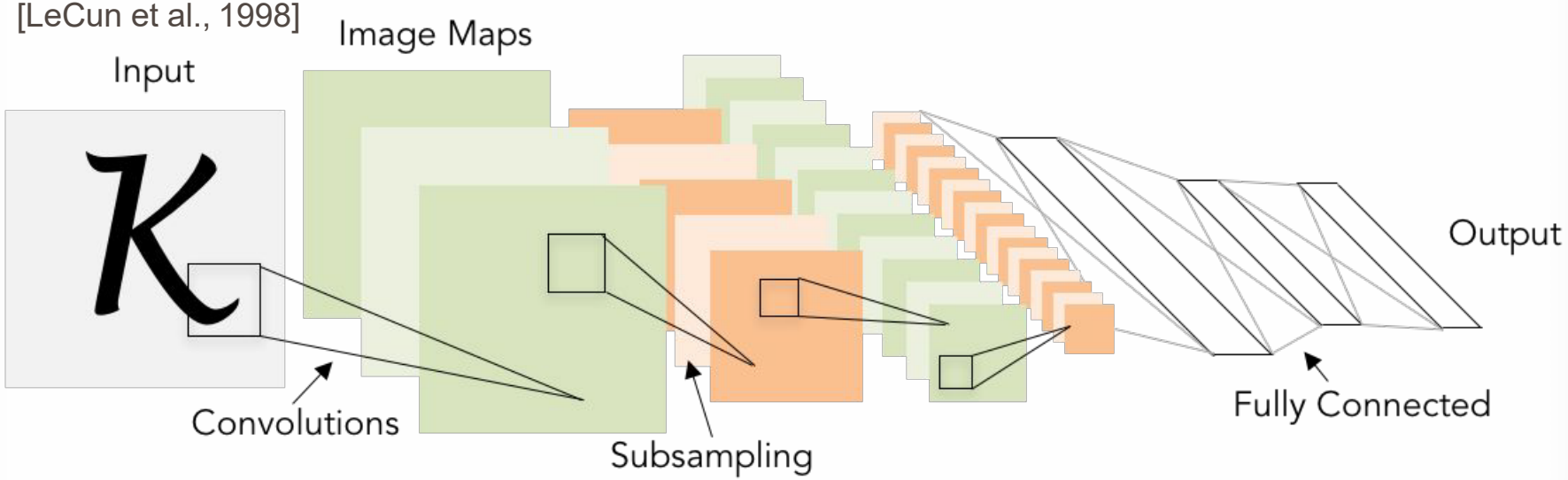
- TensorFlow: https://github.com/tensorflow/models
- PyTorch: https://github.com/pytorch/vision

Today: CNN Architectures

- Case Studies
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet
- Also....
  - SENet
  - NiN (Network in Network)
  - Wide ResNet
  - ResNeXT
  - DenseNet
  - FractalNet
  - MobileNets
  - NASNet
  - SK/EfficientNet/ViT,…never stop

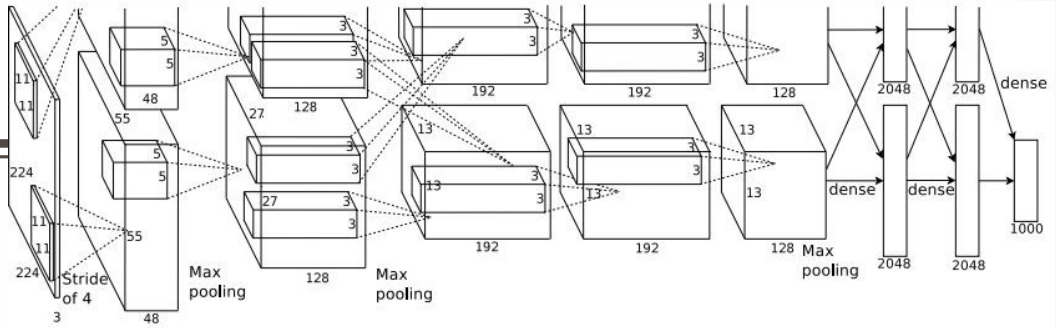
Review: LeNet-5

[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]

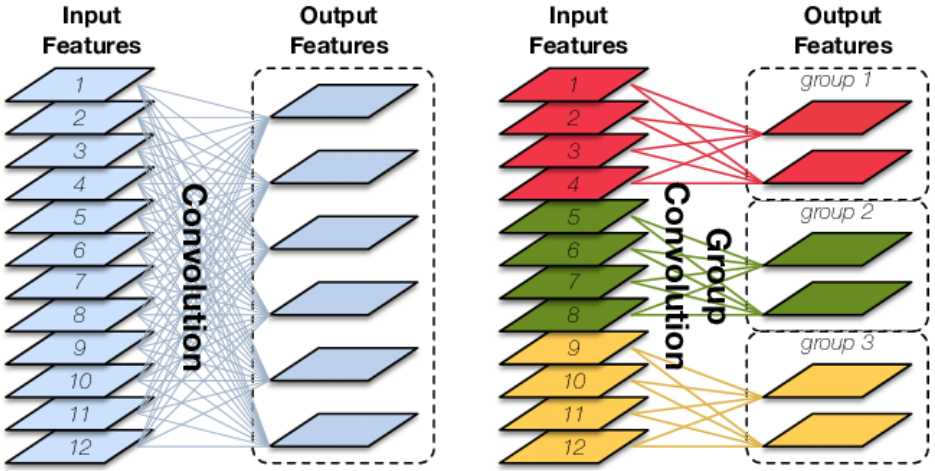
Case Study: AlexNet [Krizhevsky et al. 2012]



Architecture:

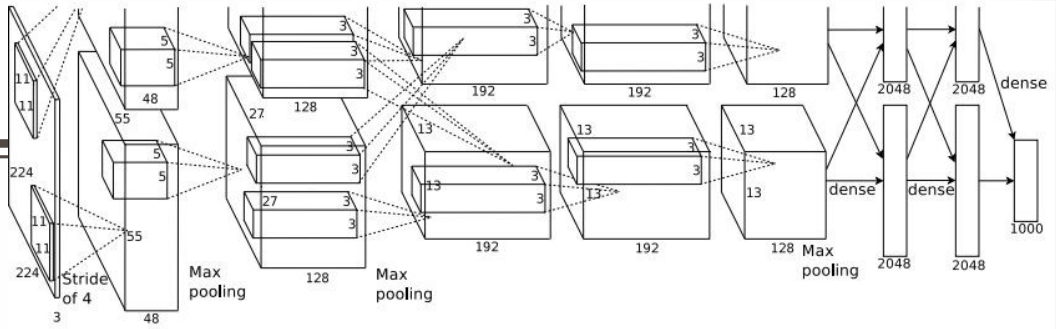
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Case Study: AlexNet [Krizhevsky et al. 2012]



- Convolution:
  - Input feature maps: C∗H∗W ➔ N-channeled output feature maps
  - #parameters: N * C * K * K
- Group convolution:
  - #parameters: N * (C/G) * K * K (G=#groups)

Case Study: AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Review: AlexNet

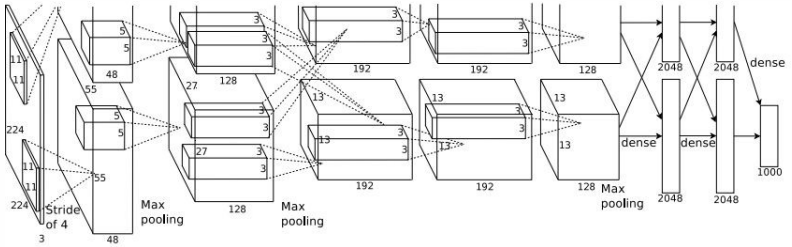
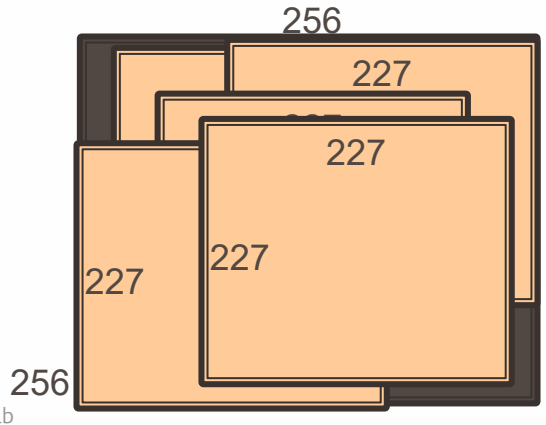


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Input: 227x227x3 images

Q: Why 227*227? Not 256*256?

Data augmentation: Randomly cropping!!
Assumed that the input size = 256*256
We randomly crop the image sized of 227*227



Case Study: AlexNet [Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

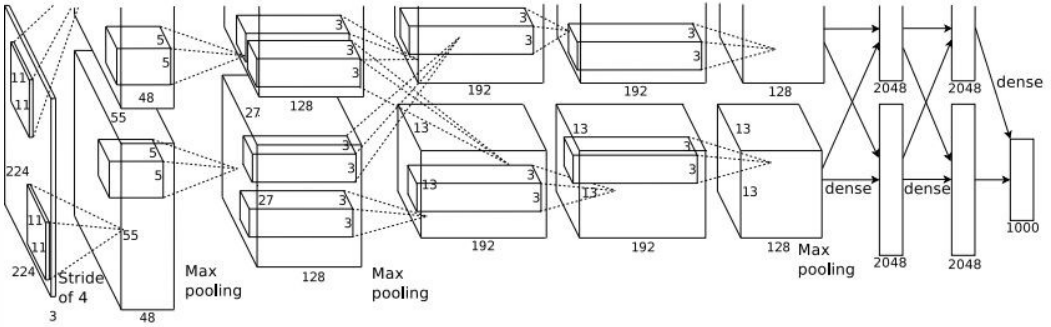
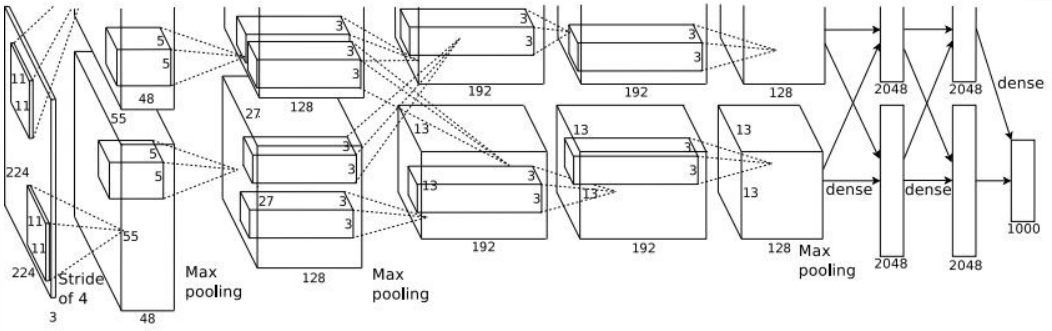


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

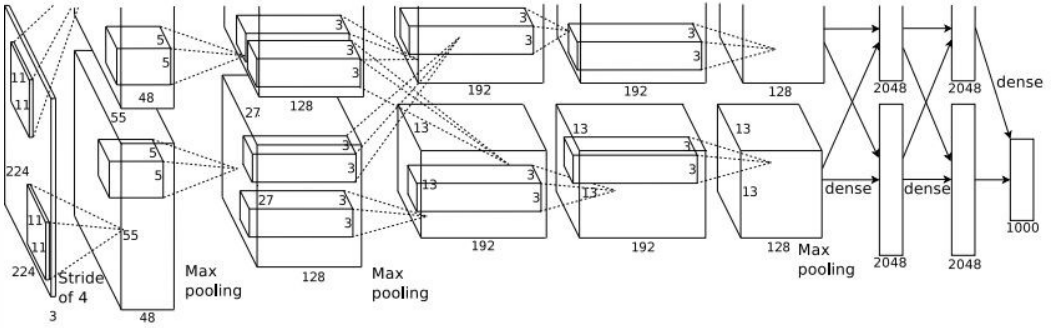
Case Study: AlexNet [Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**



Case Study: AlexNet [Krizhevsky et al. 2012]

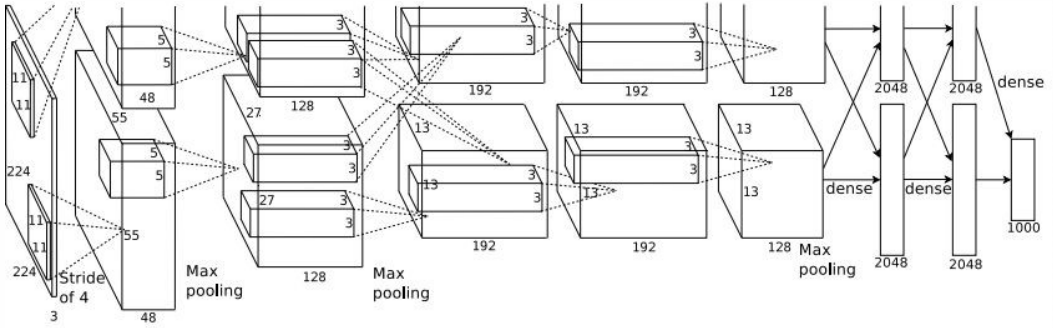


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet [Krizhevsky et al. 2012]



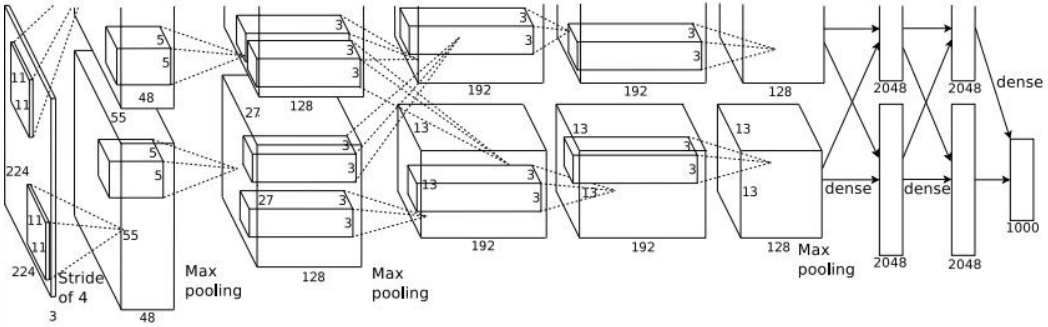
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet [Krizhevsky et al. 2012]

Input: 227x227x3 images After CONV1: 55x55x96



Second layer (POOL1): 3x3 filters applied at stride 2 Output volume: 27x27x96

Parameters: 0!

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet [Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

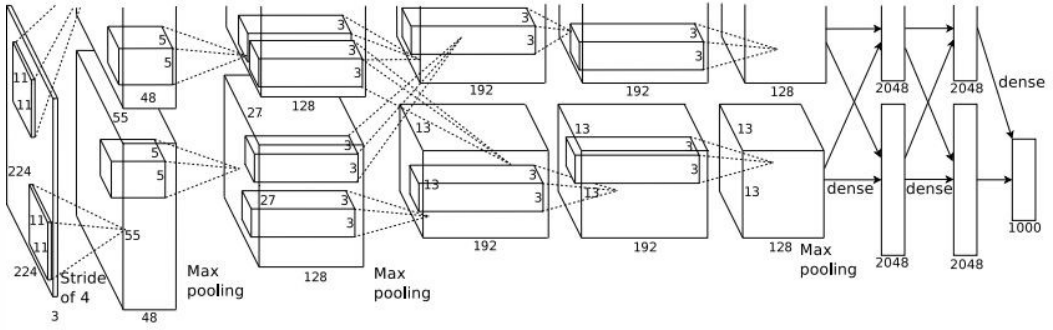
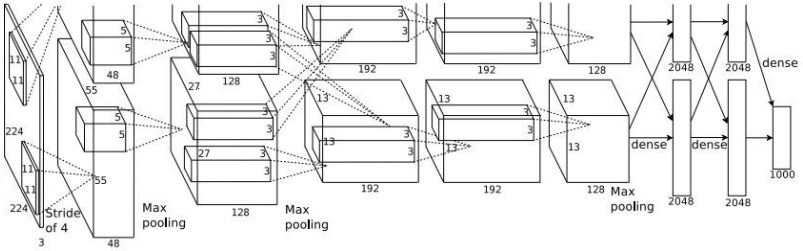


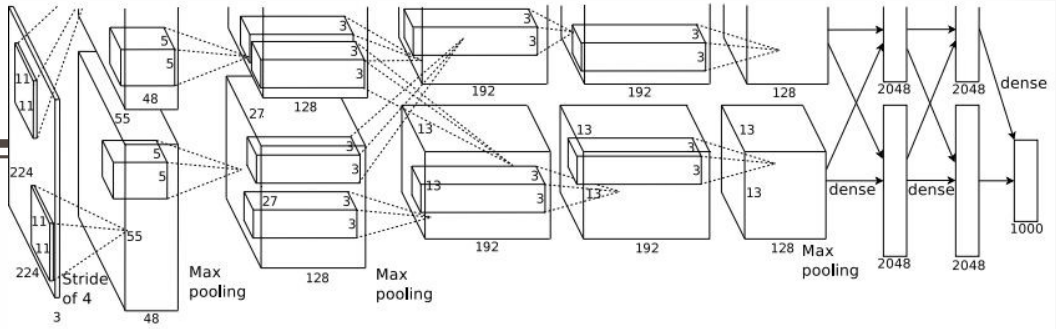
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Case Study: AlexNet [Krizhevsky et al. 2012]

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] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



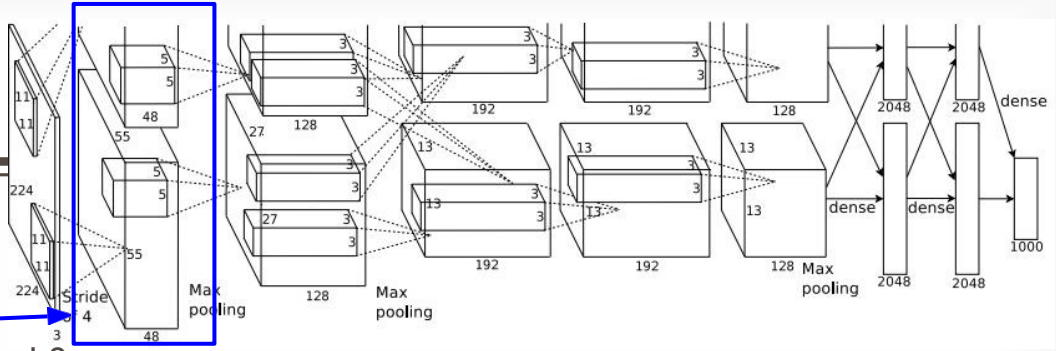
Case Study: AlexNet [Krizhevsky et al. 2012]



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] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
-first use of ReLU
-used Norm layers (not common anymore)
-heavy data augmentation
-dropout 0.5
-batch size 128
-SGD Momentum 0.9
-Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: AlexNet [Krizhevsky et al. 2012]



[55x55x48] x 2

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] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

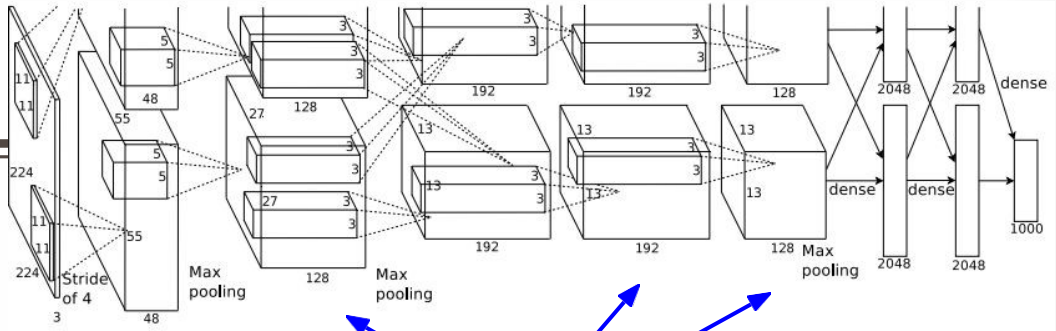
[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

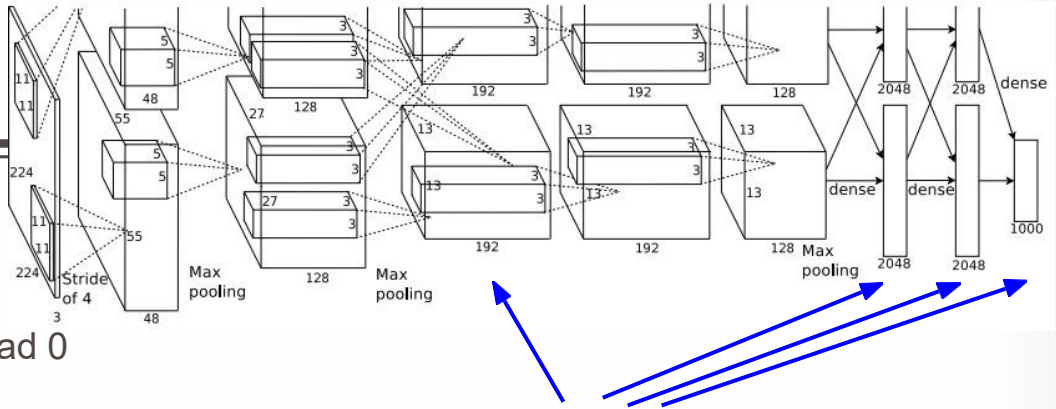
Case Study: AlexNet [Krizhevsky et al. 2012]



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] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

CONV1, CONV2, CONV4, CONV5:
Connections only with feature maps on same GPU

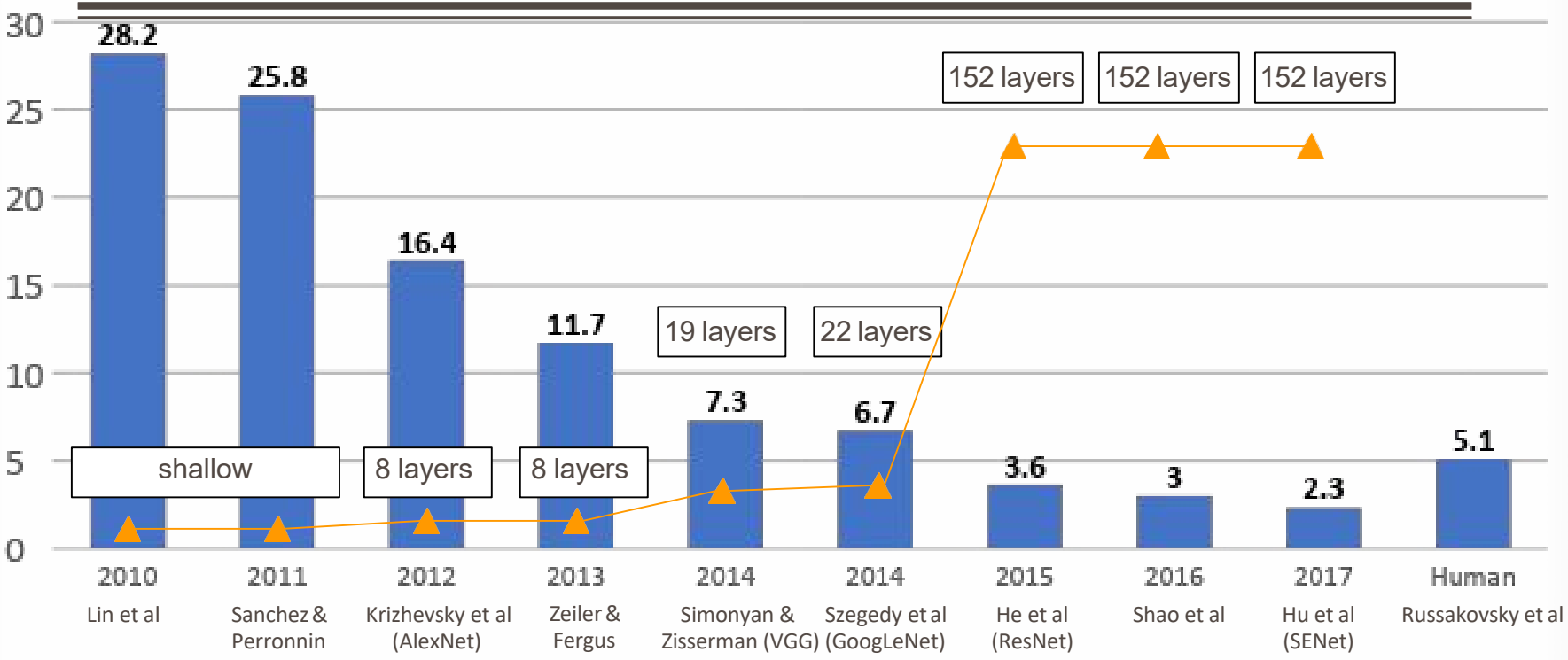
Case Study: AlexNet [Krizhevsky et al. 2012]



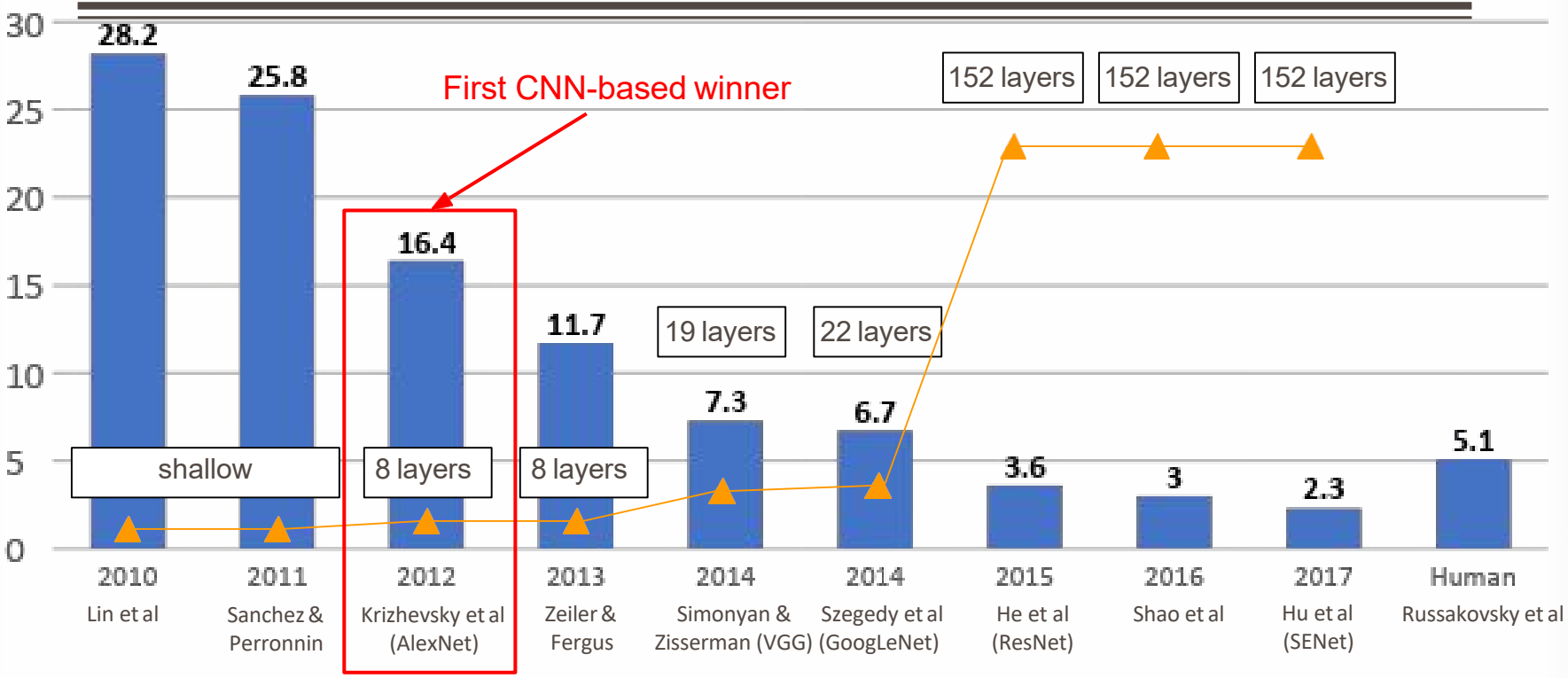
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

CONV3, FC6, FC7, FC8:
Connections with all feature maps in preceding layer, communication across GPUs

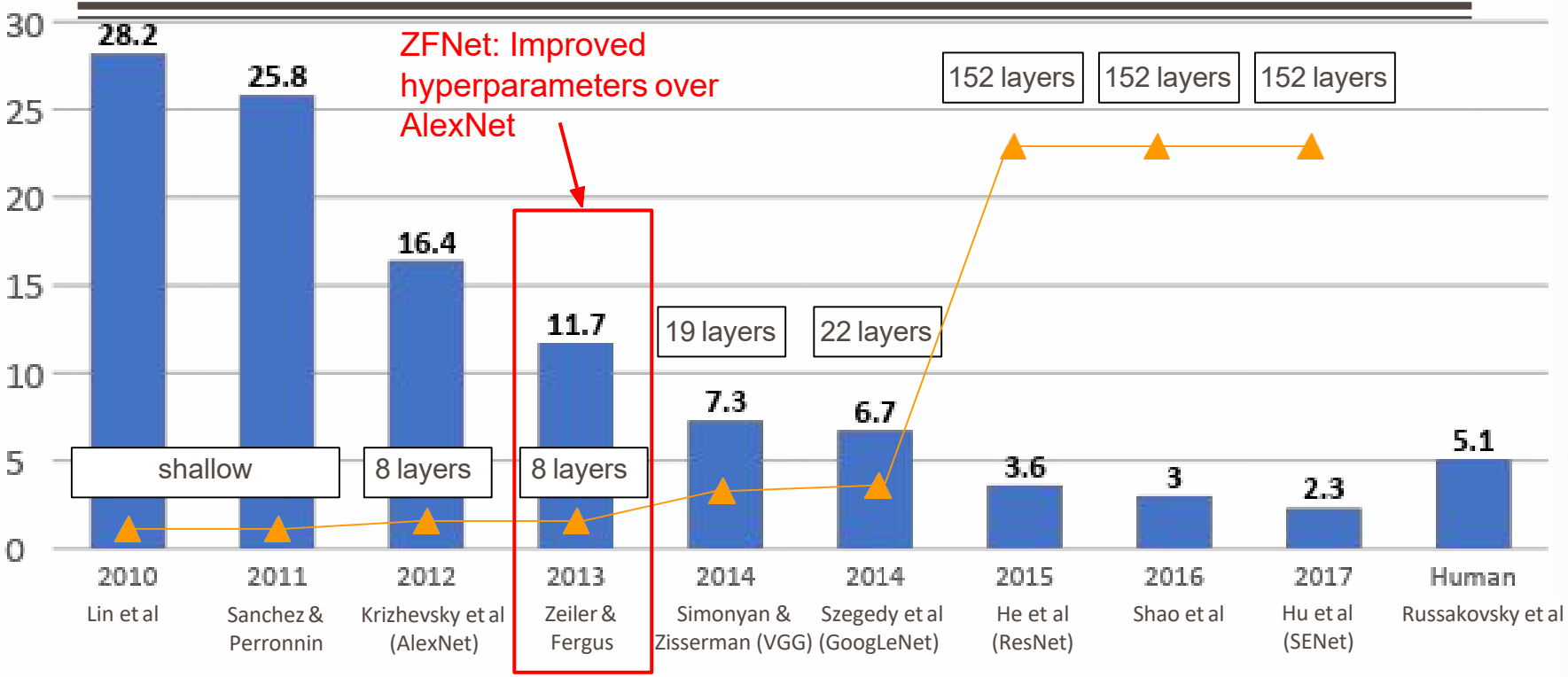
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

