ARCHITECTURES FOR CONVOLUTIONAL NEURAL NETWORKS

INF5860 — Machine Learning for Image Analysis

Ole-Johan Skrede

14.03.2018

University of Oslo

OUTLINE

- Introduction and motivation
- LeNet
- AlexNet
- VGG
- Inception / GoogLeNet
- ResNet
- Recent examples

INTRODUCTION AND MOTIVATION

WHY IS THIS IMPORTANT

- Important to know the history for reference
- The chosen architectures are amongst the most cited works in machine learning
- Many works refer to these architectures
- They have served, and still serve as a basis for other classification network, and also segmentation, localization e.t.c.
- Interesting to see how others has been creative in this field
- We can learn from previous mistakes, and successes

LEARNING OUTCOME

- Recognise different network architectures
- For each architecture:
    - How does it work?
    - What ideas are introduced?
    - Why is it successful?
    - What can we learn from it?
- How to apply the ideas

LENET

Paper	Gradient Based Learning Applied to Document Recognition
Authors	Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner
Year	1998
Citations	11 135
Link to	pdf

INITIAL NOTES

- Very influential, and successful in its time
- First "modern" cnn
- We start to see tendencies of the familiar cnn composition, but it is not the first cnn
- The paper discusses a lot of central aspects
- Also uses a lot of deprecated techniques:
    - Originally uses a *stochastic diagonal Levenberg-Marquardt* optimization routine
    - Originally uses distance from an "ideal" set of ASCII characters as loss
    - The "idea" of the method holds with SGD and softmax
    - Originally a complicated scheme of which filters to apply on which feature maps
    - Also uses non-linearity after pooling

ACTIVATION FUNCTIONS

- Convolution nodes uses a scaled $\tanh$ non-linearity

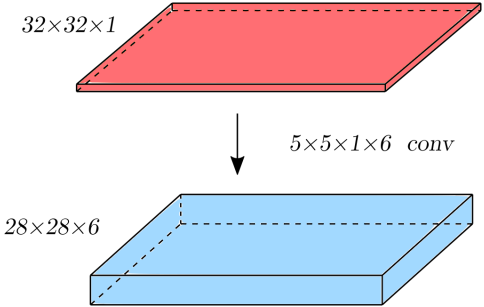
$$g(z) = A \tanh(Sz) \tag{1}$$

- Sets $A = 1.7259$, and $S = 2/3$
- This makes $g(-1) = -1$ and $g(1) = 1$, which is chosen for convenience

LENET — INPUT

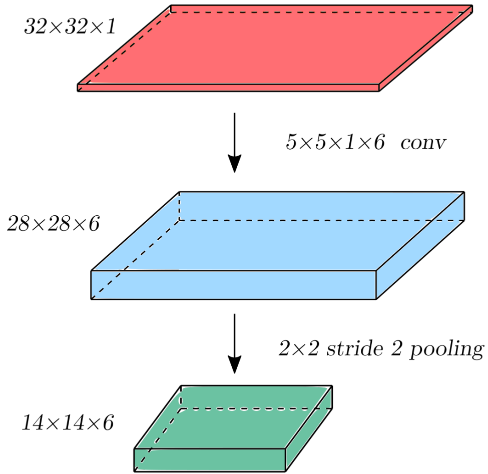
- $32 \times 32 \times 1$
- Used for character recognition
- Normalized to zero mean and unit variance

- Input shape: $32 \times 32 \times 1$
- 6 convolutions with kernel shape $5 \times 5 \times 1$, no padding
- $5 \cdot 5 \cdot 6 + 1 \cdot 6 = 156$ trainable parameters
- Output shape: $28 \times 28 \times 6$



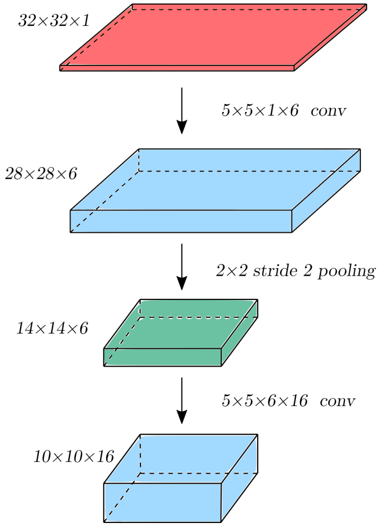
LENET — FIRST SUBSAMPLE (POOLING) LAYER

- Input shape: $28 \times 28 \times 6$
- Window shape: 2×2 with stride 2
- Output shape: $14 \times 14 \times 6$
- Activation for a unit:
  $a = g\left(\frac{x_1+x_2+x_3+x_4}{w} + b\right)$
- w and b is shared by all units in a feature map
- w and b are trainable, resulting in $6 \cdot (1+1) = 12$ parameters
- Very similar to an average pool layer



