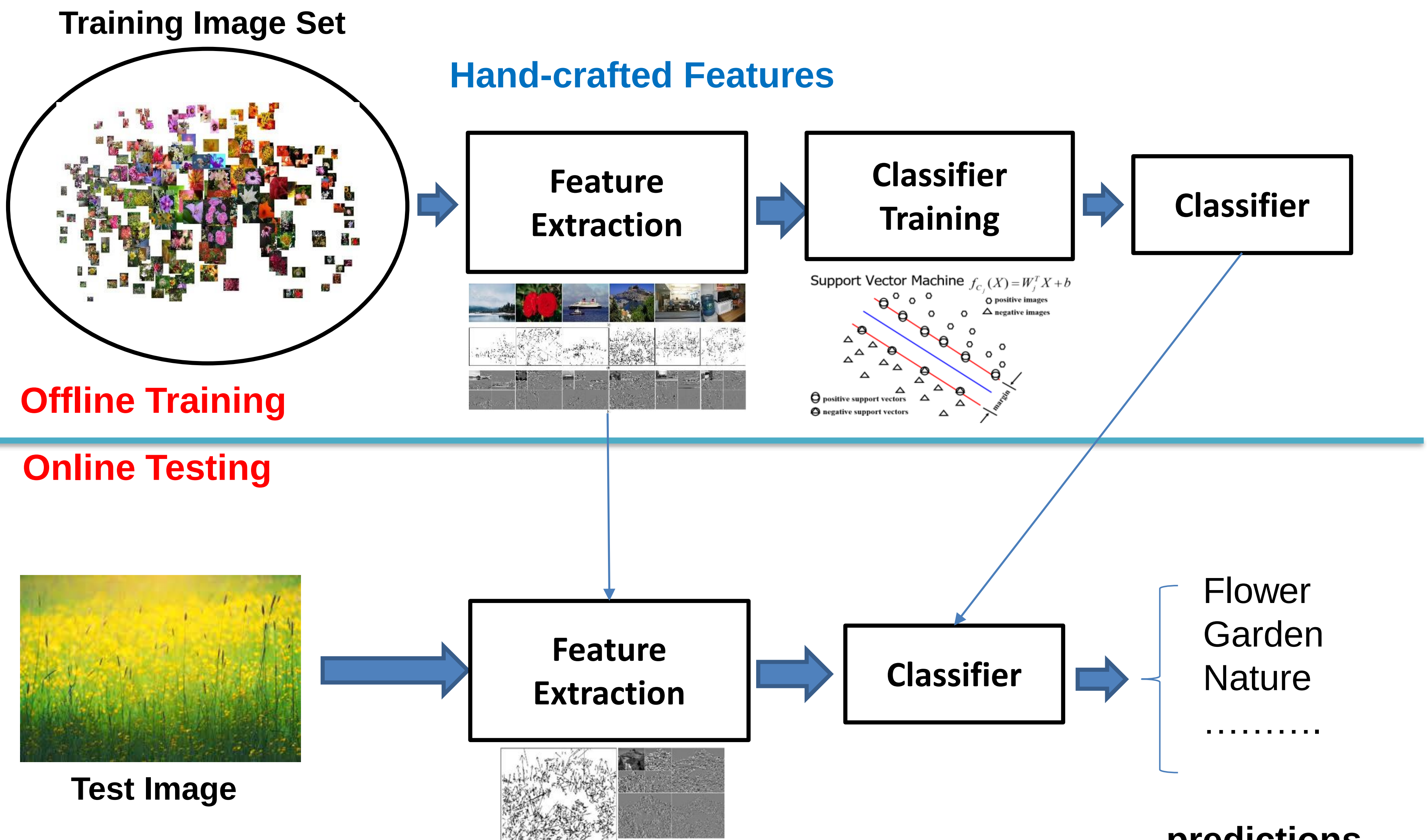
Deep Networks for Image Classification

Jianping Fan
Dept of Computer Science
UNC-Charlotte

Course Website:
http://webpages.uncc.edu/jfan/itcs5152.html

Pipeline for Traditional Image Classification System



Separating feature extraction from classifier training

Feature Extractor & Classifier

Class Probabilities

Dog (0.7)
Cat (0.1)
Bike (0.02)
Car (0.02)
Plane (0.02)
House (0.04)

Review for Feature Extraction: Hand-Crafted Features

- **Hand-crafted features from neighboring pixels: operations over 7x7, 5x5, 3x3 neighboring blocks because image pixels are spatially correlated!**

Review for Feature Extraction: Hand-Crafted Features

- Hand-crafted features from neighboring pixels: operations over 7x7, 5x5, 3x3 neighboring blocks because image pixels are spatially correlated!
- We expect that such features are scale, translation, even affine transformation invariant!

Review for Feature Extraction: Hand-Crafted Features

- **Hand-crafted features from neighboring pixels: 7x7, 5x5, 3x3 neighboring blocks!**
- **Such features are transformation-invariant!**
- **Feature quality is most important! Feature quality is more important than classifier!**

Review for Feature Extraction: Hand-Crafted Features

- Hand-crafted features from neighboring pixels: 7x7, 5x5, 3x3 neighboring blocks!
- Such features are transformation-invariant!
- Feature quality is important than classifier!
- Feature dimensions are meaningful for classifier! Dimension reduction should be there!

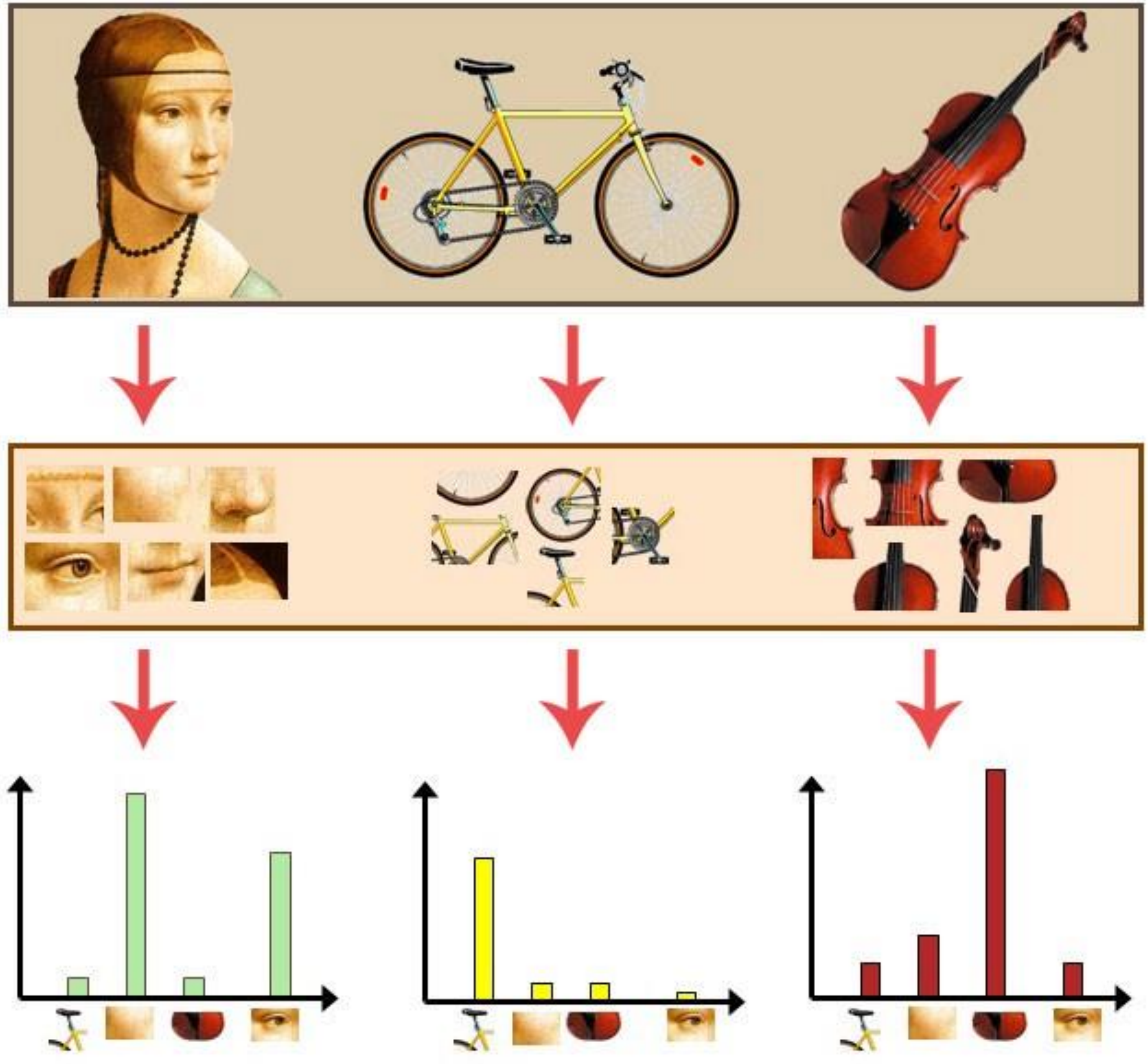
Review for Feature Extraction: Hand-Crafted Features

- Hand-crafted features from neighboring pixels: 7x7, 5x5, 3x3 neighboring blocks!
- Such features are transformation-invariant!
- Feature quality is important than classifier!
- Feature dimension reduction!
- High-level features vs. low-level features! Feature extraction from objects, parts of objects, ………

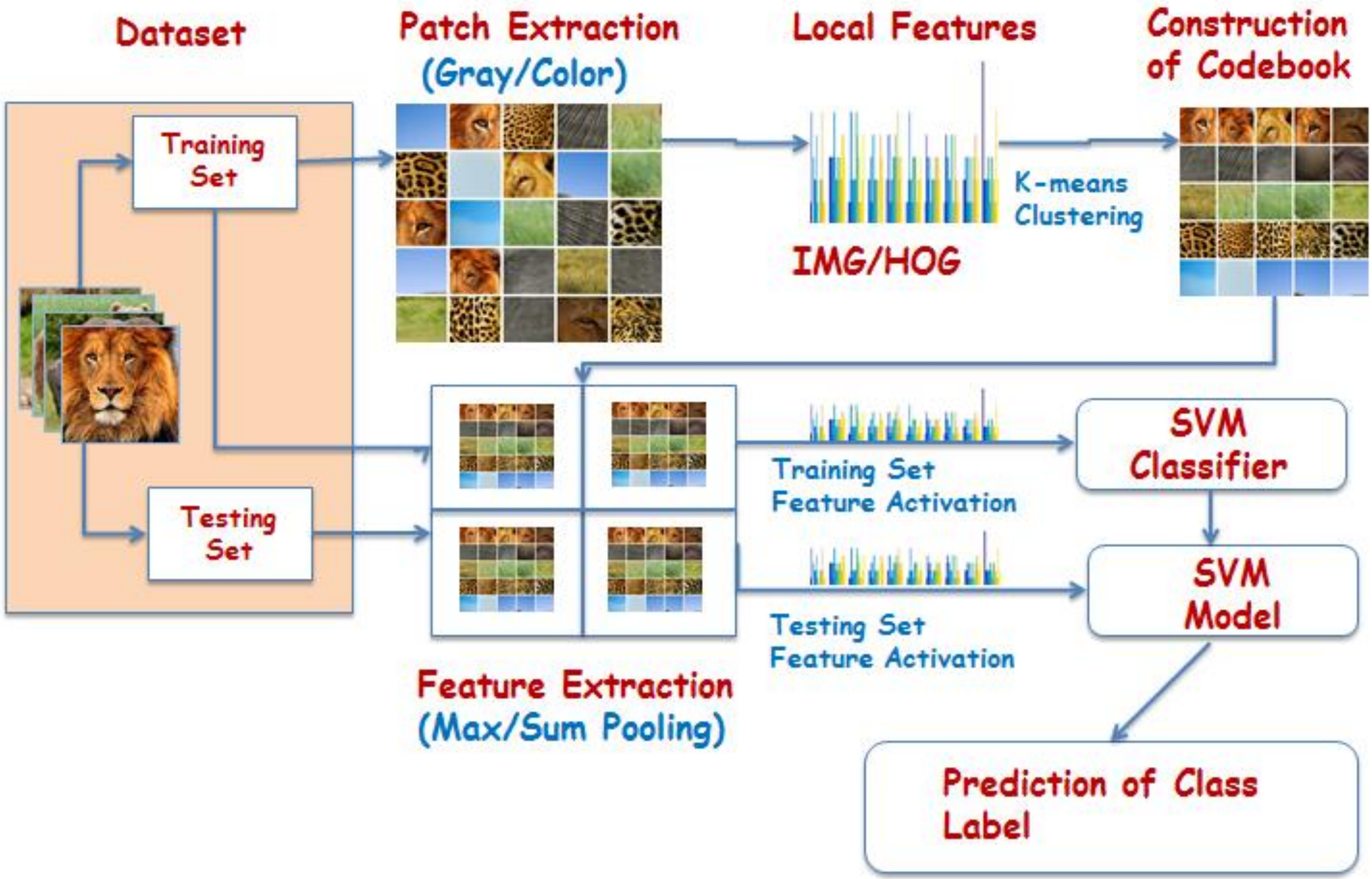
Review for Feature Extraction: Hand-Crafted Features

- Hand-crafted features about neighborhoods: 7x7, 5x5, 3x3 neighboring blocks!
- Such features are transformation-invariant!
- Feature quality is important than classifier!
- Feature dimension reduction!
- Semantics-driven features

Bag of Visual Words



Patch-based Visual Features

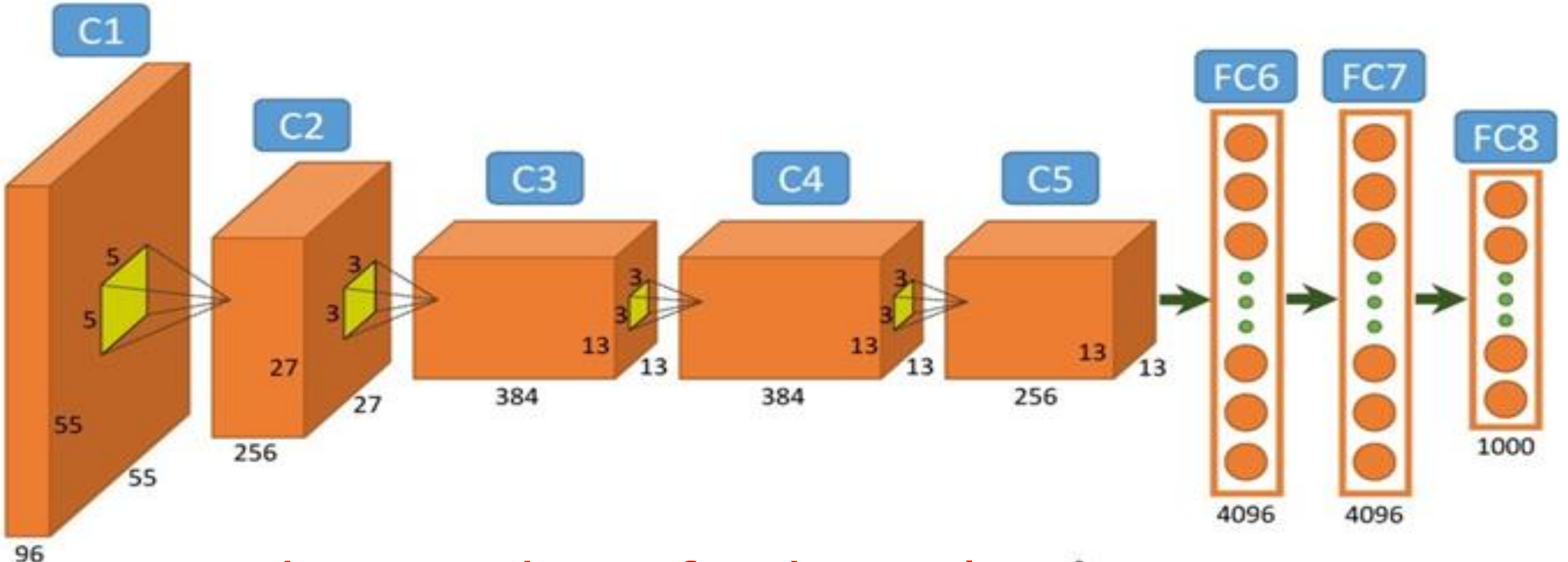


Review for Feature Extraction: Hand-Crafted Features

- Hand-crafted features from neighboring pixels: 7x7, 5x5, 3x3 neighboring blocks!
- Such features are transformation-invariant!
- Feature quality is important than classifier!
- Feature dimension reduction!
- Semantics-driven features
- Feature normalization
- ----------

Deep Learning Approach

Joint process for **feature learning** & **classifier training**

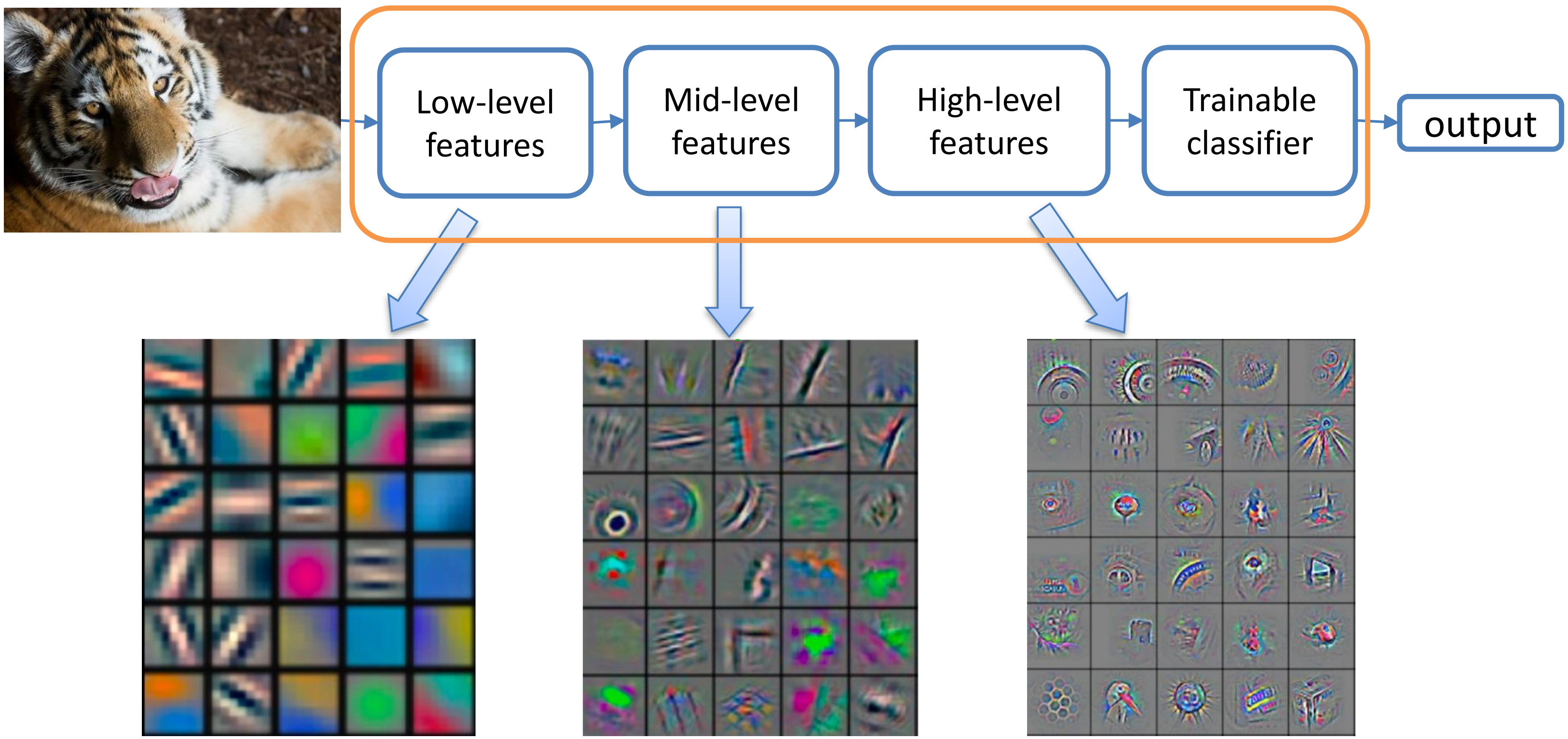


Let data speak out for themselves!

SGD for back-propagation

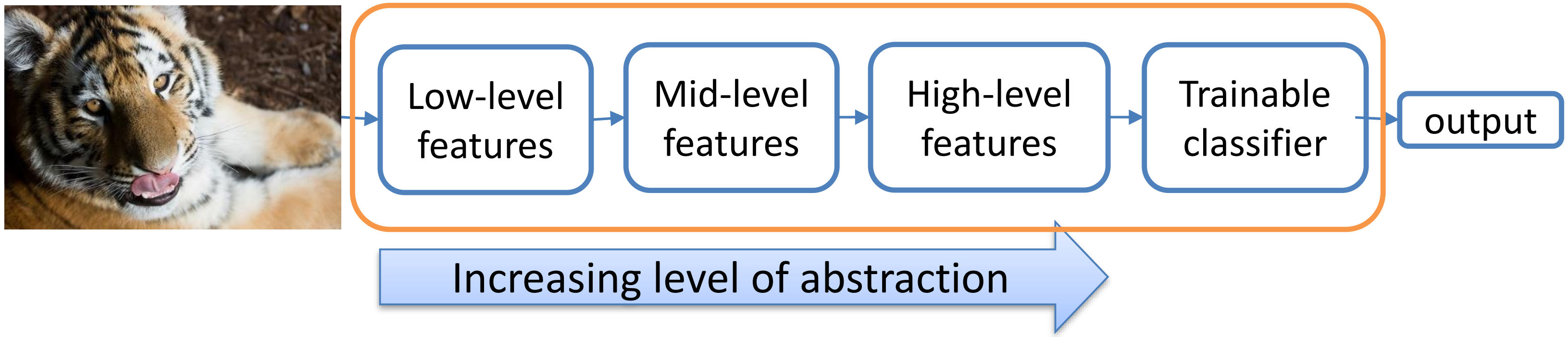
Deep Learning

- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features at multiple levels) automatically through multiple stage of feature learning process.



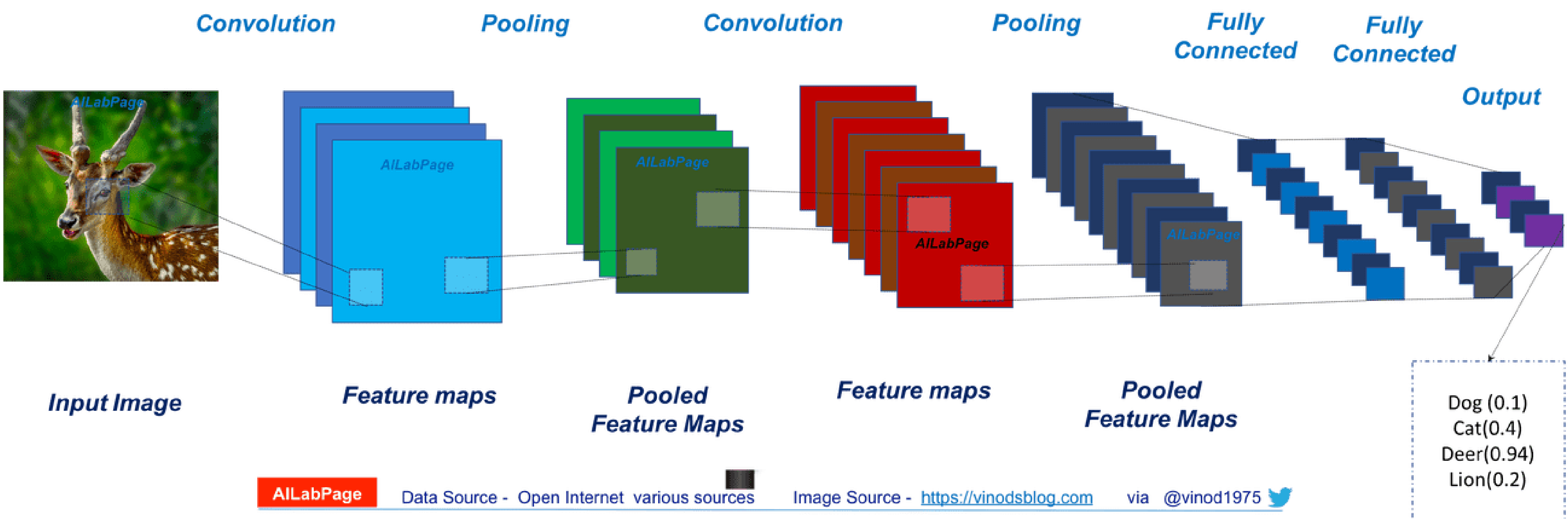
Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

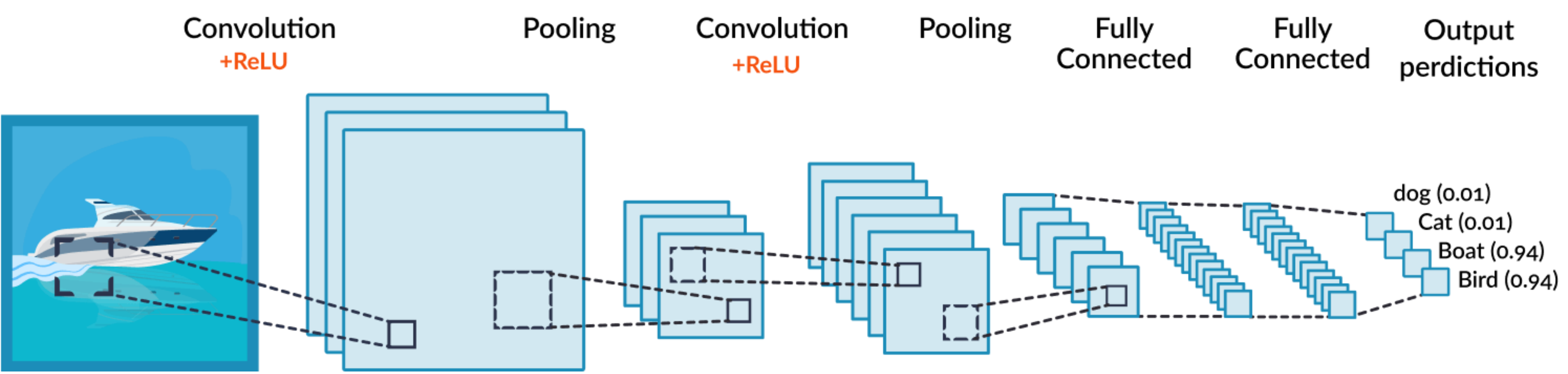
Learning Hierarchical Representations



- Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable nonlinear feature transform
- Hierarchical Image Representation & Recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story

Convolution Neural Network





Key Operations

1. Convolution, 2. ReLU, 3. Pooling, 4. Softmax

5. Data Augmentation, 6. Fine-tune, 7. Batch Normalization, 8. Drop out

Convolutional Neural Network (CNN)

- A standard CNN for image classification is composed of:
  - **Convolutional layers**
  - **Down-sampling layers**
    - Strided convolution
    - Max **pooling**
    - Avg. Pooling
  - **Batch normalization**
  - **Activation functions (e.g. ReLU)**

Basic Operators for CNN

1. Convolution;
2. ReLU
3. Pooling
4. Softmax

Basic Tools for CNN Training

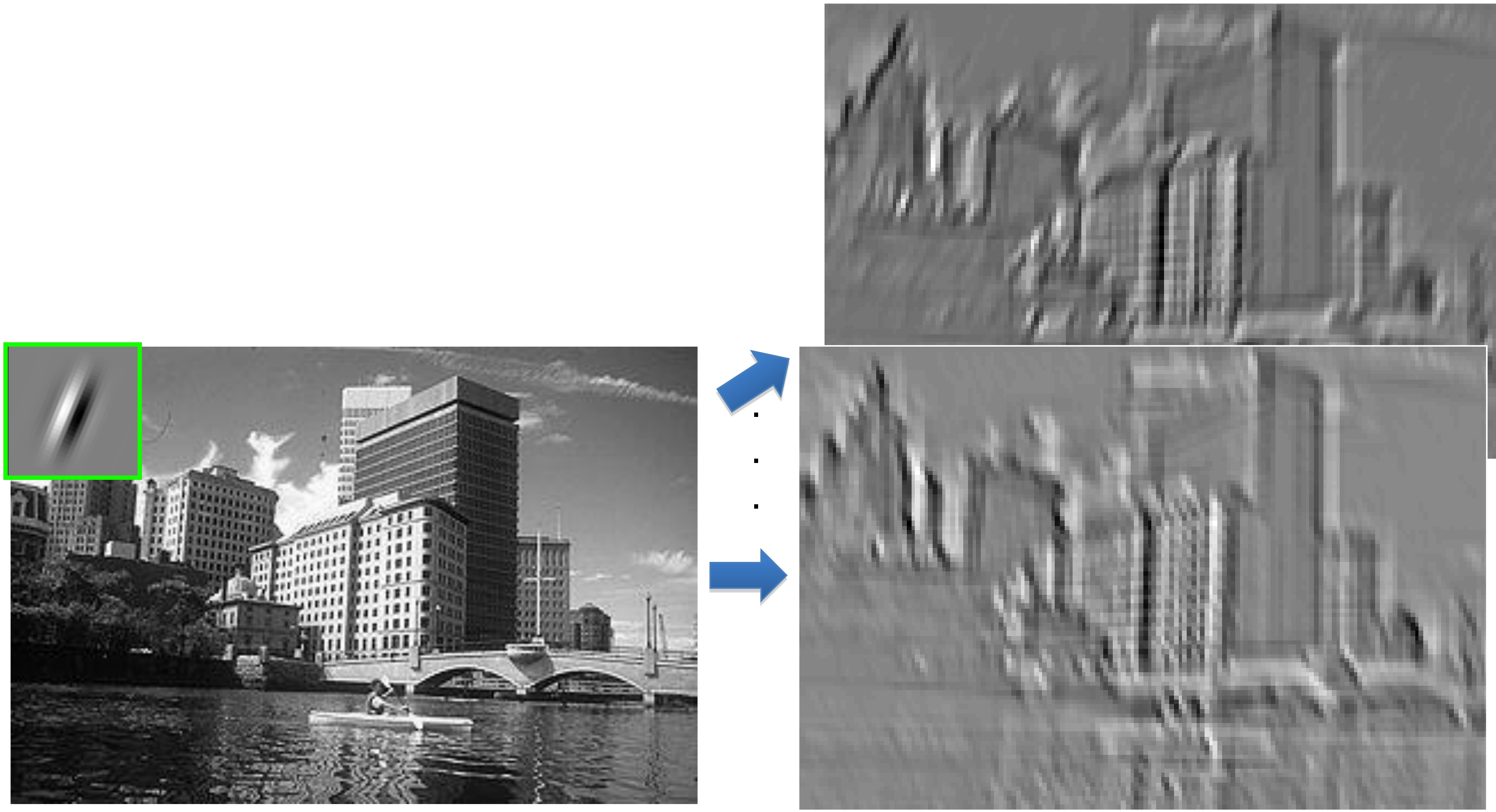
1. Data Augmentation
2. Fine-tune
3. Batch Normalization
4. Drop-out

Convolutional Layer

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

Convolution works on neighboring pixels!

Convolution as neighbor-based feature extraction



Input

Feature Map

Slide: Lazebnik

Discrete convolution

- A discrete convolution is a linear transformation
  - Sparse – only few inputs contribute to a given output unit
  - Reuses parameters – same kernel is applied over multiple input elements

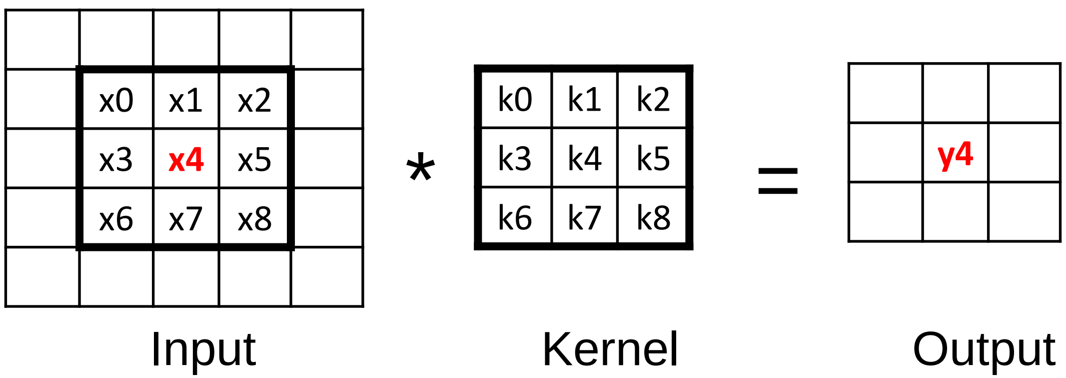


Figure: In this example, each output element is computed using 9 pixels

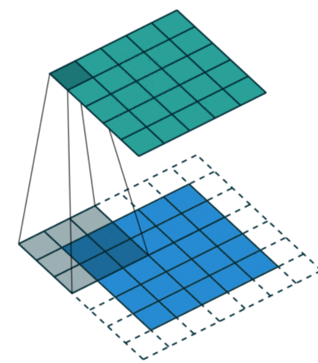
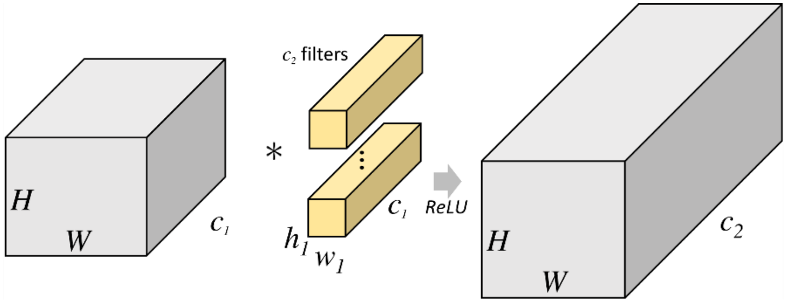


Figure: Kernel strides over input

Convolution Layer

- Convolution layer takes an input feature map of dimension $W \times H \times N$ and produces an output feature map of dimension $\widehat{W} \times \widehat{H} \times M$
- Each layer is defined using following parameters:
  - # Input channels (N)
  - # Output channels (M)
  - **Kernel size**
  - Padding
  - Stride



# of parameters learned by convolution layer is n^2NM

Convolution Layer

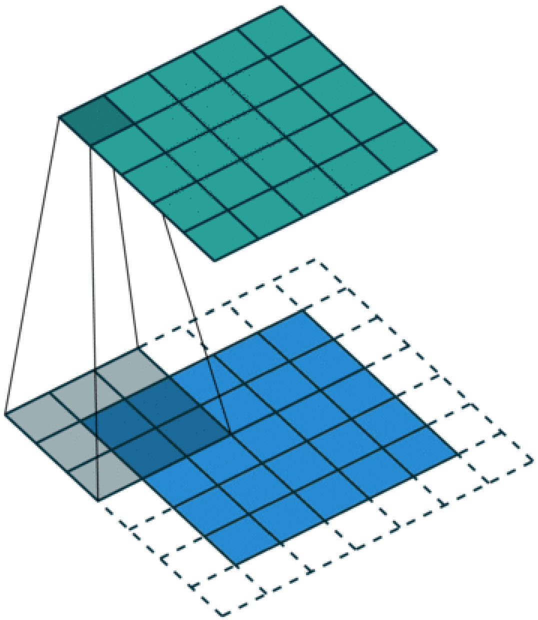


Figure: In this example, 5x5 input is convolved with 3x3 kernel with stride=padding=1 to produce an output of size 5x5.

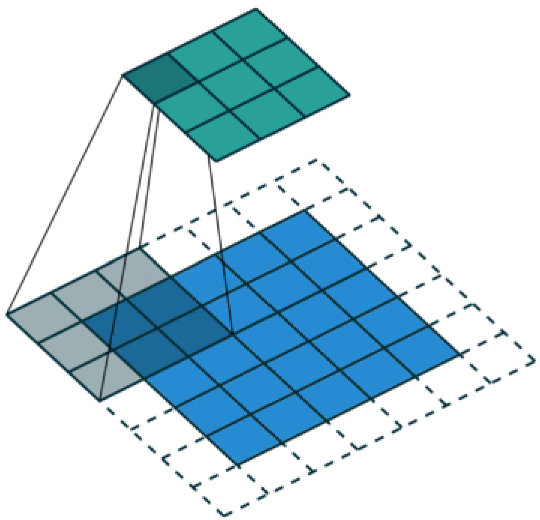
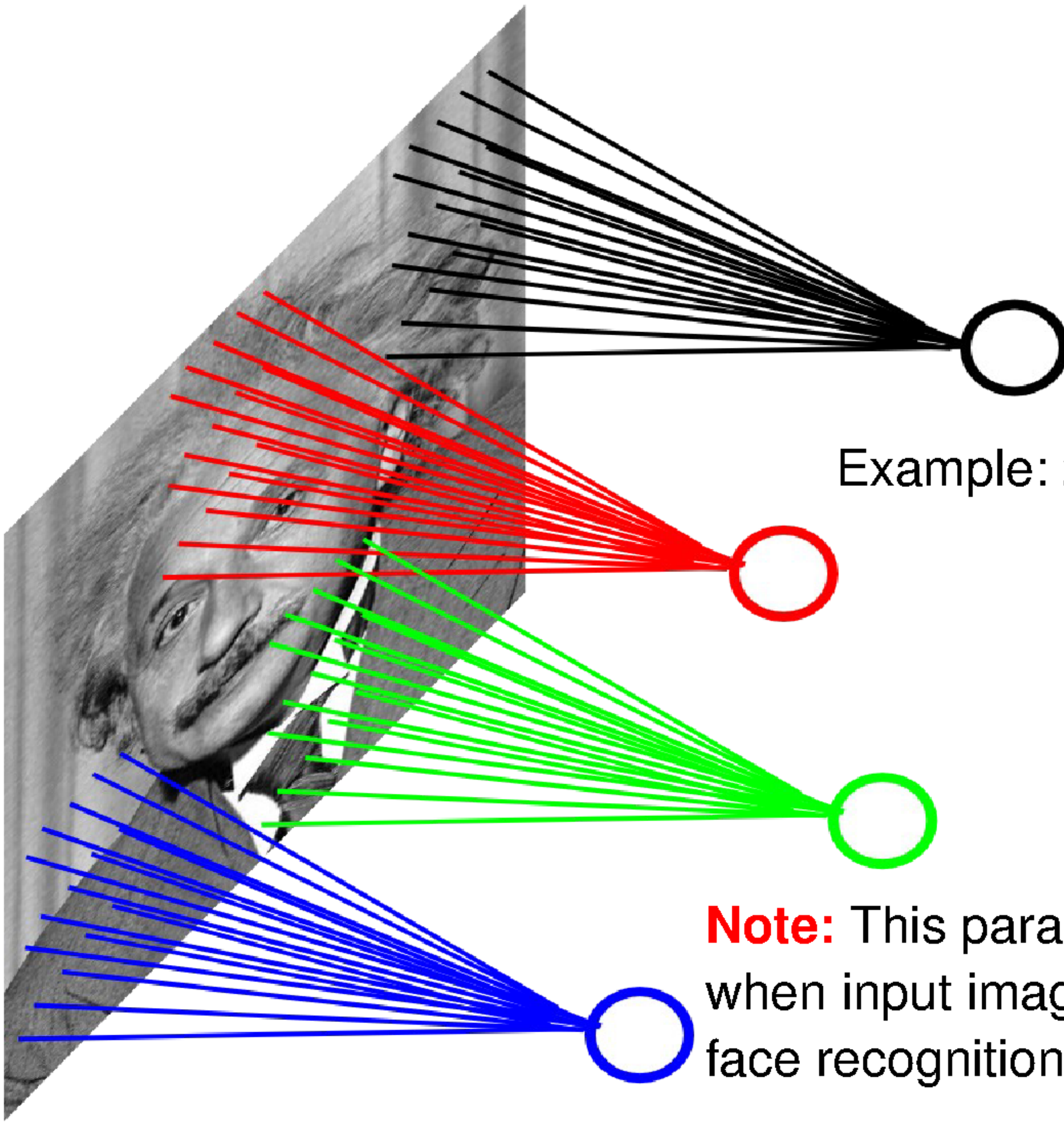


Figure: In this example, 5x5 input is convolved with 3x3 kernel with **stride=2** and padding=1 to produce an output of size 3x3.

Locally Connected Layer

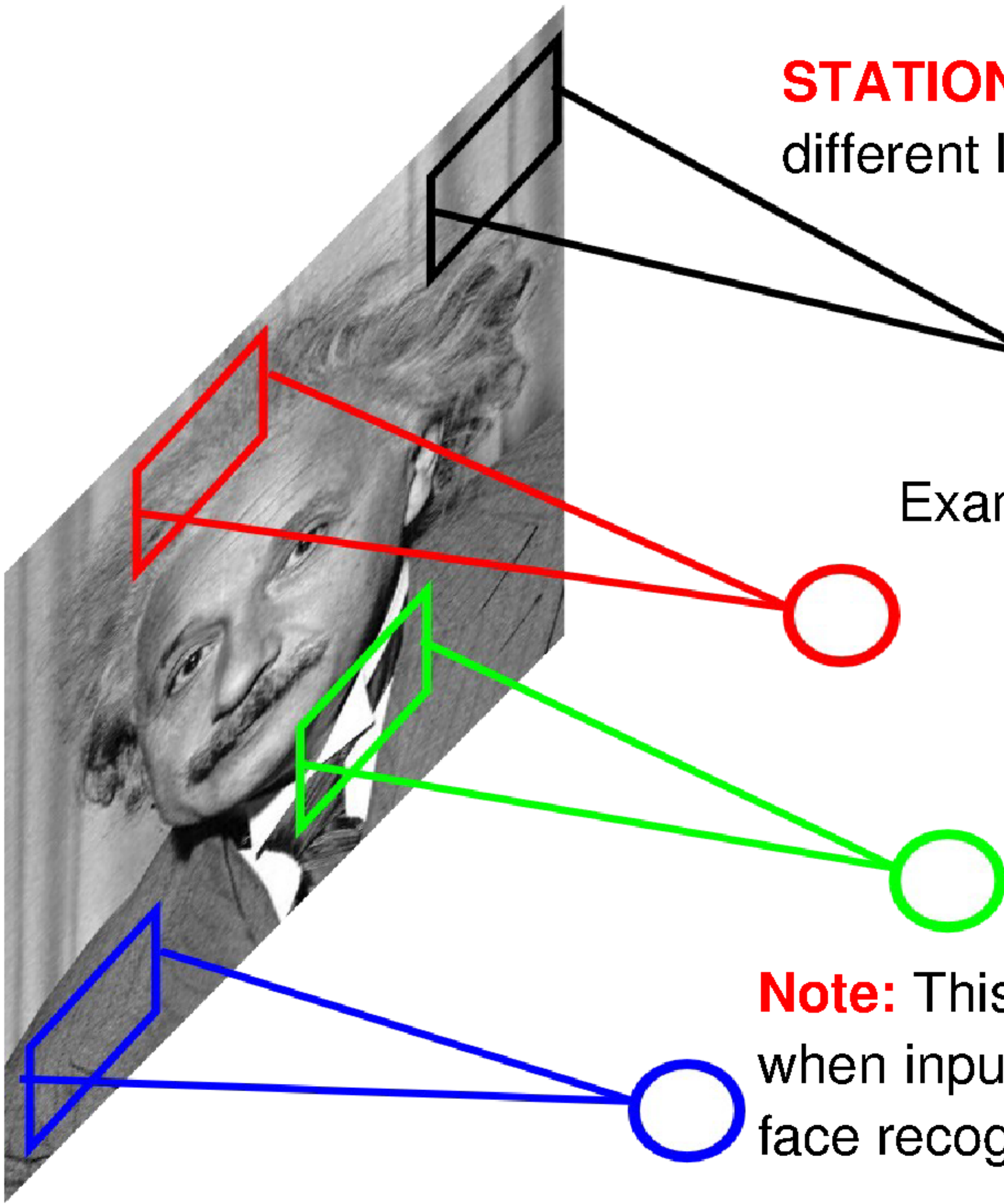


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Ranzato

Locally Connected Layer



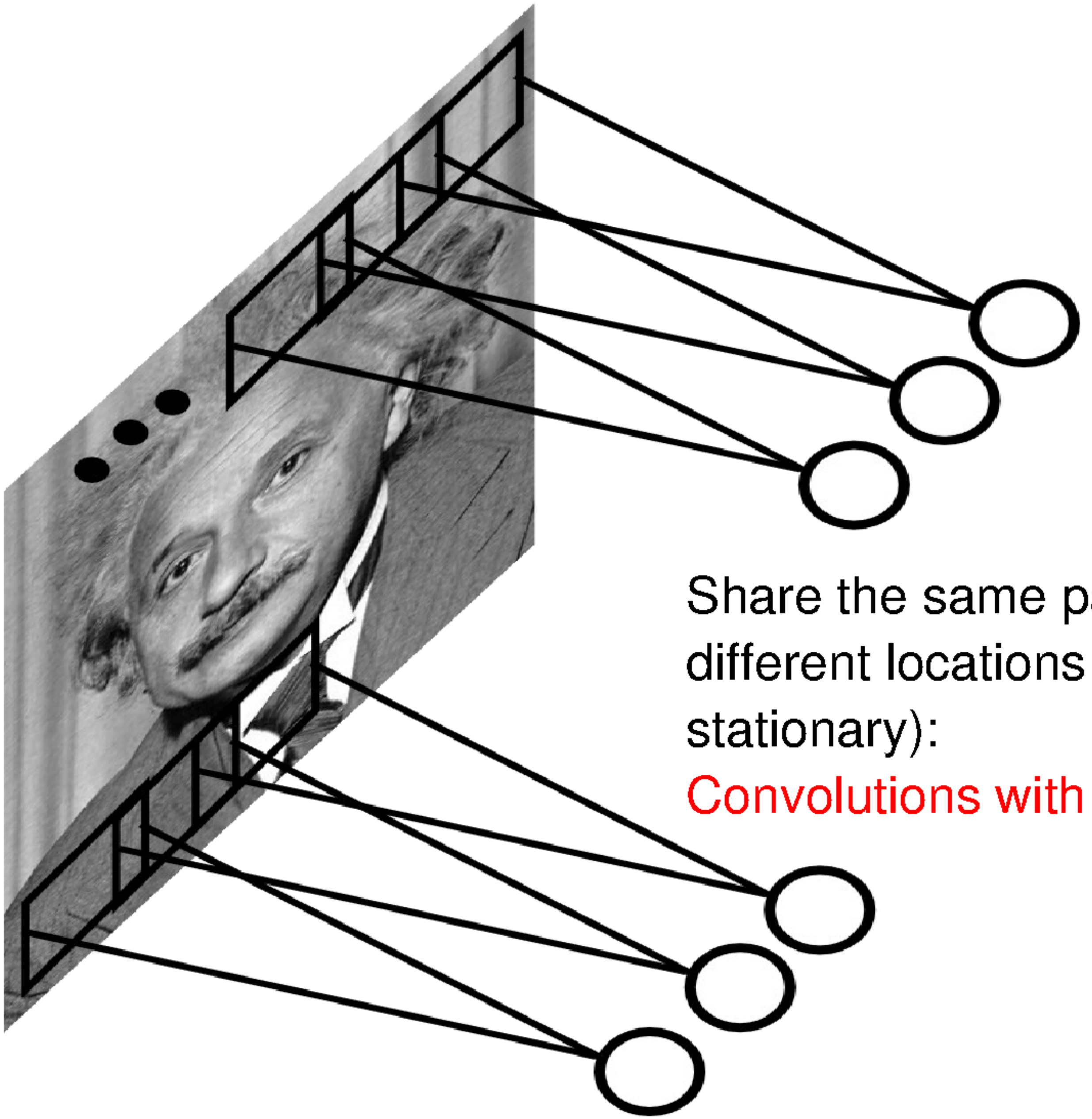
STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