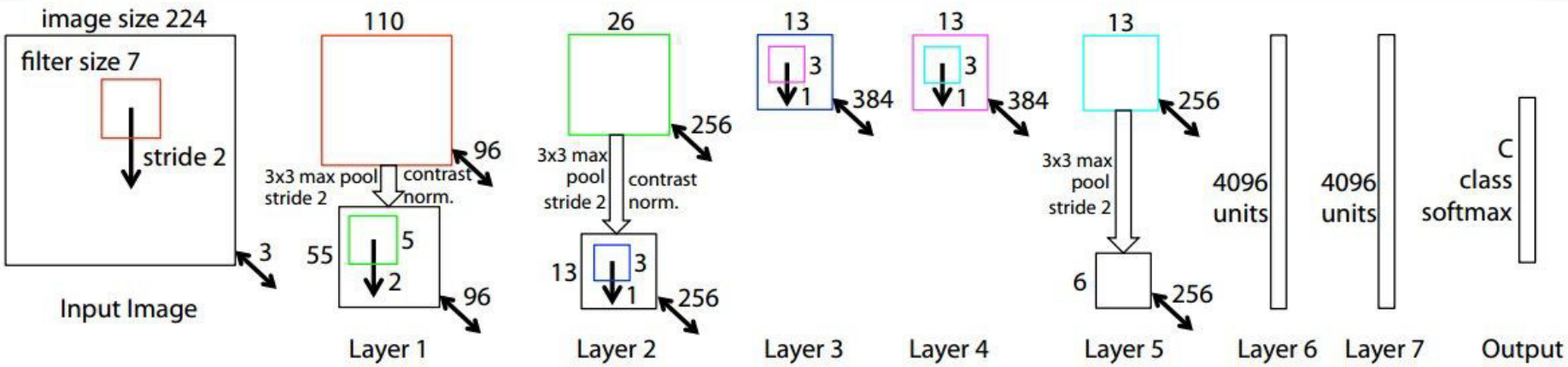


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



[Zeiler and Fergus, 2013]

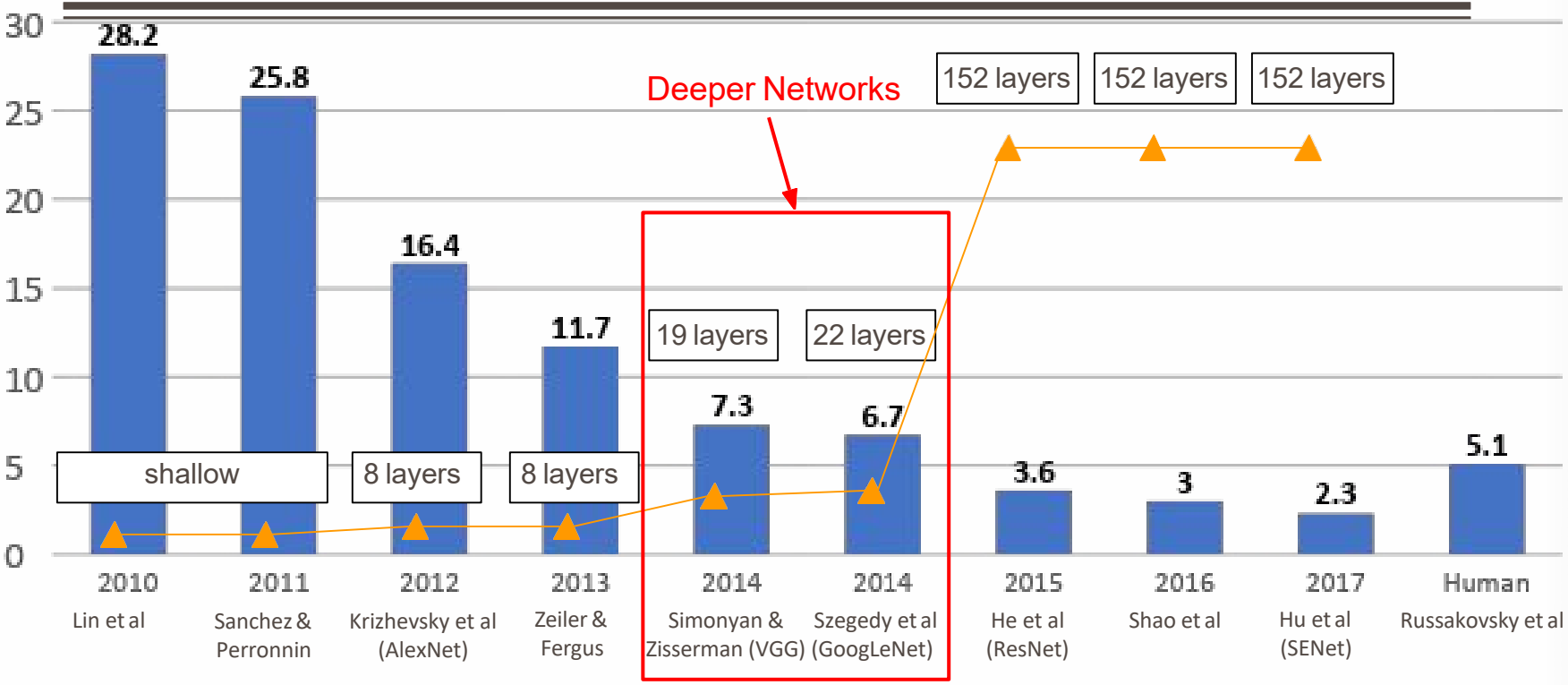
ZFNet



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



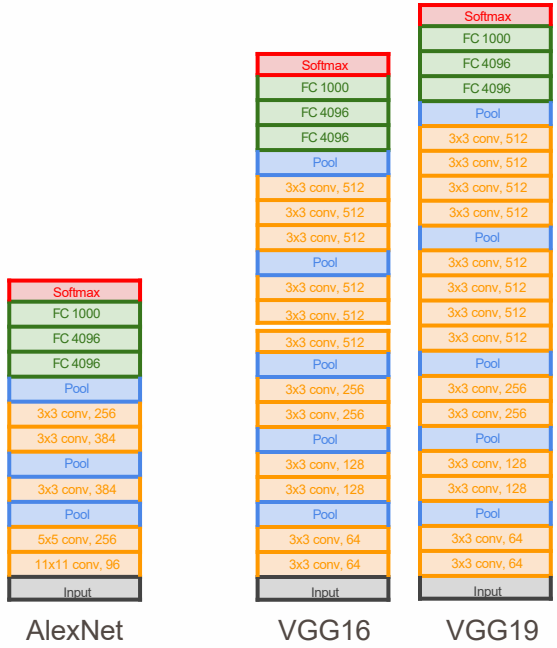
VGGNet
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

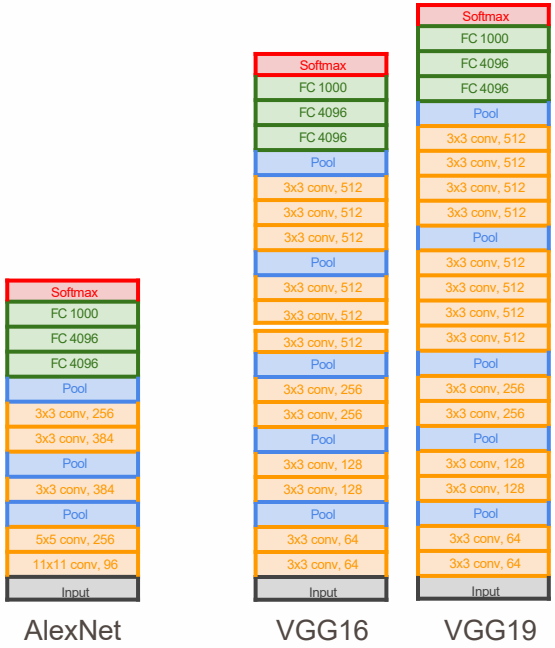
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



VGGNet

Q: Why use smaller filters? (3x3 conv)

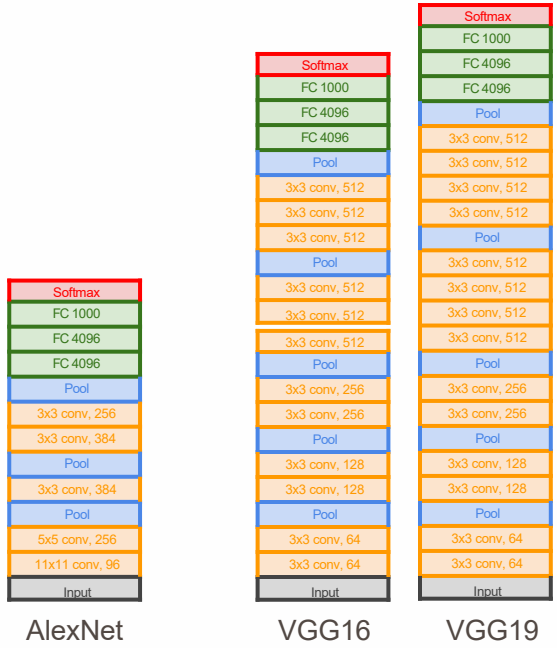


VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

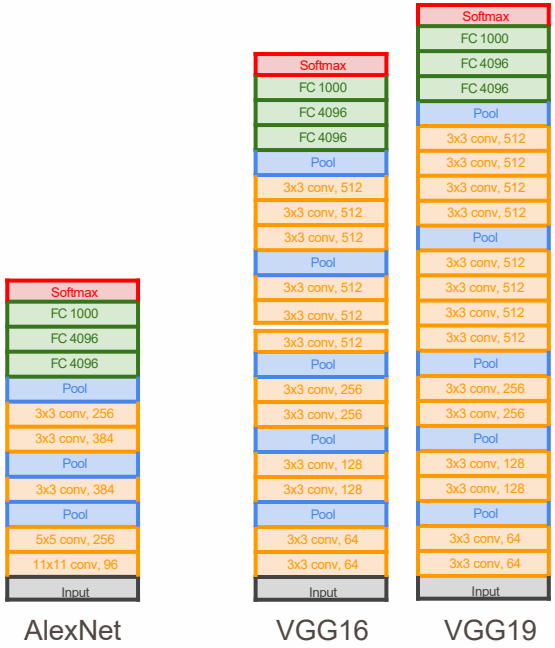


VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

[7x7]



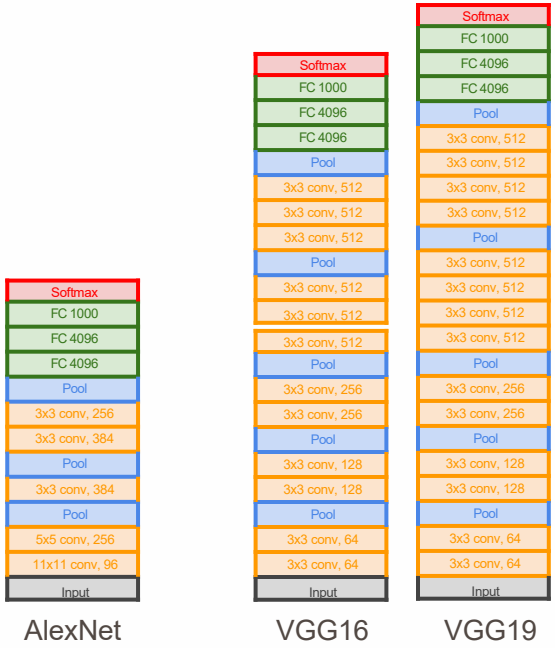
VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

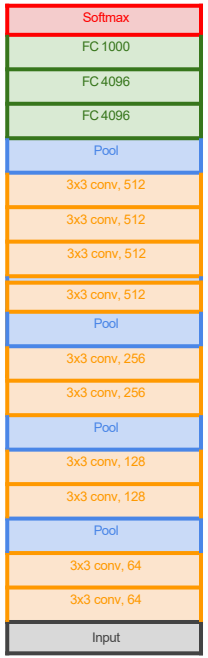
But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
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
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
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
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

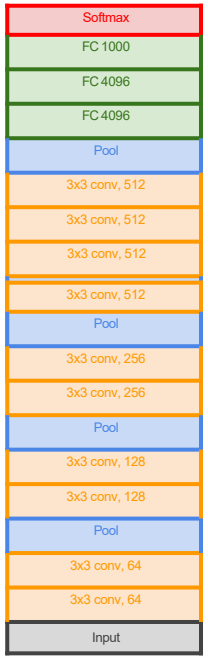


VGG16

#Parameters (not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
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
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
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
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

#Parameters (not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
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
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
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
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

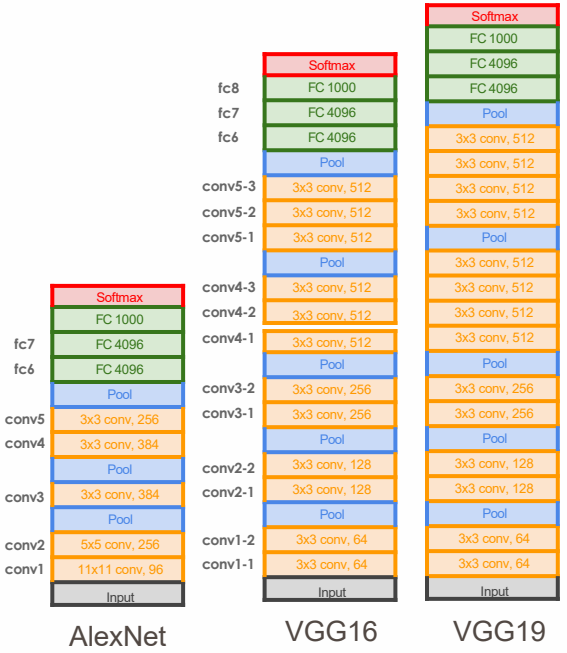
Note:

Most memory is in early CONV

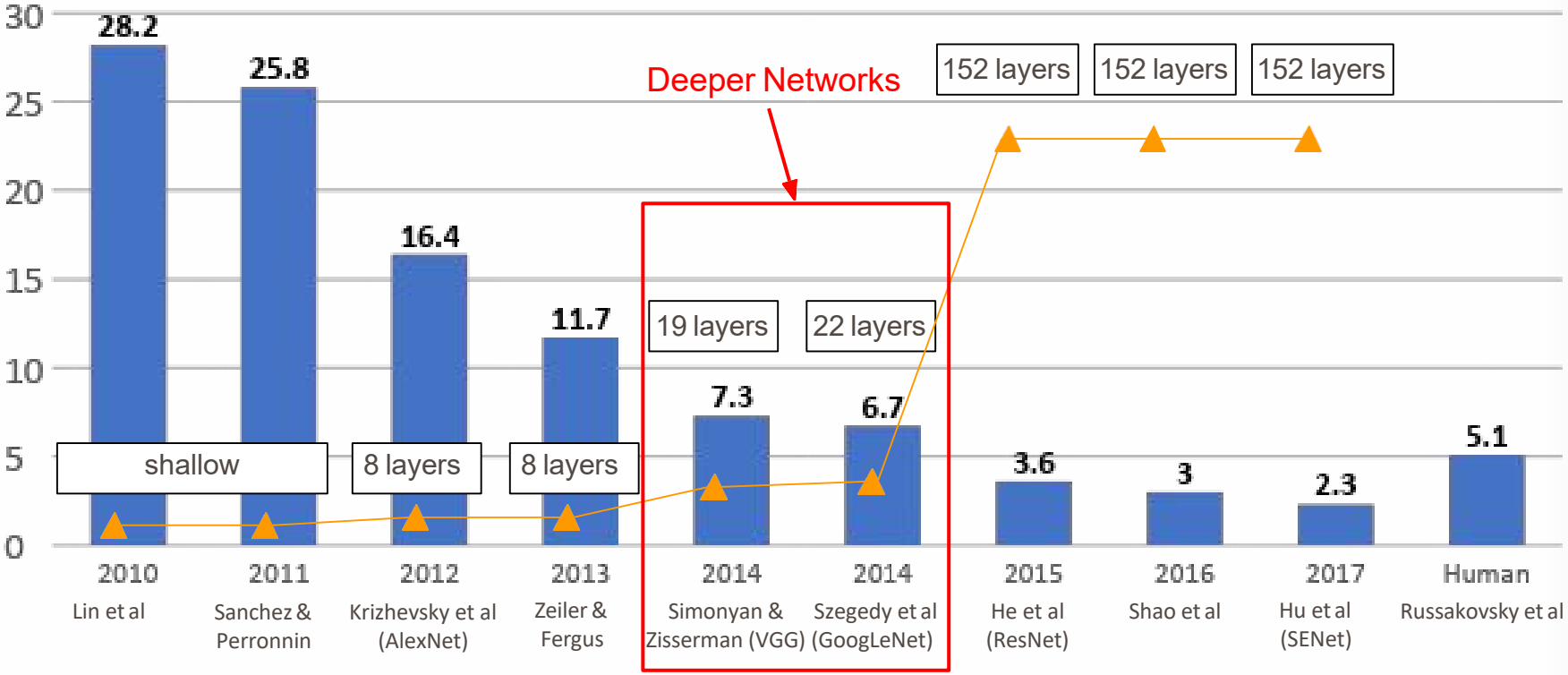
Most params are in late FC

VGGNet

- Details
  - ILSVRC'14 2nd in classification, 1st in localization
  - Similar training procedure as Krizhevsky 2012
  - No Local Response Normalisation (LRN)
  - Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
  - Use ensembles for best results
  - FC7 features generalize well to other tasks



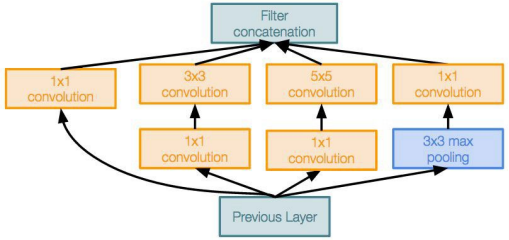
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



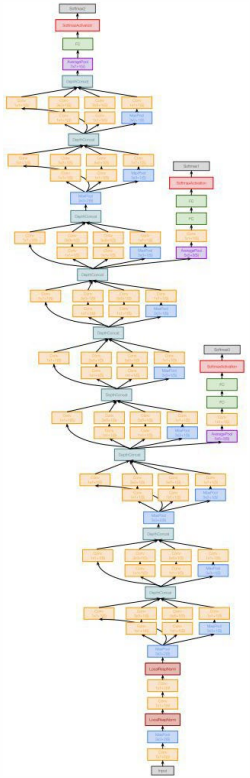
GoogLeNet
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner
  (6.7% top 5 error)

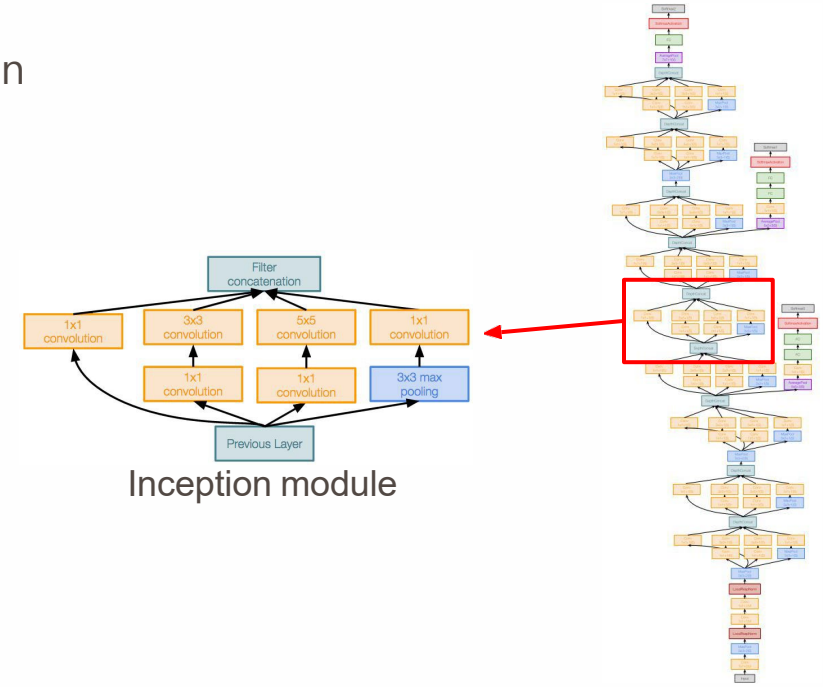


Inception module

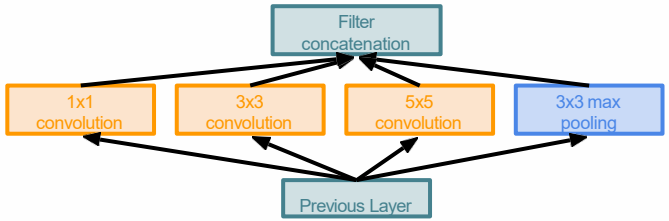


GoogLeNet
[Szegedy et al., 2014]

- "Inception module" : design a good local network topology (network within a network) and then stack these modules on top of each other



GoogLeNet
[Szegedy et al., 2014]



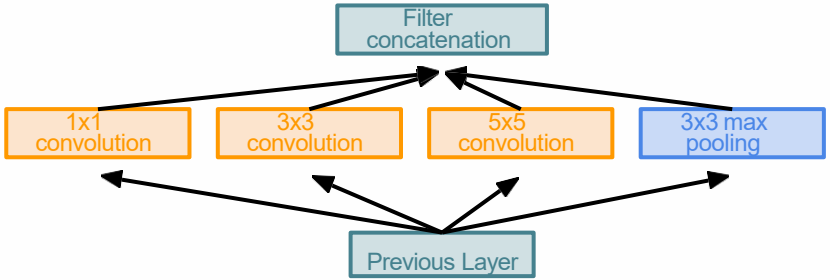
Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

GoogLeNet
[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

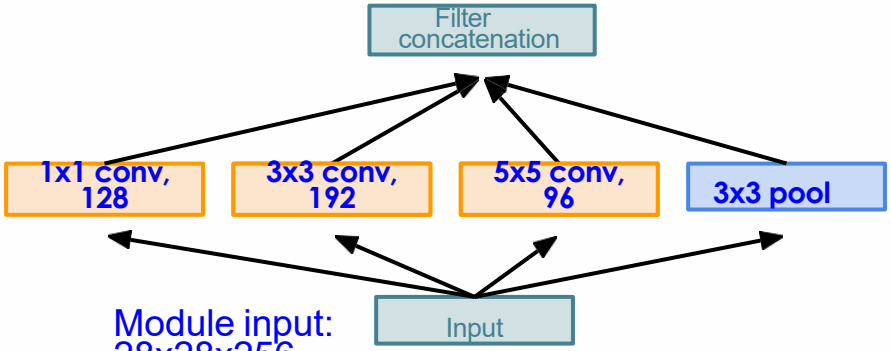
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Q: What is the problem with this?
[Hint: Computational complexity]

GoogLeNet
[Szegedy et al., 2014]

Example:



Module input: 28x28x256

Q: What is the problem with this? [Hint: Computational complexity]

Q1: What is the output size of the 1x1 conv, with 128 filters?

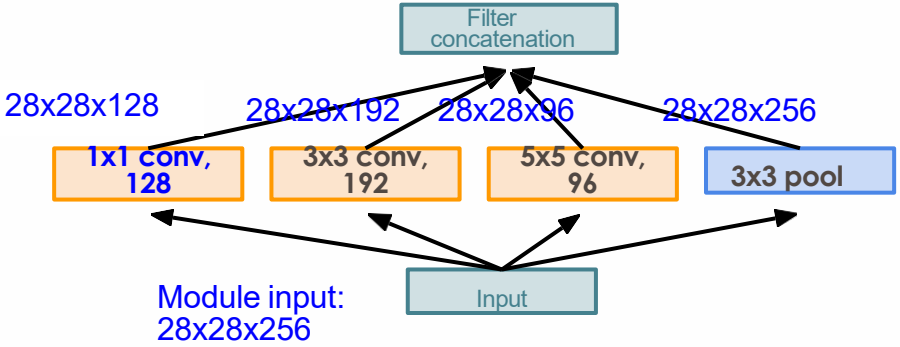
Naive Inception module

GoogLeNet

[Szegedy et al., 2014]

Example:

28x28x(128+192+96+256) = 28x28x672=**529K**



Naive Inception module

Q: What is the problem with this?
[Hint: Computational complexity]

Q1: What is the output size of the 1x1 conv, with 128 filters?

Q2: What are the output sizes of all different filter operations?

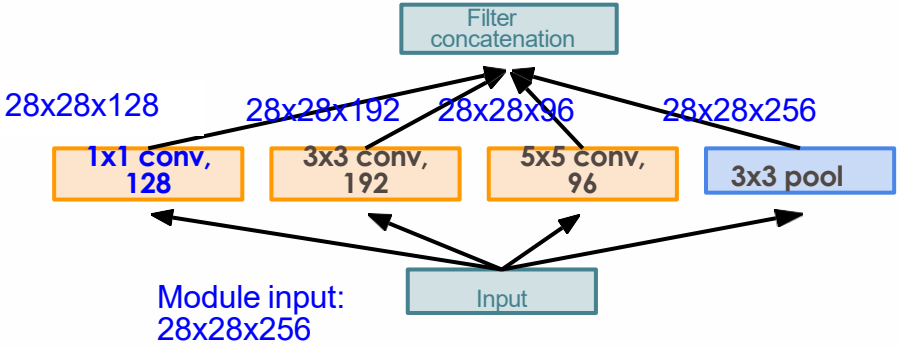
Q3:What is output size after filter concatenation?

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

GoogLeNet

[Szegedy et al., 2014]

Example:



Naive Inception module

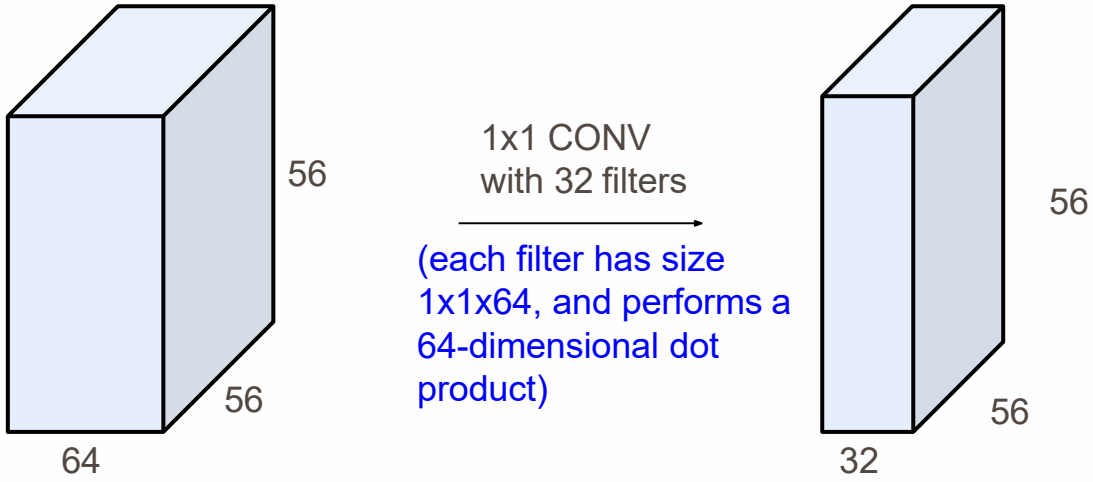
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

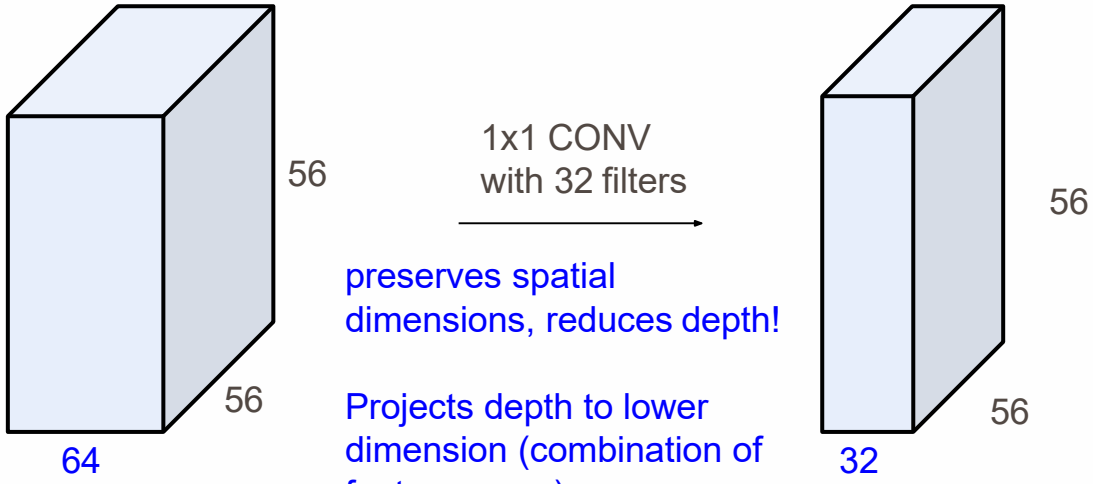
Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

SOLUTION: “BOTTLENECK” LAYERS THAT USE 1X1 CONVOLUTIONS TO REDUCE FEATURE DEPTH

Reminder: 1x1 convolutions

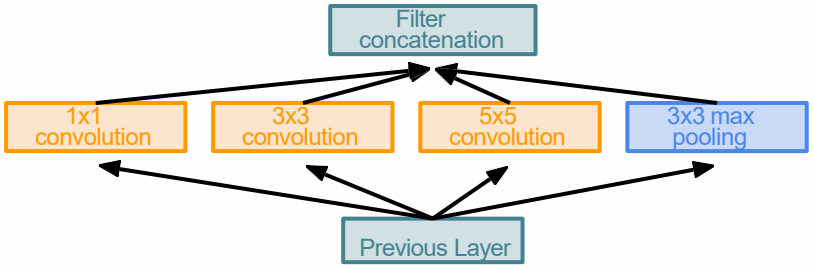


Reminder: 1x1 convolutions

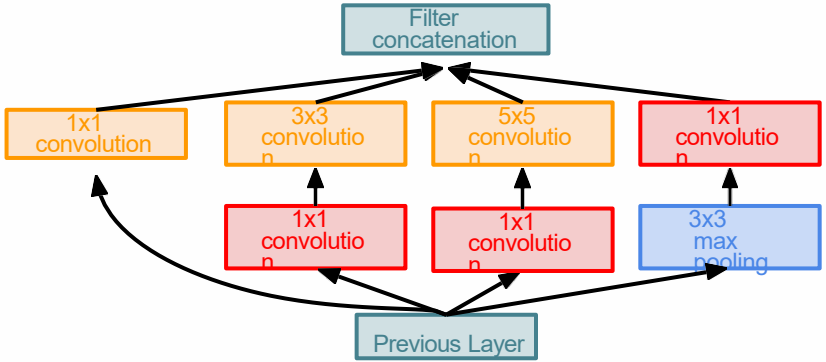


GoogLeNet
[Szegedy et al., 2014]

1x1 conv “bottleneck” layers

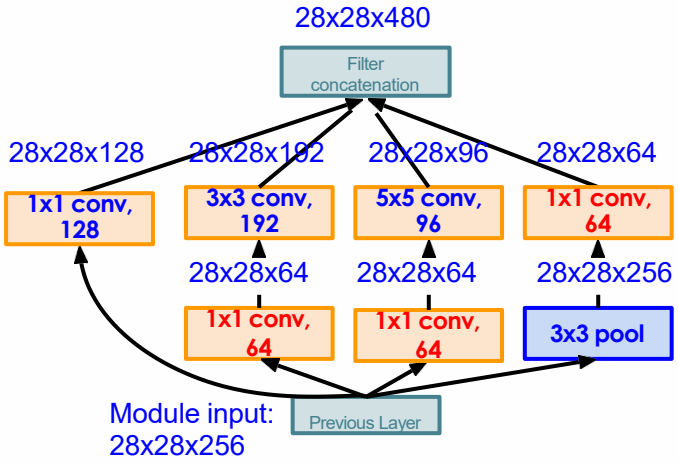


Naive Inception module



Inception module with dimension reduction

GoogLeNet
[Szegedy et al., 2014]



