Lecture 6: CNNs I - Architectures

Shenghua Gao

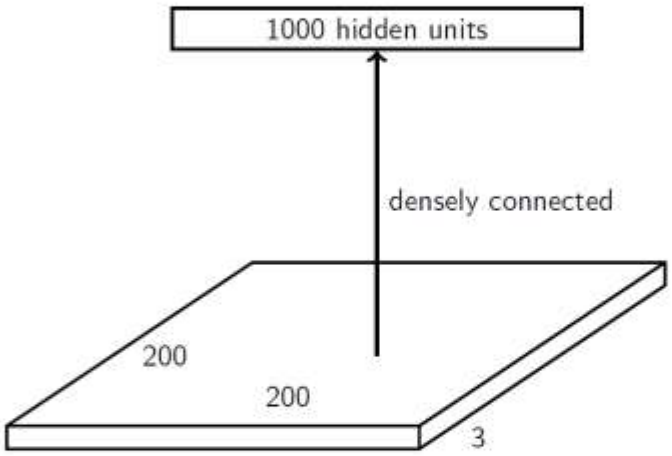
Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

Motivation

- Visual recognition
  - Suppose we aim to train a network that takes a 200x200 RGB image as input



  - What is the problem with have full connections in the first layer?
    - Too many parameters! 200x200x3x1000 = 120 million
    - What happens if the object in the image shifts a little?

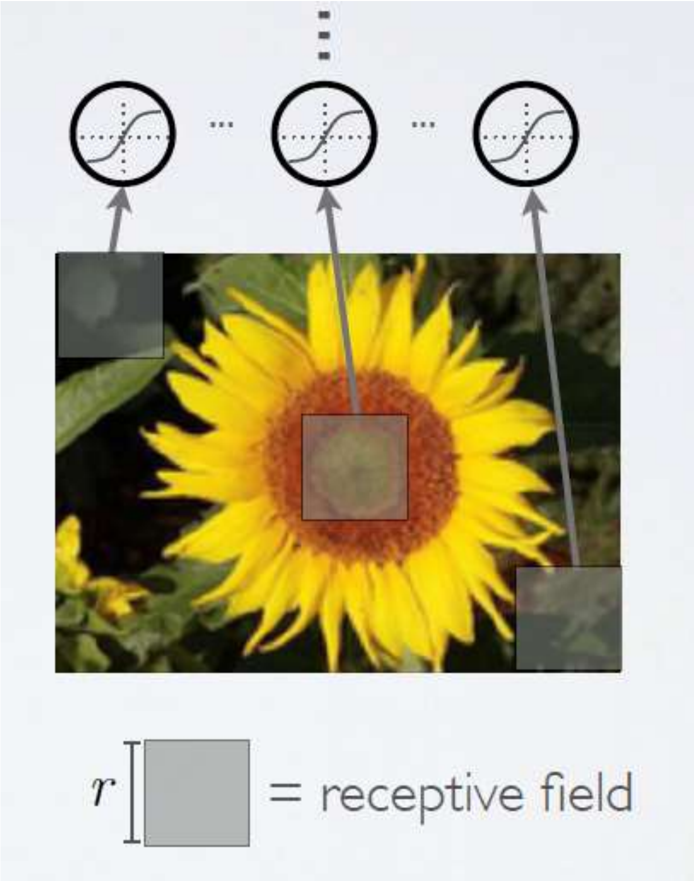
Our goal

- Visual Recognition: Design a neural network that
  - Much deal with very high-dimensional inputs
  - Can exploit the 2D topology of pixels in images
  - Can build in invariance/equivariance to certain variations we can expect
    - Translation, small deformations, illumination, etc.

- Convolution networks leverage these ideas
  - Local connectivity
  - Parameter sharing
  - Pooling/subsampling hidden units

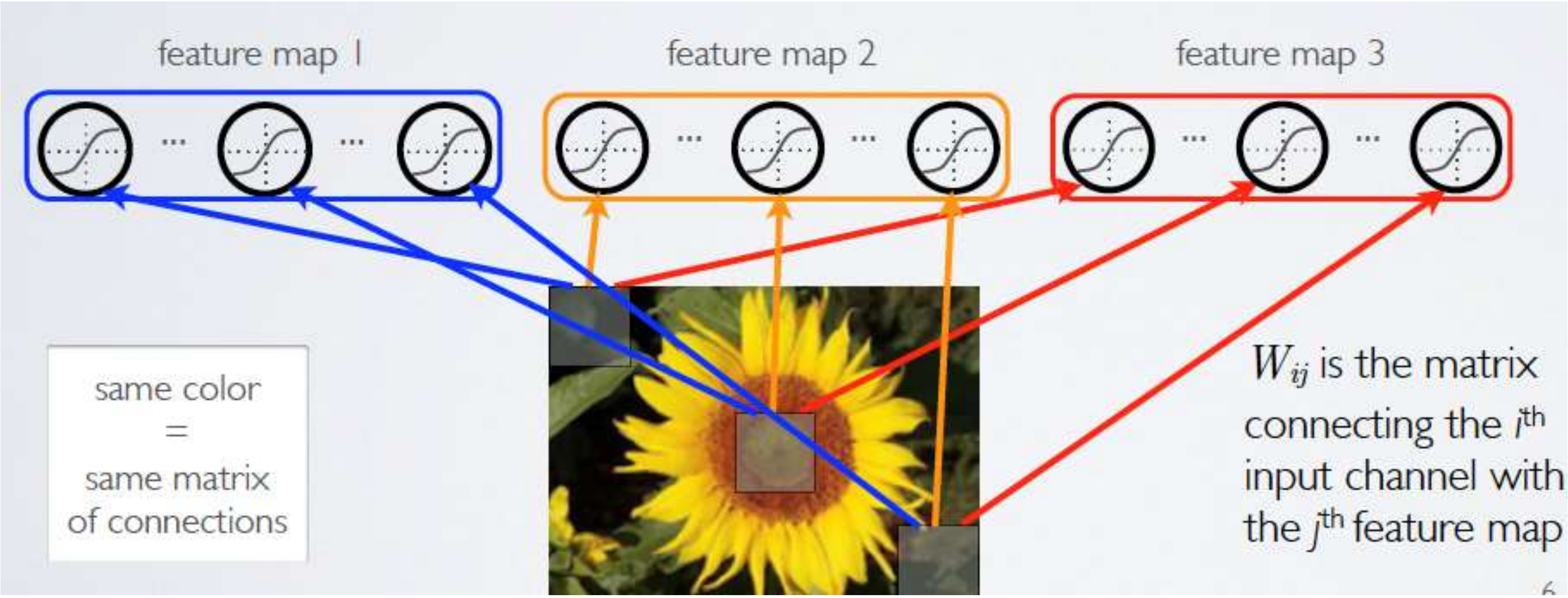
Overview of CNNs

- First idea: **Use a local connectivity of hidden units**
  - Each hidden unit is connected only to a subregion (patch) of the input image
  - Usually it is connected to all channels
  - Each neuron has a local receptive field



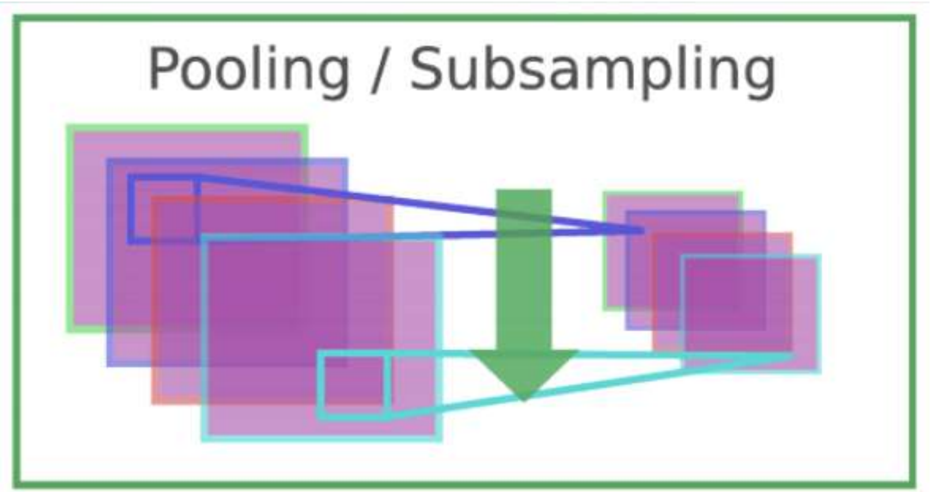
Overview of CNNs

- **Second idea: share weights across certain units**
  - Units organized into the same "feature map" share weight parameters
  - Hidden units within a feature map cover different positions in the image



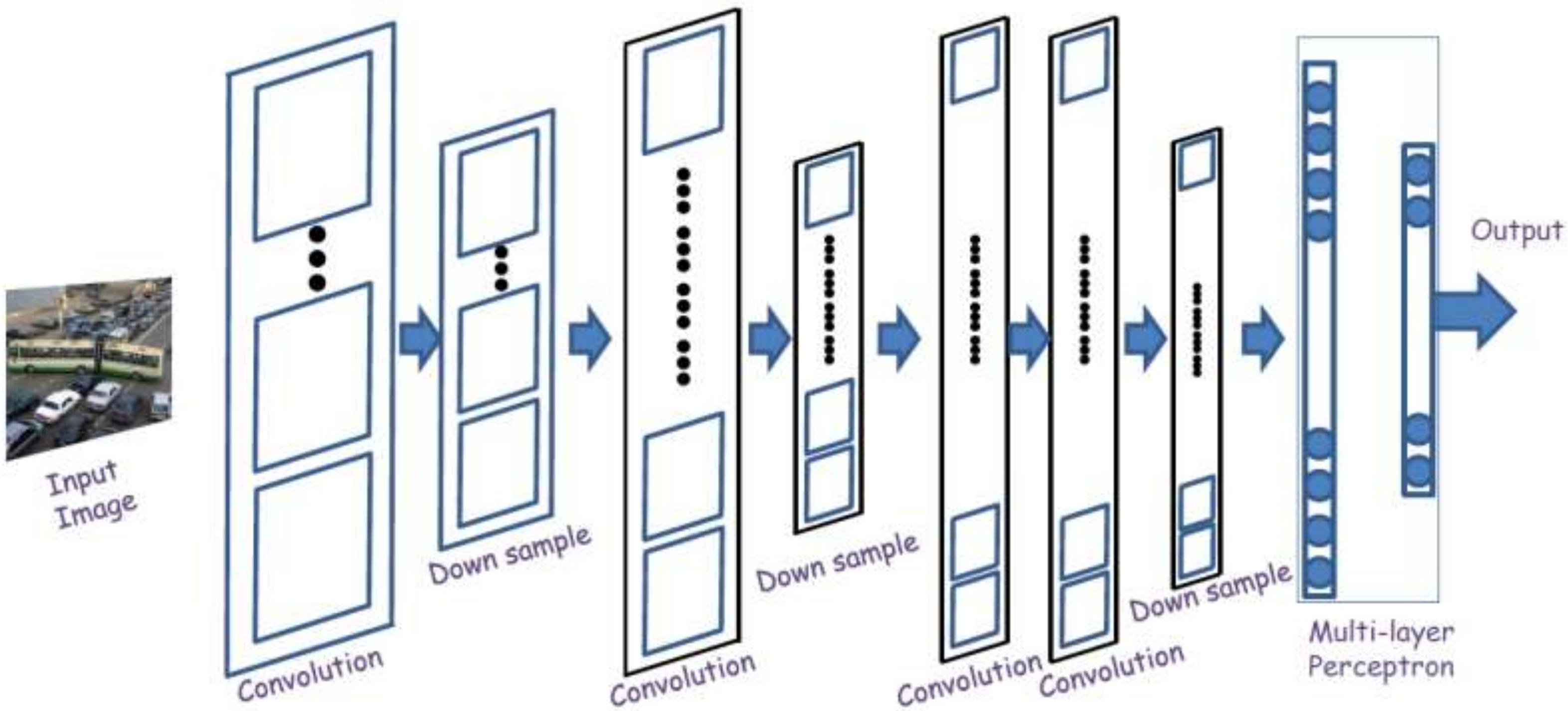
Overview of CNNs

- **Third idea: pool hidden units in the same neighborhood**
  - Averaging or Discarding location information in a small region
  - Robust toward small deformations in object shapes by ignoring details.



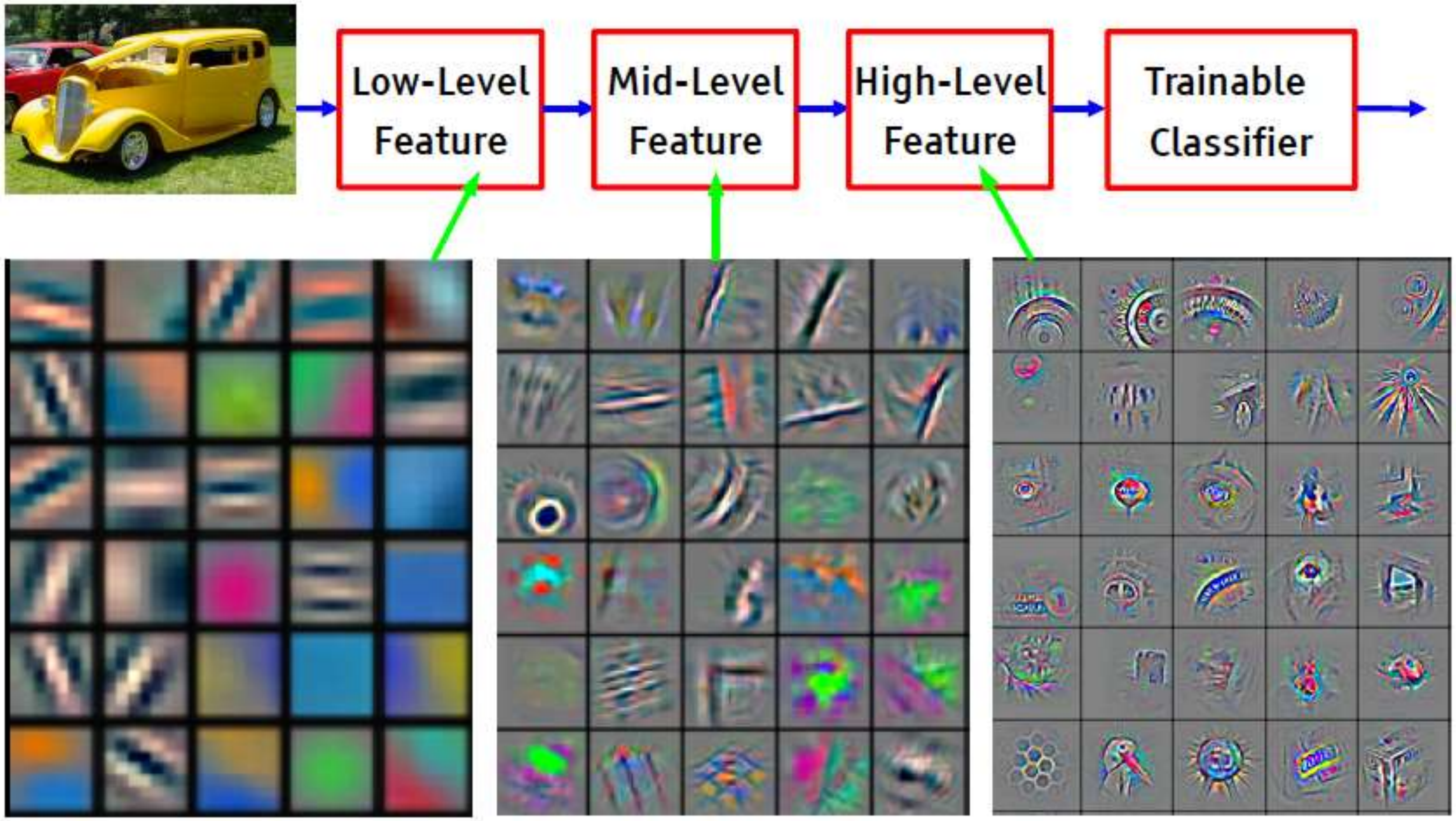
Overview of CNNs

- **Fourth idea: Interleaving feature extraction and pooling operations**
  - Extracting abstract, compositional features for representing semantic object classes



Overview of CNNs

- Artificial visual pathway: from images to semantic concepts (Representation learning)



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

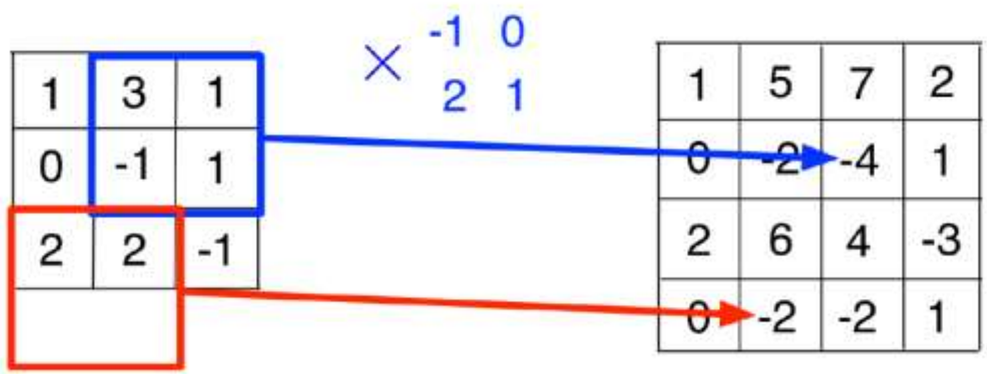
Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

2D Convolution

If A and B are two 2-D arrays, then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.$$

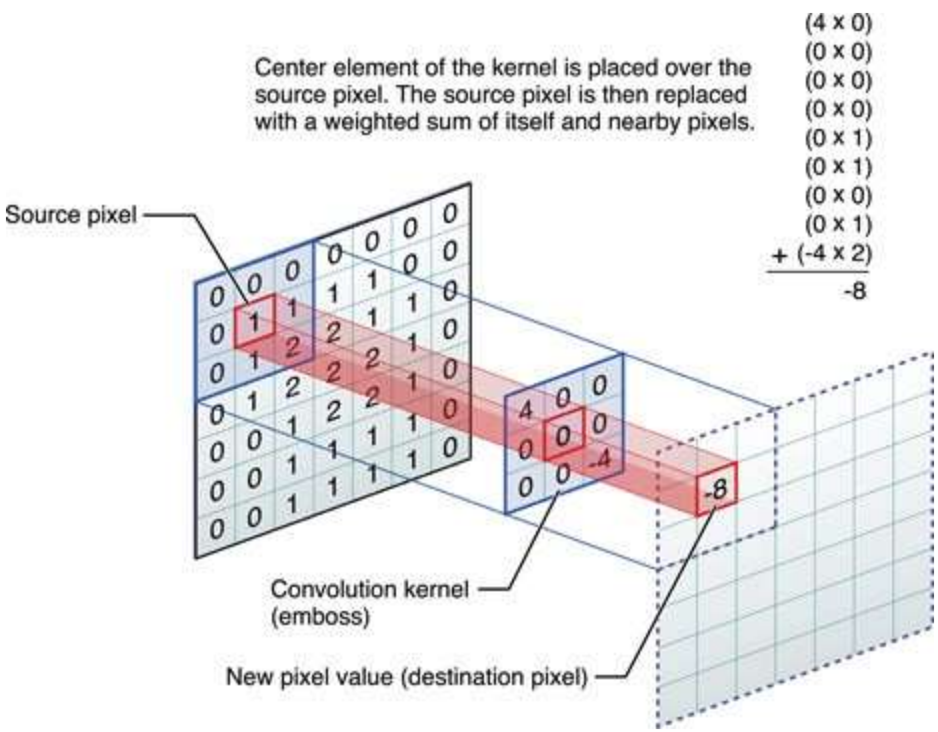
$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} \ast \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array}$$

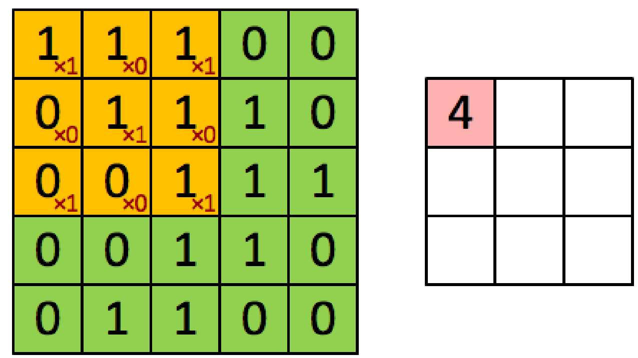


2D Convolution

If A and B are two 2-D arrays, then:

$$(A * B)_{ij} = \sum_{s} \sum_{t} A_{st} B_{i-s,j-t}.$$





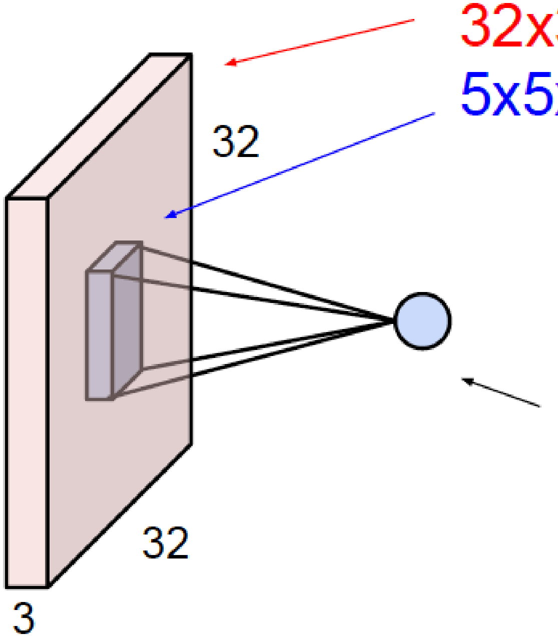
Image

Convolved Feature

Picture Courtesy: developer.apple.com

Convolution Layers

- Formal definition

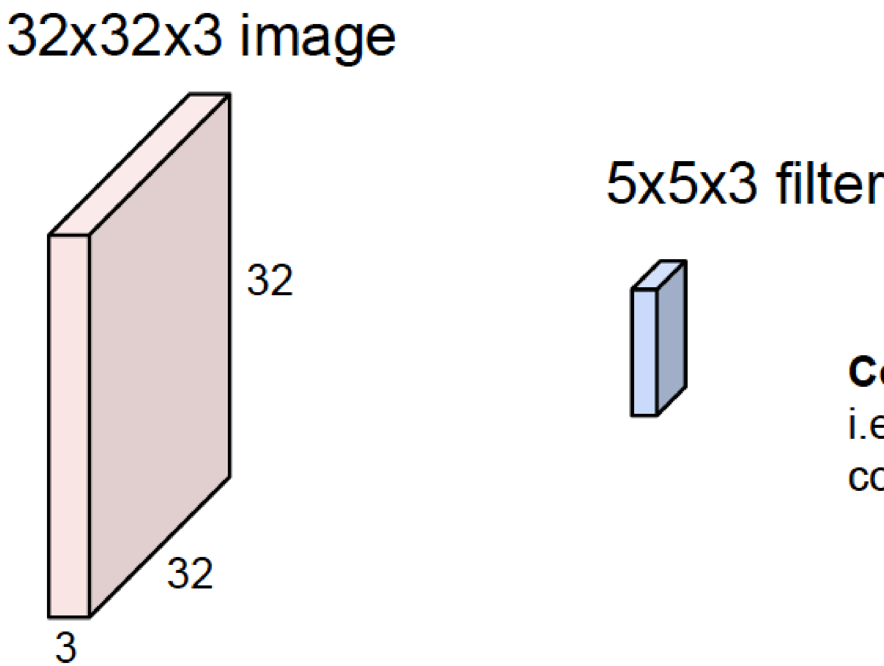


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layers

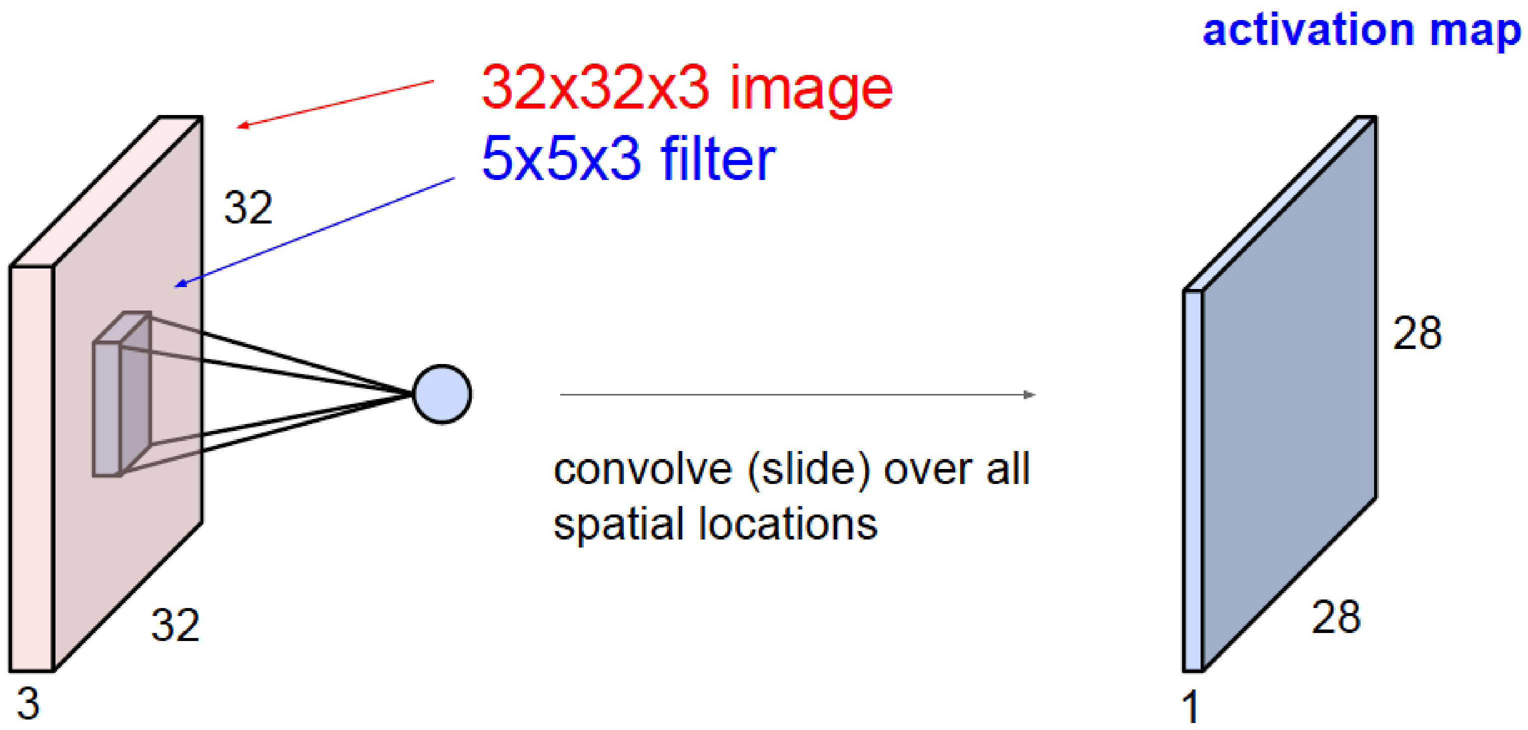
- Define a neuron corresponding to a 5x5 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

