Statistical Methods in AI (CS7.403)

Introduction to CNN/Deep Learning

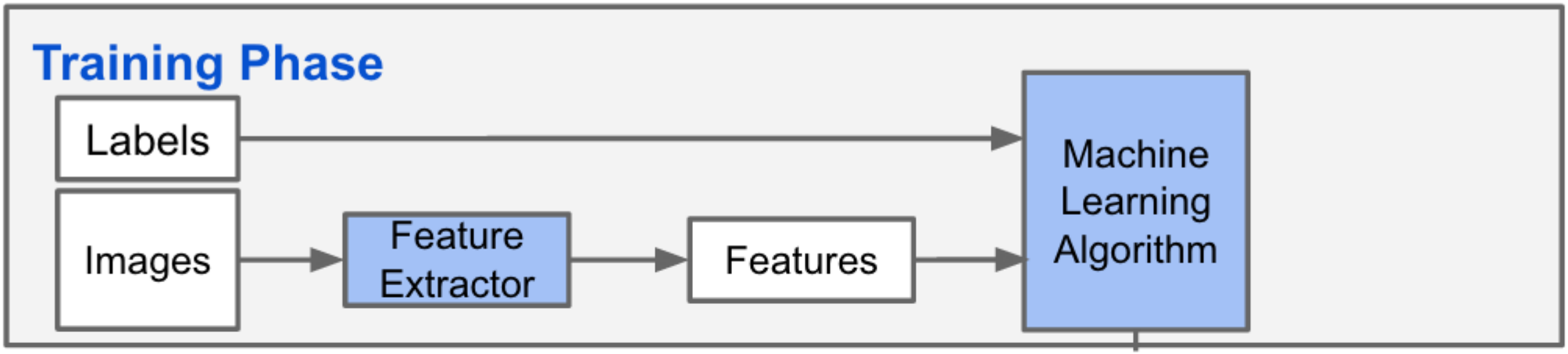
Ravi Kiran (ravi.kiran@iiit.ac.in)

https://ravika.github.io

@vikataravi

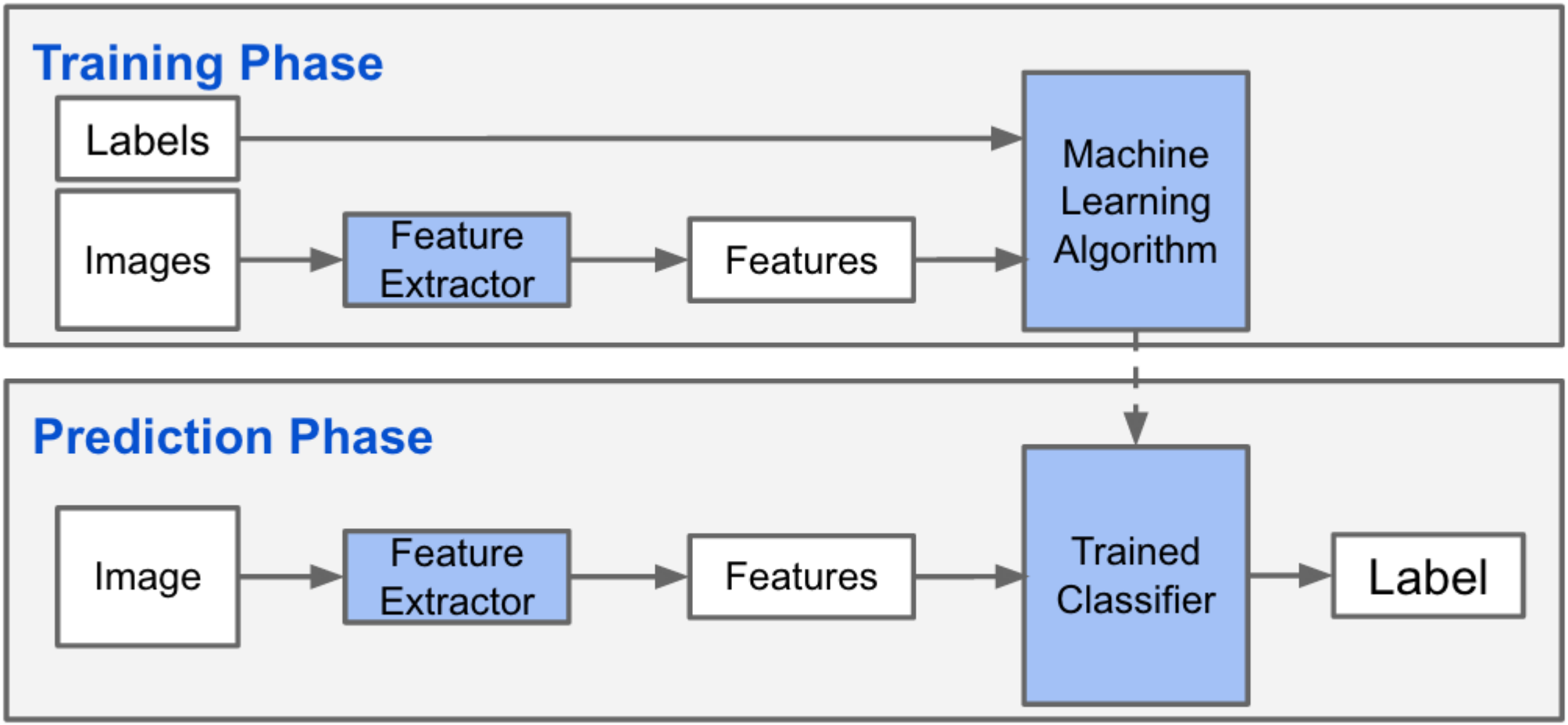
Center for Visual Information Technology (CVIT)
IIIT Hyderabad

Classification



Source: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Classification



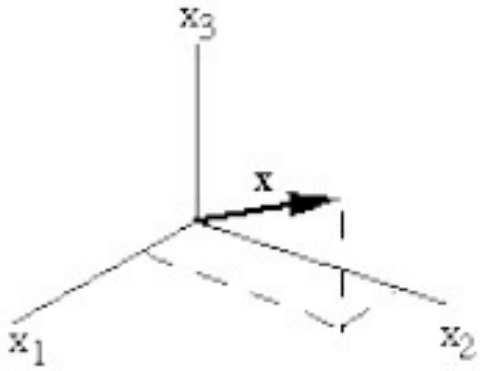
Source: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Feature Vectors

- Usually a single object can be represented using several features, e.g.
  - **x1** = shape (e.g. nr of sides)
  - **x2** = size (e.g. some numeric value)
  - **x3** = color (e.g. rgb values)
  - ...
  - **xd** = some other (numeric) feature.

- **X** becomes a feature vector
  - x is a point in a d-dimensional **feature space.**

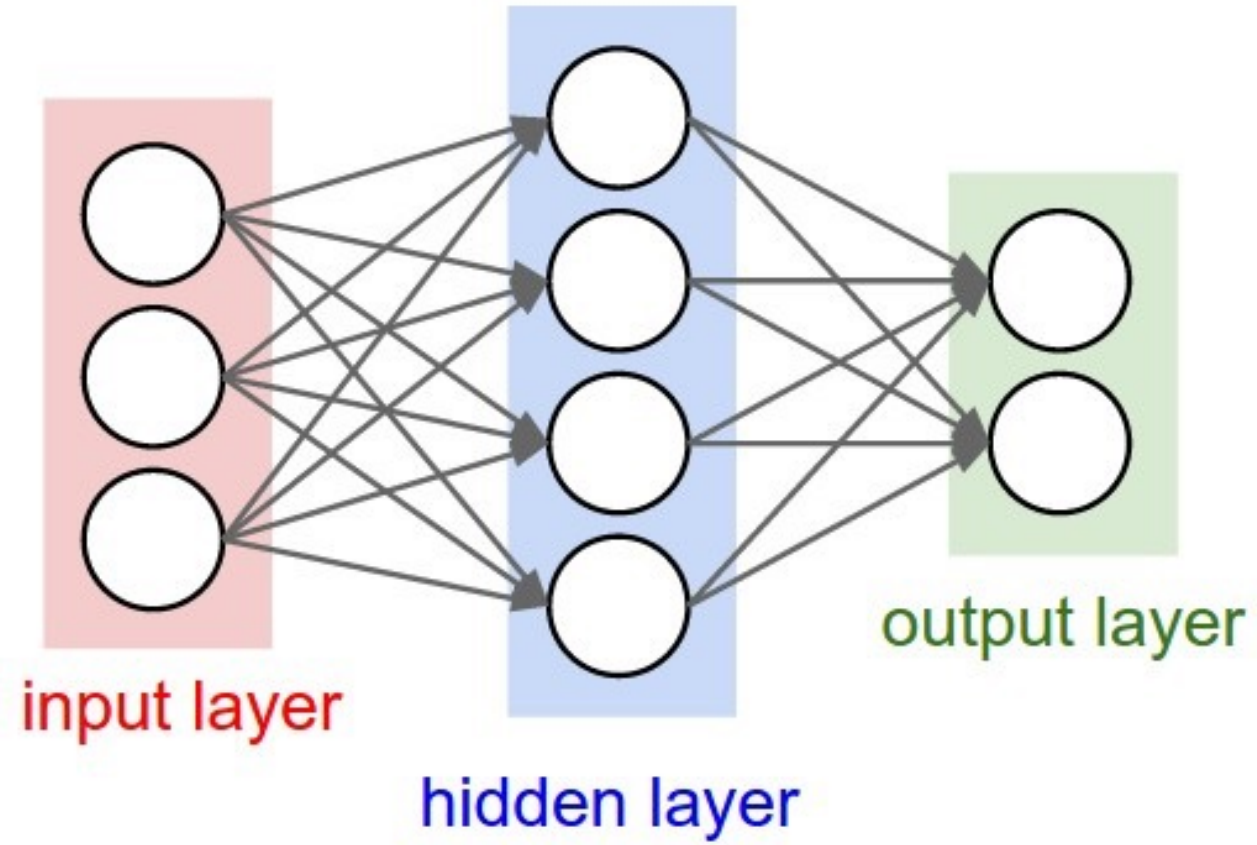
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_d \end{bmatrix}$$



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{x}$$

Source: https://w

Traditional approach: Extract features, feed to NN

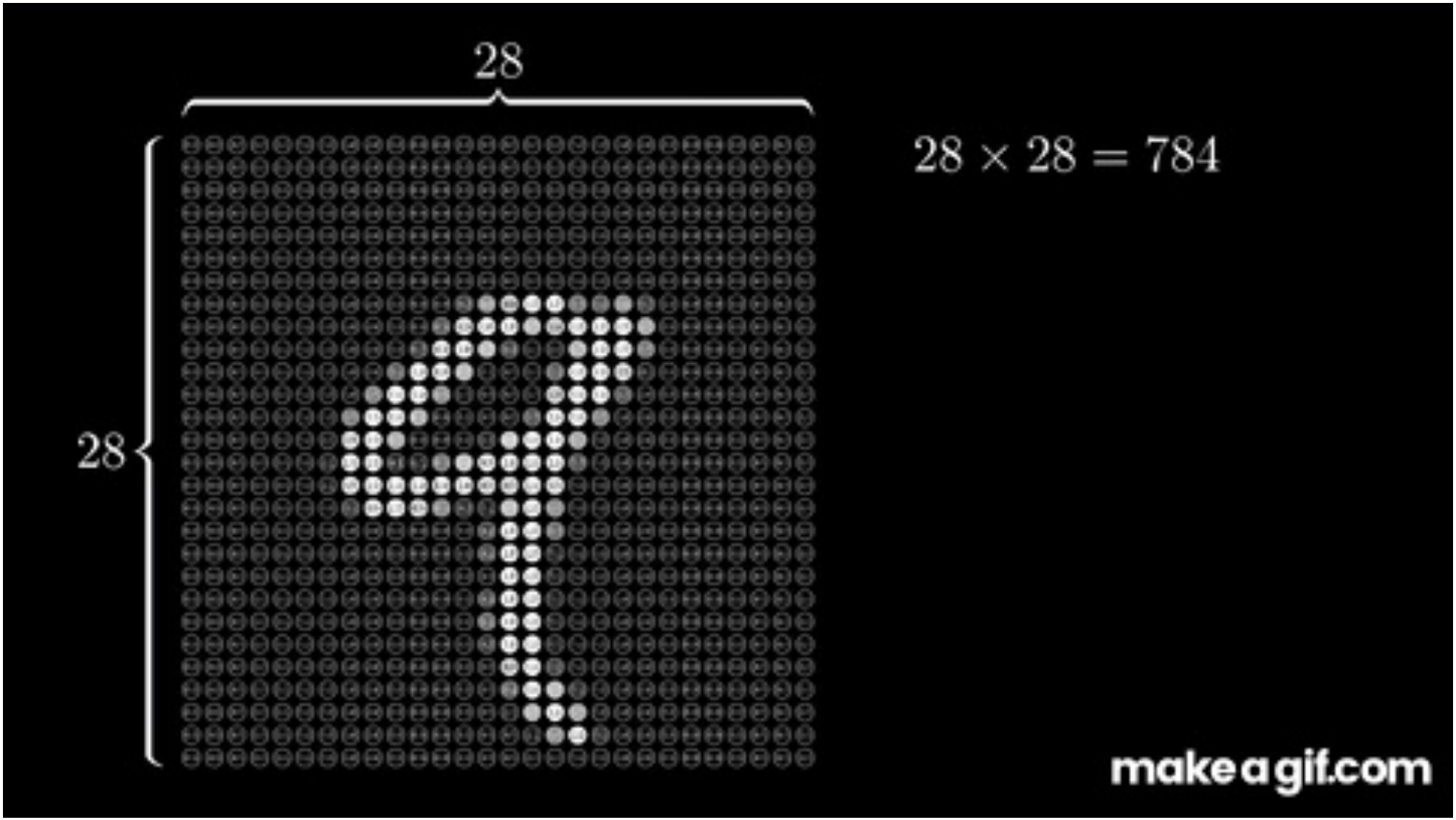


Wouldn't it be nice if we could feed the image directly ?

… and let the “learning process” figure out which features to extract ?