Inception module with dimension reduction

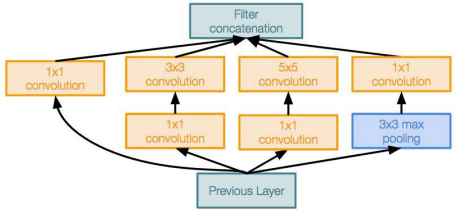
Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

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

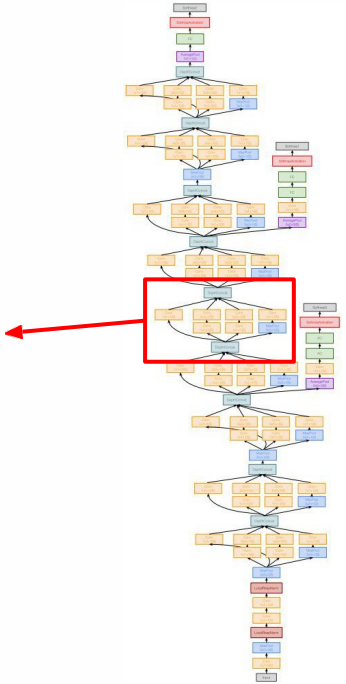
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

GoogLeNet
[Szegedy et al., 2014]

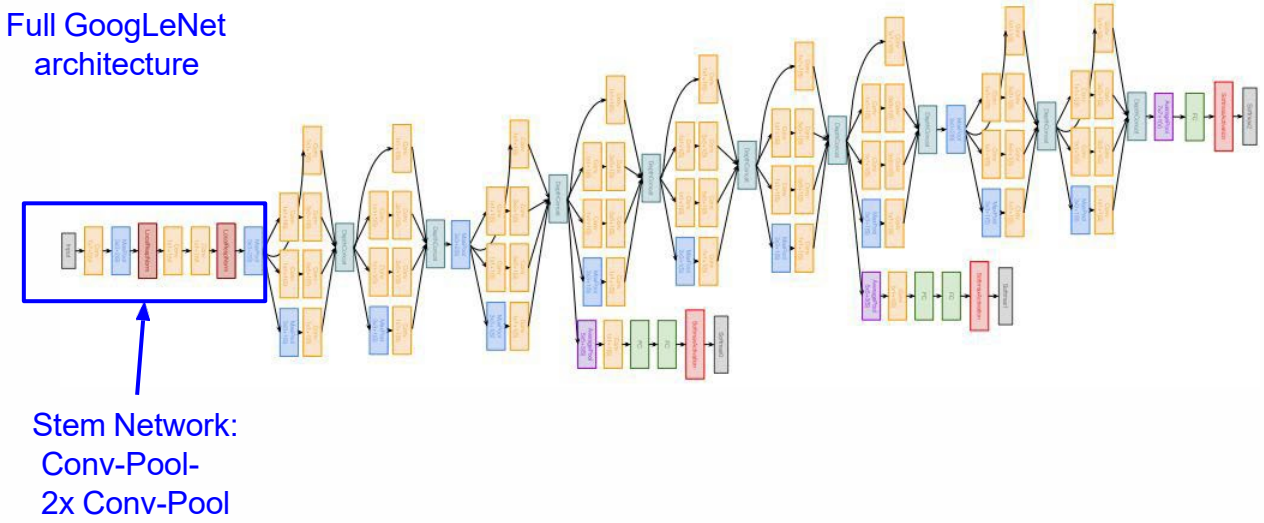
Stack Inception modules with dimension reduction on top of each other



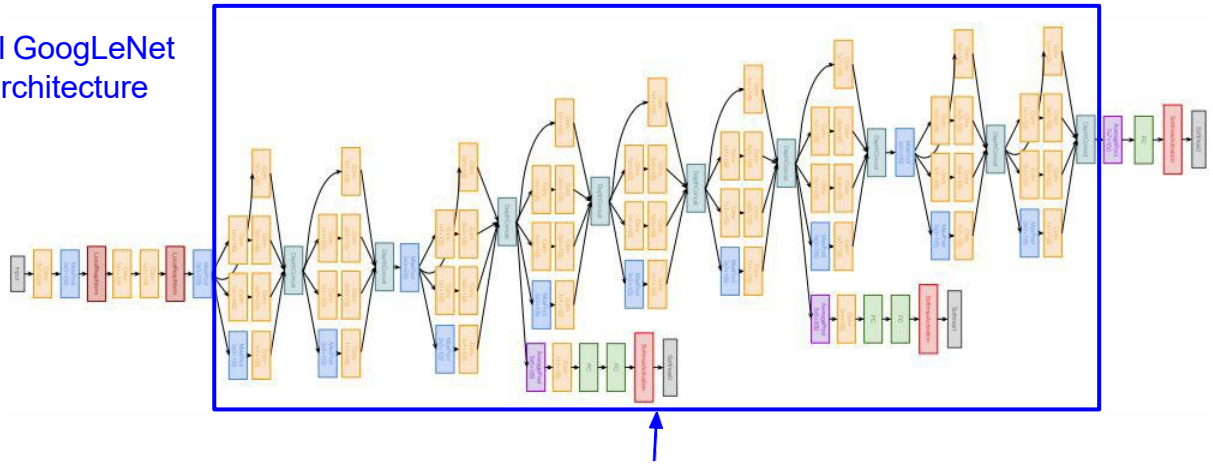
Inception module



GoogLeNet
[Szegedy et al., 2014]

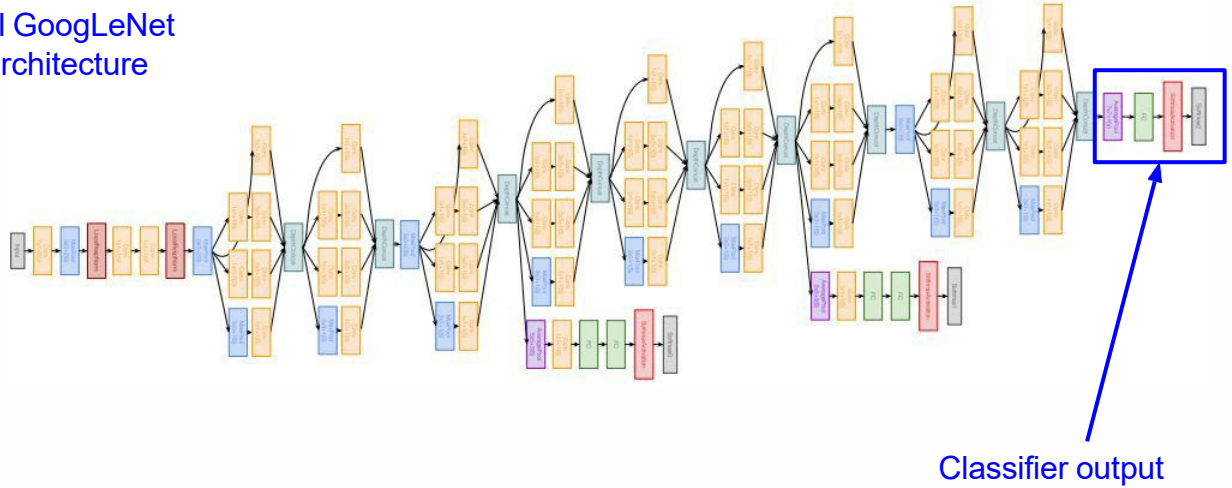


GoogLeNet
[Szegedy et al., 2014]



GoogLeNet
[Szegedy et al., 2014]

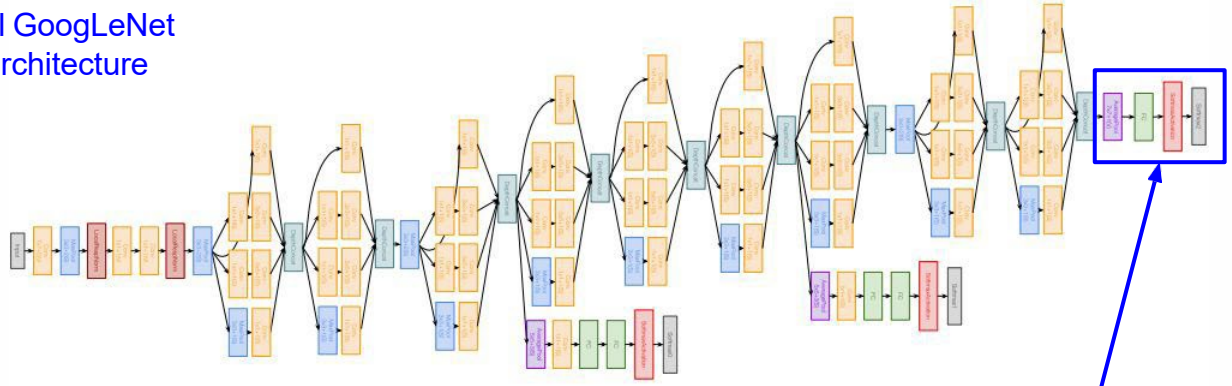
Full GoogLeNet architecture



Classifier output

GoogLeNet
[Szegedy et al., 2014]

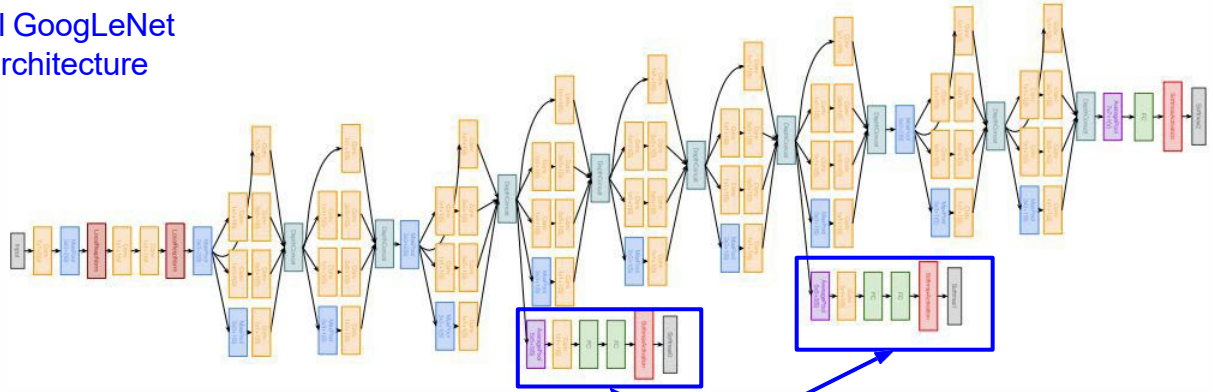
Full GoogLeNet architecture



Classifier output
(removed expensive FC layers!)

GoogLeNet
[Szegedy et al., 2014]

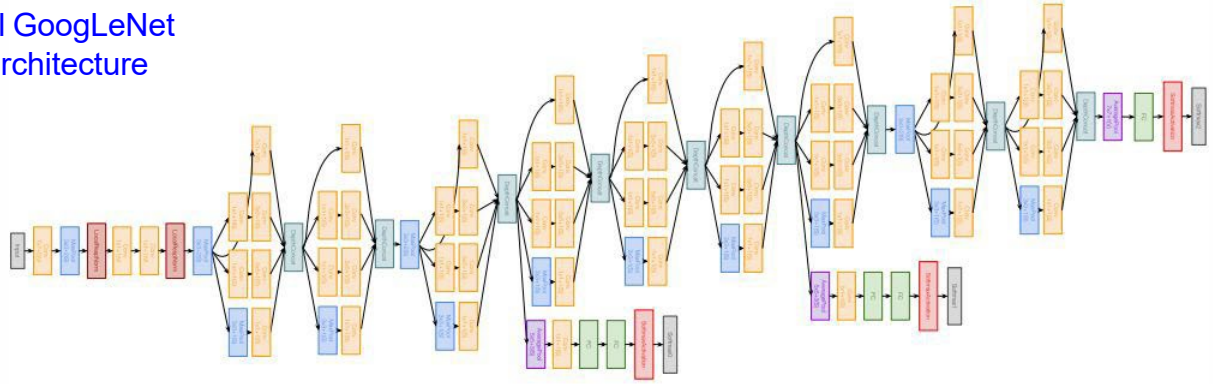
Full GoogLeNet architecture



Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

GoogLeNet
[Szegedy et al., 2014]

Full GoogLeNet architecture

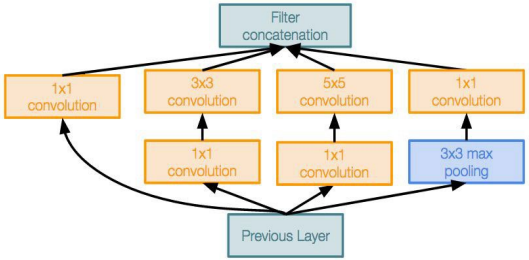


22 total layers with weights (including each parallel layer in an Inception module)

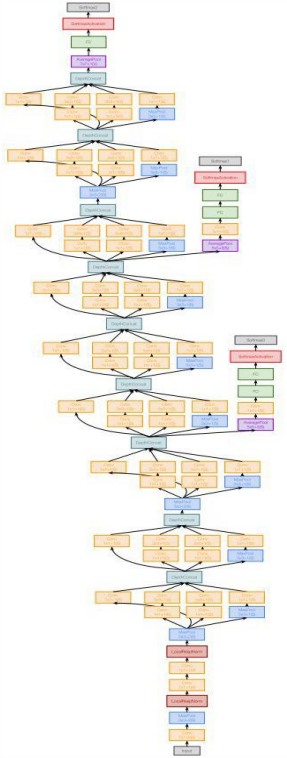
GoogLeNet
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

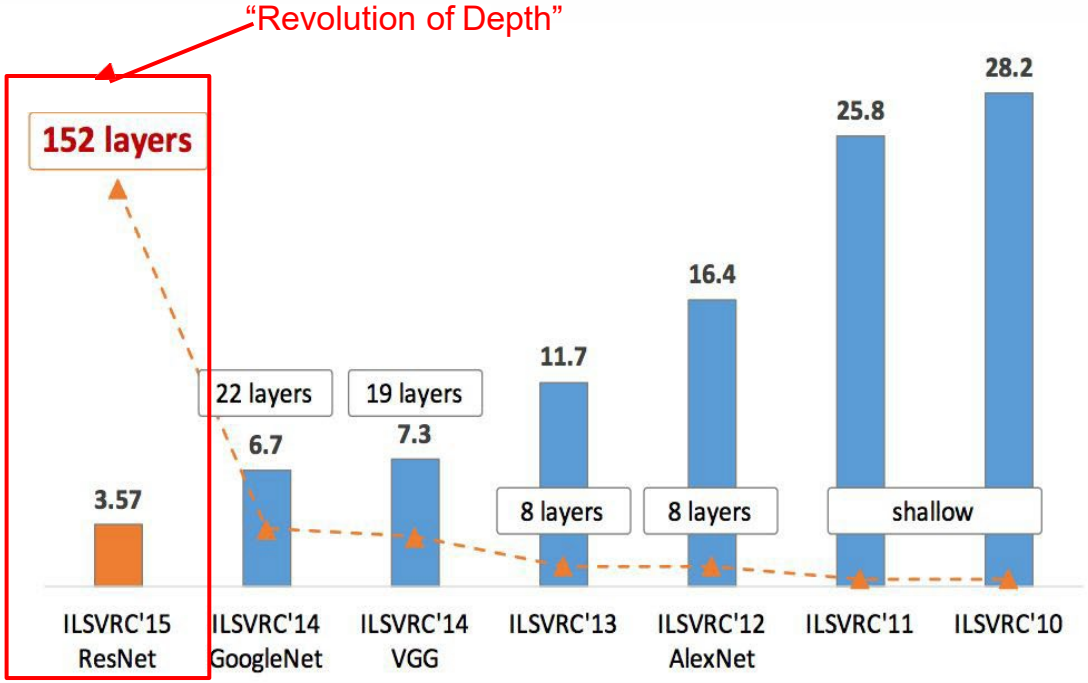
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)



Inception module



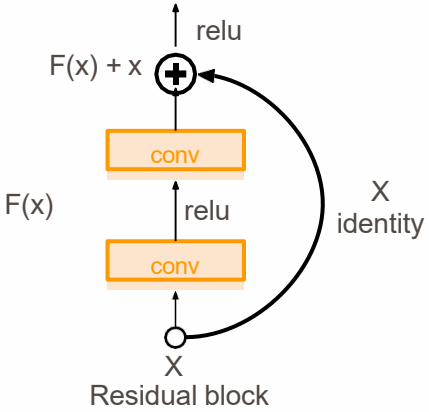
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

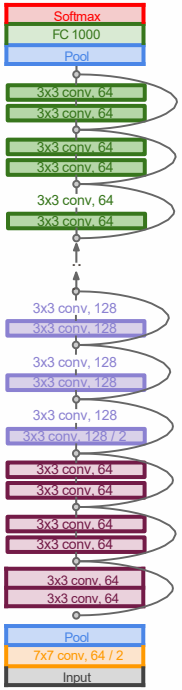


ResNet
[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



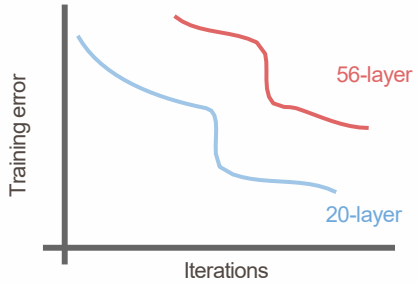


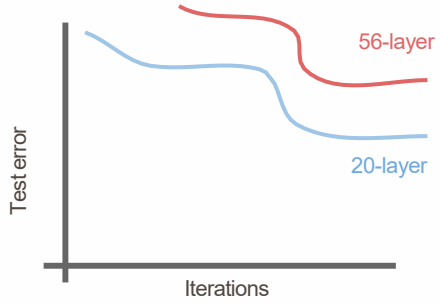
ResNet
[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

ResNet
[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

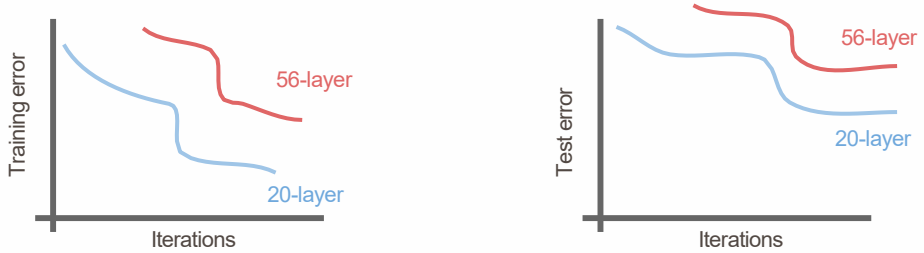




Q: What’s strange about these training and test curves?
[Hint: look at the order of the curves]

ResNet
[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

ResNet
[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

ResNet
[He et al., 2015]

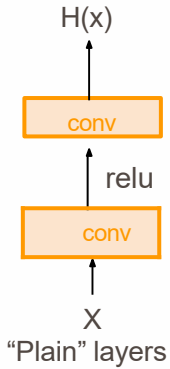
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

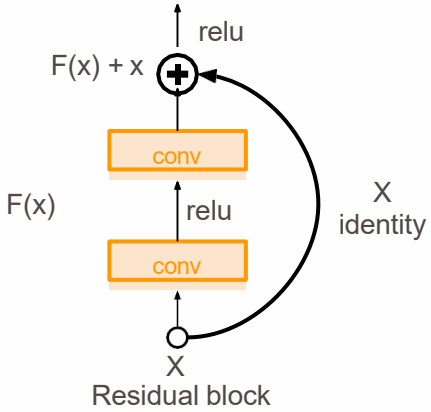
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

ResNet
[He et al., 2015]

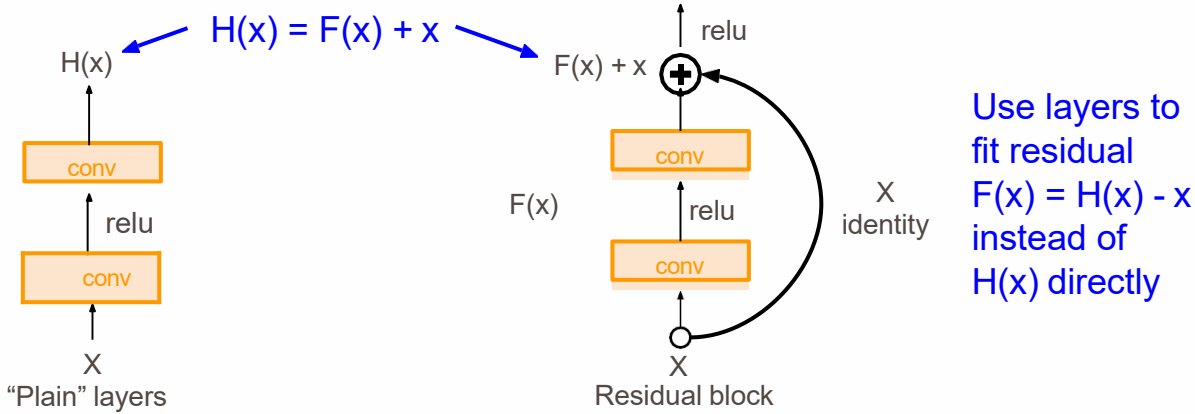
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping





ResNet
[He et al., 2015]

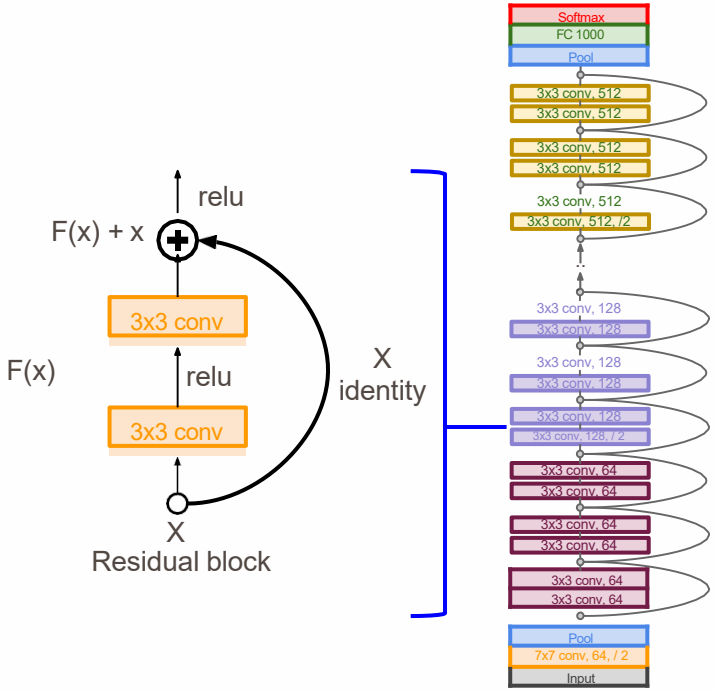
Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

ResNet
[He et al., 2015]

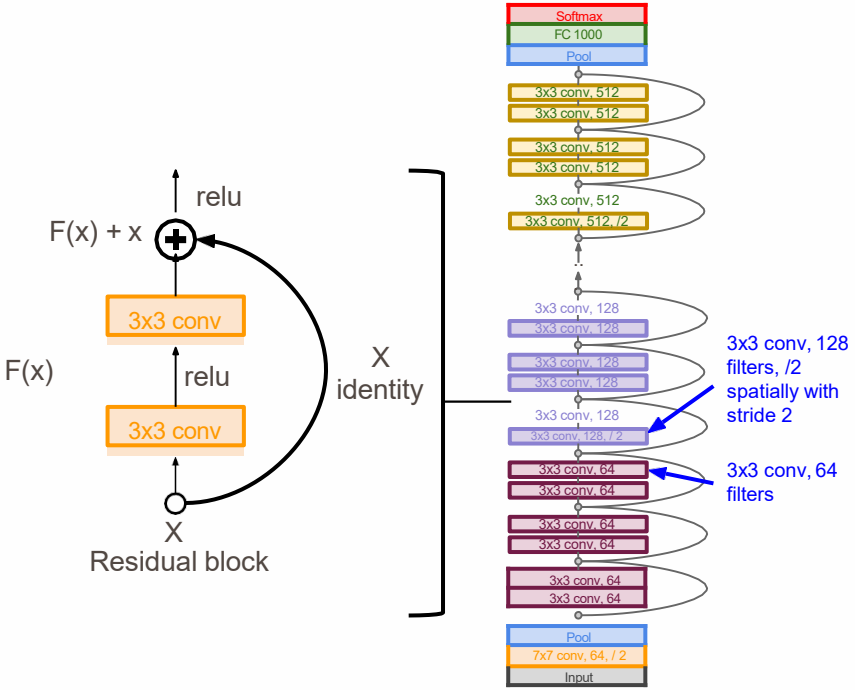
Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



ResNet
[He et al., 2015]

Full ResNet architecture:

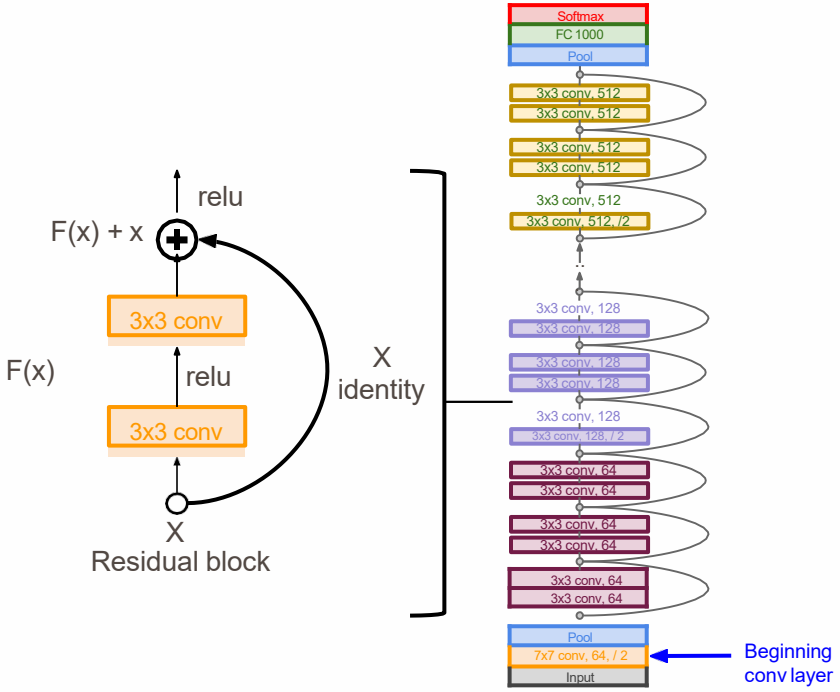
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



ResNet
[He et al., 2015]

Full ResNet architecture:

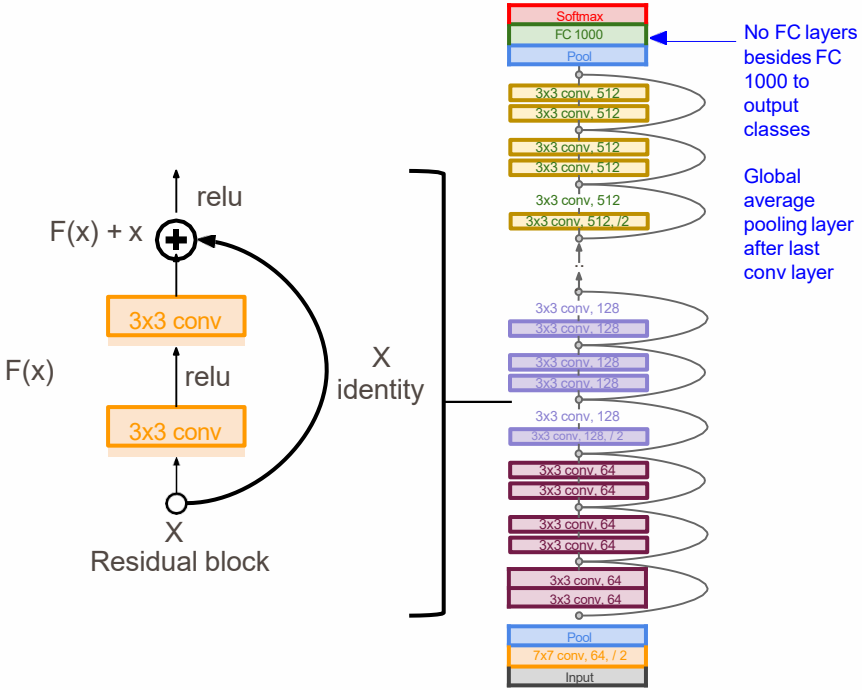
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



ResNet
[He et al., 2015]

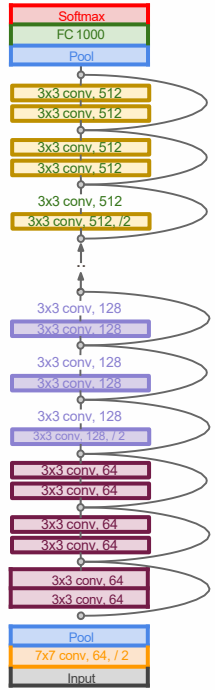
Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



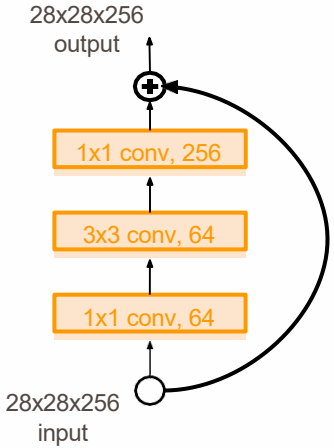
ResNet
[He et al., 2015]

Total depths of 34, 50, 101, or
152 layers for ImageNet



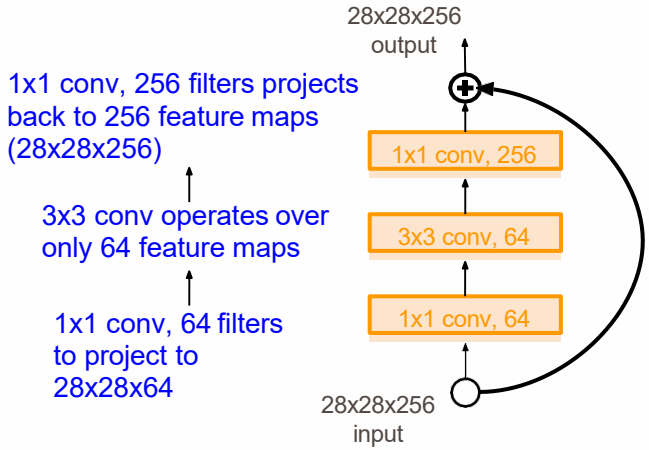
ResNet
[He et al., 2015]

For deeper networks
(ResNet-50+), use “bottleneck”
layer to improve efficiency
(similar to GoogLeNet)



ResNet
[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



ResNet
[He et al., 2015]

- Training ResNet in practice:
  - Batch Normalization after every CONV layer
  - Xavier/2 initialization from He et al.
  - SGD + Momentum (0.9)
  - Learning rate: 0.1, divided by 10 when validation error plateaus
  - Mini-batch size 256
  - Weight decay of 1e-5
  - No dropout used

ResNet
[He et al., 2015]