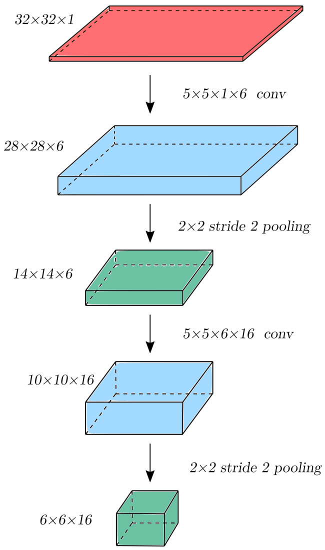
LENET — SECOND CONVOLUTIONAL LAYER

- Input shape: $14 \times 14 \times 6$
- 16 convolutions with shape $5 \times 5 \times 6$, no padding
- Output shape: $10 \times 10 \times 16$
- In total $25 \cdot (6 \cdot 3 + 6 \cdot 4 + 3 \cdot 4 + 6) + 16 = 1516$ trainable parameters

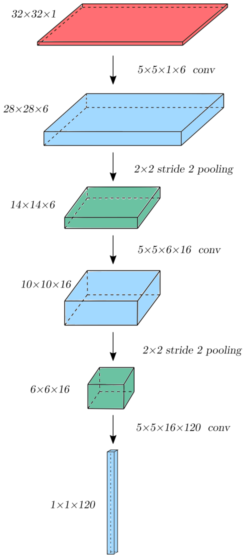


LENET — SECOND SUBSAMPLE (POOLING) LAYER

- Input shape: $10 \times 10 \times 16$
- Window shape: 2×2 with stride 2
- Output shape: $5 \times 5 \times 16$
- Activation for a unit:
  $a = g\left(\frac{x_1+x_2+x_3+x_4}{w} + b\right)$
- w and b is shared by all units in a feature map
- w and b are trainable, resulting in $16 \cdot (1+1) = 32$ parameters

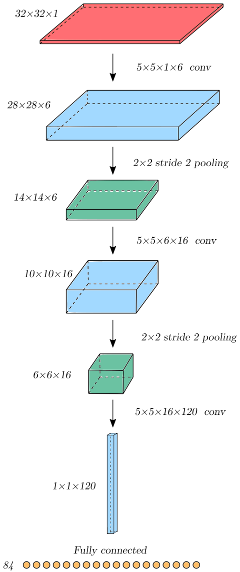


- Input shape: $5 \times 5 \times 16$
- 120 convolutions with shape $5 \times 5 \times 16$, no padding
- Output shape: $1 \times 1 \times 120$
- In total $5 \cdot 5 \cdot 16 \cdot 120 + 1 \cdot 120 = 48120$ trainable parameters



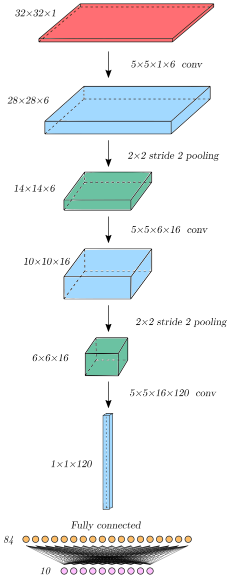
LENET — FULLY CONNECTED LAYER

- Input nodes: 120
- Output nodes: 84
- In total $120 \cdot 84 + 84 = 10164$ parameters



LENET — FULLY CONNECTED OUTPUT LAYER

- Input nodes: 84
- Output nodes: 10 (number of classes in MNIST)
- In total $84 \cdot 10 + 10 = 850$ parameters



ARCHITECTURE — SUMMARY

- Alternates between convolution and pooling layers, finishing with dense layers
- Propagating through the network:
  - Number of channels (feature maps) increase
  - Feature map dimensions reduce
- Number of trainable parameters: 60 850

ALEXNET

Paper	ImageNet Classification with Deep Convolutional Neural Networks
Authors	Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton
Year	2012
Citations	20 340
Link to	pdf

INITIAL NOTES

- At the time superior performance on the ImageNet challenge
- Kick-started the machine-learning renaissance
- Hinted at the importance of depth
- Successful use of dropout and ReLU
- Very efficient convolution implementation
- Distributed the network over 2 GPU's

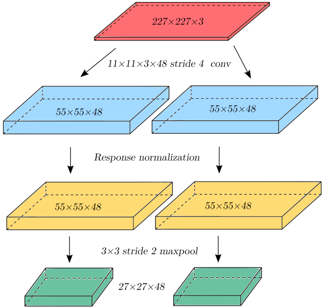
RESPONSE NORMALIZATION

- For a volume of feature maps, with shape $H \times W \times C$
- Activation a_{ijk} at location (i,j) in feature map k is being normalized
- Normalizes w.r.t. neighbouring activations across depth, not w.r.t. spatial neighbours
- Let b_{ijk} be the result, then local response normalization is