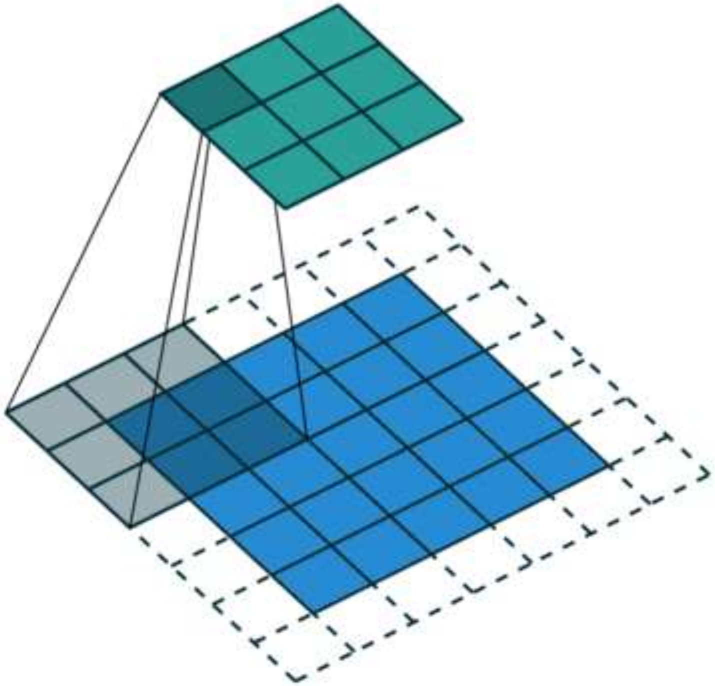
Convolution Layers

- Convolution operation
  - Parameter sharing
  - Spatial information



Convolution Layers

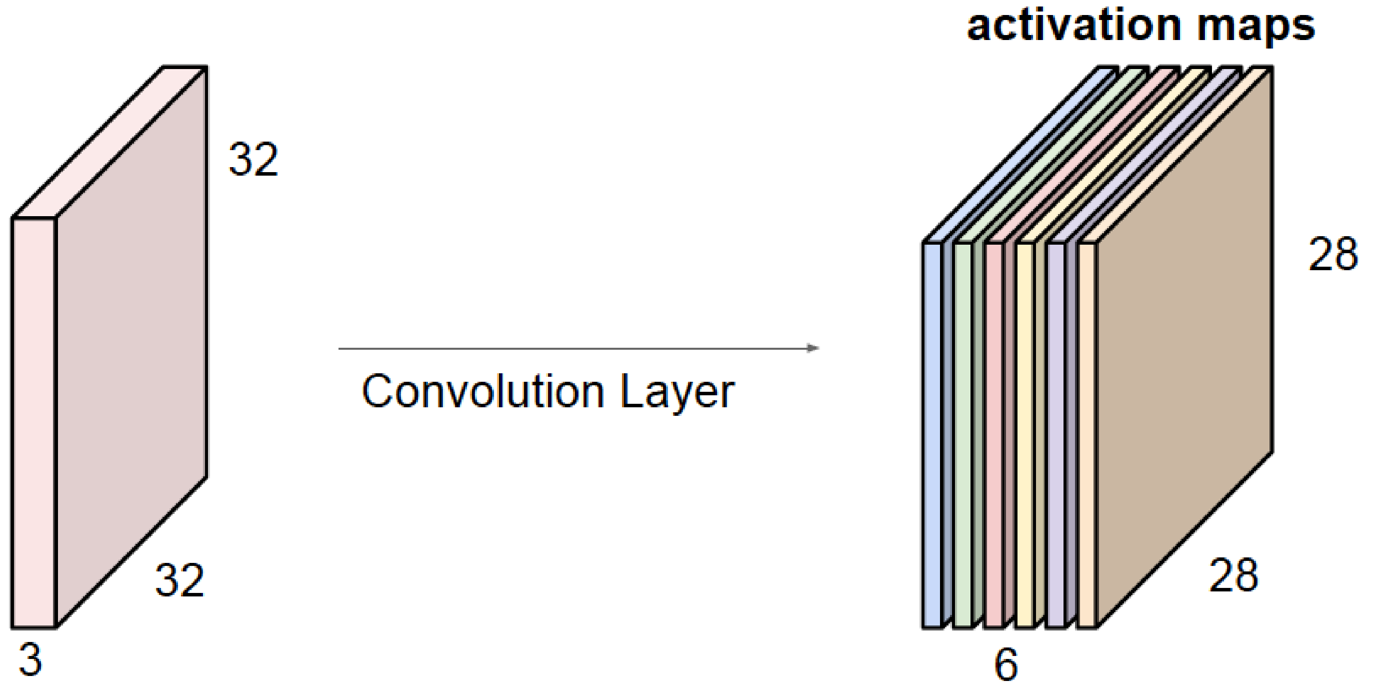
- Convolution operation
  - Parameter sharing
  - Spatial information



Convolution Layers

- Multiple kernels/filters

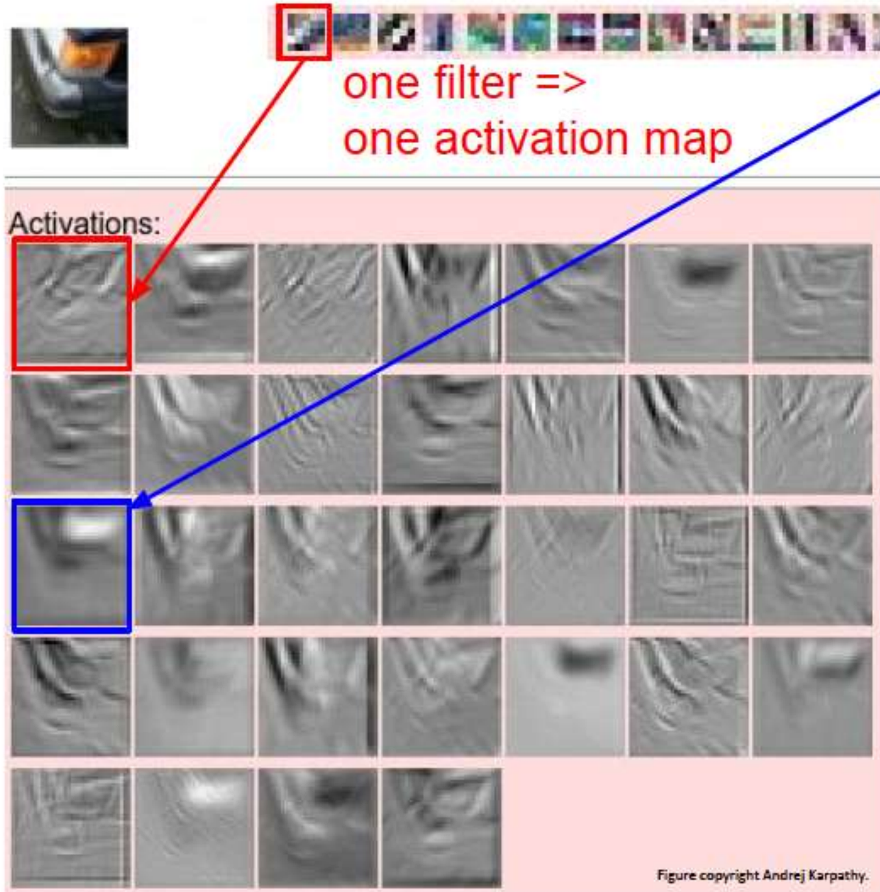
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolution Layers

- Visualizing the filters and their outputs



example 5x5 filters
(32 total)

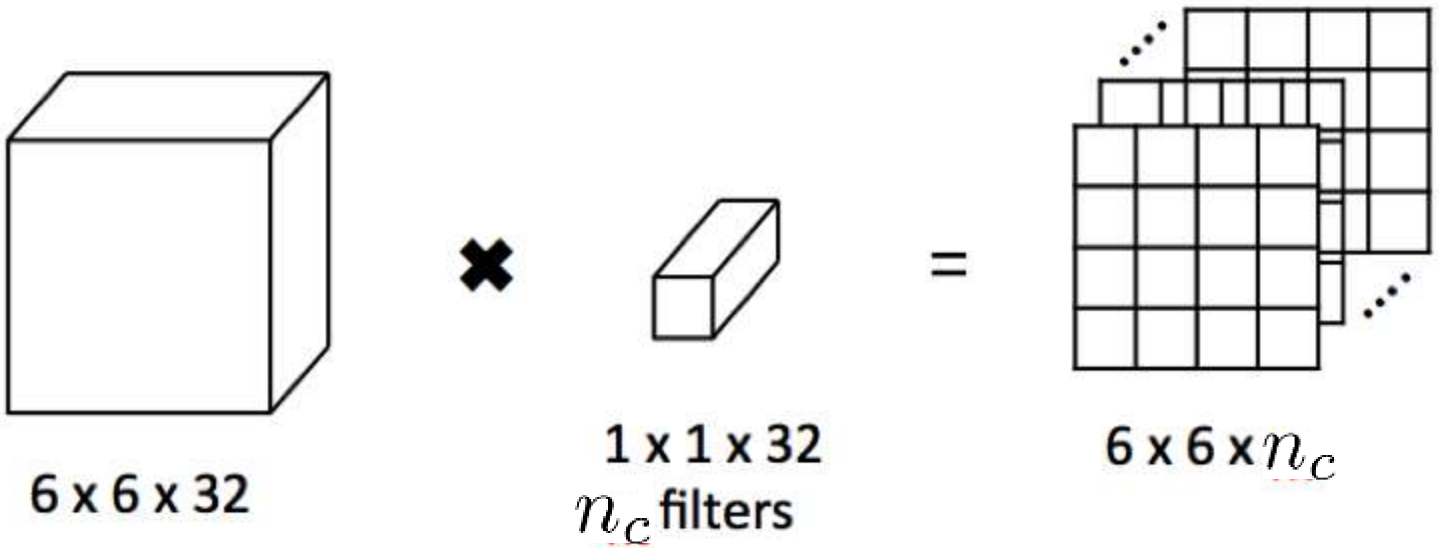
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

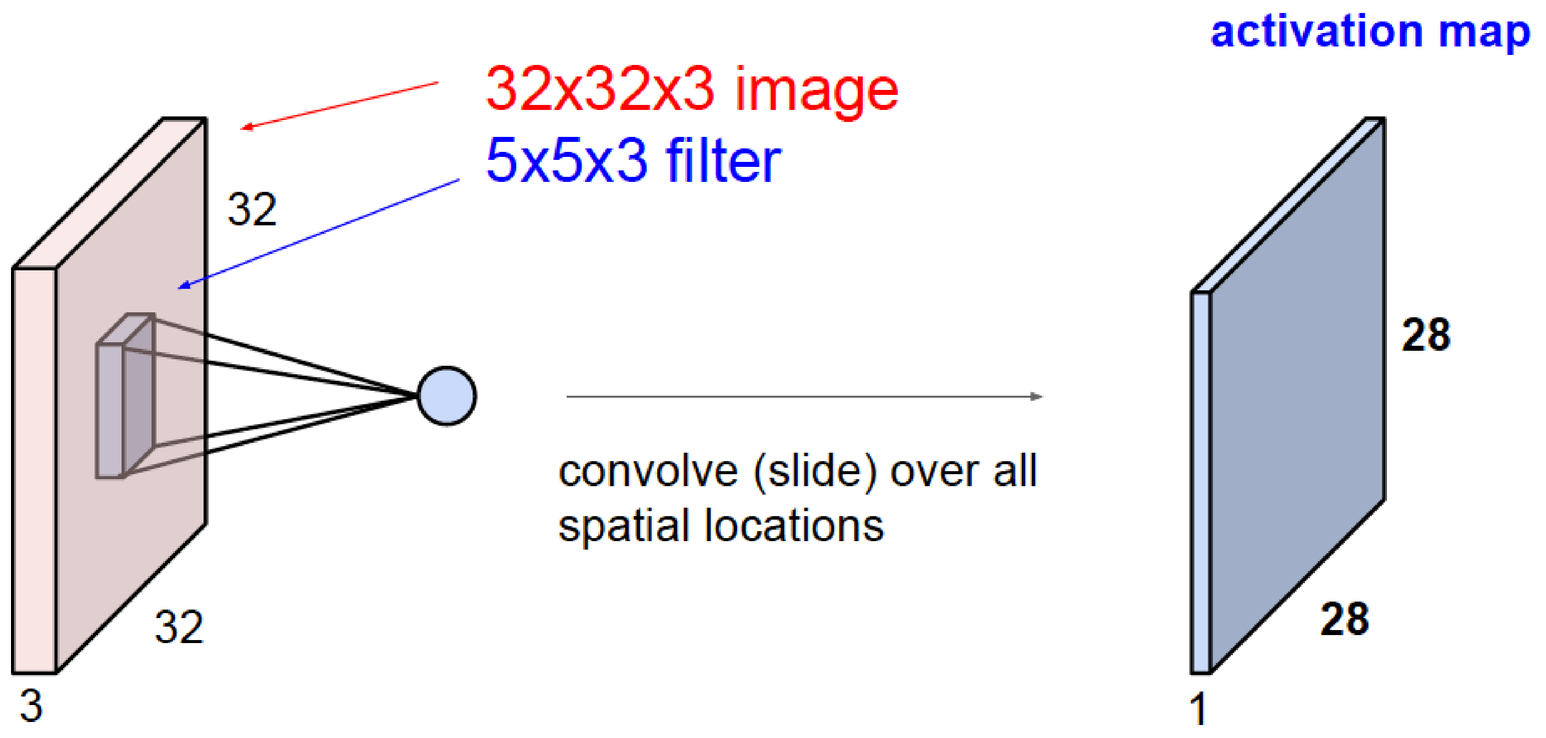
Special Convolutions

- 1x1 convolutions
  - Used in Network-in-network, GoogleNet
  - Reduce or increase dimensionality
  - Can be considered as ‘feature pooling”



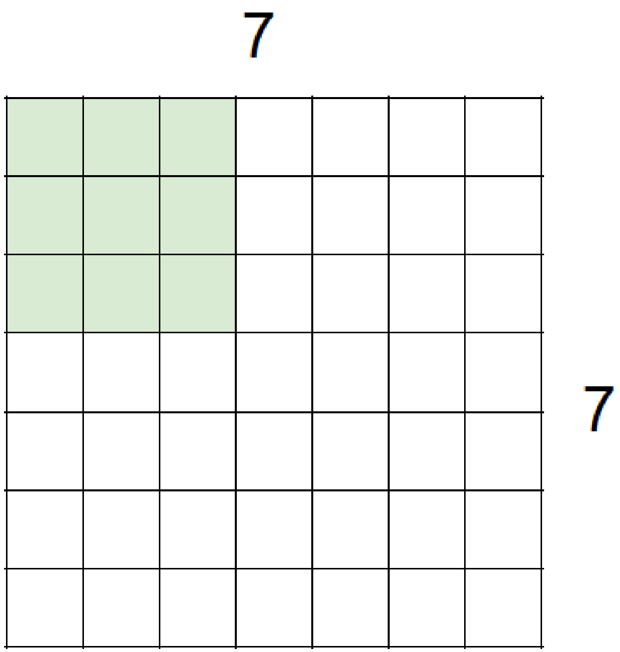
Complexity of Convolution Layers

- Sizes of activation maps and number of parameters



Complexity of Convolution Layers

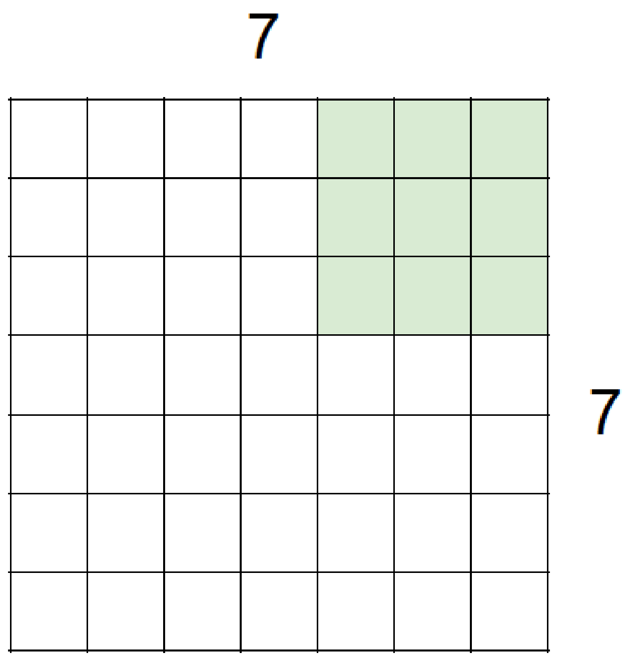
- Size of activation maps



7x7 input (spatially)
assume 3x3 filter

Complexity of Convolution Layers

- Size of activation maps

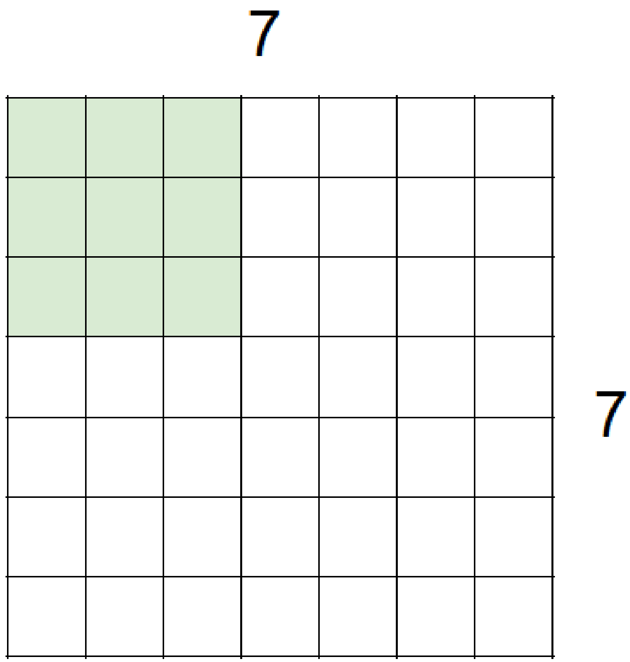


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Complexity of Convolution Layers

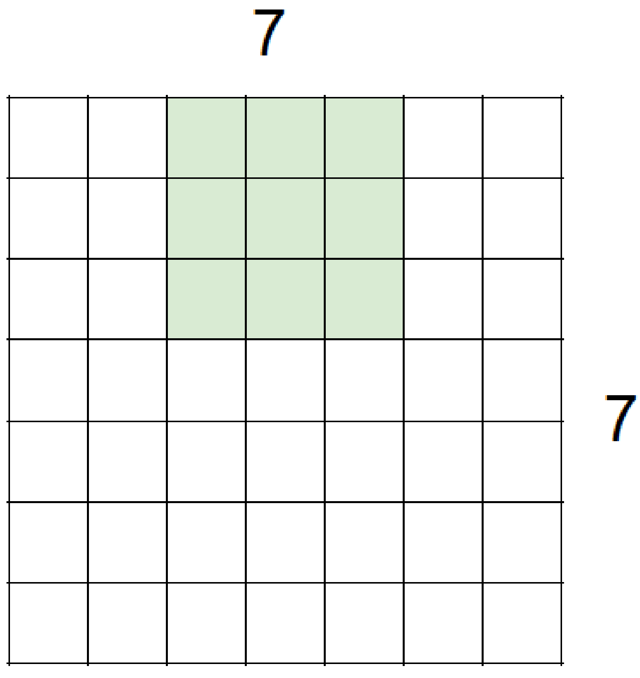
- Case: Stride > 1



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Complexity of Convolution Layers

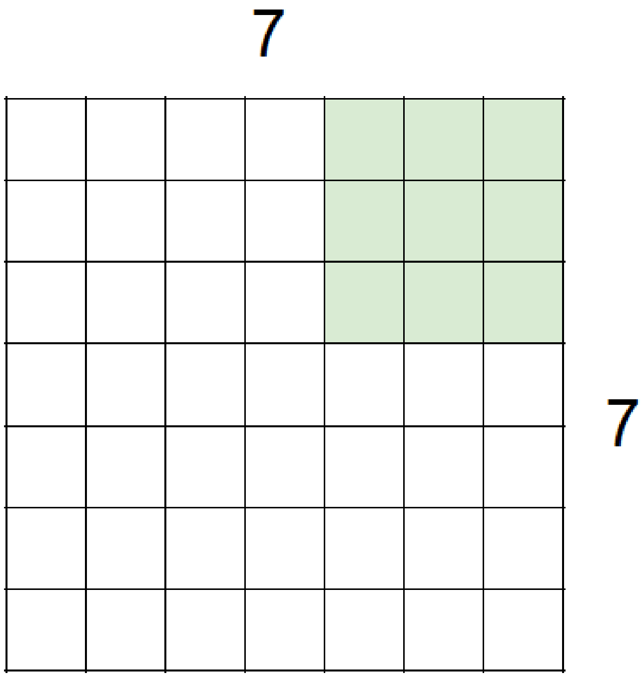
- Case: Stride > 1



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Complexity of Convolution Layers

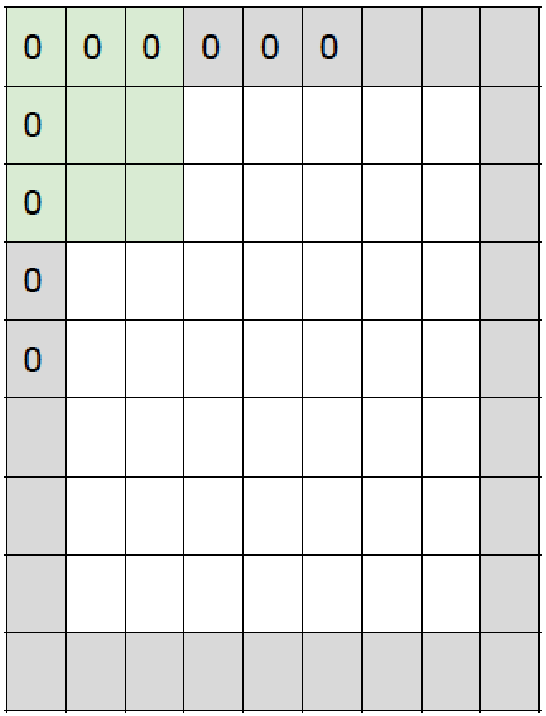
- Case: Stride > 1



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Complexity of Convolution Layers

- Zero padding to handle non-integer cases or control the output sizes



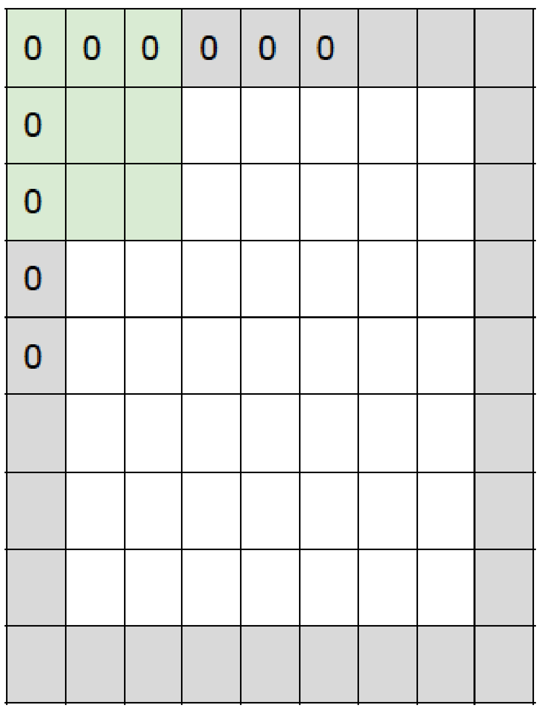
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N - F) / stride + 1

Complexity of Convolution Layers

- Zero padding to handle non-integer cases or control the output sizes

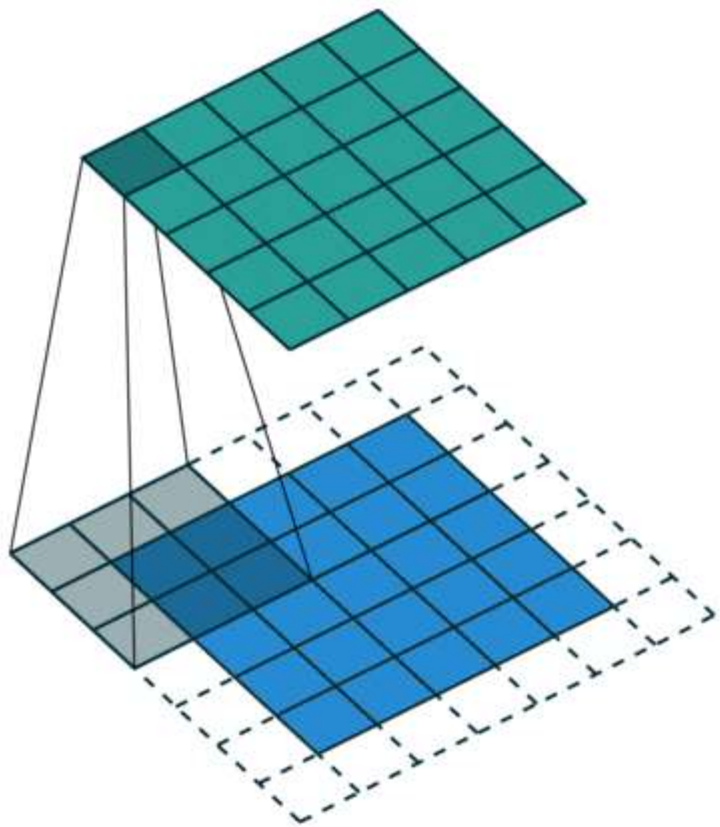


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

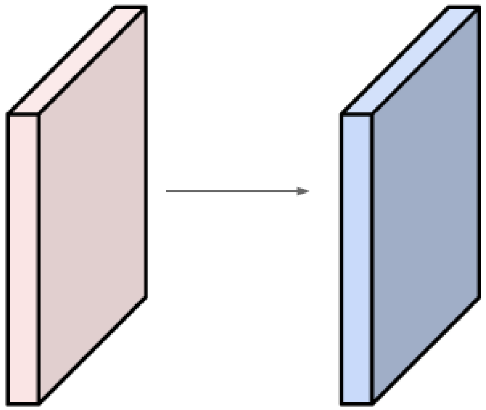
Complexity of Convolution Layers

- Zero padding to handle non-integer cases or control the output sizes



Complexity of Convolution Layers

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

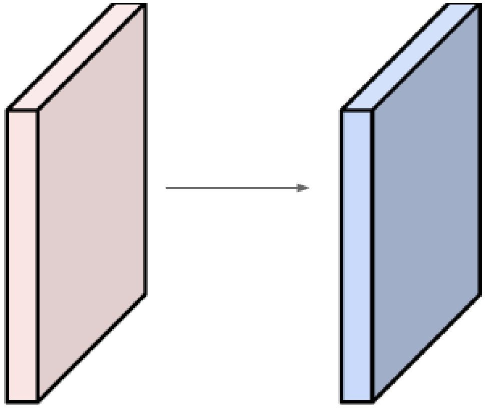
Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

