CNN Architectures

CS60010: Deep Learning

Abir Das

IIT Kharagpur

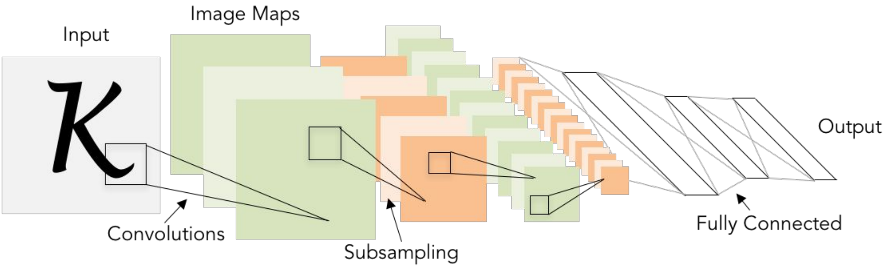
Feb 12 and 13, 2020

Agenda

To discuss in detail about some of the highly successful deep CNN architectures

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]

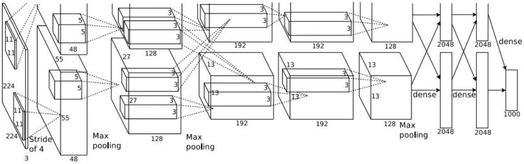
§ Citation of the paper as on Feb 06, 2020 is 24,262

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

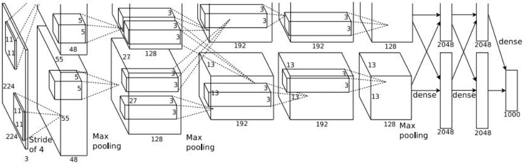
§ Citation of the paper as on Feb 06, 2020 is 56,120

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Input: 227x227x3 images

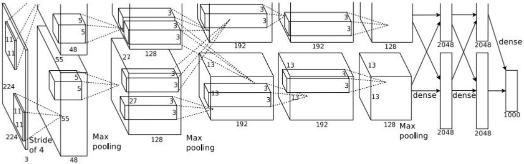
First layer (CONV1): 96 11x11 filters applied at stride 4 =>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Input: 227x227x3 images

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

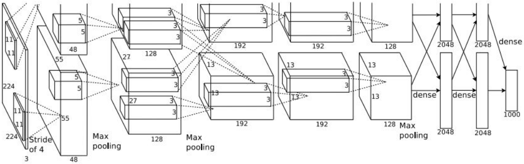
Q: What is the total number of parameters in this layer?

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Input: 227x227x3 images

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

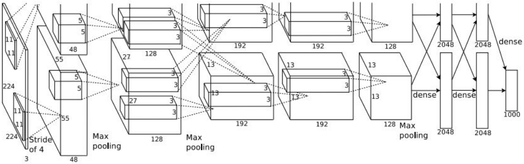
Parameters: (11*11*3)*96 = 35K

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

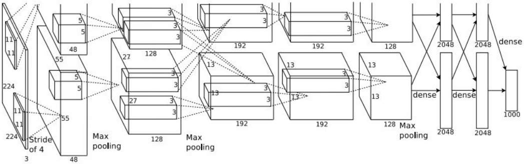
Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2
Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2.
Output volume: 27x27x96

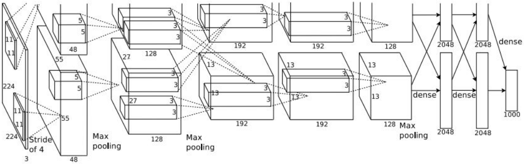
Q: what is the number of parameters in this layer?

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



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

Input: 227x227x3 images
After CONV1: 55x55x96
Second layer (POOL1): 3x3 filters applied at stride 2.
Output volume: 27x27x96

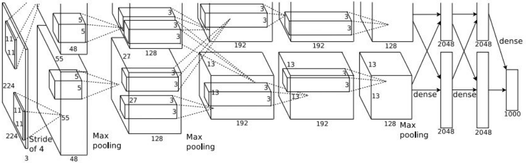
Parameters: 0!

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



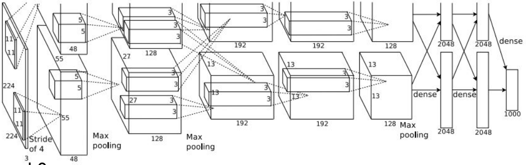
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

Source: CS231n course, Stanford University

AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



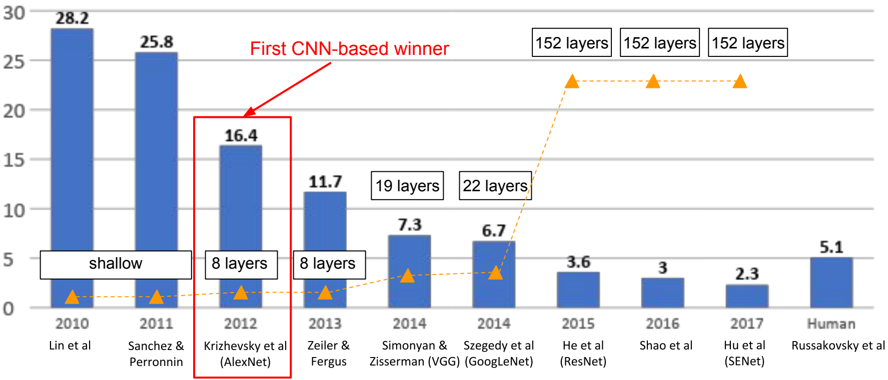
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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
- Pooling is overlapping
- 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%

Source: CS231n course, Stanford University

Imagenet Leaderboard

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Source: CS231n course, Stanford University

Just In

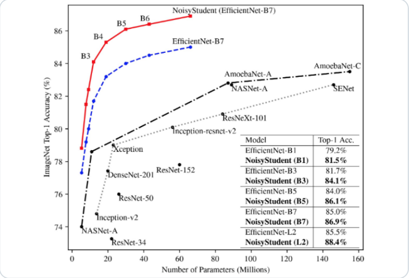
Quoc Le
@quocleix

Happy to announce that we've released a number of models trained with Noisy Student (a semi-supervised learning method). The best model achieves 88.4% top-1 accuracy on ImageNet (SOTA).

