CNN Popular Architectures and Transfer Learning

Palacode Narayana Iyer Anantharaman

16th Oct 2018

Motivation : Why study these?

Understanding popular architectures help us in many ways:

- We develop a better understanding of the subject by studying high performant architectures
- This helps perform our own research on newer models
- Learning their design philosophy help us to design our models more effectively.
- Use them as a backbone for transfer learning, selecting the right architecture

References

Deep Residual Learning for Image Recognition

Kaiming He   Xiangyu Zhang   Shaoqing Ren   Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Squeeze-and-Excitation Networks

Jie Hu[1*] hujie@momenta.ai
Li Shen[2*] lishen@robots.ox.ac.uk
Gang Sun[1] sungang@momenta.ai
[1] Momenta [2] Department of Engineering Science, University of Oxford

Going Deeper with Convolutions

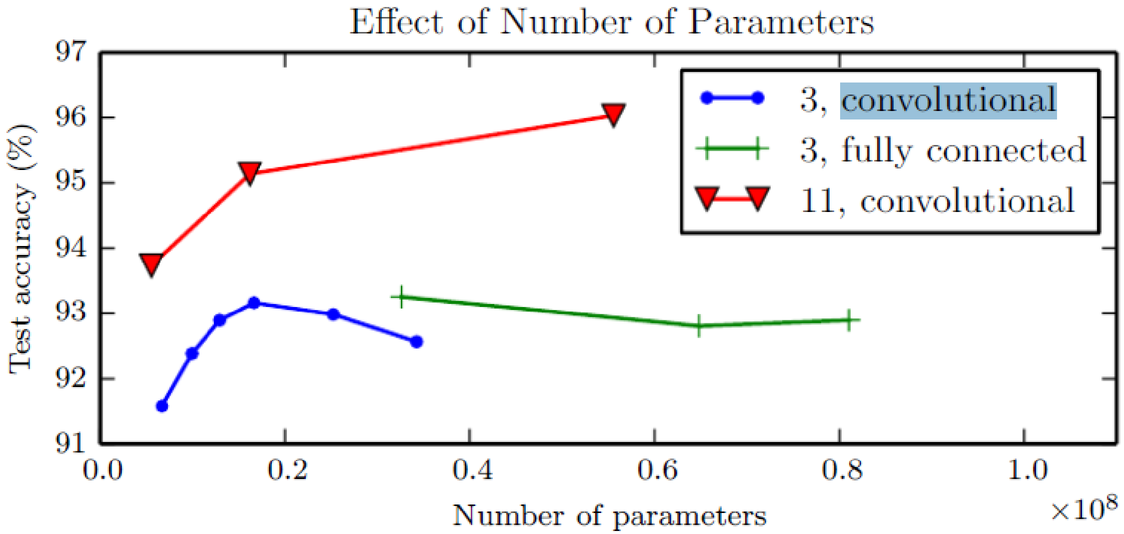
Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy, jiayq, sermanet, dragomir, dumitru, vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

[PDF] May 1, 2018 Lecture 9 - CS231n
cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf
May 1, 2018 - Fei-Fei Li & Justin Johnson & Serena Yeung. Lecture 9 -. May 1, 2018. 20. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) ...

https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/

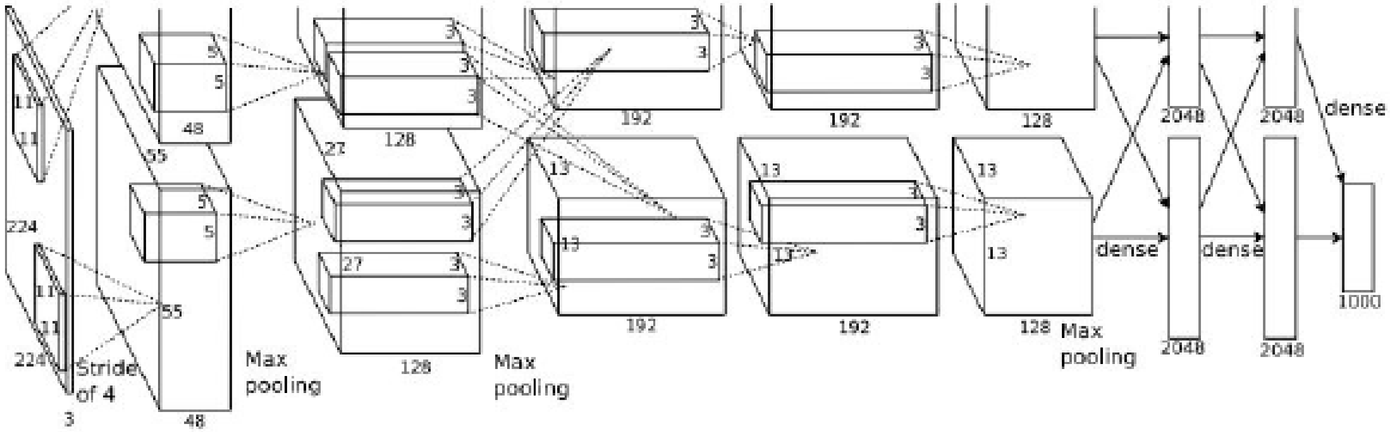
Current trend: Deeper Models work better

- CNNs consistently outperform other approaches for the core tasks of CV
- Deeper models work better
- Increasing the number of parameters in layers of CNN without increasing their depth is not effective at increasing test set performance.
- Shallow models overfit at around 20 million parameters while deep ones can benefit from having over 60 million.
- Key insight: Model performs better when it is architected to reflect composition of simpler functions than a single complex function. This may also be explained off viewing the computation as a chain of dependencies



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%

VGG Net

(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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image		
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64
maxpool		
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

