



---

# Introduction to Convolutional Neural Networks / Deep Learning

Ravi Kiran Sarvadevabhatla

<https://ravika.github.io/>



G. Hinton  
(U.Toronto,  
Google DeepMind)

Yann LeCun  
(Facebook AI)

Y. Bengio  
(MILA, Montreal)



## Hinton's Prayer

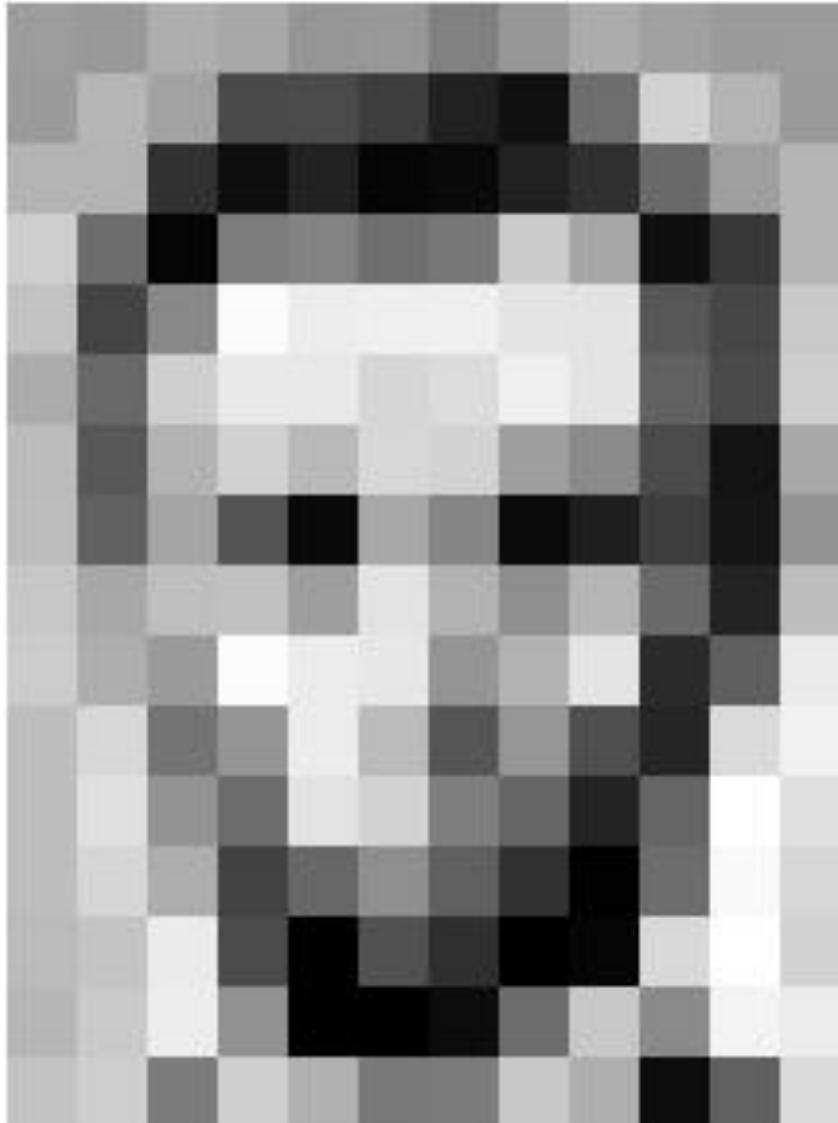
Our father who art in  $n$ -dimensions,  
    hallowed be thy backprop,  
    thy loss be minimized,  
    thy gradients unvanished,  
        on earth as it is in Euclidean space.  
Give us this day our daily hyperparameters,  
    and forgive us our large learning rates,  
    as we forgive those whose parameters diverge,  
    and lead us not into discrete optimization,  
        but deliver us from local optima.  
For thine are the dimensions,  
    and the GPUs, and the glory,  
    for ever and ever. Dropout.

From the *Gospel According to Christopher*, circa 2016



---

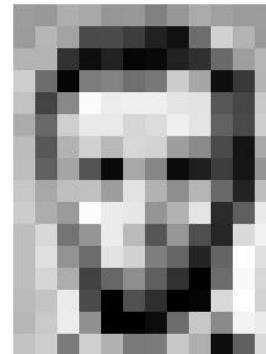
# Image Representation



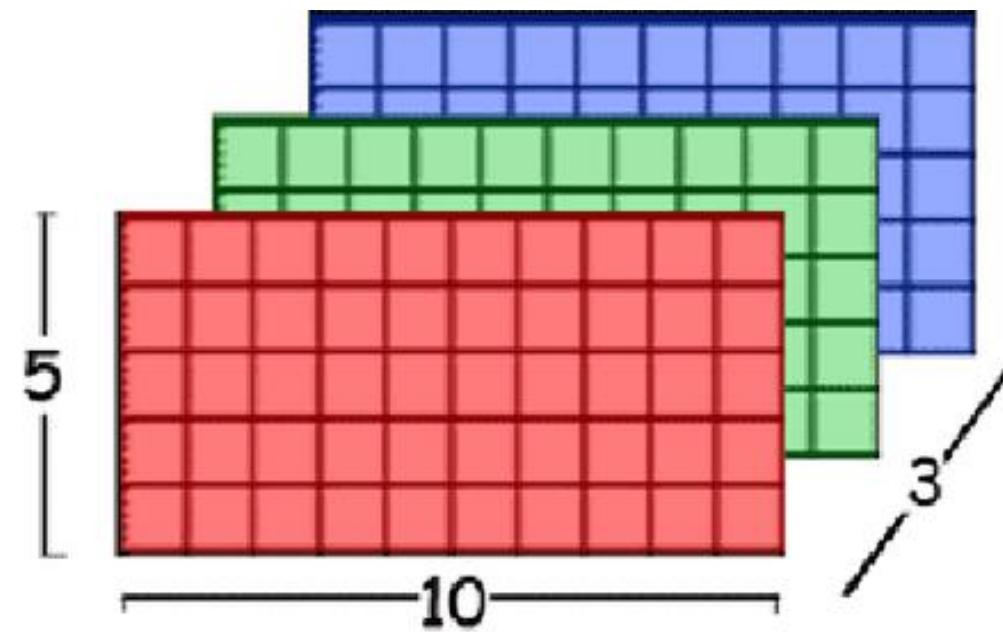
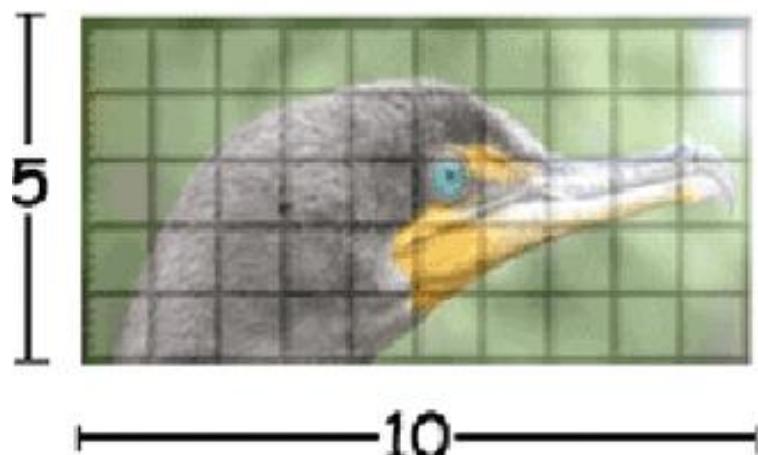
157	153	174	168	150	152	129	151	172	161	165	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	45	105	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	156	252	236	231	149	178	228	43	95	234
160	216	116	149	236	187	85	150	79	38	218	241
160	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218



# Image Representation



157	153	174	168	160	152	129	151	172	161	155	156
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	54	6	10	53	48	106	159	181
206	109	6	124	131	111	120	204	168	15	56	180
194	58	137	251	237	239	230	228	227	87	71	201
172	106	207	233	233	214	220	239	228	96	74	206
188	68	175	209	185	215	211	158	139	75	20	169
189	97	165	64	70	168	134	11	31	62	22	148
199	168	191	193	158	227	177	143	182	101	36	190
205	174	156	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	172	66	103	143	95	50	2	109	249	216
187	195	238	75	1	81	47	0	6	217	255	211
183	292	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	122	200	175	13	96	218





# A Case Study: Fish Classification

- Problem:
  - sort incoming fish on a conveyor belt according to species
  - Assume only two classes exist:
    - Sea Bass and Salmon





# Features

- **Features are properties of an object:**
  - Ideally representative of a specific type (i.e. class) of objects
  - Compact (memory efficient)
  - Computationally simple (CPU efficient)
  - Perceptual relevant (if trying to implement a human inspired classifier)
- => **Should pave the way to a good discrimination of different classes of objects!**



# Features

- Take a group of graphical objects
  - Possible features:
    - Shape
    - Color
    - Size
    - ...
  - Allows to group them into different classes:





# Feature Vectors

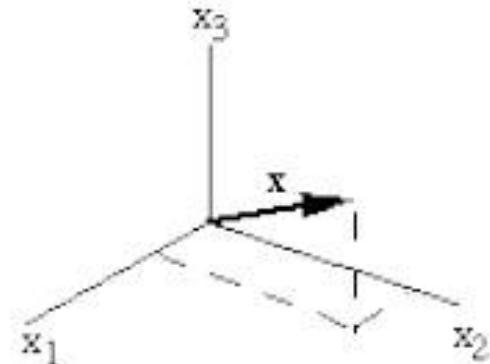
- Usually a single object can be represented using several features, e.g.

- $x_1$  = shape (e.g. nr of sides)
- $x_2$  = size (e.g. some numeric value)
- $x_3$  = color (e.g. rgb values)
- ...
- $x_d$  = some other (numeric) feature.

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

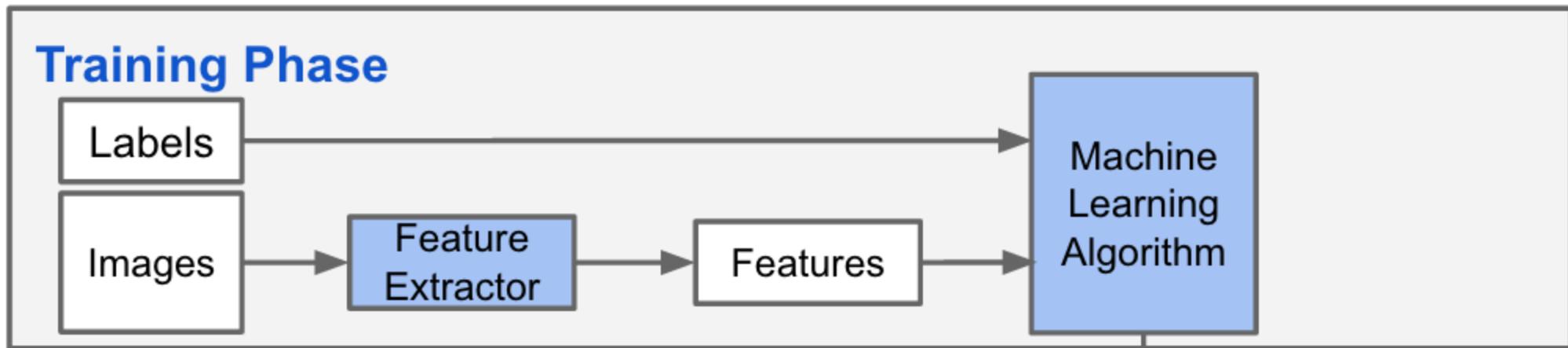


$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_d \end{bmatrix}$$



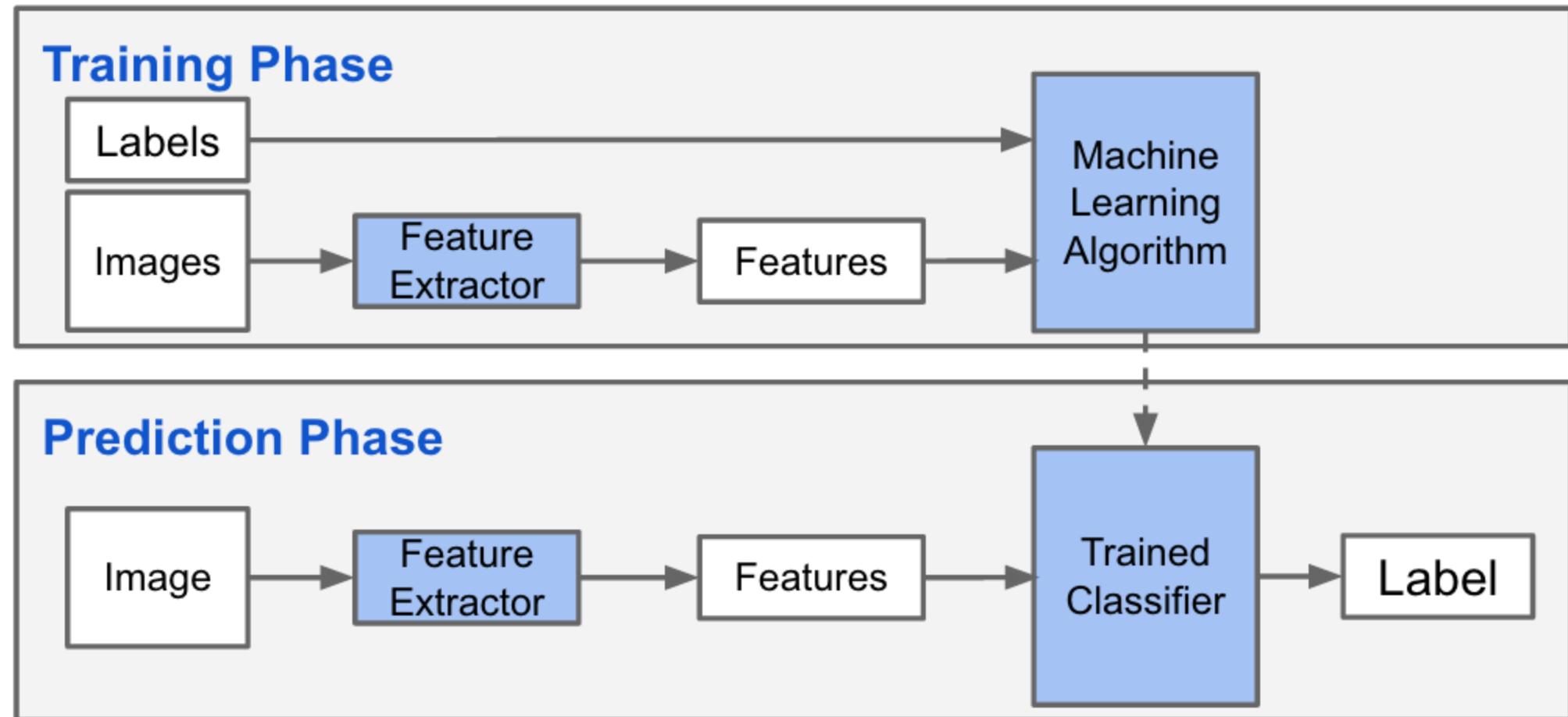


# Classification





# Classification





# Feature Vectors

- Usually a single object can be represented using several features, e.g.

- $x_1$  = shape (e.g. nr of sides)
- $x_2$  = size (e.g. some numeric value)
- $x_3$  = color (e.g. rgb values)
- ...
- $x_d$  = some other (numeric) feature.

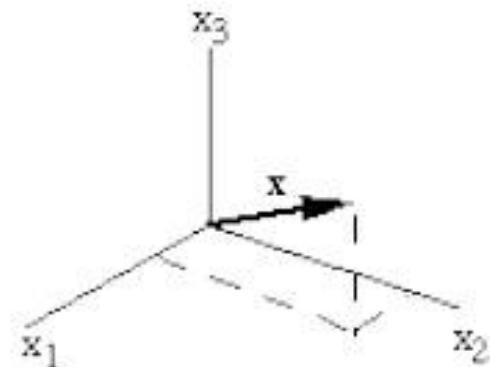
- $\mathbf{X}$  becomes a feature vector
  - $\mathbf{x}$  is a point in a d-dimensional **feature space**.



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



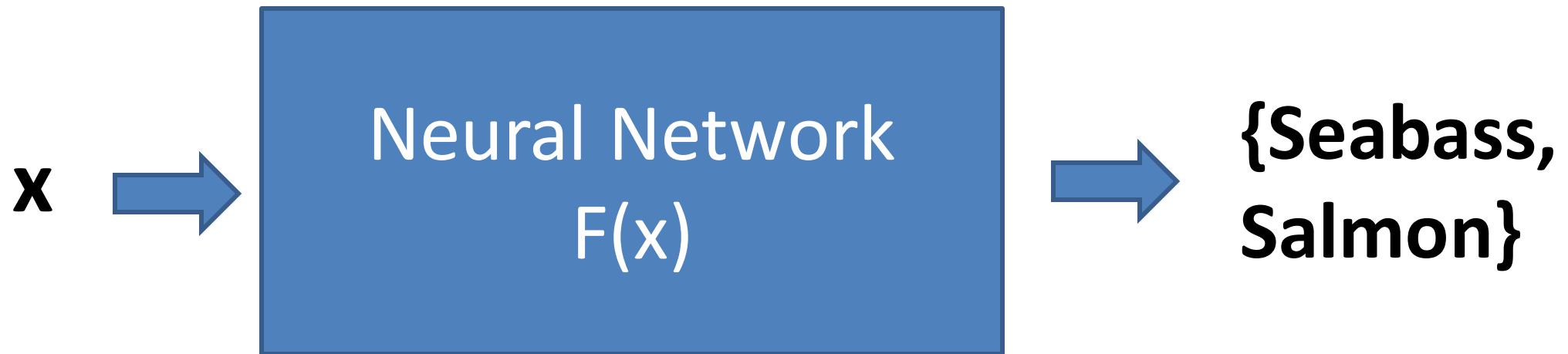
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$





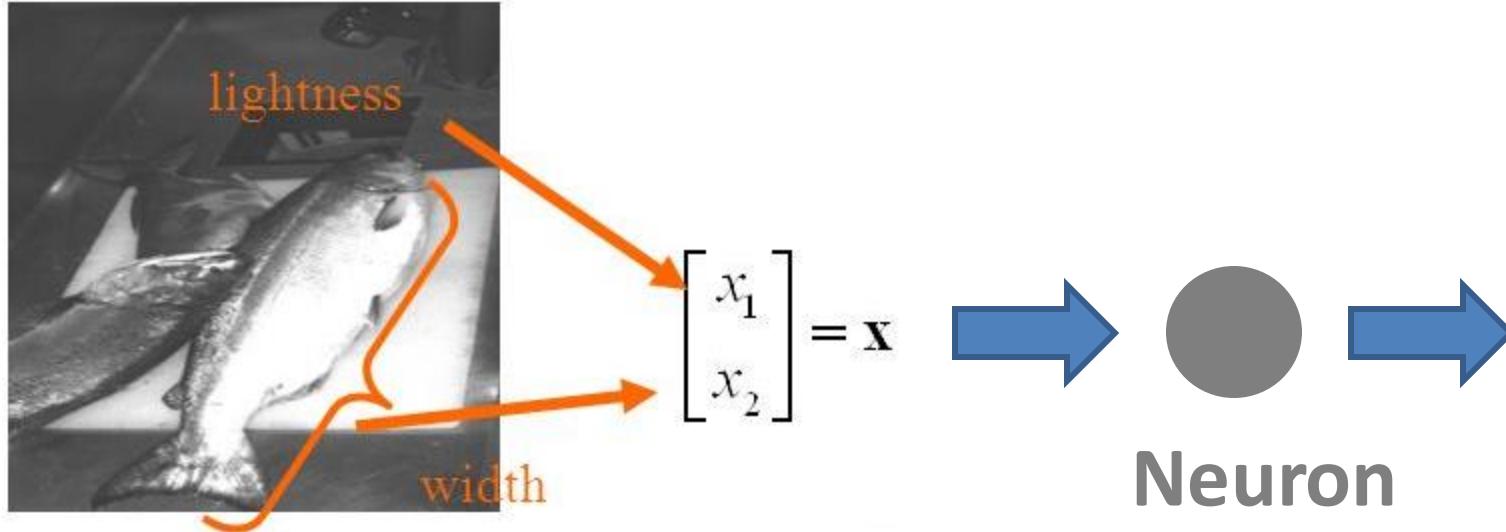
---

Neural Network = A function approximator





# The simplest neural network



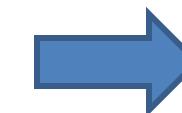
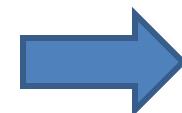
{Seabass,  
Salmon}



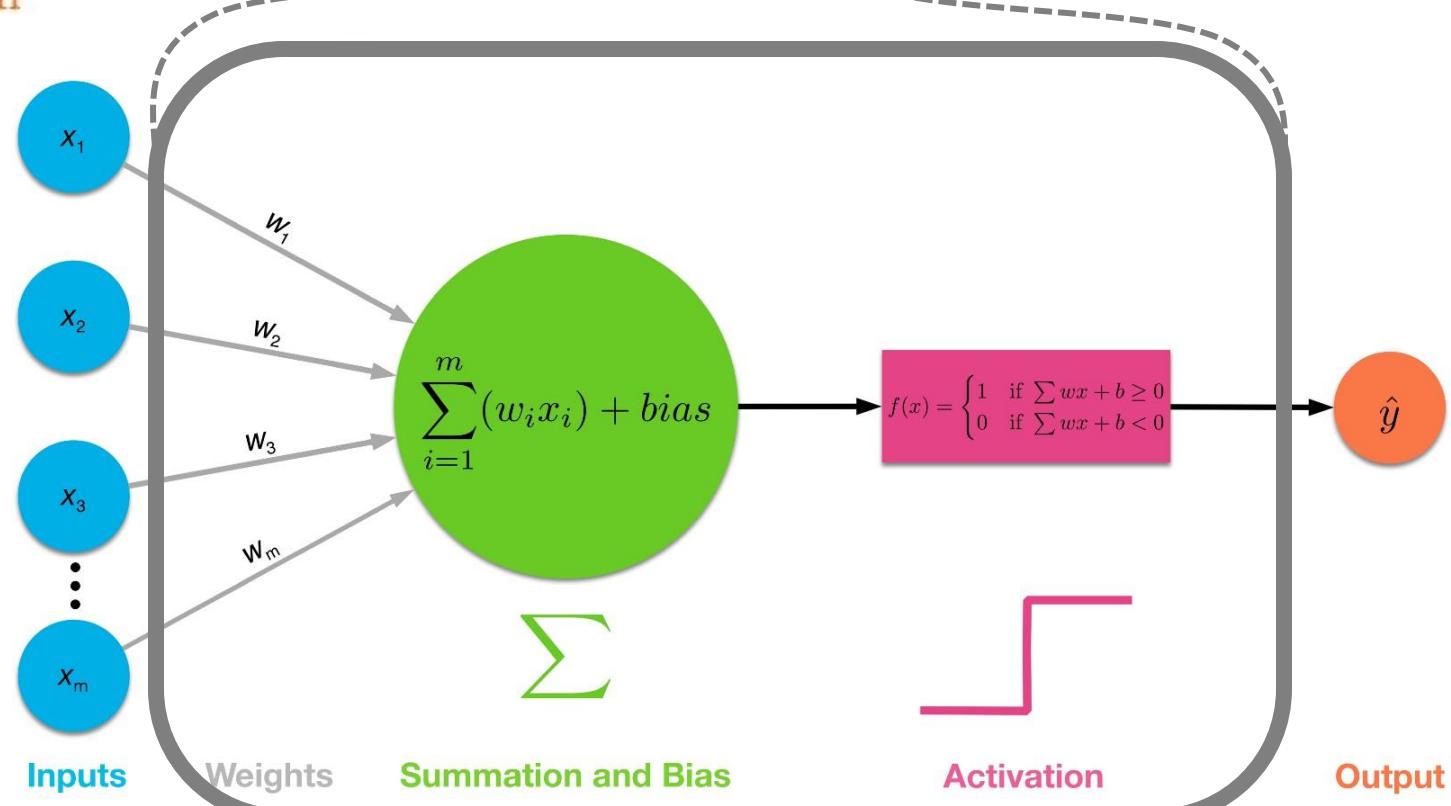
# The simplest neural network



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

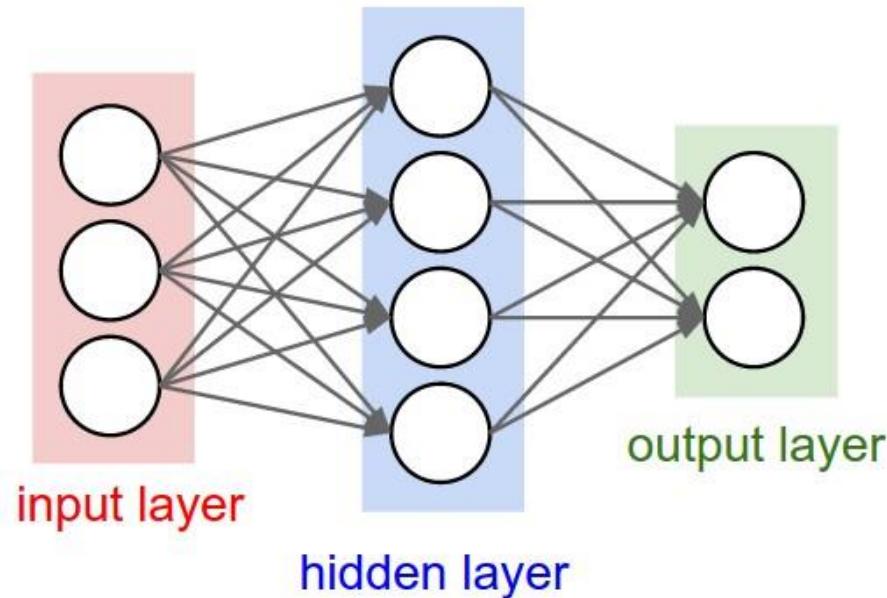


{Seabass,  
Salmon}





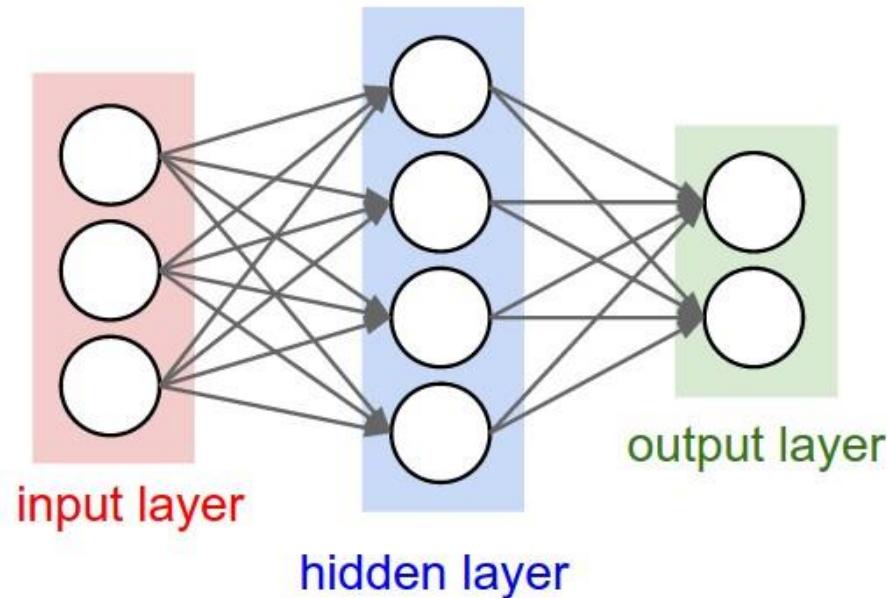
# Writing “code” (Keras) ...



```
model = Sequential()
model.add(Dense(4, input_dim=3)
model.add(Activation('tanh'))
model.add(Dense(2))
model.compile(loss='mean_squared_error', optimizer=adam)
```



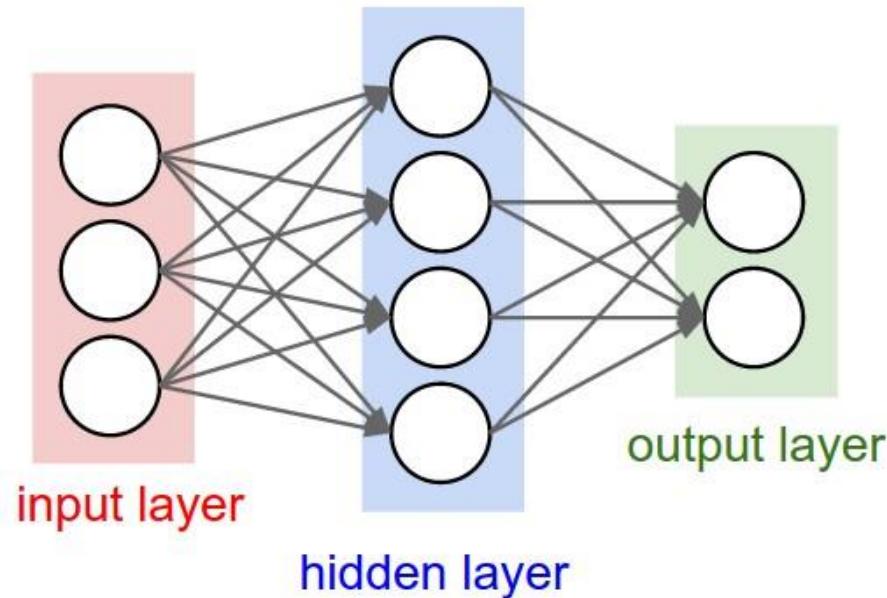
# Training the model



```
# Training  
# X_train : [num-train-samples x 3]  
# Y_train : [num-train-samples x 2]  
model.fit(X_train, y_train)
```