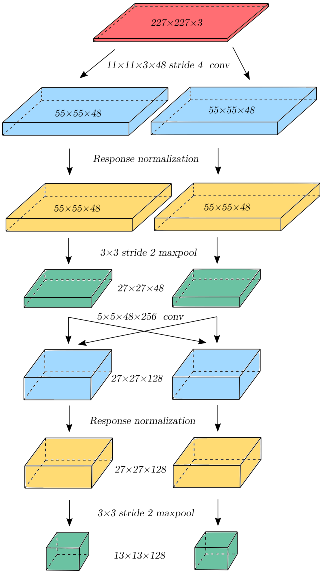
$$b_{ijk} = a_{ijk} \left(\kappa + \alpha \sum_{l=\max\{0,k-n/2\}}^{\min\{N-1,k+n/2\}} a_{ijl}^2 \right)^{-\beta}$$

- N: Number of feature maps in the layer
- n: Number of neighbouring nodes to include
- κ, α, β, hyperparameters

- Input shape: $227 \times 227 \times 3$
- On each gpu: 48 $11 \times 11 \times 3$ convolutions with stride 4
- Response normalization
- 3×3 max pool with stride 2
- Output shape: $27 \times 27 \times 48$ on each gpu

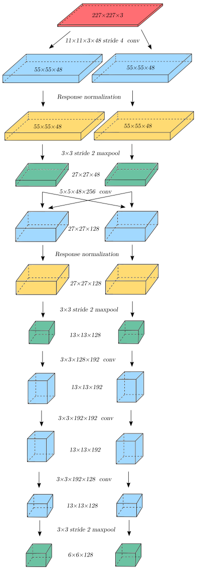


- Input shape: $27 \times 27 \times 48$
- On each gpu: 128 $5 \times 5 \times 256$ convolutions
- Notice the communication between gpus
- Response normalization
- 3×3 max pool with stride 2
- Output shape: $13 \times 13 \times 128$ on each gpu



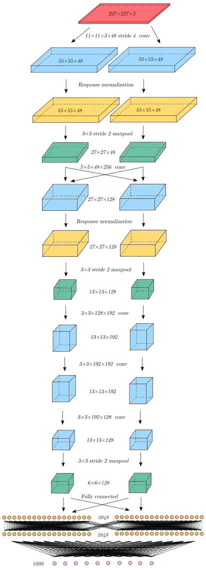
ARCHITECTURE — LAST CONVOLUTION LAYERS

- On each gpu:
    - Input shape: $13 \times 13 \times 128$
    - Conv3: 192 $3 \times 3 \times 128$ convolutions
    - Conv4: 192 $3 \times 3 \times 192$ convolutions
    - Conv5: 128 $3 \times 3 \times 192$ convolutions
- 3×3 max pool with stride 2
- Output shape: $6 \times 6 \times 128$ on each gpu



ARCHITECTURE — DENSE OUTPUT LAYERS

- On each gpu:
    - Input shape: $6 \times 6 \times 128$
    - Dense1: $9216 (= 2 \cdot 6 \cdot 6 \cdot 128) \to 4096$
    - Dense2: $2048 \to 4096$
    - Dense3: $2048 \to 1000$
- Notice communication between gpus
- Final output (1000) is the number of classes



SUMMARY

- Alternating convolution and pooling, finalizing with dense layers
- Reducing spatial dimension, and increasing number of feature maps
- Uses ReLU
- Uses data augmentation, weight decay, and dropout
- Very many parameters compared to LeNet, about 60 million

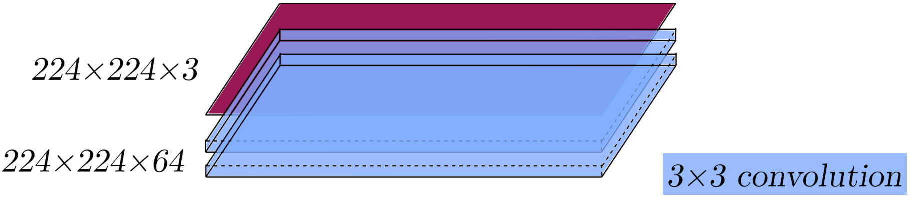
VGG

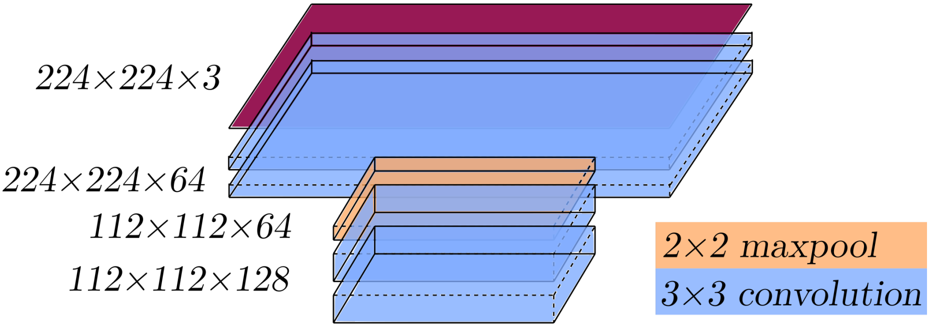
Paper	Very Deep Convolutional Networks for Large-Scale Image Recognition
Authors	Karen Simonyan, and Andrew Zisserman
Year	2014
Citations	9428
Link to	pdf