Experimental Results

- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

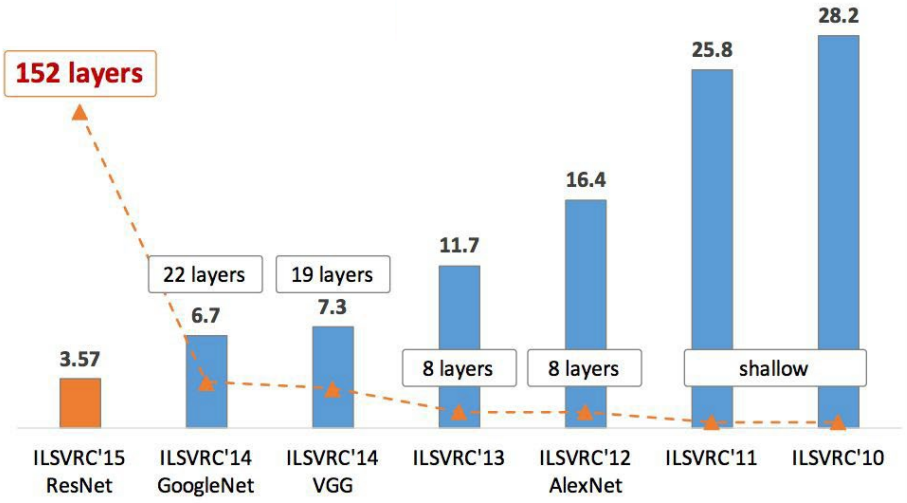
ResNet
[He et al., 2015]

Experimental Results

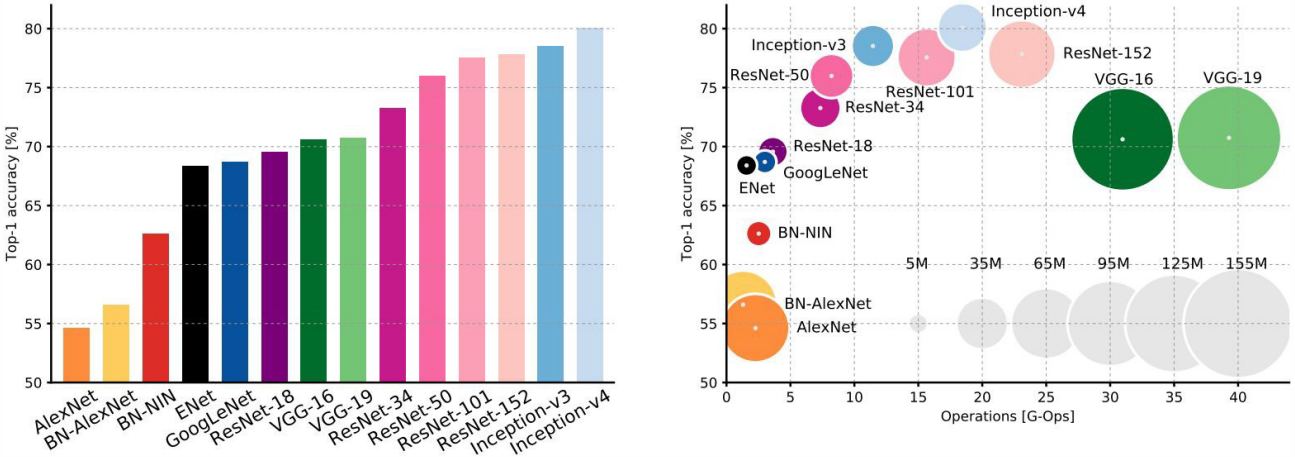
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

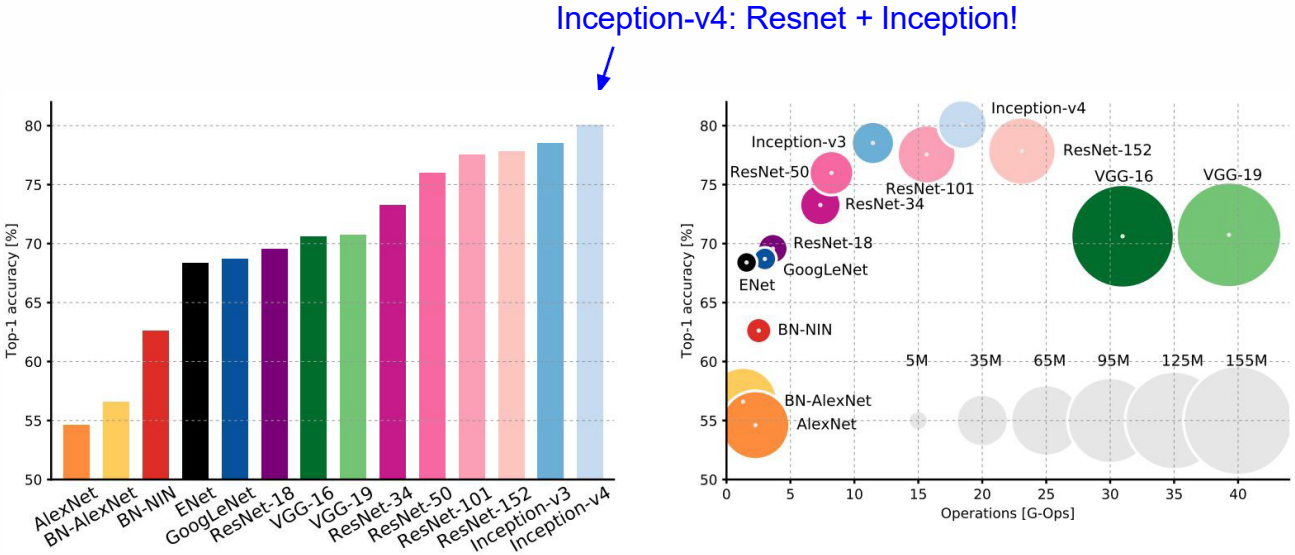


Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

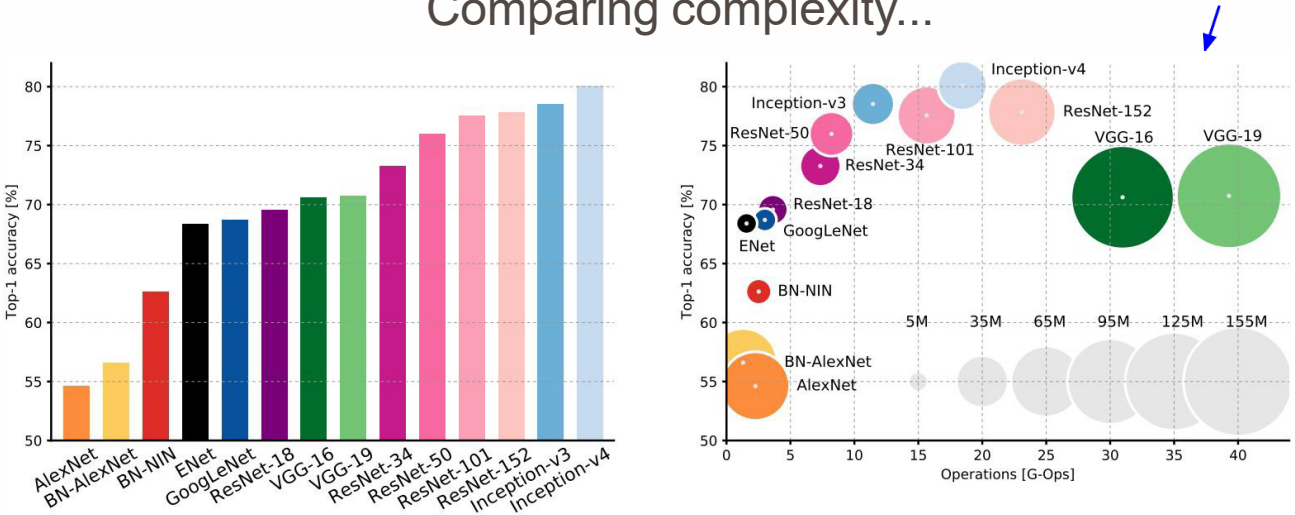
Comparing complexity…



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

VGG: Highest memory, most operations

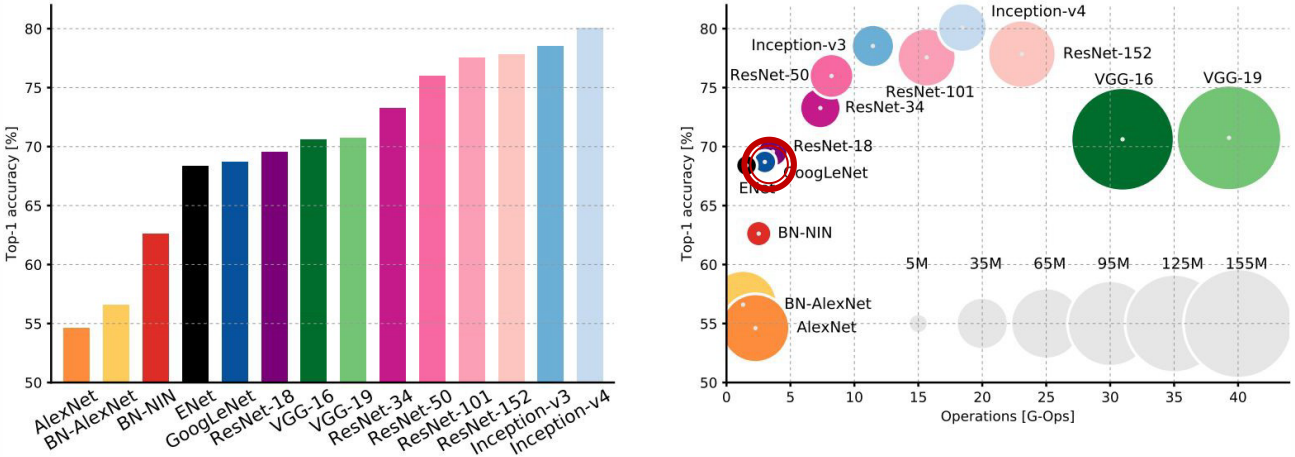
Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

GoogLeNet: most efficient

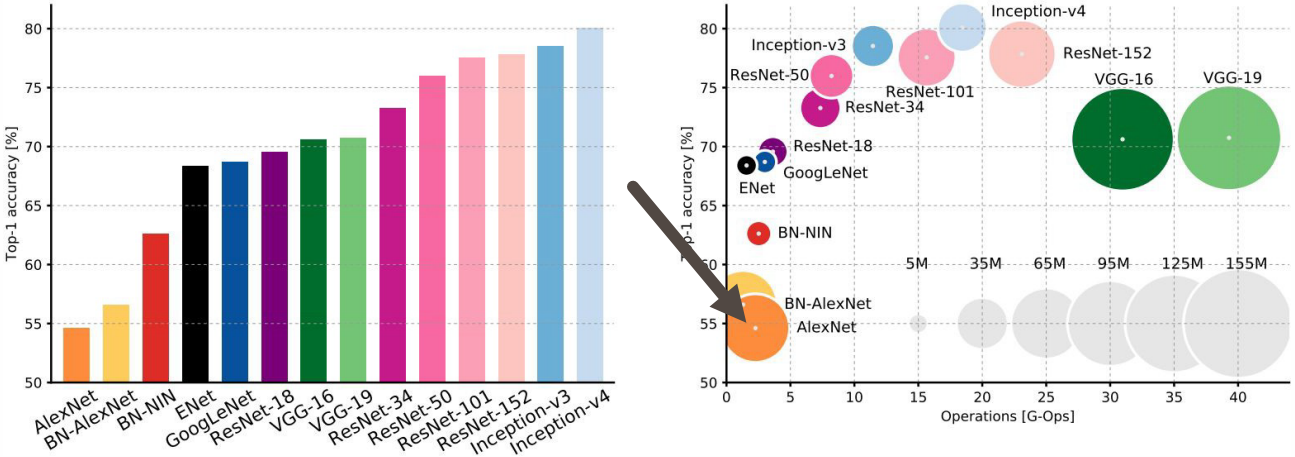
Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

AlexNet:
Smaller compute, still memory heavy, lower accuracy

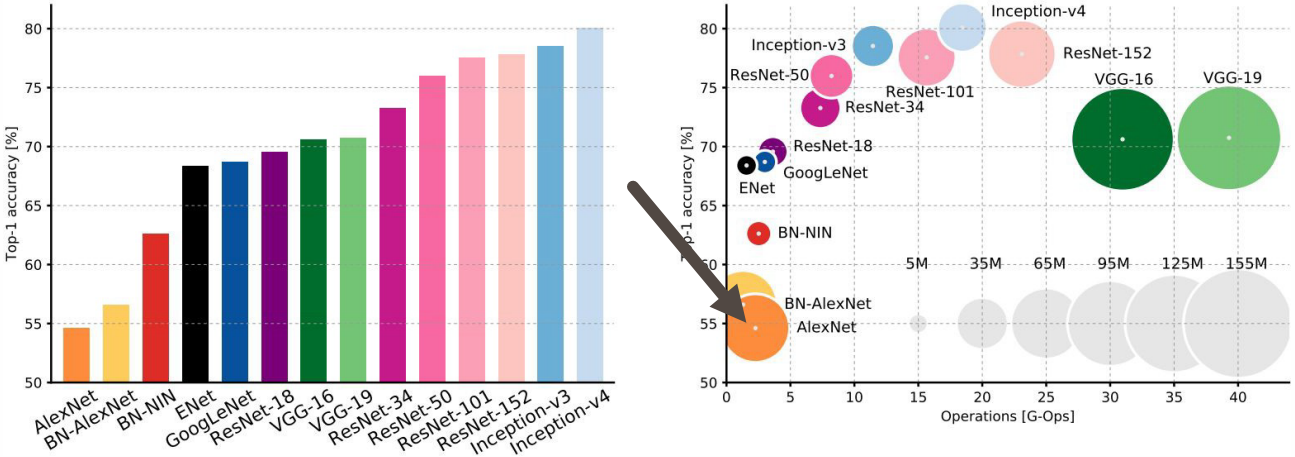
Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

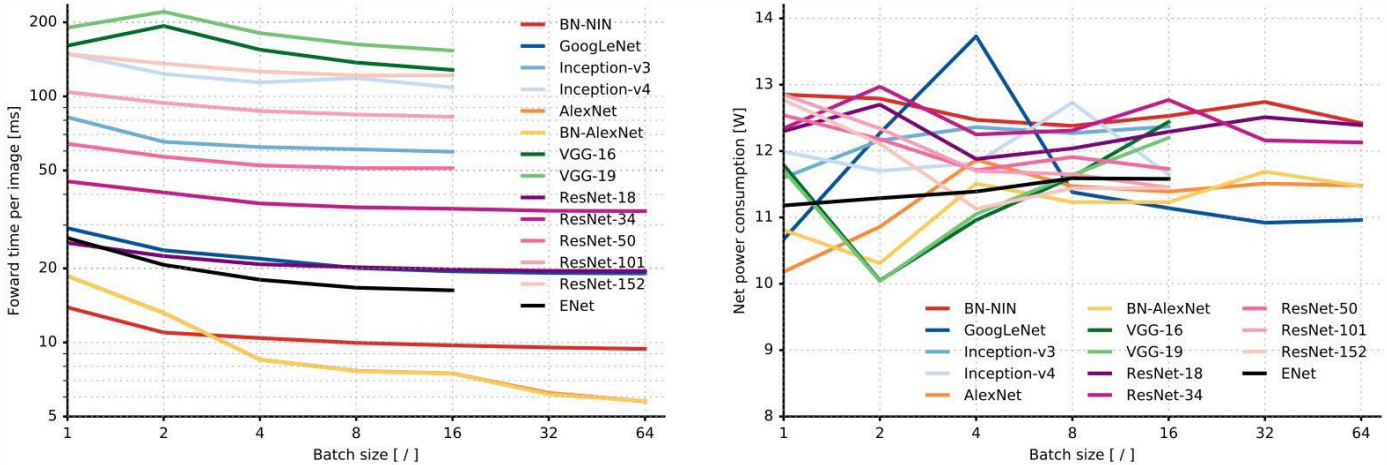
ResNet:
Moderate efficiency depending on model, highest accuracy

Comparing complexity...



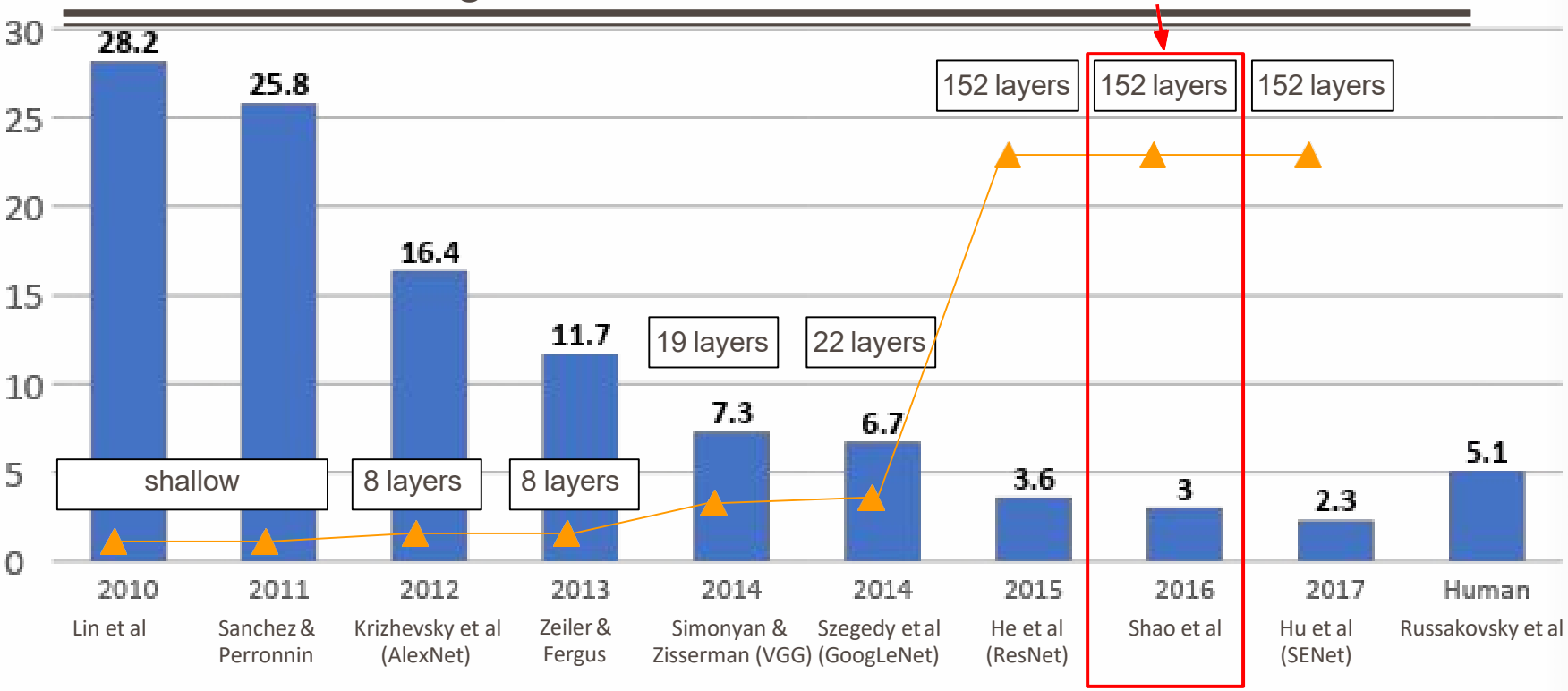
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Forward pass time and power consumption



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
Network ensembling



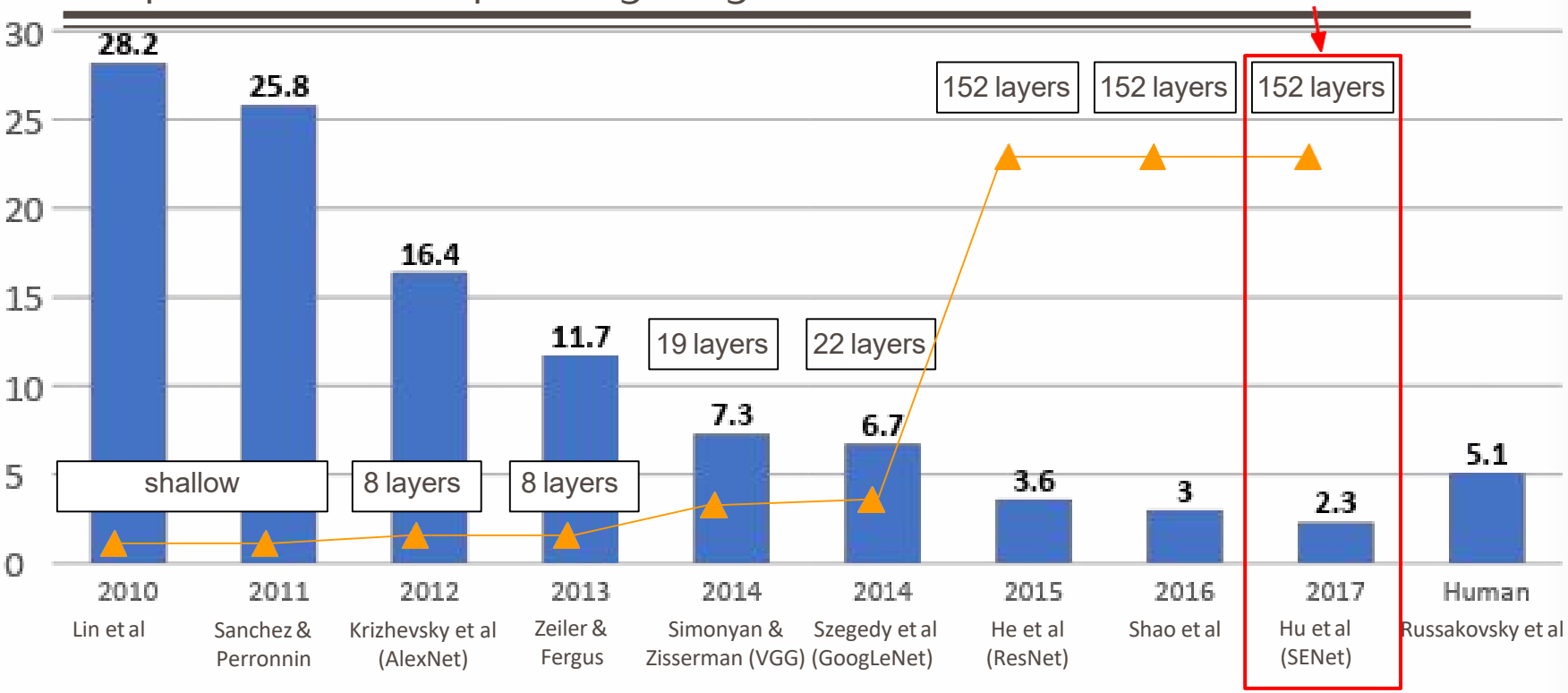
Improving ResNets...

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC'16 classification winner
- "Good Practices for Deep Feature Fusion"

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

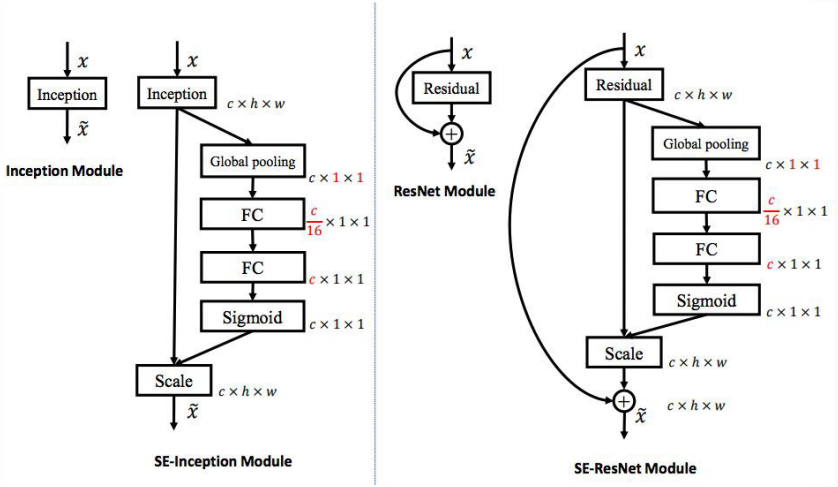
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
Adaptive feature map reweighting

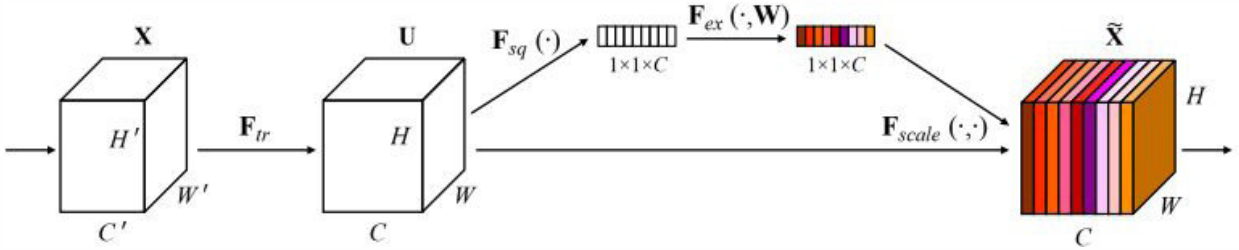


Improving ResNets...
Squeeze-and-Excitation Networks (SENet)

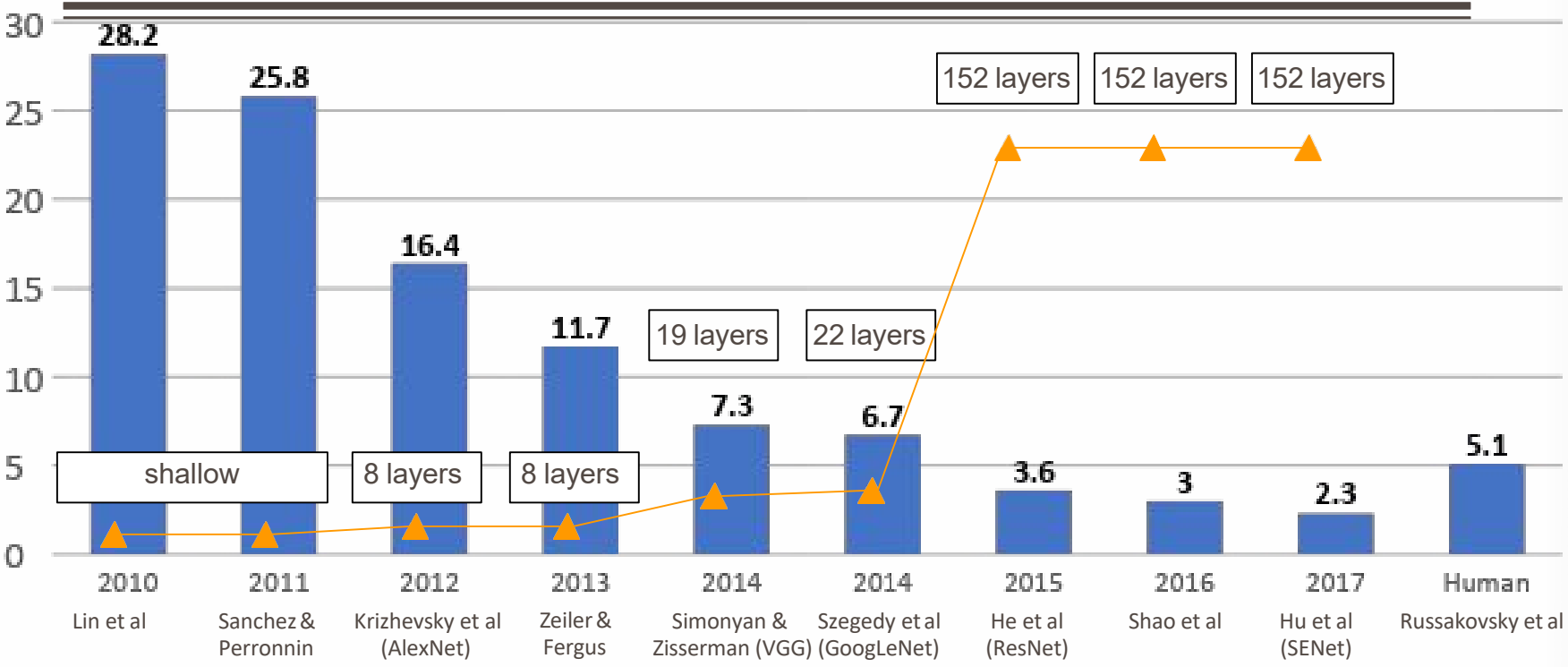
[Hu et al. 2017]

- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)

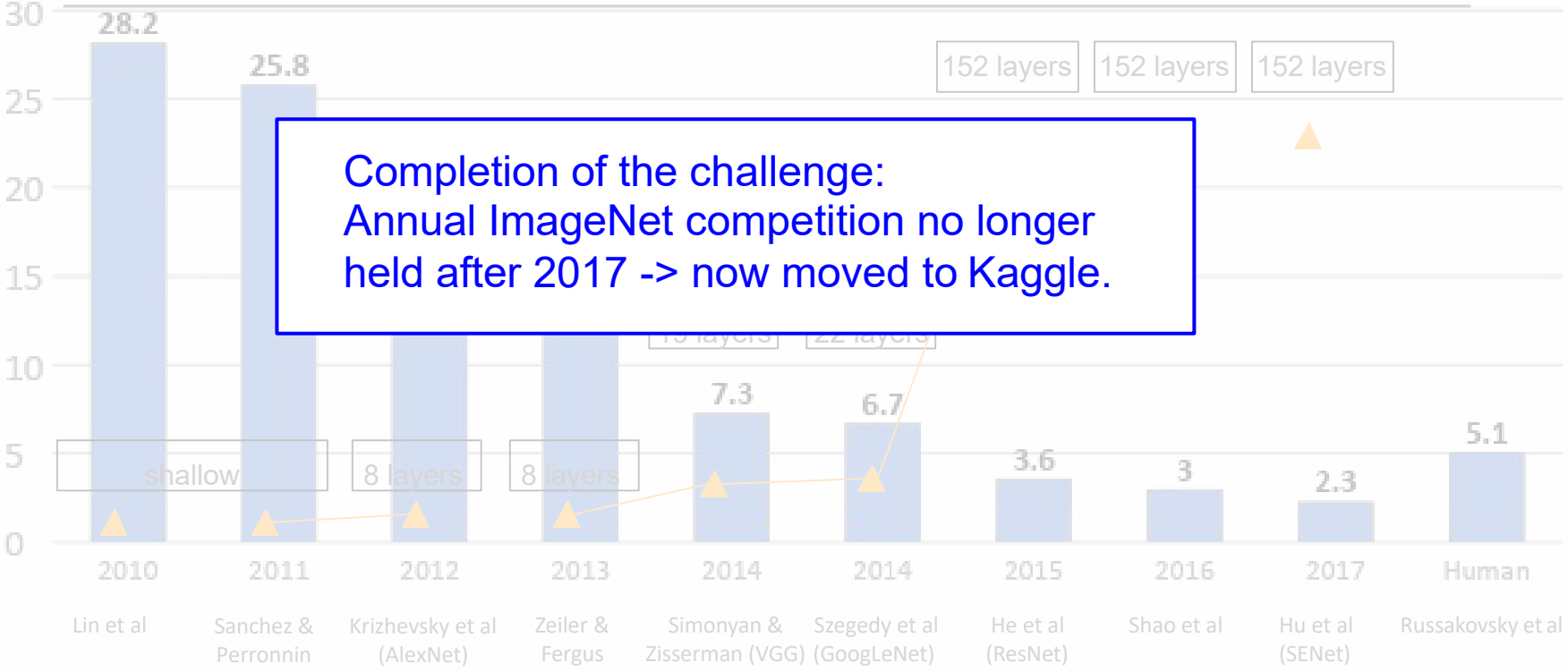




ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



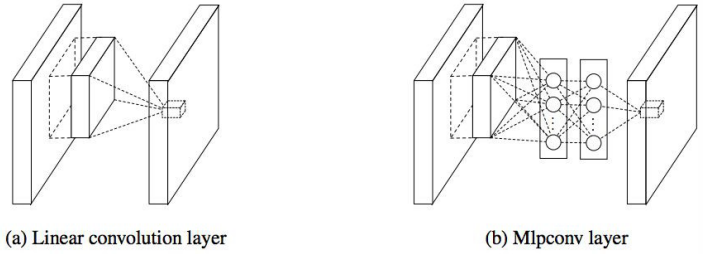
Completion of the challenge:
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.