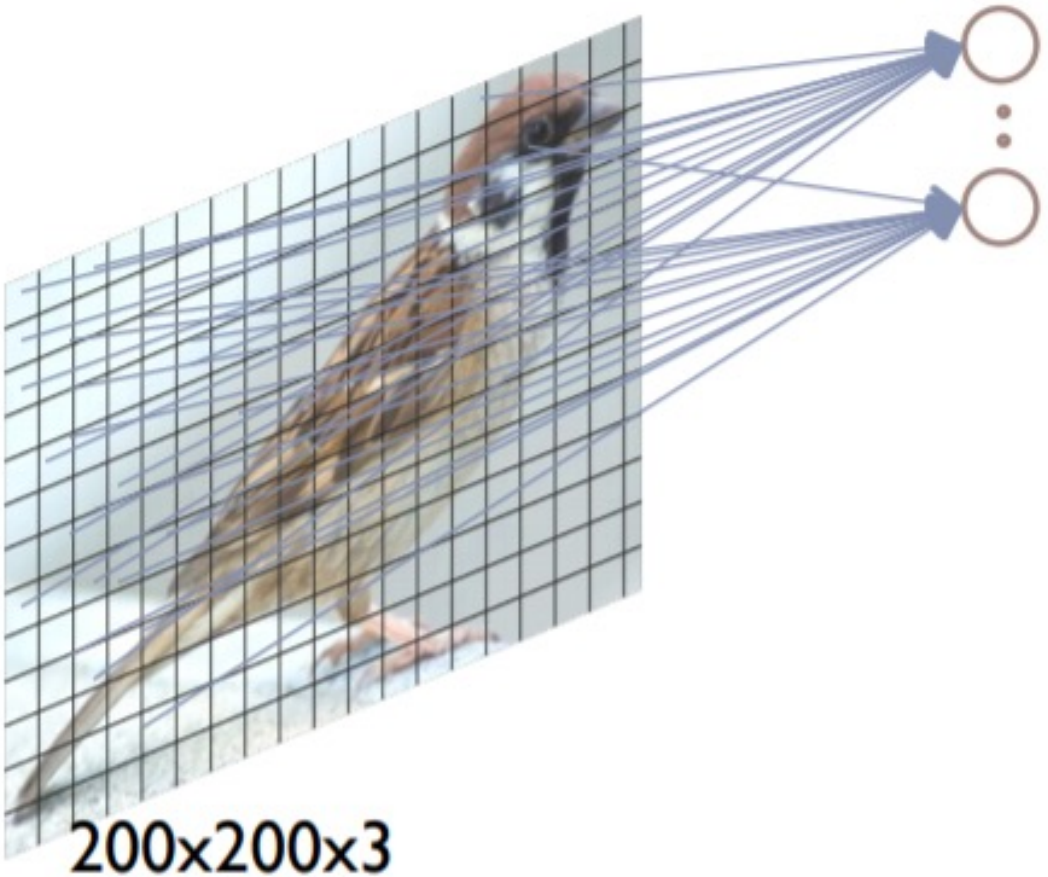
MNIST Handwritten Digits Dataset

Flatten image , Feed to NN



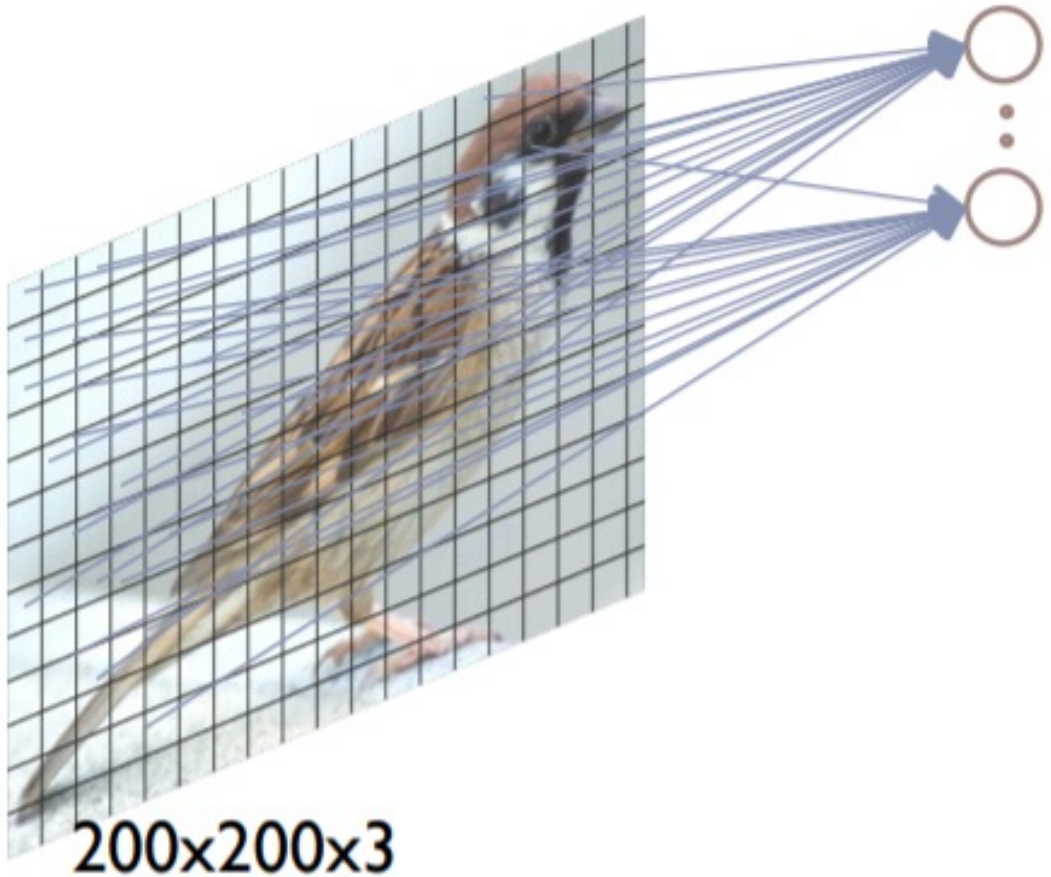
https://www.youtube.com/watch?v=aircAruvnKk

For color images ?



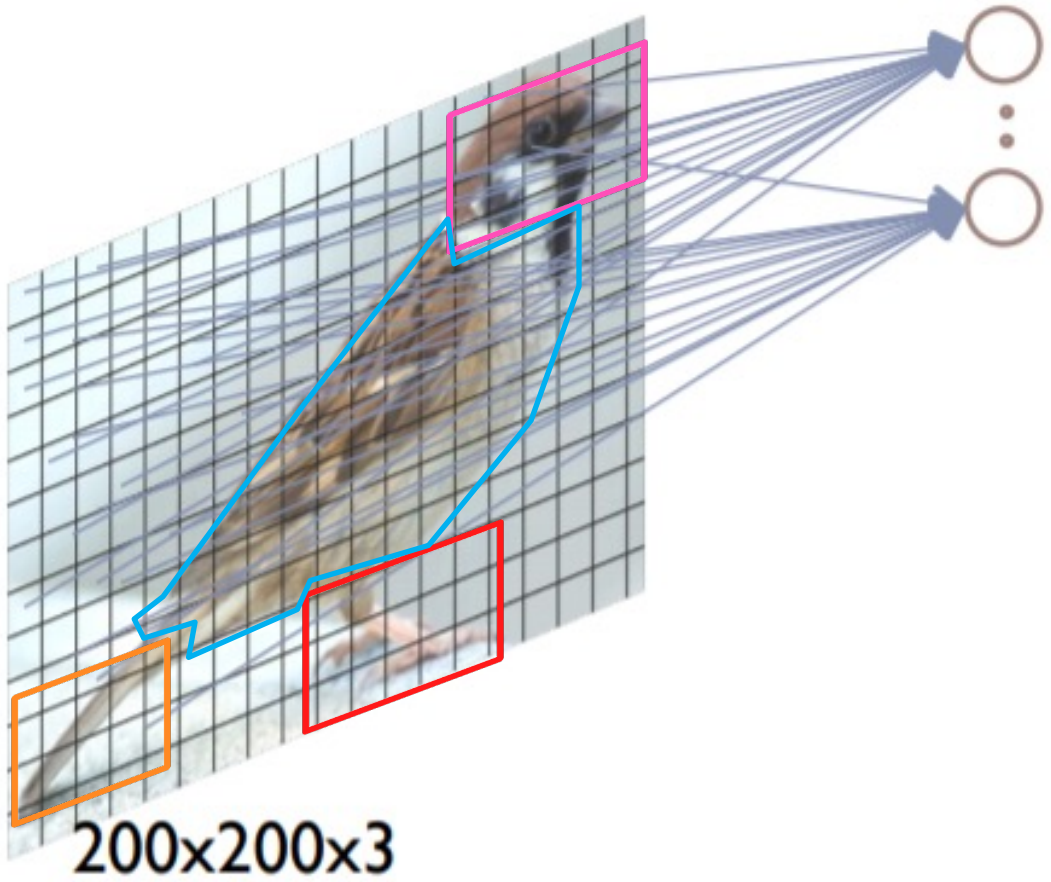
- #Hidden Units: 120,000
- #Params: 14.4 billion
- Need huge training data to prevent over-fitting!

Do we really need full connectivity ?



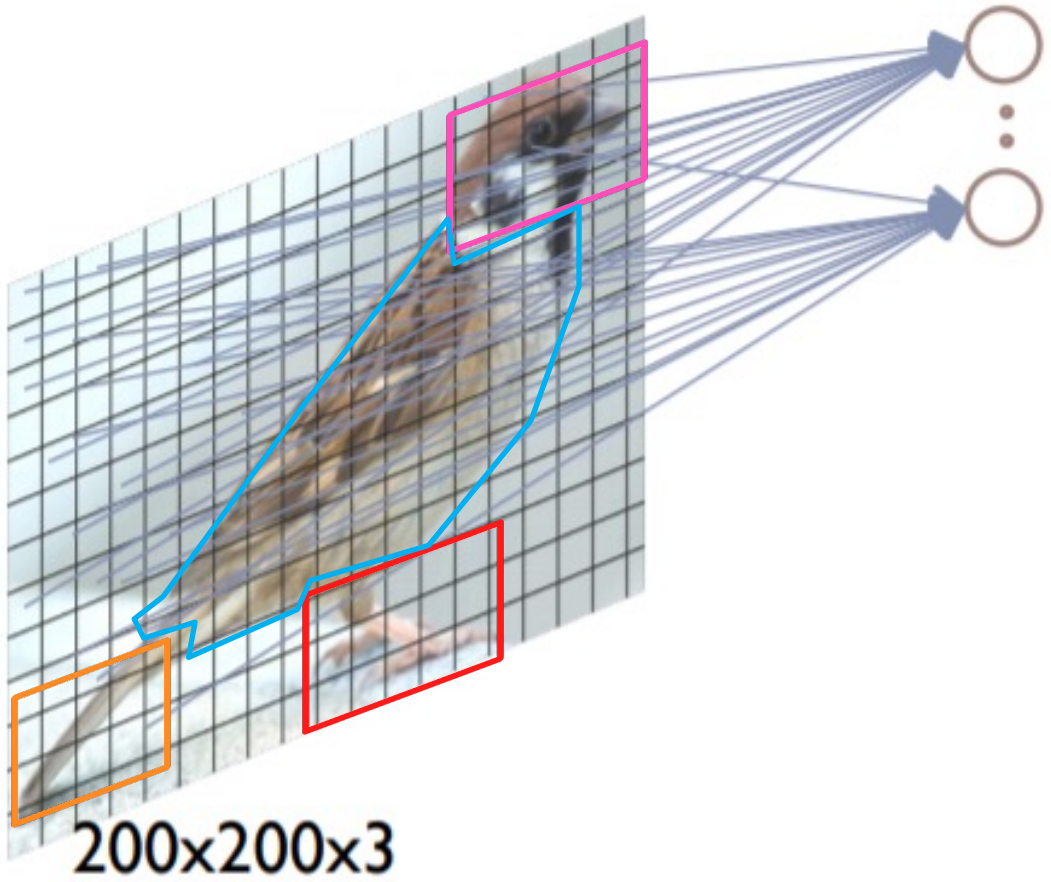
- #Hidden Units: 120,000
- #Params: 14.4 billion
- Need huge training data to prevent over-fitting!

Do we really need full connectivity ?



- Head + Body + Tail + Legs = sparrow

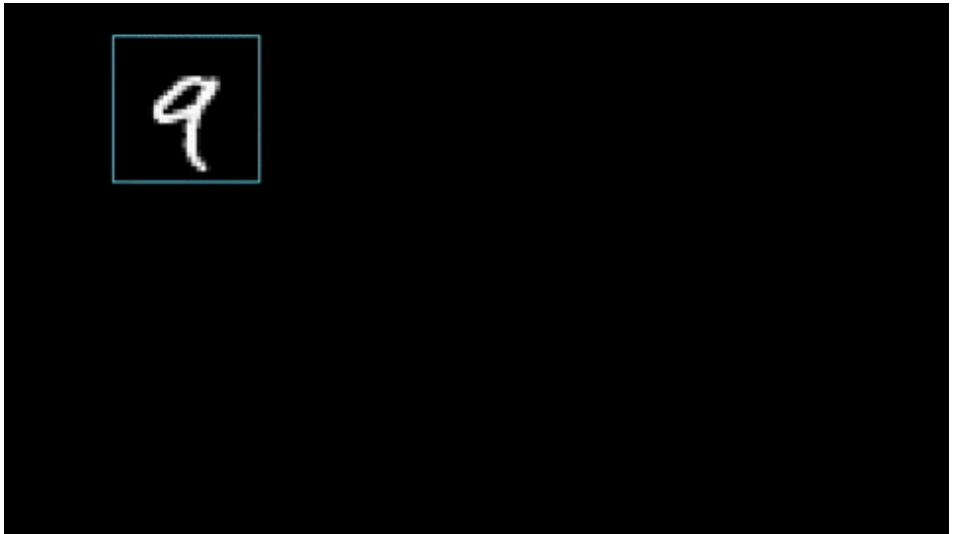
Do we really need full connectivity ?



- Head + Body + Tail + Legs = sparrow

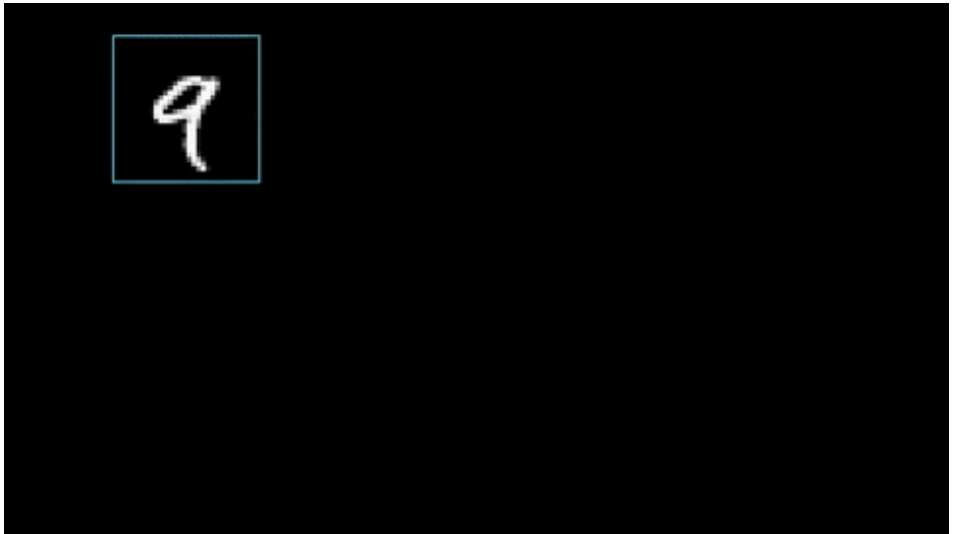
https://www.youtube.com/watch?v=aircAruvnKk

- Images are 2D.
- Assumption: Object image = combination of **2D image patterns**



- Images are 2D.
- Assumption: Object image = combination of **2D image patterns**

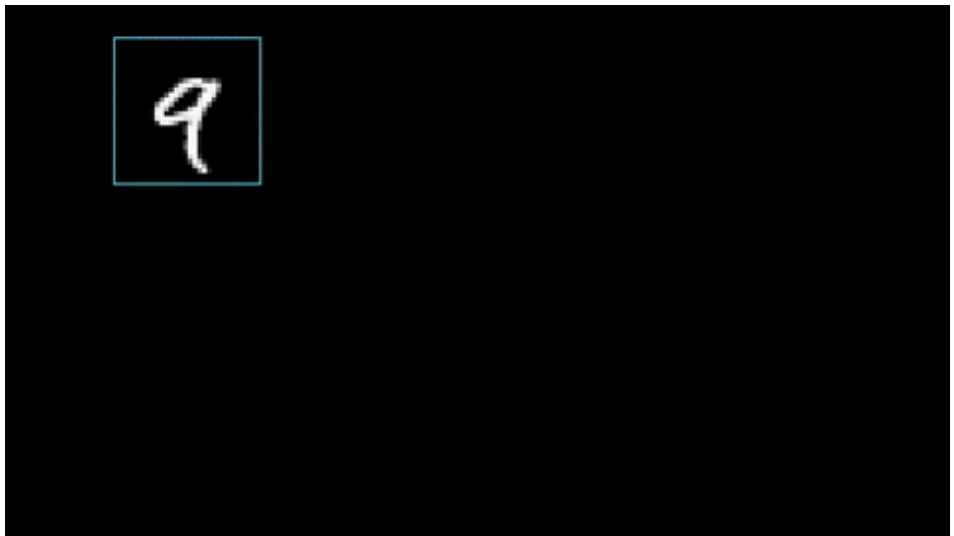
- Machine Learning Strategy:
  a) [Pre-final layer] Determine 2D image patterns
  b) Map 2D image patterns → Target label