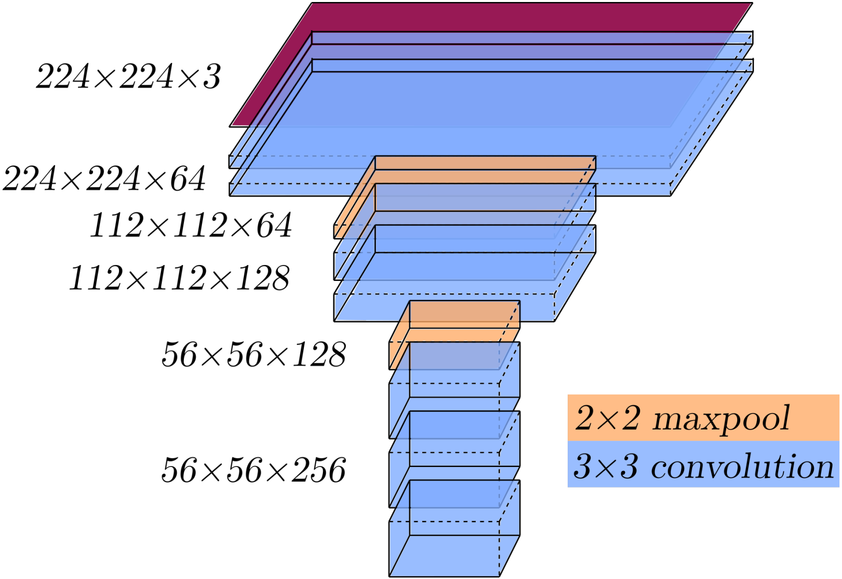
INITIAL NOTES

- Simple and elegant design
- Further investigates the importance of deep nets
- Very good performance on ImageNet
- Very large

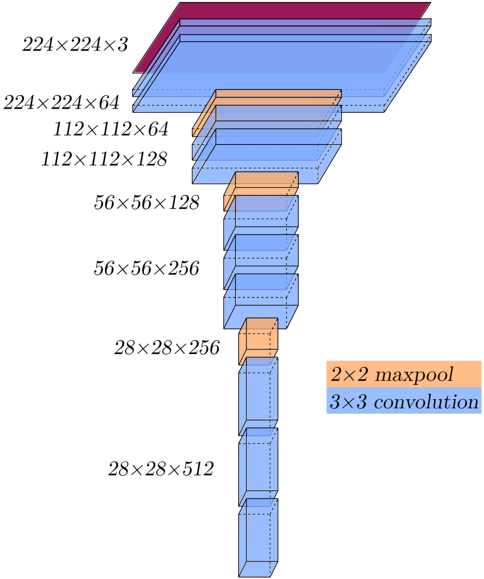
VGG16 — FIRST CONVOLUTION



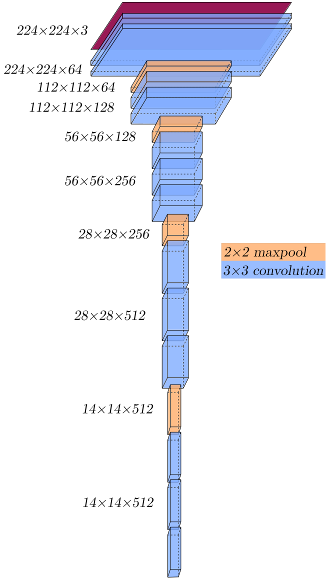

$224\times224\times3$
$224\times224\times64$
$112\times112\times64$
$112\times112\times128$
2×2 maxpool
3×3 convolution


$224\times224\times3$
$224\times224\times64$
$112\times112\times64$
$112\times112\times128$
$56\times56\times128$
$56\times56\times256$
2×2 maxpool
3×3 convolution

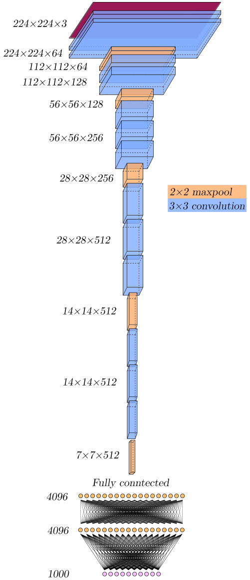
VGG16 — THIRD DOWNSAMPLING



VGG16 — FOURTH DOWNSAMPLING



VGG16 — OUTPUT LAYERS



SUMMARY

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	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	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