VGG net

(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

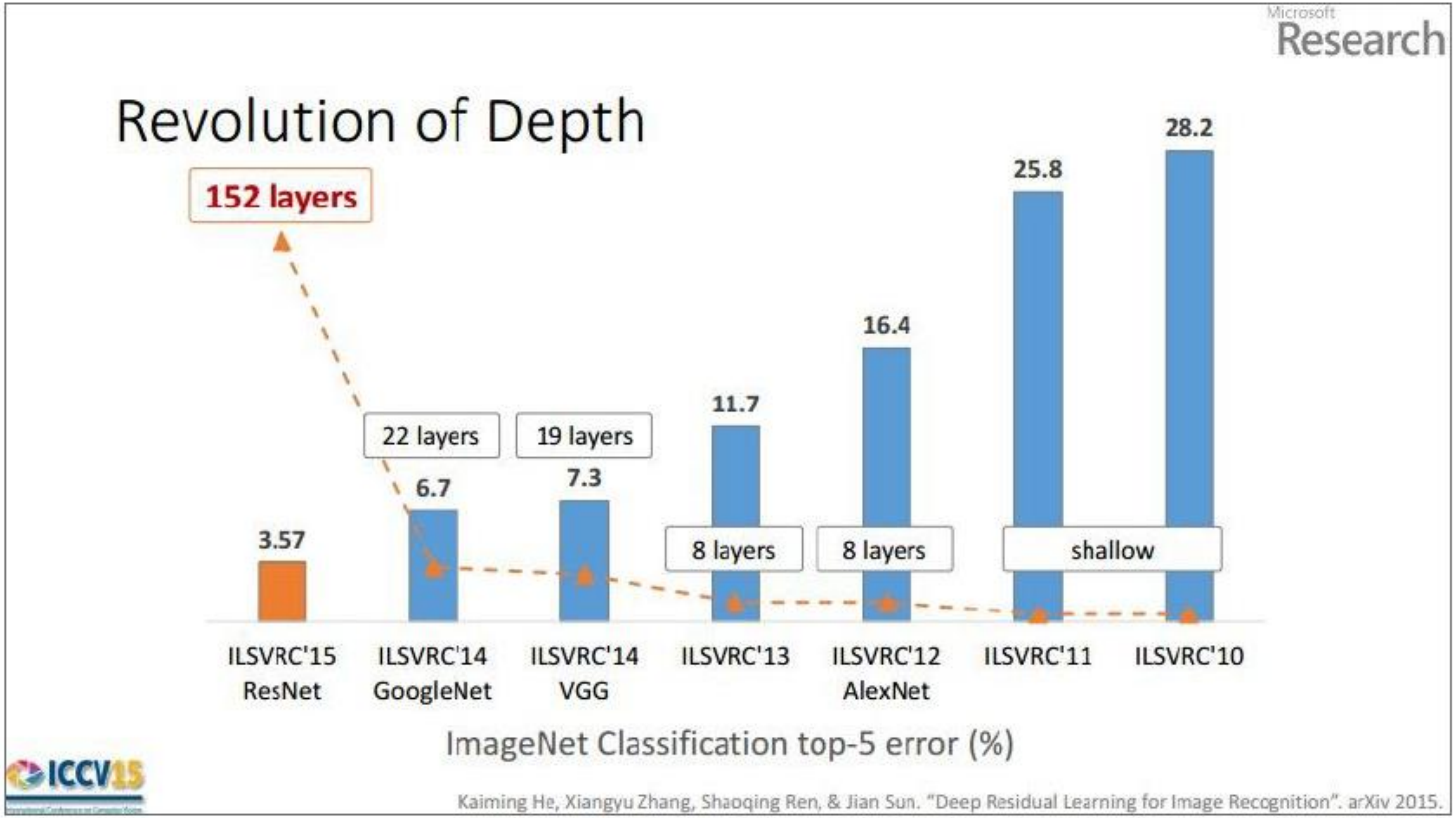
Note:

Most memory is in early CONV

Most params are in late FC

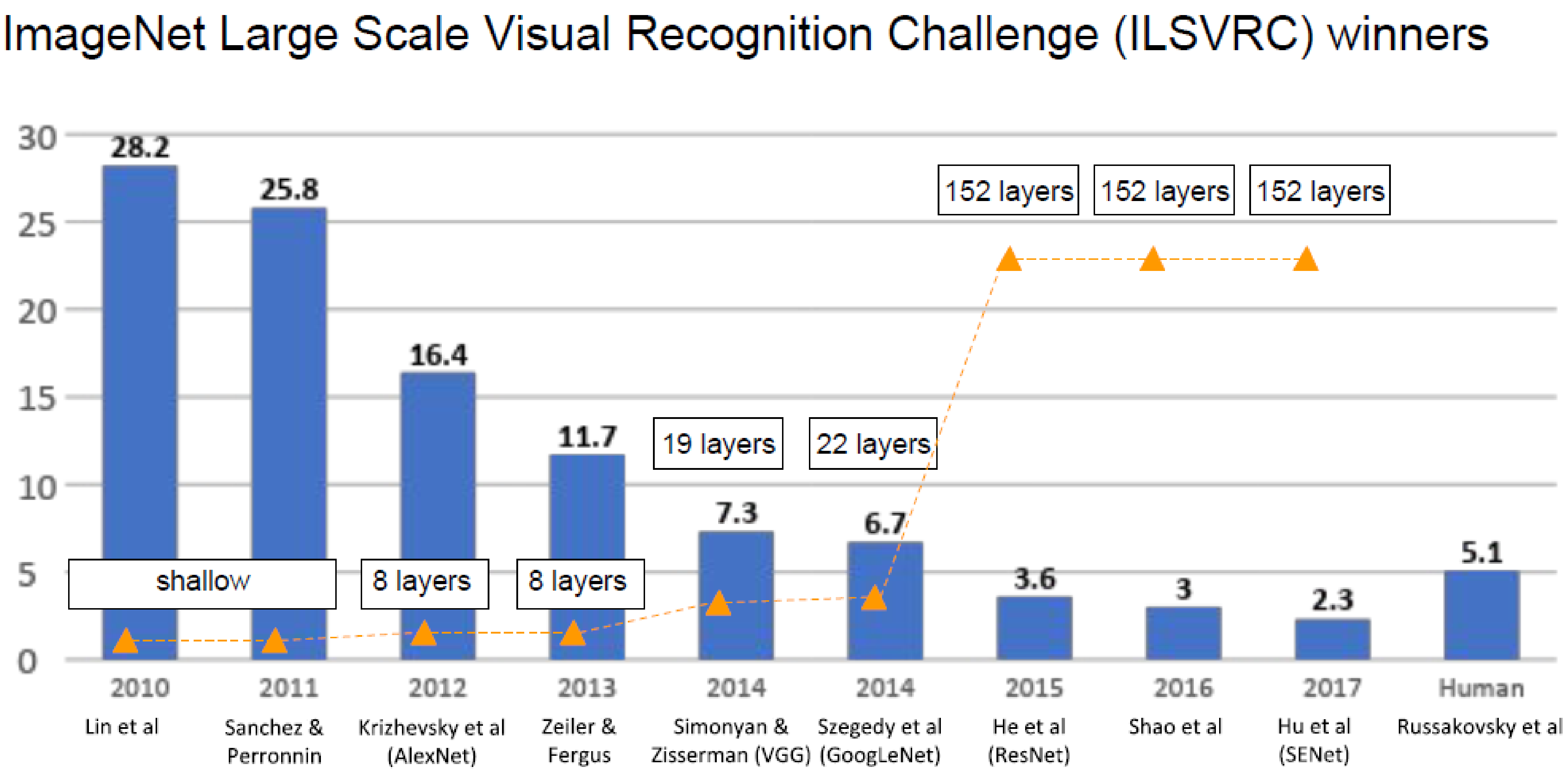
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ResNet