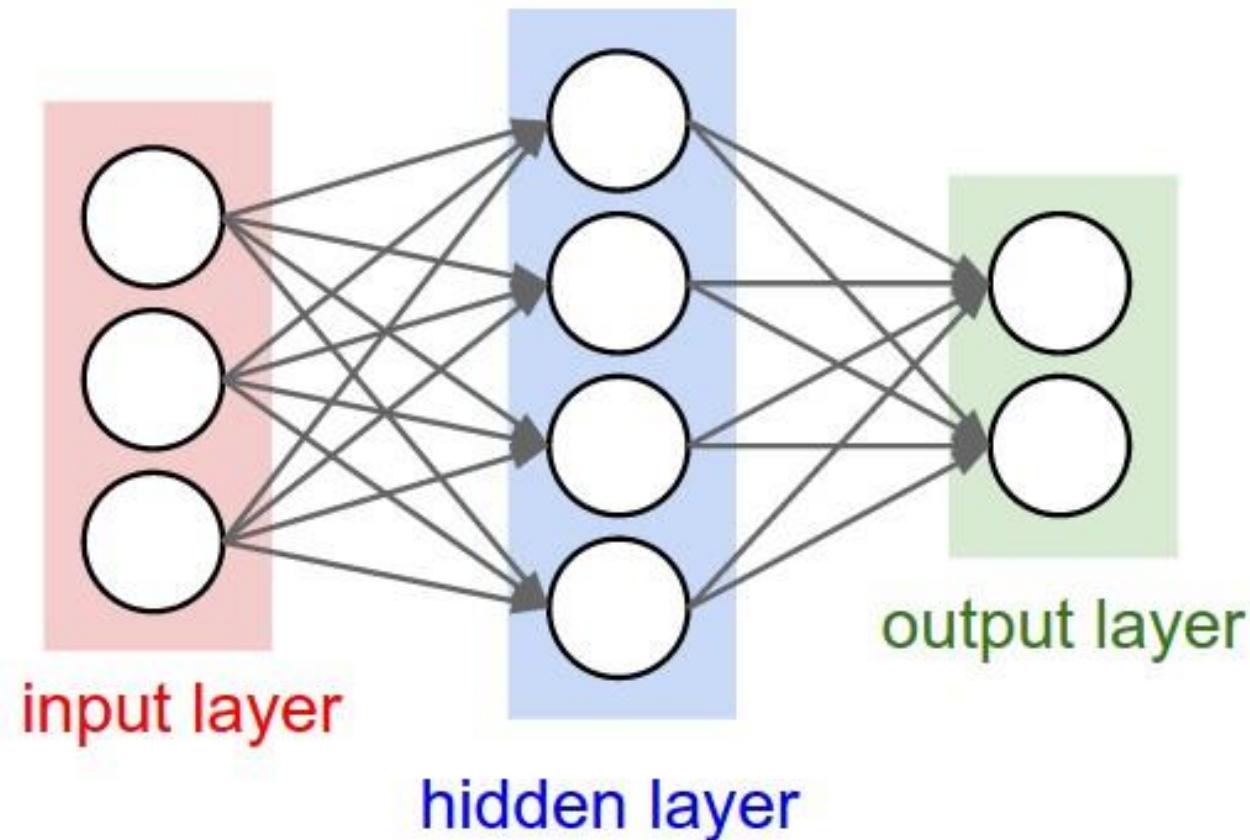
# Model evaluation / Prediction



```
# Prediction
# X_test : [num-test-samples x 3]
# Y_test : [num-test-samples x 2]
score = model.evaluate(X_test, y_test)
```



# 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 ?**



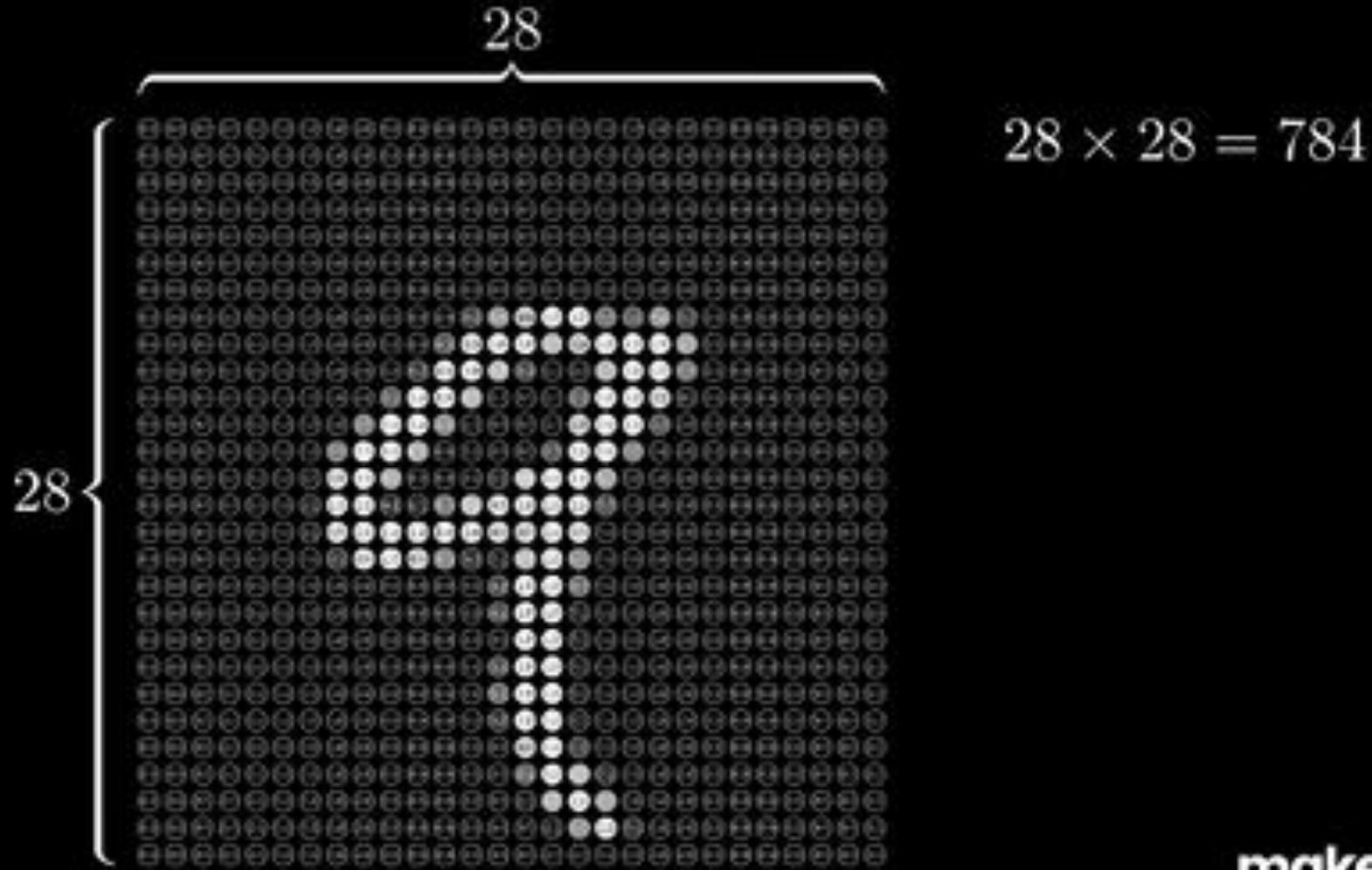
---

# MNIST Handwritten Digits Dataset

0  
1  
2  
3  
4  
5  
6  
7  
8  
9 9



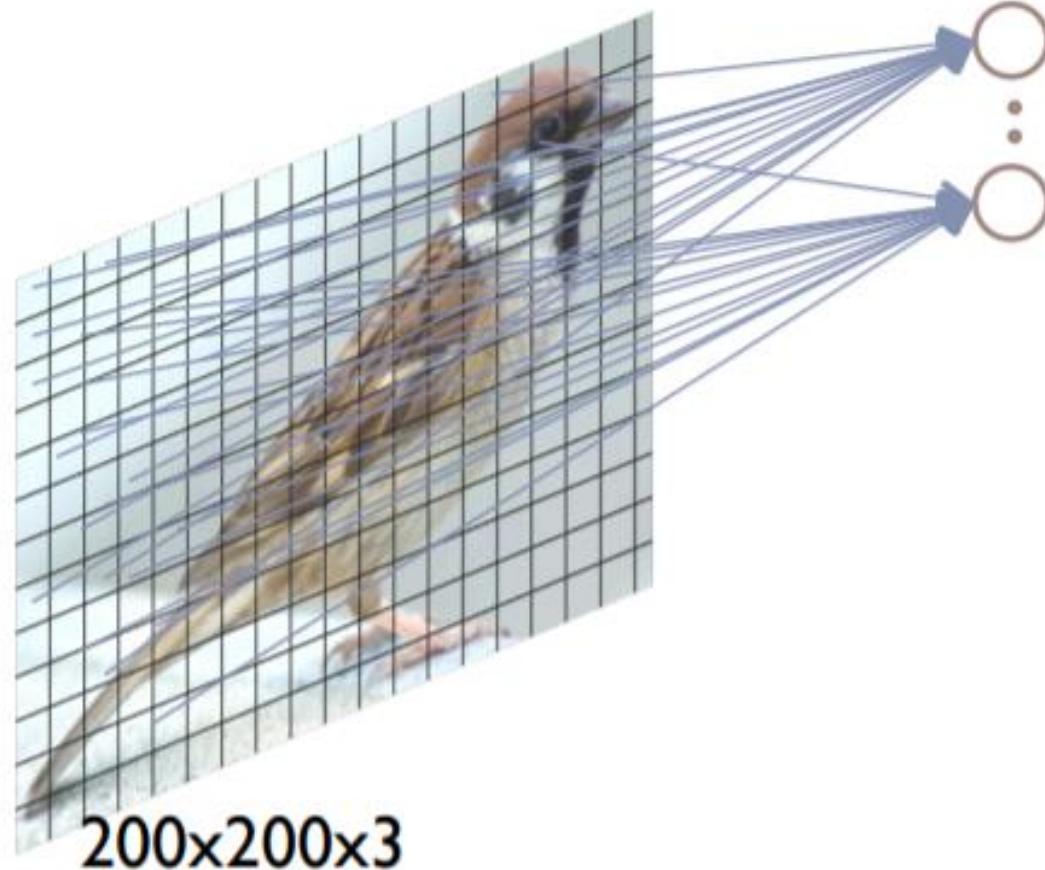
# Flatten image , Feed to NN



makeagif.com



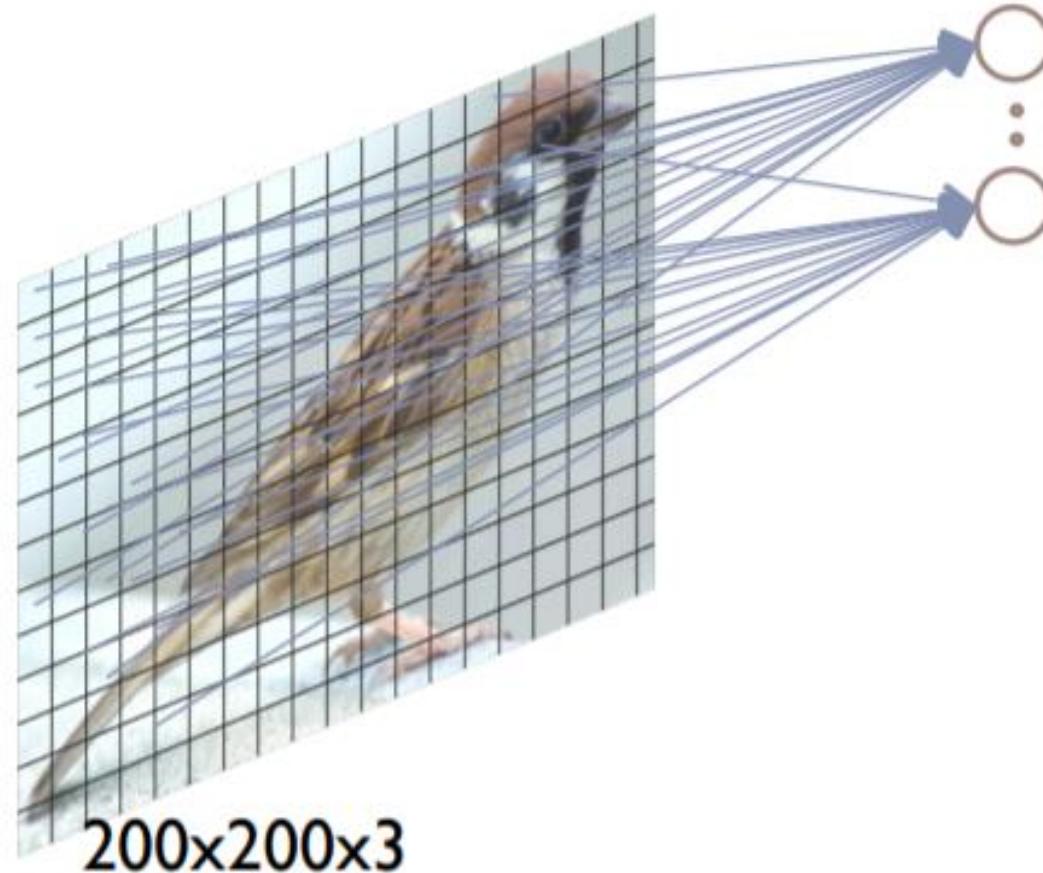
# For color images ?



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



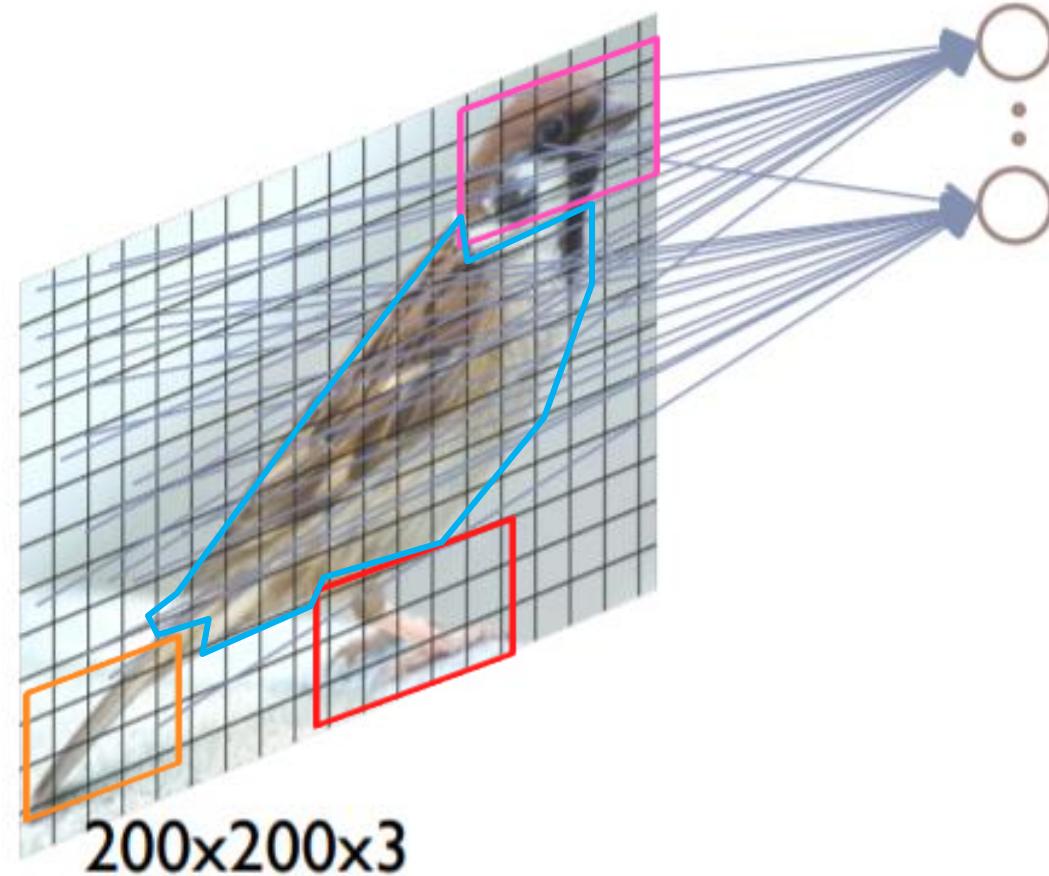
# Do we really need full connectivity ?



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



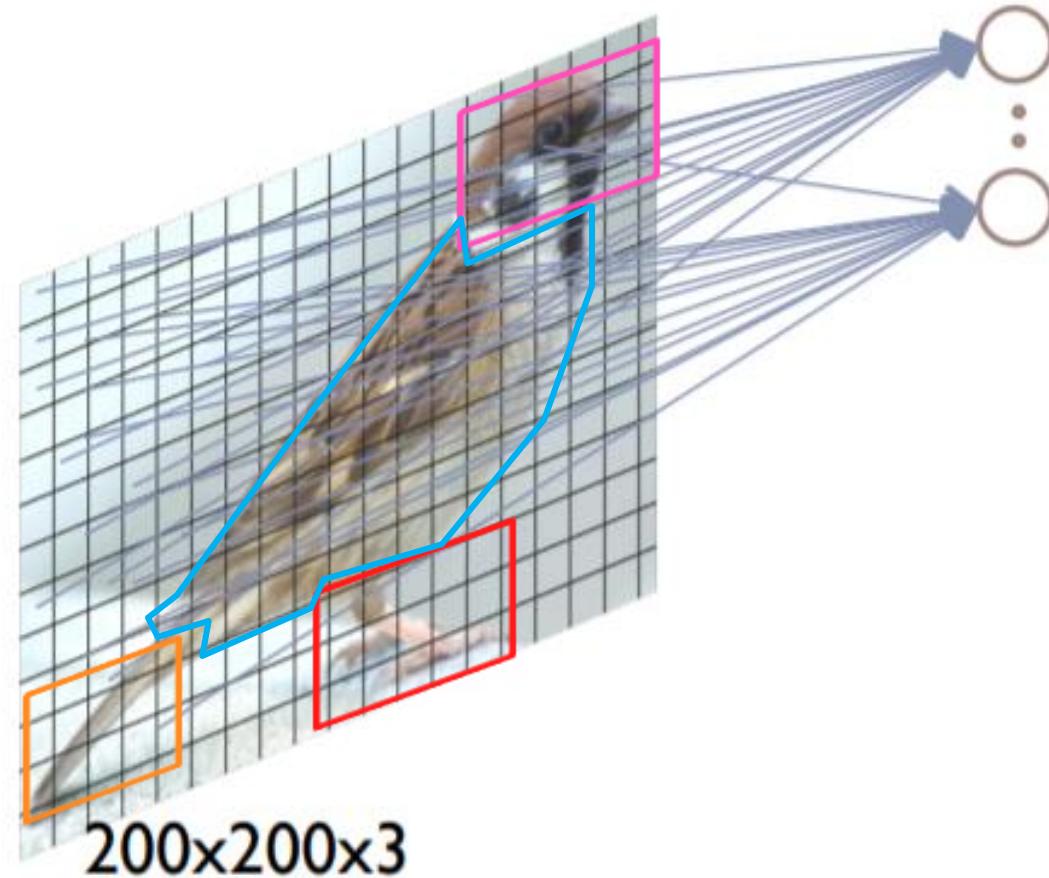
# Do we really need full connectivity ?



- Head + Body + Tail + Legs = sparrow



# Do we really need full connectivity ?

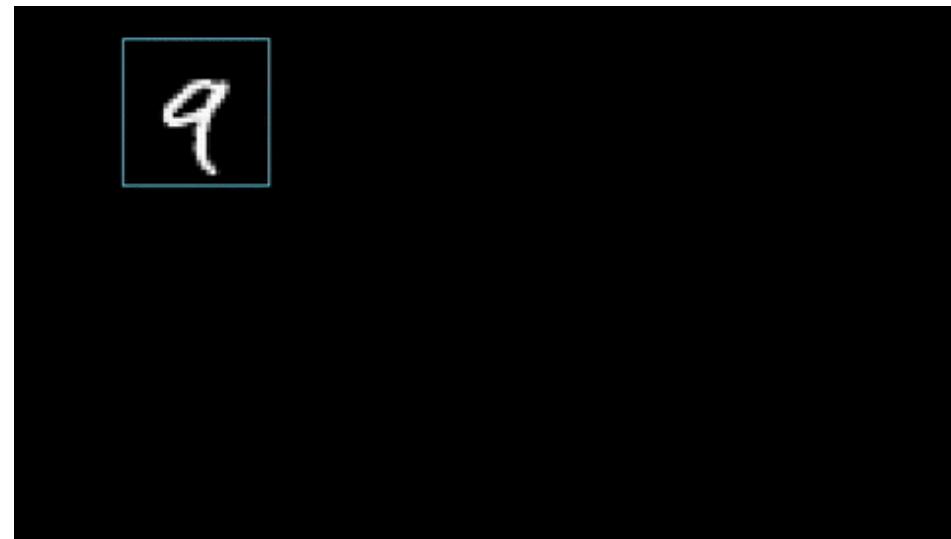


- Head + Body + Tail + Legs = sparrow



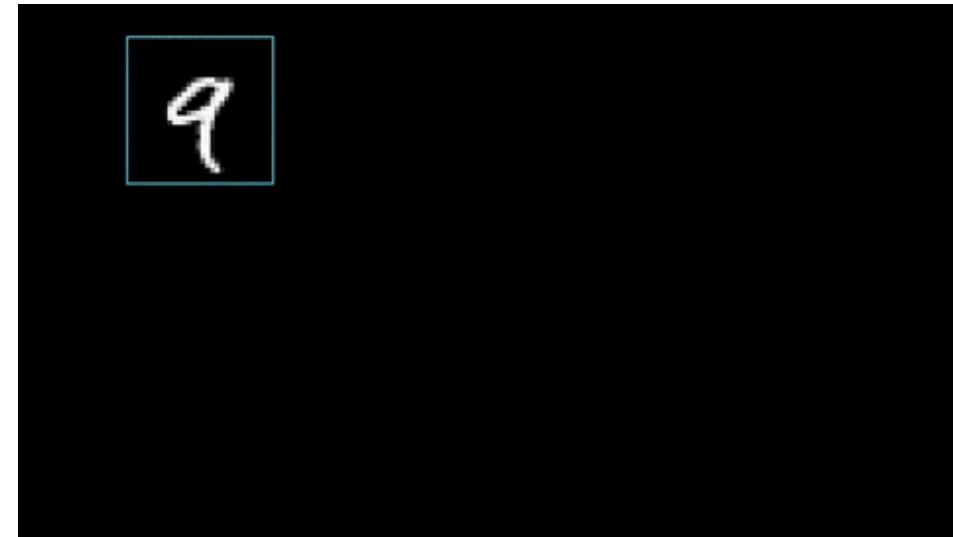


- 
- Images are 2D.
  - Assumption: Object image = combination of **2D image patterns**





- 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



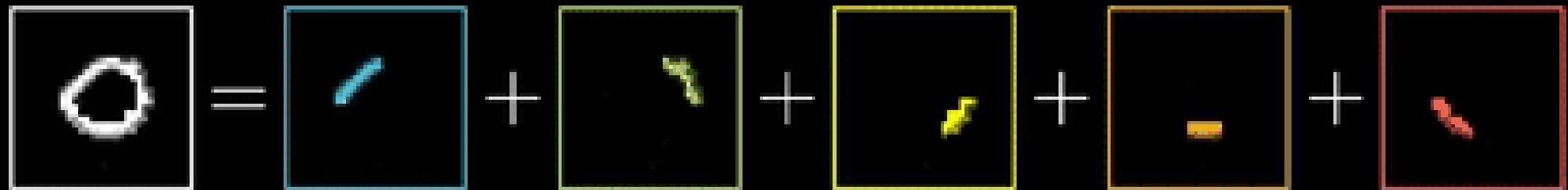


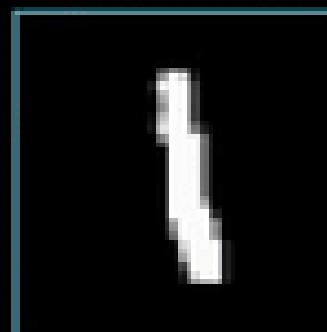
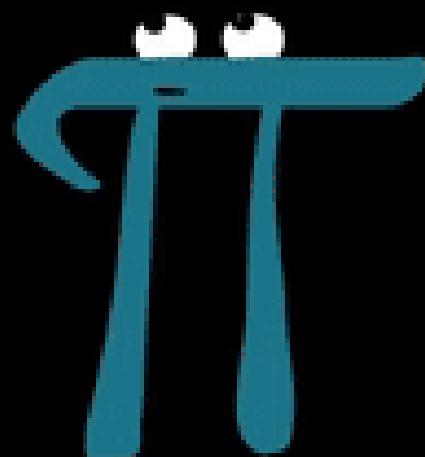
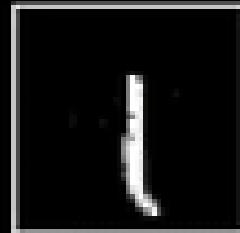
- Images are 2D.
- Assumption: Object image = combination of **2D image patterns**
- Machine Learning Strategy:
  - [Pre-final layer] Determine 2D image patterns
  - Map 2D image patterns → Target label
- NOTE: 2D image patterns are smaller than image





# Hierarchical decomposition of input/features

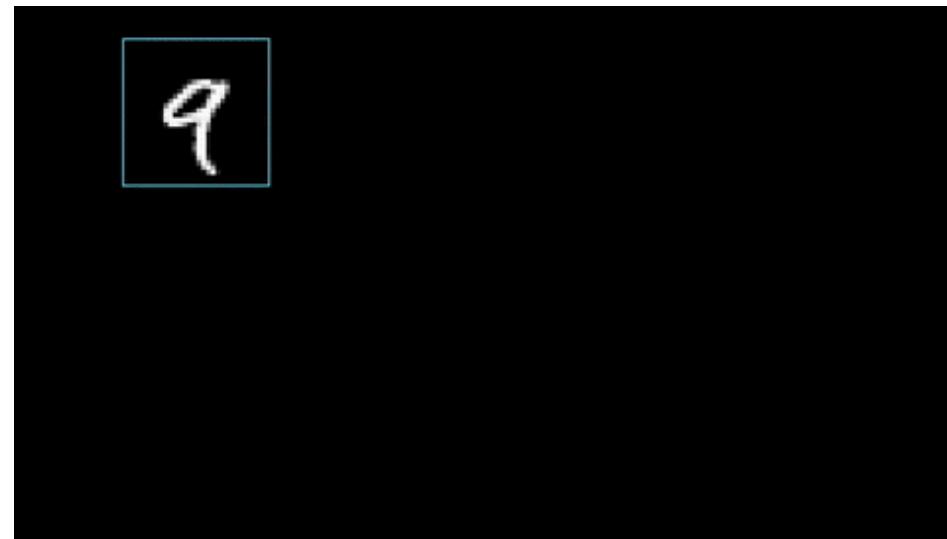
$$\text{Input Image} = \text{Feature 1} + \text{Feature 2} + \text{Feature 3} + \text{Feature 4} + \text{Feature 5}$$




makeagif.com



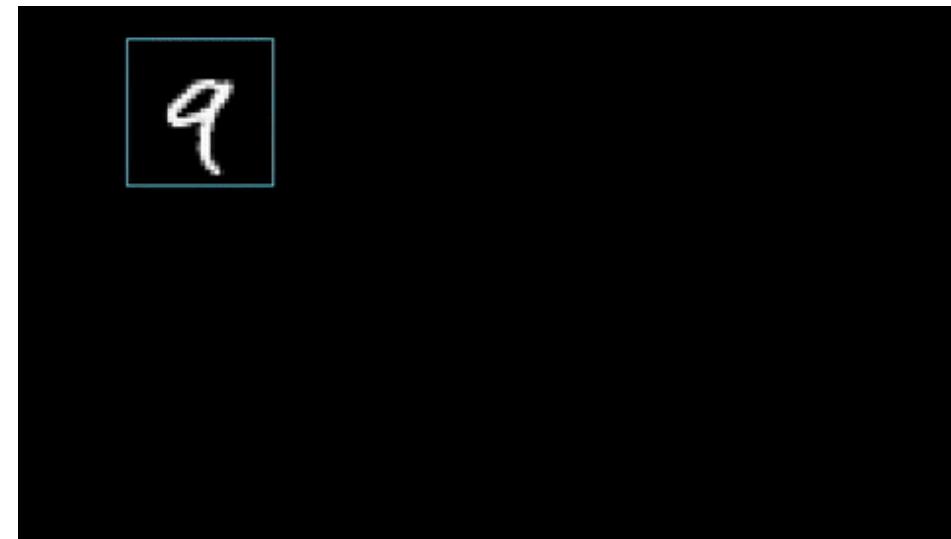
- Images are 2D.
- Assumption: Object image = **hierarchical** combination of **2D image patterns**
- **Machine Learning Strategy:**
  - [All except pre-final] Determine 2D image patterns
  - [Pre-final layer] Map 2D image patterns → Target label
- NOTE: 2D image patterns are smaller than image