INCEPTION / GOOGLENET

Paper Going deeper with convolutions

Authors Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich

Year 2014

Citations 6282

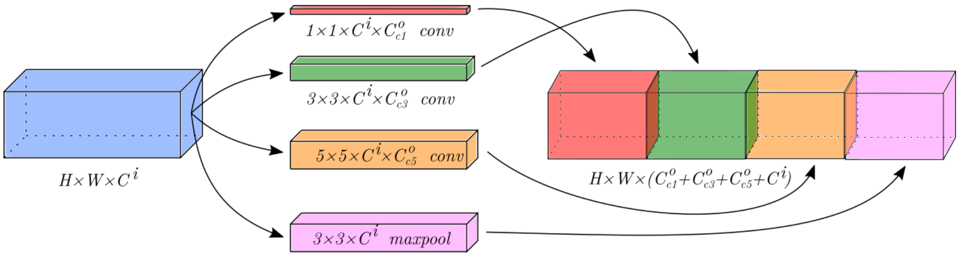
Link to pdf

INITIAL NOTES

- Impressive ImageNet result
- Complex structure with few parameters (anti-thesis of VGG networks)
- 12 times fewer parameters than AlexNet

- Deeper models seem to be key
- Deeper models means more parameters
  - More prone to overfitting
  - Computationally more expensive
- If weights are close to zero, this is wasted
- Tries to create a sparse structure using dense components

- Utilize local correlations on multiple scales
- Common operations
  - 1×1 Convolution
  - 3×3 Convolution
  - 5×5 Convolution
  - Max pooling
- Use all of them



COMPUTATIONAL COST IN CONVOLUTIONS

- 3×3 and 5×5 convolutions are expensive on all input channels
- Solution: Do C_c^b $1 \times 1 \times C^i$ convolutions first
- This creates a *bottleneck* layer with shape $H \times W \times C_c^b$
- Then take C_c^o $5 \times 5 \times C_c^b$ convolutions, where $C_c^b < C_c^i$

- Computational savings

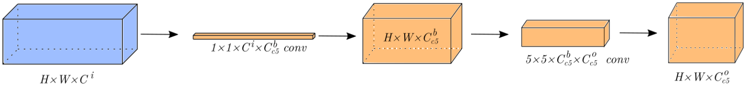
$$n_{ops}^{naive} = (H \cdot W \cdot C_c^o) \cdot (5 \times 5 \times C^i)$$

$$n_{ops}^{improved} = (H \cdot W \cdot C_c^o) \cdot (1 \times 1 \times C^i) + (H \cdot W \cdot C_c^o$$

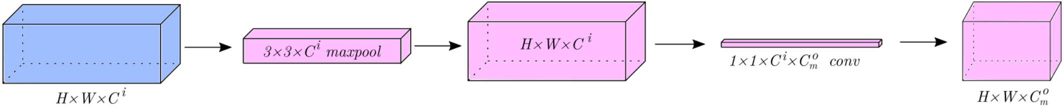
$$\frac{n_{ops}^{naive}}{n_{ops}^{improved}} = \frac{C^i \cdot C_c^o \cdot 5 \cdot 5}{C_c^b(C^i + 5 \cdot 5 \cdot C_c^o)}$$

- With $C^i = 192$, $C_c^b = 16$, $C_c^o = 32$

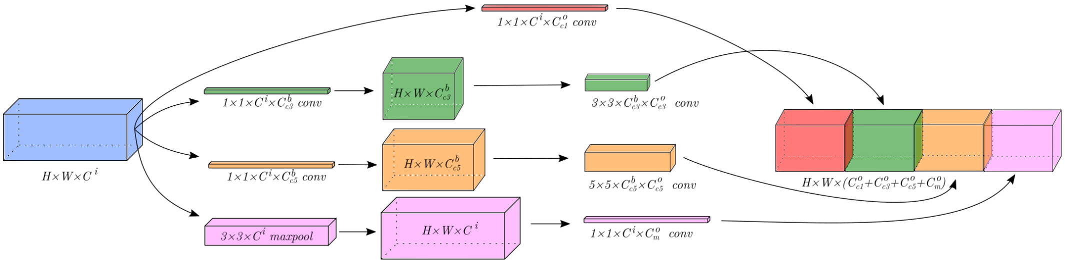
$$\frac{n_{ops}^{naive}}{n_{ops}^{improved}} = 9.68$$



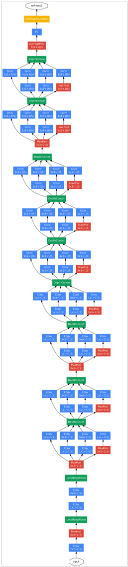
- Maxpool with window size 3×3 and stride 1
- Yields as many output channels C^i as input channels
- Solve this by adding C_m^o $1 \times 1 \times C^i$ convolutions