(slide from Kaiming He's recent presentation)

Recent Results (Credits: CS231n Stanford)



Resnet Motivation

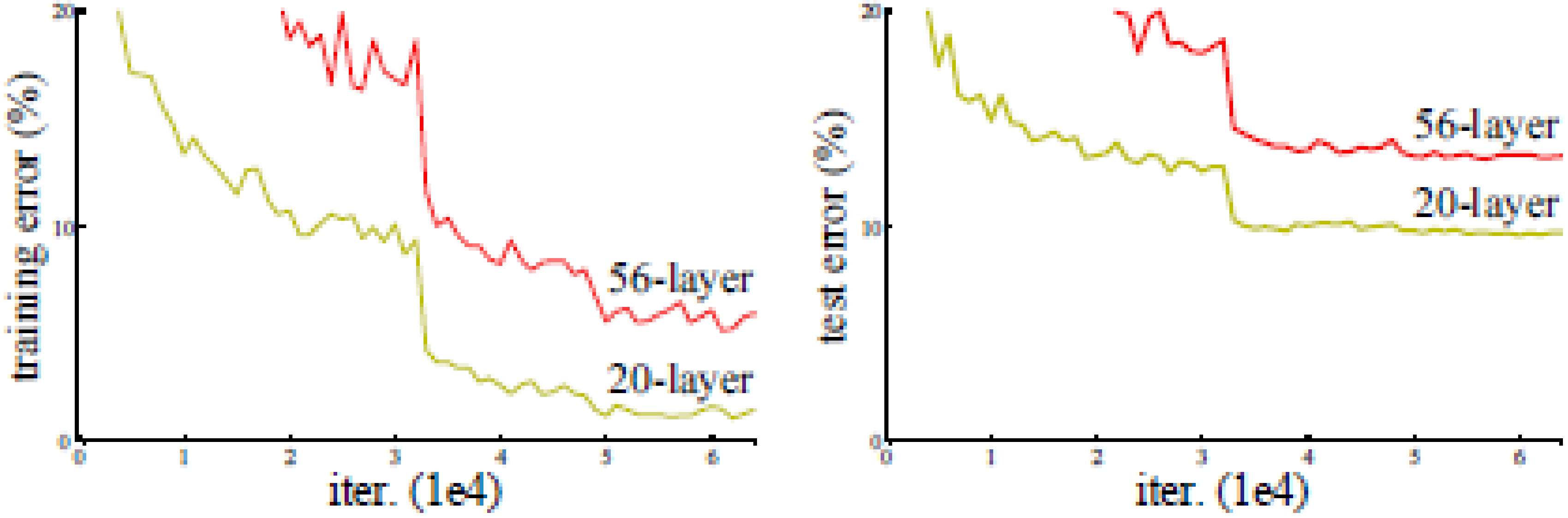
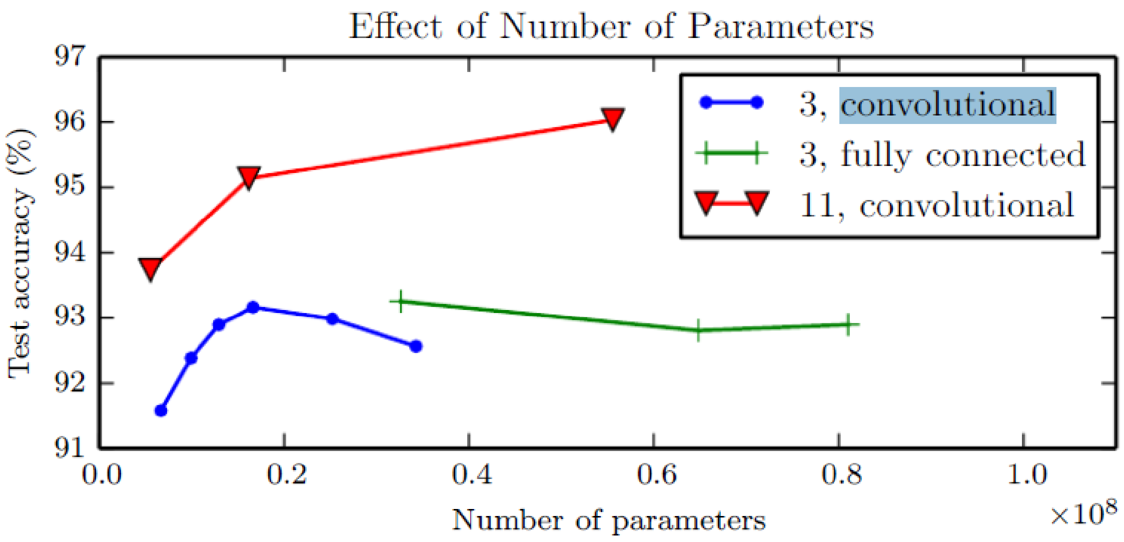
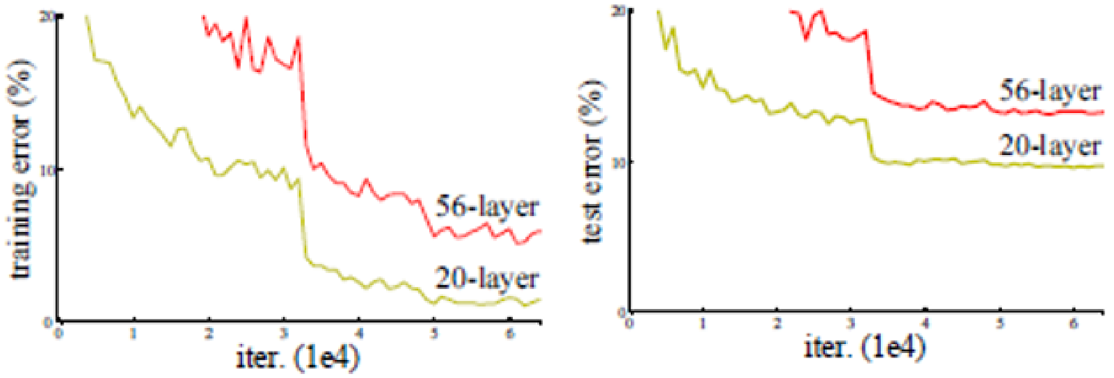


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet Approach

- Generally: Deeper the better - See the fig
- Issue: Hard to train deeper networks effectively : Validation error goes up not just due to overfitting but increase in training error
- Solution: Use skip connections to propagate the activations to reduce the impact
- Rationale: Imagine how to get an identity output through a deep network accurately.