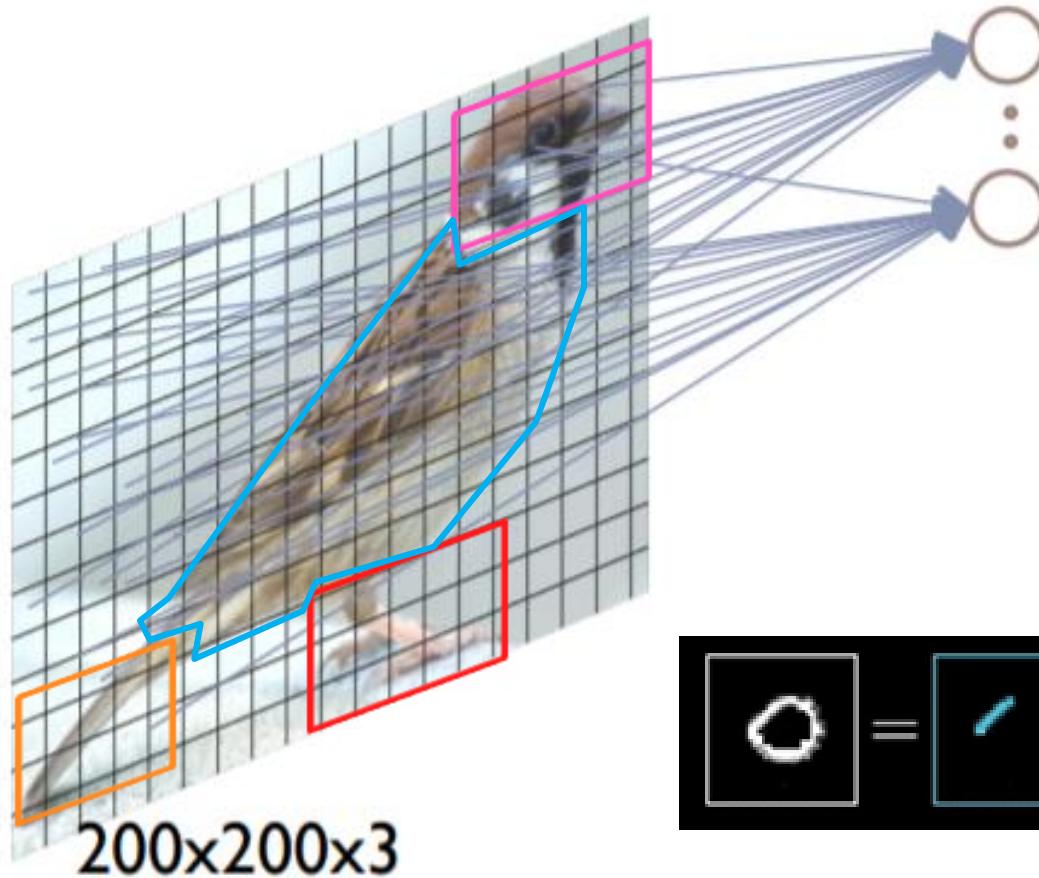
**Efficient !**

- 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

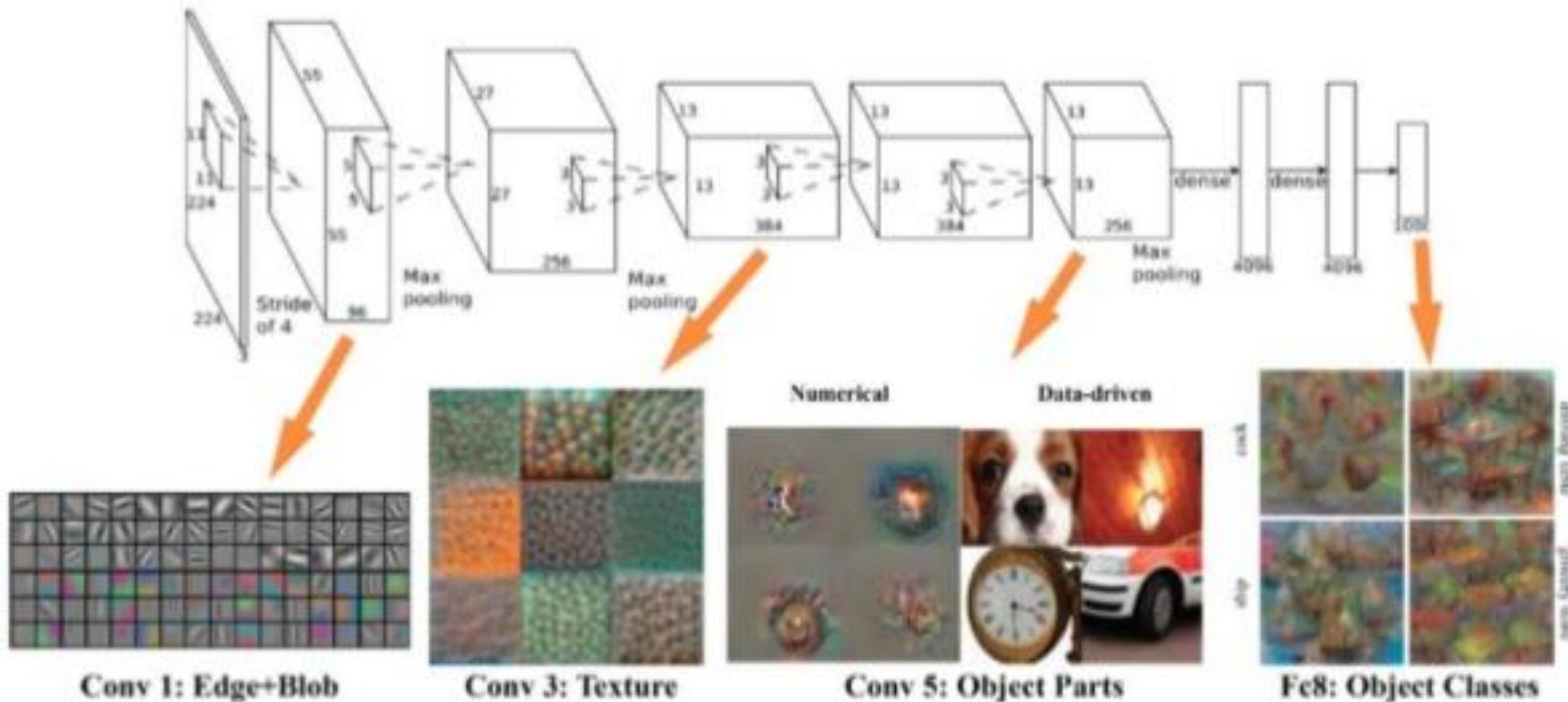




**SPOILER ALERT!**



$$\text{Output Node} = \text{Bias} + \text{Blue Feature} + \text{Green Feature} + \text{Yellow Feature} + \text{Orange Feature} + \text{Red Feature}$$



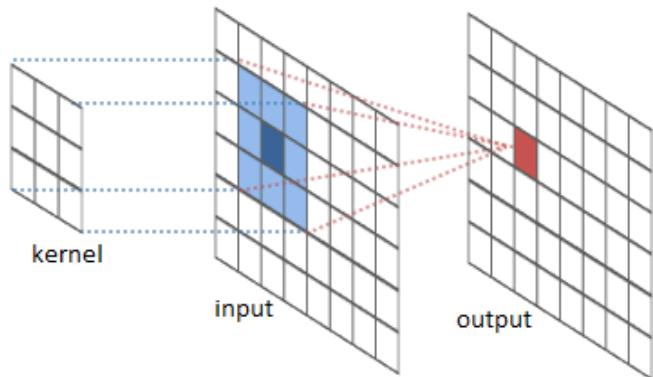
From: [mNeuron: A Matlab Plugin to Visualize Neurons from Deep Models](#), Donglai Wei et. al.

- 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 ?

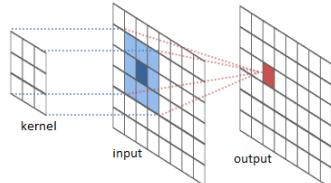
- Determine the filter
- ‘Scan’ the filter on input image / representation
- Output = A record of all input locations that ‘match’ the filter



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



# How do filters work ?



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



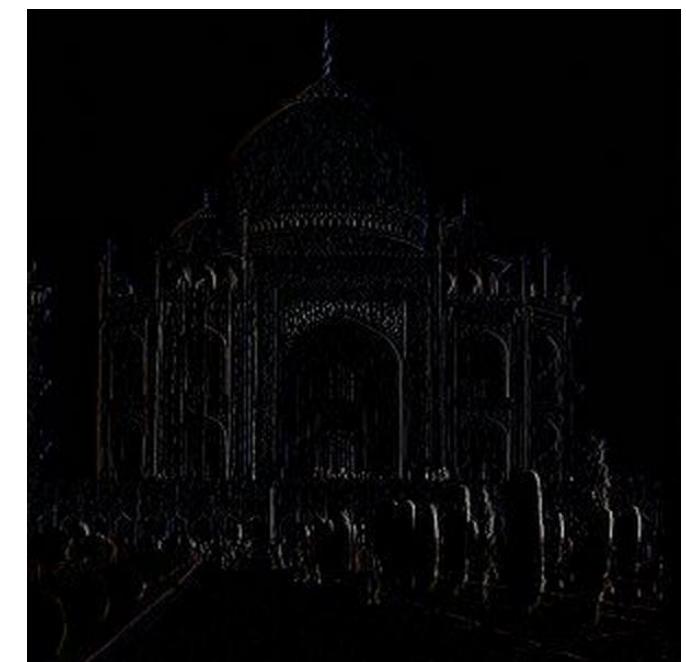
Input

A

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)**



---

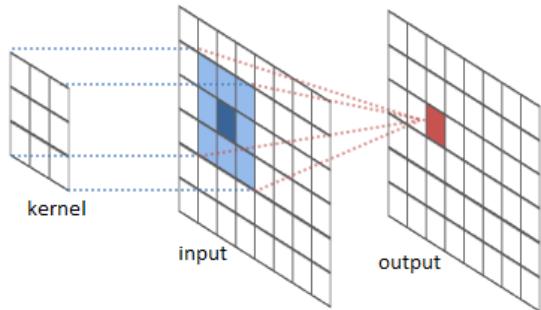
# Convolution (in general)

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

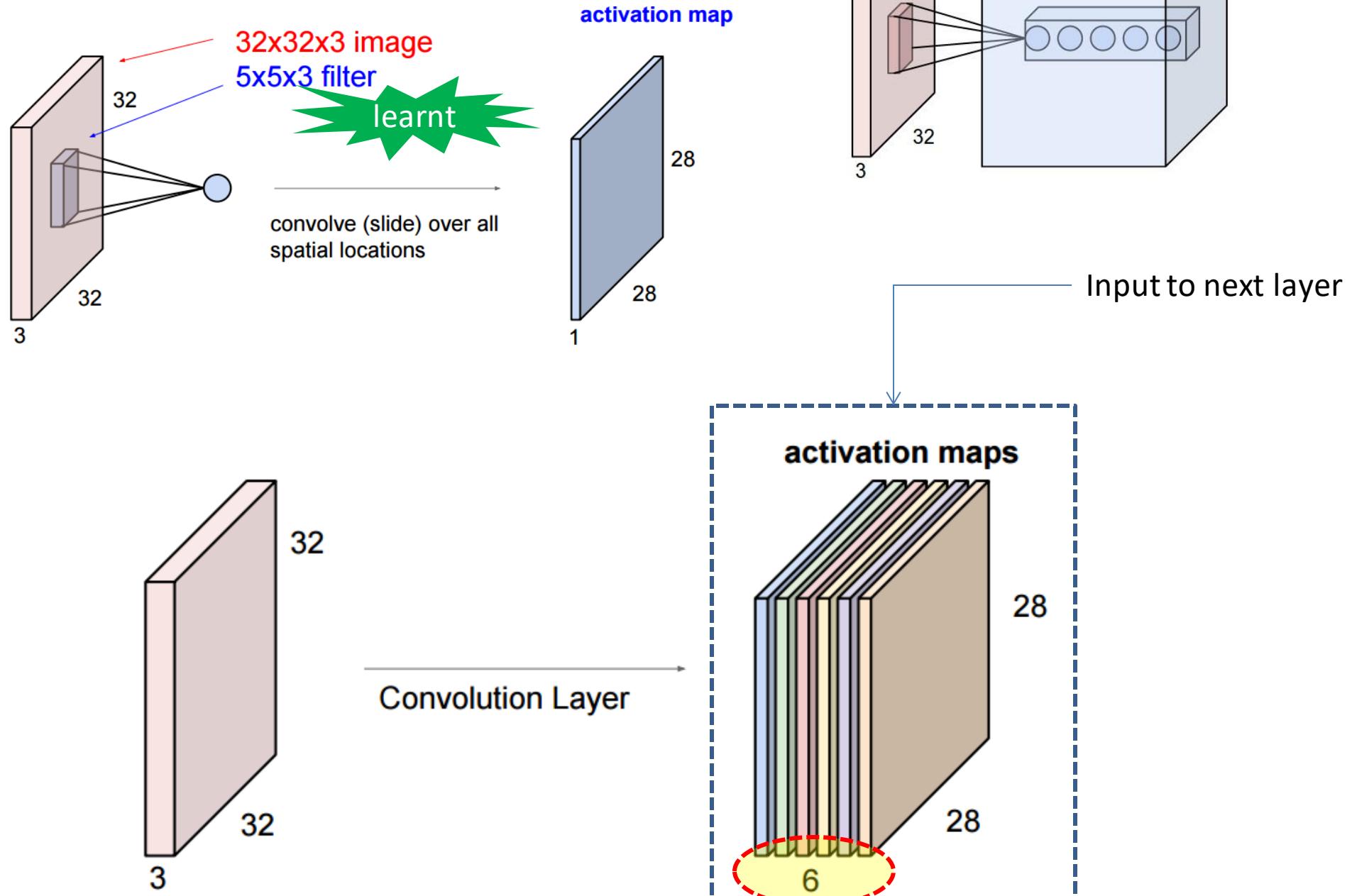


# Multiple filters, multiple ‘feature maps’



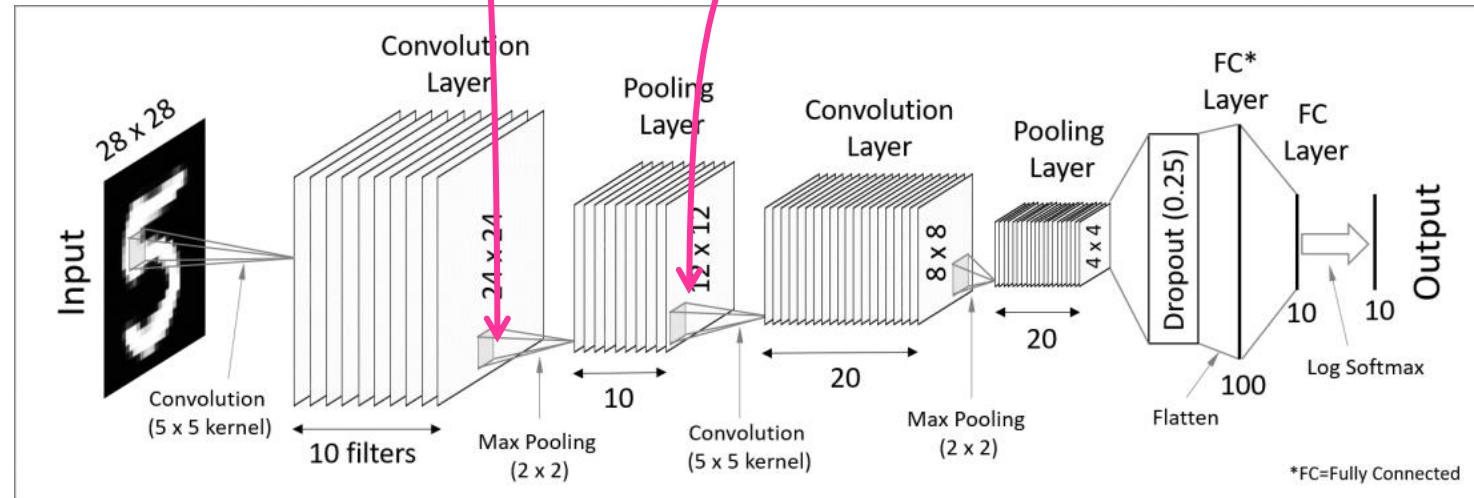
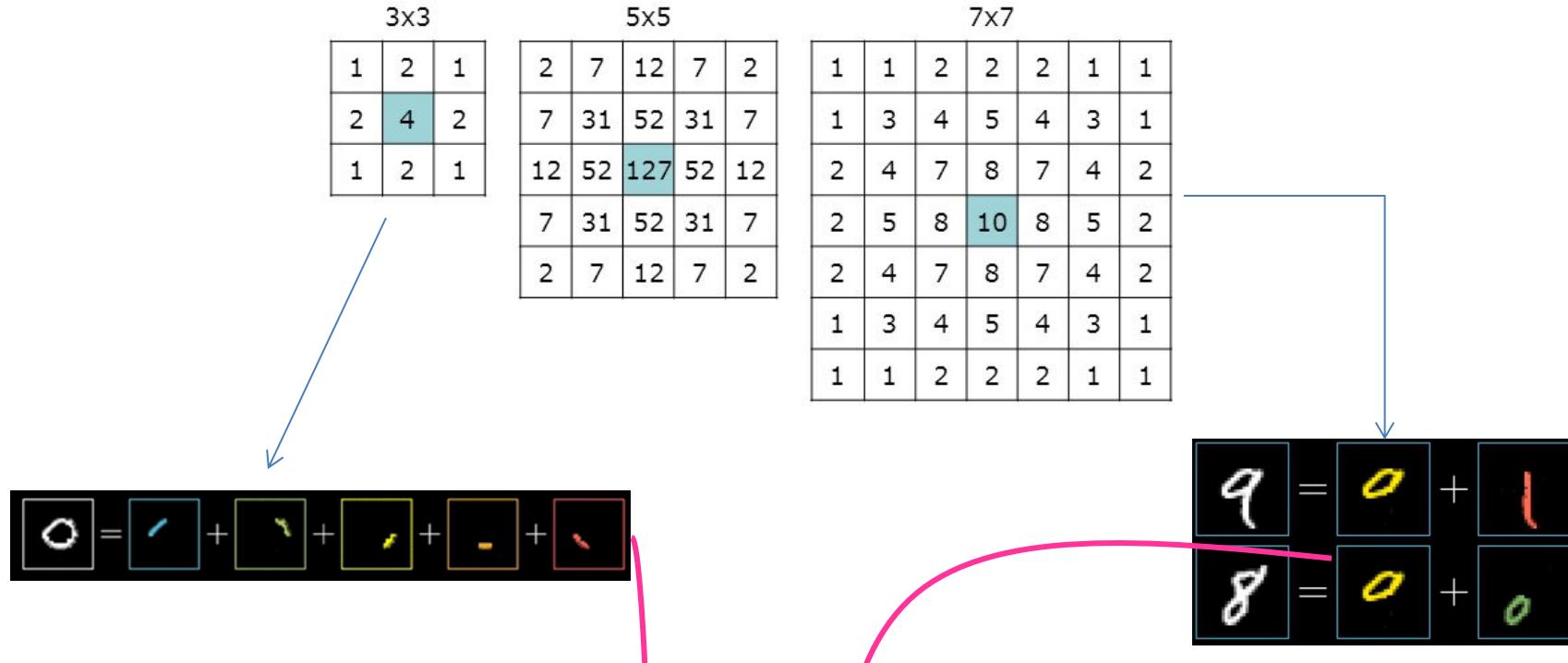


# Convolution Layer



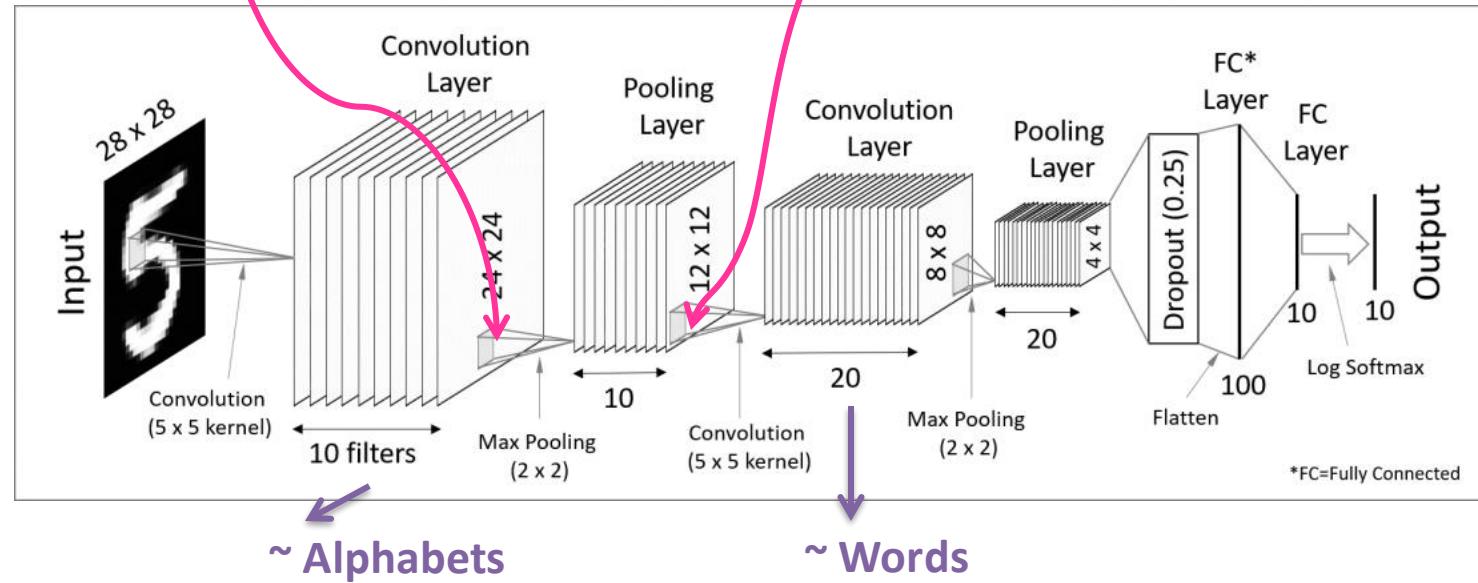
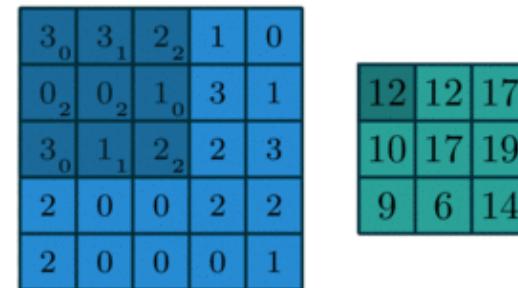