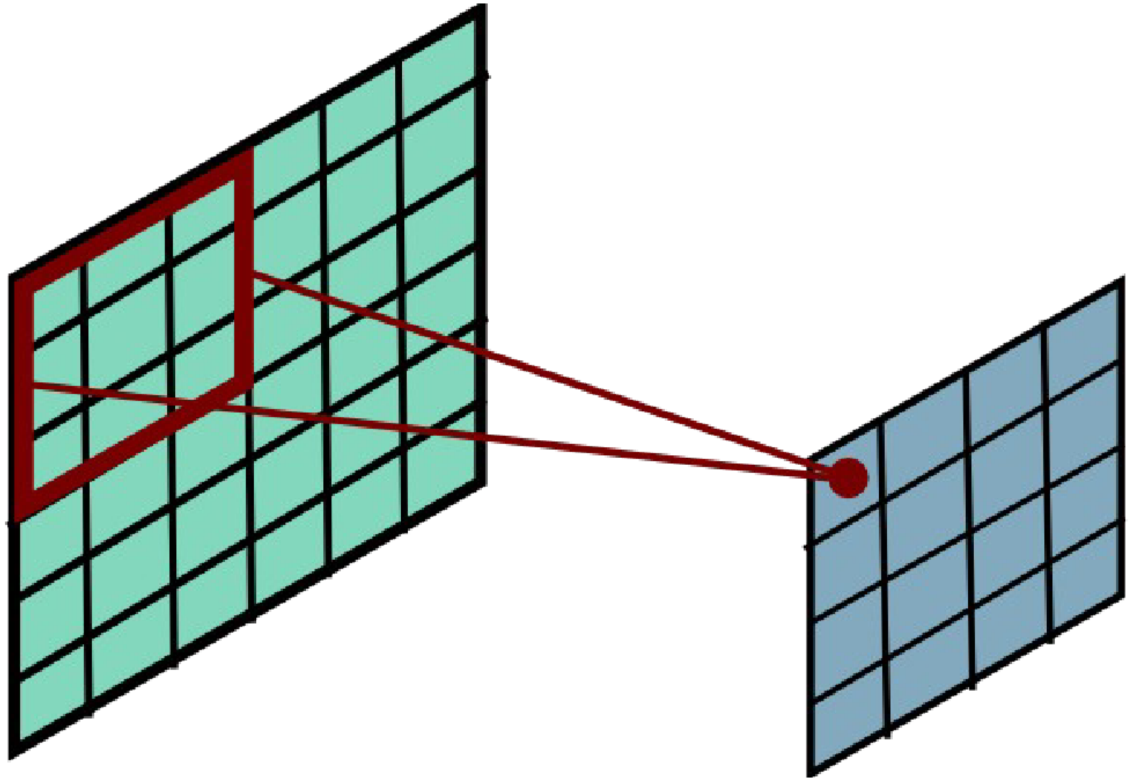
Ranzato

Convolutional Layer



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

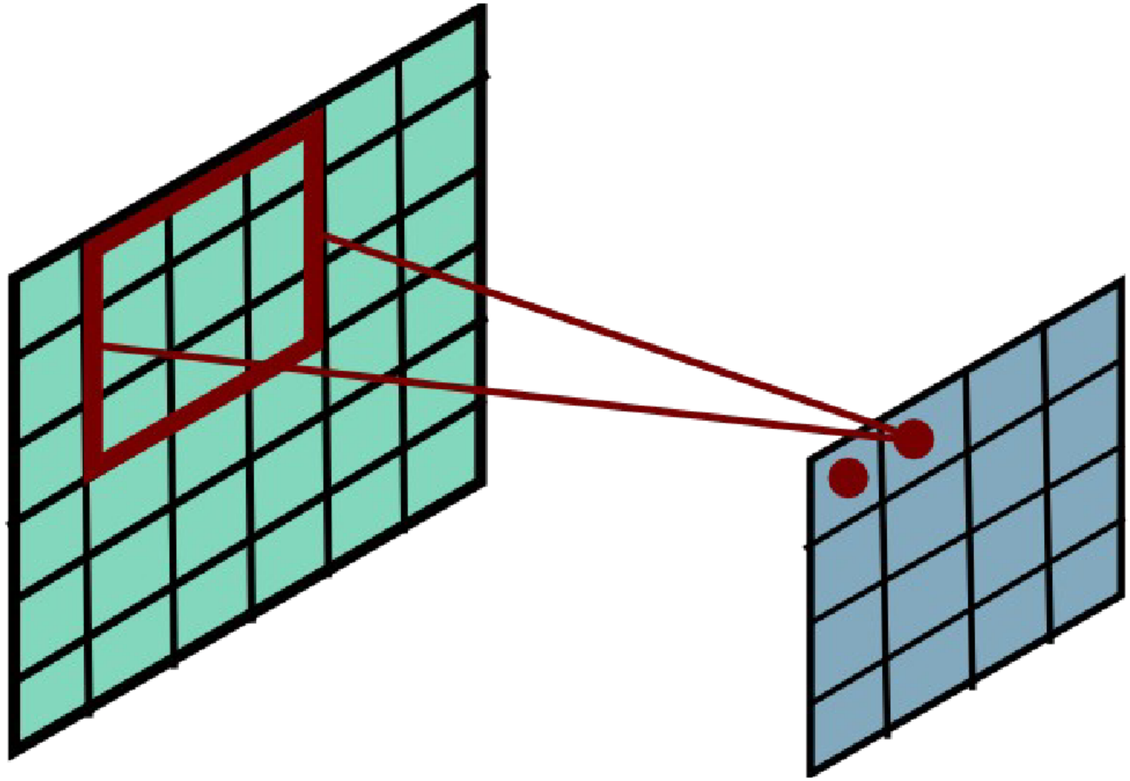
Convolutional Layer



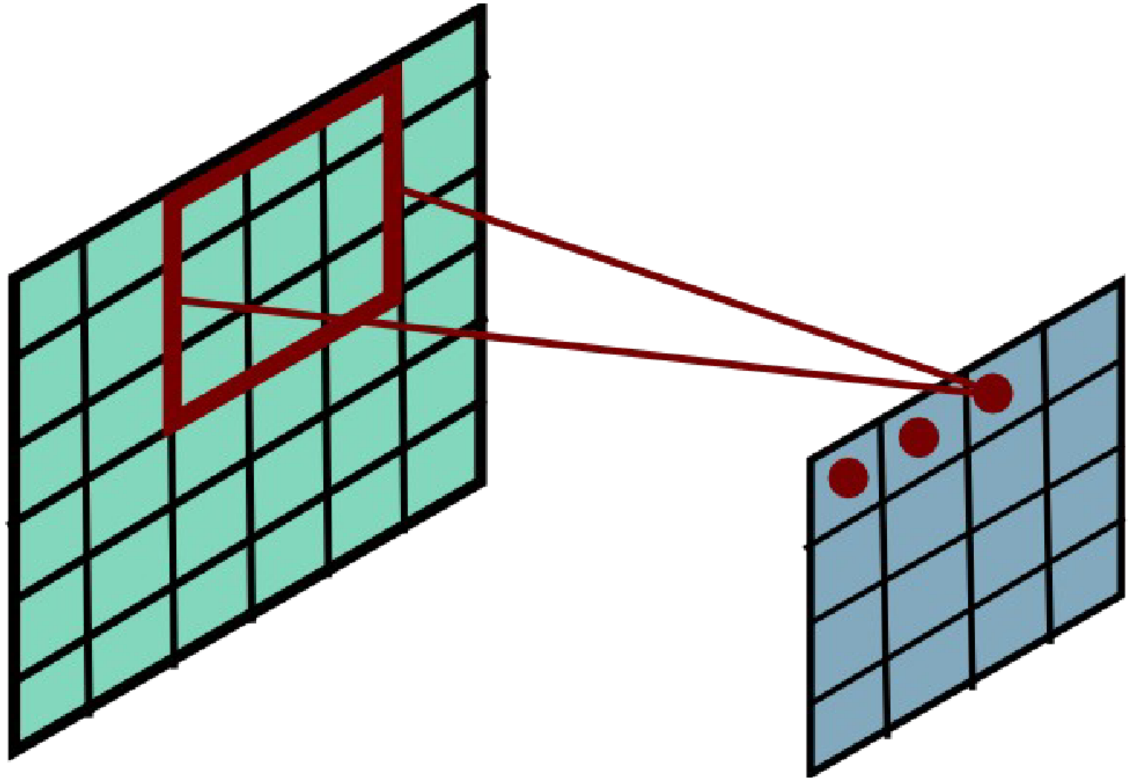
Slide Credit: Marc'Aurelio Ranzato

Ranzato

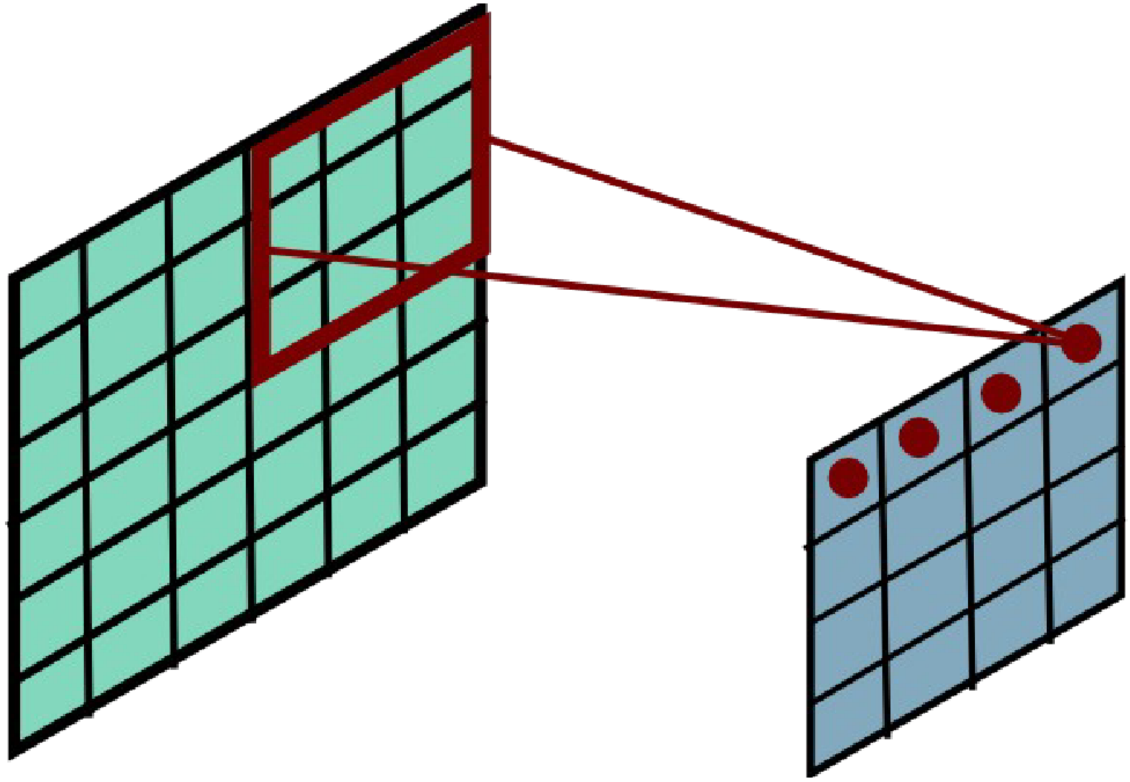
Convolutional Layer



Convolutional Layer

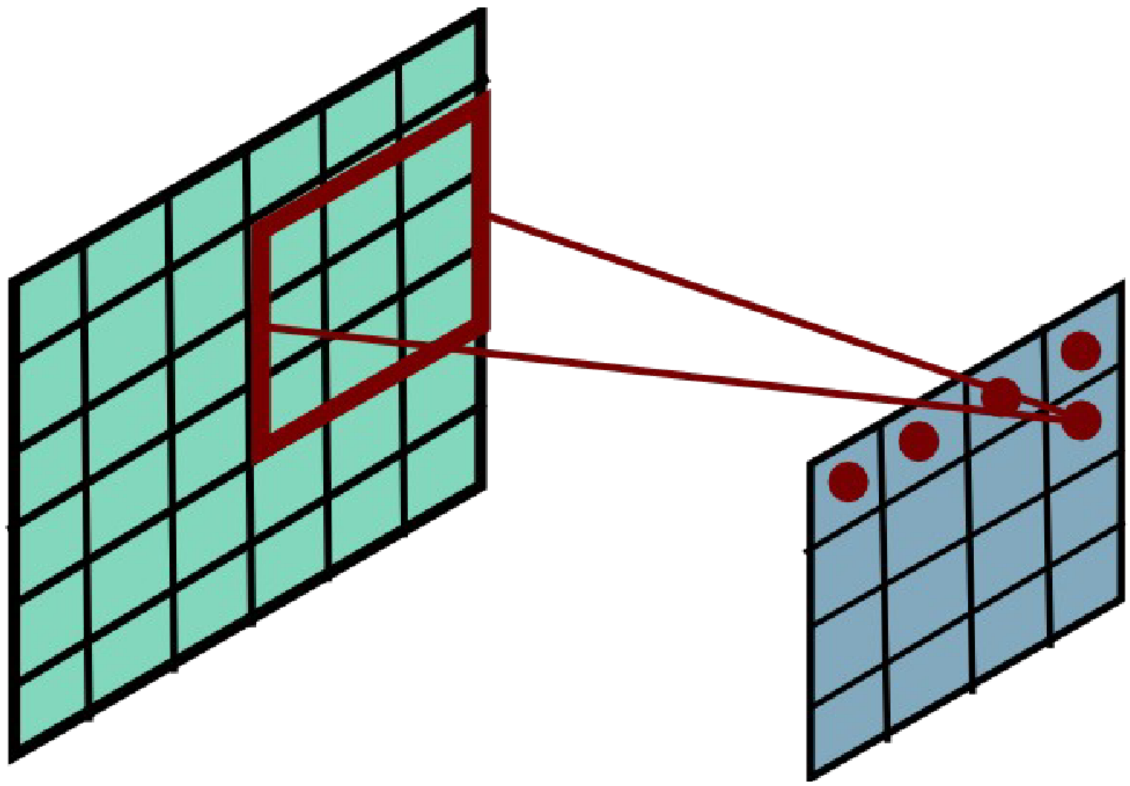


Convolutional Layer

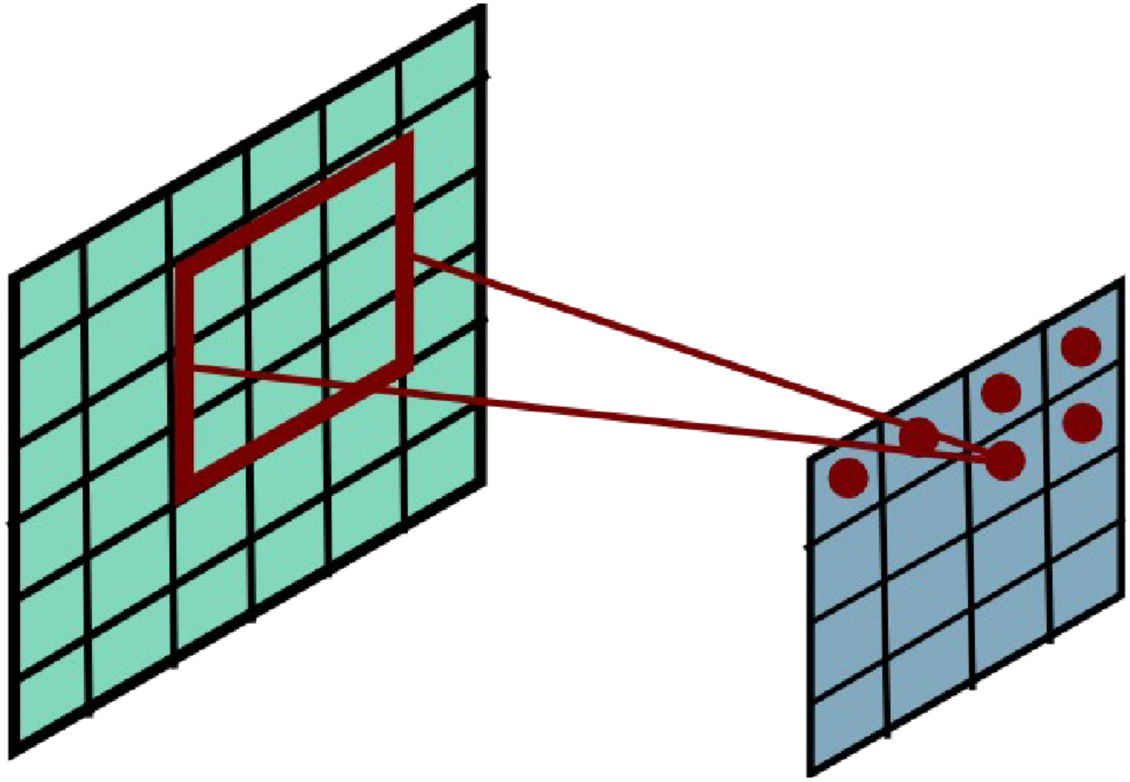


Ranzato

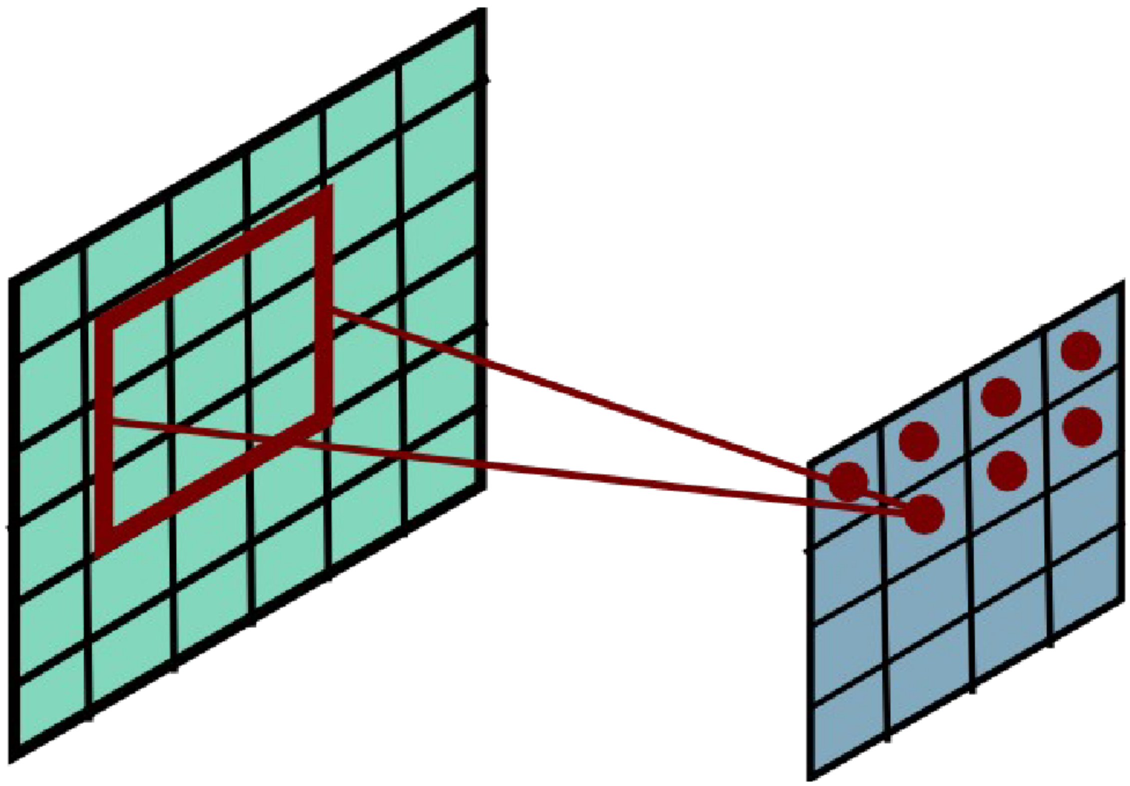
Convolutional Layer



Convolutional Layer

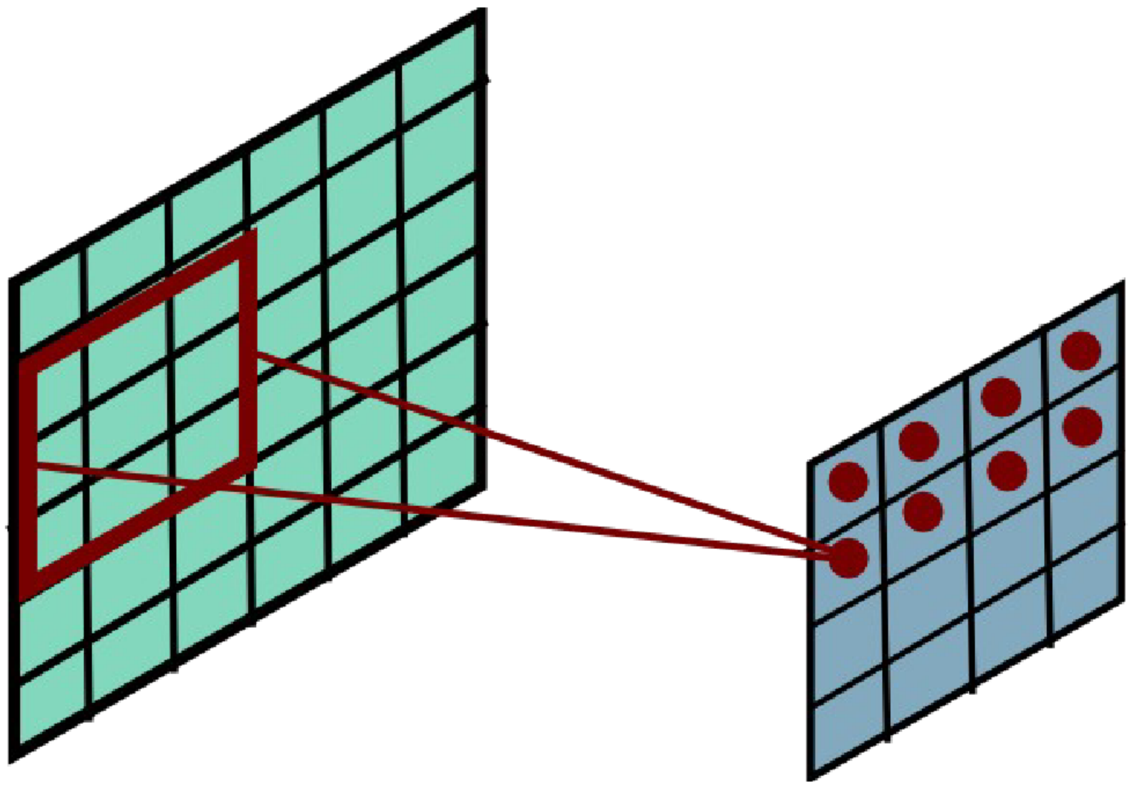


Convolutional Layer

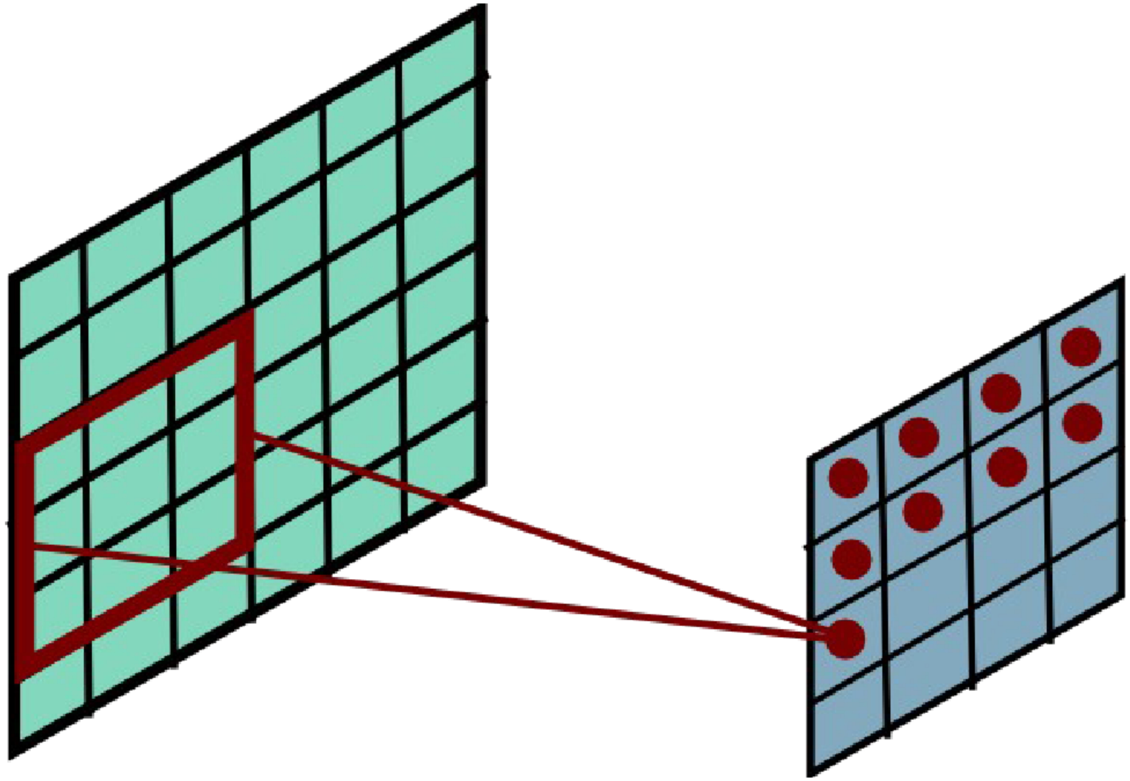


Ranzato

Convolutional Layer

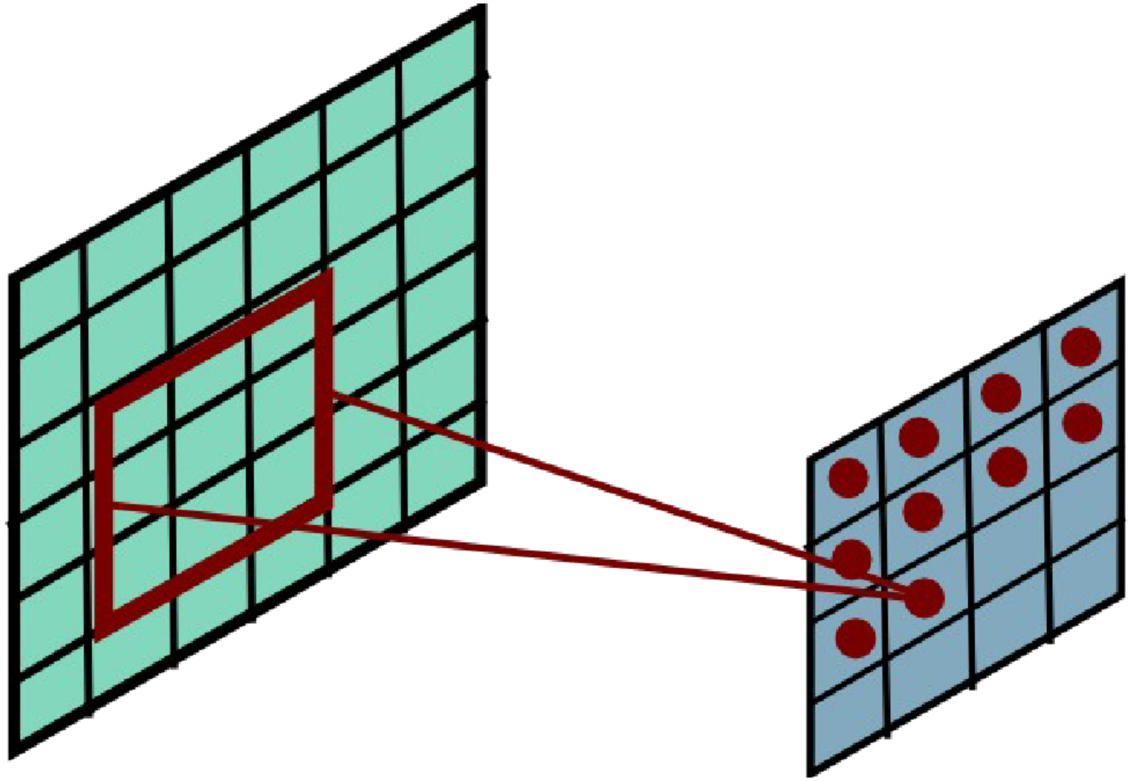


Convolutional Layer

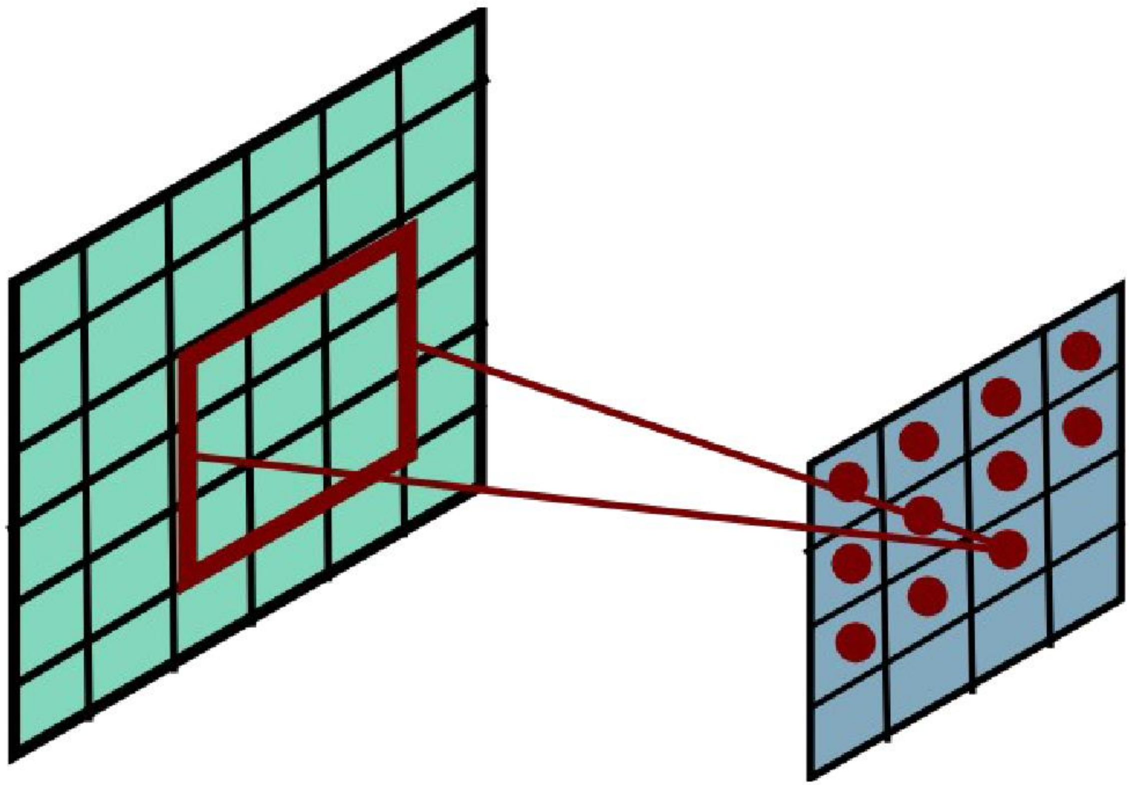


Ranzato

Convolutional Layer

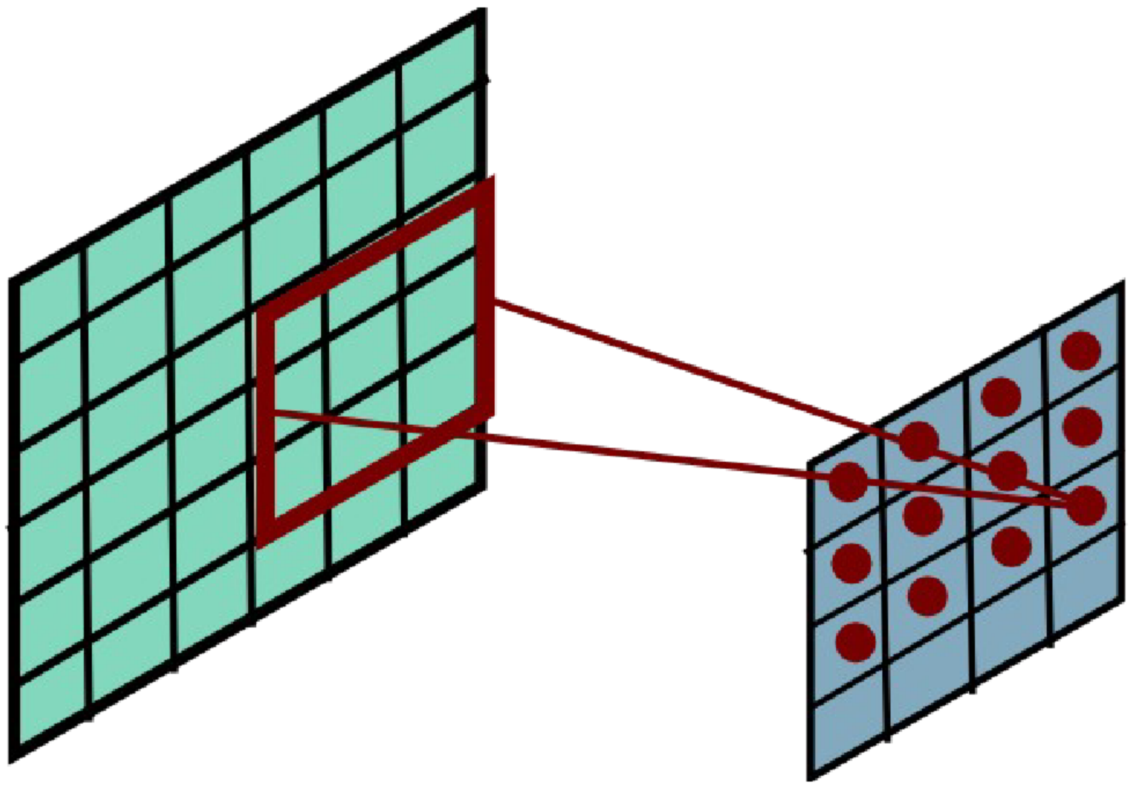


Convolutional Layer



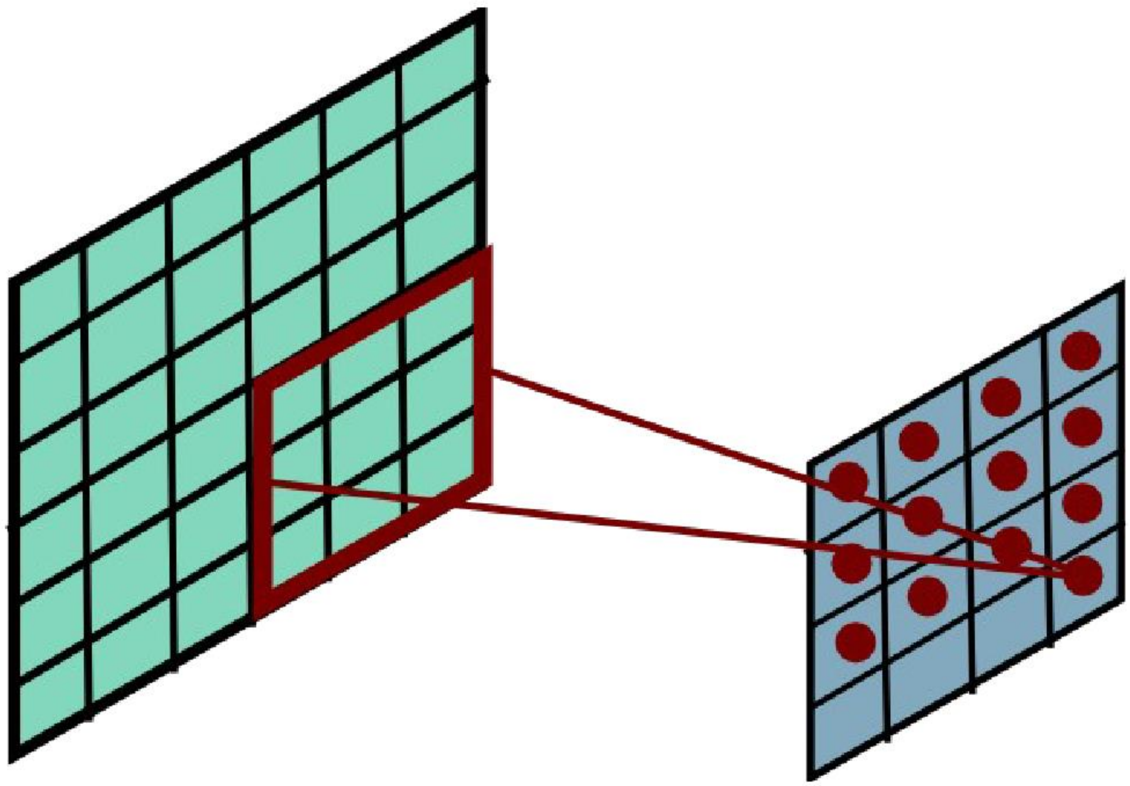
Ranzato

Convolutional Layer



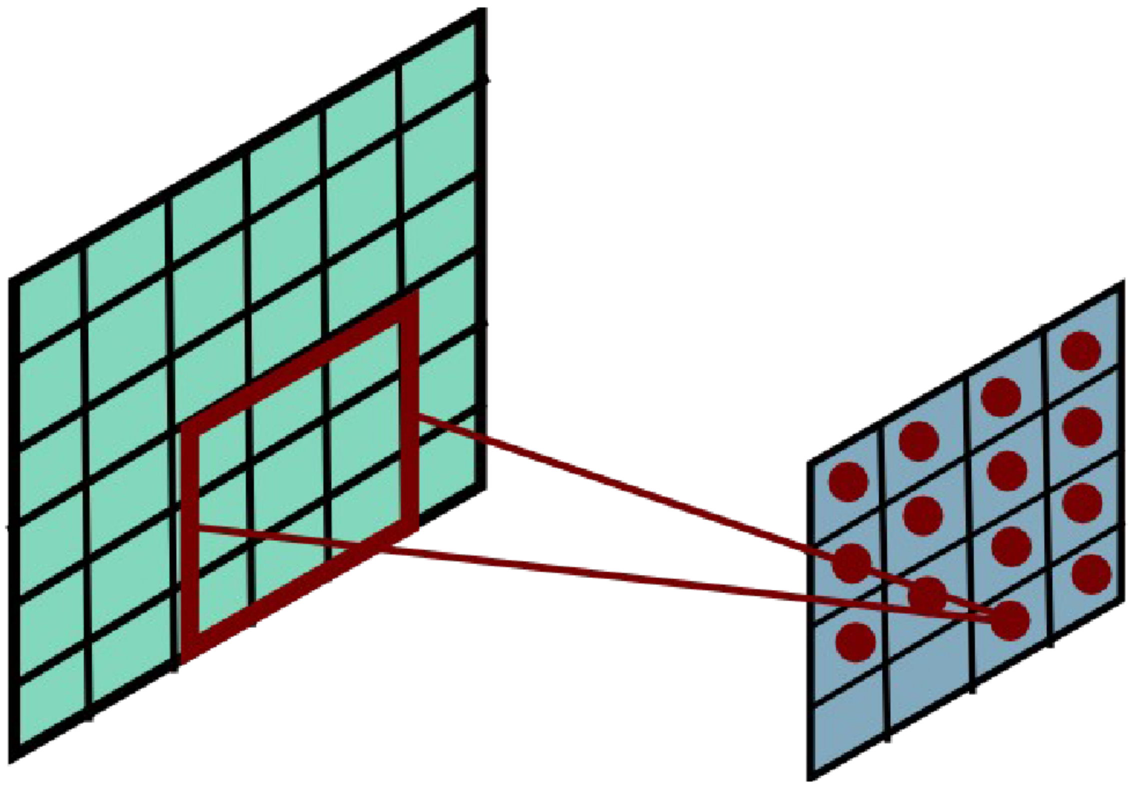
Ranzato

Convolutional Layer



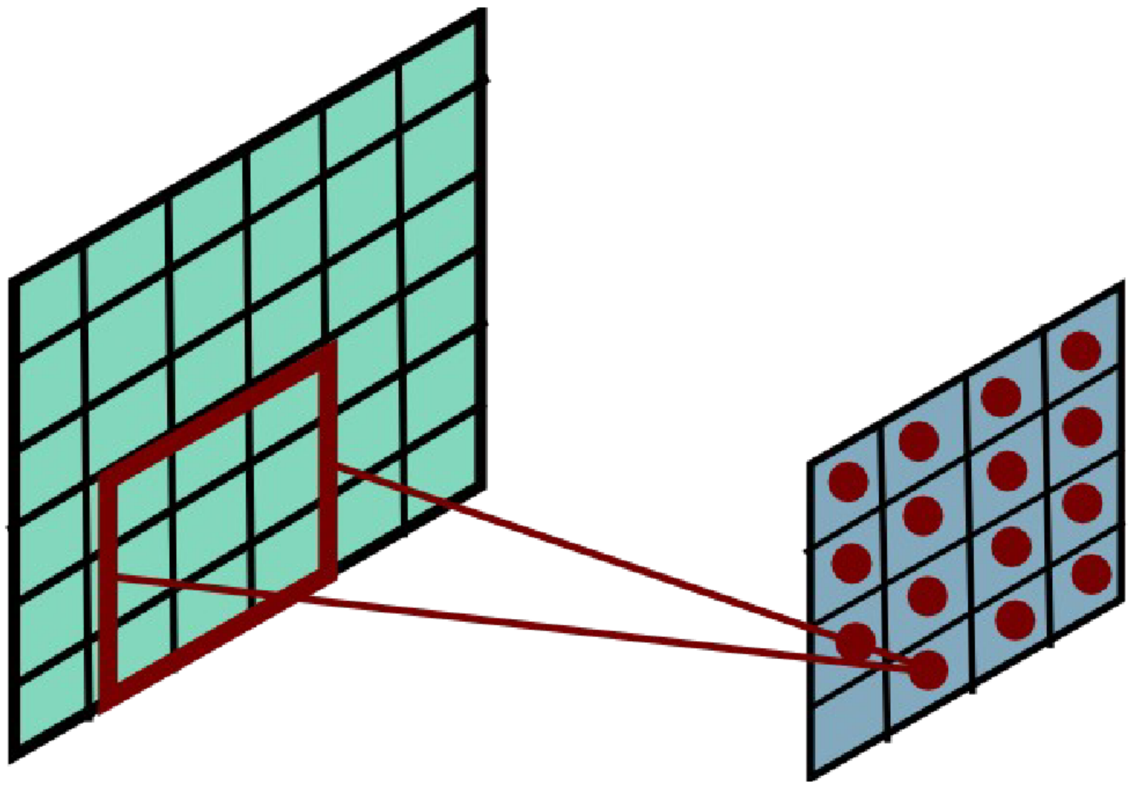
Ranzato

Convolutional Layer

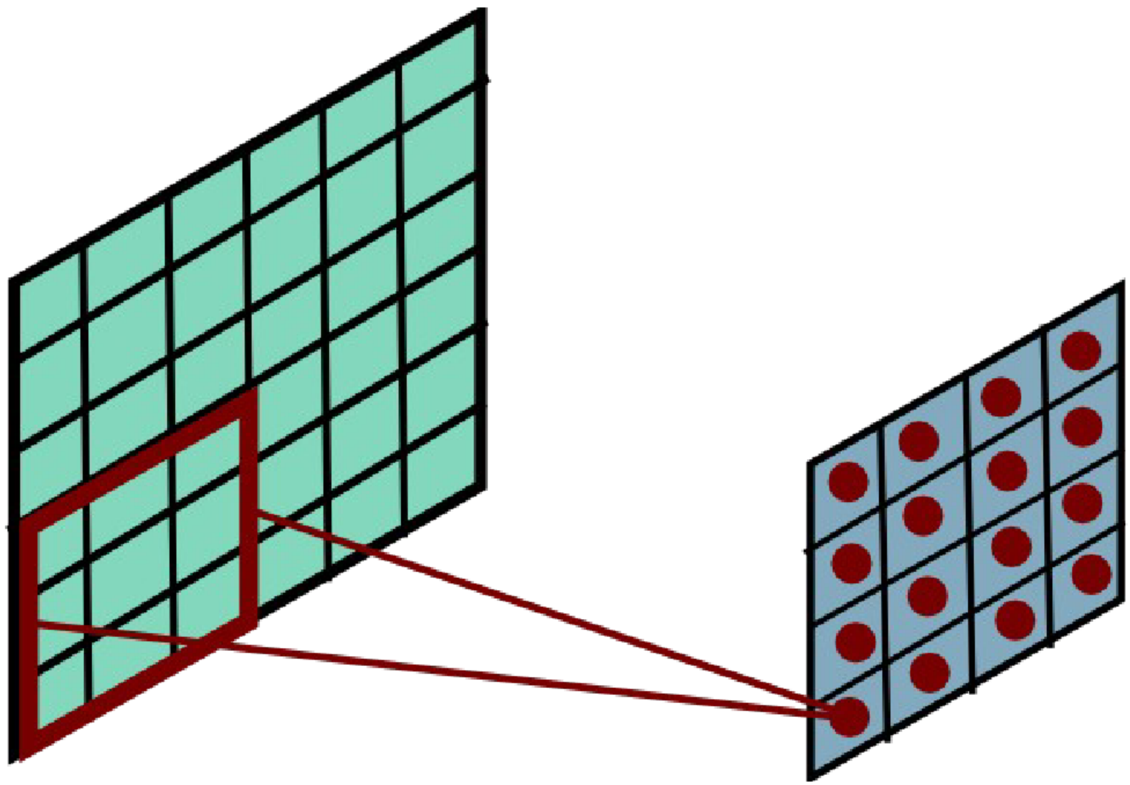


Ranzato

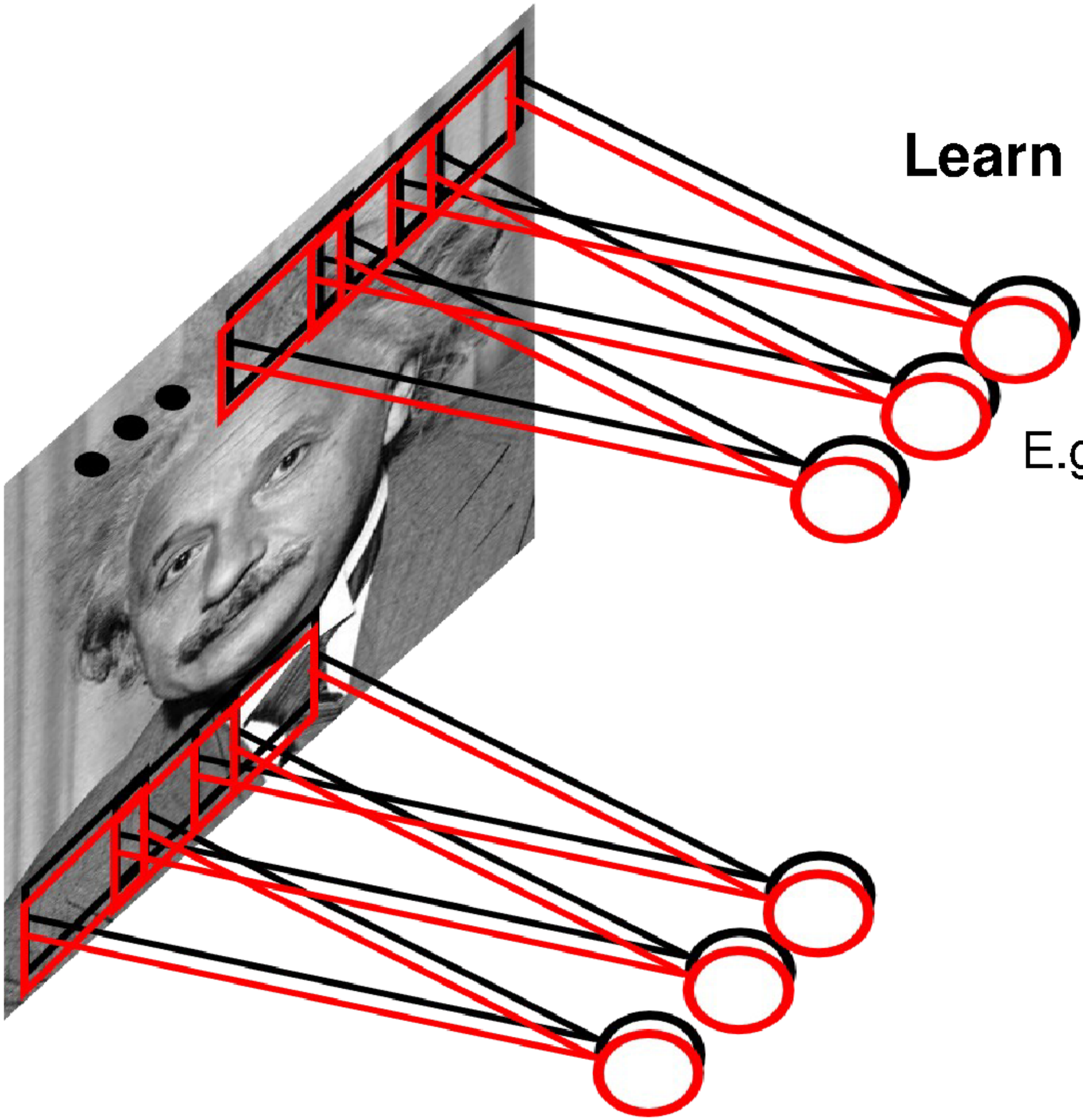
Convolutional Layer



Convolutional Layer



Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

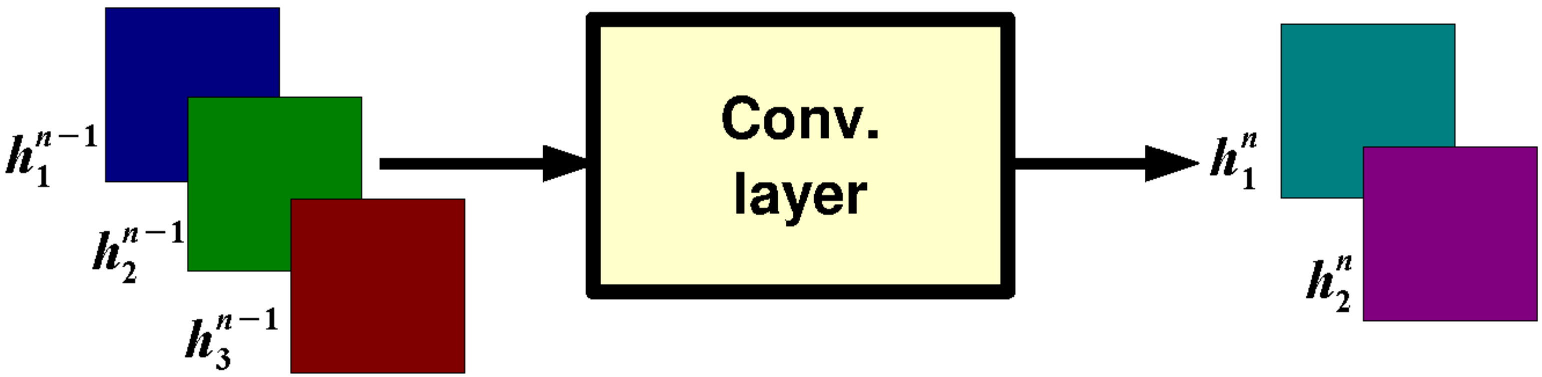
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



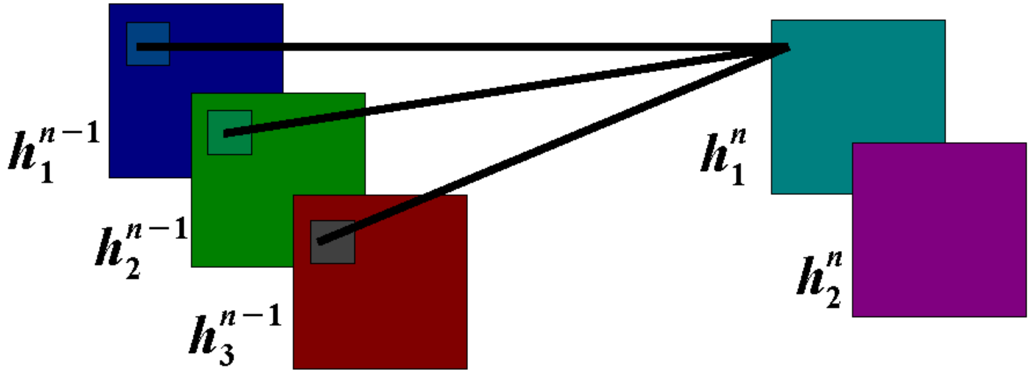
Convolutional Layer

$$h_j^n = max\,(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n)$$

output feature map

input feature map

kernel



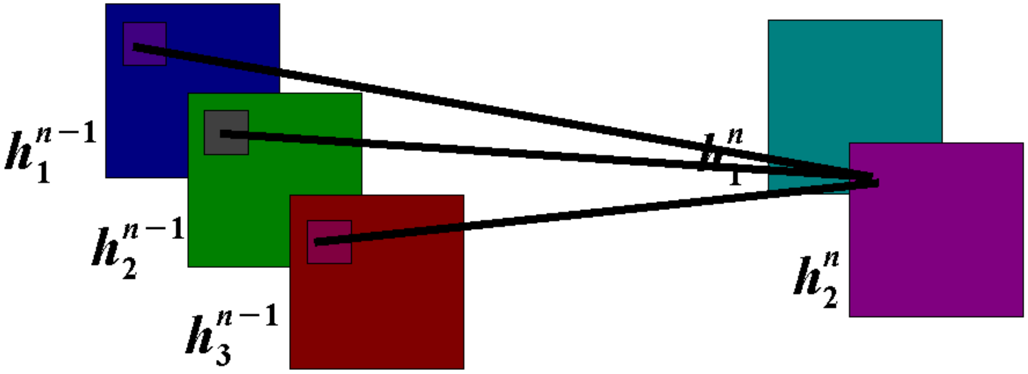
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

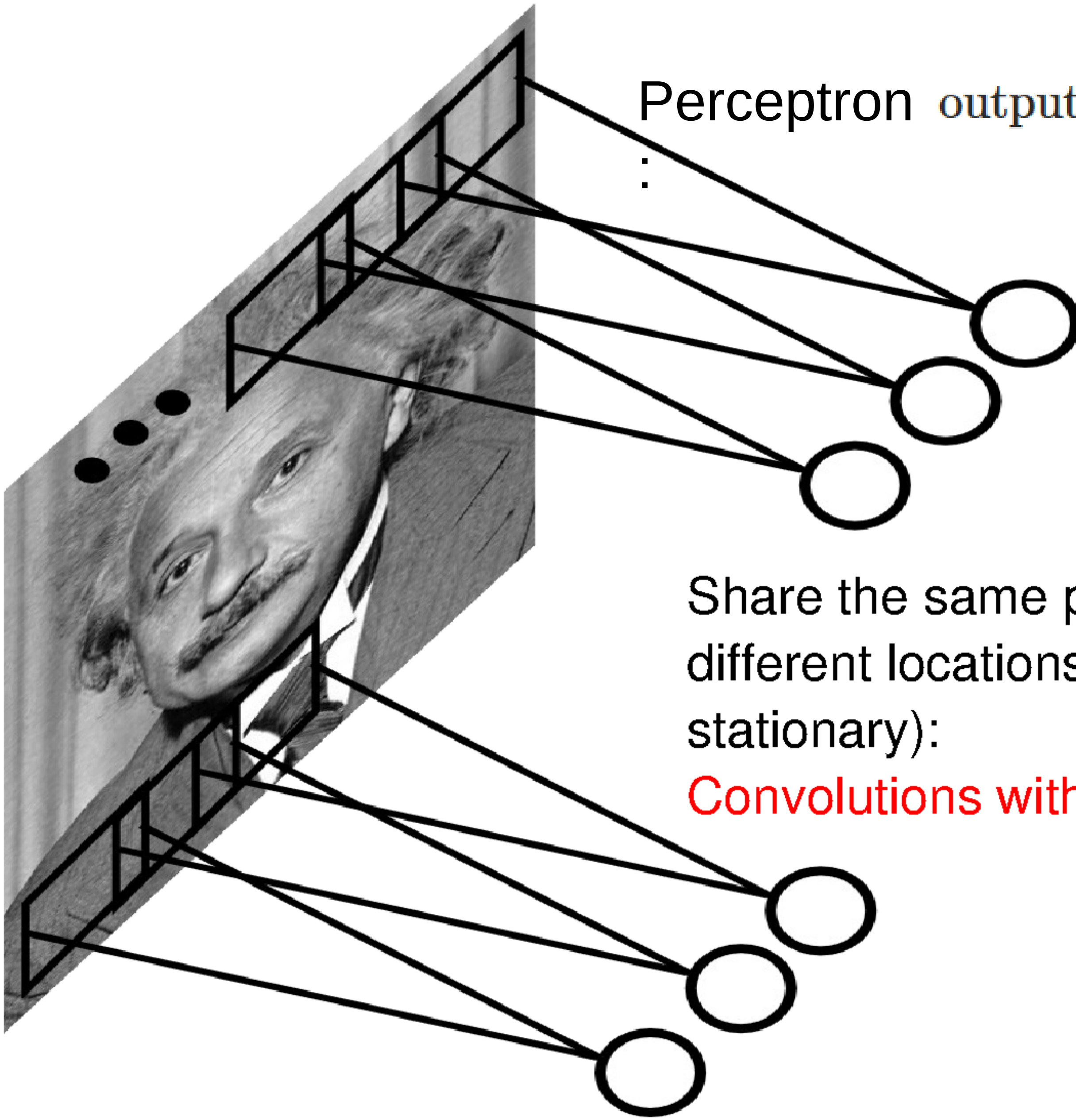
output feature map

input feature map

kernel



Convolutional Layer



Perceptron :

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \le 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

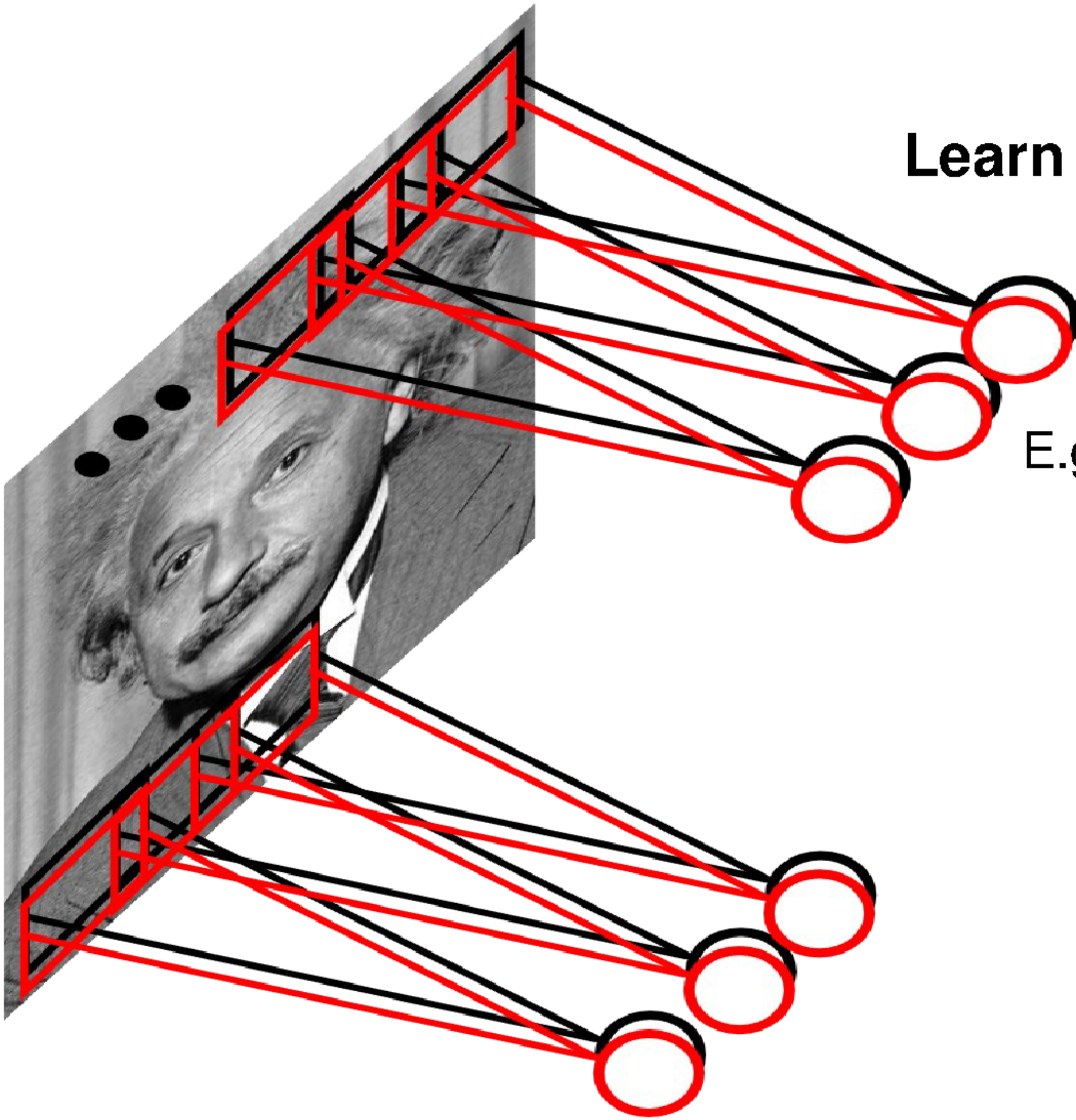
This is convolution!

