Convolution Neural Network Architecture

Deep Learning Course

E. Fatemizadeh

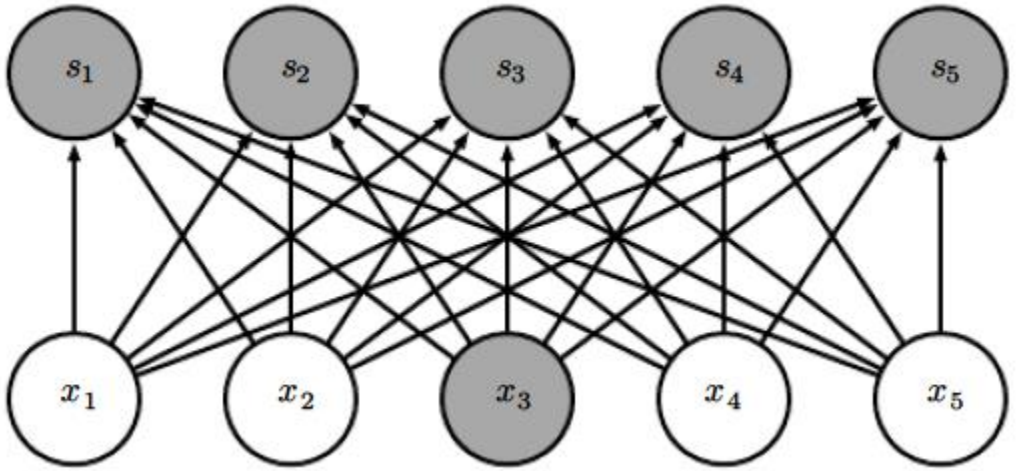
Fall 2021

Fully Connected Network (FCN)

› A simple matrix representation

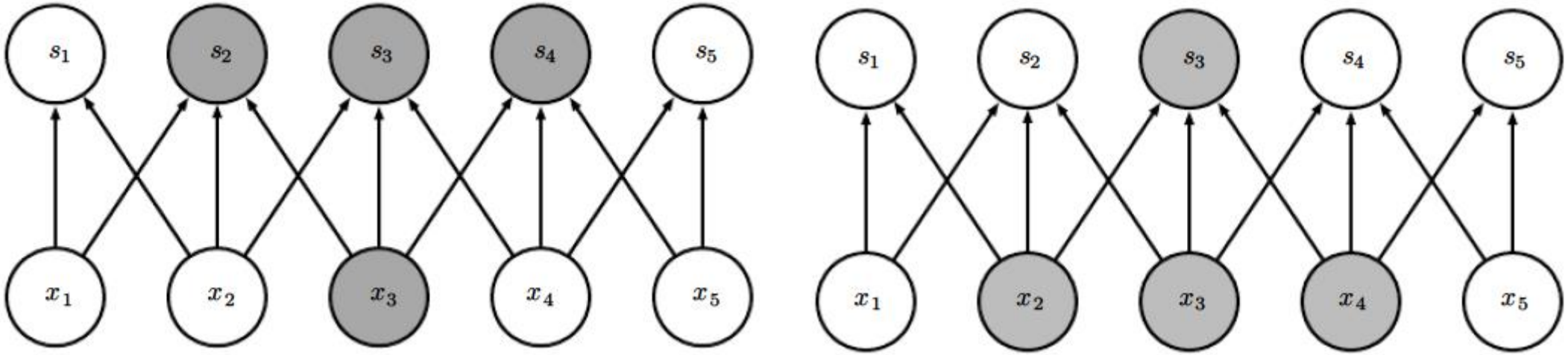
$$Y = ReLu(WX), w_{ij}\ is\ arbitrary$$

› *Fully Connected* (FC) or *Dense*!



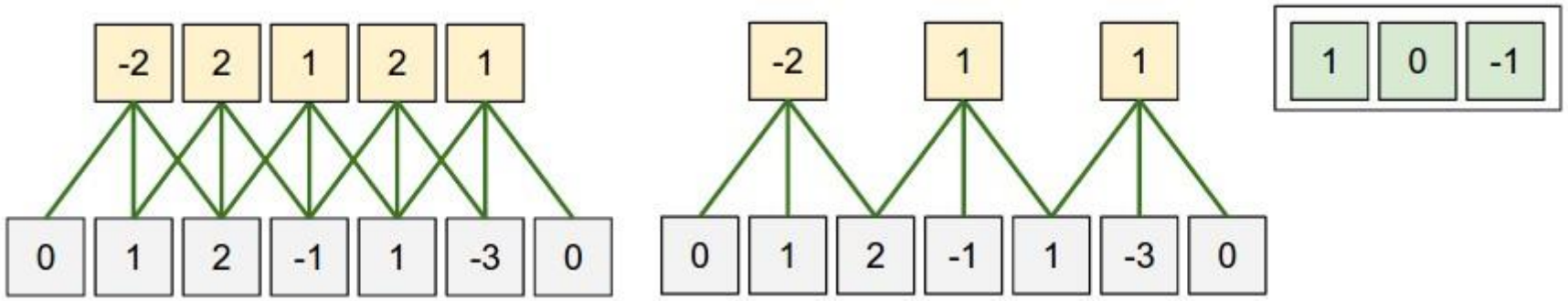
How to reduce number of parameters?

- Sparse Connection/interaction



How to reduce number of parameters?

- Parameter Sharing (and Stride):



Convolution/Correlation

› Linear Convolution (Linear Shift Invariant Systems)

$$y[n] = \sum_m x[m]h[n-m] = \sum_k h[k]x[n-k]$$

› Correlation:

$$y[n] = \sum_m x[m]h[n+m] = \sum_k h[k]x[n+k]$$

Convolution/Correlation

- Sparse Connectivity:
  - Yes, short length $h[n]$, kernel
- Parameter Sharing:
  - Yes, Linear Shift Invariant, a single $h[n]$ and NOT $h[n,m]$!
- Equivariant Representations:
  - Linearity and Shift Invariant! $f(g(x)) = g(f(x))$, some certain $g - operator$!

Convolution/Correlation

- Matrix-Multiplication Representation:
  - Linear Convolution, Toeplitz Matrix:
    - Output size: *full*, *same*, *valid*

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \end{bmatrix} = \begin{bmatrix} h[0] & 0 & 0 \\ h[1] & h[0] & 0 \\ h[2] & h[1] & h[0] \\ h[3] & h[2] & h[1] \\ 0 & h[3] & h[2] \\ 0 & 0 & h[3] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \end{bmatrix}$$

$$Toeplitz(h) = \begin{bmatrix} h[0] & 0 & 0 \\ h[1] & h[0] & 0 \\ h[2] & h[1] & h[0] \\ h[3] & h[2] & h[1] \\ 0 & h[3] & h[2] \\ 0 & 0 & h[3] \end{bmatrix}$$

Numerical Example

› Linear Convolution:

$$x[n]=\{\underline{1},\ 2,\ 2\},\quad h[n]=\{\underline{1},\ -1\},\quad N_1=3, N_2=2$$

$$\mathbf{y}=\mathbf{Hf}=\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 0 \\ -2 \end{bmatrix}\quad y[n]=\{\underline{1},\ 1,\ 0,\ -2\}$$

Numerical Example

› Circular Convolution, Circulant Matrix:

$$C = \begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-1} \\ c_{n-1} & c_0 & c_1 & c_2 & \cdots \\ c_{n-2} & c_{n-1} & c_1 & \cdots & \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ c_1 & c_2 & \cdots & c_{n-1} & c_0 \end{pmatrix}$$

Numerical Example

› Circular Convolution:

$$x[n] = \{\underline{1},\ 2,\ 2\},\quad h[n] = \{\underline{1},\ -1\},\quad N_1 = 3, N_2 = 2$$

$$\mathbf{y = Hf} = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

$$y[n] = \{-\underline{1},\ 1,\ 0\}$$

2D Convolution

› Definition:

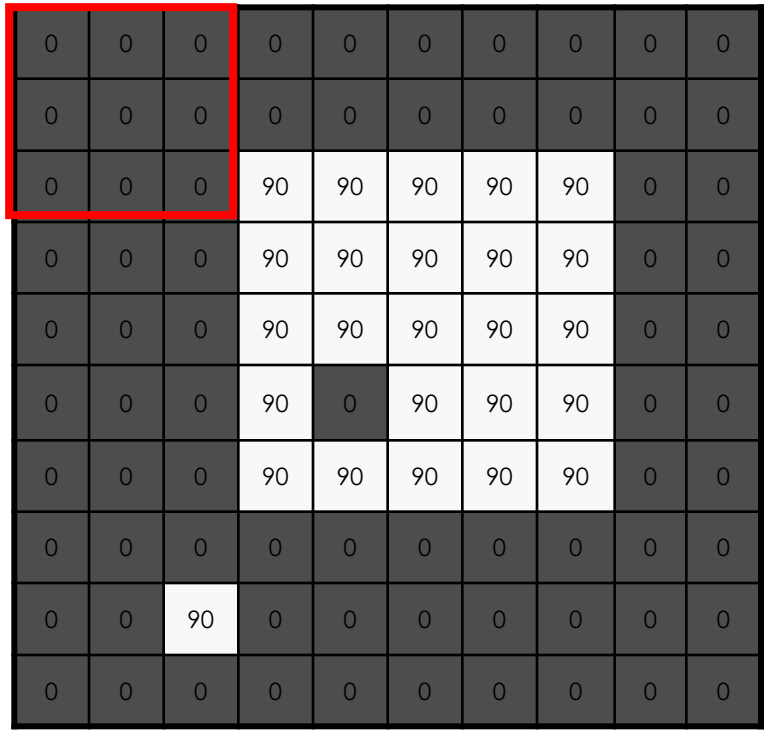
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{9.4}$$

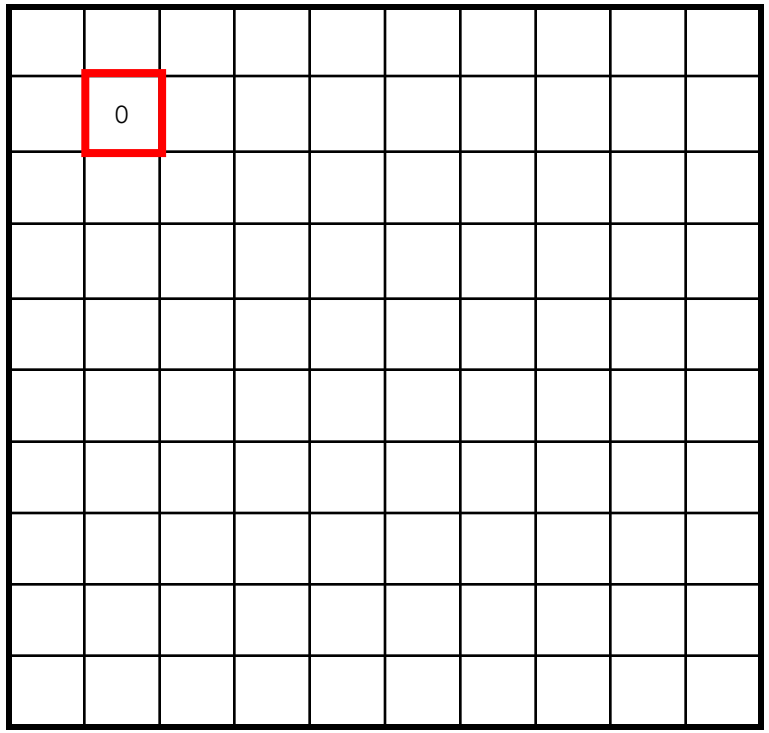
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{9.5}$$

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Image/Matrix Filtering

› Step #1:



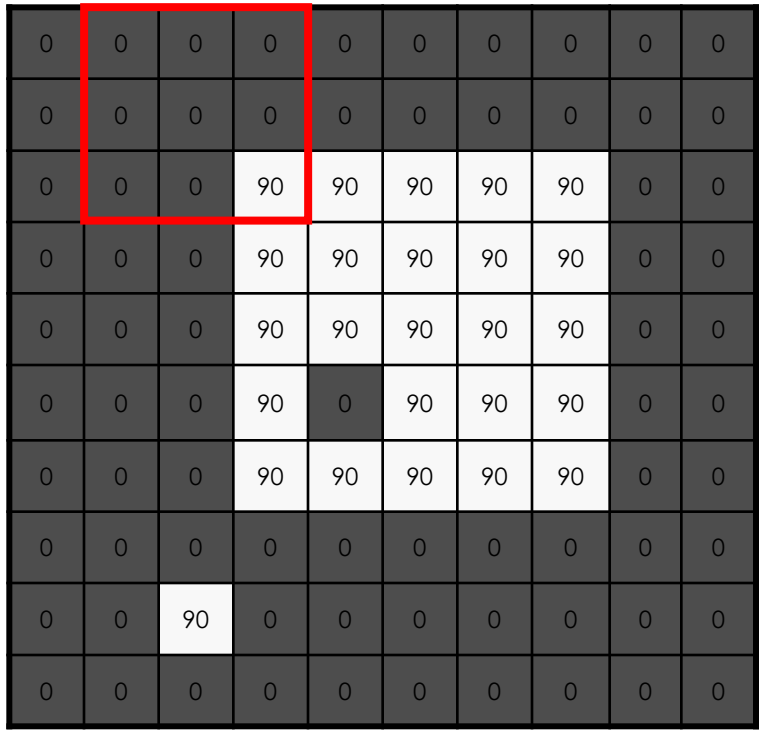


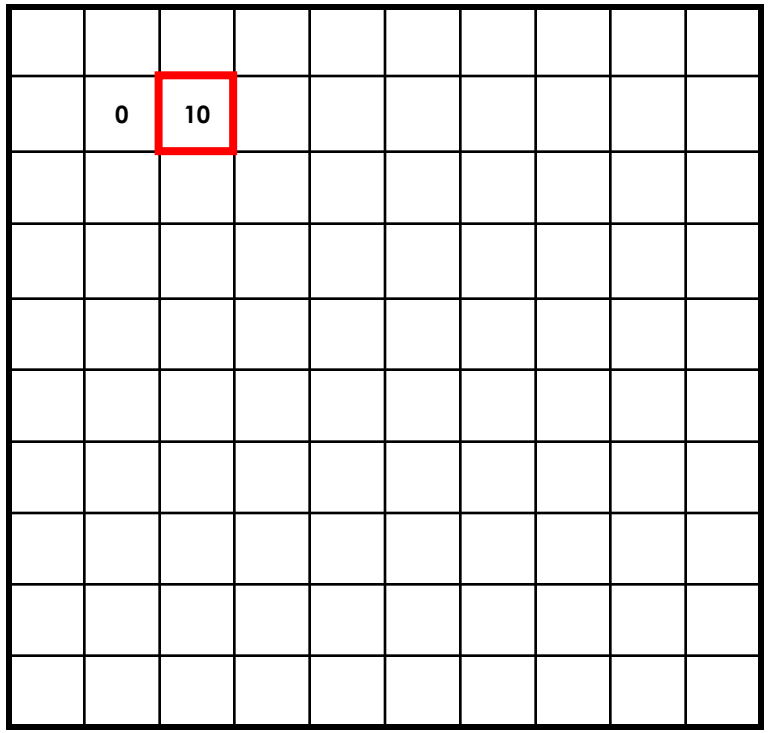
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #2



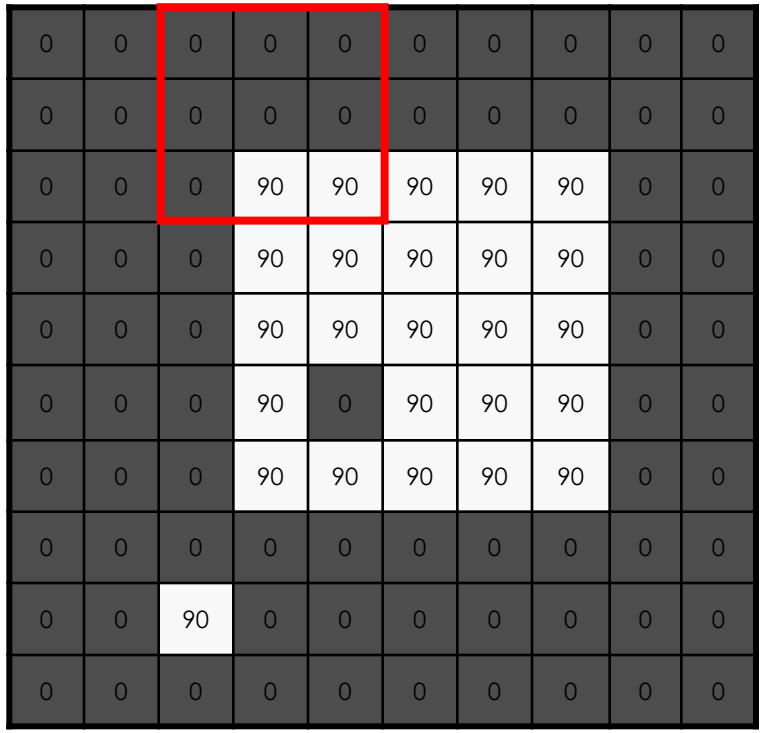


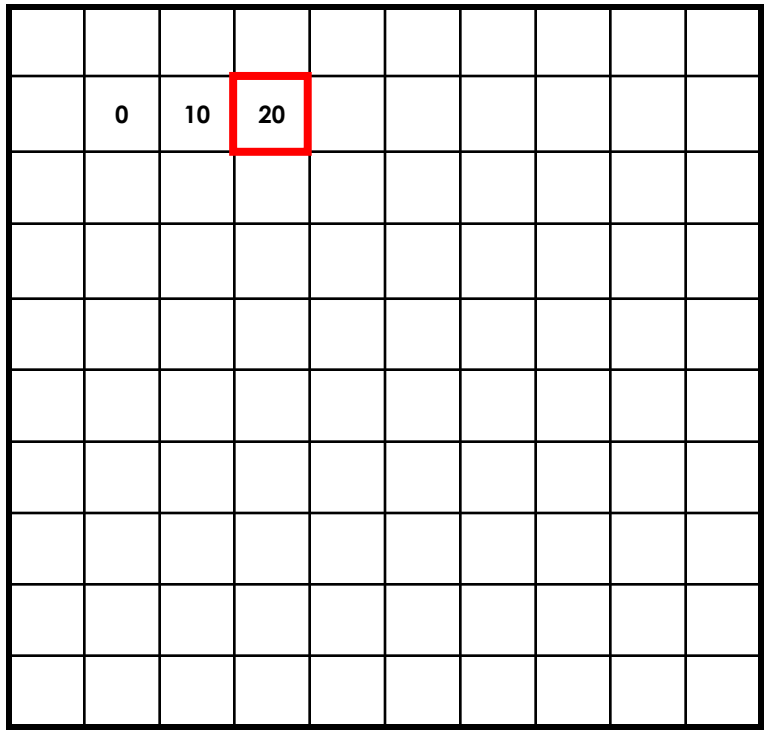
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #3





$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #4

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30					

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #5

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30				

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #6

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30				
				50					

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Step #7

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30				
						90			
					50				

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Image/Matrix Filtering

› Final Results

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

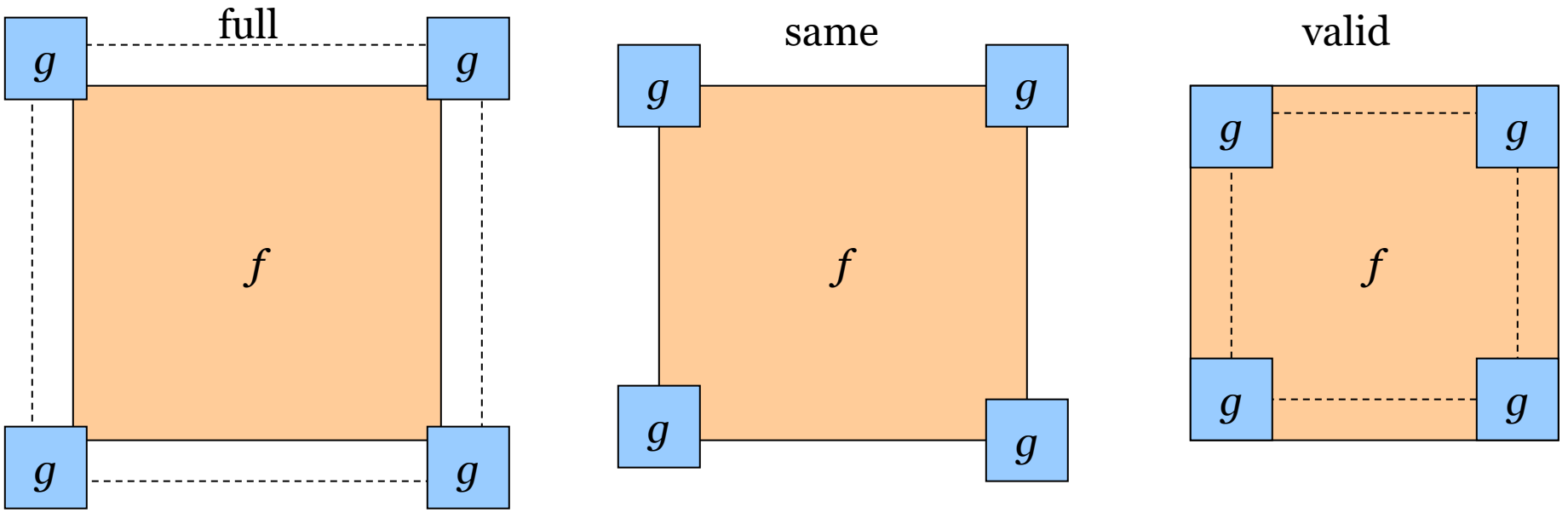
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

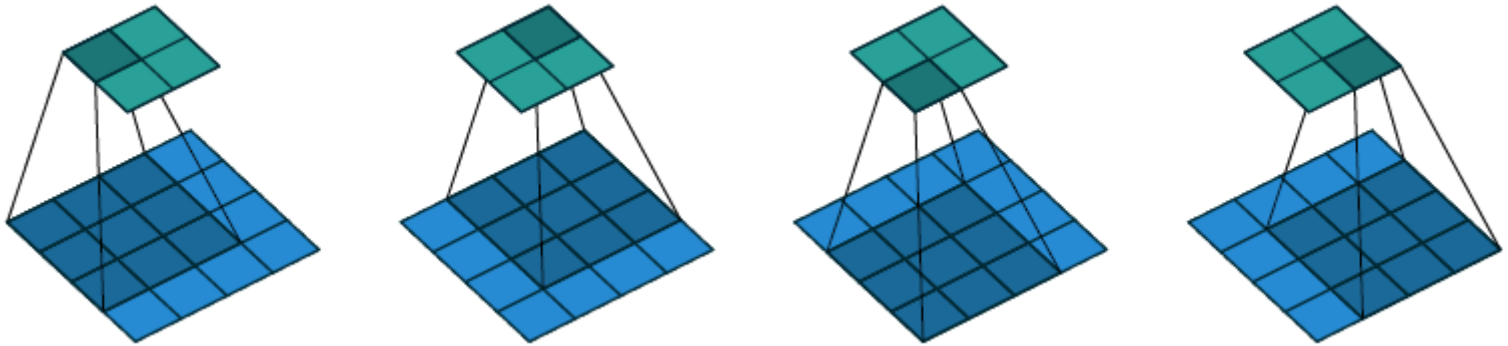
Practical matters

› Padding: Full, Same (Half), and Valid Scenario!



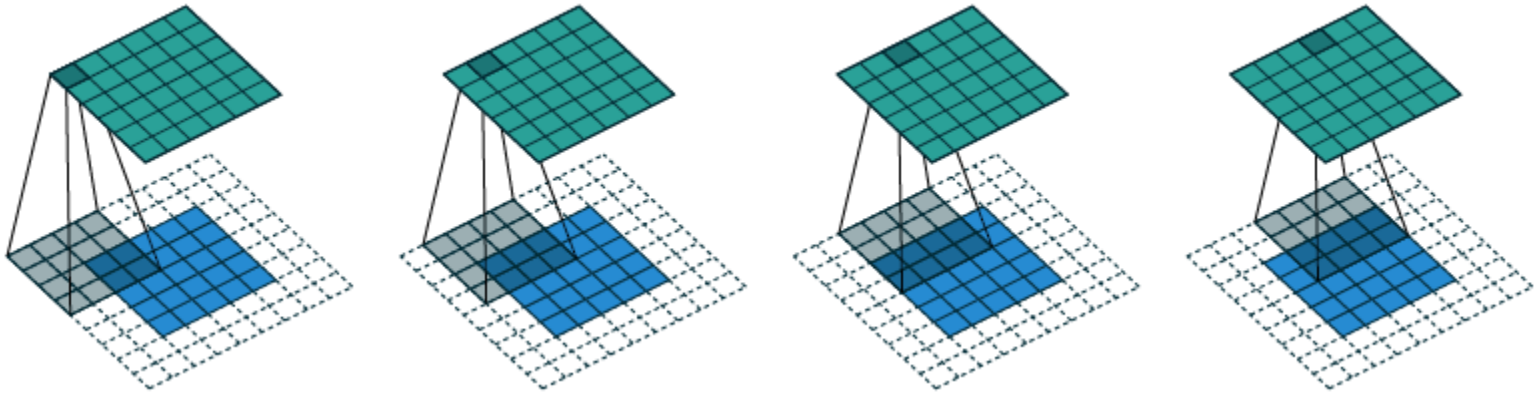
Practical matters – Padding and Stride

› No Padding (Valid) – Unit Stride



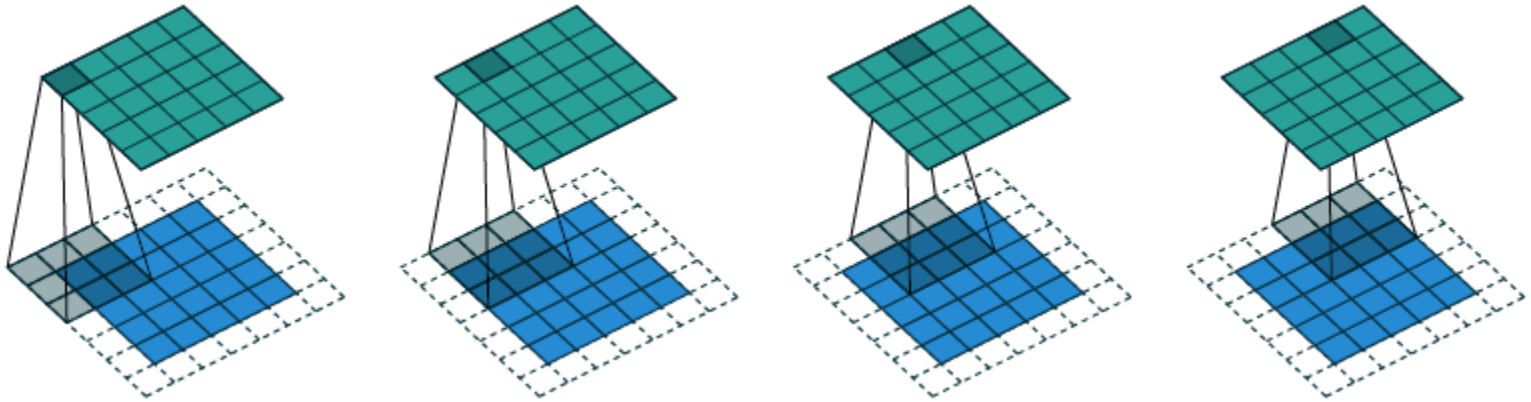
Practical matters – Padding and Stride

› Padding – Unit Stride



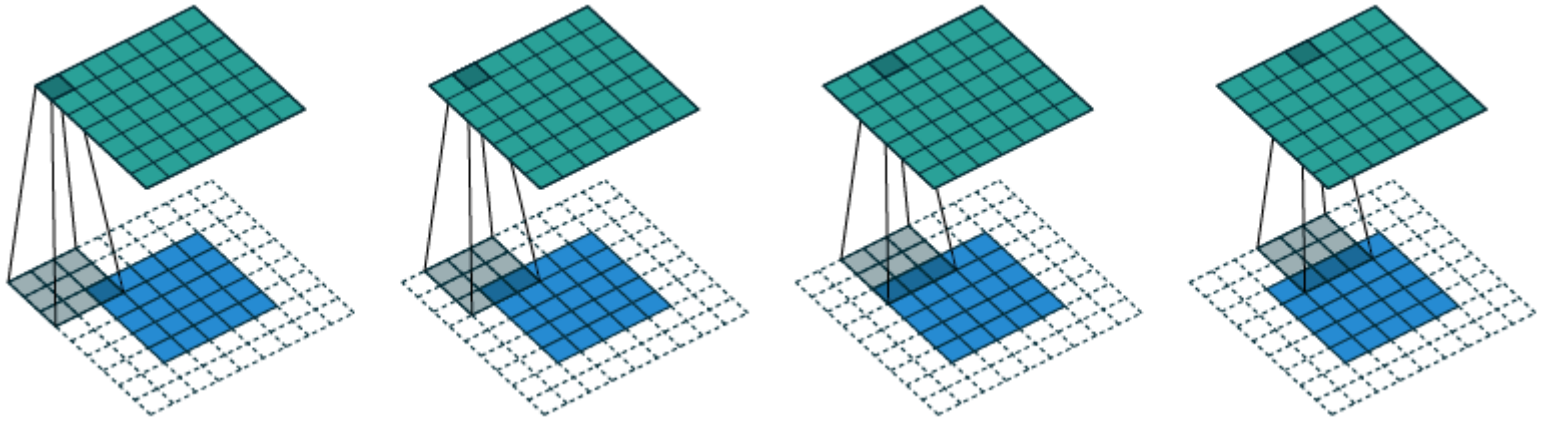
Practical matters – Padding and Stride

› Half Padding (Same) – Unit Stride



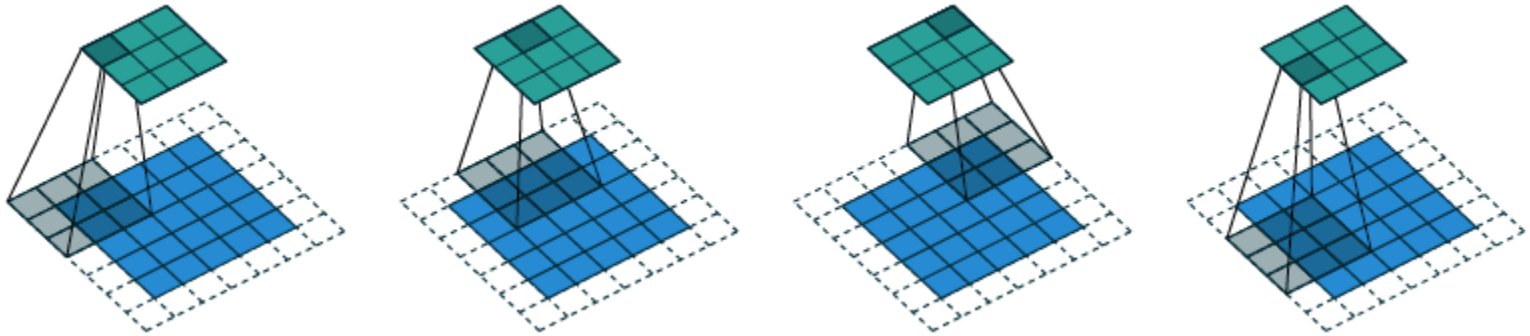
Practical matters – Padding and Stride

› Full Padding (Full) – Unit Stride



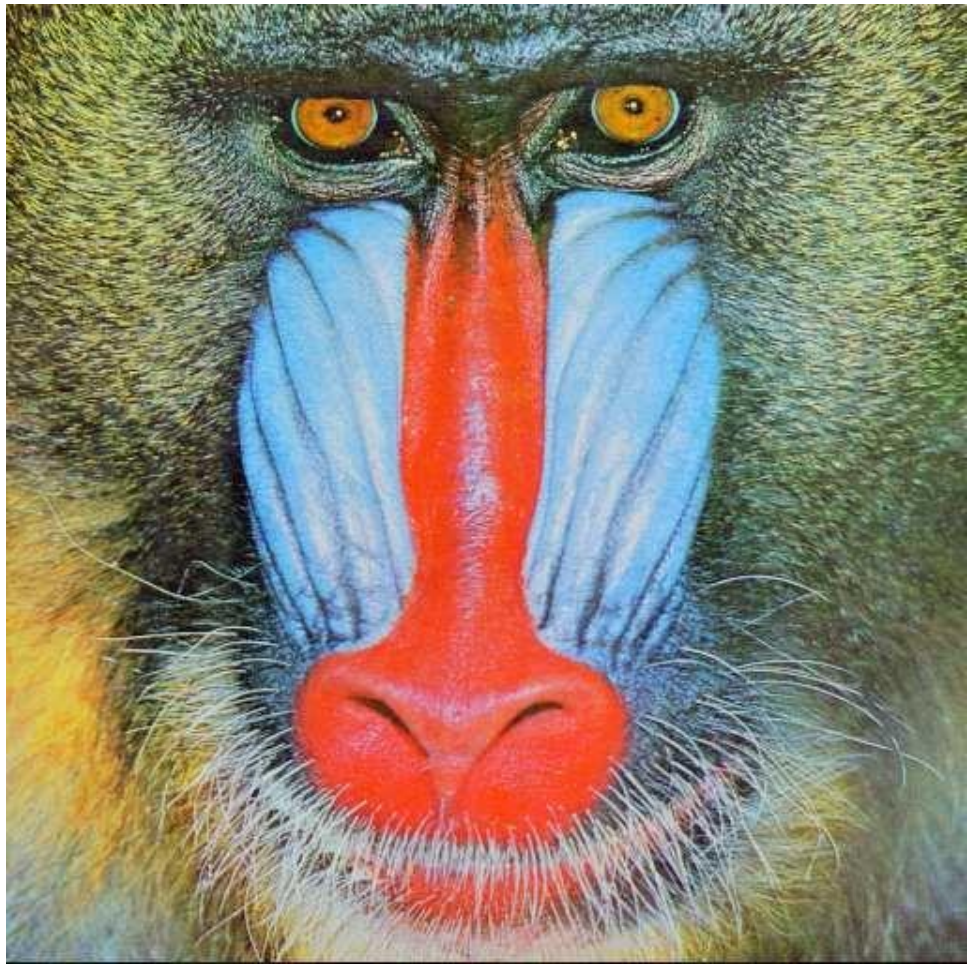
Practical matters – Padding and Stride

› Padding– 2x2 Stride (Subsampling)



Convolution Effect!

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



Convolution Effect!

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution Effect!

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Convolution Effect!

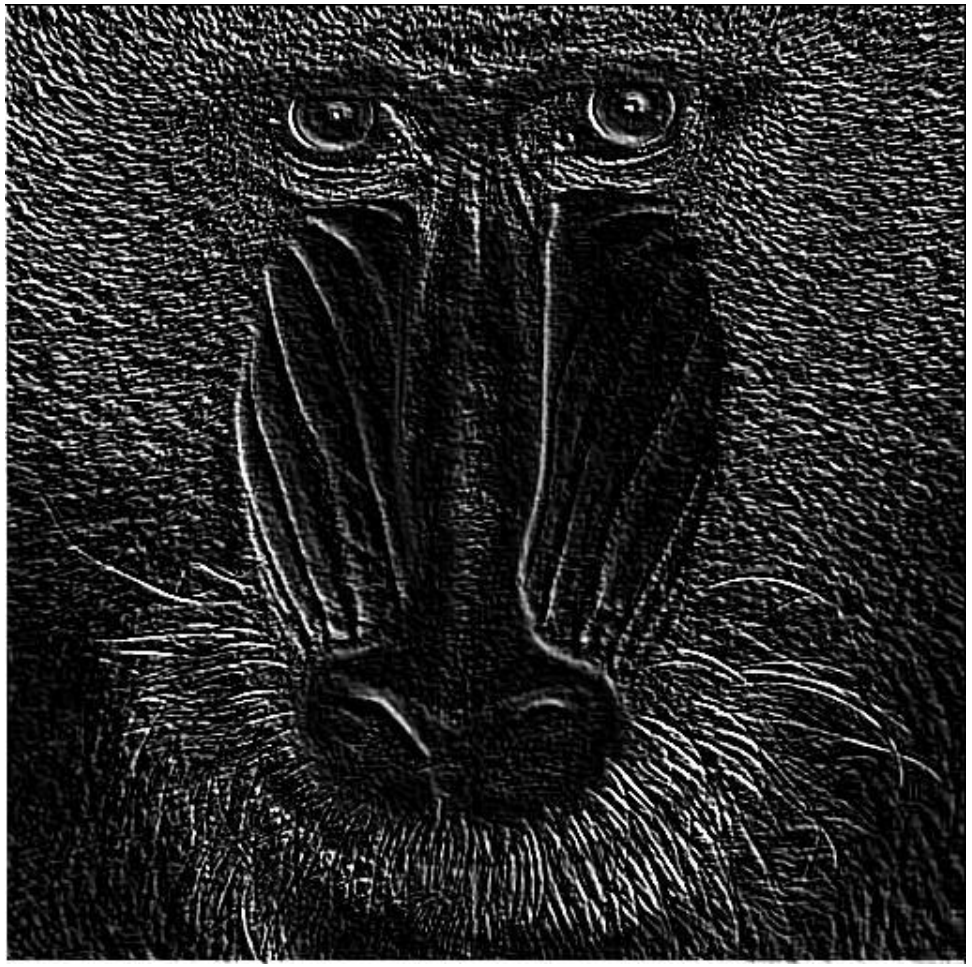
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Convolution Effect!