Enjoy finetuning!

Link: github.com/tensorflow/tpu...

Paper: arxiv.org/abs/1911.04252



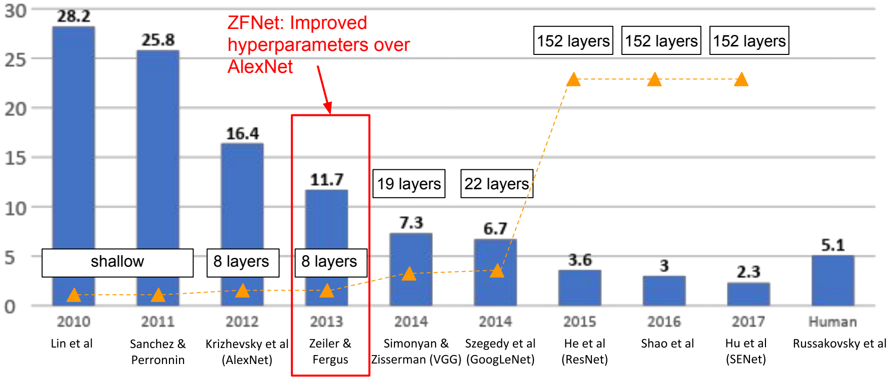
Model	Top-1 Acc.
EfficientNet-B1	79.2%
NoisyStudent (B1)	**81.5%**
EfficientNet-B3	81.7%
NoisyStudent (B3)	**84.1%**
EfficientNet-B5	84.0%
NoisyStudent (B5)	**86.1%**
EfficientNet-B7	85.0%
NoisyStudent (B7)	**86.9%**
EfficientNet-L2	85.5%
NoisyStudent (L2)	**88.4%**

3:51 AM · Feb 12, 2020 · Twitter Web App

Source: Twitter

Imagenet Leaderboard

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

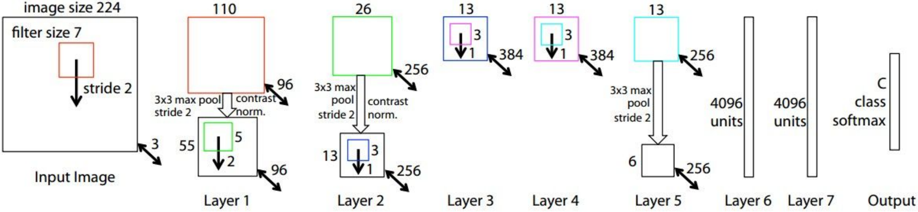


Source: CS231n course, Stanford University

ZFNet

ZFNet

[Zeiler and Fergus, 2013]



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

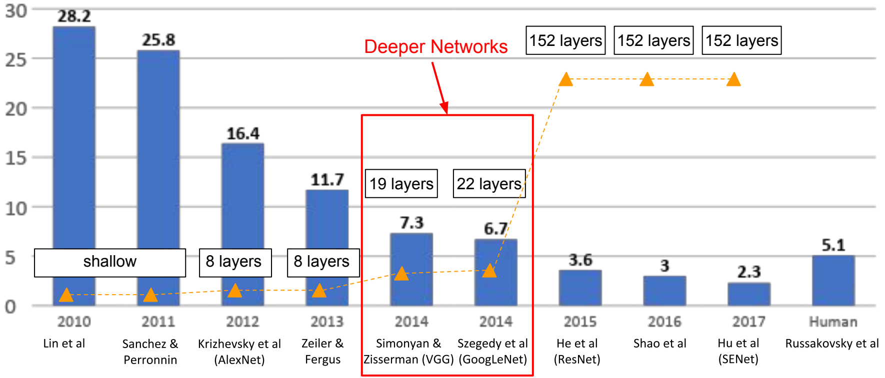
ImageNet top 5 error: 16.4% -> 11.7%

§ Citation of the paper as on Feb 06, 2020 is 8,360

Source: CS231n course, Stanford University

ZFNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Source: CS231n course, Stanford University

VGG

Case Study: VGGNet

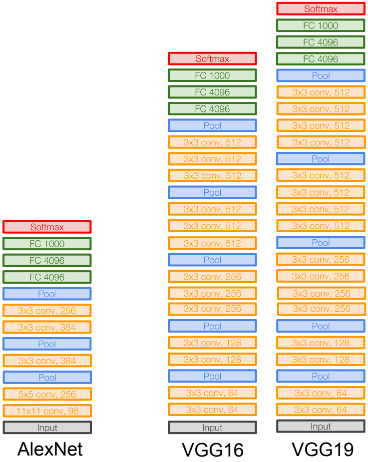
[Simonyan and Zisserman, 2014]

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



§ Citation of the paper as on Feb 06, 2020 is 33,431

Source: CS231n course, Stanford University

VGG

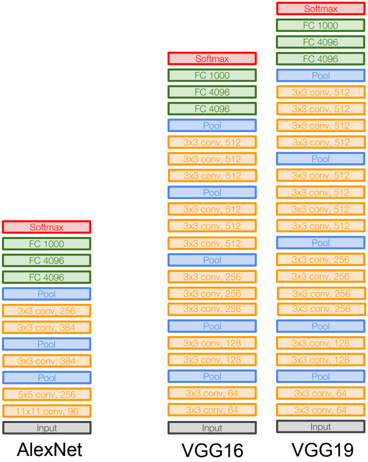
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

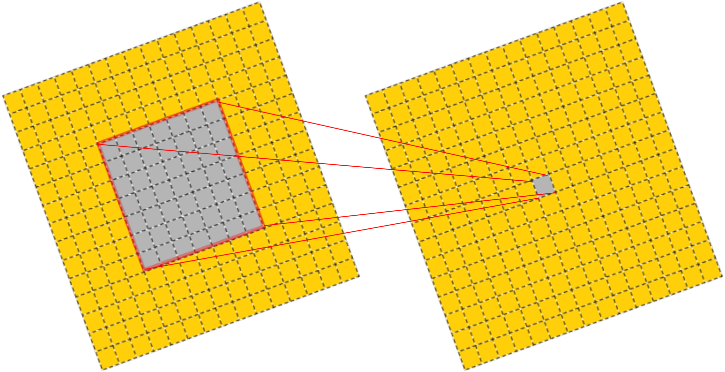
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?