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

Ranzato

Convolutional Layer



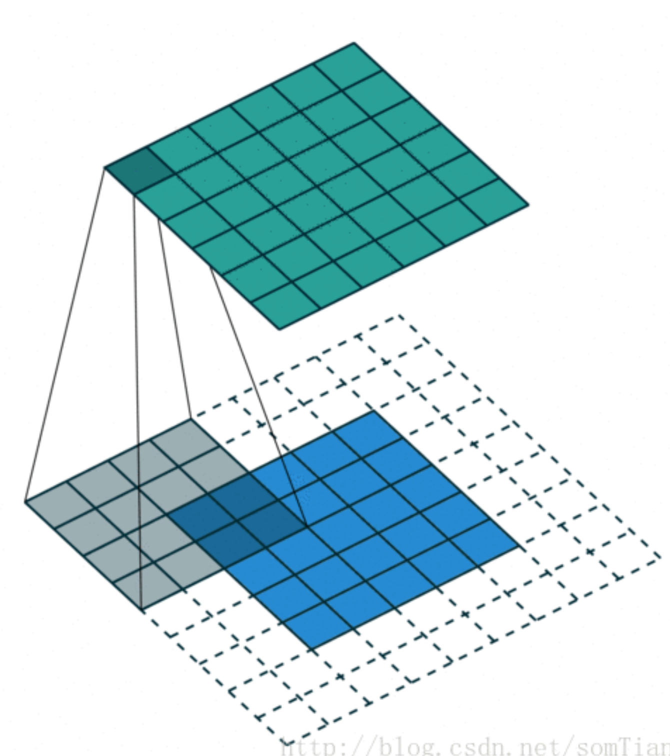
Learn multiple filters.

Filter = 'local' perceptron.
Also called *kernel.*

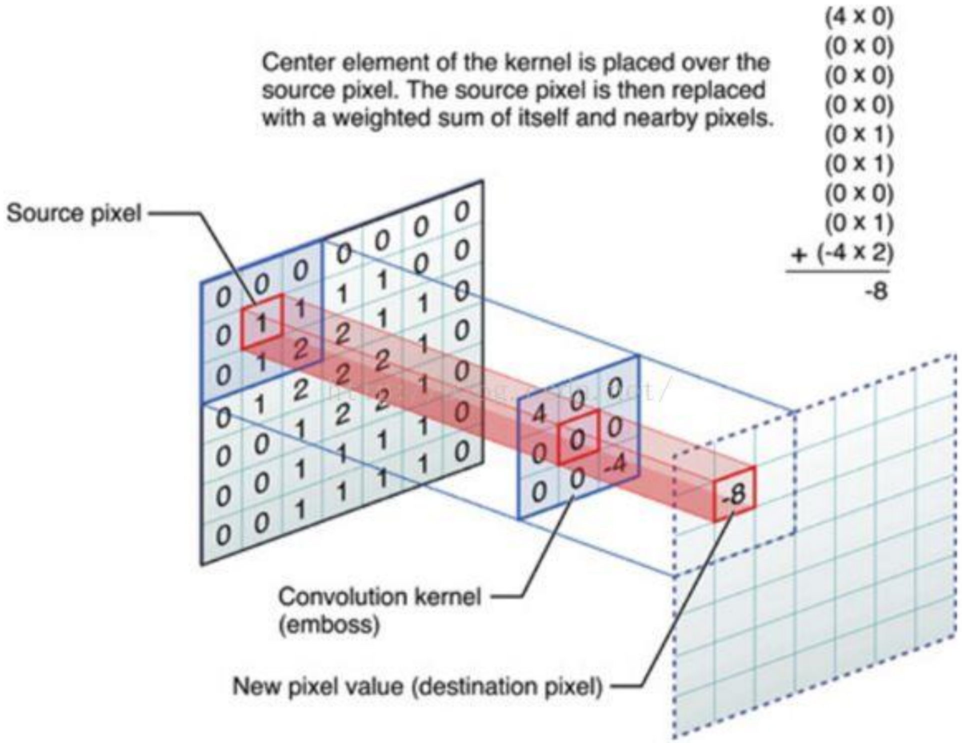
E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

Convolutional kernel



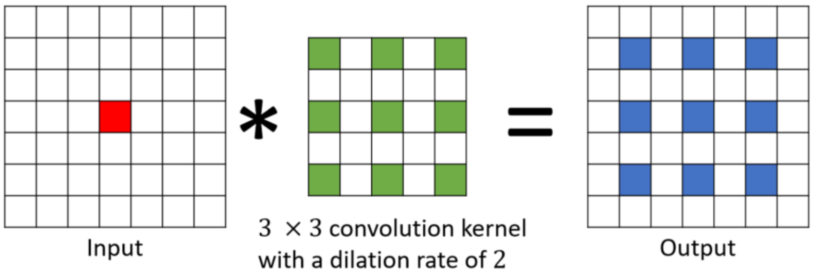
Convolutional kernel



Padding on the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input

Dilated Convolution Layer

- Inserts spaces between the kernel element to increase the effective size of kernel
- Same as the convolutional layer except it has additional parameter, dilation rate, that controls the spacing
- Each layer is defined using following parameters:
  - # Input channels (C_1)
  - # Output channels (C_2)
  - Kernel size ($w_1 \times h_1$)
  - Padding
  - Stride
  - **Dilation rate** (r)



Group Convolution Layer

- Input and kernel are split into g groups across channel dimension
- Each group then performs the convolutions independently
- Each layer is defined using following parameters:
  - # Input channels (C_1)
  - # Output channels (C_2)
  - Kernel size ($w_1 \times h_1$)
  - Padding
  - Stride
  - Dilation rate (r)
  - **# of groups** (g)
- Parameter reduction??

Group vs Standard Convolution Layer

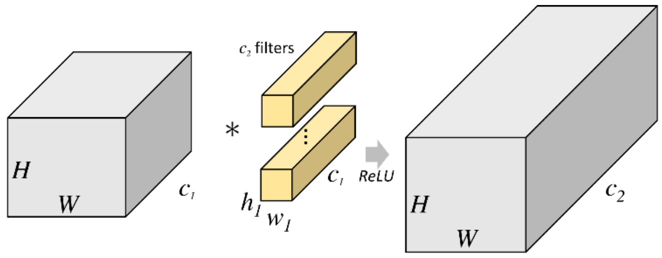


Figure: Standard convolution

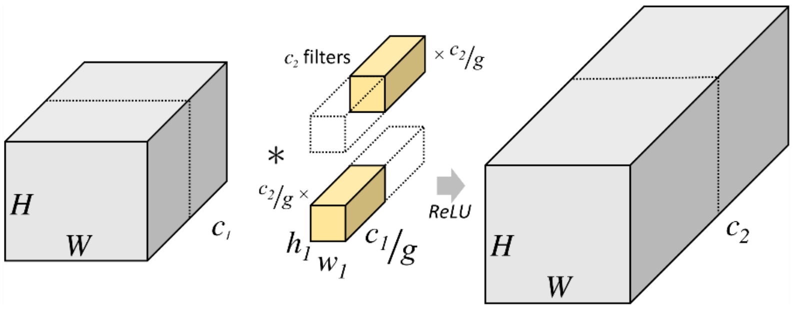


Figure: Grouped convolution

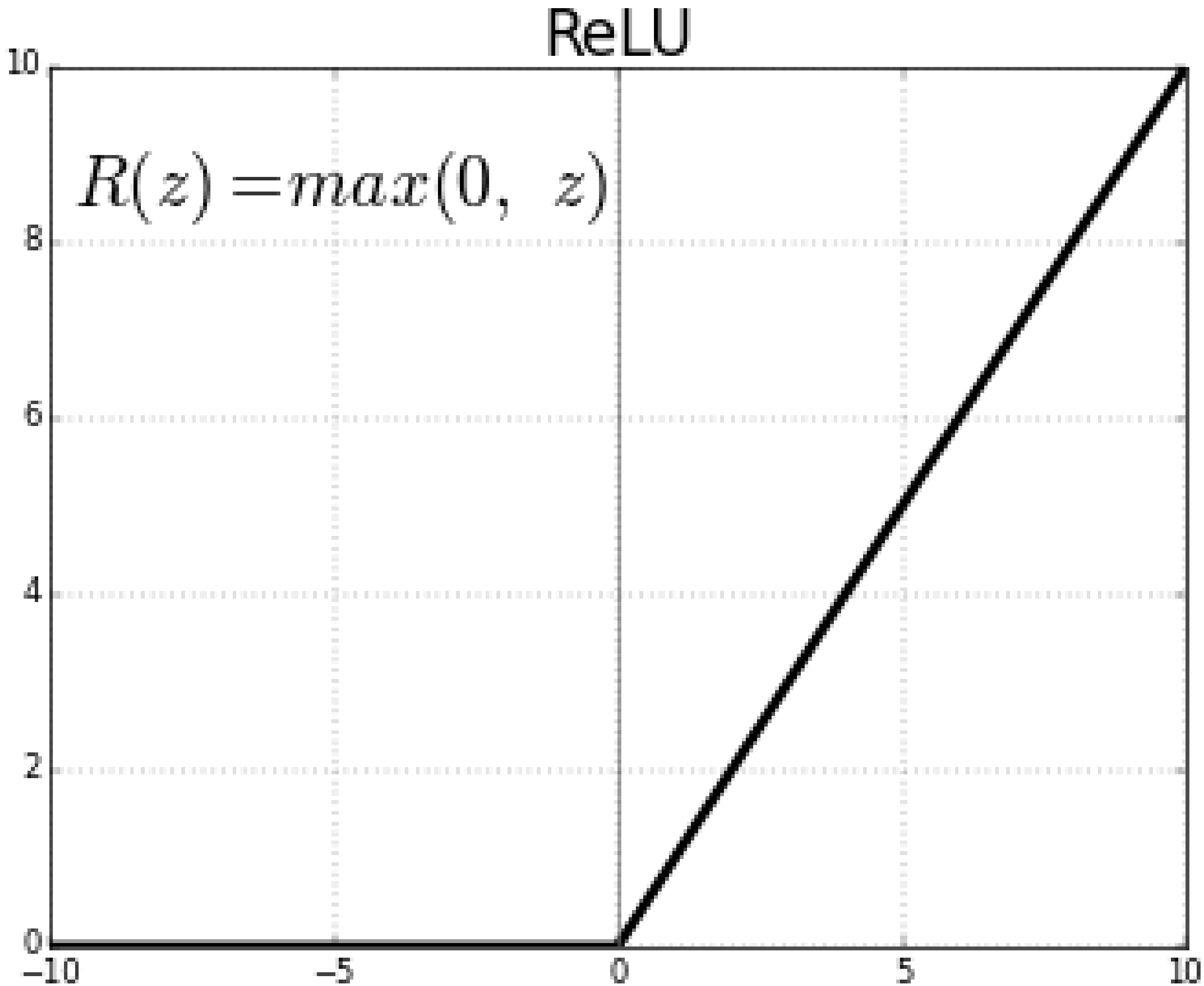
Depth-wise Convolution

- Special case of group convolution where each channel is processed independently

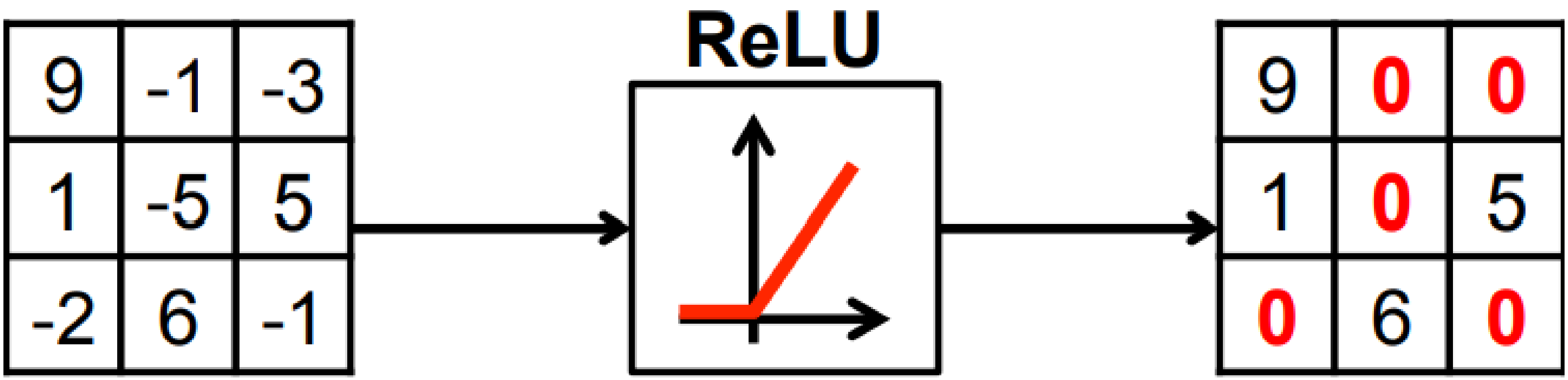
  \# input channels = # groups = # output channels

- Parameter reduction??

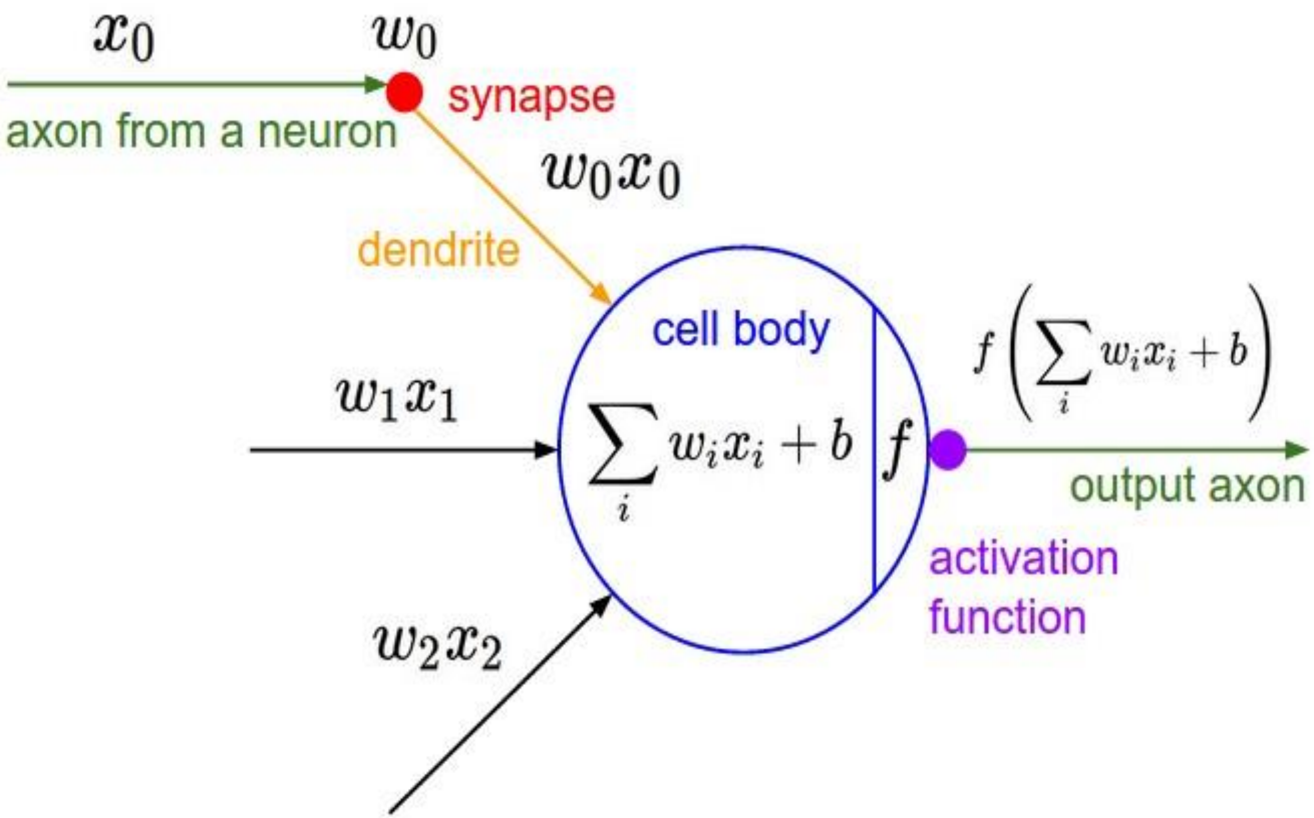
ReLU



ReLU



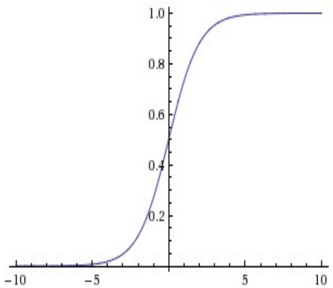
ReLU



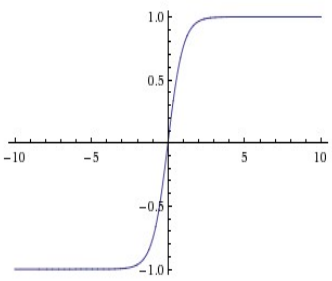
Feature Normalization

Sigmoid

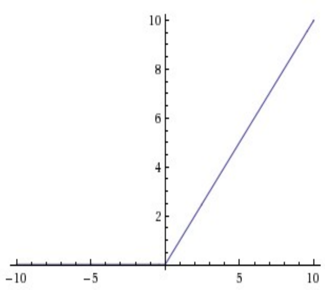
$$\sigma(x) = 1/(1+e^{-x})$$



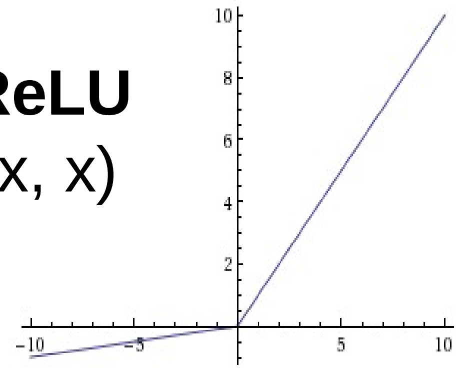
tanh tanh(x)



ReLU max(0,x)

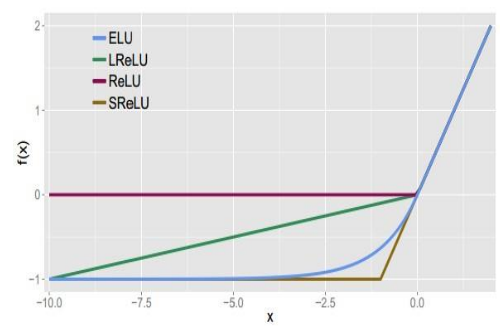


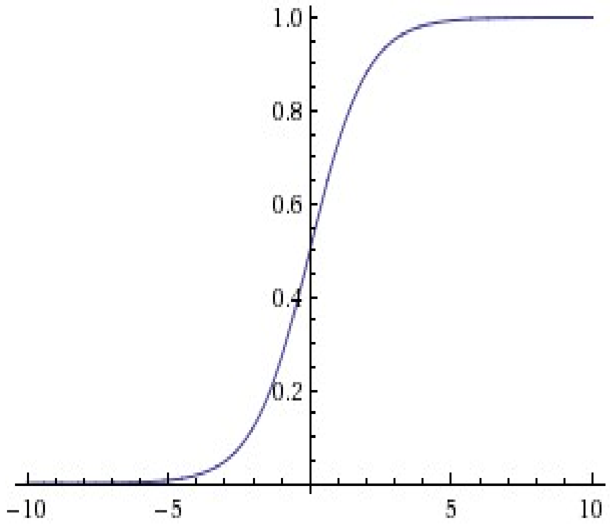
Leaky ReLU
max(0.1x, x)



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

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

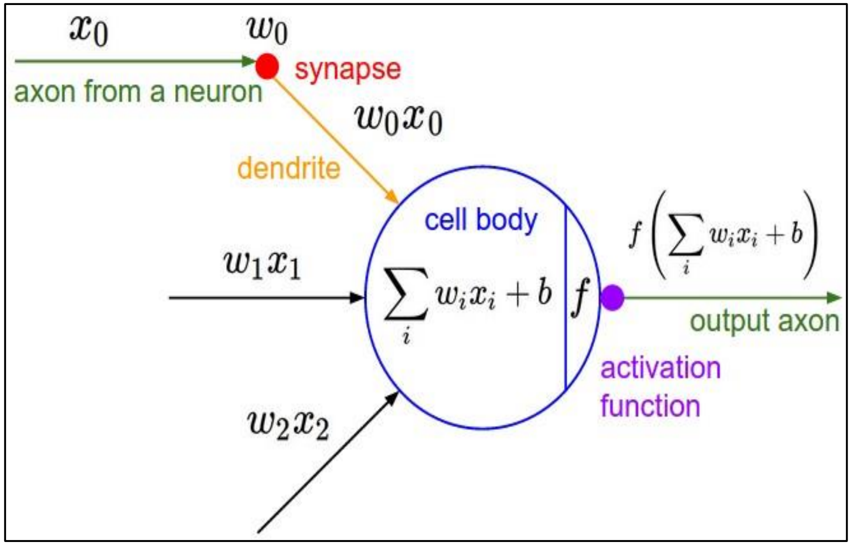




Sigmoid

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

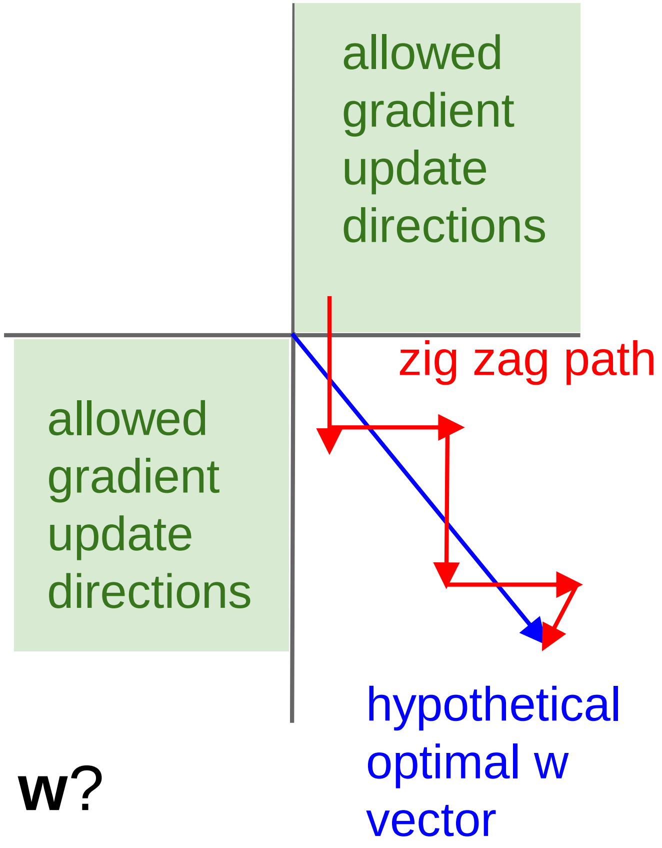
- Squashes numbers to range [0,1] – can kill gradients.
- A key element in LSTM networks – “control signals”
- Best for learning “logical” functions – i.e. functions on binary inputs.
- Not as good for image networks (replaced by RELU)
- Not zero-centered



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

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

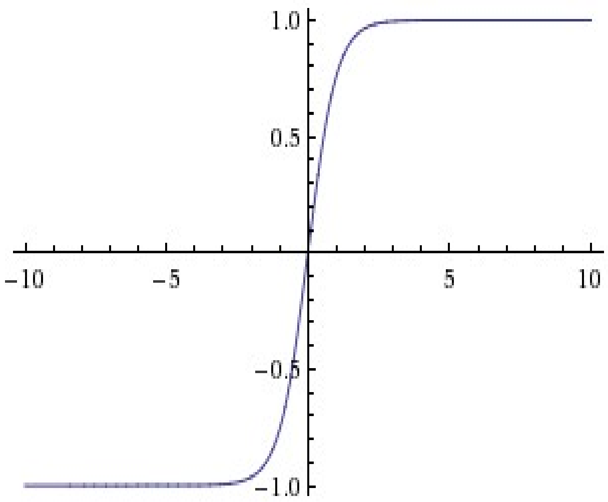
$$f\left(\sum_i w_i x_i + b\right)$$



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

Always all positive or all negative :(

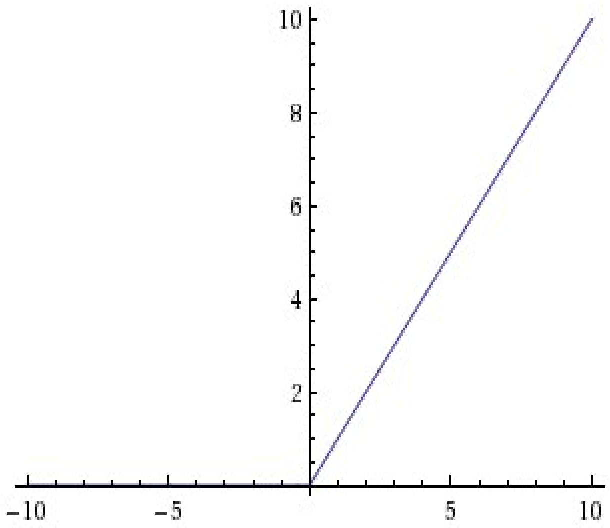
(this is also why you want zero-mean data!)



tanh(x)

- Squashes numbers to range [-1,1]
- Zero centered (nice)
- Still kills gradients when saturated :(
- Also used in LSTMs for bounded, signed values.
- Not as good for binary functions

[LeCun et al., 1991]

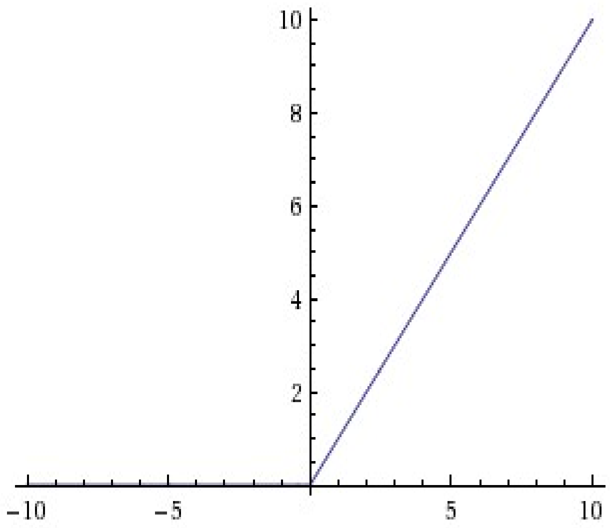


ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**
- Does not saturate (in +region)
- Converges faster than sigmoid/tanh on image data (e.g. 6x)
- Not suitable for logical functions
- Not for control in recurrent nets

[Krizhevsky et al., 2012]

Activation Functions



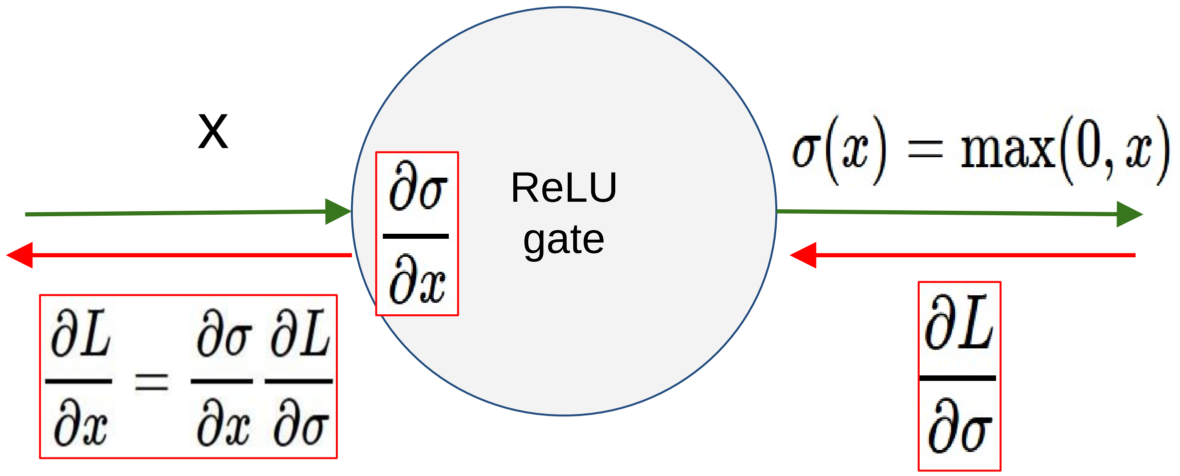
ReLU
(Rectified Linear Unit)

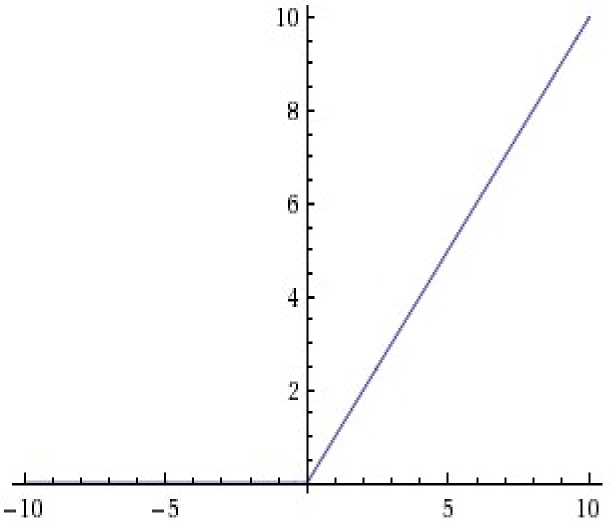
- Computes **f(x) = max(0,x)**

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

- Not zero-centered output
- An annoyance:

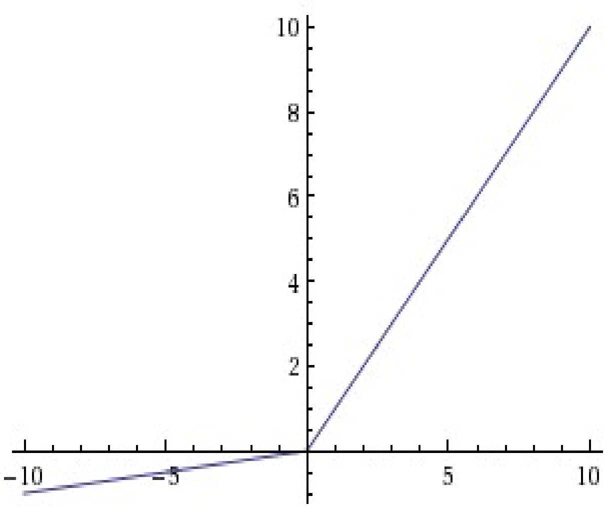
hint: what is the gradient when $x < 0$?





What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

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

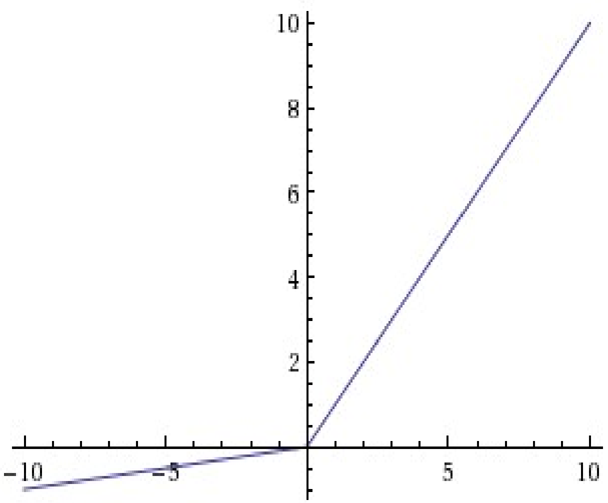


- Does not saturate
- Converges faster than sigmoid/tanh on image data(e.g. 6x)
- **will not “die”.**

Leaky ReLU

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

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



- Does not saturate
- Converges faster than sigmoid/tanh on image data (e.g. 6x)
- **will not “die”.**

Leaky ReLU

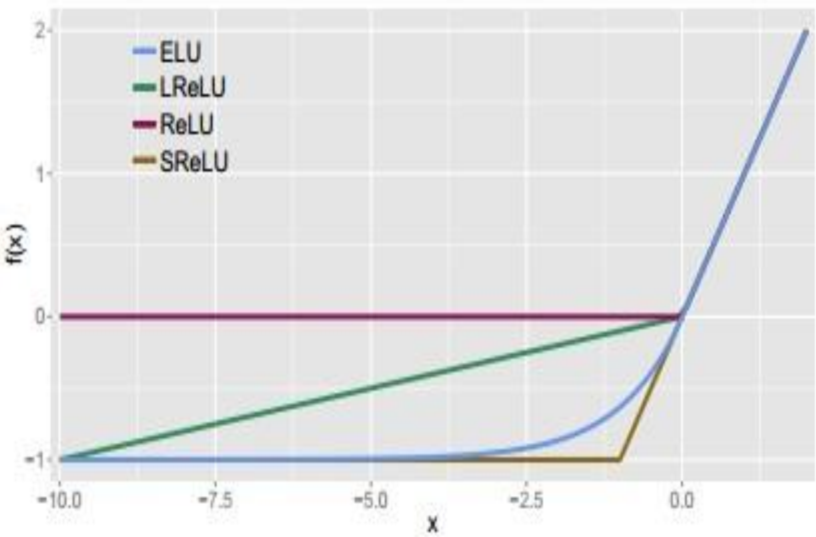
$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

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

backprop into \alpha
(parameter)

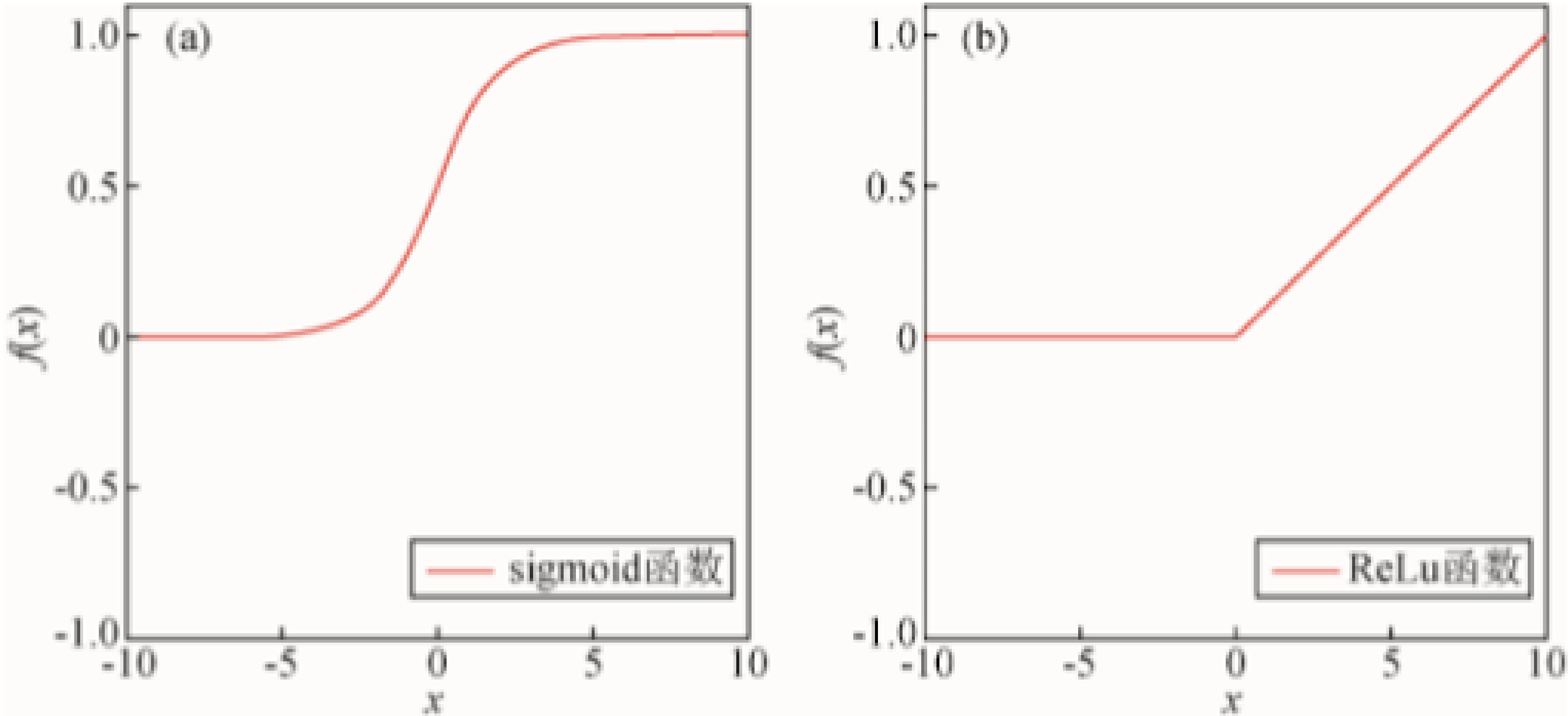
Exponential Linear Units (ELU)



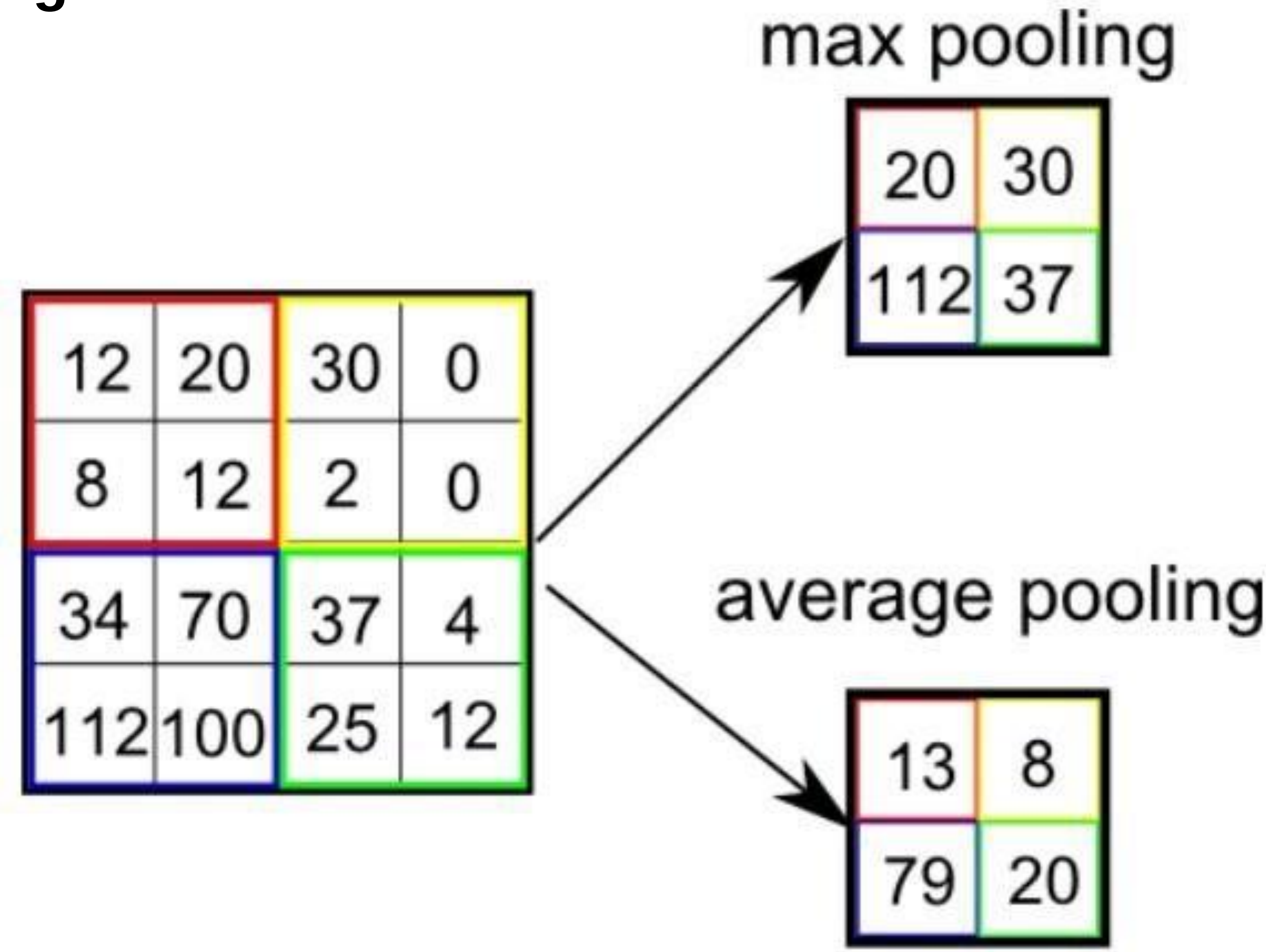
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \le 0 \end{cases}$$

- All benefits of ReLU
- Does not die
- Closer to zero mean outputs

ReLU

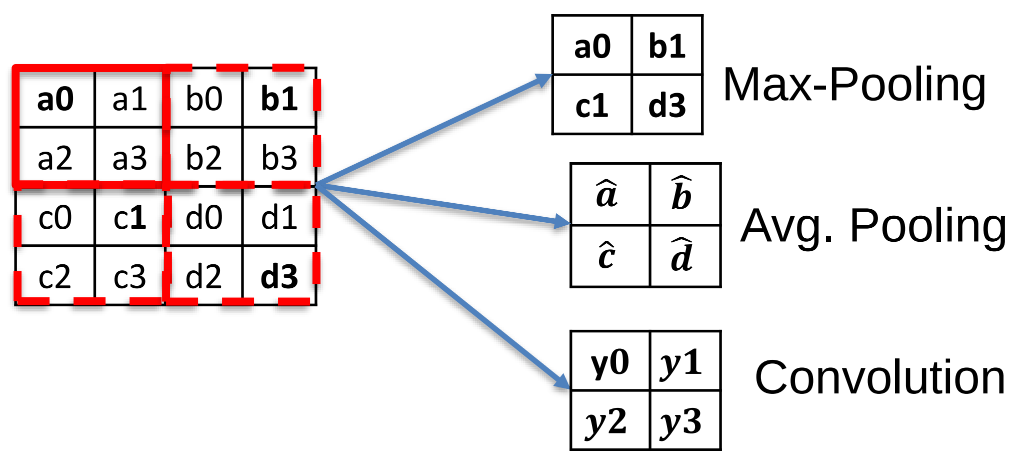

(a)
1.0
0.5
0
-0.5
-1.0
f(x)
-10
-5
0
5
10
x
sigmoid函数
(b)
1.0
0.5
0
-0.5
-1.0
f(x)
-10
-5
0
5
10
x
ReLu函数

Pooling for Dimension Reduction

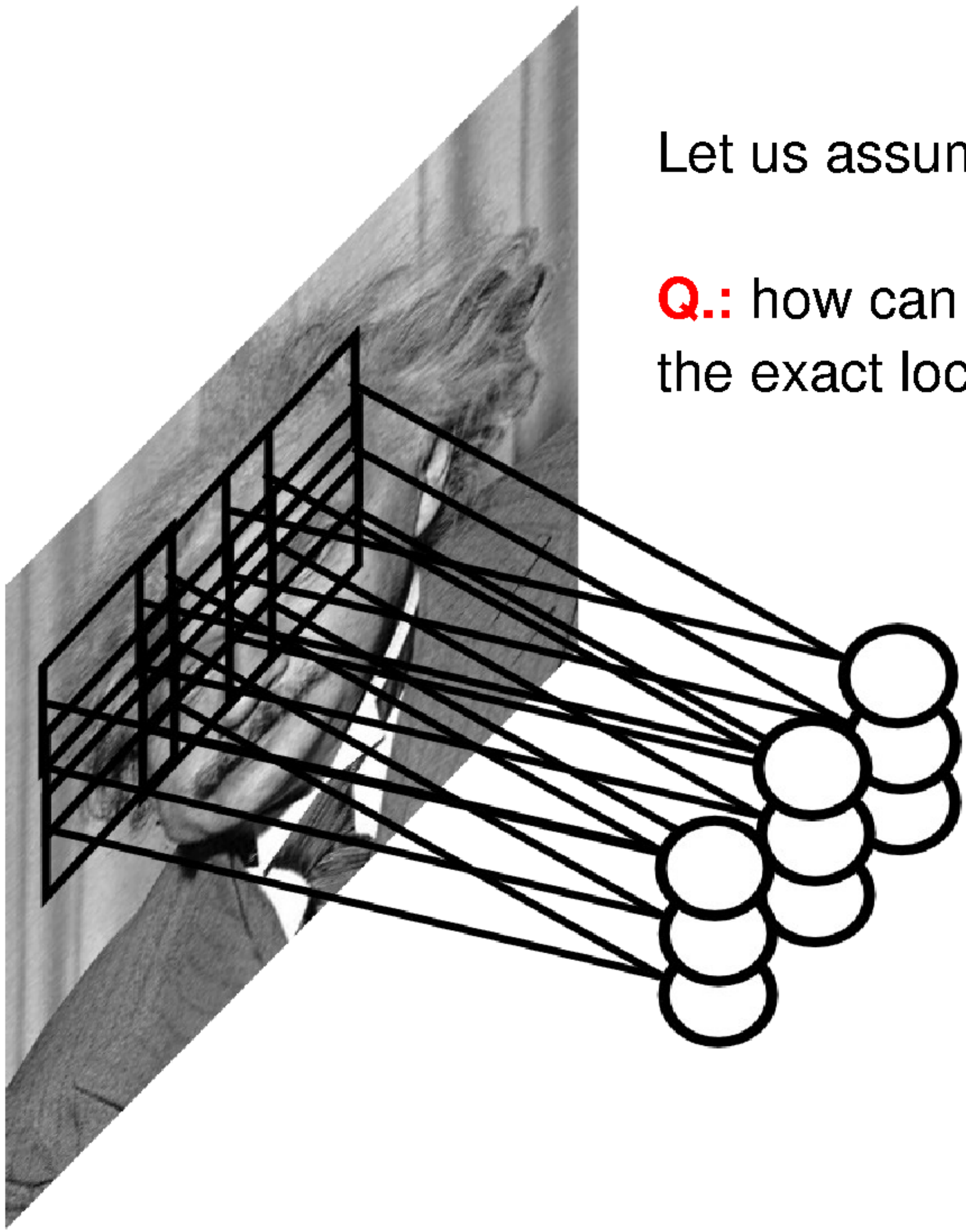


Down-sampling

- Learning representations at multiple scales is a fundamental step in computer vision
  - Laplacian Pyramids
  - SIFT, etc.
- Down-sampling in CNNs
  - Strided convolution
  - Max pooling
  - Avg. Pooling



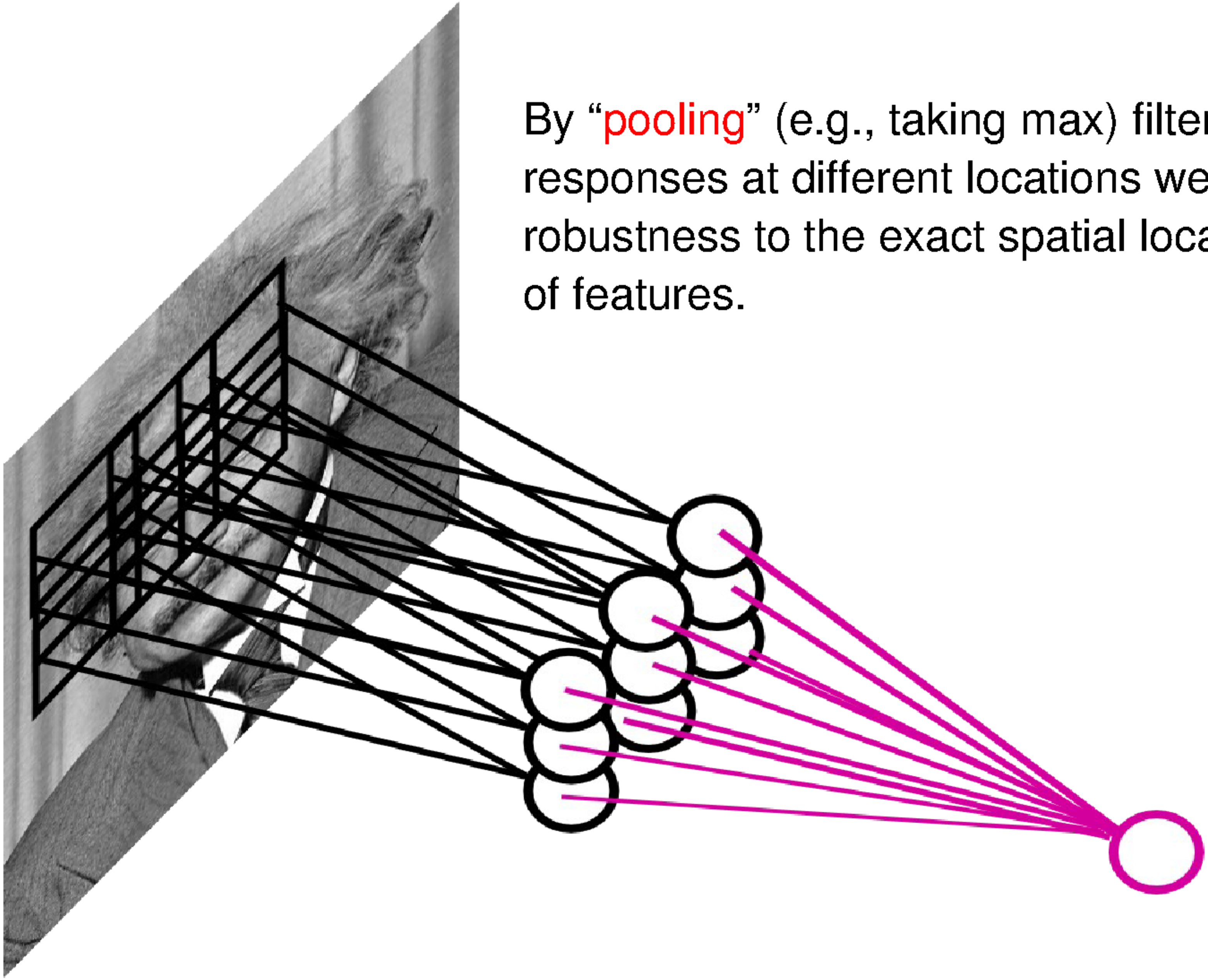
Pooling Layer



Let us assume filter is an "eye" detector.

Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})$$

L2-pooling:

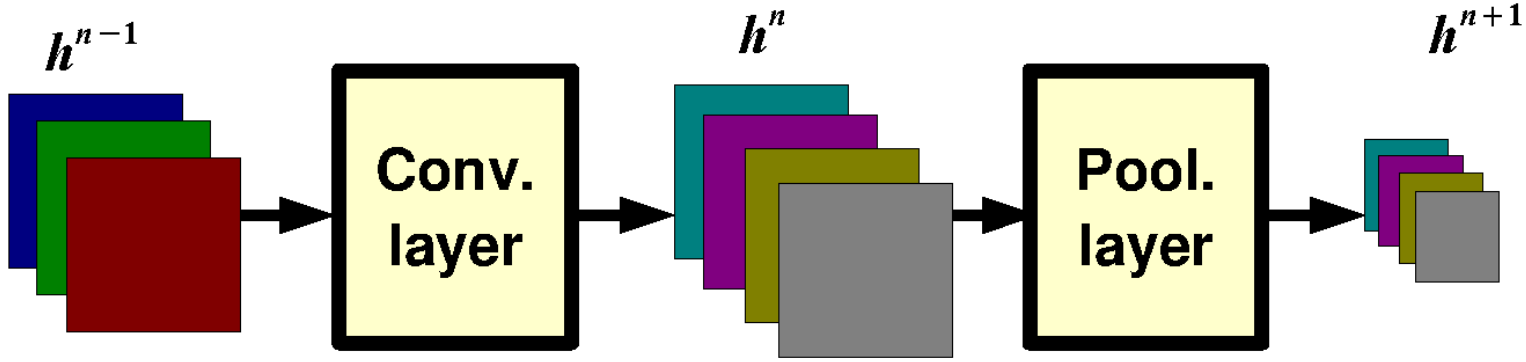
$$h_j^n(x,y) = \sqrt{\sum_{\bar{x} \in N(x),\, \bar{y} \in N(y)} h_j^{n-1}(\bar{x},\bar{y})^2}$$

L2-pooling over features:

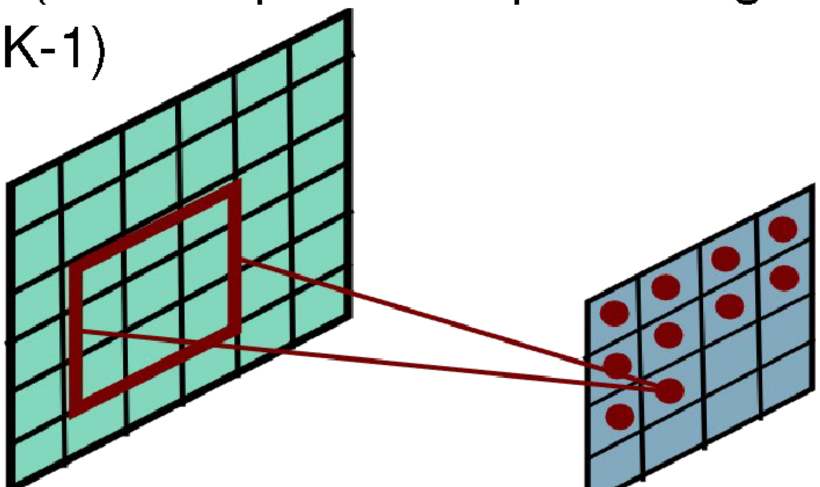
$$h_j^n(x,y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x,y)^2}$$

Ranzato

Pooling Layer: Receptive Field Size

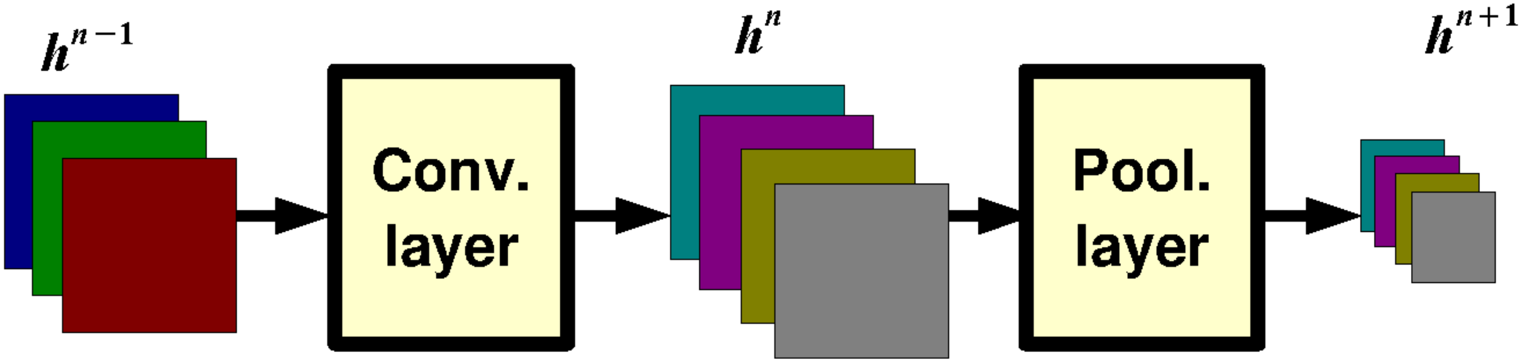


If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)

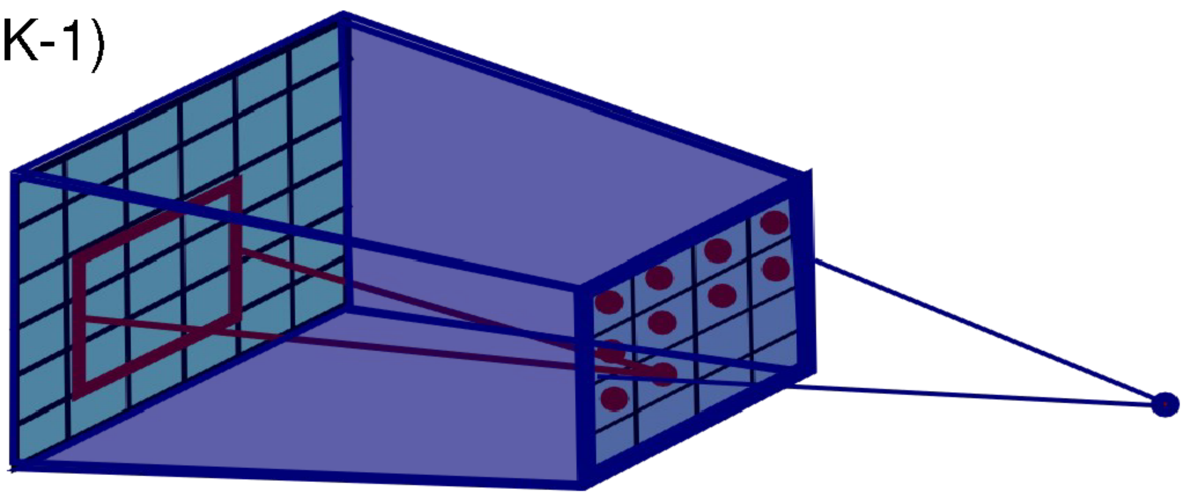


Ranzato

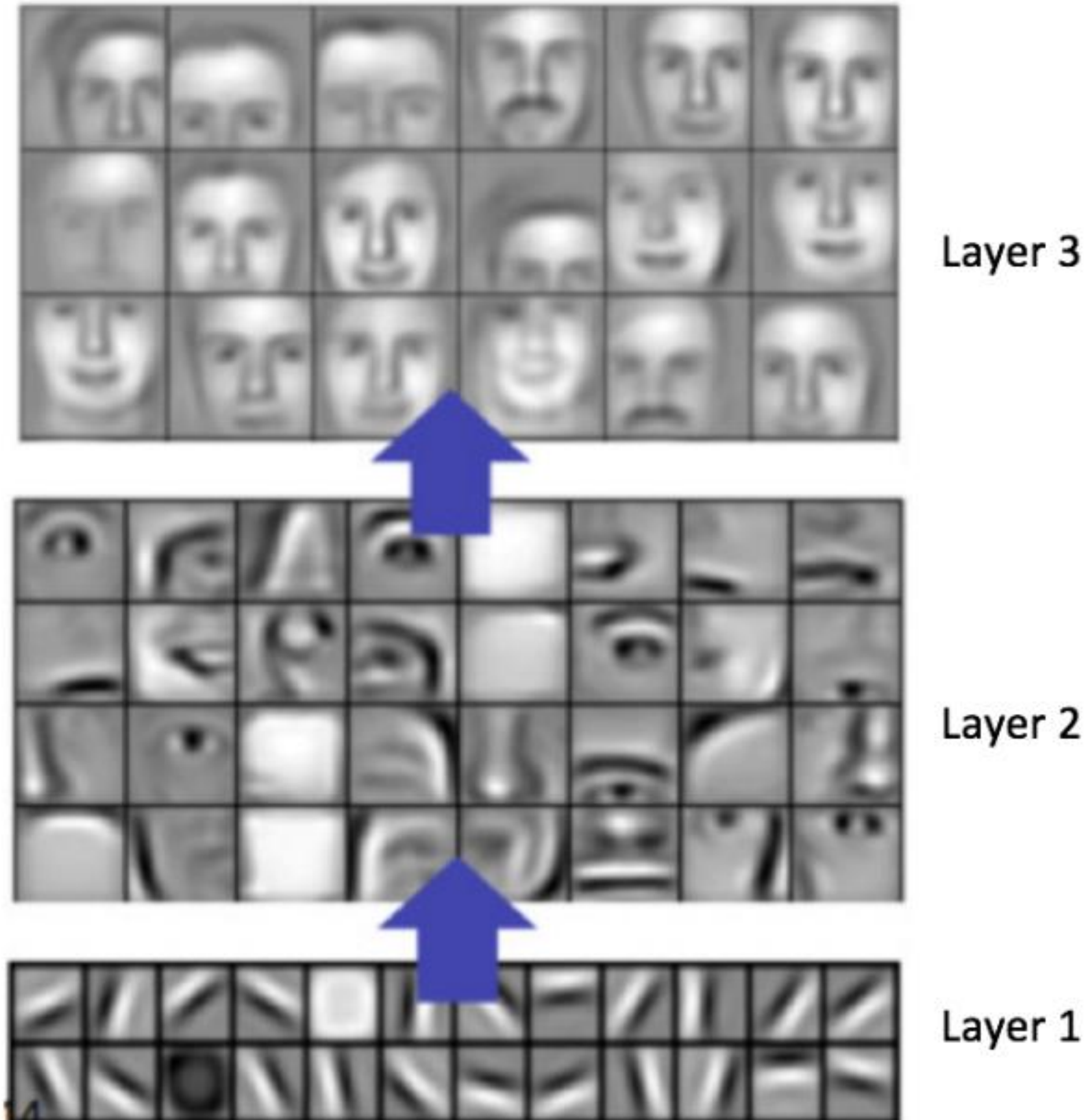
Pooling Layer: Receptive Field Size



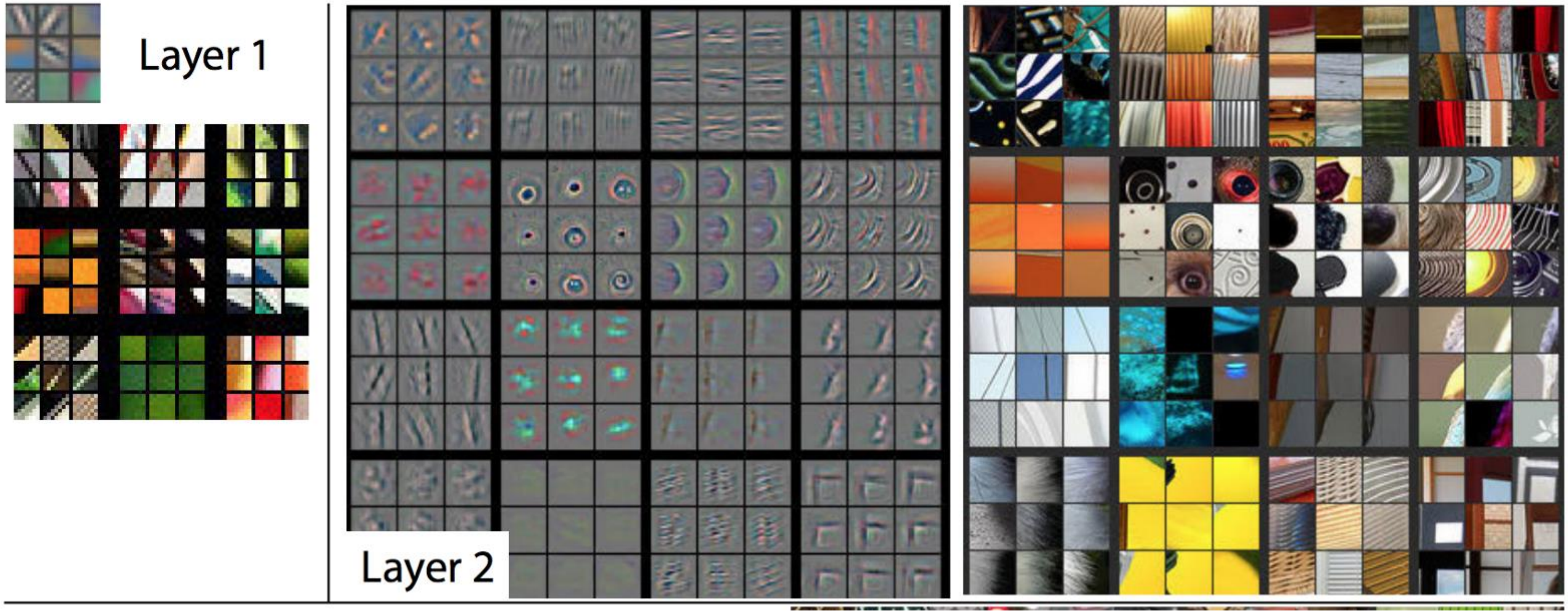
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)

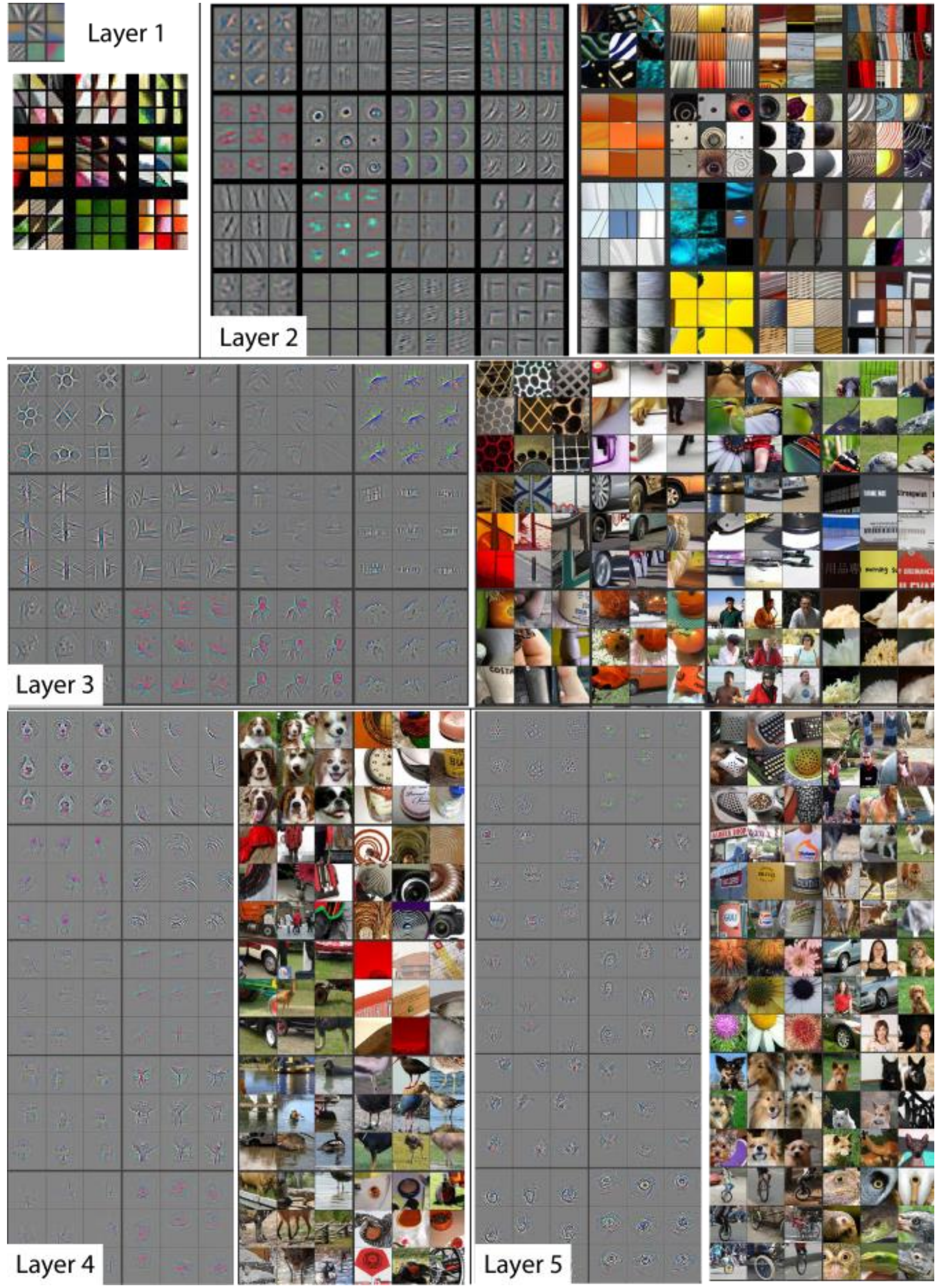


Features at Different Levels

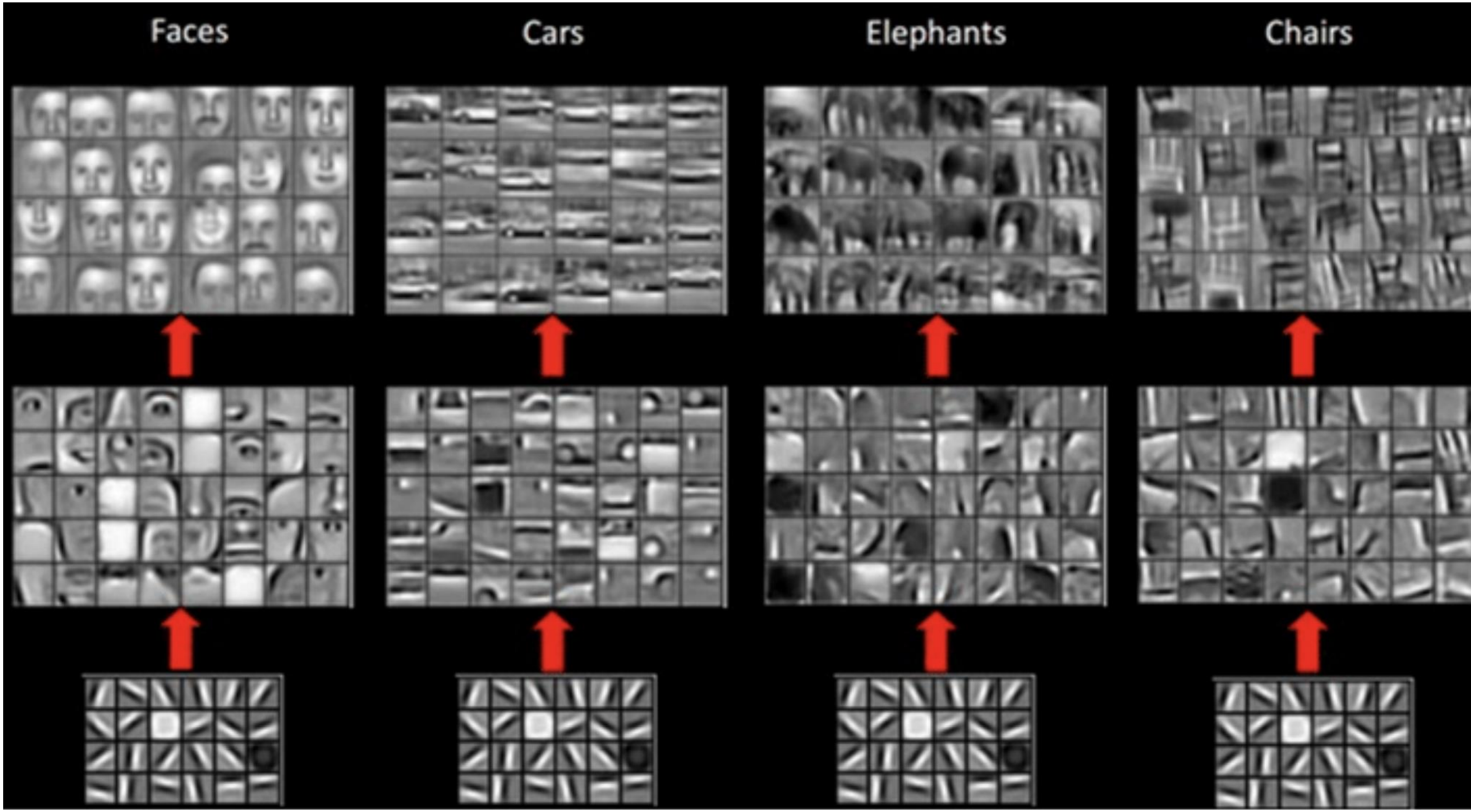


Features at Different Levels

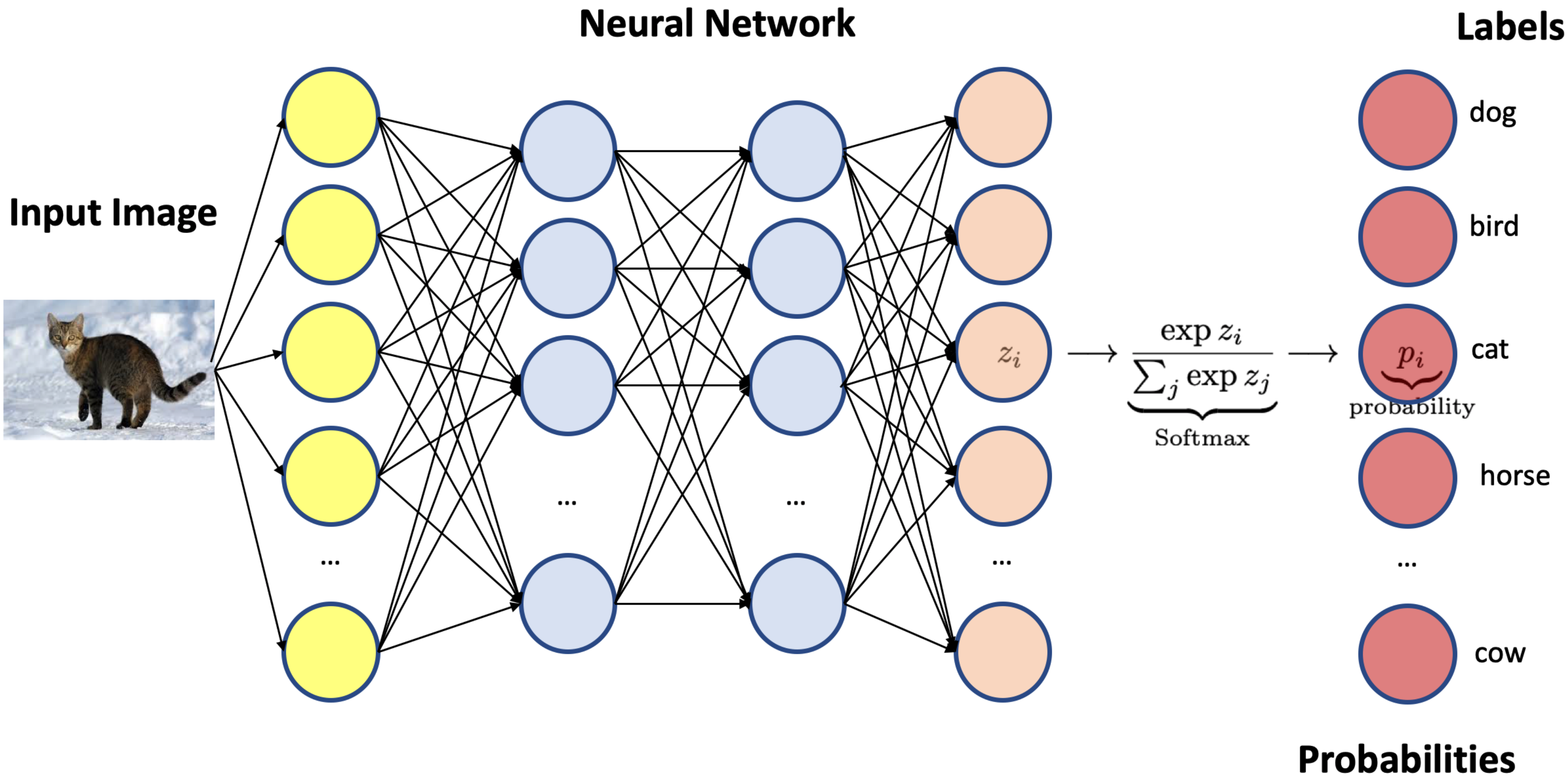



Layer 1
Layer 2
Layer 3
Layer 4
Layer 5

Features at Different Levels



Softmax



Classifier for prediction

Operations for Network Training

1. **Data Augmentation**: extracting transformation-invariant features

2. **Fine-tune**: optimizing feature extractor & classifier

3. **Batch Normalization:** feature normalization & shift reduction

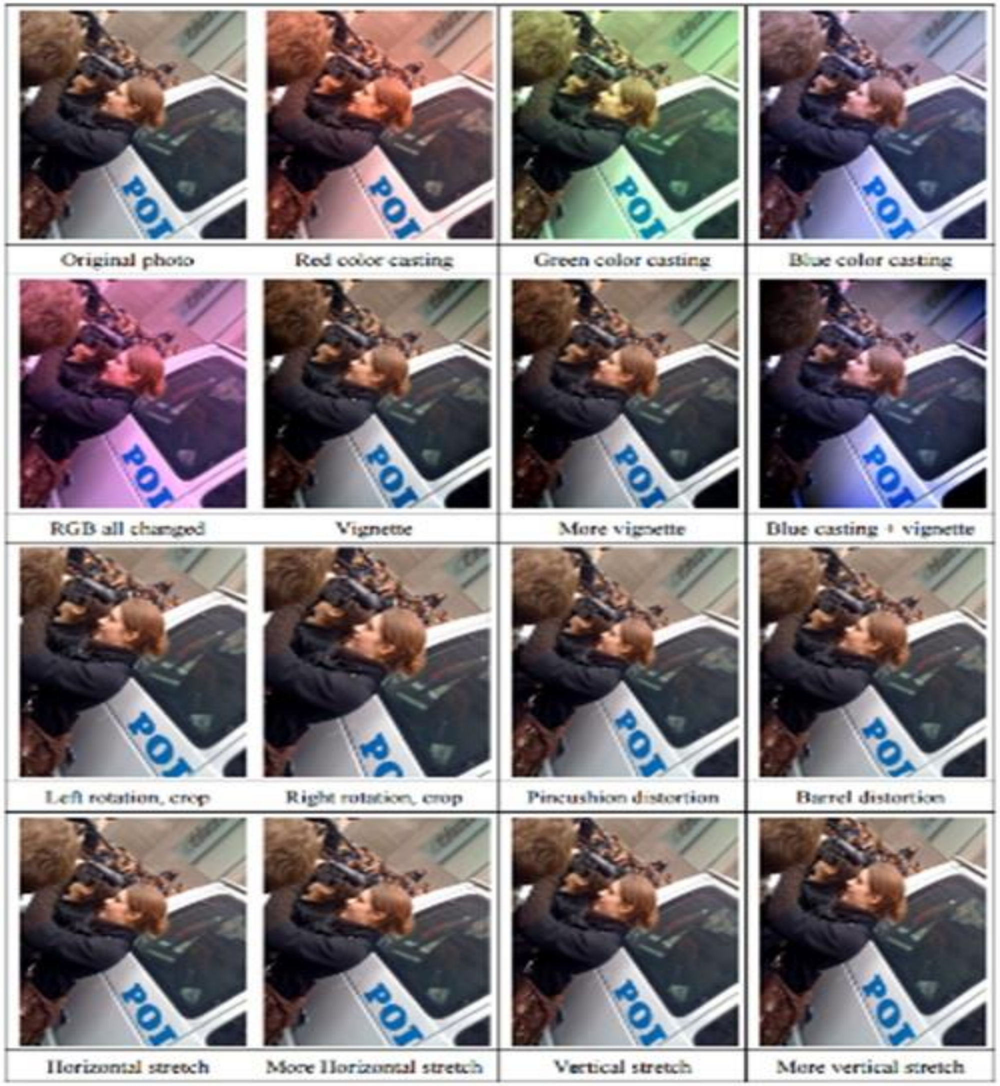
4. **Drop-out**: uncertain & vote

Slide: Jiabin Huang

Data Augmentation (Jittering)

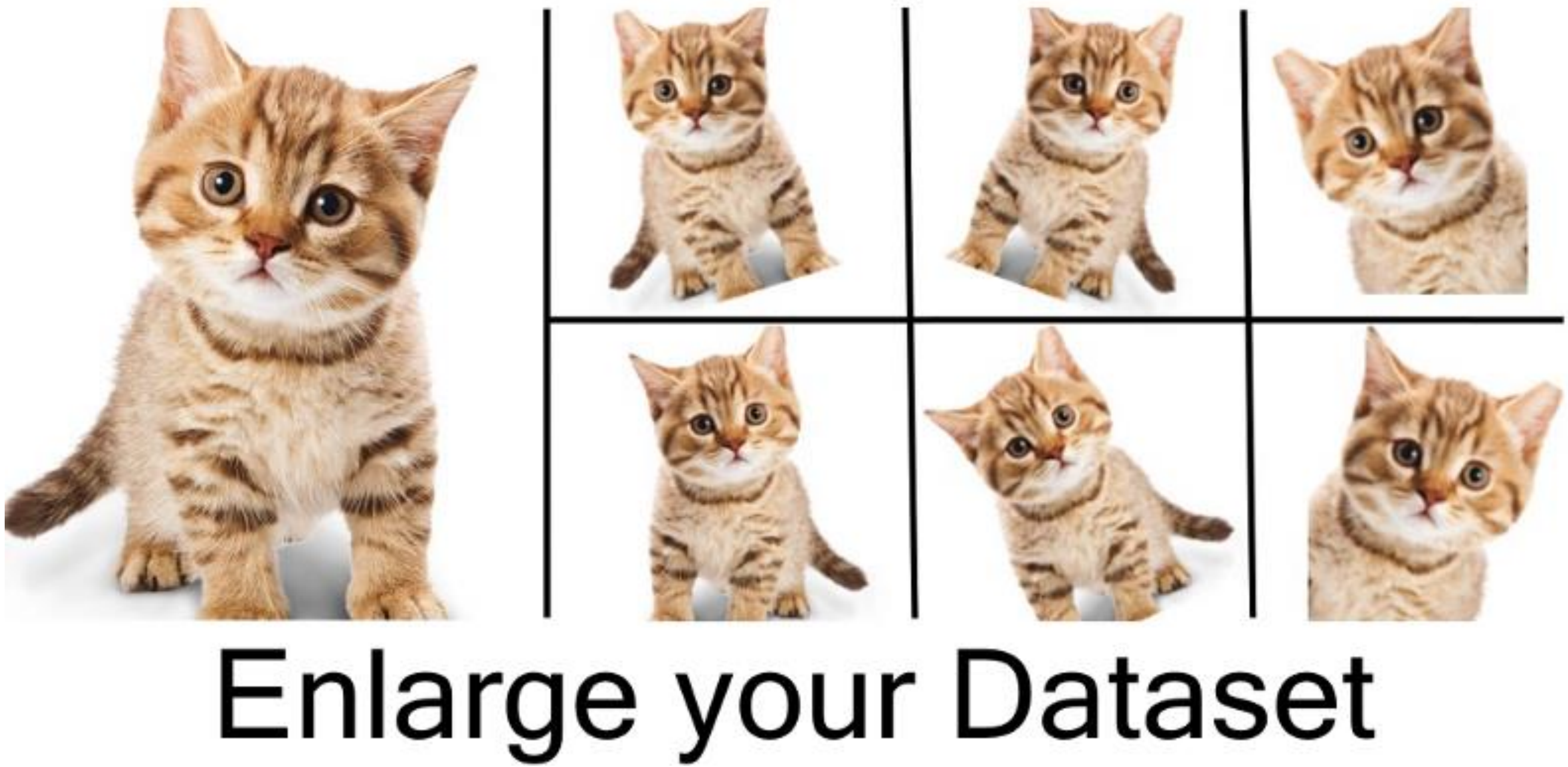
- Create *virtual* training samples
  – Horizontal flip
  – Random crop
  – Color casting
  – Geometric distortion

- Idea goes back to Pomerleau 1995 at least (neural net for car driving)

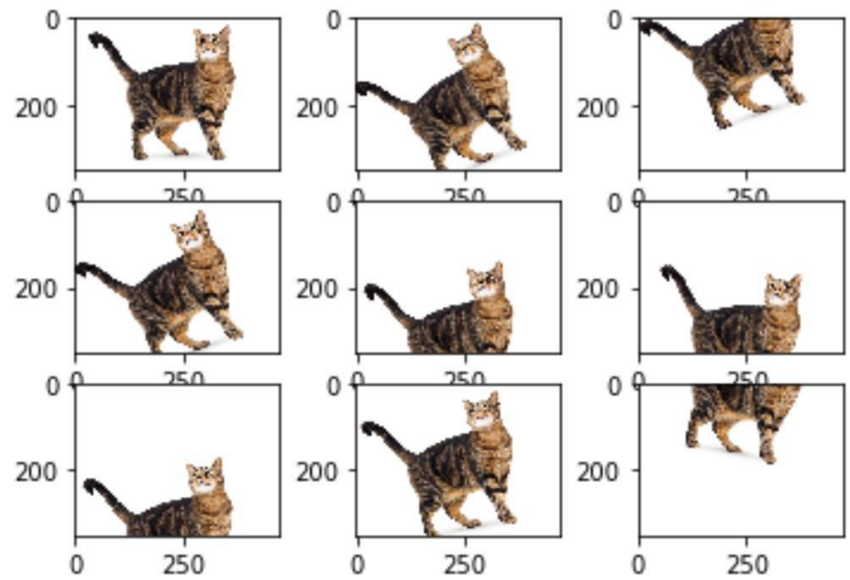


Deep Image [Wu et al. 2015]