Complexity of Convolution Layers

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

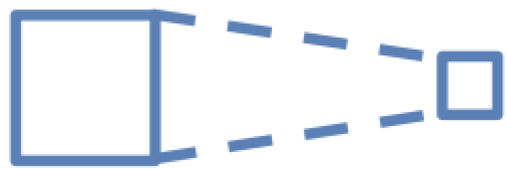
Complexity of Convolution Layers

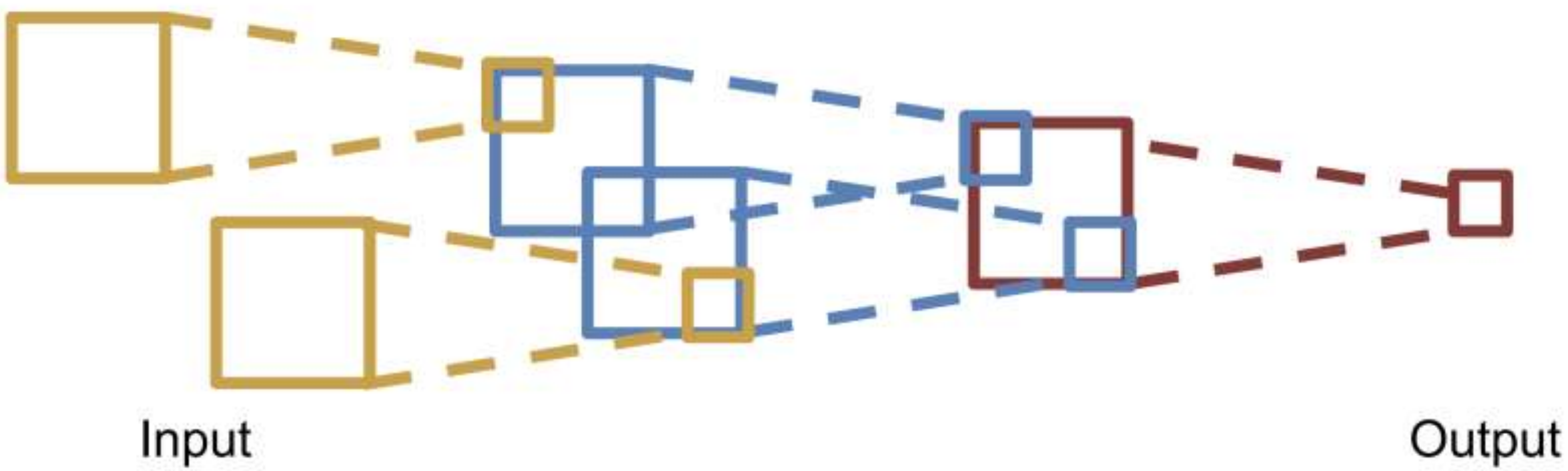
- Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
    - Number of filters K,
    - their spatial extent F,
    - the stride S,
    - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F + 2P)/S + 1$
    - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
    - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Receptive Fields

- For convolution with kernel size K, each element in the output depends on a K x K receptive field in the input





Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

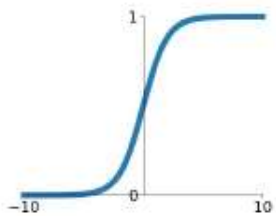
Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

Review: Activation Function

- Zoo of Activation functions

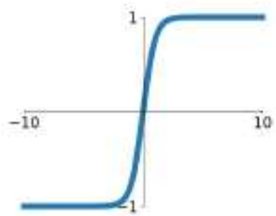
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



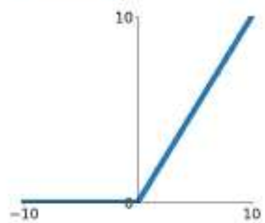
tanh

$\tanh(x)$



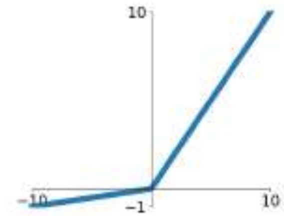
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

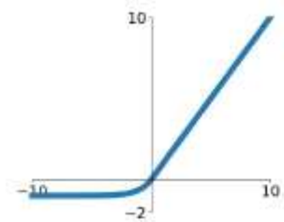


Maxout

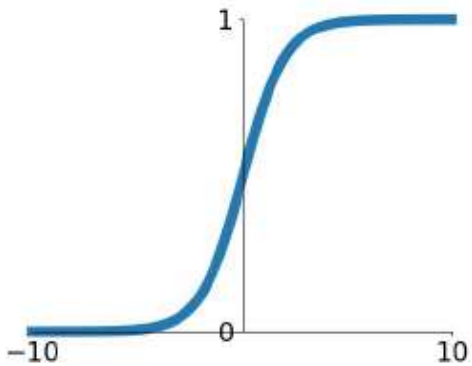
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Sigmoid function



Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

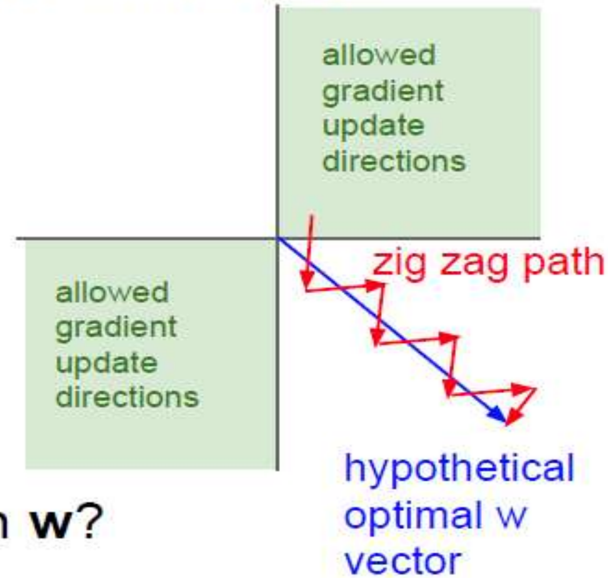
3 problems:

1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Sigmoid function

Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?

Always all positive or all negative :(

(this is also why you want zero-mean data!)

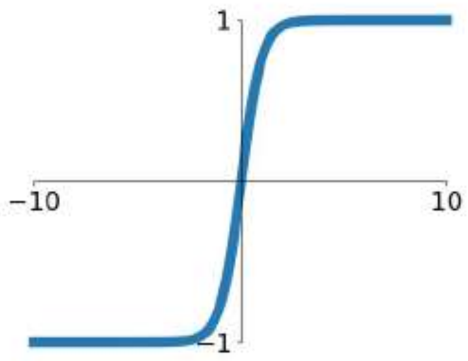
$$f = \sum w_i x_i + b$$

$$\frac{df}{dw_i} = x_i$$

$$\frac{dL}{dw_i} = \frac{dL}{df}\frac{df}{dw_i} = \frac{dL}{df}x_i$$

because $x_i > 0$, the gradient $\frac{dL}{dw_i}$ always has the same sign as $\frac{dL}{df}$ (all positive or all negative

Tanh function



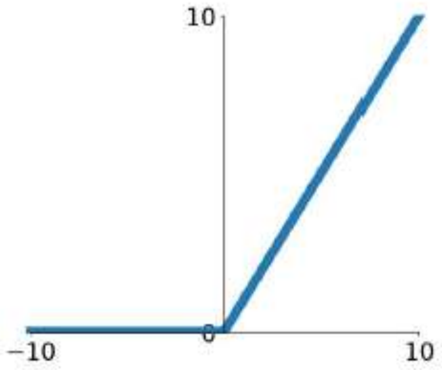
tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

[LeCun et al., 1991]

- Recurrent neural networks: LSTM, GRU

Rectified Linear Unit



ReLU
(Rectified Linear Unit)

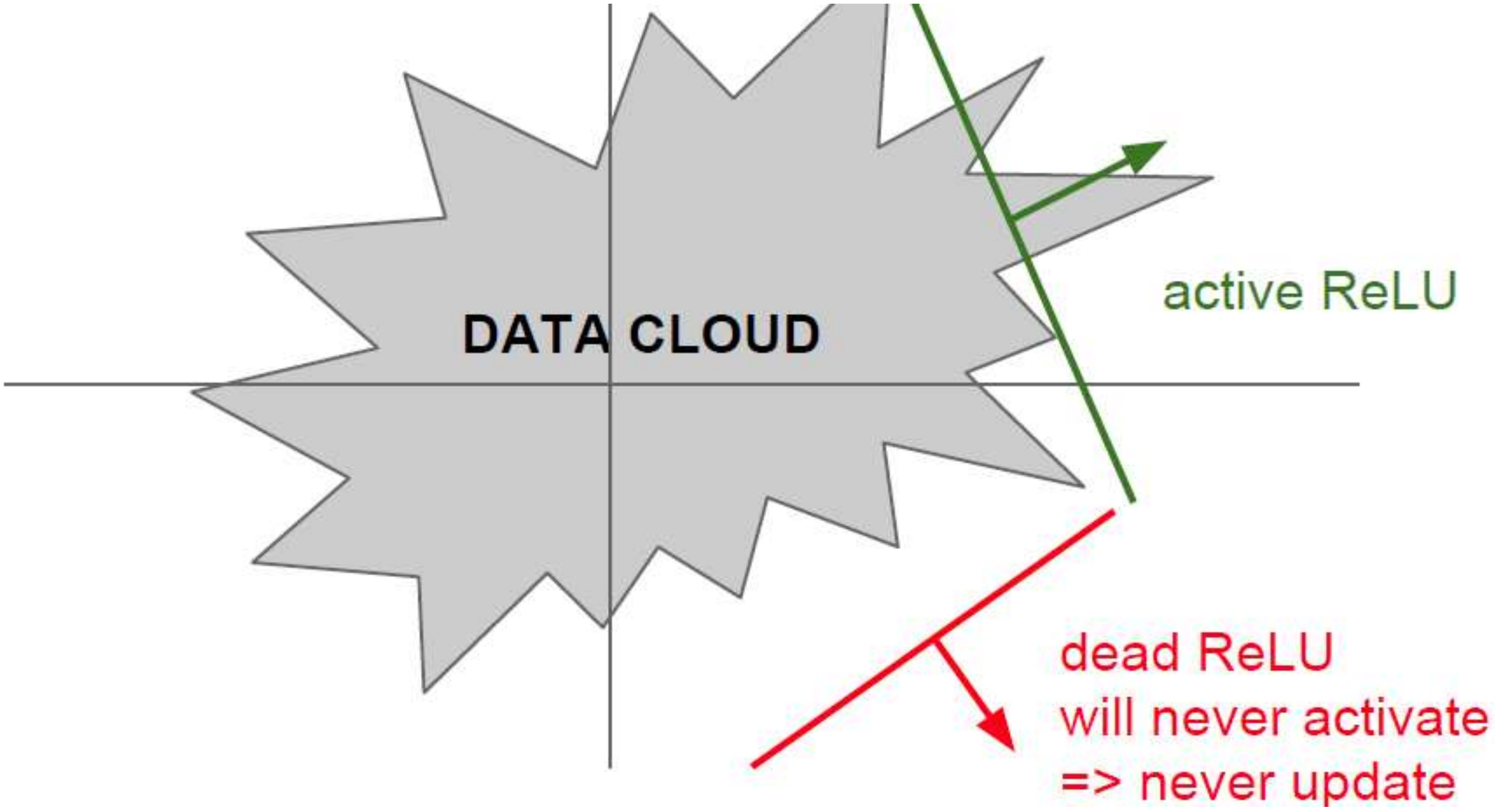
- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

- Not zero-centered output
- An annoyance:

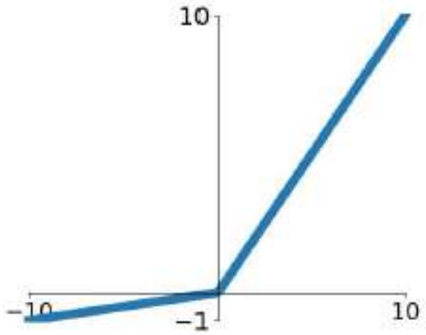
hint: what is the gradient when x < 0?

Rectified Linear Unit



Leaky ReLU

[Mass et al., 2013]
[He et al., 2015]



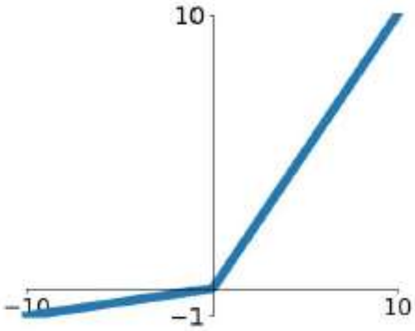
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Leaky ReLU

[Mass et al., 2013]
[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

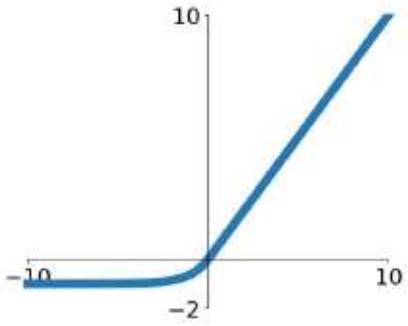
$$f(x) = \max(\alpha x, x)$$

backprop into \alpha
(parameter)

Exponential Linear Units (ELU)

[Clevert et al., 2015]

Exponential Linear Units (ELU)



- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky ReLU adds some robustness to noise

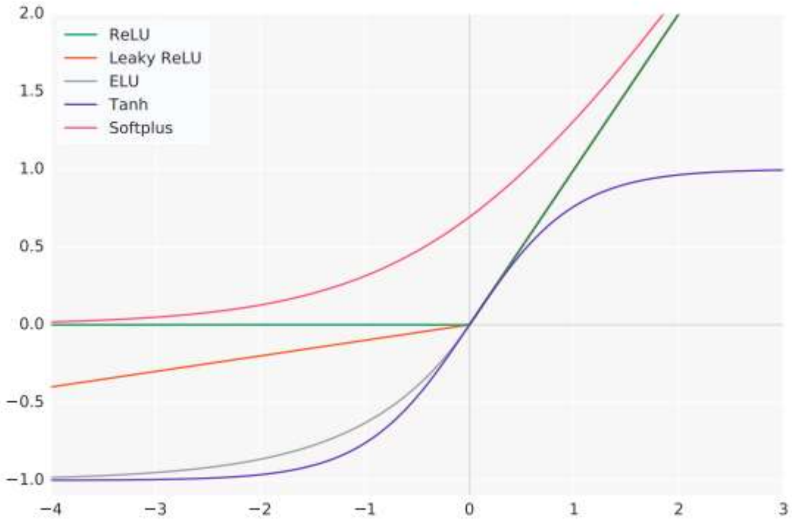
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$

- Computation requires exp()

Summary: Activation function

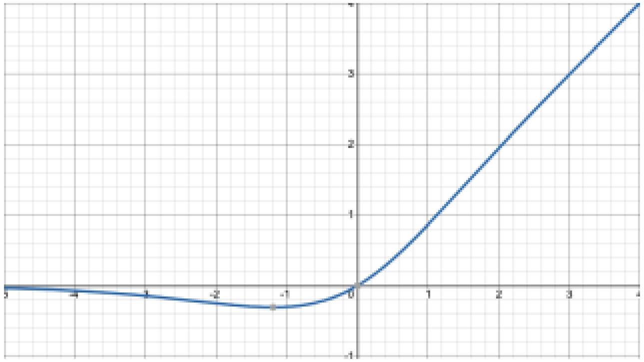
- For internal layers in CNNs
  - Use ReLU. Be careful with your learning rates
  - Try out Leaky ReLU / Maxout / ELU
  - Try out tanh but don't expect much
  - Don't use sigmoid

- For output layers
  - Task dependent
  - Related to your loss function



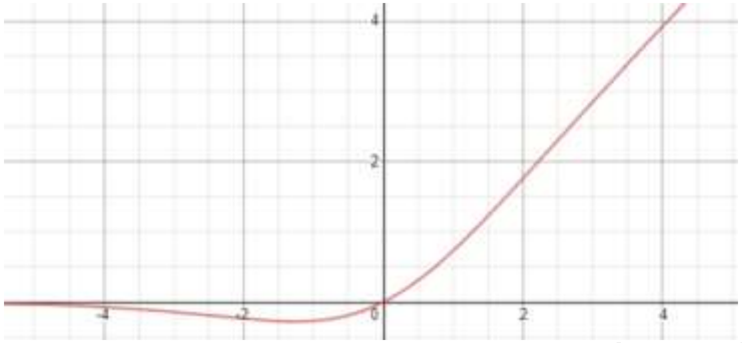
Summary: Activation function

- Recent progresses
  - Mish $f(x) = x \cdot \tanh(\varsigma(x))$, $\varsigma(x) = \ln(1+e^{x})$,



https://arxiv.org/abs/1908.08681

  - Swish $f(x) = x * (1 + \exp(-x))^{-1}$



https://arxiv.org/abs/1710.05941

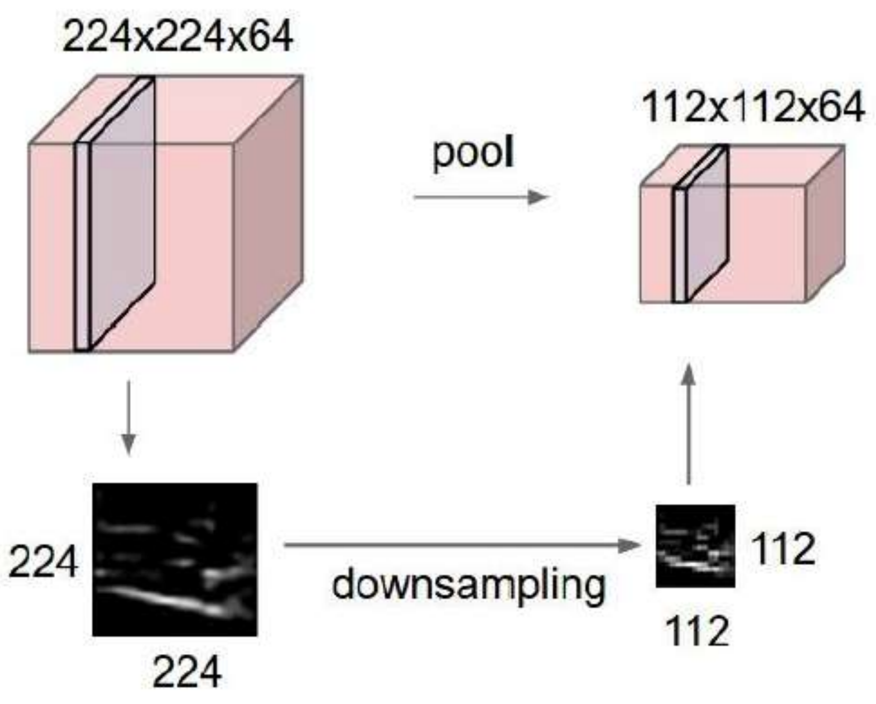
Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

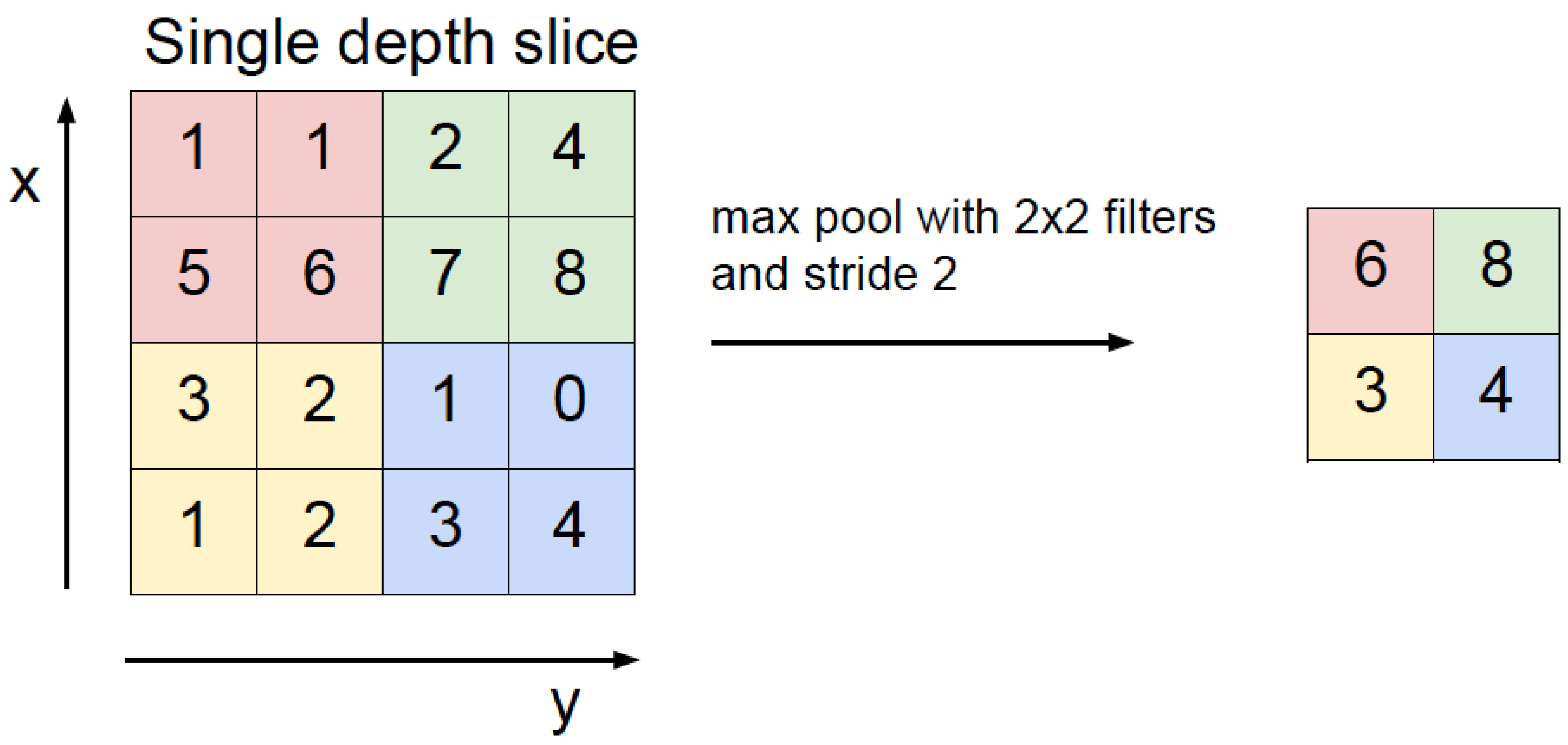
Pooling Layers

- Reducing the spatial size of the feature maps
  - Smaller representations
  - On each activation map independently
  - Low resolution means fewer details



Pooling Layers

- Example: max pooling
- Spatial invariance; no learnable parameters!



Complexity of Pooling Layers

- Summary

  - Accepts a volume of size $W_1 \times H_1 \times D_1$
  - Requires three hyperparameters:
    - their spatial extent F,
    - the stride S,
  - Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F)/S + 1$
    - $H_2 = (H_1 - F)/S + 1$
    - $D_2 = D_1$
  - Introduces zero parameters since it computes a fixed function of the input
  - Note that it is not common to use zero-padding for Pooling layers

Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

Math Properties of CNNs

- What representations a CNN can capture in general?
- Consider a representation ϕ as an abstract function

$$\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^d$$

- We want to look at how the representation changes upon transformations of input image.
  - Transformations represent the potential variations in the natural images
  - Translation, scale change, rotation, local deformation etc.