Source: CS231n course, Stanford University

VGG

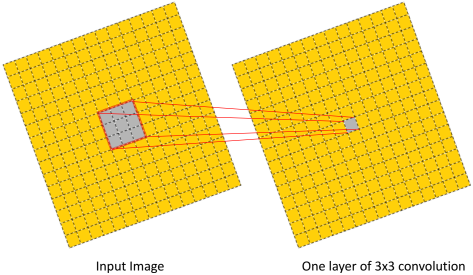
§ Receptive field is the region in the input space that a particular CNN's feature (activation value) is looking at (or getting computed due to)



Input Image

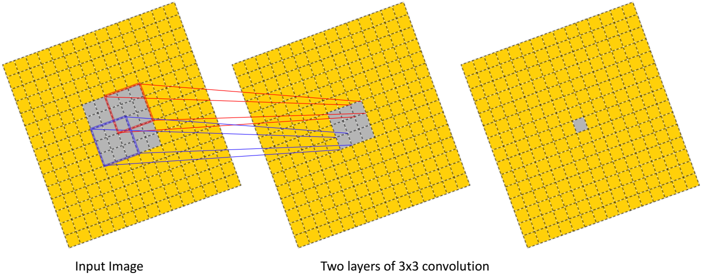
One layer of 7x7 convolution

VGG



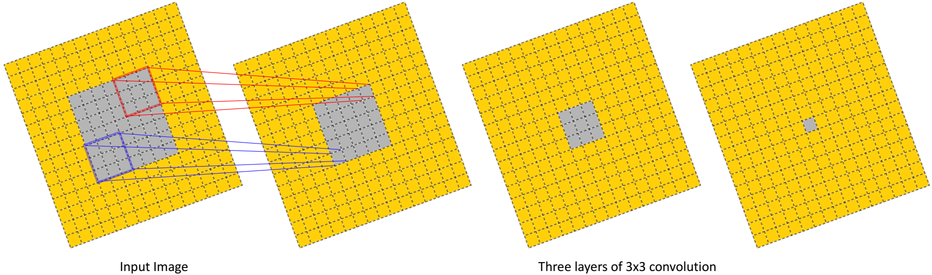
- Receptive field is 3×3

VGG



- Receptive field is 5×5

VGG



- Receptive field is 7×7

VGG

Case Study: VGGNet

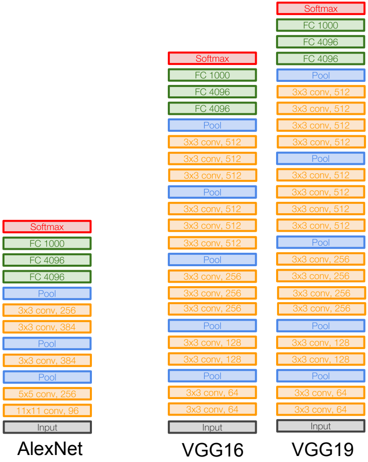
[Simonyan and Zisserman, 2014]

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



Source: CS231n course, Stanford University

VGG

(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

TOTAL memory: 15.2M * 4 bytes ~= 58MB / image (for a forward pass)
TOTAL params: 138M parameters

Source: CS231n course, Stanford University

VGG

(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

Note:

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 15.2M * 4 bytes ~= 58MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

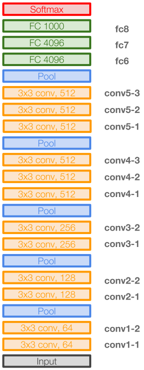
Source: CS231n course, Stanford University

VGG

(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: 15.2M * 4 bytes ~= 58MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

Common names

Source: CS231n course, Stanford University

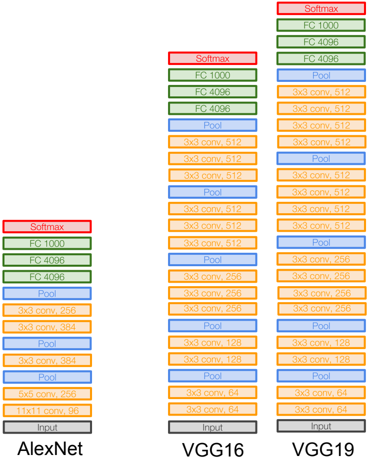
VGG

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

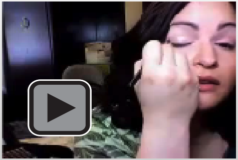


Source: CS231n course, Stanford University

Video Classification

Examples from UCF-101 dataset.

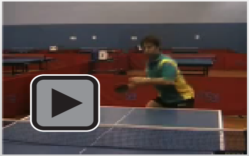
ApplyEyeMakeup



CuttingInKitchen

BalanceBeam

TableTennisShot



C3D

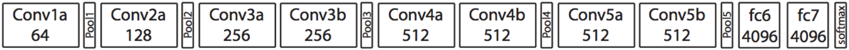


Figure 3. **C3D architecture**. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. Number of filters are denoted in each box. The 3D pooling layers are denoted from `pool1` to `pool5`. All pooling kernels are $2 \times 2 \times 2$, except for `pool1` is $1 \times 2 \times 2$. Each fully connected layer has 4096 output units.

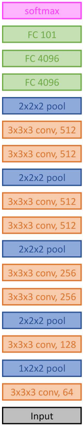
§ Citation of the paper as on Feb 12, 2020 is 2,907

C3D

INPUT: [16x112x112x3] memory: 16*112*112*3=602K params: 0 (not counting biases)
CONV1a: [16x112x112x64] memory: 16*112*112*64=12.8M params: 3*(3*3*3)*64 = 5,184
POOL1: [16x56x56x64] memory: 16*56*56*64=3.2M params: 0
CONV2a: [16x56x56x128] memory: 16*56*56*128=6.4M params: 64*(3*3*3)*128 = 221,184
POOL2: [8x28x28x128] memory: 8*28*28*128=802K params: 0