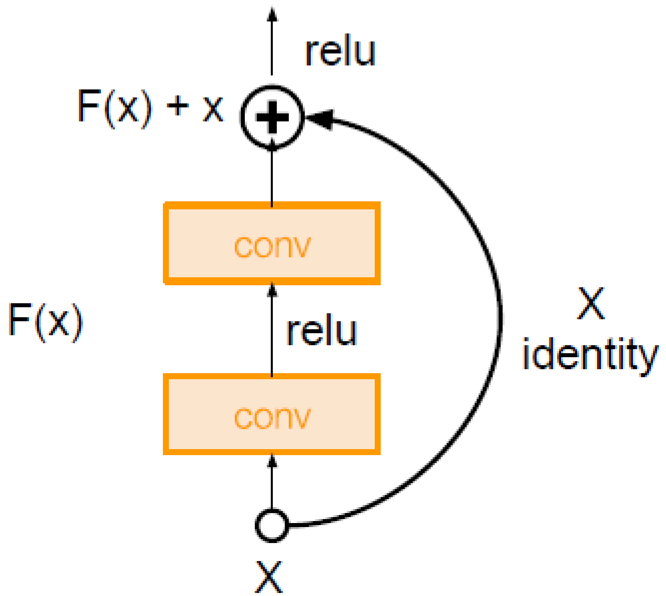
ResNet Hypothesis : Rationale

- Deeper models should perform at least as well as shallower models
  - More parameters, more degrees of freedom to get the training error down
  - More depth, more ability to model abstractions

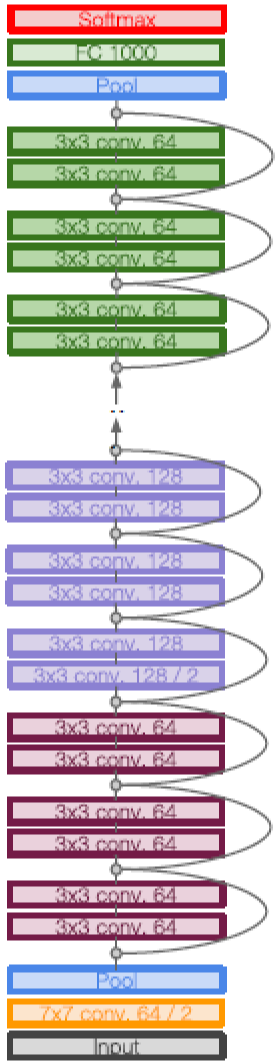
- Solution by construction is copying the learned layers of a shallower model and setting the additional layers to identity mapping

ResNet

- Very deep network: Uses 152 layers
- Shallower versions (e.g. Resnet50) are available
- The “go to” backbone network for many applications such as Faster RCNN
- Pre trained weights are available for Keras, TensorFlow



Residual block



ResNet Details

- Default input size: (224, 224, 3)
- Each stage reduces the width, height dimensions by a factor of 2
- This property is leveraged in later implementations such as pyramidal networks

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ImageNet Results summary table

2015 ResNet (ILSVRC'15) 3.57

Year	Codename	Error (percent)	99.9% Conf Int
2014	**GoogLeNet**	**6.66**	**6.40 - 6.92**
2014	VGG	7.32	7.05 - 7.60
2014	MSRA	8.06	7.78 - 8.34
2014	AHoward	8.11	7.83 - 8.39
2014	DeeperVision	9.51	9.21 - 9.82
2013	Clarifai†	11.20	10.87 - 11.53
2014	CASIAWS†	11.36	11.03 - 11.69
2014	Trimps†	11.46	11.13 - 11.80
2014	Adobe†	11.58	11.25 - 11.91
2013	**Clarifai**	**11.74**	**11.41 - 12.08**
2013	NUS	12.95	12.60 - 13.30
2013	ZF	13.51	13.14 - 13.87
2013	AHoward	13.55	13.20 - 13.91
2013	OverFeat	14.18	13.83 - 14.54
2014	Orange†	14.80	14.43 - 15.17
2012	SuperVision†	15.32	14.94 - 15.69
2012	**SuperVision**	**16.42**	**16.04 - 16.80**
2012	ISI	26.17	25.71 - 26.65
2012	VGG	26.98	26.53 - 27.43
2012	XRCE	27.06	26.60 - 27.52
2012	UvA	29.58	29.09 - 30.04