[?]

Convolution Effect!

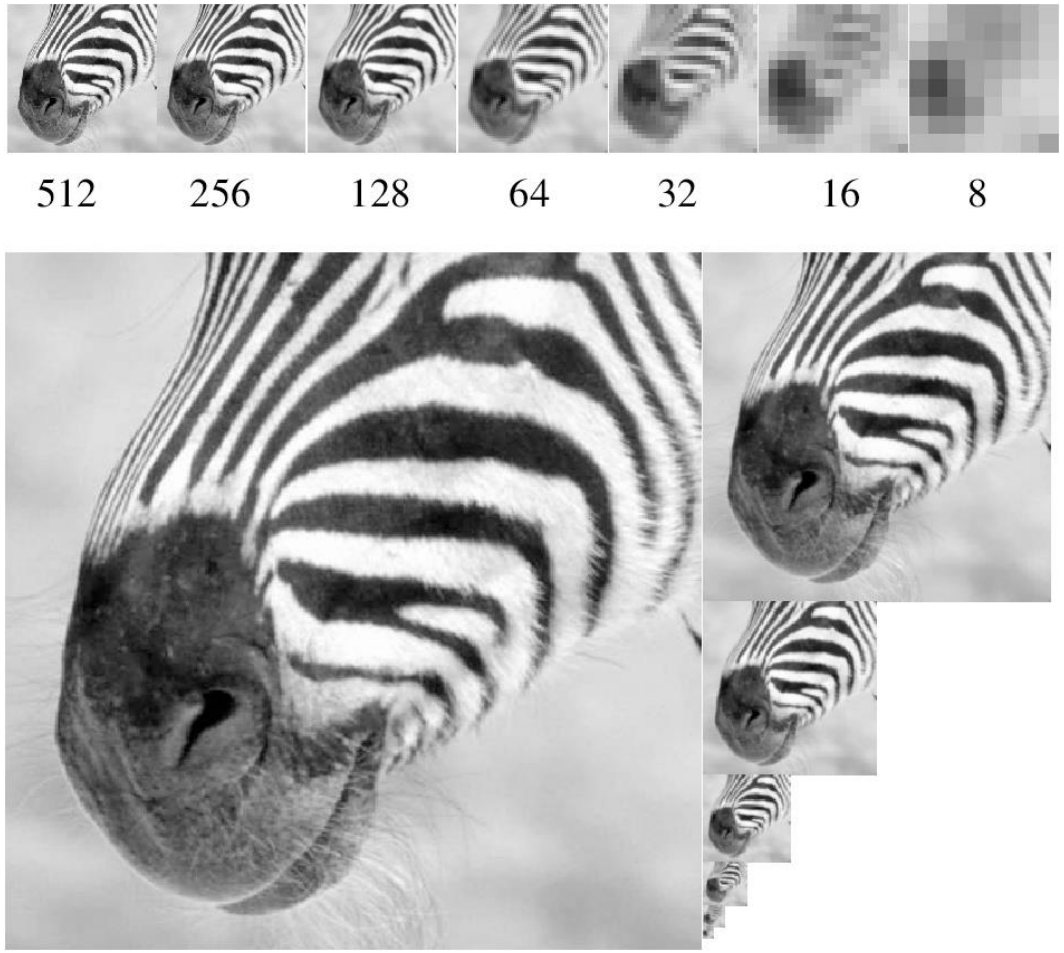
[?]



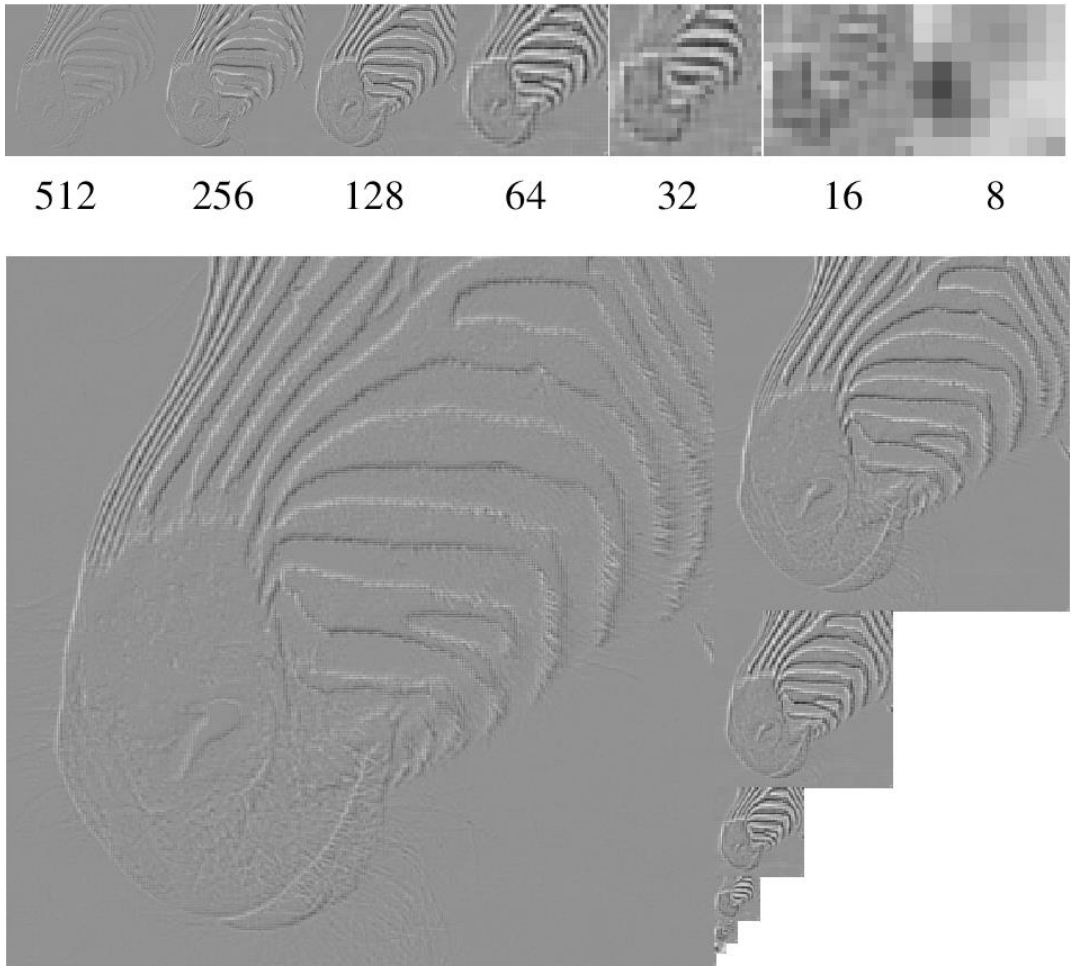
Convolution Effect!

[?]

Convolution+Scaling!

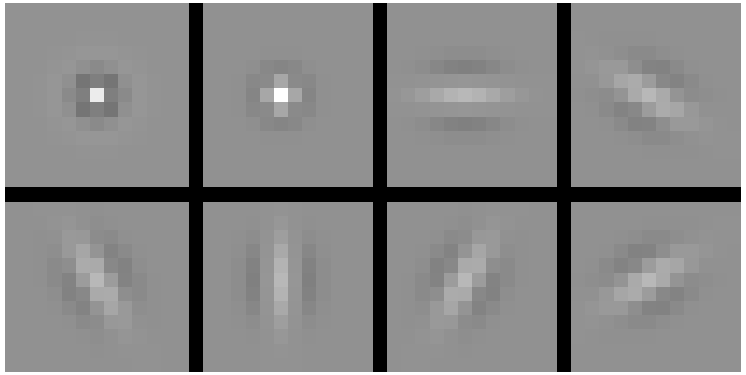


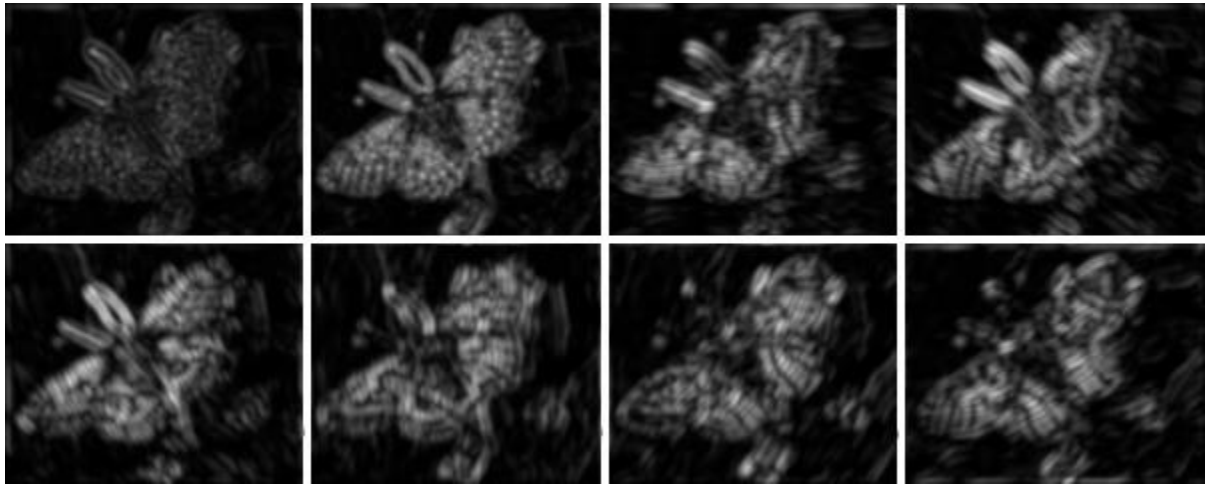
Convolution + Scaling!



Filter banks!

› Eight Filters



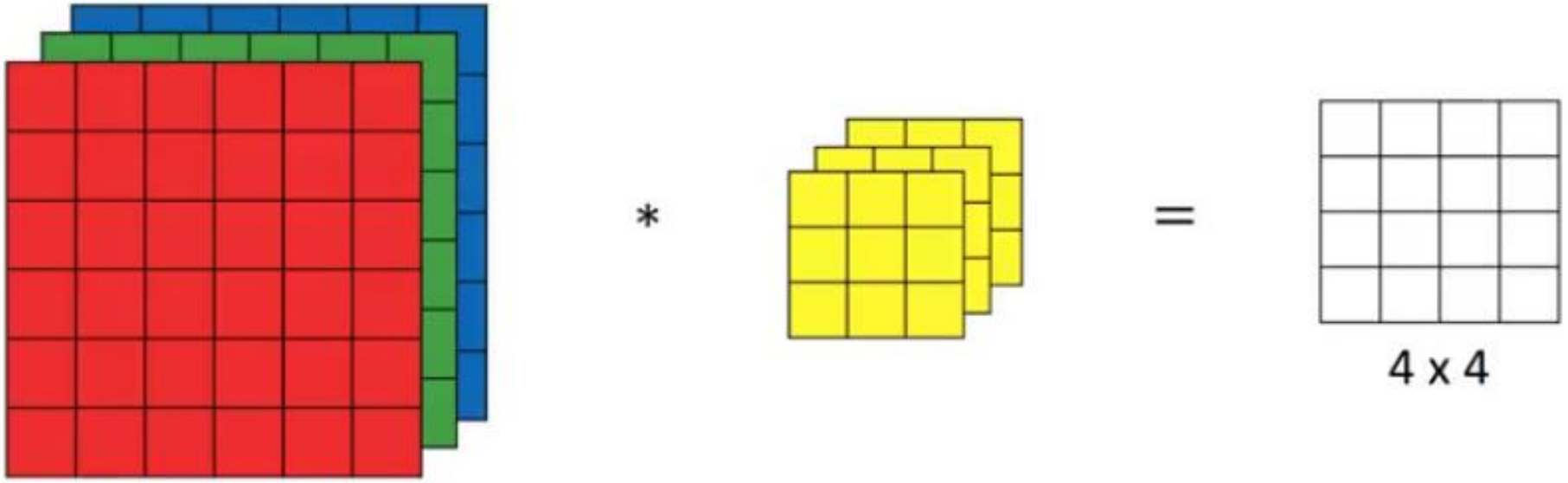


Vocabulary

- Input, x
- Kernel, h
- Feature Map (Output), y

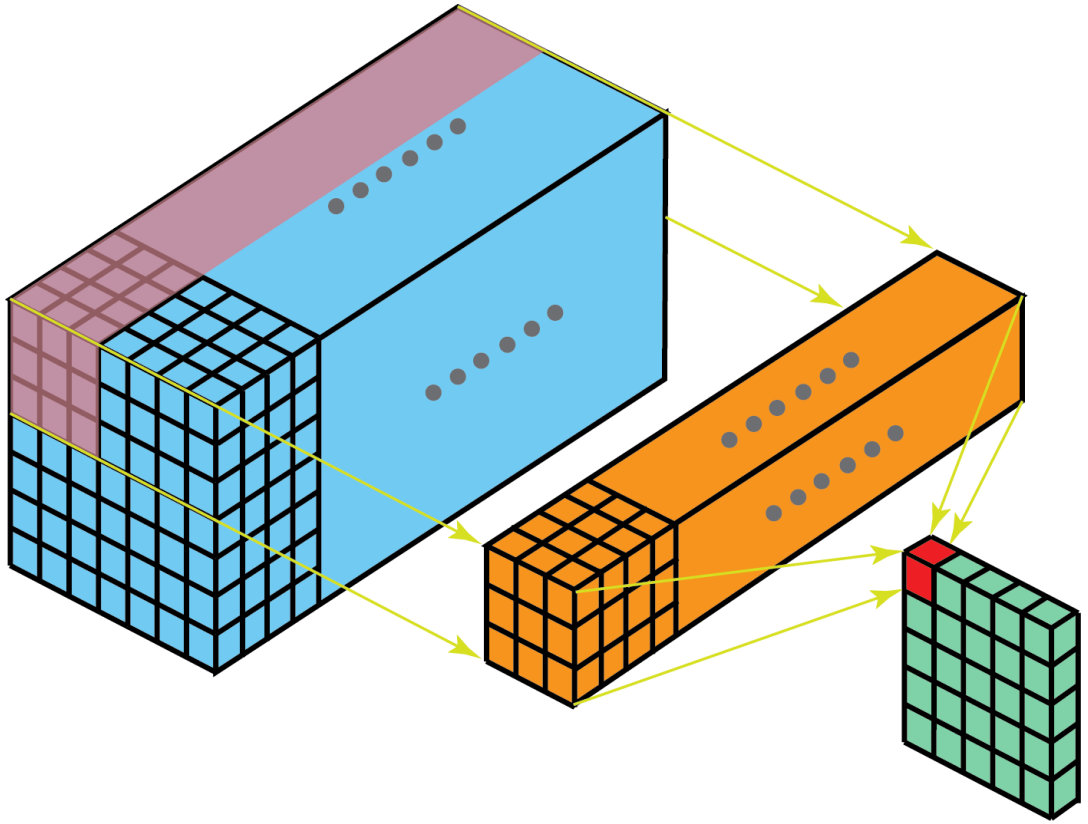
3D - ConvLayer

› Is Not Fully 3D Convolution



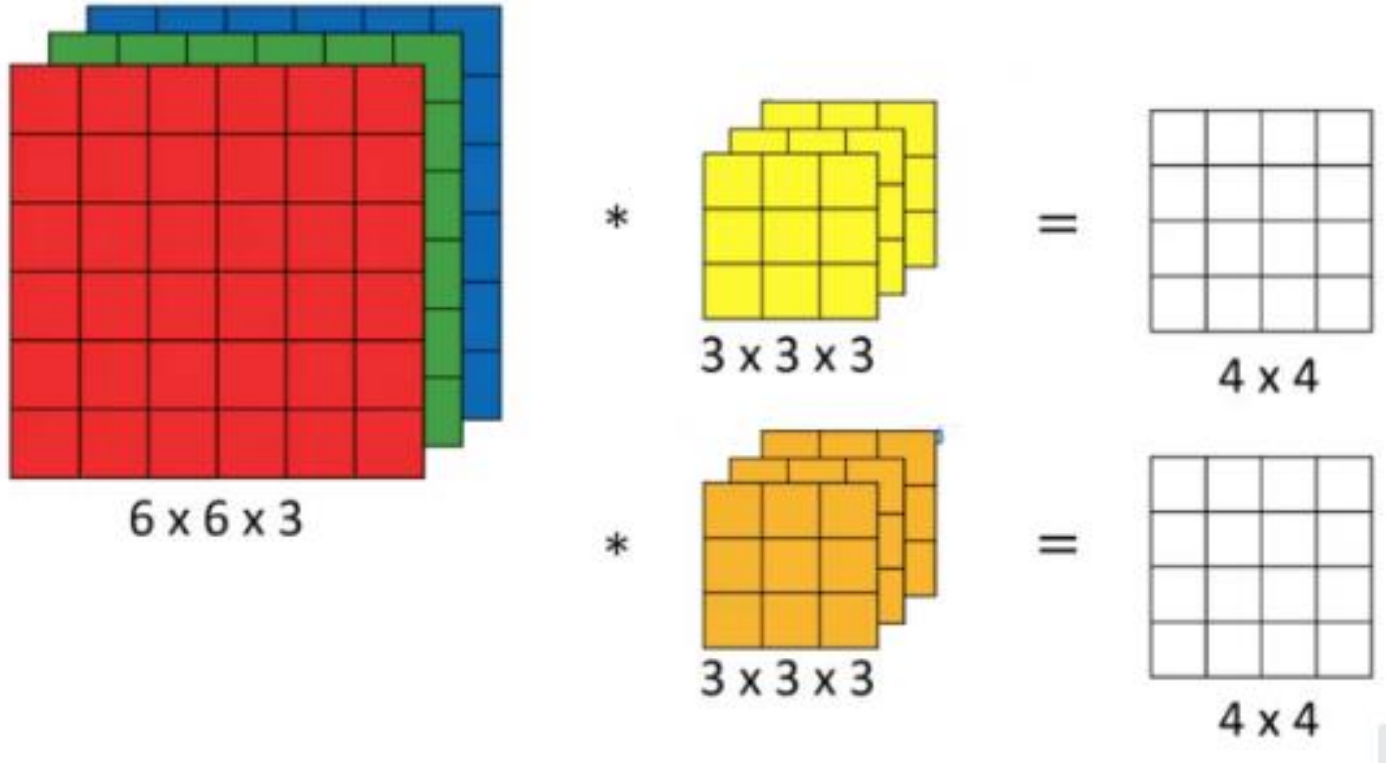
3D - ConvLayer

› Is Not Fully 3D Convolution



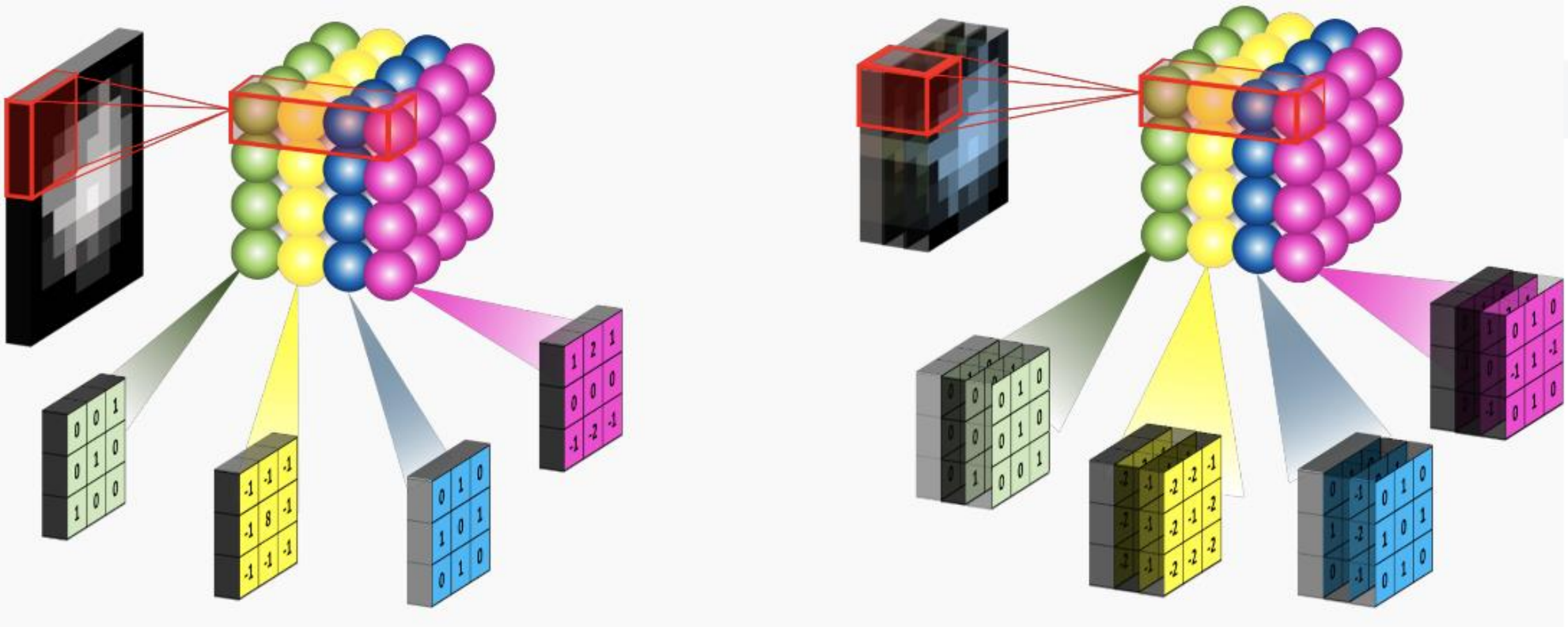
3D - ConvLayer

› Grouped Convolutions (Not true 3D Convolution)



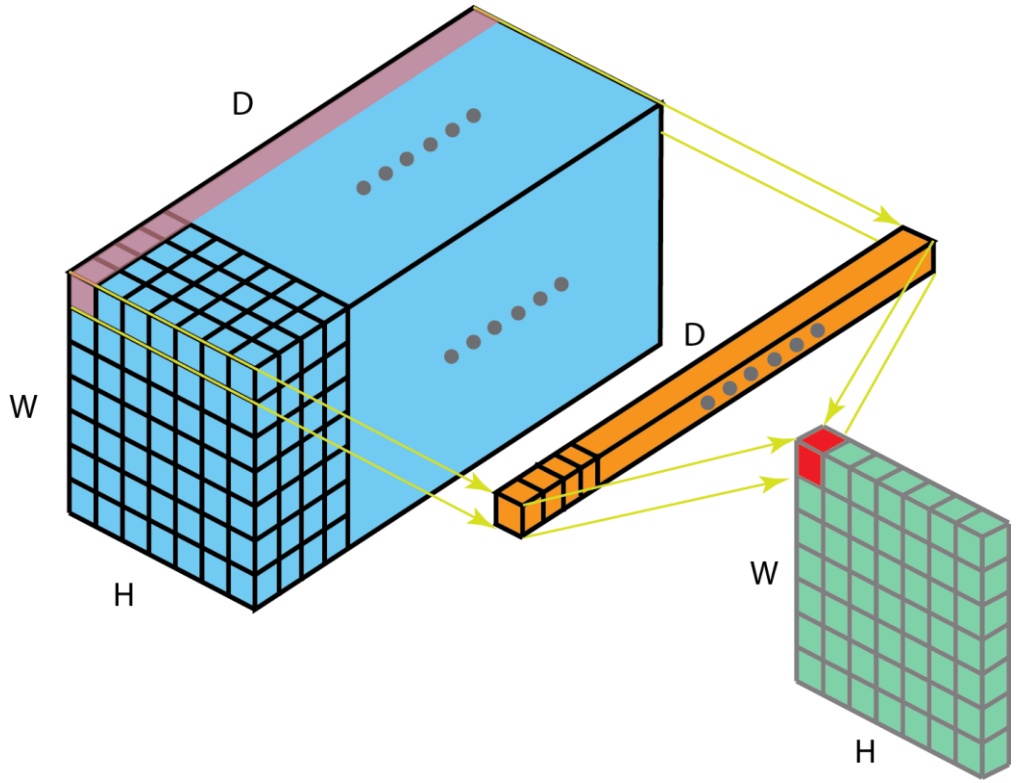
3D - ConvLayer

› Grouped Convolutions (Not true 3D Convolution)



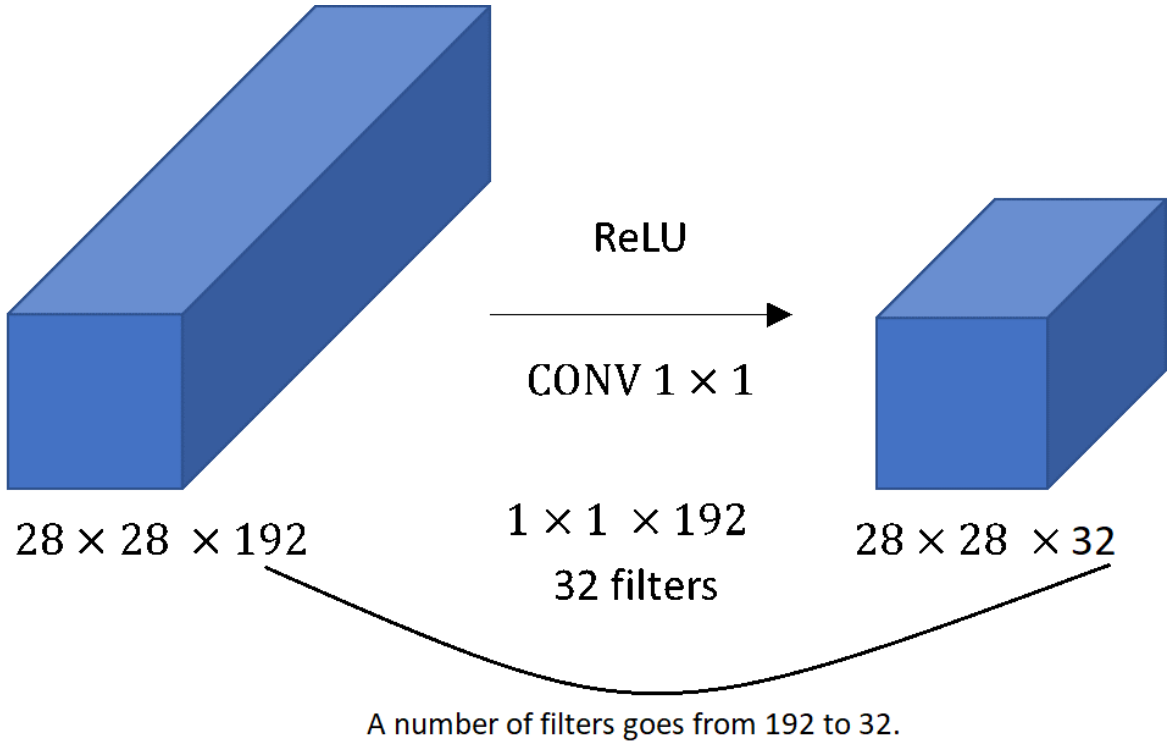
1D - ConvLayer

› 1D or 1×1 (Single) Convolutions



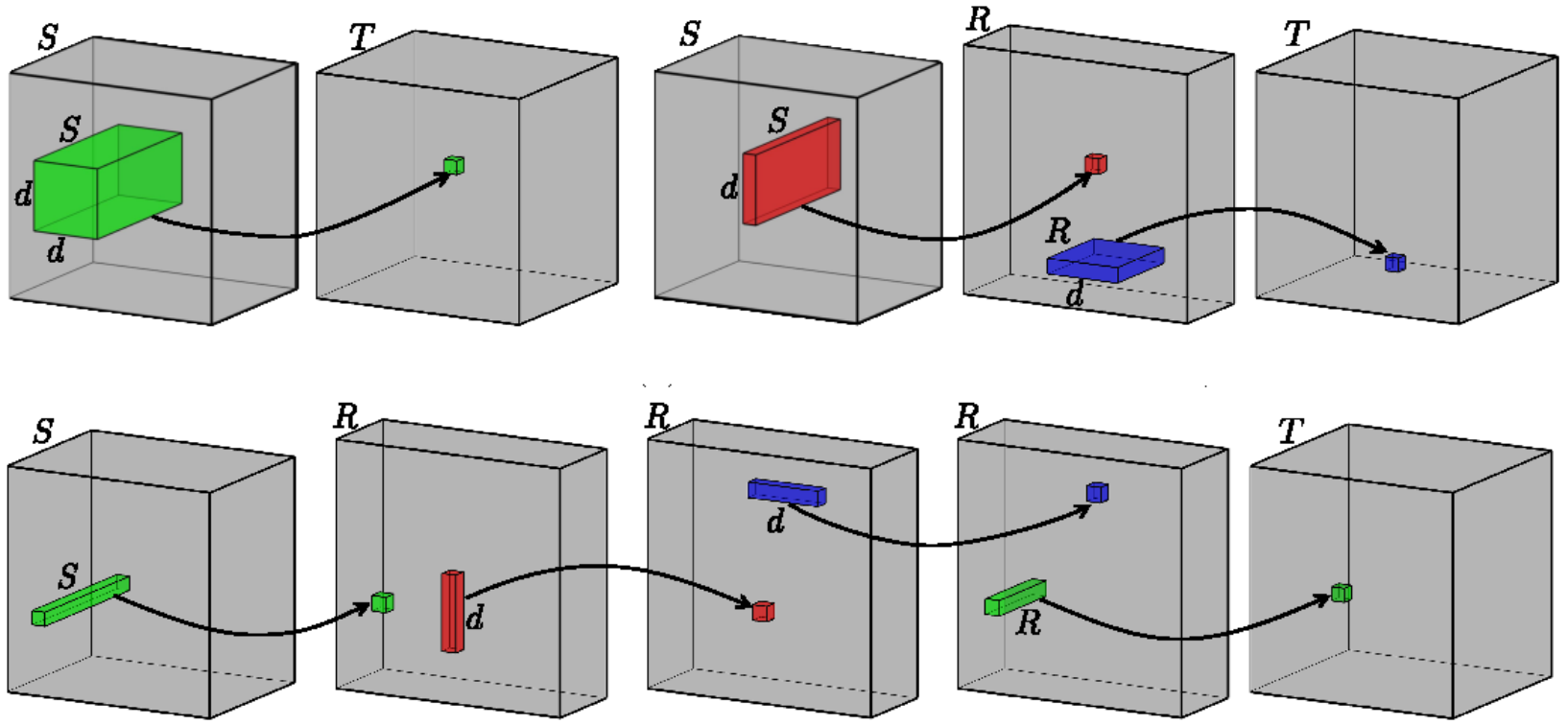
1D - ConvLayer

› 1D or 1×1 (Grouped) Convolutions, Channel Reduction

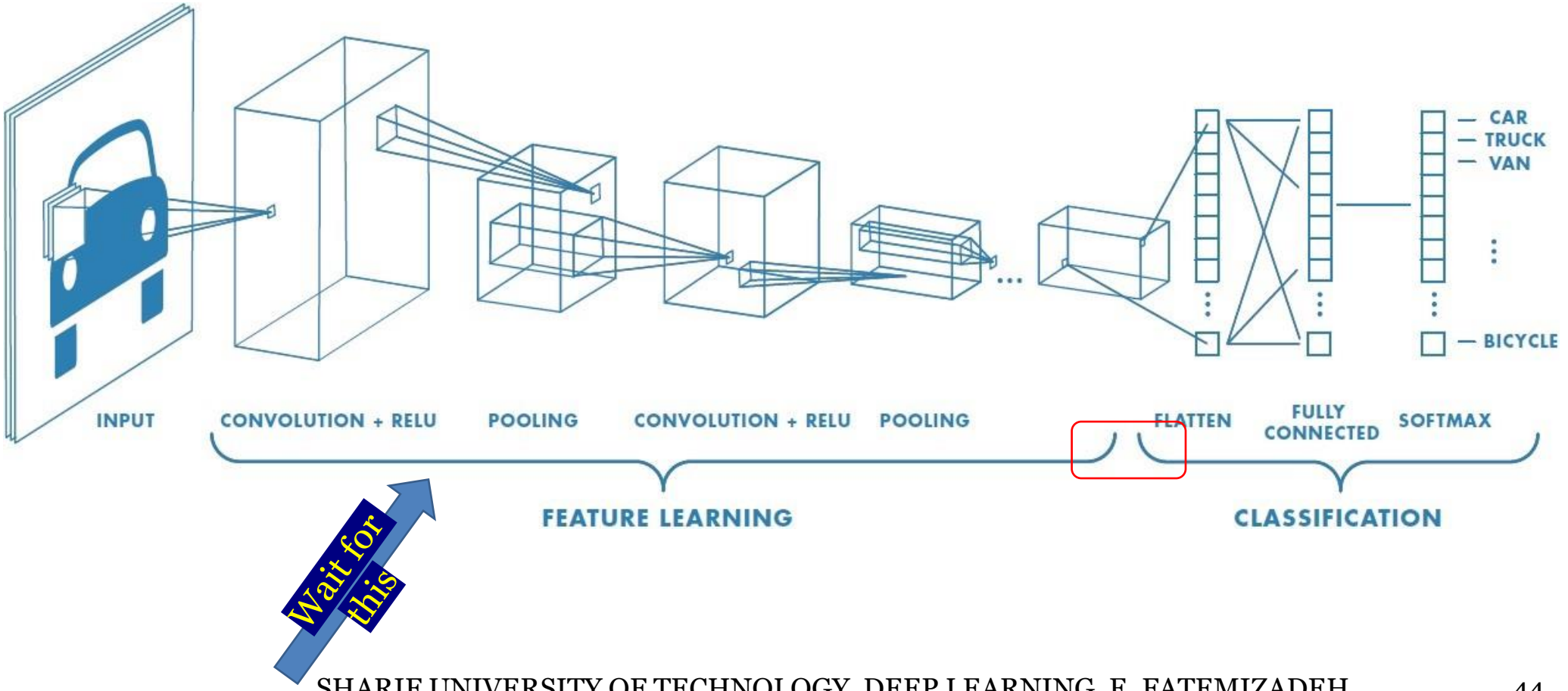


Variation

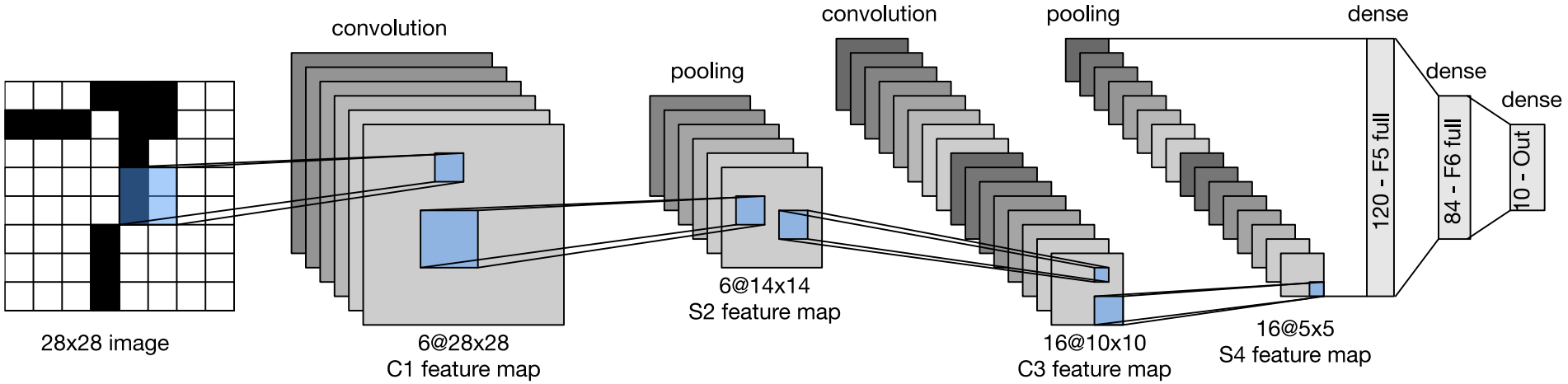
› Higher Order (Complete or Not)