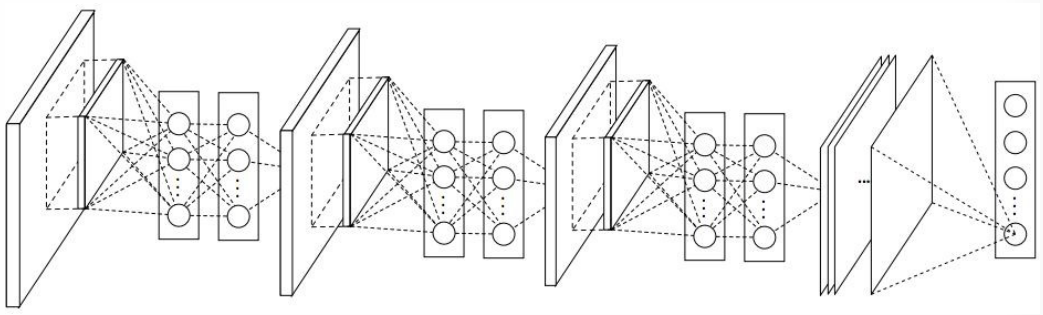
BUT RESEARCH INTO CNN ARCHITECTURES IS STILL FLOURISHING

Of historical note… Network in Network (NiN)

[Lin et al. 2014]

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron
- Precursor to GoogLeNet and ResNet “bottleneck” layers
- Philosophical inspiration for GoogLeNet



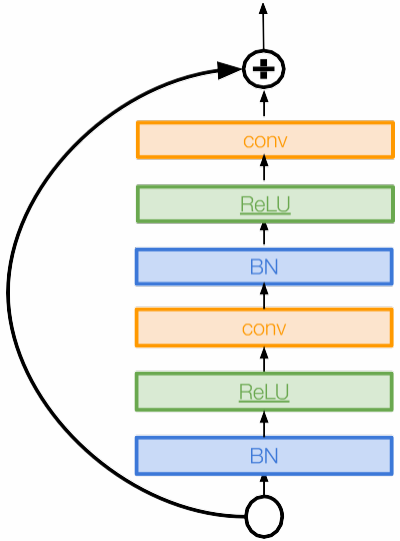


Figures copyright Lin et al., 2014. Reproduced with permission.

Identity Mappings in Deep Residual Networks

[He et al. 2016]

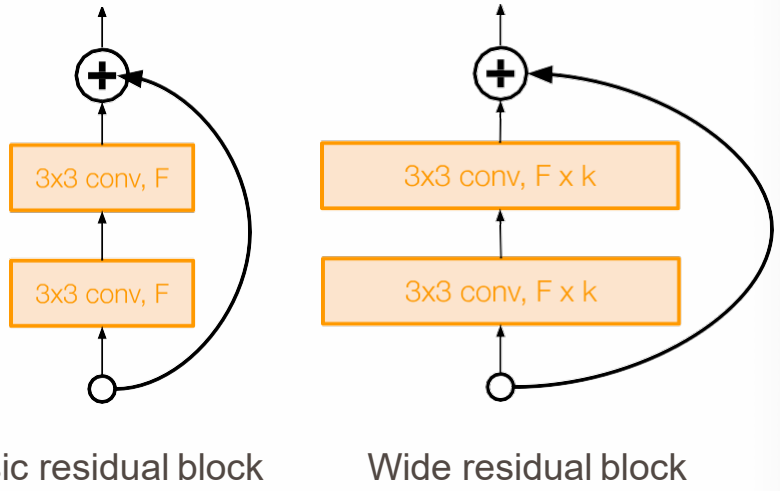
- Improving ResNets...
- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance



Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)

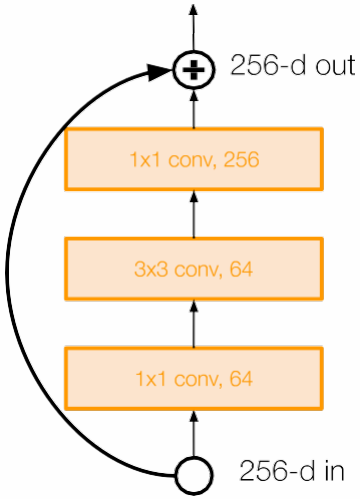


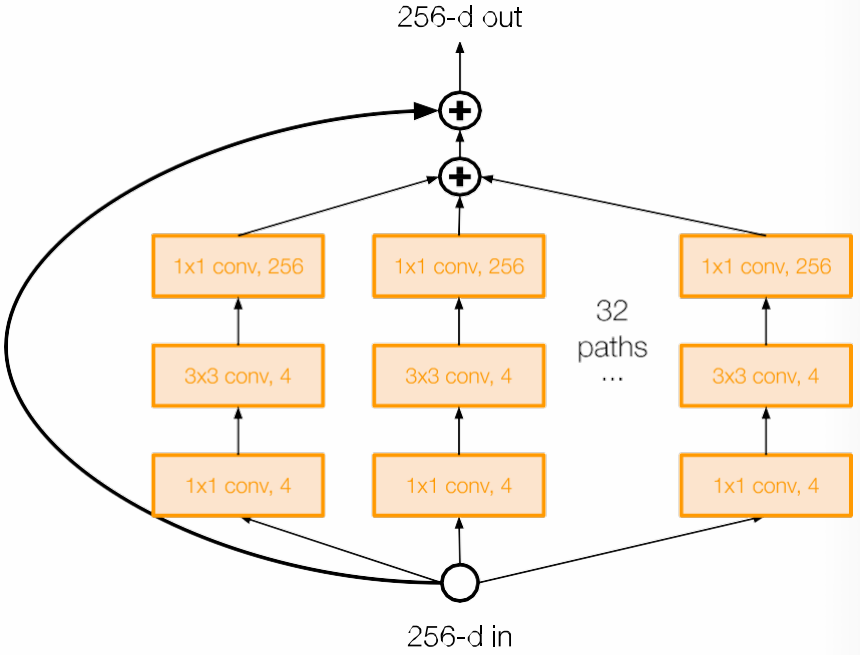
Basic residual block Wide residual block

Neural Networks (ResNeXt)

[Xie et al. 2016]

- Aggregated Residual Transformations for Deep
- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways (“cardinality”)
- Parallel pathways similar in spirit to Inception module

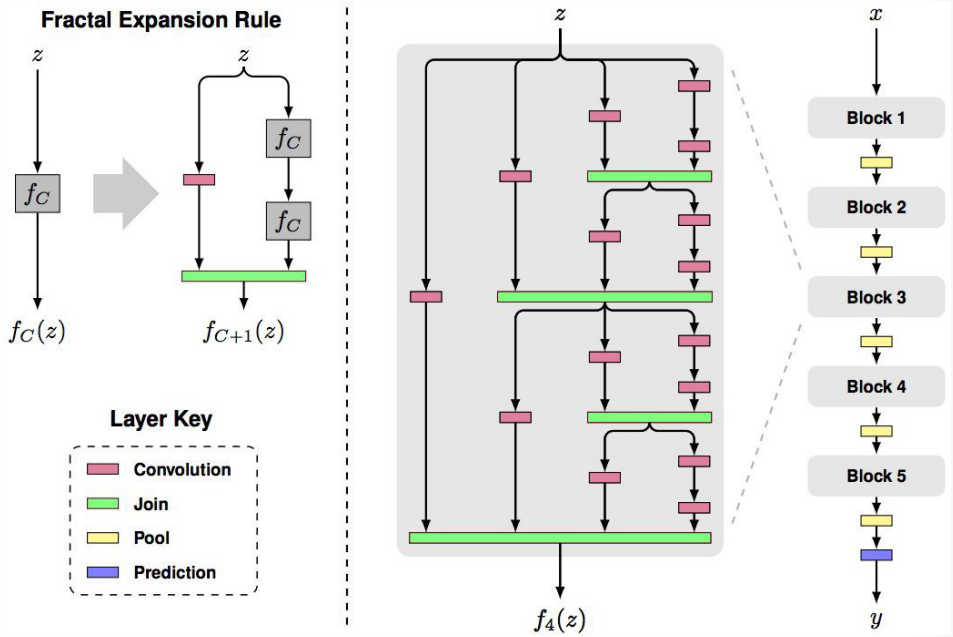




FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time

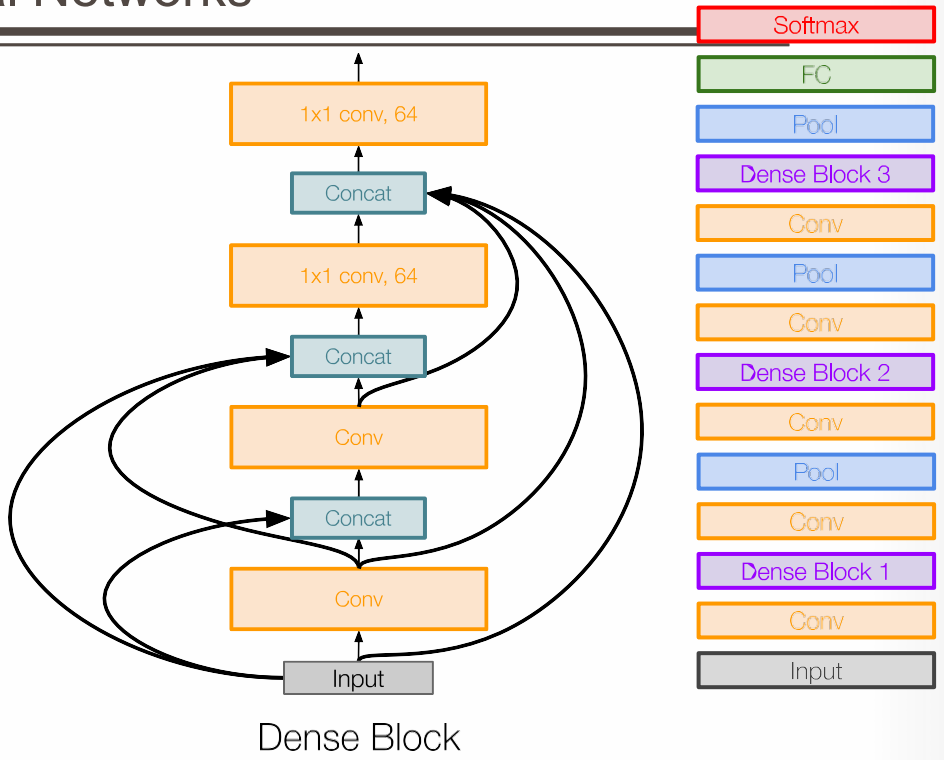


Figures copyright Larsson et al., 2017. Reproduced with permission.

Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
- Best paper in CVPR 2017!!



Dense Block

Dense Block: Densely Connected

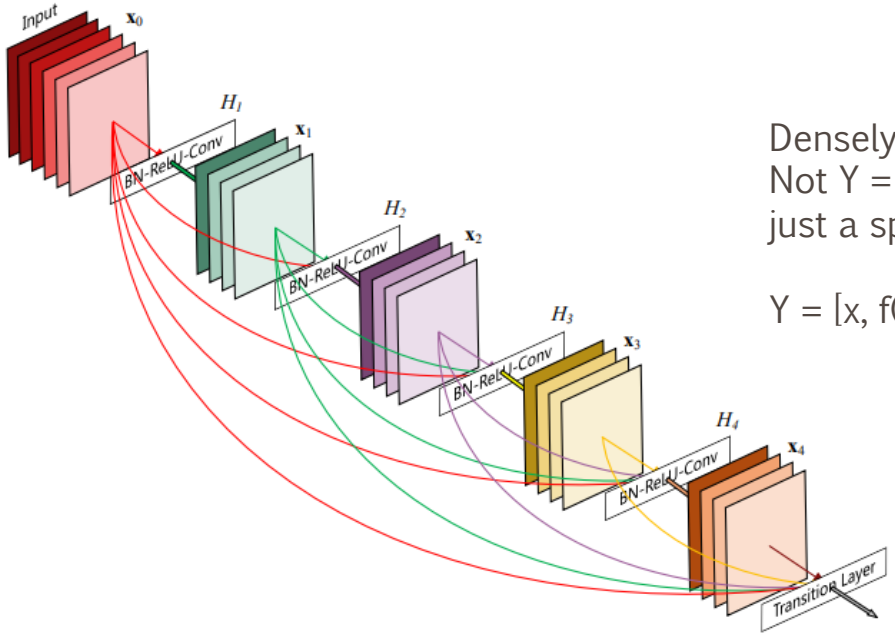
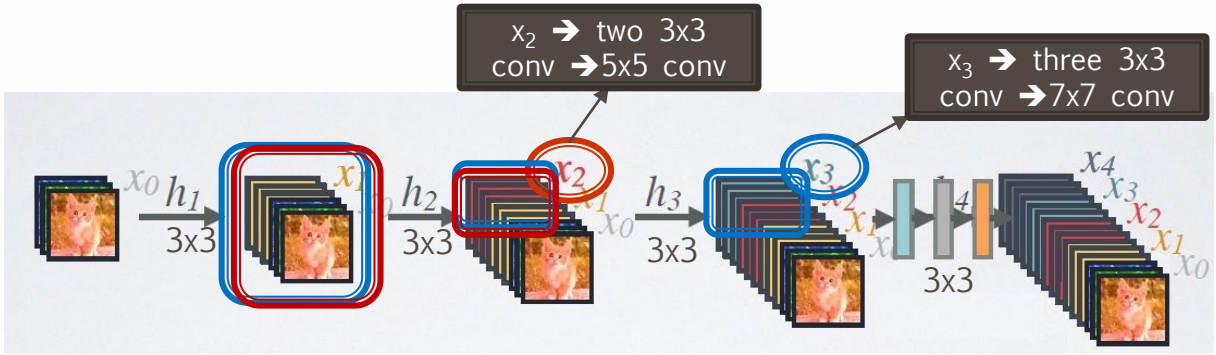


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Densely connected layers:
Not Y = x + f(x) + f(f(x)) ➔ It is just a special case of ResNet…

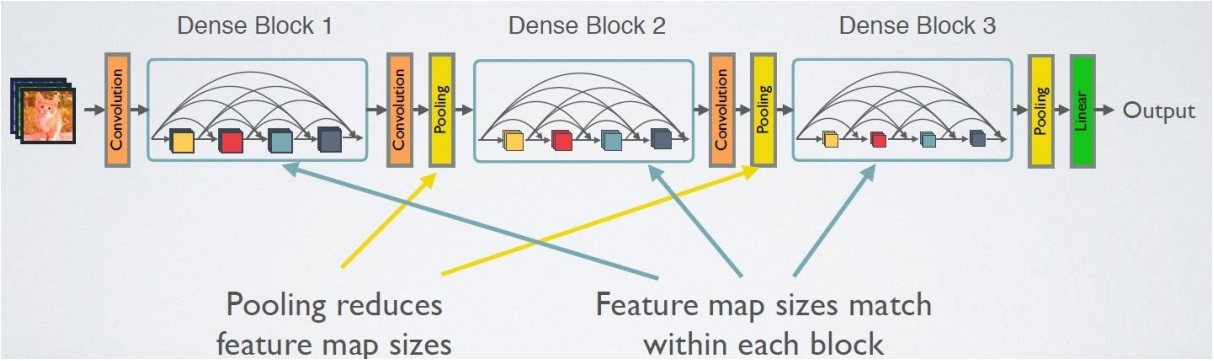
Y = [x, f(x)]!! Concatenation!

Dense Block



- Why it is excellent?
- Feature reuse!!
  - Multiscale feature representation
  - Recall that: two 3x3 conv ➔ 5x5 conv, three 3x3 conv➔7x7 conv.

Problem in DenseNet (cont.)



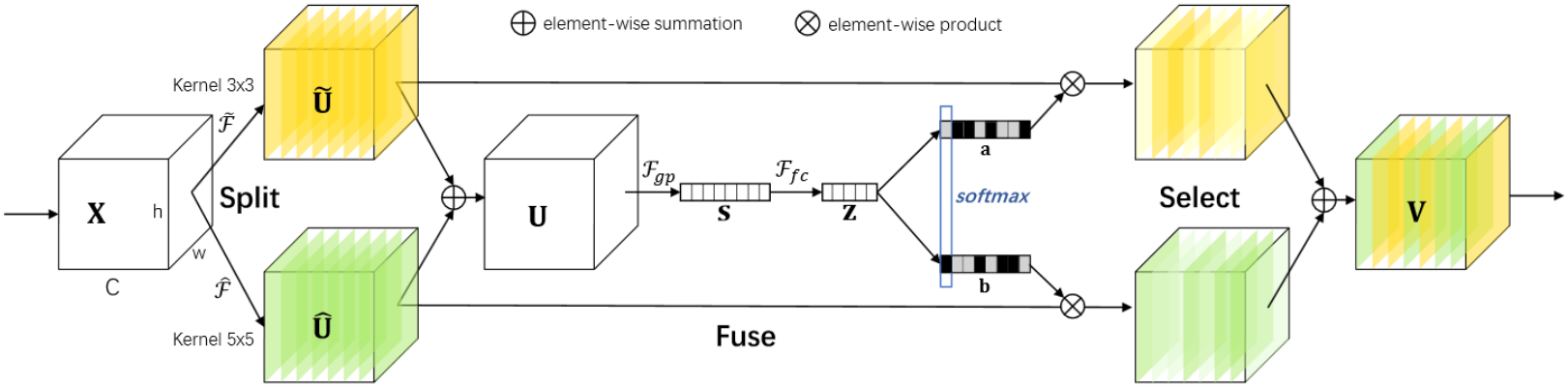
- Dense connection is performed in a dense block
  - Same feature map size
- Pooling (by max-pooling or strided conv.) between two dense blocks
- ResNet/DenseNet: Feature map reusing (large memory usage)

DenseNet

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

SKNet (Selective Kernel Convolution)

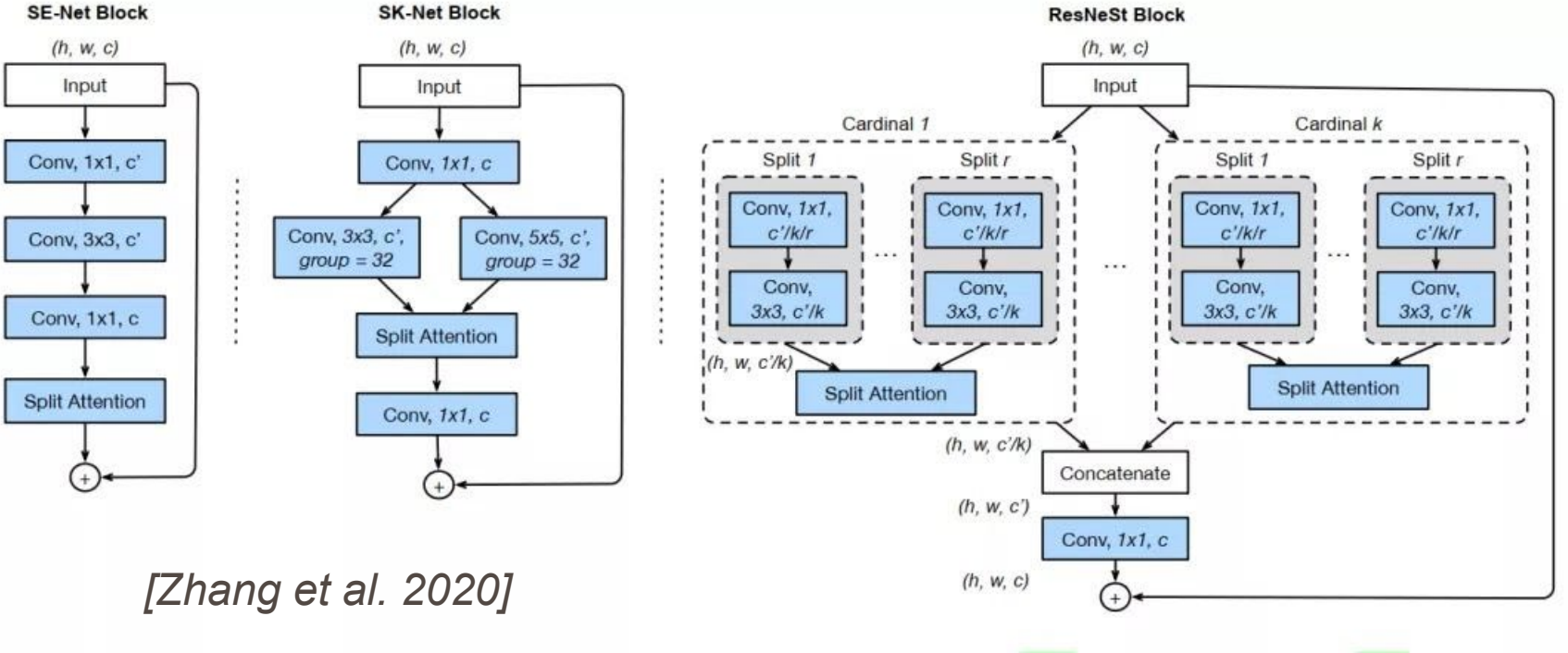
[Li et al. 2019]



Similar to SENet

Get the channeled-features and use split-attention to automatically decide which channel is important!

ResNeSt：Split-Attention Networks



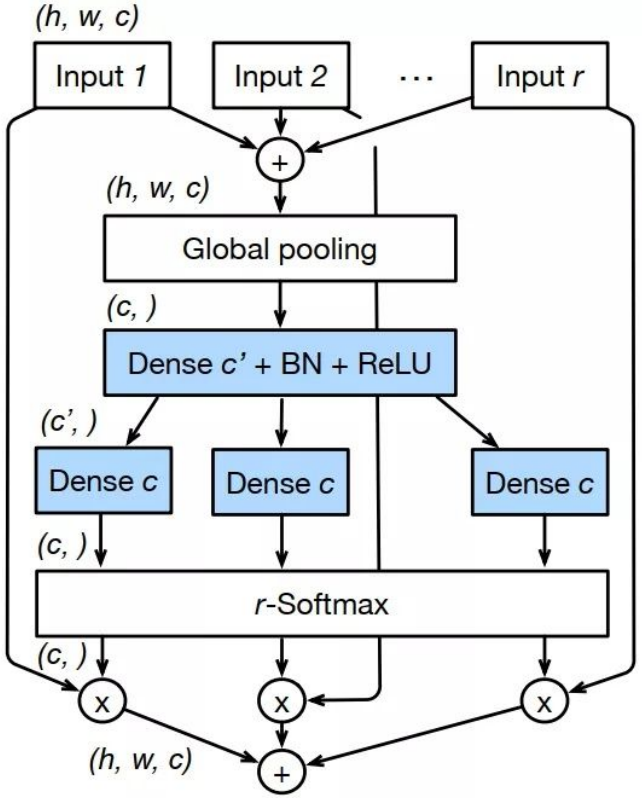
[Zhang et al. 2020]

ResNeSt：Split-Attention Networks

[Zhang et al. 2020]

Combine different existing modules

- Multi-path module in Inception
  - Different kernel sizes
  - Group conv. is used.
- Feature map attention (SKNet)
- Channel attention (SENet)

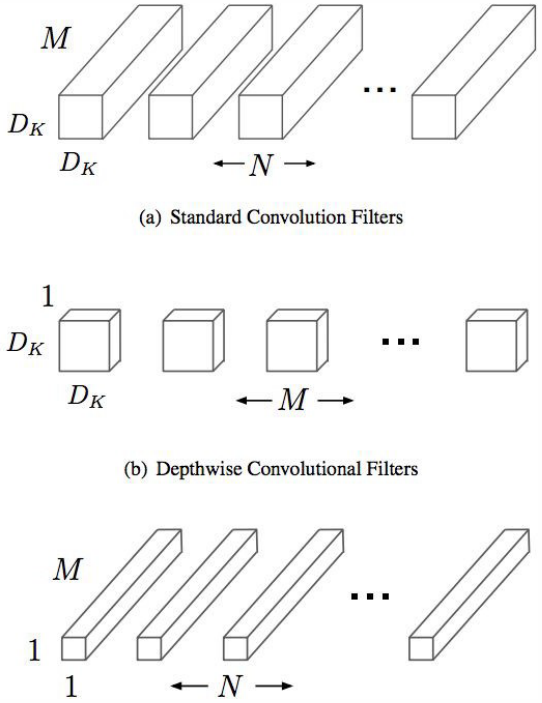


Efficient networks...

[Howard et al. 2017]

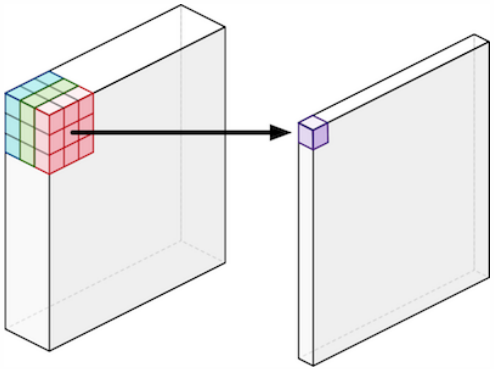
- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution that is much more efficient
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- Other works in this space e.g. ShuffleNet (Zhang et al. 2017)
- Latest: EfficientNet-b7
- Latest: RegNet (Statics for search range, [He et.al., 2020])

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

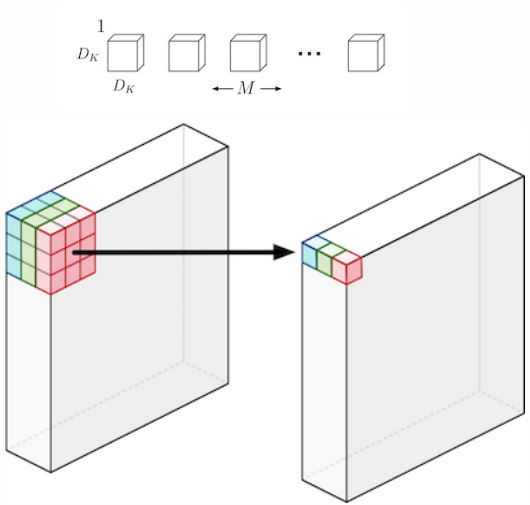


Depth-wise separable Convolution

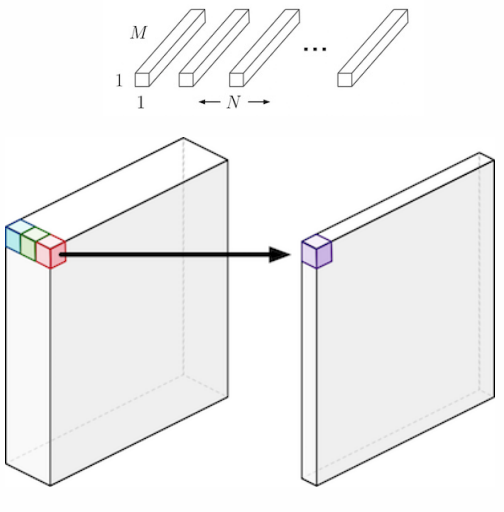
[Howard et al. 2017]



Standard conv.

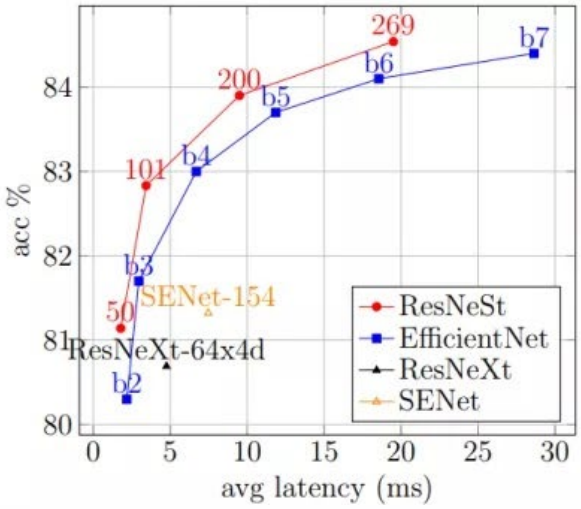


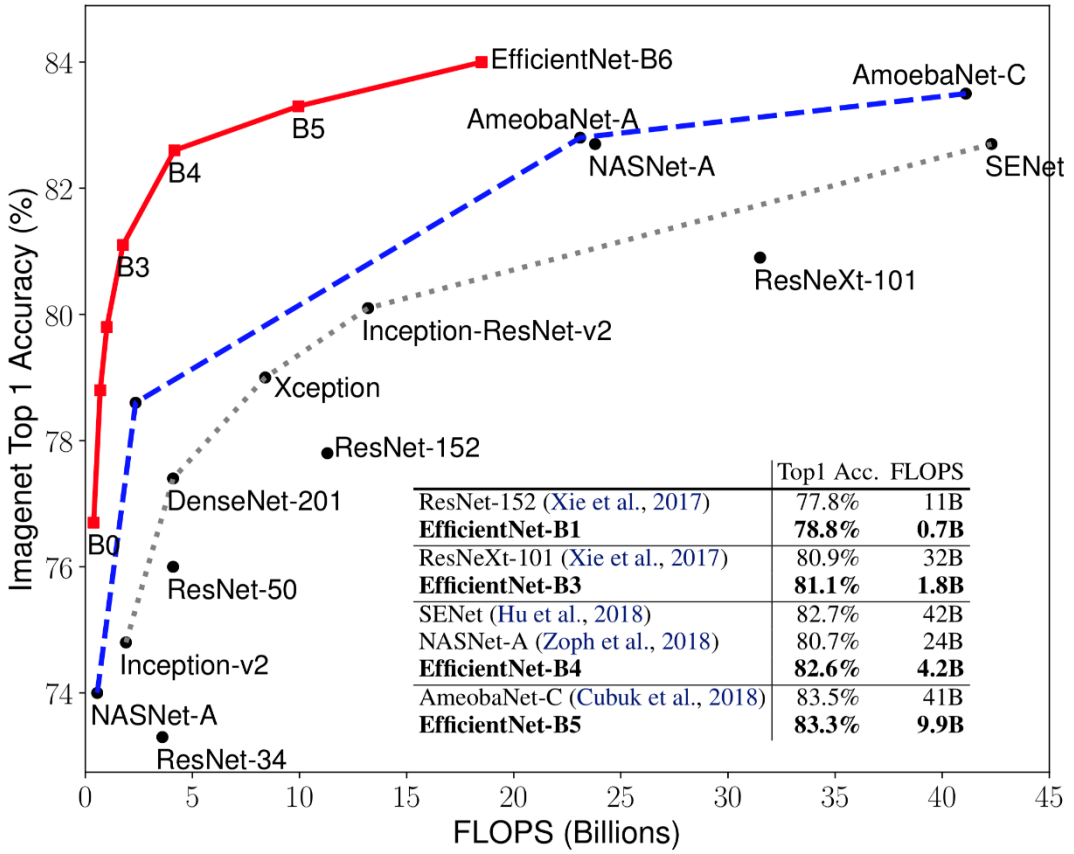
Depth-wise conv.



Point-wise conv.