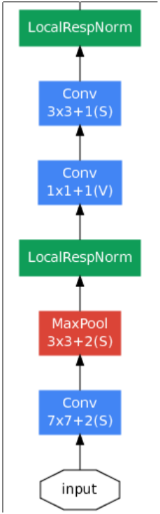
INCEPTION MODULE — FINAL VERSION

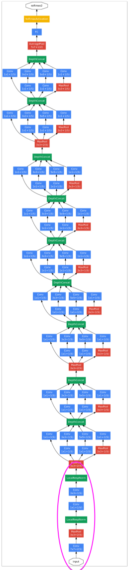


- The inception module controls the number of feature maps
- Can stack multiple inception modules
- Put max-pool layers in between occasionally

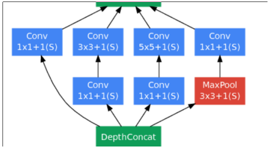


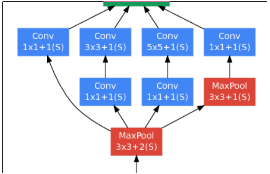
GOOGLENET — INPUT

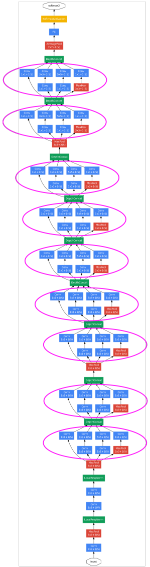




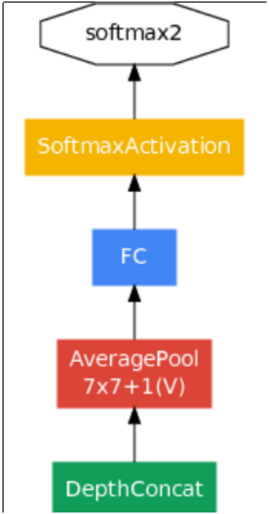
GOOGLENET — INCEPTION MODULES

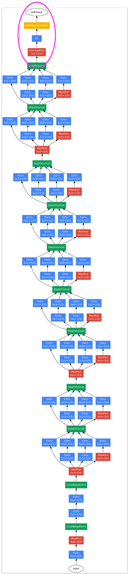






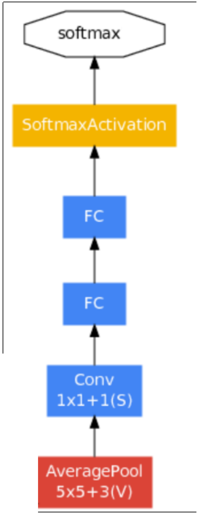
GOOGLENET — OUTPUT

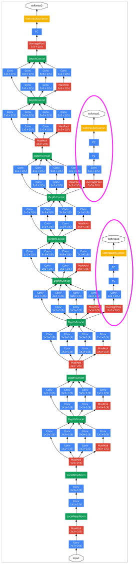




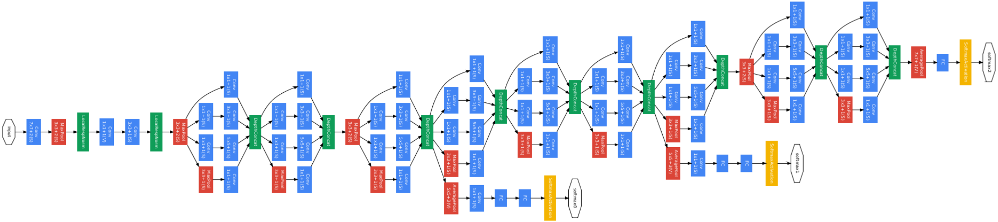
GOOGLENET — VANISHING GRADIENTS

- Backpropagation throught deep network
- Can suffer from “vanishing gradients”
- Potential solution:
  - Intermediate layers can be used discriminatively
  - Add classifiers to intermediate layers
  - Total loss is then a combination of multiple losses


softmax
SoftmaxActivation
FC
FC
Conv
1x1+1(S)
AveragePool
5x5+3(V)



GOOGLENET — FINAL ARCHITECTURE



RESNET

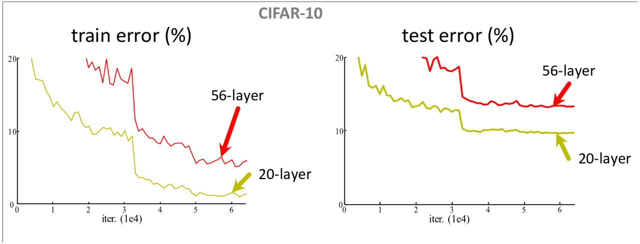
Paper	Deep Residual Learning for Image Recognition
Authors	Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun
Year	2015
Citations	6598
Link to	pdf