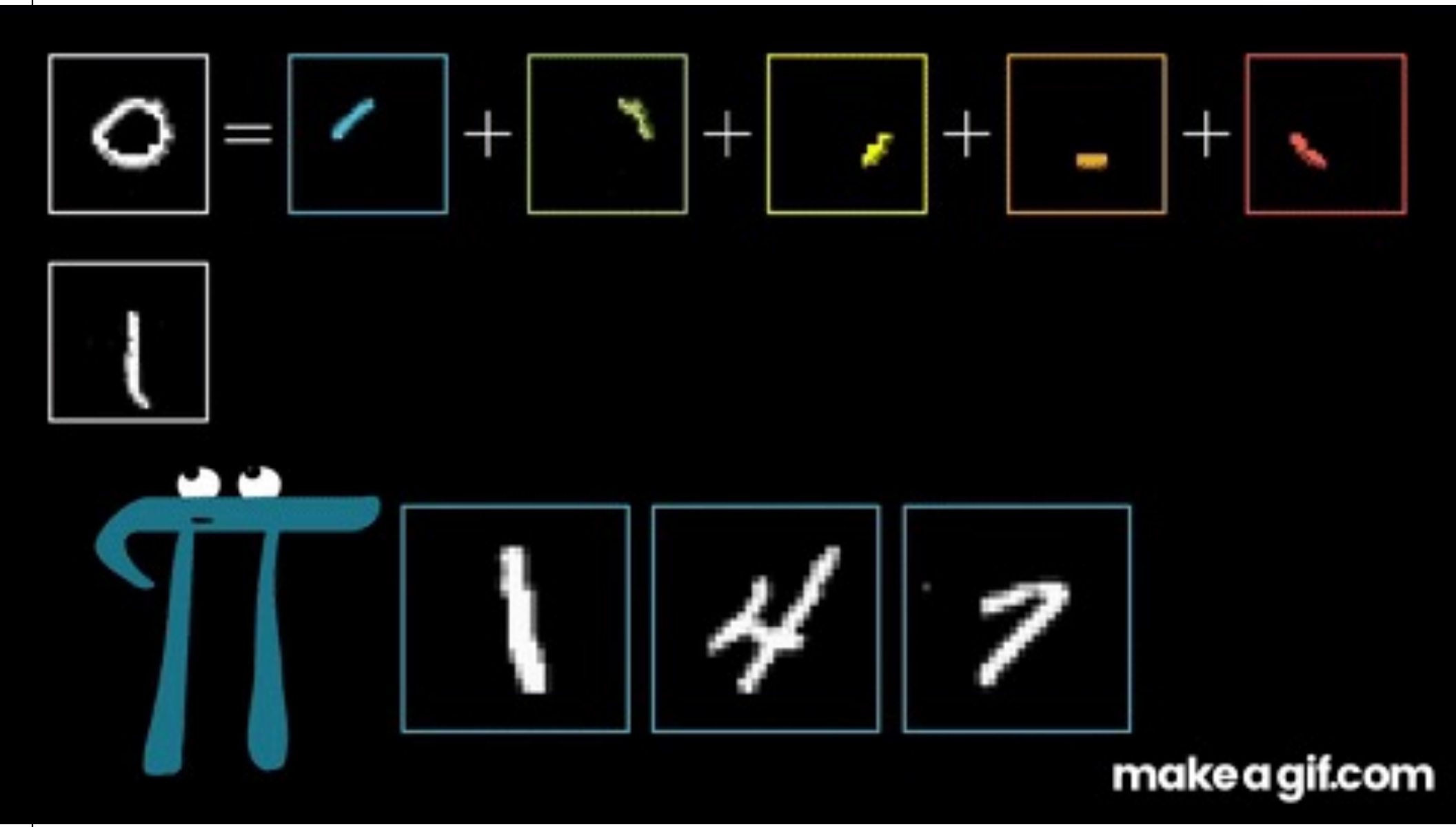
- Images are 2D.
- Assumption: Object image = combination of **2D image patterns**

- Machine Learning Strategy:
  - [Pre-final layer] Determine 2D image patterns
  - Map 2D image patterns → Target label

- NOTE: 2D image patterns are smaller than image



Hierarchical decomposition of input/features

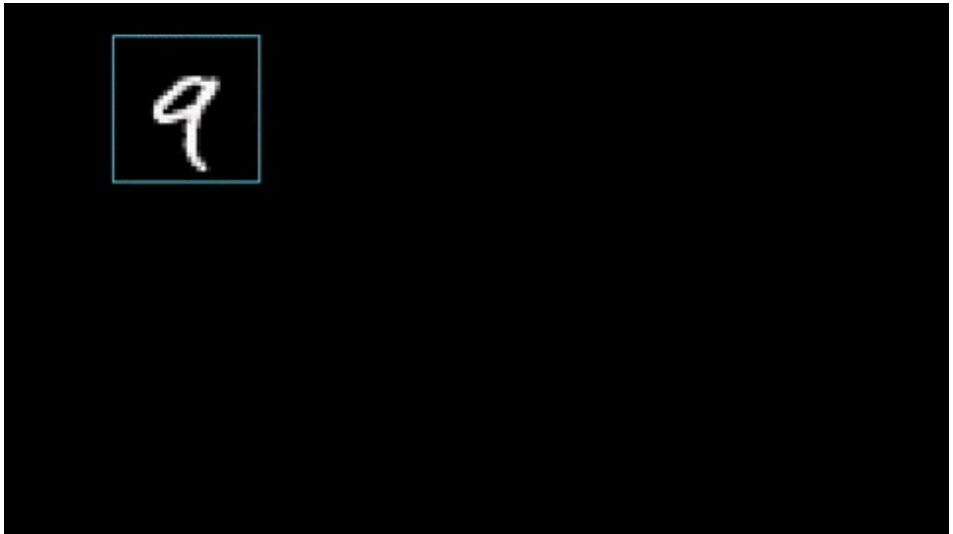


https://www.youtube.com/watch?v=aircAruvnKk

- Images are 2D.
- Assumption: Object image = **hierarchical** combination of **2D image patterns**

- Machine Learning Strategy:
  - [All except pre-final] Determine 2D image patterns
  - [Pre-final layer] Map 2D image patterns → Target label

- NOTE: 2D image patterns are smaller than image

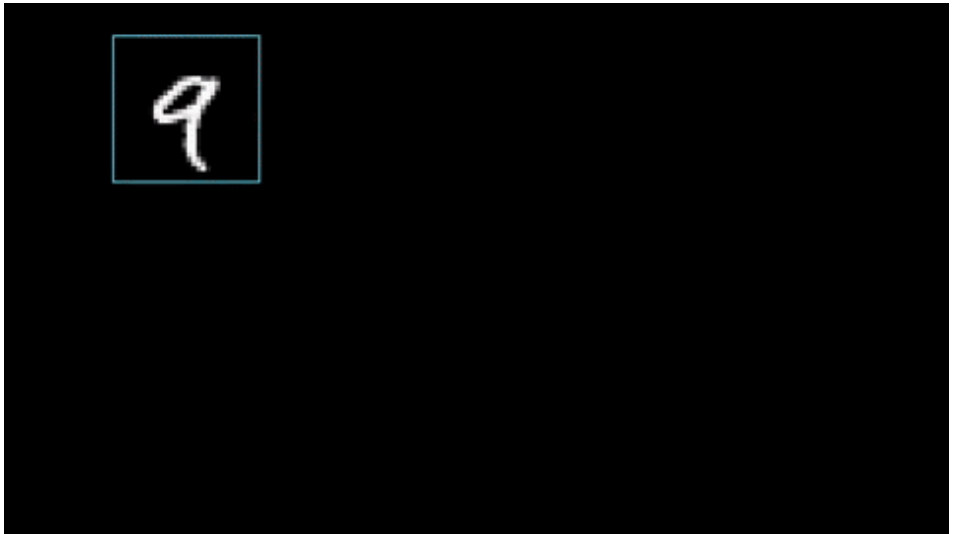


Efficient !

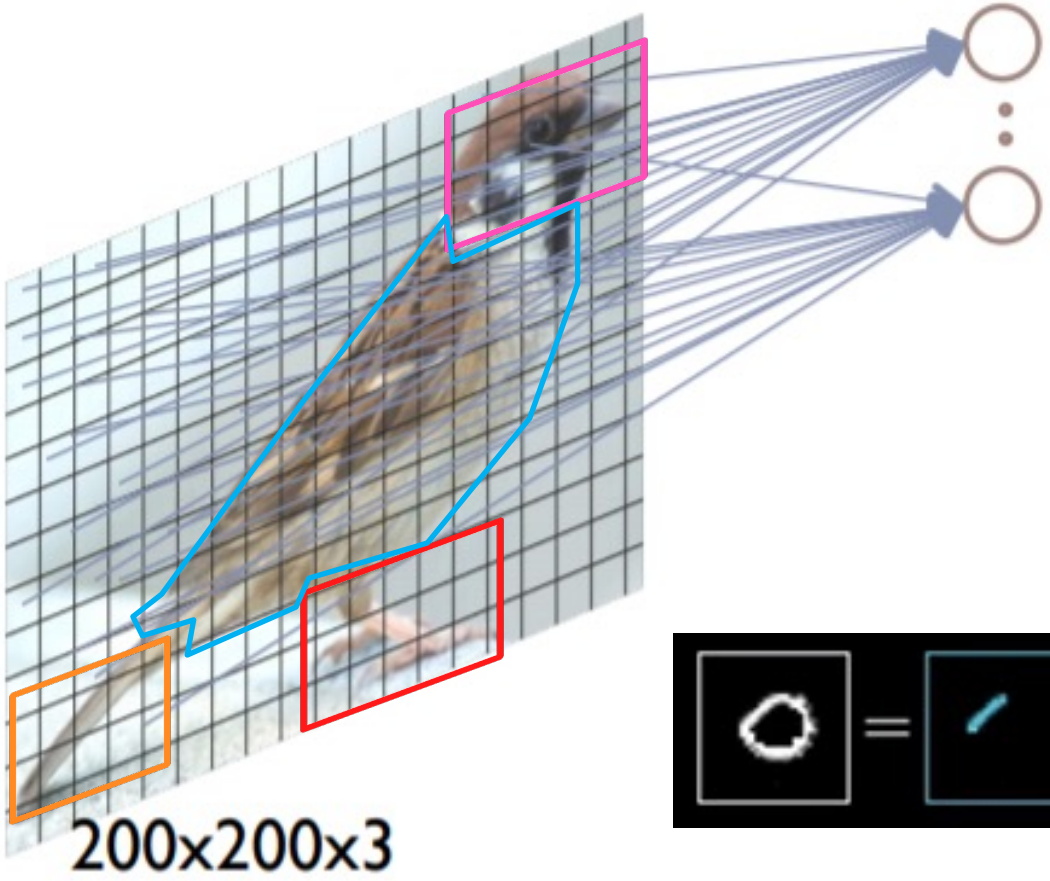
- Images are 2D.
- Assumption: Object image = **hierarchical** combination of **2D image patterns**

- Machine Learning Strategy:
  - [All except pre-final] Determine 2D image patterns
  - [Pre-final layer] Map 2D image patterns → Target label

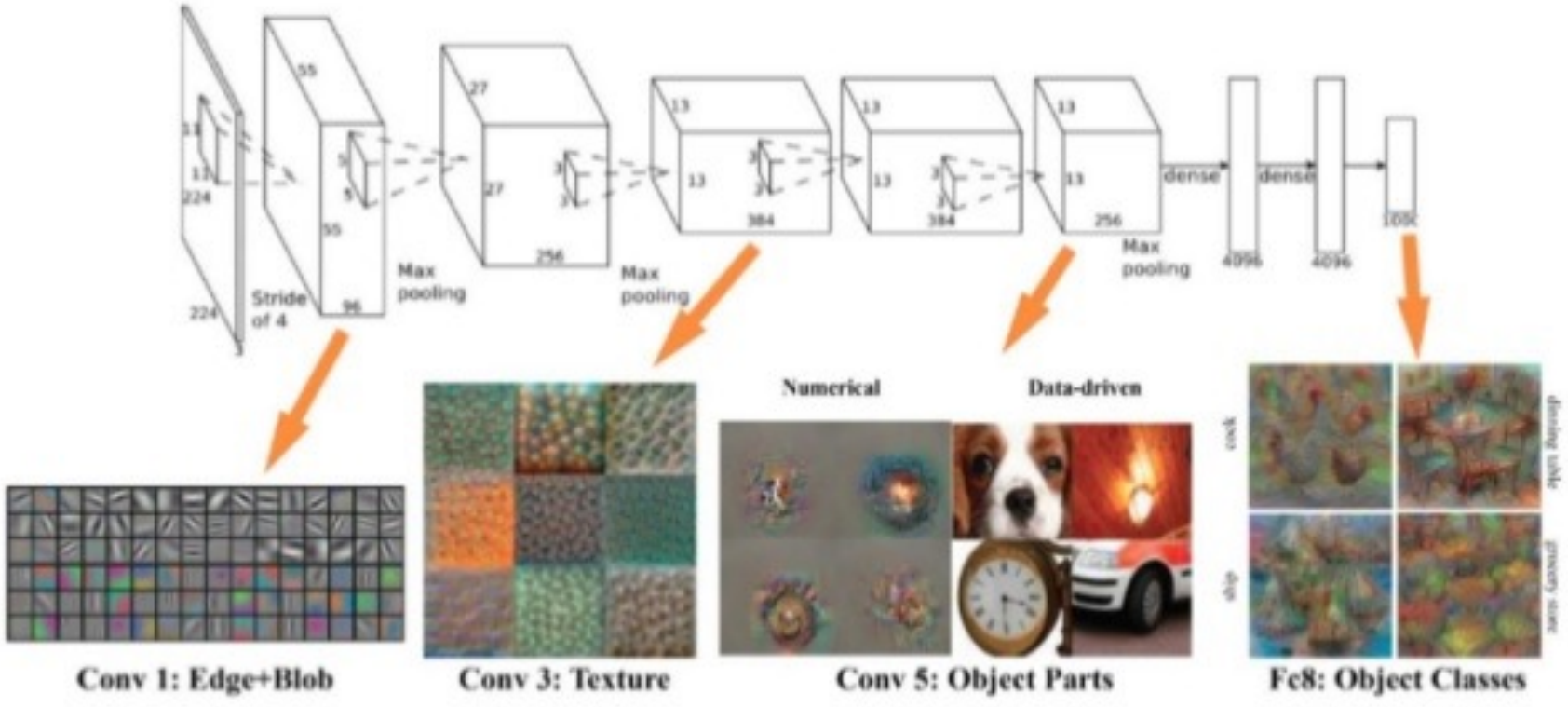
- NOTE: 2D image patterns are smaller than image



Neural Networks don’t really
learn such conveniently interpretable
Representations – this is just a toy illustration



SPOILER ALERT!

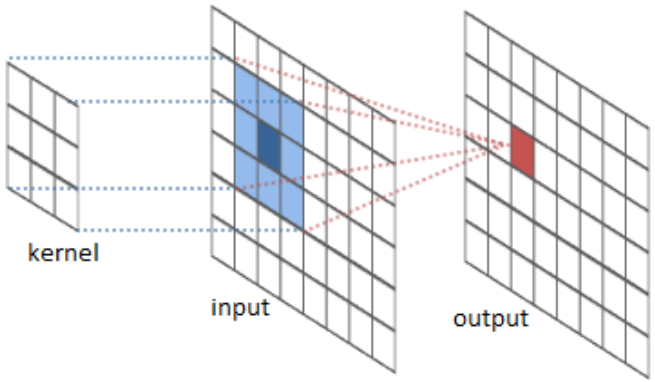


From: mNeuron: A Matlab Plugin to Visualize Neurons from Deep Models, Donglai Wei et. al.

- Pattern detectors = Filters
- What should be the size of filters ?
- How many filters do we need at each level ?
- How many levels ?

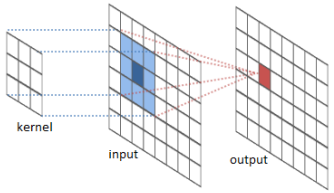
How do filters work ?

- Determine the filter
- ‘Scan’ the filter on input image / representation
- Output = A record of all input locations that ‘match’ the filter



$$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

How do filters work ?



0	0	0
0	-1	1
0	0	0

Input

Kernel Convolution

-1	-1	-1
-1	8	-1
-1	-1	-1

Feature Map

Features

Wouldn't it be nice if we could feed the image directly ?

… and let the “learning process” figure out which features to extract ?

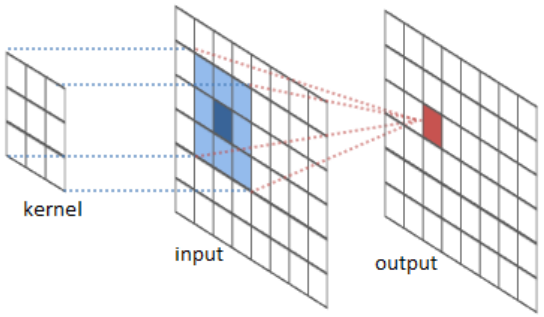
Wouldn't it be nice if we could feed the image directly ?