# Design choices : Filter size, # of filters





# Design choices : Filter size, # of filters





# Pooling Layer

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2



6	8
3	4



# Pooling Layer

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2



6	8
3	4



# Pooling Layer

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2



6	8
3	4

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



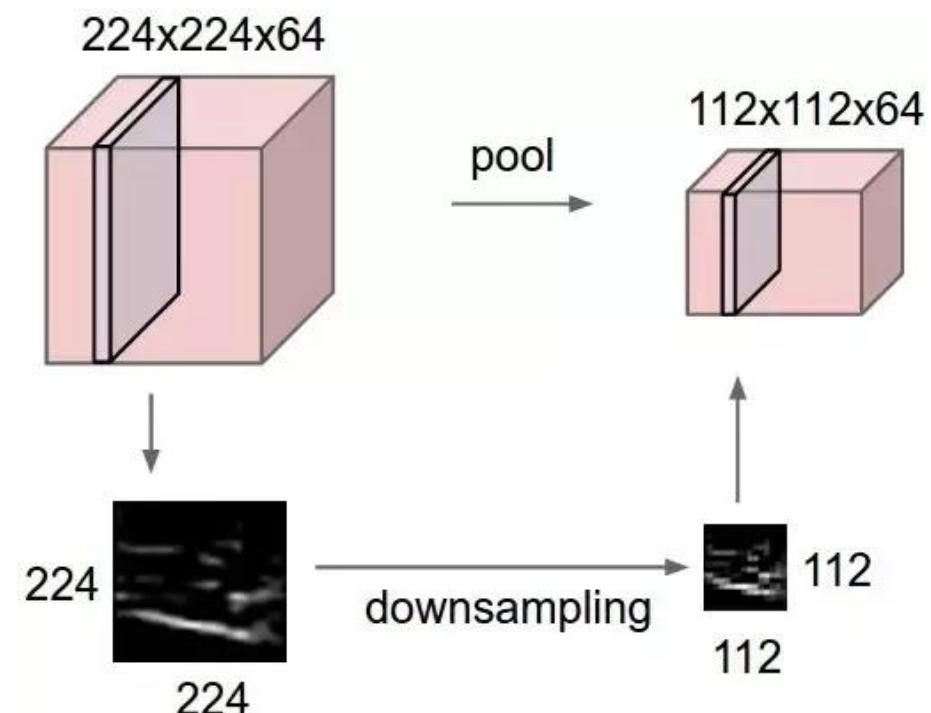
# Pooling Layer

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2

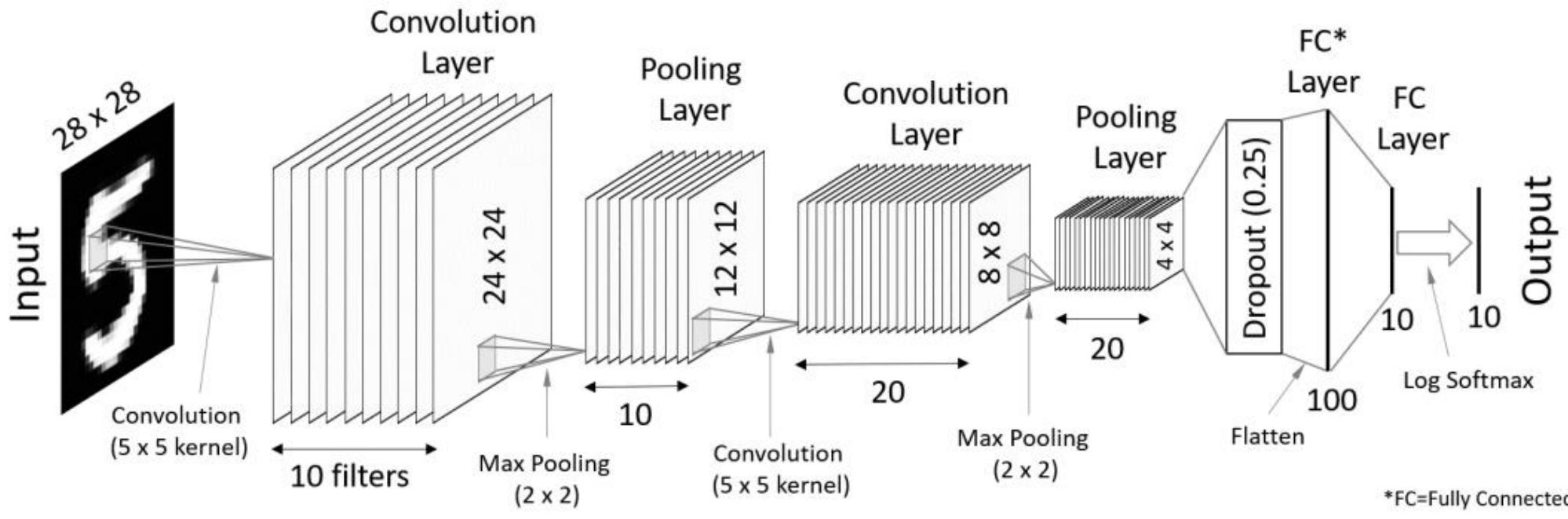
6	8
3	4

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





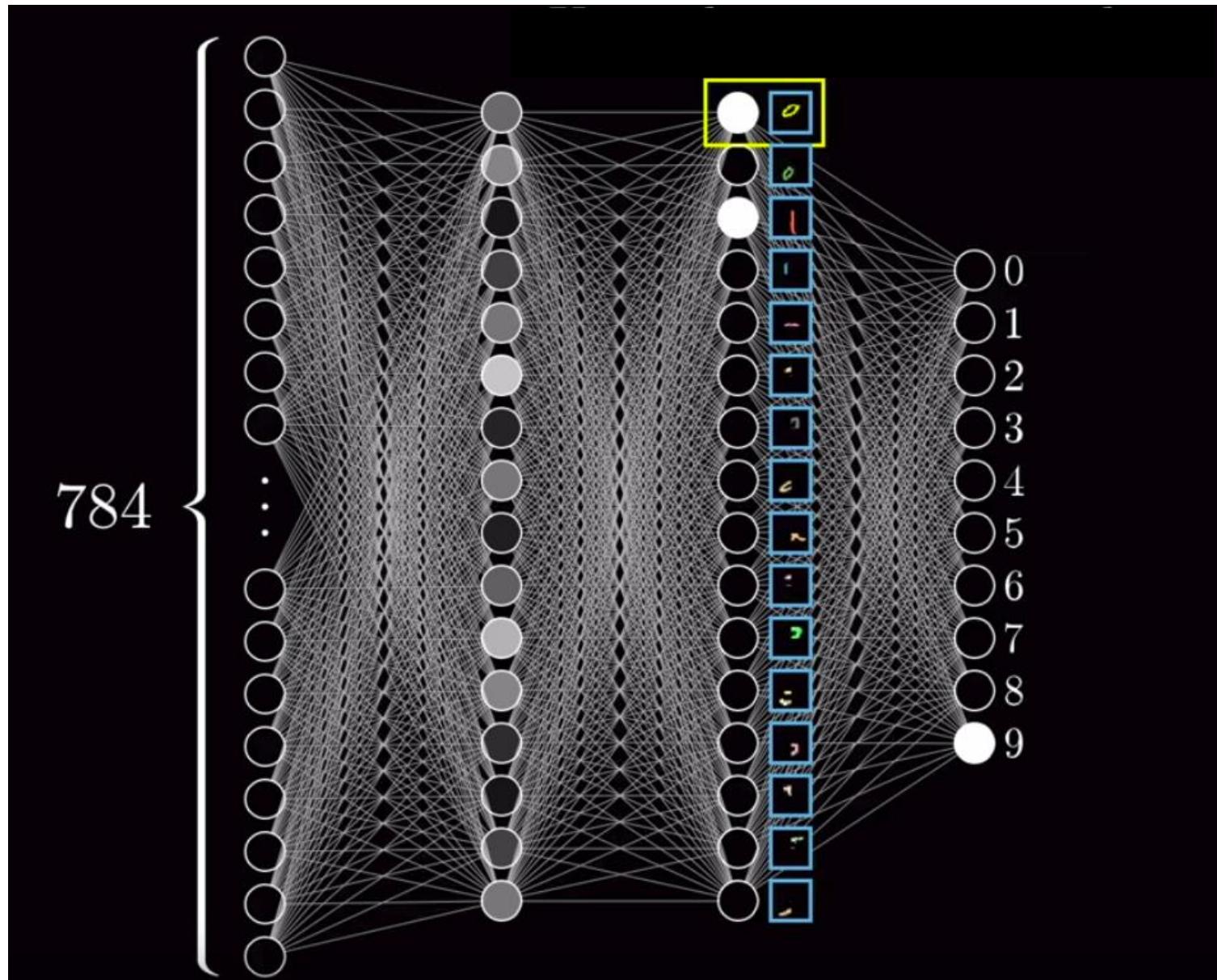
# Fully-connected layers





---

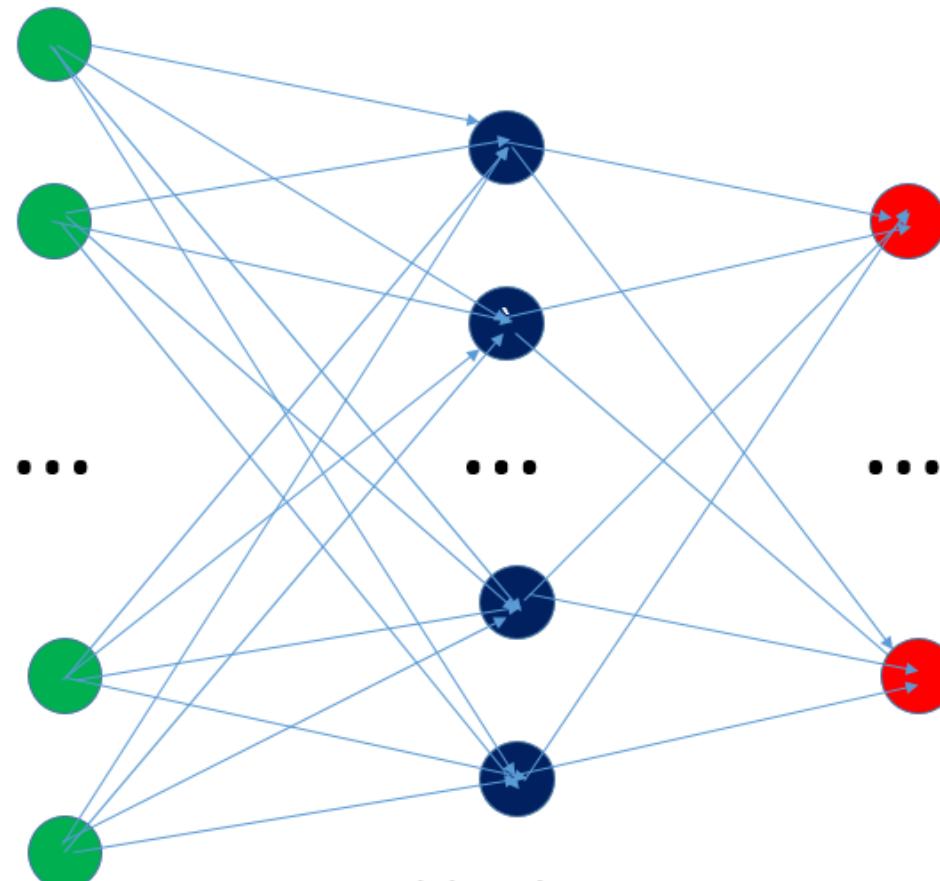
# Fully Connected Layers





7

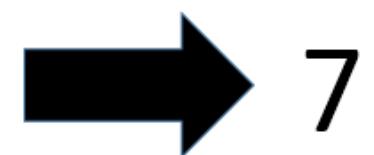
Input image:  
28x28 pixels



Input layer:  
784 neurons,  
one per pixel

Hidden layer:  
100 neurons

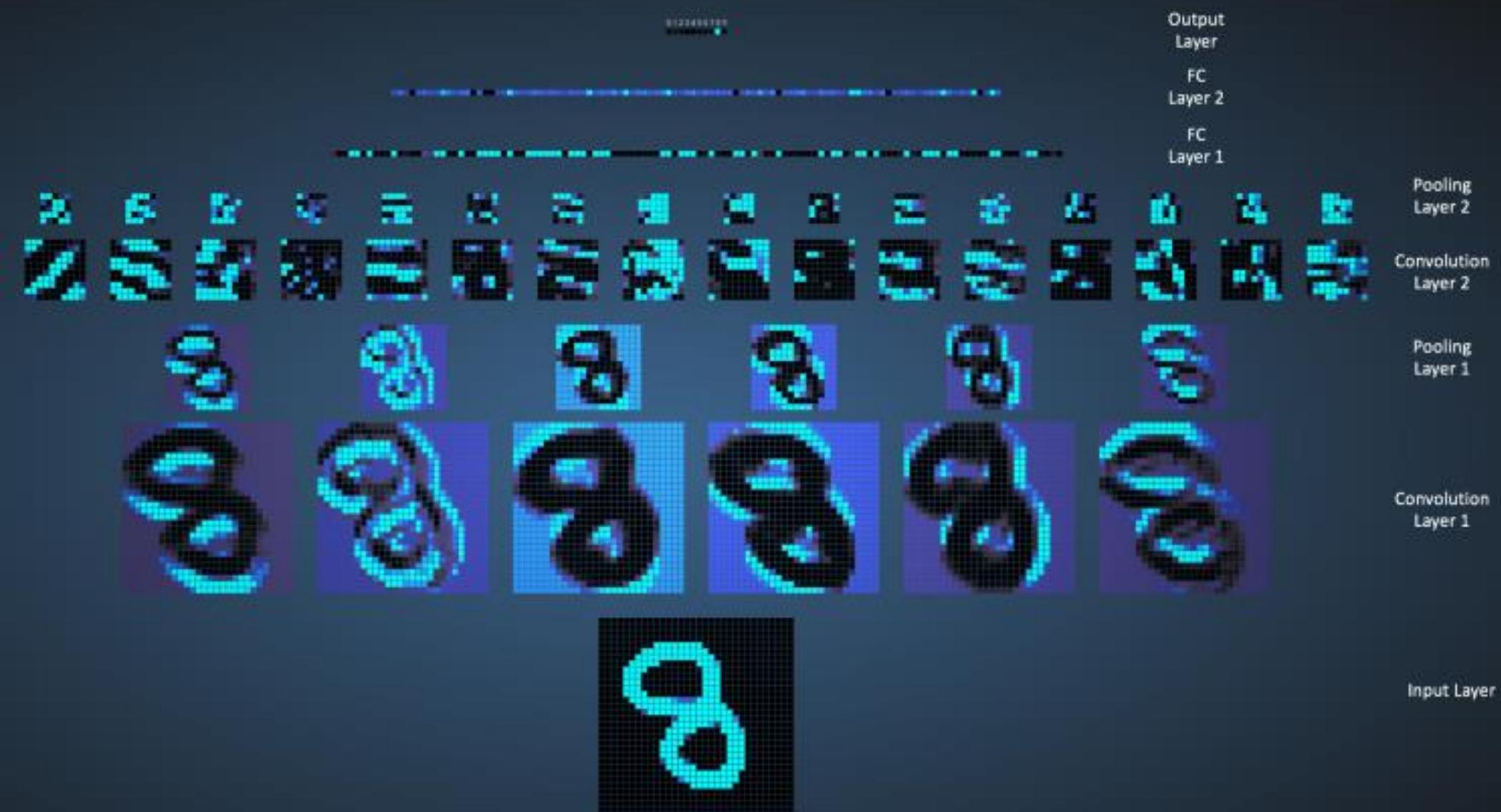
Output layer:  
10 neurons



7

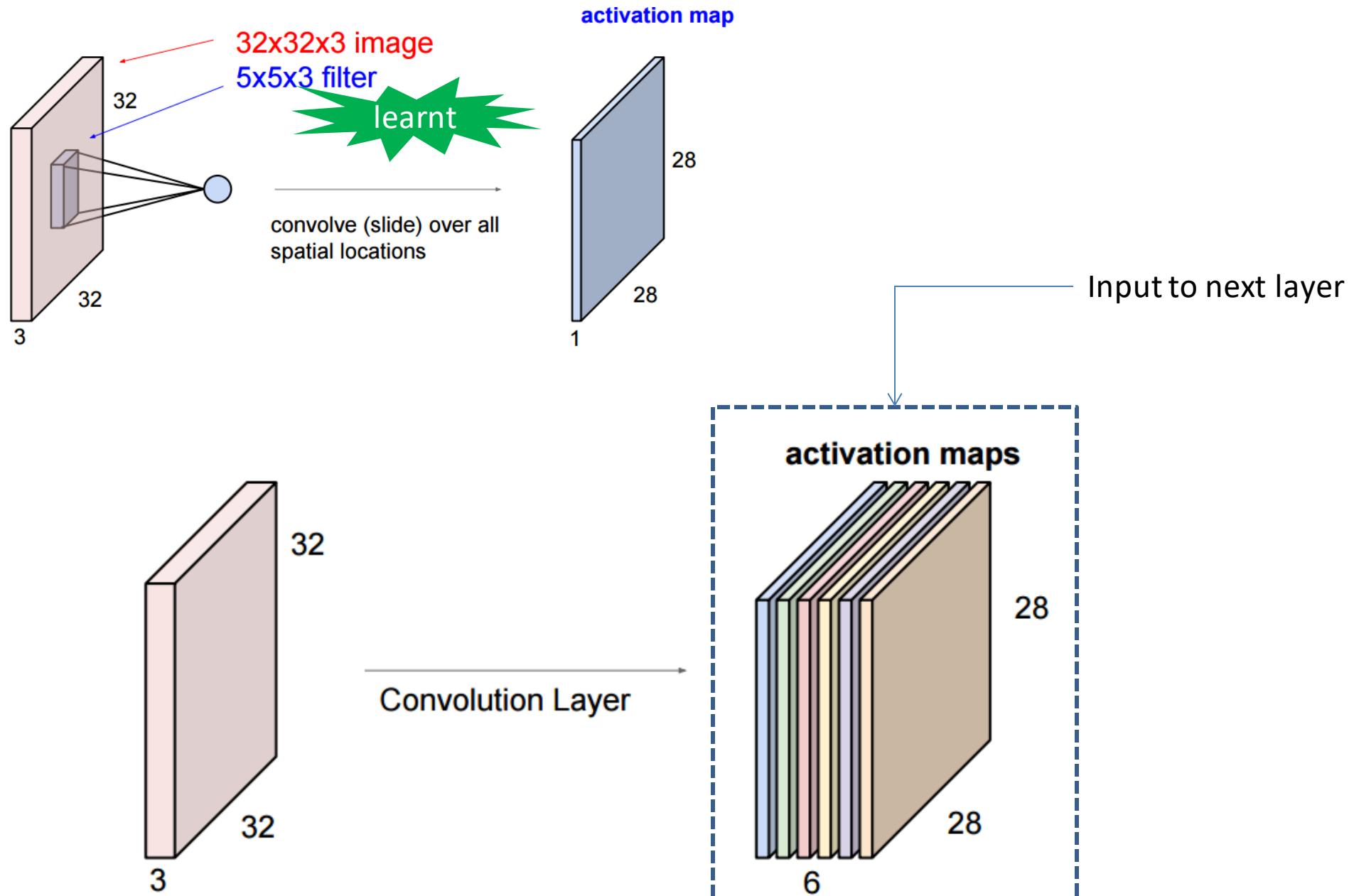
Output:  
predicted  
digit value

# MNIST features visualized



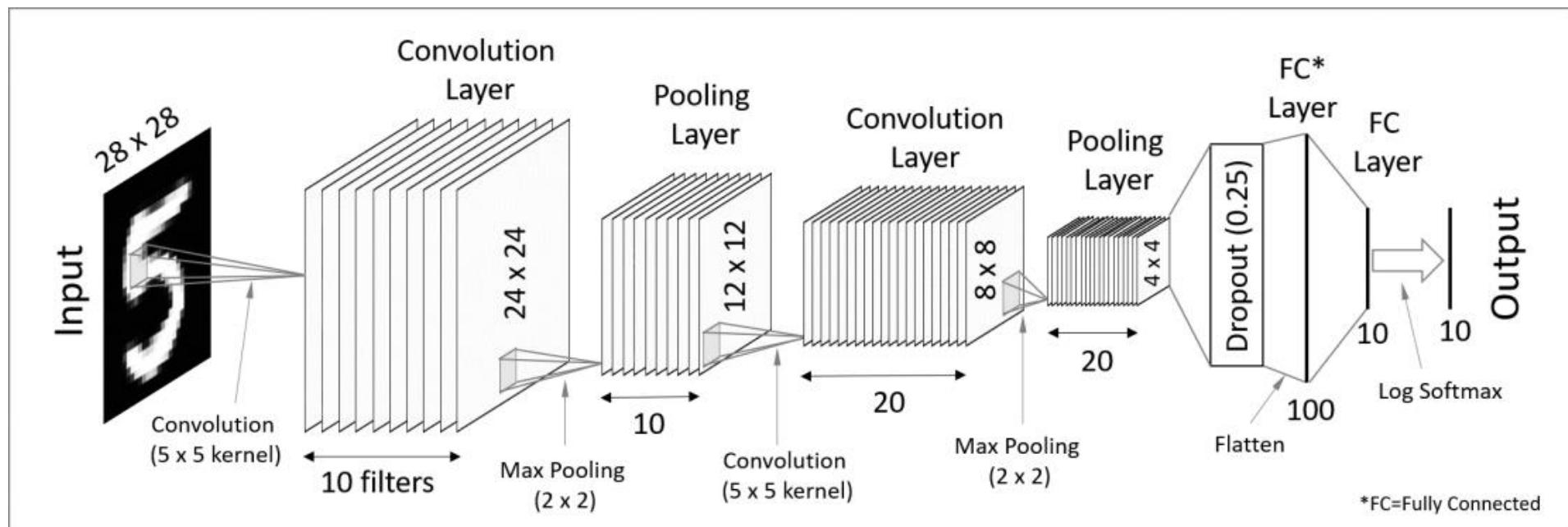


# Convolution Layer





# Convolutional Neural Network





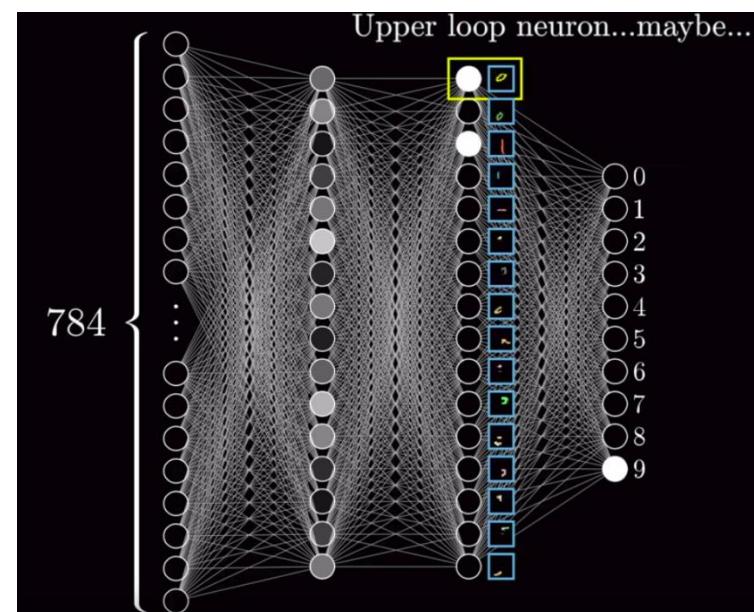
# Pooling Layer

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2

6	8
3	4

# Fully-connected Layer

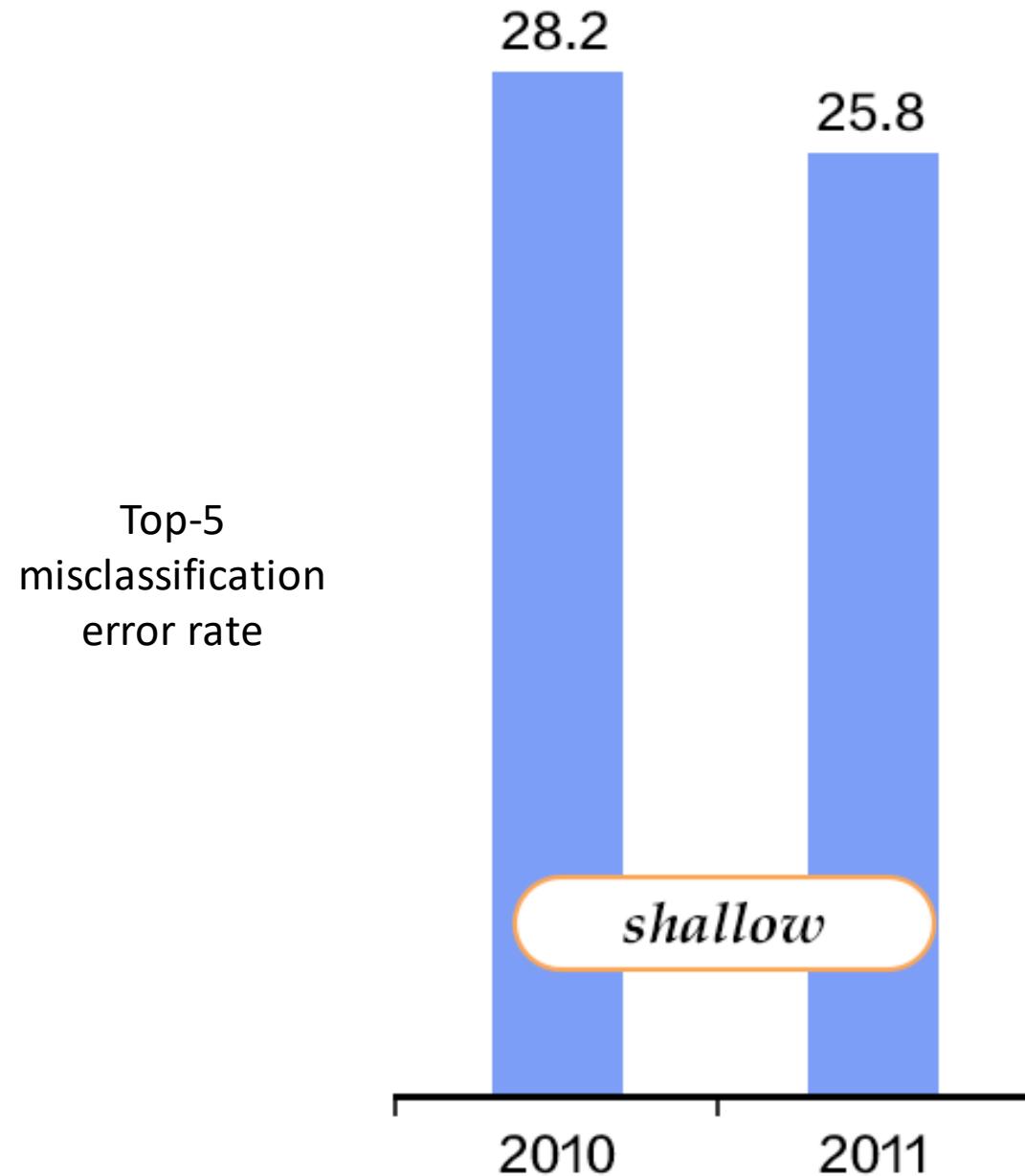


# ImageNet Challenge

IMAGENET

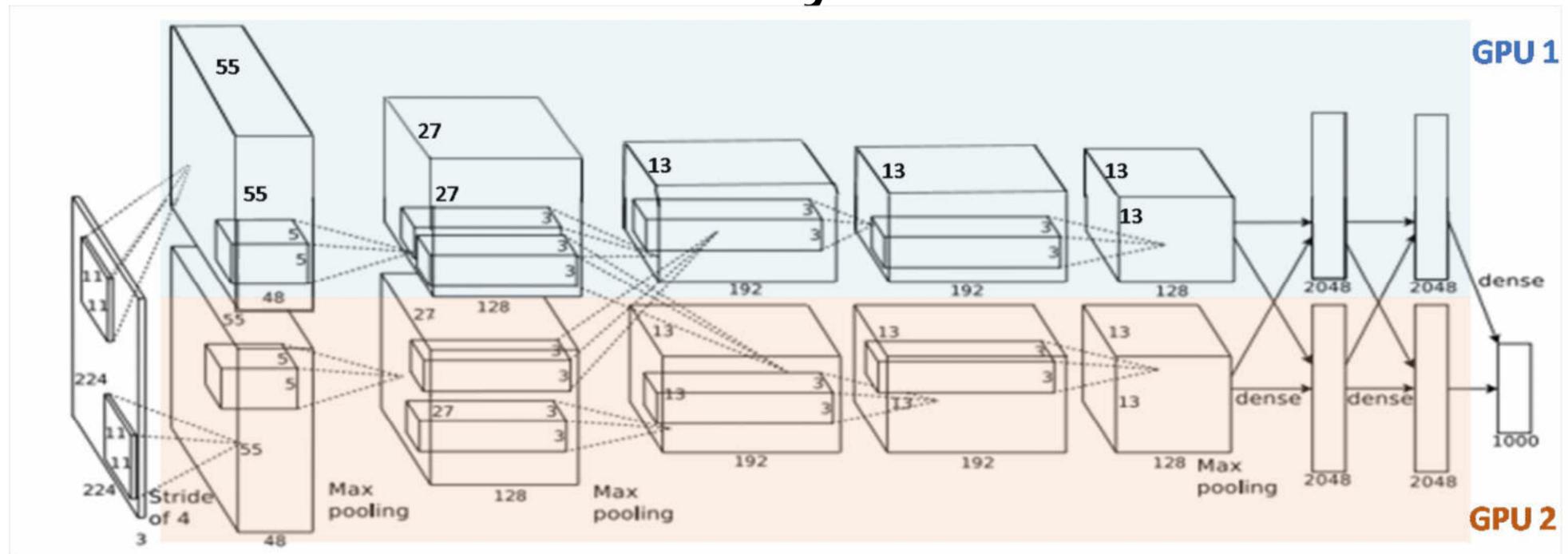
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.







# Case Study: AlexNet

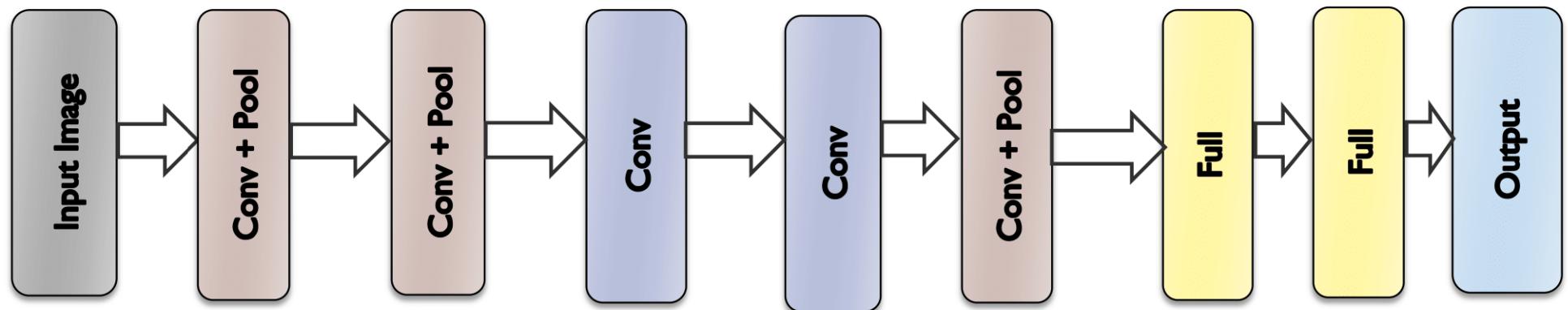
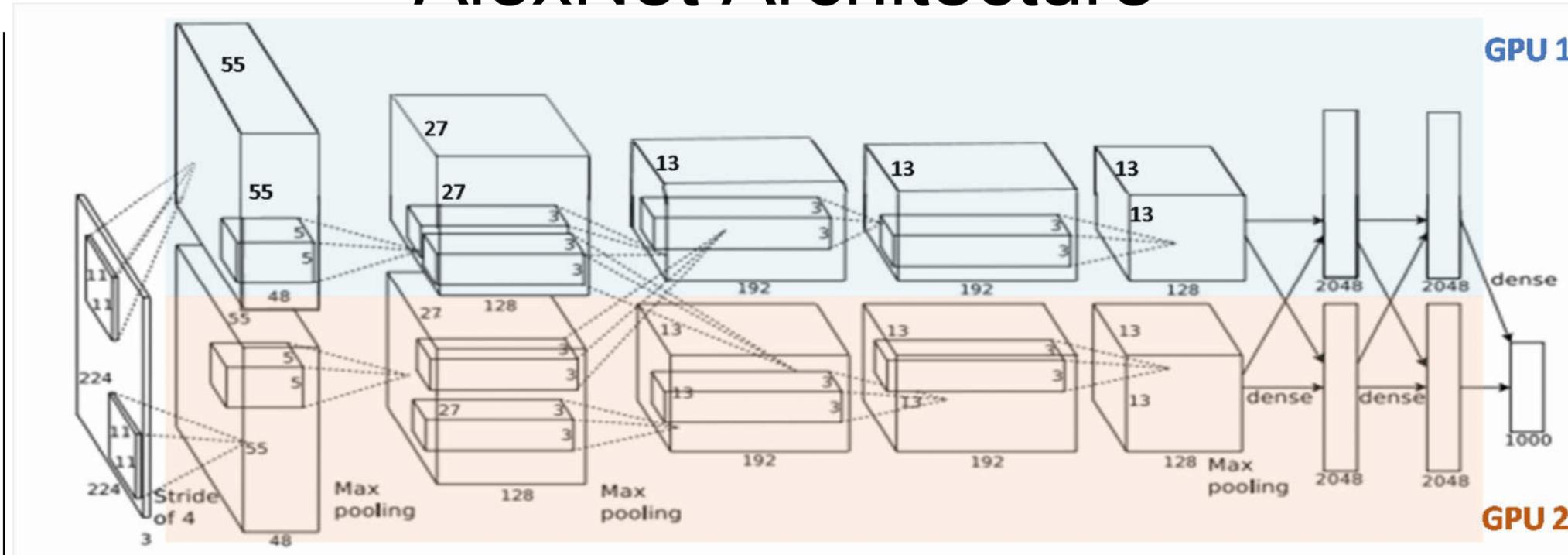


- Winner of ImageNet LSVRC-2010.
- Trained over 1.2M images using SGD with regularization.
- Deep architecture (60M parameters.)
- Optimized GPU implementation (cuda-convnet)