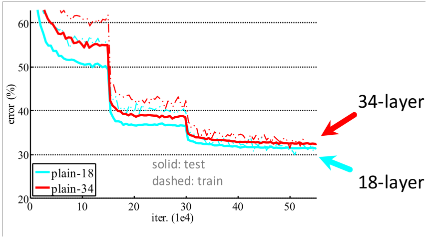
INITIAL NOTES

- “Solved” ImageNet
- Elegant solution to a concrete problem

PROBLEM

- Deeper models seems to be better
- However, very deep models perform worse
- Not due to overtraining
- Degradation problem

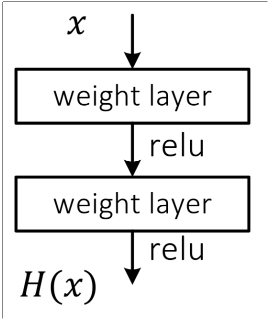




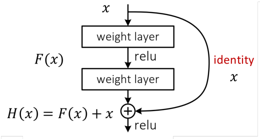
DEEP MODELS SHOULD BE AT LEAST AS GOOD AS SHALLOW

- A deeper model should not have higher training error
- “Proof” by construction
    - Take a shallow model
    - Insert extra layers as identity mappings
    - This deeper mode should have at least as good training error
- How to solve this is the key

- Stack a couple of layers
- Input x
- Let $H(x)$ be the desired mapping to be learned



- Explicitly compose the output as $H(x) = F(x) + x$, by adding the input x
- This means that what has been learnt is the residual $F(x) = H(x) - x$
- This should make identities $H(x) = x$ easier to learn
- Easier to train very deep networks



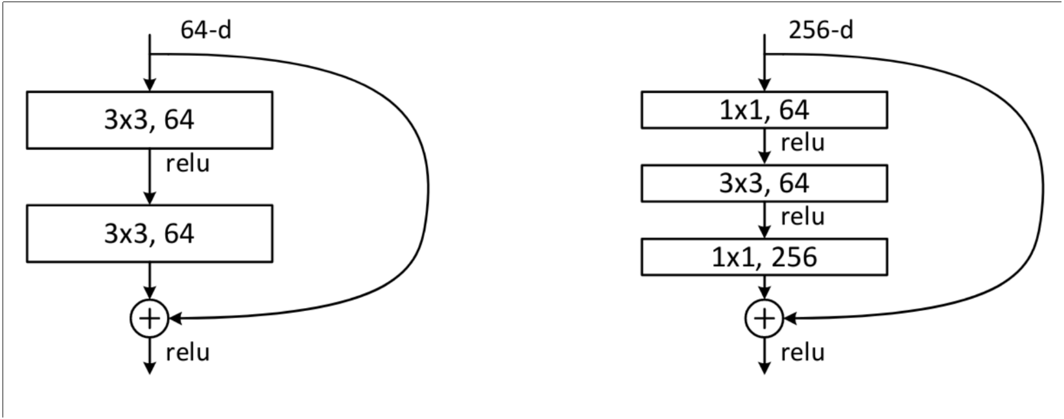
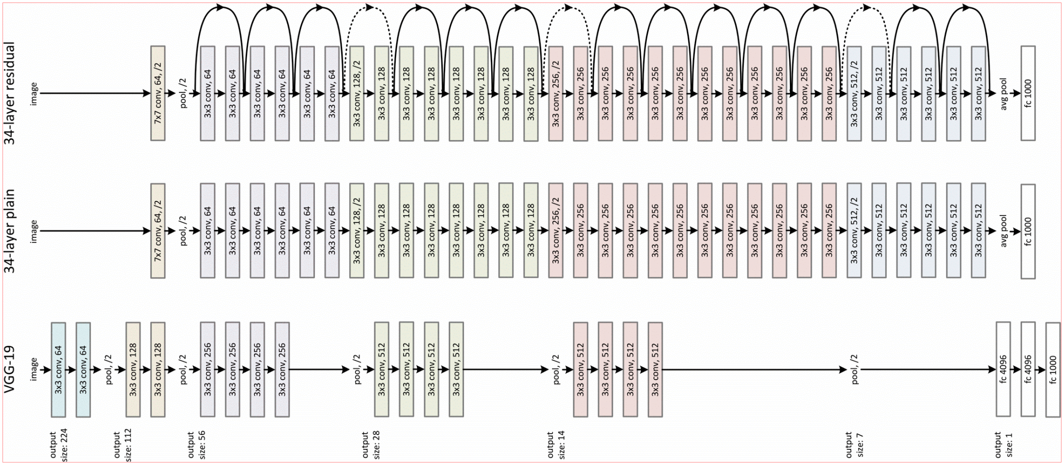


Figure 1: Left: a regular residual block. Right: a “bottleneck” residual block


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

ARCHITECTURE VARIANTS

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

WHY ARE RESIDUAL NETWORKS EFFECTIVE? — IDENTITY MAPPINGS

- Traditional model
  - Layer 3 would benefit from a result in layer 1
  - Layer 2 overrides/destroy this result
  - Layer 3 cannot make use of it, does not adapt to do it either
  - Layer 2 do not adapt so that layer 3 gets to use it, since it does not know that layer 3 needs it
- In resnets, layers compute the identity mapping easily
- Information can “skip” layers
- Layers “only” contribute when they are beneficial

WHY ARE RESIDUAL NETWORKS EFFECTIVE? — UNROLLED ITERATIVE ESTIMATION

Paper: Highway and Residual Networks learn Unrolled Iterative Estimation

- Traditional view:
  - Hierarchical data
  - Models generate increasingly abstract representations
  - Deep models are successful because of their deep representations
- Studies challenges this in resnets:
  - Removing blocks has surprisingly little effect
  - Shuffling blocks has surprisingly little effect
- Instead, blocks (with the same dimensionality “stage”) “collaborate” on refining initial representations

Paper: Residual Networks Behave Like Ensembles of Relatively Shallow Networks

- Residual networks behave like ensambles of shallow networks
- Based on some of the same observations as the previous slide

WHY ARE RESIDUAL NETWORKS EFFECTIVE — ADDITIONAL IDEAS

- It is easier to reuse features in higher layers
    - Deep Networks with Stochastic Depth
    - Highway Networks
- Better gradients, easier optimization
    - The Shattered Gradients Problem: If resnets are the answer, then what is the question?

PRELUDE

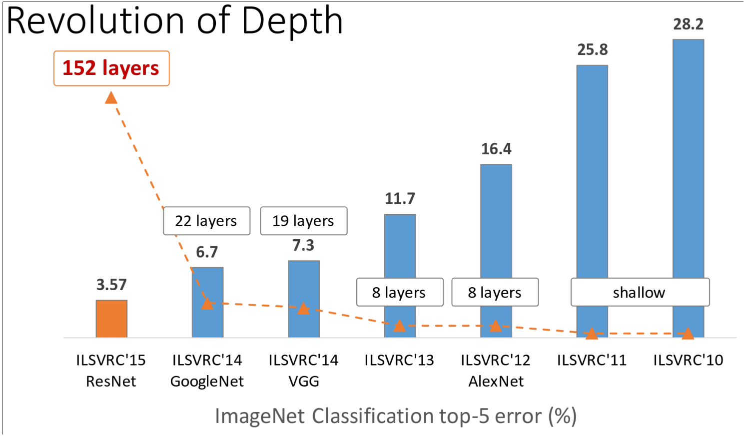


Figure 2: Source: *An analysis of Deep Neural Network Models for Practical Applications.* Canziani, A., Paszke, A., Culurciello, E., 2016

IMAGENET ACCURACY AND SIZE

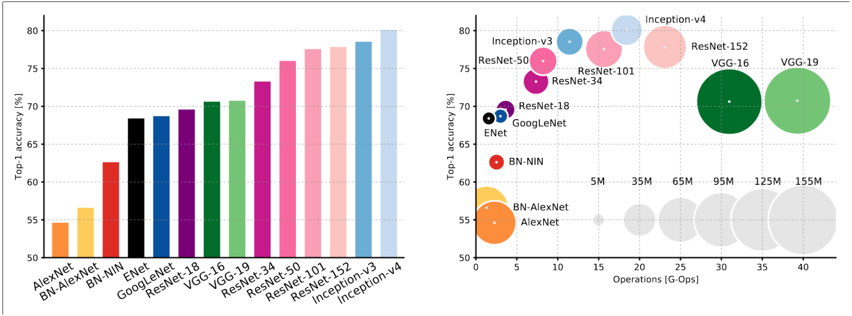


Figure 3: Size and accuracy comparison. Blob size reflects the number of parameters. Source: *An analysis of Deep Neural Network Models for Practical Applications*. Canziani, A., Paszke, A., Culurciello, E., 2016

FINAL THOUGHTS

- As with everything: choose the tool best suited for your problem
- ImageNet top accuracy is not necessarily your ideal metric
- Some things (non-exhaustive) to take into account, in addition to accuracy
  - Training time
  - Inference time
  - Power consumption
  - Memory consumption
  - Processing power consumption
  - Amount of training data
- Hard constraints on the above have shaped current models

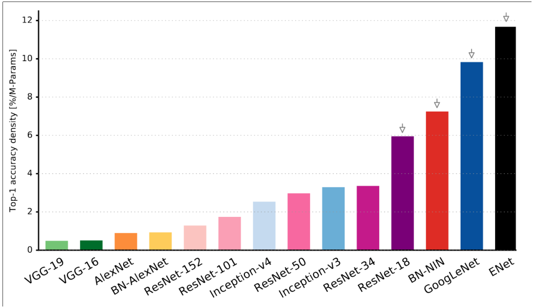


Figure 4: Source: *An analysis of Deep Neural Network Models for Practical Applications*. Canziani, A., Paszke, A., Culurciello, E., 2016

QUESTIONS?