human error is around *5.1%* on a subset

Microsoft ResNet, a 152 layers network

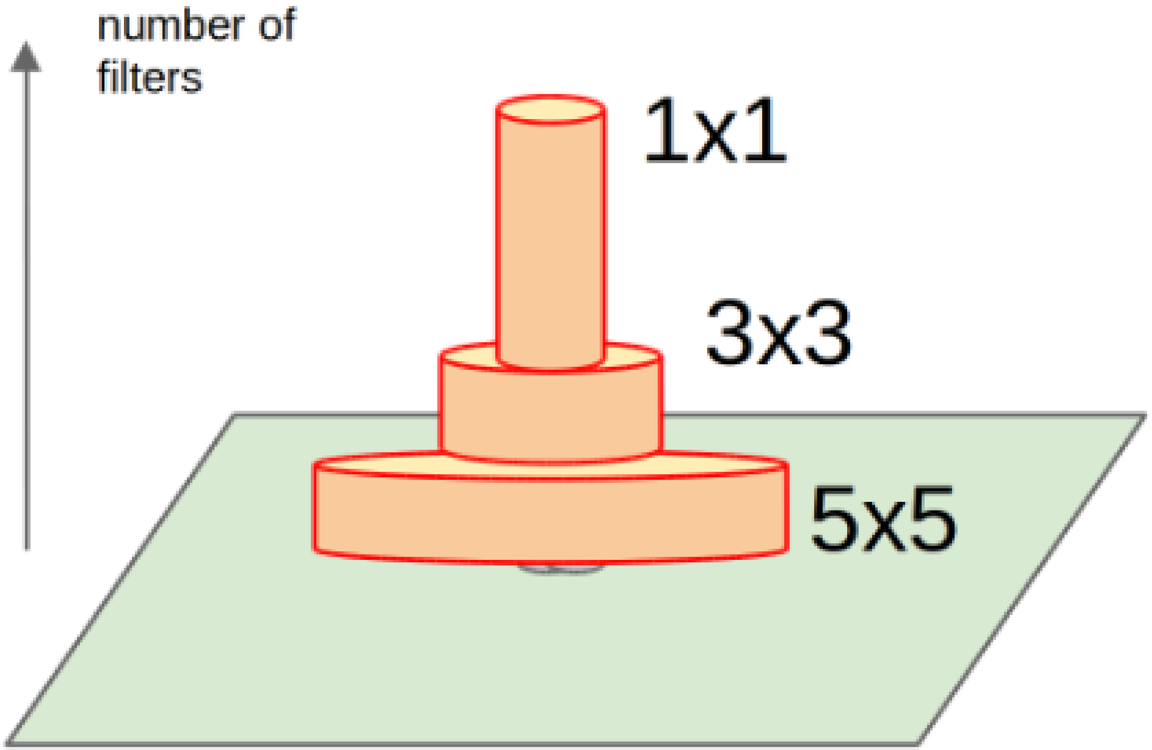
GoogLeNet, 22 layers network

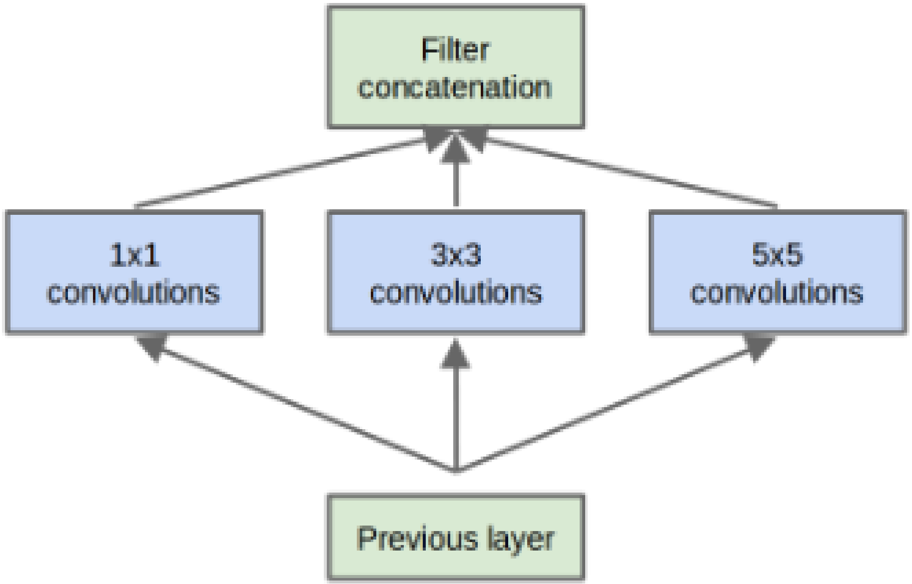
U. of Toronto, SuperVision, a 7 layers network

GoogleNet

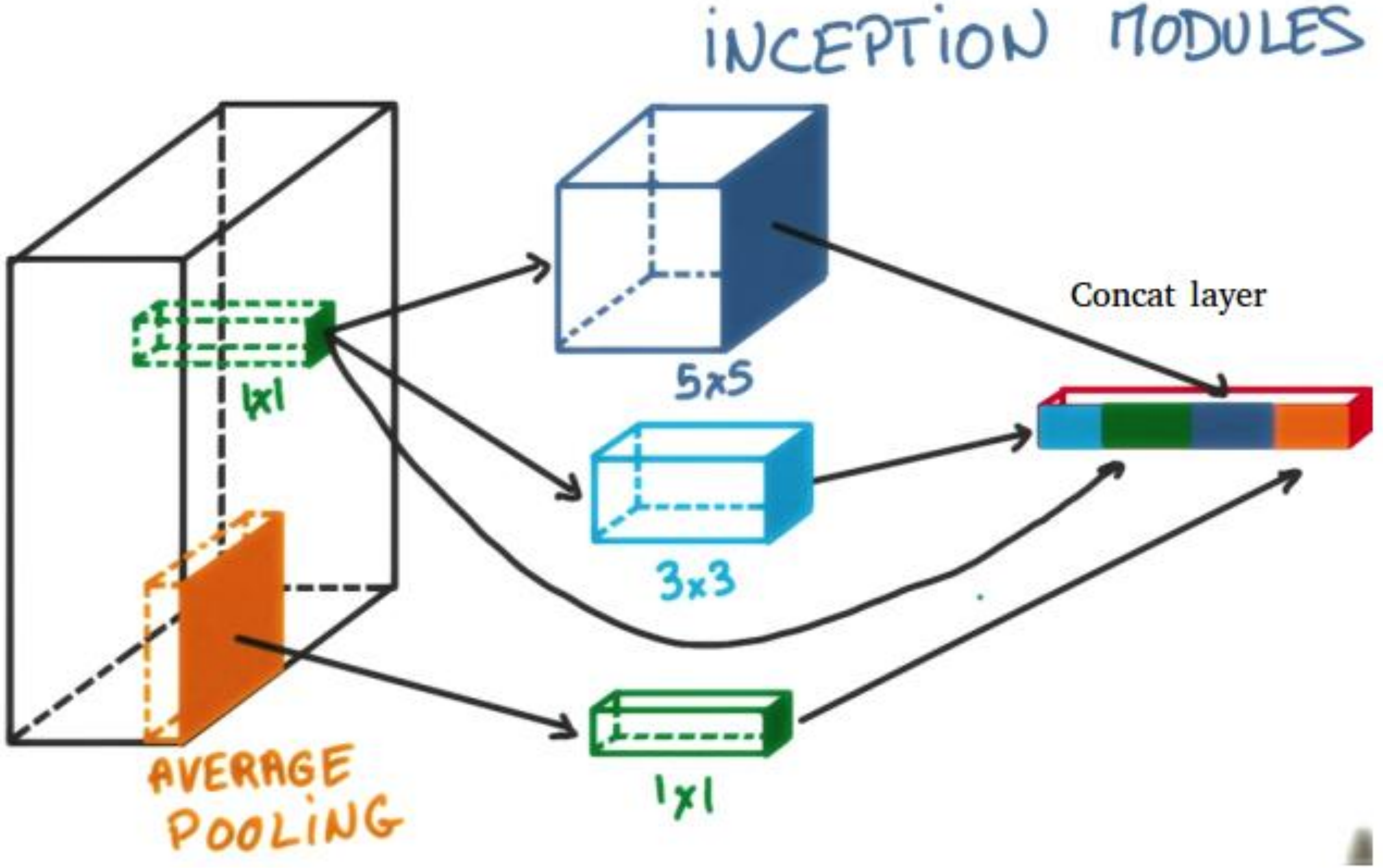
- Design choices of filter sizes: (3, 3), (5, 5) and so on – Which to choose for each convolution layer?
- Why not try all of them and choose the best?
- Trying each possible value on every layer and experimenting manually is not a solution
- Inception layer allows multiple filter sizes and learn their contributions through parameters automatically