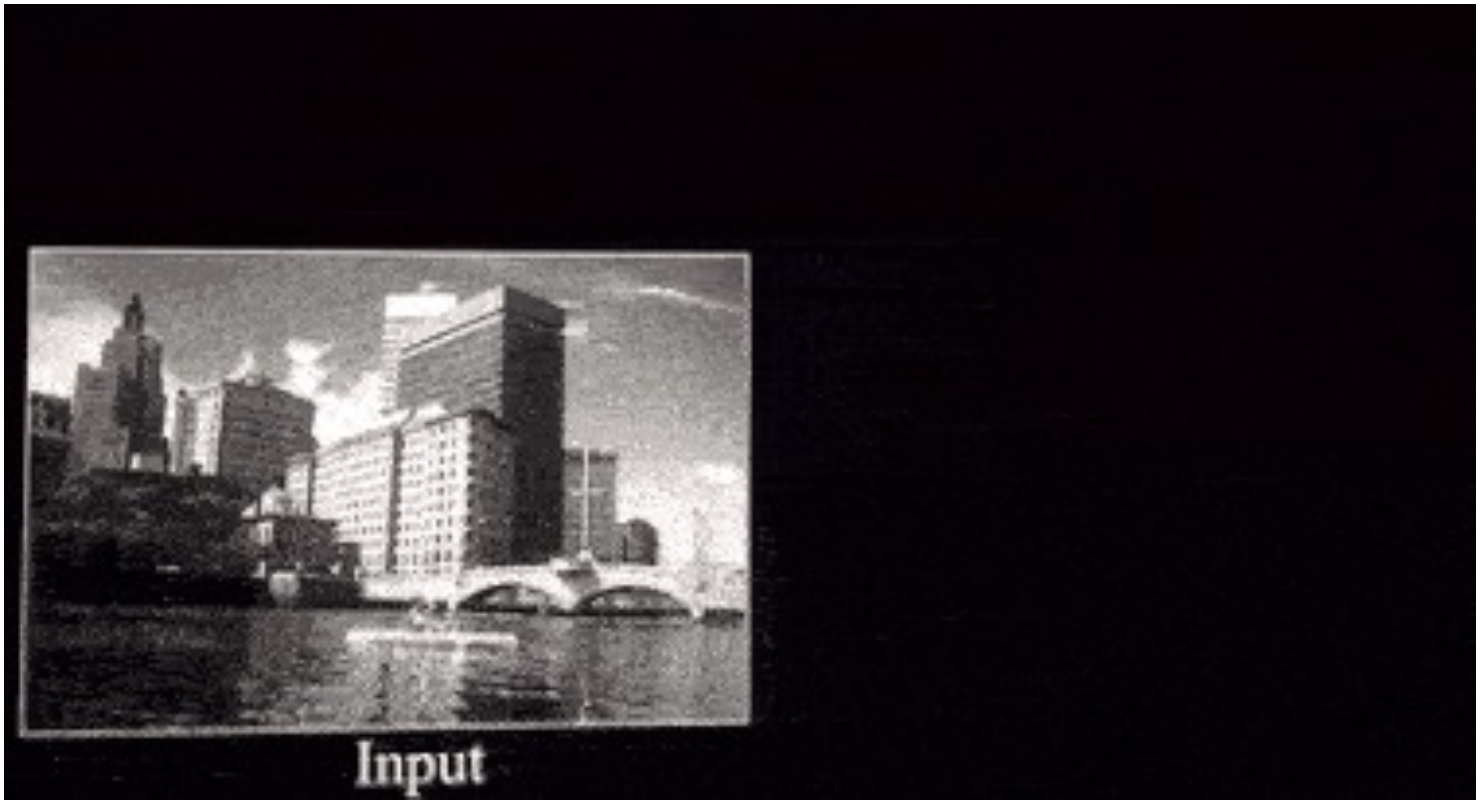
… and let the “learning process” figure out which **features** to extract ?

(which filters to construct)

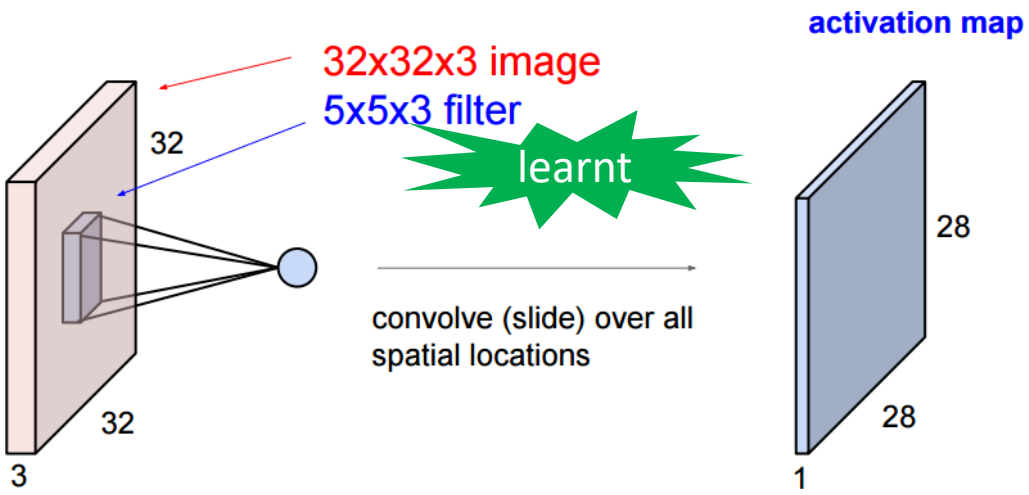
Convolution (in general)

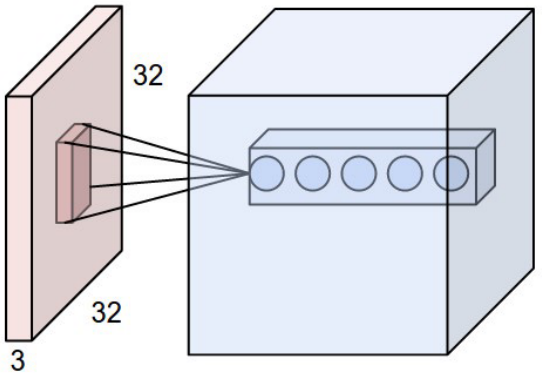
Multiple filters, multiple ‘feature maps’



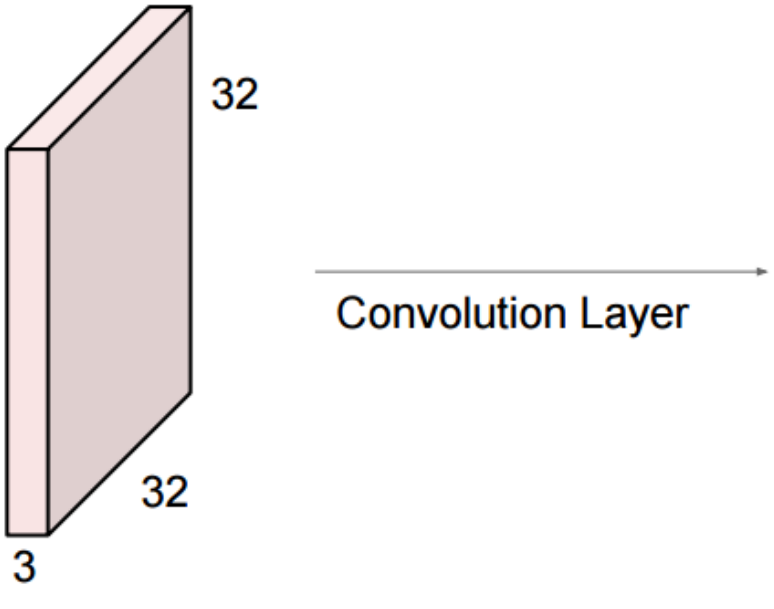


Convolution Layer





Input to next layer



Design choices : Filter size, # of filters

3x3

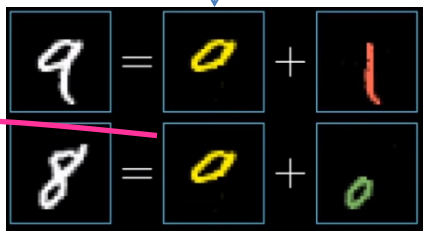
1	2	1
2	4	2
1	2	1

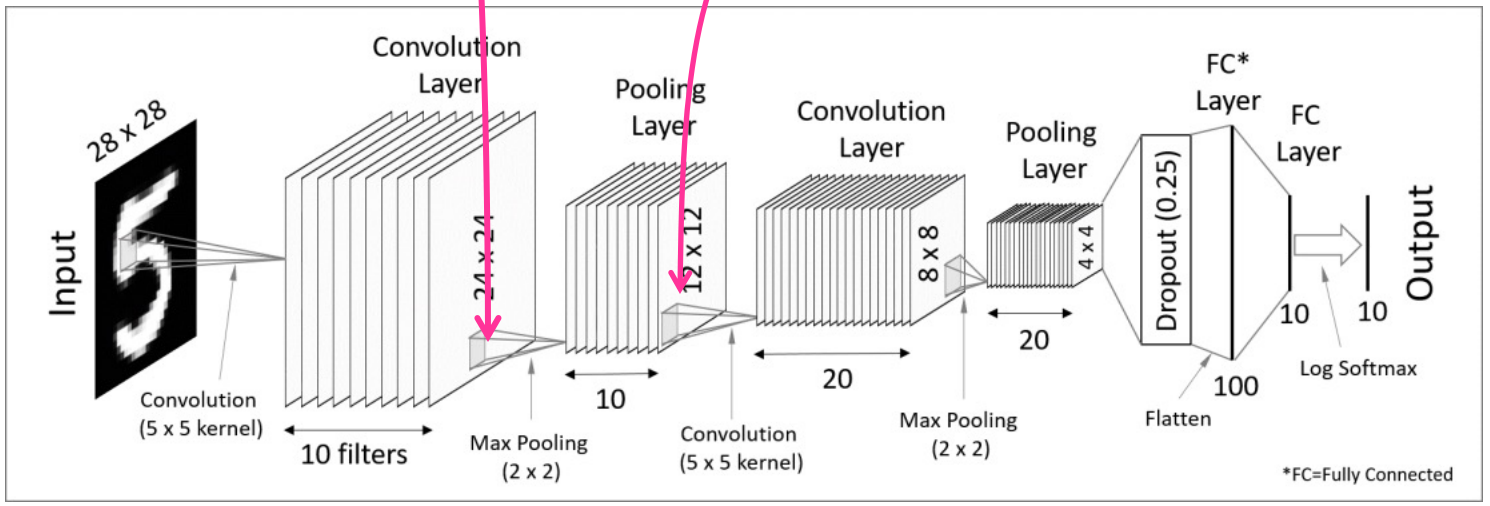
5x5

2	7	12	7	2
7	31	52	31	7
12	52	127	52	12
7	31	52	31	7
2	7	12	7	2

7x7

1	1	2	2	2	1	1
1	3	4	5	4	3	1
2	4	7	8	7	4	2
2	5	8	10	8	5	2
2	4	7	8	7	4	2
1	3	4	5	4	3	1
1	1	2	2	2	1	1

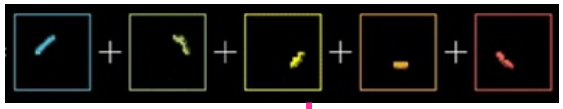


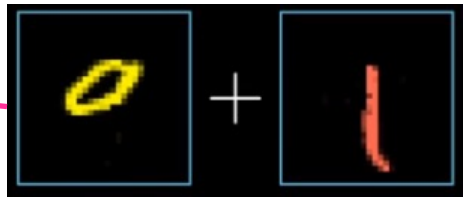


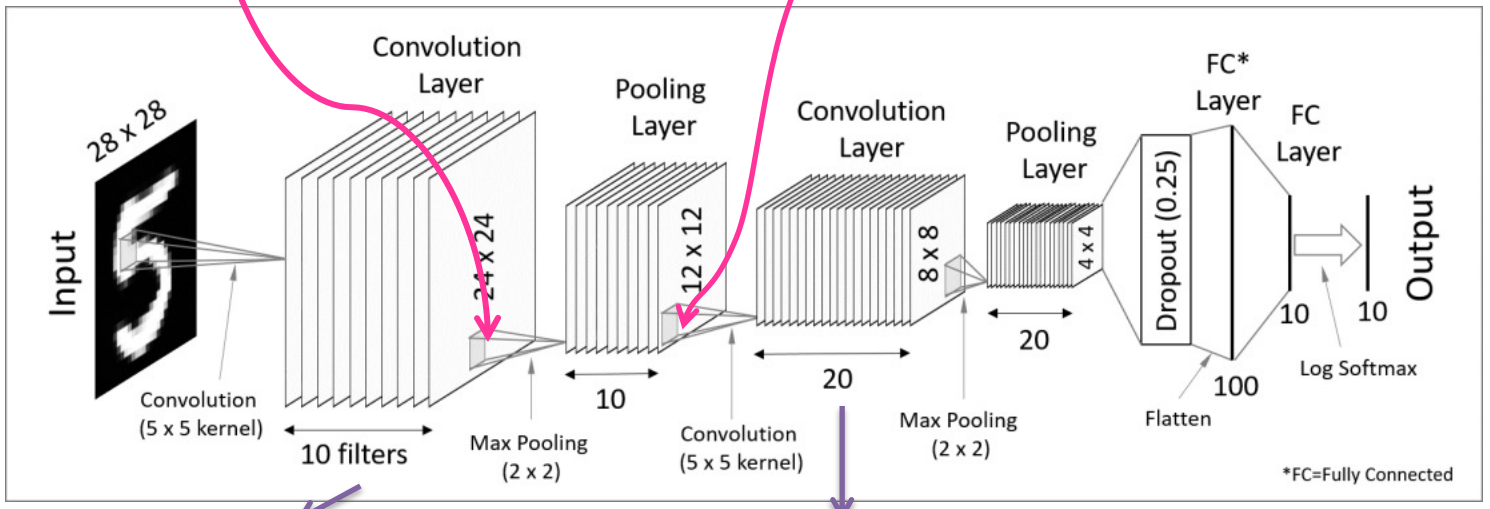
Design choices : Filter size, # of filters