CONV3a: [8x28x28x256] memory: 8*28*28*256=1.6M params: 128*(3*3*3)*256 = 884,736
CONV3b: [8x28x28x256] memory: 8*28*28*256=1.6M params: 256*(3*3*3)*256 = 1,769,472
POOL3: [4x14x14x256] memory: 4*14*14*256=200K params: 0
CONV4a: [4x14x14x512] memory: 4*14*14*512=401K params: 256*(3*3*3)*512 = 3,538,944
CONV4b: [4x14x14x512] memory: 4*14*14*512=401K params: 512*(3*3*3)*512 = 7,077,888
POOL4: [2x7x7x512] memory: 2*7*7*512=50K params: 0
CONV5a: [2x7X7x512] memory: 2*7*7*512=50K params: 512*(3*3*3)*512 = 7,077,888
CONV5b: [2x7X7x512] memory: 2*7*7*512=50K params: 512*(3*3*3)*512 = 7,077,888
POOL5: [1x4x4x512] memory: 1*4*4*512=8,192 params: 0
FC6: [4096] memory: 4096 params: 4*4*512*4096 = 33,554,432
FC: [4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [101] memory: 101 params: 4096*101 = 413,696

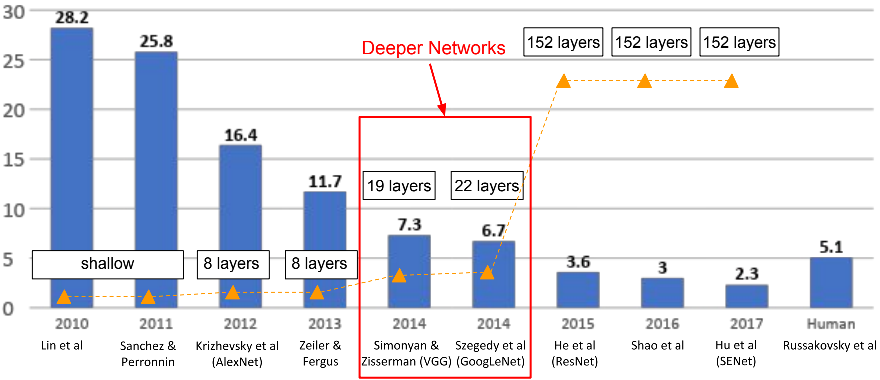
TOTAL memory: 28.3M * 4 bytes ~= 107.82MB / image (for a forward pass)
TOTAL params: 78.4M parameters



C3D

GoogLeNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

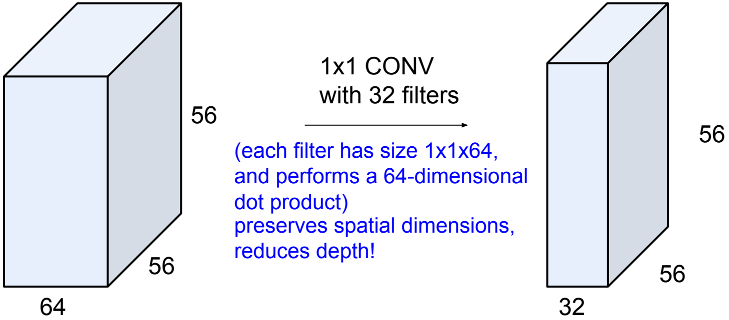


§ Citation of the paper as on Feb 12, 2020 is 19,313

Source: CS231n course, Stanford University

GoogLeNet

1x1 convolutions



Source: CS231n course, Stanford University

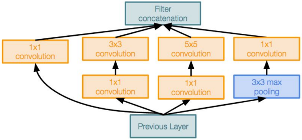
GoogLeNet

Case Study: GoogLeNet

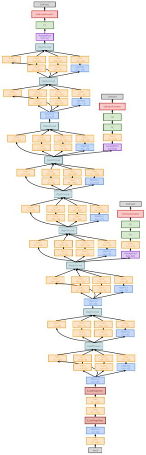
[Szegedy et al., 2014]

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



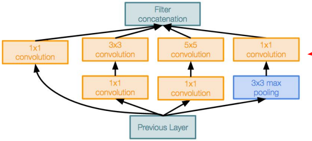
Source: CS231n course, Stanford University

GoogLeNet

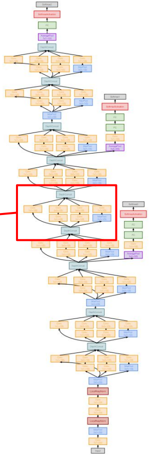
Case Study: GoogLeNet

[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



Inception module

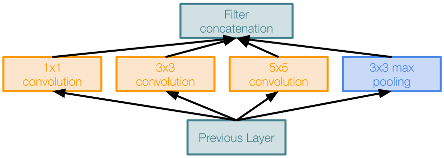


Source: CS231n course, Stanford University

GoogLeNet

Case Study: 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)

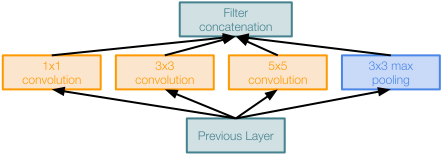
Concatenate all filter outputs together depth-wise

Source: CS231n course, Stanford University

GoogLeNet

Case Study: 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]

Source: CS231n course, Stanford University

GoogLeNet

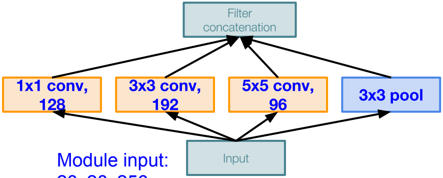
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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



Module input: 28x28x256

Naive Inception module

Source: CS231n course, Stanford University

GoogLeNet

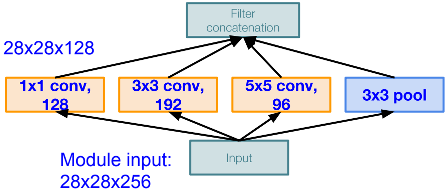
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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



Naive Inception module

Source: CS231n course, Stanford University

GoogLeNet

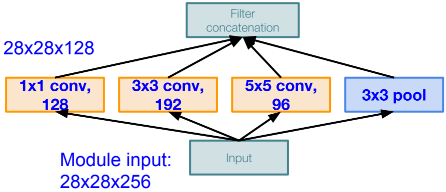
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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