A Typical CNN

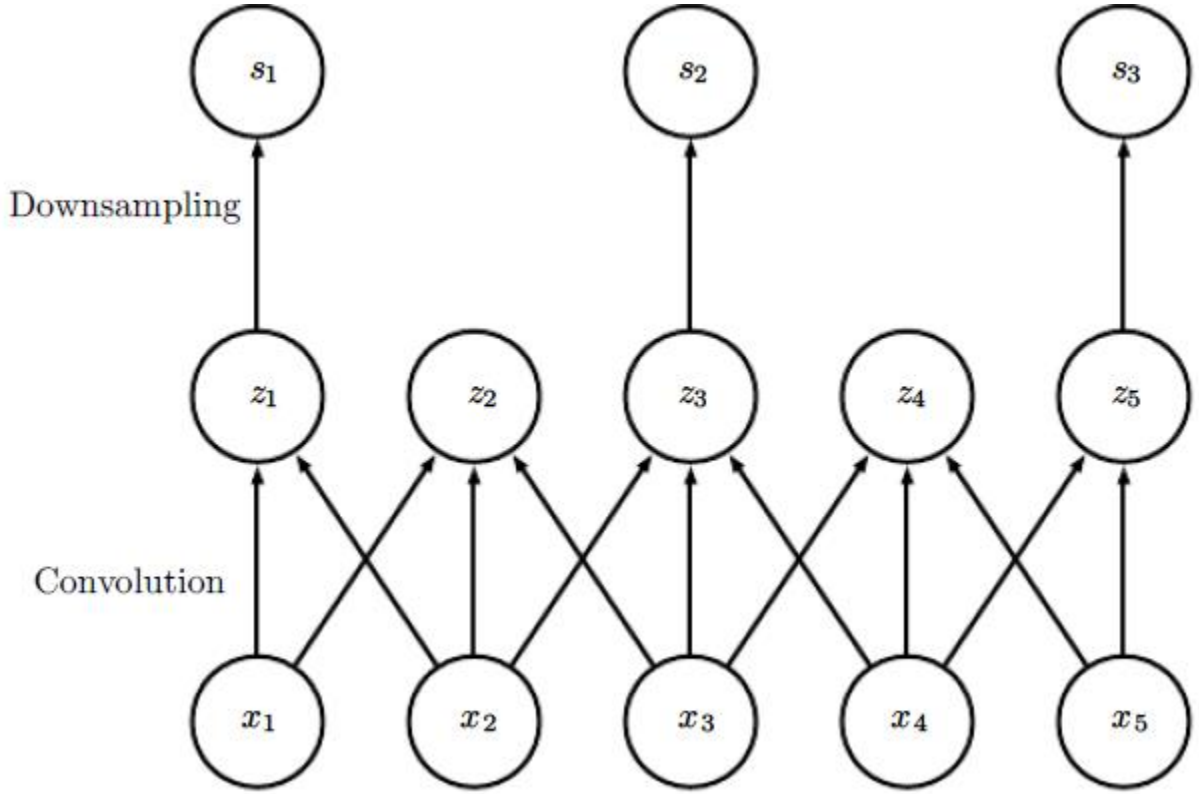


An Old CNN: LeNet (LeCun, 1998)



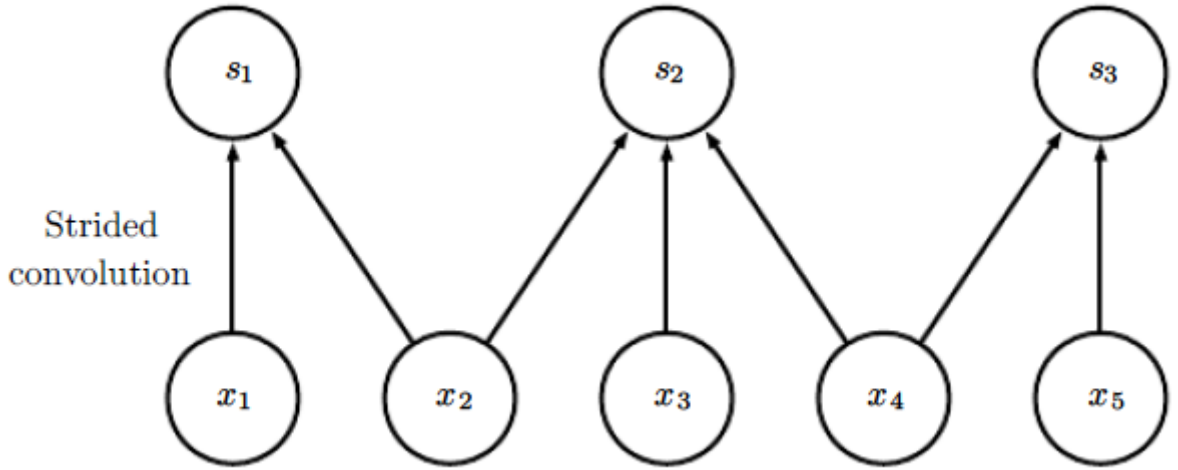
Variation

› Convolution and Downsampling:



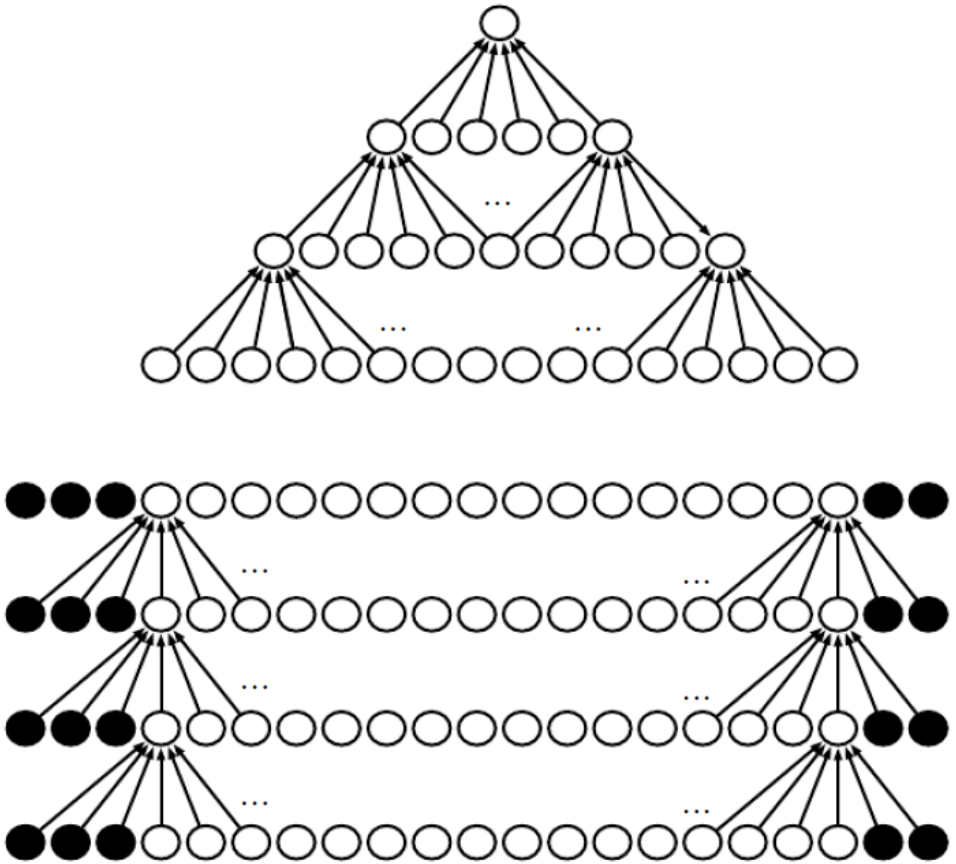
Variation

› Convolution and Downsampling:



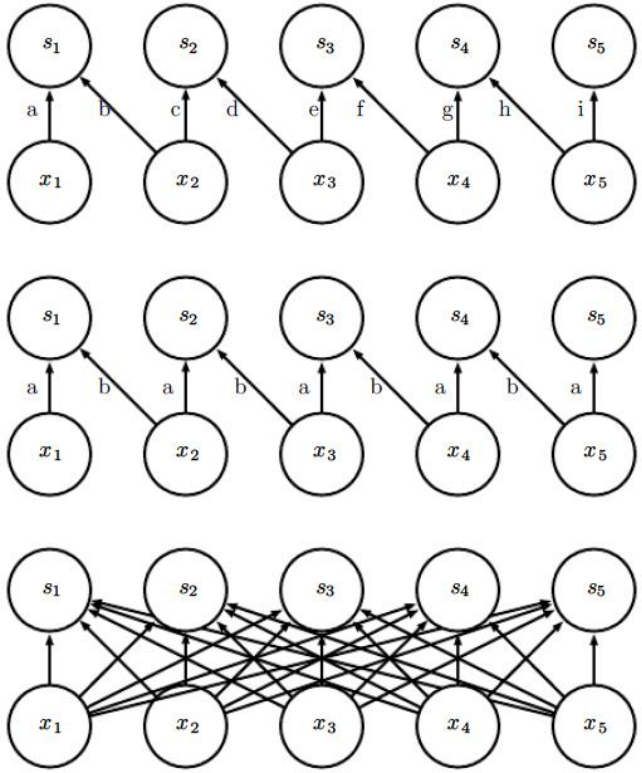
Variation

› Zero Padding (Size effect)



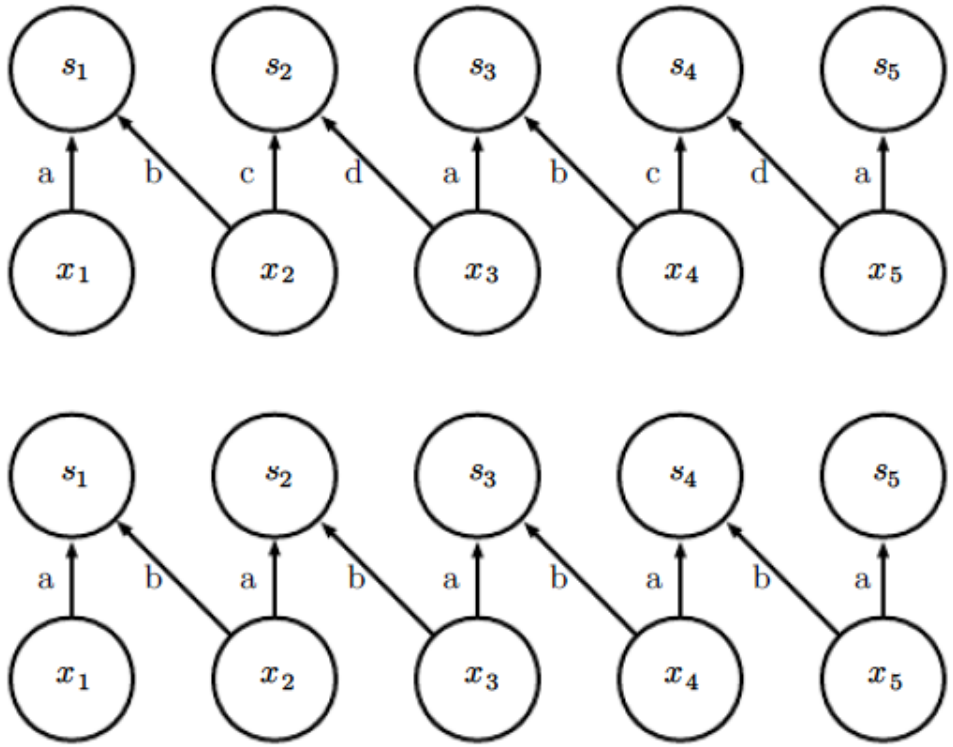
Variation

- Unshared Convolution (Shift Variant Convolution)
  - Each Node has its own weights (Locally Connected Layer)



Variation

› Tiled Convolution:
  – Two or More Kernel in each Layer

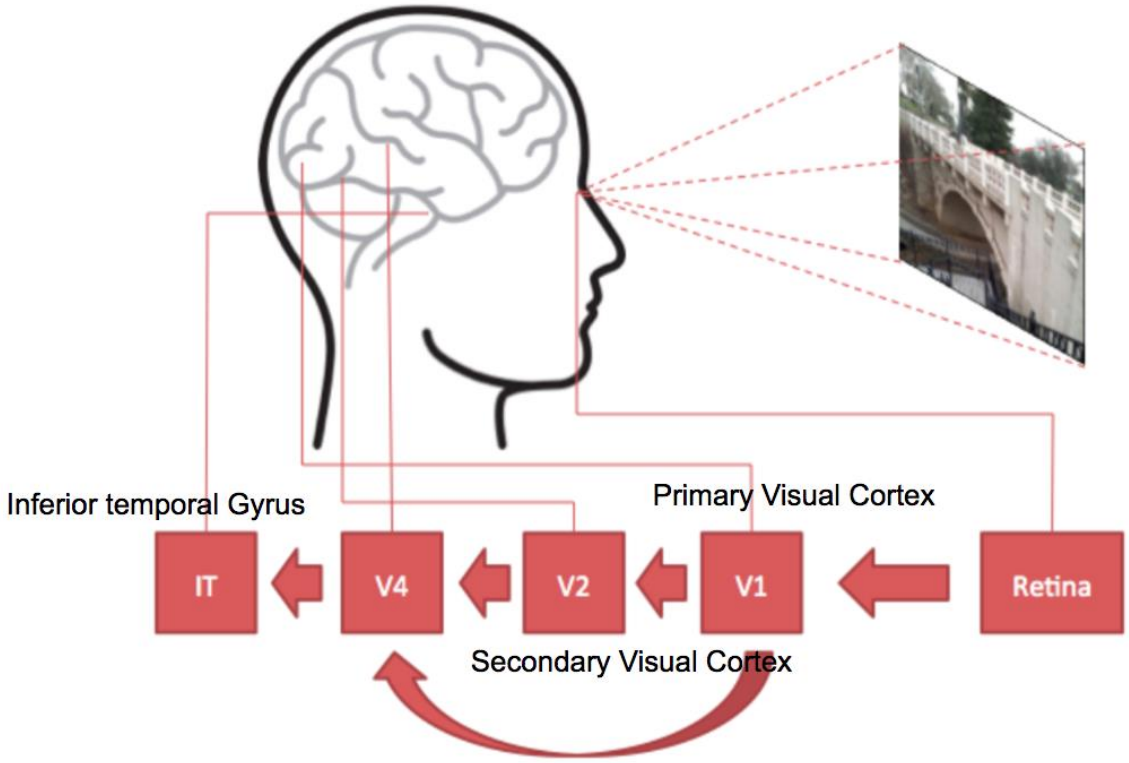


CNN History

- 1959: Neurobiological Experiments (Layers, Receptive Field, ….)
- 1980-1999: Neocognitron (Fukushima), ConvNet (LeCun)
- Early 2000: Winter Sleep!
- 2006-2011: Resurrection of CNN
- 2012-2014: Rise of CNN
- 2015-Present: Rapid Architectural Innovations and Applications

The Neuroscientific Basis

› Primary Visual Cortex, Theory: from 1959-1985



V1: Edge detection, etc.

V2: Extract simple visual properties (orientation, spatial frequency, color, etc)

V4: Detect object features of intermediate complexity

TI: Object recognition.

The Neuroscientific Basis

› V1 Cell in Primary Visual Cortex, Model by Gabor Transform

$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y). \tag{9.15}$$

Specifically, $w(x, y)$ takes the form of a Gabor function:

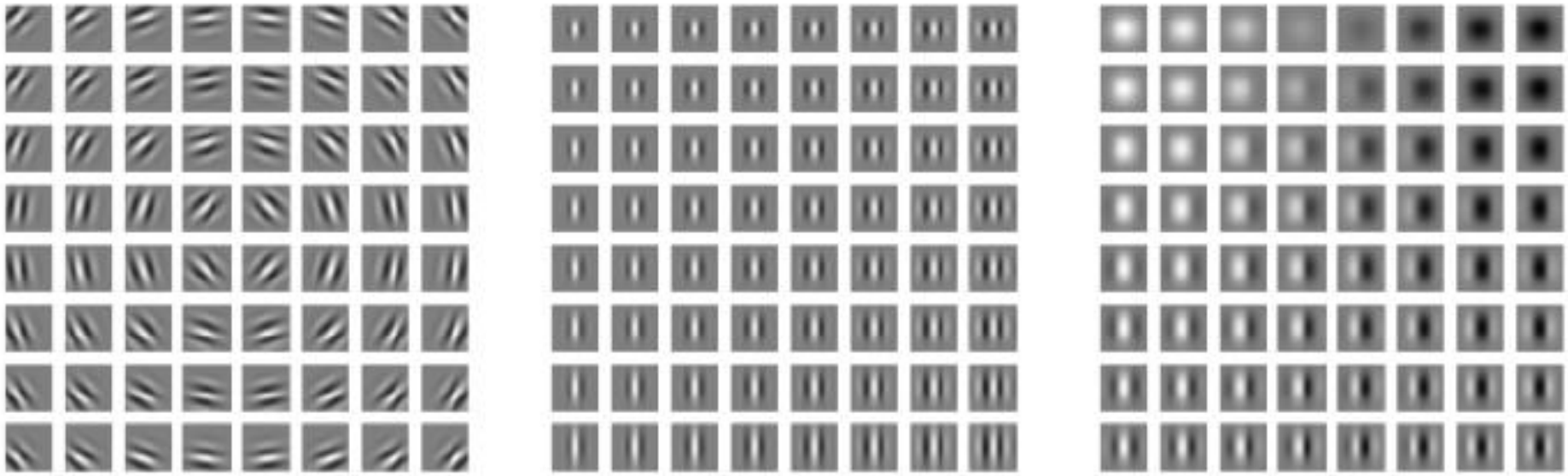
$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp\left(-\beta_x x'^2 - \beta_y y'^2\right) \cos(f x' + \phi), \tag{9.16}$$

where

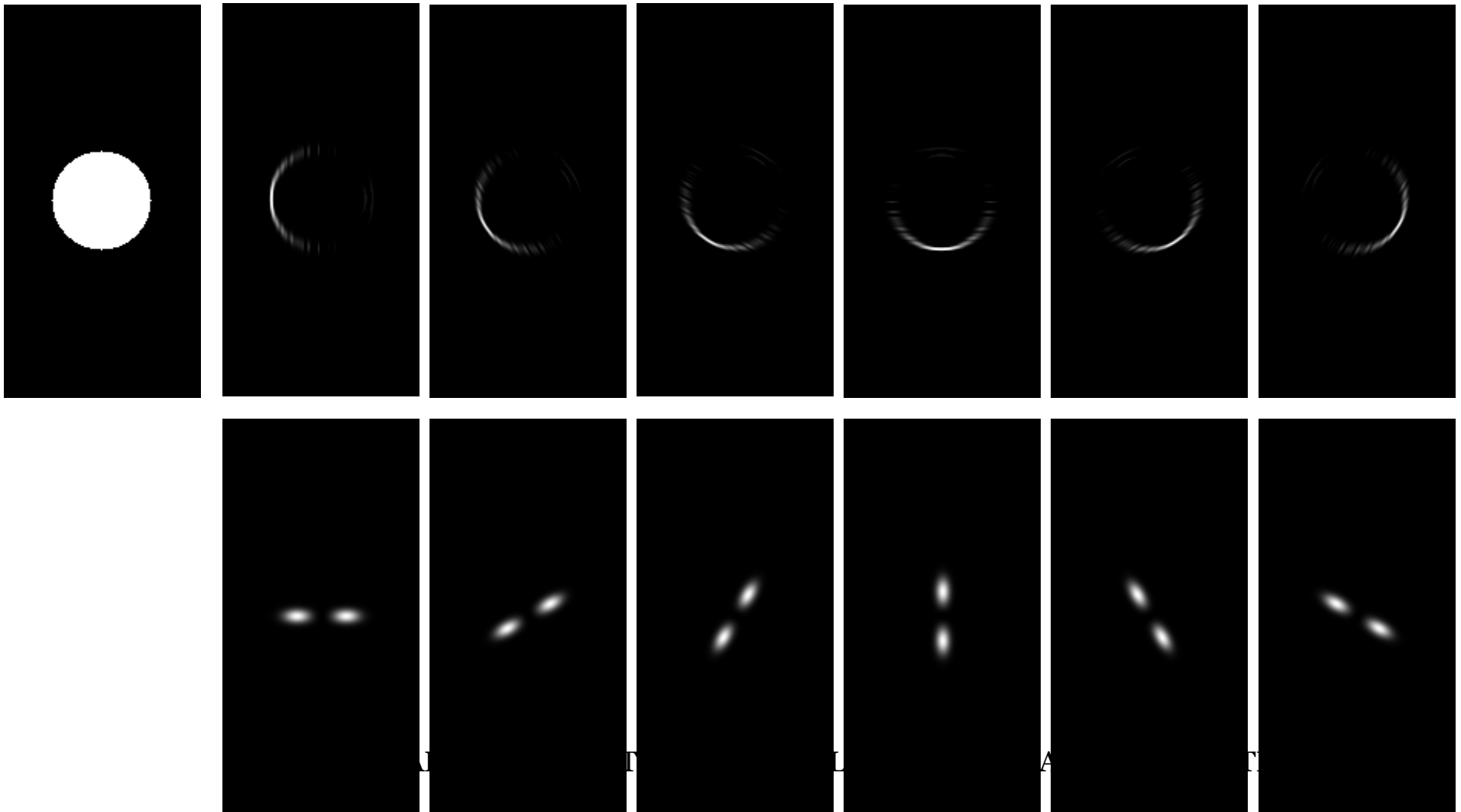
$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau) \tag{9.17}$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau). \tag{9.18}$$

Gabor Function



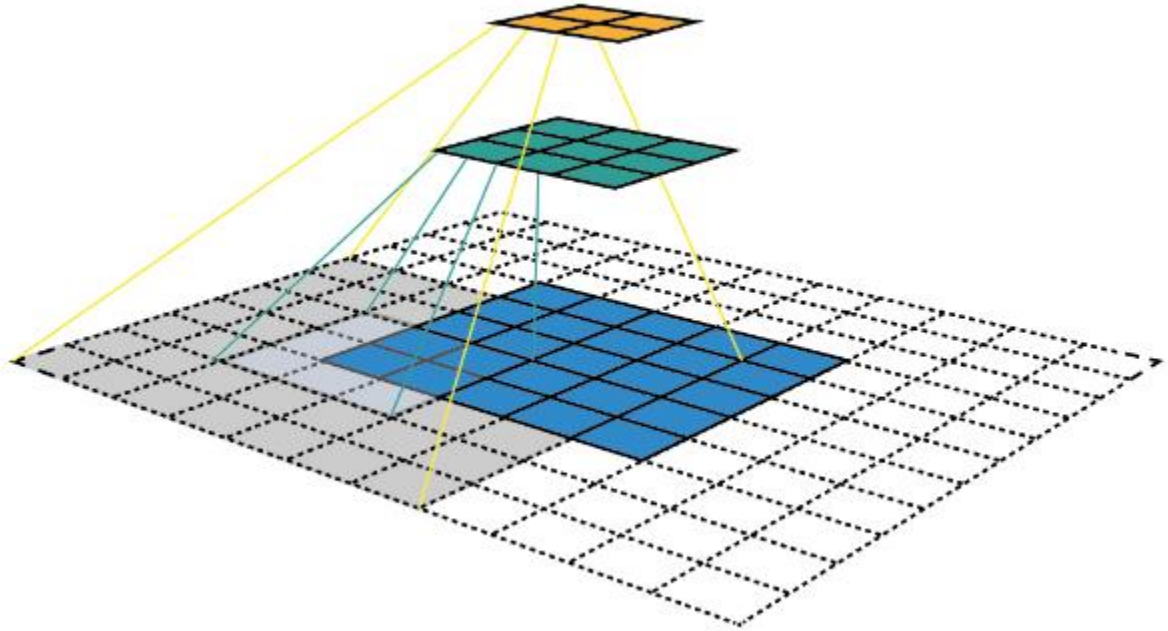
Gabor Filter Bank



Receptive Field in CNN

› Definition:

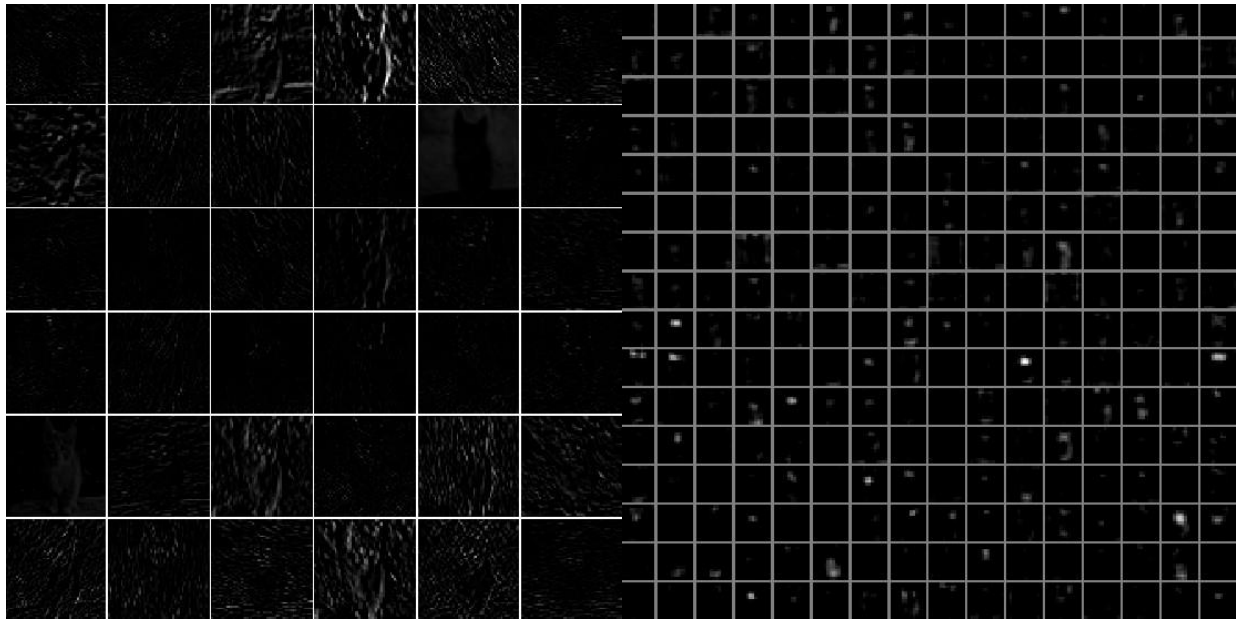
The region in the input space that a particular CNN's feature is looking at (or affected by)



What *Feature Layers* Learn

› Visualizing Strategies: #1 Visualize the activations of the network in response to an input

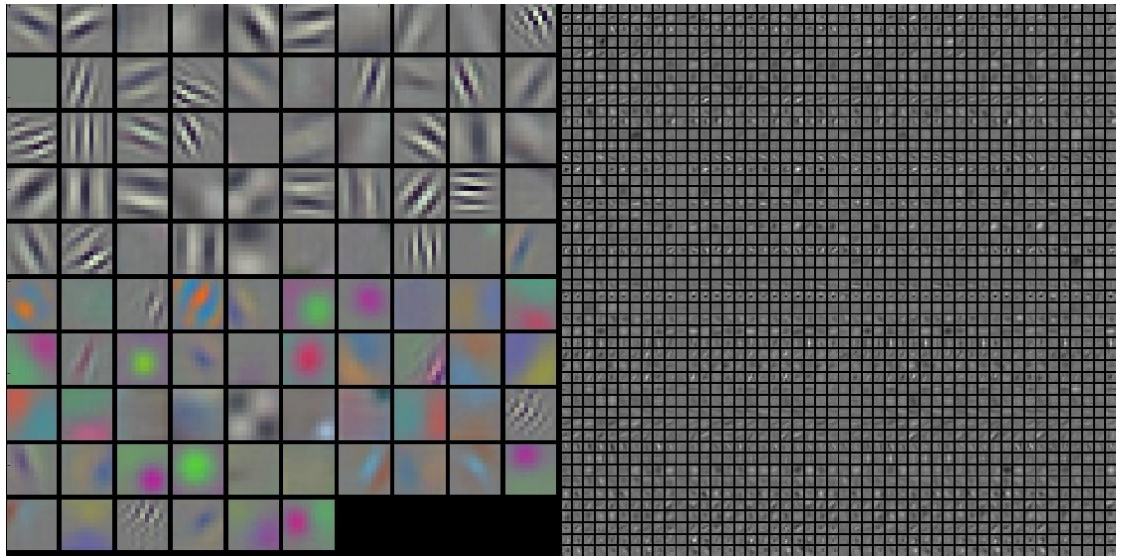
- Input is Cat!
- Layer #1/#5 of AlexNet
- Map is Sparse



What *Feature Layers* Learn

› Visualizing Strategies: #2 Visualize the weights (filters)

- 1st layer is informative
- Layer #1/#2 of AlexNet

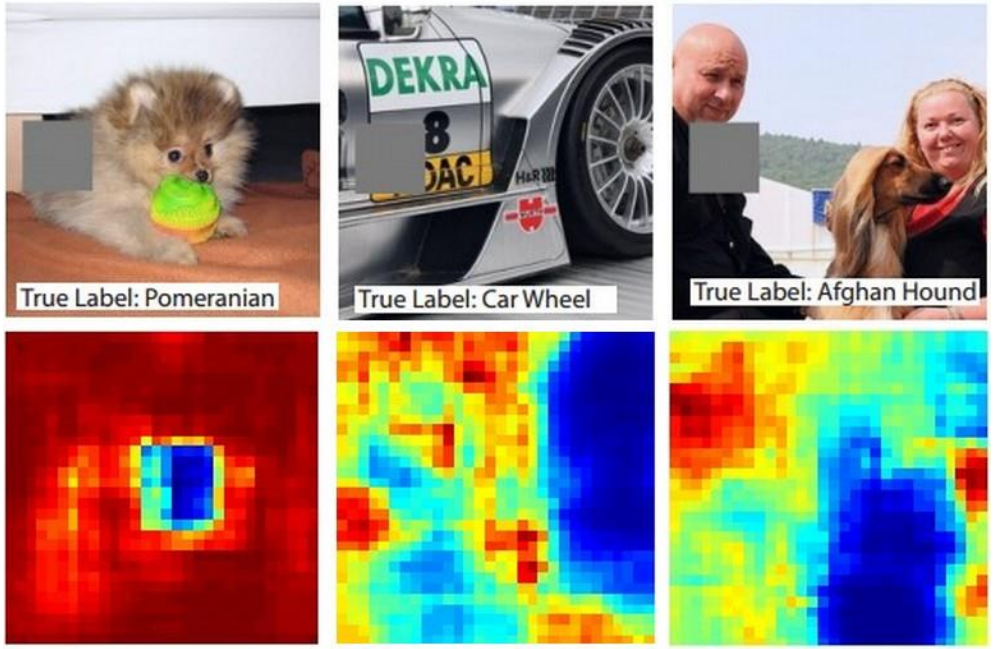


What *Feature Layers* Learn

› Visualizing Strategies: #3 visualize the images to get an understanding of what the neuron is looking for in its receptive field (5^{th} Layer of AlexNet)

What *Feature Layers* Learn

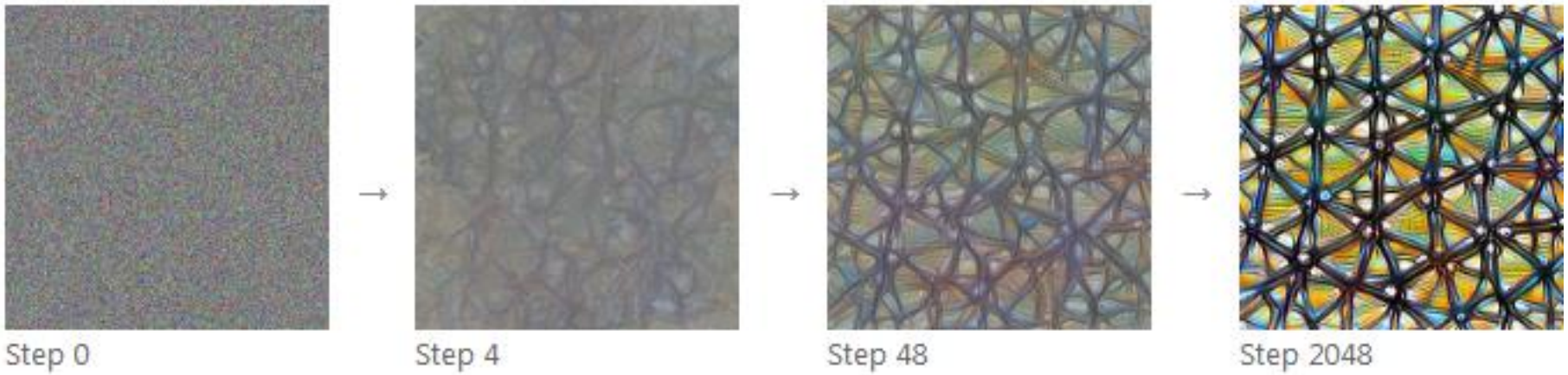
› Visualizing Strategies: #4 Cover (by a p×p square) the input and plot the prediction probability as function of covering square



What *Feature Layers* Learn

› Visualizing Strategies: #5 Optimize input to activate a particular Neuron/Layer/….

› For Example: Maximizing (Neuron #11, Layer mixed4a,GoogleNet):



› See: https://distill.pub/2017/feature-visualization/appendix/

What *Feature Layers* Learn

› Visualizing Strategies: #5 Optimize input to activate a particular Neuron/Layer/….

› A DNN paints (*DeepDream* by Google)

Baseball—or stripes?
mixed4a, Unit 6

Animal faces—or snouts?
mixed4a, Unit 240

Clouds—or fluffiness?
mixed4a, Unit 453

Buildings—or sky?
mixed4a, Unit 492

What *Feature Layers* Learn

- Visualizing Strategies: #5 Optimize input to activate a particular Neuron/Layer/….
- A DNN paints (*DeepDream* by Google)




What *Feature Layers* Learn

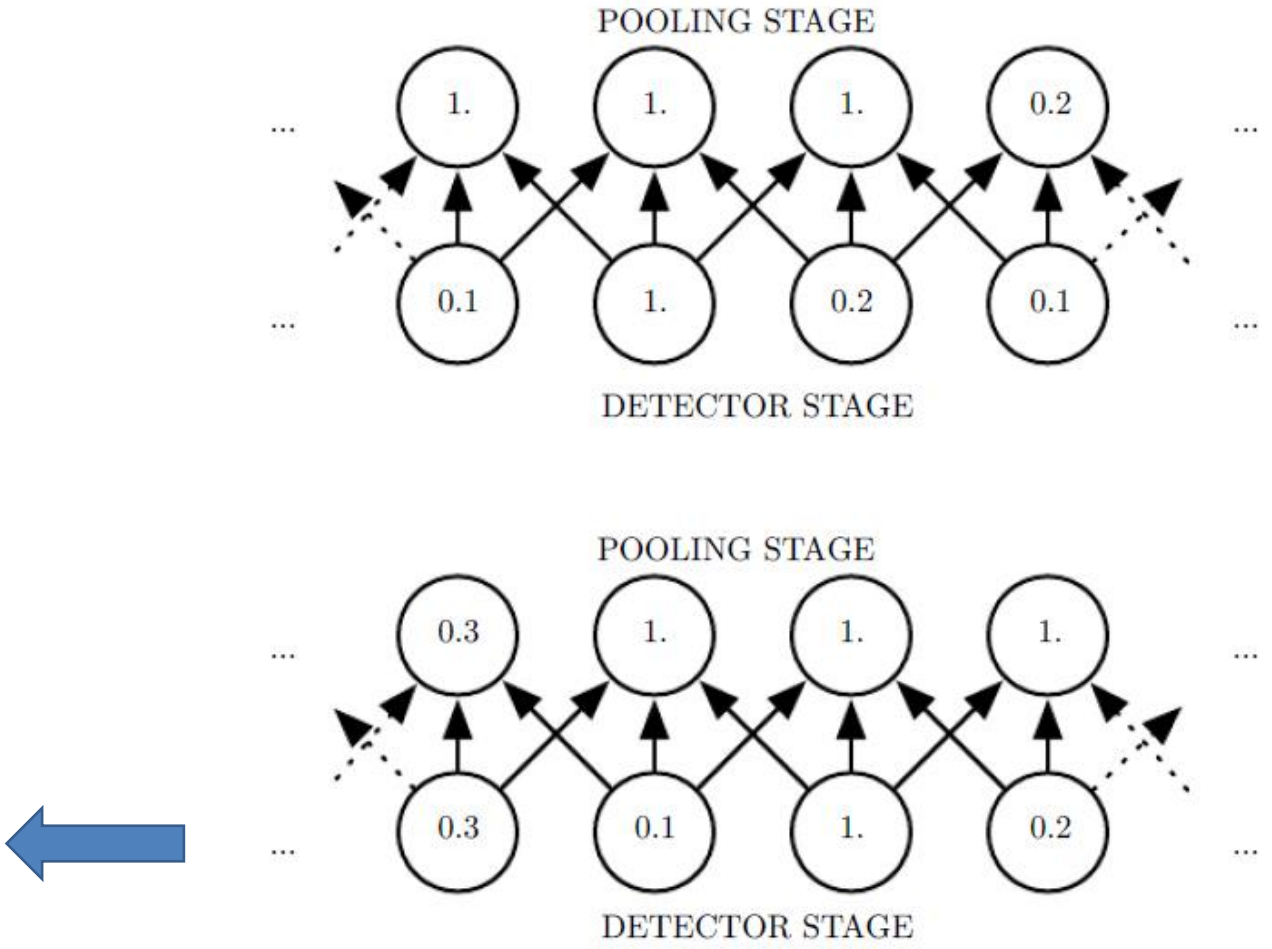
› Layers Learn High Level Features
  – Top: Nine 7x7 filters in the 1st ConvLayer
  – Bottom: grid of 7x7 patches from actual images which maximally activated each of the 9 filters.