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

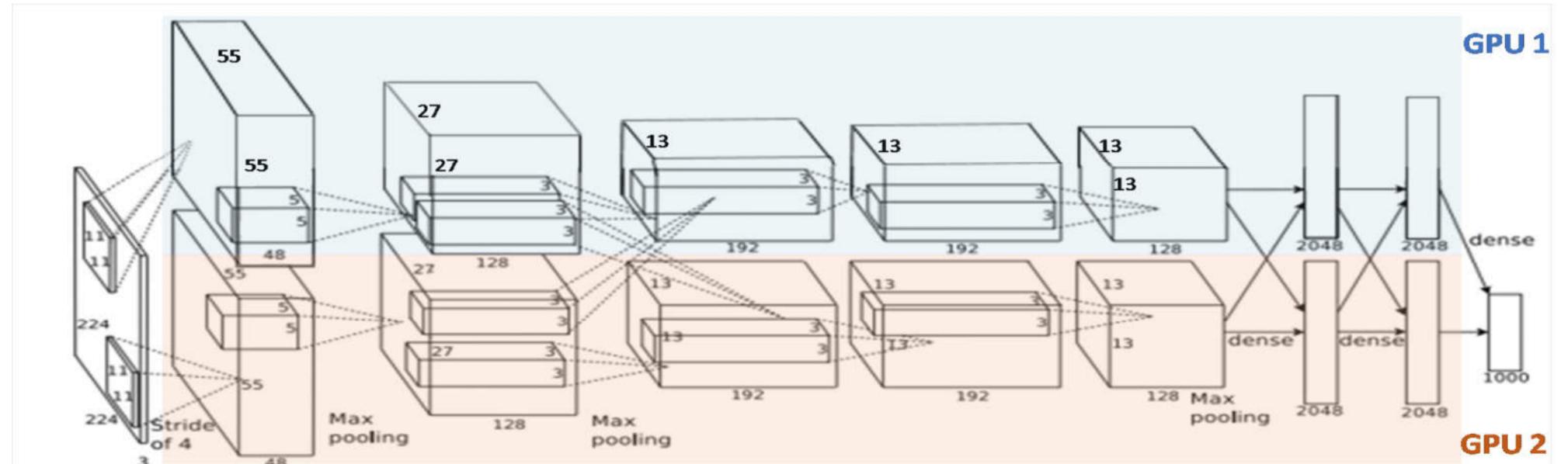


# AlexNet Architecture





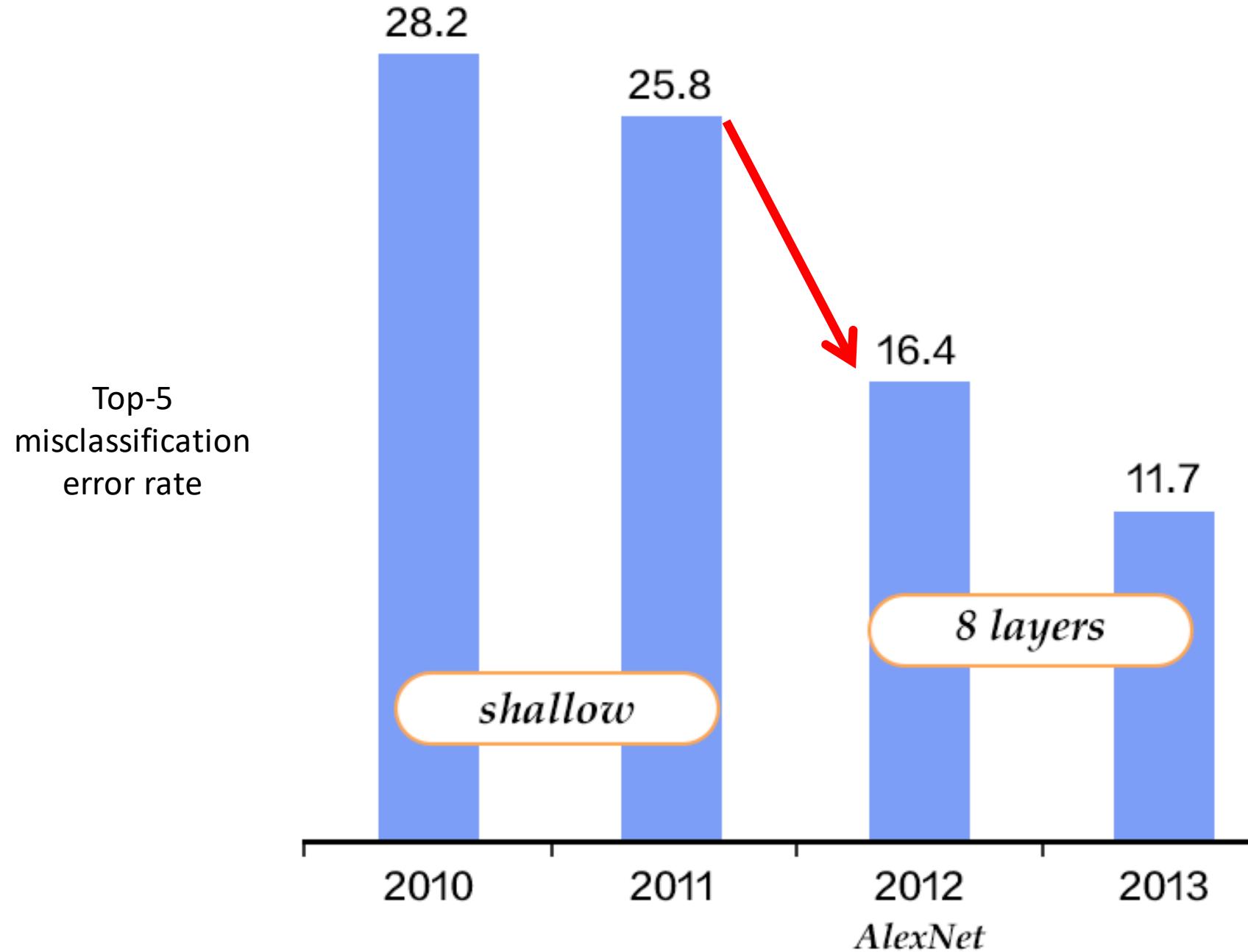
# AlexNet Architecture

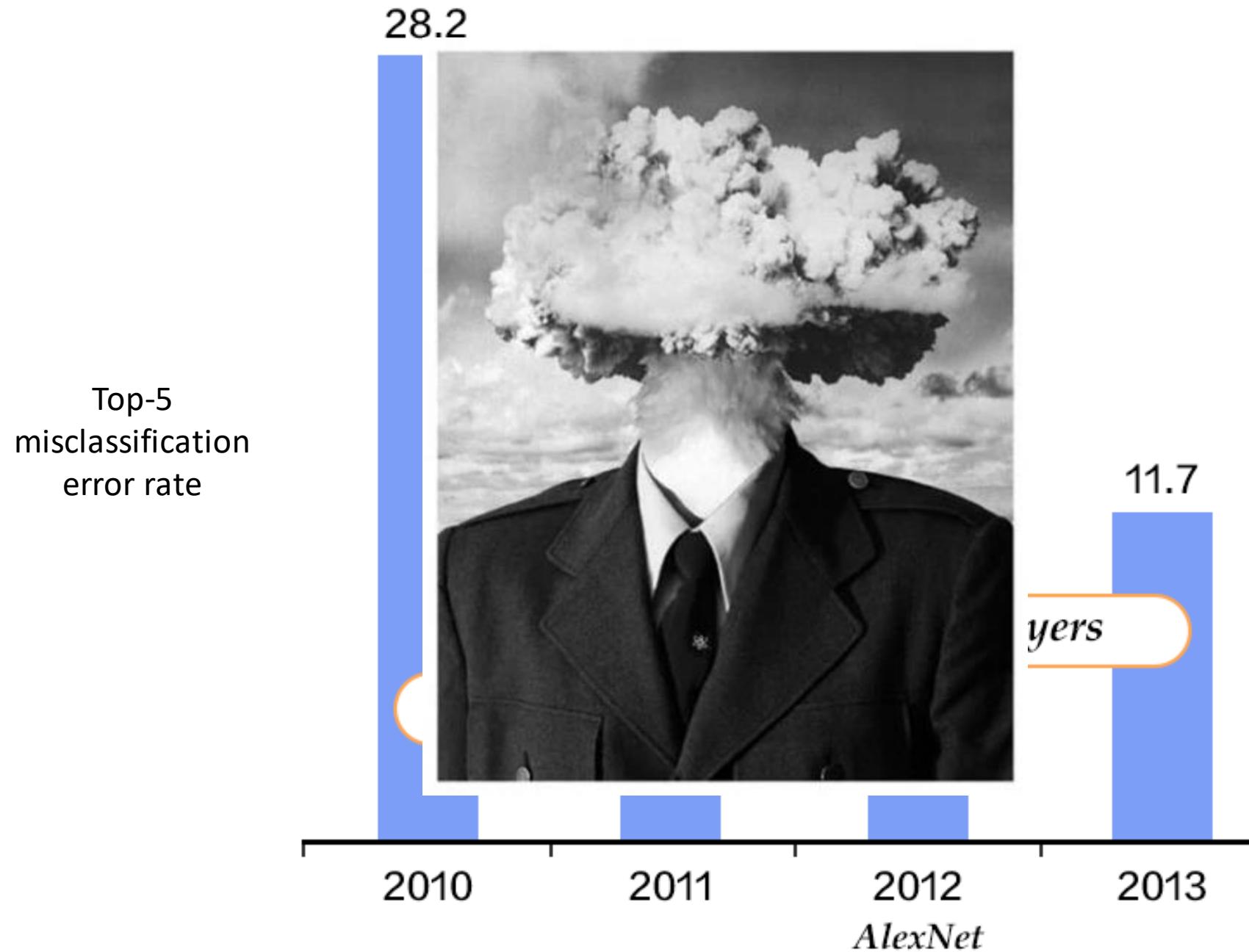


About 57 M parameters are in the fully connected layers

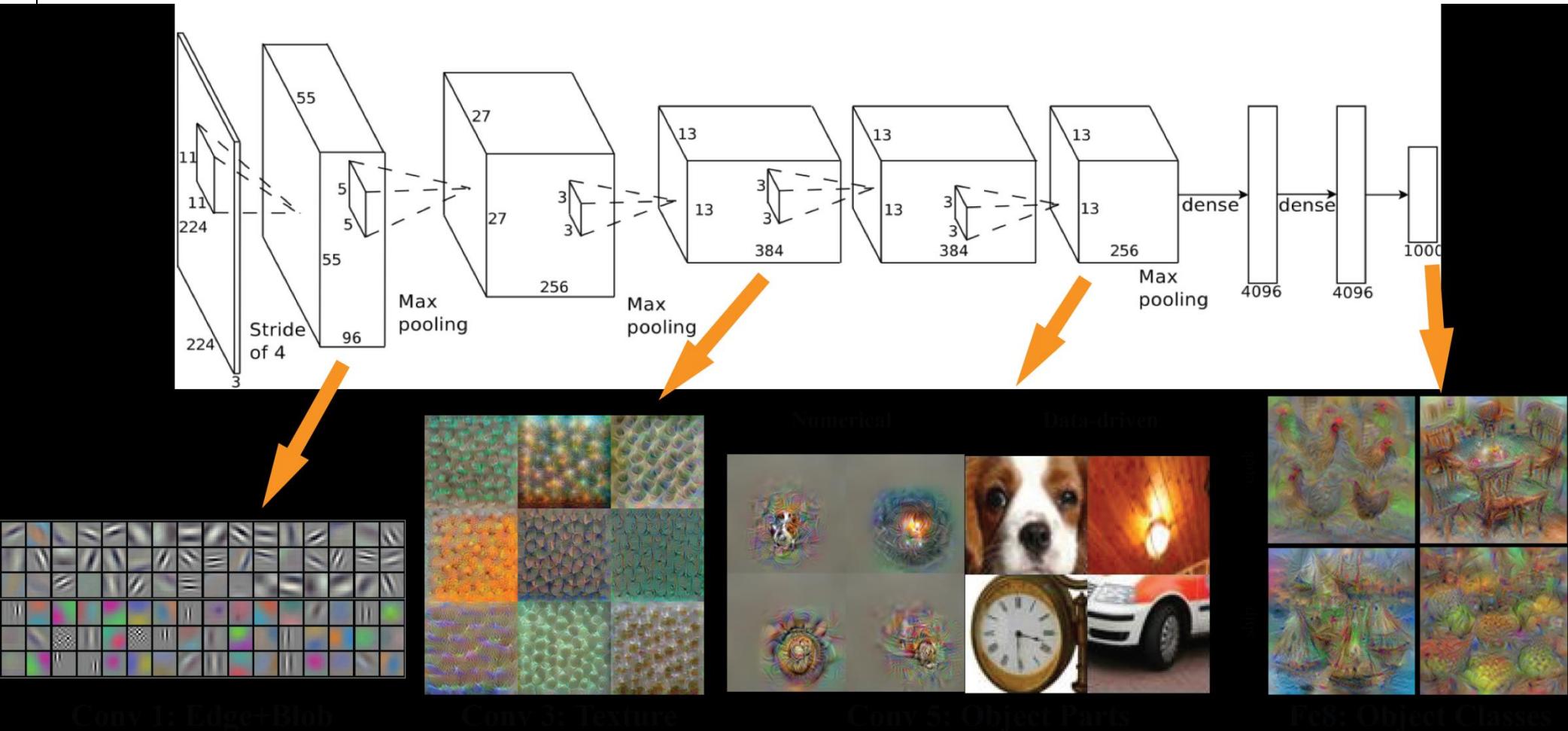
<b>Parameters :</b>	$[(11 \times 11 \times 3) + 1] \times 96 = 35 \text{ K}$	$[5 \times 5 \times 48] \times 256 = 307 \text{ K}$	$[3 \times 3 \times 256] \times 384 = 884 \text{ K}$	663 K	442 K	37 M	16 M	4 M	60 M
<b>Neurons :</b>	253,440	$27 \times 27 \times 256 = 186,624$	$13 \times 13 \times 384 = 64,896$	$13 \times 13 \times 384 = 64,896$	$13 \times 13 \times 256 = 43,264$	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.



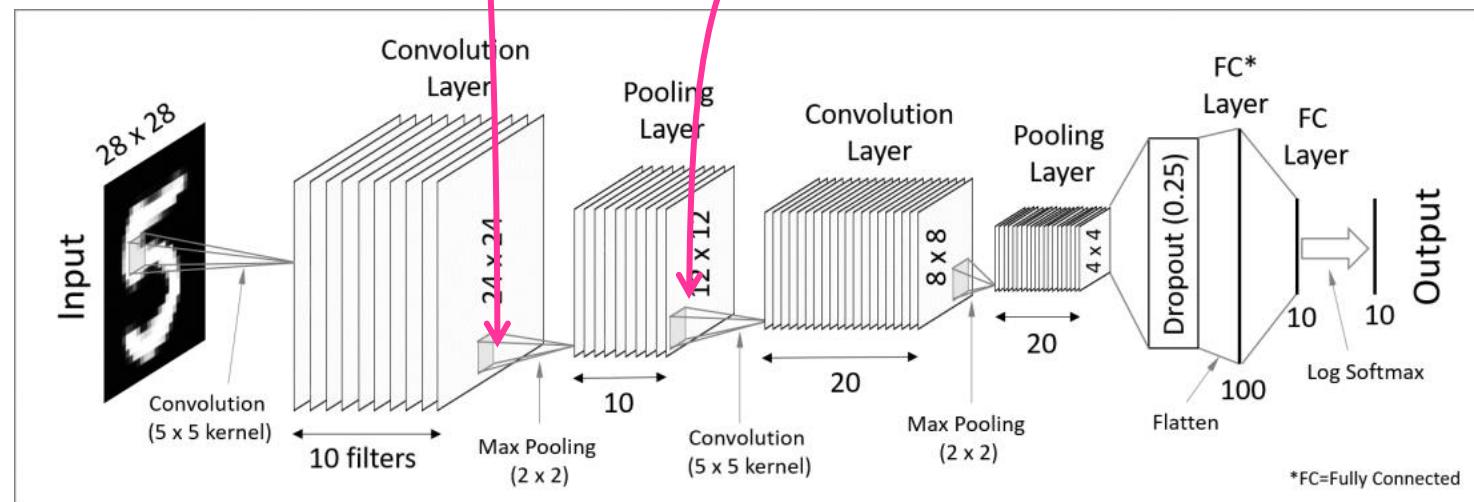
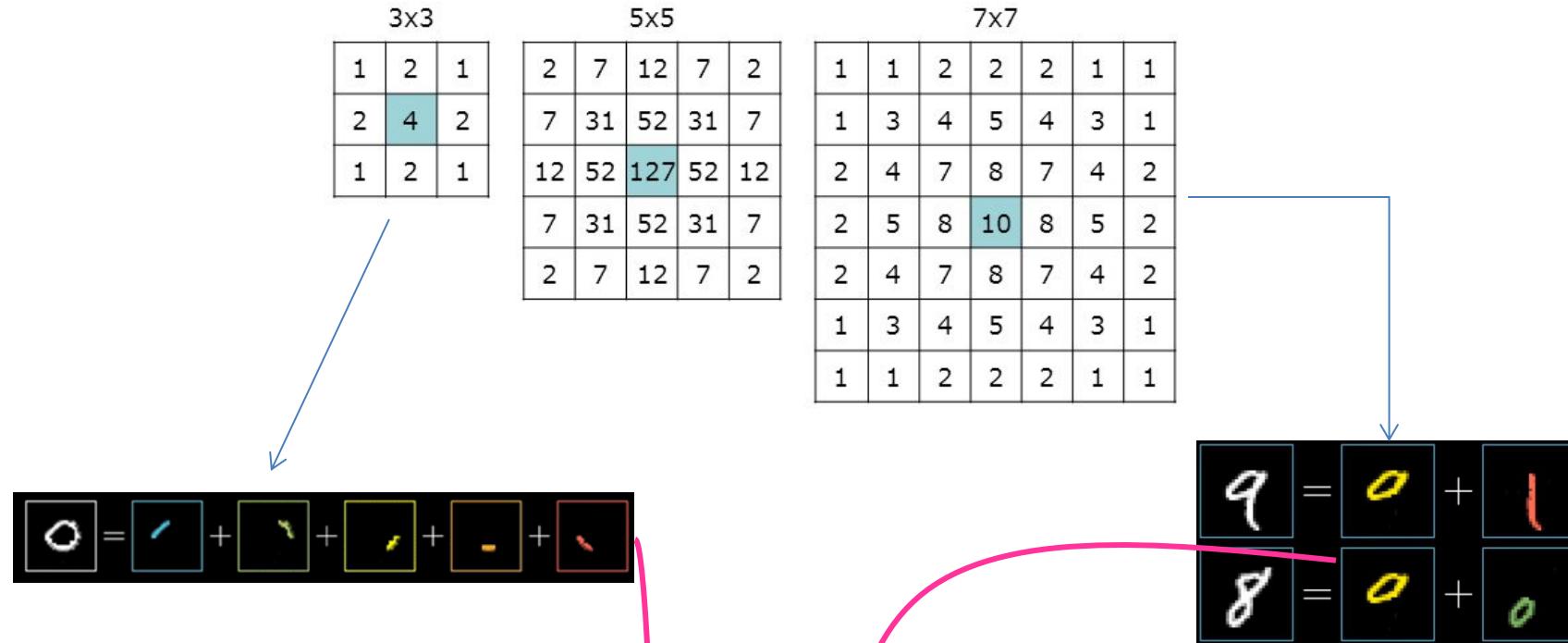


# Filters learnt by AlexNet





Committing to a single filter size at each layer seems somewhat rigid !

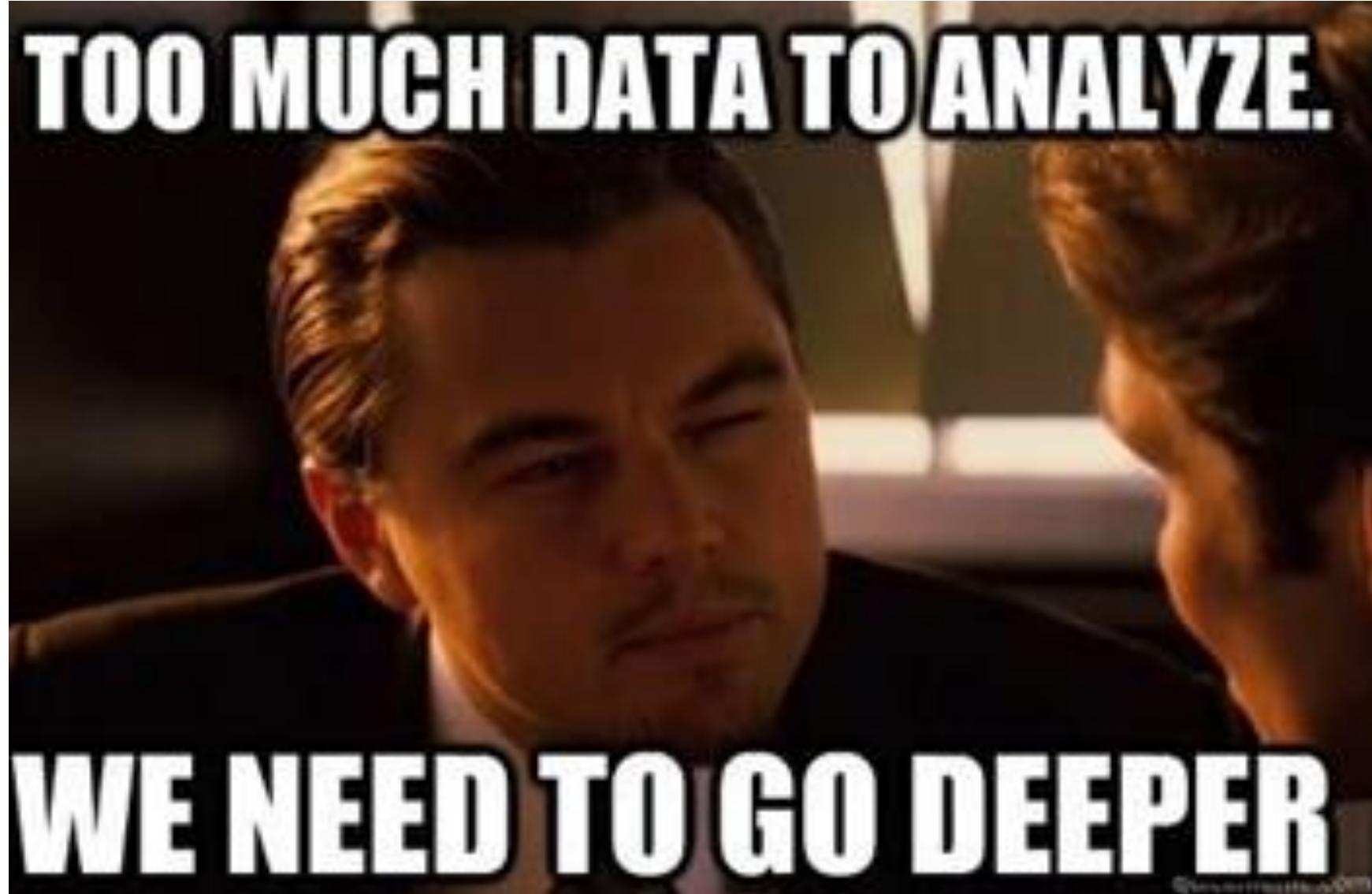




---

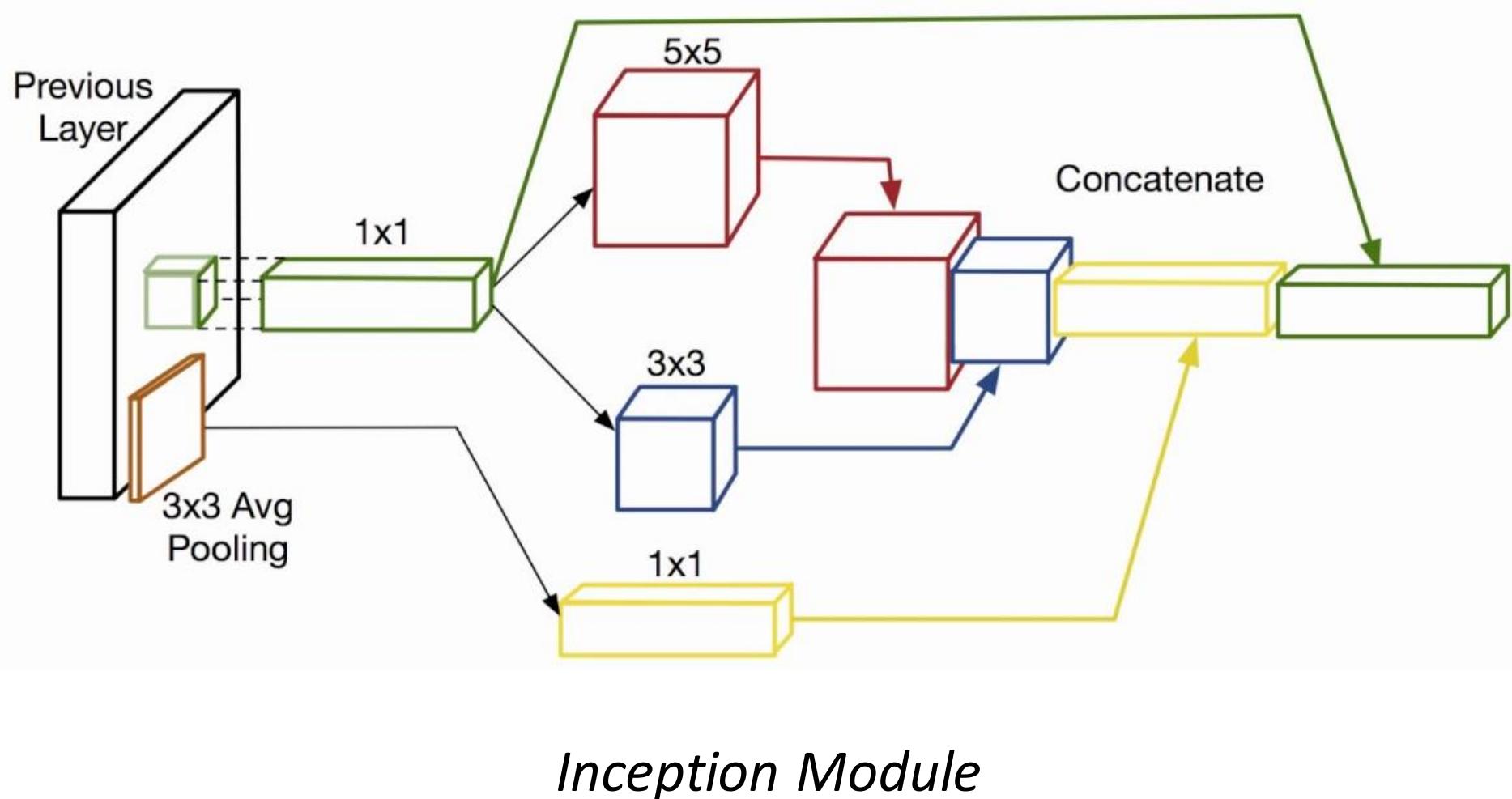
---

Why not have multiple levels (scales)  
of filtering ?





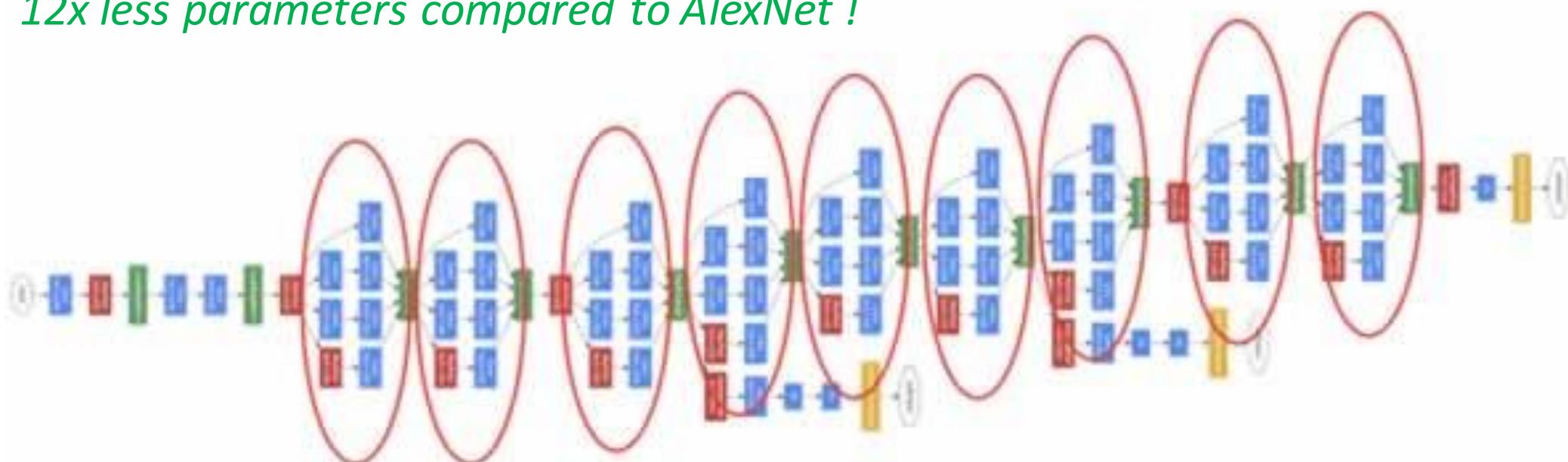
# Why not have multiple levels (scales) of filtering ?





# GoogLeNet

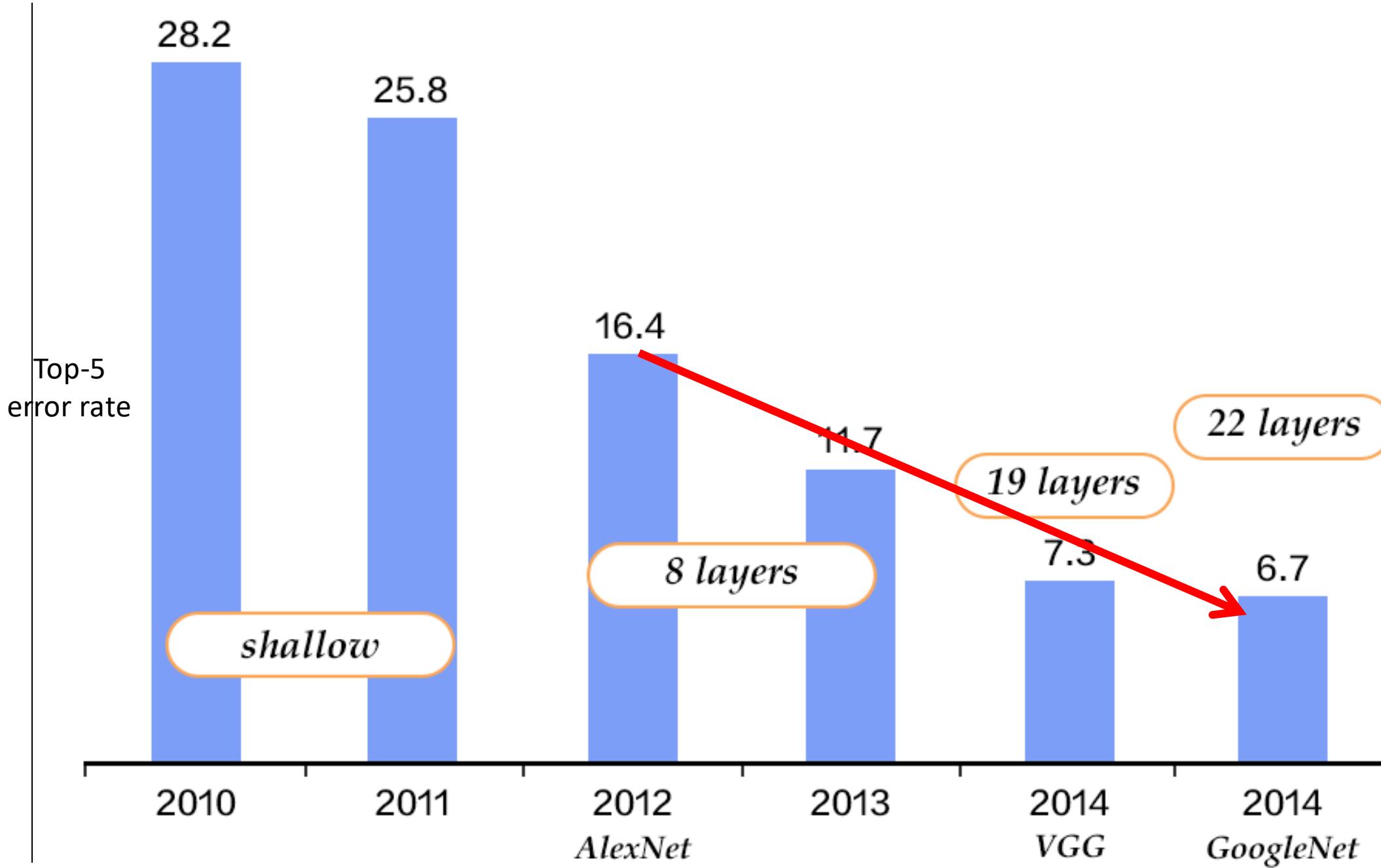
*12x less parameters compared to AlexNet !*



## 9 Inception modules

Network in a network in a network...

Convolution  
Pooling  
Softmax  
Other





---

# How deep can the network get ?

- 50 layers ? 100 ? 1000 ?