Naive Inception module

Source: CS231n course, Stanford University

GoogLeNet

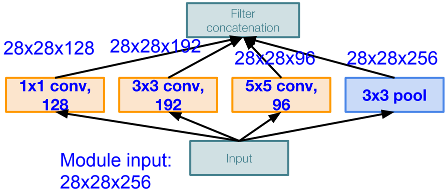
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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



Naive Inception module

Source: CS231n course, Stanford University

GoogLeNet

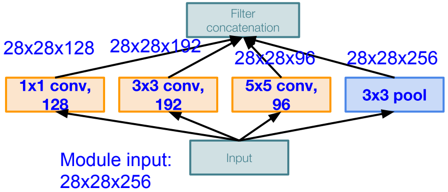
Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

Source: CS231n course, Stanford University

GoogLeNet

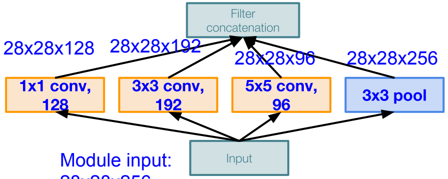
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?

28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Naive Inception module

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

Source: CS231n course, Stanford University

GoogLeNet

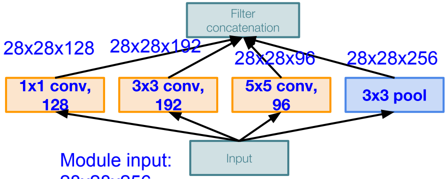
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?

28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Naive Inception module

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

Conv Ops:

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

Source: CS231n course, Stanford University

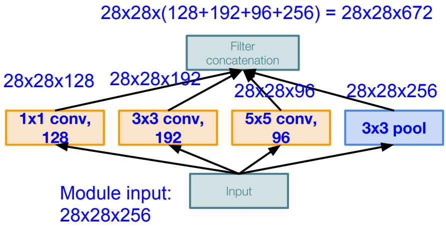
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

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

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!

Source: CS231n course, Stanford University

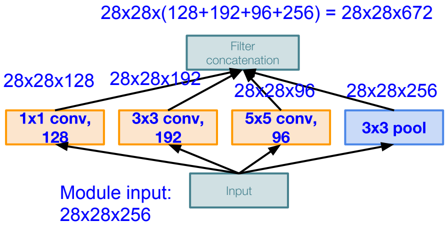
GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3:What is output size after filter concatenation?



Naive Inception module

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

Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth

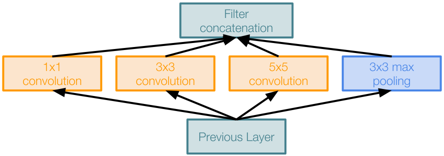
Source: CS231n course, Stanford University

GoogLeNet

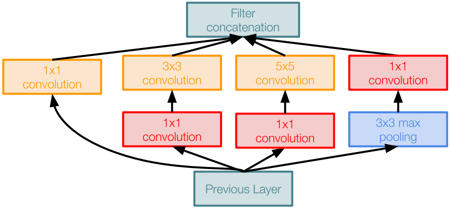
Case Study: GoogLeNet

[Szegedy et al., 2014]

1x1 conv "bottleneck" layers



Naive Inception module



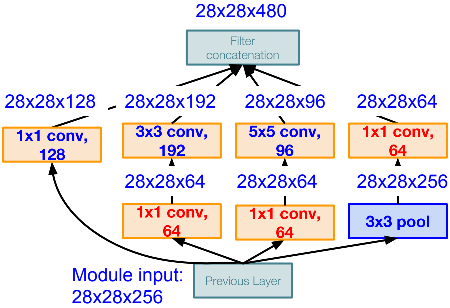
Inception module with dimension reduction

Source: CS231n course, Stanford University

GoogLeNet

Case Study: 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

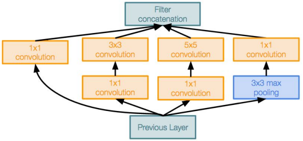
Source: CS231n course, Stanford University

GoogLeNet

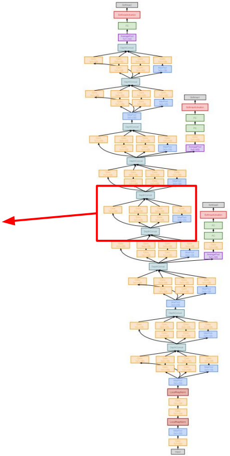
Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules with dimension reduction on top of each other



Inception module



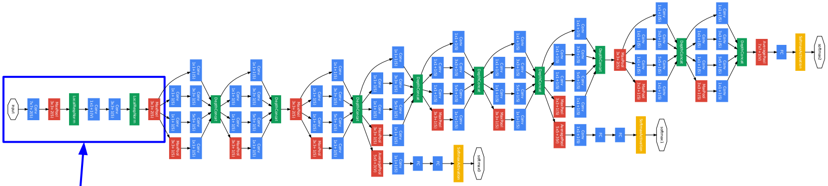
Source: CS231n course, Stanford University

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Stem Network: Conv-Pool- 2x Conv-Pool

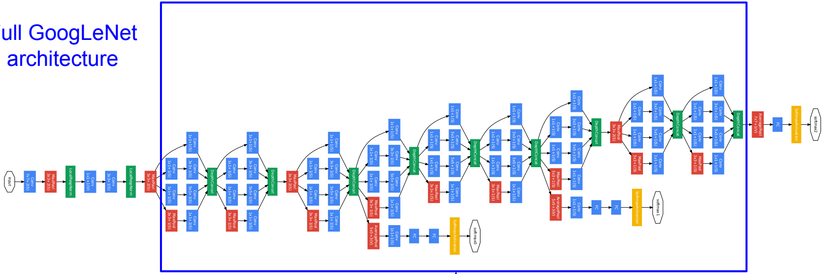
Source: CS231n course, Stanford University

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Stacked Inception Modules

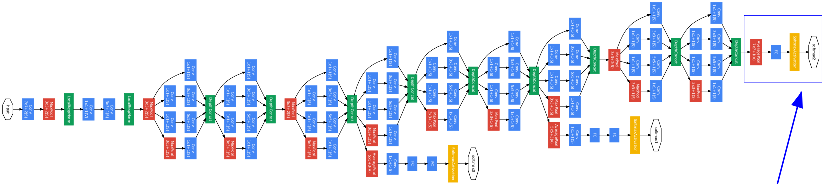
Source: CS231n course, Stanford University

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Classifier output
(removed expensive FC layers!)