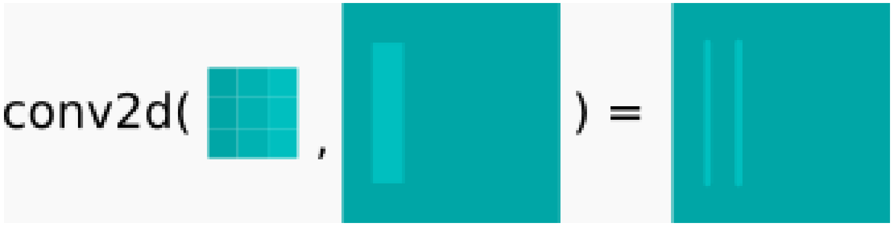
Math Properties of CNNs

- Two key properties of representations
  - Equivariance

    A representation ϕ is equivariant with a transformation g if the transformation can be transferred to the representation output.

$$\exists \text{ a map } M_g : \mathbb{R}^d \to \mathbb{R}^d \text{ such that:}$$
$$\forall \mathbf{x} \in \mathcal{X} : \phi(g\mathbf{x}) \approx M_g \phi(\mathbf{x})$$

  - Example: convolution w.r.t. translation



Math Properties of CNNs

- Two key properties of representations
  - Invariance

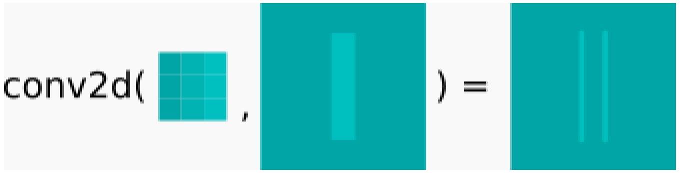
    A representation ϕ is invariant with a transformation g if the transformation has no effect on the representation output.

$$\forall \mathbf{x} \in \mathcal{X} : \phi(g\mathbf{x}) \approx \phi(\mathbf{x})$$

  - Example: convolution+pooling+FC w.r.t. translation

Math Properties of CNNs

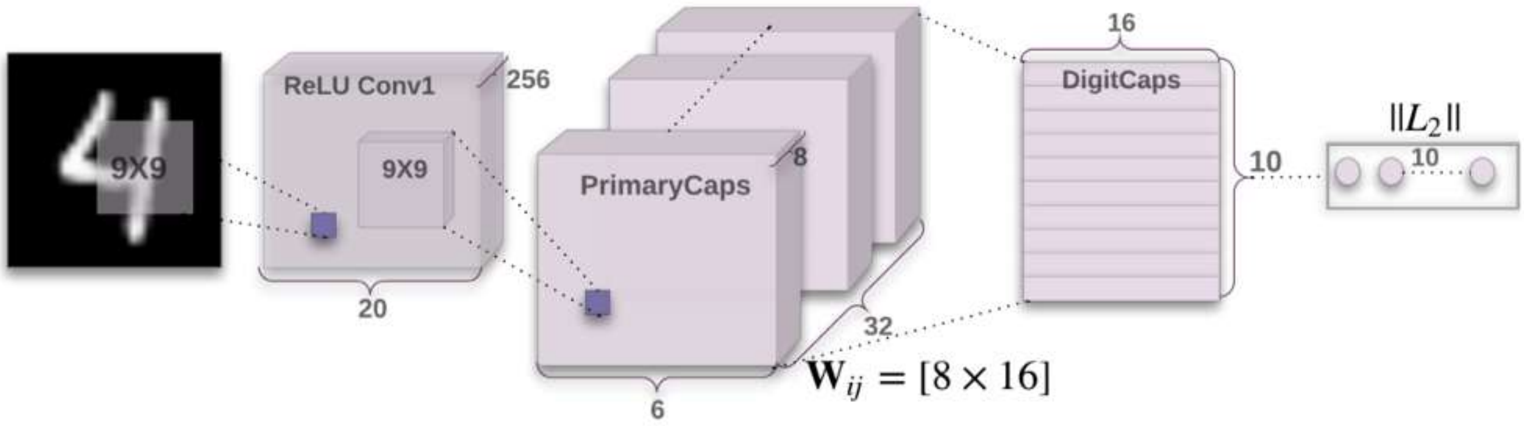
- Recent results on convolution layers
  - Convolutions are equivariant to translation
  - Convolutions are not equivariant to other isometries of the sampling lattice, e.g., rotation



  - What if a CNN learns rotated copies of the same filter?
    - The stack of feature maps is equivariant to rotation.

Math Properties of CNNs

- Recent results on convolution layers
  - Ordinary CNNs can be generalized to Group Equivariant Networks (Cohen and Welling ICML'16, Kondor and Trivedi ICML'18)
    - Redefining the convolution and pooling operations
    - Equivariant to more general transformation from some group *G*
  - Replacing pooling by other network designs
    - Capsule network (Sabour et al, 2017) https://arxiv.org/abs/1710.09829



Outline

- Why Convolutional Neural Network (CNN)?
  - Motivation and overview
- What is the CNN?
  - Convolution layers & model complexity
  - Closer look at activation functions
  - Pooling layers & model complexity
  - Math properties
- Examples of CNNs

Acknowledgement: Roger Grosse@UofT & Feifei Li's cs231n notes

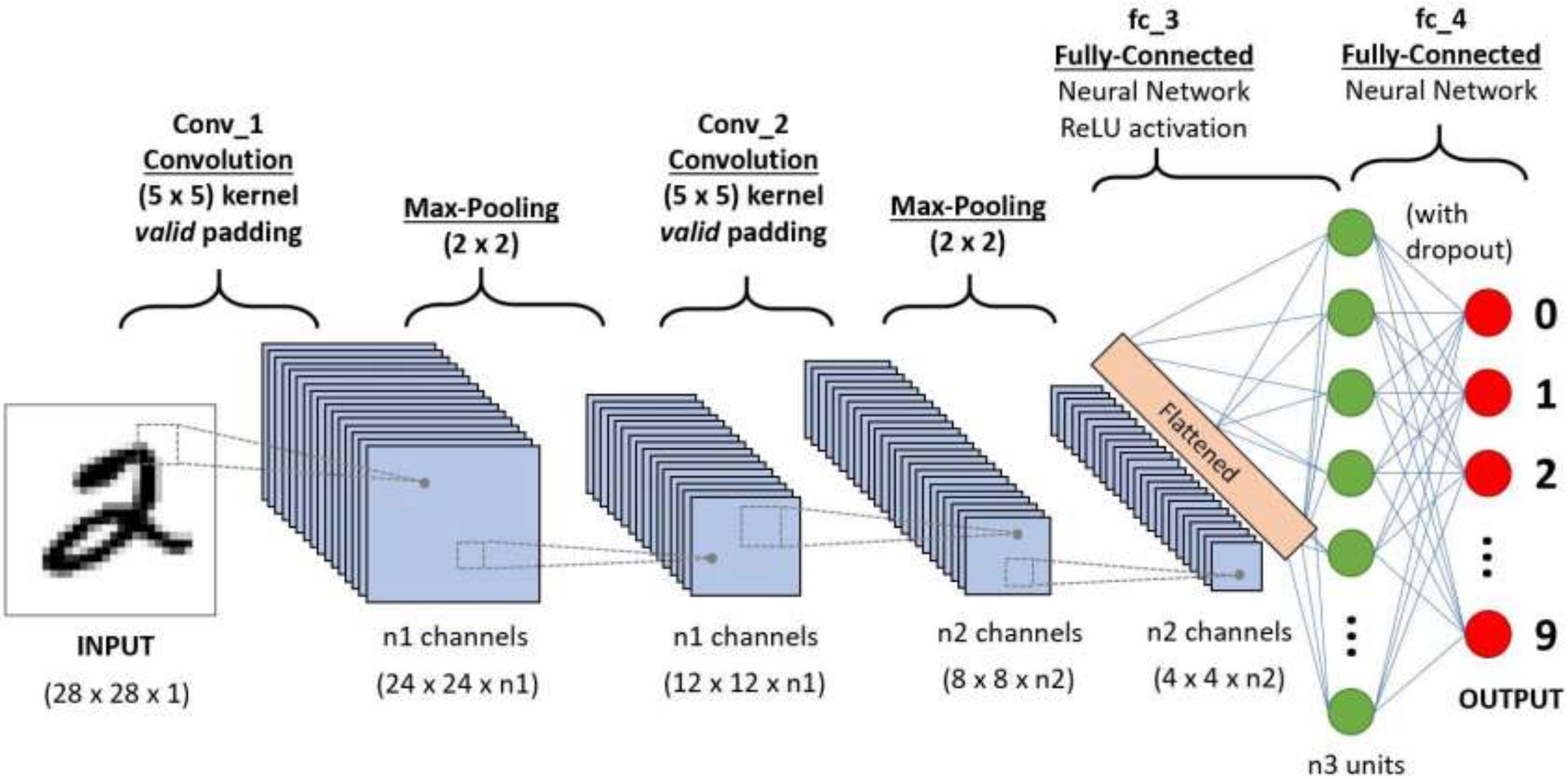
Outline

- CNN architectures
  - Sequential structure: LeNet/AlexNet/VGGNet
  - Parallel branches: GoogLeNet
  - Residual structure: ResNet/DenseNet
  - Network Architecture Search

Acknowledgement: Zemel et al’s CSC411 and Feifei Li et al’s cs231n notes

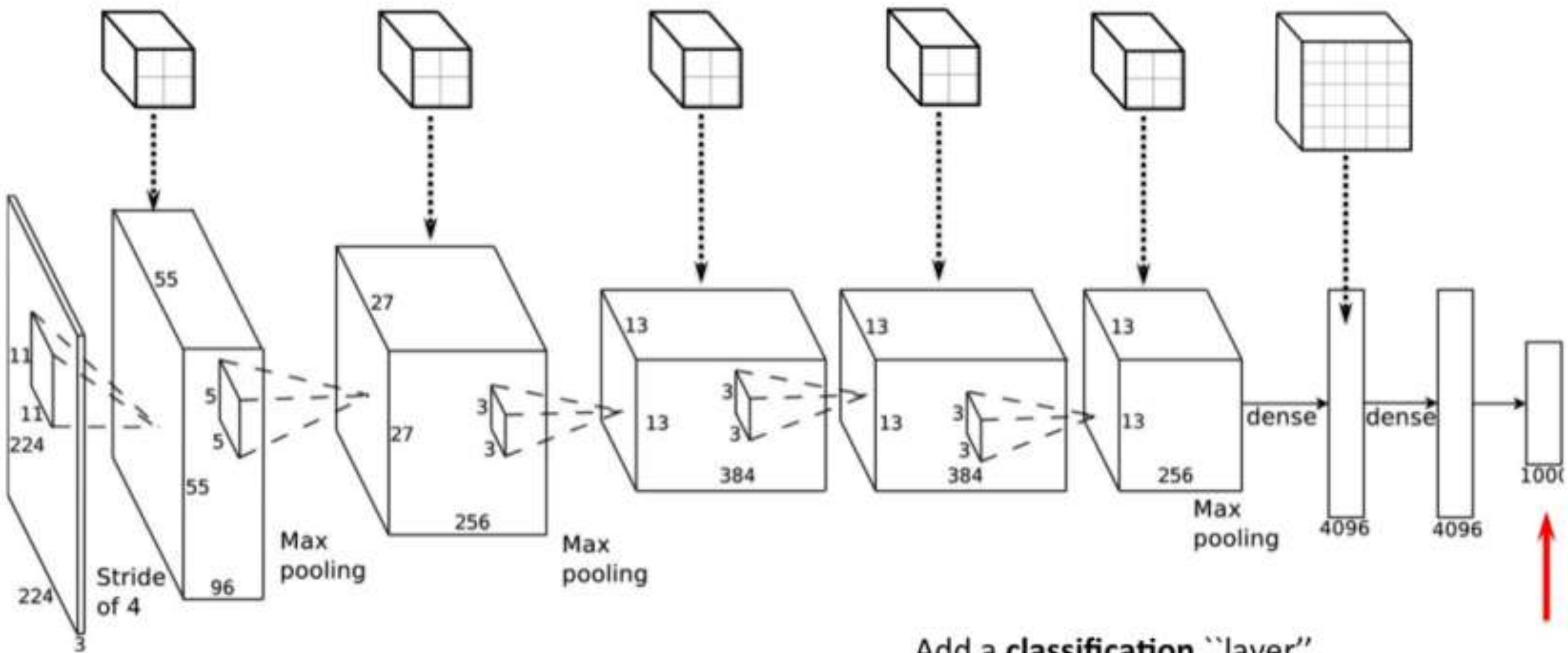
LeNet-5

- Handwritten digit recognition



AlexNet

- Deeper network structure

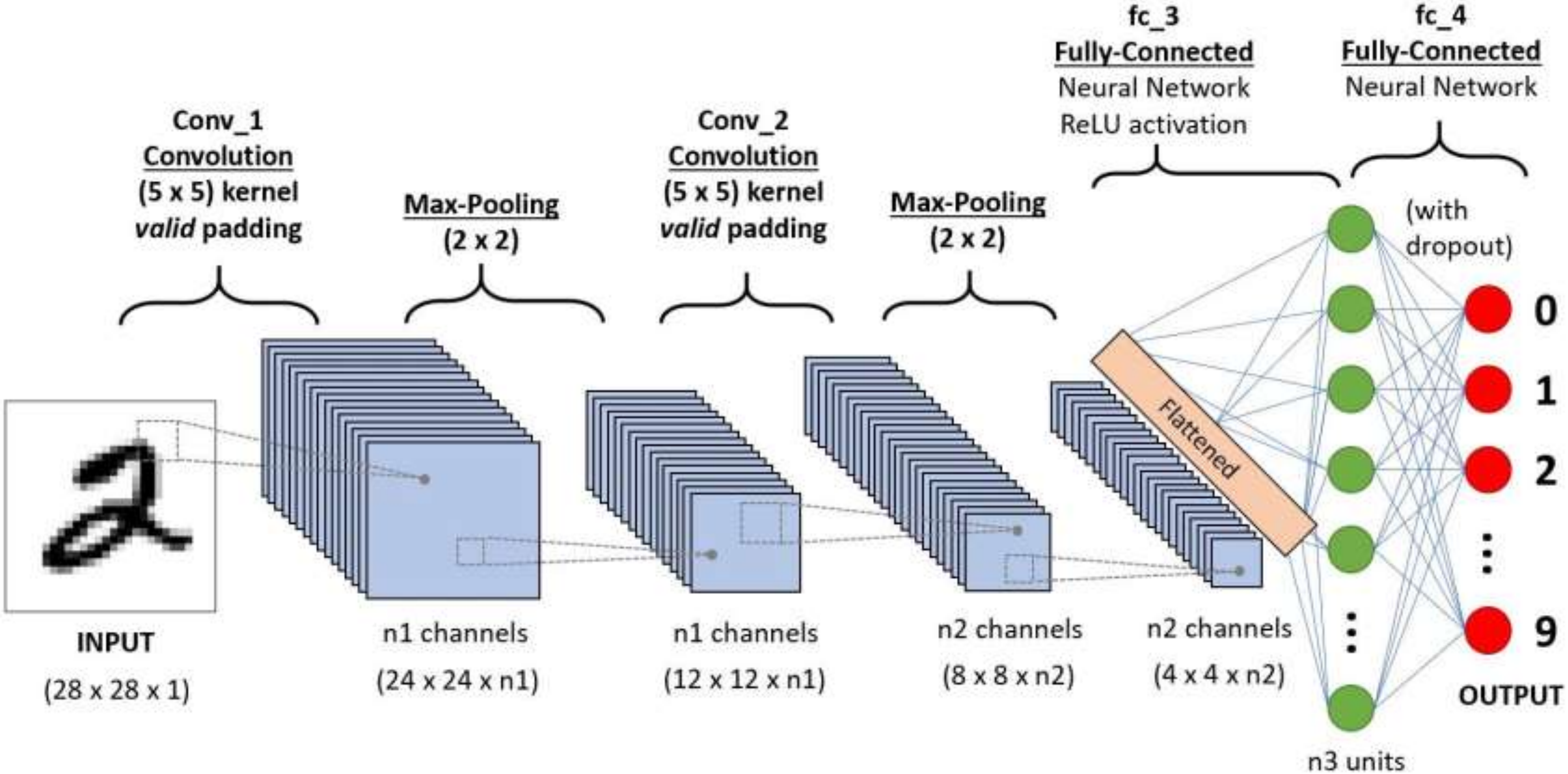


Add a **classification** ``layer''.

For an input image, the value in a particular dimension of this vector tells you the probability of the corresponding object class.

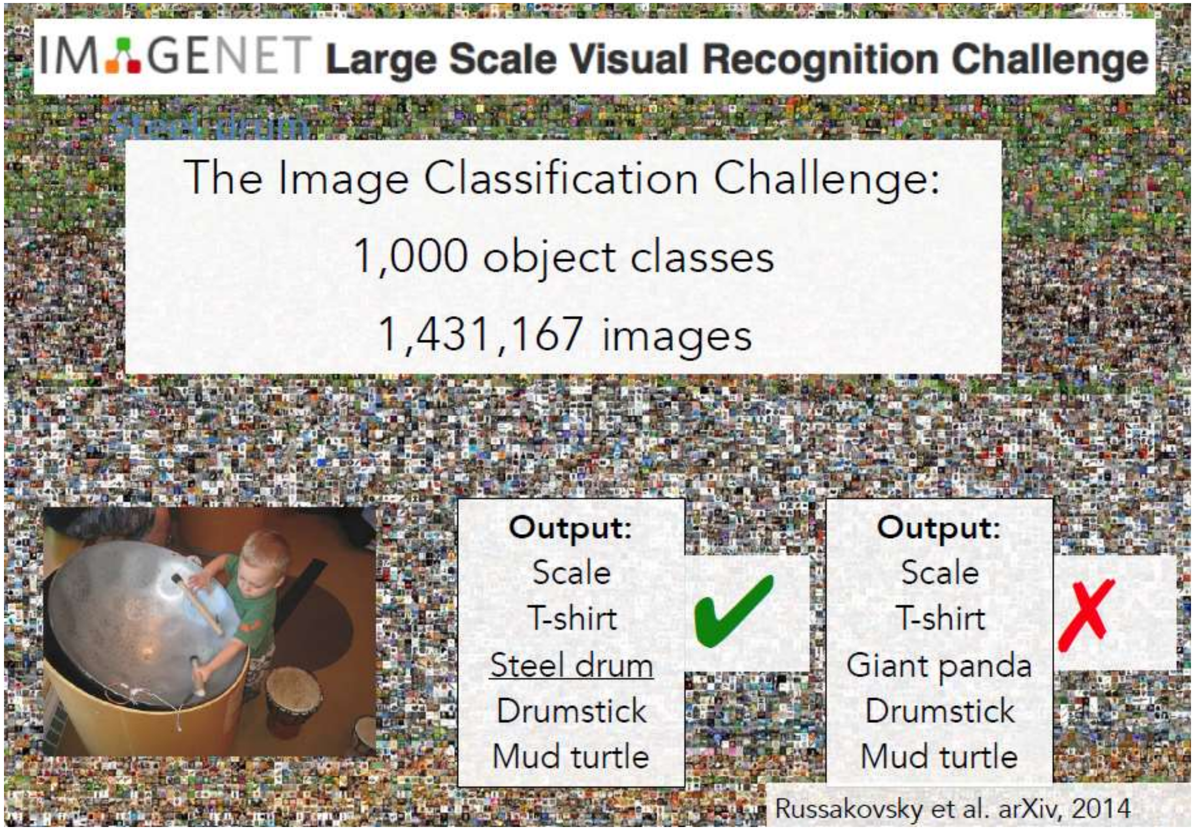
LeNet-5

- Handwritten digit recognition
- LeCun et al., 1998

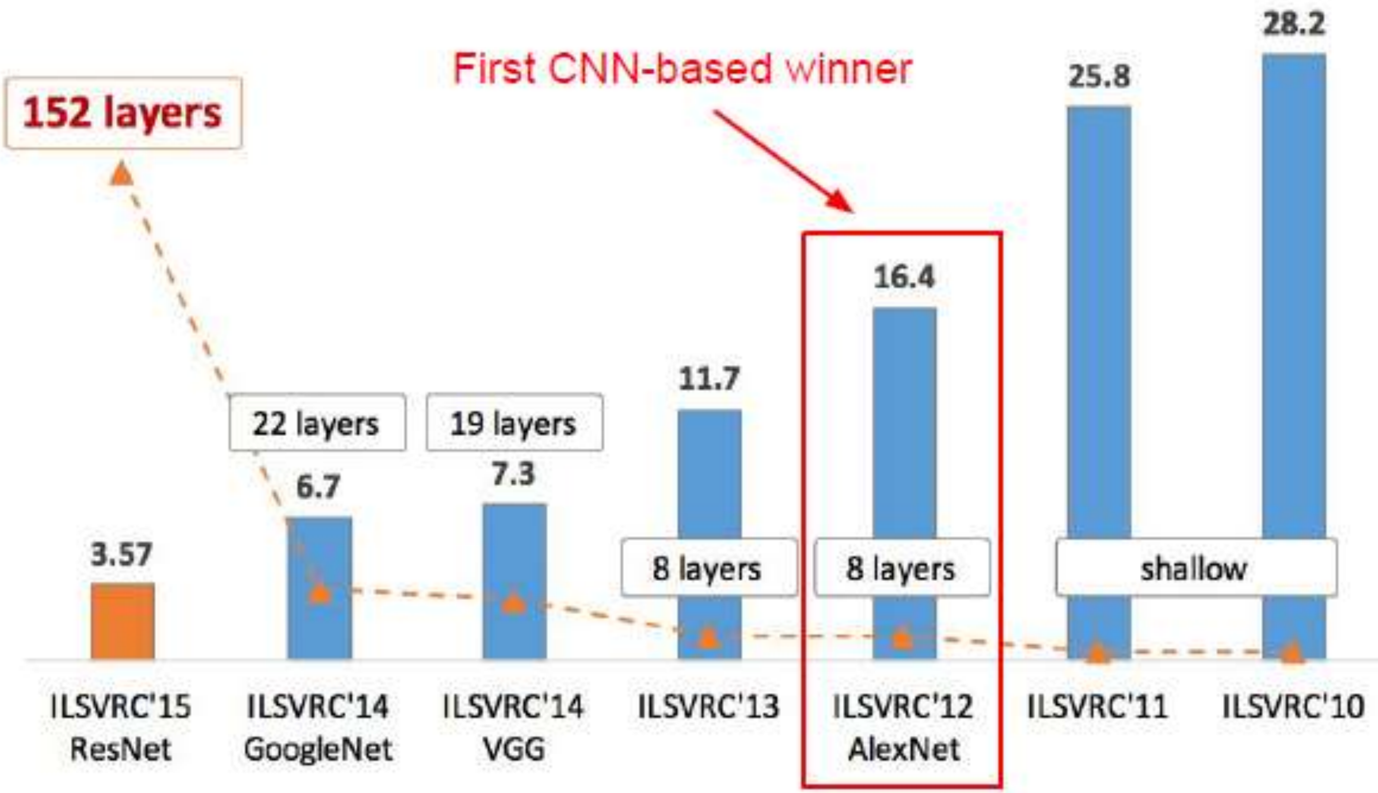


Background: Image/Object Classification

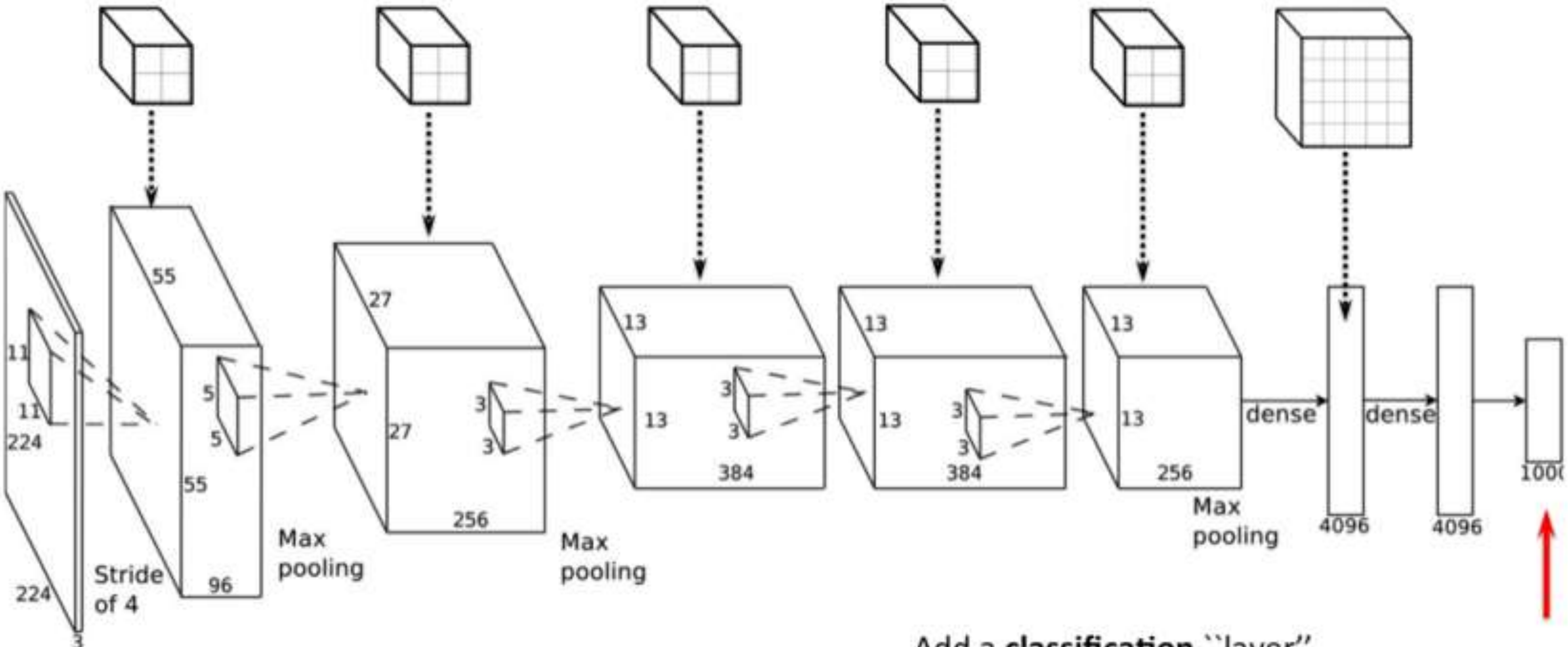
- Problem Setup
  - Input: Image
  - Output: Object class



ImageNet (ILSVRC)



AlexNet



Add a **classification** ``layer''.

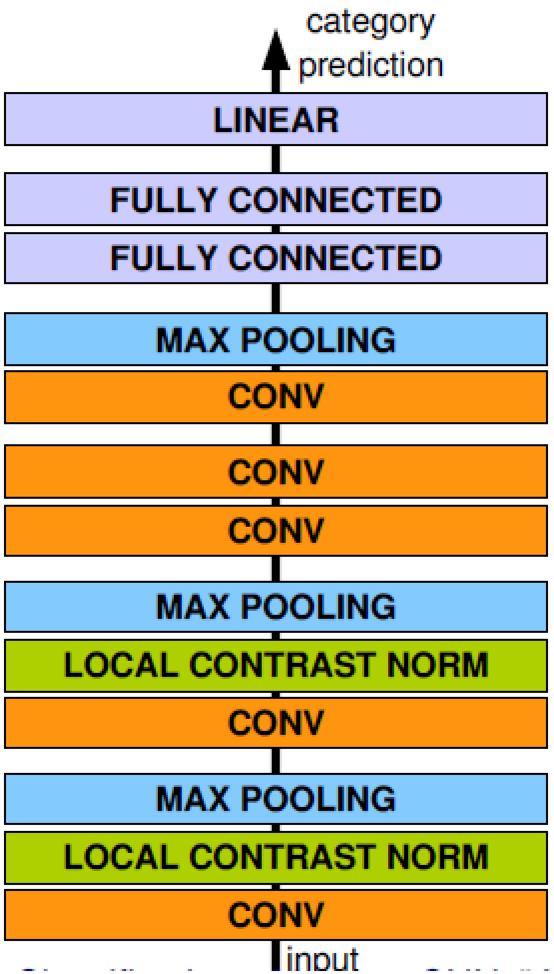
For an input image, the value in a particular dimension of this vector tells you the probability of the corresponding object class.