SOTA for Image Recognition (2020)





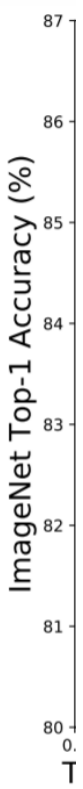
	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**78.8%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.1%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.6%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2018)	83.5%	41B
EfficientNet-B5	**83.3%**	**9.9B**

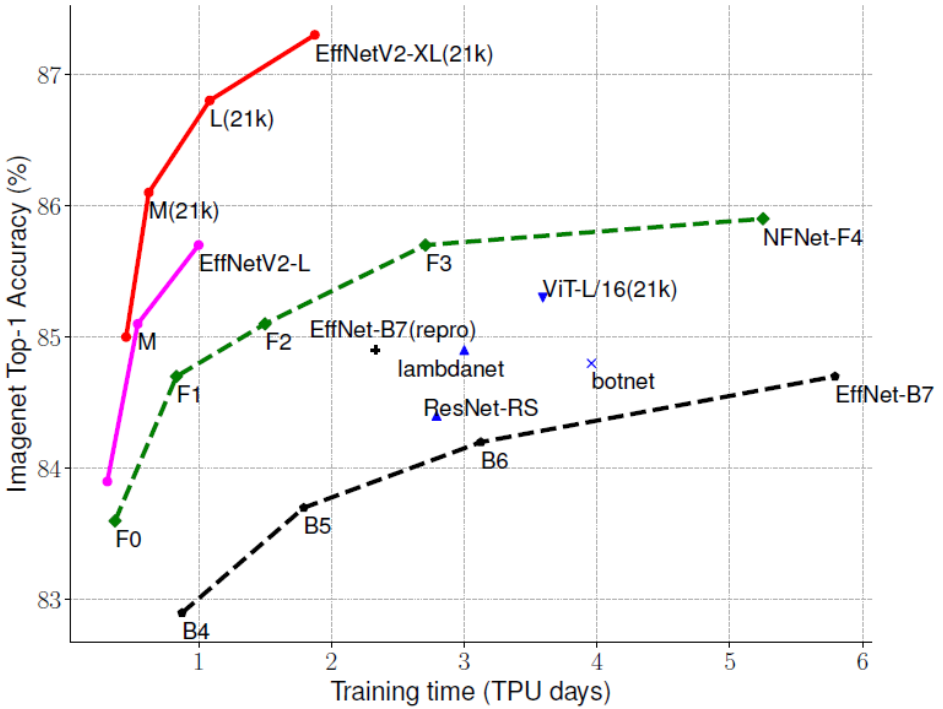
SOTA for Image Recogni

- Meta pseudo label
- Self-supervised learning
- Trained on 2,048 TPUv3 cores...?????WT
  - [Google 2021]

- DeepMine proposed NFNet
- Without normalization!!
  - Normalizer-Free ResNet
- AGC (gradient clip) [2021]

- **EfficientNet v2**



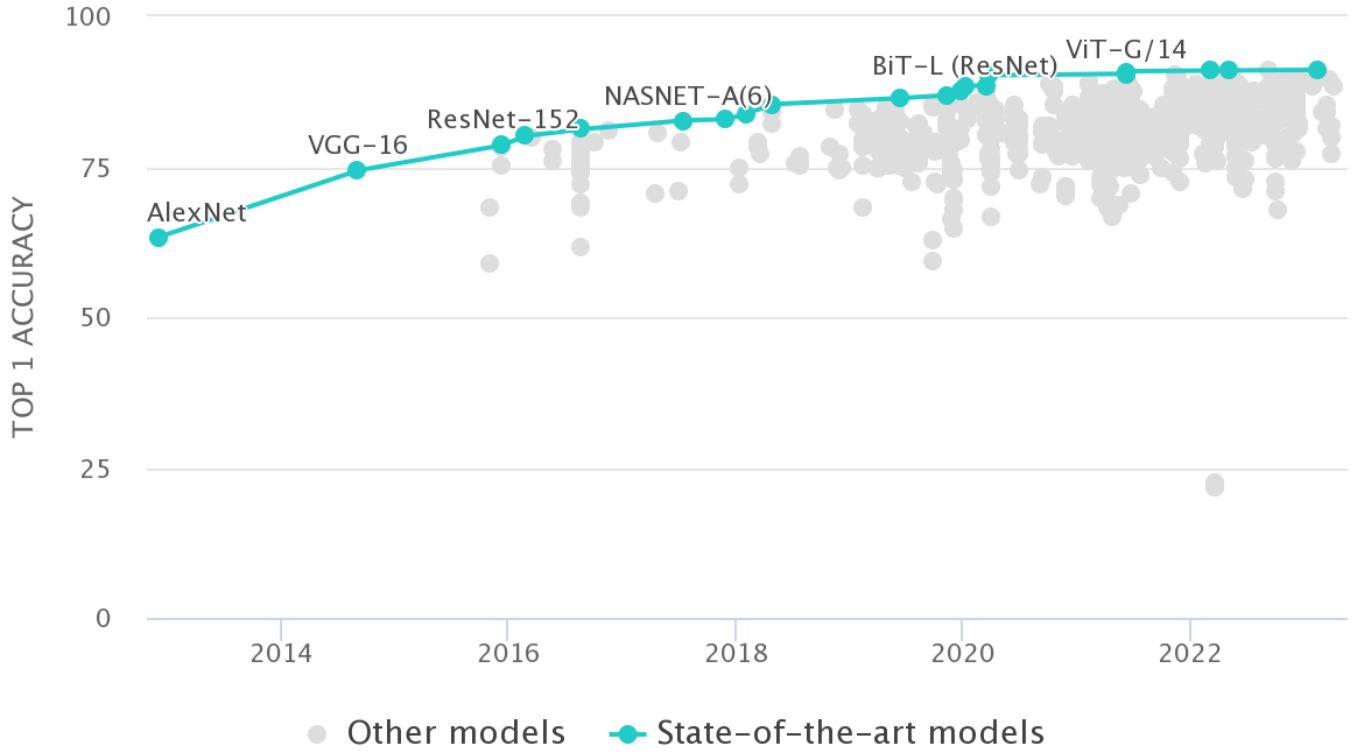


(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

Public Benchmark (2023) of ImageNet Validation Set



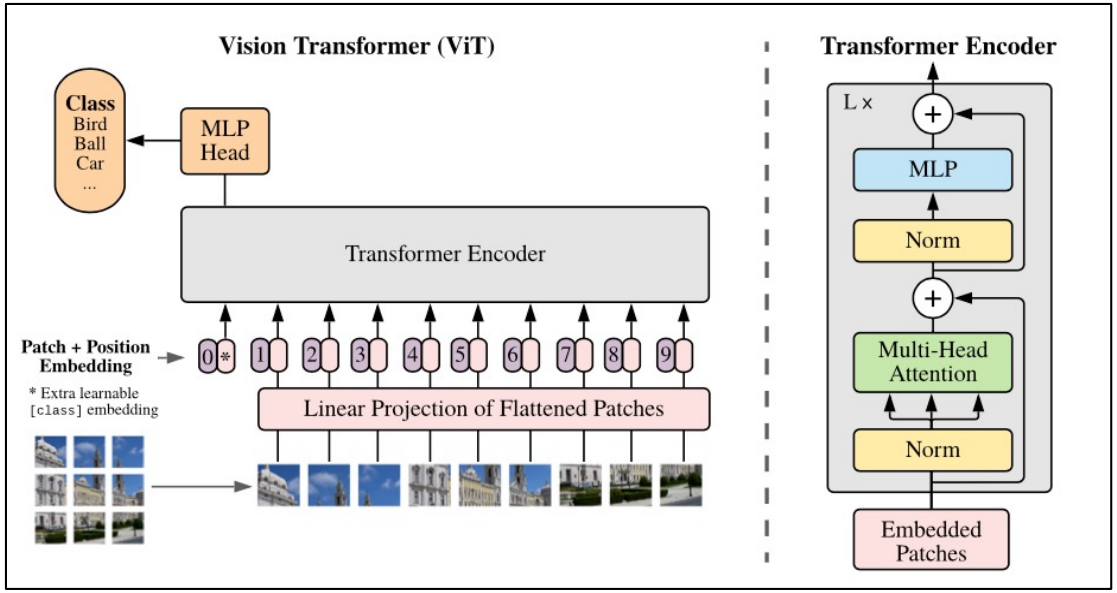
AGC (Adaptive Gradient Clipping) for NFNet

$$G \rightarrow \begin{cases} \lambda \frac{G}{\| G \|} & \text{if } \| G \| > \lambda \\ G & \text{otherwise} \end{cases}$$

$$G_i^\ell \rightarrow \begin{cases} \lambda \frac{\| W_i^\ell \|_F^\star}{\| G_i^\ell \|_F} G_i^\ell & \text{if } \frac{\| G_i^\ell \|_F}{\| W_i^\ell \|_F^\star} > \lambda \\ G_i^\ell & \text{otherwise.} \end{cases}$$

Other than SOTA CNNs

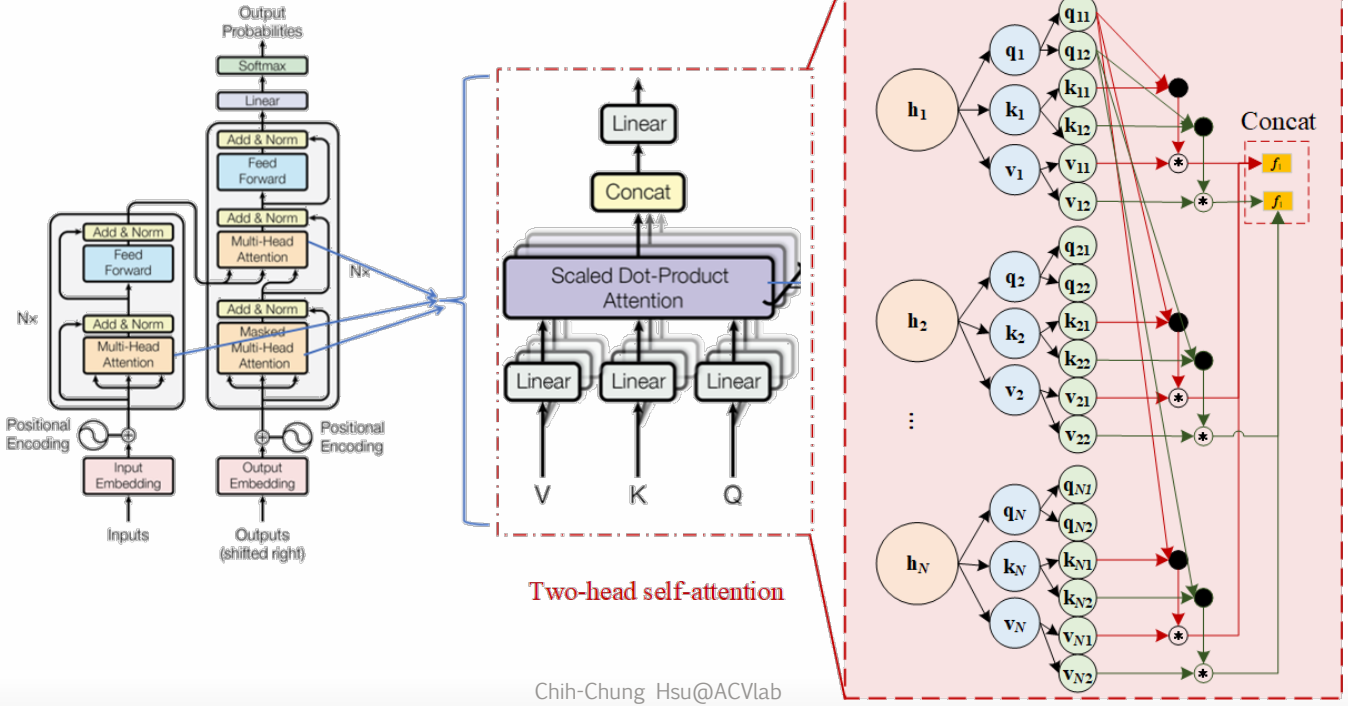
- Visual Transformer [ICLR 2021]!!
  - A feed-forward network for visual information without CNNs
  - Transformer: we talk it later



What’s Different?

Visual Transformer

- To make fully connected layer great again!!
  - So, what's attention?

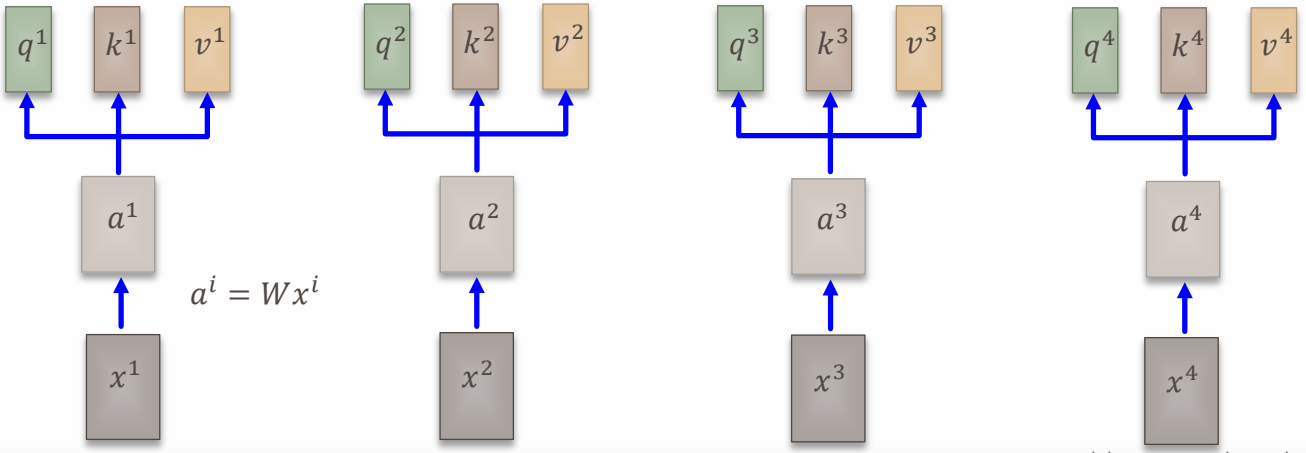


Self-attention

q: query (to match others) $\quad q^i = W^q a^i$

k: key (to be matched) $\quad k^i = W^k a^i$

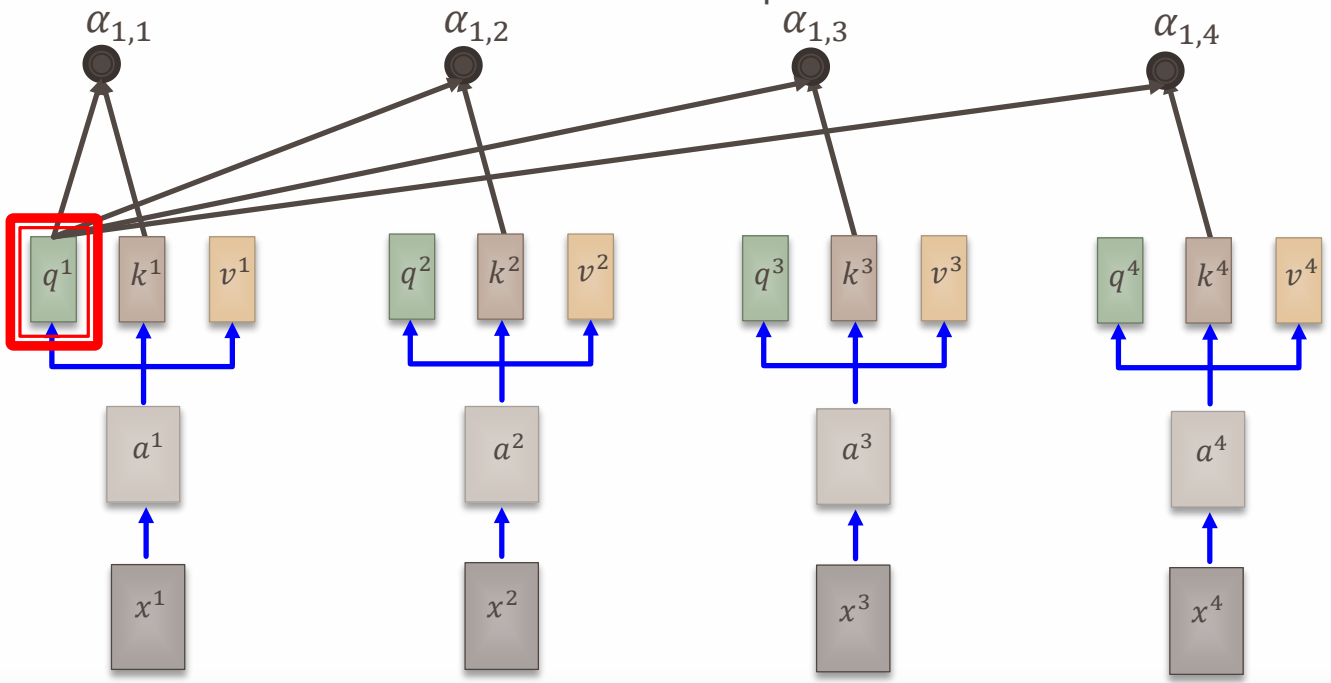
v: information to be extracted $\quad v^i = W^v a^i$



https://arxiv.org/abs/1706.03762

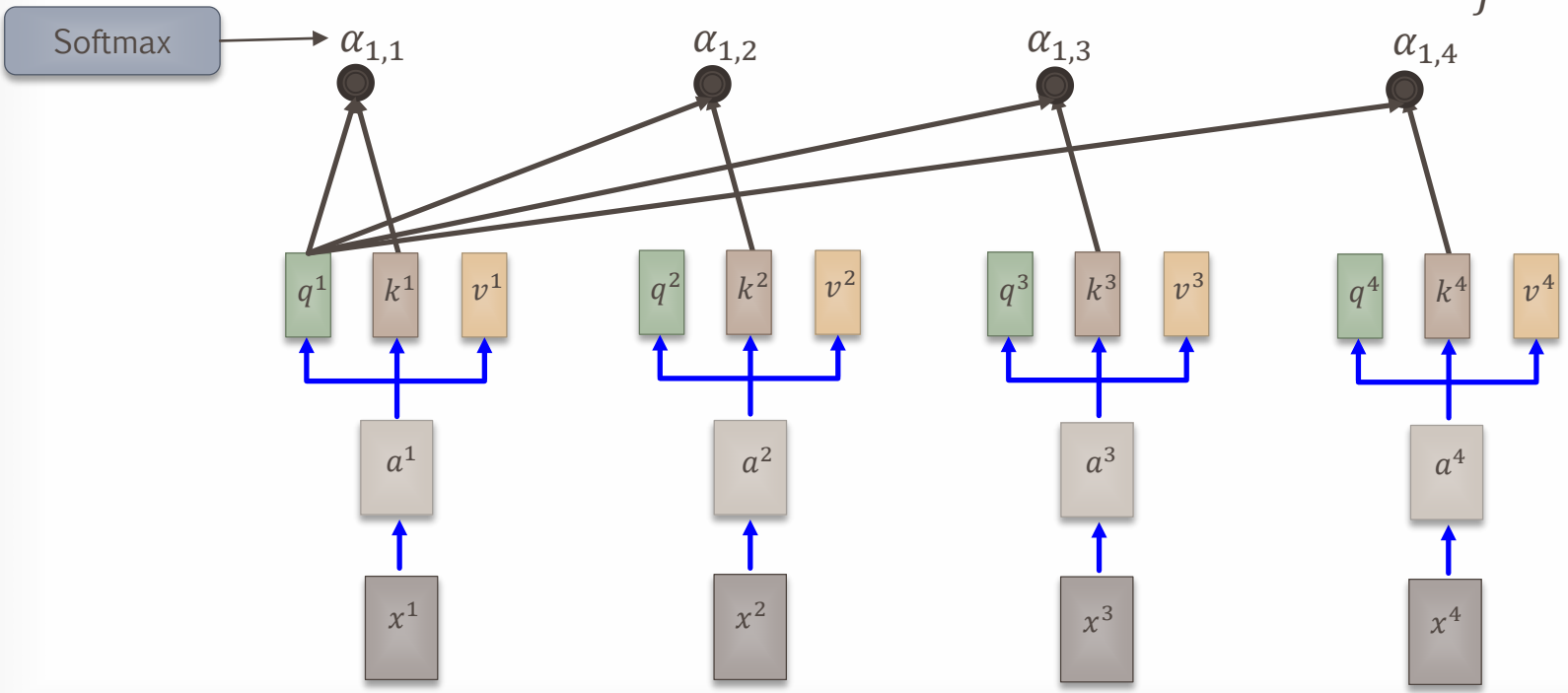
Self-attention

Scaled Dot-Product Attention: $\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$ d is the dim of q and k



Self-attention

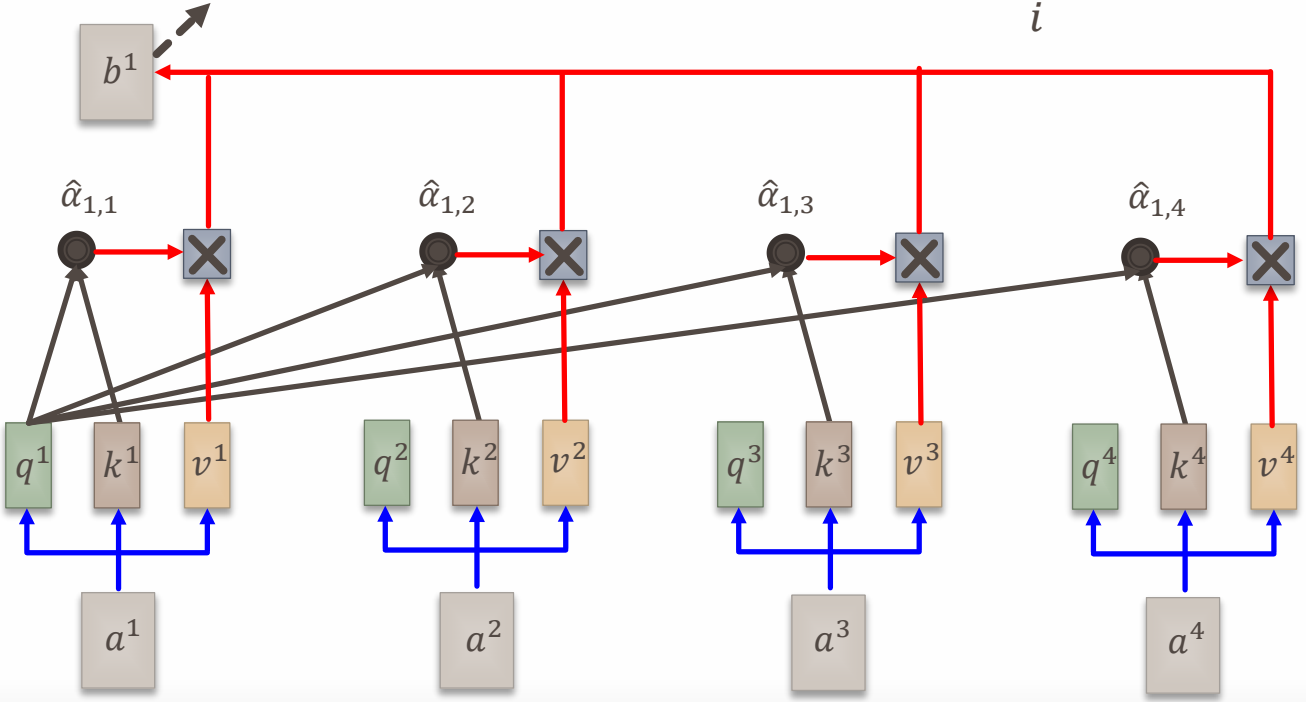
$$\hat{\alpha}_{1,i} = exp(\alpha_{1,i}) / \sum_j exp(\alpha_{1,j})$$



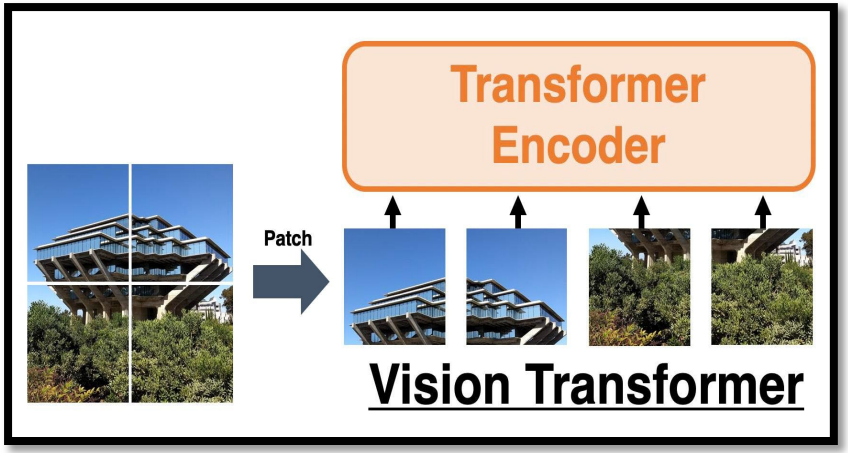
Self-attention

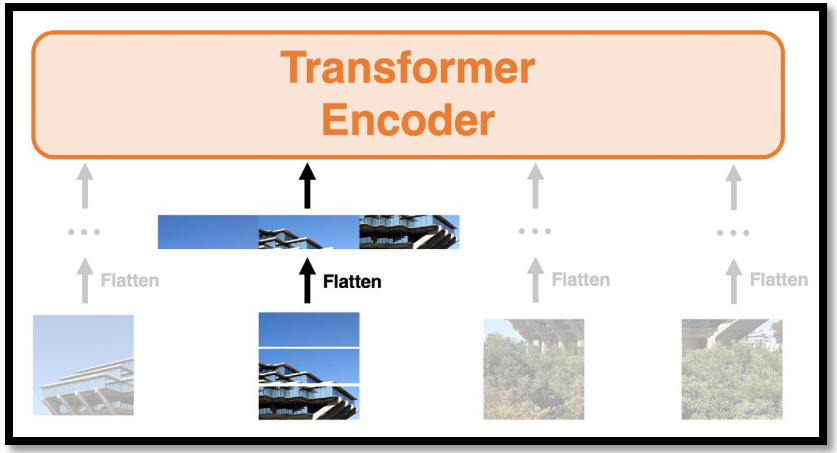
Attention works on everywhere!!

$$b^1 = \sum_i \hat{\alpha}_{1,i} v^i$$



ViT - Input Section



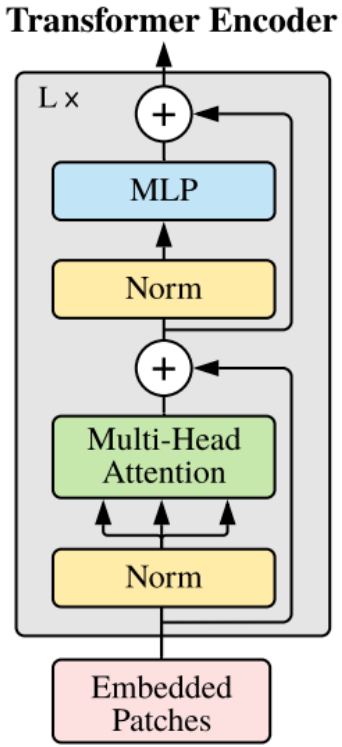


$$x \in \mathbb{R}^{H \times W \times C}$$

$$x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$$

$$N = HW/P^2$$

ViT - Transformer Encoder



A learnable embedding is added to the sequence of embedded patches ($z_0^0 = x_{class}$), where state at the output of the Transformer encoder (z_L^0) serves as the image representation y (Eq. 4).

Layer-norm (LN) is applied before every block, and residual connections after every block.

The MLP (Fully-connected layer) contains two layers with a GELU non-linearity.