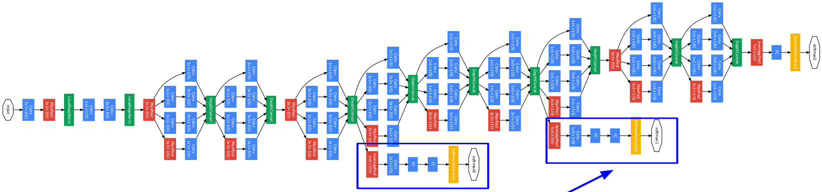
Source: CS231n course, Stanford University

GoogLeNet

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



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

Source: CS231n course, Stanford University

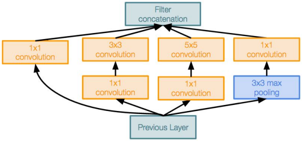
GoogLeNet

Case Study: GoogLeNet

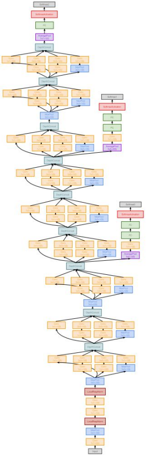
[Szegedy et al., 2014]

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)



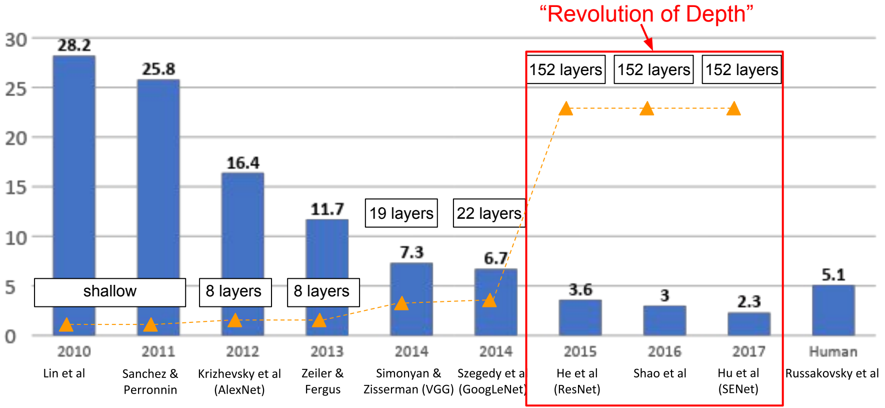
Inception module



Source: CS231n course, Stanford University

ResNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



§ Citation of the paper as on Feb 12, 2020 is 39,180

Source: CS231n course, Stanford University

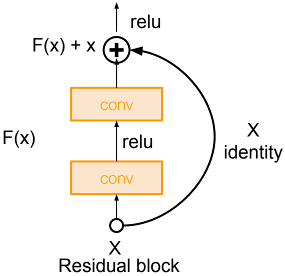
ResNet

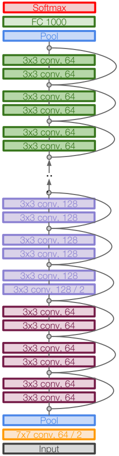
Case Study: ResNet

[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!





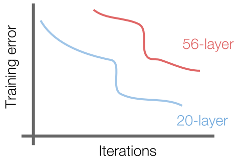
Source: CS231n course, Stanford University

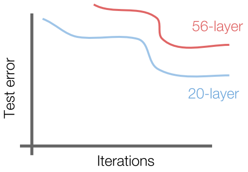
ResNet

Case Study: 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]

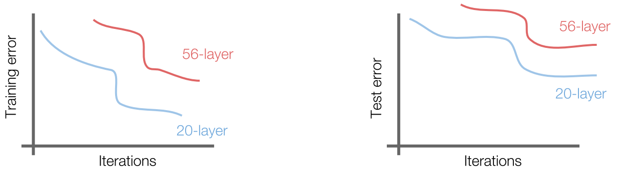
Source: CS231n course, Stanford University

ResNet

Case Study: 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!

Source: CS231n course, Stanford University

ResNet

Case Study: 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.

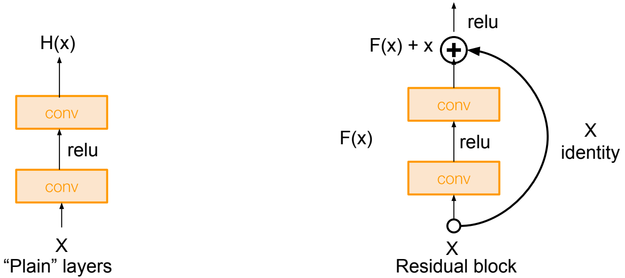
Source: CS231n course, Stanford University

ResNet

Case Study: ResNet

[He et al., 2015]

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



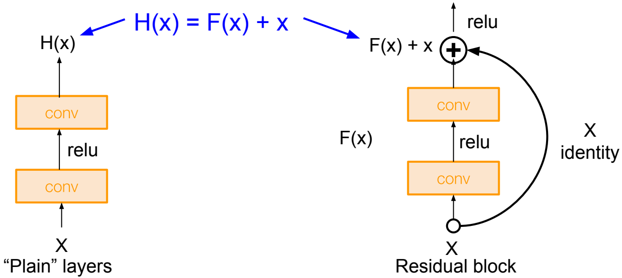
Source: CS231n course, Stanford University

ResNet

Case Study: 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

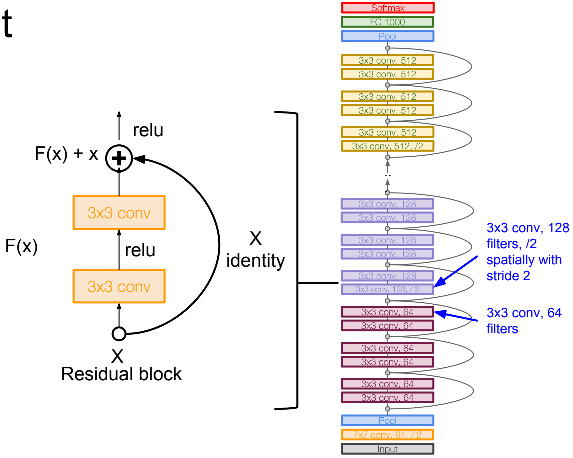
Source: CS231n course, Stanford University