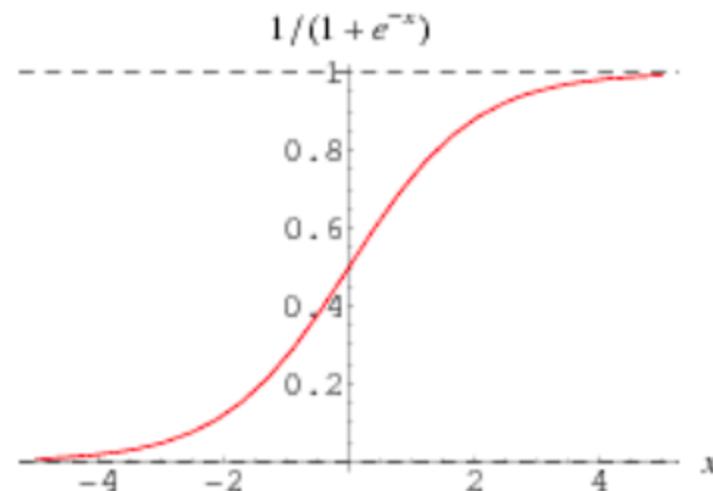


# How deep can the network get ?

- 50 layers ? 100 ? 1000 ?
- Issue: Vanishing gradients

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

Sigmoid



Plot of Sigmoid function



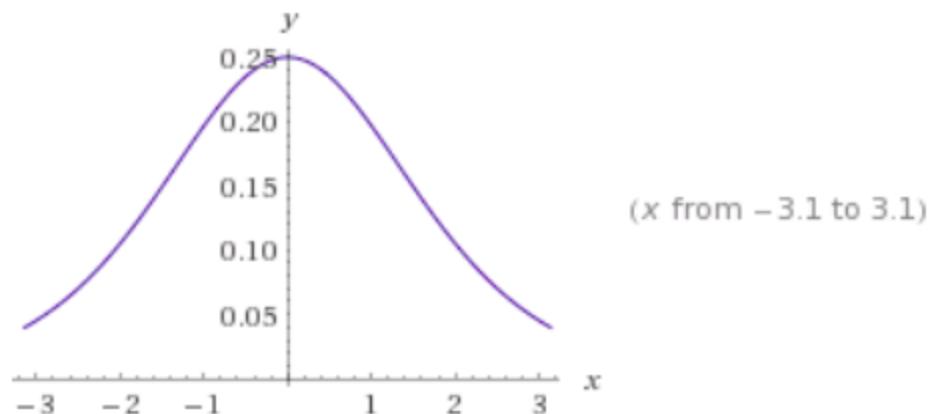
---

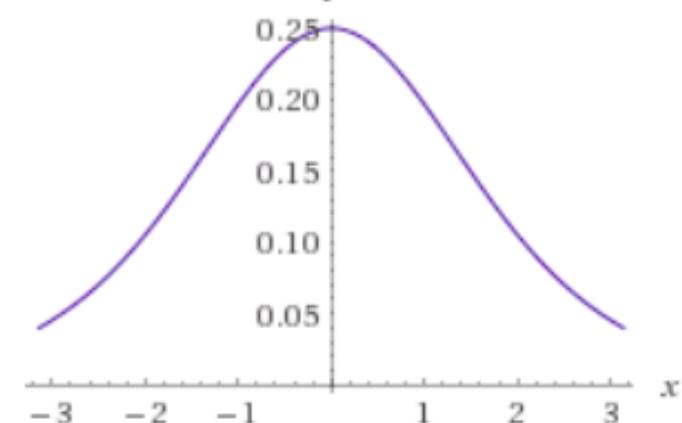
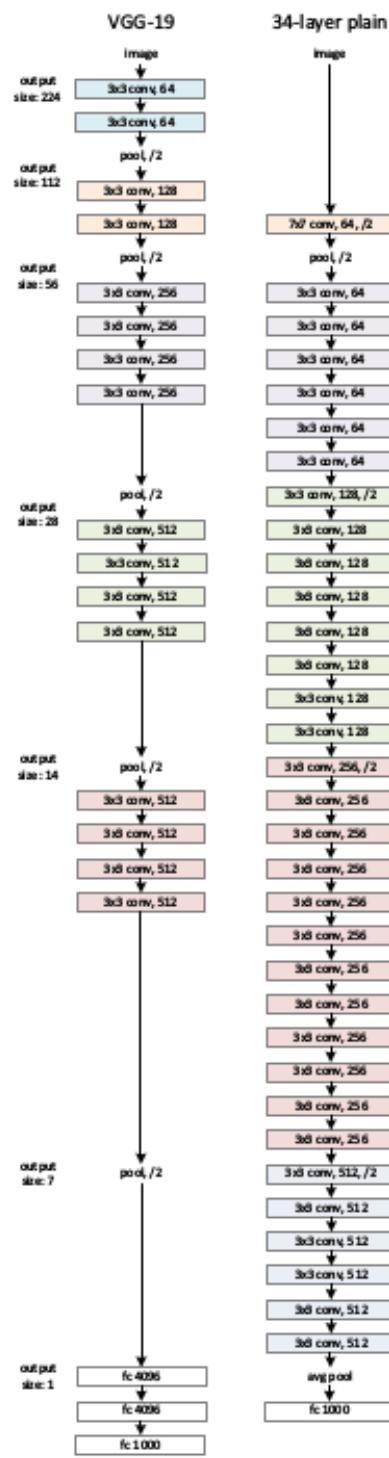
# Vanishing gradients

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

$$\sigma'(x) = \sigma(1 - \sigma)$$

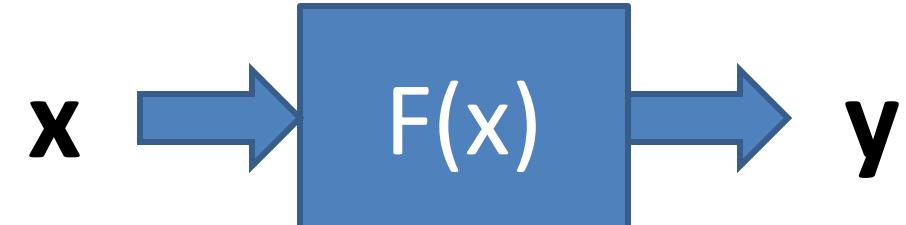
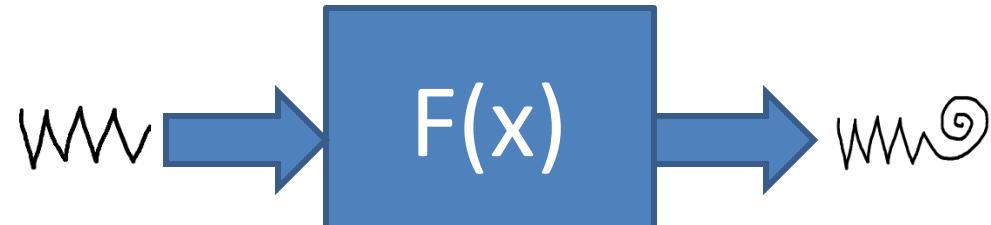
Derivative of sigmoid can be represented as





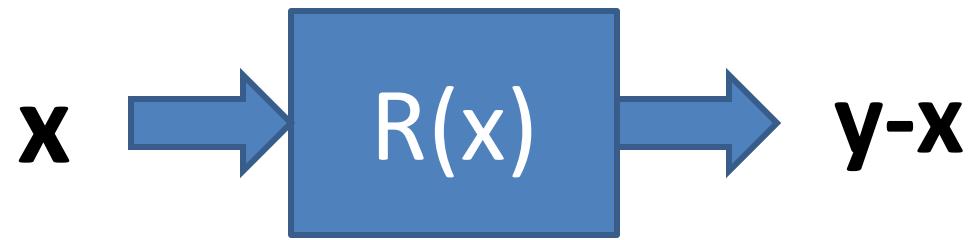
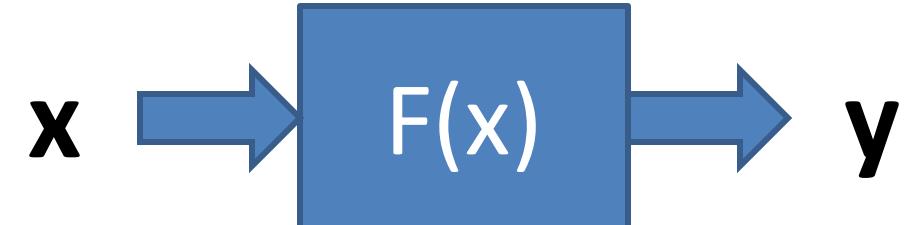
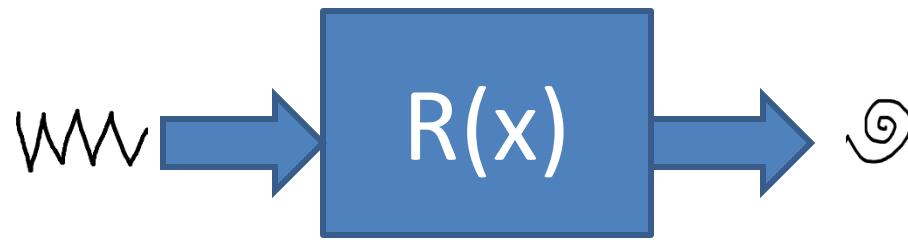


# Residual Networks – A crayon analogy



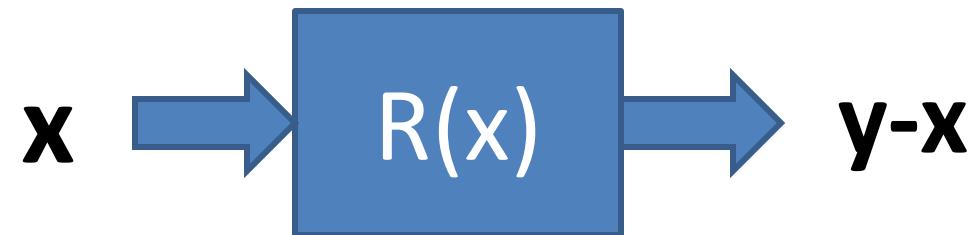
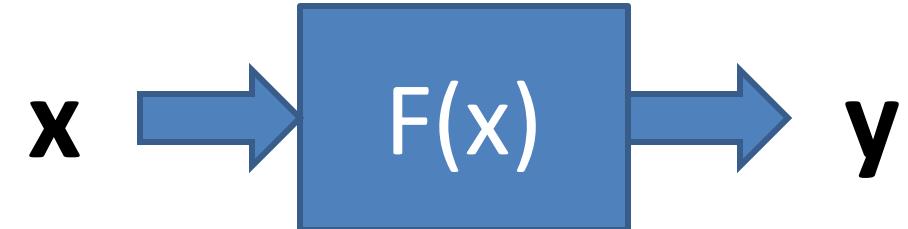
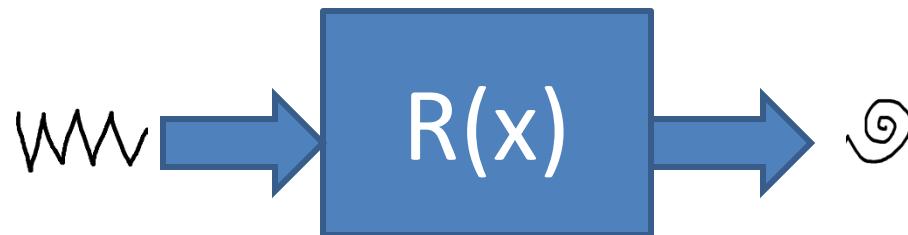


# Residual Networks – A crayon analogy





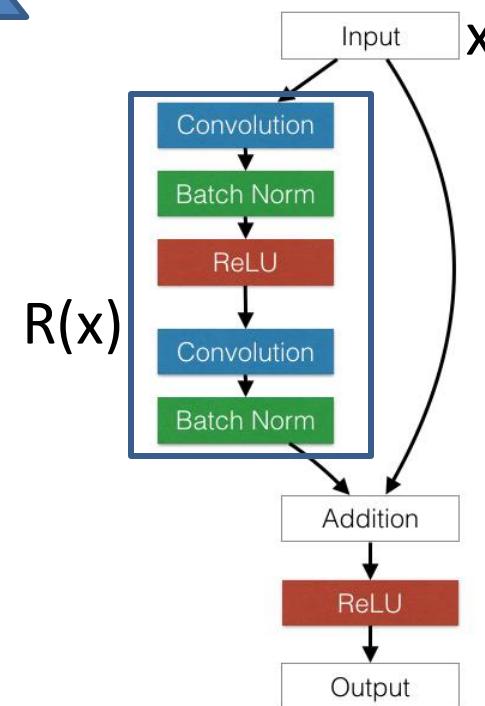
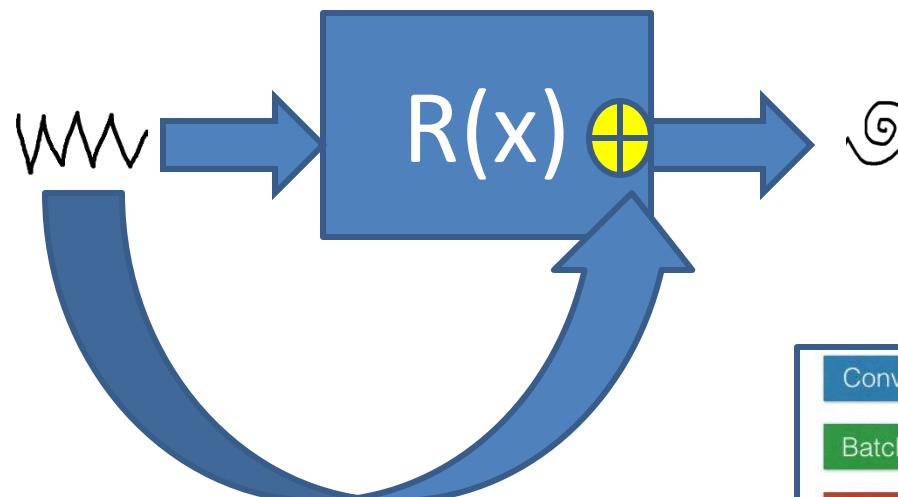
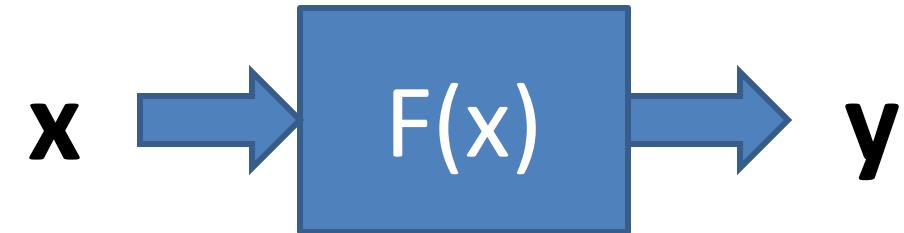
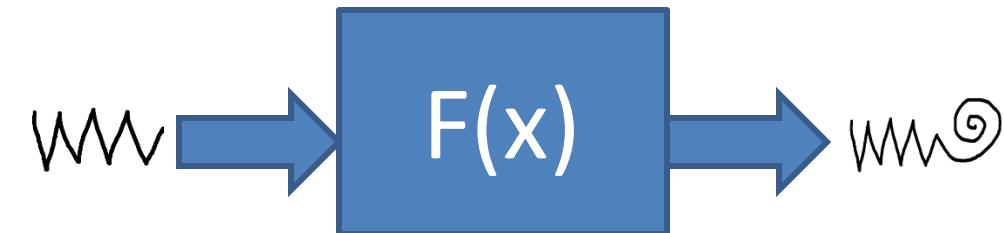
# Residual Networks – A crayon analogy



$$y = F(x) = x + R(x)$$



# Residual Networks – A crayon analogy

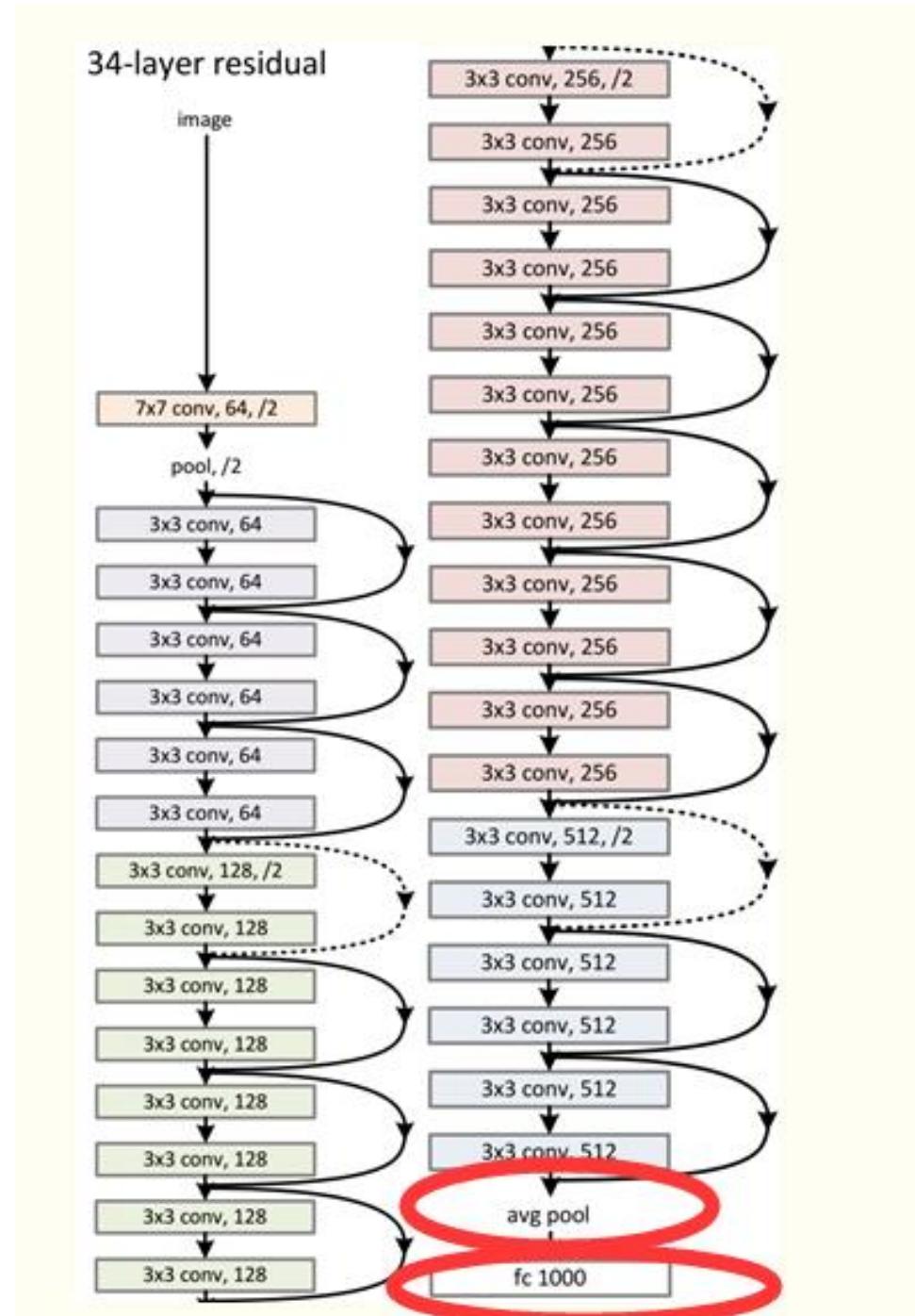


*Architecture of a residual block*

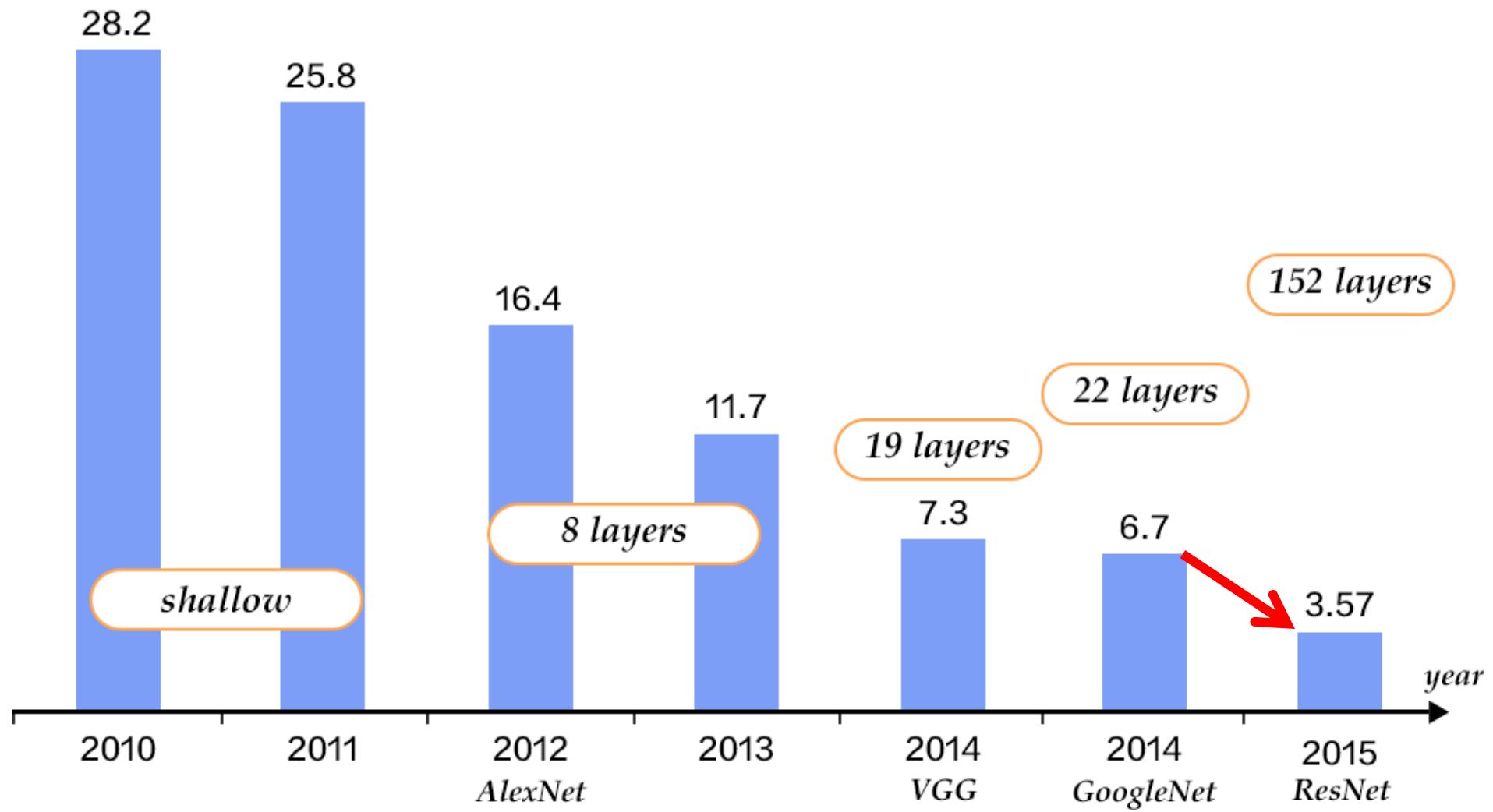
$$F(x) = x + R(x)$$



# Residual Networks



*Gradients can be supplied to shallower layers directly if needed (via skips)*



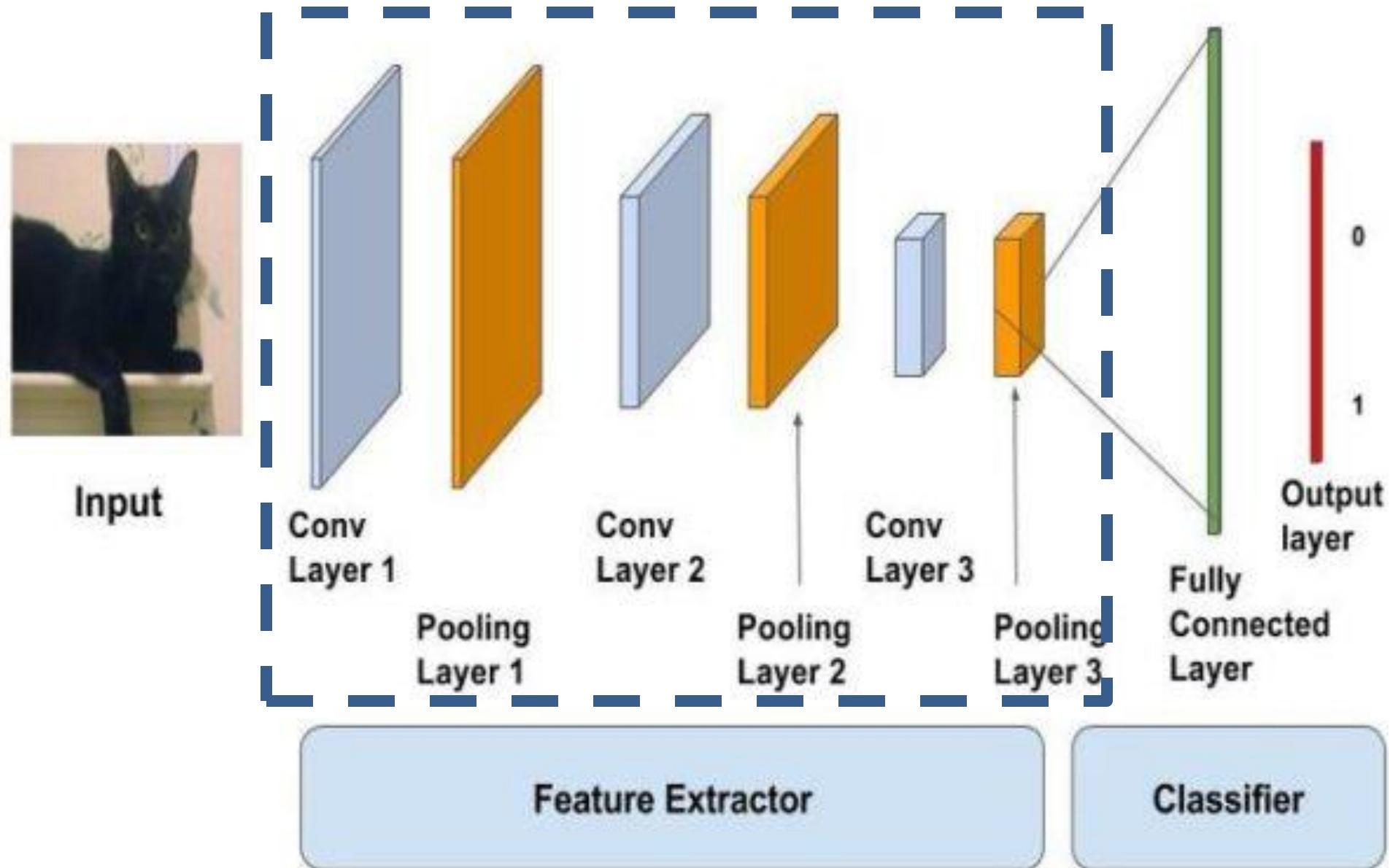


---

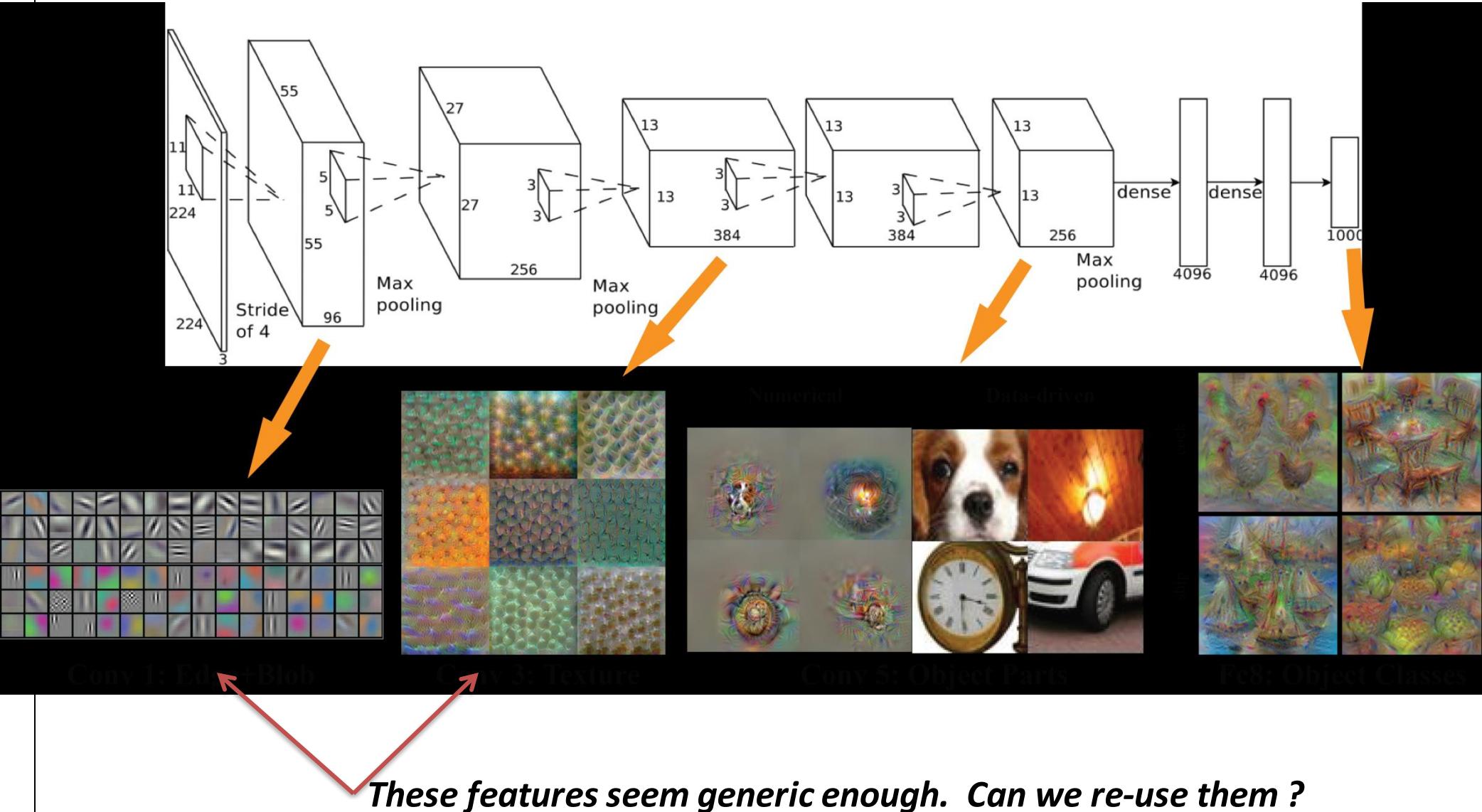
# Landscape of CNNs : Applications and Architectures