- 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%

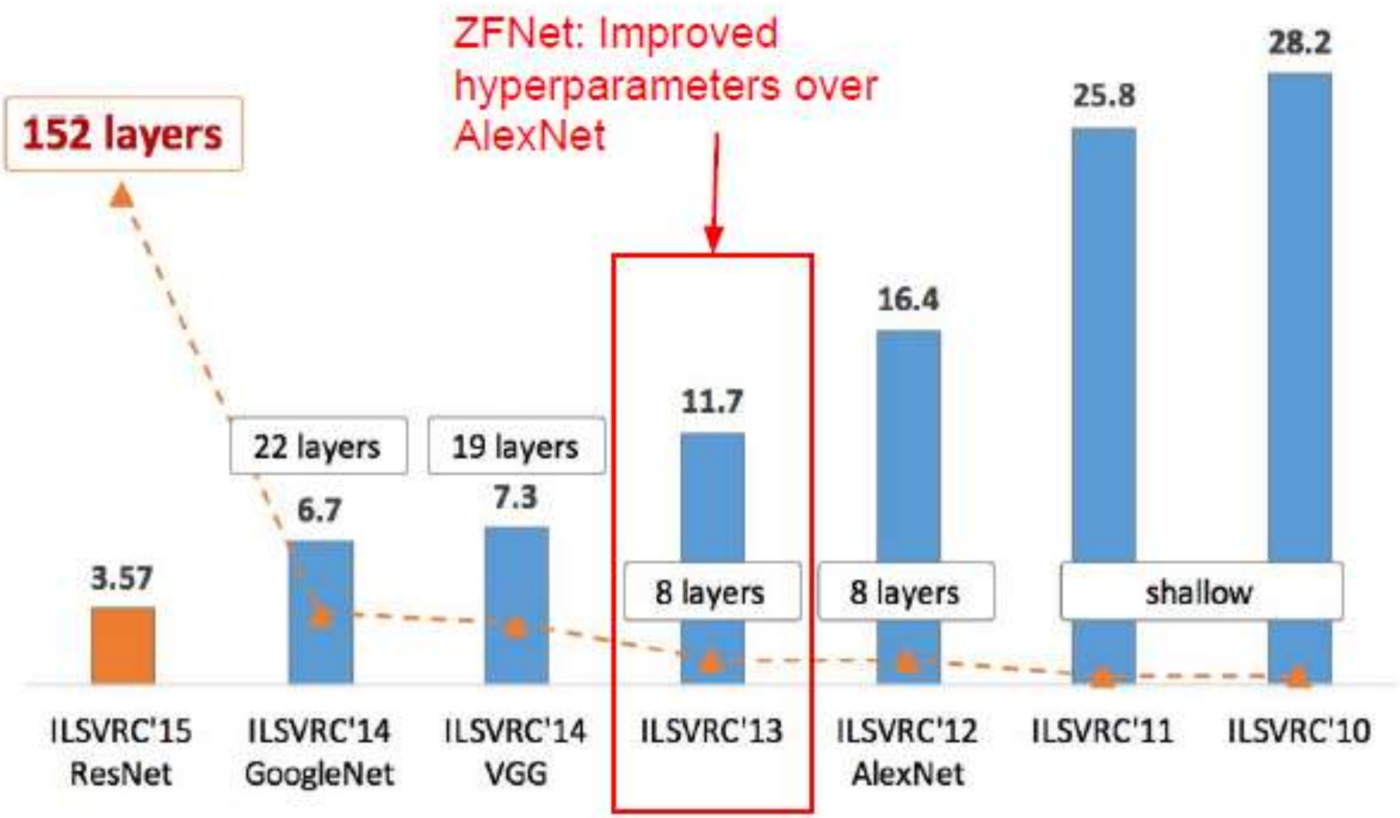
AlexNet

- Deeper network structure
  - More convolution layers
  - Local contrast normalization
  - ReLu instead of Tanh
  - Dropout as regularization

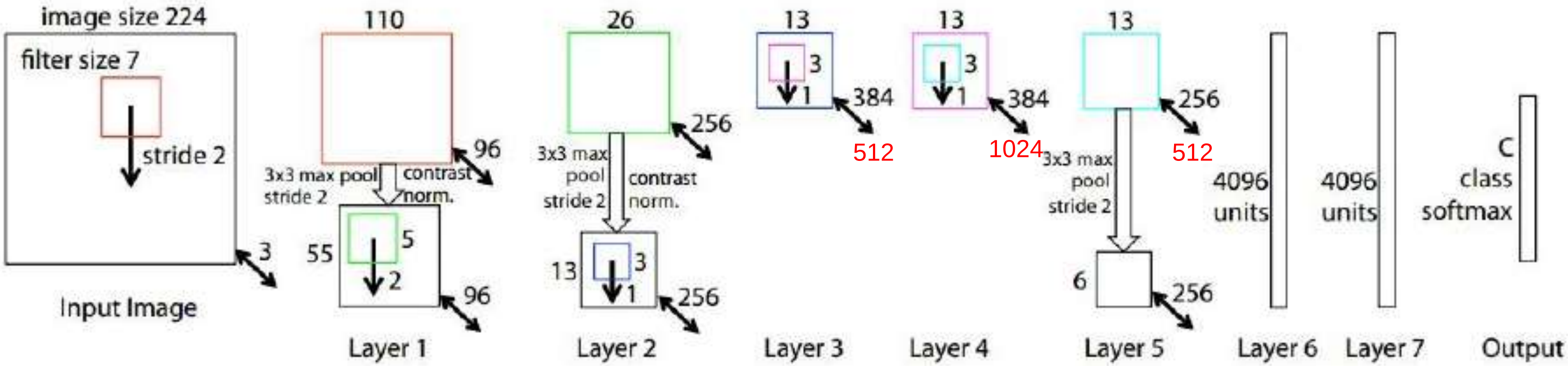
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] 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)



ImageNet (ILSVRC)



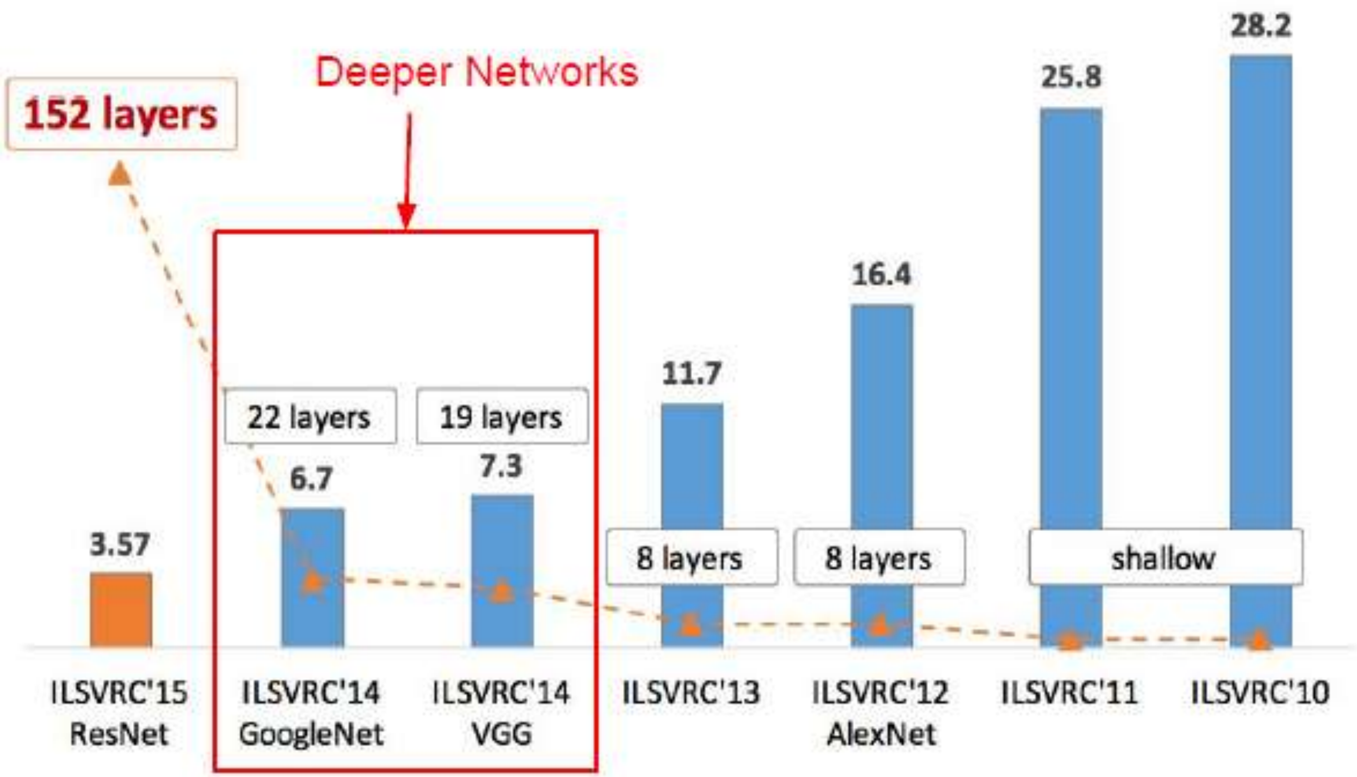
ZFNet



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

ImageNet top 5 error: 16.4% -> 11.7%

ImageNet (ILSVRC)



VGGNet

Case Study: VGGNet

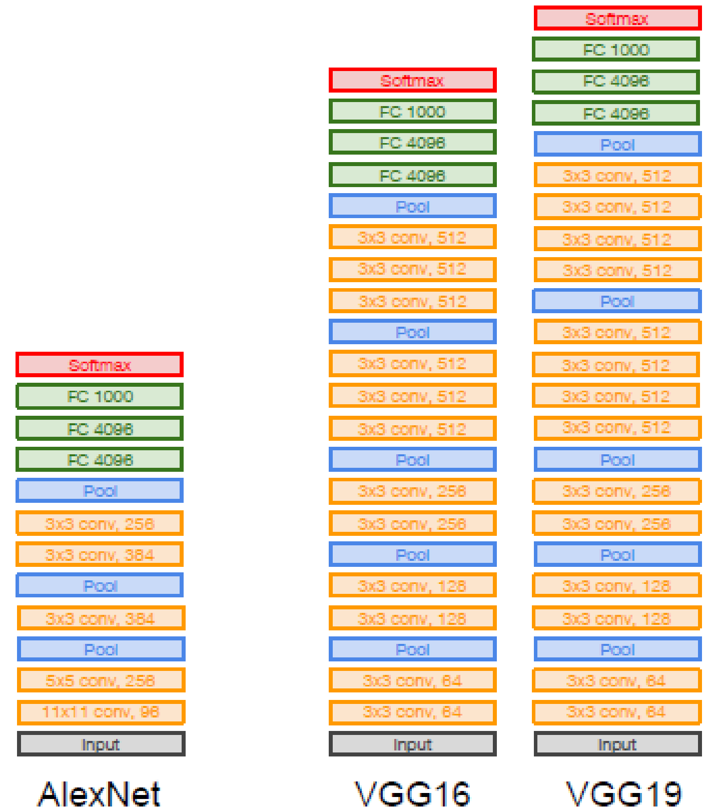
[Simonyan and Zisserman, 2014]

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



VGGNet

Case Study: VGGNet

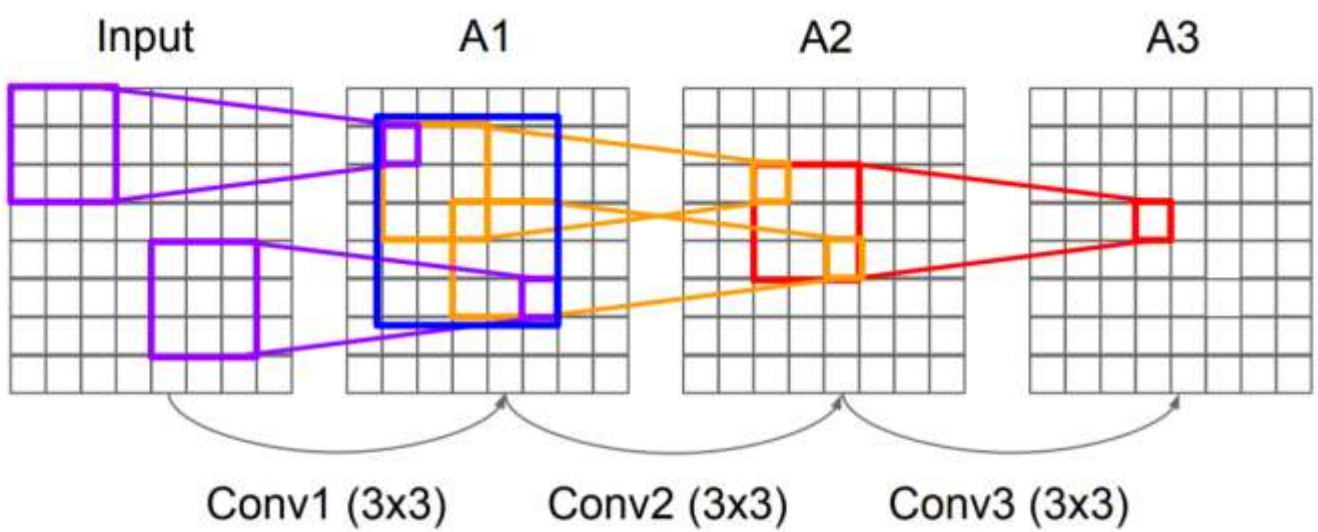
[Simonyan and Zisserman, 2014]

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

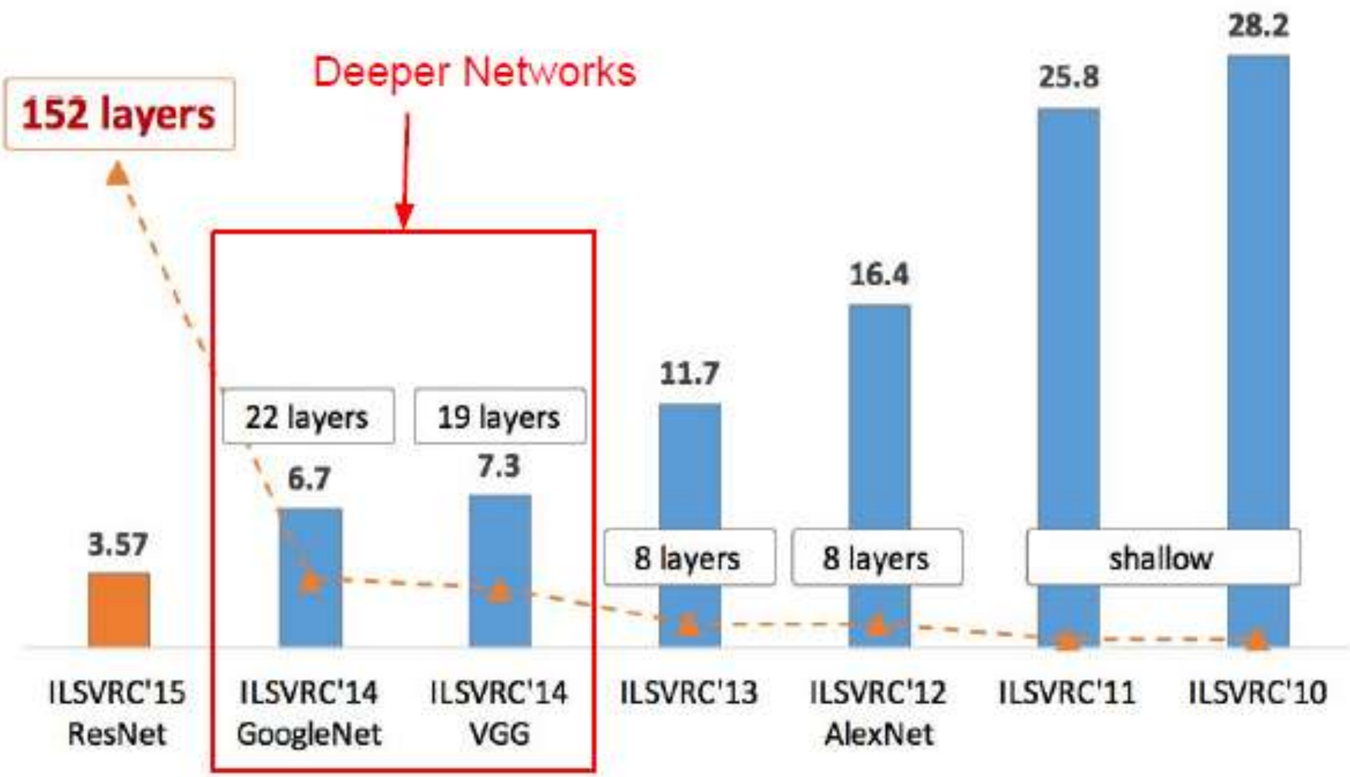


Outline

- CNN architectures
  - Sequential structure: LeNet/AlexNet/VGGNet
  - Parallel branches: GoogLeNet
  - Residual structure: ResNet/DenseNet
  - Network Architecture Search

Acknowledgement: Zemel et al’s CSC411 and Feifei Li et al’s cs231n notes

ImageNet (ILSVRC)



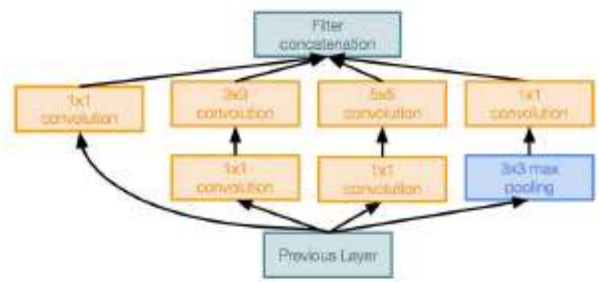
GoogLeNet

Case Study: GoogLeNet

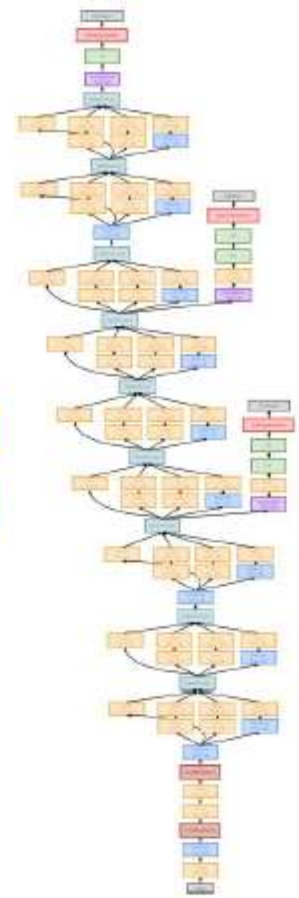
[Szegedy et al., 2014]

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

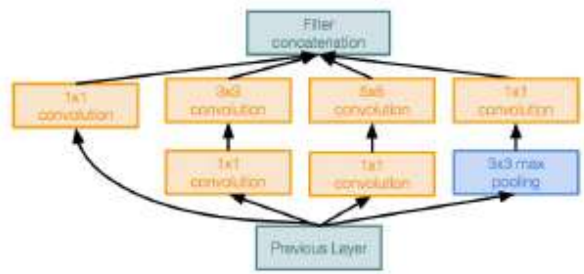


GoogLeNet

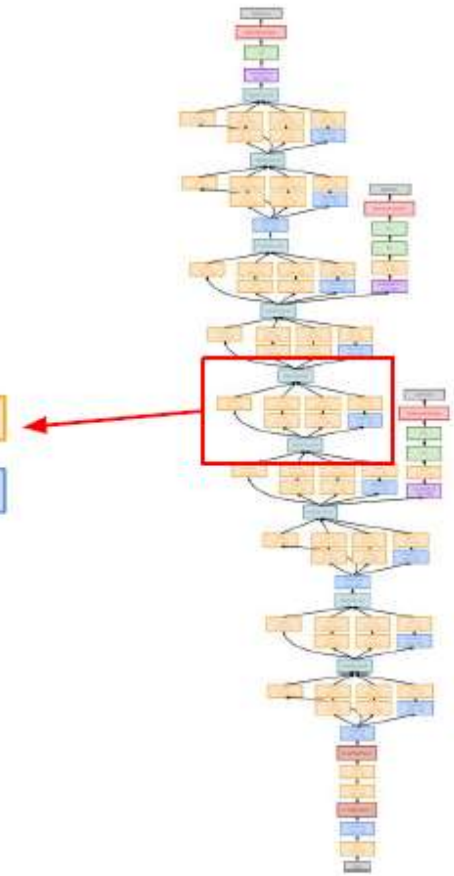
Case Study: GoogLeNet

[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

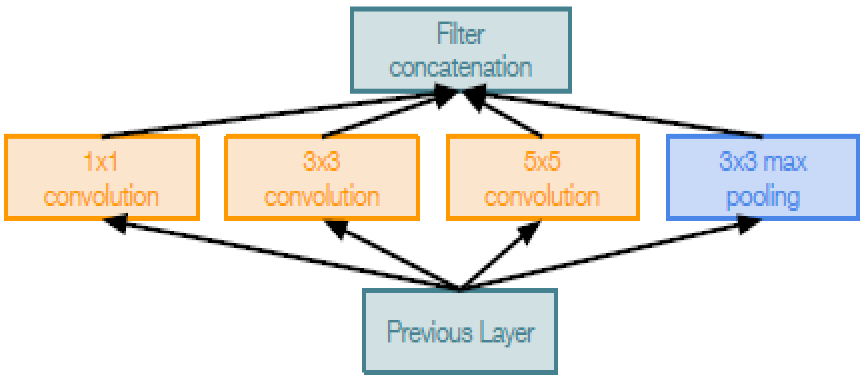


Inception module



GoogLeNet

- Inception Module



Naive Inception module

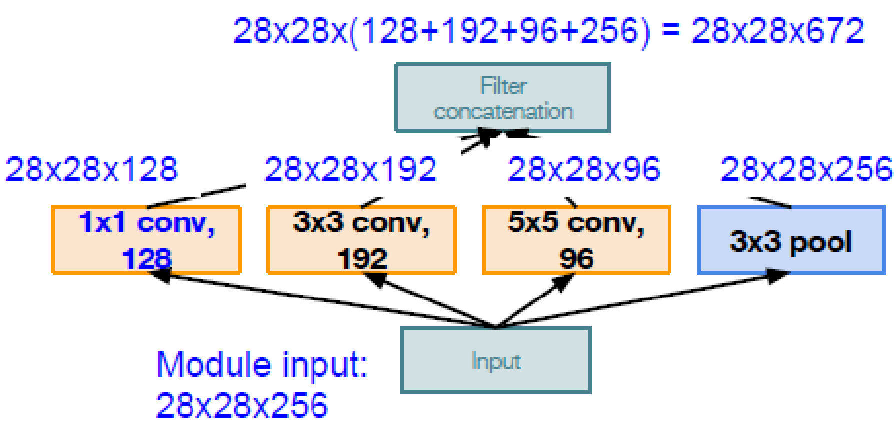
Apply parallel filter operations on the input from previous layer:

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

Concatenate all filter outputs together depth-wise

GoogLeNet

- Inception Module



Naive Inception module

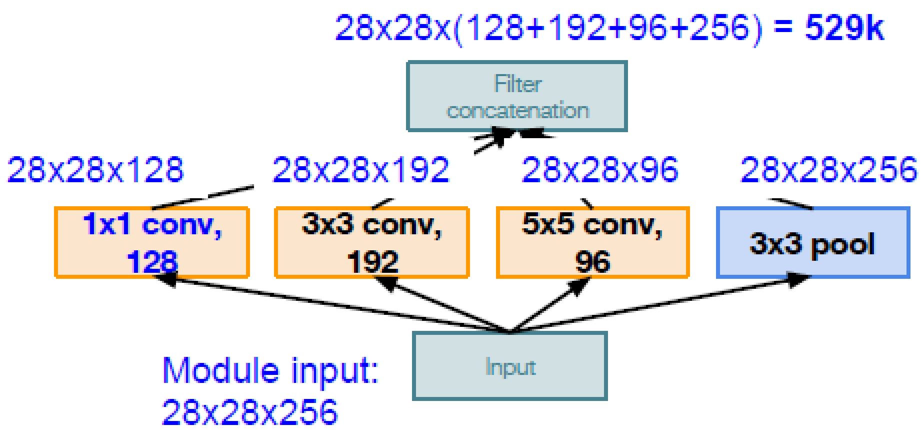
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