Data Augmentation

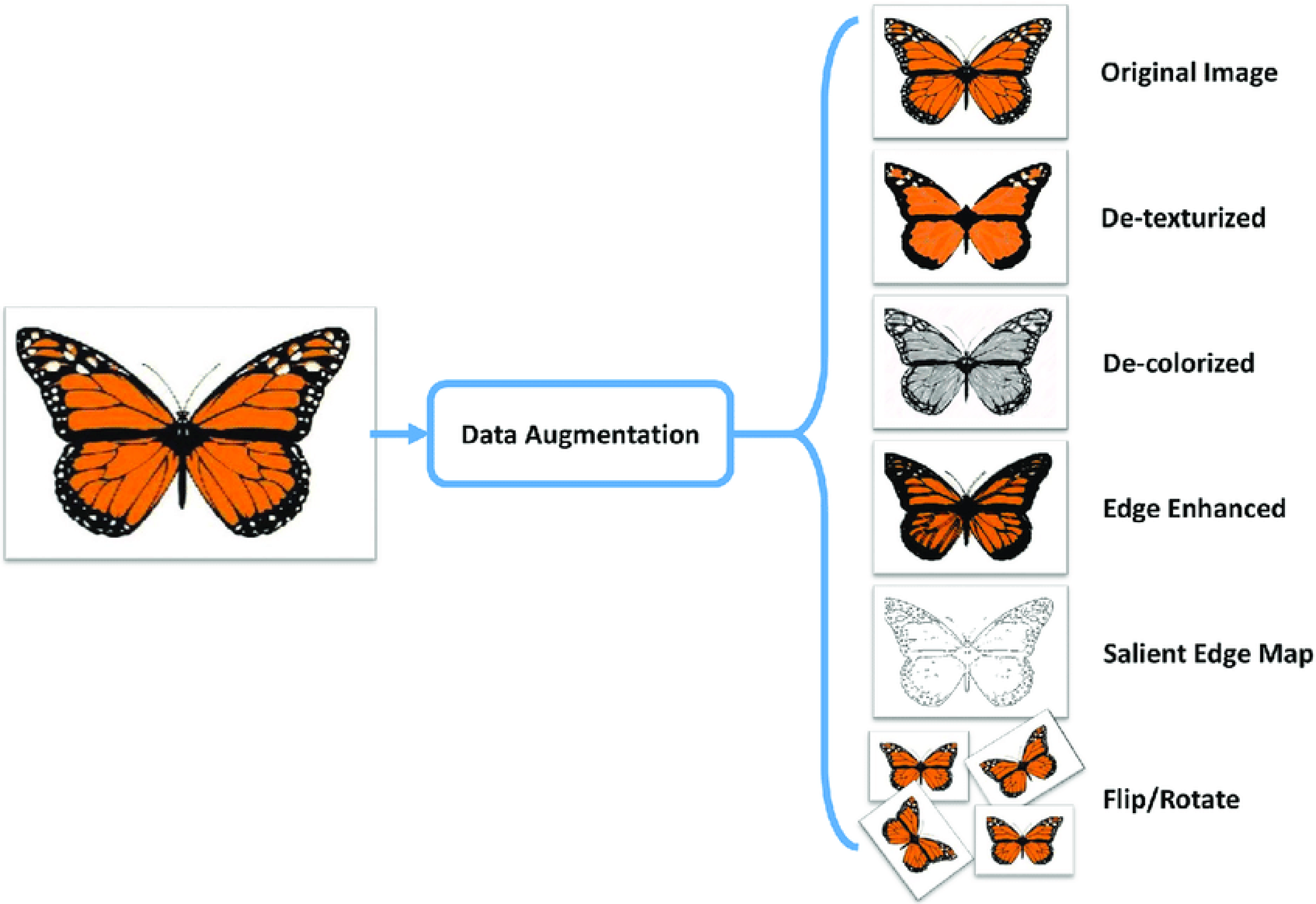


Data Augmentation

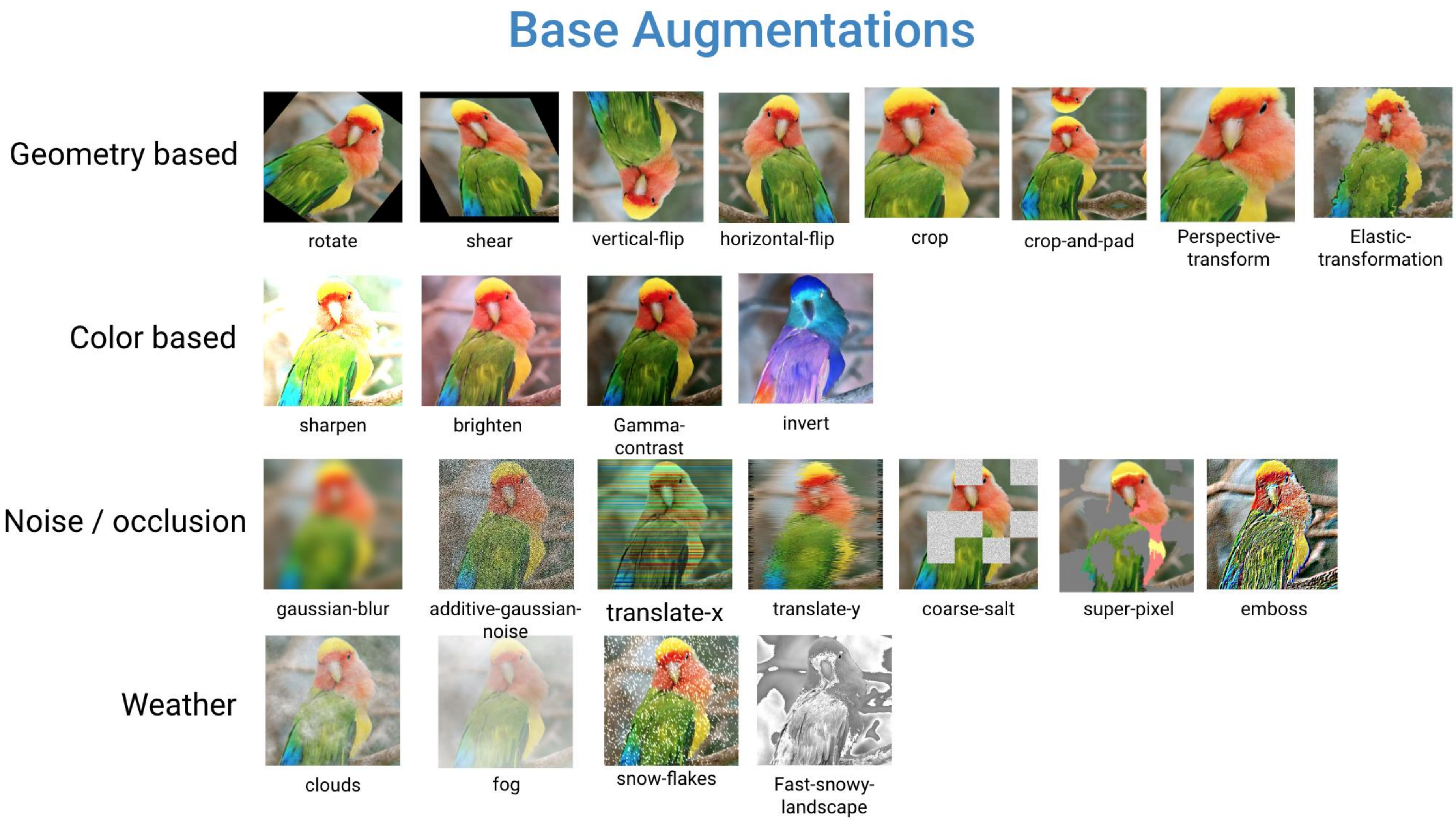


Affine Transformation for handling objects under different views

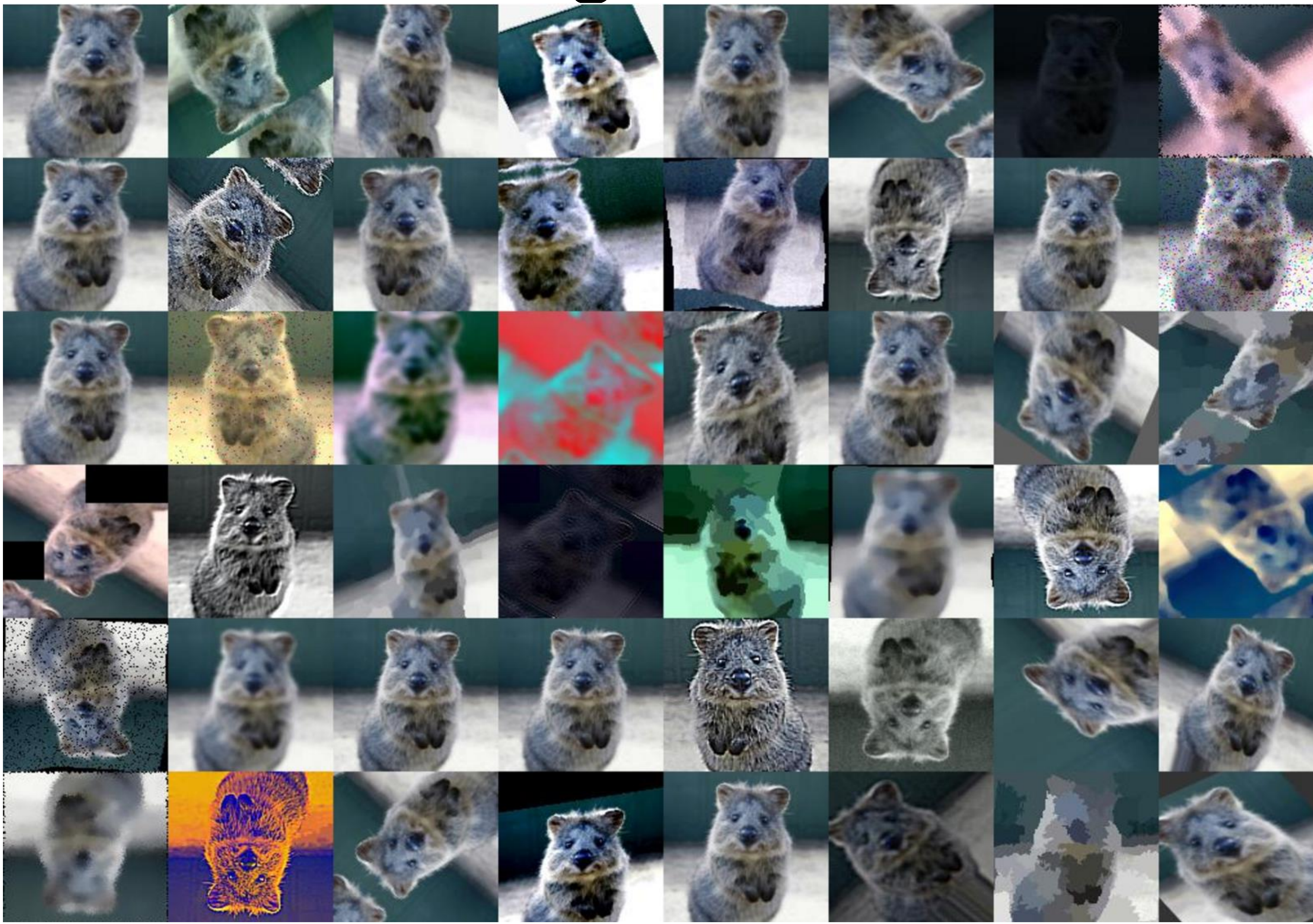
Data Augmentation



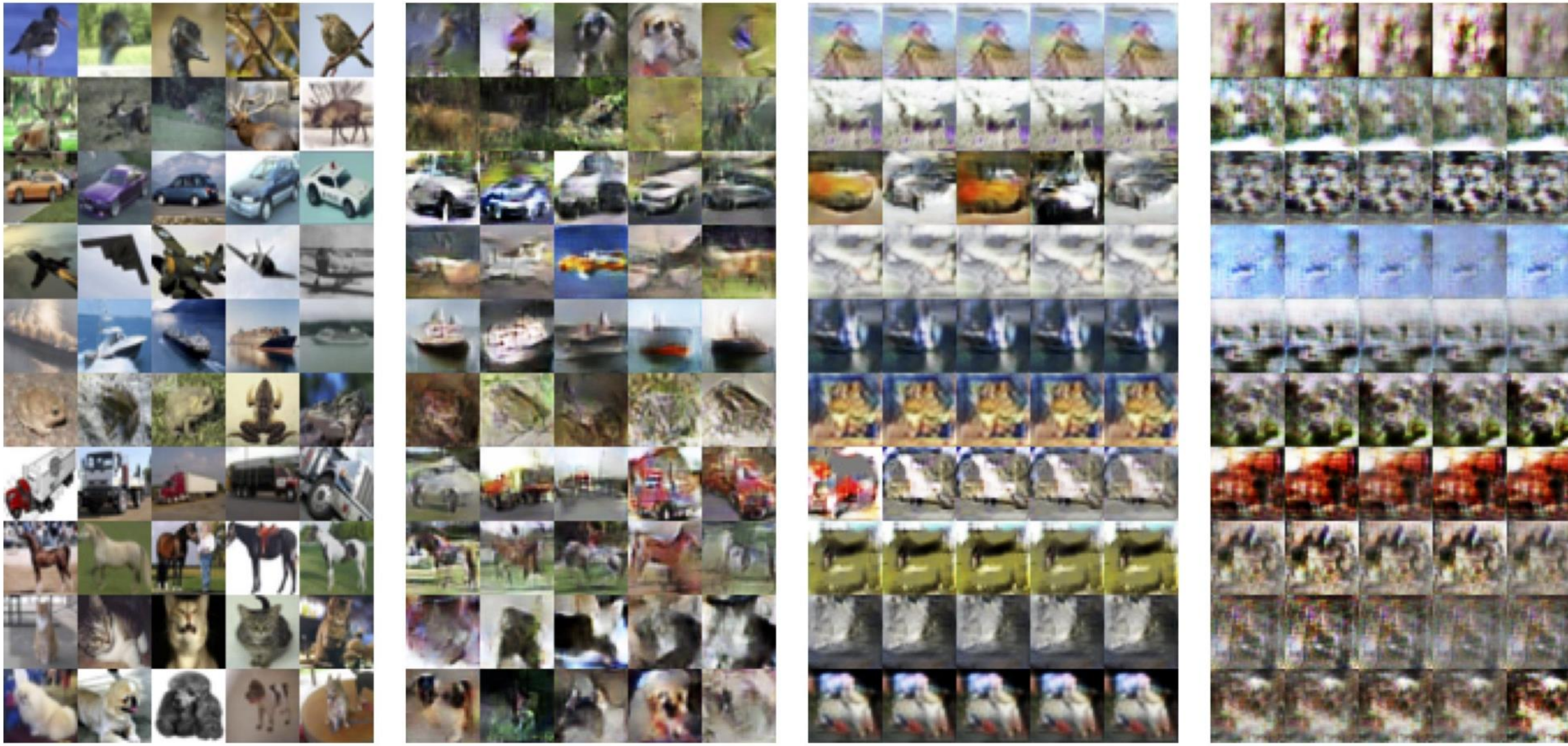
Data Augmentation



Data Augmentation



Data Augmentation

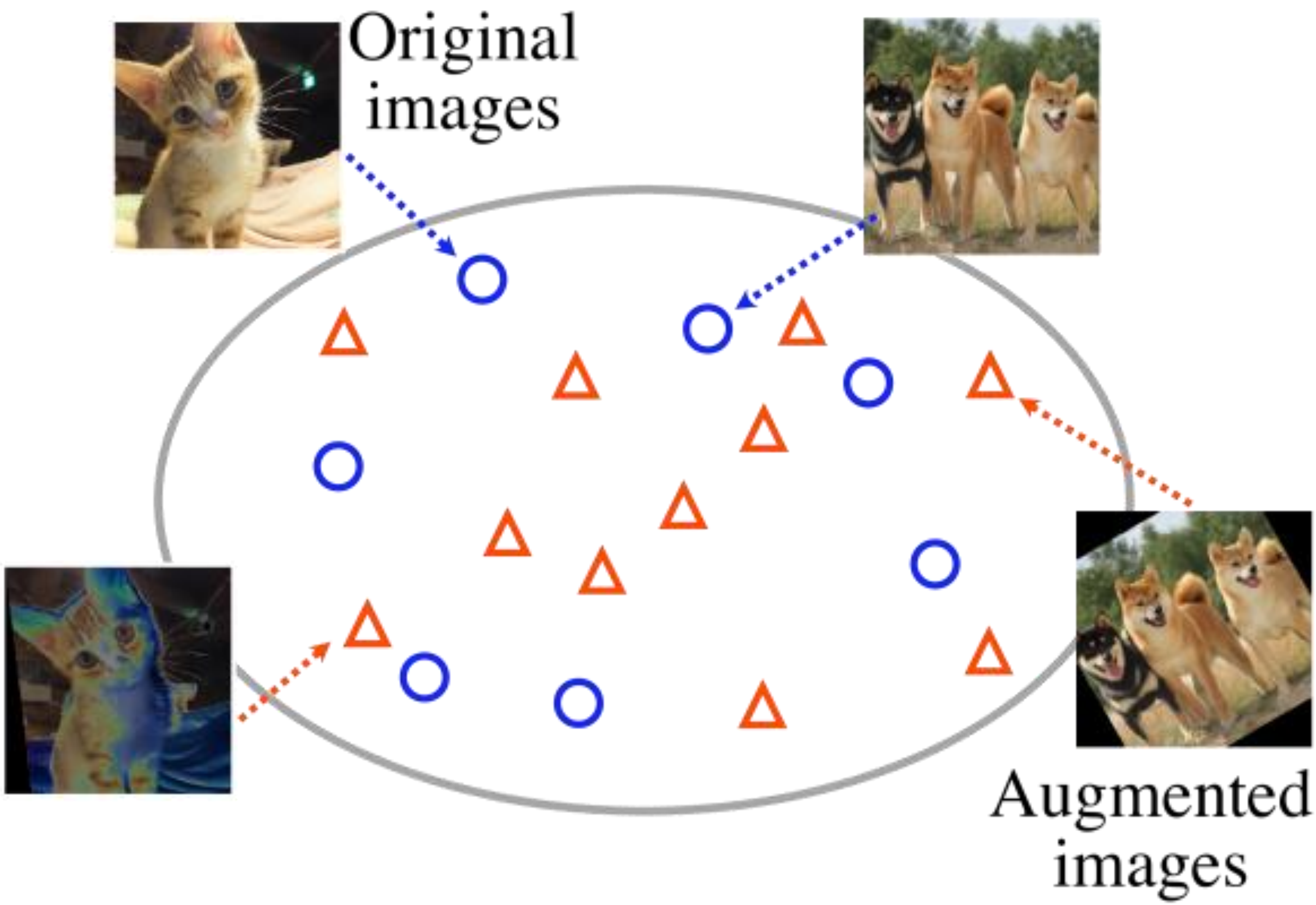


(a) Real image samples (b) *BAGAN* (c) *ACGAN* (d) *Simple GAN*

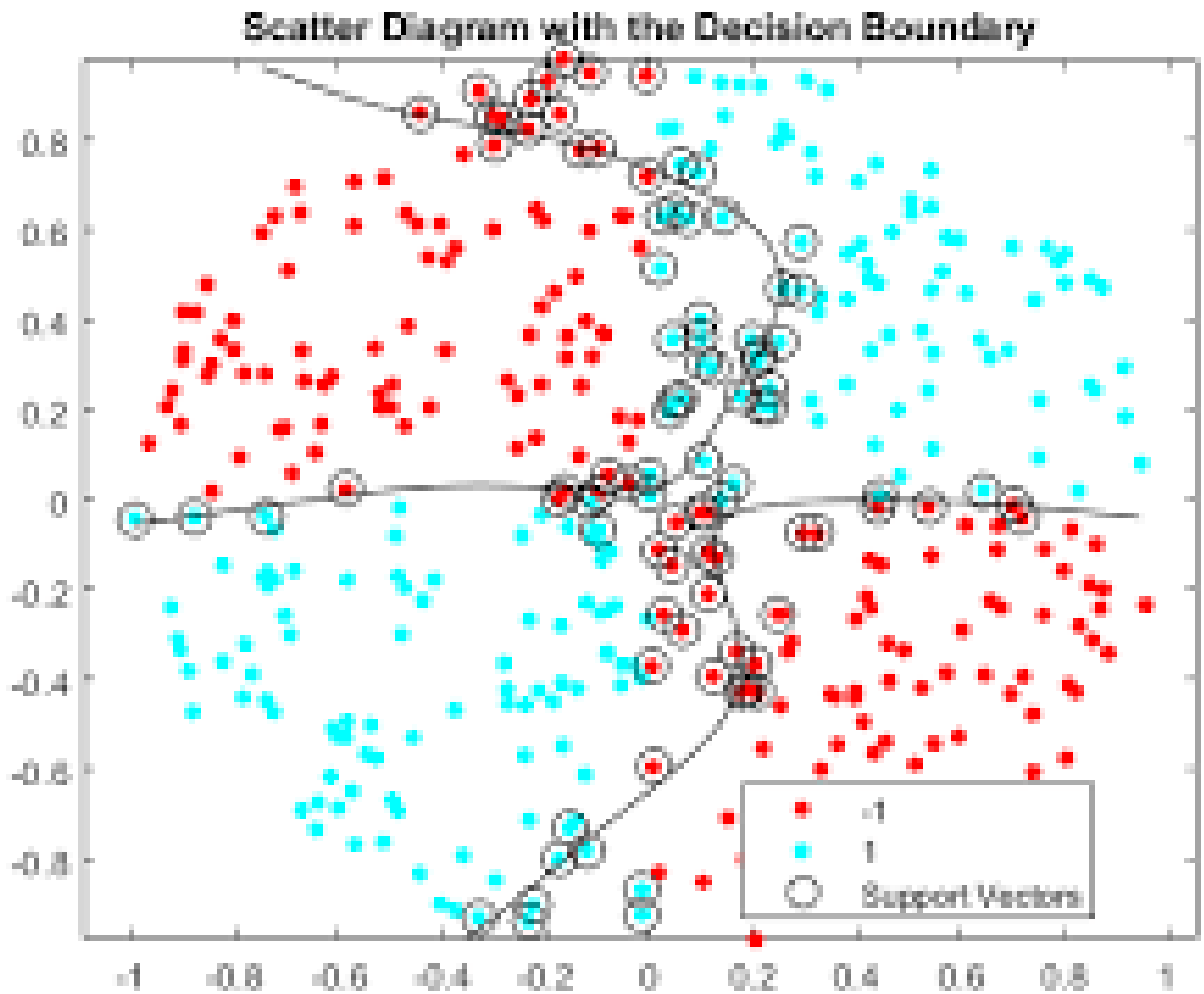
GAN-based Data Augmentation

Data Augmentation

Data Augmentation



Data Augmentation



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

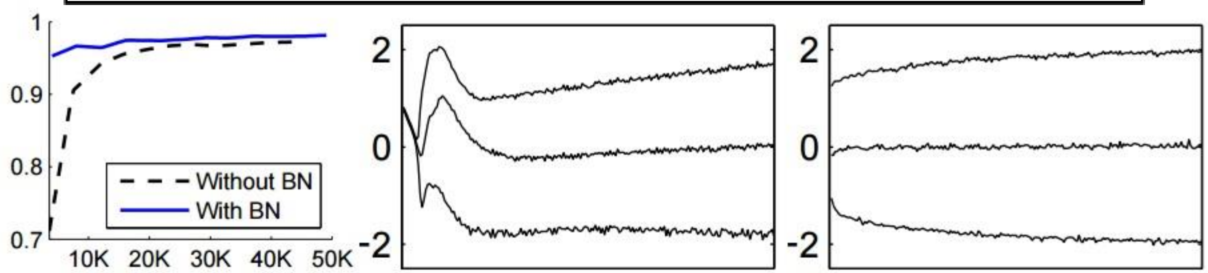
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

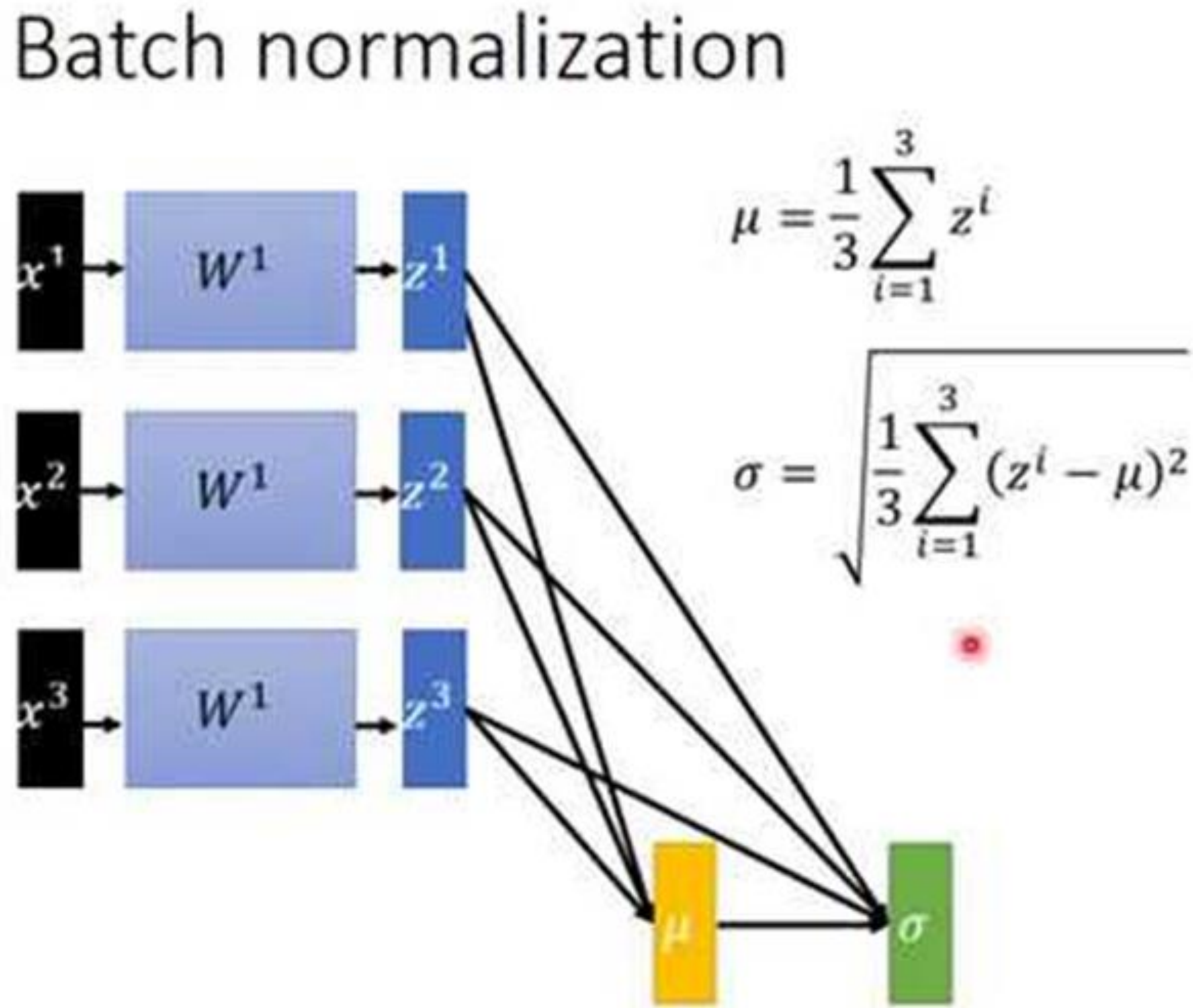
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



(a) (b) Without BN (c) With BN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

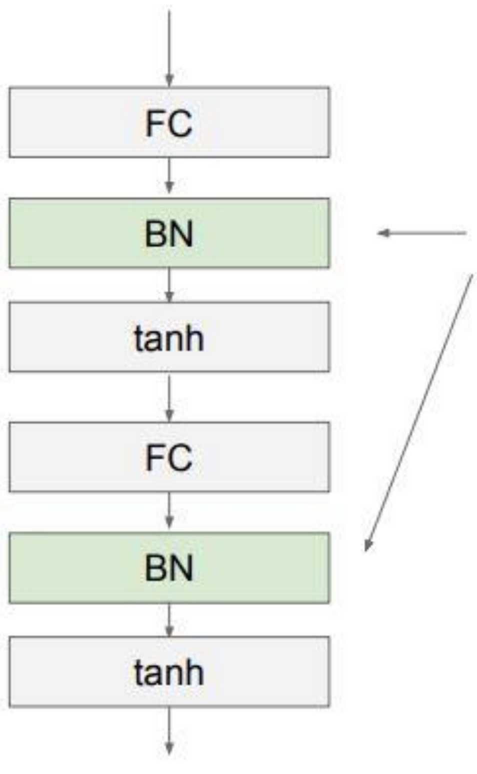
Batch Normalization



Batch Normalization

Batch Normalization

[Ioffe and Szegedy, 2015]

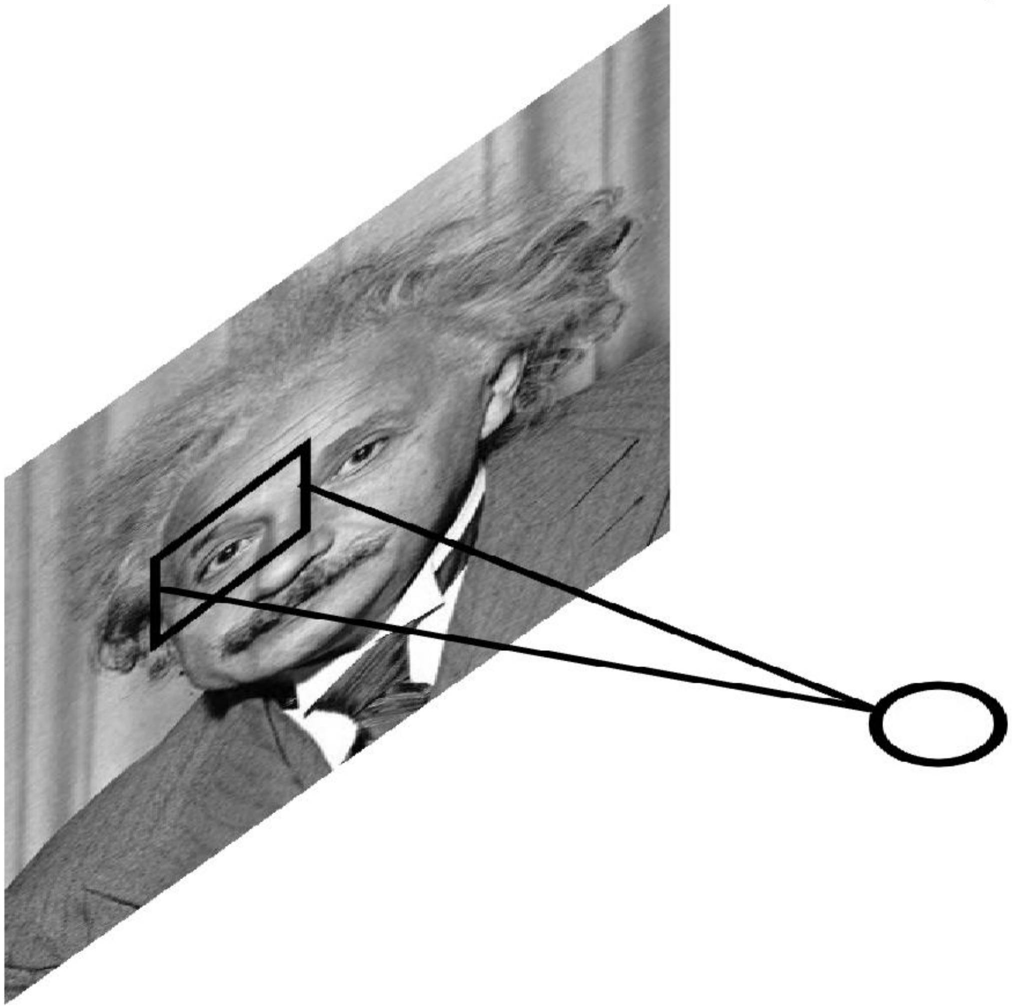


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

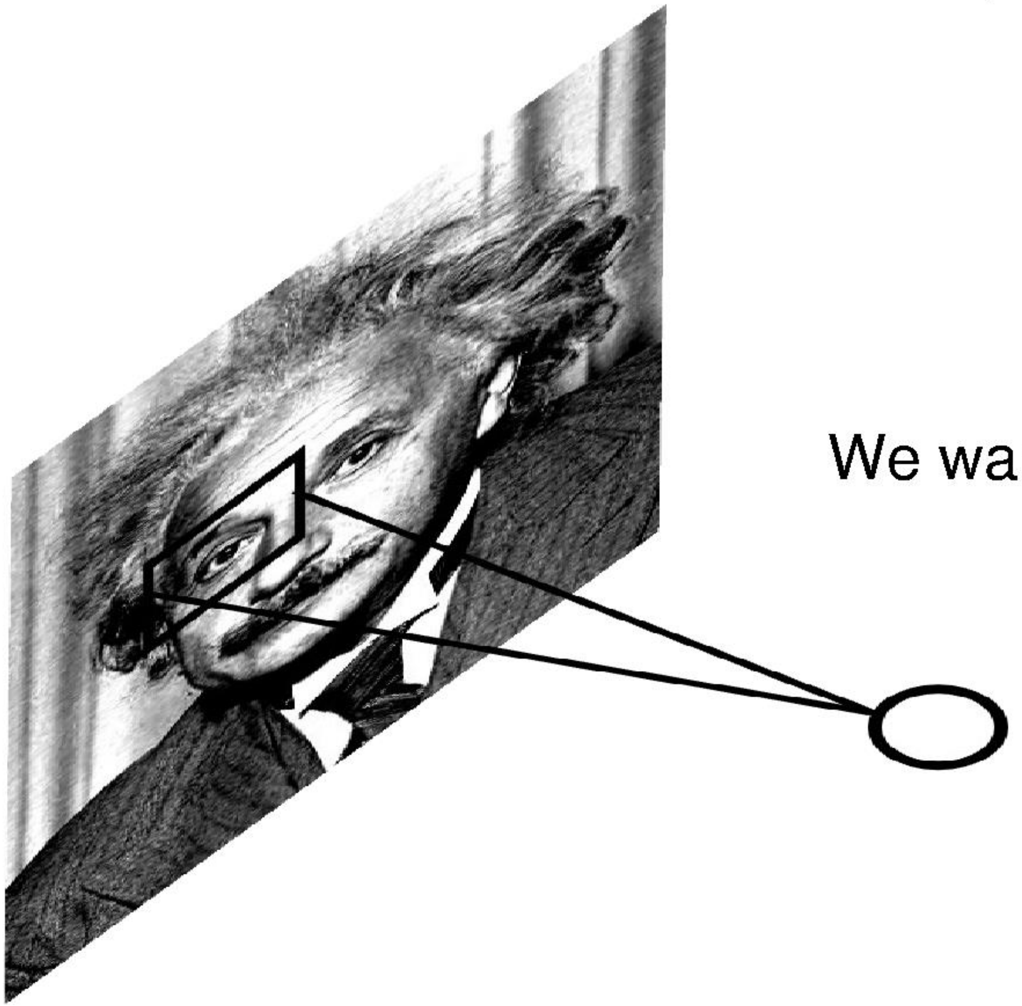
Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$



Local Contrast Normalization

$$h^{i+1}(x,y) = \frac{h^i(x,y) - m^i(N(x,y))}{\sigma^i(N(x,y))}$$

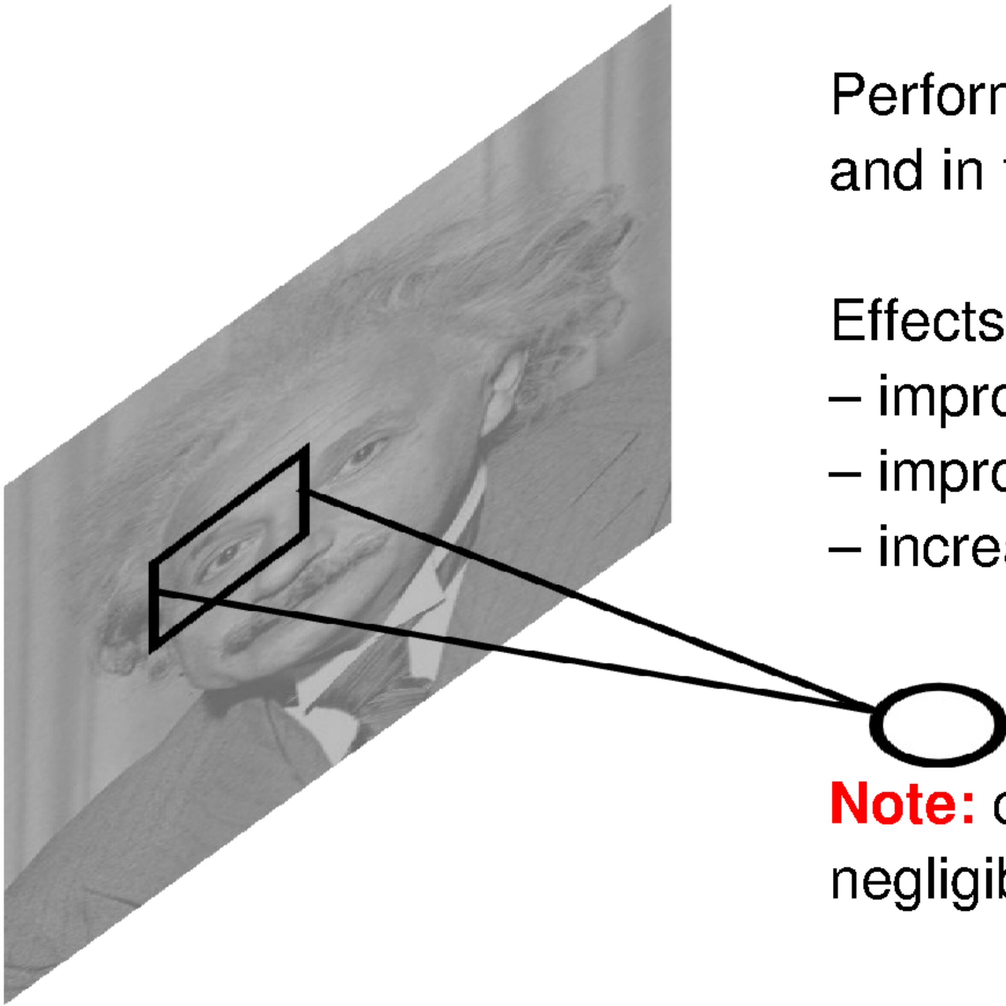


We want the same response.

Ranzato

Local Contrast Normalization

$$h^{i+1}(x,y)=\frac{h^{i}(x,y)-m^{i}(N(x,y))}{\sigma^{i}(N(x,y))}$$

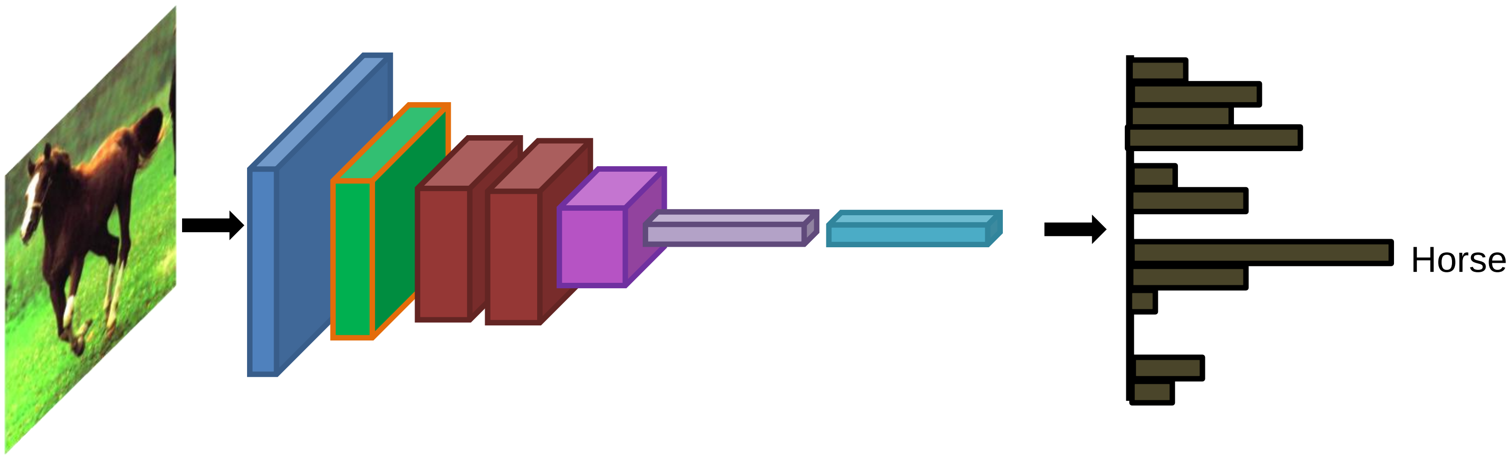


Performed also across features and in the higher layers..

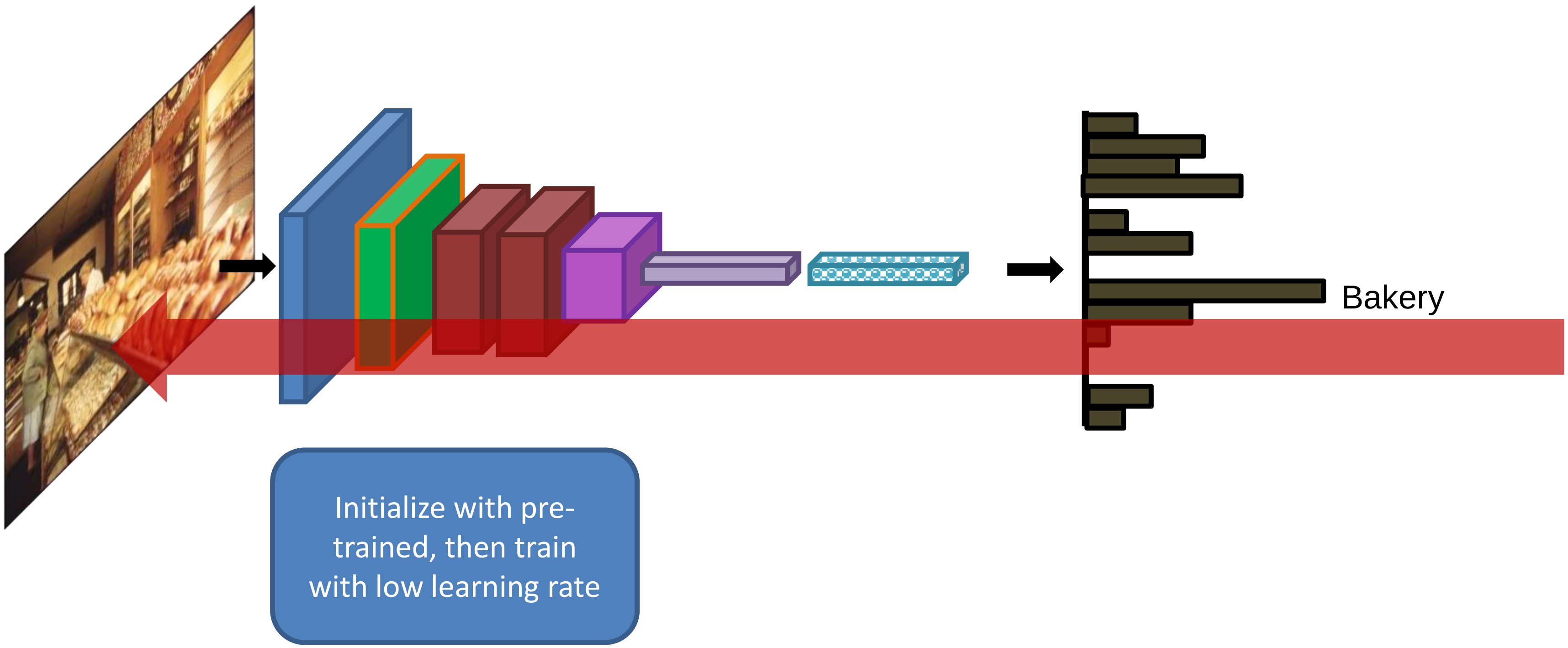
Effects:
– improves invariance
– improves optimization
– increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

Fine-tuning

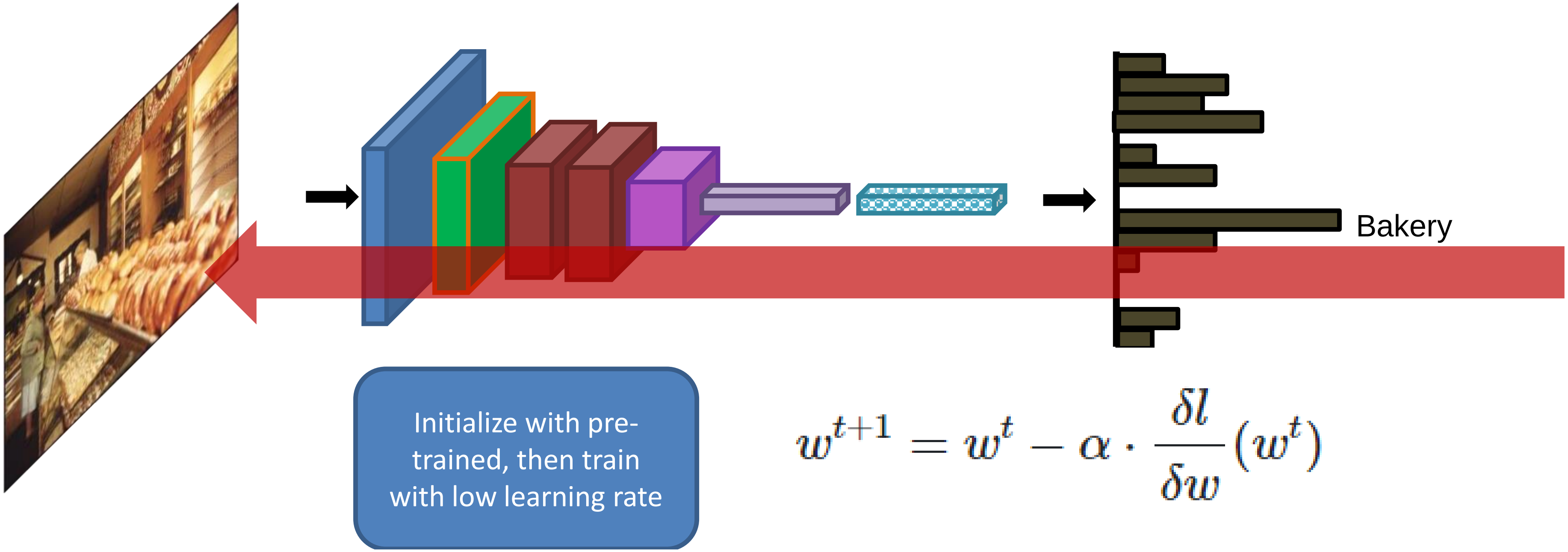


Fine-tuning



$$w^{t+1} = w^t - \alpha \cdot \frac{\delta l}{\delta w}(w^t)$$

Fine-tuning



$$w^{t+1} = w^t - \alpha \cdot \frac{\delta l}{\delta w}(w^t)$$

Start with:

θ_s : shared parameters
θ_o : task specific parameters for each old task
X_n, Y_n : training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \textsc{Cnn}(X_n, \theta_s, \theta_o)$ // compute output of old tasks for new data
$\theta_n \leftarrow \textsc{RandInit}(|\theta_n|)$ // randomly initialize new parameters

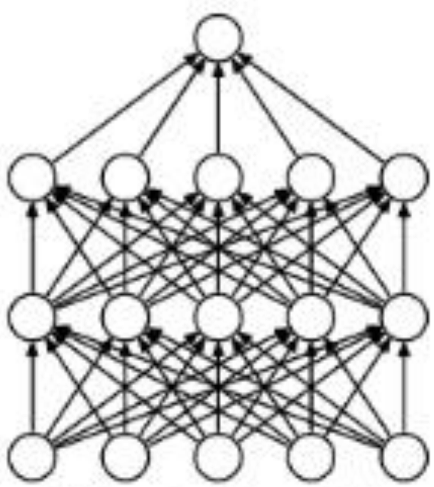
Train:

Define $\hat{Y}_o \equiv \textsc{Cnn}(\hat{X}_n, \hat{\theta}_s, \hat{\theta}_o)$ // old task output
Define $\hat{Y}_n \equiv \textsc{Cnn}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // new task output

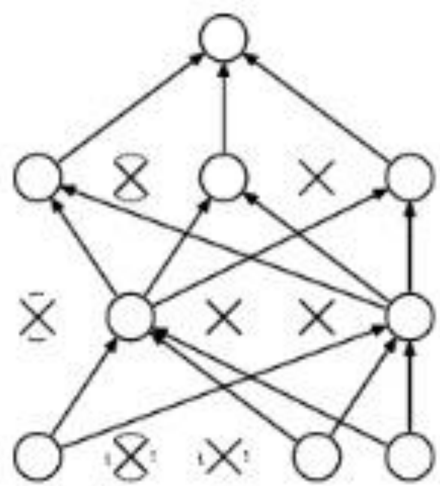
$$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{argmin} \left(\lambda_o L_{old}(Y_o, \hat{Y}_o) + L_{new}(Y_n, \hat{Y}_n) + R(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n)\right)$$

Dropout

- Similar to bagging (approximation of bagging)
- Act like **regularizer** (reduce overfit)
- Instead of using all neurons, “dropout” some neurons randomly (usually 0.5 probability)

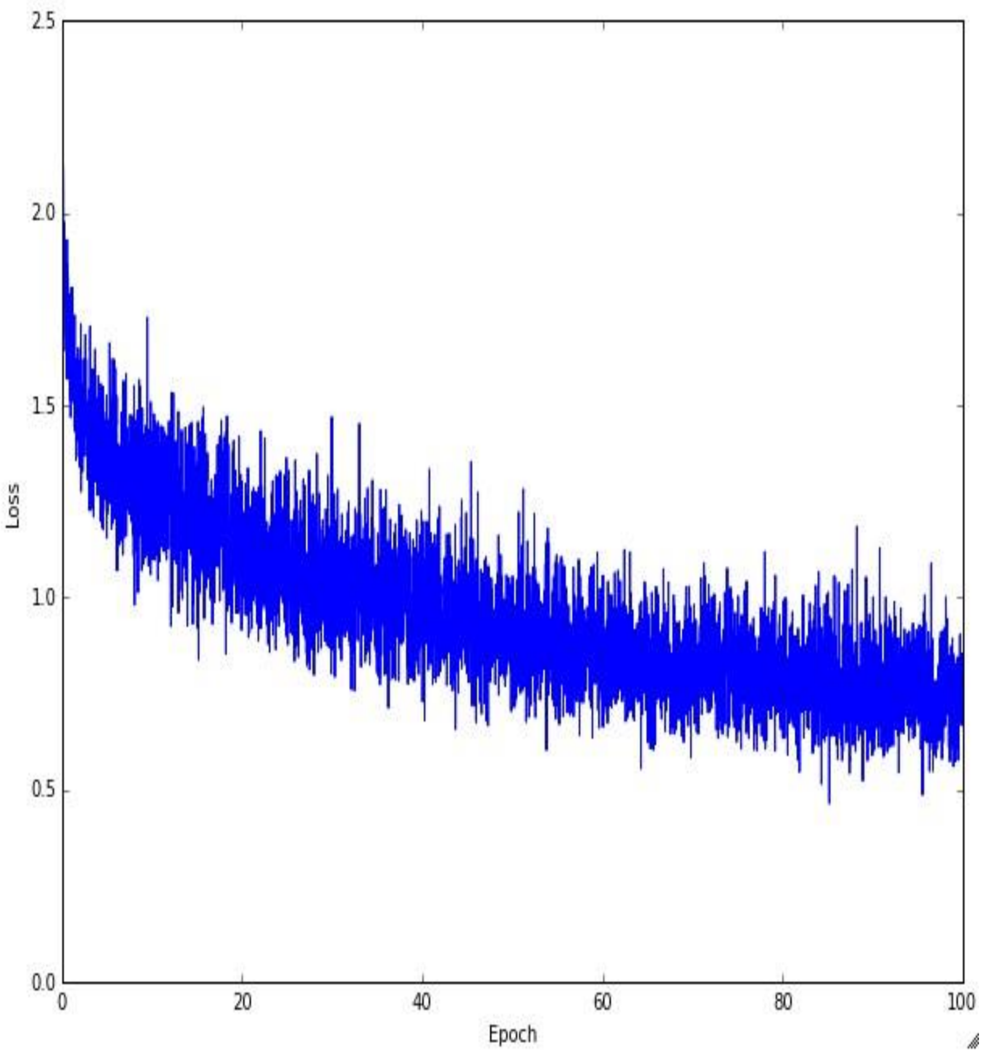


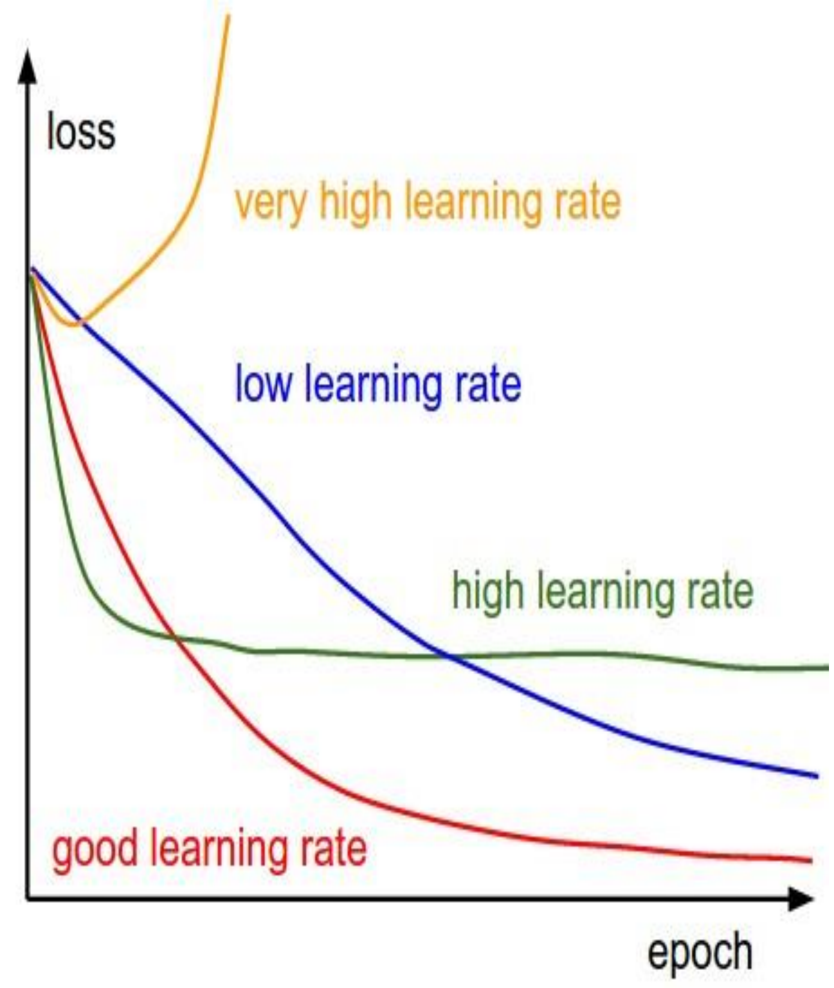
(a) Standard Neural Net



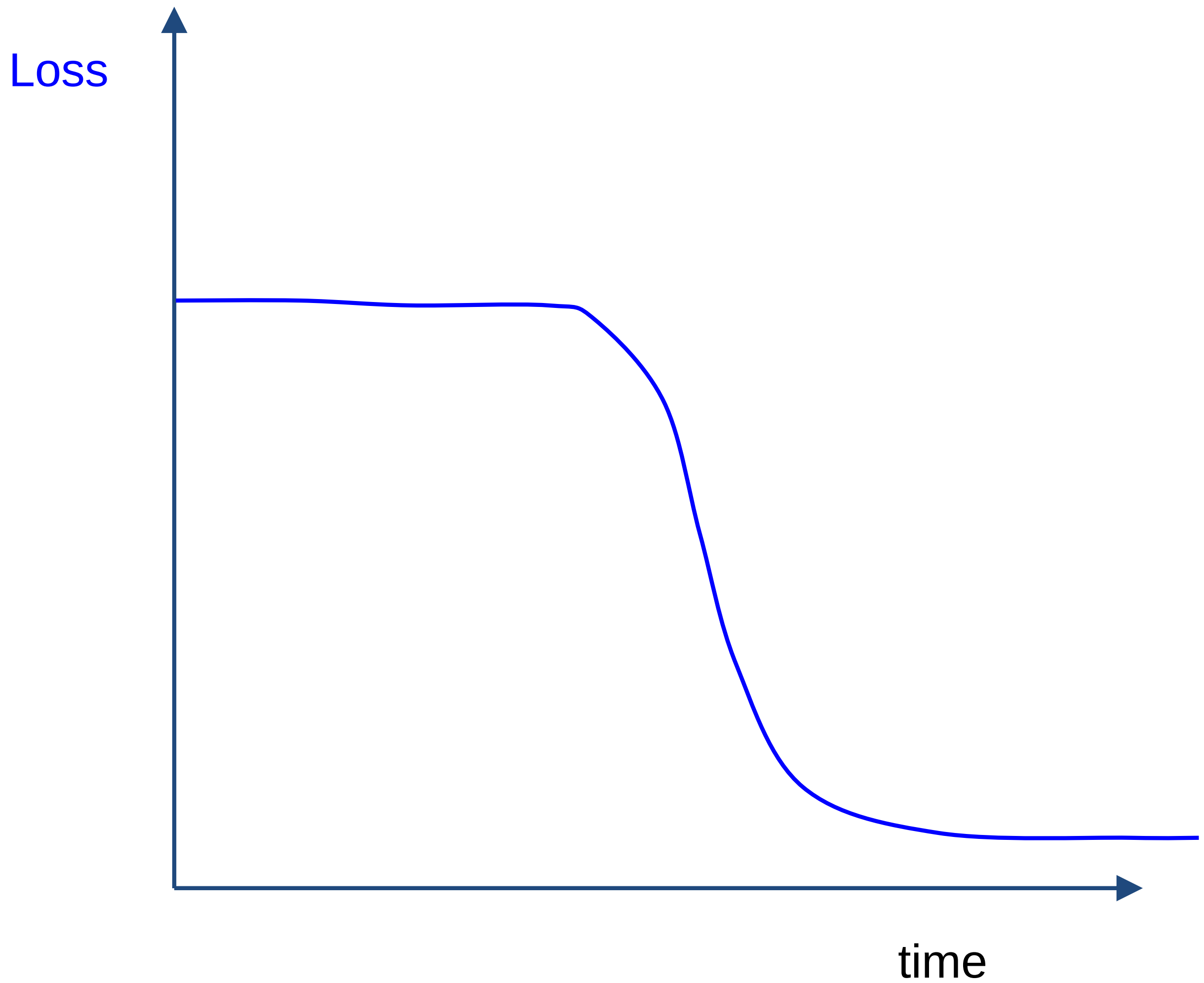
(b) After applying dropout.