Layer 1

Operators - Pooling

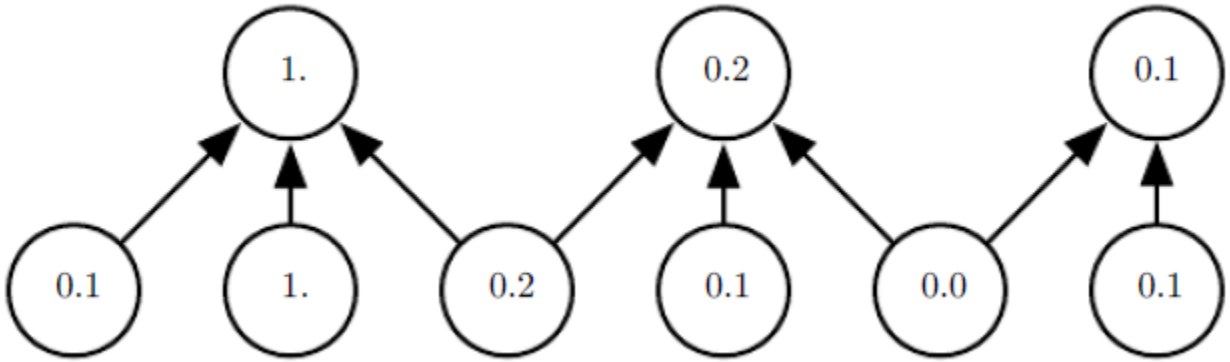
- Modify and Subsample output of detector (ReLu):
  - Using neighbors
    - Max pooling
    - Means pooling
    - Median pooling
    - Lx-Pooling
    - Weighted Mean pooling
  - Make Invariant to small variation (like shift)
  - Reduce # of parameters
  - Regularization

Example – One Pixel Shift



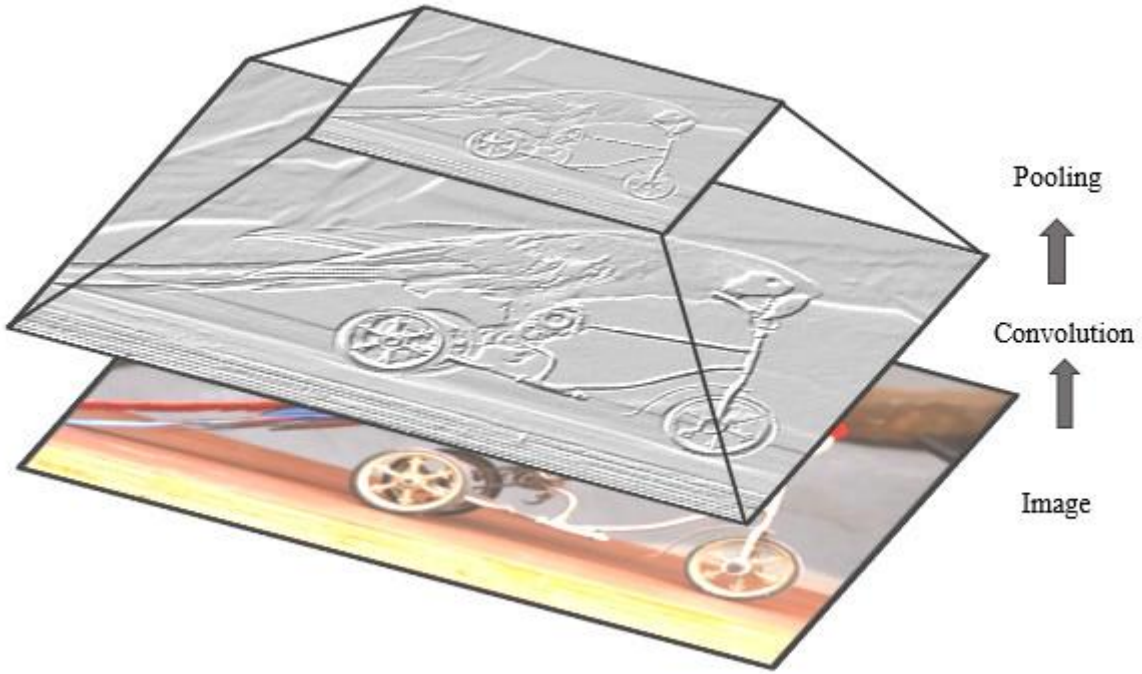
Pooling with Downsampling

› With or Without overlap/stride Pooling



Pooling Layer

› Pooling Effect



Pooling Strategy

- Deterministic
- Stochastic:

Pooling Strategy - Deterministic

› Max pooling, idea!

$$a_{kij} = \max_{(p,q)\in R_{ij}} (a_{kpq})$$

› Mean pooling:

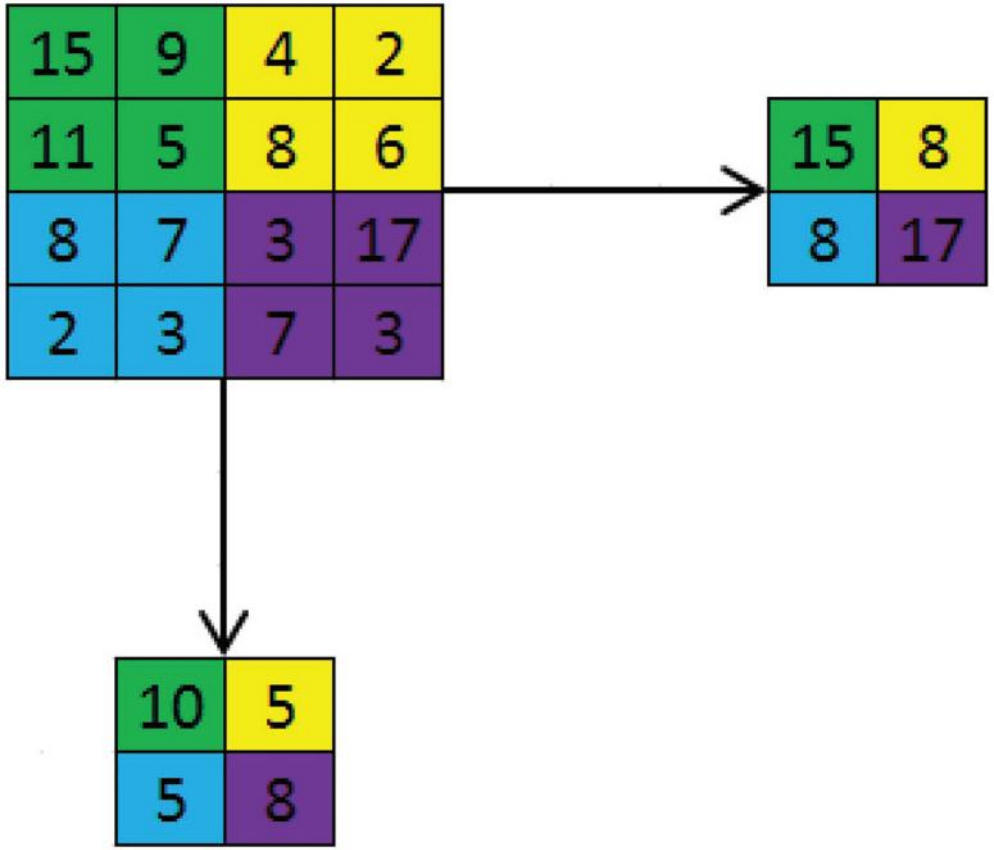
$$a_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} a_{kpq}$$

› Mixed pooling (a random parameter):

$$a_{kij} = \lambda \cdot \max_{(p,q)\in R_{ij}} (a_{kpq}) + (1-\lambda) \cdot \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} a_{kpq}$$

Pooling Strategy - Deterministic

› Max/Mean Example



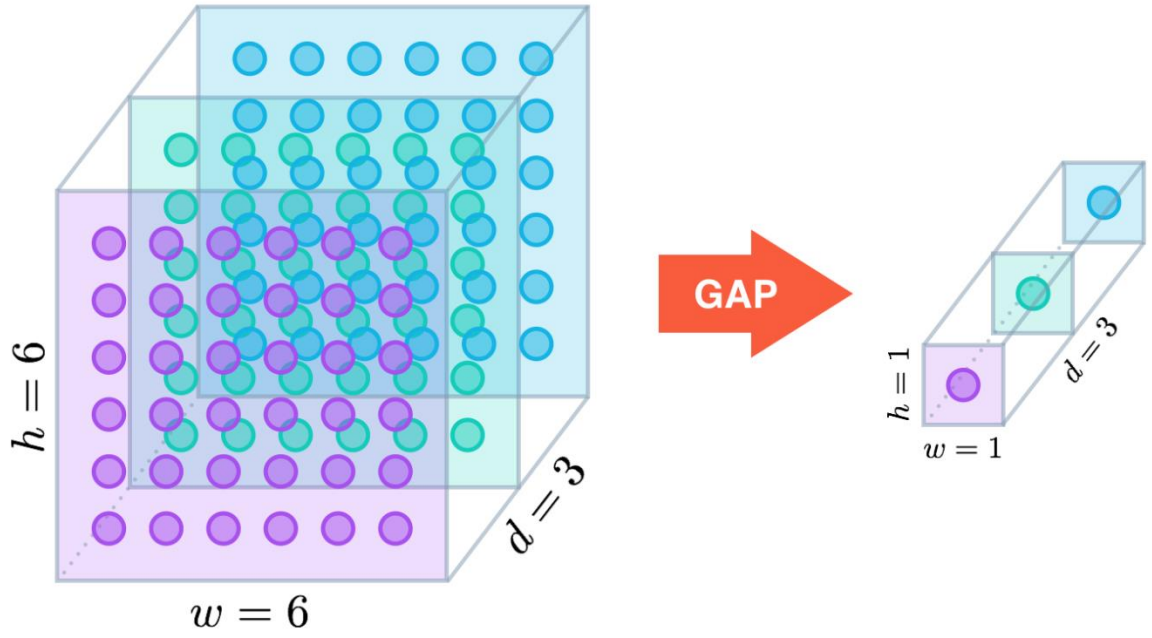
Pooling Strategy - Deterministic

› (Discrete Cosine Transform Spectral Pooling Layers for Convolutional Neural Networks)-ICAISC 2018

› Spectral Pooling:
  – c: # of channels,
  – Input: $X \in \Re^{C \times m \times n}$
  – Output: $\hat{X} \in \Re^{C \times p \times q}$
  – $Y = Transform(X) \in \Re^{C \times m \times n}$, Transform: DFT/DCT/Wavelet/...
    › May be *Transform* and *filter*!
  – $ProperTruncate\ Y\ to\ \hat{Y} \in \Re^{C \times p \times q}$
  – $\hat{X} = InverseTransfor(\hat{Y}) \in \Re^{C \times p \times q}$

Pooling Strategy - Deterministic

› Global Averaging Pooling:
  – Just before FC layer



Pooling Strategy - Stochastic

› Stochastic (Stochastic Pooling for Regularization of Deep Convolutional Neural Networks- ICLR2013)

– **Roulette wheel Pooling (RWP)**, pick a random activation from region of pooling, activity related probability:

$$P_i = \frac{a_i}{\sum_{k \in Region} a_k}$$

– To avoid some noisy behavior:

$$S_j = \sum_{i \in R_j} a_i p_i$$

Pooling Strategy - Stochastic

› Results

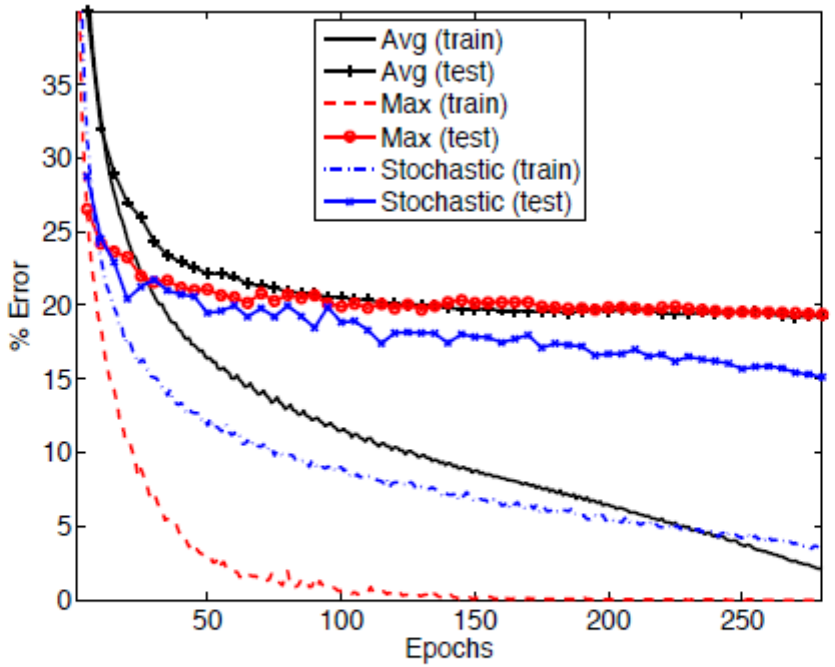


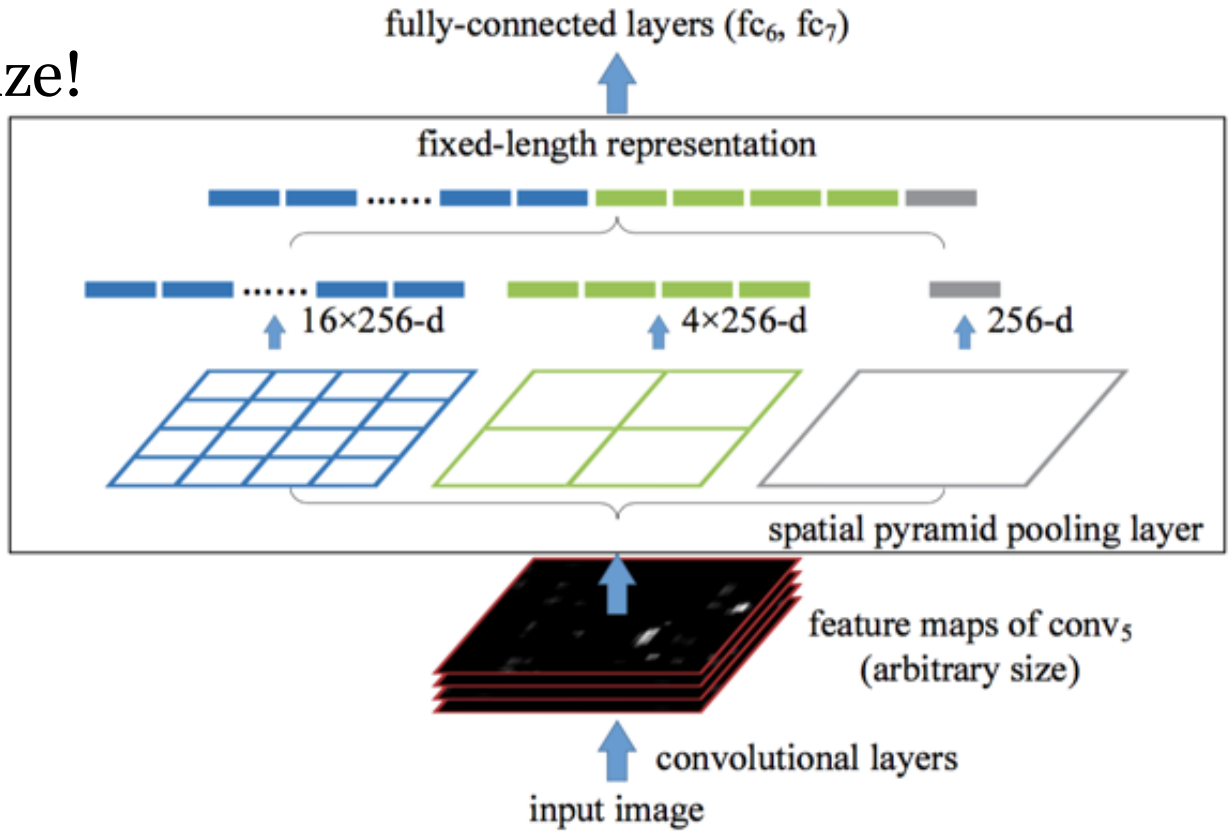
Figure 3: CIFAR-10 train and test error rates throughout training for average, max, and stochastic pooling. Max and average pooling test errors plateau as those methods overfit. With stochastic pooling, training error remains higher while test errors continue to decrease.[1]

Pooling Strategy - Stochastic

- Stochastic:
  - **Tournament pooling**: Take any two (tournament size) nonzero activations with *equal* probability, and select larger one.
  - **Fractional Max Pooling (FP):** Random Region size and Location!

Pooling Strategy - Spatial Pyramid Pooling

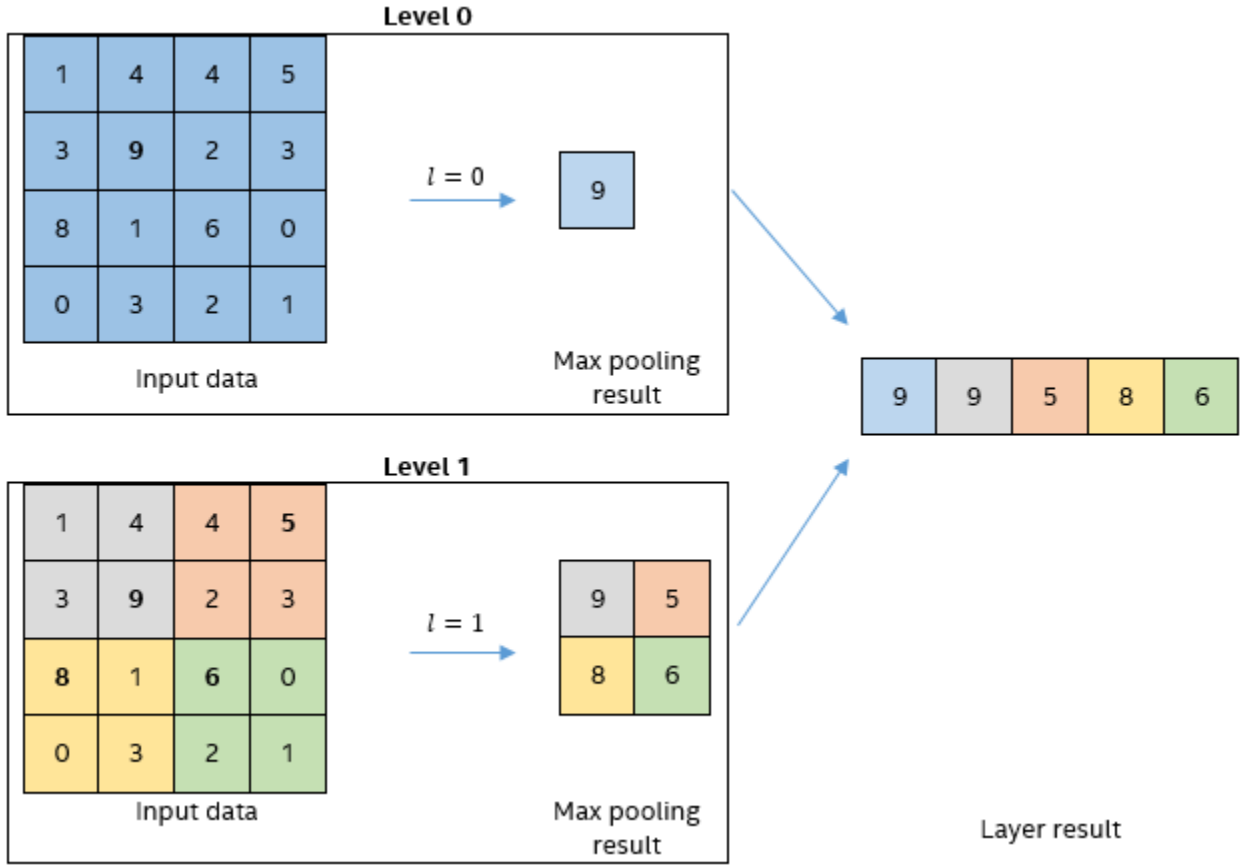
› Spatial Pyramid Pooling
  – 256 is # of filters
  – FCL has fixed input size!



Pooling Strategy - Spatial Pyramid Pooling

https://software.intel.com/sites/products/documentation/doclib/daal/daal-user-and-reference-guides/daal_prog_guide/GUID-97371455-C579-4E7B-BDBB-5274BC2F338C.htm

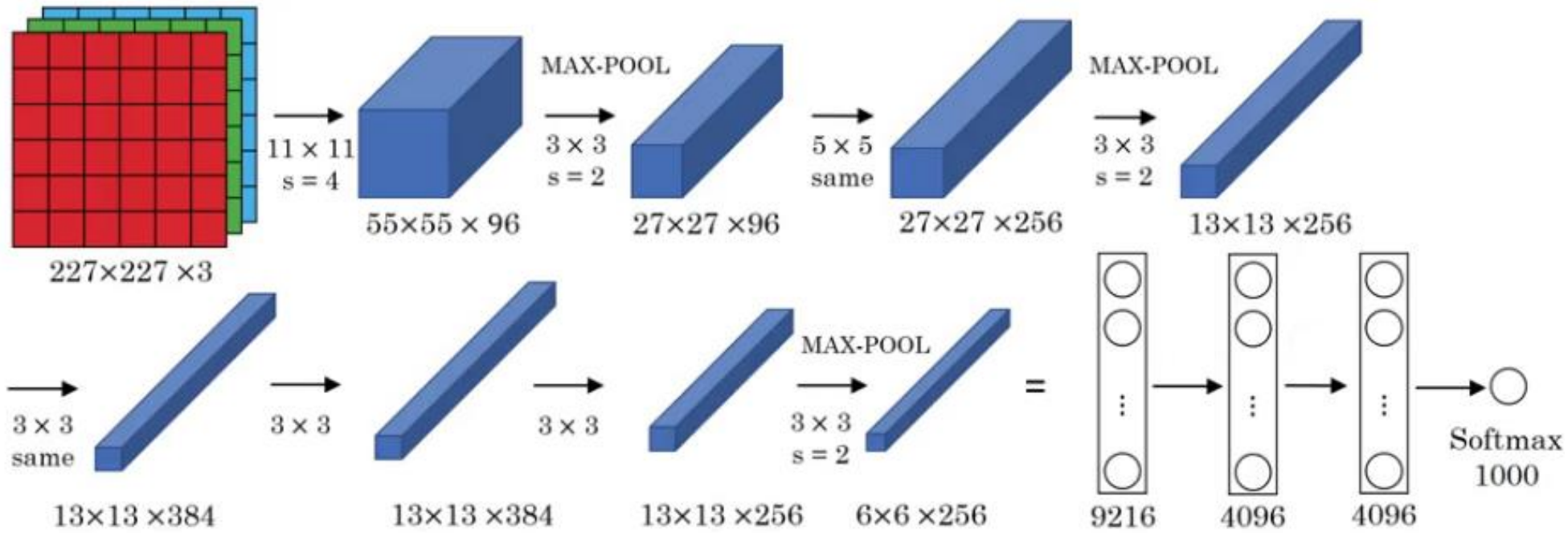
› Spatial Pyramid Pooling



A Taste! AlexNet

› AlexNet

– 1st Layer: (11+4*(k-1))=227 ➔ k=55

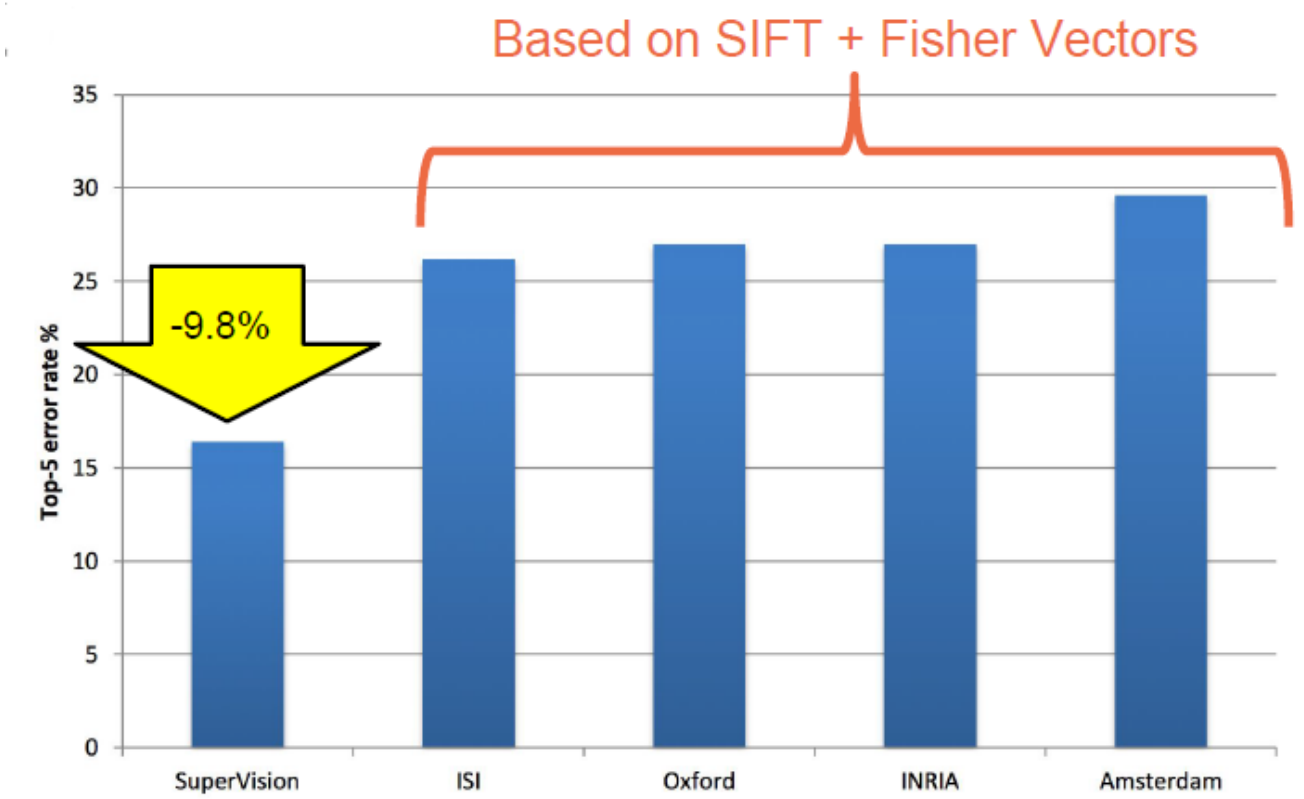


A Challenging Problem: IMAGENET!

- **1000 Classes**
- **Train: 1.2 M**
- **Test: 100 K**

A Challenging Problem: IMAGENET!

› ImageNet ILSRVC 2012:
(https://cs.nyu.edu/%7Efergus/pmwiki/pmwiki.php)



AlexNet, Alex Krizhevsky et al. 2012

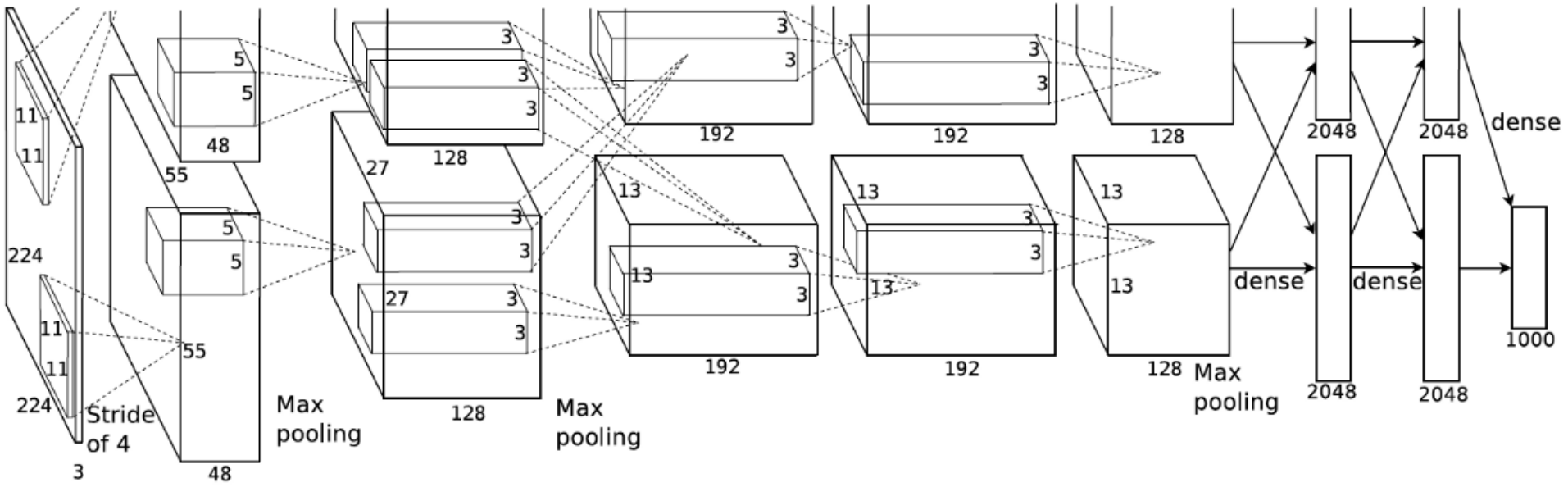
- In Brief:
  - ReLU (x6 faster than tanh)
  - Regularization: 50% Dropout (x2 training time)
  - 5 CL (Convolution Layer), 11×11, 5×5, 3×3
  - 3 FC (Fully Connected), 4096-4096-1000
  - 3 MaxPool Layer: 3×3
  - ReLu after all CL and FC
  - Image Size: 256x256
  - Input Size: 227x227x3

AlexNet, Krizhevsky et al. 2012]

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

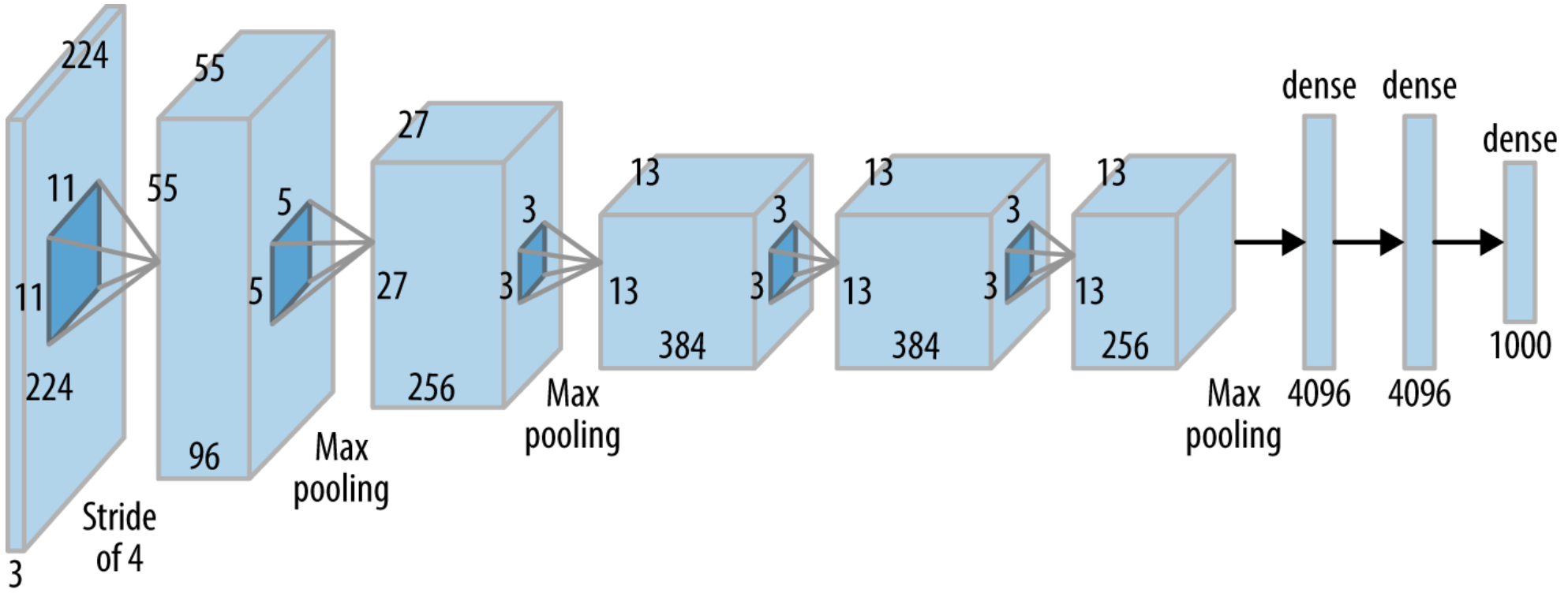
AlexNet, Krizhevsky et al. 2012]

› Original (paper) Figure:

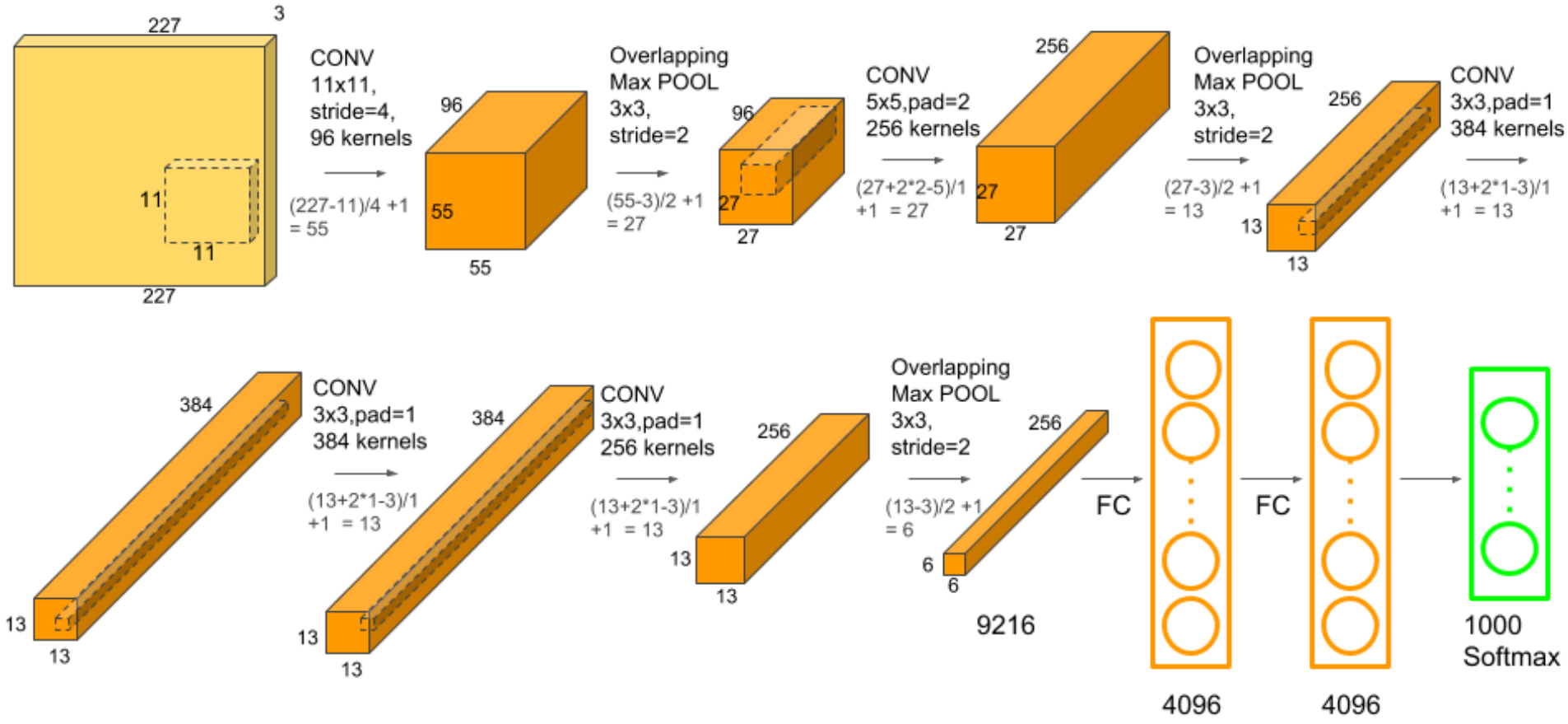


AlexNet, Krizhevsky et al. 2012]