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

GoogLeNet

- Inception Module

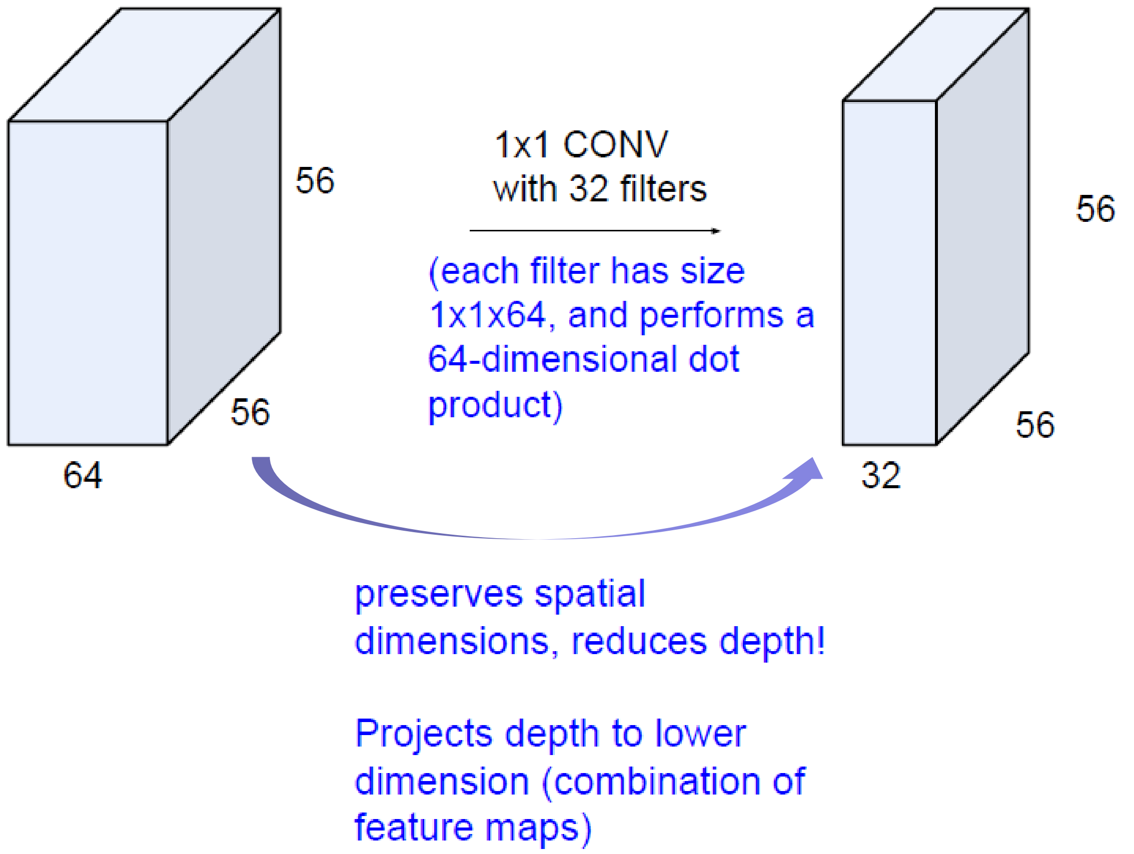


Naive Inception module

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

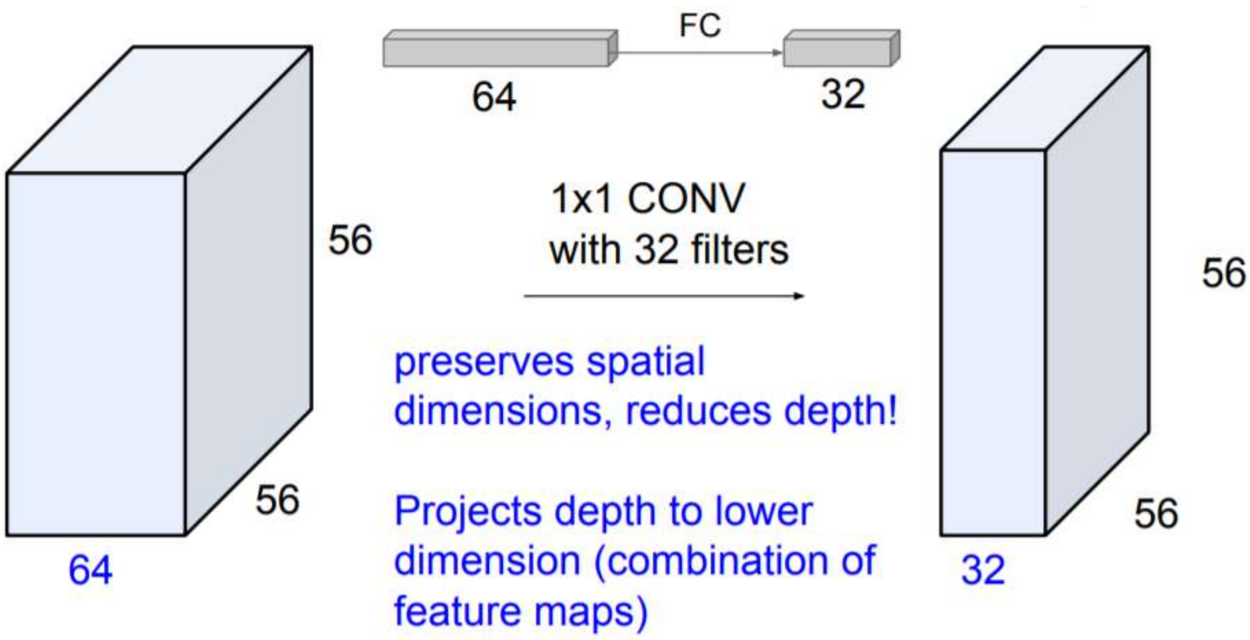
GoogLeNet

- Bottleneck layer



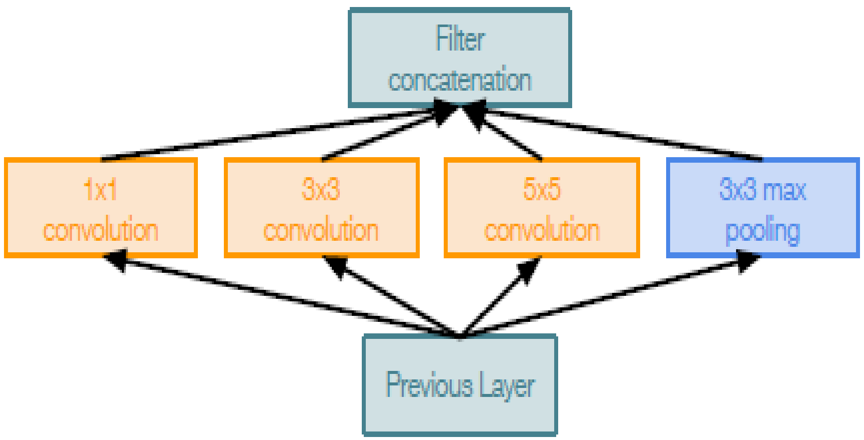
GoogLeNet

- 1x1 Convolutions
  - Alternatively, interpret it as applying the same FC layer on each input pixel



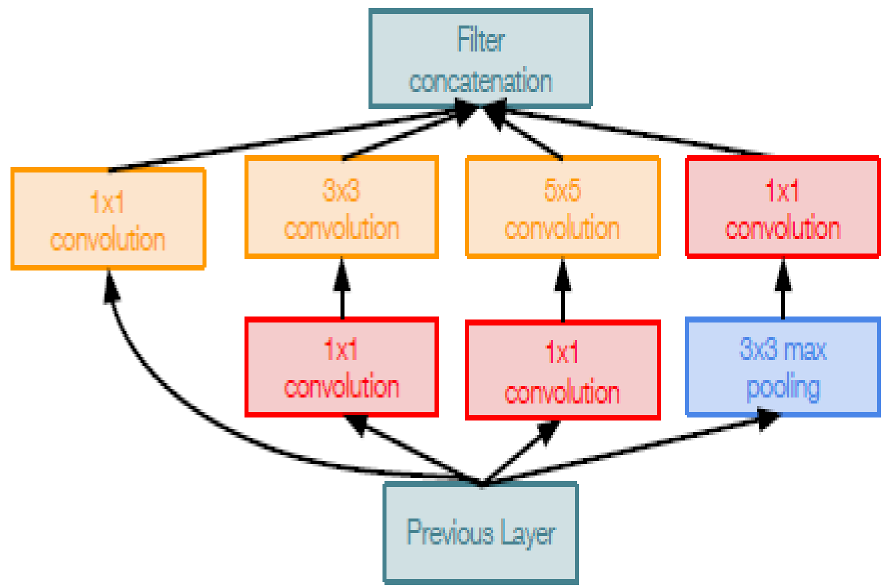
GoogLeNet

- Inception Module



Naive Inception module

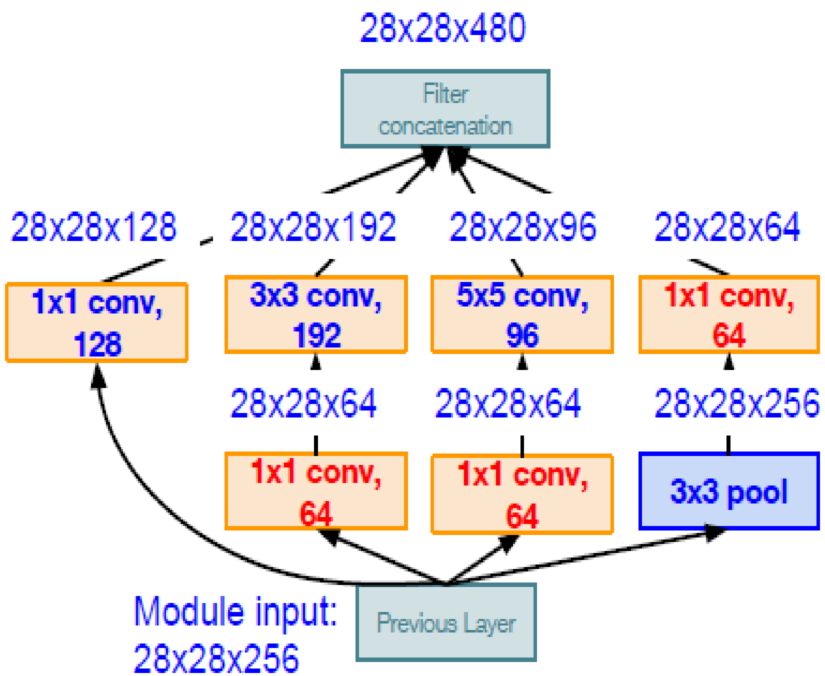
1x1 conv "bottleneck" layers



Inception module with dimension reduction

GoogLeNet

- Inception Module



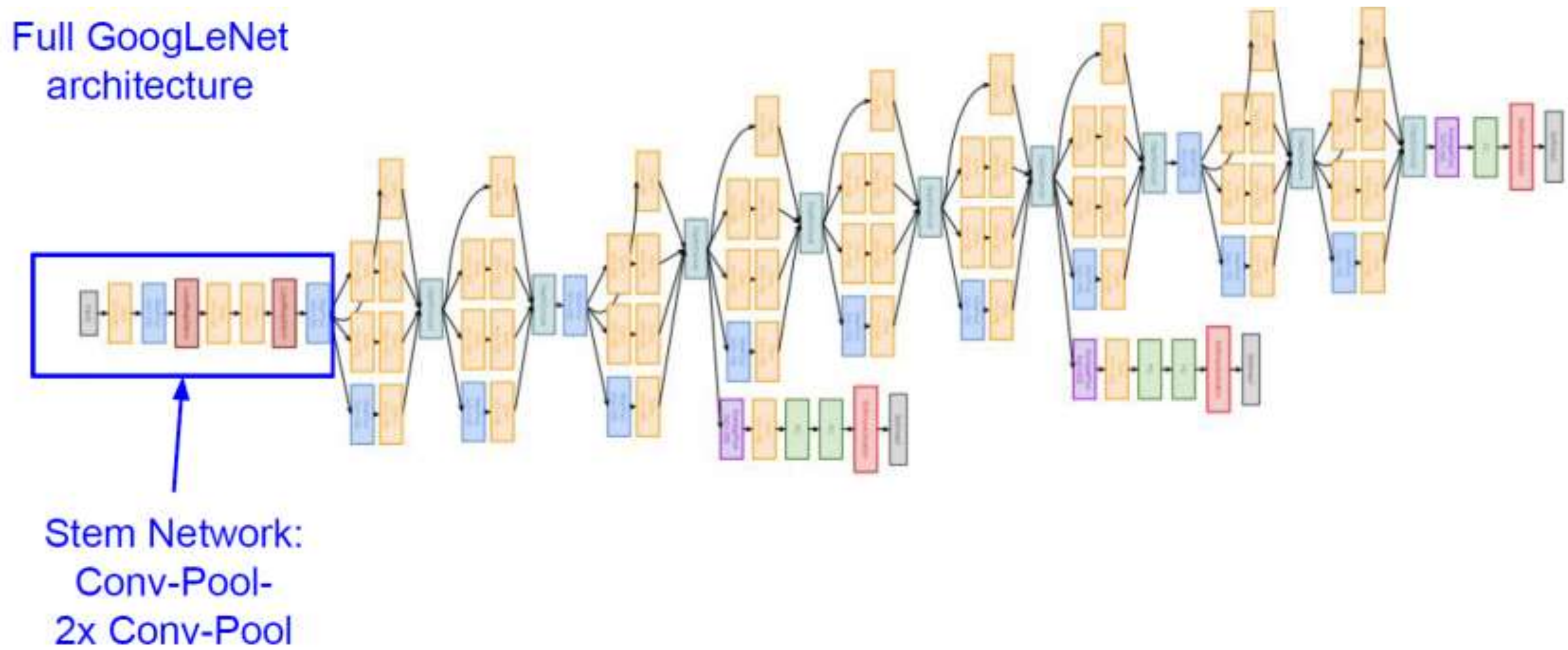
Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

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

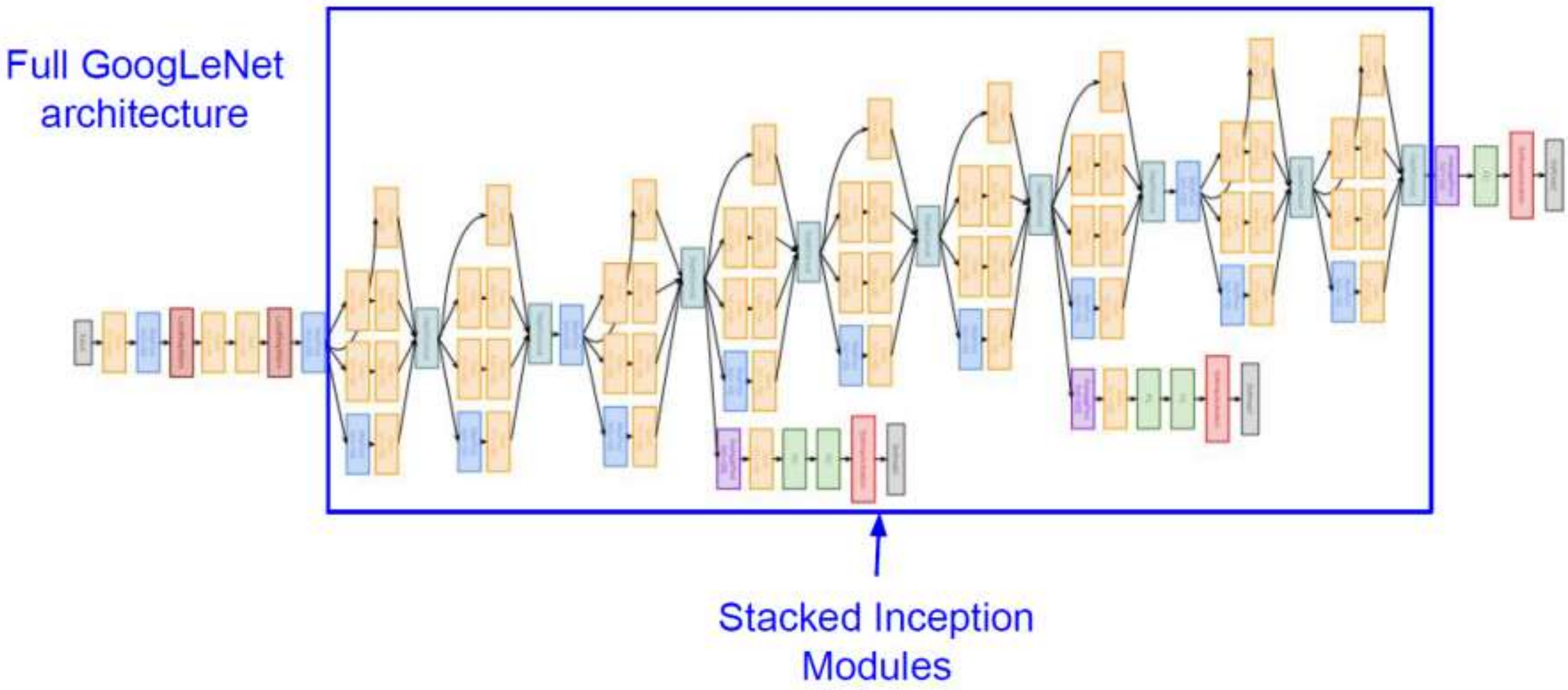
GoogLeNet

- Overall network structure



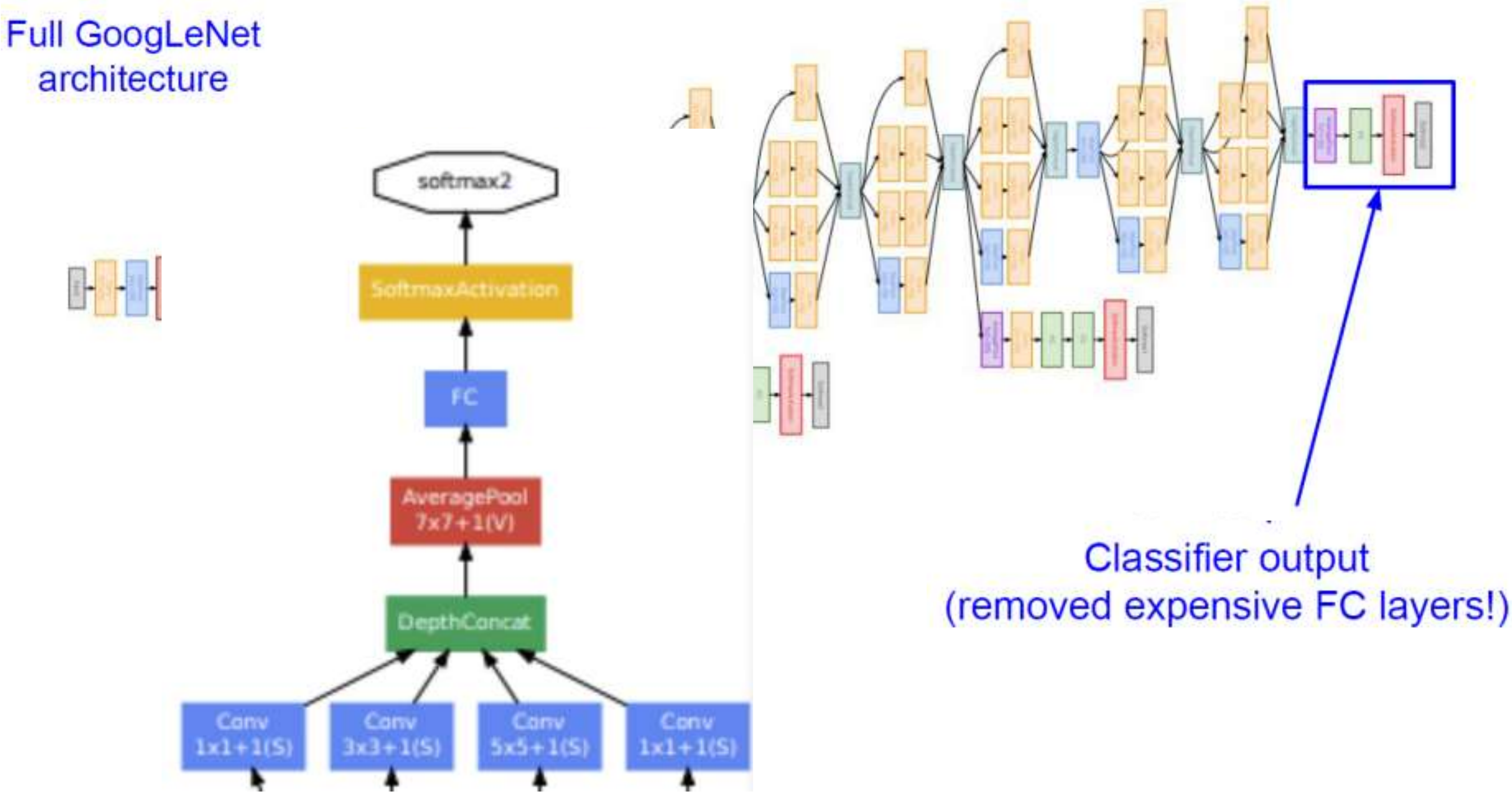
GoogLeNet

- Overall network structure



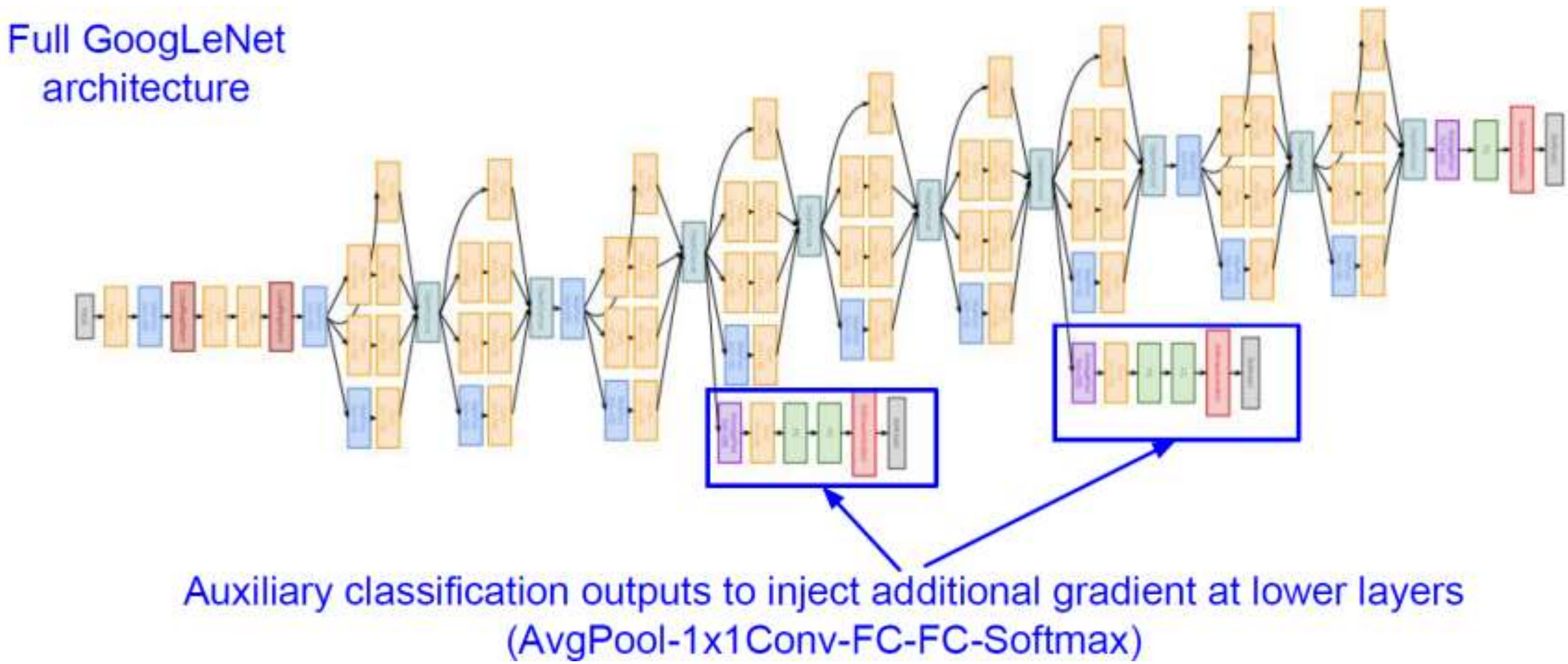
GoogLeNet

- Overall network structure



GoogLeNet

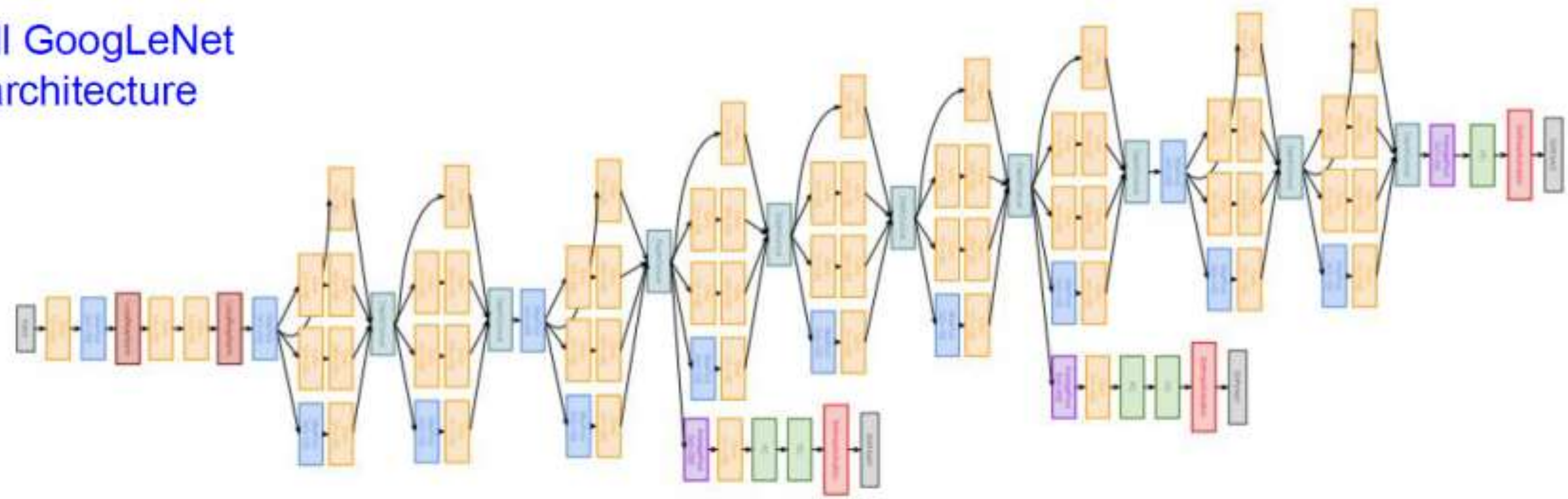
- Overall network structure



GoogLeNet

- Overall network structure

Full GoogLeNet architecture



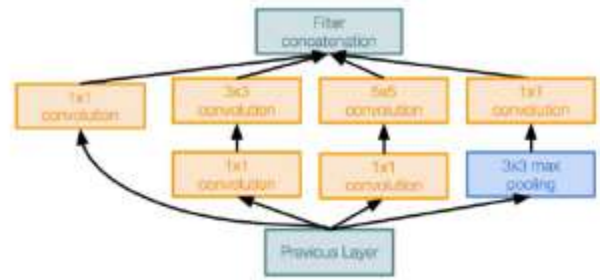
22 total layers with weights (including each parallel layer in an Inception module)

GoogLeNet

- Summary

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)