ResNet

Case Study: 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)



Source: CS231n course, Stanford University

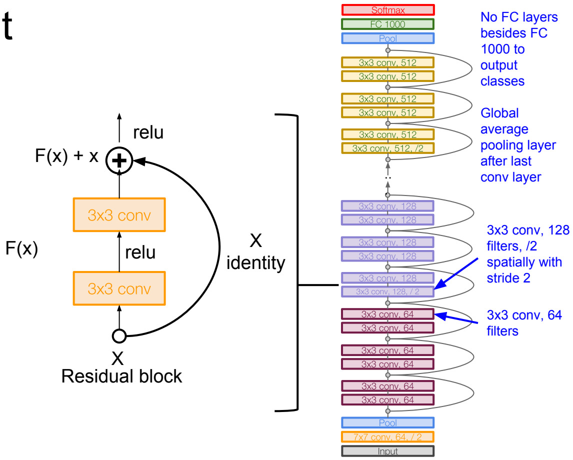
ResNet

Case Study: 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)



Source: CS231n course, Stanford University

ResNet

Case Study: ResNet

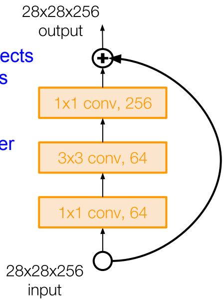
[He et al., 2015]

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

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64



Source: CS231n course, Stanford University

ResNet

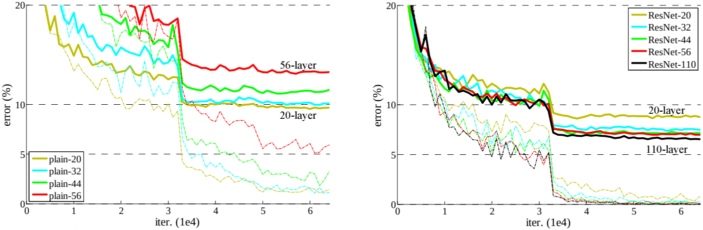


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error

Source: He et. al., 2015

ResNet

Case Study: 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

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

Source: CS231n course, Stanford University

MobileNet-v1

- ConvNets, in general, are compute and memory heavy
- MobileNet-v1 from Google [in 2017] describes an efficient network in terms of compute and memory so that many real world vision applications can be performed in mobile or similar embedded platforms

MobileNet-v1

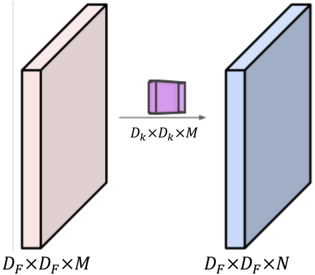
- ConvNets, in general, are compute and memory heavy
- MobileNet-v1 from Google [in 2017] describes an efficient network in terms of compute and memory so that many real world vision applications can be performed in mobile or similar embedded platforms
- Used depthwise separable convolution which is depthwise convolution and then pointwise convolution

MobileNet-v1

- ConvNets, in general, are compute and memory heavy
- MobileNet-v1 from Google [in 2017] describes an efficient network in terms of compute and memory so that many real world vision applications can be performed in mobile or similar embedded platforms
- Used depthwise separable convolution which is depthwise convolution and then pointwise convolution
- Also introduced two simple scaling hyperparameters

Depthwise Separable Convolution

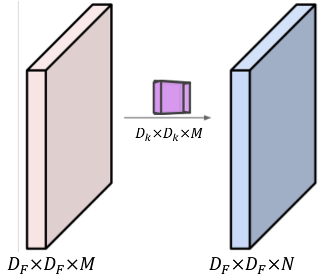
- Suppose, we have $D_F \times D_F \times M$ input feature map, $D_F \times D_F \times N$ output feature map and $D_k \times D_k$ spatial sized conventional convolution filters.



- What is the computational cost for such a convolution operation?

Depthwise Separable Convolution

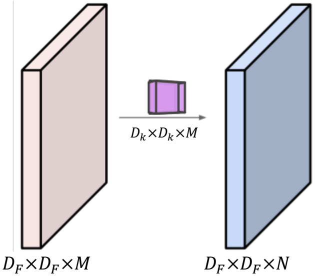
- Suppose, we have $D_F \times D_F \times M$ input feature map, $D_F \times D_F \times N$ output feature map and $D_k \times D_k$ spatial sized conventional convolution filters.



- What is the computational cost for such a convolution operation? —$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \cdot N$
- What is the number of parameters?

Depthwise Separable Convolution

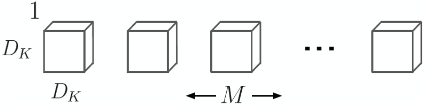
- Suppose, we have $D_F \times D_F \times M$ input feature map, $D_F \times D_F \times N$ output feature map and $D_k \times D_k$ spatial sized conventional convolution filters.



- What is the computational cost for such a convolution operation? —$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \cdot N$
- What is the number of parameters? —$D_k \cdot D_k \cdot M \cdot N$

Depthwise Separable Convolution

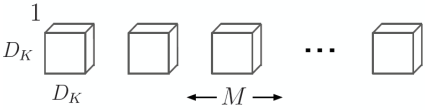
- Now, think of M filters which are $D_K \times D_K$ (not $D_K \times D_K \times M$) and think each M of these filters are operated separately on M channels of input of spatial size $D_F \times D_F$



- What is the computational cost for such a convolution operation?

Depthwise Separable Convolution

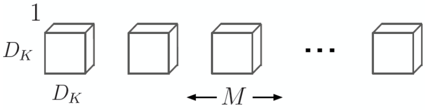
- Now, think of M filters which are $D_K \times D_K$ (not $D_K \times D_K \times M$) and think each M of these filters are operated separately on M channels of input of spatial size $D_F \times D_F$



- What is the computational cost for such a convolution operation? —$D_K \cdot D_K \cdot D_F \cdot D_F \cdot M$
- And what is the number of parameters?