› Better one:



AlexNet 2012 (https://www.learnopencv.com/understanding-alexnet/)



AlexNet 2012 -Details

- Detail:
- First use of ReLU
- Normalize ReLu Output !!!
- Several Data Augmentation
- Dropout: 0.5
- Batch Size: 128
- SGD Momentum: 0.9
- Learning rate 1e-2, reduced by 10
- ~60M parameters

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

AlexNet 2012 – Data Augmentation

- Random patches 224x224 (227x227 is true) from 256x256 input images
- Horizontal Reflection of all patches
- Add AGWN (Additive Gaussian White Noise) as follow:
  - PCA on RGB (Training set):

$$n_1\lambda_1\mathbf{v}_1 + n_2\lambda_2\mathbf{v}_2 + n_3\lambda_3\mathbf{v}_3$$

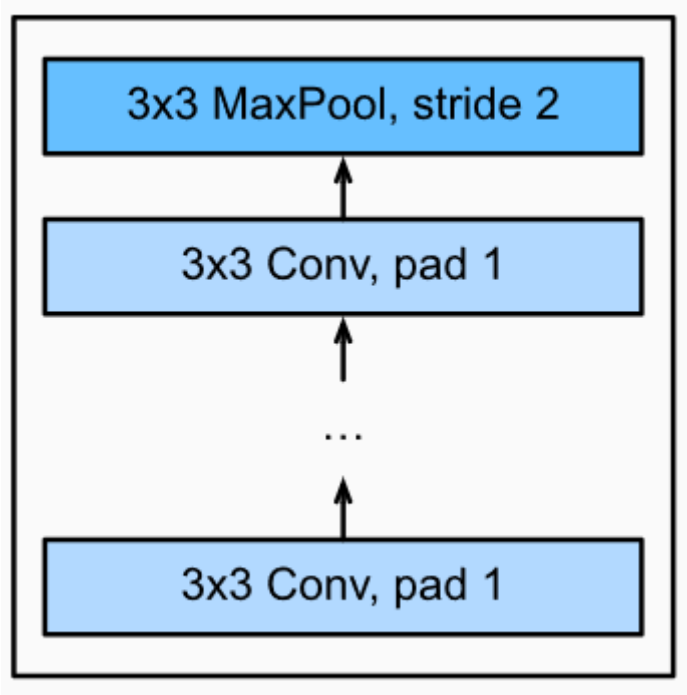
$$n_i \propto N(0, 0.1)$$

VGGnet 16&19, Simonyan & Zisserman, 2014

- VGG: **V**isual **G**eometry **G**roup
- Idea: Smaller filter but deeper!
  - 8 Layers in Alexnet → (16-19) ConvLayers
  - ConvLayer (16/19): Only 3×3, Stride: 1, Padding: 1
    - More Non-linearity
  - MaxPooling (5): 2×2, Stride: 2
  - 138M Parameters
  - 3×3 and deeper ~ 7×7 shallower
    - (3×3)*(3×3)*(3×3) ~ 7×7 (Receptive Field)

VGGnet 16&19, Simonyan & Zisserman, 2014

› VGG Main Block:

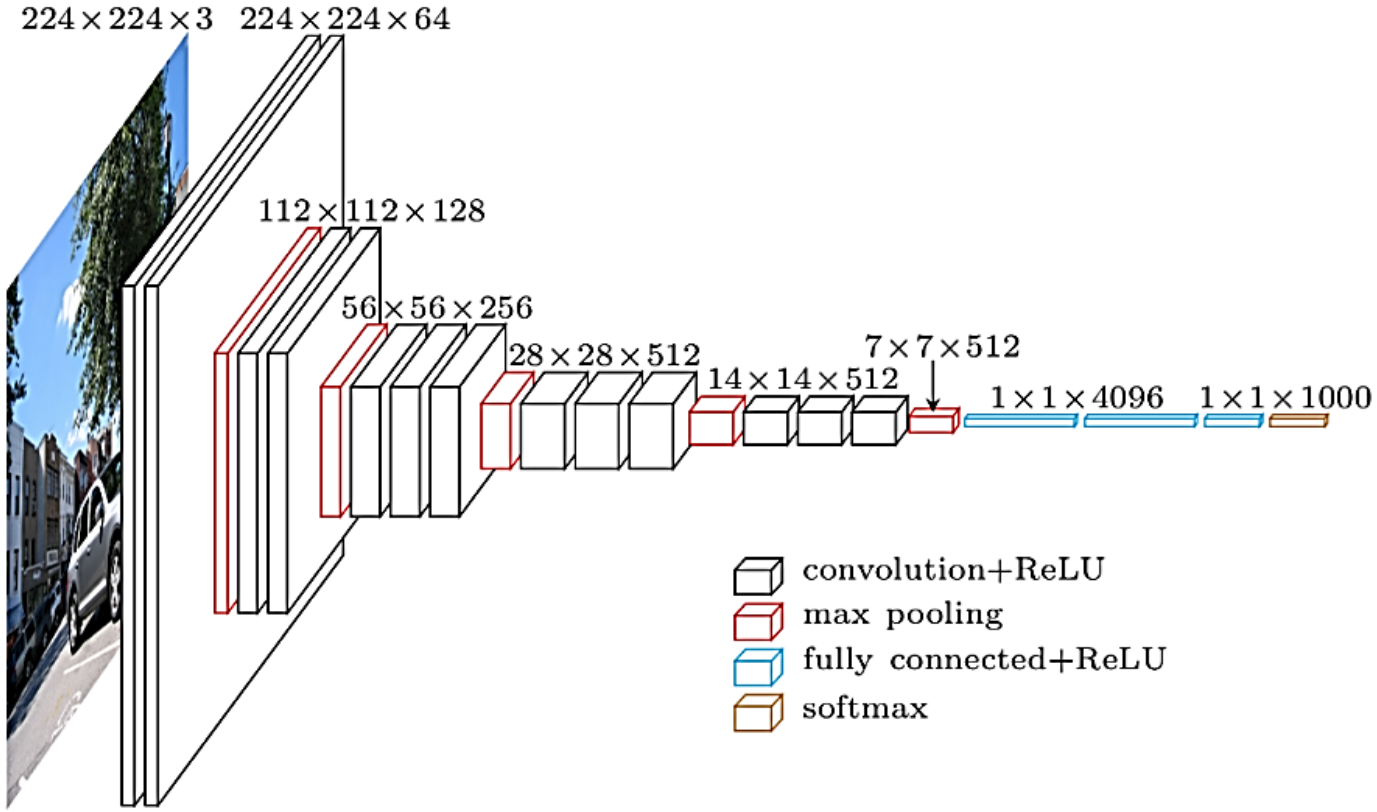


VGGnet 16, Simonyan & Zisserman, 2014

› VGG16 Structure:

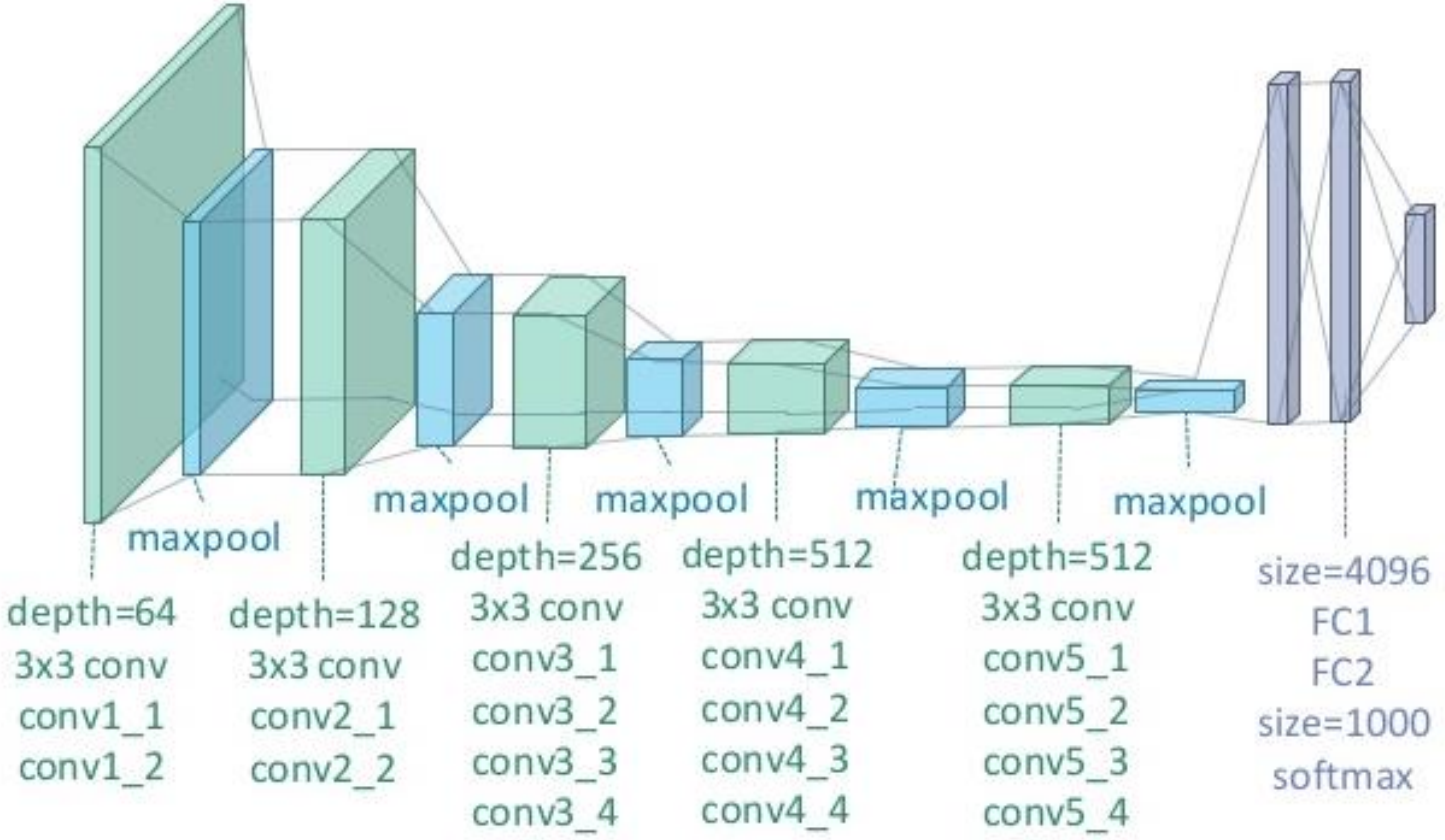
› Input to 1st FC:

– 7*7*512=25088



VGGnet 16&19, Simonyan & Zisserman, 2014

› VGG19 Structure:



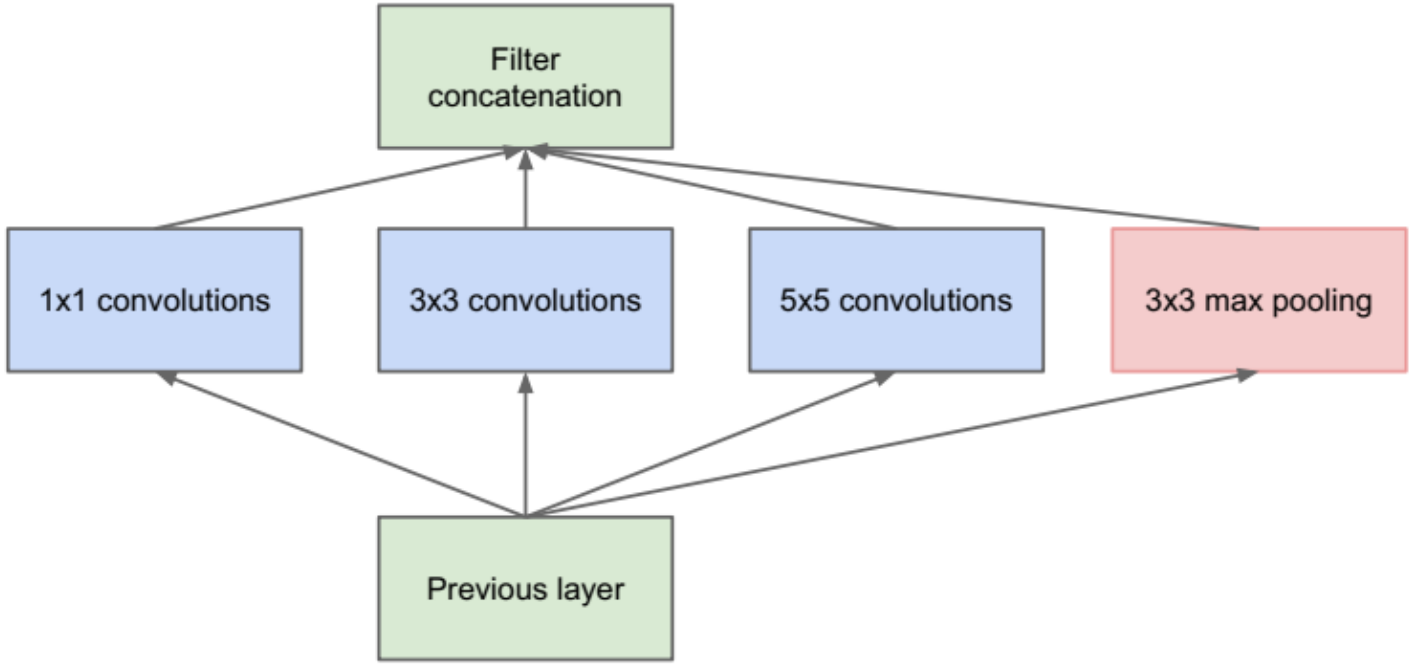
GoogLeNet - Going deeper with convolutions, 2014

- Inception Module **V1**
- Idea and Motivation:
  - Multiresolution Analysis (avoid cascade only convolution)
  - Global Average Pooling before FC (dimension reduction)
  - Bottleneck Layer (dimension reduction)
  - Auxiliary Classifier (shallow/weak output)

GoogLeNet - Going deeper with convolutions, 2014

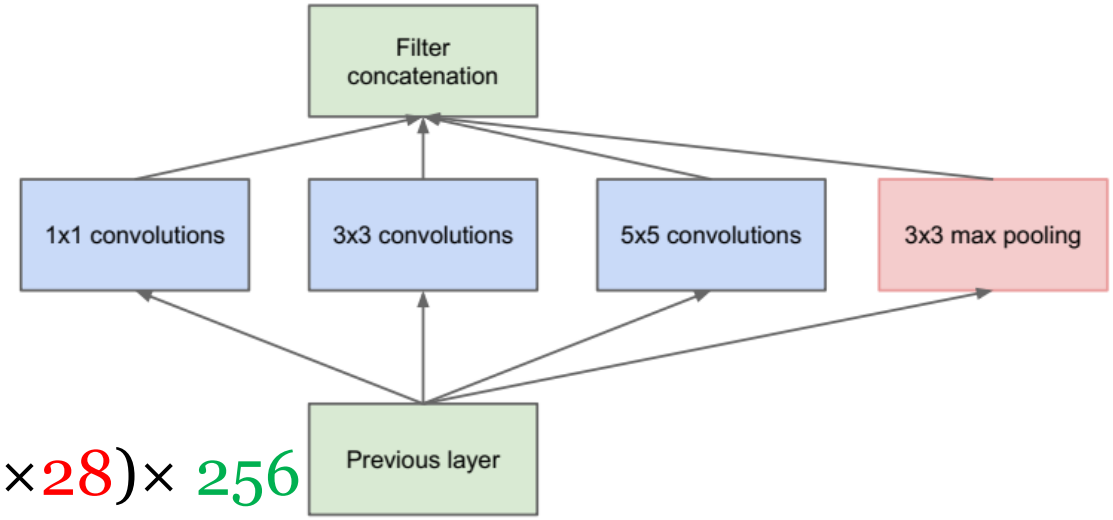
› Idea#1: Why Resolution reduction in successive Layer?

› Naïve Inception Version:



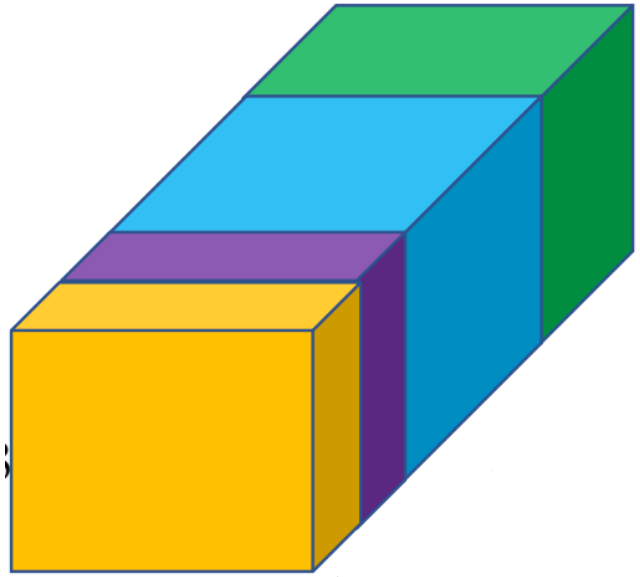
GoogLeNet - Going deeper with convolutions, 2014

- Inception V1 (Naïve Version):
- Previous Layer: (28×28)×256
- 64 (1×1)→ (28×28)×64
- 128 $(3\times3)_{same}$→ (28×28)×128
- 32 $(5\times5)_{same}$→ (28×28)×32
- (28×28) $\text{MaxPool}_{(3\times3),same}$→(28×28)× 256
- Concatenated Tensor:
  - (64+128+32+256)×(28×28)= 480×(28×28)= 376320



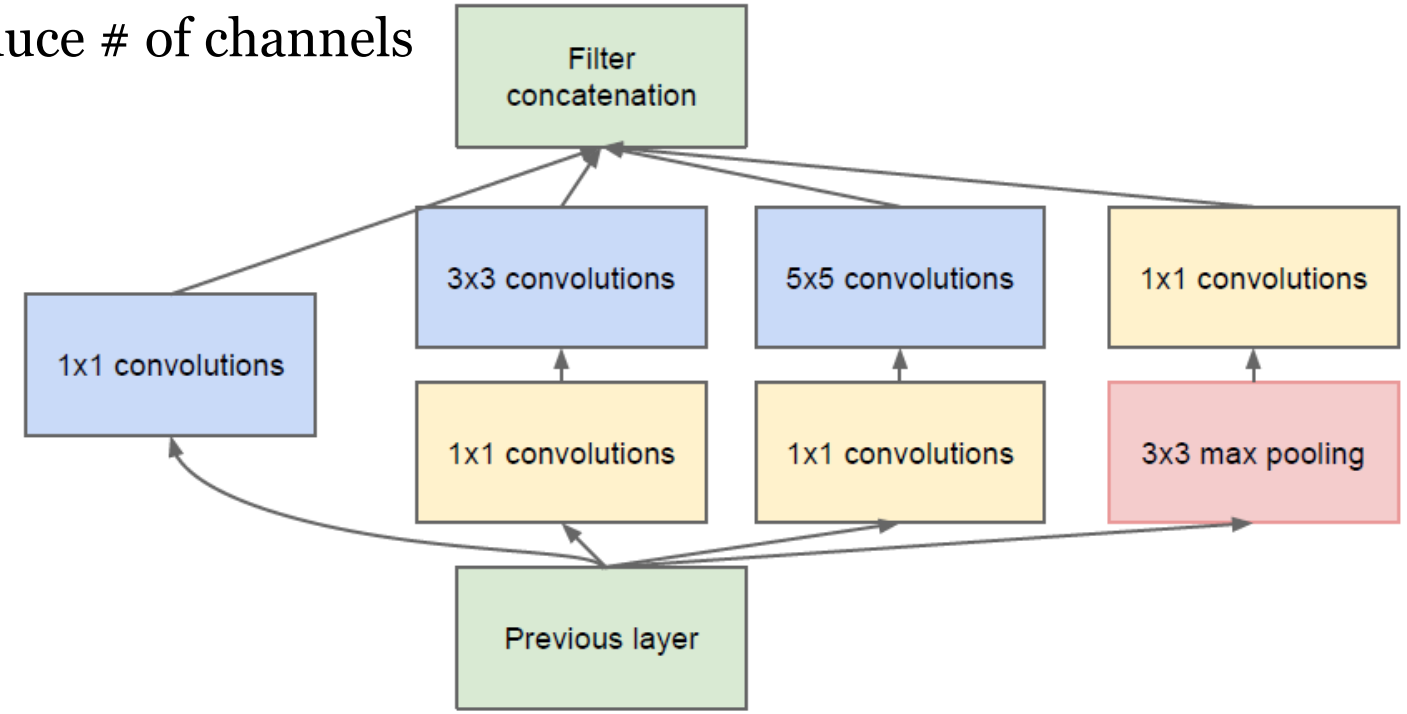
GoogLeNet - Going deeper with convolutions, 2014

- Idea#1:Resolution reduction in successive Layer
- Concatenated Tensor:
  - (64+128+32+256)×(28×28)= 480×(28×28)=
  - Memory and Computation Cost ☹



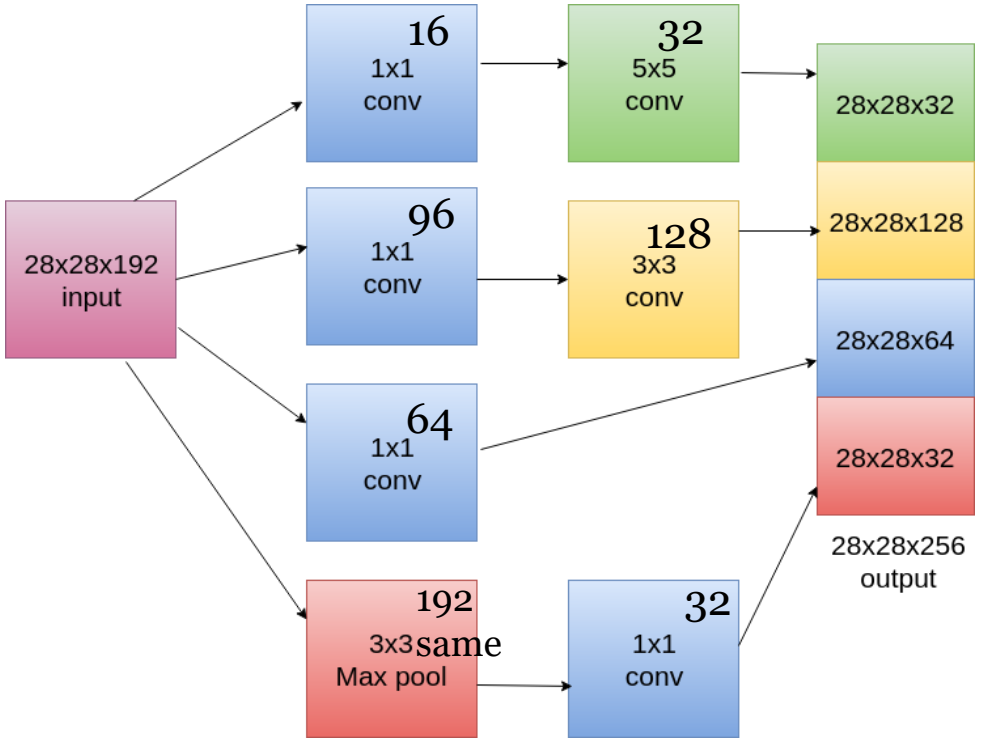
GoogLeNet - Going deeper with convolutions, 2014

- Idea#1: Inception **V1** (Inception Module with dimension Reduction):
  - Bottleneck Layer (1×1 Convolution)
    - Reduce # of channels



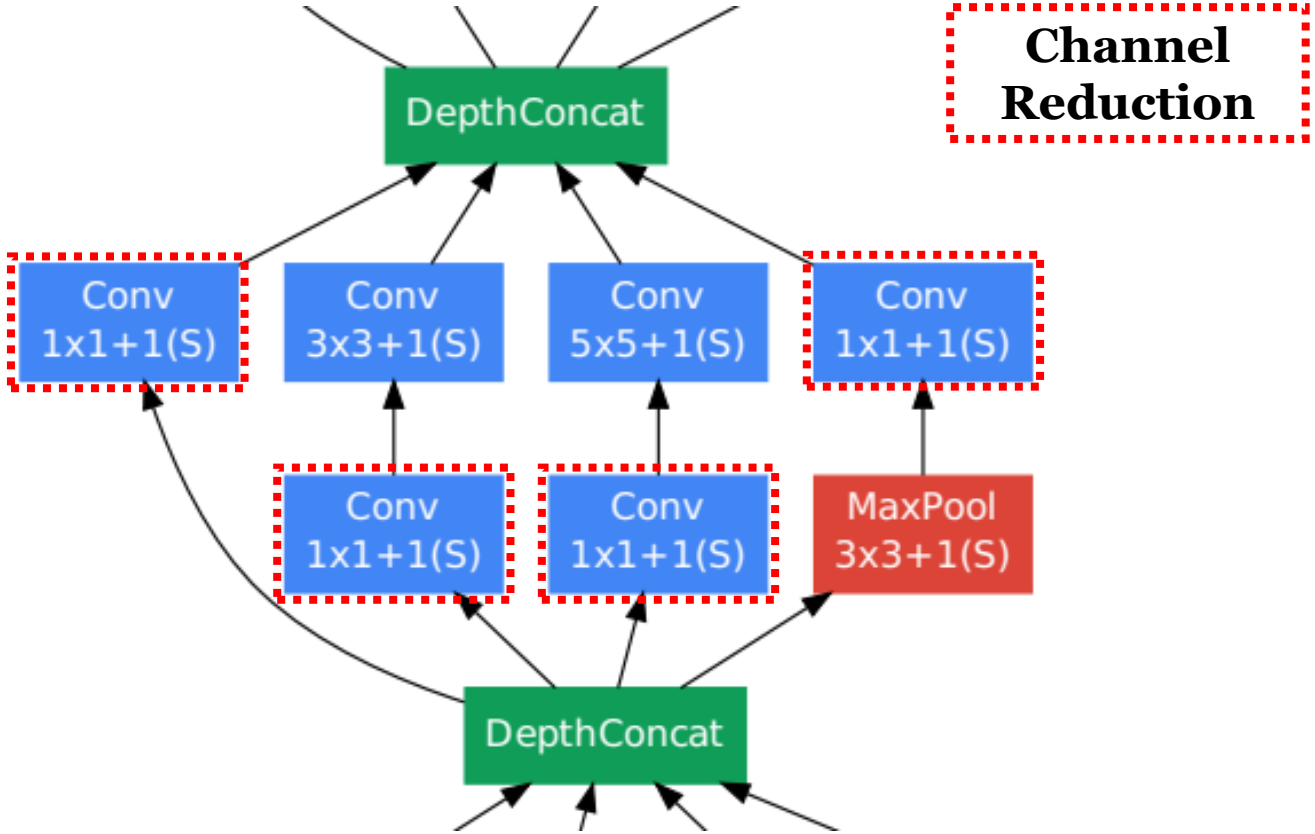
GoogLeNet - Going deeper with convolutions, 2014

› Example:



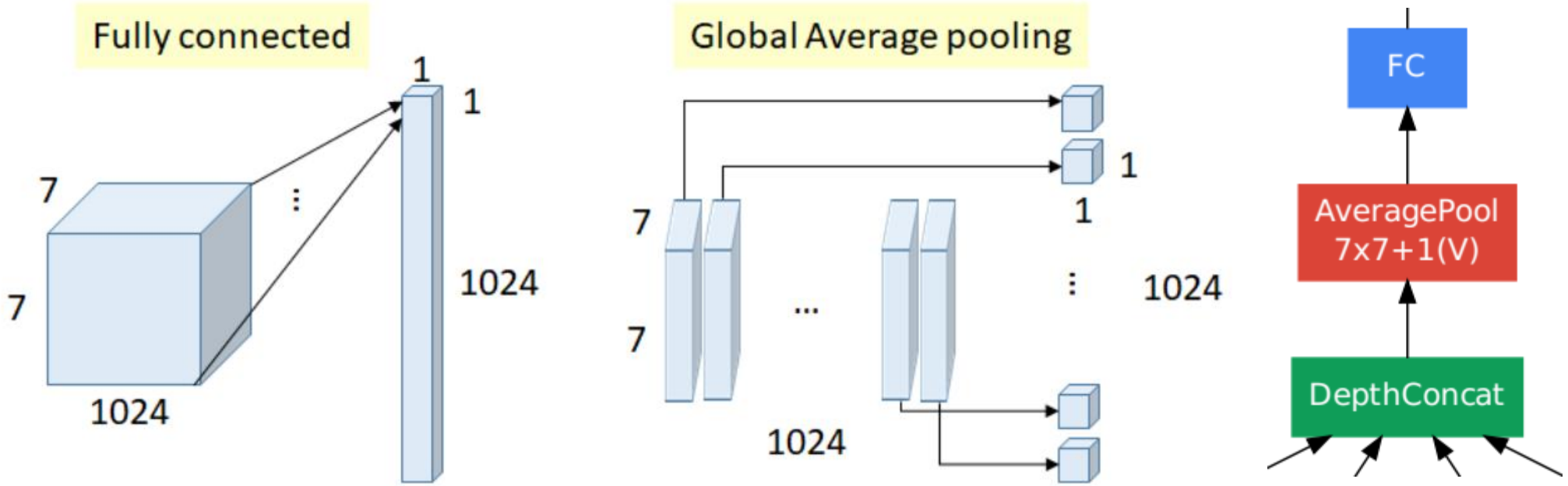
GoogLeNet - Going deeper with convolutions, 2014

› A Section from GoogLeNet:



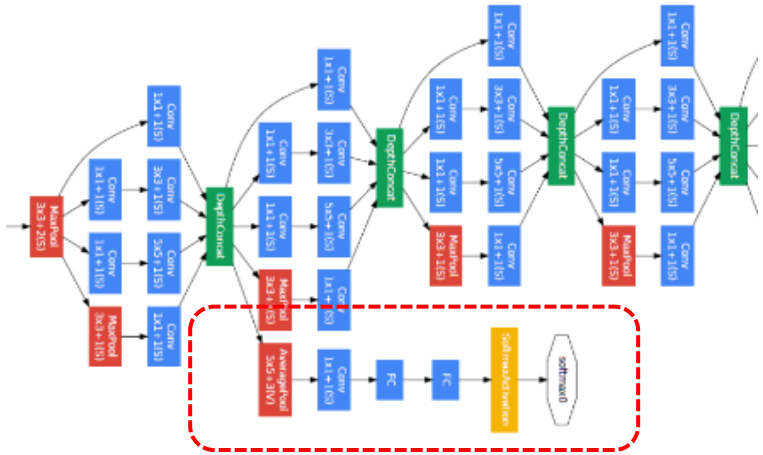
GoogLeNet - Going deeper with convolutions, 2014

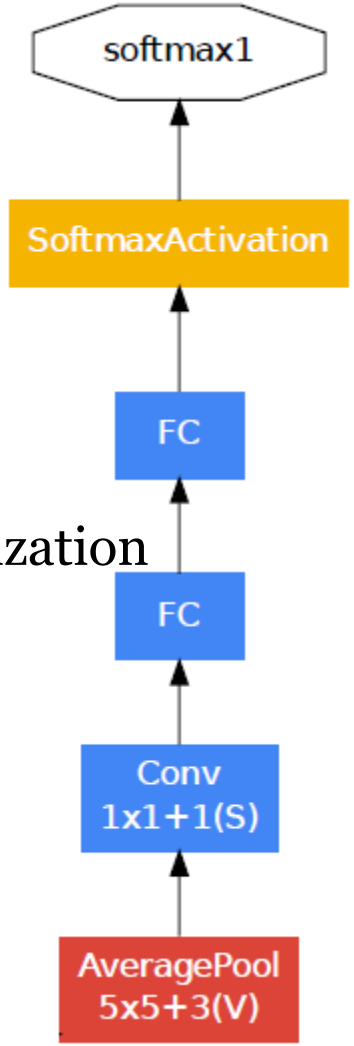
- Idea#2: Global Average Pooling Before FC
- Input: 7×7×1024
- Output: 1×1×1024



GoogLeNet - Going deeper with convolutions, 2014

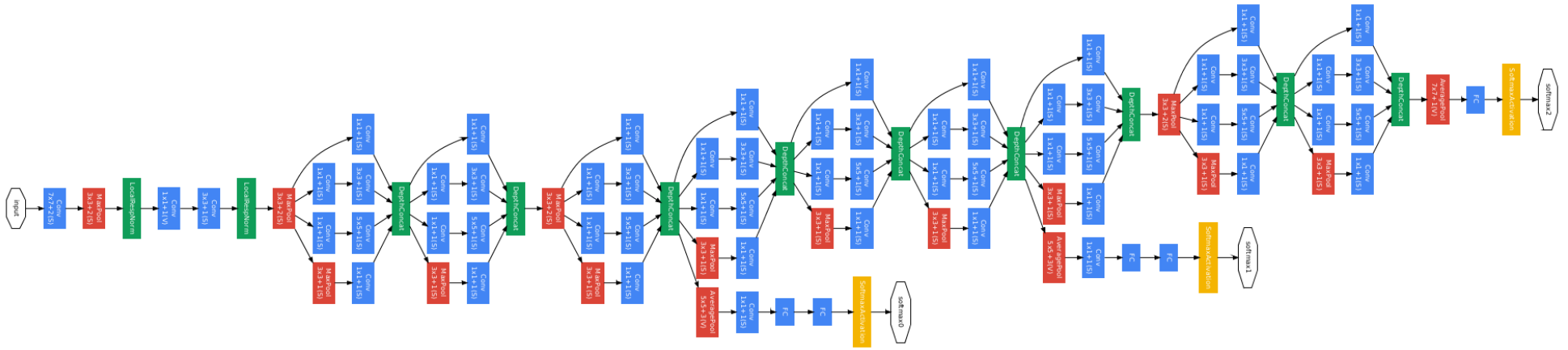
- Idea #3: Auxiliary Classifier (intermediate SoftMax branches)
  - Too Deep (22 Layers) → Gradient Flow problem!
  - Consider Shallow Outputs!
  - How?
    - Add their loss (0.3 weight)to deep loss during training→ Regularization





GoogLeNet - Going deeper with convolutions, 2014

› Complete Structure:



GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

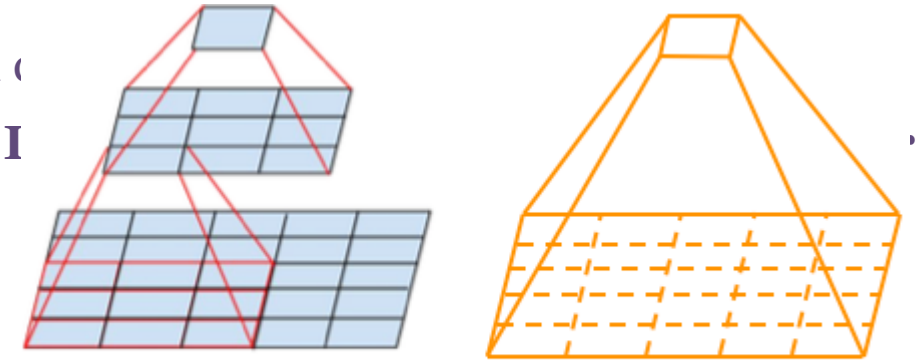
- Inception **V2** and **V3**
- Motivation and Idea:
  - Factorizing Convolutions
  - Efficient Grid Size Reduction
  - Label Smoothing as Regularization

GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

› Idea #1: (**Symmetric** Factorization)
- A 5x5 Convolution ~ Two successive 3x3 convolution
- $\text{Weights}(5\times5)_{\text{Conv}} > 2^* \text{Weights}(3\times3)_{\text{Conv}}$, (25-2*9)/25= %28
- $\text{Weights}(3\times3)_{\text{Conv}} > 2^* \text{Weights}(2\times2)_{\text{Conv}}$, (9-2*4)/9= %11
- Less parameter (28%), High Nonlinear (2×ReLU)