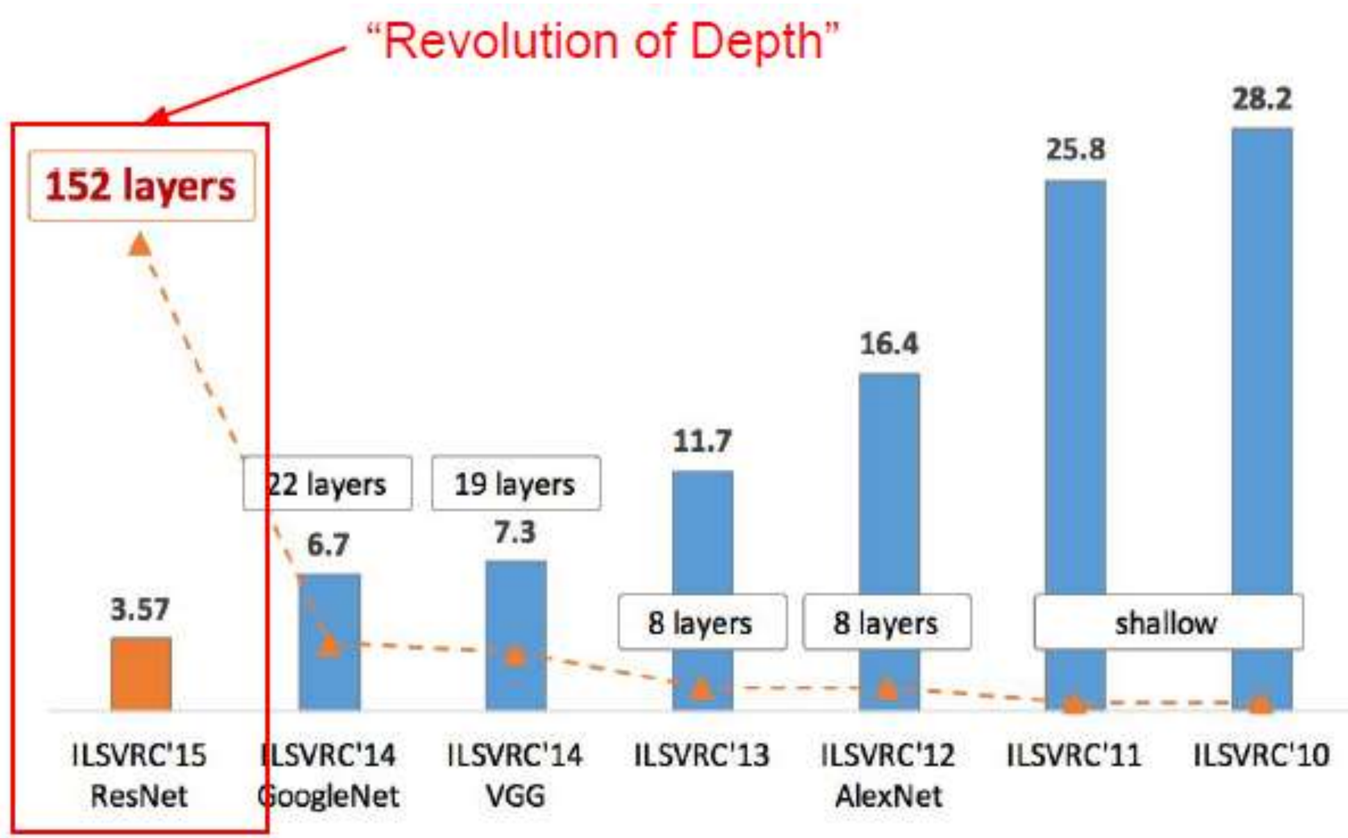
Inception module

Outline

- CNN architectures
  - Sequential structure: LeNet/AlexNet/VGGNet
  - Parallel branches: GoogLeNet
  - Residual structure: ResNet/DenseNet
  - Network Architecture Search

Acknowledgement: Zemel et al's CSC411 and Feifei Li et al's cs231n notes

ImageNet (ILSVRC)



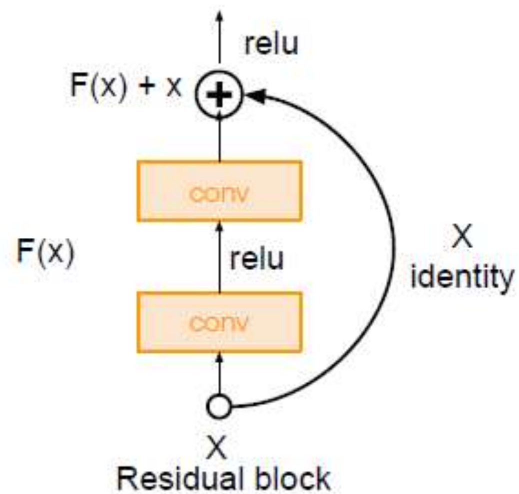
ResNet

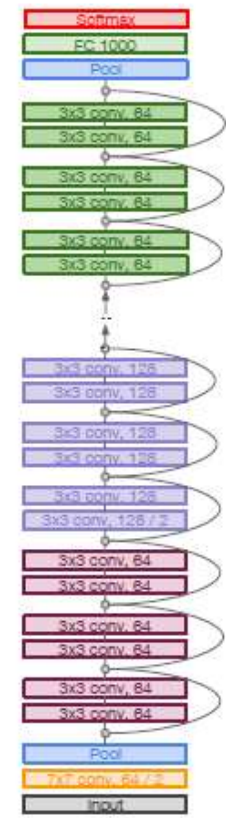
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

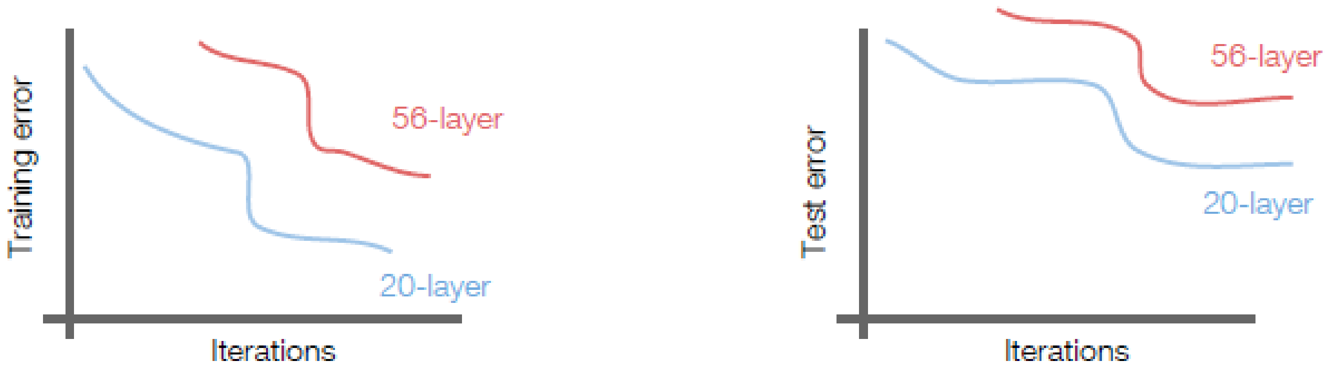
- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





ResNet

- What happens when stacking deeper plain conv layers?



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

ResNet

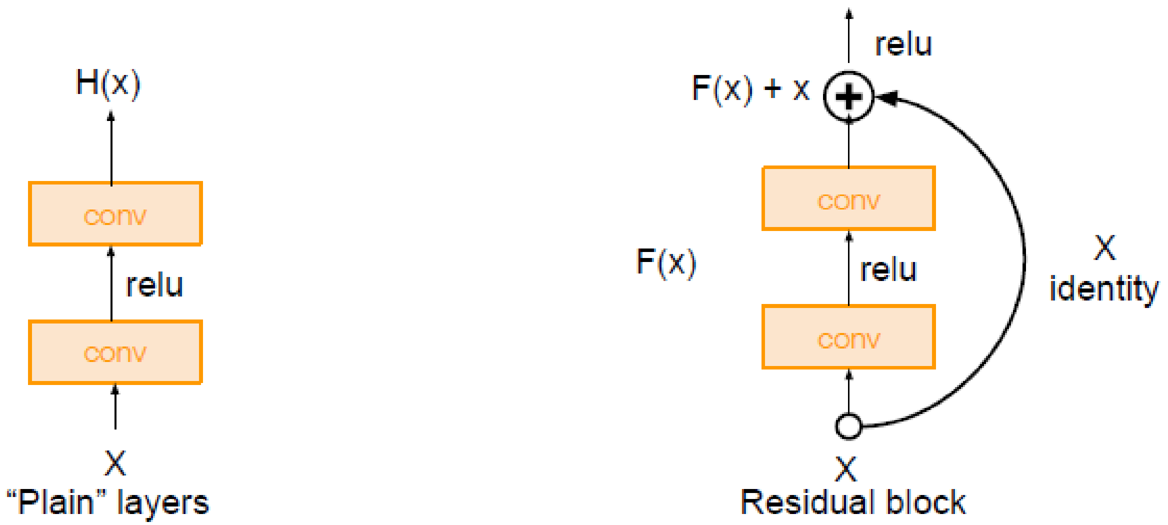
- Hypothesis:
  - The problem is an optimization problem, deeper models are harder to optimize

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

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

ResNet

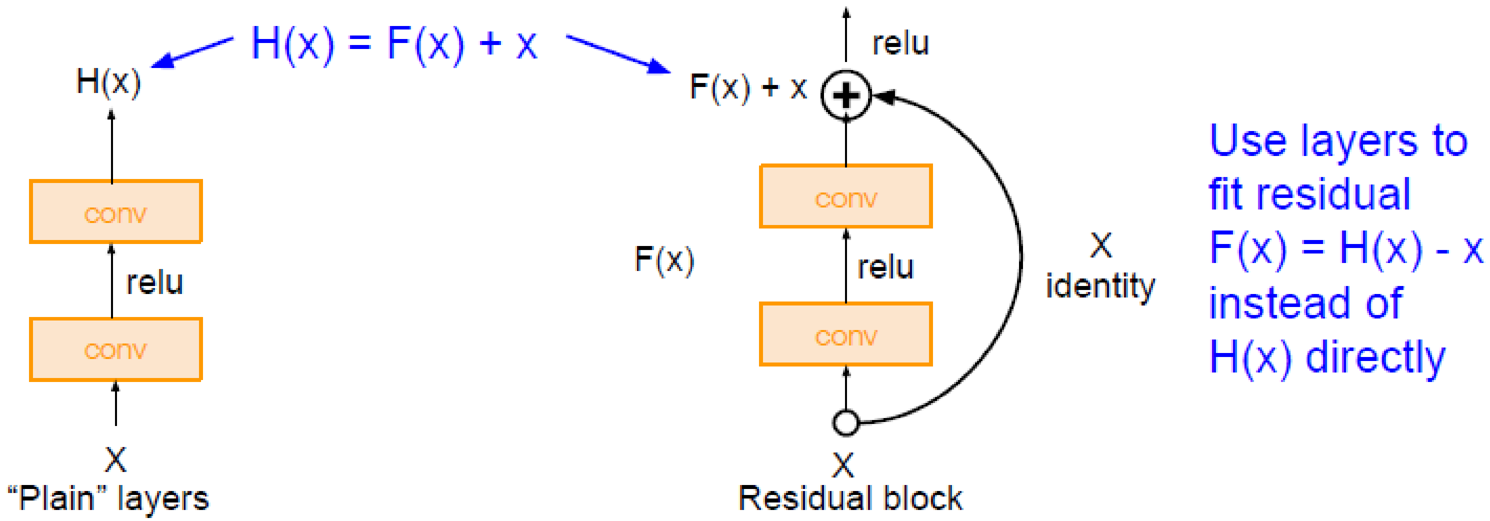
- Solution:
  - Use network layers to fit a residual mapping



He et al "Deep Residual Learning for Image Recognition", CVPR 2016

ResNet

- Solution:
  - Use network layers to fit a residual mapping

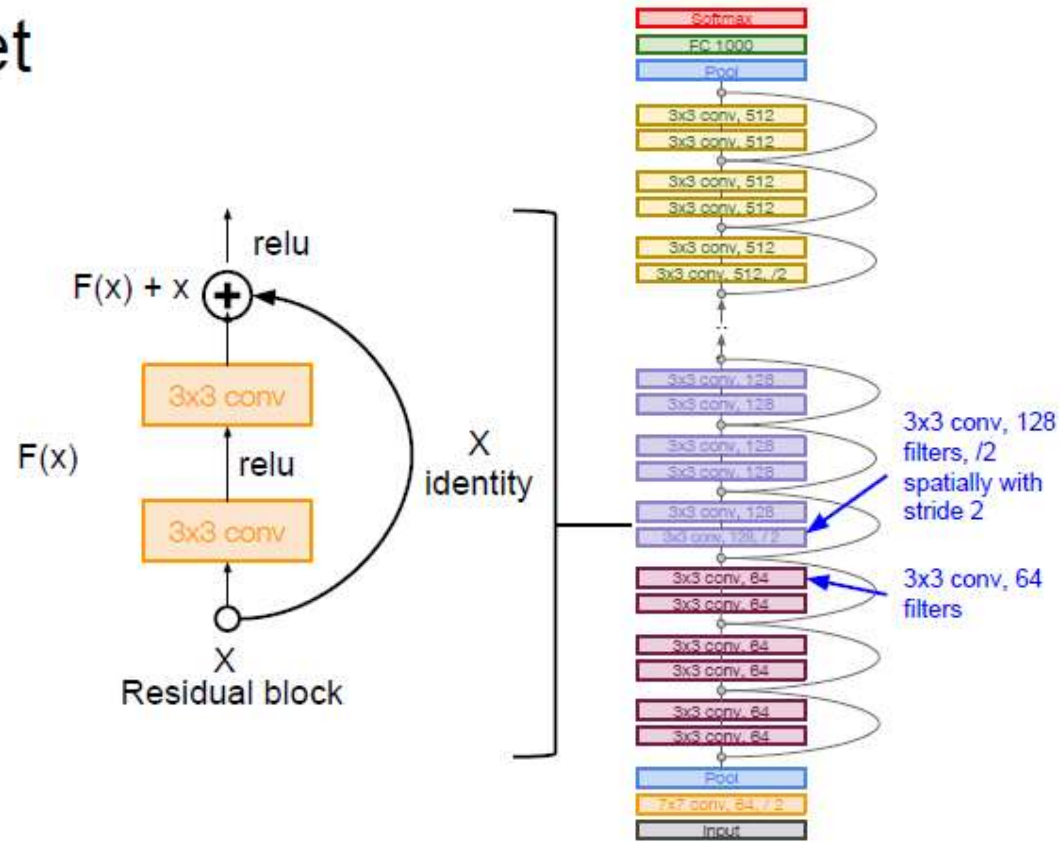


ResNet

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)

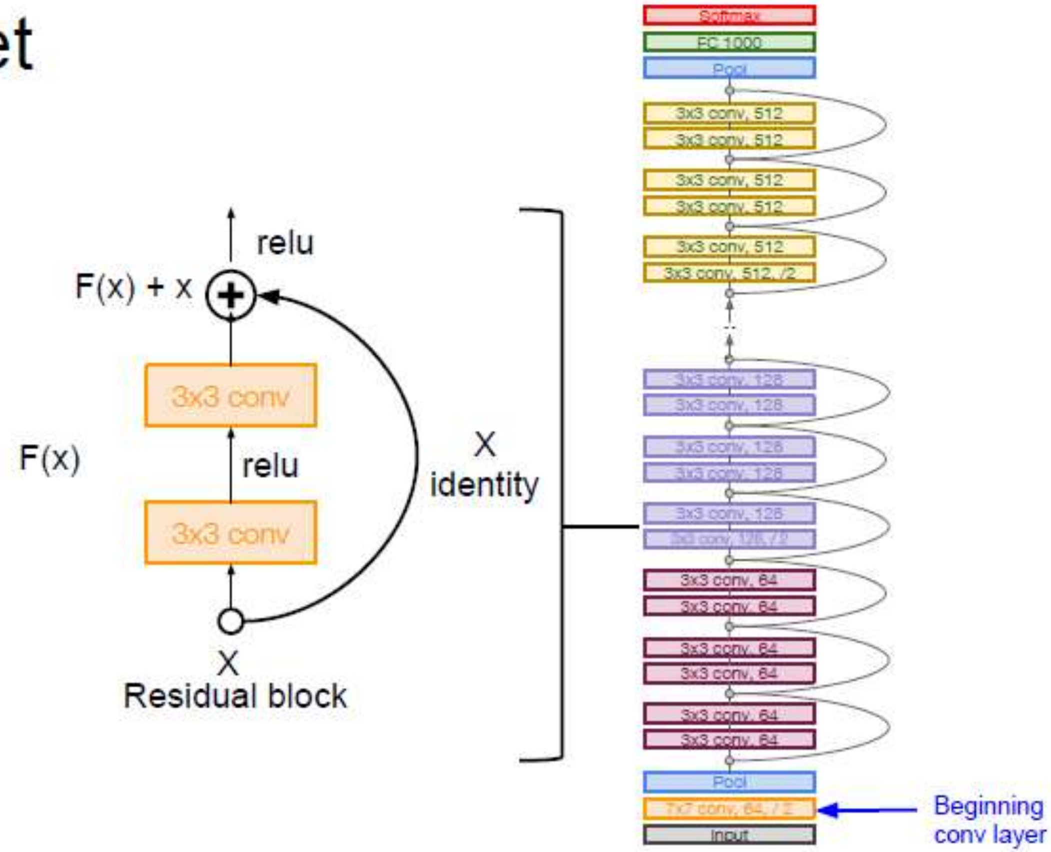


ResNet

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



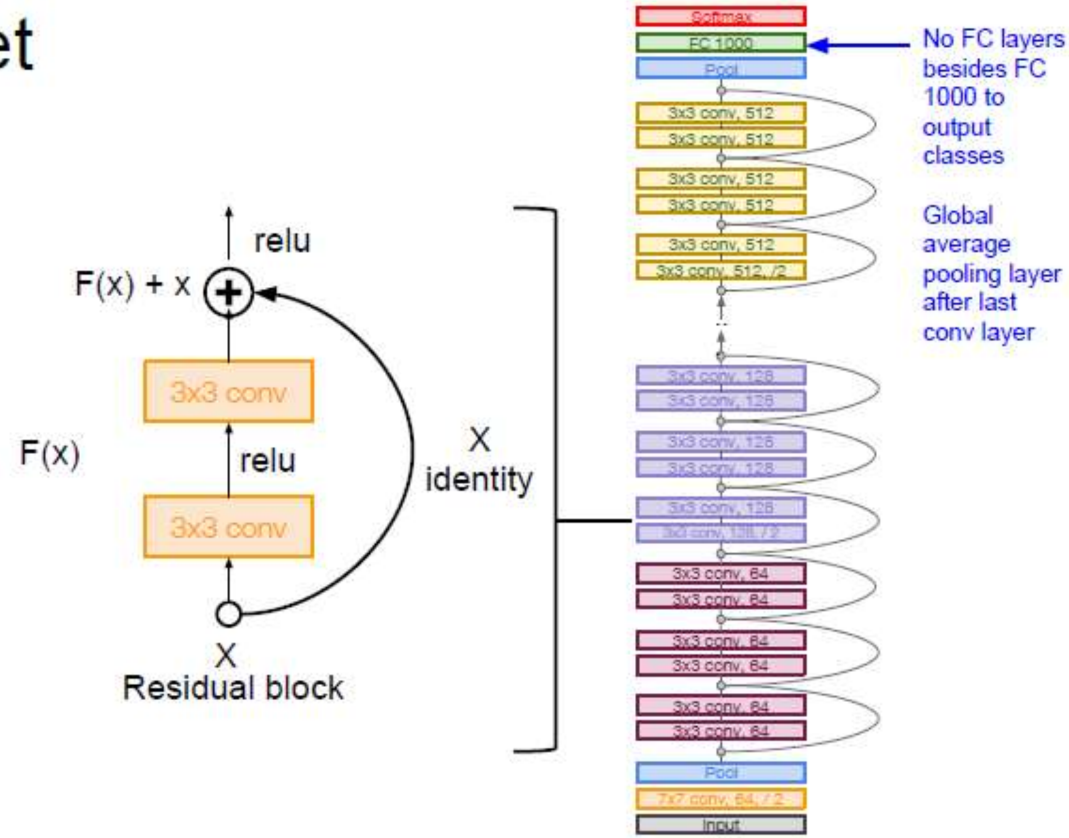
ResNet

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)

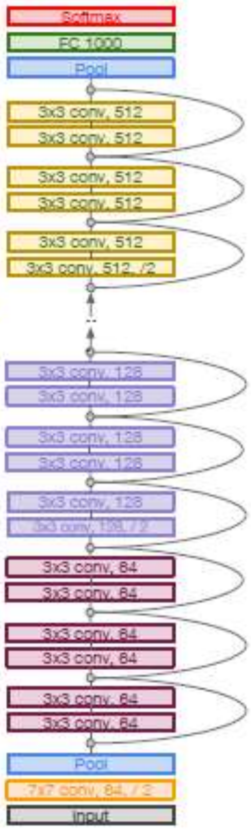


ResNet

Case Study: ResNet

[He et al., 2015]

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



ResNet

Case Study: ResNet

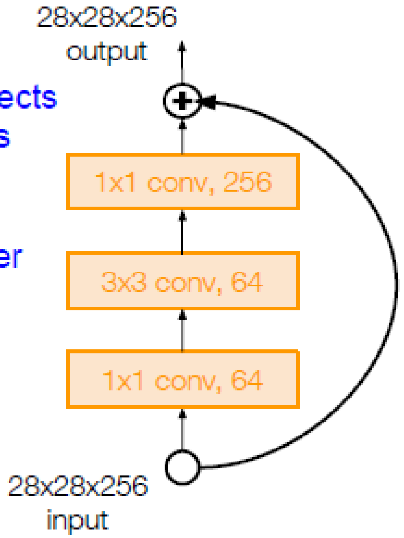
[He et al., 2015]

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



ResNet

- Training details

Training ResNet in practice:

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

ResNet

- Results

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

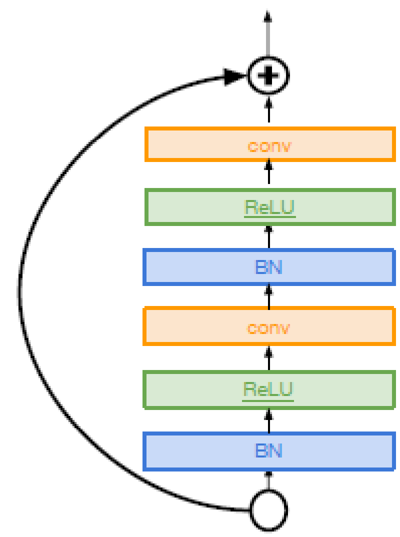
MSRA @ ILSVRC & COCO 2015 Competitions

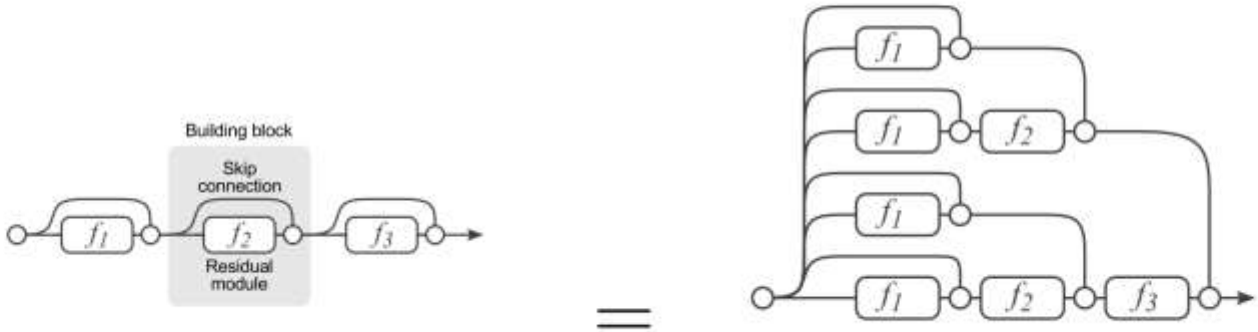
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

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

Other: Identity Mappings in ResNet

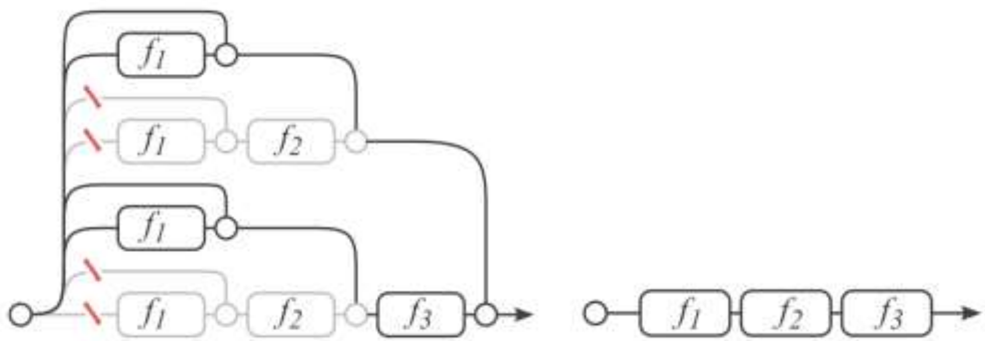
- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance





(a) Conventional 3-block residual network (b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.



(a) Deleting f_2 from unraveled view (b) Ordinary feedforward network

Figure 2: Deleting a layer in residual networks at test time (a) is equivalent to zeroing half of the paths. In ordinary feed-forward networks (b) such as VGG or AlexNet, deleting individual layers alters the only viable path from input to output.