CLS token

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

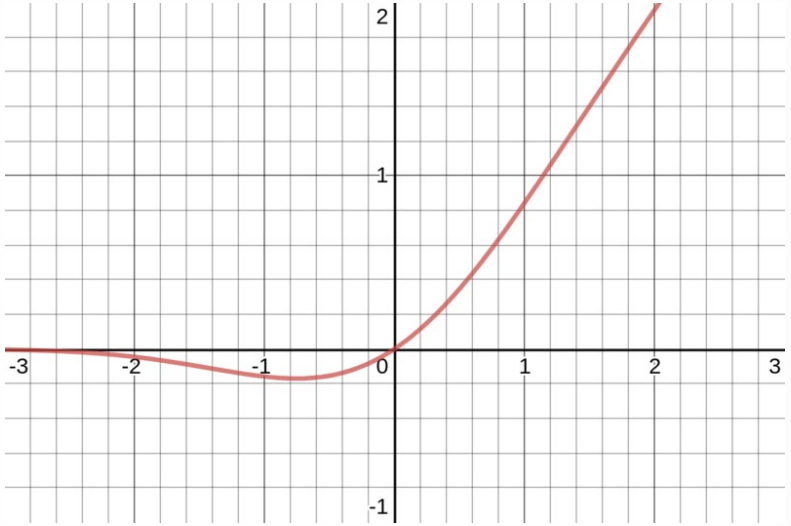
Hybrid Architecture : Raw image patches --> Feature map of a CNN