Going deeper with

Rethinking the I
Vision, 2015

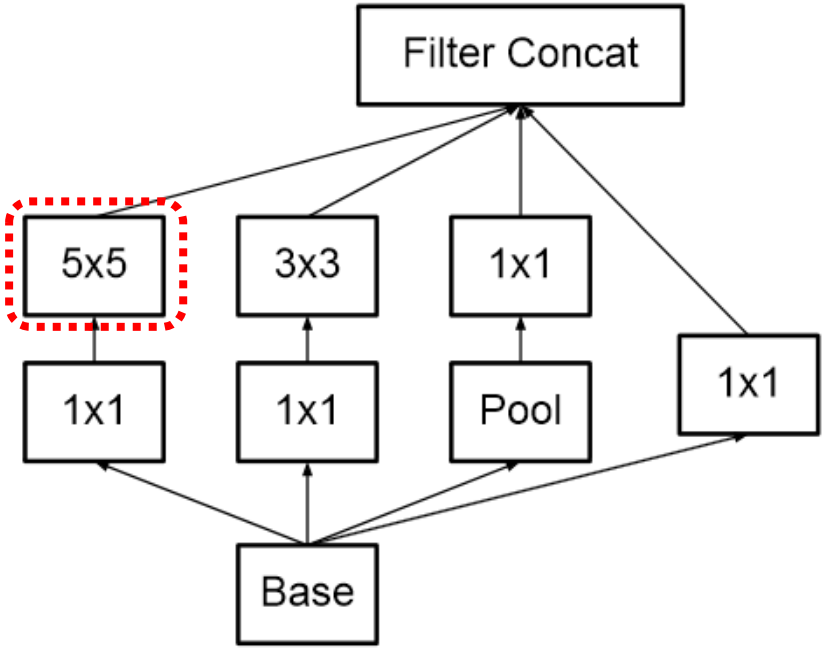


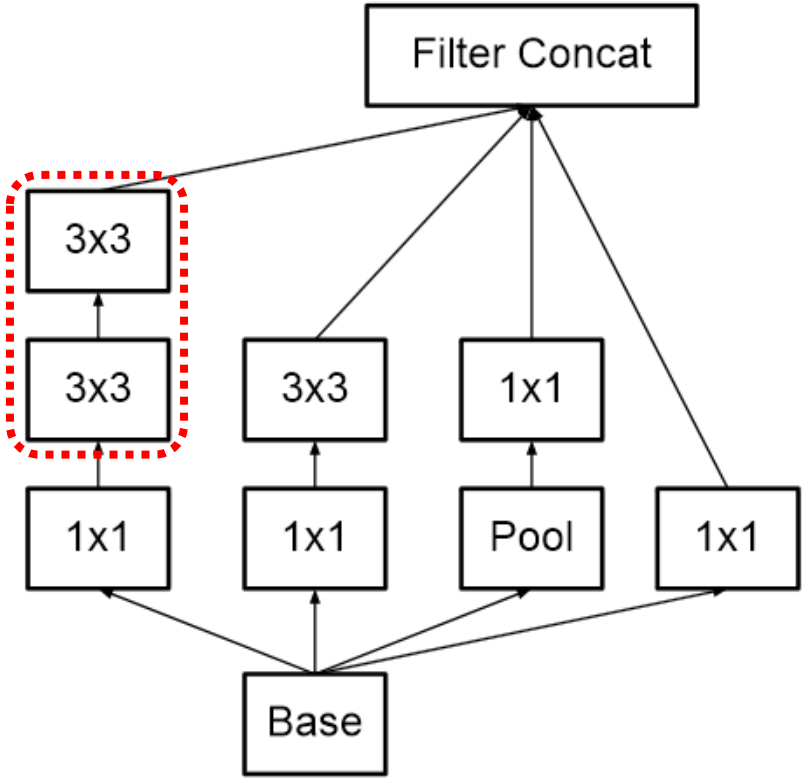
two successive
3x3 convolutions

5x5 convolution

GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

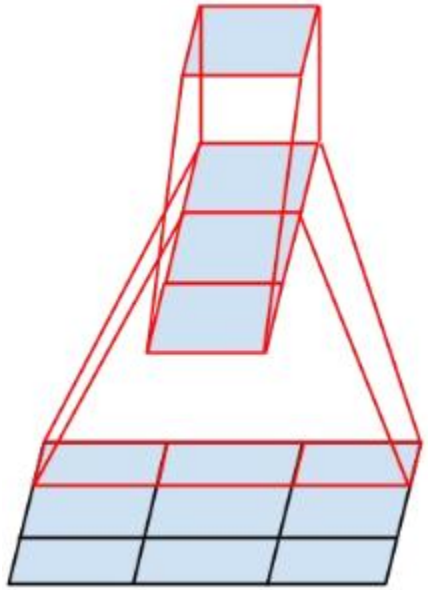
› From Inception V1 to Inception V2





GoogLeNet - GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

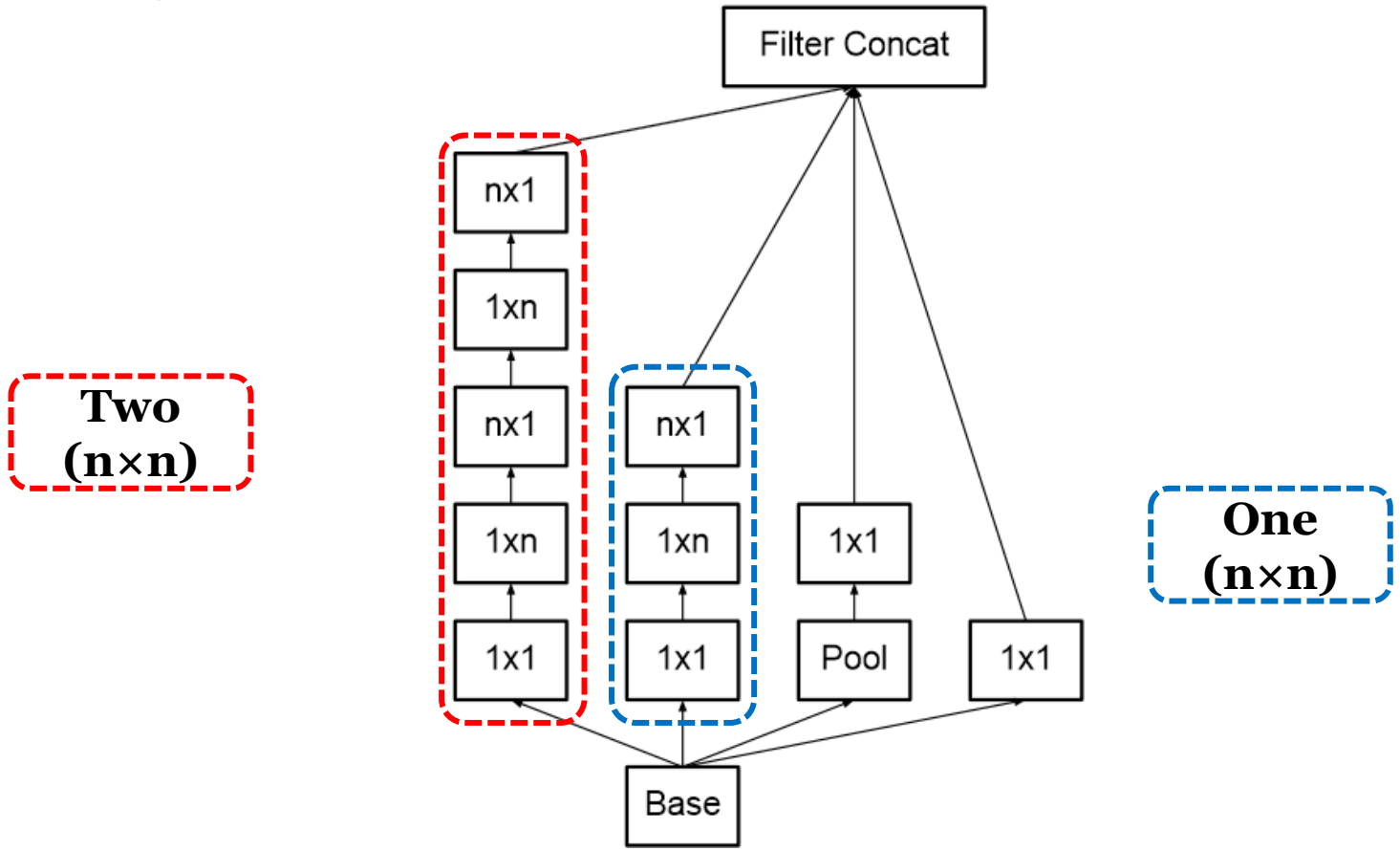
› Idea #1: More Factorization (**Asymmetric** Factorization)
- A $(3x3)_{Conv} \sim (3x1)_{Conv}$ Followed by $(1x3)_{Conv} \rightarrow (9-6)/6=33\%$
- Less Parameter (33%), More Nonlinearity (2×ReLU)



$$\begin{bmatrix}1\\2\\3\end{bmatrix} * \begin{bmatrix}4 & 5 & 6\end{bmatrix} = \begin{bmatrix}4 & 5 & 6\\8 & 10 & 12\\12 & 15 & 18\end{bmatrix}$$

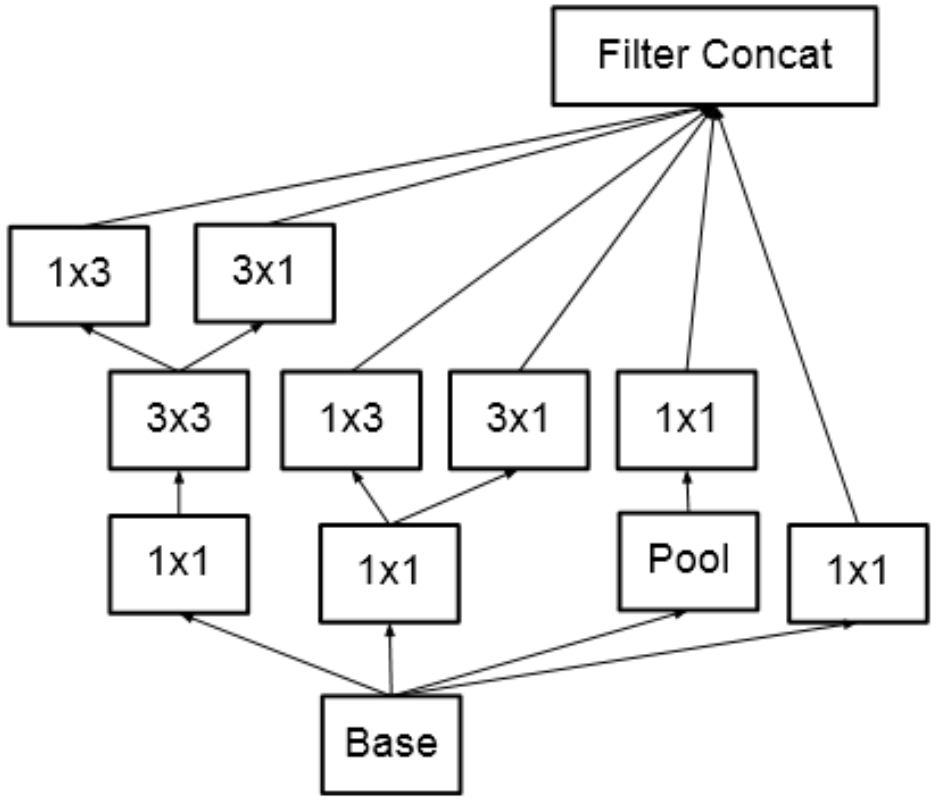
GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

› Illustration:



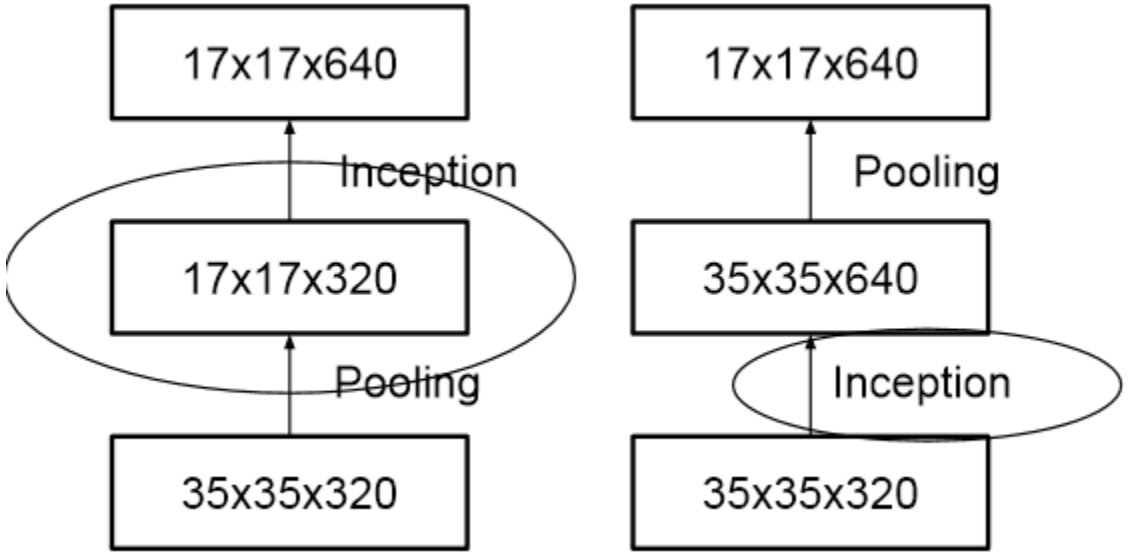
GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

› Make it wide (promote high dimensional representations)



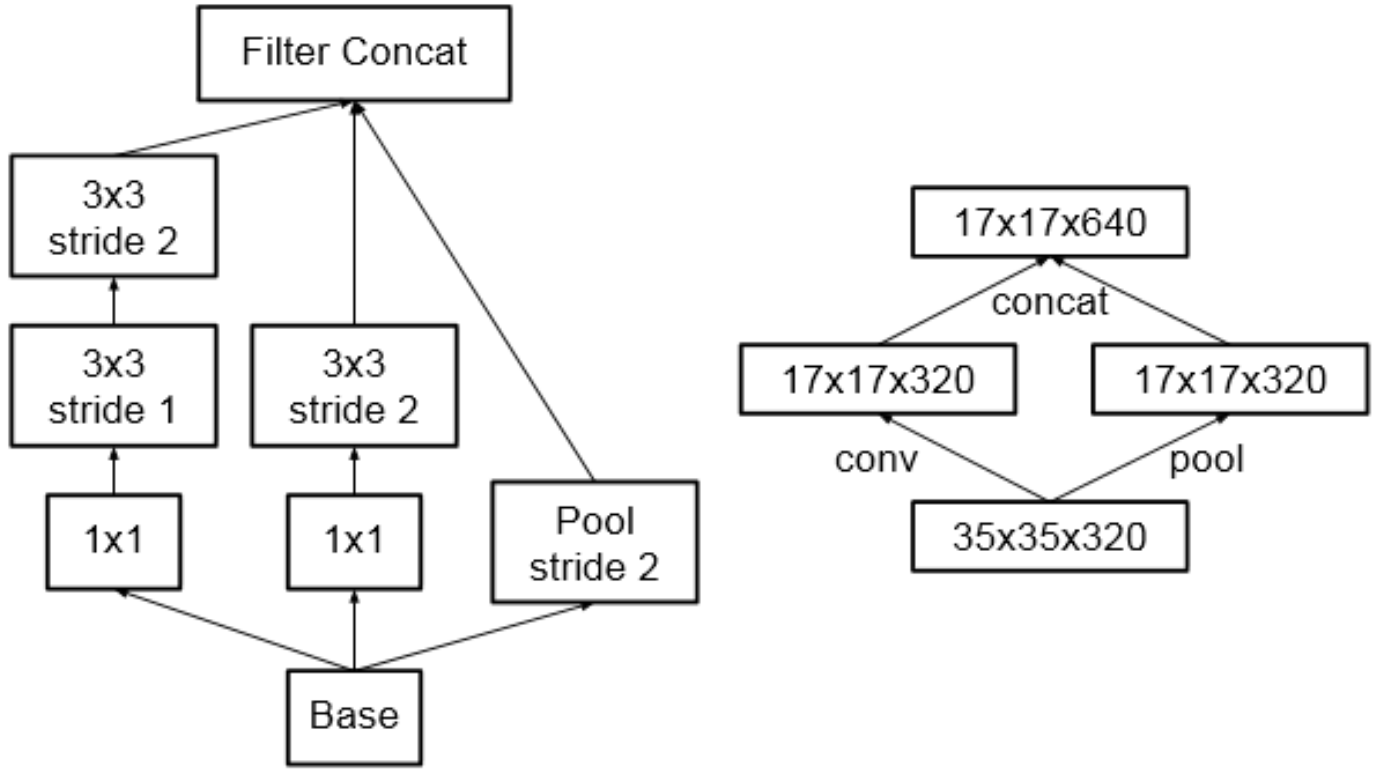
GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

- Idea #2: Efficient Grid Size Reduction
  - How to downsample?
    - Pooling then Convolution: Too Greedy (information loss)
    - Convolution then Pooling: High Cost



GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

› Idea #2: Efficient Grid Size Reduction
  – Solution:



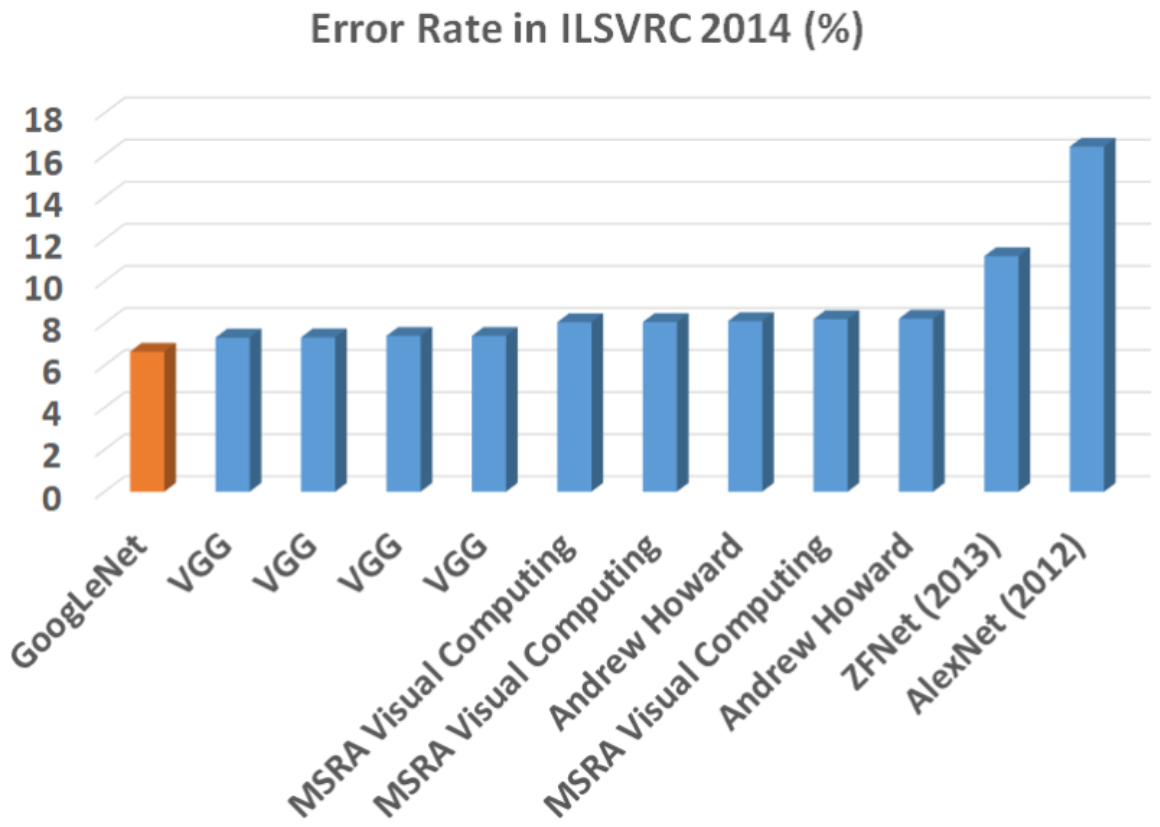
GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

› Idea #3: Label Smoothing
  – Add noise to target (label), to regularize learning strategy:
  – $\varepsilon \approx 0.1$, K = # of Classes

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\varepsilon}{K} \\ \vdots \\ (1-\varepsilon) \\ \vdots \\ \frac{\varepsilon}{K} \end{bmatrix}$$

GoogLeNet - Going deeper with convolutions, 2014

› Results:



GoogLeNet - Going deeper with convolutions, 2014
Rethinking the Inception Architecture for Computer Vision, 2015

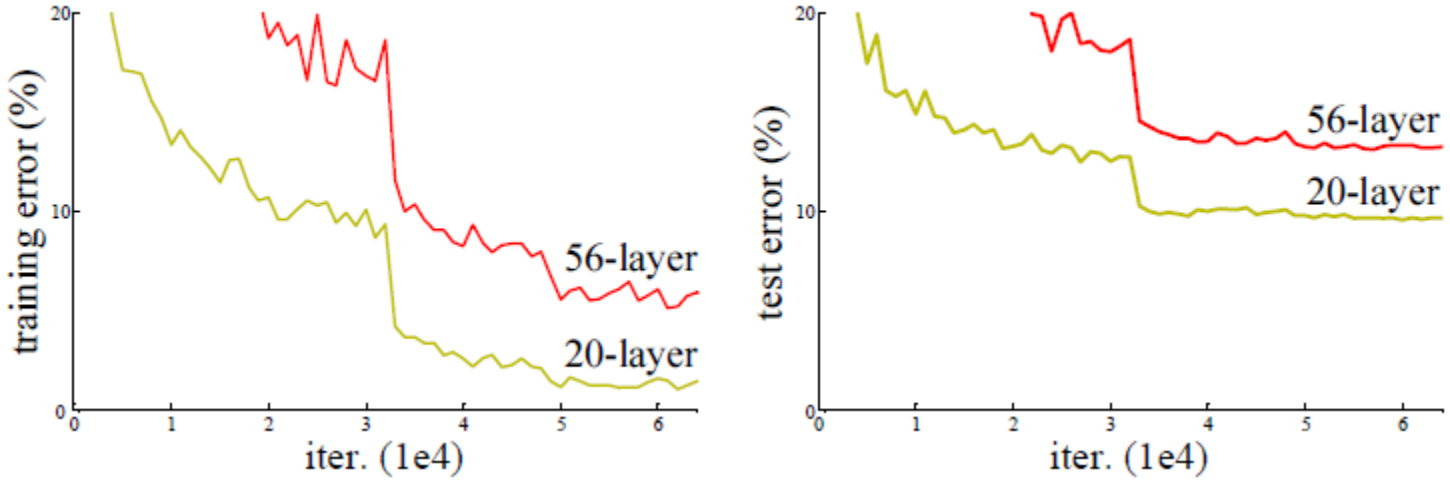
- Inception **V3**
- Motivation and Idea:
  - RMSProp Optimizer.
  - Factorized 7x7 convolutions.
  - BatchNorm in the Auxillary Classifiers.
  - Label Smoothing

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception **V4**
  - Some block modifications and ResNet (Microsoft, next slides) module

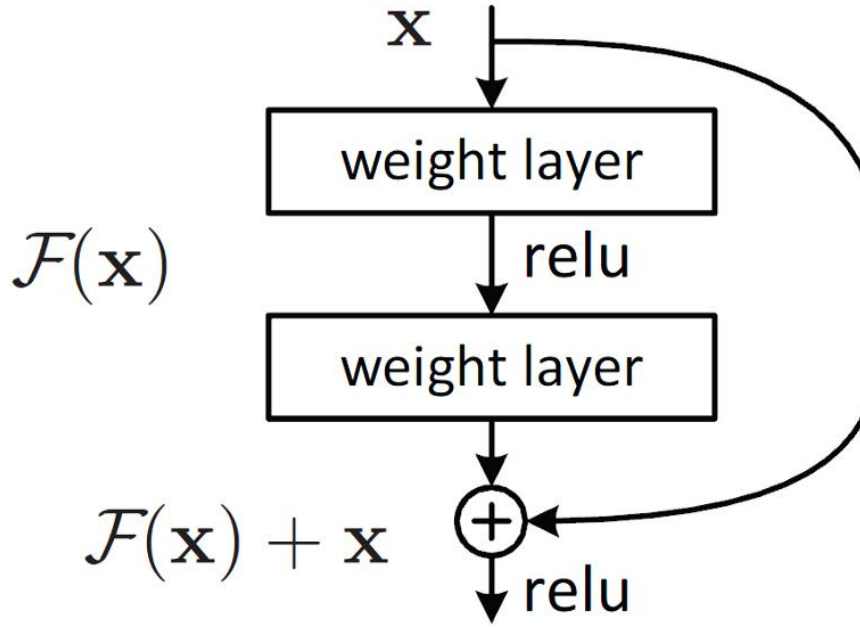
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Problems with Deeper Networks:
  – Performance degradation, Not due to overfitting
  – Gradient vanishing/exploding
  – Increasing Depth→ Accuracy Saturation then degrades rapidly!
  – For a PLAIN network (on CIFAR-10)



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Idea: New Block (***Shortcut Connection/Skip Connections***)
  - Deep Residual Learning!



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Why?: A Identity map is hard to capture by highly nonlinear map.
- In Residual Learning:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

- $H(x)=F(x)+x$, the $F(x)$ may learn to be zero!
  - ☺ No Extra Parameter!
  - ☺ No Computational Complexity!
  - ☹ Problem of dimension F and X
    - Solution:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

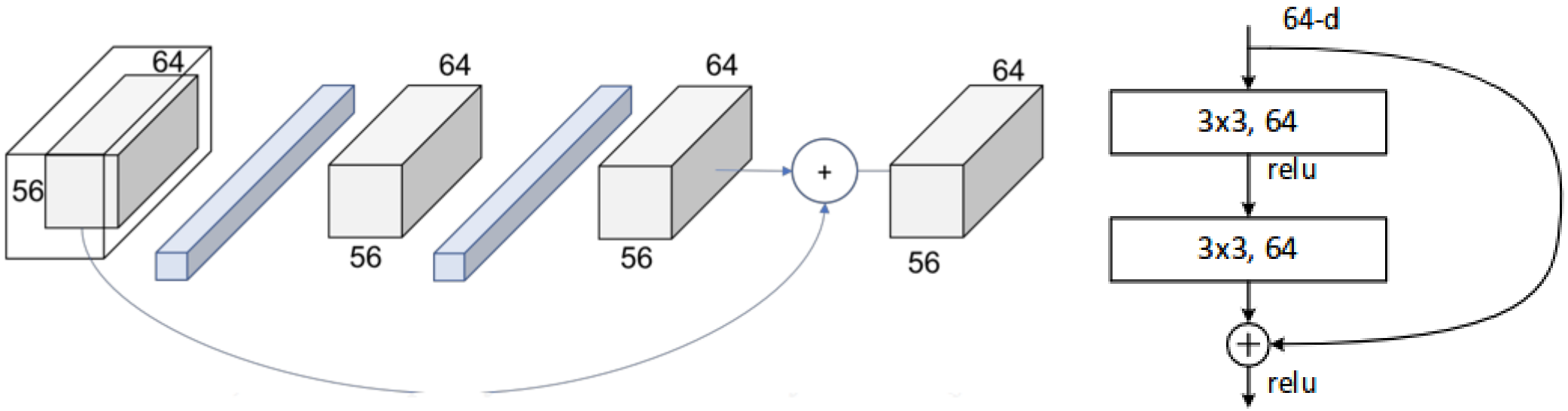
- ResNet Benefits:
  - Vanishing and exploding gradients
  - May increase # of layer

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Residual Layer (Type I):

› Input: (56×56)×64

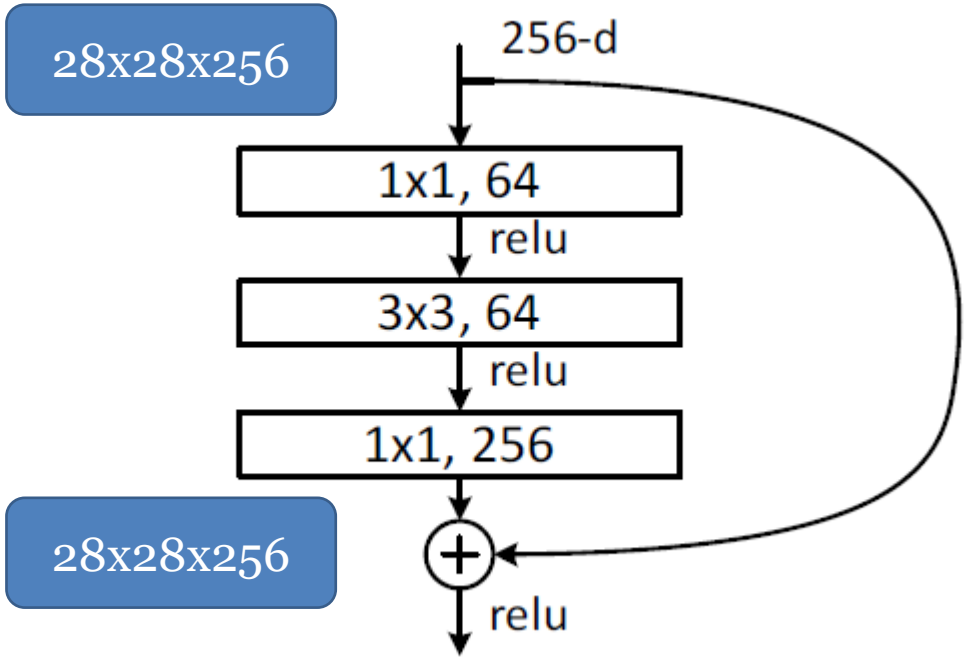
› Kernel: $(3\times3)_{same}\times64$



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

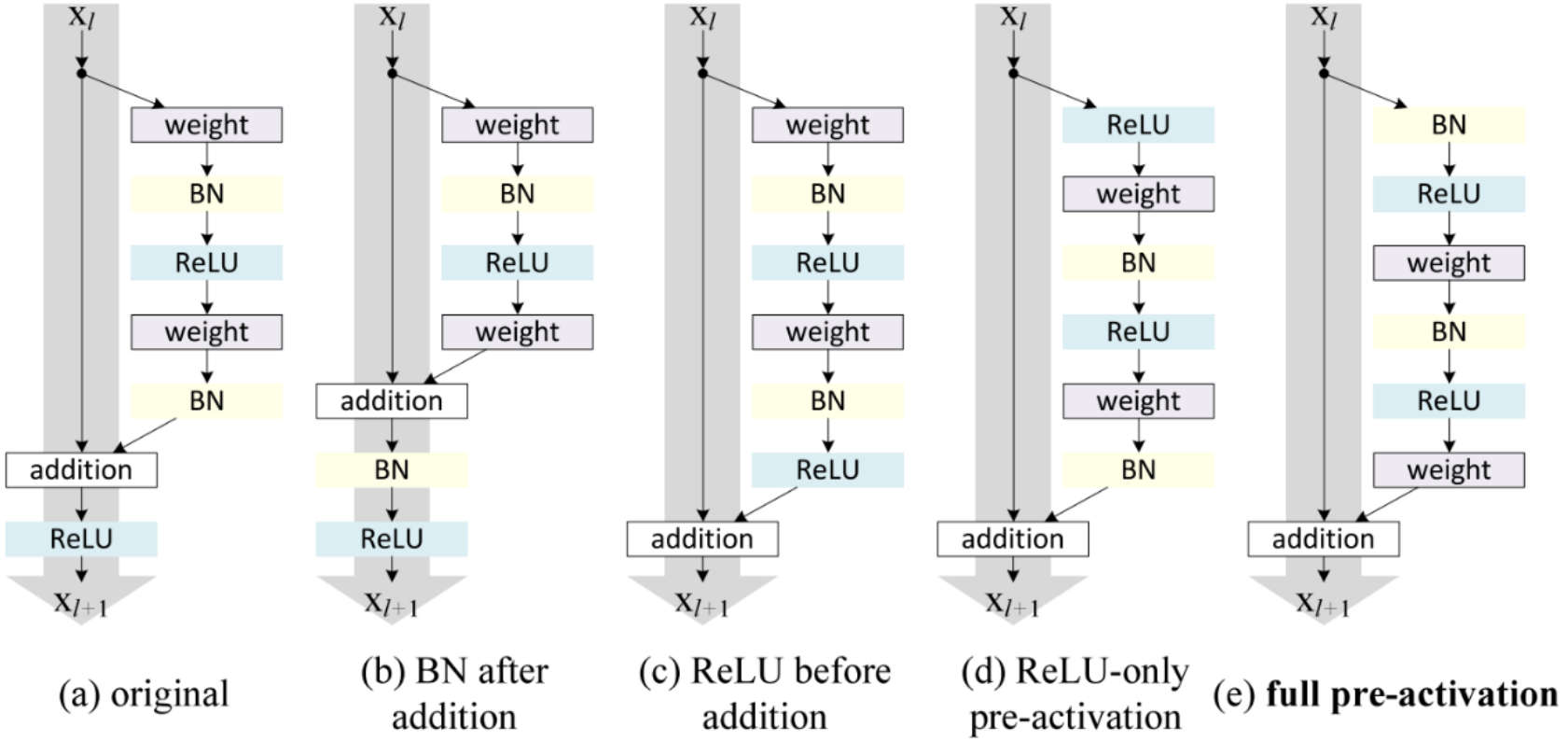
› Residual Layer (Type II):

› Another idea for Deeper Net (Bottleneck Design):



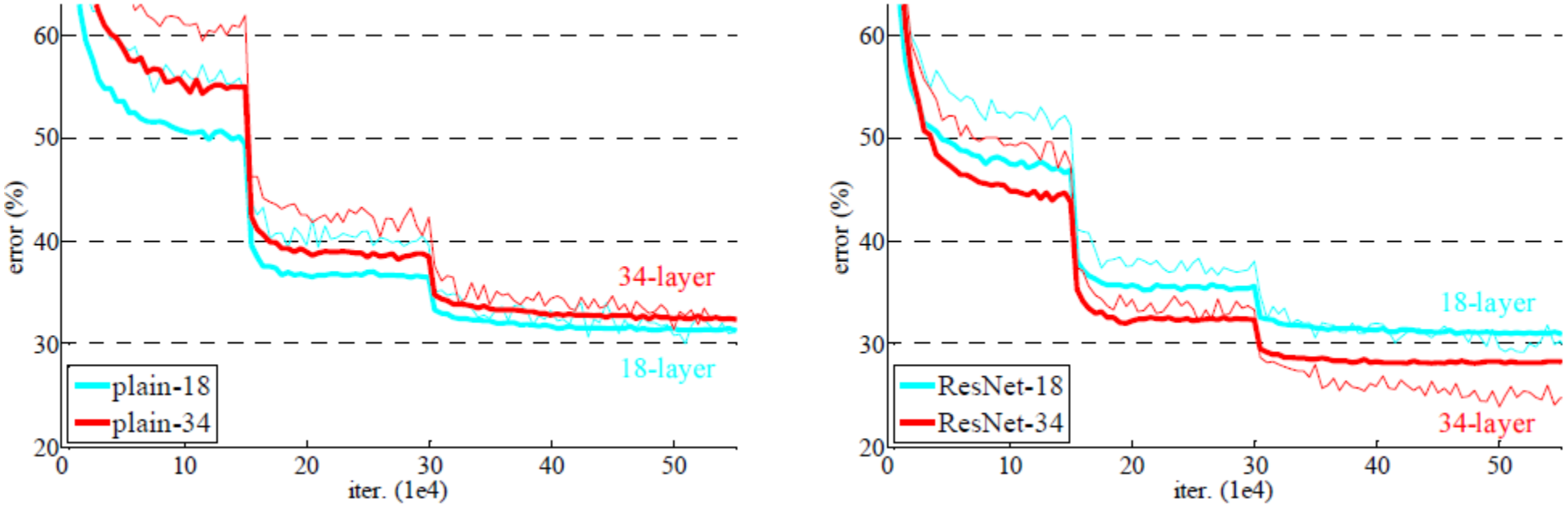
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Variation:



ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Results (ImageNet, Not too deep, 18/34):



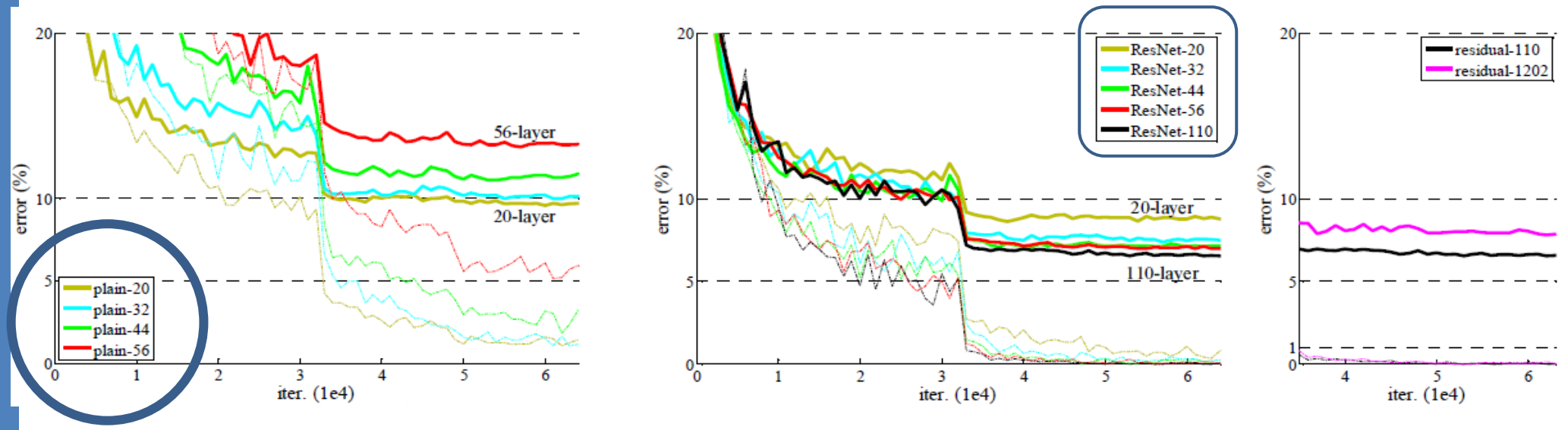
ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Results (ImageNet, very deep, 152):

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Results (CIFAR, too deep, 20/110/1202):

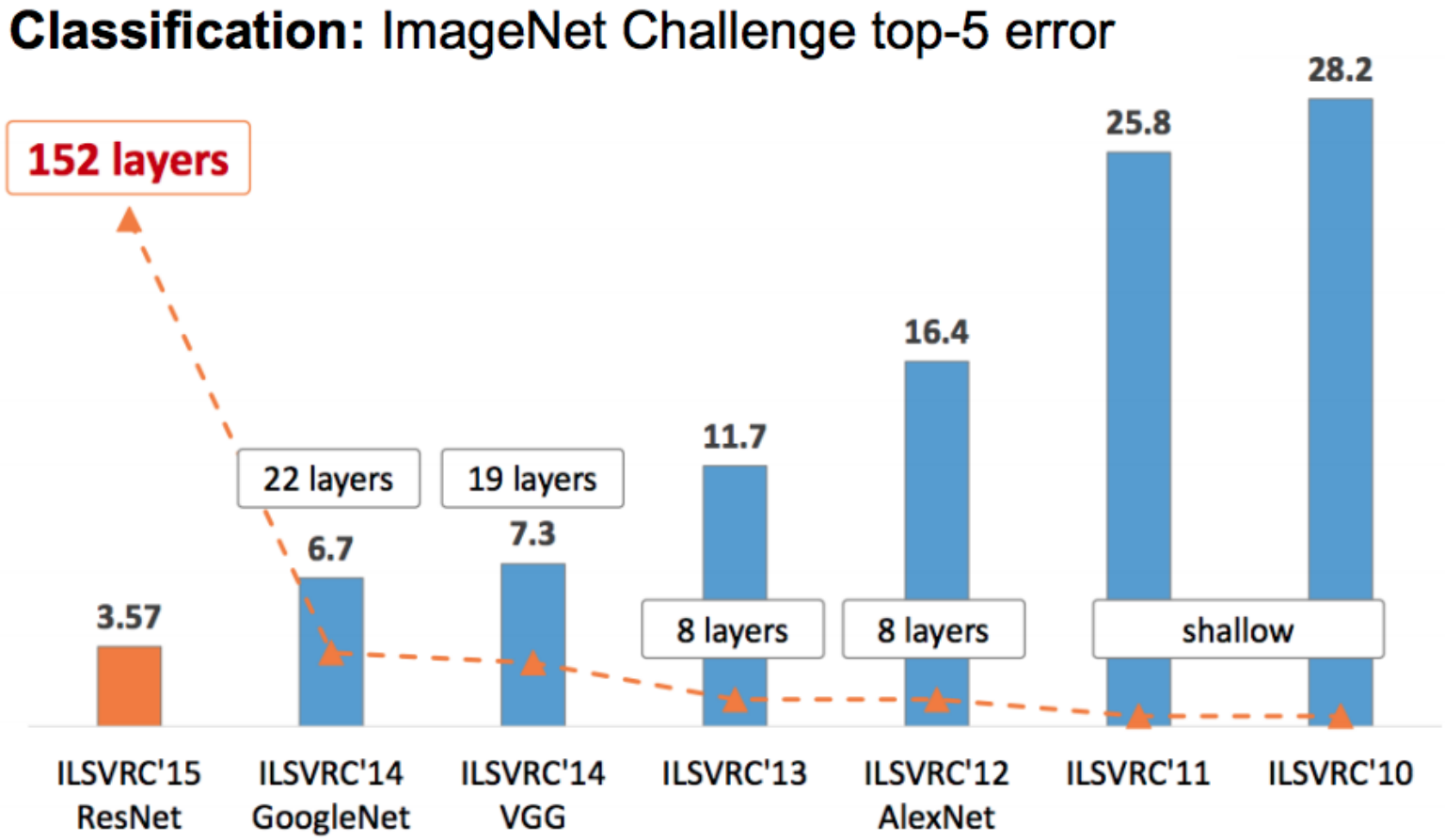


ResNet - Deep Residual Learning for Image Recognition (Microsoft)

- Specification:
  - Batch Normalization after each ConvLayer
  - SGD + momentum (0.9)
  - Minibatch Size: 256
  - No Dropout

ResNet - Deep Residual Learning for Image Recognition (Microsoft)

› Depth Effect!