Inception Layer Naïve Architecture

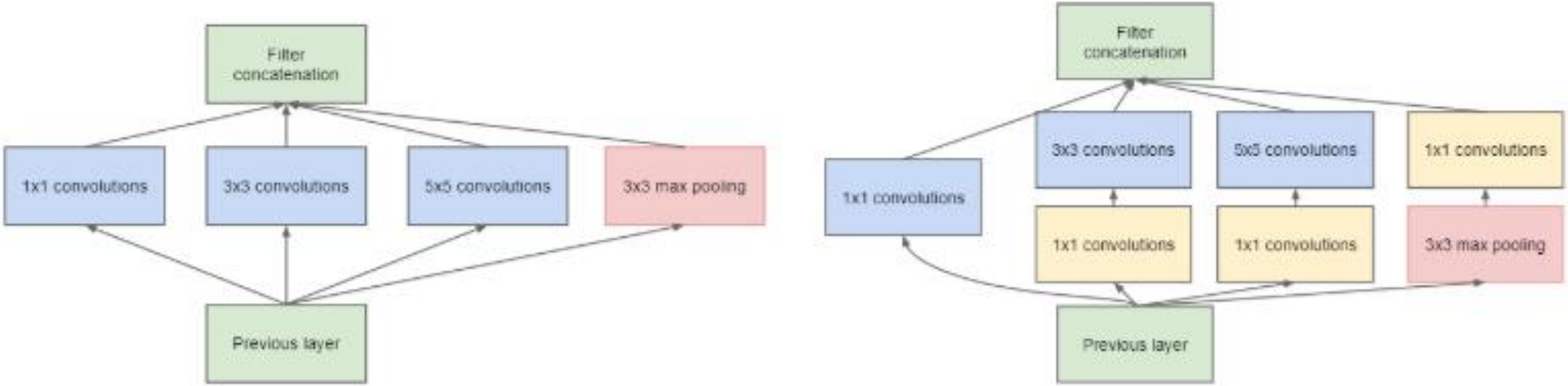




Inception Layer Naïve Architecture (Fig Udacity Deep Learning)



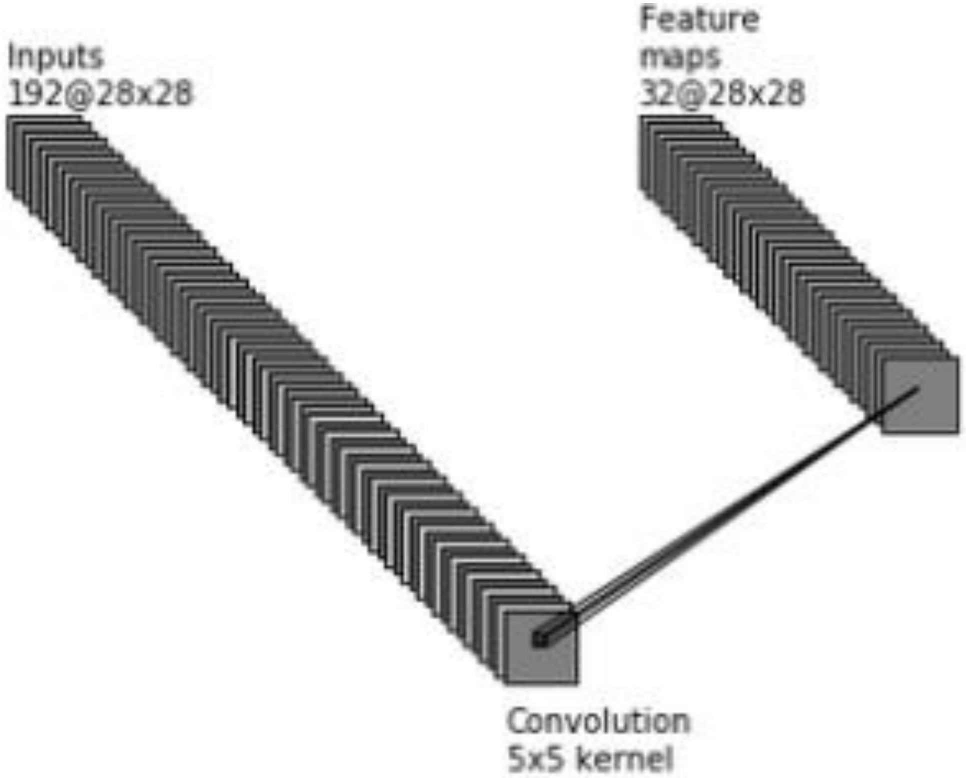
Inception Architecture with bottleneck layer