# Transfer Learning



# Filters learnt by AlexNet





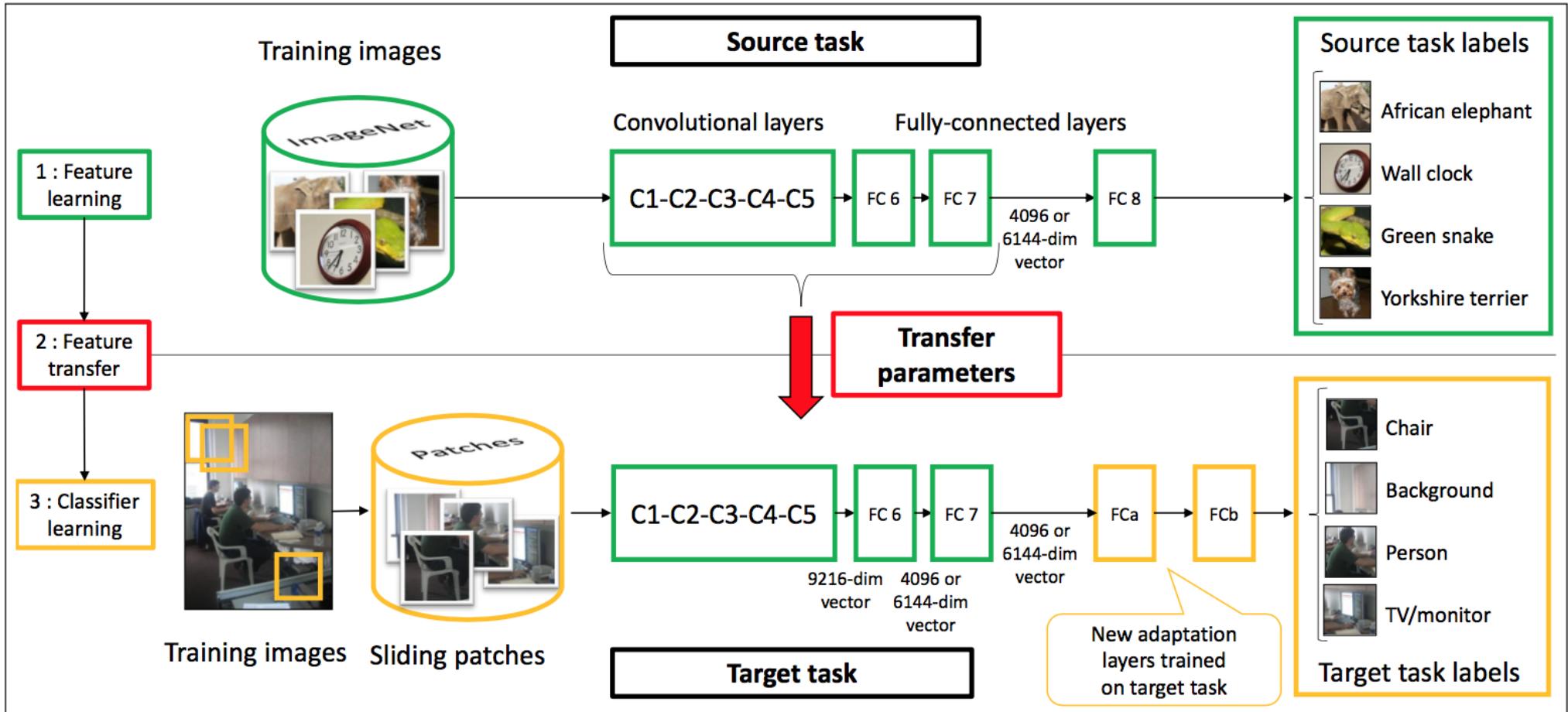
---

# Transfer Learning





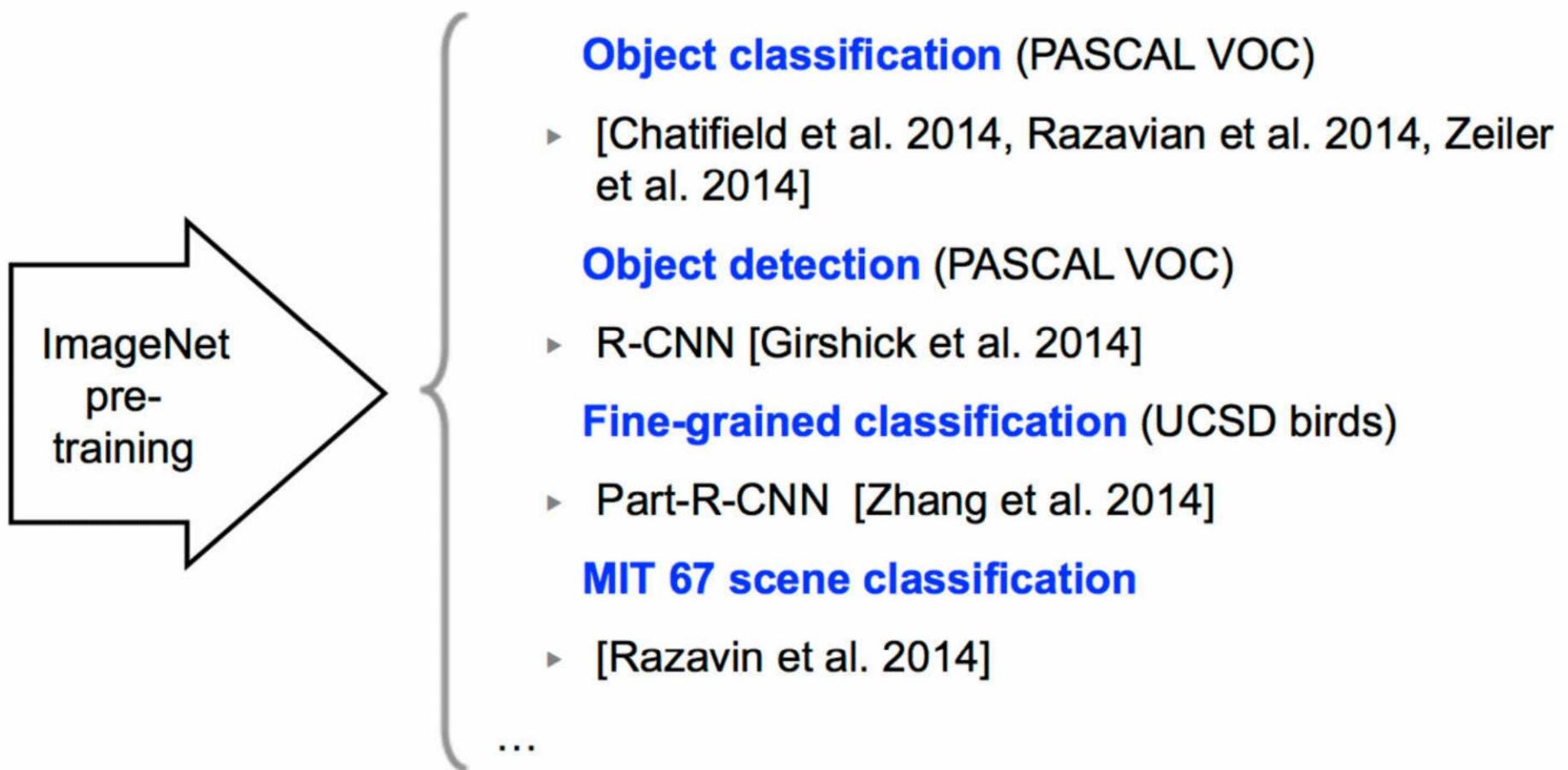
# Transfer Learning





# Examples

Pre-trained features are quite general





# Semantic segmentation



Input



Segmentation [9]



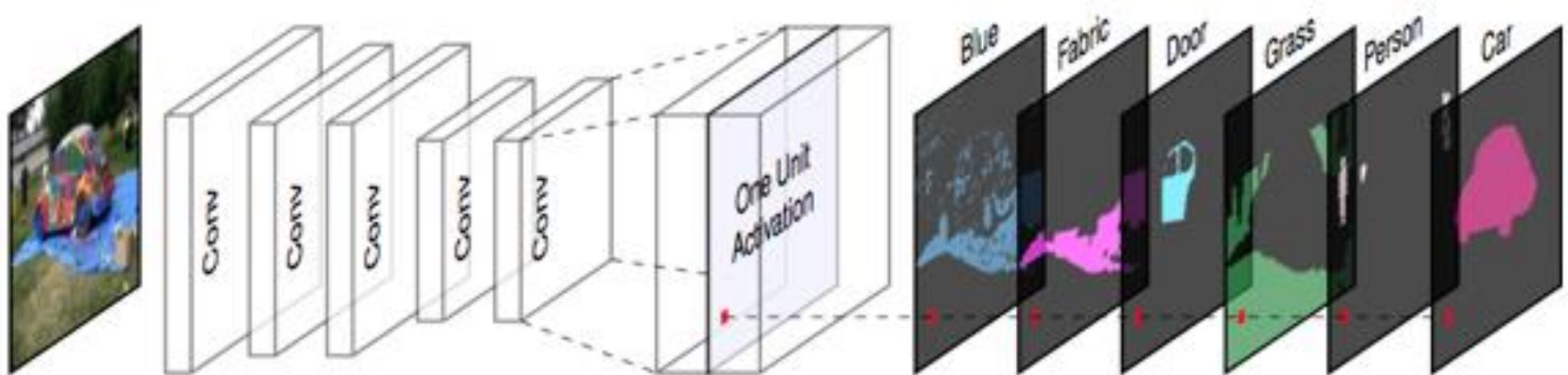
---

# Semantic segmentation





# Semantic segmentation



<http://netdissect.csail.mit.edu/thumb/slides-thumbnail.png>



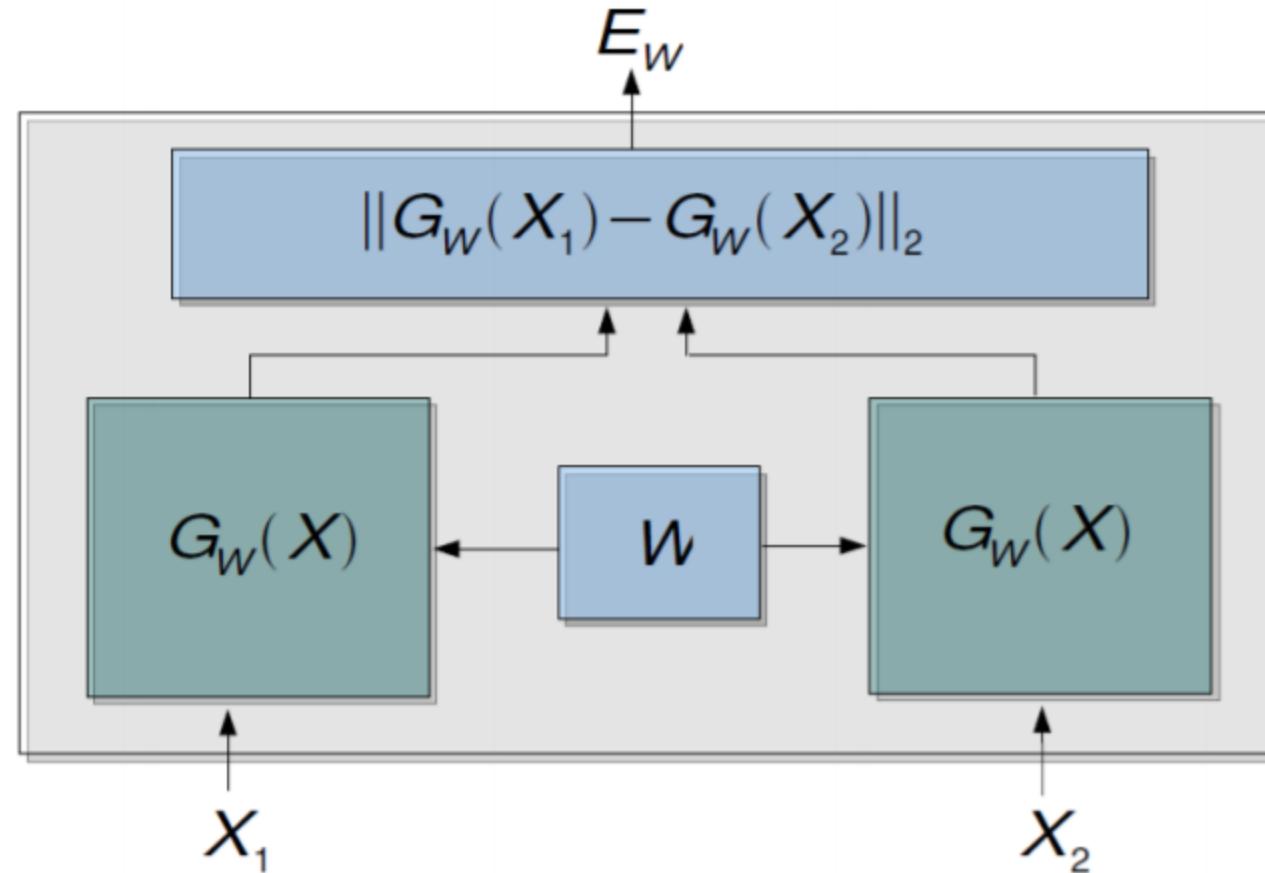
---

# Siamese Network





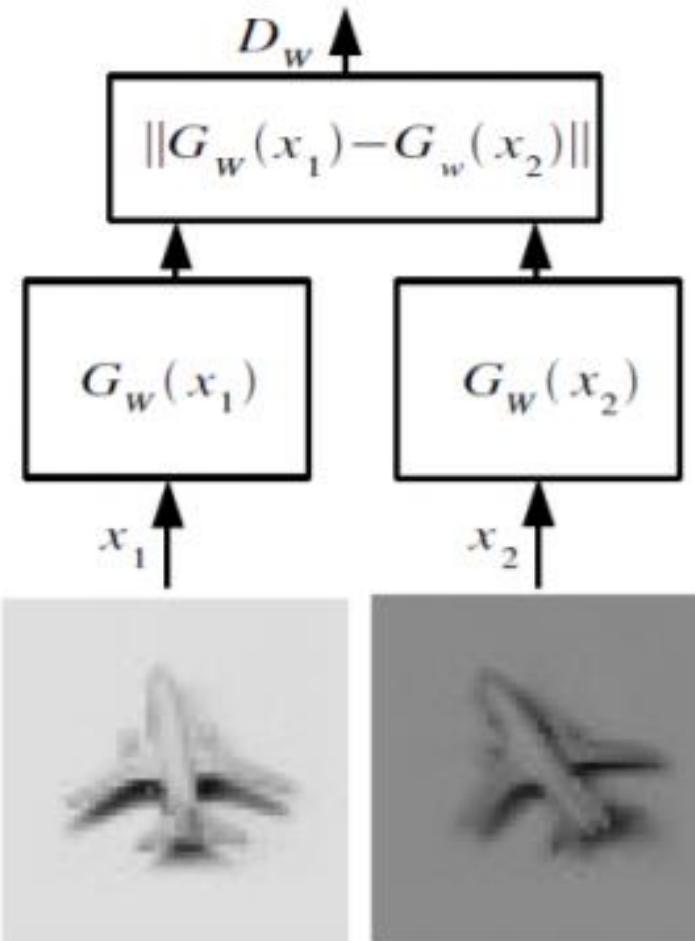
# Siamese Network





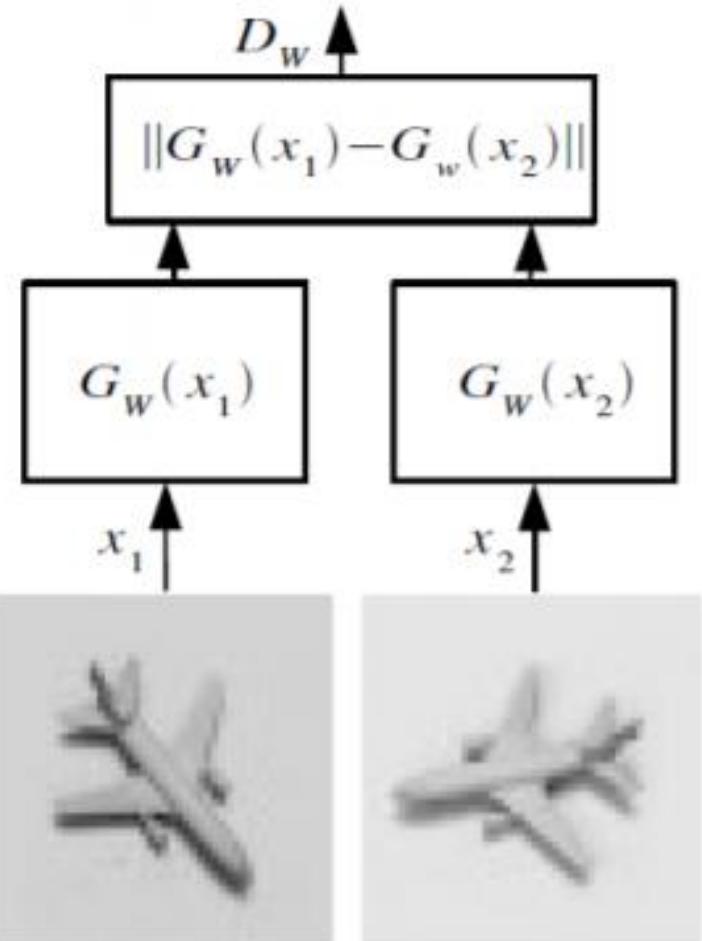
# Siamese Network

Make this small



Similar Images

Make this large



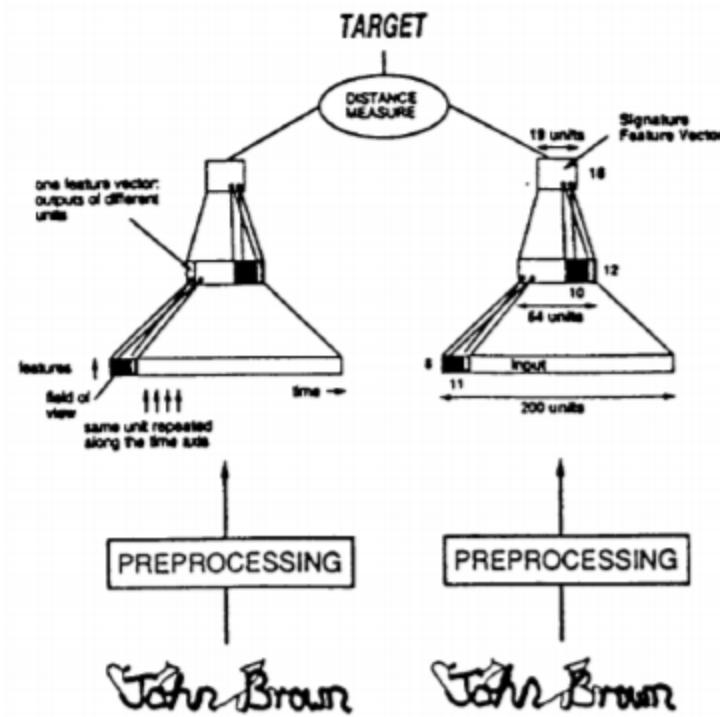
Dissimilar Images



# Siamese Network

## *Application in Signature Verification*

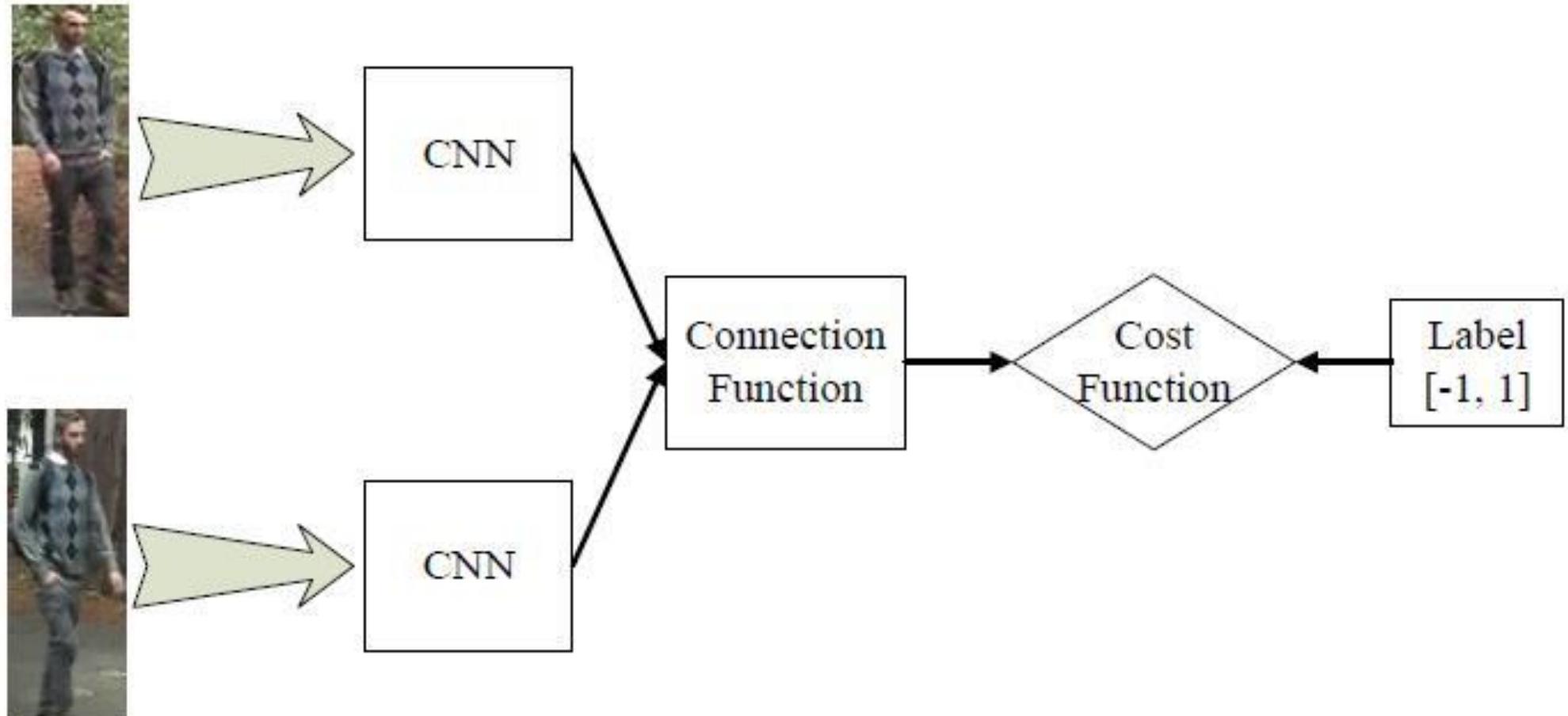
- The input is 8(feature)  $\times$  200(time) units.
- The cosine distance was used, (1 for genuine pairs, -1 for forgery pairs )



Bromley J, Guyon I, Lecun Y, et al. Signature Verification using a "Siamese" Time Delay Neural Network, NIPS Proc. 1994



# Siamese Network (Person re-id)



[http://www.fubin.org/research/Person\\_ReID/Person\\_ReID.html](http://www.fubin.org/research/Person_ReID/Person_ReID.html)