Learning Rate

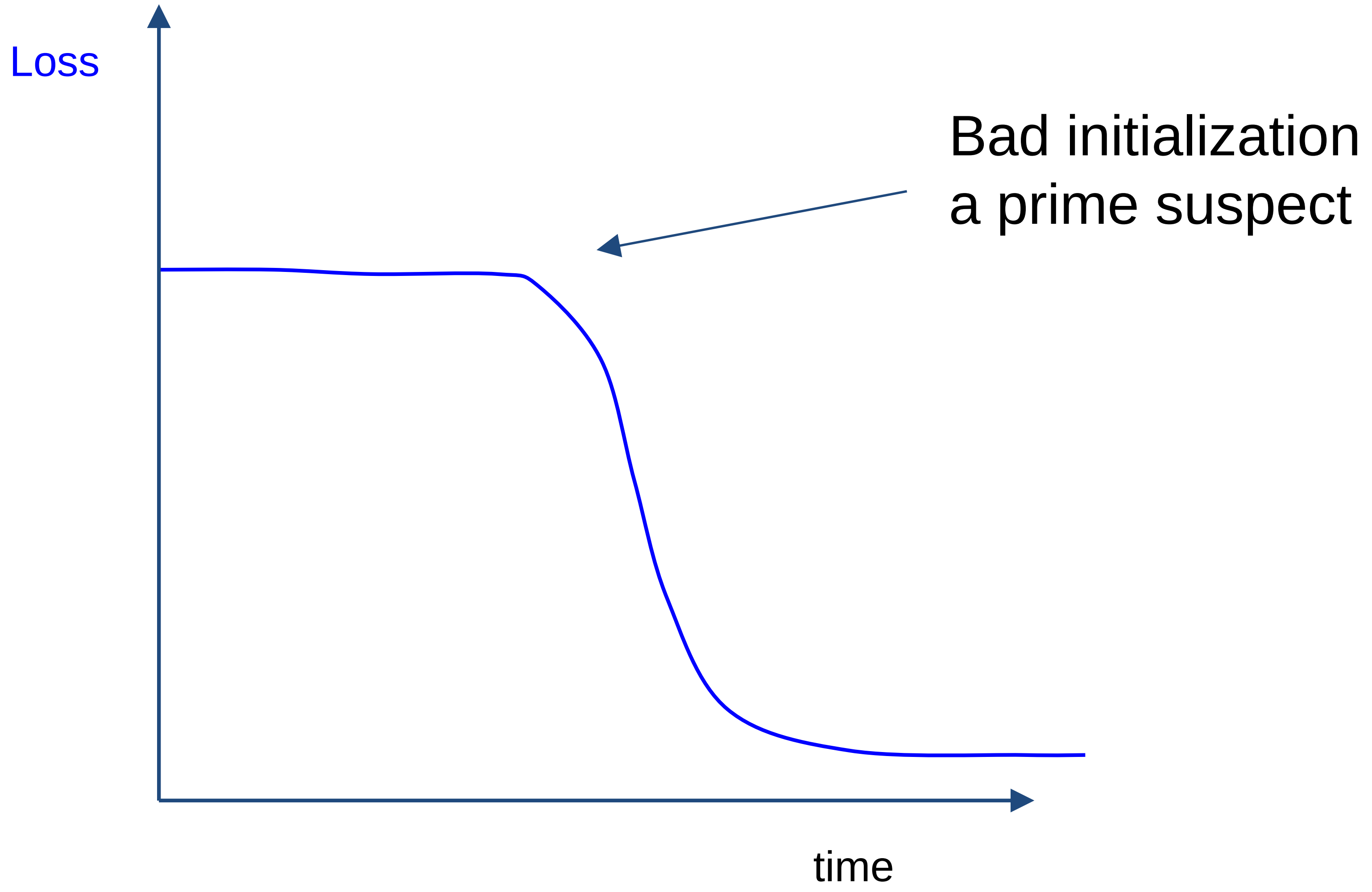




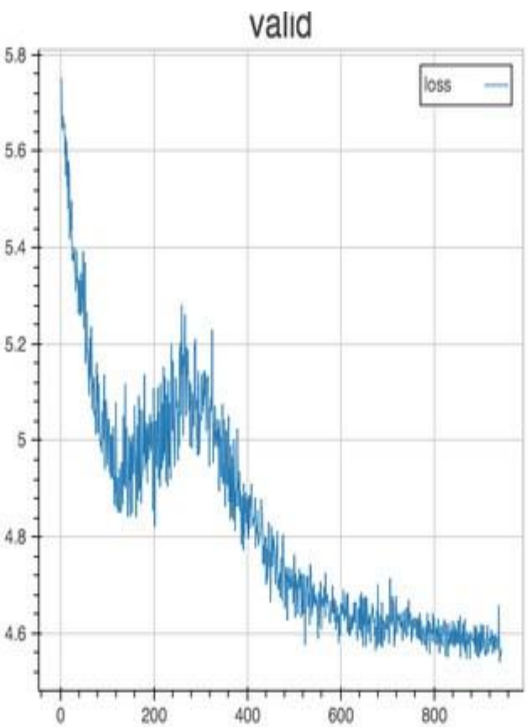
Loss Visualization

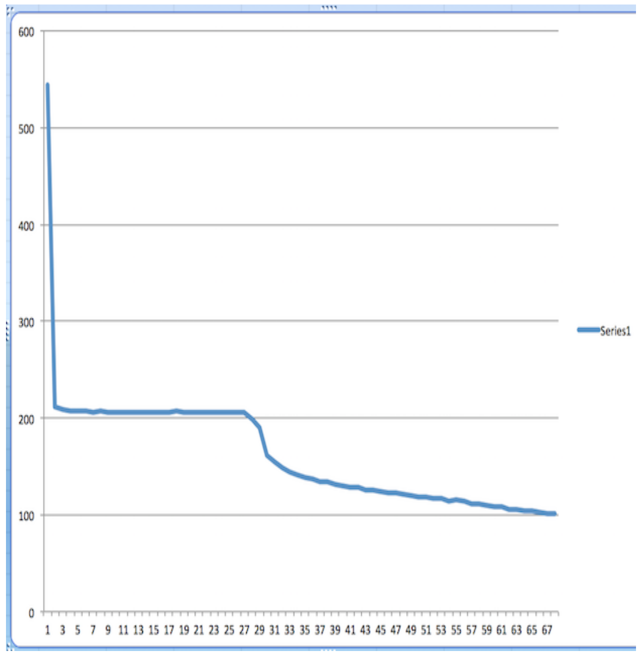


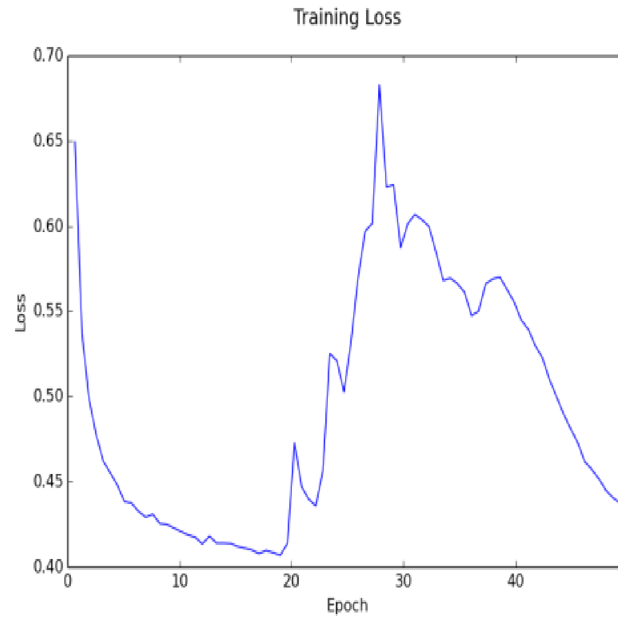
Loss Visualization



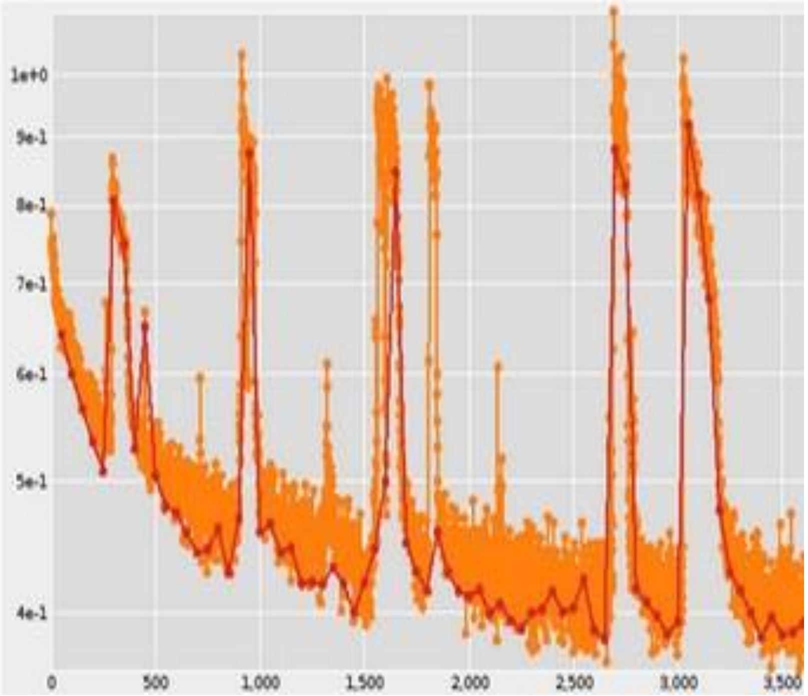
lossfunctions.tumblr.com

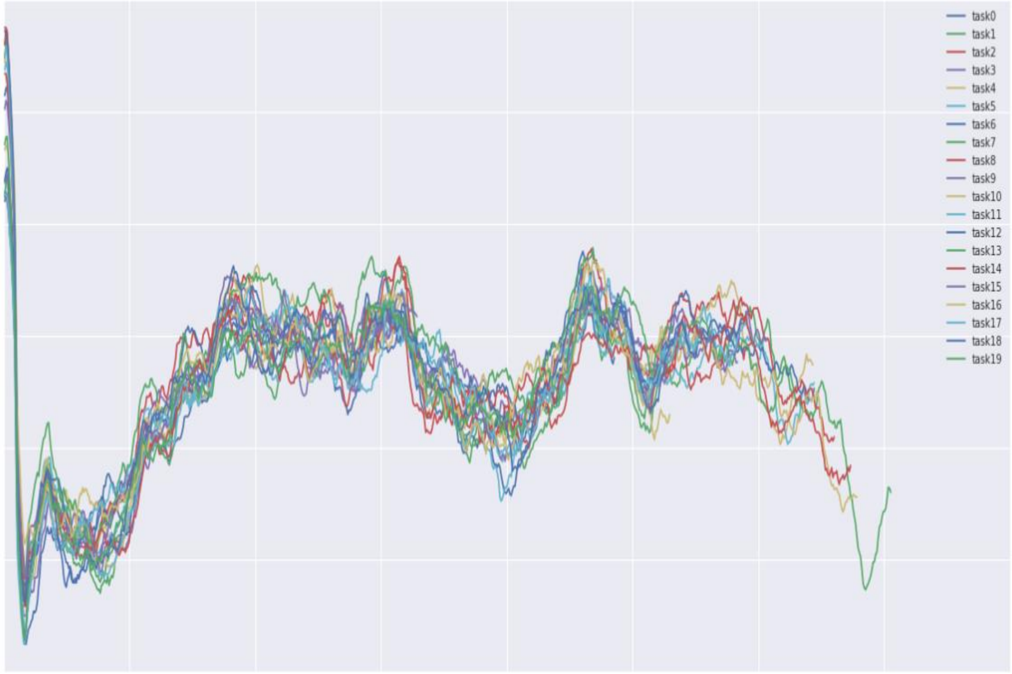


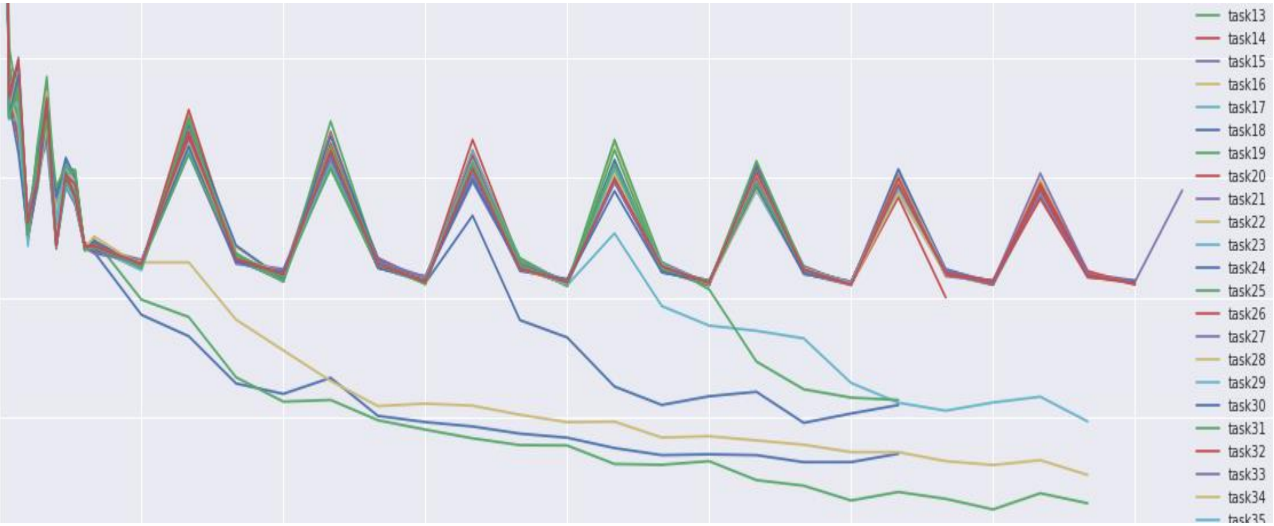




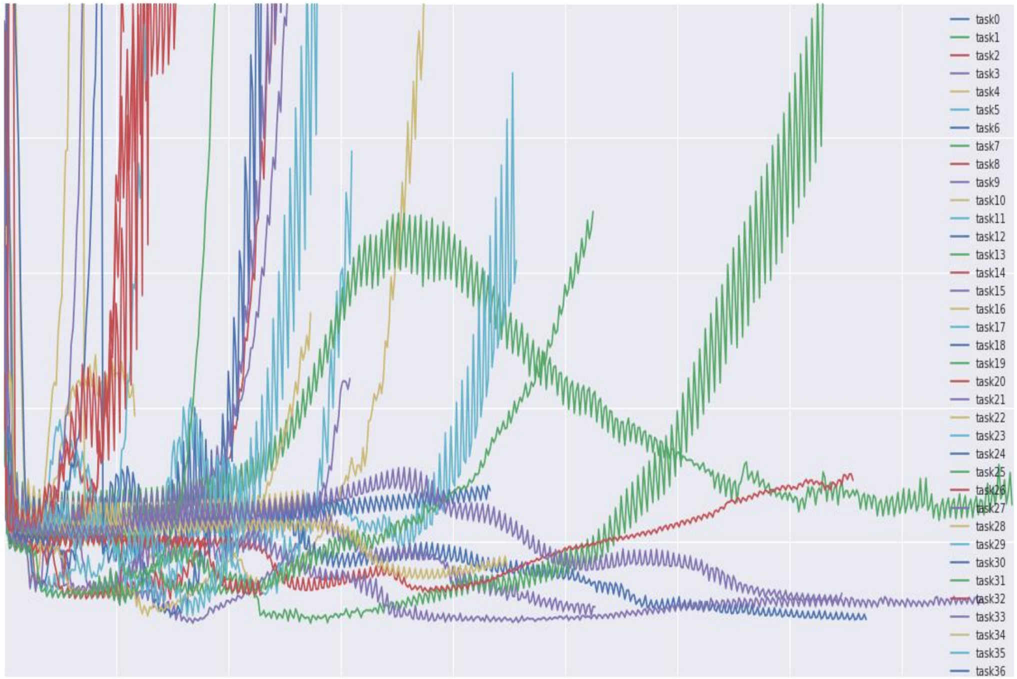
lossfunctions.tumblr.com



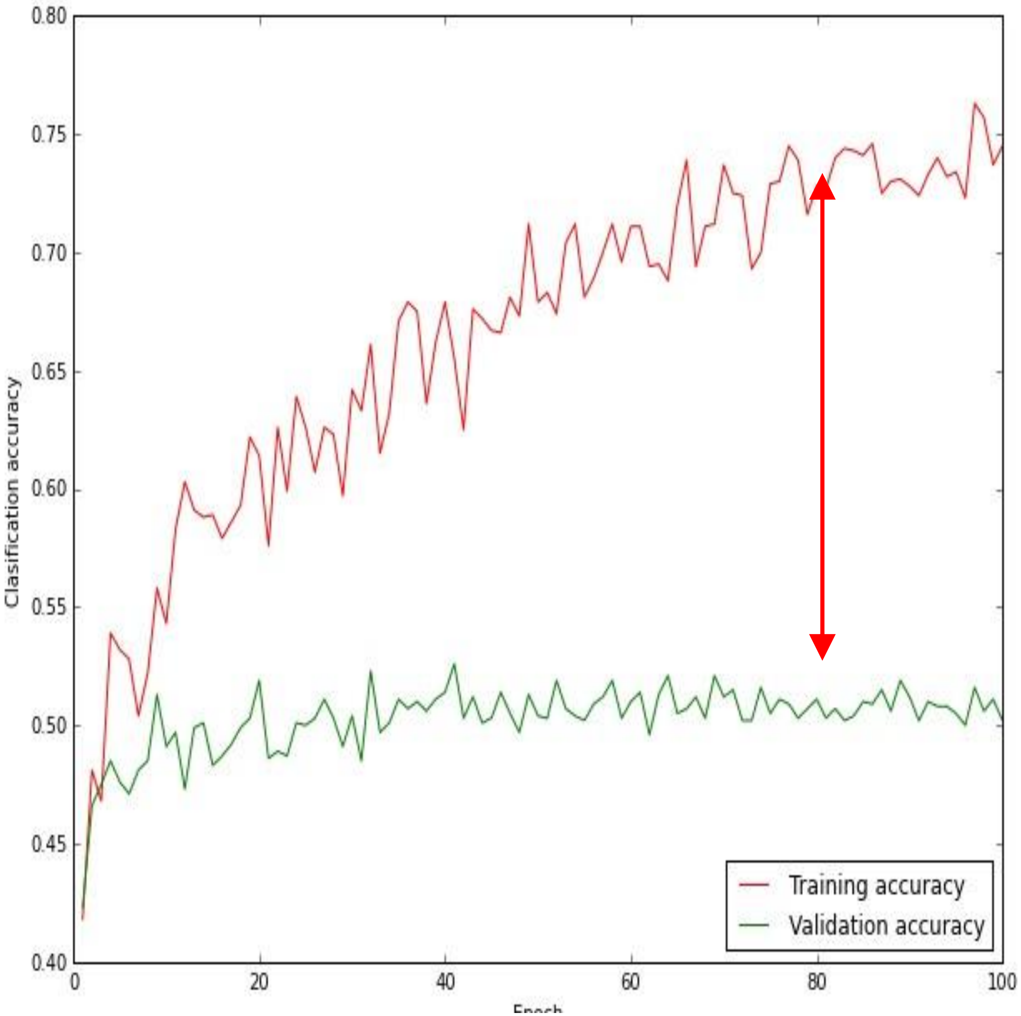




lossfunctions.tumblr.com



Accuracy Visualization



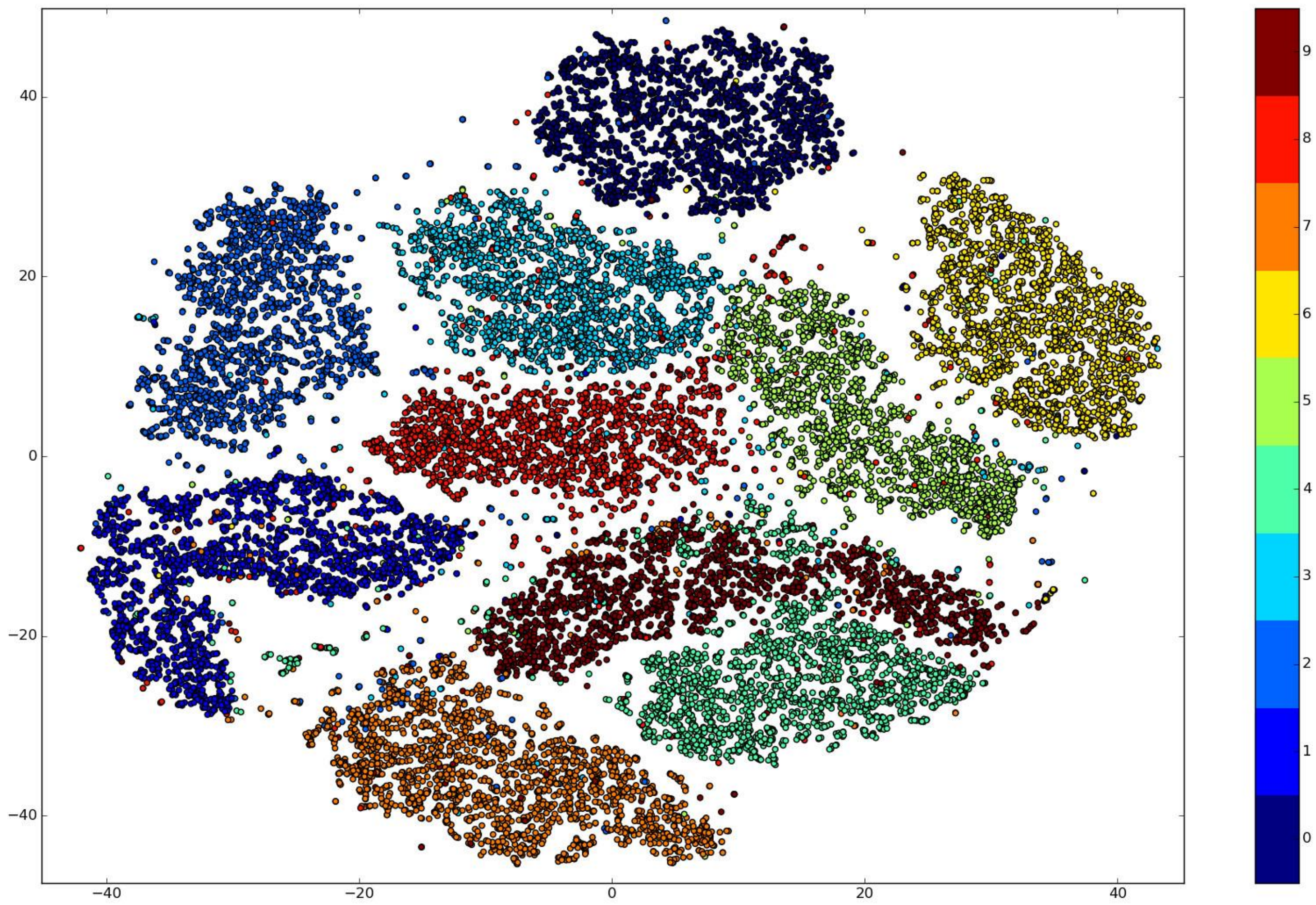
big gap = overfitting

=> increase regularization strength?

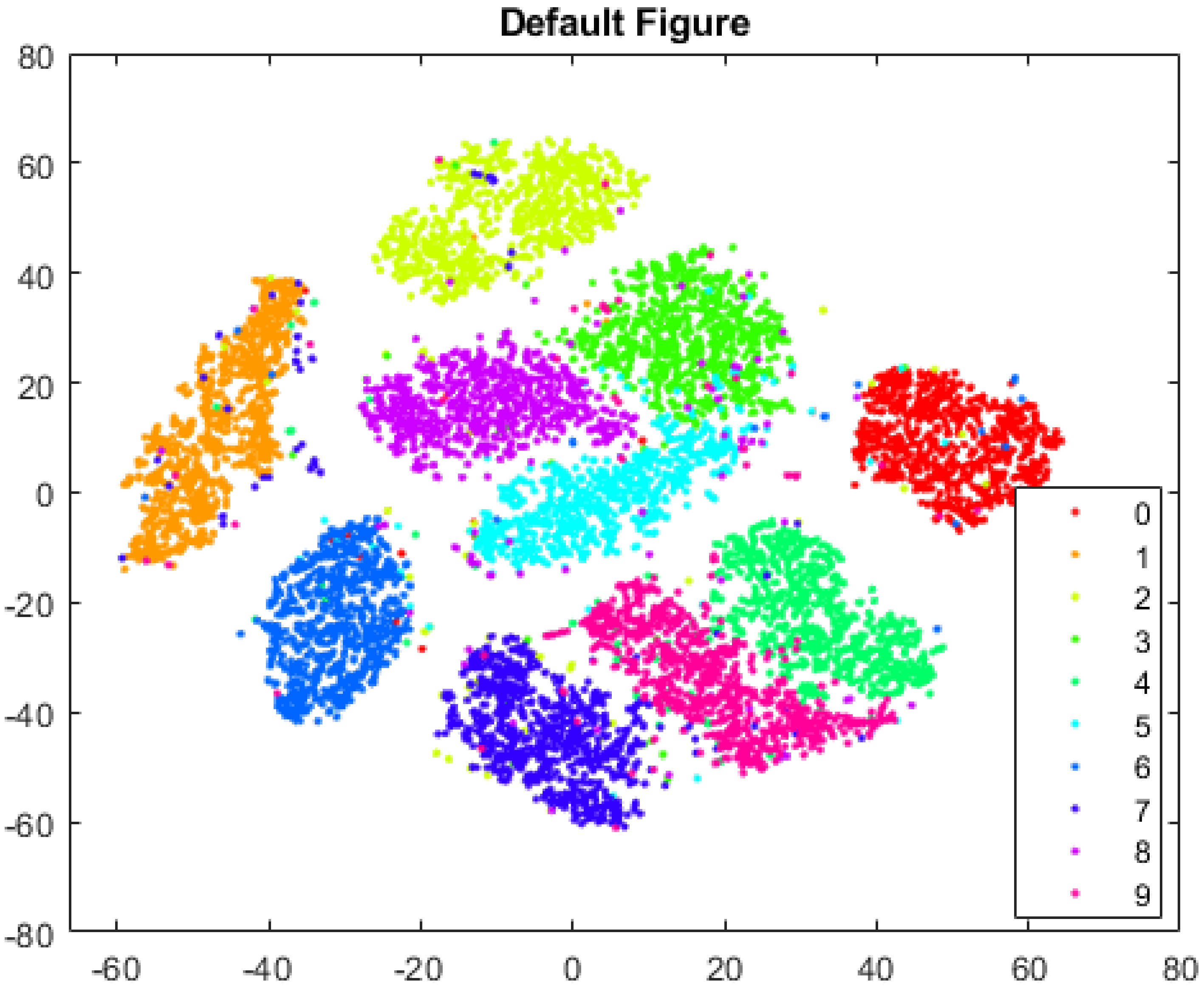
no gap

=> increase model capacity?

t-SNE

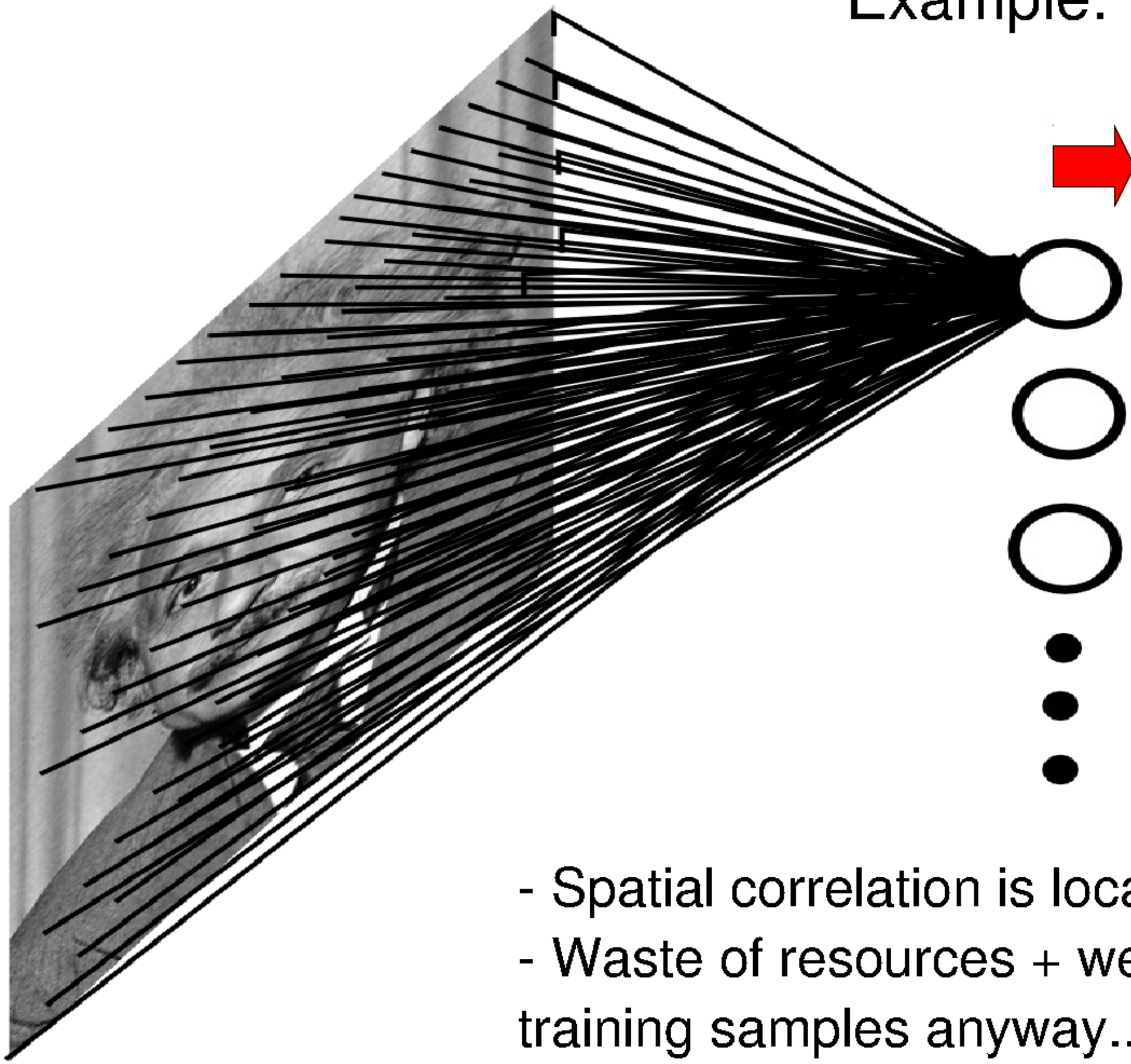


t-SNE


Default Figure
80
60
40
20
0
-20
-40
-60
-80
-60
-40
-20
0
20
40
60
80
0
1
2
3
4
5
6
7
8
9

Additional Interpretations of CNN Operators

Fully Connected Layer



Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Convolutional Layer

Question: What is the size of the output? What's the computational cost?

Answer: It is proportional to the number of filters and depends on the stride. If kernels have size KxK, input has size DxD, stride is 1, and there are M input feature maps and N output feature maps then:
- the input has size M@DxD
- the output has size N@(D-K+1)x(D-K+1)
- the kernels have MxNxKxK coefficients (which have to be learned)
- cost: M*K*K*N*(D-K+1)*(D-K+1)

Question: How many feature maps? What's the size of the filters?

Answer: Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute).
The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).

Ranzato

Key Ideas

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input
- share the weight across space

This is called: **convolutional layer.**
A network with convolutional layers is called **convolutional network.**

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998

Pooling Layer

Question: What is the size of the output? What's the computational cost?

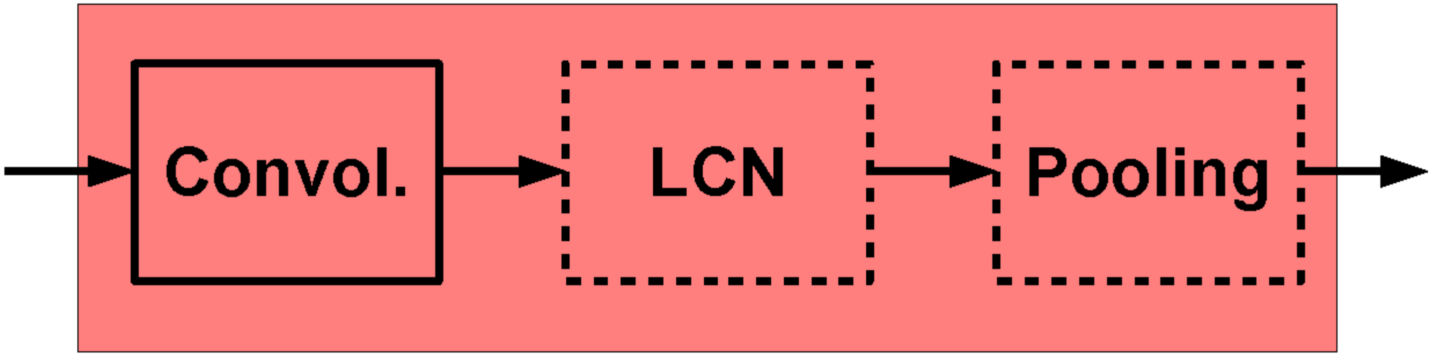
Answer: The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size KxK, and the input has size DxD with M input feature maps, then:
- output is M@(D/K)x(D/K)
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

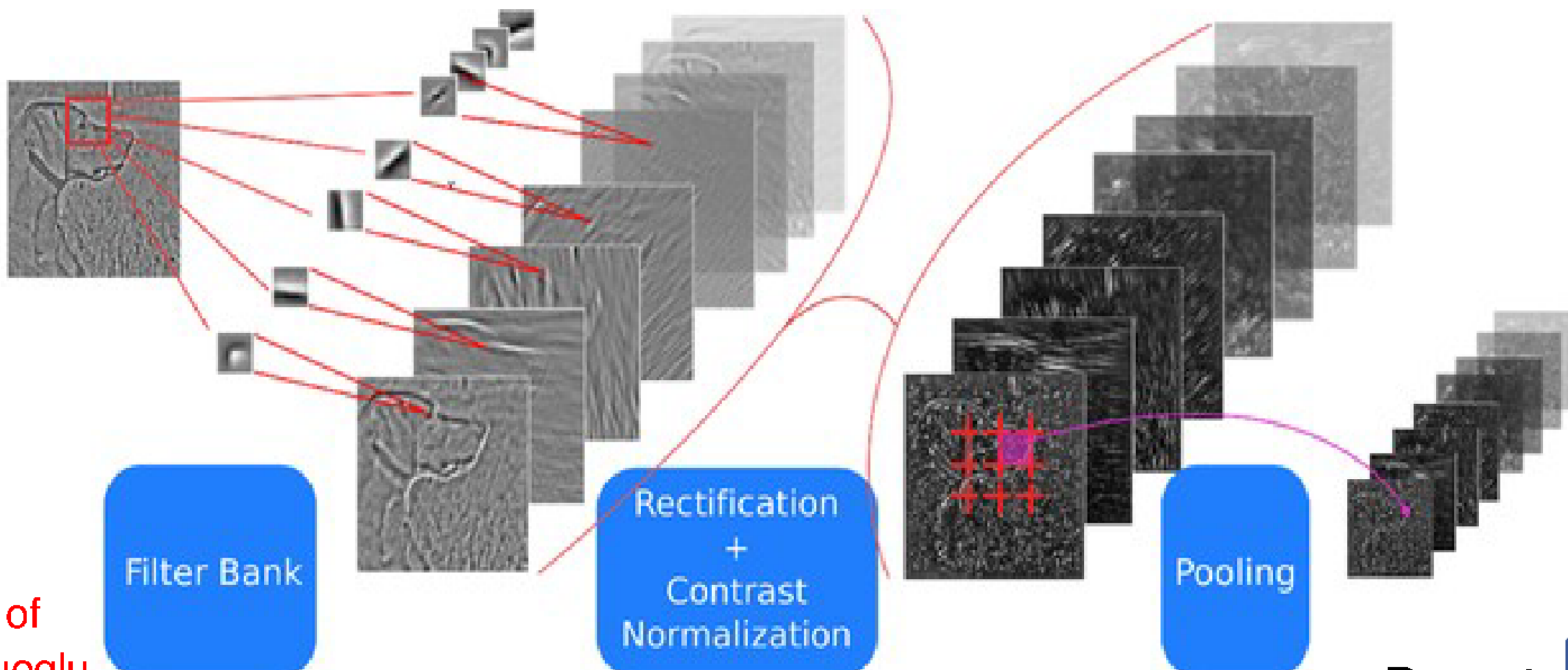
Question: How should I set the size of the pools?

Answer: It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).

ConvNets: Typical Stage

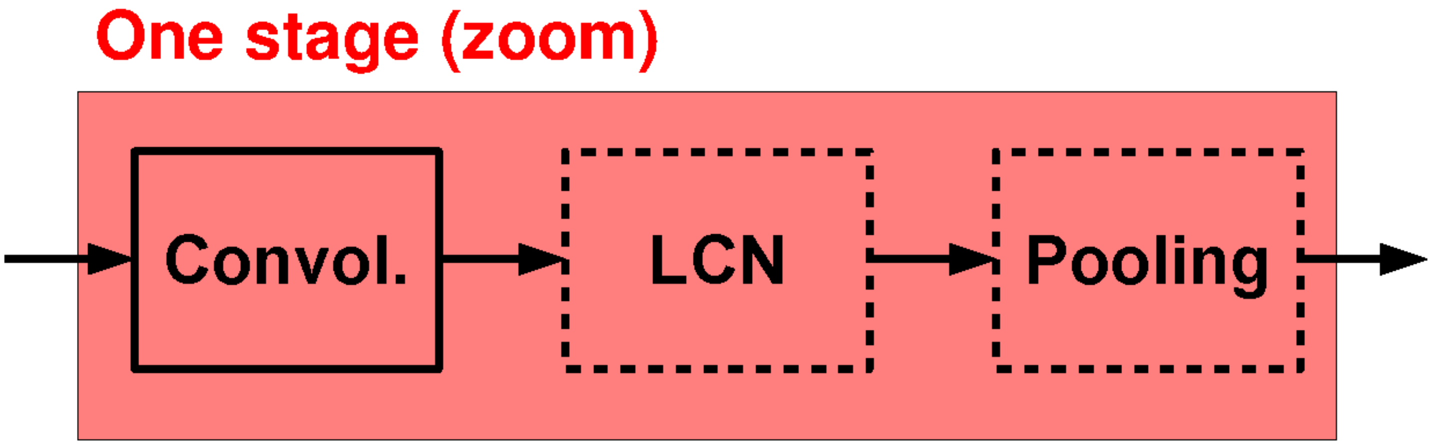
One stage (zoom)





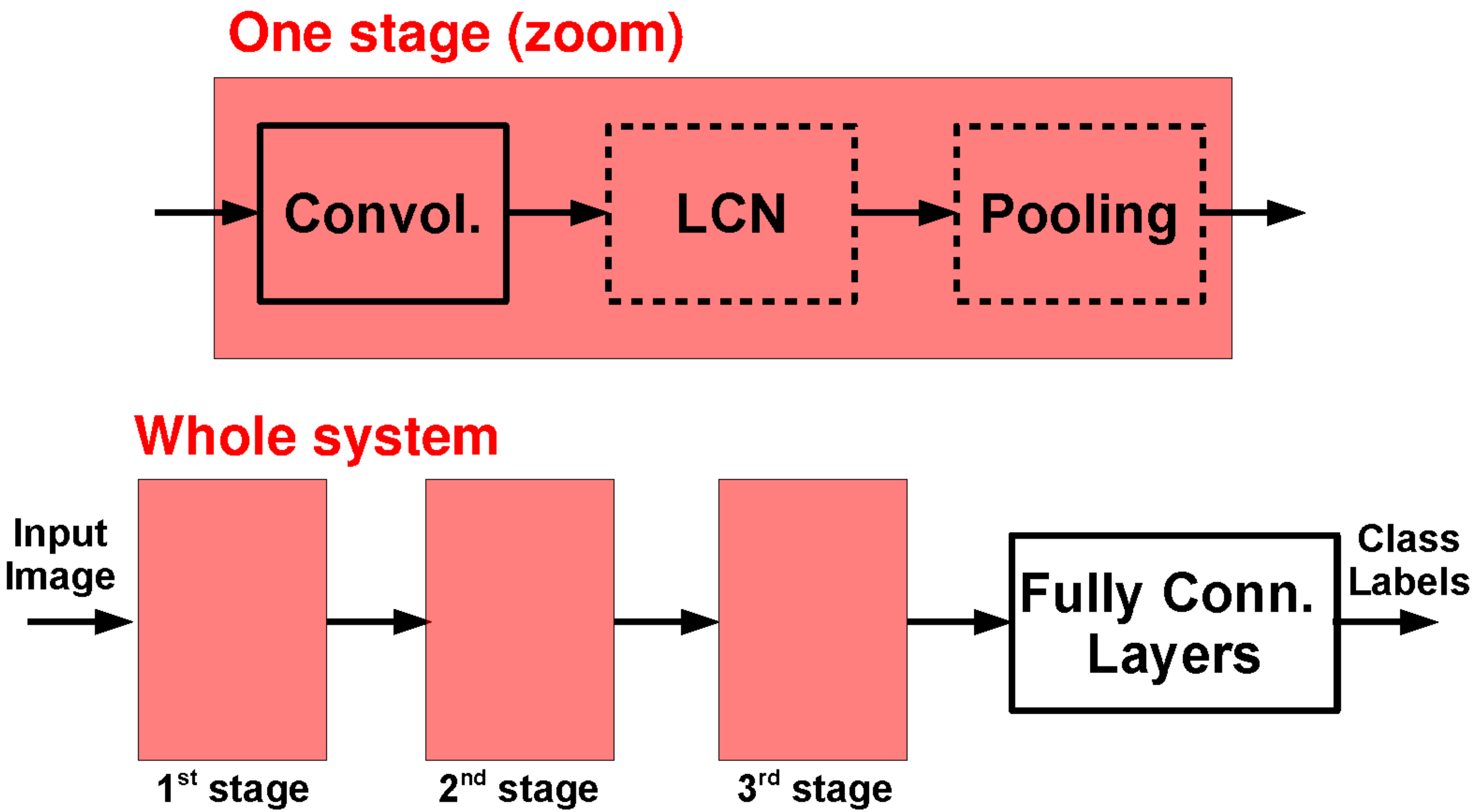
courtesy of
K. Kavukcuoglu

ConvNets: Typical Stage

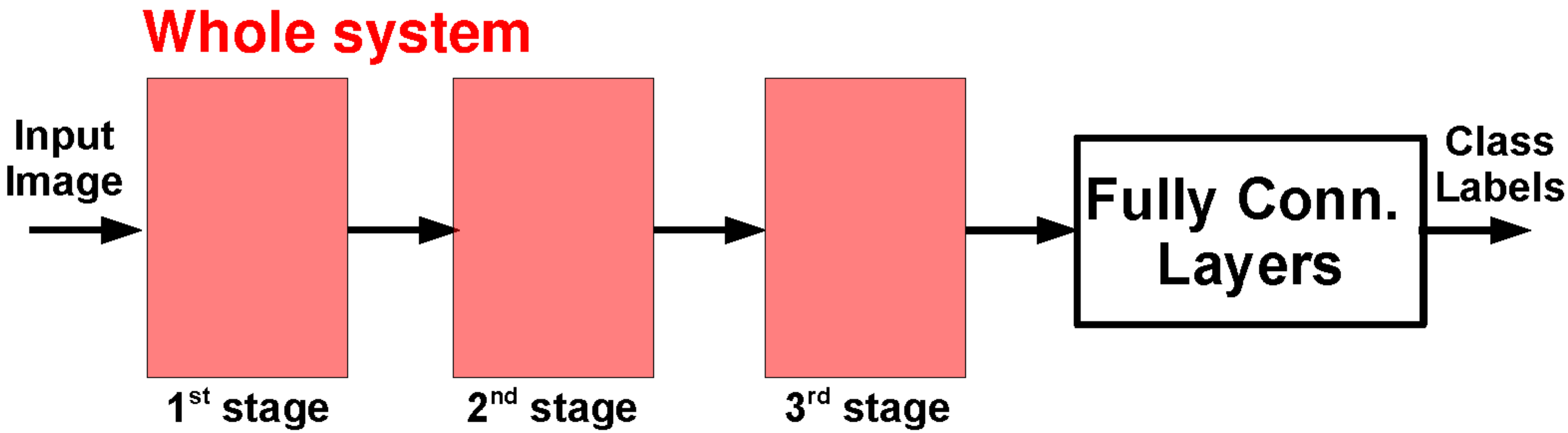


Conceptually similar to: SIFT, HoG, etc.

ConvNets: Typical Architecture



ConvNets: Typical Architecture



Conceptually similar to:

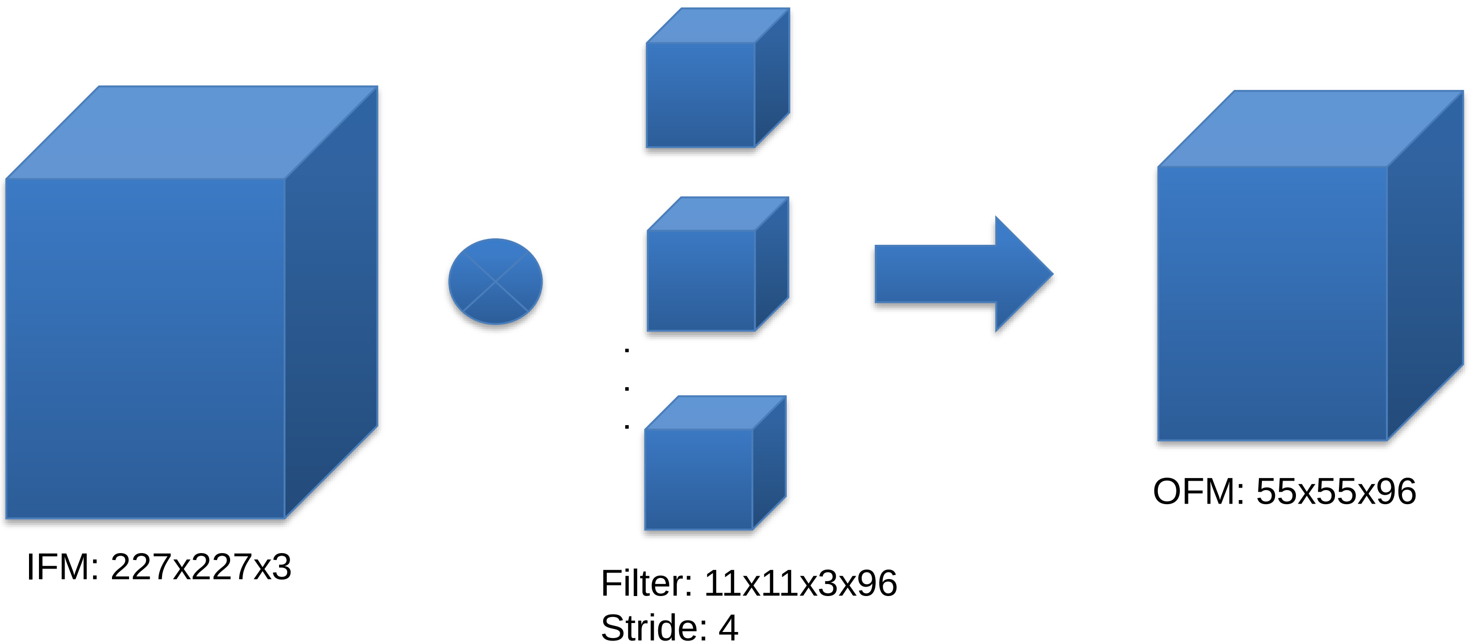
SIFT → K-Means → Pyramid Pooling → SVM

Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

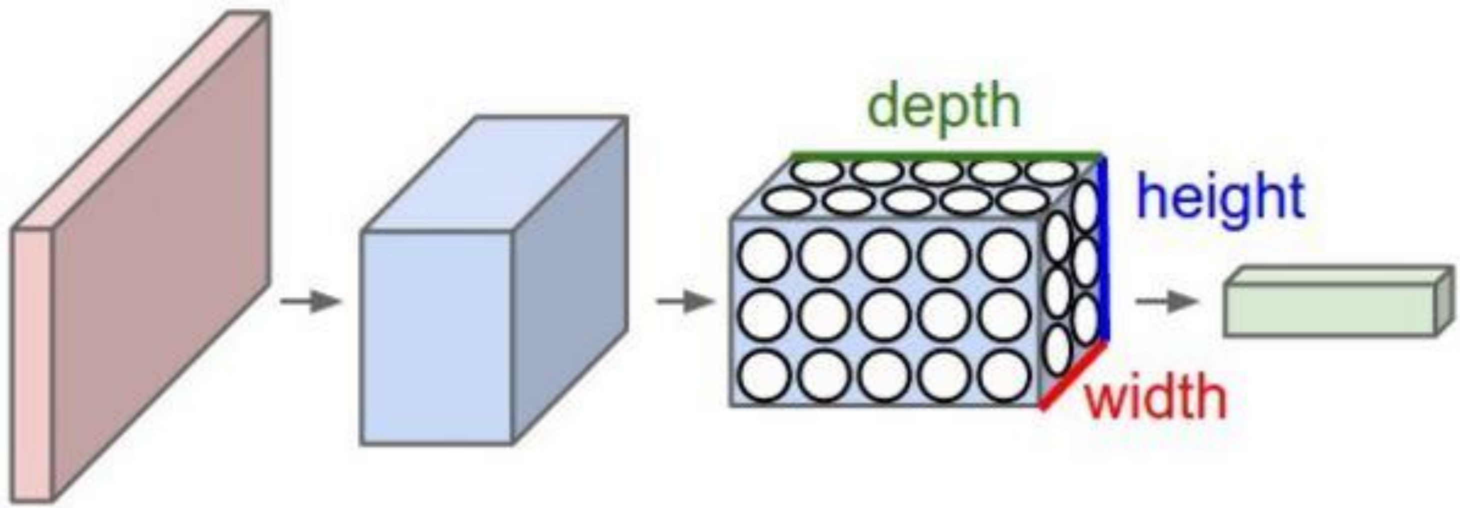
Sanchez et al. "Image classifcation with F.V.: Theory and practice" IJCV 2012

Convolution = Matrix Multiply

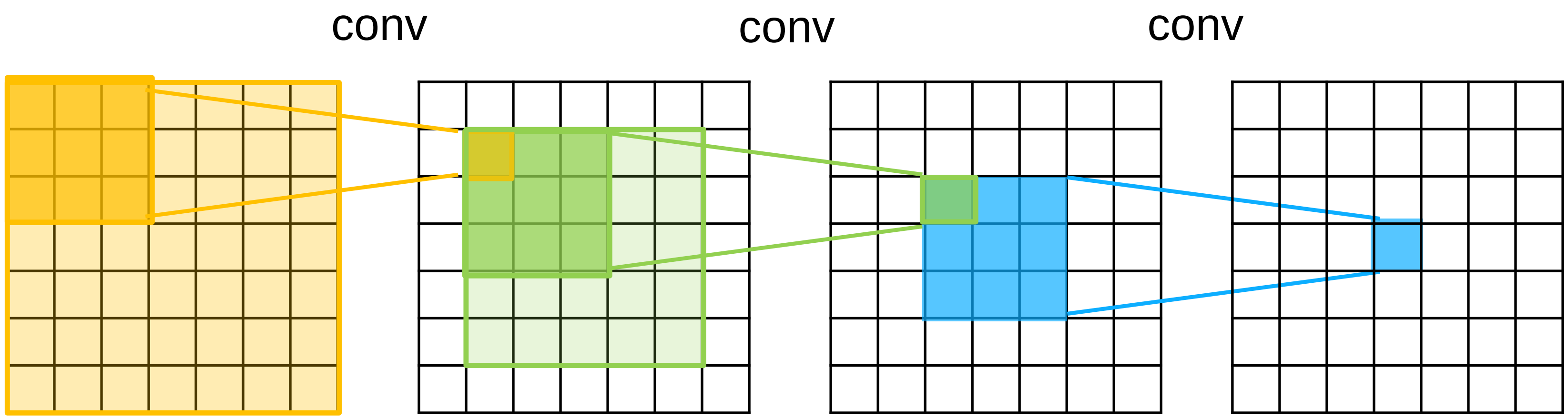


- IFM converted to 363x3025 matrix
  - filter looks at 11x11x3 input volume, 55 locations along W,H.
- Weights converted to 96x363 matrix
- OFM = Weights x IFM.
- BLAS libraries used to implement matrix multiply (GEMM)
  - MKL for CPU, CuBLAS for GPU

Convolutional Neural Network

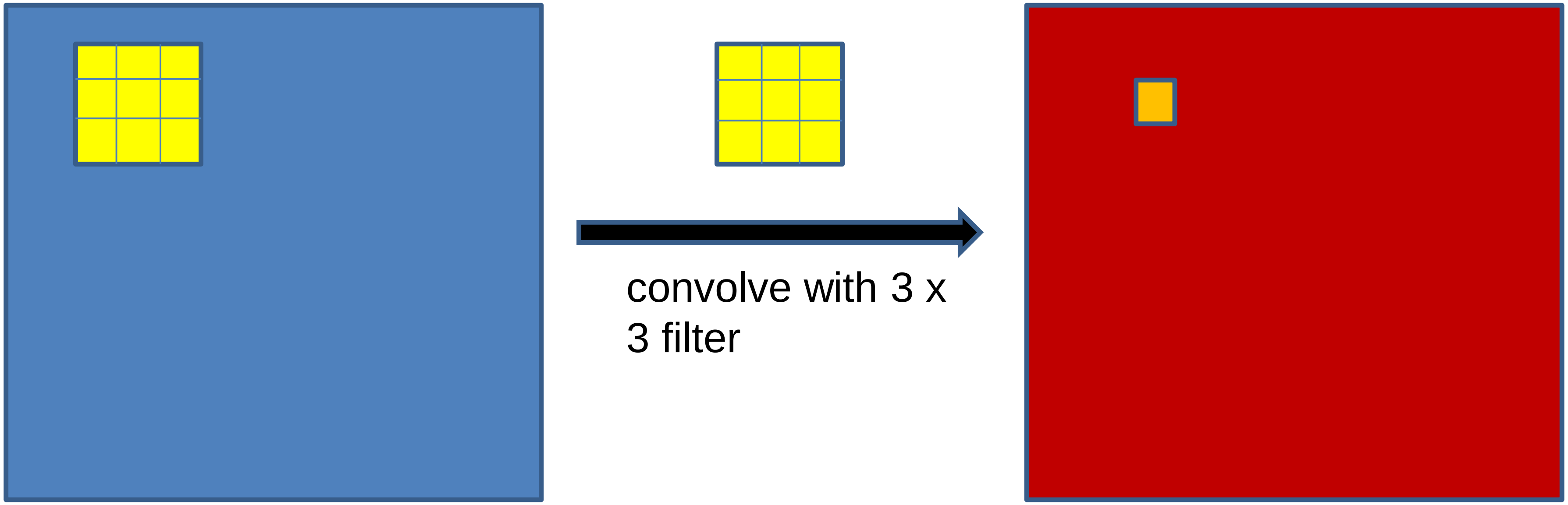


Reminder: Receptive Field

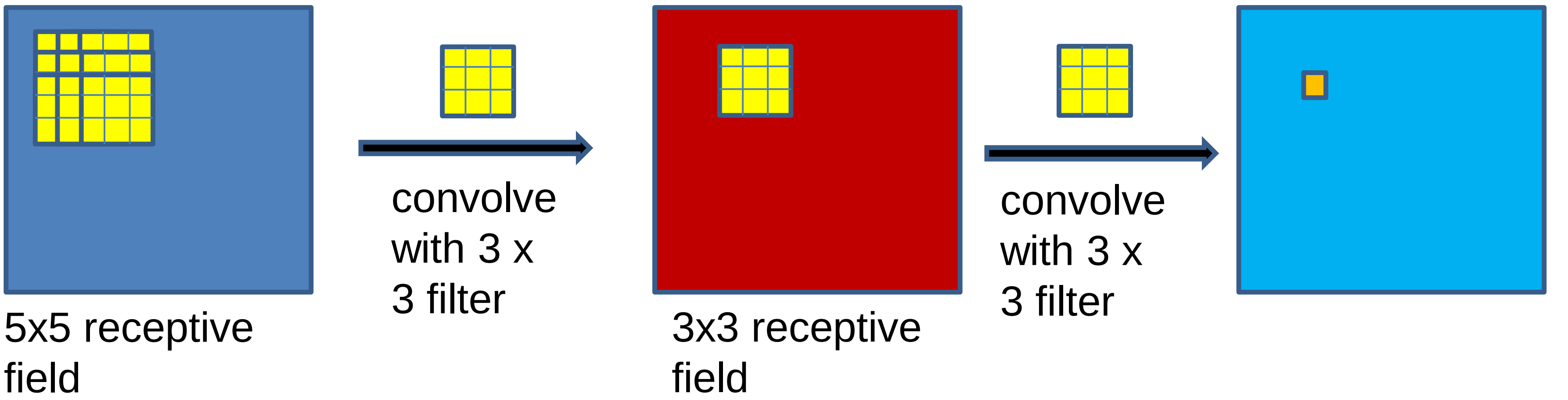


Receptive field

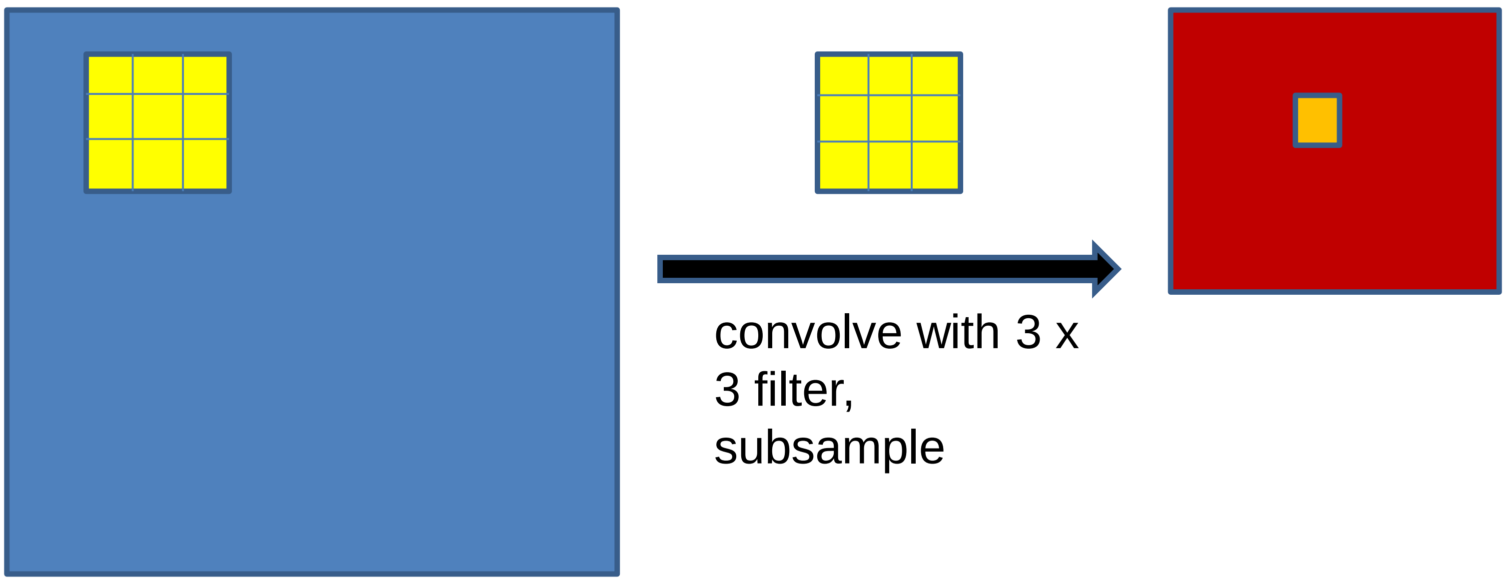
- Which input pixels does a particular unit in a feature map depends on