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14



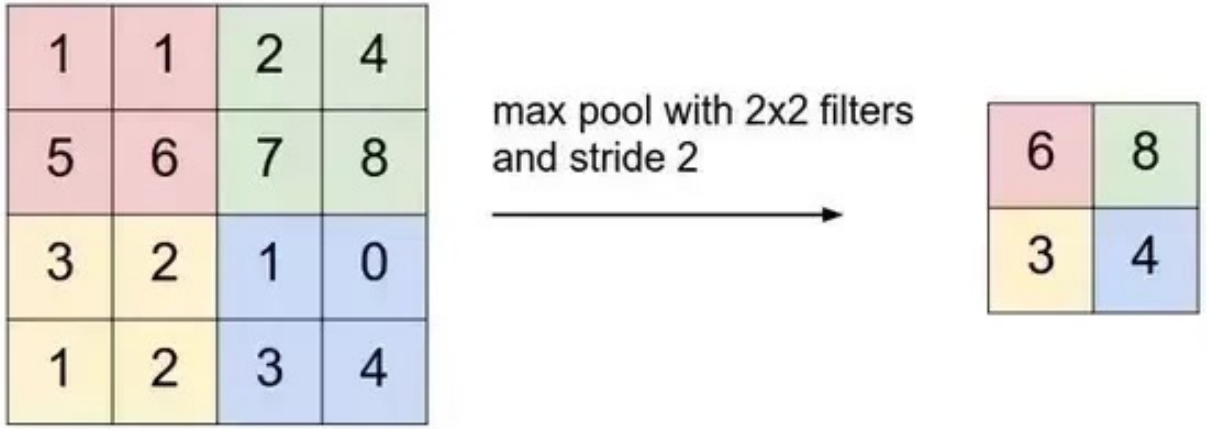




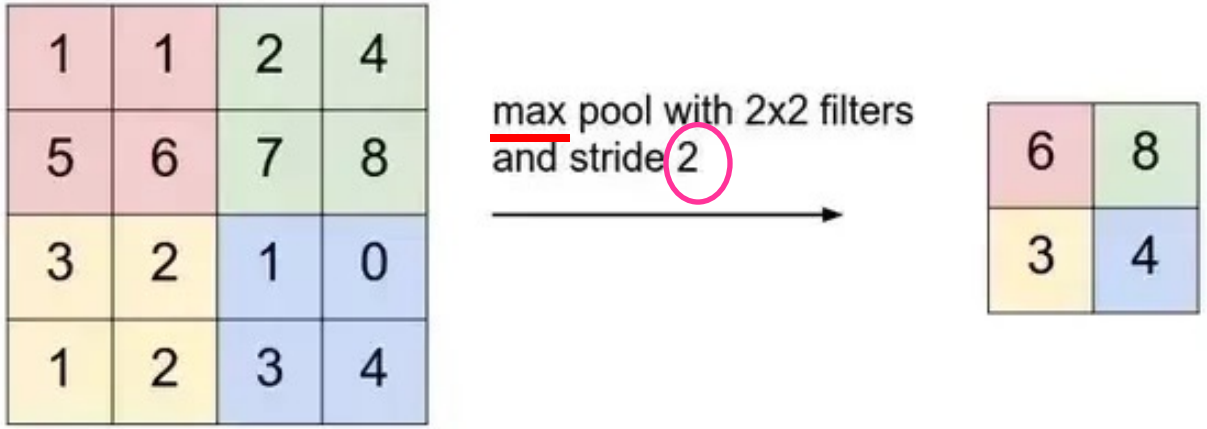
~ Alphabets

~ Words

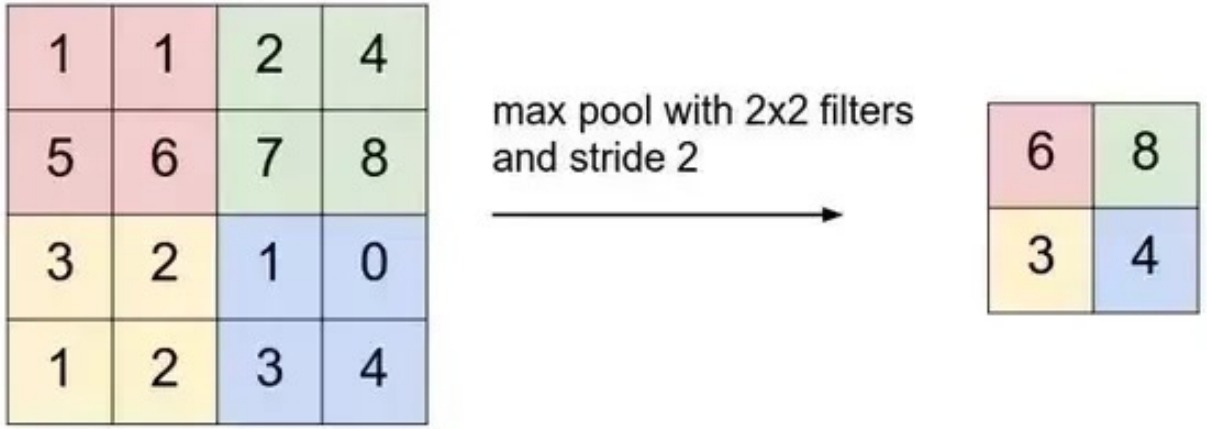
Pooling Layer



Pooling Layer

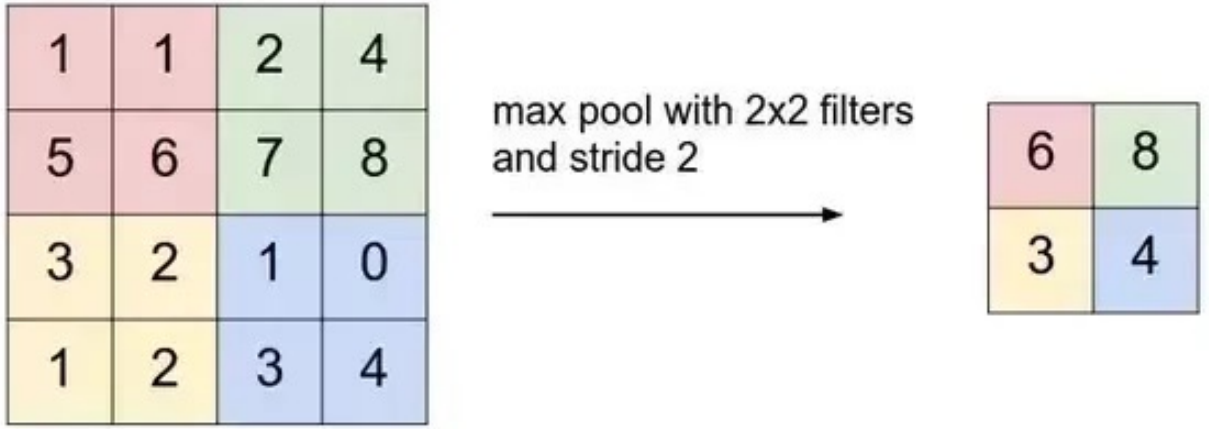


Pooling Layer

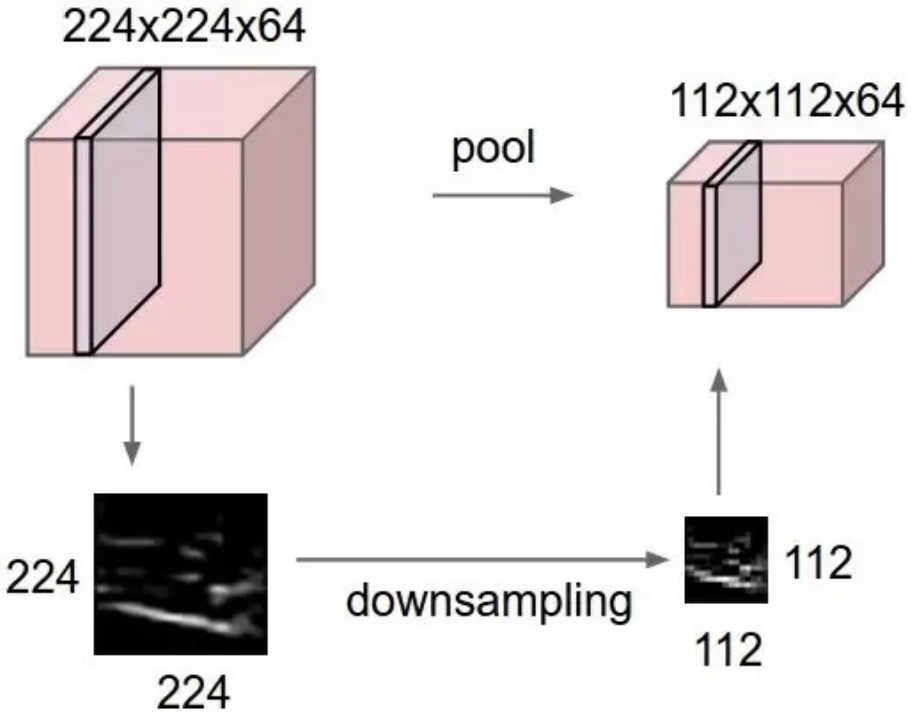


- **Motivation: We care about presence of features, not their exact location !**
- Dimensionality Reduction
- Prevents overfitting

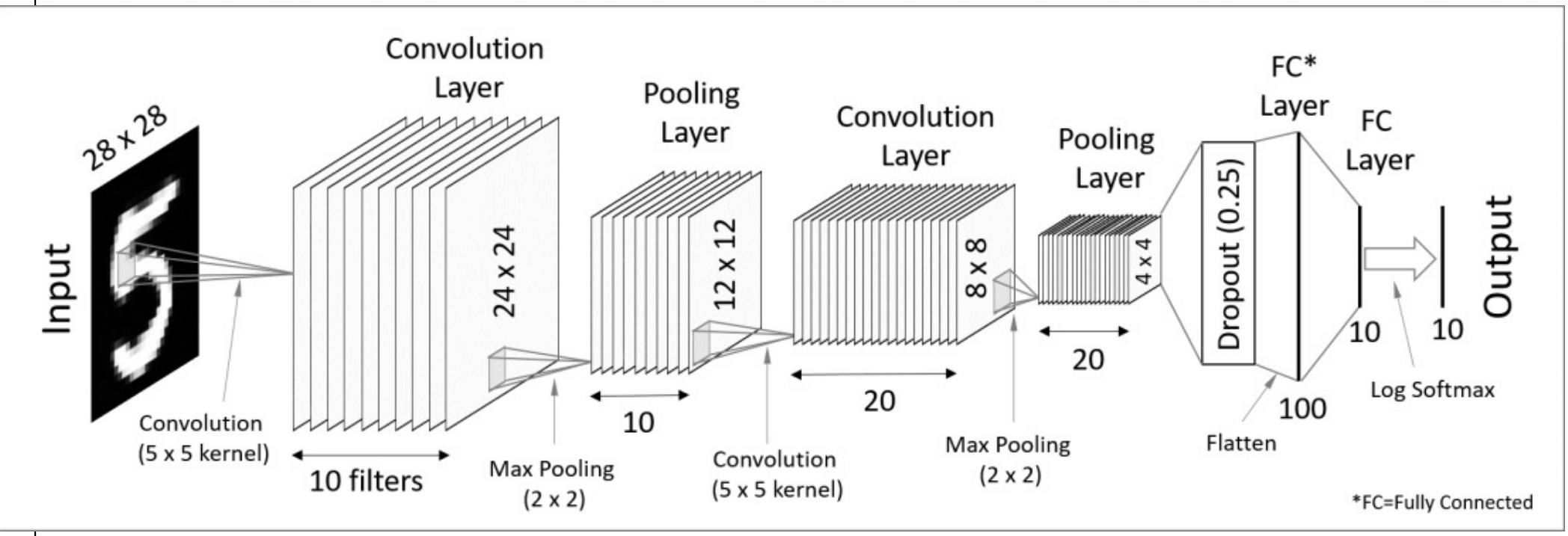
Pooling Layer



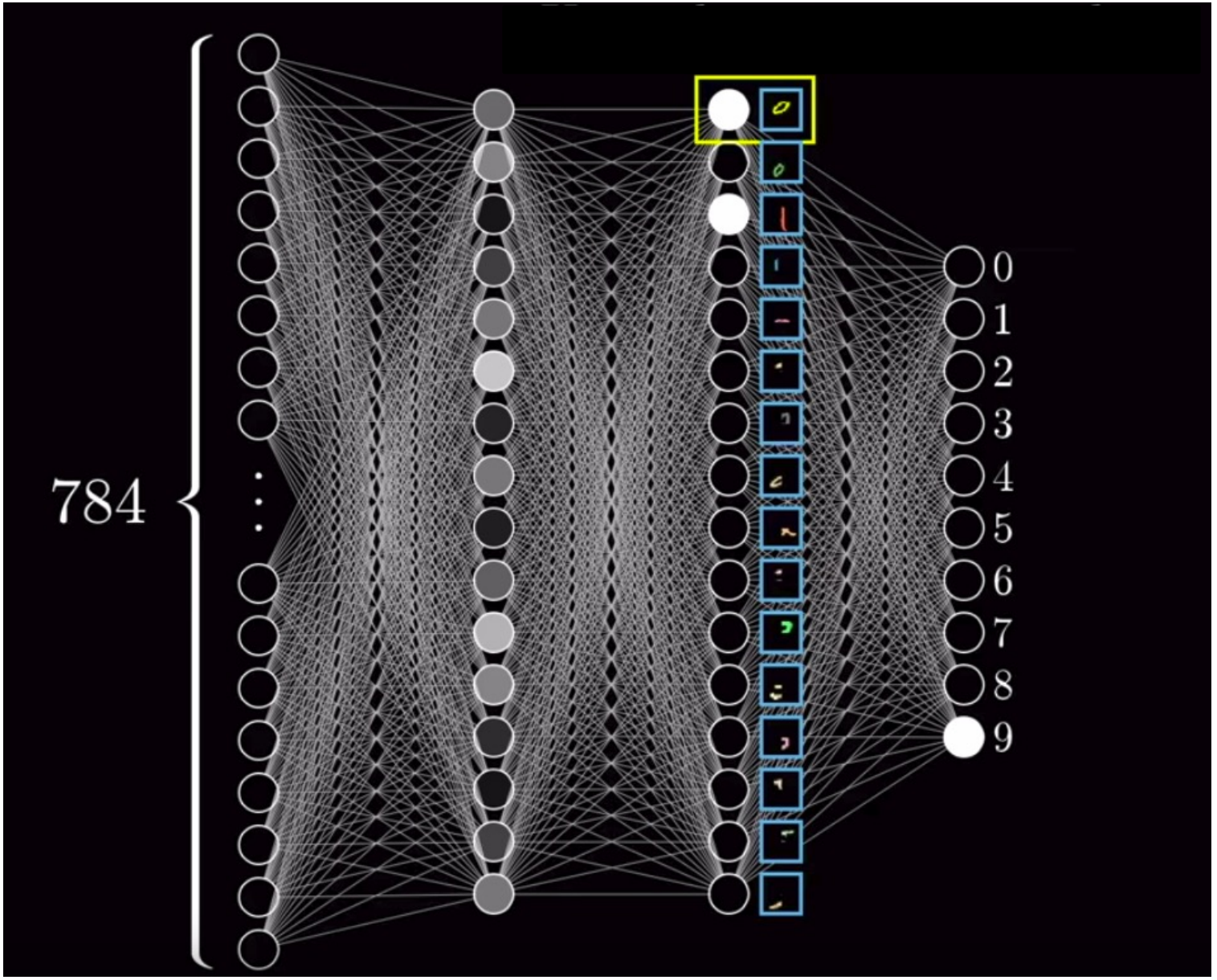
- **Motivation: We care about presence of features, not their exact location !**
- Dimensionality Reduction
- Prevents overfitting