LN & GELU

```python
class LayerNorm(nn.Module):
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```

LN



GELU

Experiments - Training

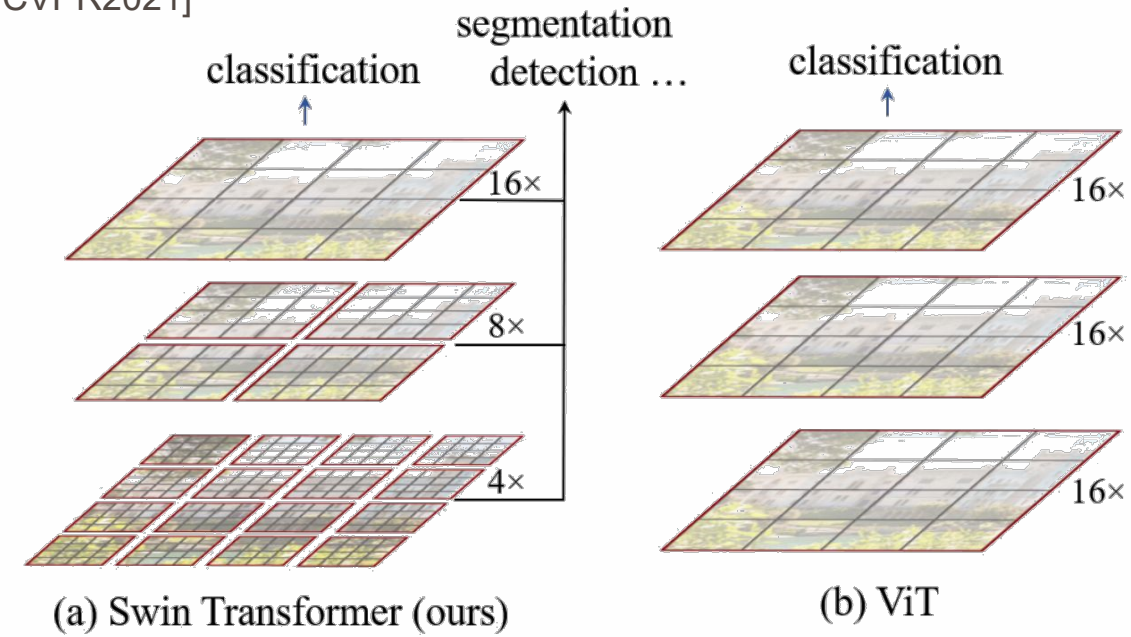
Models	Dataset	Epochs	Base LR	LR decay	Weight decay	Dropout
ViT-B/{16,32}	JFT-300M	7	$8 \cdot 10^{-4}$	linear	0.1	0.0
ViT-L/32	JFT-300M	7	$6 \cdot 10^{-4}$	linear	0.1	0.0
ViT-L/16	JFT-300M	7/14	$4 \cdot 10^{-4}$	linear	0.1	0.0
ViT-H/14	JFT-300M	14	$3 \cdot 10^{-4}$	linear	0.1	0.0
R50x{1,2}	JFT-300M	7	10^{-3}	linear	0.1	0.0
R101x1	JFT-300M	7	$8 \cdot 10^{-4}$	linear	0.1	0.0
R152x{1,2}	JFT-300M	7	$6 \cdot 10^{-4}$	linear	0.1	0.0
R50+ViT-B/{16,32}	JFT-300M	7	$8 \cdot 10^{-4}$	linear	0.1	0.0
R50+ViT-L/32	JFT-300M	7	$2 \cdot 10^{-4}$	linear	0.1	0.0
R50+ViT-L/16	JFT-300M	7/14	$4 \cdot 10^{-4}$	linear	0.1	0.0
ViT-B/{16,32}	ImageNet-21k	90	10^{-3}	linear	0.03	0.1
ViT-L/{16,32}	ImageNet-21k	30/90	10^{-3}	linear	0.03	0.1
ViT-$*$	ImageNet	300	$3 \cdot 10^{-3}$	cosine	0.3	0.1

patch size

Table 3: Hyperparameters for training. All models are trained with a batch size of 4096 and learning rate warmup of 10k steps. For ImageNet we found it beneficial to additionally apply gradient clipping at global norm 1. Training resolution is 224.

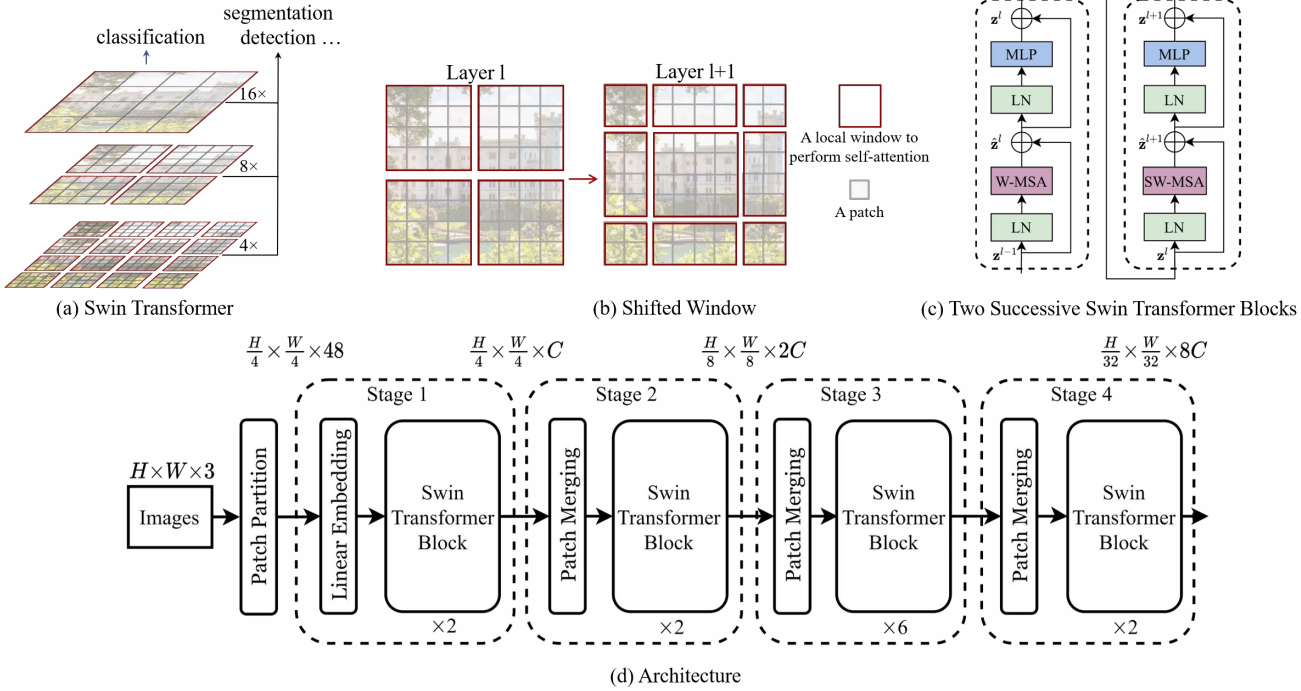
Previous SOTA (2021.4)

- Swin-Transformer V1 (vision)
  - [MSRA, CVPR2021]



arxiv.org/abs/2103.14030

Swin-Transformer V2: Swin Transformer V2: Scaling Up Capacity and Resolution (CVPR 2022)



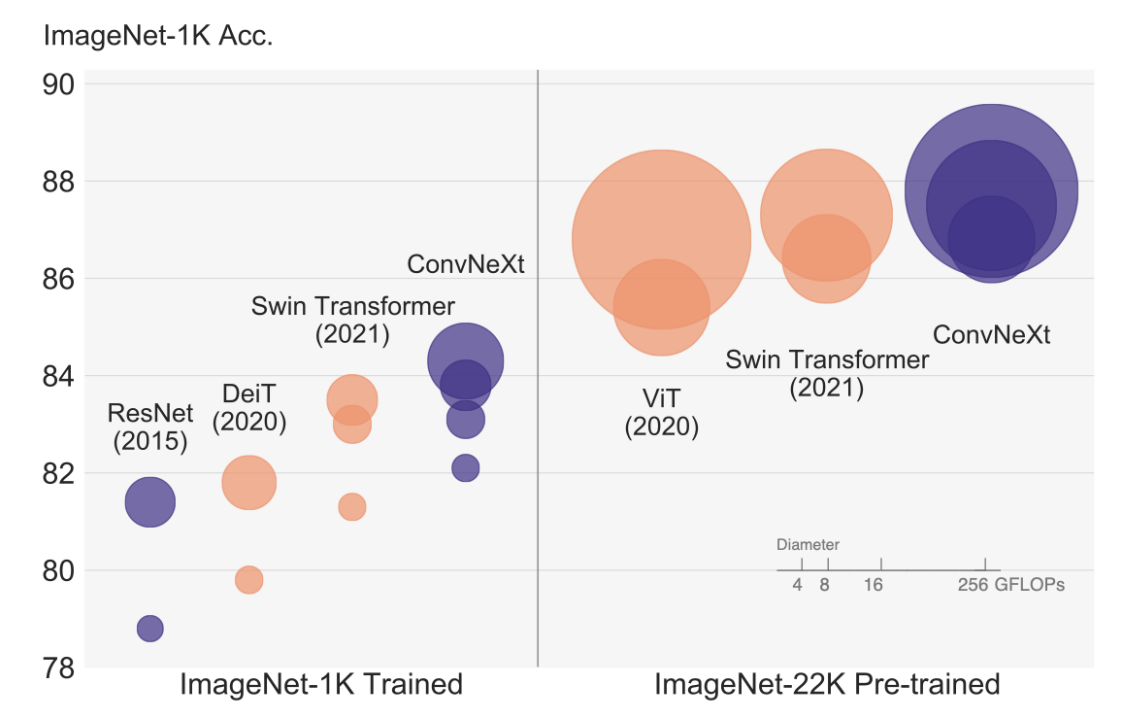
(a) Swin Transformer (b) Shifted Window (c) Two Successive Swin Transformer Blocks

(d) Architecture

TRANSFORMER BECOMES “SOTA”?

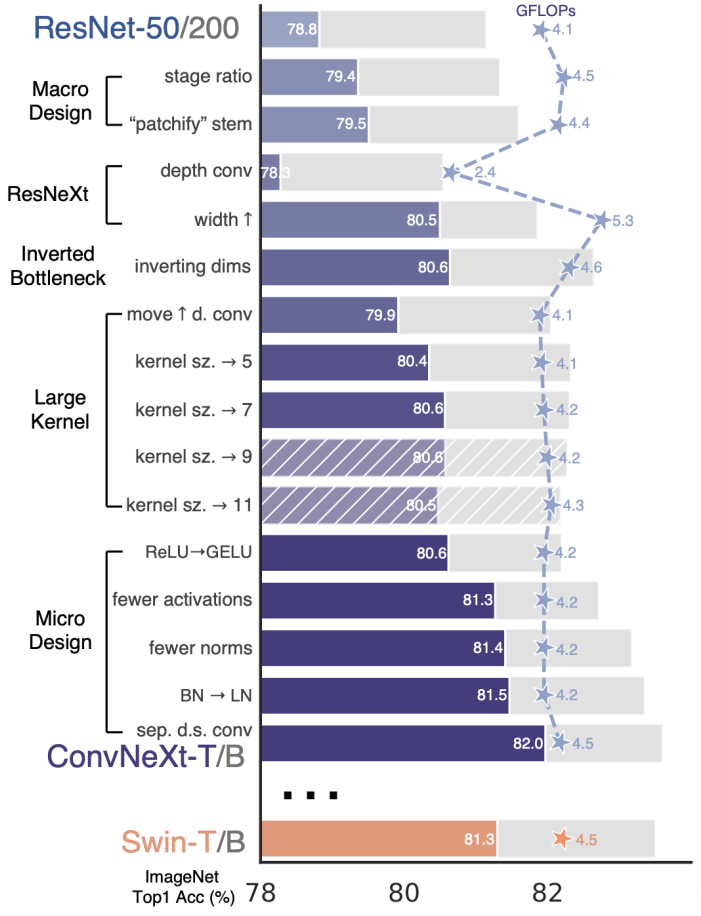
Not really.
Let’s see the truth!

ImageNet-1K Classification Benchmark



A ROADMAP

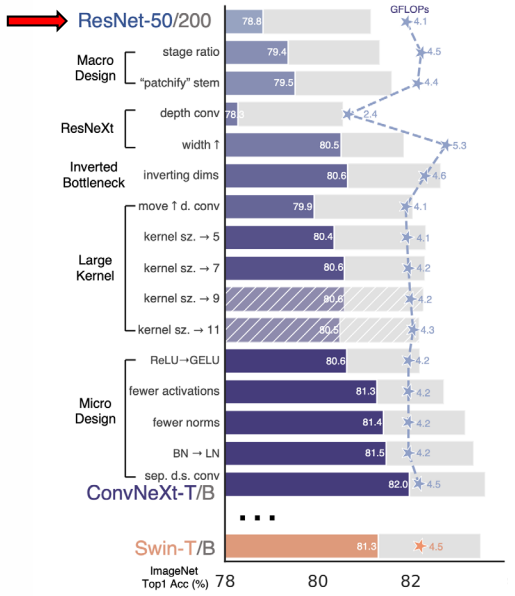
From ResNet
to ConvNeXt



Training Techniques

- More epochs: 90 -> 300
- Optimizer: AdamW
- Data augmentation: Mixup, Cutmix, Random Augment, Random Erasing
- Regularizations: Stochastic Depth, Label Smoothing

- ImageNet Top-1 acc: 76.1% -> 78.8%

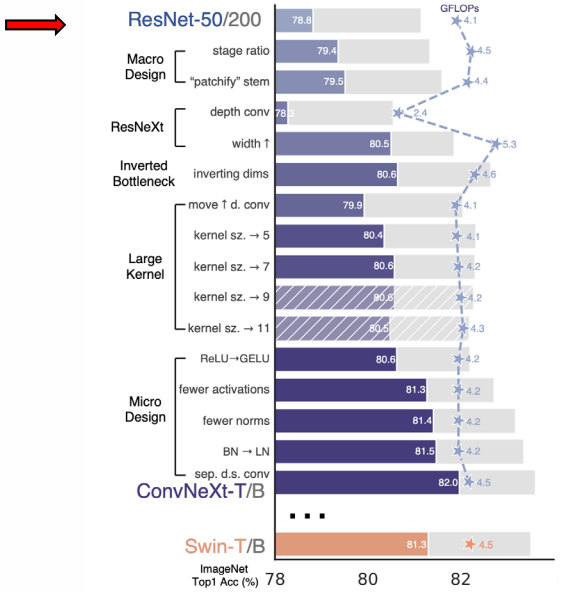


Macro Design

- Changing stage compute ratio / Changing stem to “Pachify”

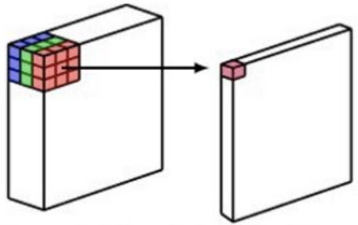
	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

Table 9. **Detailed architecture specifications** for ResNet-50, ConvNeXt-T and Swin-T.

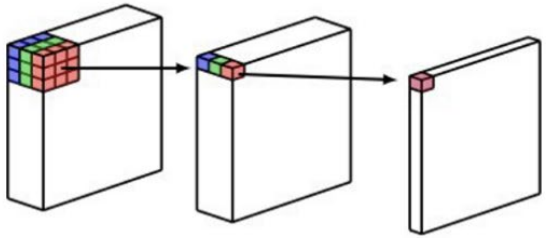


ResNeXt-ify

- Depth-wise Separable Convolution



(a) Conventional Convolutional Neural Network



(b) Depthwise Separable Convolutional Neural Network

$Given\ \ kernel\ size\ (K, K), strides\ 1,$
$input\ resolution\ (H_1, W_1), channels\ C_1,$
$output\ resolution\ (H_2, W_2),, channels\ C_2,$

Standard Convolution

FLOPs = $H_2 W_2 (K^2 C_1 C_2)$

$$\frac{1}{C_2} + \frac{1}{K^2}$$

Depth-wise Separable Convolution

FLOPs = $H_2 W_2 (K^2 C_1 + C_1 C_2)$

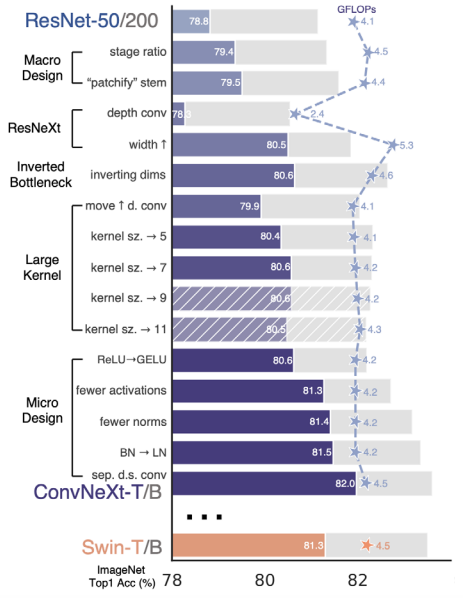
All calculation ignore bias add, batch normalization and activation function.
The resulting FLOPs should be double if multiply accumulate (MAC) matters

ResNeXt-ify

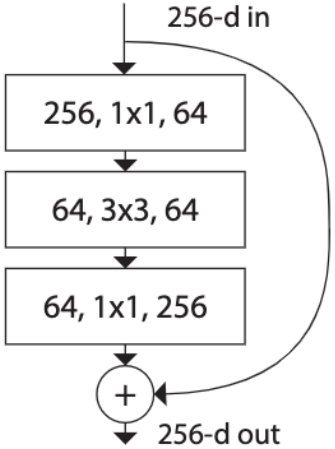
- Depth-wise Separable Convolution

	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

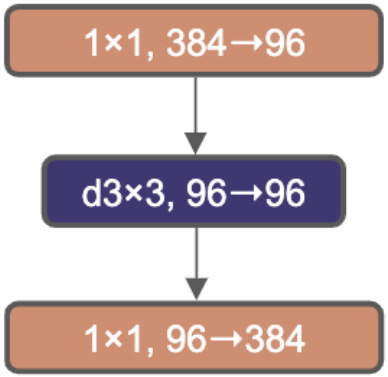
Table 9. **Detailed architecture specifications** for ResNet-50, ConvNeXt-T and Swin-T.



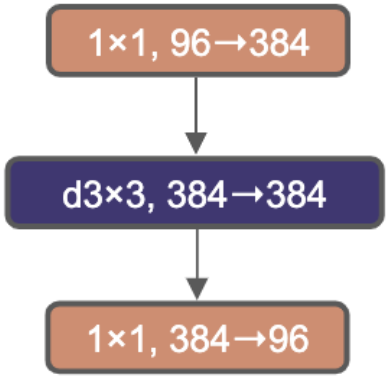
Inverted Bottleneck / Large Kernel Size



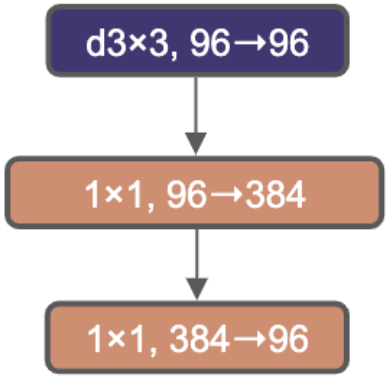
RESNET Block



(a)



(b)

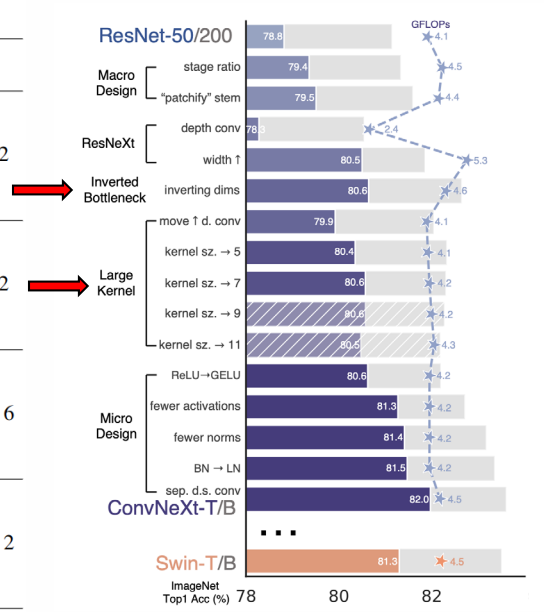


(c)

Inverted Bottleneck / Large Kernel Size

	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

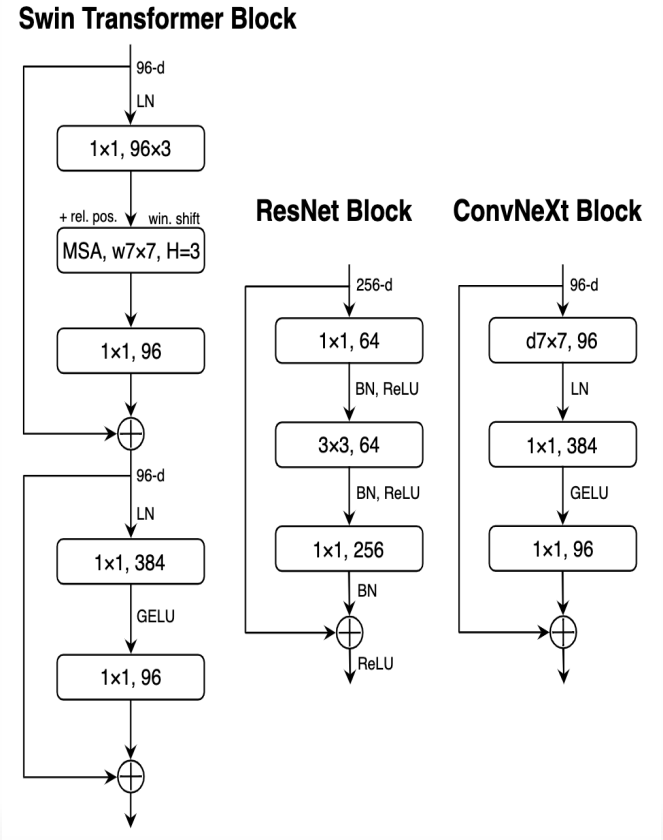
Table 9. **Detailed architecture specifications** for ResNet-50, ConvNeXt-T and Swin-T.



Micro Design

- RELU -> GELU
- Fewer activation
- Fewer norms
- Batch Norm -> Layer Norm
- Separate downsampling conv

- ImageNet Top-1 acc: 82.0%



Micro Design

- Layer Normalization

6.7 Convolutional Networks

We have also experimented with convolutional neural networks. In our preliminary experiments, we observed that layer normalization offers a speedup over the baseline model without normalization, but batch normalization outperforms the other methods. With fully connected layers, all the hidden units in a layer tend to make similar contributions to the final prediction and re-centering and re-scaling the summed inputs to a layer works well. However, the assumption of similar contributions is no longer true for convolutional neural networks. The large number of the hidden units whose receptive fields lie near the boundary of the image are rarely turned on and thus have very different statistics from the rest of the hidden units within the same layer. We think further research is needed to make layer normalization work well in ConvNets

https://arxiv.org/abs/1607.06450

Different ConvNeXt variants

- ConvNeXt-T
- ConvNeXt-S
- ConvNeXt-B
- ConvNeXt-L
- ConvNeXt-XL

Channels (C)	Blocks (B)
(96, 192, 384, 768)	(3, 3, 9, 3)
(96, 192, 384, 768)	(3, 3, 27, 3)
(128, 256, 512, 1024)	(3, 3, 27, 3)
(192, 384, 768, 1536)	(3, 3, 27, 3)
(256, 512, 1024, 2048)	(3, 3, 27, 3)

Results

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
● RegNetY-16G [54]	224^2	84M	16.0G	334.7	82.9
● EffNet-B7 [71]	600^2	66M	37.0G	55.1	84.3
● EffNetV2-L [72]	480^2	120M	53.0G	83.7	85.7
○ DeiT-S [73]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [73]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
● ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
● ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
● ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
● ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
● ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
● ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**

Results

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
• RegNetY-16G [54]	224^2	84M	16.0G	334.7	82.9
• EffNet-B7 [71]	600^2	66M	37.0G	55.1	84.3
• EffNetV2-L [72]	480^2	120M	53.0G	83.7	85.7
○ DeiT-S [73]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [73]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
• ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
• ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
• ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
• ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
• ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
• ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**

Results

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
● RegNetY-16G [54]	224^2	84M	16.0G	334.7	82.9
● EffNet-B7 [71]	600^2	66M	37.0G	55.1	84.3
● EffNetV2-L [72]	480^2	120M	53.0G	83.7	85.7
○ DeiT-S [73]	224^2	22M	4.6G	978.5	79.8
○ DeiT-B [73]	224^2	87M	17.6G	302.1	81.8
○ Swin-T	224^2	28M	4.5G	757.9	81.3
● ConvNeXt-T	224^2	29M	4.5G	774.7	**82.1**
○ Swin-S	224^2	50M	8.7G	436.7	83.0
● ConvNeXt-S	224^2	50M	8.7G	447.1	**83.1**
○ Swin-B	224^2	88M	15.4G	286.6	83.5
● ConvNeXt-B	224^2	89M	15.4G	292.1	**83.8**
○ Swin-B	384^2	88M	47.1G	85.1	84.5
● ConvNeXt-B	384^2	89M	45.0G	95.7	**85.1**
● ConvNeXt-L	224^2	198M	34.4G	146.8	**84.3**
● ConvNeXt-L	384^2	198M	101.0G	50.4	**85.5**

Results

ImageNet-22K pre-trained models

● R-101x3 [39]	384^2	388M	204.6G	-	84.4
● R-152x4 [39]	480^2	937M	840.5G	-	85.4
● EffNetV2-L [72]	480^2	120M	53.0G	83.7	86.8
● EffNetV2-XL [72]	480^2	208M	94.0G	56.5	87.3
○ ViT-B/16 (☎) [67]	384^2	87M	55.5G	93.1	85.4
○ ViT-L/16 (☎) [67]	384^2	305M	191.1G	28.5	86.8
● ConvNeXt-T	224^2	29M	4.5G	774.7	**82.9**
● ConvNeXt-T	384^2	29M	13.1G	282.8	**84.1**
● ConvNeXt-S	224^2	50M	8.7G	447.1	**84.6**
● ConvNeXt-S	384^2	50M	25.5G	163.5	**85.8**
○ Swin-B	224^2	88M	15.4G	286.6	85.2
● ConvNeXt-B	224^2	89M	15.4G	292.1	**85.8**
○ Swin-B	384^2	88M	47.0G	85.1	86.4
● ConvNeXt-B	384^2	89M	45.1G	95.7	**86.8**
○ Swin-L	224^2	197M	34.5G	145.0	86.3
● ConvNeXt-L	224^2	198M	34.4G	146.8	**86.6**
○ Swin-L	384^2	197M	103.9G	46.0	87.3
● ConvNeXt-L	384^2	198M	101.0G	50.4	**87.5**
● ConvNeXt-XL	224^2	350M	60.9G	89.3	**87.0**
● ConvNeXt-XL	384^2	350M	179.0G	30.2	**87.8**

Results

model	#param.	FLOPs	throughput (image / s)	training mem. (GB)	IN-1K acc.
○ ViT-S	22M	4.6G	978.5	4.9	79.8
● ConvNeXt-S (*iso.*)	22M	4.3G	1038.7	4.2	79.7
○ ViT-B	87M	17.6G	302.1	9.1	81.8
● ConvNeXt-B (*iso.*)	87M	16.9G	320.1	7.7	82.0
○ ViT-L	304M	61.6G	93.1	22.5	82.6
● ConvNeXt-L (*iso.*)	306M	59.7G	94.4	20.4	82.6

TRANSFORMER IS BACK…AGAIN

Masked Autoencoders Are Scalable Vision Learners
Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, Ross Girshick Facebook AI Research (FAIR)

In a nutshell…

- Masked Autoencoder (MAE)
  - MAE are scalable self-supervised learners for computer vision
  - Mask random patches of the input image and reconstruct the missing pixels
  - Asymmetric Encoder-Decoder Architecture
    - Visible subset of patches (without mask tokens) → Encoder
    - Latent representation & Mask tokens → Decoder (lightweight)
  - Masking high proportion of the input image, e.g., 75%, yields a non-trivial and meaningful self-supervisory task
  - Accelerate training (by 3× or more) and improves accuracy
  - Learning high-capacity models that generalizes well
    - Vanilla ViT-H [] achieves the best accuracy (87.8%) among methods that use only ImageNet-1K data
  - Transfer performance in downstream tasks outperforms supervised pre- training and shows promising scaling behavior

[1] He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Background

- Masked Language Modeling
  - Success of self-supervised pre-training in NLP
    - Masked language modeling (e.g., BERT [])
    - Autoregressive language modeling (e.g., GPT [])
  - Method: remove a portion of the input sequence and learn to predict the missing content
  - These methods have been shown to scale excellently
  - These pre-trained representations generalize well to various downstream tasks

[1] He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.
[2]

Background

- Autoencoder
  - Encoder maps an input to a latent representation
  - Decoder reconstructs the input
  - E.g., PCA and k-means are autoencoders
  - Denoising autoencoders (DAE) [1] are a class of autoencoders that corrupt an input signal and learn to reconstruct the original, uncorrupted signal
  - A series of methods can be thought of as a generalized DAE under different corruptions, e.g., masking pixels or removing color channels

- Masked Image Encoding
  - DAE [1] presents masking as a noise type
  - Convolution-based
    - Context Encoder inpaints large missing regions
  - Transformer-based
    - iGPT operates on sequences of pixels and predicts unknown pixels
    - The ViT studies masked patch prediction for self-supervised learning
    - Most recently, BEiT proposes to predict discrete tokens

[1] He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Background

- Self-supervised Learning
  - Early self-supervised learning approaches often focused on different pretext tasks for pre-training
  - Contrastive learning has been popular, which models image similarity and dissimilarity between two or more views
  - Contrastive and related methods strongly depend on data augmentation
  - Autoencoding pursues a conceptually different direction, and it exhibits different behaviors

What makes masked autoencoding different between vision and language?

- **Architectural Gap**
  - Convolutions typically operate on regular grids
  - It is not straightforward to integrate 'indicators' such as mask tokens or positional embeddings into convolutional networks
  - **Solution: ViT [1]**

- **Information Density**
  - Language: human-generated signals, highly semantic, information-dense, Need sophisticated language understanding to train a model to predict only a few missing words per sentence
  - Images: natural signals, heavy spatial redundancy, A missing patch can be recovered from neighboring patches with little high-level understanding of parts, objects, and scenes
  - **Solution: masking a very high portion of random patches**
  - This strategy largely reduces redundancy and creates a challenging self-supervisory task that requires holistic understanding beyond low-level image statistics

- **The Role of Autoencoder's Decoder**
  - Language: decoder predicts missing words that contain rich semantic information
  - Image: decoder reconstructs pixels, hence its output is of a lower semantic level than common recognition tasks
  - **Decoder design plays a key role in determining the semantic level of the learned latent representations for images**

Masked Autoencoder (MAE)

MAE architecture

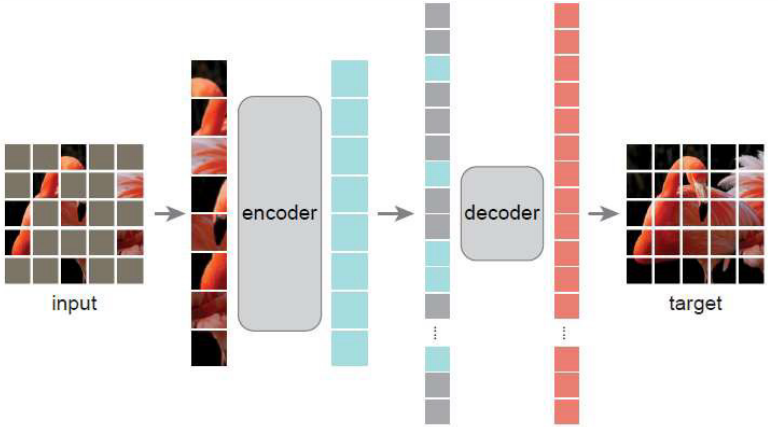


Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images to produce representations for recognition tasks.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Masked Autoencoder (MAE)

- MAE: A Simple, Effective, and Scalable Self-supervised Learner
  - MAE masks random patches from the input image and reconstructs the missing patches in the pixel space
  - Asymmetric encoder-decoder design
    - Encoder operates only on the visible subset of patches (without mask tokens)
    - (Lightweight) Decoder reconstructs the input from the latent representation along with mask tokens
    - Shifting the mask tokens to the small decoder results in a large reduction in computation
  - A very high masking ratio (e.g., 75%) can achieve a win-win scenario
    - It optimizes accuracy while allowing the encoder to process only a small portion (e.g., 25%) of patches
    - This can reduce overall pre-training time by 3 × or more and likewise reduce memory consumption, enabling us to scale our MAE to large models easily
  - With MAE pre-training,
    - Vanilla ViT-H achieves 87.8% accuracy when fine-tuned on ImageNet-1K
    - Transfer Learning on object detection, instance segmentation, semantic segmentation achieve better results than its supervised pre-training counterparts
    - Significant gains by scaling up models

MAE Reconstruction

- ImageNet validation images
  - Masking ratio 80%
  - No loss is computed on visible patches (i.e., Loss is only computed on masked patches)

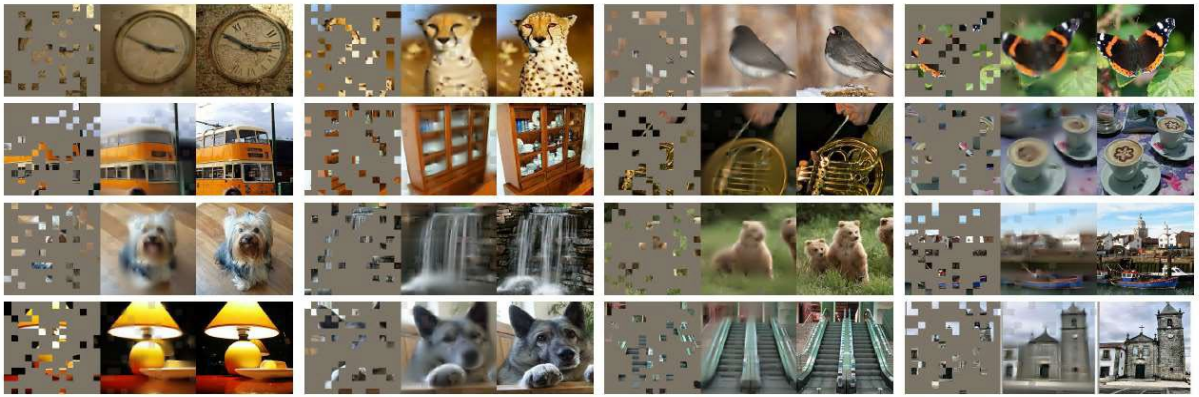


Figure 2. Example results on ImageNet *validation* images. For each triplet, we show the masked image (left), our MAE reconstruction† (middle), and the ground-truth (right). The masking ratio is 80%, leaving only 39 out of 196 patches. More examples are in the appendix.
†*As no loss is computed on visible patches, the model output on visible patches is qualitatively worse. One can simply overlay the output with the visible patches to improve visual quality. We intentionally opt not to do this, so we can more comprehensively demonstrate the method's behavior.*

MAE Reconstruction

- COCO validation images
  - MAE trained on ImageNet

Figure 3. Example results on COCO validation images, using an MAE trained on ImageNet (the same model weights as in Figure 2). Observe the reconstructions on the two right-most examples, which, although different from the ground truth, are semantically plausible.

MAE Reconstruction

- ImageNet validation images
  - MAE pre-trained with a masking ratio 75%
  - Applied on inputs with higher masking ratios (75%, 85%, 95%)

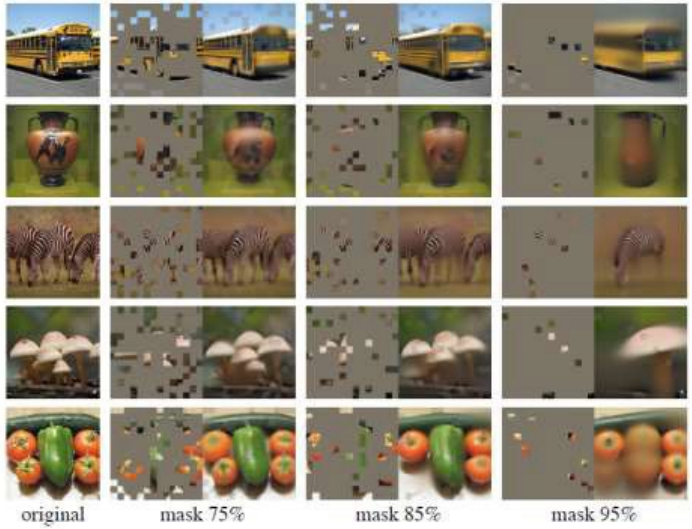


Figure 4. Reconstructions of ImageNet *validation* images using an MAE pre-trained with a masking ratio of 75% but applied on inputs with higher masking ratios. The predictions differ plausibly from the original images, showing that the method can generalize.

Masked Autoencoder (MAE)

- Masking
  - Image is divided into regular non-overlapping patches
  - Sample a subset of patches and mask (i.e., remove) the remaining ones
  - High masking ratio (e.g., 75%)
- MAE encoder
  - Encoder is a ViT [] but applied only on visible, unmasked patches
  - Encoder only operates on a small subset (e.g., 25%) of the full set
  - Masked patches are removed; No mask tokens are used
  - This allows us to train very large encoders with only a fraction of compute and memory
- MAE decoder
  - The input to the MAE decoder is the full set of tokens consisting of encoded visible patches and mask tokens
  - Positional embeddings are added to all tokens in this full set
  - MAE decoder is only used during pre-training to perform the image reconstruction task (only the encoder is used to produce image representations for recognition)
  - Decoder has <10% computation per token vs. the encoder

Masked Autoencoder (MAE)

- Reconstruction Target
  - ✓ MAE reconstructs the input by predicting the pixel values for each masked patch
  - ✓ Loss function computes the MSE between reconstructed and original images in the pixel space
  - ✓ Loss is only computed on masked patches, similar to BERT []
  - ✓ masks random patches from the input image and reconstructs the missing patches in the pixel space
- Simple Implementation
  1. Generate a token for every input patch
  2. Randomly shuffle the list of tokens and remove the last portion of the list
  3. After encoding, mask tokens are appended to the list of encoded patches, and unshuffle the full list (inverting the random shuffle operation)
  4. The decoder is applied to this full list (with positional embeddings added)

ImageNet Experiments

- Experimental Settings
  - Self-supervised pre-training on the ImageNet-1K training set →
    - supervised training to evaluate the representation with
      - End-to-end fine-tuning
      - Linear probing (freeze backbone and only train linear layer)
- ViT-L trained from scratch vs. fine-tuned from MAE
  - Backbone: ViT-L/16
  - Scratch: 200 epochs vs. Fine-tuning: 50 epochs

scratch, original [16]	scratch, our impl.	baseline MAE
76.5	82.5	84.9

✓ Fine-tuning accuracy heavily depends on pre-training

ImageNet Experiments

- Masking ratio
  - ✓ BERT typical masking ratio: 15%

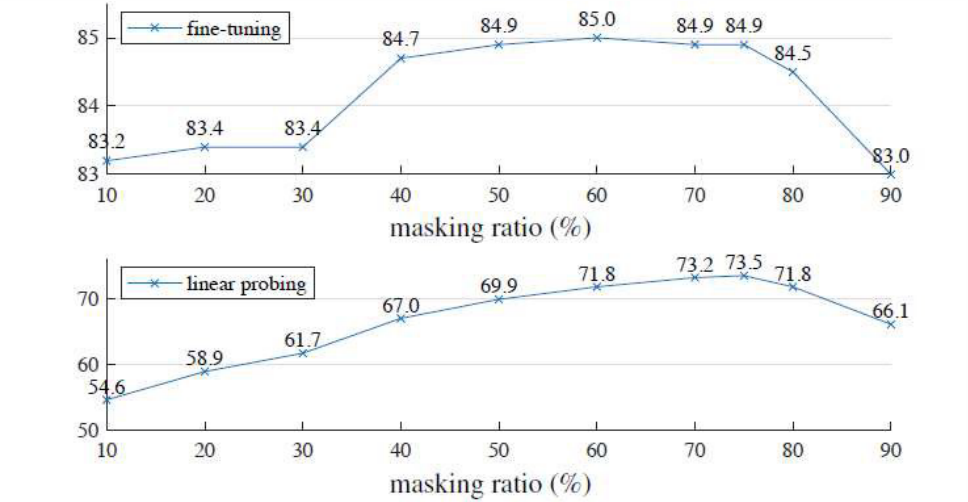


Figure 5. **Masking ratio**. A high masking ratio (75%) works well for both fine-tuning (top) and linear probing (bottom). The y-axes are ImageNet-1K validation accuracy (%) in all plots in this paper.

ImageNet Experiments: Ablations

blocks	ft	lin
1	84.8	65.5
2	**84.9**	70.0
4	**84.9**	71.9
8	**84.9**	**73.5**
12	84.4	73.3

(a) **Decoder depth**. A deep decoder can improve linear probing accuracy.

dim	ft	lin
128	**84.9**	69.1
256	84.8	71.3
512	**84.9**	**73.5**
768	84.4	73.1
1024	84.3	73.1

(b) **Decoder width**. The decoder can be narrower than the encoder (1024-d).

case	ft	lin	FLOPs
encoder w/ [M]	84.2	59.6	3.3×
encoder w/o [M]	**84.9**	**73.5**	**1**×

(c) **Mask token**. An encoder without mask tokens is more accurate and faster (Table 2).

case	ft	lin
pixel (w/o norm)	84.9	73.5
pixel (w/ norm)	**85.4**	**73.9**
PCA	84.6	72.3
dVAE token	85.3	71.6

(d) **Reconstruction target**. Pixels as reconstruction targets are effective.

case	ft	lin
none	84.0	65.7
crop, fixed size	84.7	73.1
crop, rand size	**84.9**	**73.5**
crop + color jit	84.3	71.9

(e) **Data augmentation**. Our MAE works with minimal or no augmentation.

case	ratio	ft	lin
random	75	**84.9**	**73.5**
block	50	83.9	72.3
block	75	82.8	63.9
grid	75	84.0	66.0

(f) **Mask sampling**. Random sampling works the best. See Figure 6 for visualizations.

Table 1. **MAE ablation experiments** with ViT-L/16 on ImageNet-1K. We report fine-tuning (ft) and linear probing (lin) accuracy (%). If not specified, the default is: the decoder has depth 8 and width 512, the reconstruction target is unnormalized pixels, the data augmentation is random resized cropping, the masking ratio is 75%, and the pre-training length is 800 epochs. Default settings are marked in gray.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

ImageNet Experiments: Wall-clock time (w/wo mask)

encoder	dec. depth	ft acc	hours	speedup
ViT-L, w/ [M]	8	84.2	42.4	-
ViT-L	8	84.9	15.4	2.8×
ViT-L	1	84.8	11.6	**3.7**×
ViT-H, w/ [M]	8	-	119.6†	-
ViT-H	8	85.8	34.5	3.5×
ViT-H	1	85.9	29.3	**4.1**×

Table 2. **Wall-clock time** of our MAE training (800 epochs), benchmarked in 128 TPU-v3 cores with TensorFlow. The speedup is relative to the entry whose encoder has mask tokens (gray). The decoder width is 512, and the mask ratio is 75%. †: This entry is estimated by training ten epochs.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

ImageNet Experiments: Mask sampling strategies

Figure 6. **Mask sampling strategies** determine the pretext task difficulty, influencing reconstruction quality and representations (Table 1f). Here each output is from an MAE trained with the specified masking strategy. Left: random sampling (our default). Middle: block-wise sampling [2] that removes large random blocks. Right: grid-wise sampling that keeps one of every four patches. Images are from the validation set.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

ImageNet Experiments

- Training Schedules
  - ✓ Not observed saturation of linear probing accuracy even at 1600 epochs
  - ✓ MoCo v3 saturates at 300 epochs for ViT-L

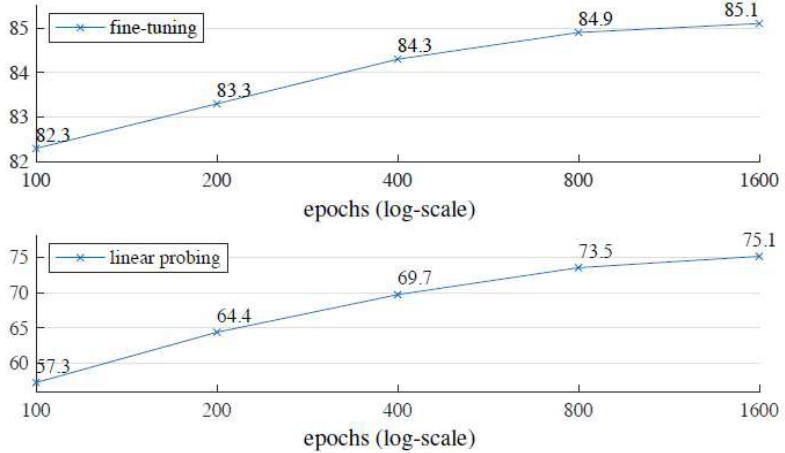


Figure 7. **Training schedules**. A longer training schedule gives a noticeable improvement. Here each point is a full training schedule. The model is ViT-L with the default setting in Table 1.

Comparisons with self-supervised methods

method	pre-train data	ViT-B	ViT-L	ViT-H	ViT-H_{448}
scratch, our impl.	-	82.3	82.6	83.1	-
DINO [5]	IN1K	82.8	-	-	-
MoCo v3 [9]	IN1K	83.2	84.1	-	-
BEiT [2]	IN1K+DALLE	83.2	85.2	-	-
MAE	IN1K	83.6	85.9	86.9	**87.8**

Table 3. **Comparisons with previous results on ImageNet-1K**. The pre-training data is the ImageNet-1K training set (except the tokenizer in BEiT was pre-trained on 250M DALLE data [43]). All self-supervised methods are evaluated by end-to-end fine-tuning. The ViT models are B/16, L/16, H/14 [16]. The best for each column is underlined. All results are on an image size of 224, except for ViT-H with an extra result on 448. Here our MAE reconstructs normalized pixels and is pre-trained for 1600 epochs.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Comparisons with Previous Results

- Comparisons with supervised pre-training

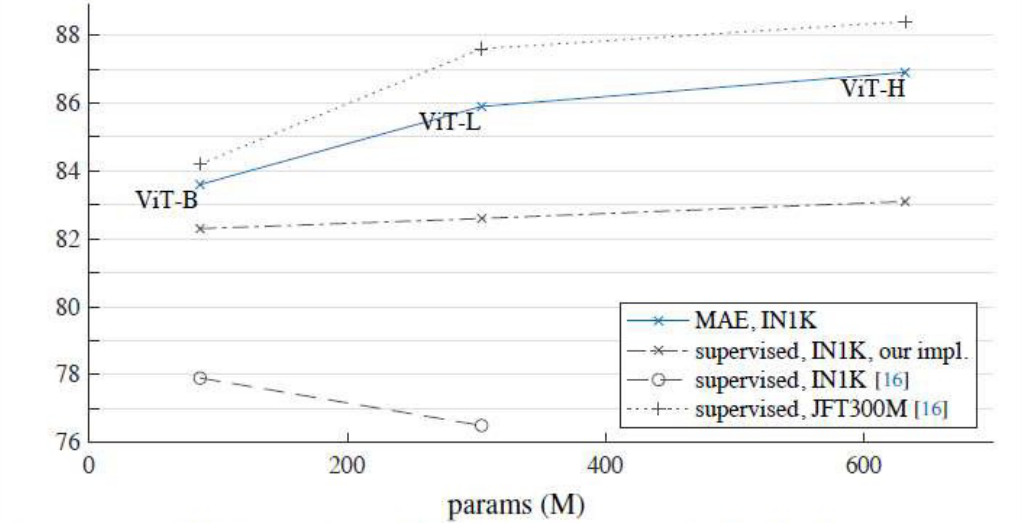


Figure 8. **MAE pre-training *vs.* supervised pre-training**, evaluated by fine-tuning in ImageNet-1K (224 size). We compare with the original ViT results [16] trained in IN1K or JFT300M.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Partial Fine-tuning

- MAE vs. MoCo v3 (w.r.t. # fine-tuned transformer blocks)

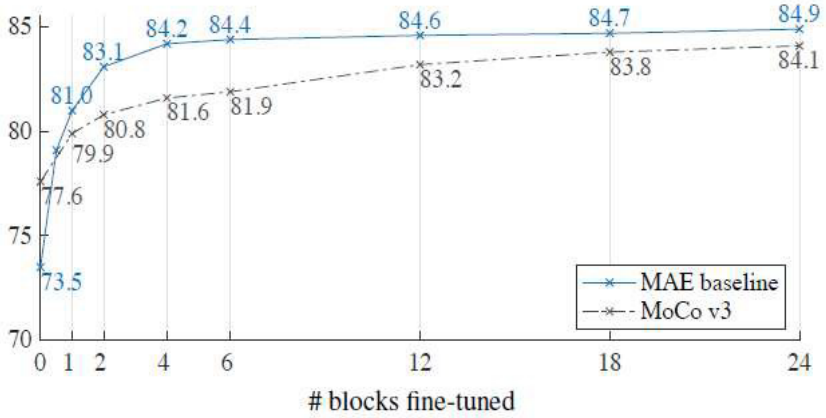


Figure 9. **Partial fine-tuning** results of ViT-L w.r.t. the number of fine-tuned Transformer blocks under the default settings from Table 1. Tuning 0 blocks is linear probing; 24 is full fine-tuning. Our MAE representations are less linearly separable, but are consistently better than MoCo v3 if one or more blocks are tuned.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Transfer Learning Experiments

- Object Detection and Segmentation

		AP^{box}		AP^{mask}	
method	pre-train data	ViT-B	ViT-L	ViT-B	ViT-L
supervised	IN1K w/ labels	47.9	49.3	42.9	43.9
MoCo v3	IN1K	47.9	49.3	42.7	44.0
BEiT	IN1K+DALLE	49.8	**53.3**	44.4	47.1
MAE	IN1K	**50.3**	**53.3**	**44.9**	**47.2**

Table 4. **COCO object detection and segmentation** using a ViT Mask R-CNN baseline. All entries are based on our implementation. Self-supervised entries use IN1K data *without* labels. Mask AP follows a similar trend as box AP.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Transfer Learning Experiments

- Semantic Segmentation

method	pre-train data	ViT-B	ViT-L
supervised	IN1K w/ labels	47.4	49.9
MoCo v3	IN1K	47.3	49.1
BEiT	IN1K+DALLE	47.1	53.3
MAE	IN1K	**48.1**	**53.6**

Table 5. **ADE20K semantic segmentation** (mIoU) using UperNet. BEiT results are reproduced using the official code. Other entries are based on our implementation. Self-supervised entries use IN1K data *without* labels.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

Transfer Learning Experiments

- Pixels vs. Tokens

	IN1K			COCO		ADE20K	
	ViT-B	ViT-L	ViT-H	ViT-B	ViT-L	ViT-B	ViT-L
pixel (w/o norm)	83.3	85.1	86.2	49.5	52.8	48.0	51.8
pixel (w/ norm)	83.6	85.9	86.9	50.3	53.3	48.1	53.6
dVAE token	83.6	85.7	86.9	50.3	53.2	48.1	53.4
△	0.0	-0.2	0.0	0.0	-0.1	0.0	-0.2

Table 6. **Pixels *vs.* tokens** as the MAE reconstruction target. △ is the difference between using dVAE tokens and using normalized pixels. The difference is statistically insignificant.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." arXiv 2021.

HOW ABOUT CNN?

ConvNext v2 is back…again

ConvNextV2

Backbone	Method	FLOPS	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
ConvNeXt V1-B	Supervised	486G	50.3	71.6	56.1	44.9	68.5	48.8
ConvNeXt V2-B	Supervised	486G	51.0	72.4	56.6	45.6	69.5	49.7
Swin-B	SimMIM	497G	52.3	–	–	–	–	–
ConvNeXt V2-B	FCMAE	486G	**52.9**	72.6	58.9	**46.6**	70.0	51.1
ConvNeXt V1-L	Supervised	875G	50.6	71.5	56.3	45.1	68.7	49.2
ConvNeXt V2-L	Supervised	875G	51.5	72.5	57.3	45.8	69.4	49.9
Swin-L	SimMIM	904G	53.8	–	–	–	–	–
ConvNeXt V2-L	FCMAE	875G	**54.4**	73.9	60.4	**47.7**	71.4	52.3
Swin V2-H	SimMIM	–	54.4	–	–	–	–	–
ConvNeXt V2-H	FCMAE	2525G	**55.7**	75.2	61.8	**48.9**	72.8	53.6

Table 6. **COCO object detection and instance segmentation results** using Mask-RCNN. FLOPS are calculated with image size (1280, 800). Swins' results are from [77]. All COCO fine-tuning experiments rely on ImageNet-1K pre-trained models.

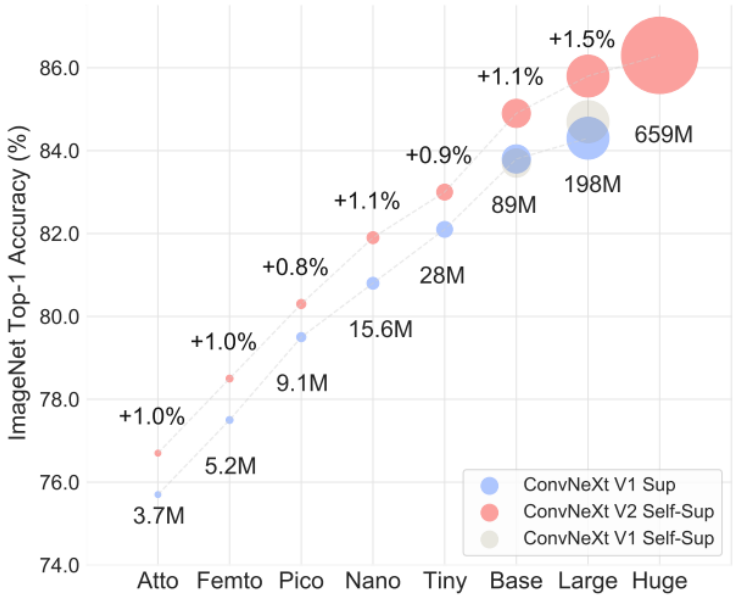
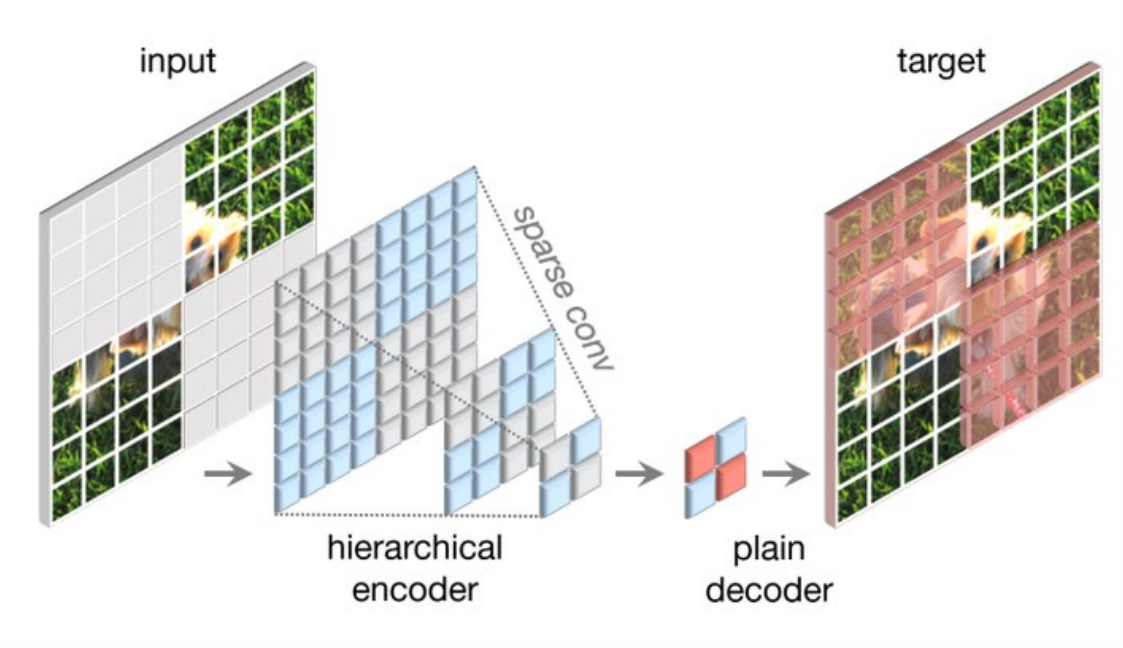


Figure 1. **ConvNeXt V2 model scaling**. The ConvNeXt V2 model, which has been pre-trained using our fully convolutional masked autoencoder framework, performs significantly better than the previous version across a wide range of model sizes.

arxiv.org/pdf/2301.00808v1.pdf

ConvNextV2

- Combine with mask auto-encoder



ConvNextV2

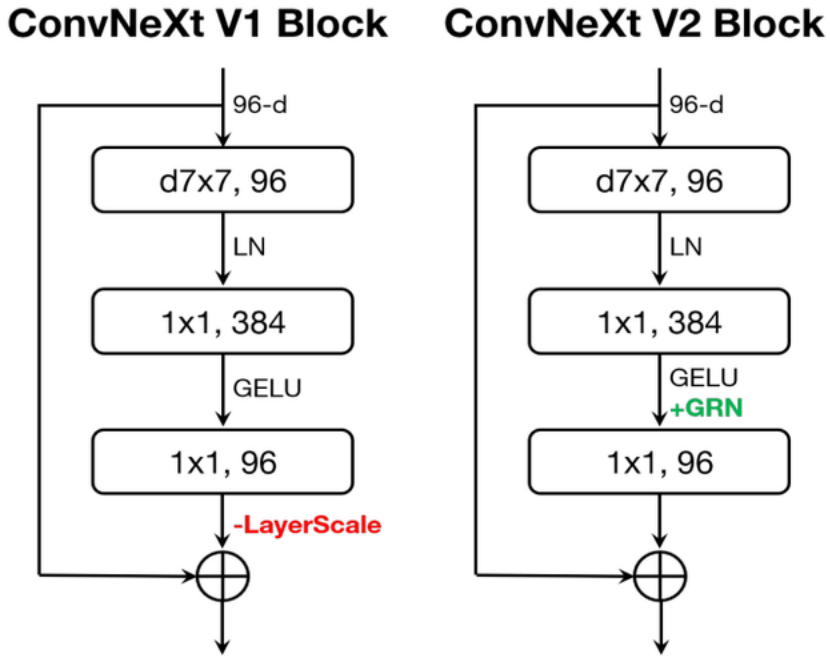


Figure 5. **ConvNeXt Block Designs**. In ConvNeXt V2, we add the GRN layer after the dimension-expansion MLP layer and drop LayerScale [65] as it becomes redundant.

GRN of ConvNextV2

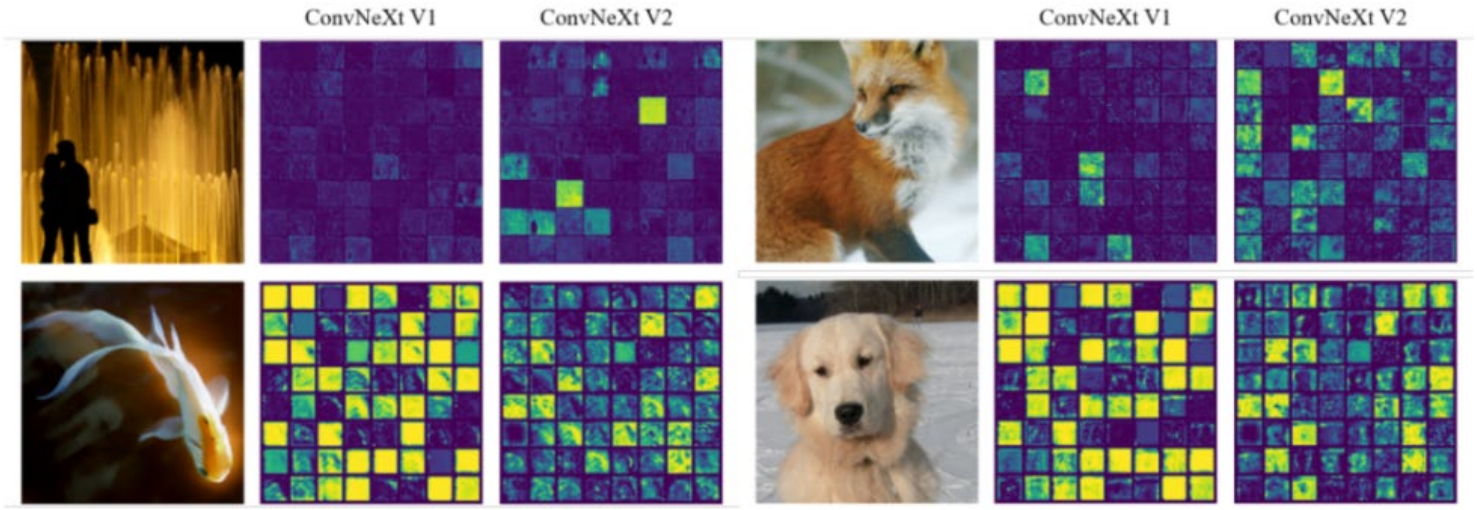
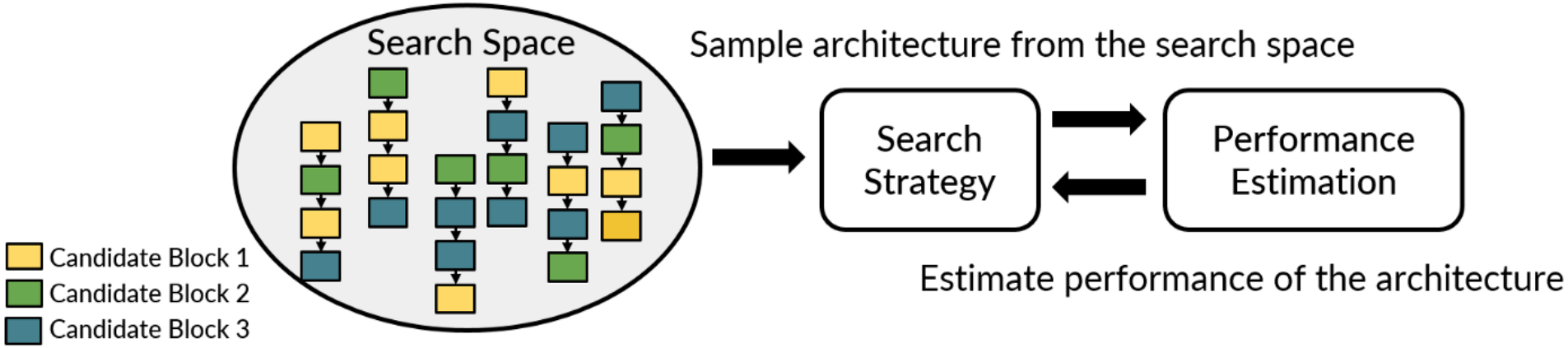


Figure 3. **Feature activation visualization.** We visualize the activation map for each feature channel in small squares. For clarity, we display 64 channels in each visualization. The ConvNeXt V1 model suffers from a feature collapse issue, which is characterized by the presence of redundant activations (dead or saturated neurons) across channels. To fix this problem, we introduce a new method to promote feature diversity during training: the global response normalization (GRN) layer. This technique is applied to high-dimensional features in every block, leading to the development of the ConvNeXt V2 architecture.

NETWORK SEARCH ARCHITECTURE (NAS)

NAS Common Architecture

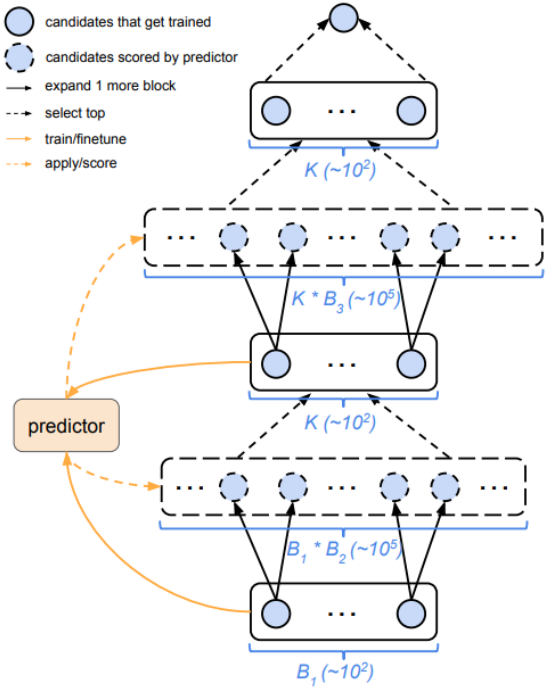


Progressive NAS

- Extra controller (called predictor)
  - Predict the "accuracy" of targeted model!

- 1. Find N possible architectures
  - Predict their accuracy
  - Find the top-k architectures
- 2. Train K models to find their result
- 3. Use the trained K models to update the predictor

- Repeat to converge!



Summary: CNN Architectures

- Many popular architectures available in model zoos
- ResNet and SENet currently good defaults to use
- Networks have gotten increasingly deep over time
- Many other aspects of network architectures are also continuously being investigated and improved
- Even more recent trend towards meta-learning/Few-shot learning

- Next time: Object detection via deep learning