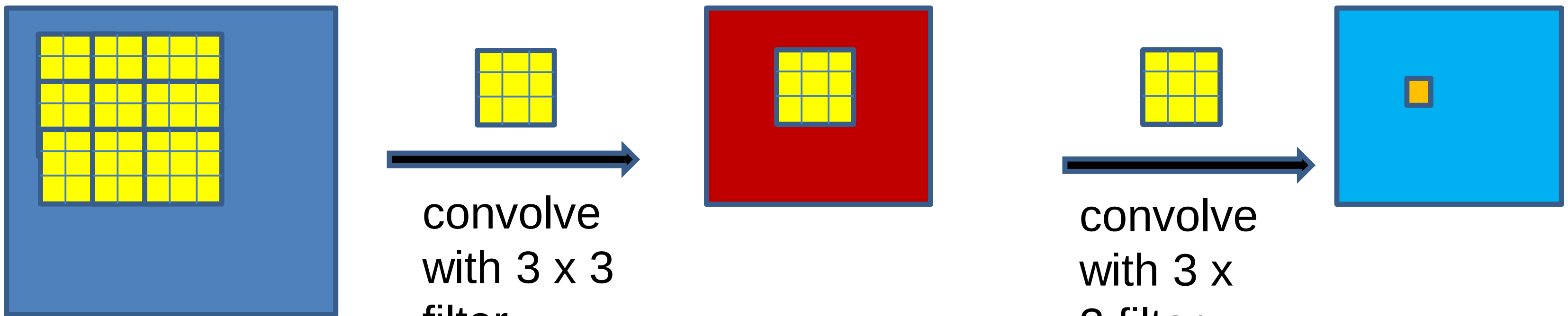
Receptive field



Receptive field



Receptive field

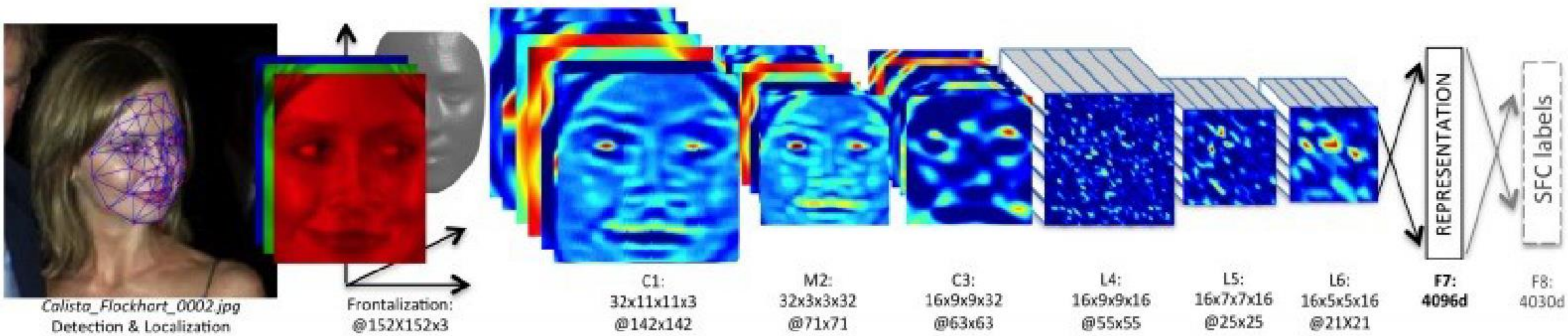


7x7 receptive field: union of 9 3x3 fields with stride of 2

3x3 receptive field

CONV NETS: EXAMPLES

- **Face Verification & Identification**



Taigman et al. “DeepFace...” CVPR 2014

Ranzato

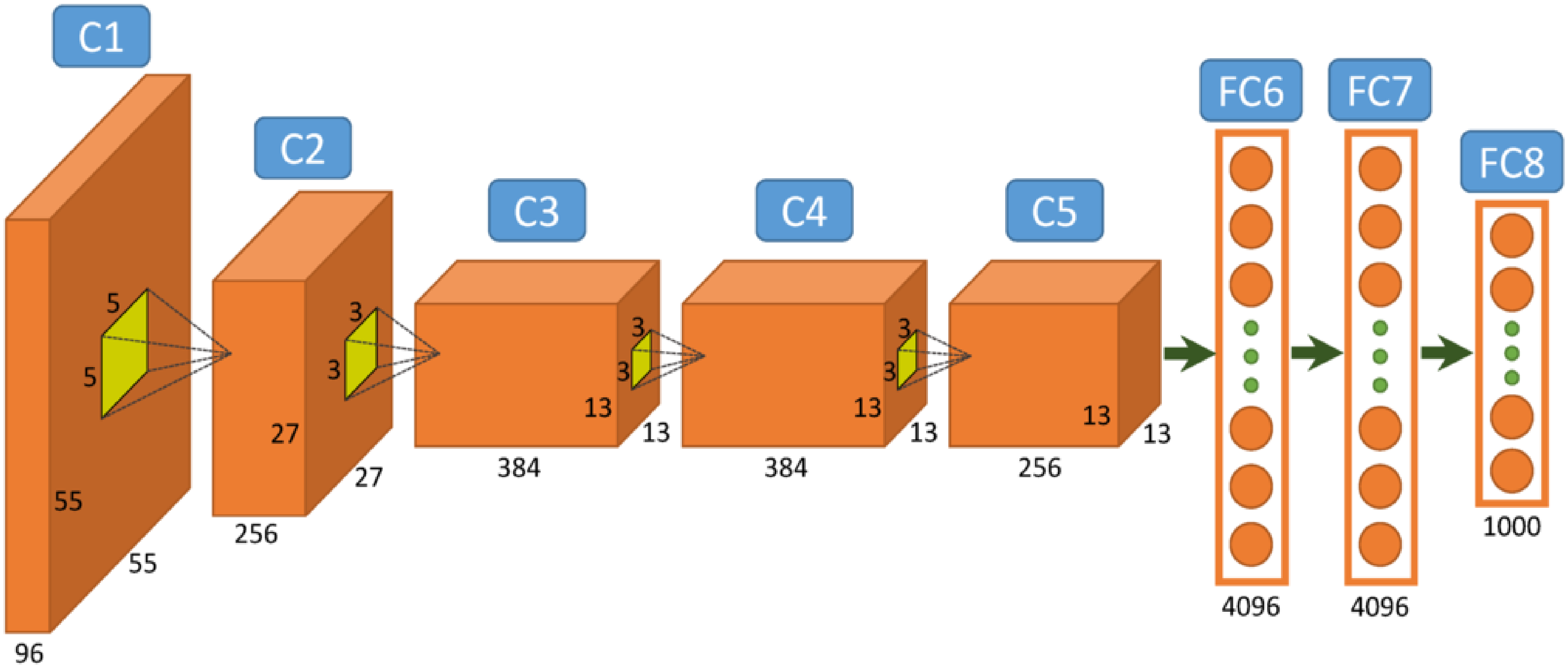
Well-known Deep Networks

1. AlexNet
2. VGG
3. GoogleNet
4. ResNet

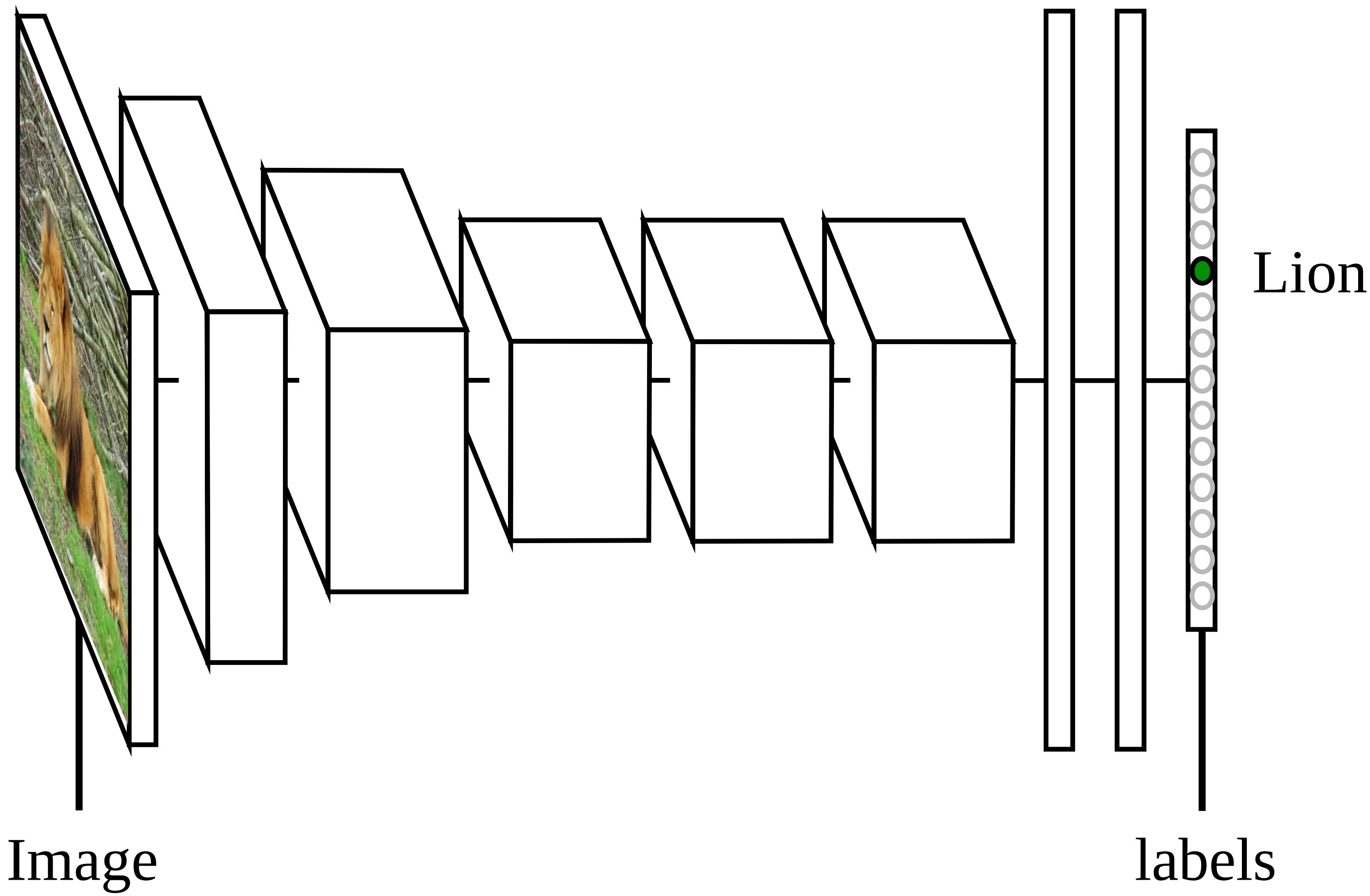
AlexNet

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Network Structure for AlexNet

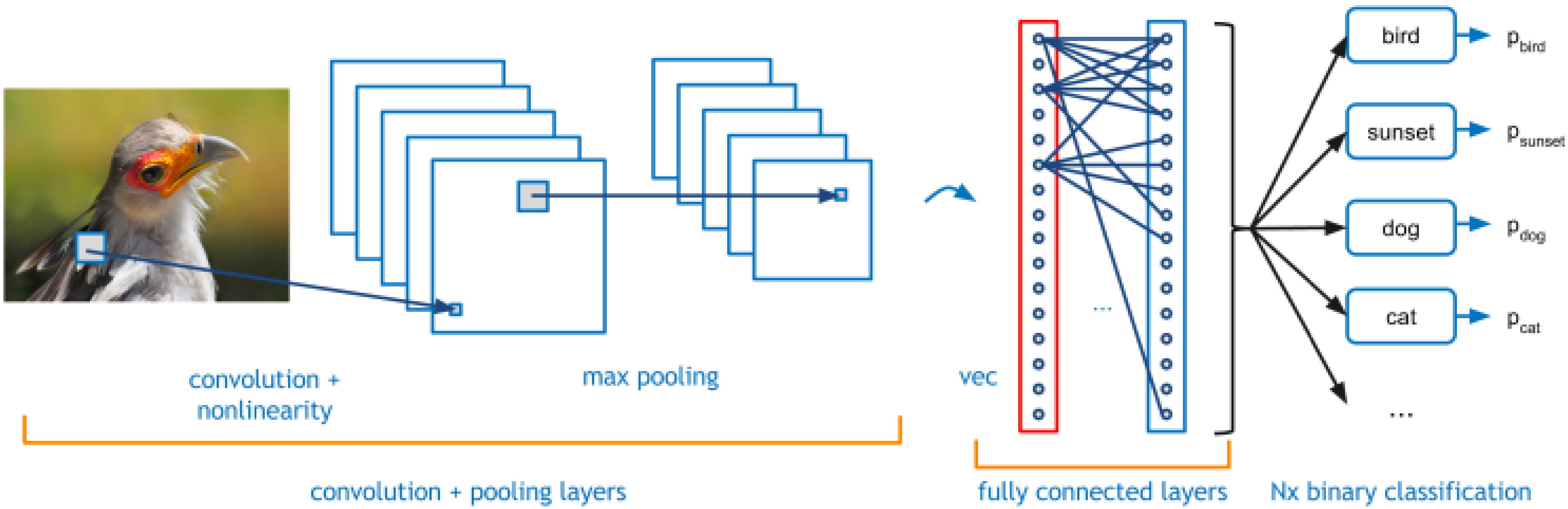


Convolutional Neural Networks: **AlexNet**

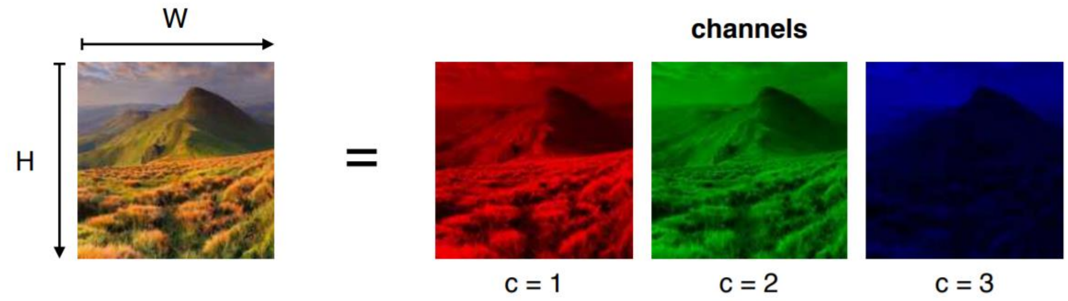


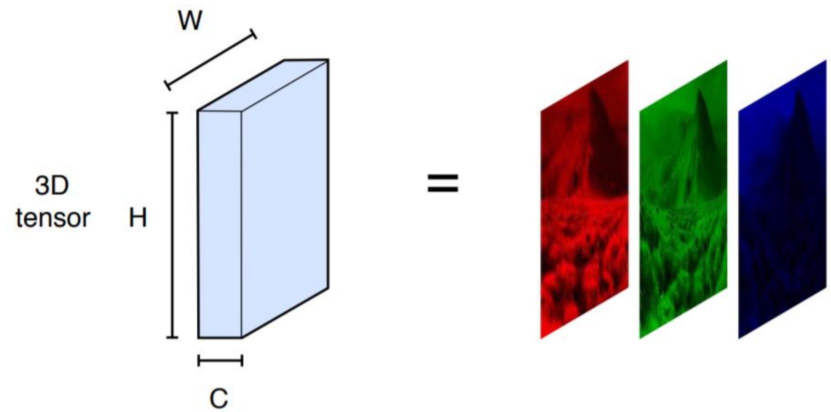
Krizhevsky, Sutskever, Hinton — NIPS 2012

Convolutional Neural Networks: **AlexNet**

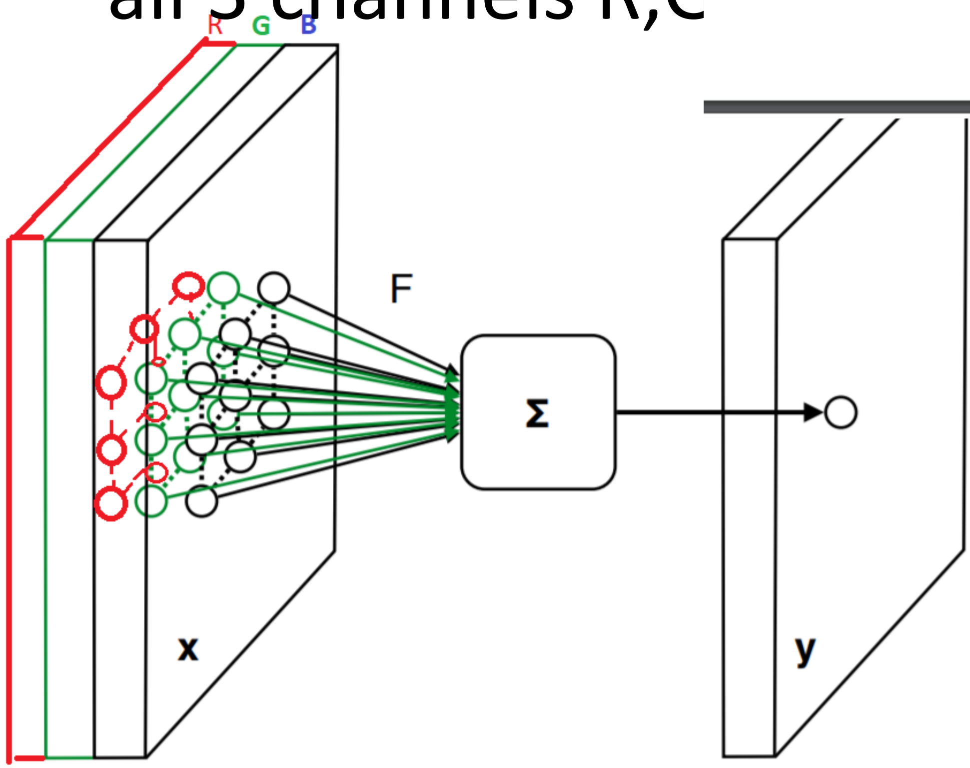


First layer



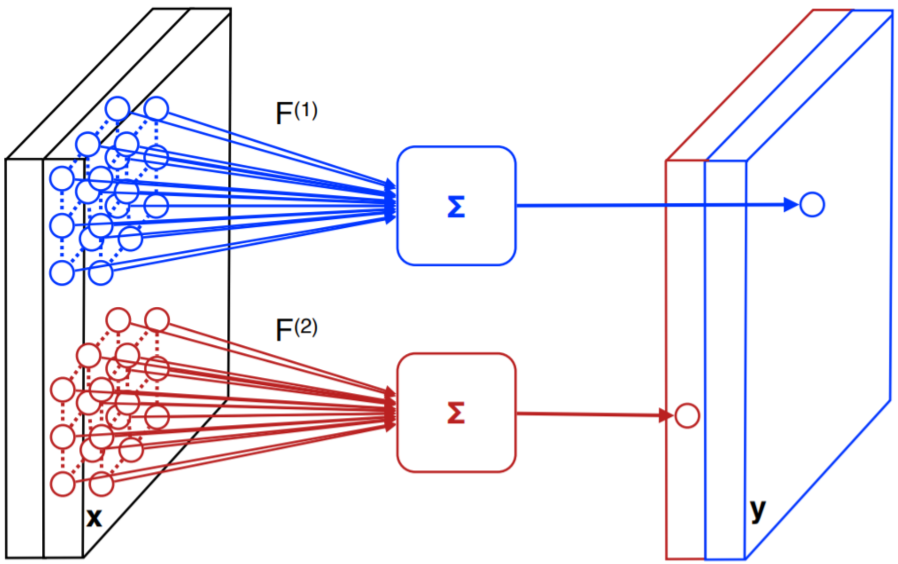


- Each filter works all 3 channels R,G



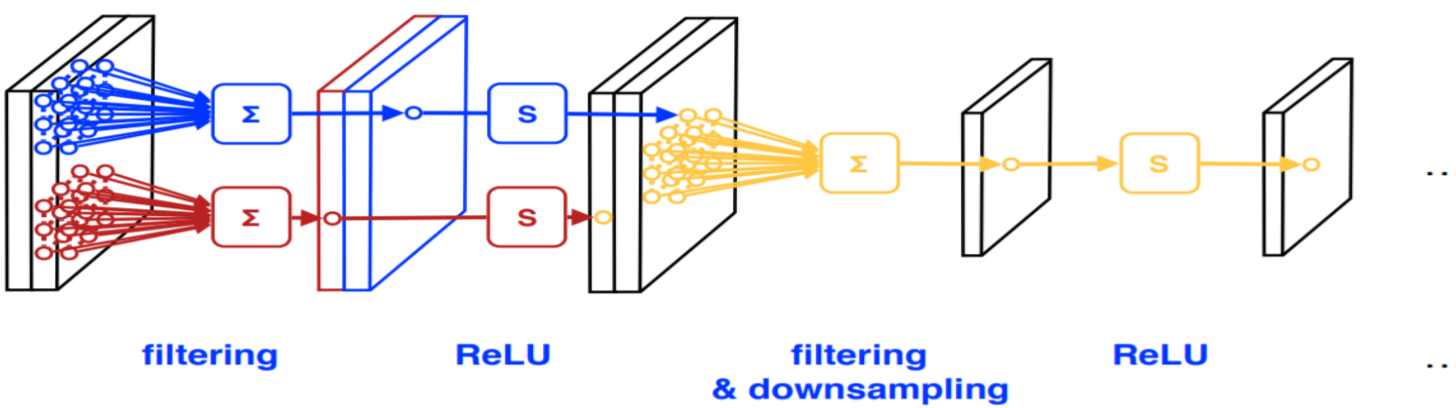
Output of Convolution Layer

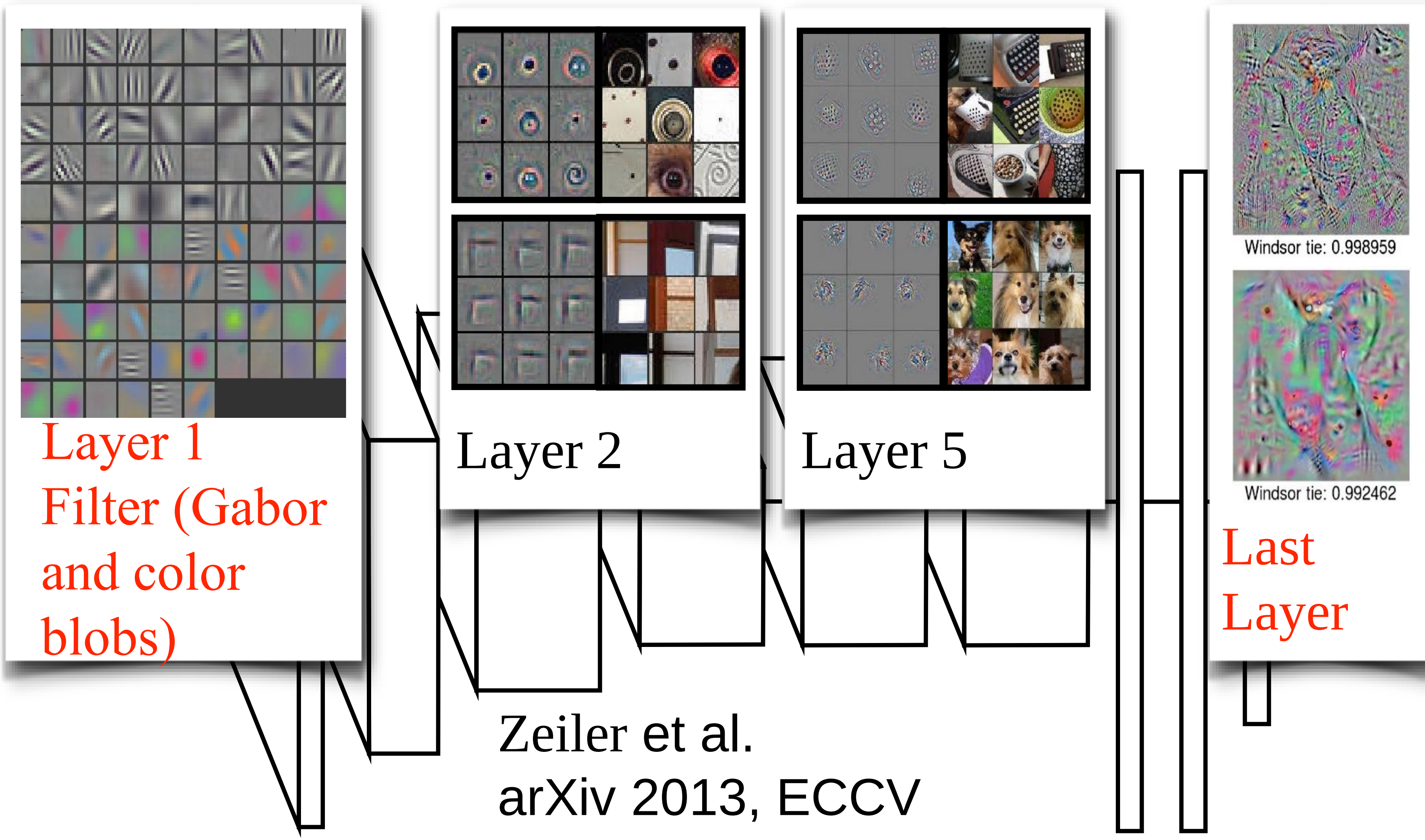
- If input =MxM and have K filters that are 3X3
  - OUTPUT = K channels of (M-2)x(M-2)



Example:
2 filters

→ 2 output channels



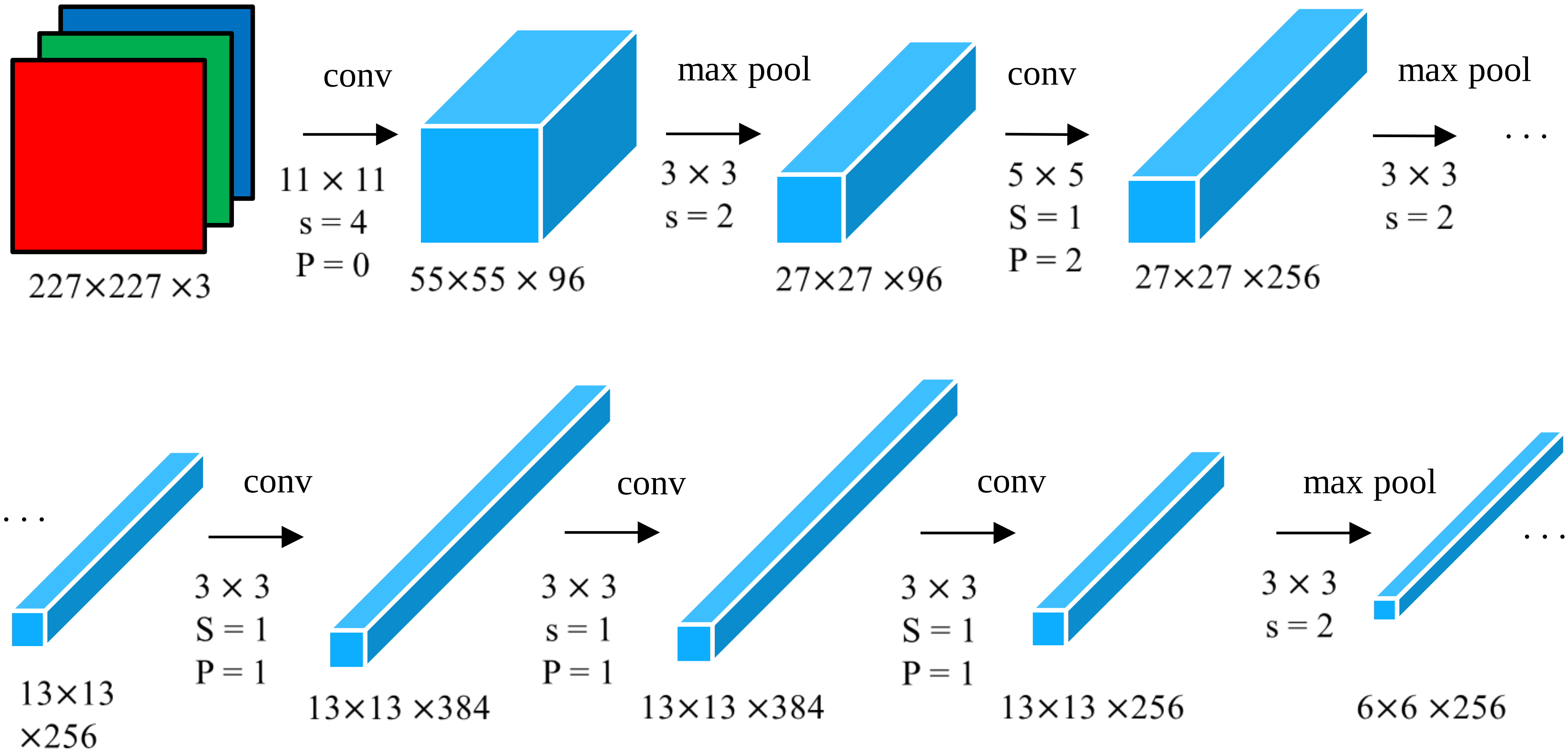


Zeiler et al.
arXiv 2013, ECCV 2014

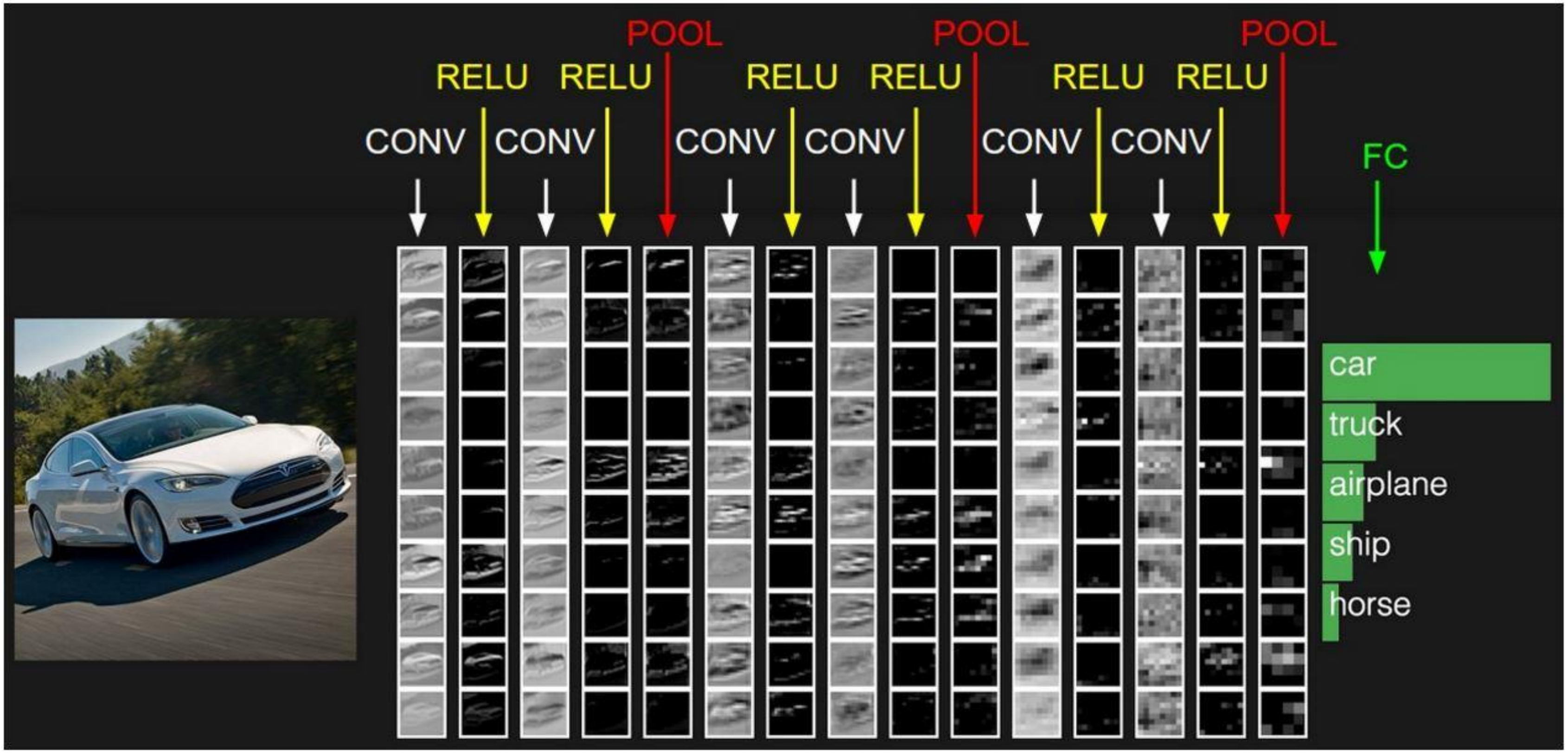
Nguyen et al.
arXiv 2014

Gabor filter: linear filters used for edge detection with similar orientation representations to the human visual system

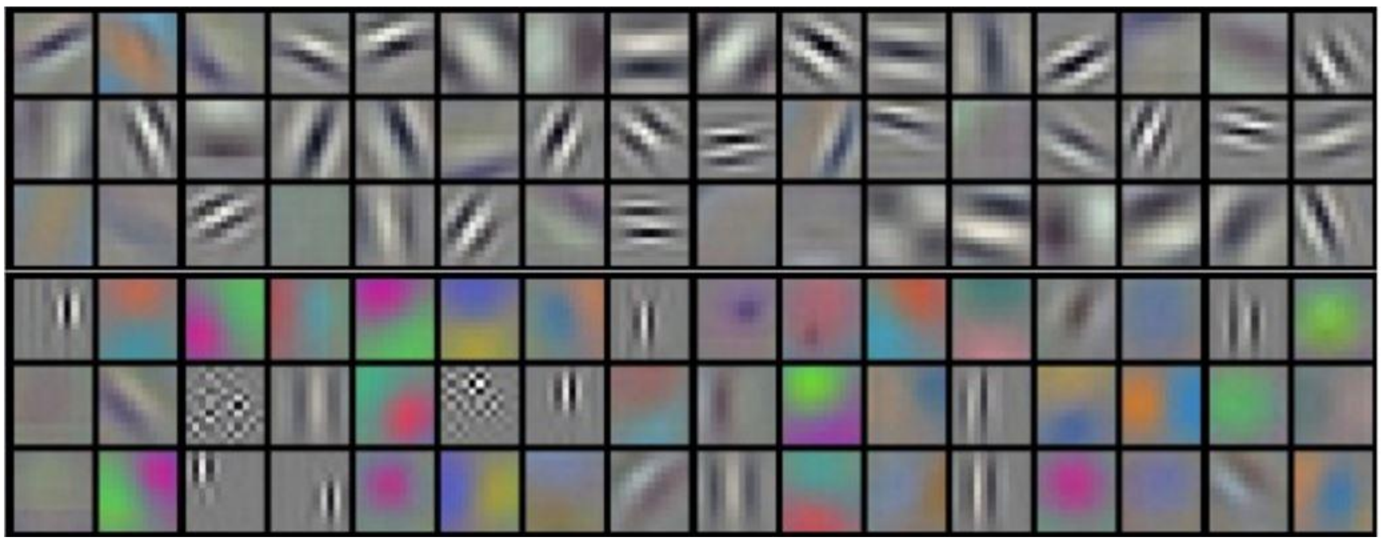
AlexNet


227×227 ×3
conv
11 × 11
s = 4
P = 0
55×55 × 96
max pool
3 × 3
s = 2
27×27 ×96
conv
5 × 5
S = 1
P = 2
27×27 ×256
max pool
3 × 3
s = 2
13×13
×256
conv
3 × 3
S = 1
P = 1
13×13 ×384
conv
3 × 3
s = 1
P = 1
13×13 ×384
conv
3 × 3
S = 1
P = 1
13×13 ×256
max pool
3 × 3
s = 2
6×6 ×256

Example of CNN layer



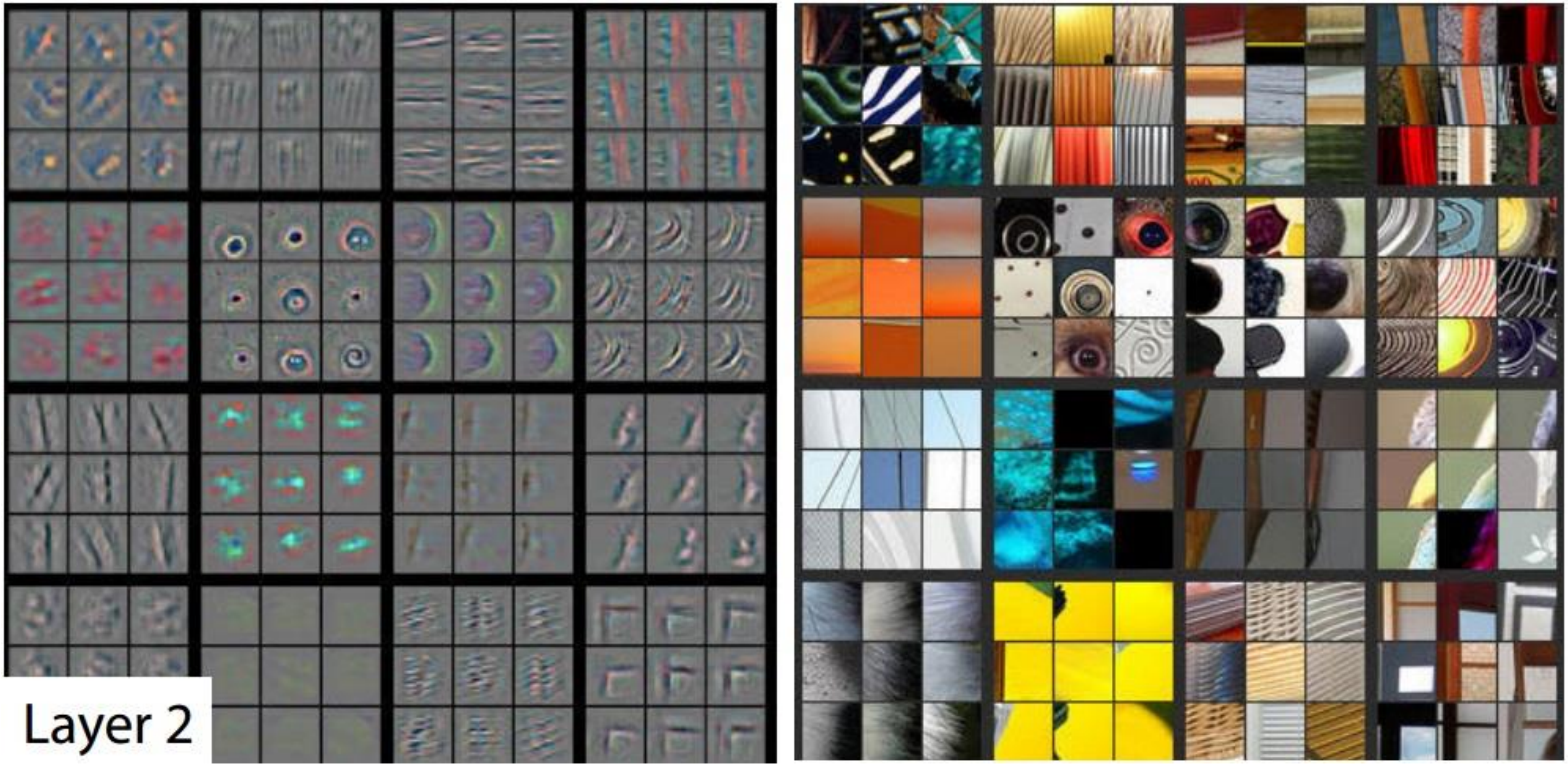
96 filters of 11x11x3 each



Layer 1

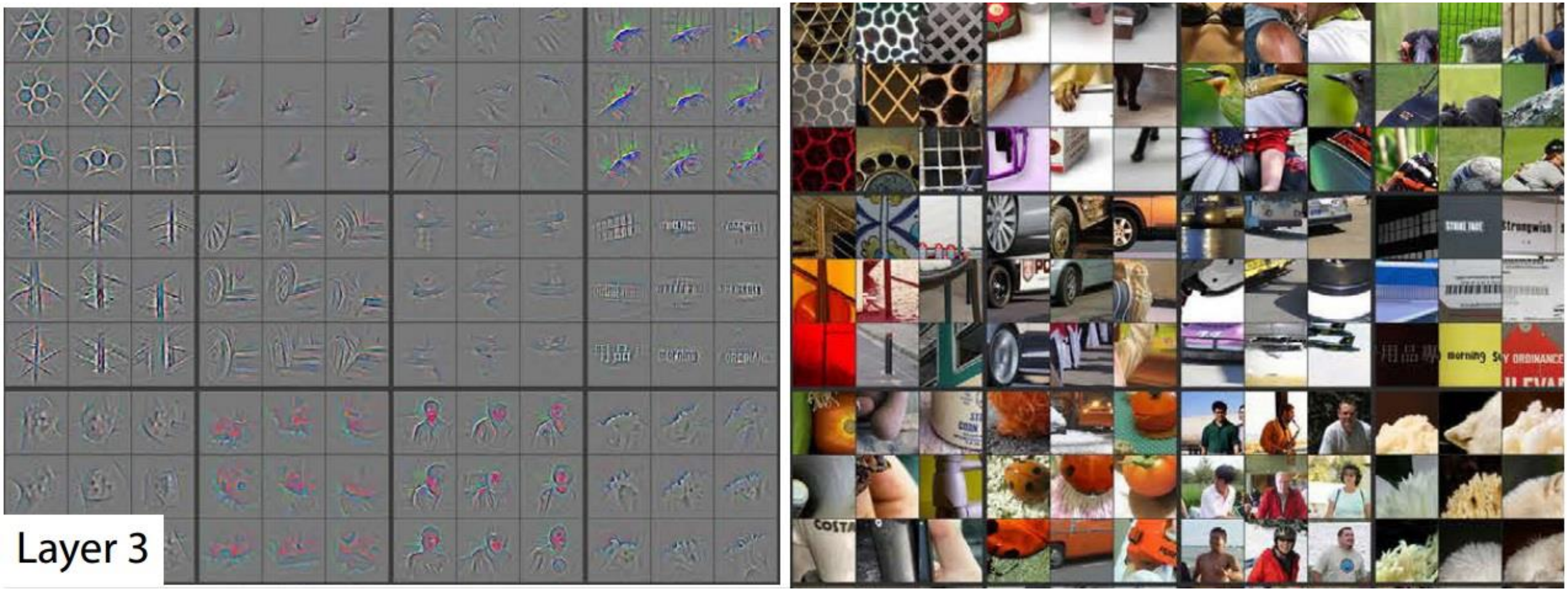
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 2



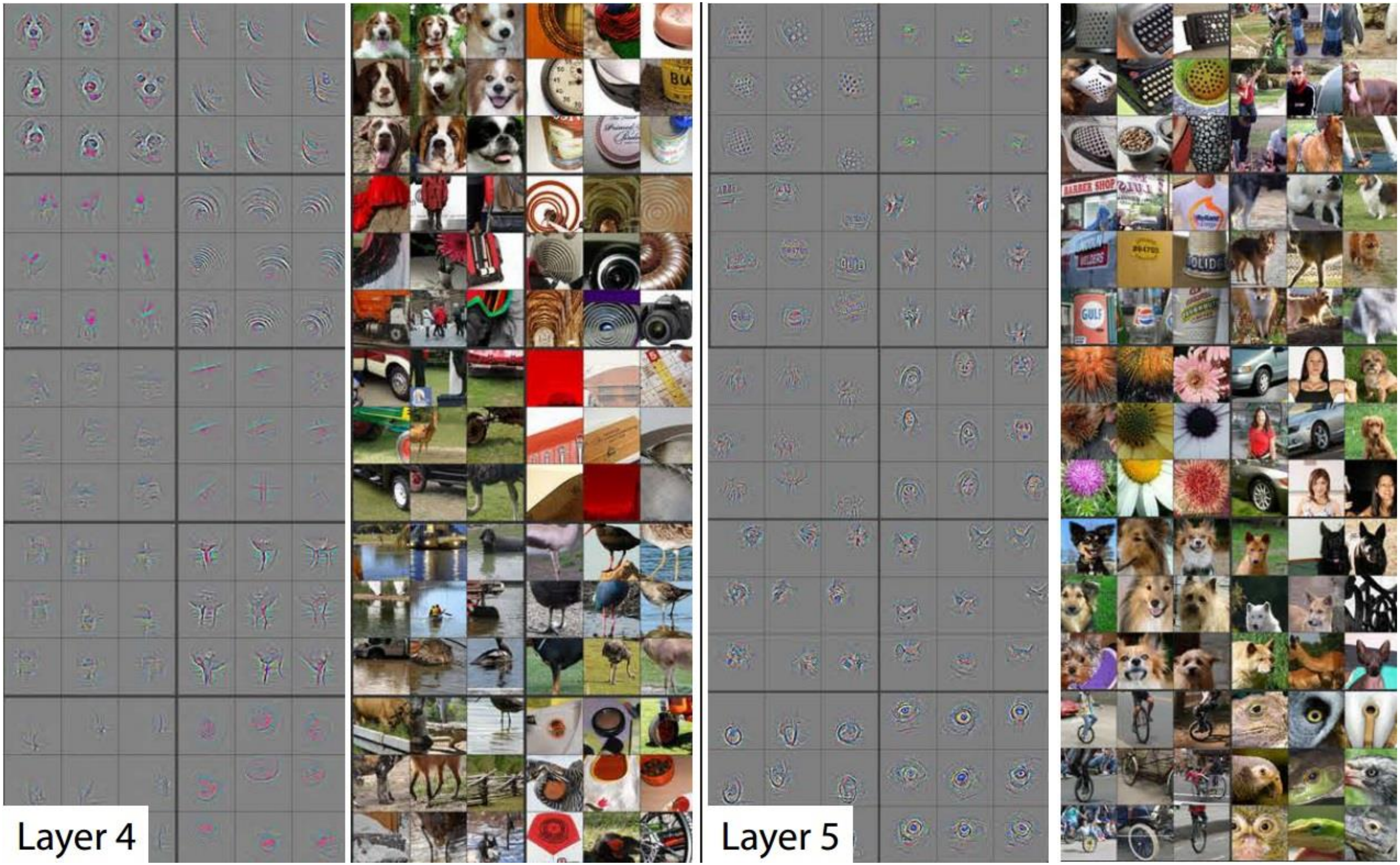
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 3



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

Layer 4 and 5



Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]

VGG

Karen Simonyan, Andrew Zisserman:**Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR, 2015**

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2

- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers

- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

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

  7x7

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Input
3x3 conv, 64
3x3 conv, 64
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

VGG16:

TOTAL memory: 24M * 4 bytes ~= 96MB / image

TOTAL params: 138M parameters

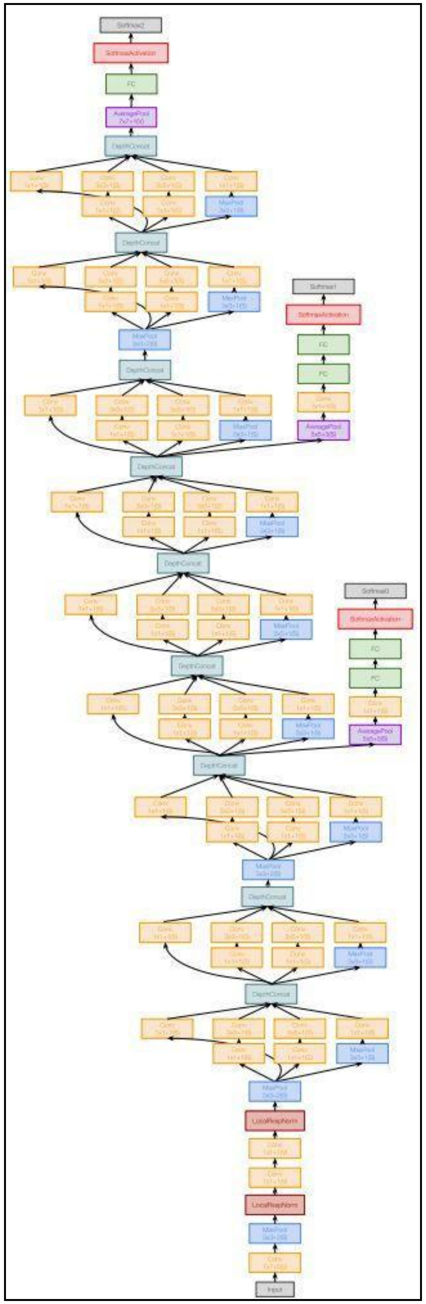
Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC 4096	memory: 4096	params: 4096*4096 = 16,777,216
FC 1000	memory: 1000	params: 4096*1000 = 4,096,000

GoogleNet

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich: **Going Deeper with Convolutions, IEEE CVPR, 2015.**

GoogleNet

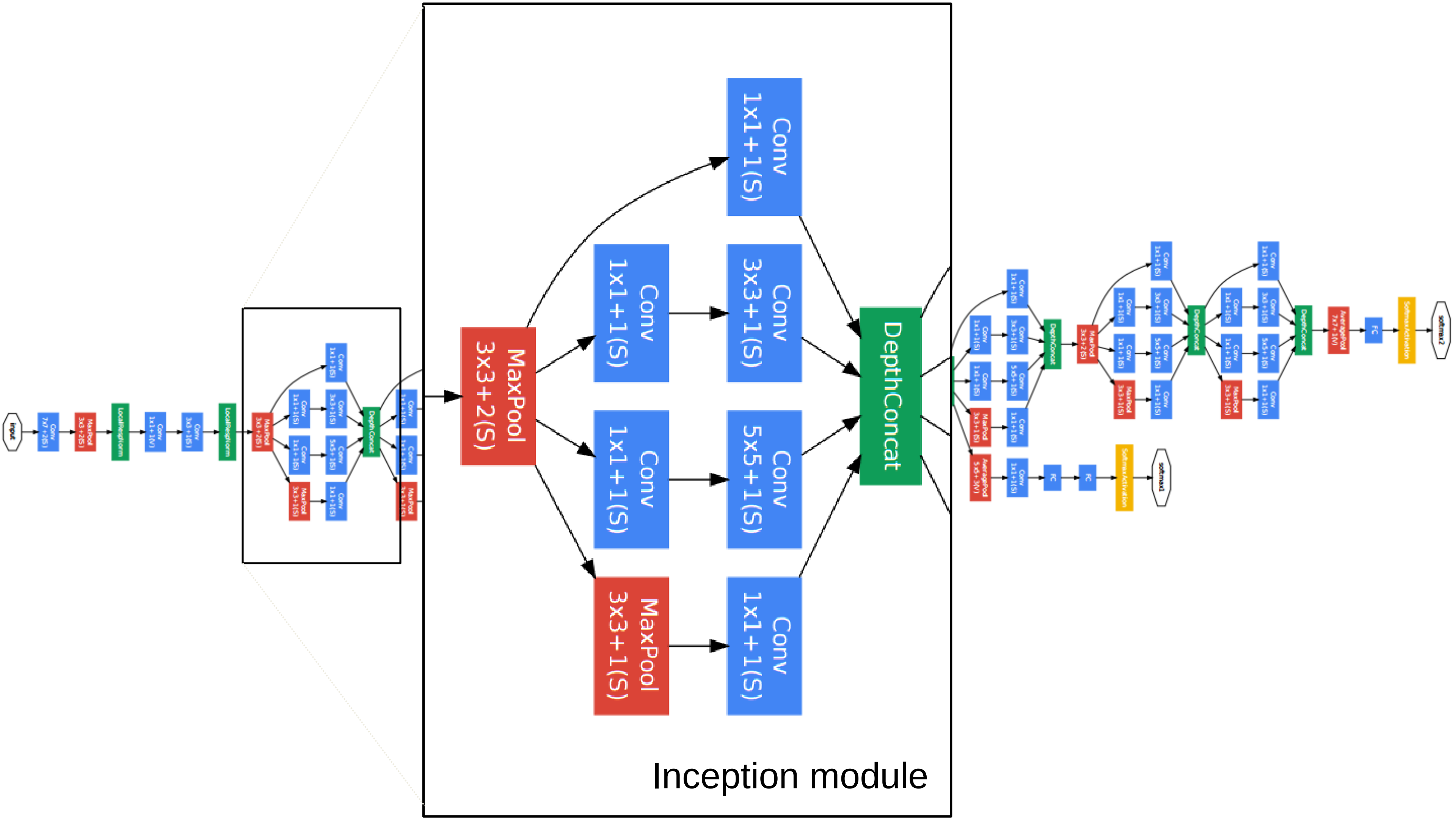
- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency



GoogleNet

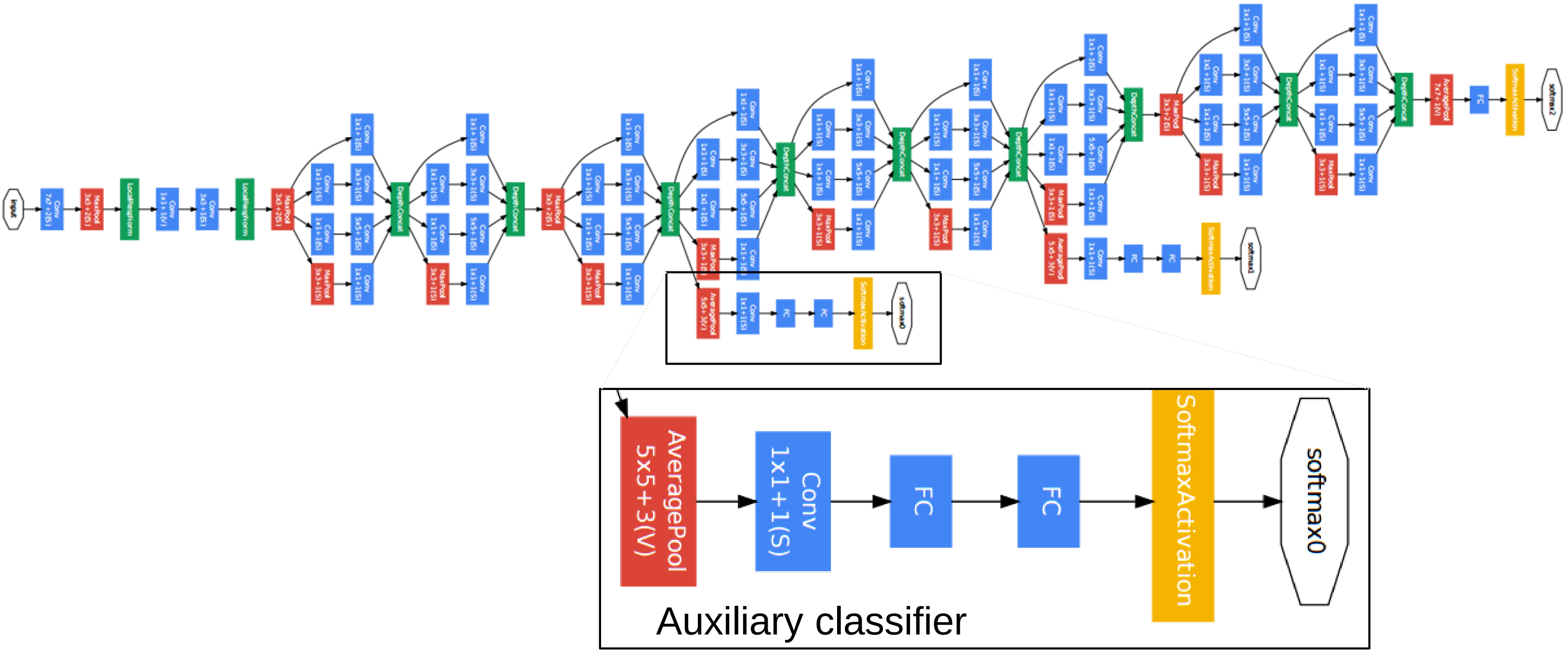
- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

GoogLeNet



C. Szegedy et al., [Going deeper with convolutions](), CVPR 2015

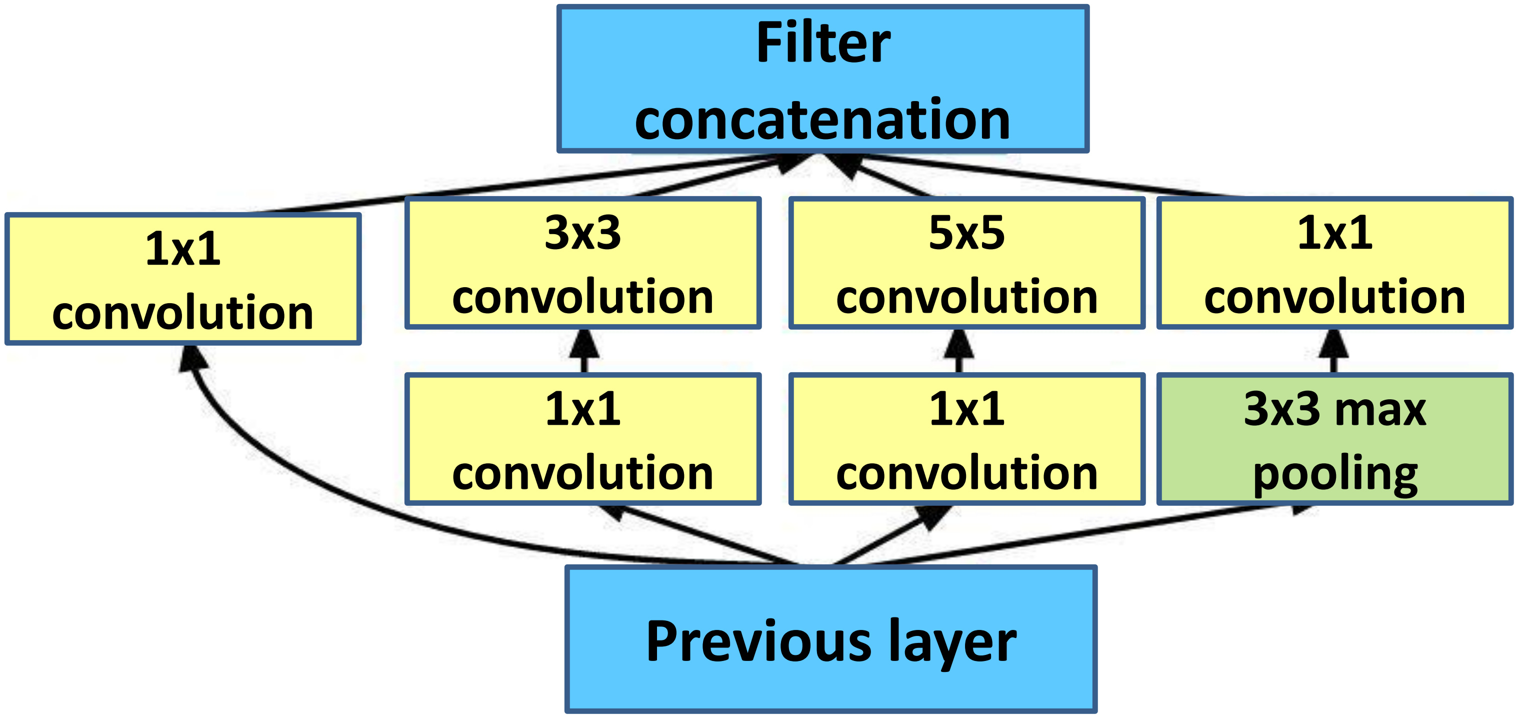
GoogLeNet



C. Szegedy et al., [Going deeper with convolutions](), CVPR 2015

GoogleNet

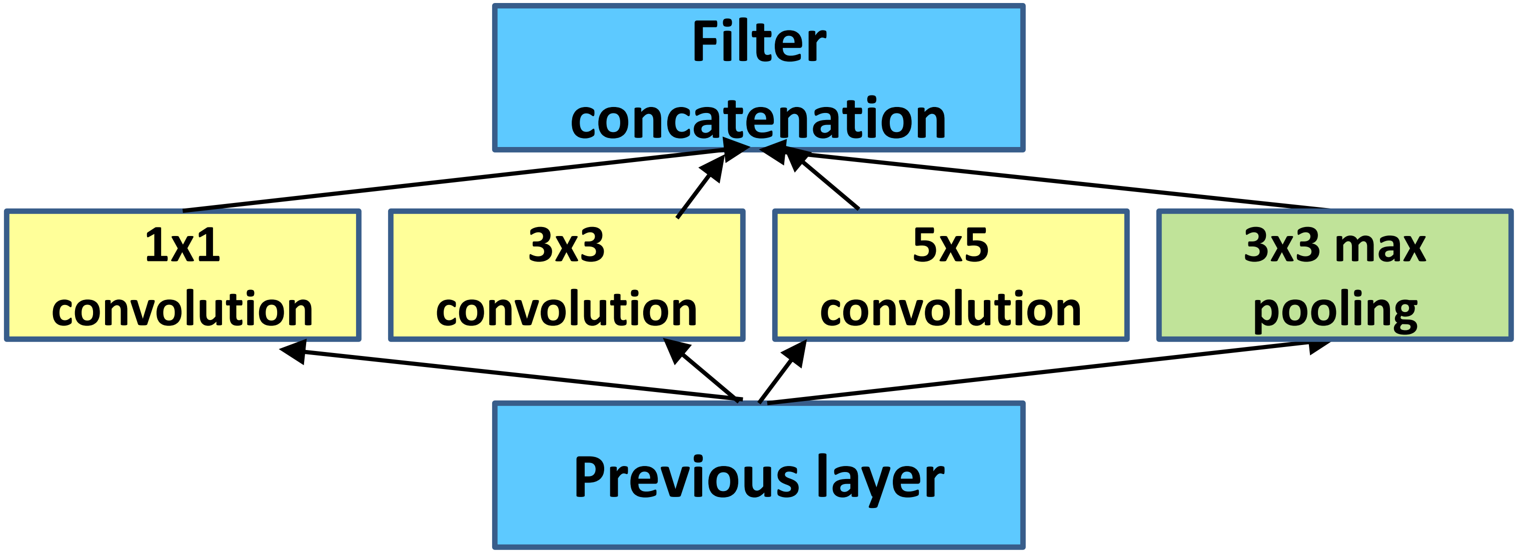
“**Inception module”:** design a good local network topology (network within a network) and then stack these modules on top of each other



GoogleNet

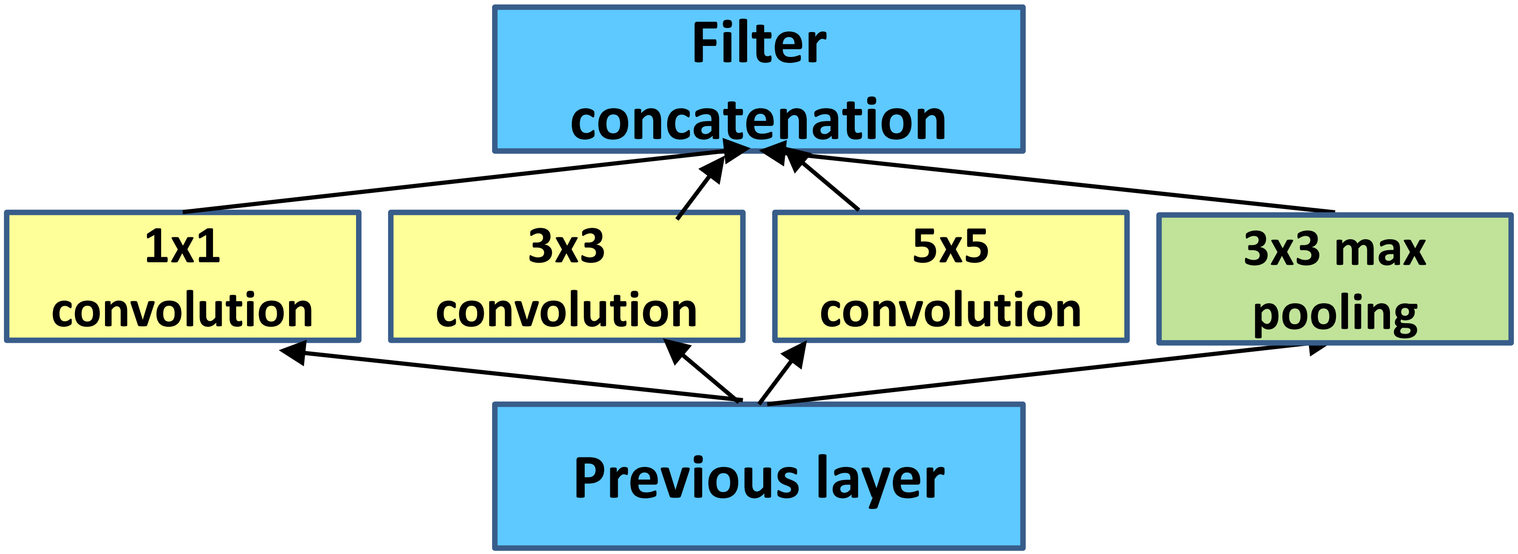
Naïve Inception Model

- Apply parallel filter operations on the input :
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



GoogleNet

- **What's the problem with this?**
  High computational complexity



GoogleNet

- **Output volume sizes:**

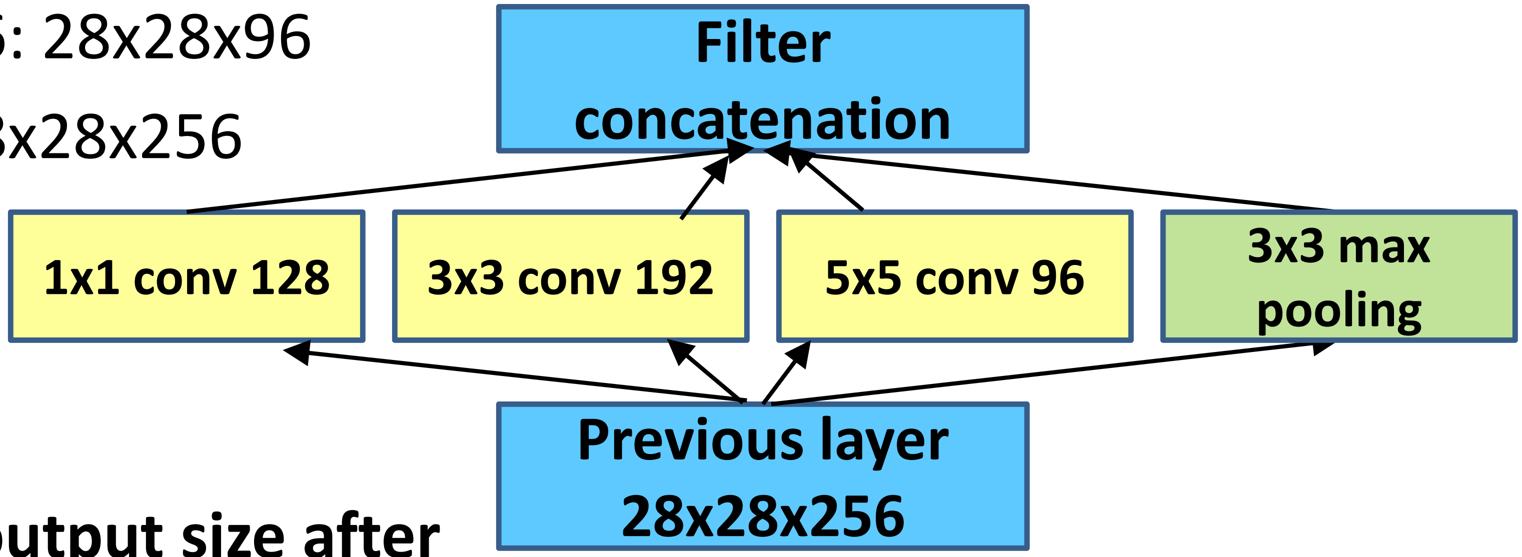
1x1 conv, 128: 28x28x128

3x3 conv, 192: 28x28x192

5x5 conv, 96: 28x28x96

3x3 pool: 28x28x256

Example:



- **What is output size after filter concatenation?**

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

GoogleNet

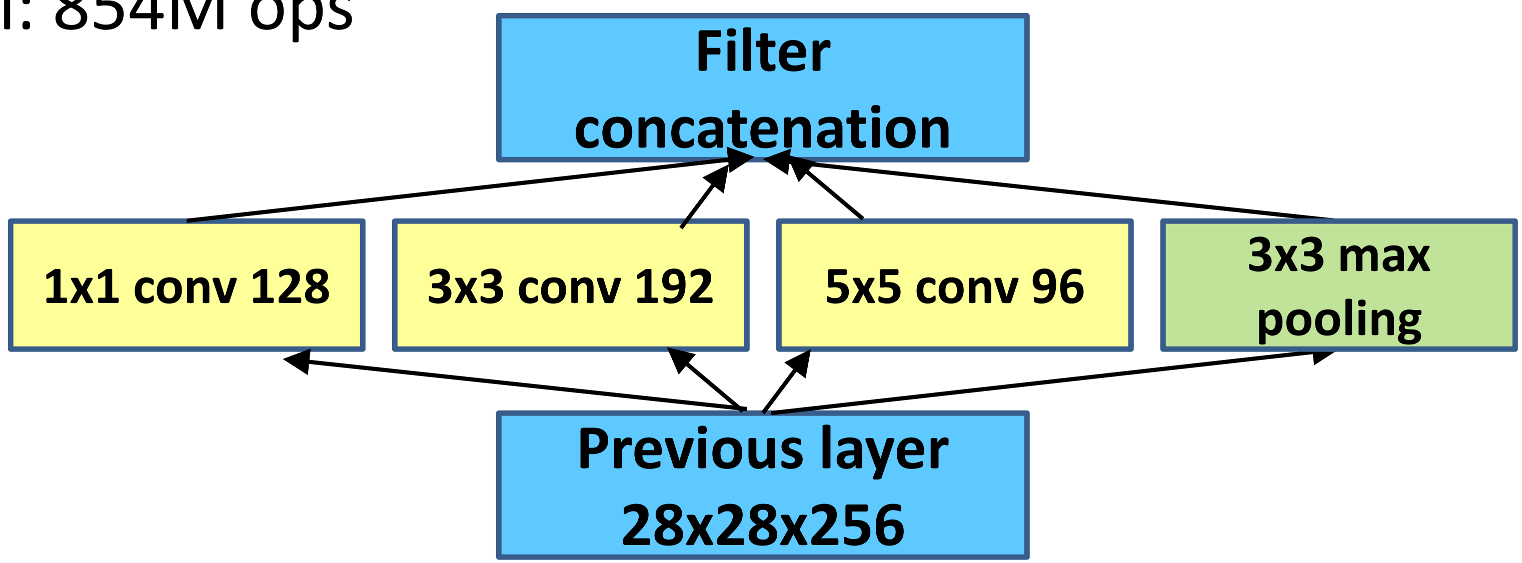
- **Number of convolution operations:**

1x1 conv, 128: 28x28x128x1x1x256

3x3 conv, 192: 28x28x192x3x3x256

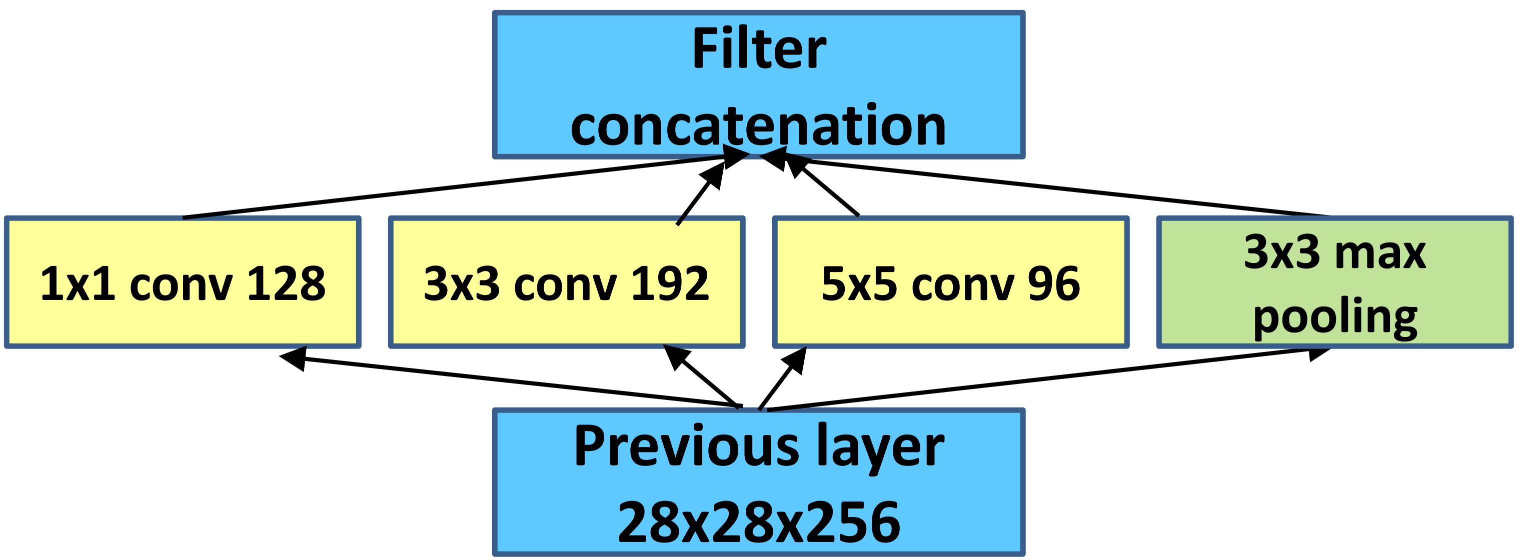
5x5 conv, 96: 28x28x96x5x5x256

Total: 854M ops



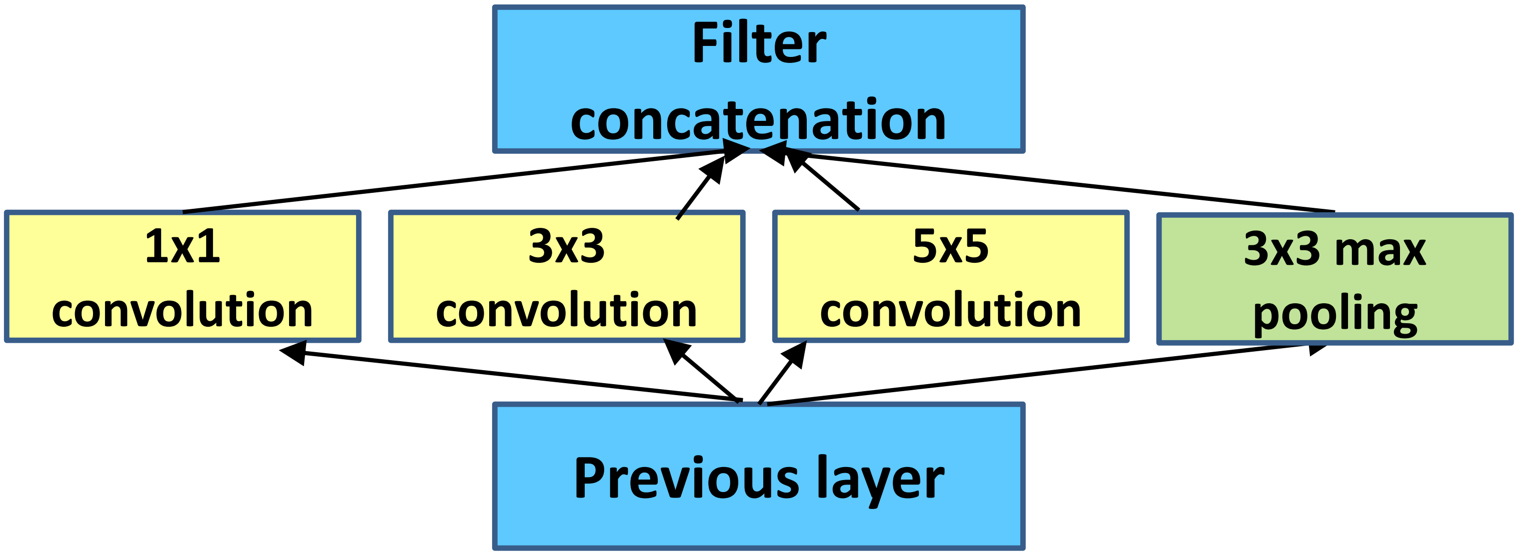
GoogleNet

- **Very expensive compute!**
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer.



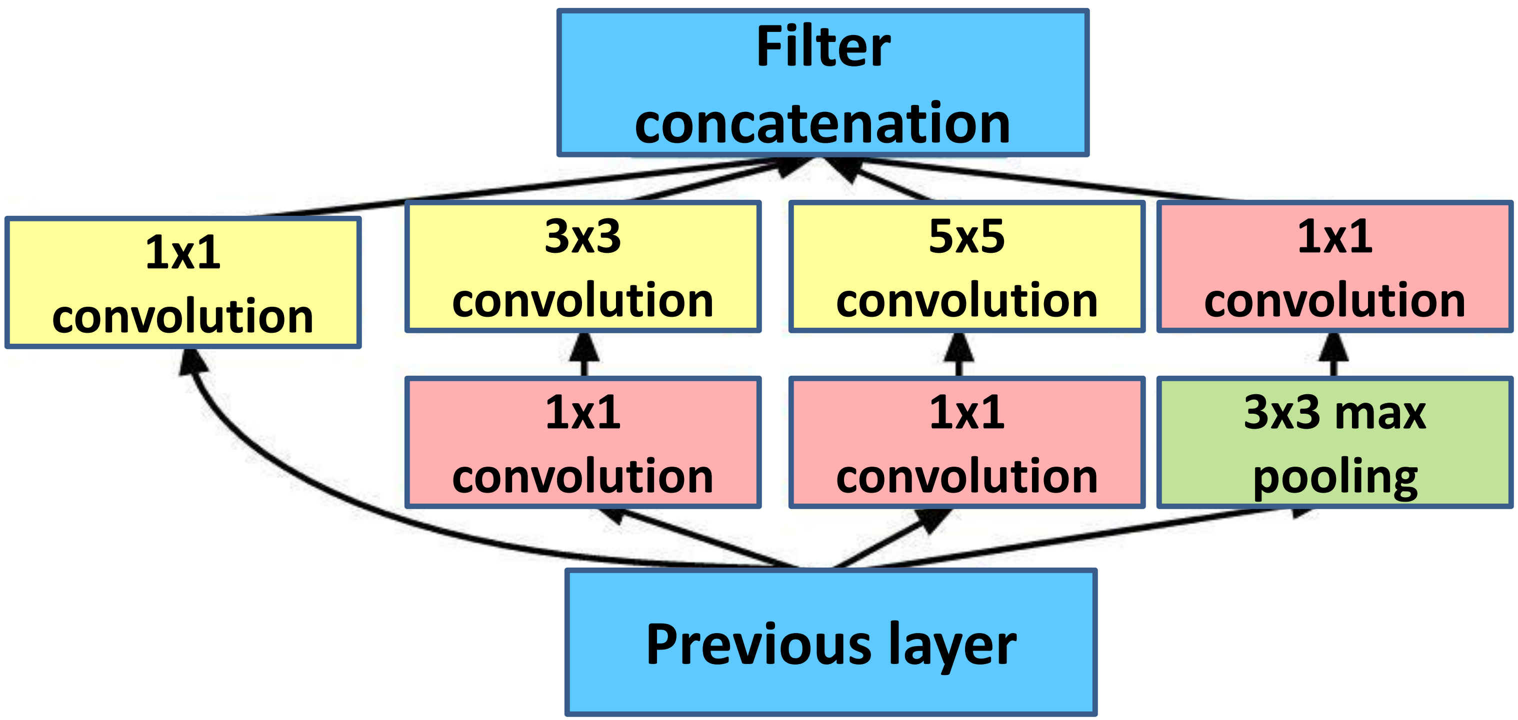
GoogleNet

- **Solution:** “bottleneck” layers that use 1x1 convolutions to reduce feature depth (from previous hour).

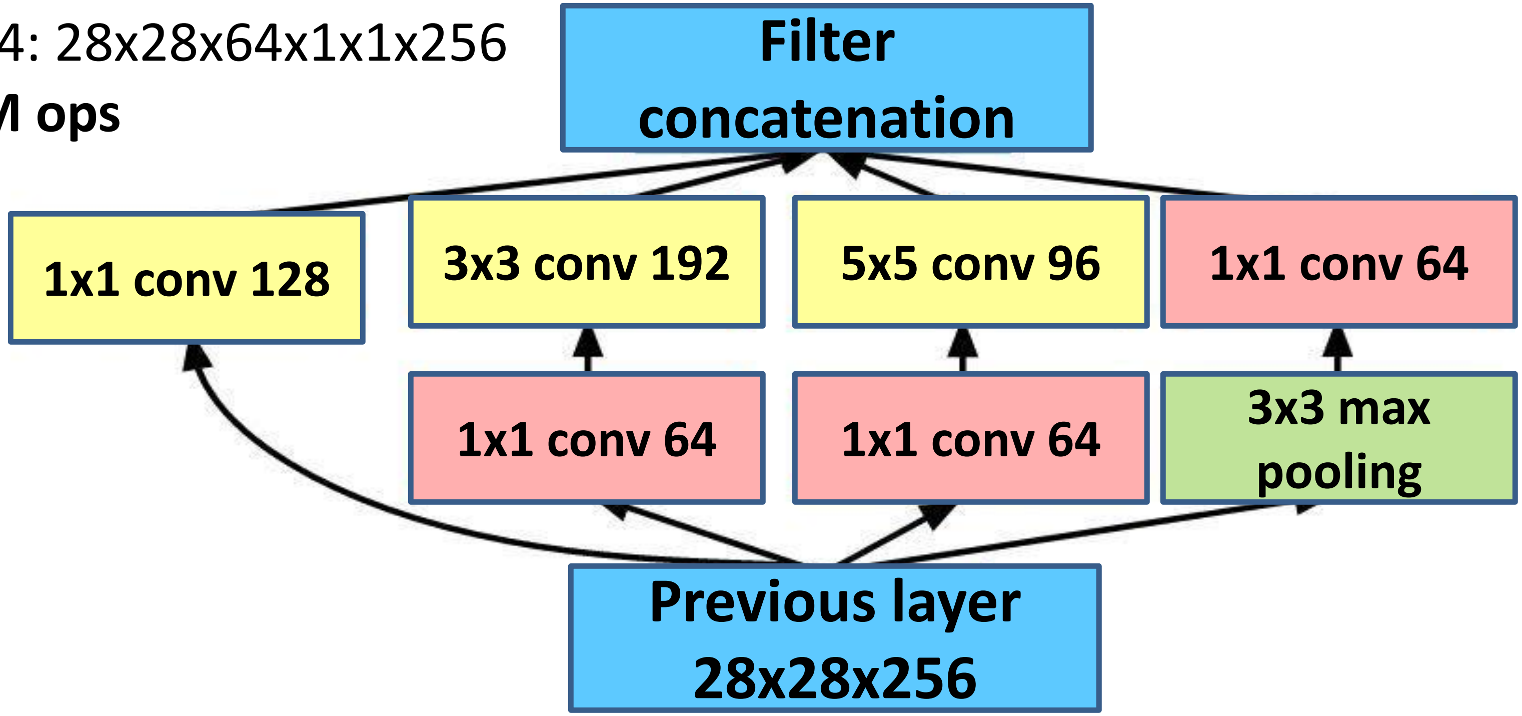


GoogleNet

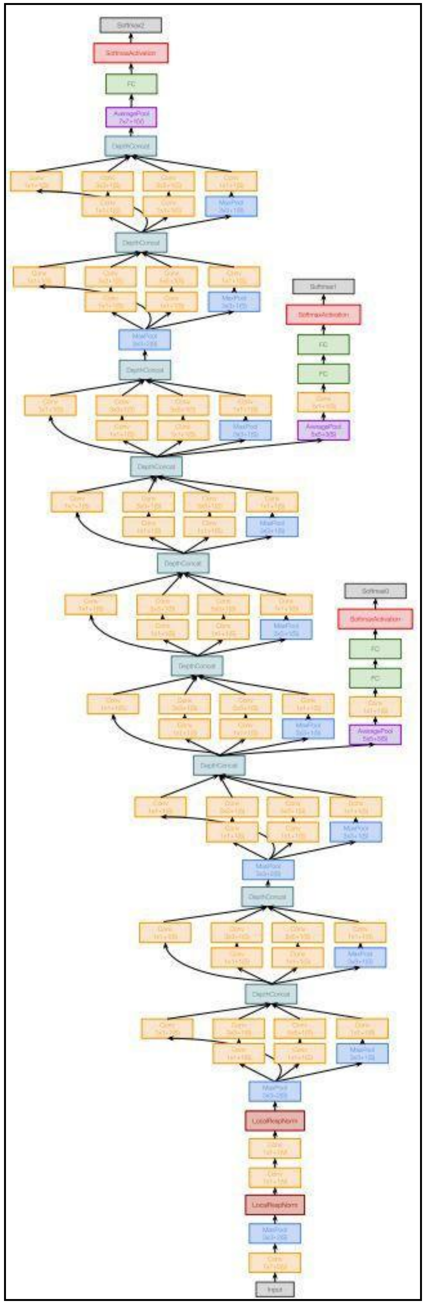
- **Solution:** “bottleneck” layers that use 1x1 convolutions to reduce feature depth (from previous hour).



- **Number of convolution operations:**
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 128: 28x28x128x1x1x256
  3x3 conv, 192: 28x28x192x3x3x64
  5x5 conv, 96: 28x28x96x5x5x264
  1x1 conv, 64: 28x28x64x1x1x256
  Total: 353M ops



- Compared to 854M ops for naive version



GoogleNet

Details/Retrospectives :

- 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)

ResNet

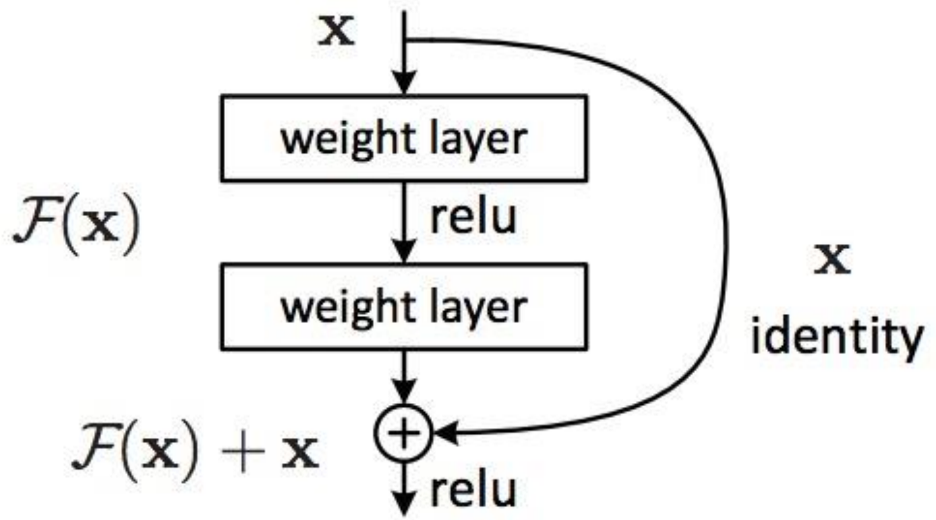
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: **Deep Residual Learning for Image Recognition,** arXiv preprint arXiv:1512.03385,2015. IEEE CVPR 2016

ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

ResNet: the residual module

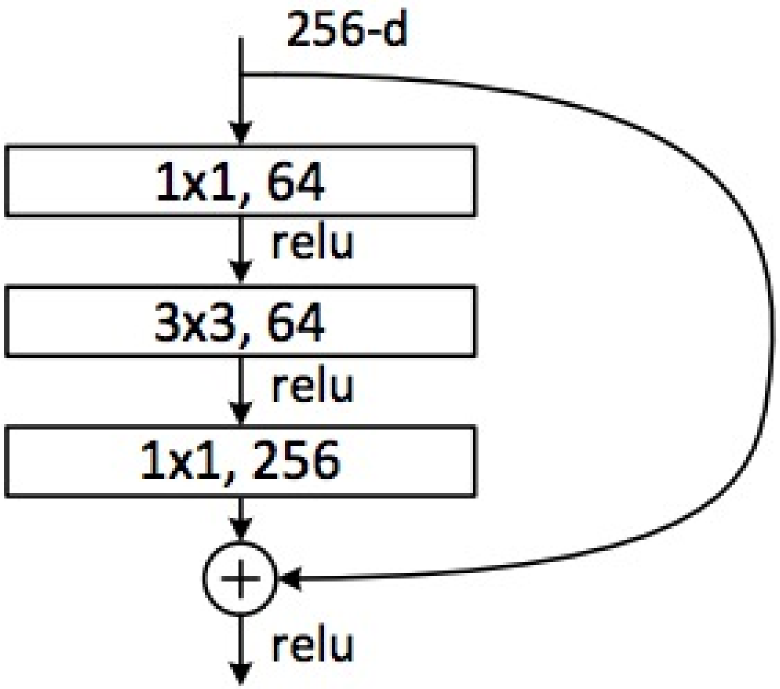
- Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
- Make it easy for network layers to represent the identity mapping
- For some reason, need to skip at least two layers



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, [Deep Residual Learning for Image Recognition], CVPR 2016 (Best Paper)

ResNet

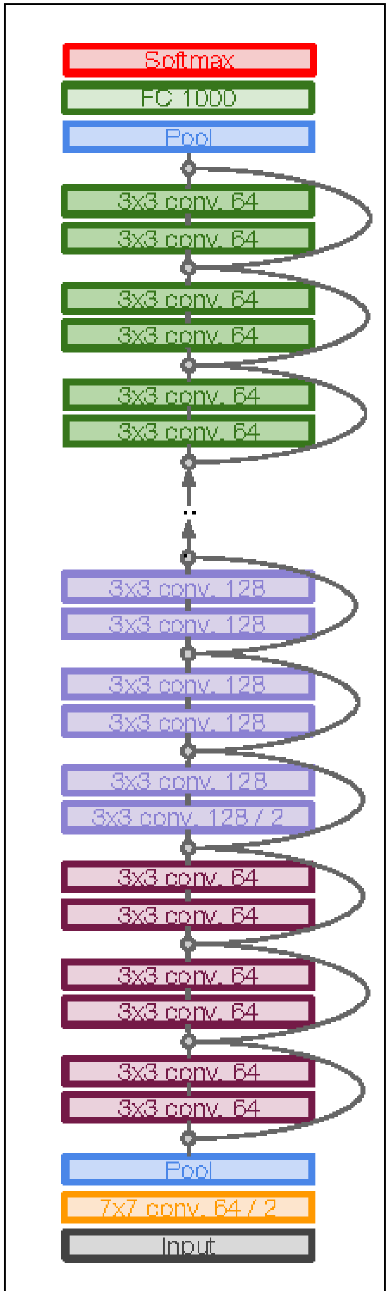
Deeper residual module (bottleneck)



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K
  operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, [Deep Residual Learning for Image Recognition](), CVPR 2016 (Best Paper)

Slide: Lazebnik

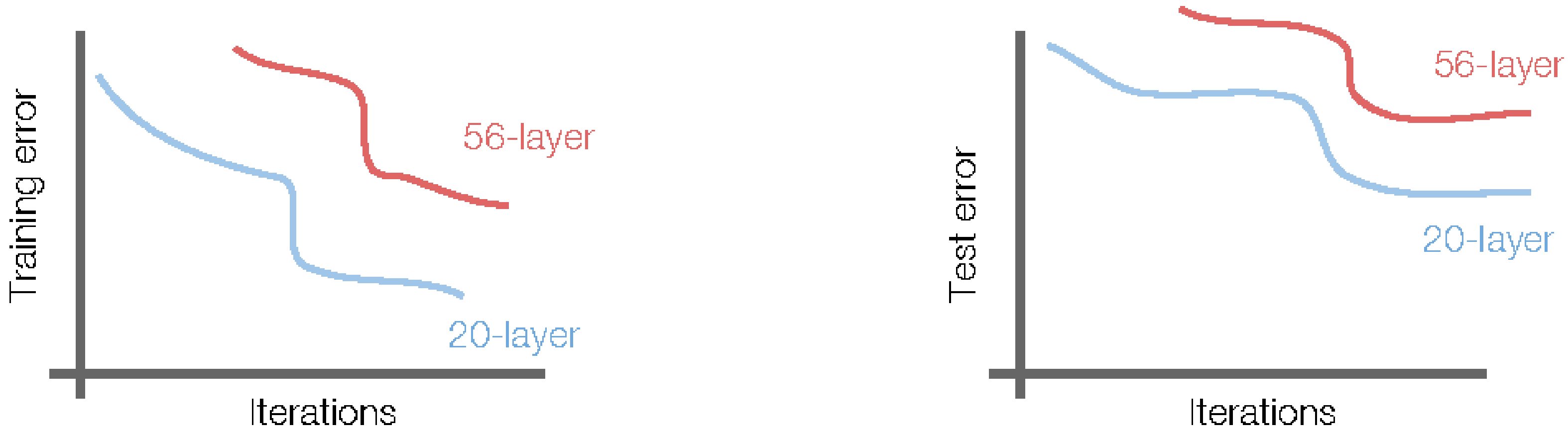


ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?



- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

ResNet

- **Hypothesis**: The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution**: Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.

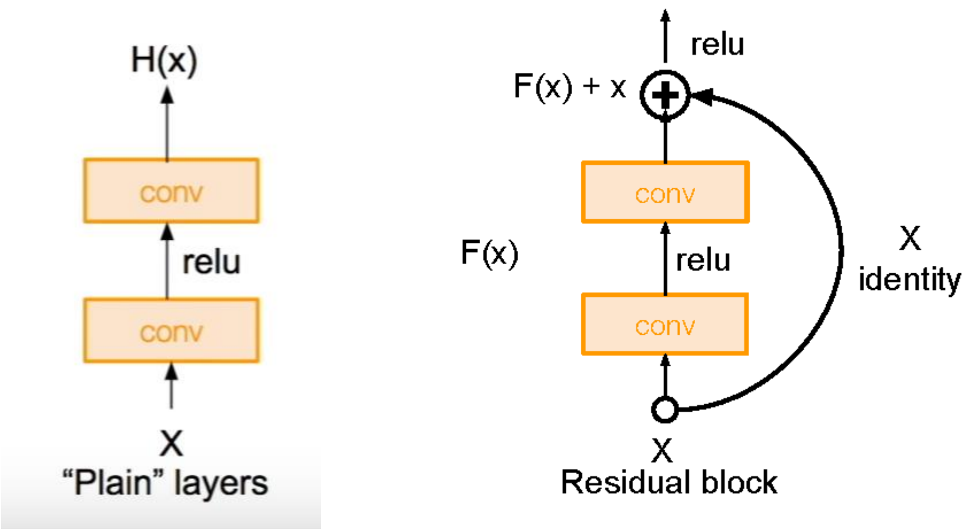
- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual $F(x) = H(x) – x$
  instead of $H(x)$ directly

ResNet

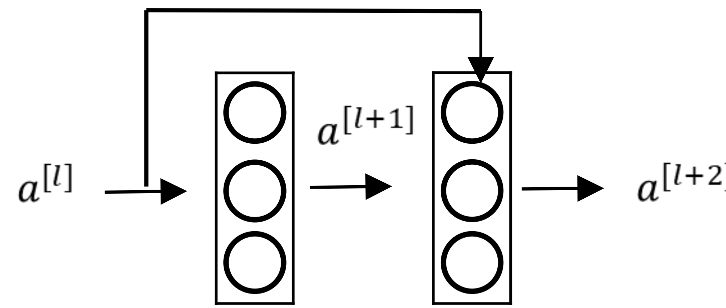
Residual Block

Input x goes through conv-relu-conv series and gives us F(x). That result is then added to the original input x. Let's call that H(x) = F(x) + x.

In traditional CNNs, H(x) would just be equal to F(x). So, instead of just computing that transformation (straight from x to F(x)), we're computing the term that we have to *add*, F(x), to the input, x.



ResNet



Short cut/ skip connection

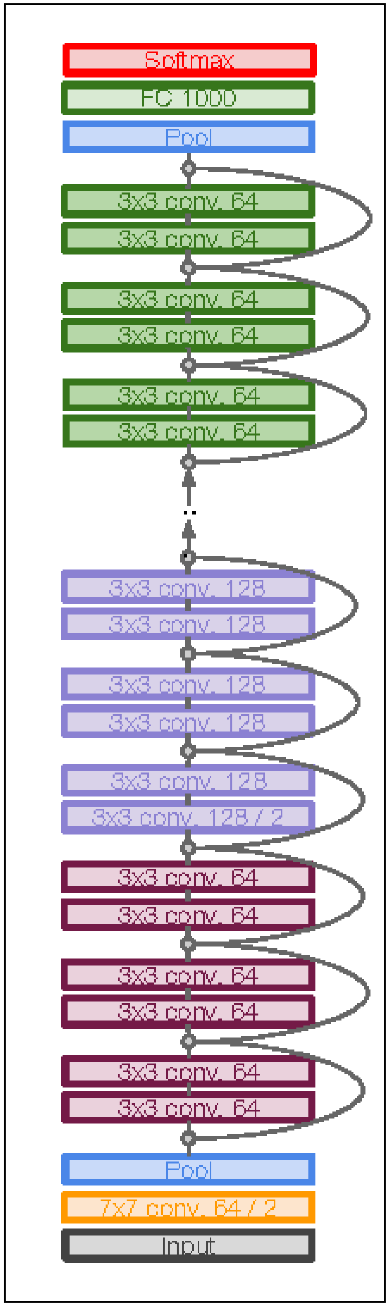
$$a^{[l]} \rightarrow \textbf{Linear} \rightarrow \textbf{ReLU} \rightarrow \textbf{Linear} \rightarrow \textbf{ReLU} \rightarrow a^{[l+2]}$$

$a^{[l+1]}$

$$\mathbf{z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}} \qquad \mathbf{z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}}$$

$$\mathbf{a^{[l+1]} = g(z^{[l+1]})} \qquad \mathbf{a^{[l+2]} = g(z^{[l+2]})}$$

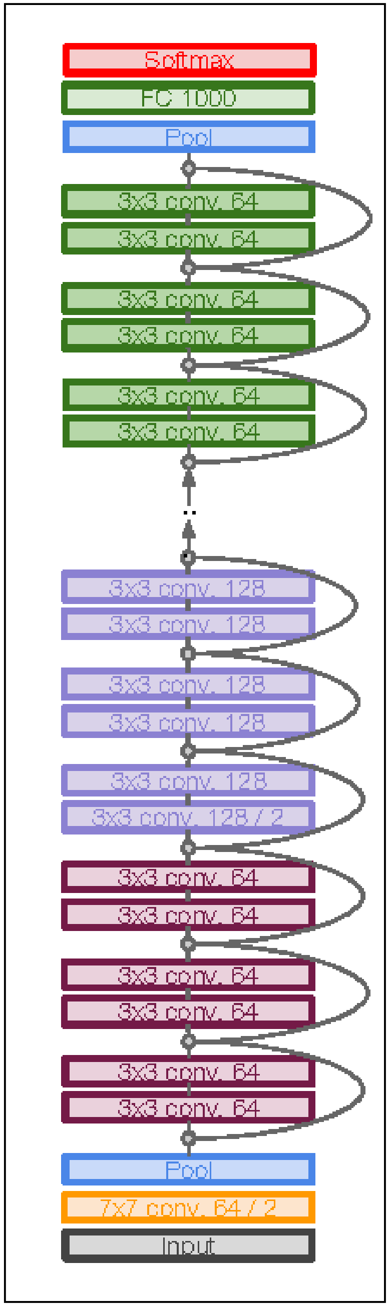
$$\mathbf{a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right) = g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})}$$



ResNet

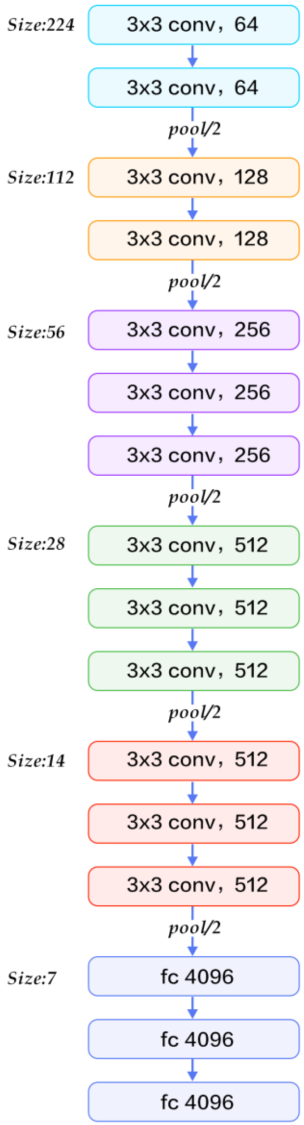
Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

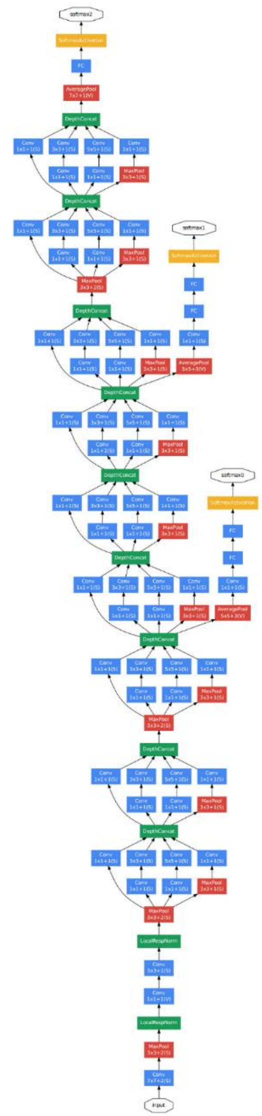


ResNet

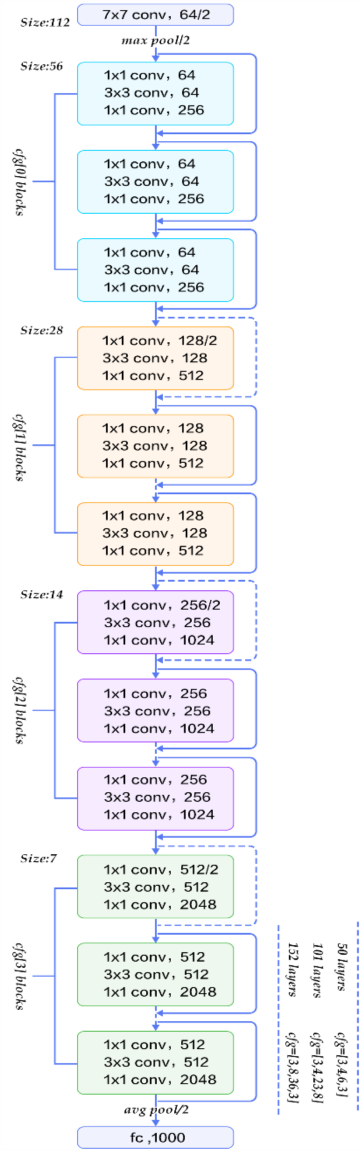
- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



VGG-16



GoogleNet



ResNet

Reading list

- https://culurciello.github.io/tech/2016/06/04/nets.html
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.
- A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- D. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, ICLR 2015
- M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014 (best paper award)
- K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
- M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014
- C. Szegedy et al., Going deeper with convolutions, CVPR 2015
- C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016
- K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (best paper award)