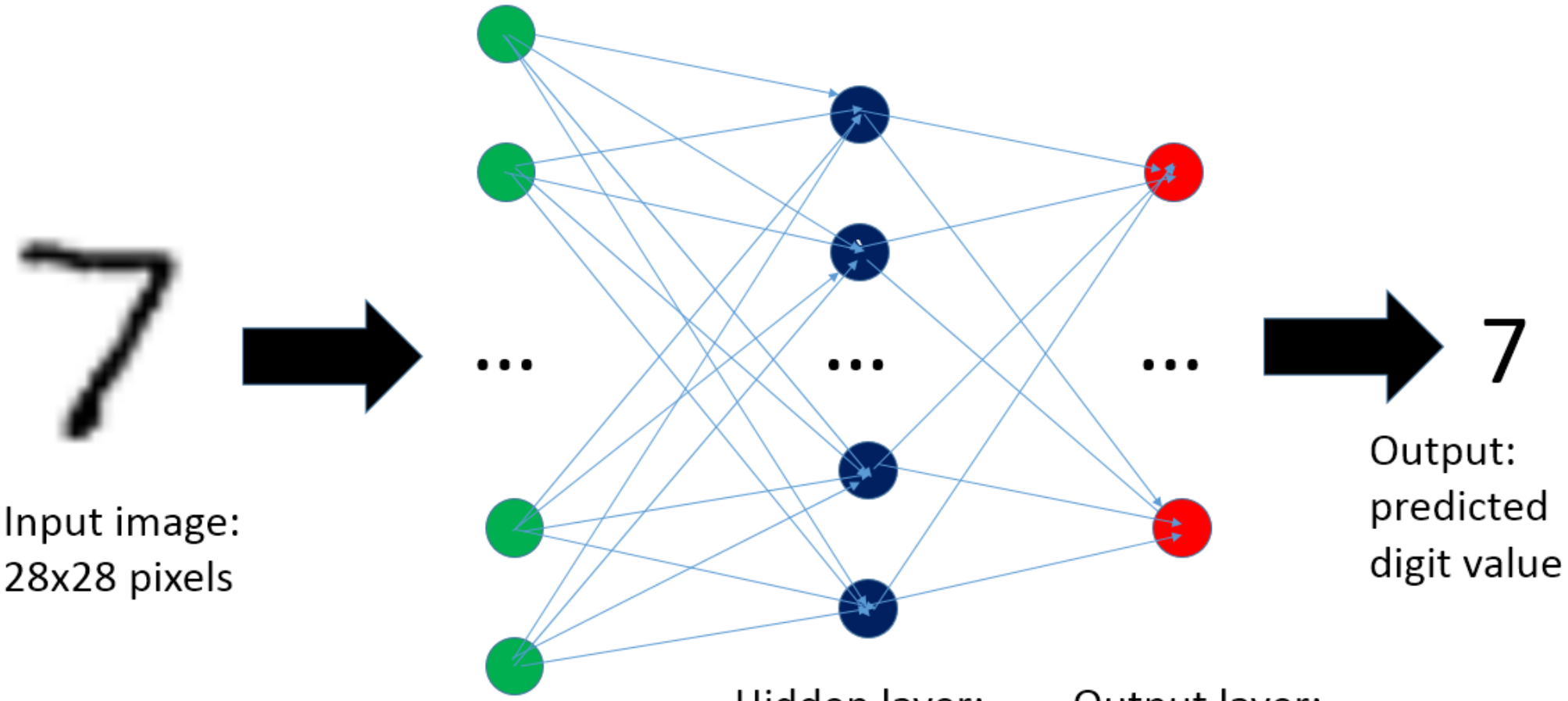


Fully-connected layers

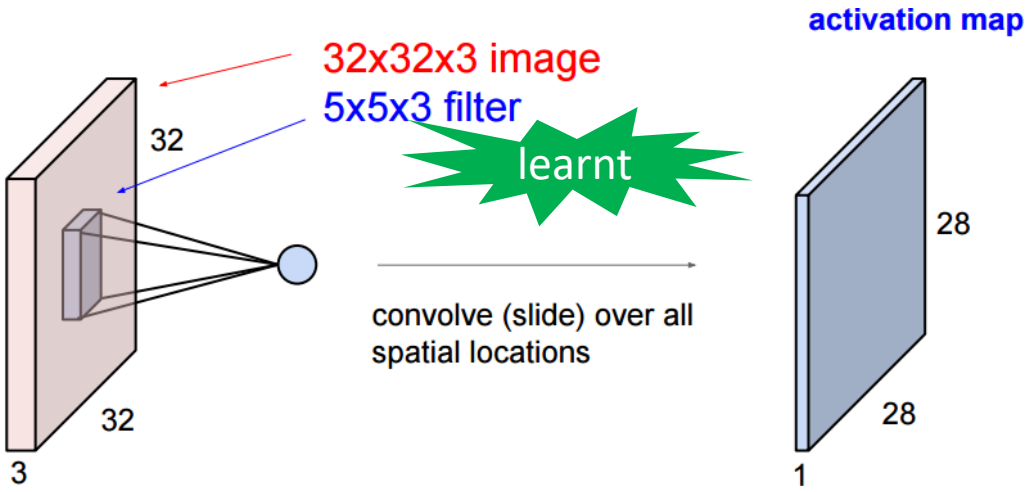


Fully Connected Layers

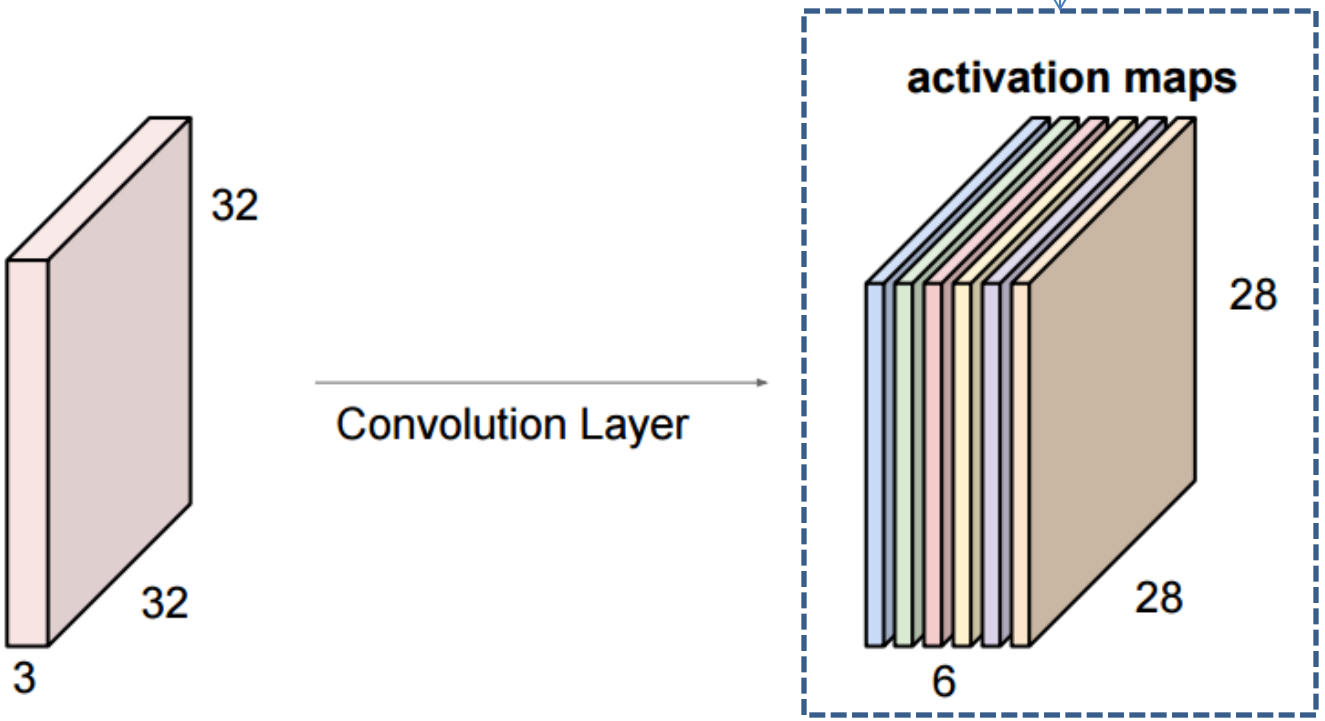



Input image:
28x28 pixels
Input layer:
784 neurons,
one per pixel
Hidden layer:
100 neurons
Output layer:
10 neurons
7
Output:
predicted
digit value

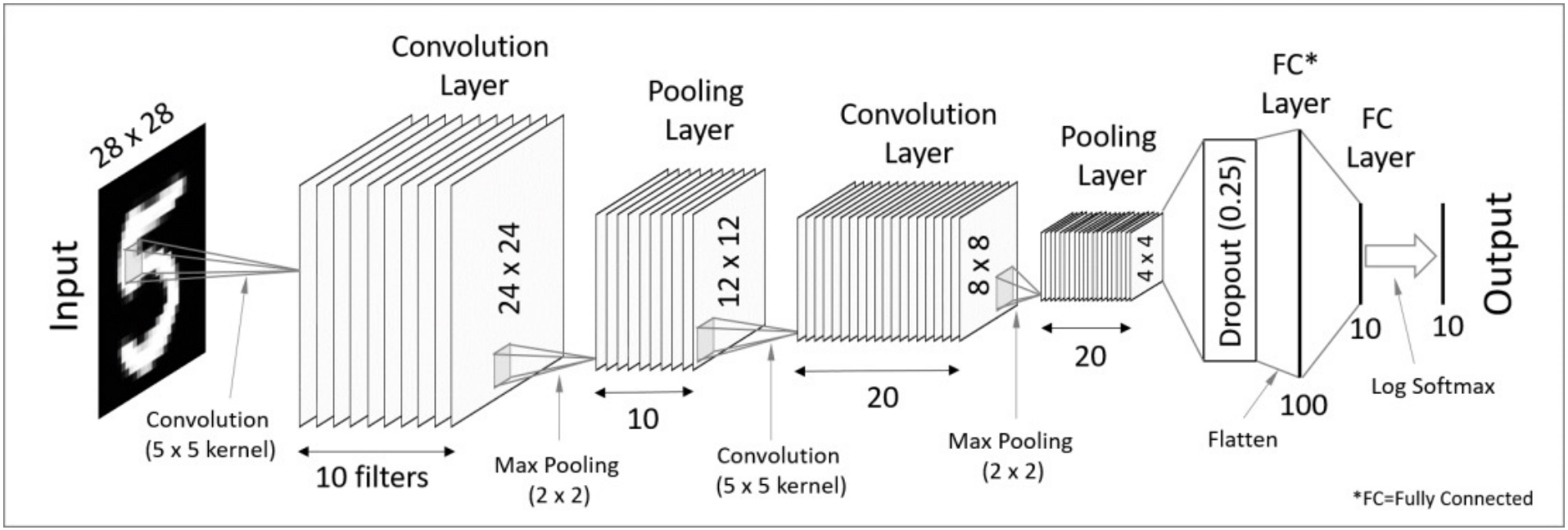
Convolution Layer



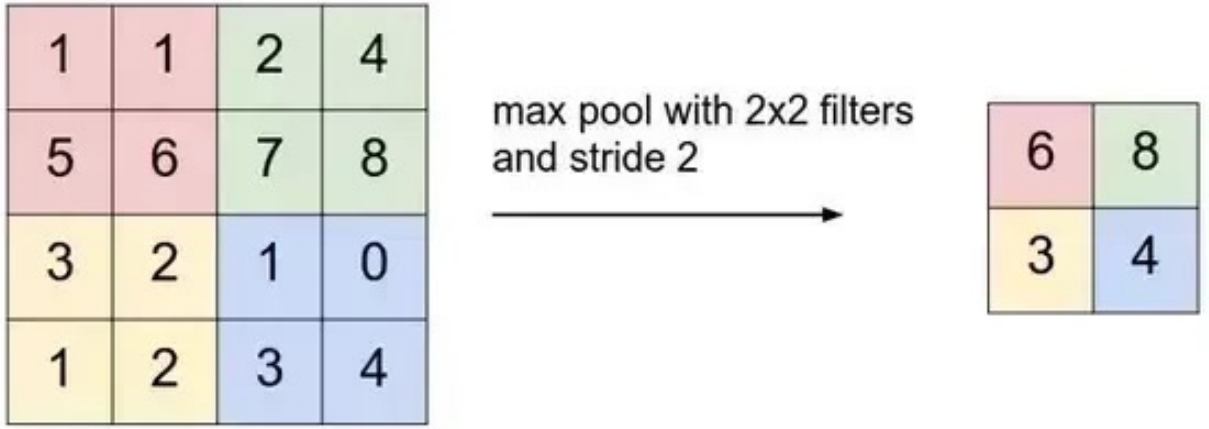
Input to next layer



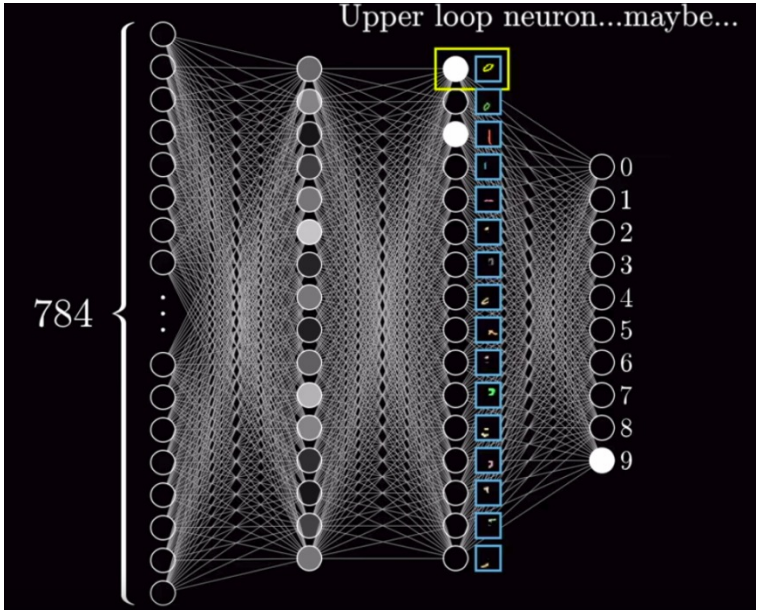
Convolutional Neural Network



Pooling Layer



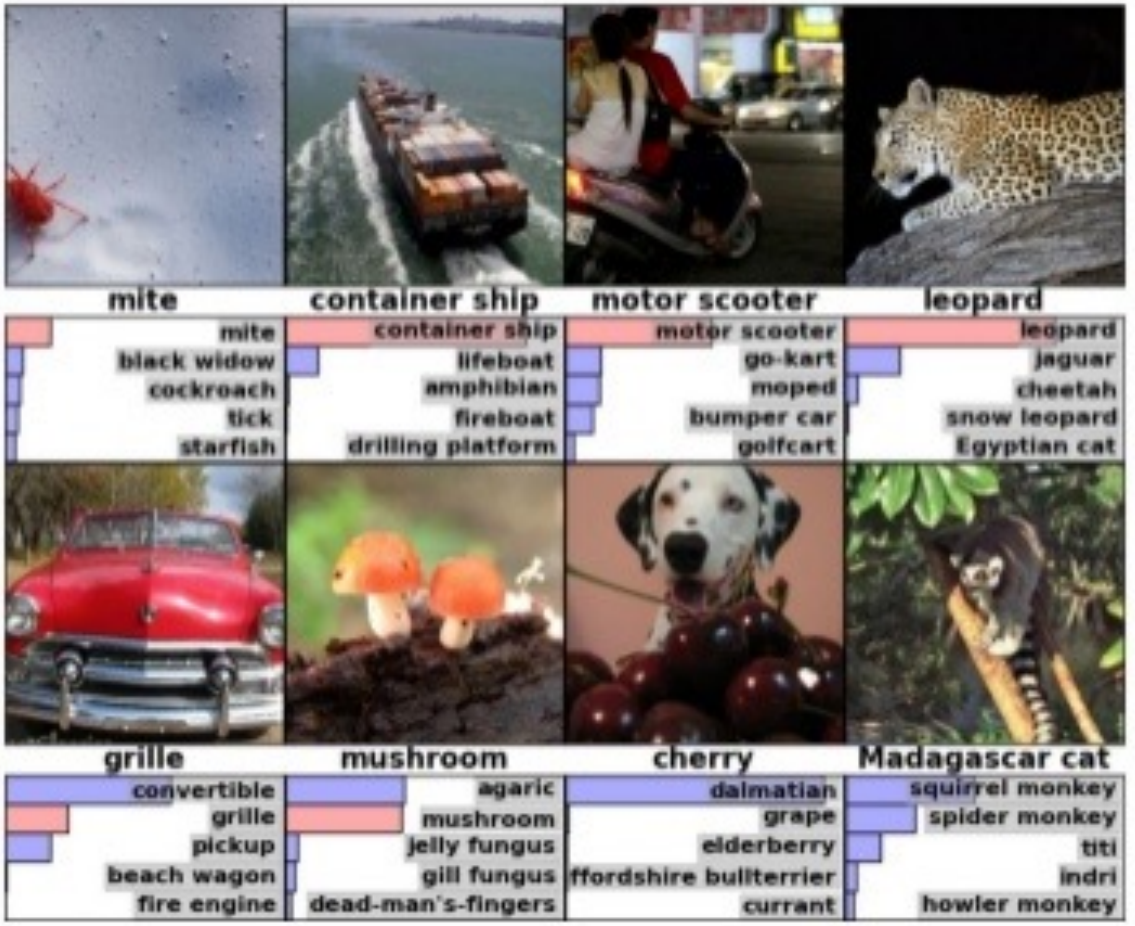
Fully-connected Layer



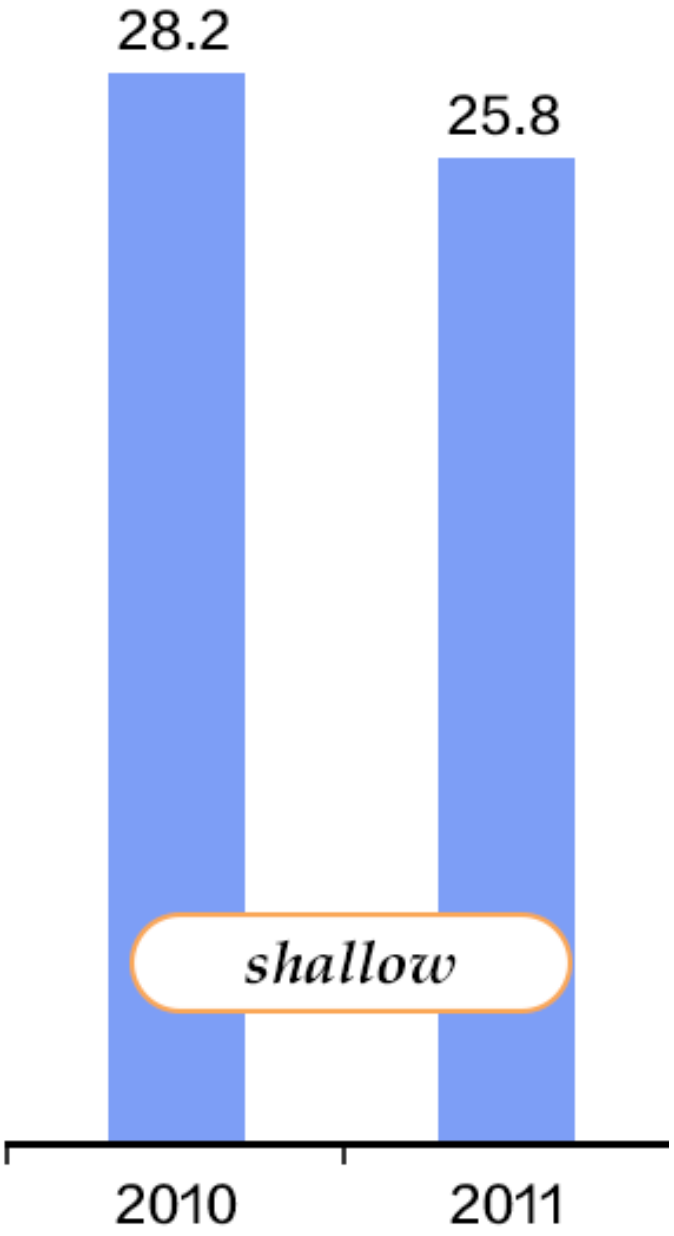
ImageNet Challenge

IMAGENET

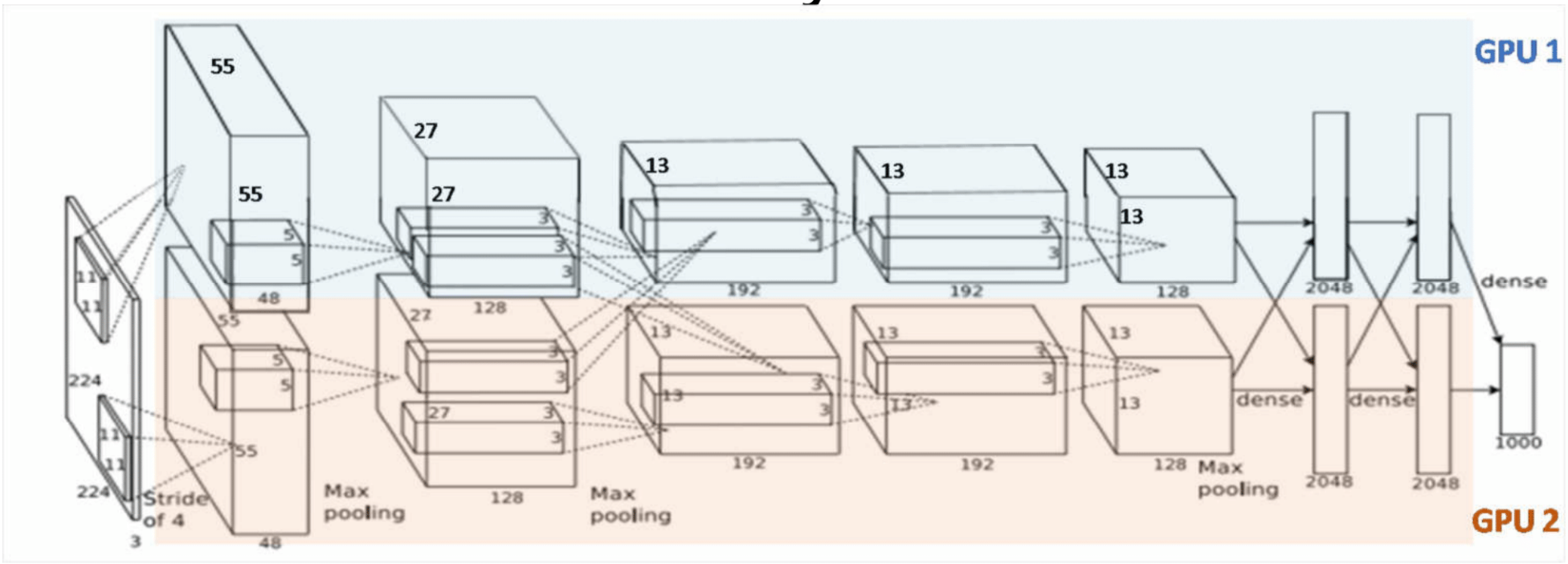
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