Depthwise Separable Convolution

§ Now, think of M filters which are $D_K \times D_K$ (not $D_K \times D_K \times M$) and think each M of these filters are operated separately on M channels of input of spatial size $D_F \times D_F$

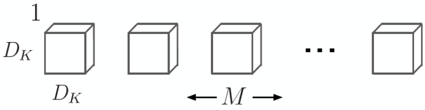


§ What is the computational cost for such a convolution operation? —$D_K \cdot D_K \cdot D_F \cdot D_F \cdot M$

§ And what is the number of parameters? —$D_K \cdot D_K \cdot M$

Depthwise Separable Convolution

- Now, think of M filters which are $D_K \times D_K$ (not $D_K \times D_K \times M$) and think each M of these filters are operated separately on M channels of input of spatial size $D_F \times D_F$



- What is the computational cost for such a convolution operation? —$D_K \cdot D_K \cdot D_F \cdot D_F \cdot M$
- And what is the number of parameters? —$D_K \cdot D_K \cdot M$
- This operation is known as Depthwise Convolution operation

Depthwise Separable Convolution

- What is the output shape now?

Depthwise Separable Convolution

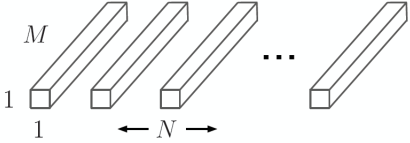
- What is the output shape now? —$D_F \times D_F \times M$
- Where did the N (output channels) go?

Depthwise Separable Convolution

- What is the output shape now? —$D_F \times D_F \times M$
- Where did the N (output channels) go?
  It is simply not there because depthwise convolution does the convolution only on input channels.

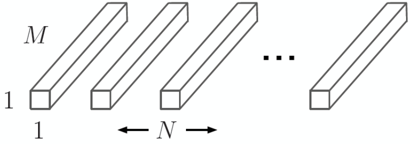
Depthwise Separable Convolution

- What is the output shape now? —$D_F \times D_F \times M$
- Where did the N (output channels) go?
  It is simply not there because depthwise convolution does the convolution only on input channels.
- Now think about 1×1 traditional convolution on $D_F \times D_F \times M$ featuremap to get $D_F \times D_F \times N$ output. What is the computation cost?



Depthwise Separable Convolution

- What is the output shape now? —$D_F \times D_F \times M$
- Where did the N (output channels) go?
  It is simply not there because depthwise convolution does the convolution only on input channels.
- Now think about 1×1 traditional convolution on $D_F \times D_F \times M$ featuremap to get $D_F \times D_F \times N$ output. What is the computation cost?

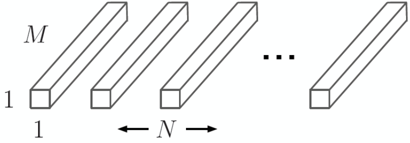


—$1 \cdot 1 \cdot M \cdot D_F \cdot D_F \cdot N = D_F \cdot D_F \cdot M \cdot N$

- What is the number of parameters?

Depthwise Separable Convolution

- What is the output shape now? —$D_F \times D_F \times M$
- Where did the N (output channels) go?
  It is simply not there because depthwise convolution does the convolution only on input channels.
- Now think about 1×1 traditional convolution on $D_F \times D_F \times M$ featuremap to get $D_F \times D_F \times N$ output. What is the computation cost?



—$1 \cdot 1 \cdot M \cdot D_F \cdot D_F \cdot N = D_F \cdot D_F \cdot M \cdot N$

- What is the number of parameters? —$1 \cdot 1 \cdot M \cdot N$
- This operation is called 1×1 pointwise convolution

Depthwise Separable Convolution

- So, traditional convolution with $D_K \times D_K \times M \times N$ filters, we get feature map of size $D_F \times D_F \times N$
- Also with depthwise separable convolution (*i.e.*, depthwise convolution $+ 1 \times 1$ pointwise convolution), we get $D_F \times D_F \times N$ feature map
- The computation is less
  - $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \cdot N$ vs
  - $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + D_F \cdot D_F \cdot M \cdot N$
- The reduction in computation $\frac{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + D_F \cdot D_F \cdot M \cdot N}{D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \cdot N} = \frac{1}{N} + \frac{1}{D_K^2}$
- Also the reduction in number of parameters $\frac{M \cdot D_K \cdot D_K + M \cdot N}{D_K \cdot D_K \cdot M \cdot N} = \frac{1}{N} + \frac{1}{D_K^2}$

MobileNet-v1 Structure

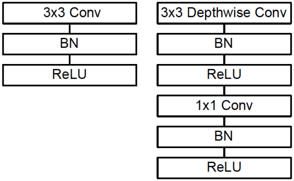


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Image taken from: *MobileNet Paper*

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Image taken from: *MobileNet Paper*

Width and Resolution Multiplier

- The role of the width multiplier $\alpha \in (0, 1]$ is to thin a network uniformly at each layer
- the number of input channels M becomes αM and the number of output channels N becomes αN
- The computational cost of a depthwise separable convolution with width multiplier α is $D_k \cdot D_k \cdot \alpha M \cdot D_F \cdot D_F + D_F \cdot D_F \cdot \alpha M \cdot \alpha N$
- Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α^2

Width and Resolution Multiplier

- The role of the width multiplier $\alpha \in (0, 1]$ is to thin a network uniformly at each layer
- the number of input channels M becomes αM and the number of output channels N becomes αN
- The computational cost of a depthwise separable convolution with width multiplier α is $D_k \cdot D_k \cdot \alpha M \cdot D_F \cdot D_F + D_F \cdot D_F \cdot \alpha M \cdot \alpha N$
- Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α^2
- Resolution multiplier $\rho \in (0, 1]$ reduces the image resolution by this factor and the internal representation of every layer is subsequently reduced by the same multiplier
- With width multiplier α and resolution multiplier ρ, the computational cost is α is $D_k \cdot D_k \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \rho D_F \cdot \rho D_F \cdot \alpha M \cdot \alpha N$
- Resolution multiplier has the effect of reducing computational cost by ρ^2

Experimental Results

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Image taken from: *MobileNet Paper*

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Image taken from: *MobileNet Paper*