Results - https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/googlenet.html

› ImageNet 2015

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

Microsoft ResNet, a 152 layers network

GoogLeNet, 22 layers network

U. of Toronto, SuperVision, a 7 layers network

human error is around *5.1%* on a subset

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

- Inception **V4**
  - Introduce 3 types of *inception module*, 2 types of *reduction module*
  - Build *Inception-ResNet*, *Inception-ResNet reduction* Block

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception **V4**, Inception Modules A, B, C

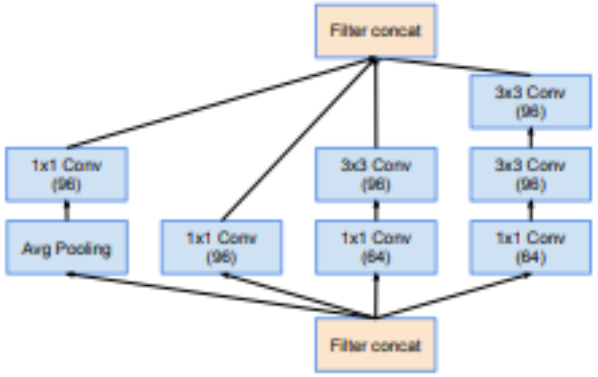


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

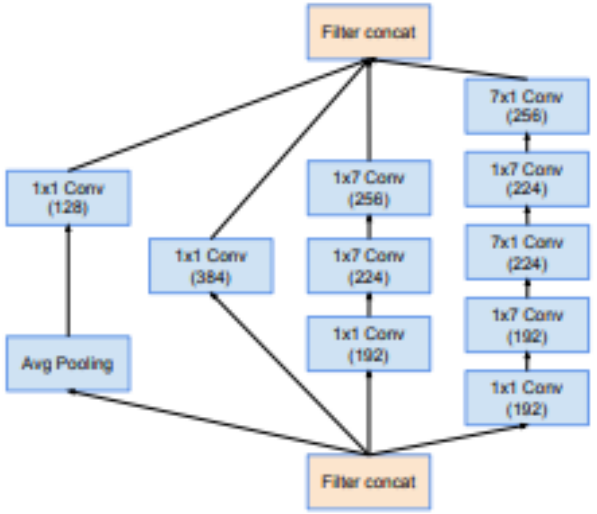


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

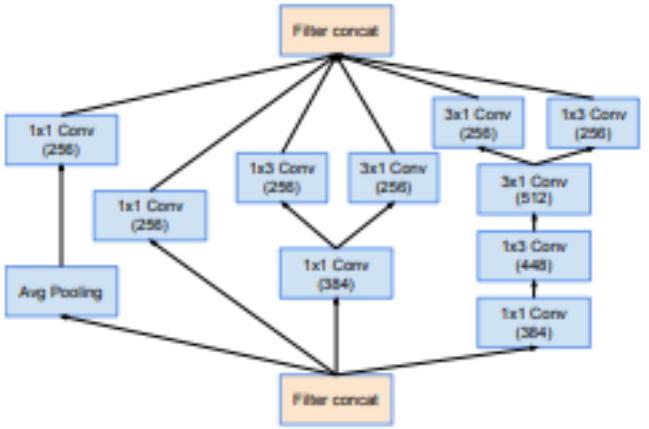


Figure 6. The schema for 8 × 8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception **V4**, Inception-Reduction Modules A, B

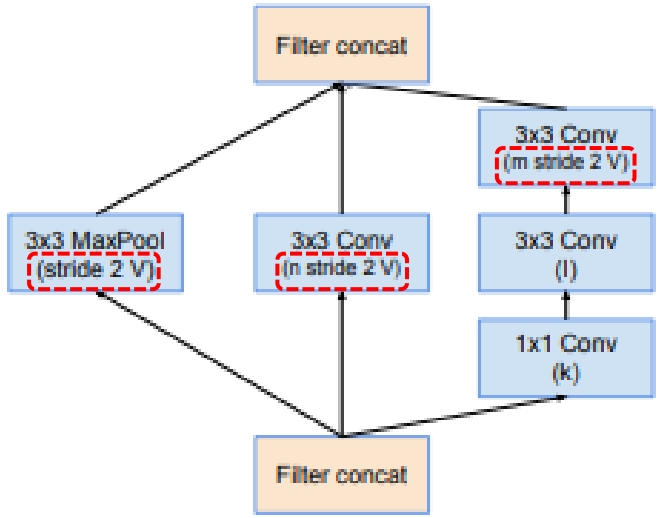


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

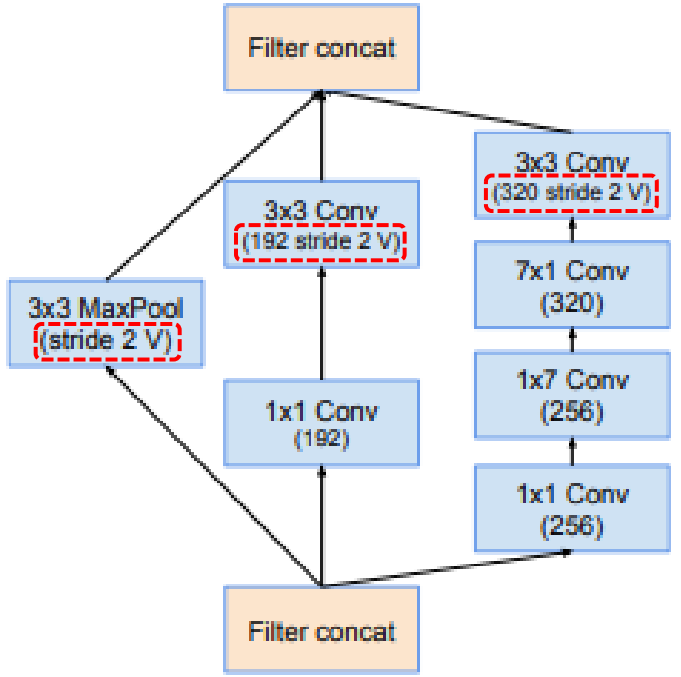


Figure 8. The schema for 17 × 17 to 8 × 8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception **V4**, Inception-ResNet Modules A, B, C

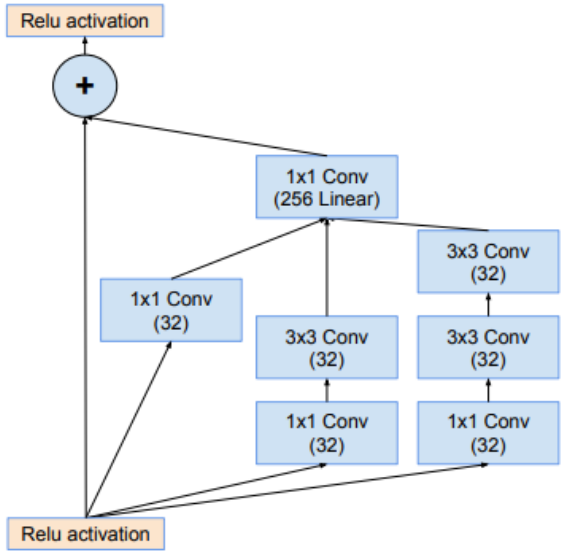


Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

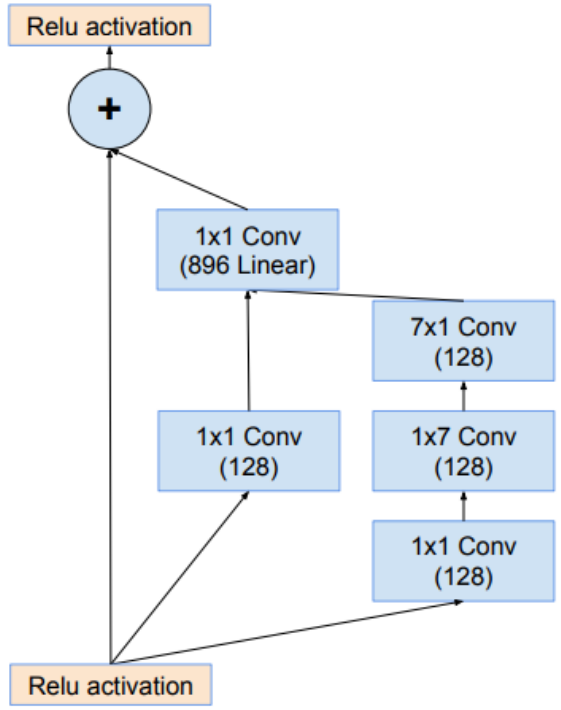


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

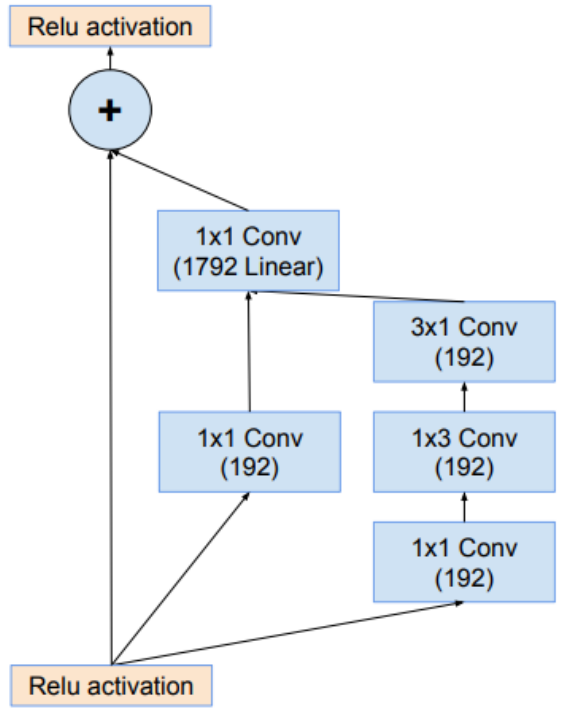
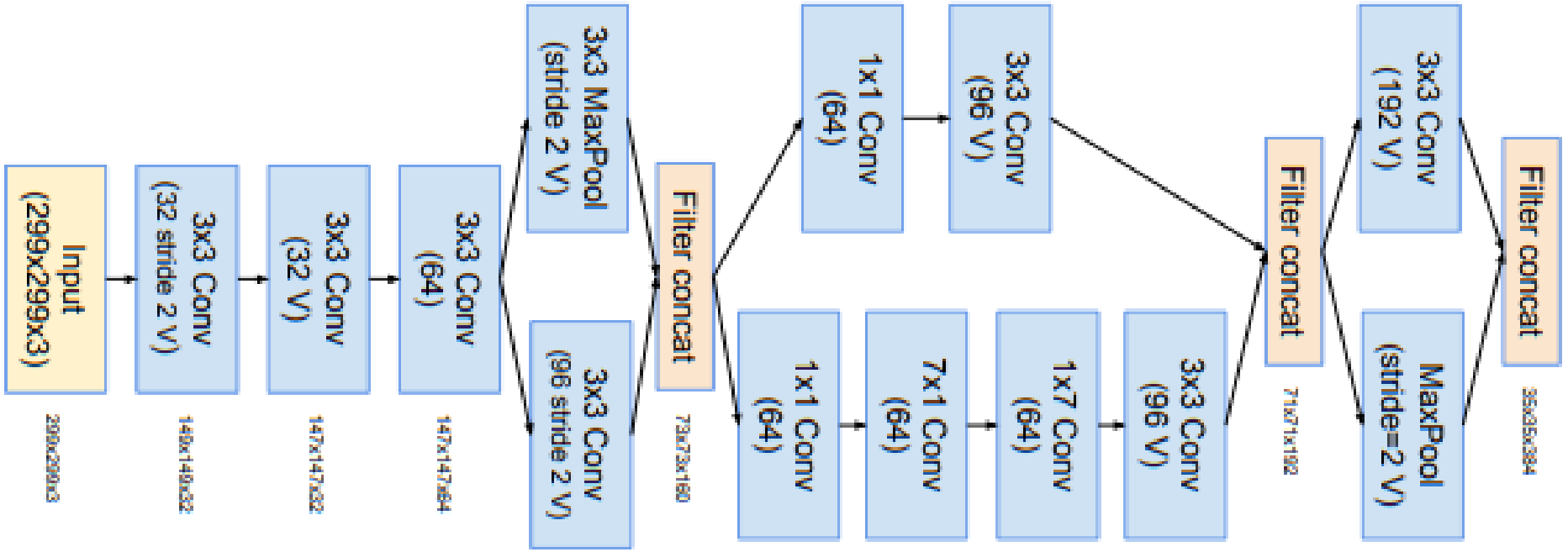


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

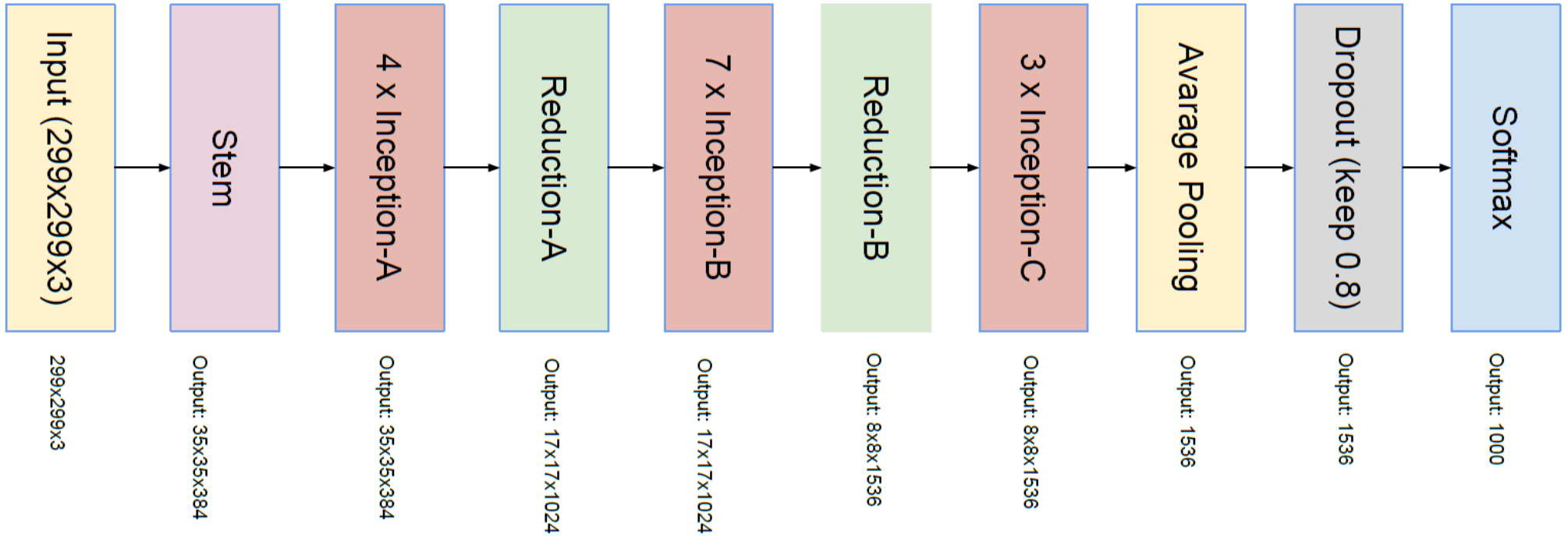
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception **V4**, stem module



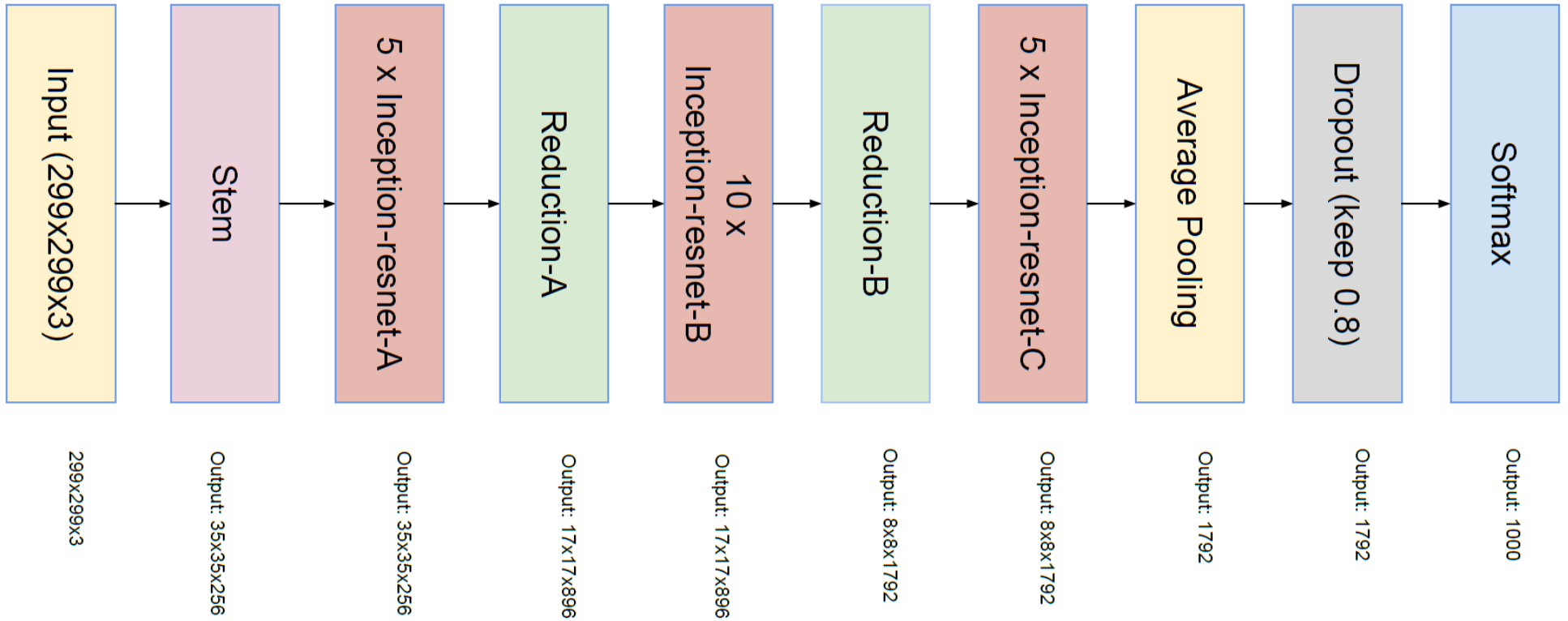
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception **V4**, Overall Scheme



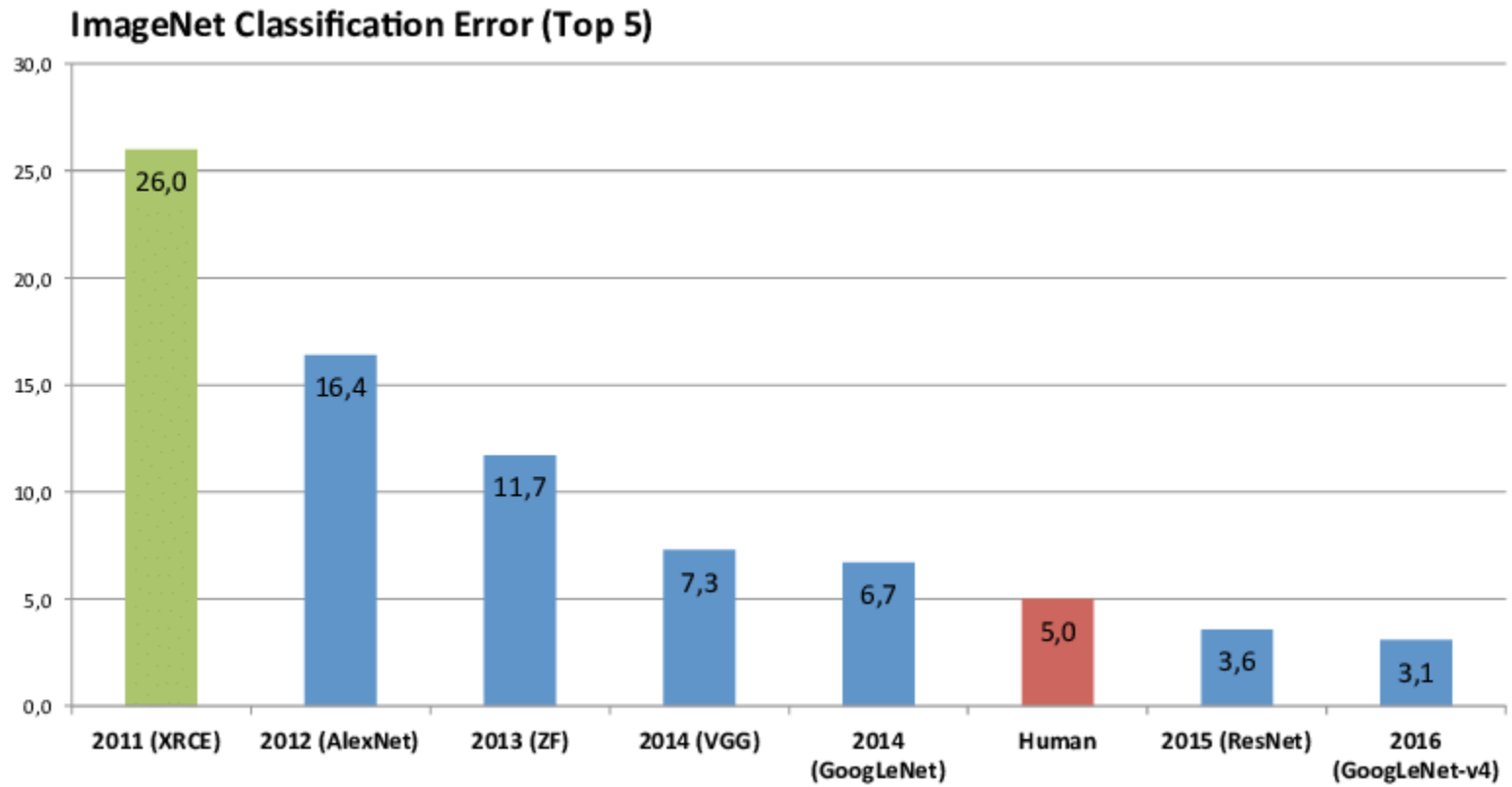
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Inception-ResNet **V4**, Overall Scheme



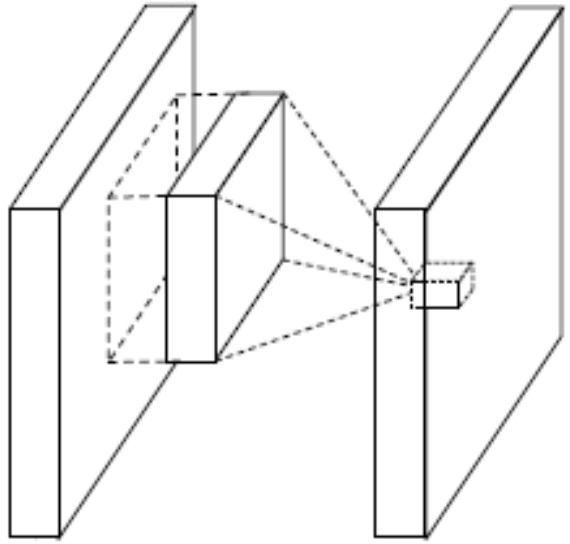
GoogLeNet - Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, Google, 2016

› Results



NiN - Network In Network, 2013

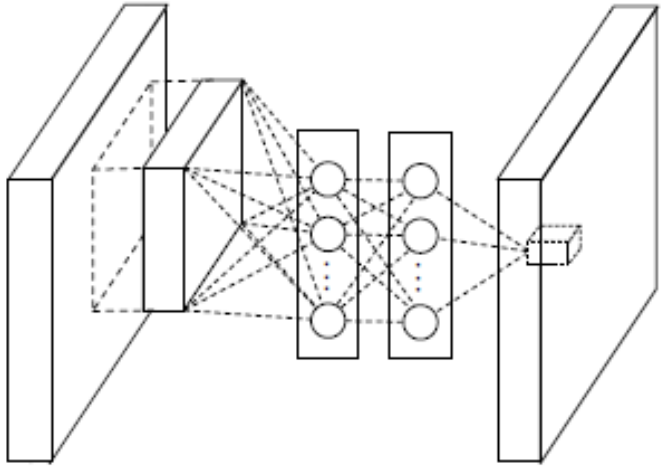
› Network in Network (NiN)
  – Conventional Convolution Layer:



$$f_{i,j,k} = \max(w_k^T x_{i,j}, 0).$$

NiN - Network In Network, 2013

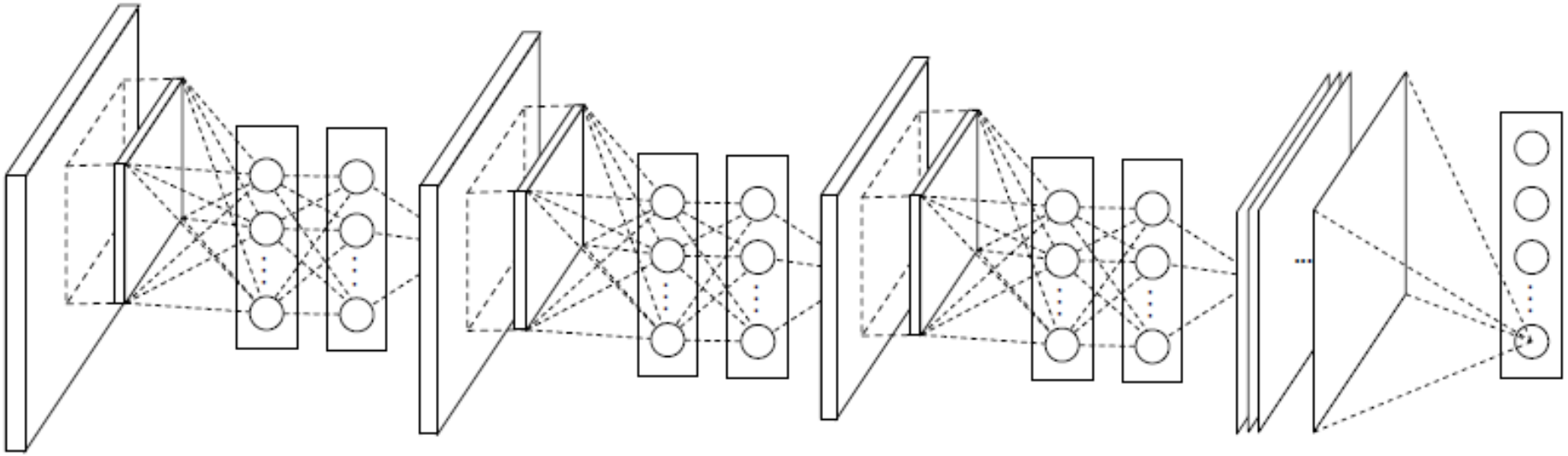
› Network in Network (NiN)
  – MLP Convolution Layer:



$$
\begin{aligned}
f^1_{i,j,k_1} &= \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0). \\
&\vdots \\
f^n_{i,j,k_n} &= \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0).
\end{aligned}
$$

NiN - Network In Network, 2013

› Network in Network (NiN)
  – Global Average Pooling Before FC



DenseNet - Densely Connected Convolutional Networks, CVPR 2017

› Idea:

› Conventional CNN:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}).$$

› ResNet:

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}.$$

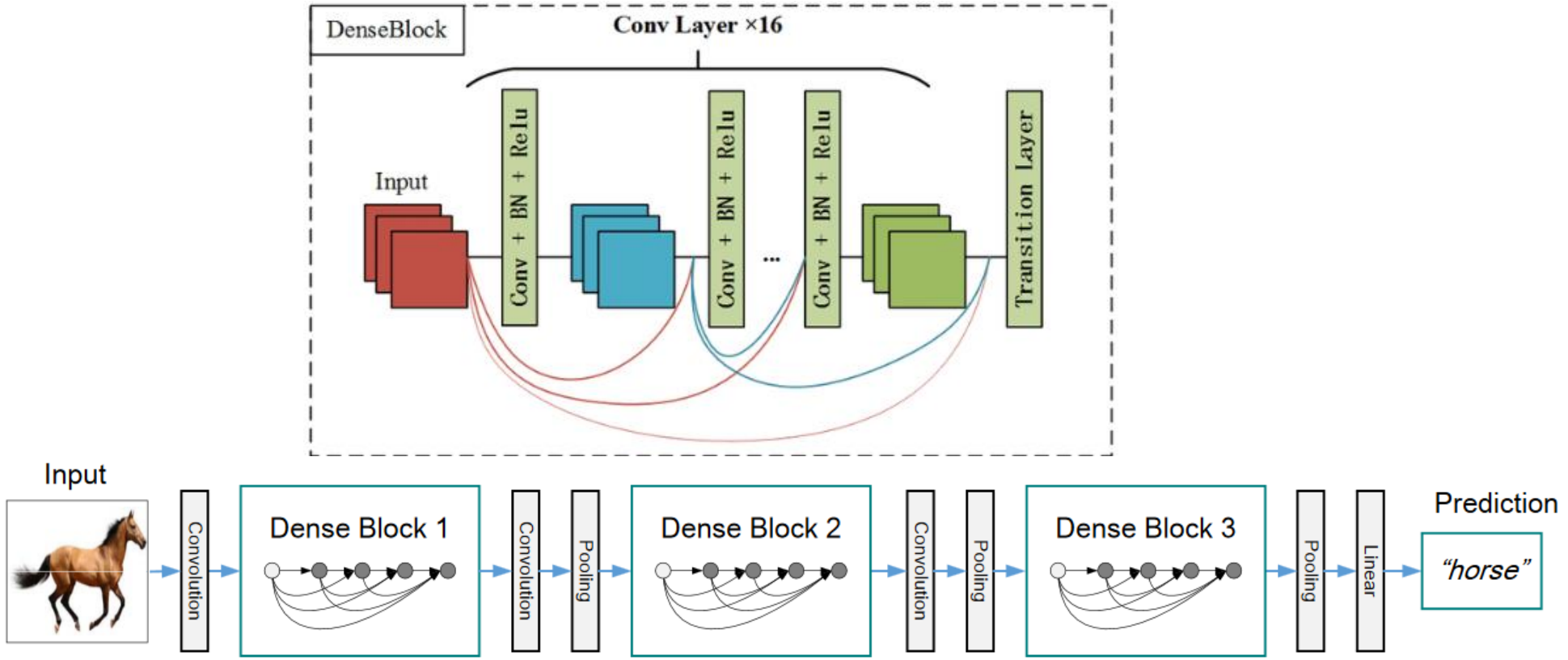
› DenseNet:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]),$$

› $[x_0, x_1, \ldots, x_{l-1}]$: Concatenation of the feature-maps in previous layer

DenseNet - Densely Connected Convolutional Networks, CVPR 2017

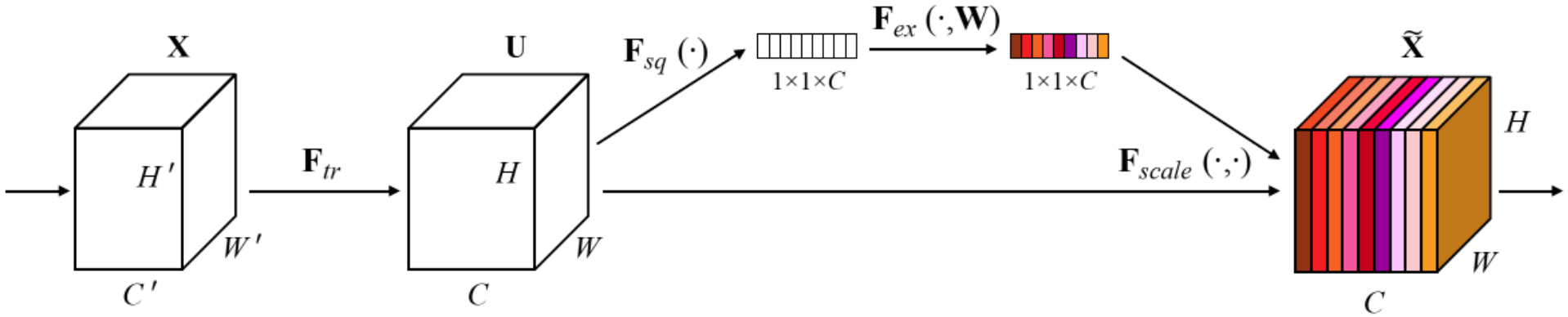
› $H(.)$ functionality: Batch Normalization, ReLU, $(3\times3)_{\text{Conv}}$



SENet - Squeeze-and-Excitation Networks, CVPR 2017

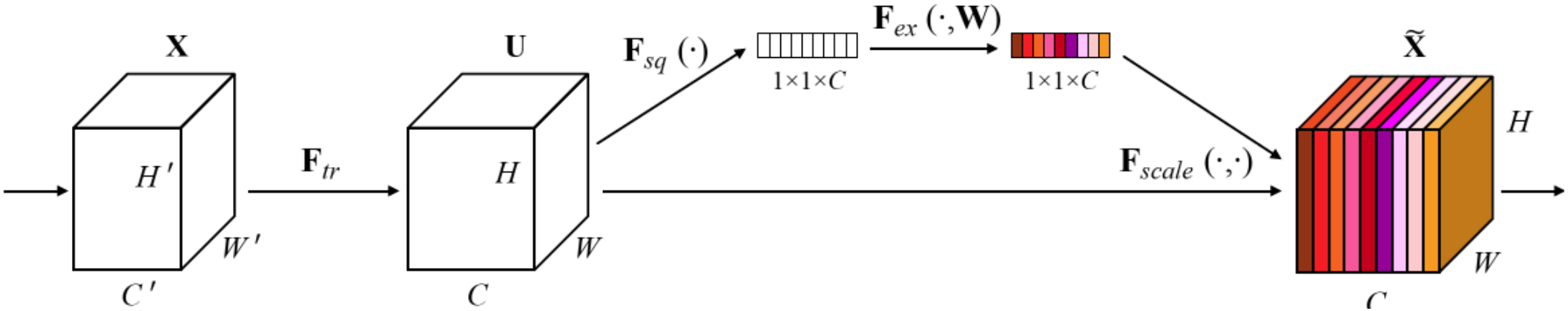
› Main Idea: Weight Channels by learning

› Squeeze-and-Excitation (SE) Block:



SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Squeeze-and-Excitation (SE) Block:



› Transform: $U = F_{tr}(X)$: $\mathbb{R}^{H' \times W' \times C'} \rightarrow \mathbb{R}^{H \times W \times C}$

› Squeezing: $z = F_{sq}(U)$: $\mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{1 \times 1 \times C}$

› Excitation: $s = F_{ex}(U)$: $\mathbb{R}^{C} \rightarrow \mathbb{R}^{C}$

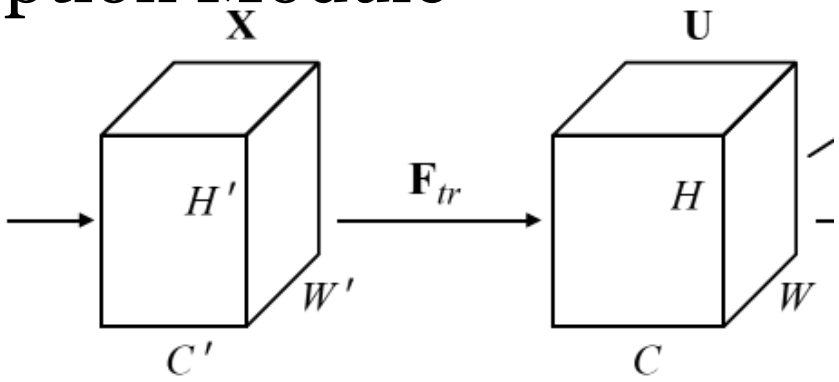
SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Transform: 3D Convolution

› Transform: $U = F_{tr}(X)$: $\mathbb{R}^{H' \times W' \times C'} \rightarrow \mathbb{R}^{H \times W \times C}$

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s.$$

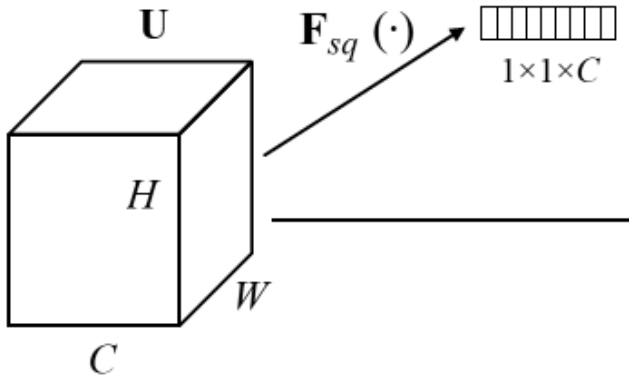
› Implement by Residual or Inception Module



SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Squeeze: Global Information Embedding

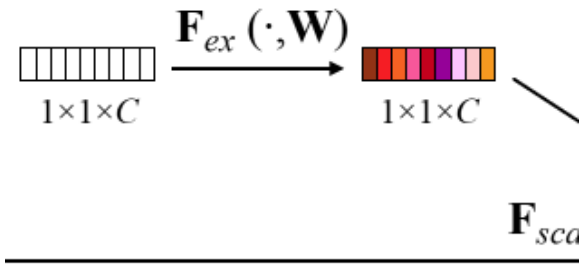
$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j).$$



› Global pooling, A Channel Descriptor!

SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Excitation (SE) Block:



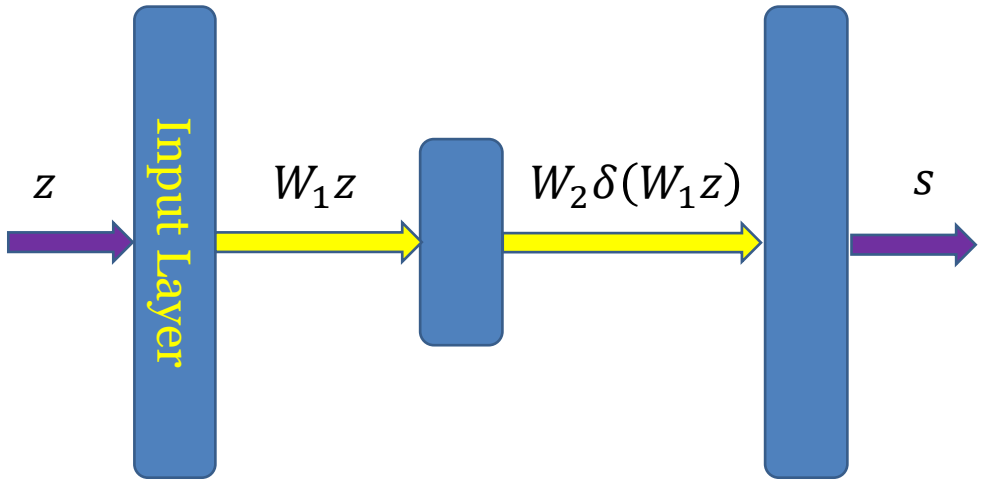
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z})),$$

› $W_1 \in \mathbb{R}^{\frac{C}{r}\times C}$: FC

› δ: ReLU

› $W_2 \in \mathbb{R}^{C\times\frac{C}{r}}$: FC

› σ: Sigmoid

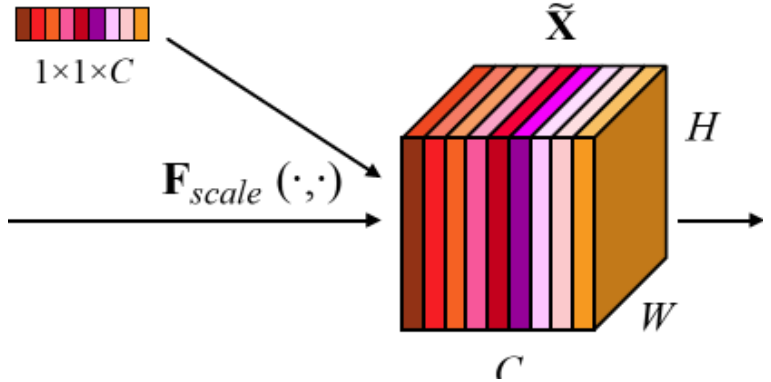


SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Scaling:

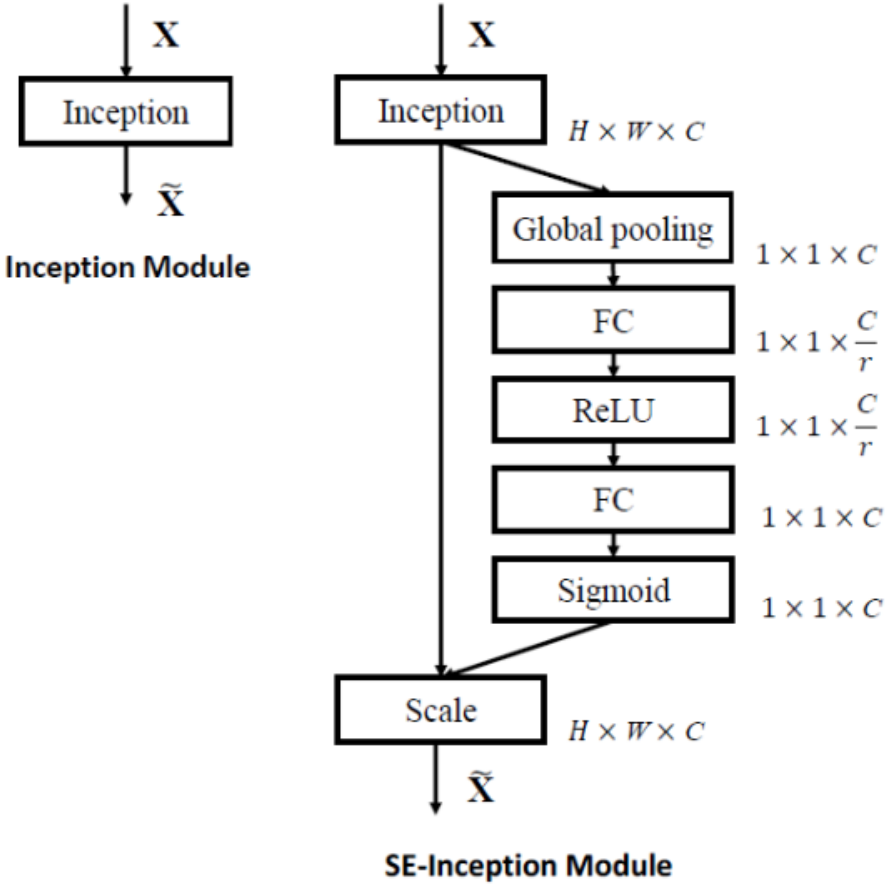
› Channel-wise multiplication

› S_i (scalar) multiply by $X(:,:,i) \rightarrow \tilde{X}(:,$



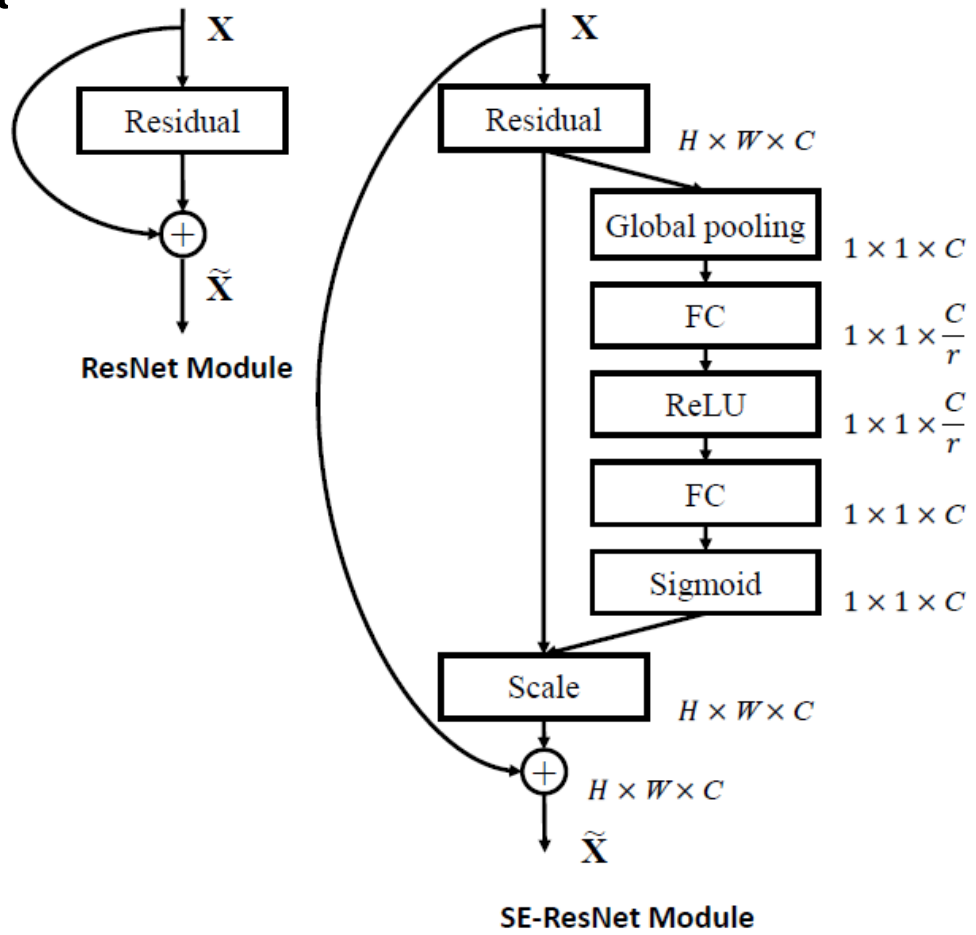
SENet - Squeeze-and-Excitation Networks, CVPR 2017

› SE-Inception



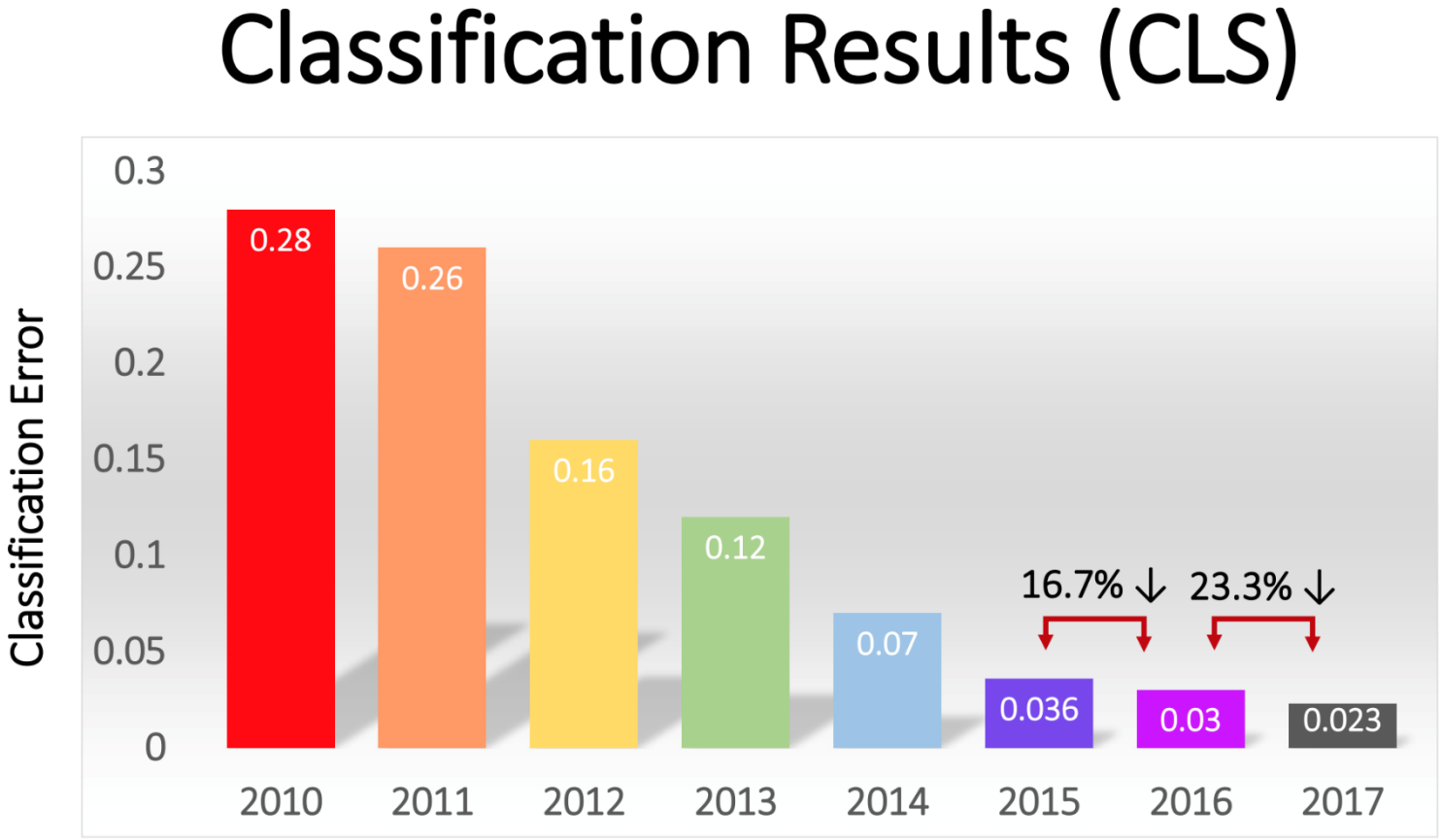
SENet - Squeeze-and-Excitation Networks, CVPR 2017

› SE-Residua[1]



SENet - Squeeze-and-Excitation Networks, CVPR 2017

› Results



CNN for Signal Processing

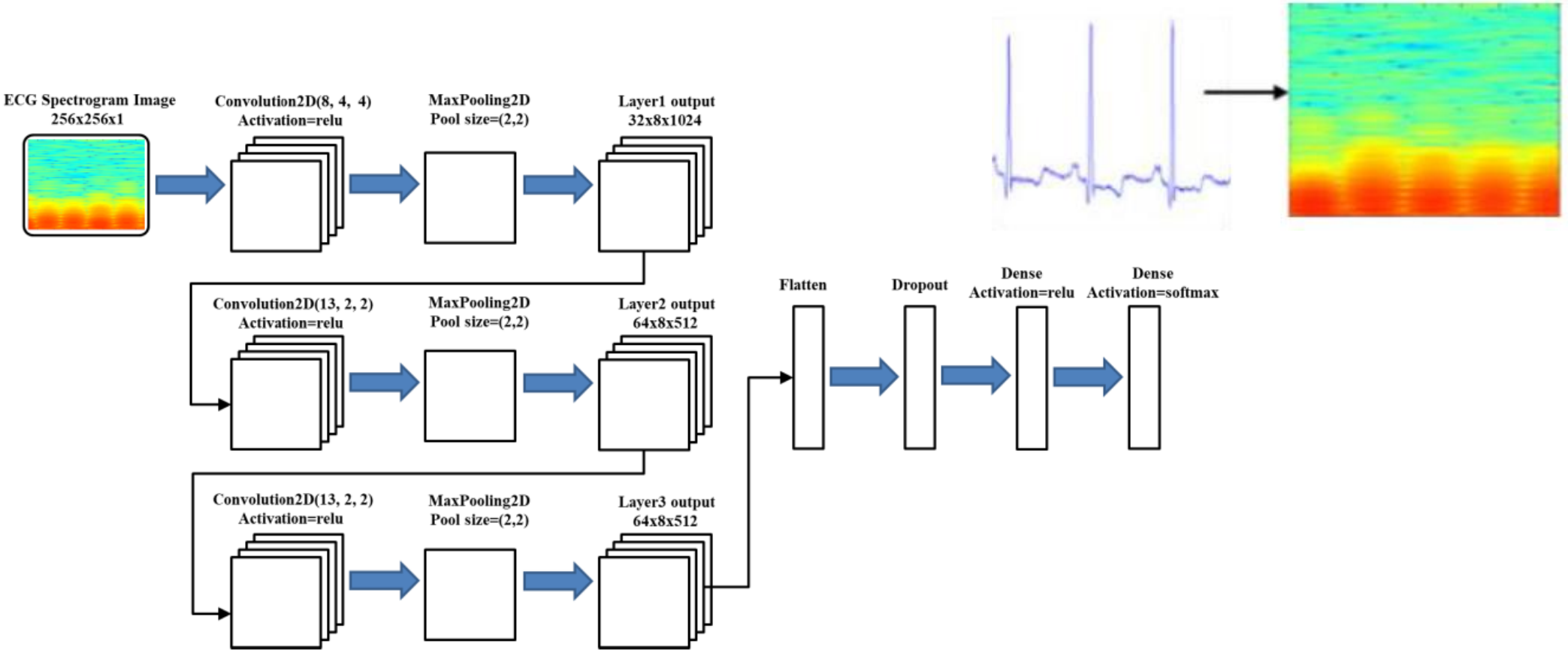
- CNN and 1D signals:
  - 2D reshape → Conventional CNN architecture:
    - Multichannel Signals (Biological signals, market data, array sensor., ….)
    - Group of Subjects
    - 2D (time and transform/feature/…) representation (STFT[1], EMD[2], …)
  - 1D CNN:
    - One dimensional Convolution, Pooling, …..
- Time series deep structure (RNN, LSTM, …)

1 – Short Time Fourier Transform

2 - Empirical Mode Decomposition

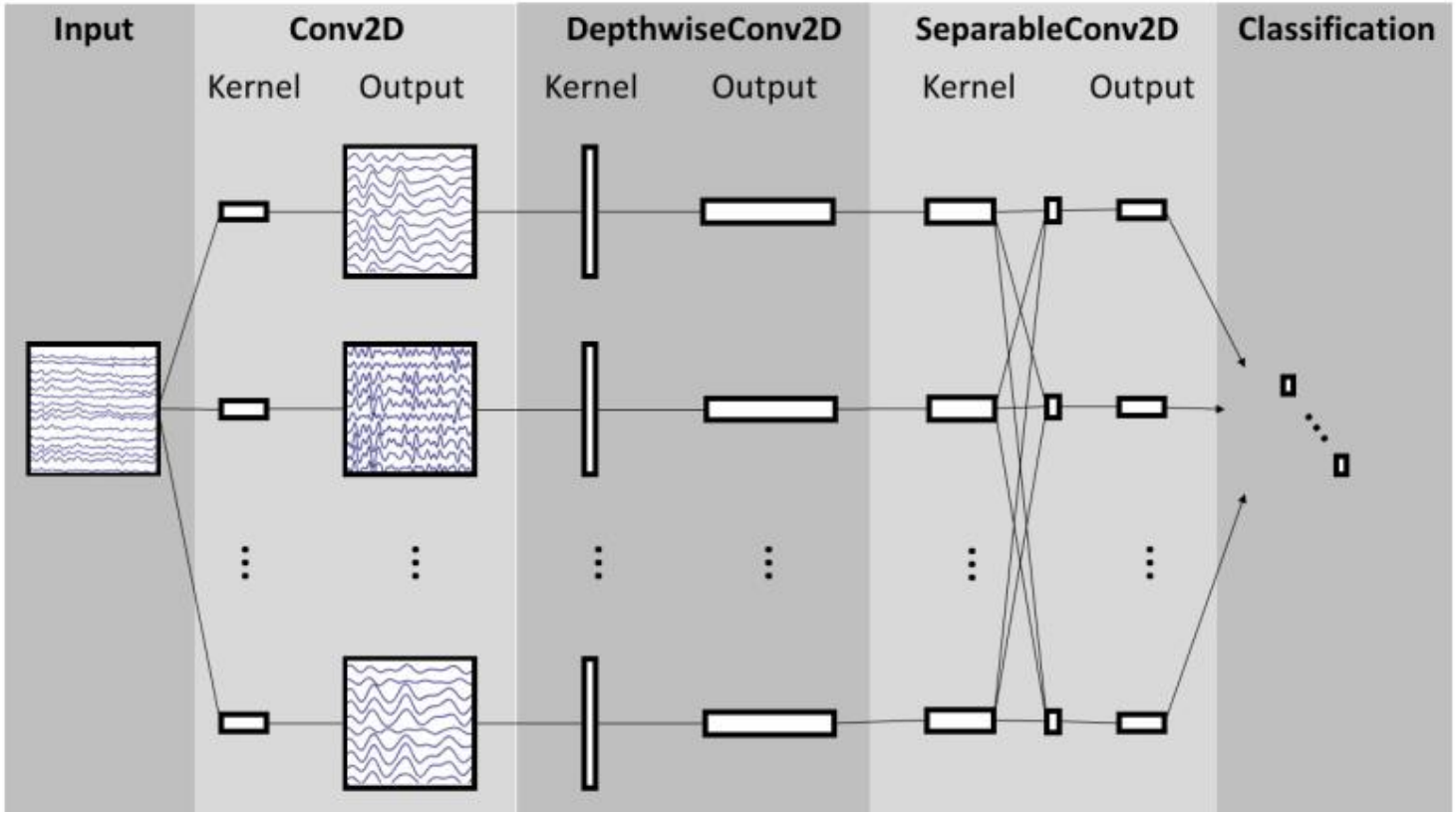
CNN for Signal Processing - ECG Arrhythmia Classification Using STFT-Based Spectrogram and CNN - 2019

› 2D representation via STFT



CNN for Signal Processing - EEGNet: A Compact Convolutional Neural Network for EEG-based Brain-Computer Interfaces

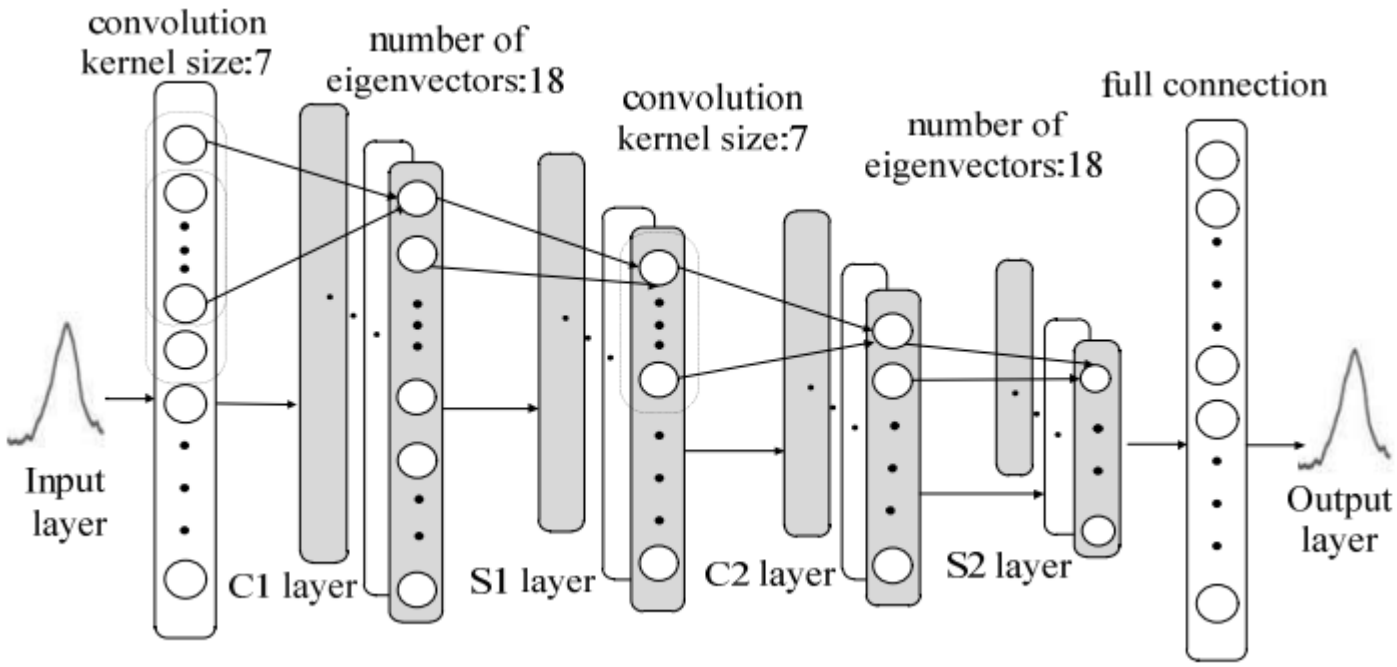
› 2D representation via Multichannel recording



CNN for Signal Processing - Classification of ECG Signals

Based on 1D Convolution Neural Network

› 1D Signal processing

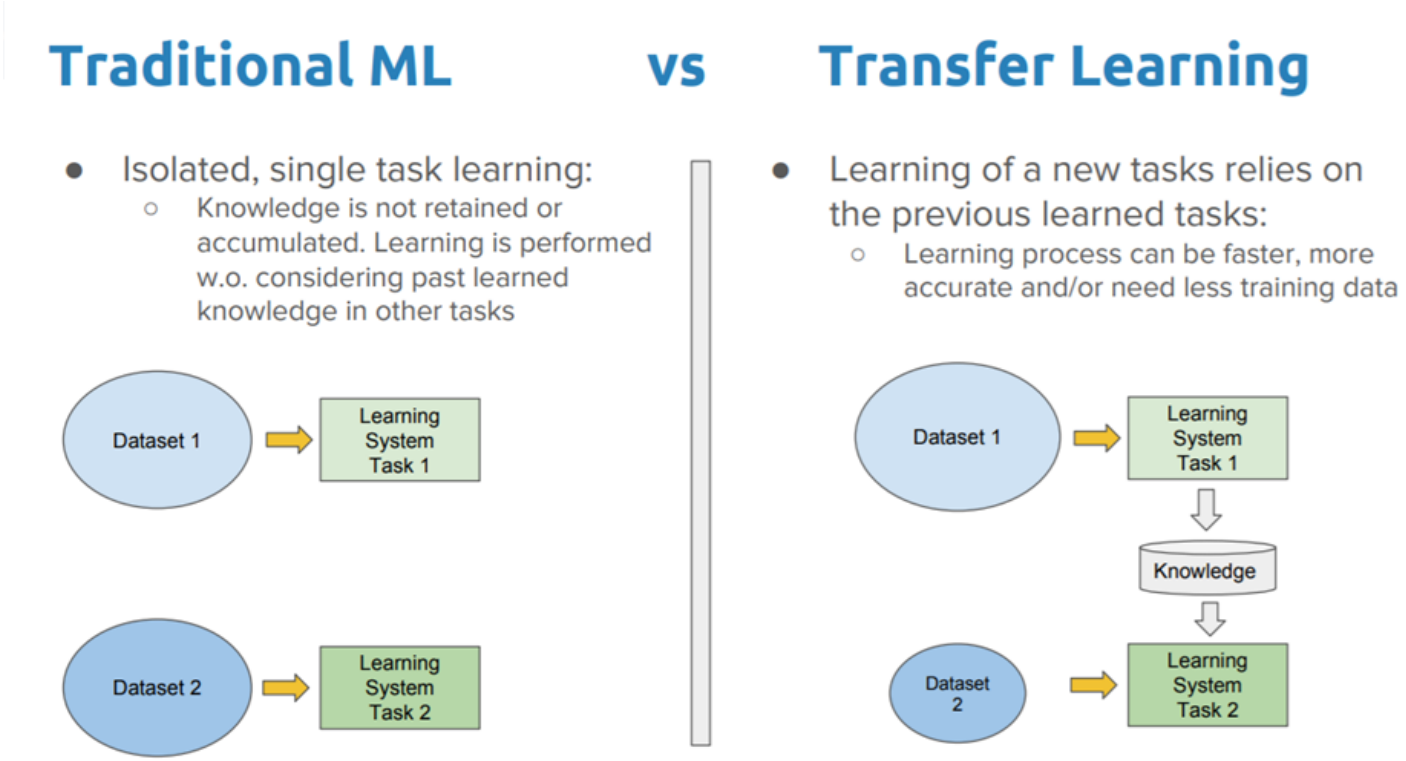


Transfer Learning

- A trained model one dataset#1 used for decision on dataset#2
- Vocabulary
  - Pre-trained Model
  - Fine-Tuning
  - Freezing

Transfer Learning

› Definition:



Transfer Learning

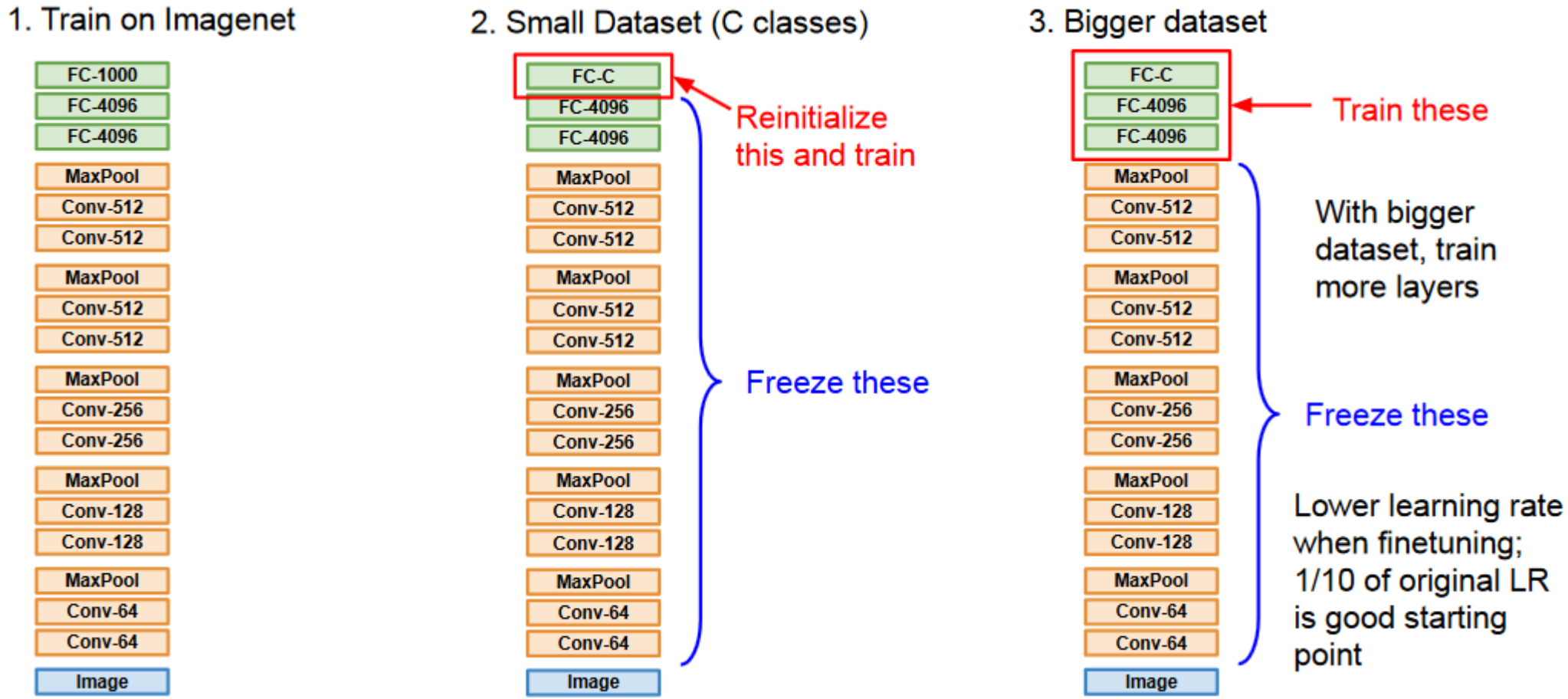
- Methodology in CNN:
  - ConvNet as fixed feature extractor, remove FC layer (4096-d feature vector)
  - Fine-tuning the ConvNet, fine-tune the weights of the pre-trained network by continuing the backpropagation with new data (all or some layers)
  - Pretrained models: Use without modification

Transfer Learning - http://cs231n.github.io/transfer-learning/#tf

› When and How:

1. New dataset is small and similar to original dataset:
   - Use final feature layer and train a linear or simple classifier
2. New dataset is large and similar to the original dataset:
   - Fine-tune the full network
3. New dataset is small but very different from the original dataset:
   - Train a SVM classifier from activations (middle layer, not final) in the network
4. New dataset is large and very different from the original dataset:
   - Train from scratch (Initialize with weights from a pre-trained model)

Transfer Learning - cs231n_2019_lecture09.pdf



CNN - Summary

- This is never-end story!
- Lots of models and growing
- Depth is growing!
- New architecture are developing!
- ResNet and SENet: good choice!
- Model Zoo:
  - https://modelzoo.co/framework/tensorflow
  - https://github.com/BVLC/caffe/wiki/Model-Zoo
  - https://github.com/onnx/models
- New sight: Capsule Net, Meta Learning (Learn the architecture, …)