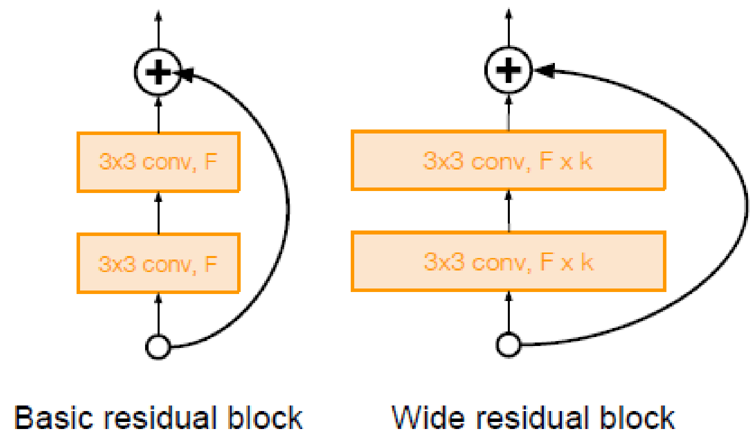
Residual Networks Behave Like Ensembles of Relatively Shallow Networks, Veit etal, CVPR 2016

Other: Wide ResNets

Wide Residual Networks

[Zagoruyko et al. 2016]

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

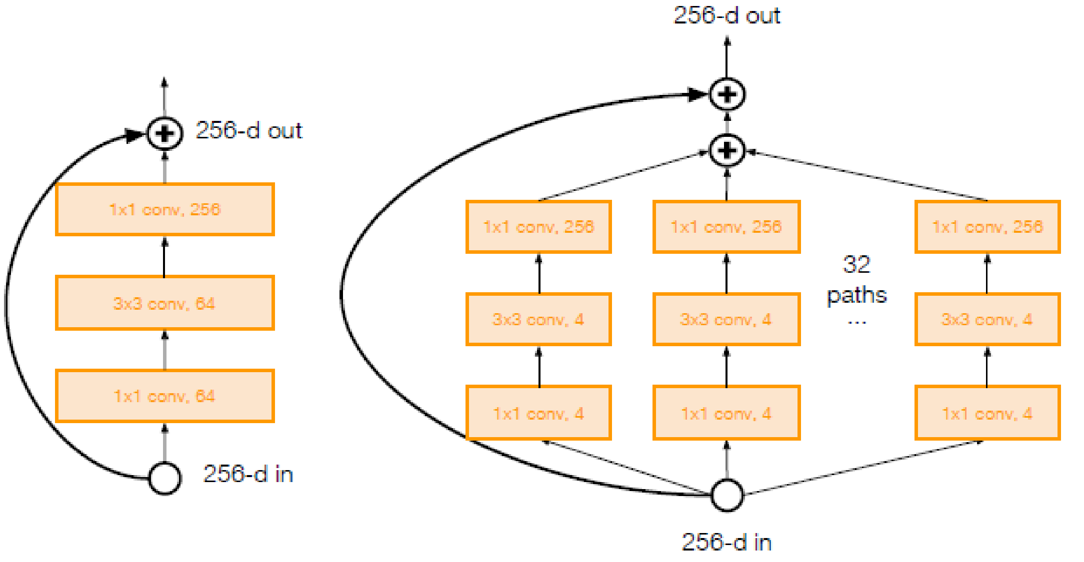


Other: ResNeXt

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module

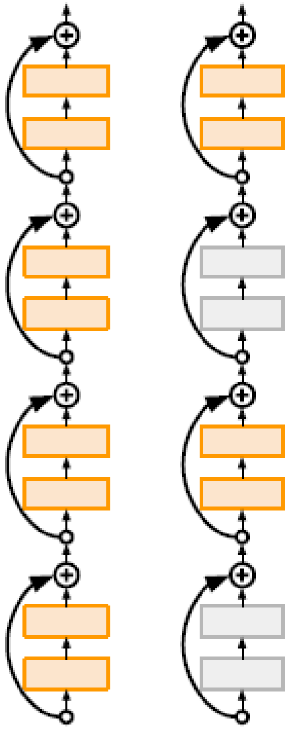


Other:ResNet with Stochastic Depth

Deep Networks with Stochastic Depth

[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time

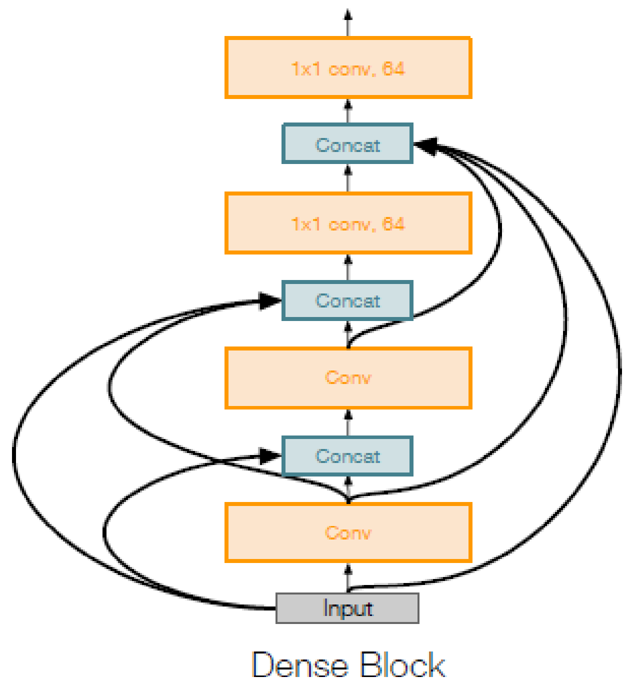


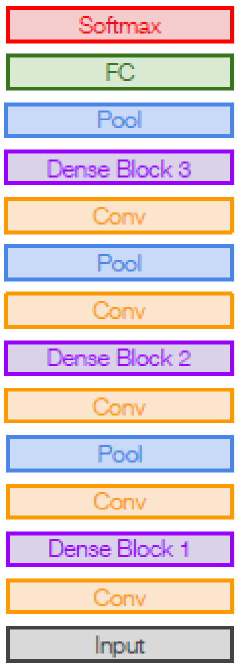
DenseNet

Densely Connected Convolutional Networks

[Huang et al. 2017]

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

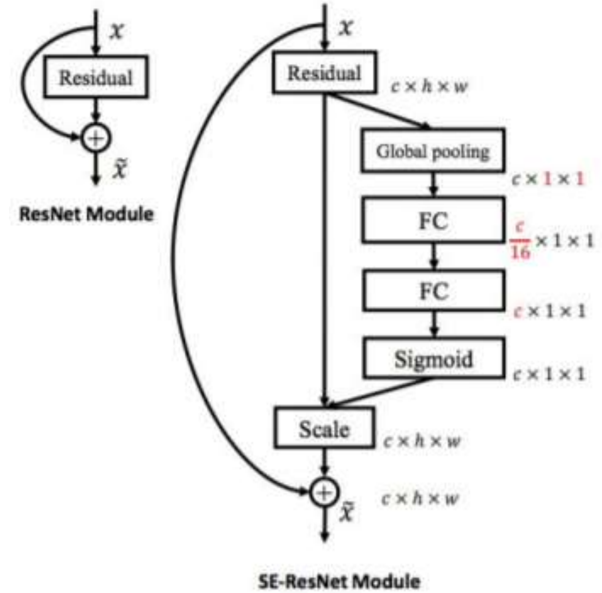


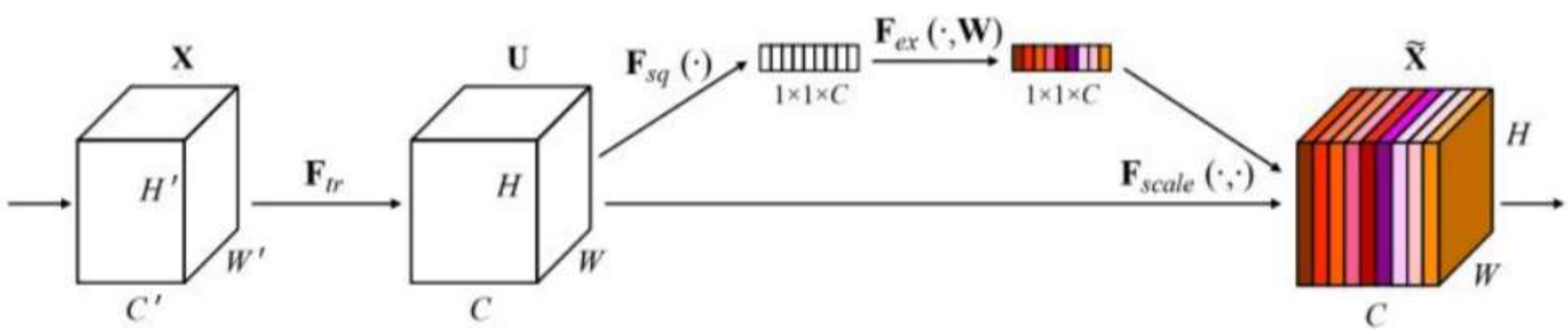


Squeeze-and-Excitation Networks (SENet)

[Hu et al. 2017]

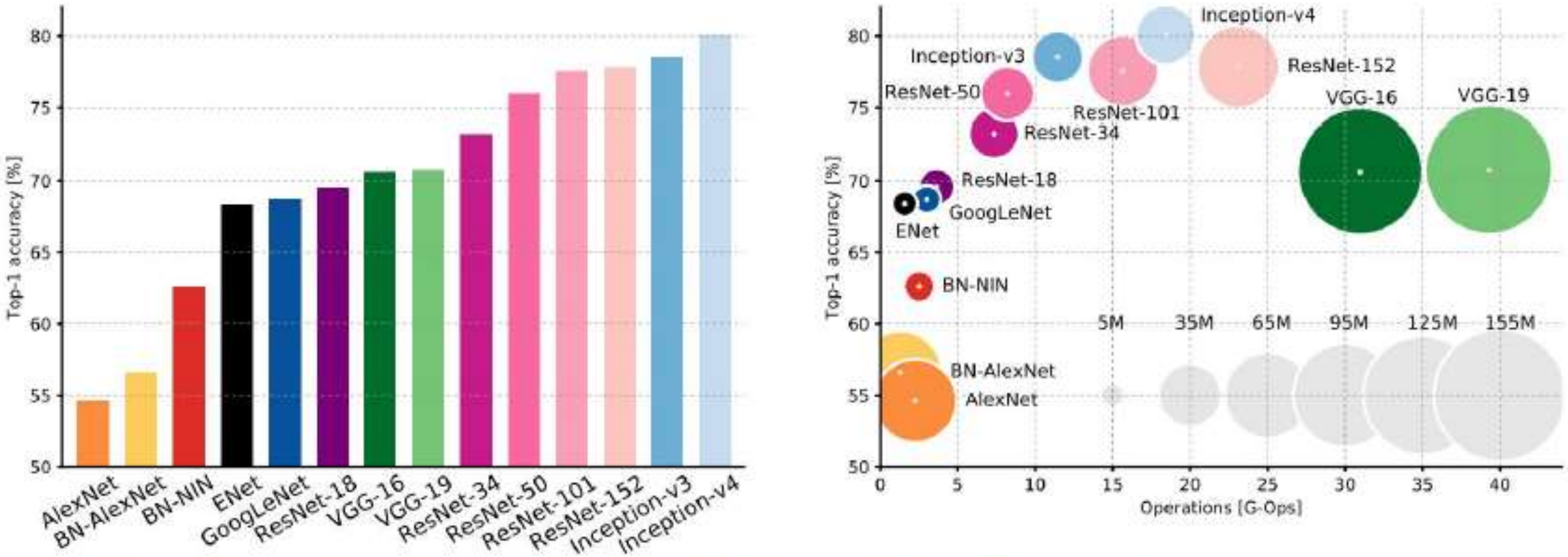
- Add a "feature recalibration" module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)





Model complexity

Comparing complexity...



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

Outline

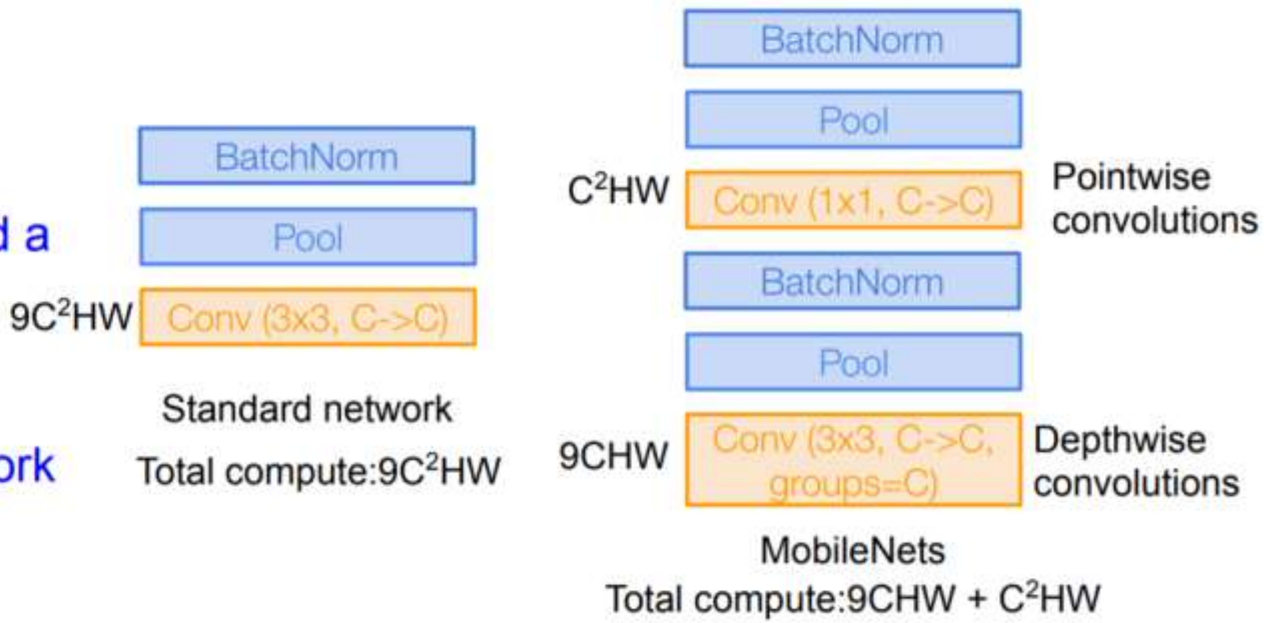
- CNN architectures
  - Sequential structure: LeNet/AlexNet/VGGNet
  - Parallel branches: GoogLeNet
  - Residual structure: ResNet/DenseNet
  - Network Architecture Search

Acknowledgement: Zemel et al's CSC411 and Feifei Li et al's cs231n notes

Efficient networks

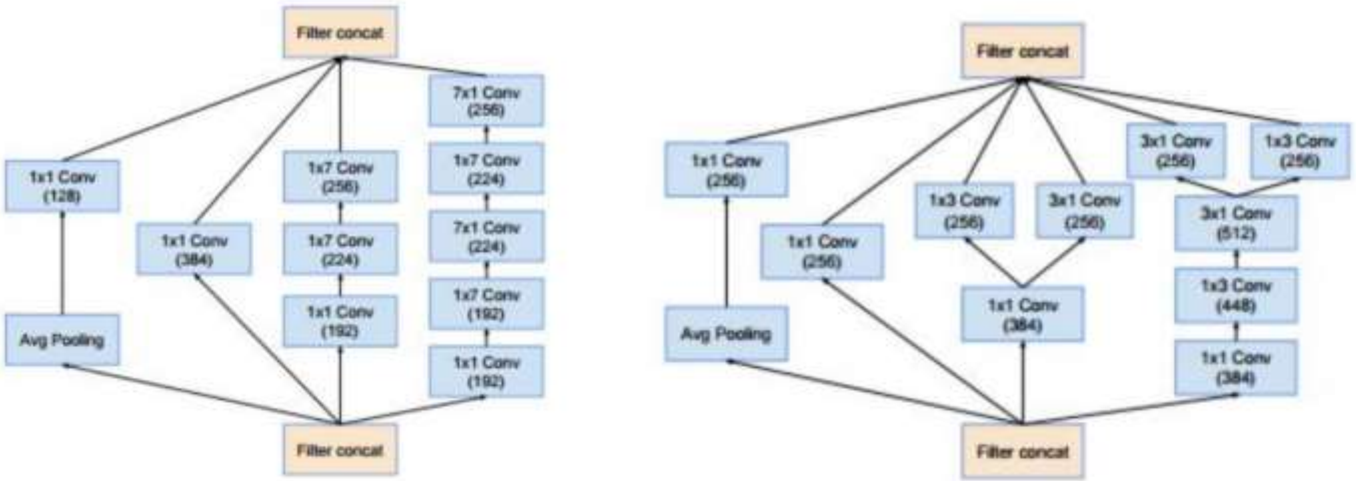
- MobileNets: Efficient Convolutional Neural Networks for Mobile Applications [Howard et al. 2017]

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018



Network Architecture

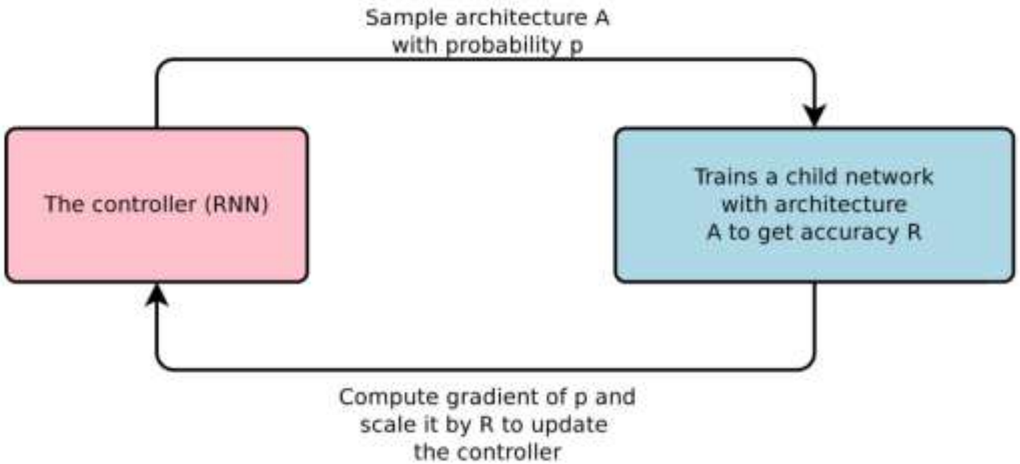
- Problems with network architecture
  - Designing NA is hard
  - Lots of human efforts go into tuning them
  - Not a lot of intuition into how to design them well
  - Can we learn good architectures automatically?



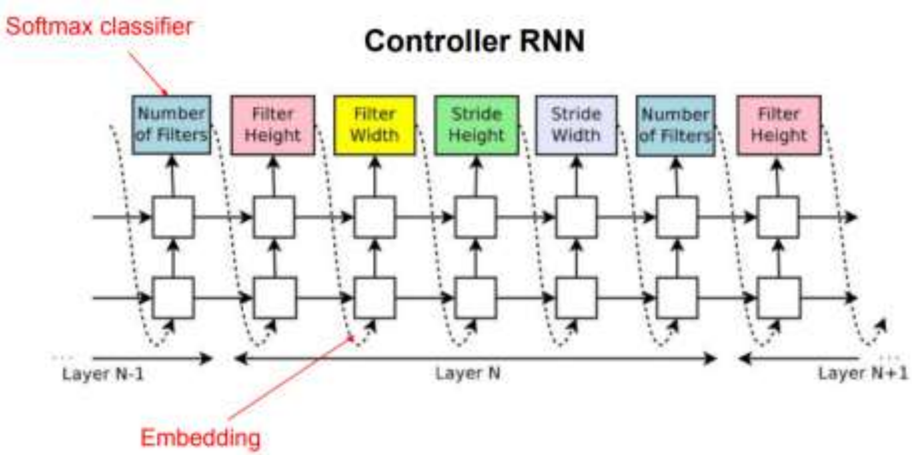
Two layers from the famous Inception V4 computer vision model.
Szegedy et al, 2017

Network Architecture

- Neural architecture search (Zoph and Le, ICLR 2016)

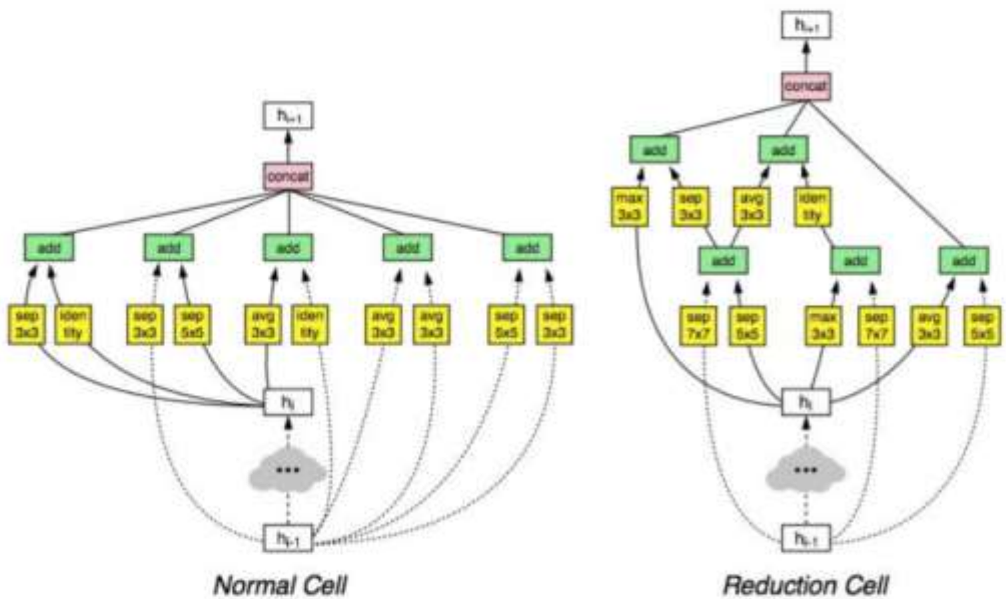


- "Controller" network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a "reward" R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)



Network Architecture

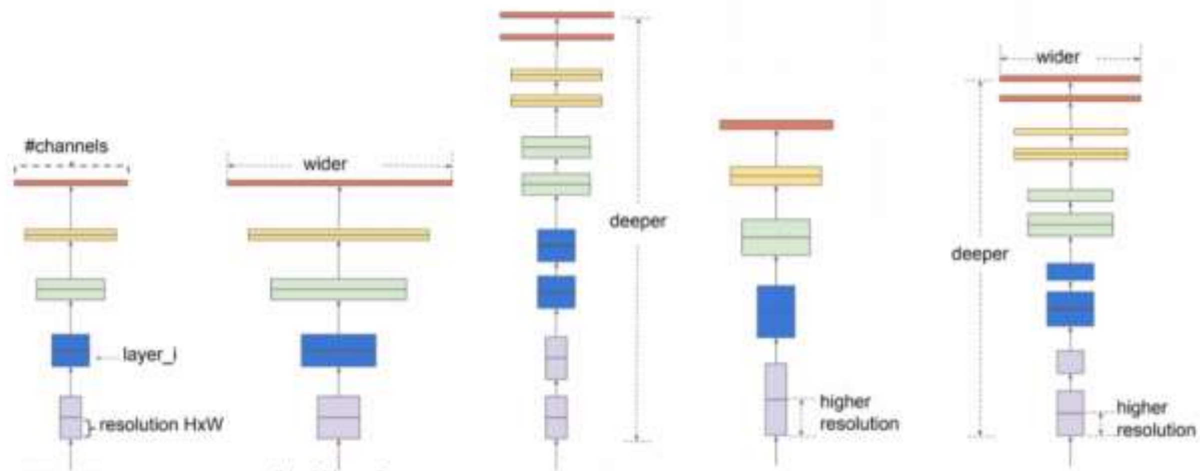
- Neural architecture search (Zoph et al. 2017)
  - Design a search space of building blocks ("cells") that can be flexibly stacked
  - NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet
  - Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)



Network Architecture

- EfficientNet: Smart Compound Scaling [Tan and Le. 2019]
  - Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
  - Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
  - Scale up using smart heuristic rules

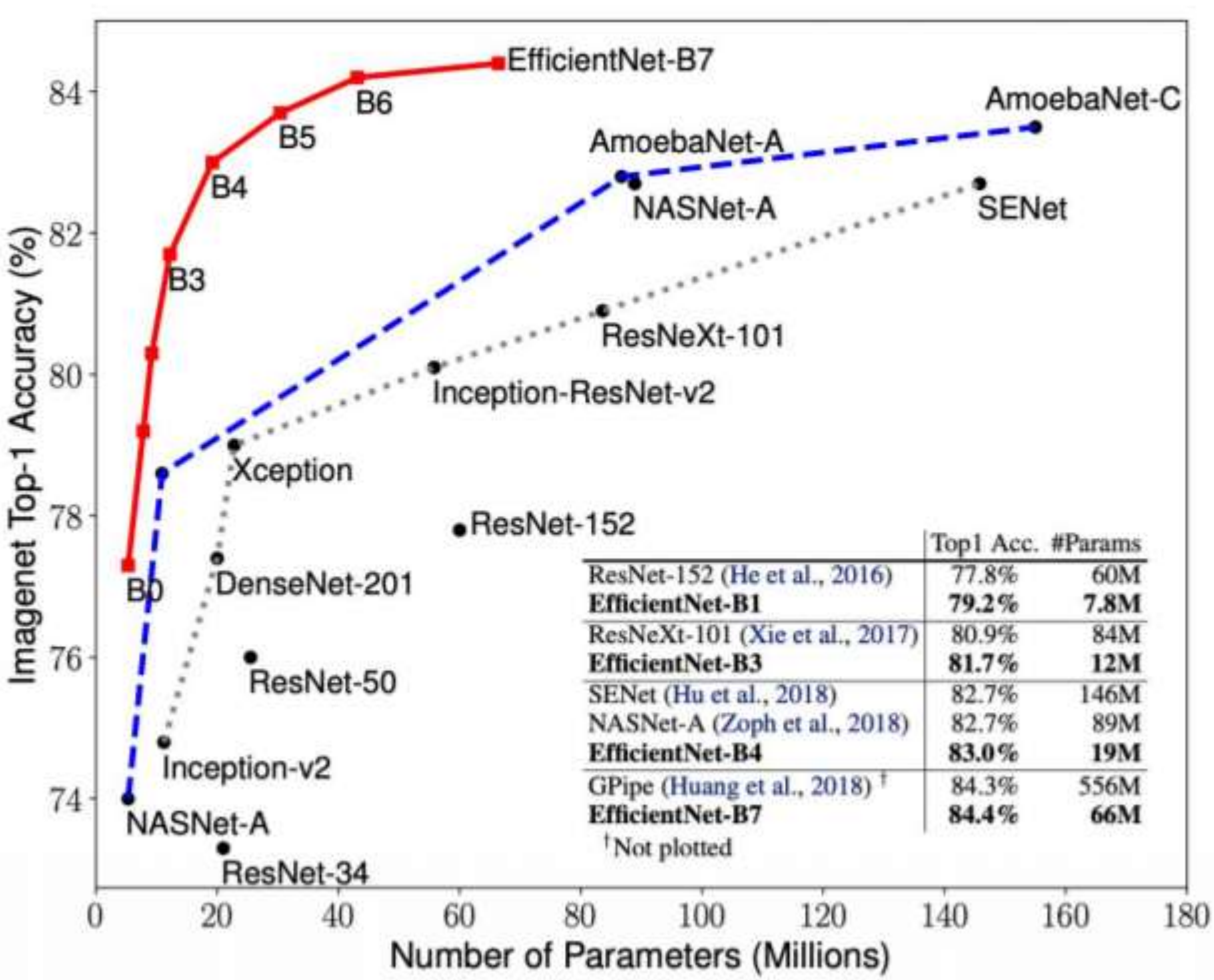
$$\text{depth: } d = \alpha^{\phi}$$
$$\text{width: } w = \beta^{\phi}$$
$$\text{resolution: } r = \gamma^{\phi}$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$



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

Network Architecture

- EfficientNet: Smart Compound Scaling [Tan and Le. 2019]



Network structure summary

- AlexNet showed that you can use CNNs to train Computer Vision models.
- ZFNet, VGG shows that bigger networks work better
- GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
- ResNet showed us how to train extremely deep networks
  - Limited only by GPU & memory!
  - Showed diminishing returns as networks got bigger
- After ResNet: CNNs were *better than the human metric* and focus shifted to Efficient networks:
  - Lots of tiny networks aimed at mobile devices: MobileNet, ShuffleNet
- Neural Architecture Search can now automate architecture design

- Read the ResNet paper
- Read the codes of ResNet
- Use ResNet for image classification on CIFAR10, you can either use the model pretrained from imagenet or trained from scratch..
- What’s the performance difference between the model finetuned from a pretrained ResNet and that trained from scratch?
- Use ResNet for crowd counting task, and report the performance on ShanghaiTech Crowd counting dataset.
- You can choose differents parameters for your experiments.
- Due date: Oct 29, 2023
- Submission: codes+report
- Email the codes and report to TA
- Late submission: zero point!