Top-5 misclassification error rate



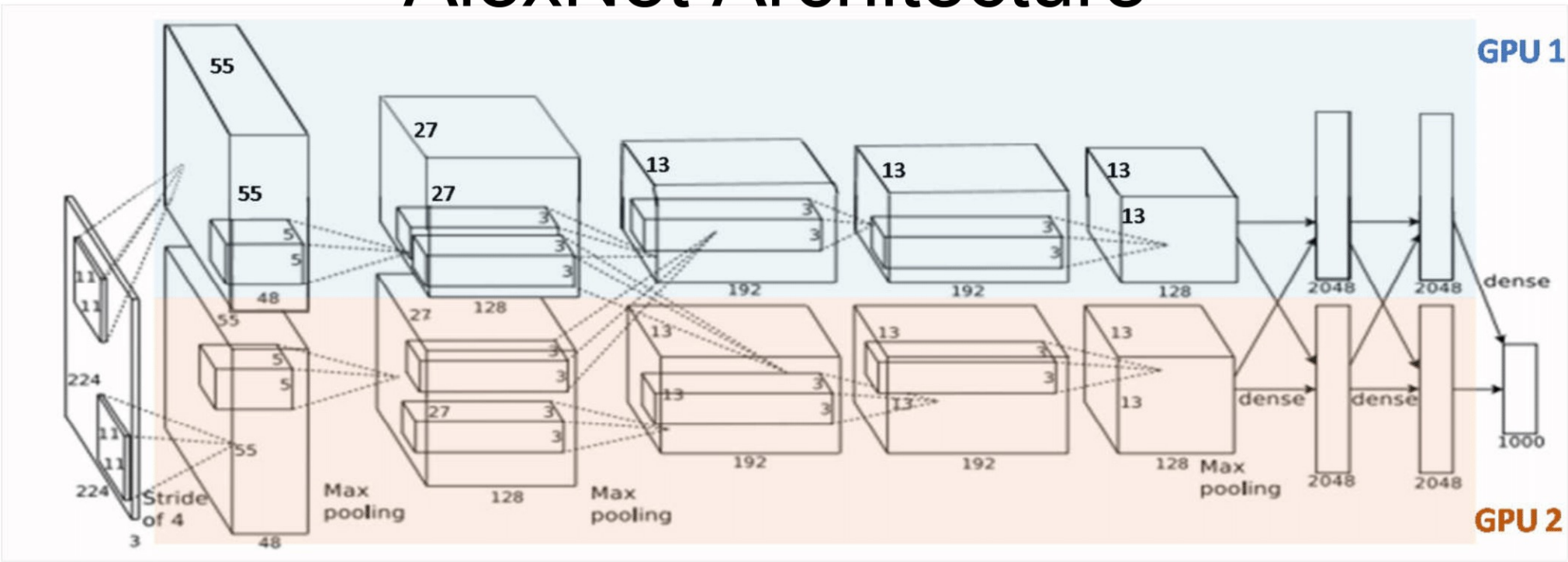
Case Study: AlexNet

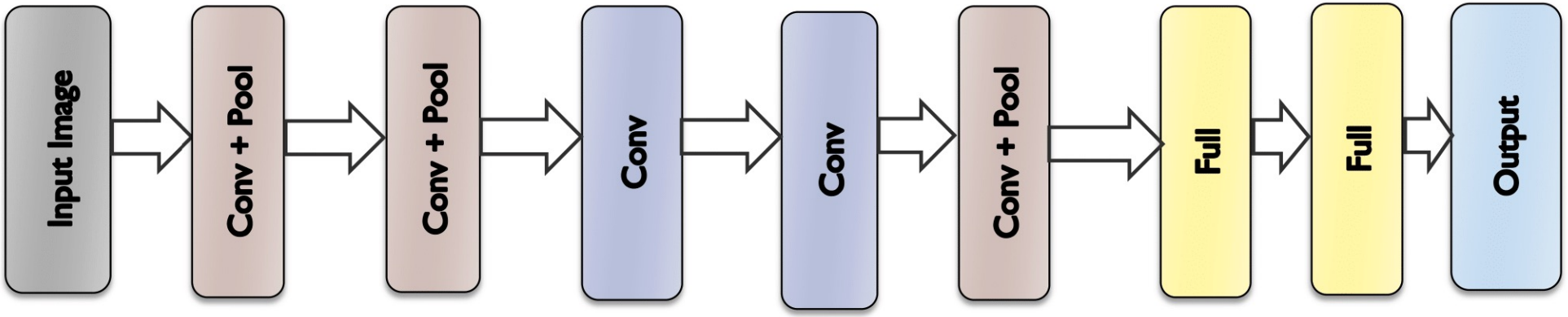


- Winner of ImageNet LSVRC-2010.
- Trained over 1.2M images using SGD with regularization.
- Deep architecture (60M parameters.)
- Optimized GPU implementation (cuda-convnet)

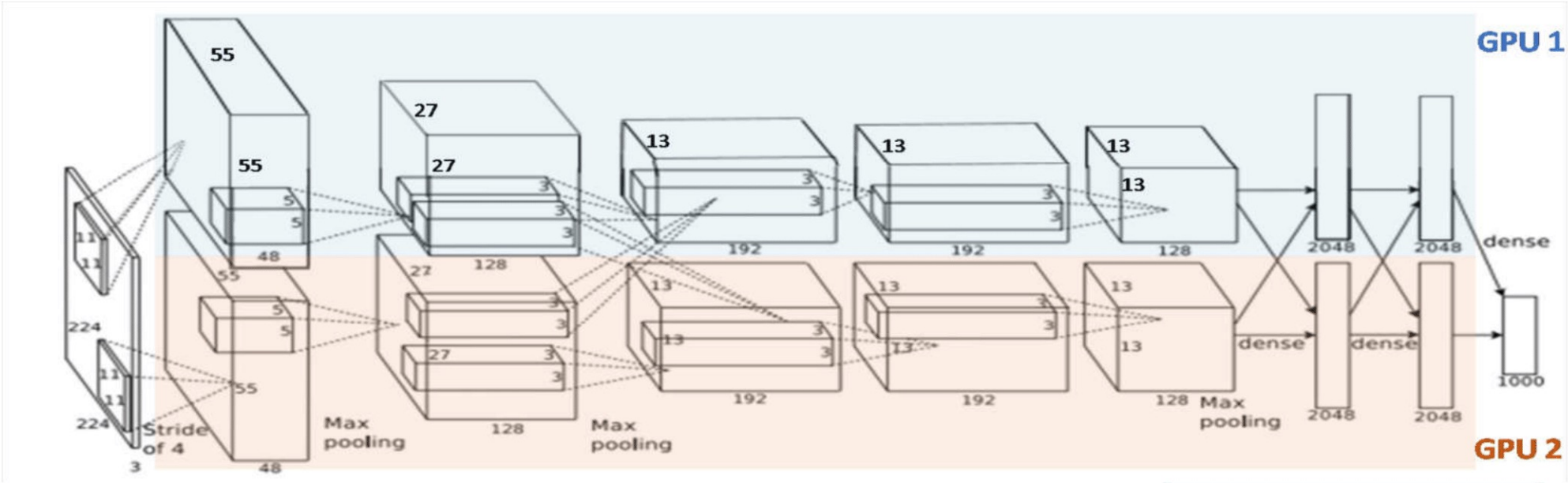
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *NIPS* 2012.

AlexNet Architecture





AlexNet Architecture

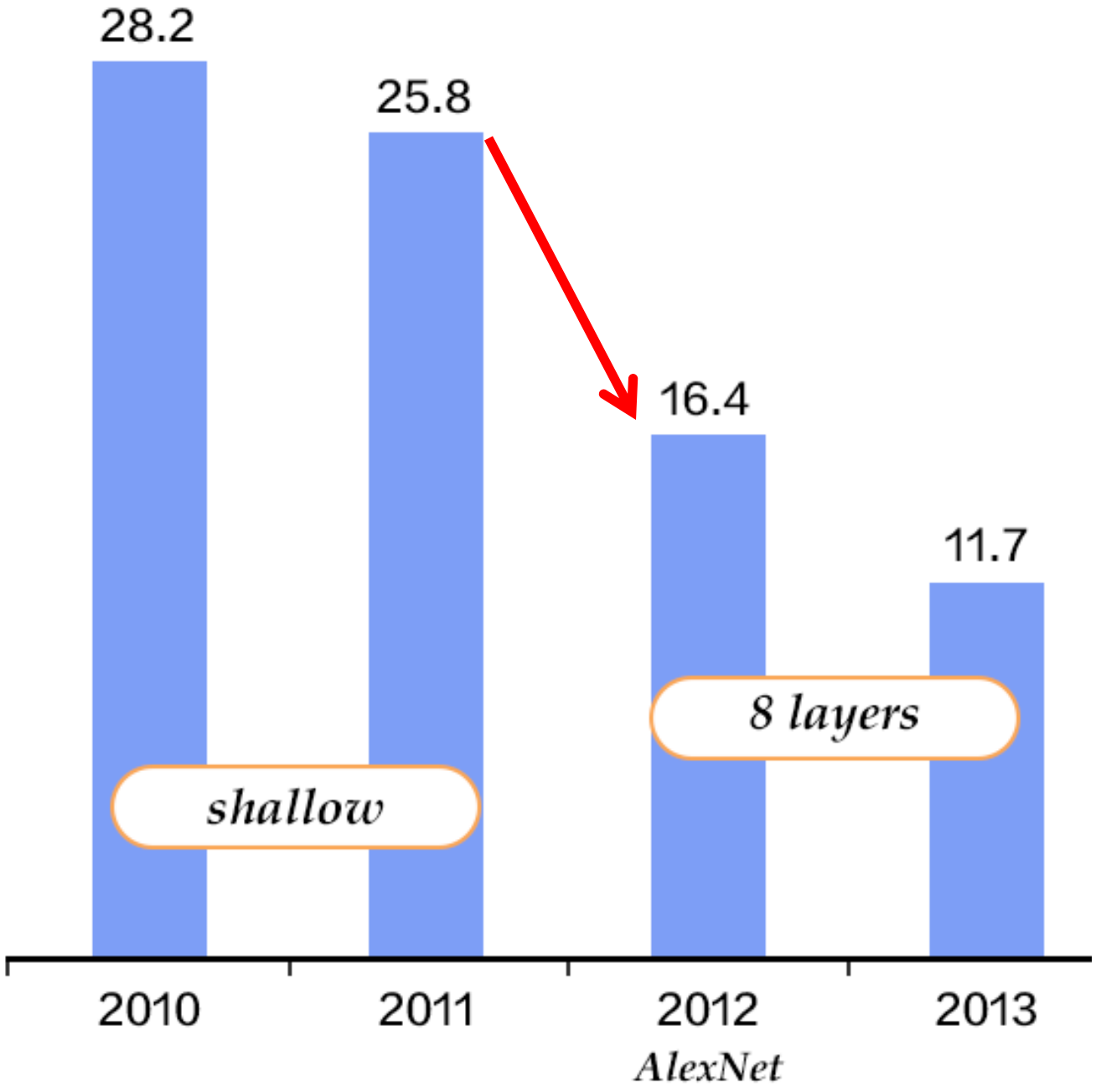


About 57 M parameters are in the fully connected layers

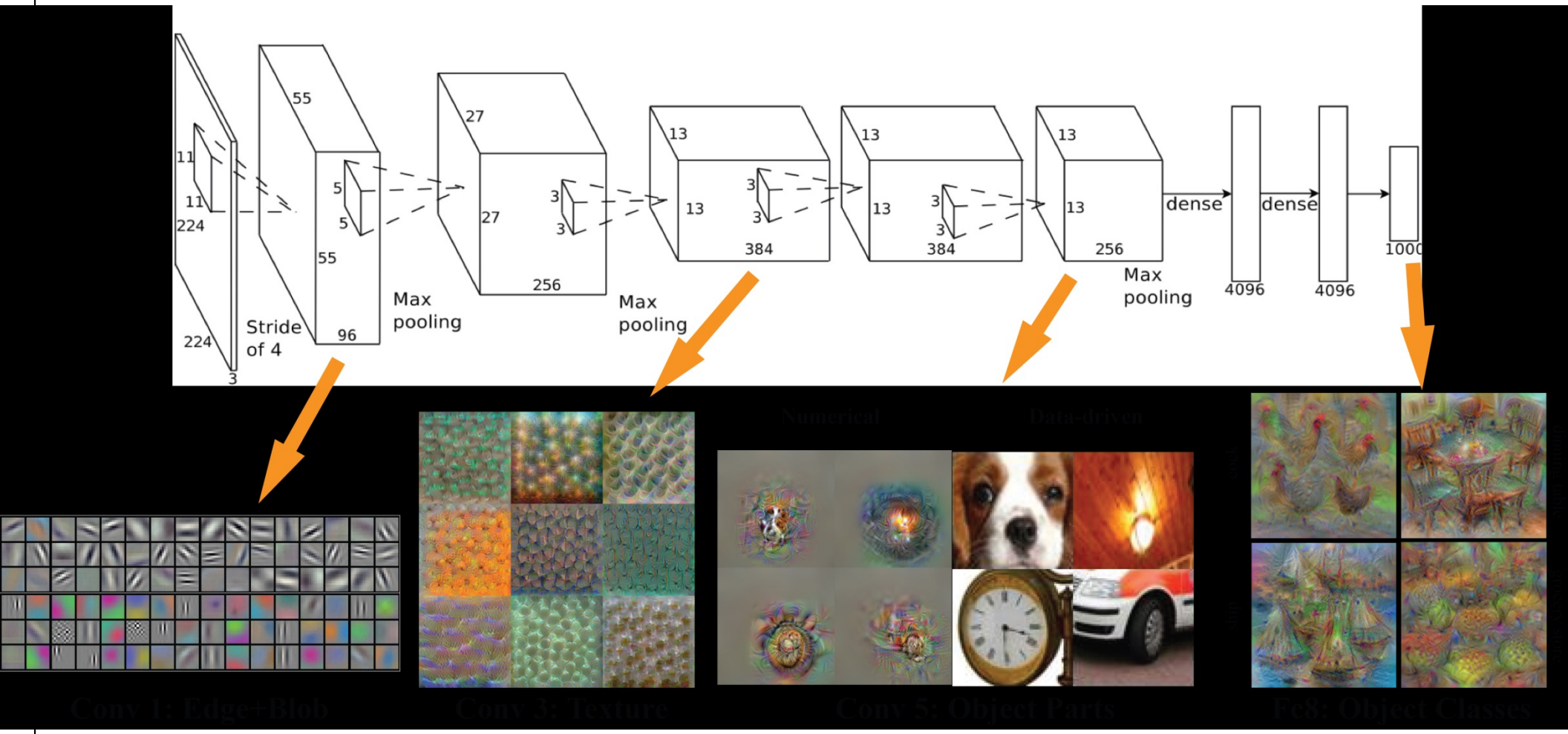
									Total
Parameters :	[(11 x 11 x 3) + 1] x 96 = 35 K	[5 x 5 x 48] x 256 = 307 K	[3 x 3 x 256] x 384 = 884 K	663 K	442 K	37 M	16 M	4 M	60 M
Neurons :	253,440	27 x 27 x 256 = 186,624	13 x 13 x 384 = 64,896	13 x 13 x 384 = 64,896	13 x 13 x 256 = 43,264	4096	4096	1000	0.63 M

- Convolutional layers cumulatively contain about 90-95% of computation, only about 5% of the parameters
- Fully-connected layers contain about 95% of parameters.