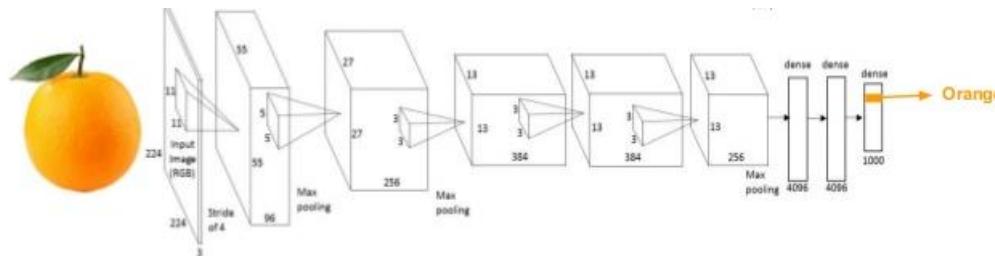
---

# Landscape of CNNs : Architectures

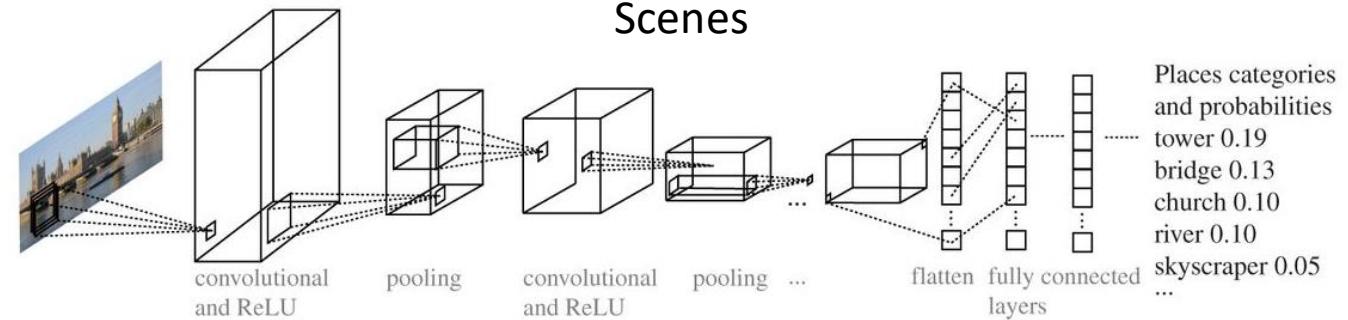
# Image → Label



## Objects

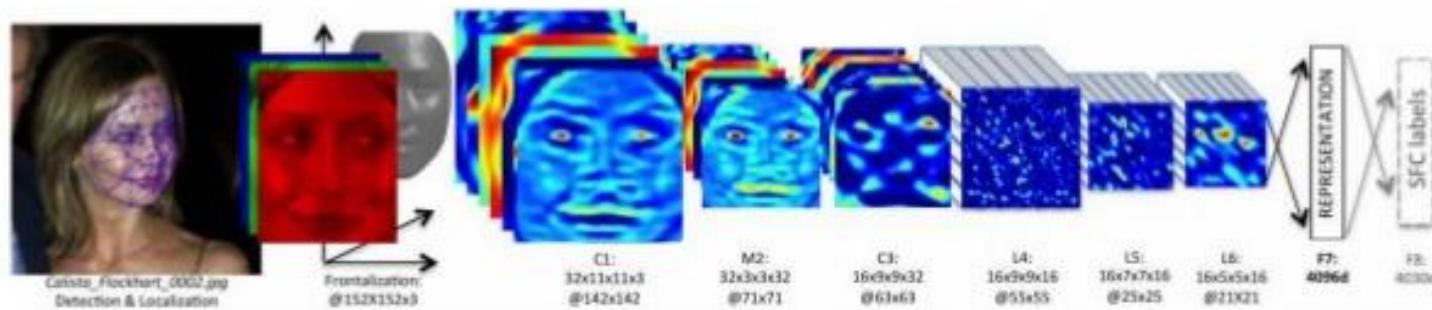


## Scenes



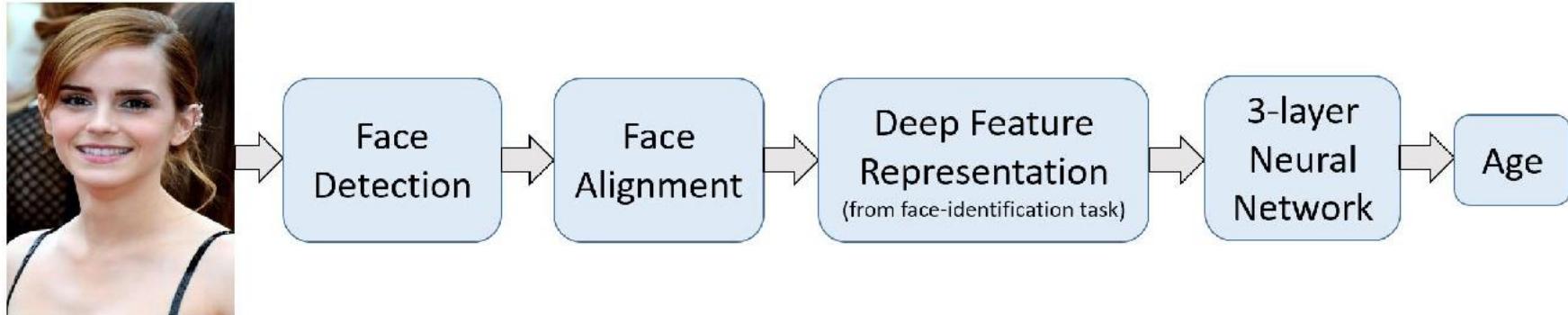
## Faces

### DeepFace Architecture

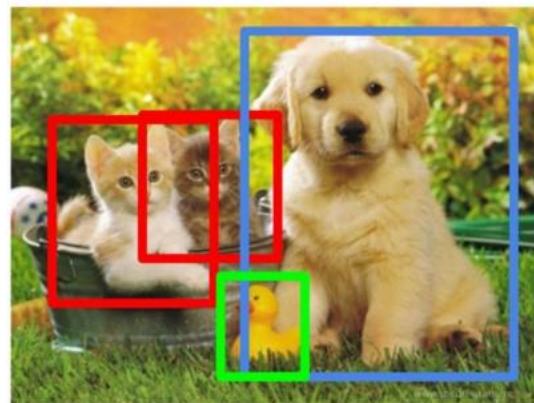




# Image → Number



## Object Detection

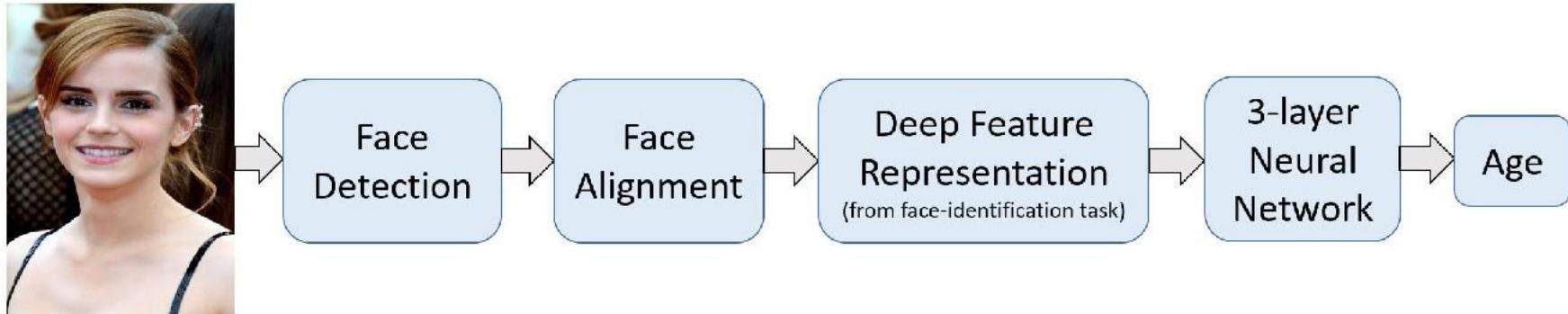


CAT, DOG, DUCK

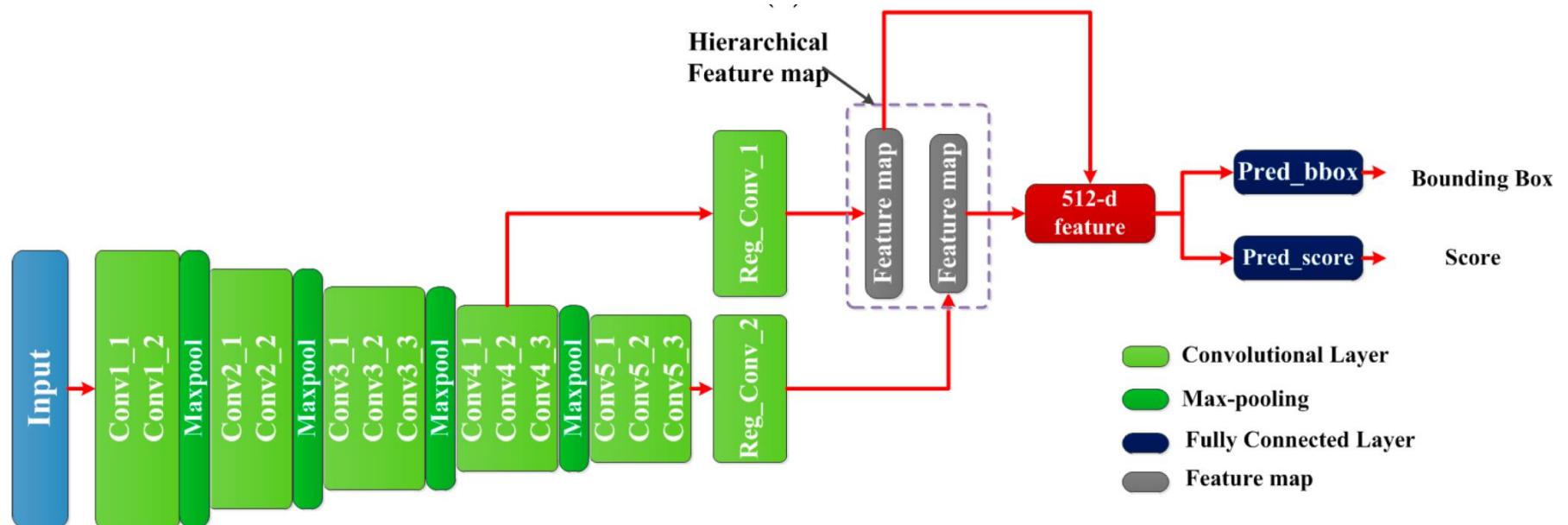


# Image → Number

## Age Estimation



## Object detection

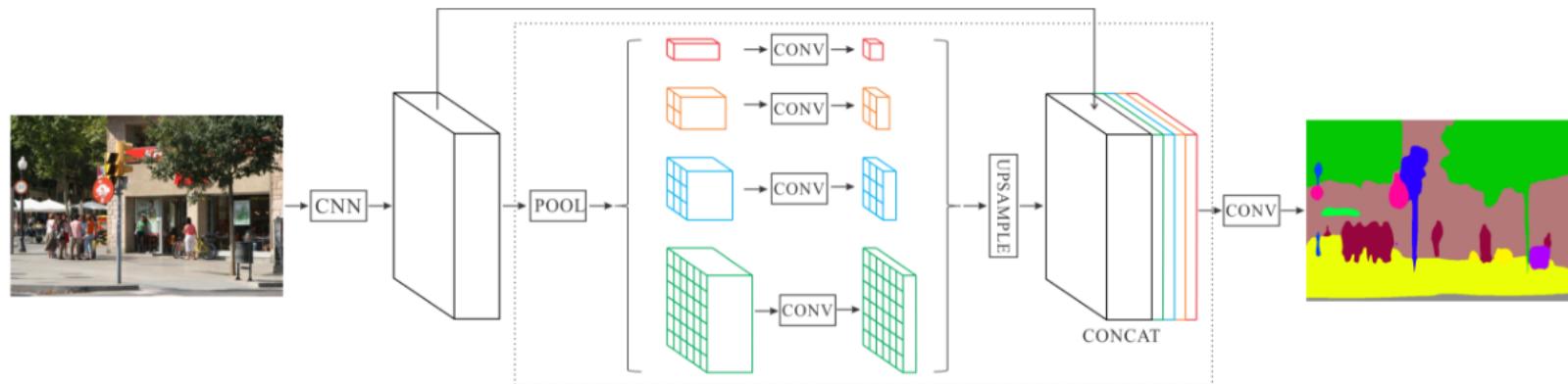




# Image → Label Image

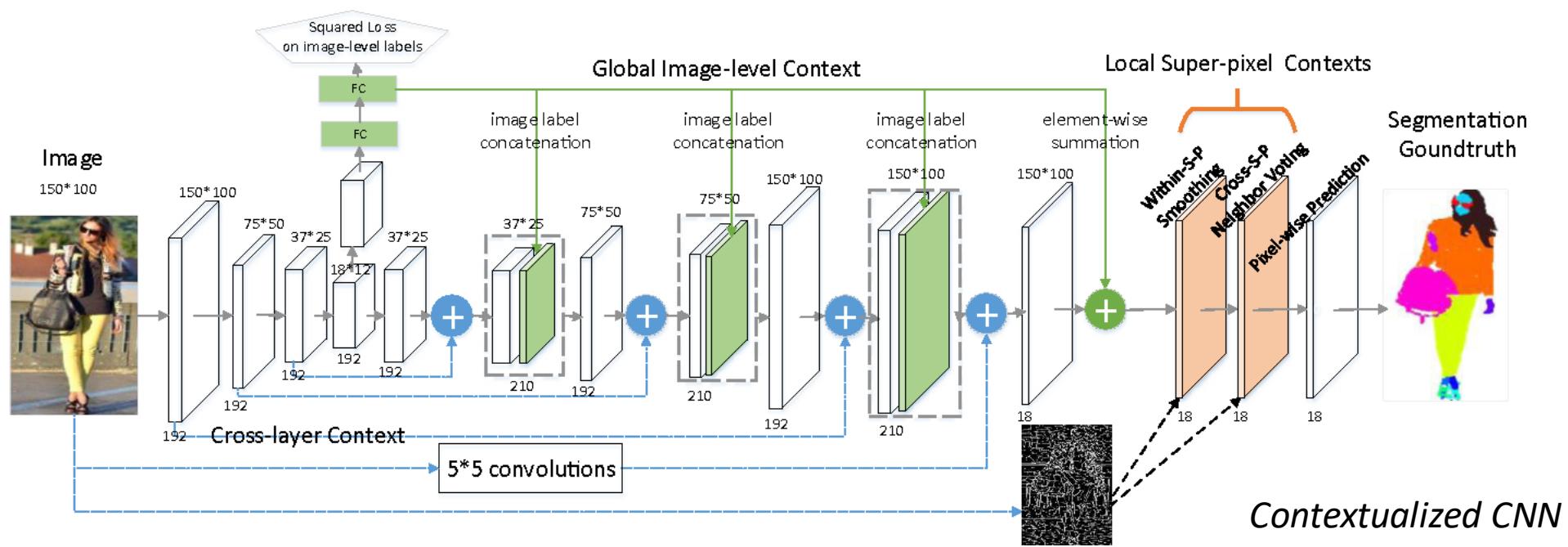


## Scene Parsing



PSPNet

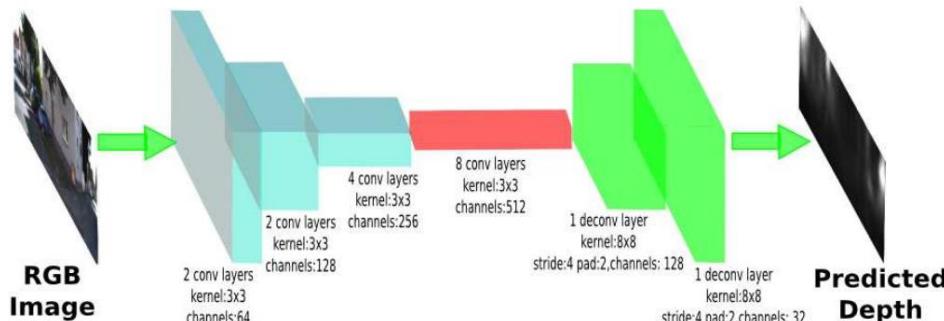
## Object Parsing



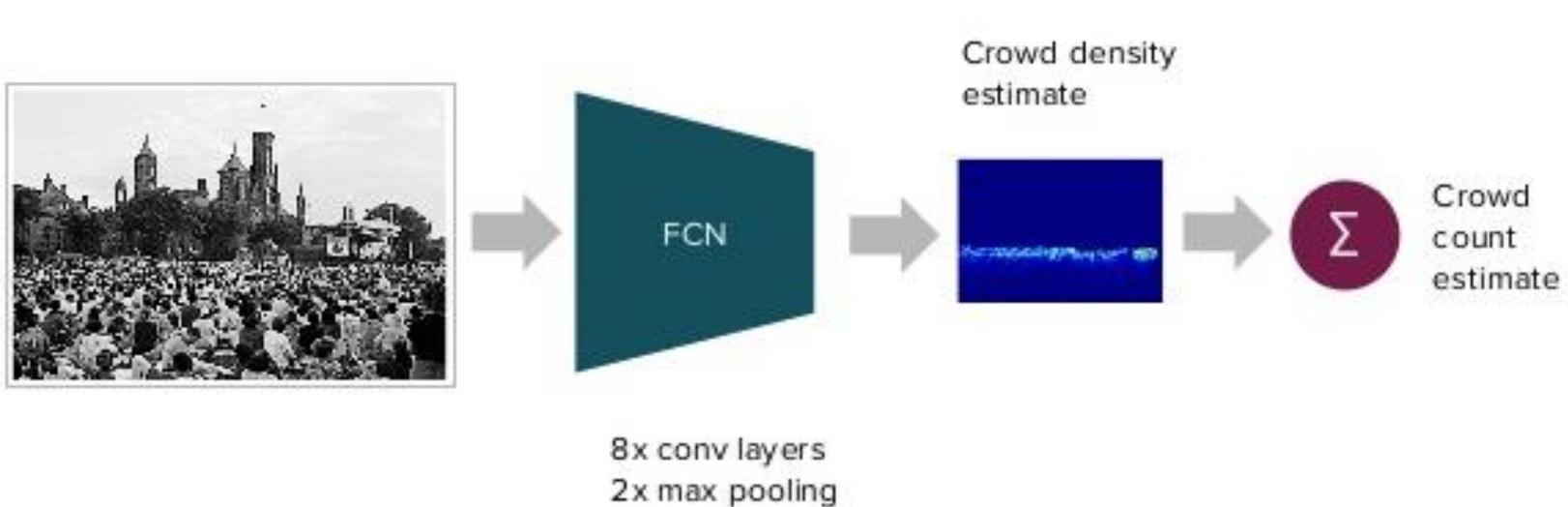
# Image → Image



## Depth Estimation



## Crowd Counting



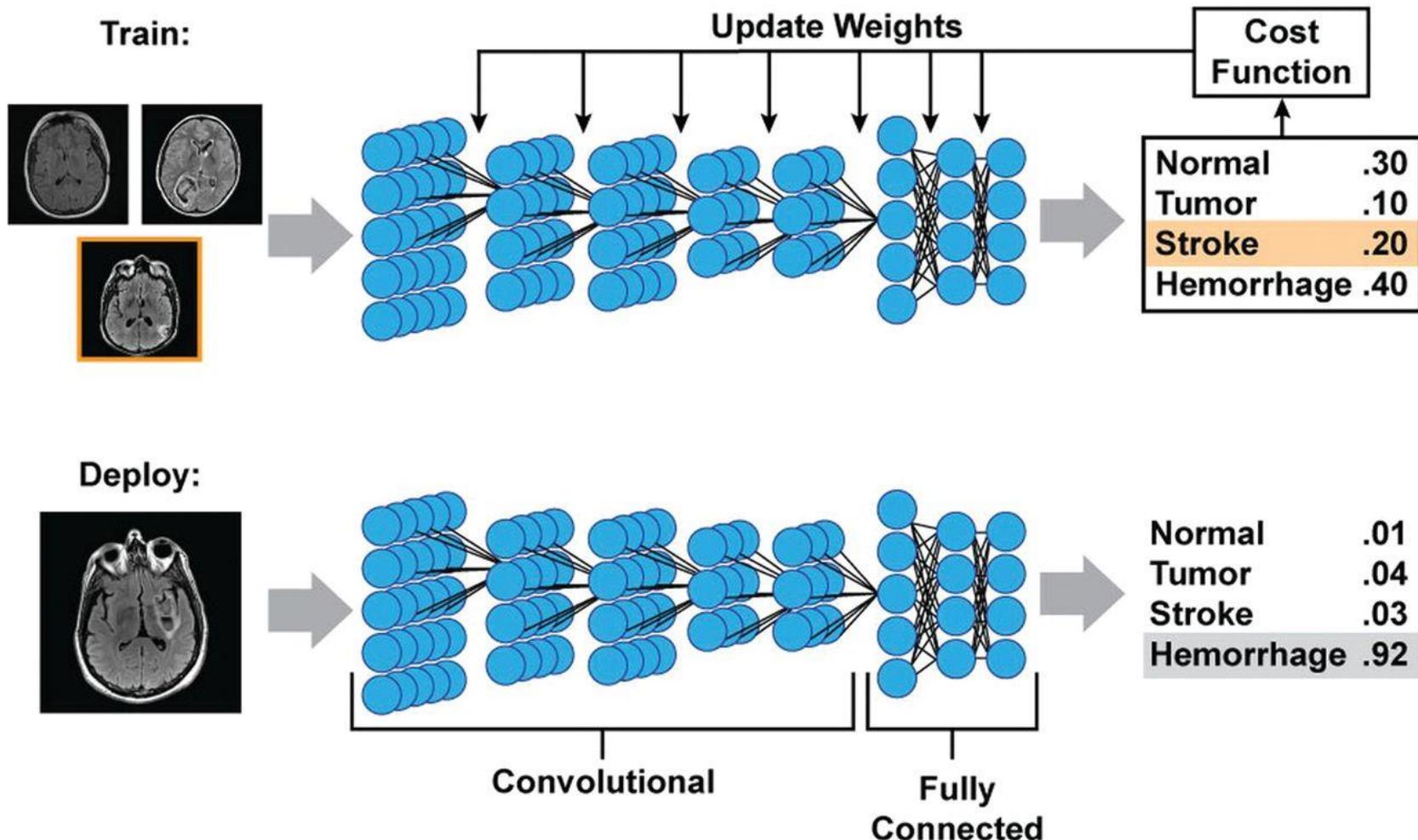


# Multi-channel input



NeuroRadiology (fMRI)

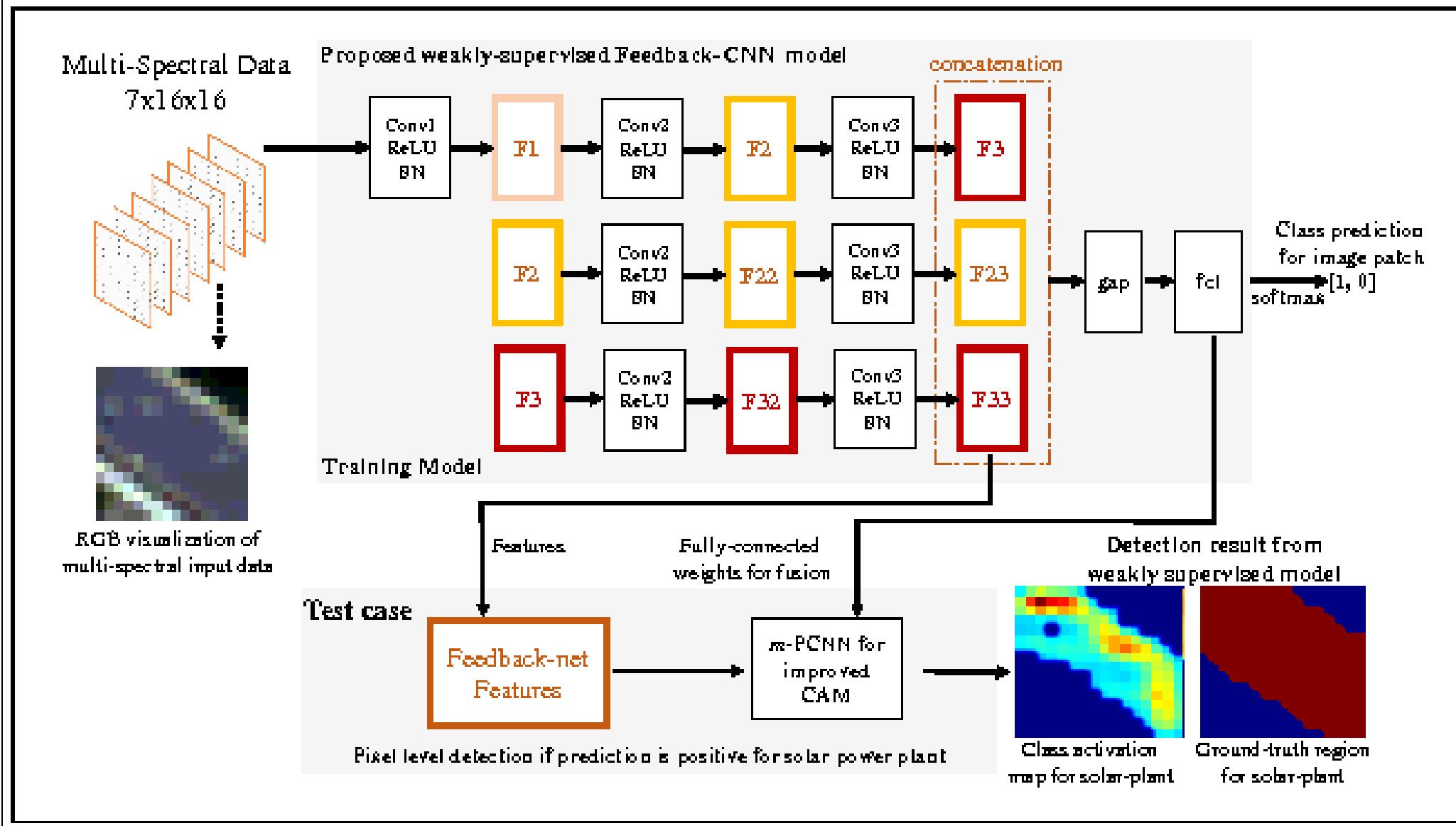
## Convolutional Neural Networks





# Multi-channel input

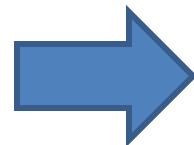
Solar-power plant detection from multi-spectral data



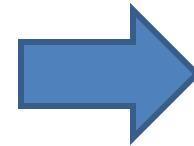


# Multi-branch input

Image Colorization



SOUTHREPORT.COM



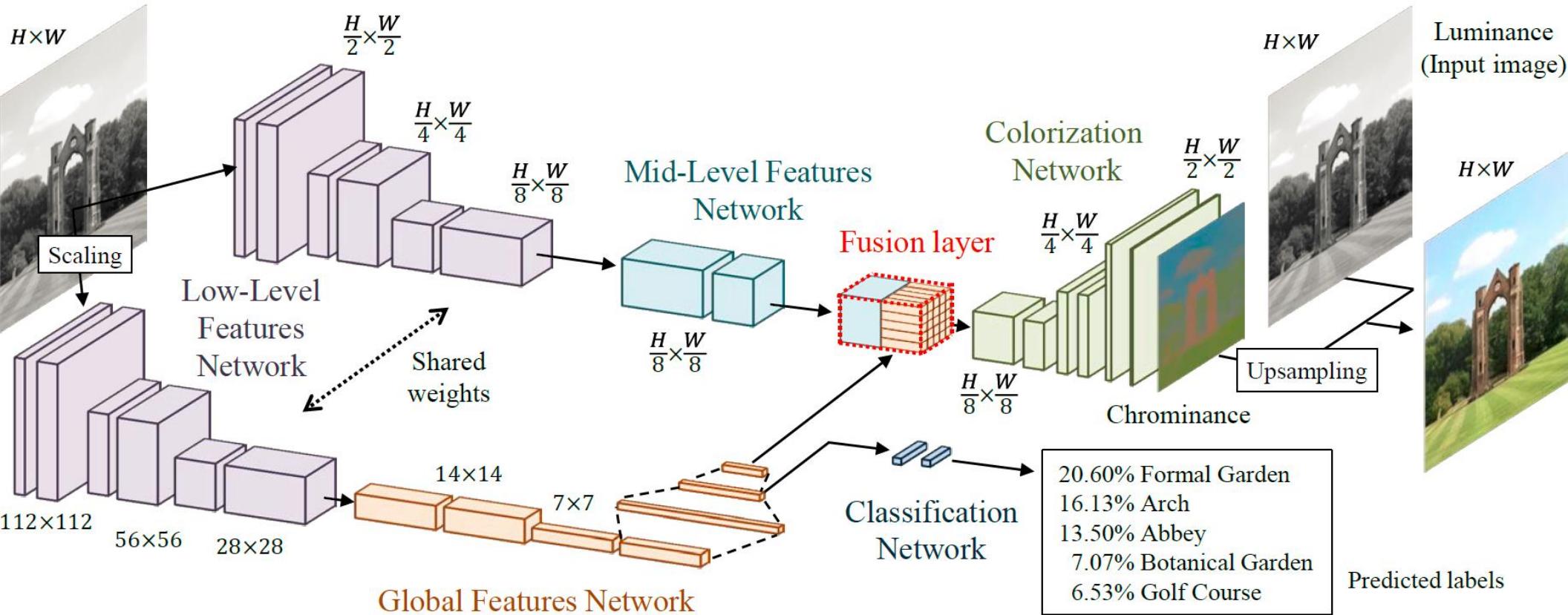
SOUTHREPORT.COM





# Multi-branch input

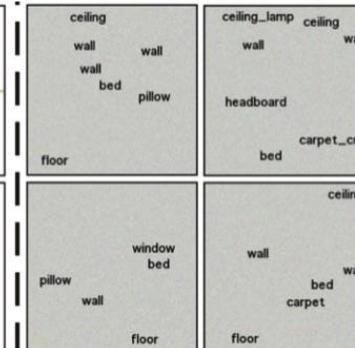
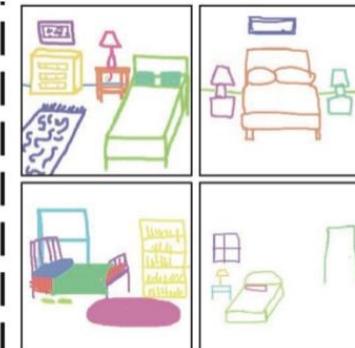
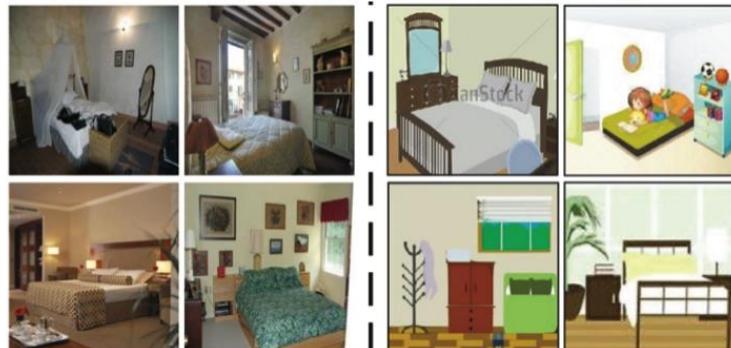
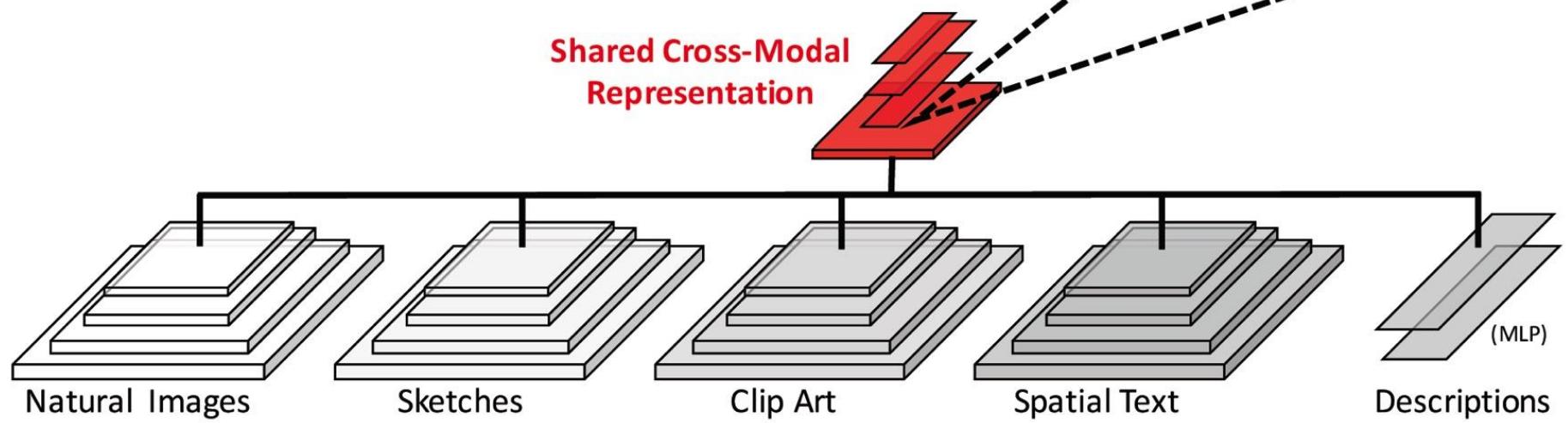
## Image Colorization





# Multi-branch input

Cross-modal Scene Classification



There is a bed with a st  
this is a nightstand with  
tall dresser and a chair  
the dresser is a jewelry

I am inside a room surro  
things. This room is fille  
comfortable bed. There  
everywhere. I have pos  
jewelry box is on the dr



# Multi-branch input

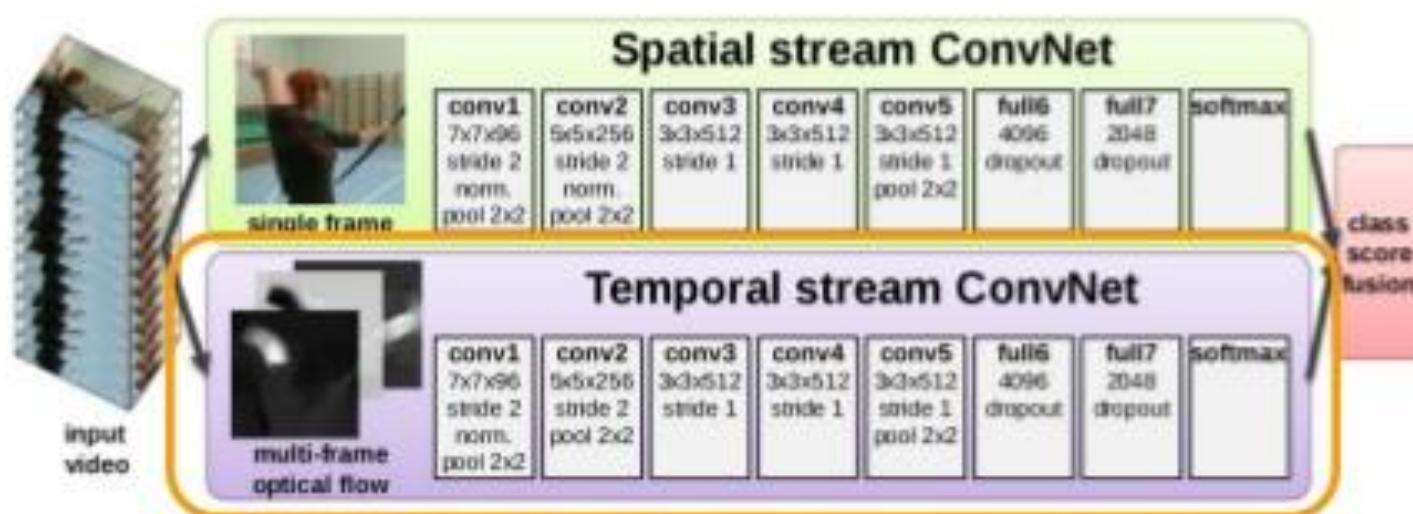
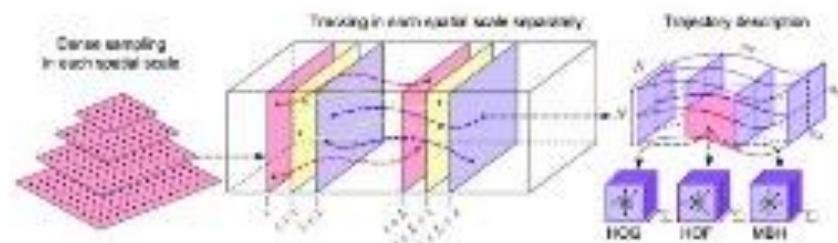
Activity Recognition

## Recognition: Two stream

Two CNNs in parallel:

- One for RGB images
- One for Optical flow (hand-crafted features)

Fusion after the softmax layer

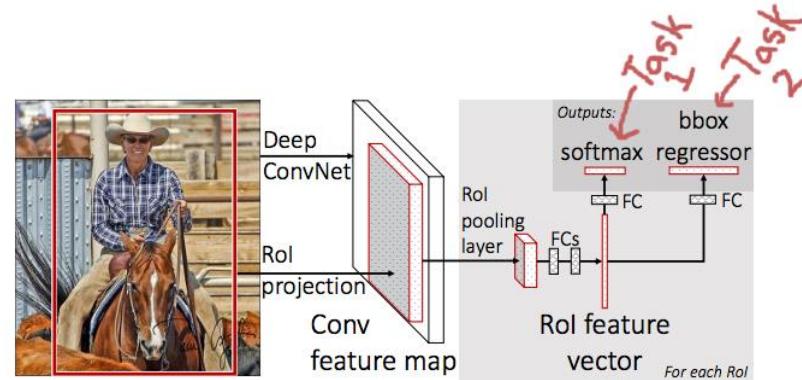


Simonyan, Karen, and Andrew Zisserman. ["Two-stream convolutional networks for action recognition in videos."](#) NIPS 2014.

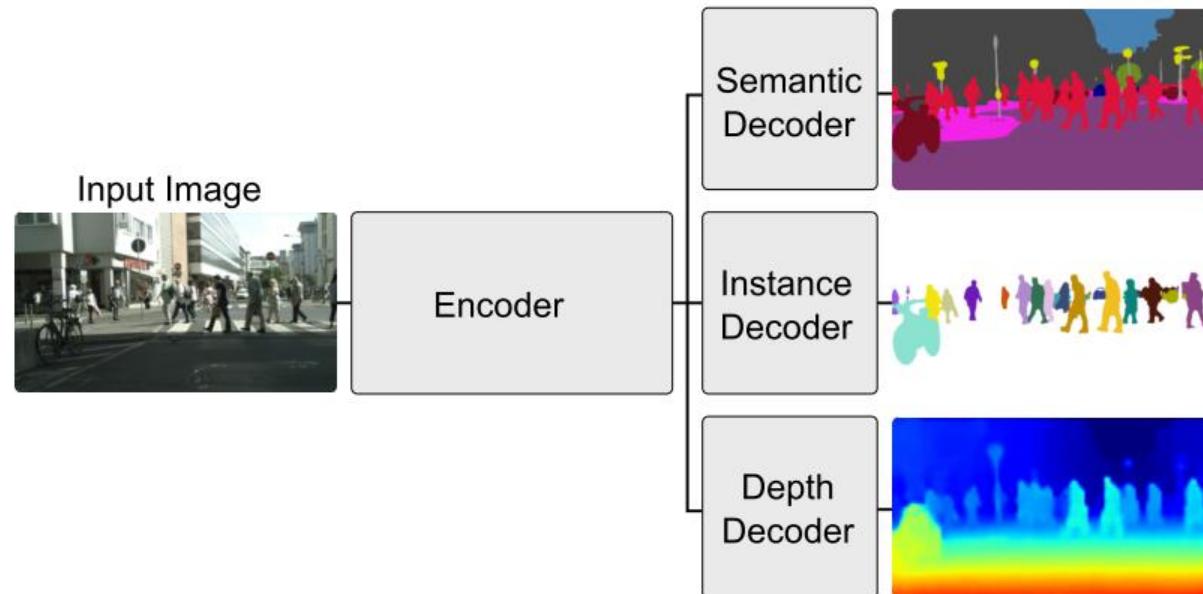


# Multi-branch output

Object Detection, Classification



Scene Parsing





# Image CNNs for non-image data

## Audio Beat Detection