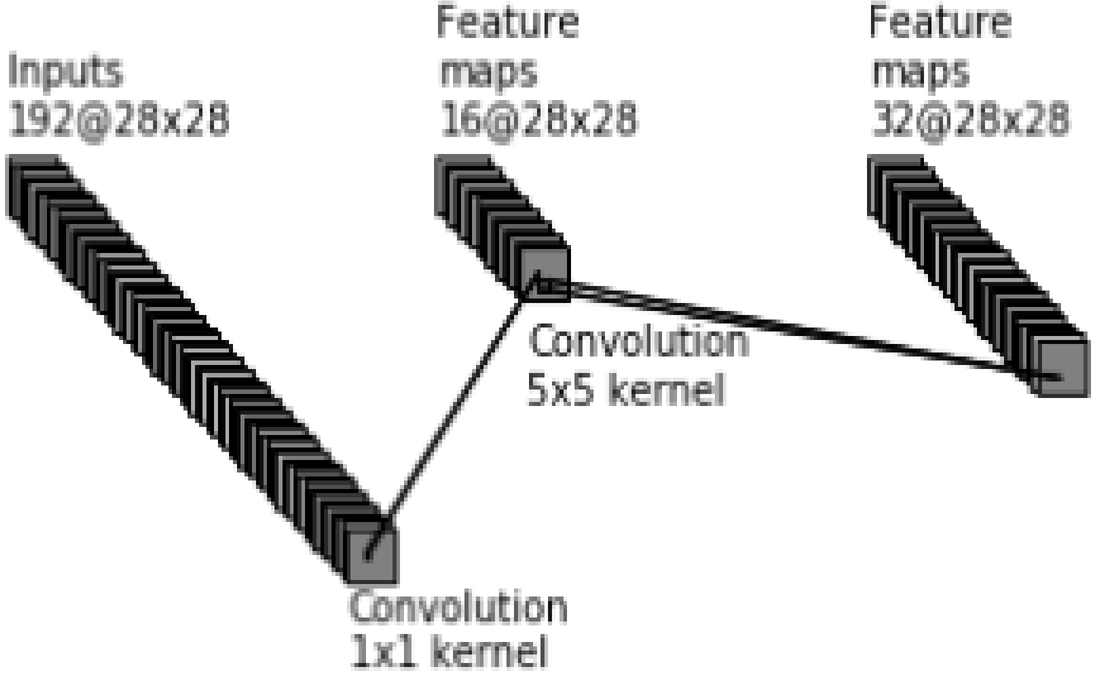


(a) Inception module, naïve version

(b) Inception module with dimension reductions

Bottleneck Layer





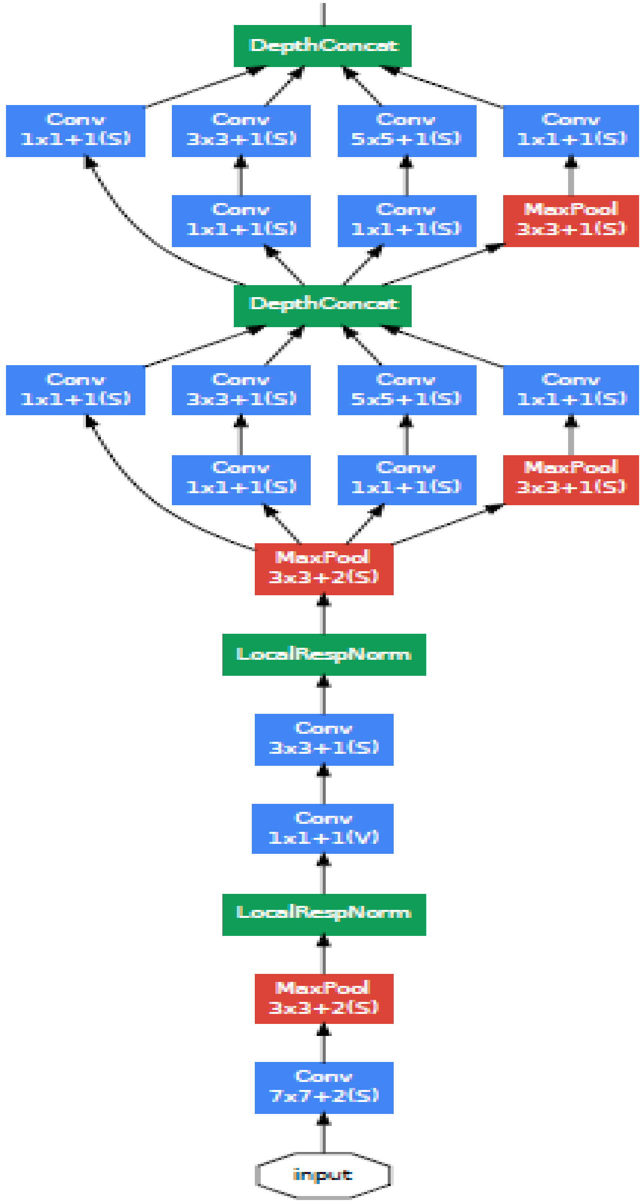
Example

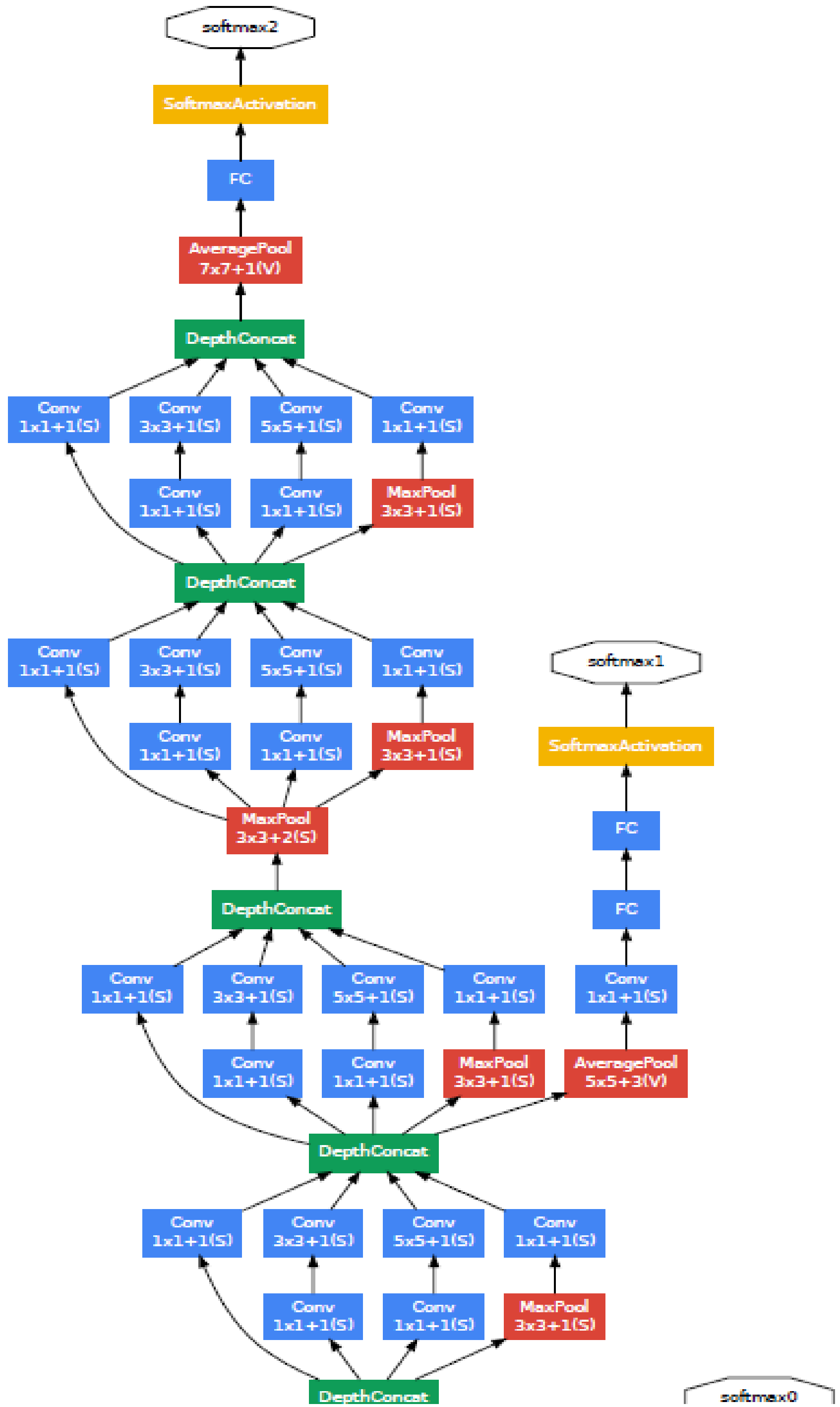
- Consider an input volume (28, 28, 192) and an output volume (28, 28, 32)
- How many computations are needed if we use a 5 x 5 filter?
- Each filter will be 5 x 5 x 192, we will be moving this over a 28 x 28 surface and we have 32 of them
  - 28 x 28 x 32 x 5 x 5 x 192 = 120M
- If we need multiple such filters, we need to add up corresponding computations for each of them
- On a very deep network these many computations are prohibitively large even when we use powerful hardware
- By reducing the dimensionality of the input before final convolutions, we get a manageable number of computations

Example with bottleneck layer

- In our example, we can transform the 28 x 28 x 192 in to same sized surface but much reduced depth (say 16) using 1 x 1 convolutions
- The bottleneck layer has the shape (28 x 28 x 16)
- Perform the required convolutions (e.g. 5 x 5) on the bottleneck layer to generate the final output volume
- Computations: #computations between input to bottleneck layer + #computations between bottleneck to output. In our example this is 12M

GoogleNet architecture with Inception layer





State of the art : SENet

- ImageNet 2017 topper in multiple categories

- A novel technique to weight the contributions of the channels of a convolutional layer

SeNet : Squeeze and Excitation Network

- SeNet is the winning architecture of ImageNet 2017 in multiple categories

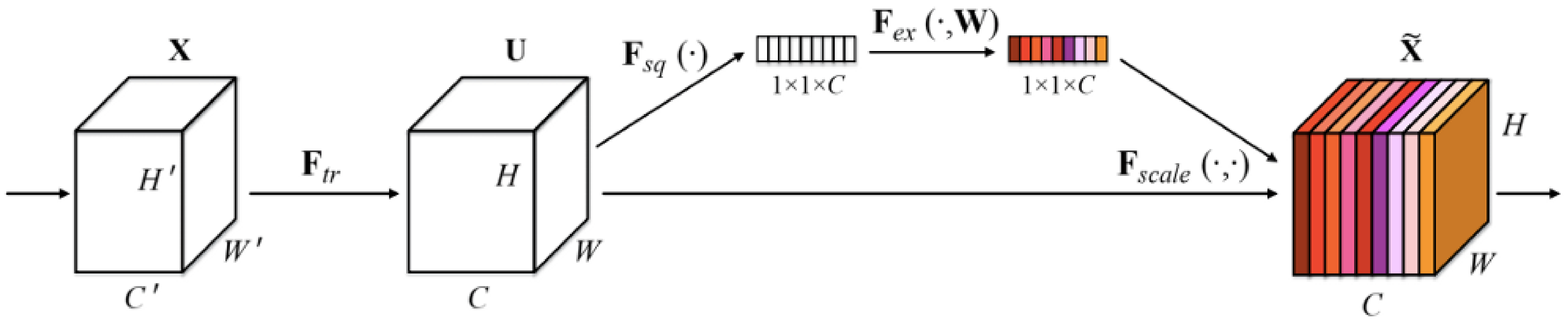
- Error rate on image classification: 2.251%

- Key Idea:
  - In authors’ words: “Improve the representational power of the network by explicitly modelling interdependencies between channels of its convolutional features”
  - Simple explanation: Add parameters to each channel of a convolutional block so that network can adaptively adjust the weighting of each feature map

SeNet Rationale

- Deep CNN’s learn increasing levels of abstractions from lower to higher layers. Lower layers have higher resolution and can extract basic elements of information
- Higher layers can detect faces or generate text etc and deal with abstract information
- All of this works by fusing the spatial and channel information of an image.
- The network weights each of its channels equally when creating the output feature maps.
- SENets change this by adding a content aware mechanism to weight each channel adaptively. In it’s most basic form this could mean adding a single parameter to each channel and giving it a linear scalar how relevant each one is.

SENet Architecture



- Get a global understanding of each channel by squeezing the feature maps to a single numeric value. This results in a vector of size *n*, where *n* is equal to the number of convolutional channels.
- Afterwards, it is fed through a two-layer neural network, which outputs a vector of the same size. These *n* values can now be used as weights on the original features maps, scaling each channel based on its importance.

Code Illustration of the key idea

```
def se_block(in_block, ch, ratio=16):
    x = GlobalAveragePooling2D()(in_block)
    x = Dense(ch//ratio, activation='relu')(x)
    x = Dense(ch, activation='sigmoid')(x)
    return multiply()([in_block, x])
```

Ref: https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7
Credits: Paul-Louis Prove

High level steps

1. The function is given an input convolutional block and the current number of channels it has
2. We squeeze each channel to a single numeric value using average pooling
3. A fully connected layer followed by a ReLU function adds the necessary nonlinearity. It's output channel complexity is also reduced by a certain ratio.
4. A second fully connected layer followed by a Sigmoid activation gives each channel a smooth gating function.
5. At last, we weight each feature map of the convolutional block based on the result of our side network.

Ref: https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7
Credits: Paul-Louis Prove

Adding squeeze excitation technique to ResNet