Top-5 misclassification error rate

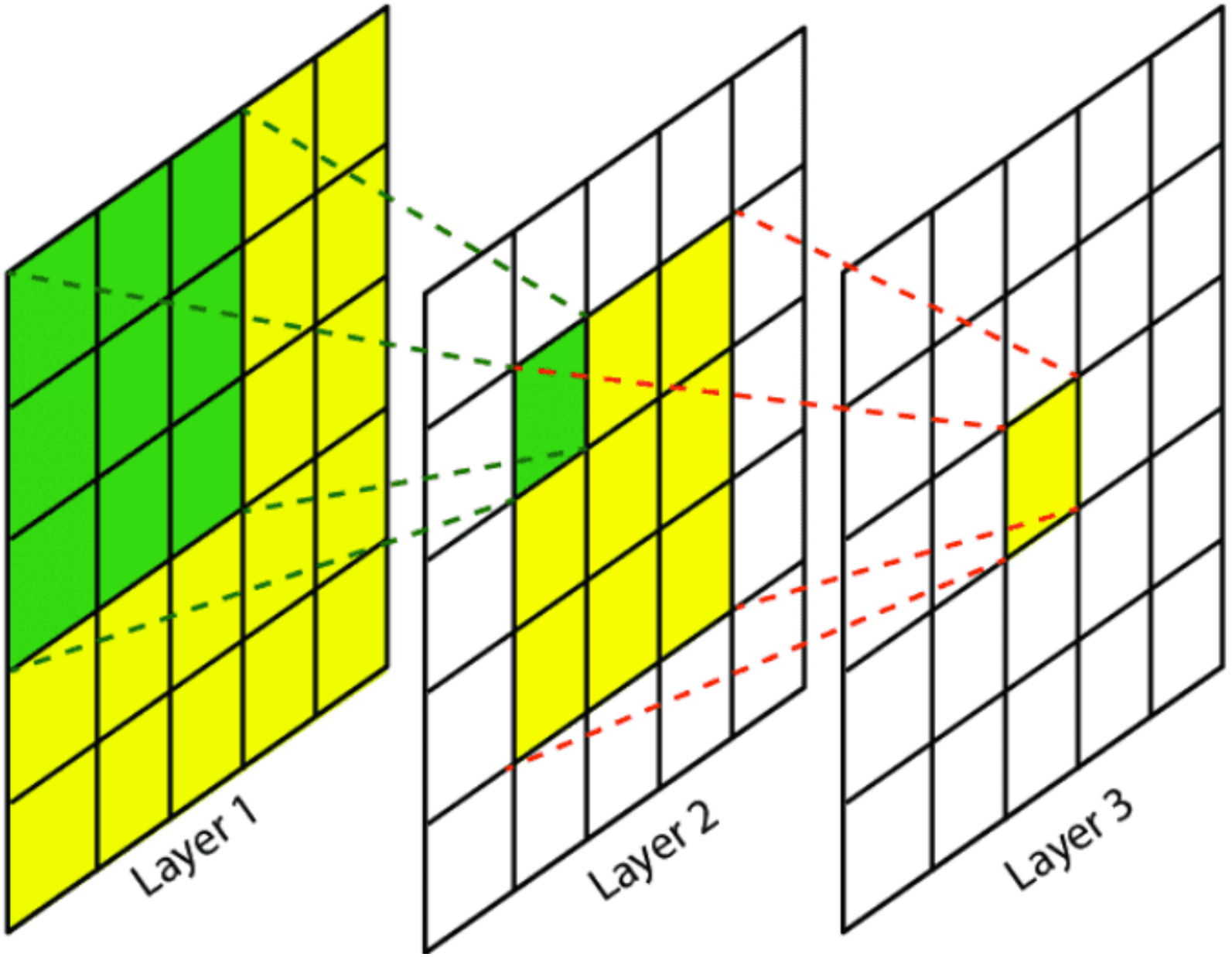


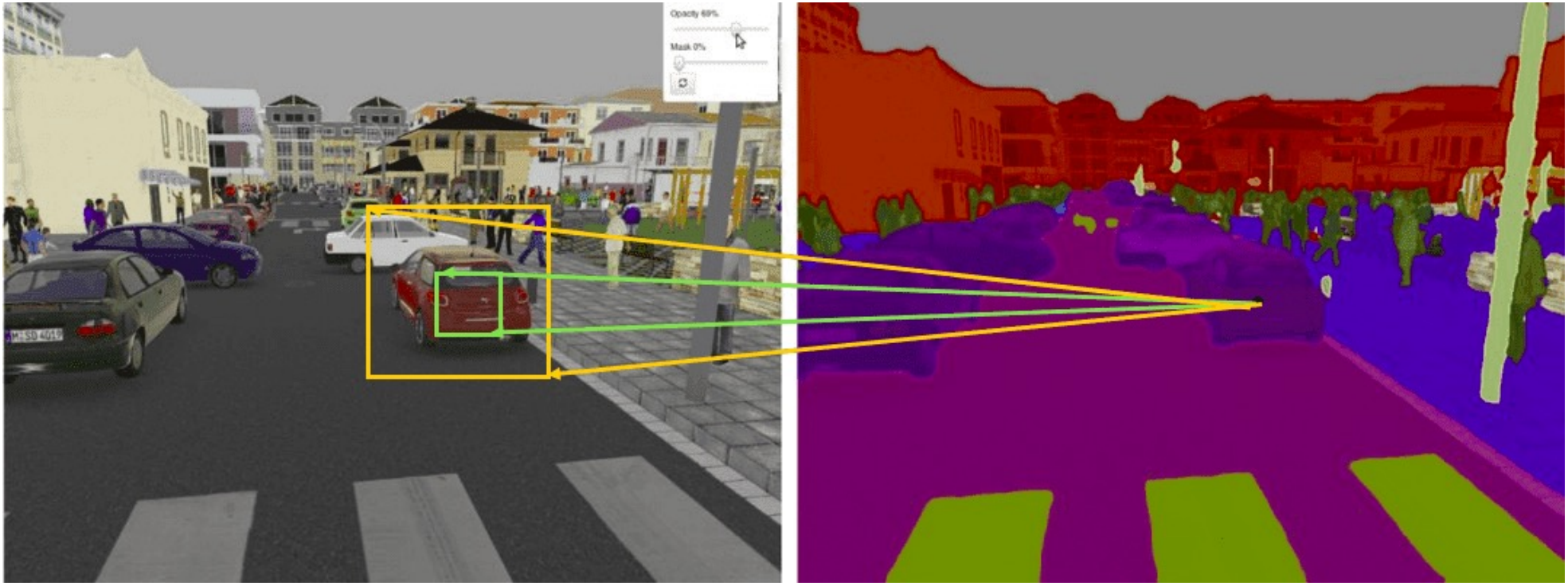
Filters learnt by AlexNet



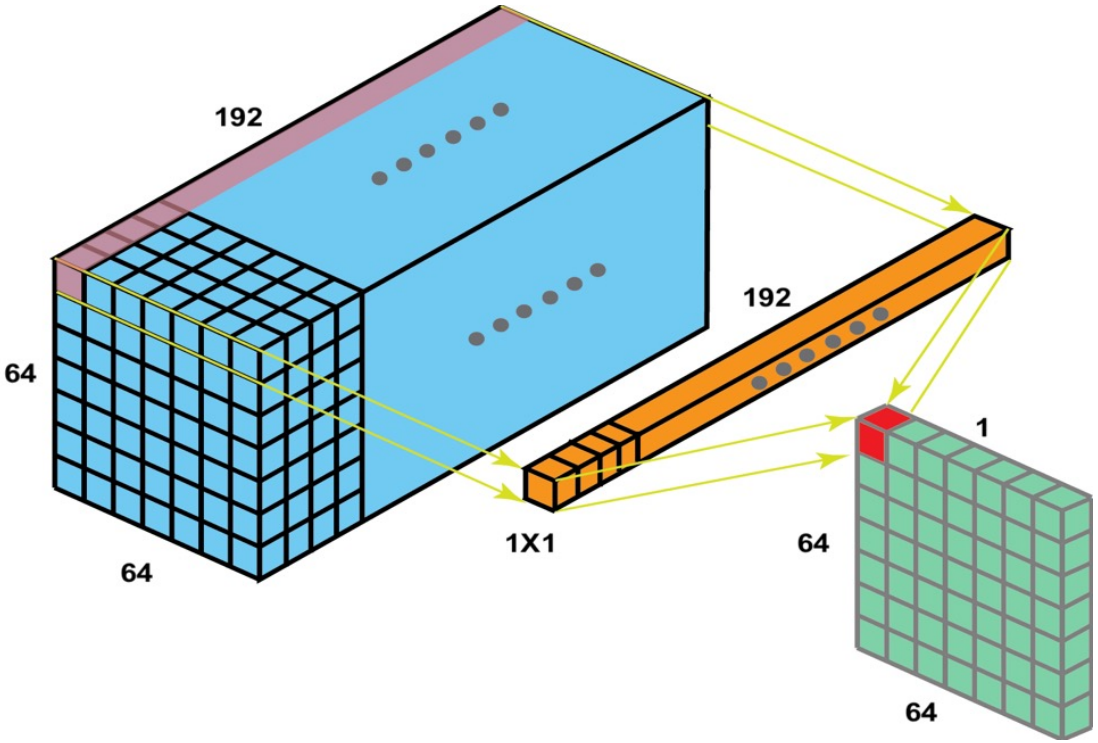
https://www.cc.gatech.edu/~hays/compvision/proj6/deepNetVis.png

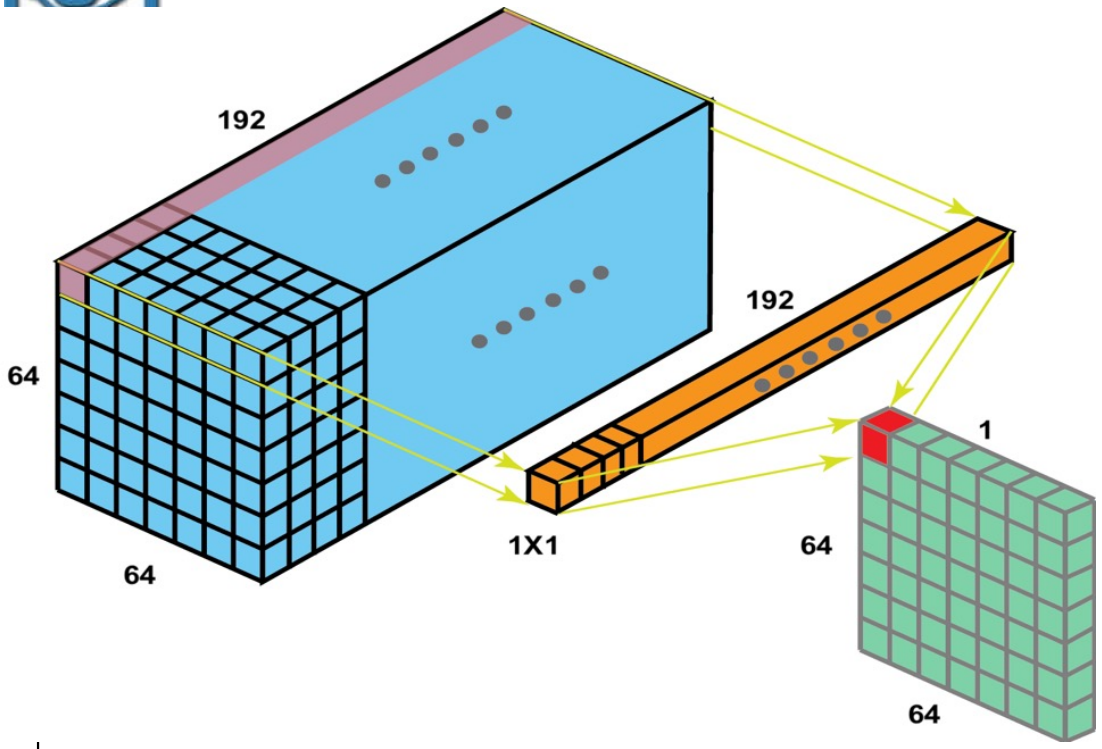
Receptive Field

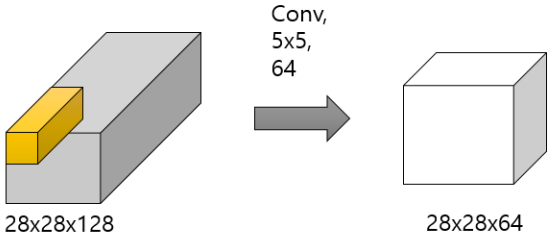



Opacity 69%
Mask 0%

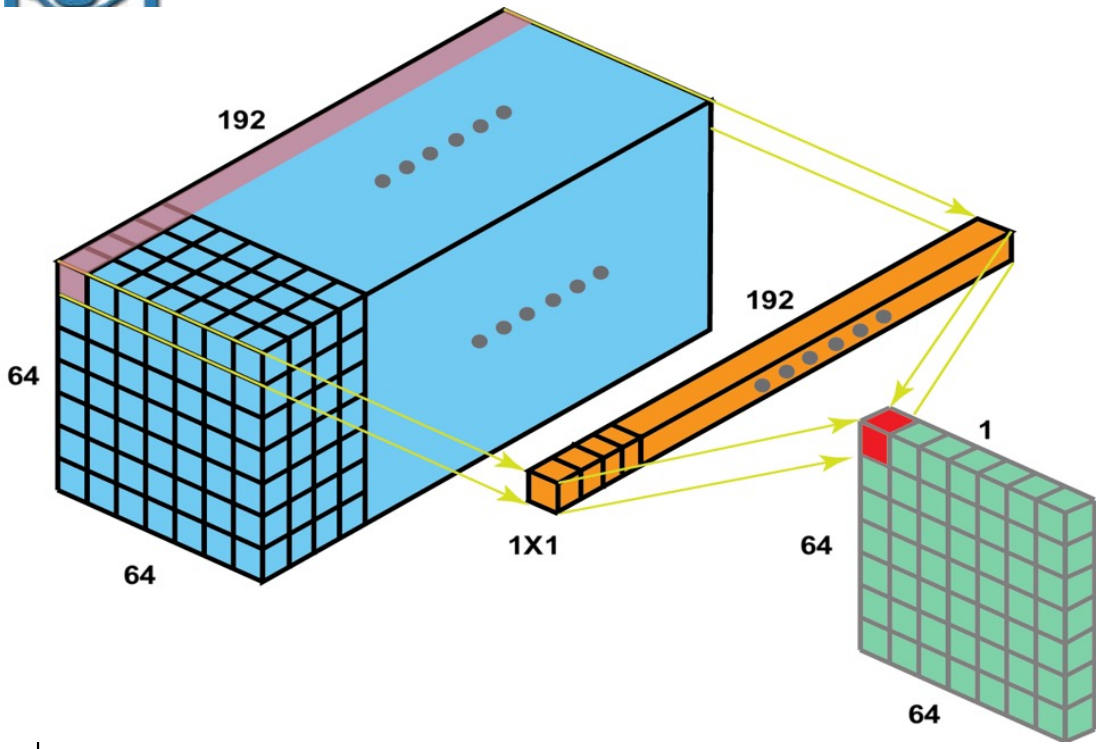
1x1 Convolutions

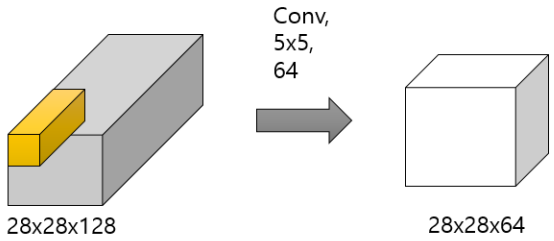




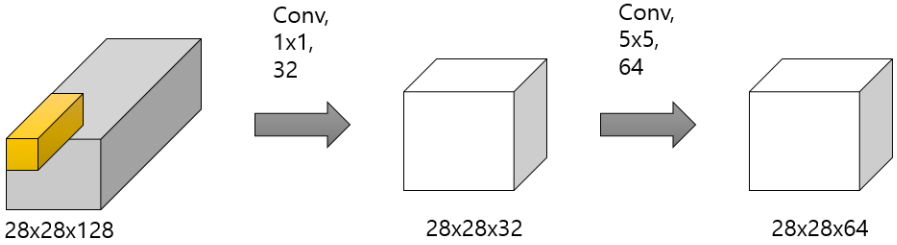


#params = 28x28x64x5x5x128 = 160M





#params = 28x28x64x5x5x128 = 160M



#params = 28x28x32x128x1x1 = 4.8M

#params = 28x28x64x5x5x32 = 40M

#total = 44.8M

Dilated Convolutions

https://github.com/vdumoulin/conv_arithmetic

Resources

- https://www.youtube.com/@far1din619/videos
- What is a convolution: https://www.youtube.com/watch?v=KuXjwB4LzSA
- https://github.com/vdumoulin/conv_arithmetic
- BP in CNN: https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf