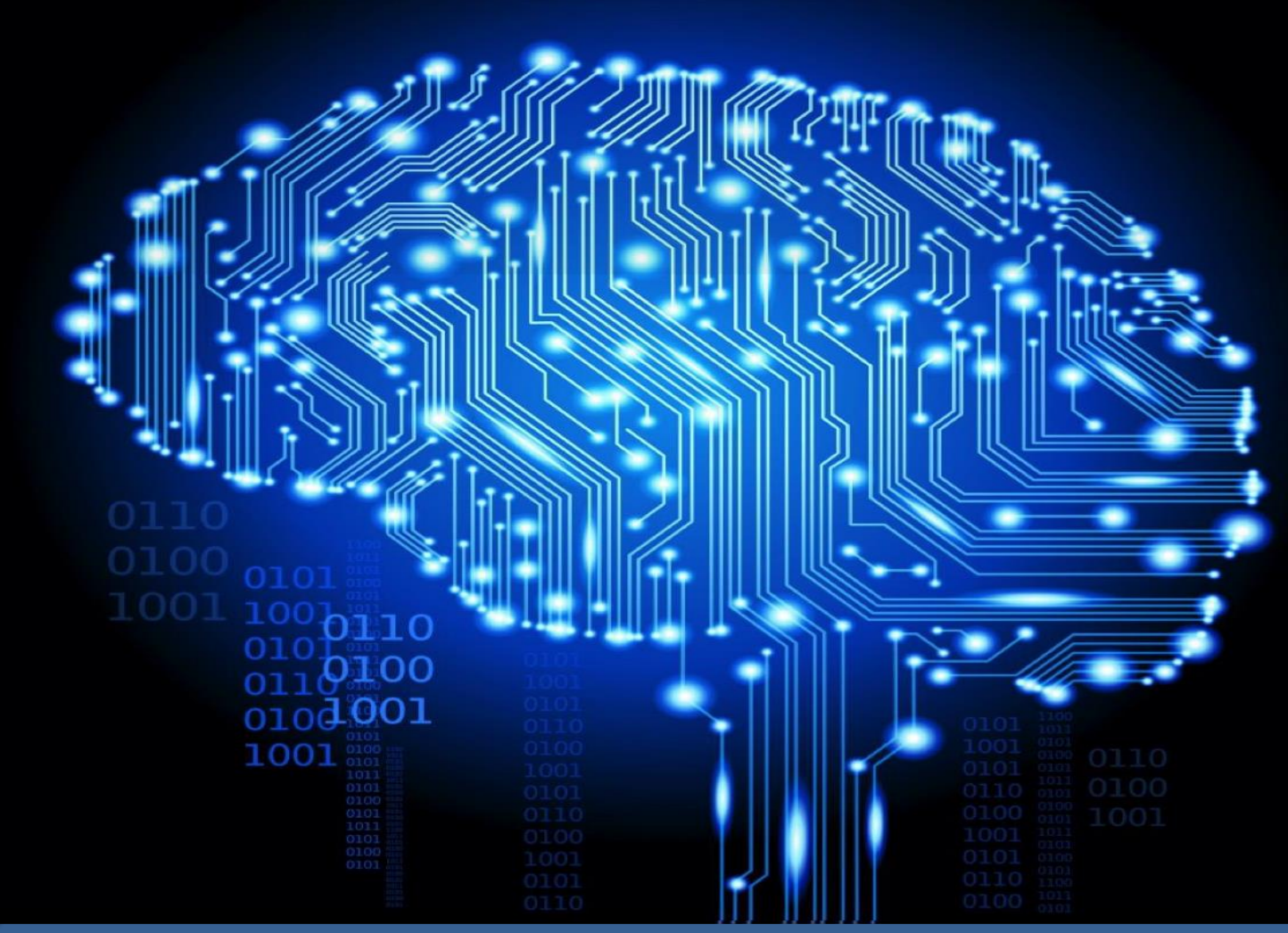


Prof. Jaewook Lee

Statistical Learning and Computational Finance Lab.
Department of Industrial Engineering
jaewook@snu.ac.kr
http://slcf.snu.ac.kr

Convolutional Neural Networks for Computer Vision

Reference

- [Goodfellow et al. (2016)] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning, The MIT Press, 2016.
- [Hinton. (2015)] G. Hinton, Online course on Neural Networks for Machine Learning.

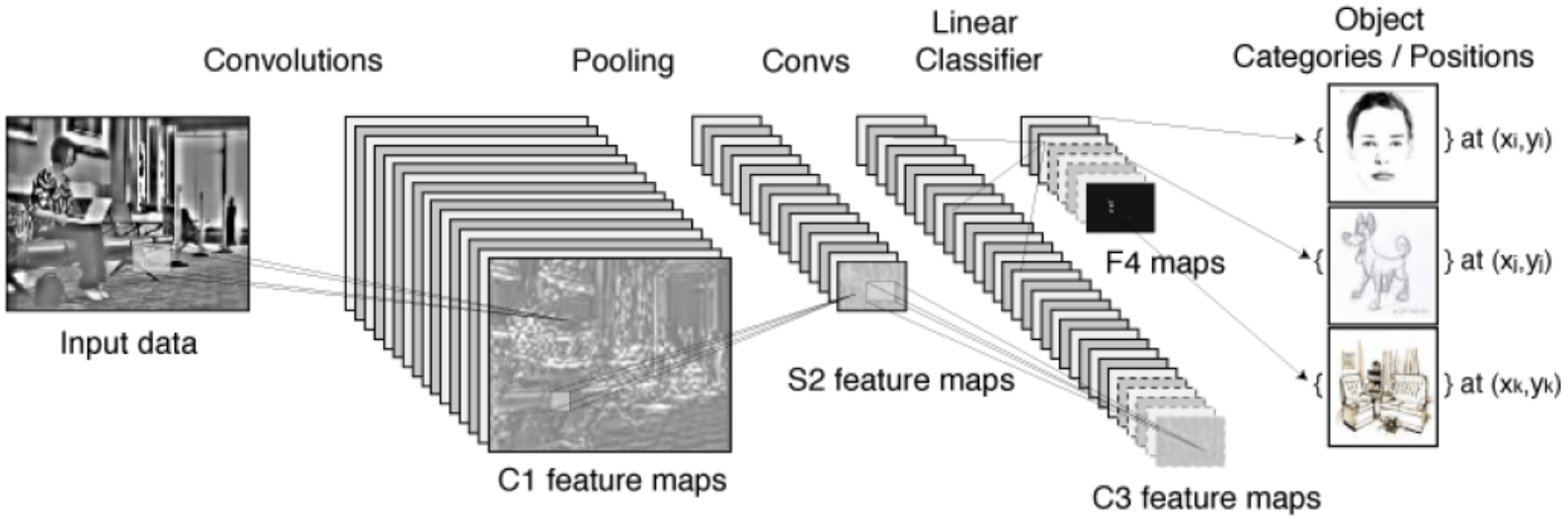
  https://www.coursera.org/learn/neural-networks
- [Larochelle. (2014)] H. Larochelle, Online course on Neural Networks.

  http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
- [Gavves. (2017)] E. Gavves, UvA Deep Learning Course.

  http://uvadlc.github.io/

- Borrows slides (some modified) from Larochelle & Hinton & Gavves

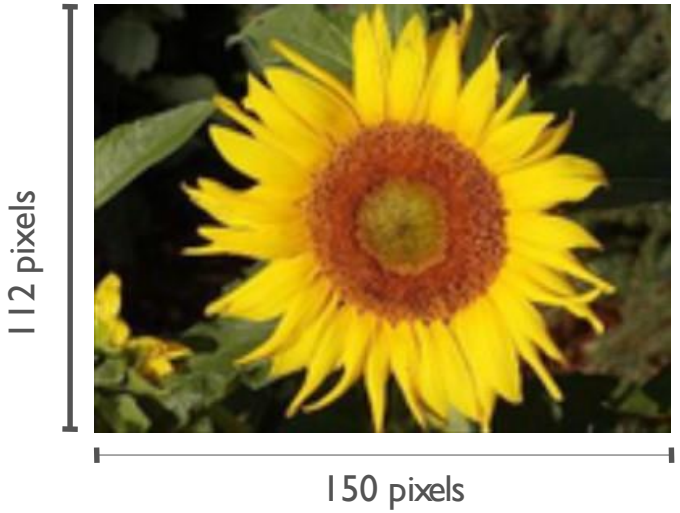
Convolutional Neural Networks



Source: http://uvadlc.github.io/

COMPUTER VISION

- Computer vision, object recognition
  - Computer vision is the design of computers that can process visual data and accomplish some given task
    - we will focus on object recognition: given some input image, identify which object it contains

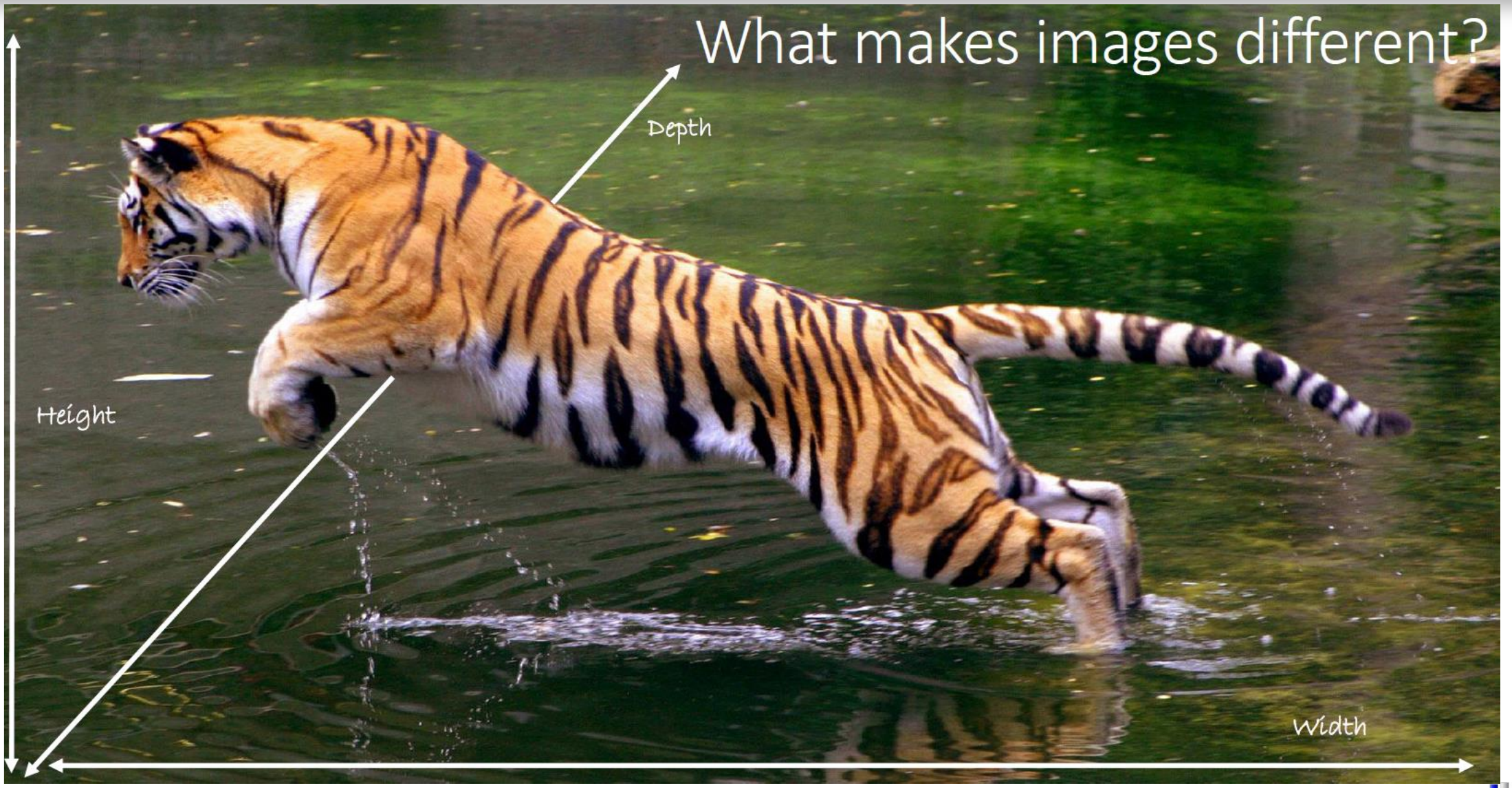


Caltech 101 dataset

"sun flower"

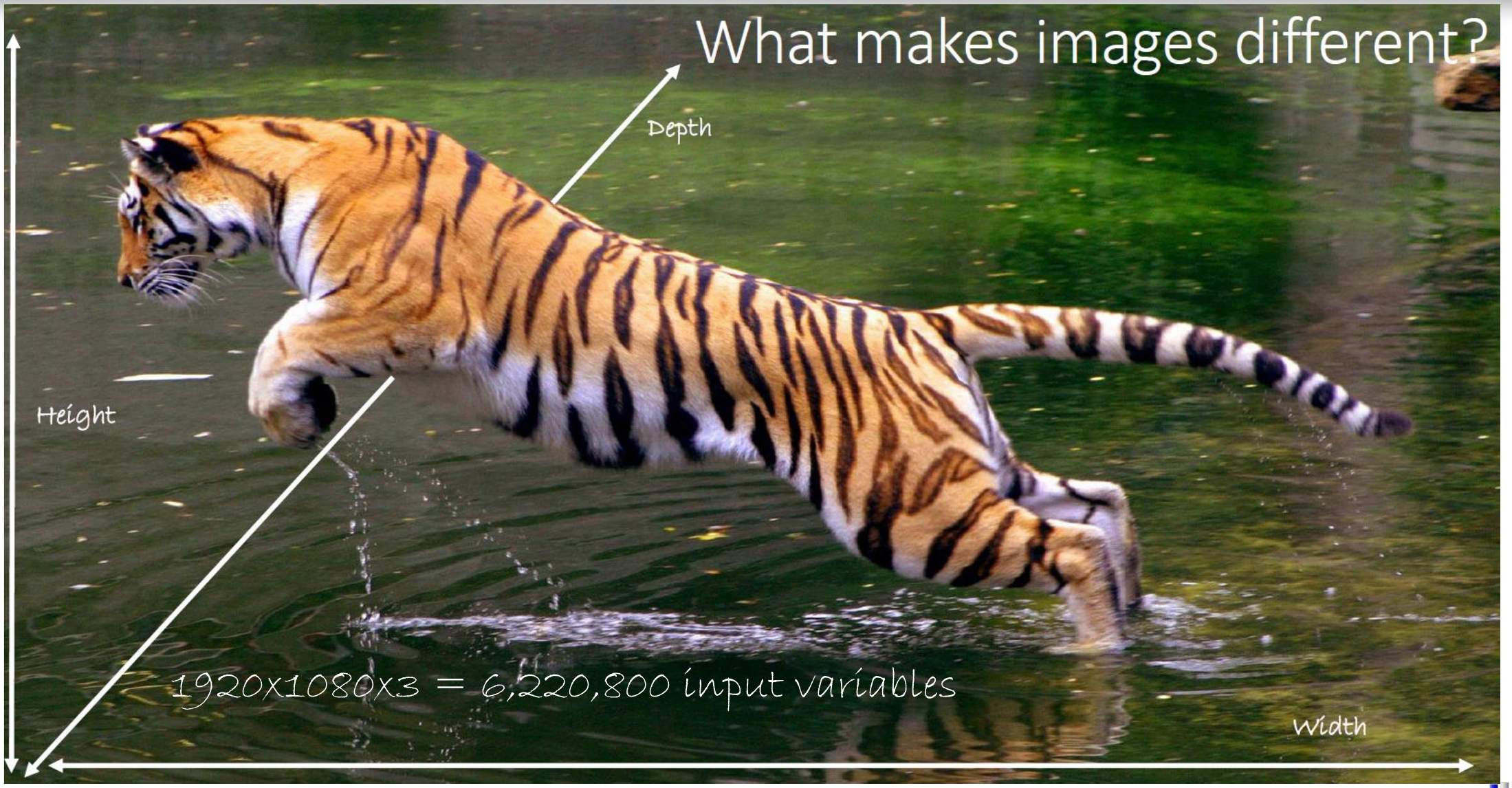
Source: http://uvadlc.github.io/

What makes images different?



Source: http://uvadlc.github.io/

What makes images different?



What makes images different?

What makes images different?

What makes images different

- An image has spatial structure
- Huge dimensionality
  - A 256x256 RGB images amounts to ~200K input variables
  - 1-layered NN with 1,000 neurons → 200 million parameters
- Images are stationary signal → they share features
  - After variances images are still meaningful
  - Small visual changes (often invisible to naked eye) → big changes to input vector
  - Still, semantics remain
  - Basic natural image statistics are the same

Input dimensions are correlated

Traditional task: Predict my salary!

Shift 1 dimension

Level of education	Age	Years of experience	Previous job	Nationality
"Higher"	28	6	Researcher	Spain

Level of education	Age	Years of experience	Previous job	Nationality
Spain	"Higher"	28	6	Researcher

Vision task: Predict the picture!

First 5x5 values

```
array([[51, 49, 51, 56, 55],
       [53, 53, 57, 61, 62],
       [67, 68, 71, 74, 75],
       [76, 77, 79, 82, 80],
       [71, 73, 76, 75, 75]], dtype=uint8)
```

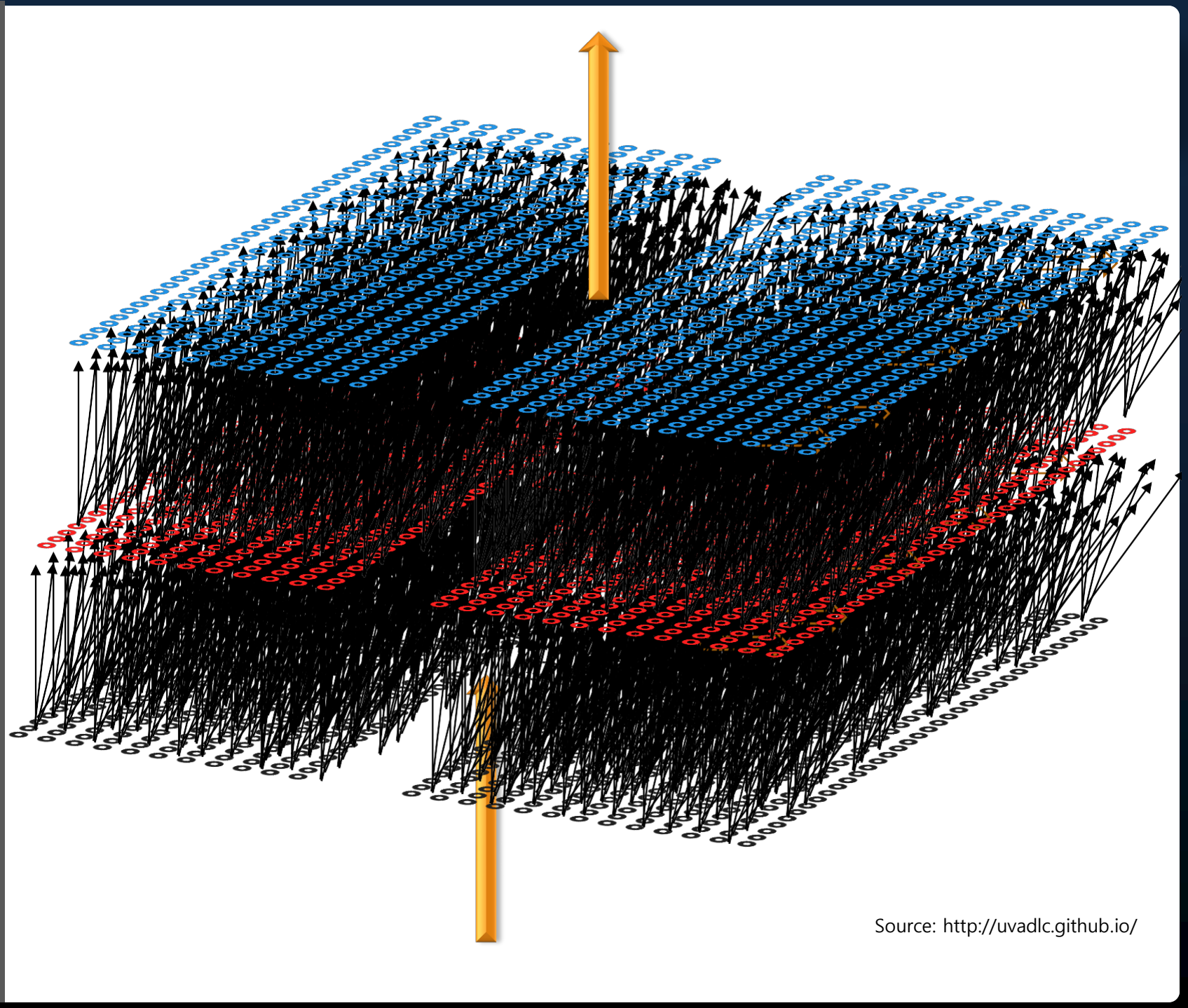
First 5x5 values

```
array([[58, 57, 57, 59, 59],
       [58, 57, 57, 58, 59],
       [59, 58, 58, 58, 58],
       [61, 61, 60, 60, 59],
       [64, 63, 62, 61, 60]], dtype=uint8)
```

Convolutional Neural Networks

- Convolutional networks
  - simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- Preserve spatial structure by convolutional filters
- Tackle huge input dimensionalities by local connectivity and parameter sharing
- Robust to local variances by spatial pooling

Filters *vs* Convolutional k-d filters

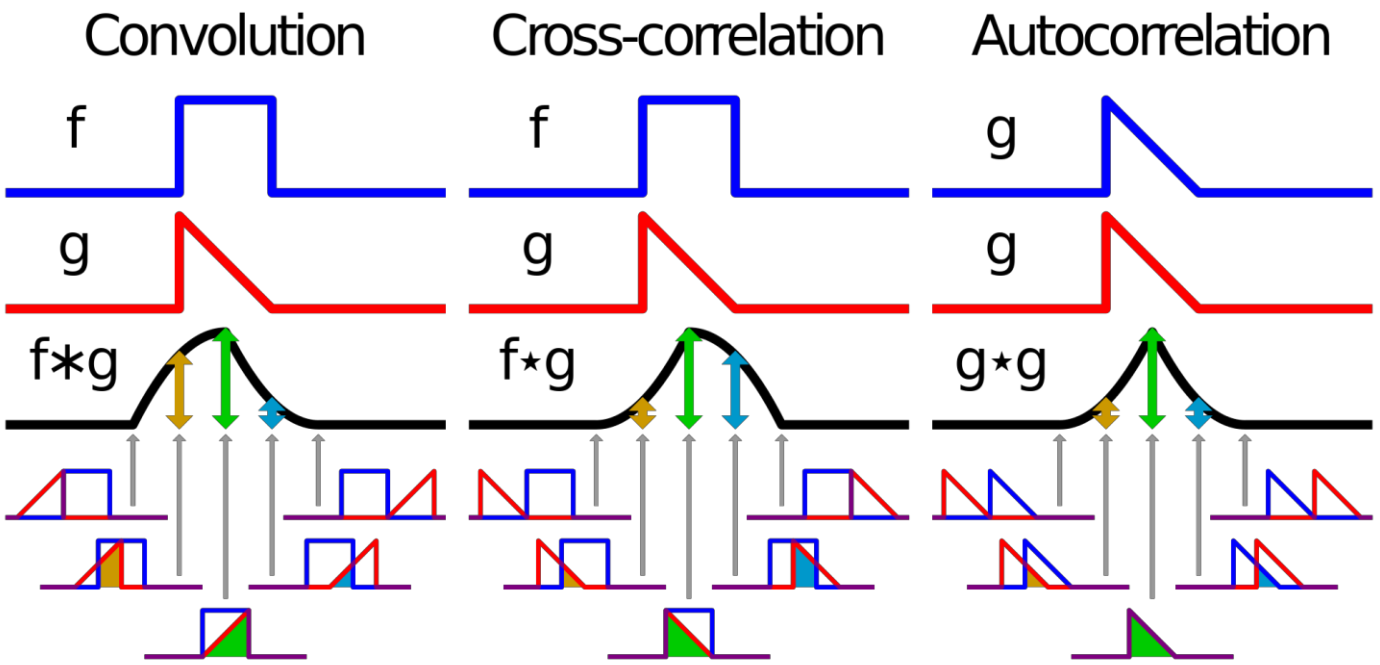


Source: http://uvadlc.github.io/

Convolution

- Definition: The convolution of two functions f and g is denoted by * as the integral of the product of the two functions after one is reversed and shifted

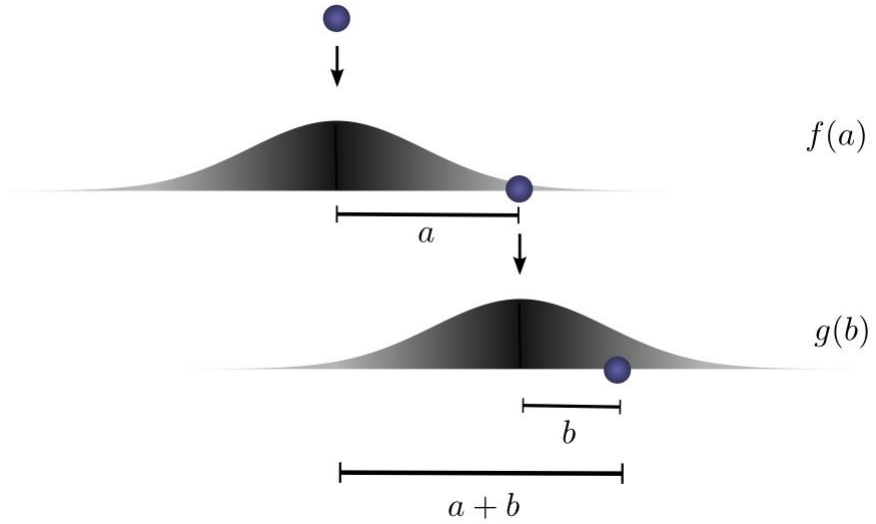
$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau$$



Source: http://uvadlc.github.io/

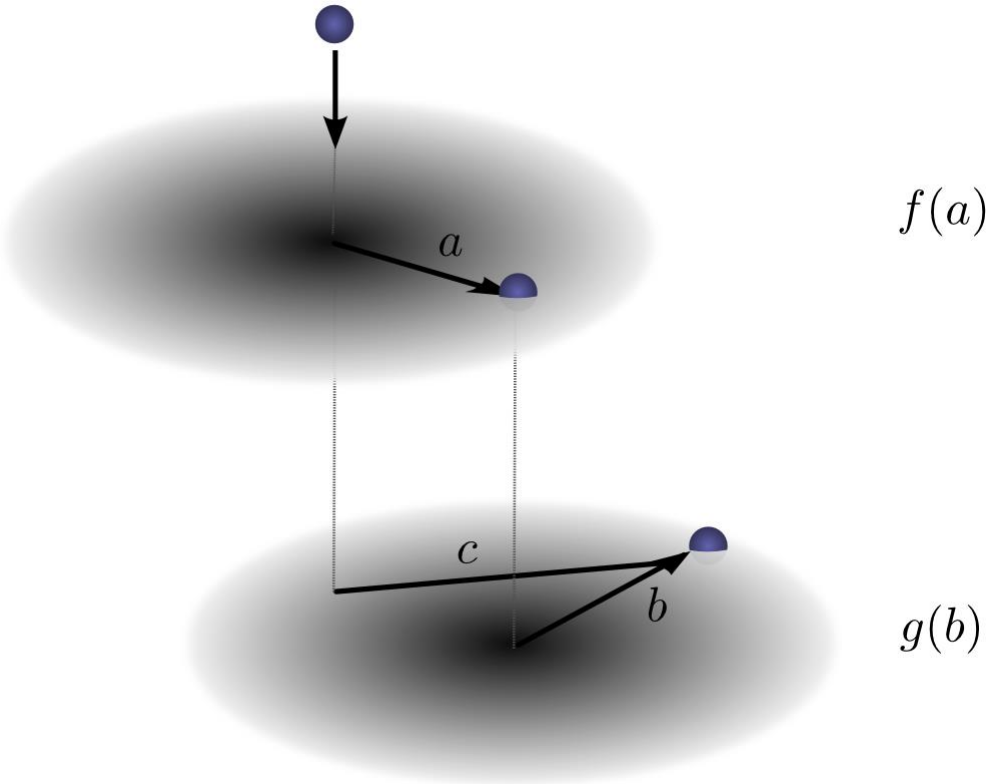
Understanding Convolution

$$(f * g)(c) = \sum_{a+b=c} f(a)g(b)$$



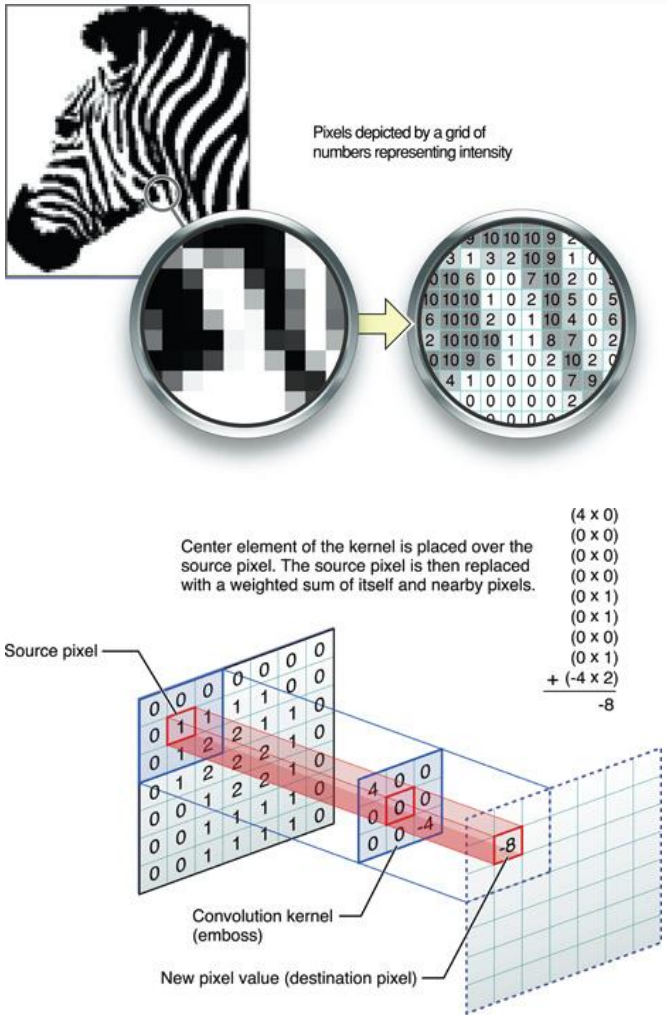
Total likelihood of the ball reaching a total distance of c =consider all the possible ways of partitioning c into two drops a and b and sum over the probability of each way

$$(f * g)(c_1, c_2) = \sum_{\substack{a_1+b_1=c_1 \\ a_2+b_2=c_2}} f(a_1, a_2)g(b_1, b_2)$$



Source; http://colah.github.io/posts/2014-07-Understanding-Convolutions/

Convolution: detector of a feature



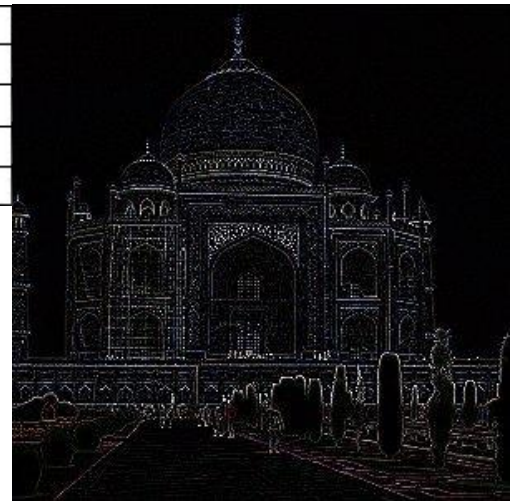
https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Blur

	0	1	0	
	1	-4	1	
	0	1	0	

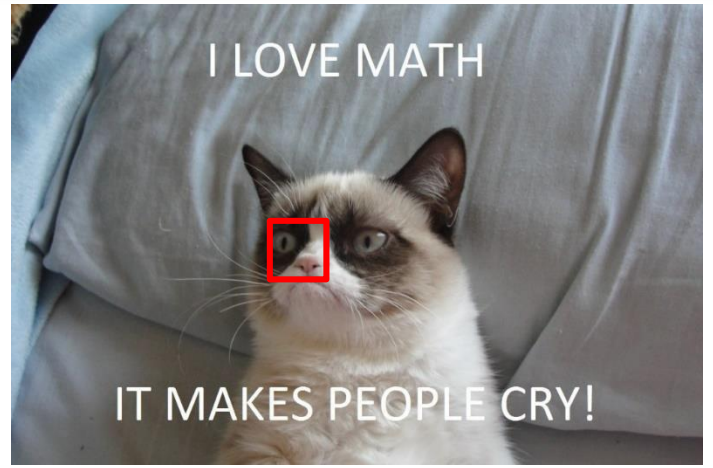
Edge detect



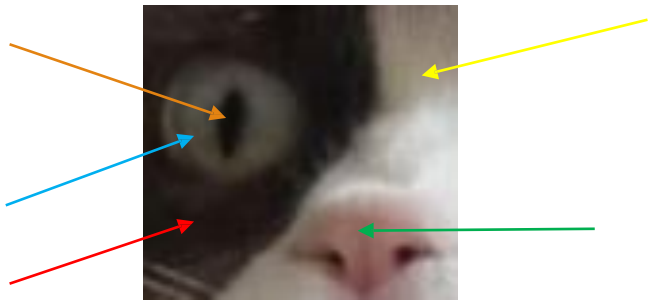
Source; http://colah.github.io/posts/2014-07-Understanding-Convolutions/

Why spatial?

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.

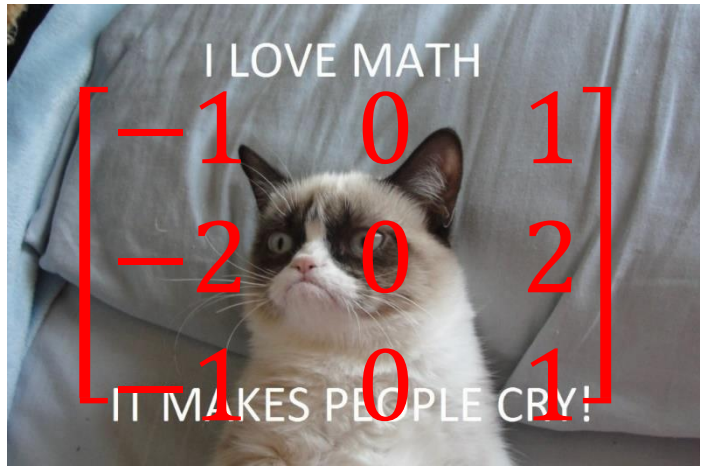


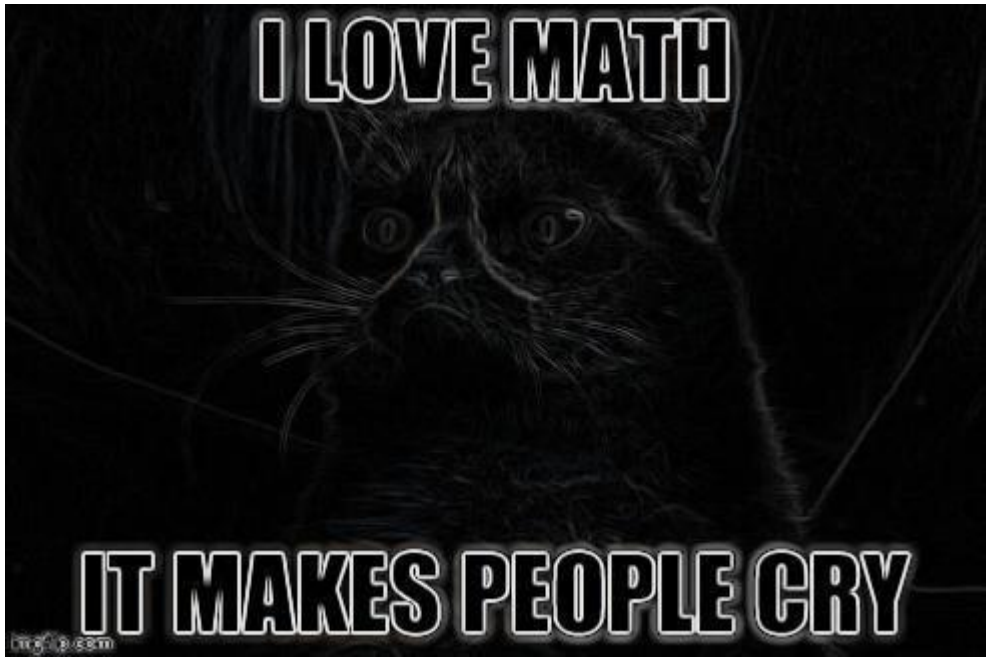
- What does a 2-D input really mean?
  - Neighboring variables are locally correlated



Example filter when k=1

e.g. Sobel 2-D filter





Sobel 2-D filter

- An edge is shown by the "jump" in intensity

Source: http://uvadlc.github.io/

Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel → compute the gradient with smoothing

Horizontal changes

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * ([-1 \quad 0 \quad 1] * I)$$

Vertical changes

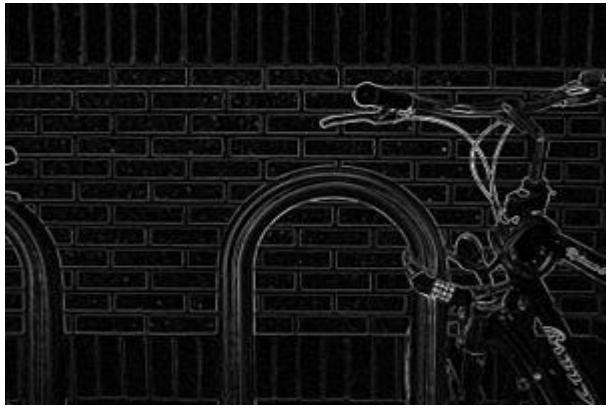
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I = [-1 \quad 0 \quad 1] * (\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * I)$$

Approximation of the gradient at each point of the image

$$G = \sqrt{G_x^2 + G_y^2} \cong |G_x| + |G_y|$$

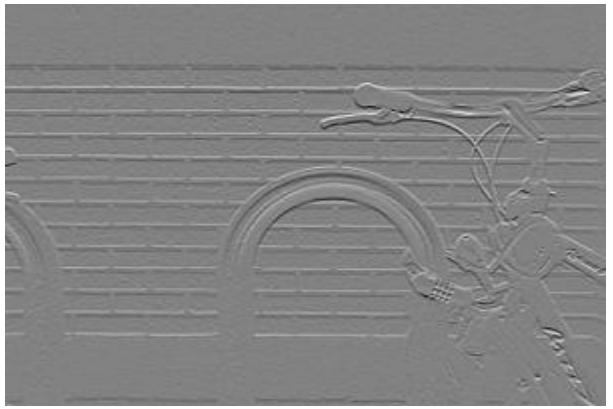
Source: https://en.wikipedia.org/wiki/Sobel_operator

Grayscale test image of brick wall and bike rack



Normalized gradient magnitude from Sobel operator

Normalized *x*-gradient from Sobel operator



Normalized *y*-gradient from Sobel operator

Learnable filters

- Several, handcrafted filters in computer vision
  - Canny, Sobel, Gaussian blur, smoothing, low-level segmentation, morphological filters, Gabor filters
- Are they optimal for recognition?
- Can we learn them from our data?
- Are they going resemble the handcrafted filters?

$$\begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ \theta_7 & \theta_8 & \theta_9 \end{bmatrix}$$

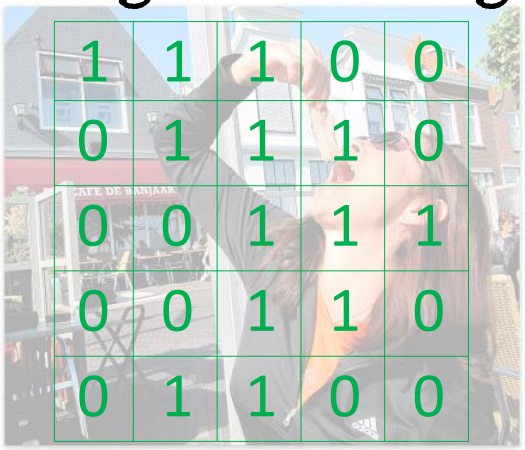
Source: http://uvadlc.github.io/

Shared 2-D filters → Convolutions

Original image

Shared 2-D filters → Convolutions

Original image



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

1	0	1
0	1	0
1	0	1

$k = \widetilde{W} = W$

k_{11}	k_{12}	k_{13}
k_{21}	k_{22}	k_{23}
k_{31}	k_{32}	k_{33}

$k = \widetilde{W}$

⇨

k_{33}	k_{32}	k_{31}
k_{23}	k_{22}	k_{21}
k_{13}	k_{12}	k_{11}

W

$k_{ij} = \widetilde{W_{ij}}$ from W_{ij} with its rows and columns flipped

$$I(x,y) * \widetilde{W} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x+i, y+j) \cdot W(i,j)$$

⇧

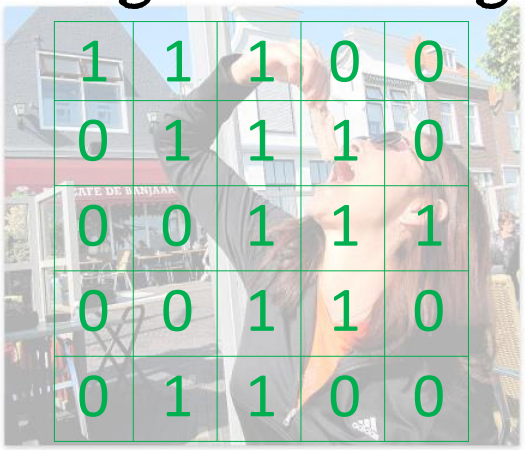
Inner product

$$I(x,y) * k = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot k(i,j)$$

Source: http://uvadlc.github.io/

Shared 2-D filters → Convolutions

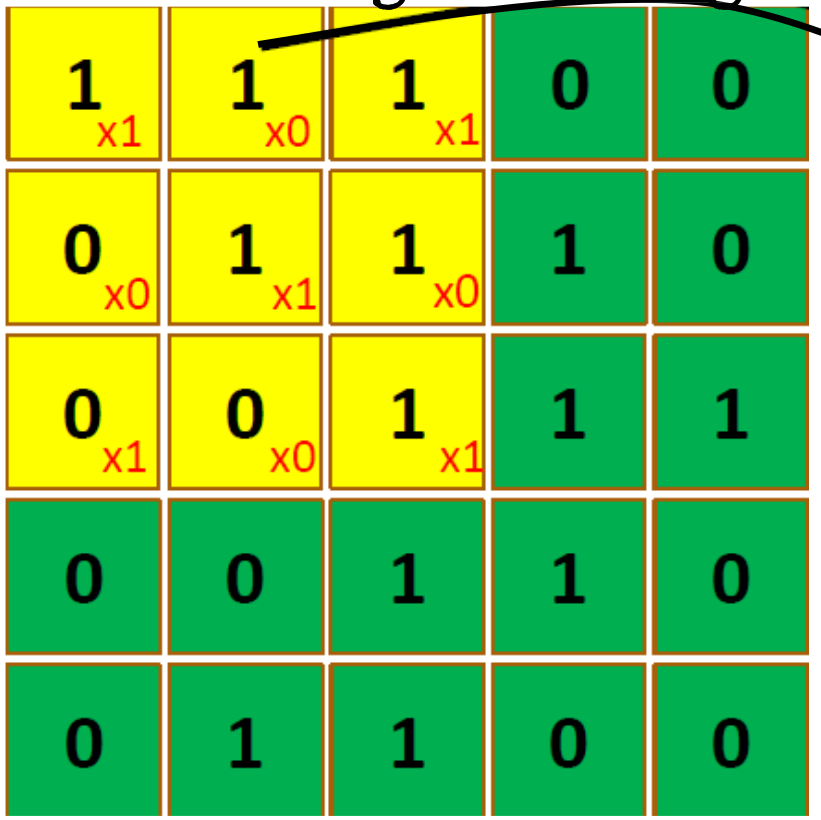
Original image



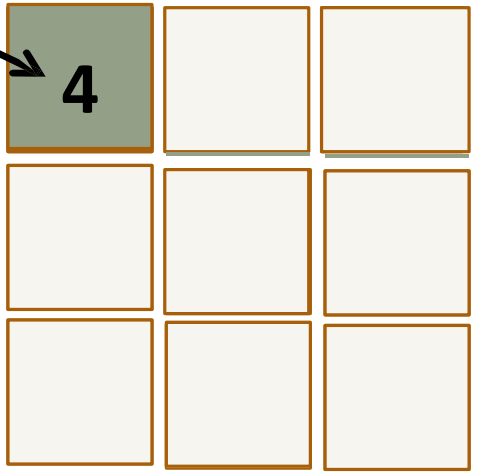
Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image



Result

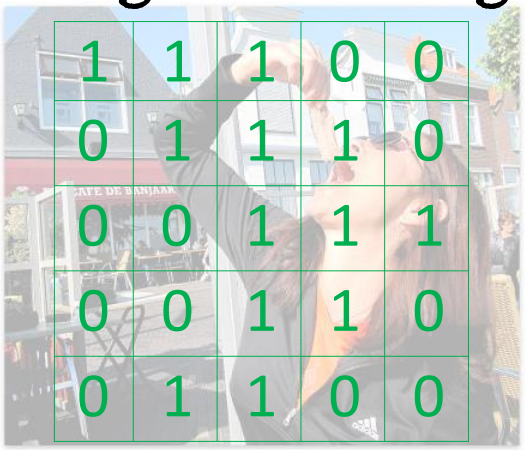


Inner product

$$I(x,y) * k = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot k(i,j)$$

Shared 2-D filters → Convolutions

Original image



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image

1	1 x1	1 x0	0 x1	0
0	1 x0	1 x1	1 x0	0
0	0 x1	1 x0	1 x1	1
0	0	1	1	0
0	1	1	0	0

Result

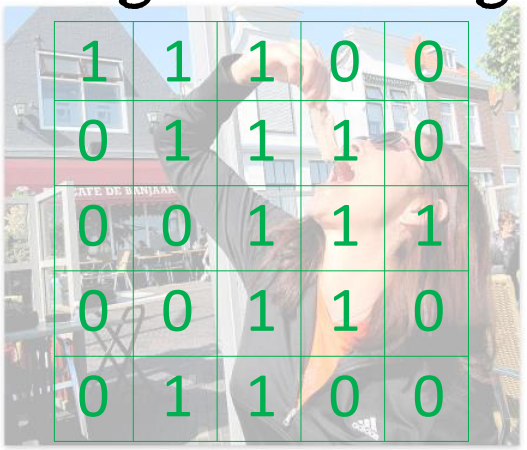
4	3	

Inner product

$$I(x,y) * k = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot k(i,j)$$

Shared 2-D filters → Convolutions

Original image



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

1	0	1
0	1	0
1	0	1

Convolving the image

1	1	1	0	0
0	1	1	1	0
0	0	1_{x1}	1_{x0}	1_{x1}
0	0	1_{x0}	1_{x1}	0_{x0}
0	1	1_{x1}	0_{x0}	0_{x1}

Result

4	3	4
2	4	3
2	3	4

Inner product

$$I(x,y) * k = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot k(i,j)$$

Convolution activations are “images”

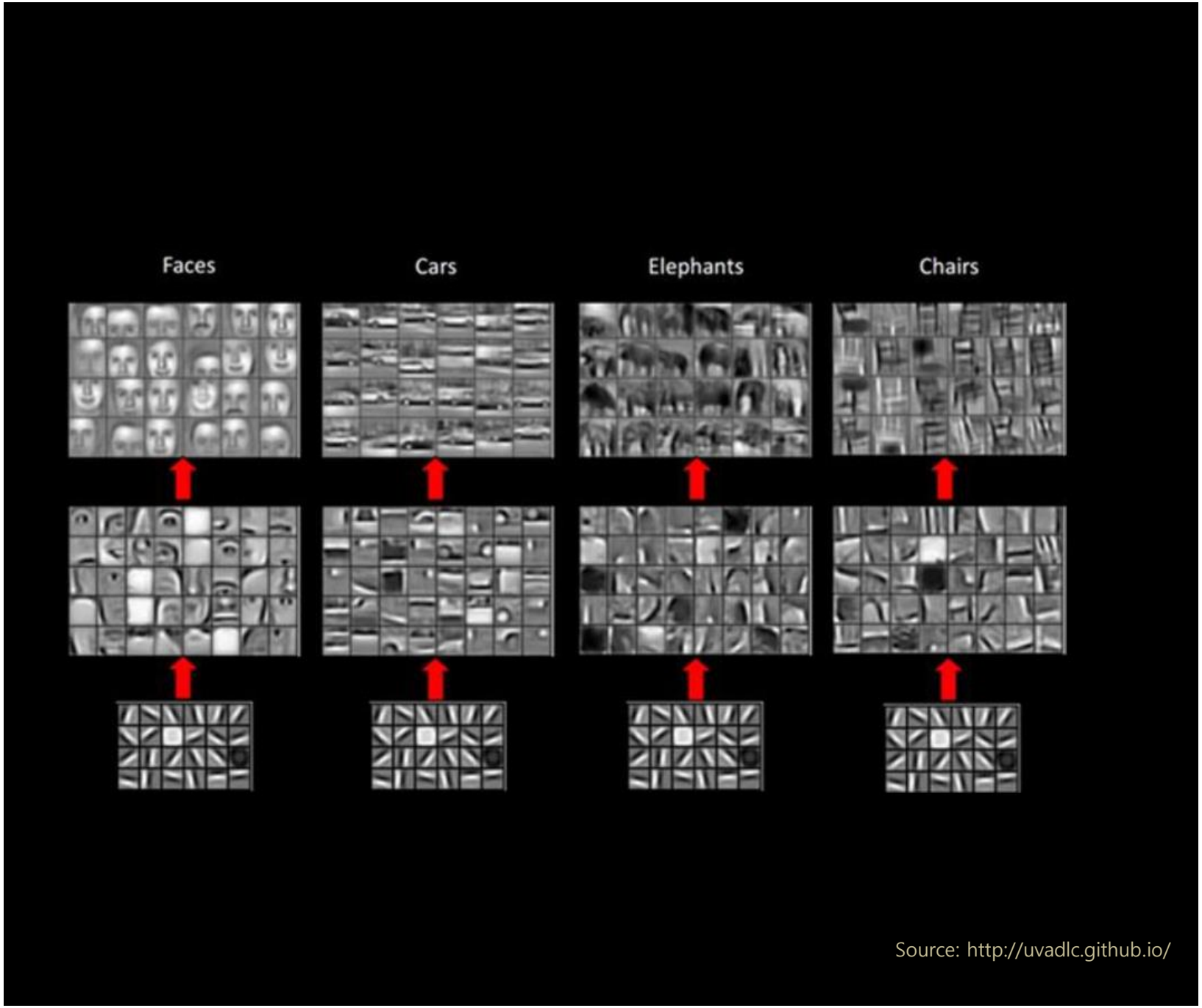
- $h_I(x,y) = I(x,y) * k = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x-i, y-j) \cdot k(i,j)$
- For every image pixel we compute a convolved image pixel
- After each convolution we end up with a new image

Convolution Operation

- Discrete convolution
  - Pre-activations from channel X_i into feature map Y_j can be computed by:
    - ij-th convolution kernel where $K_{ij} = \widetilde{W_{ij}}$ from the connection matrix W_{ij} with its rows and columns flipped
    - applying the convolution $X_i * K_{ij}$
    - This is equivalent to computing the cross correlation of x_i with W_{ij}
  - Many neural network libraries implement cross-correlation, which is the same as convolution without flipping the kernel, but call it convolution.
  - Activation function: $a_{rc} = (x * k)_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} k_{ij} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r+i,c+j} W_{ij}$
  - Essentially a dot product, similar to linear layer: $a_{rc} \sim x_{region}^{T} W$
  - Gradient wrt the parameters

$$\frac{\partial a_{rc}}{\partial W_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r+i,c+j}$$

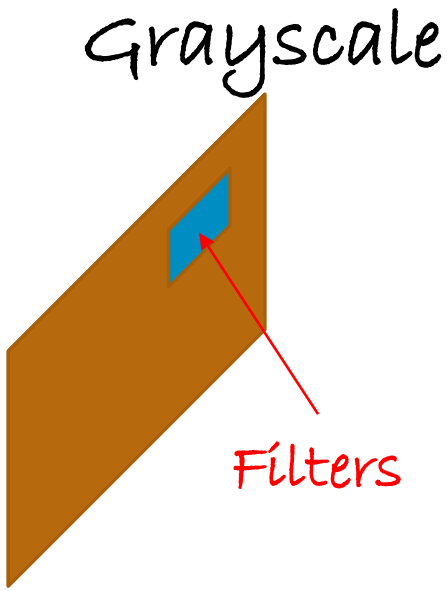
Preserving Spatial structure



Source: http://uvadlc.github.io/

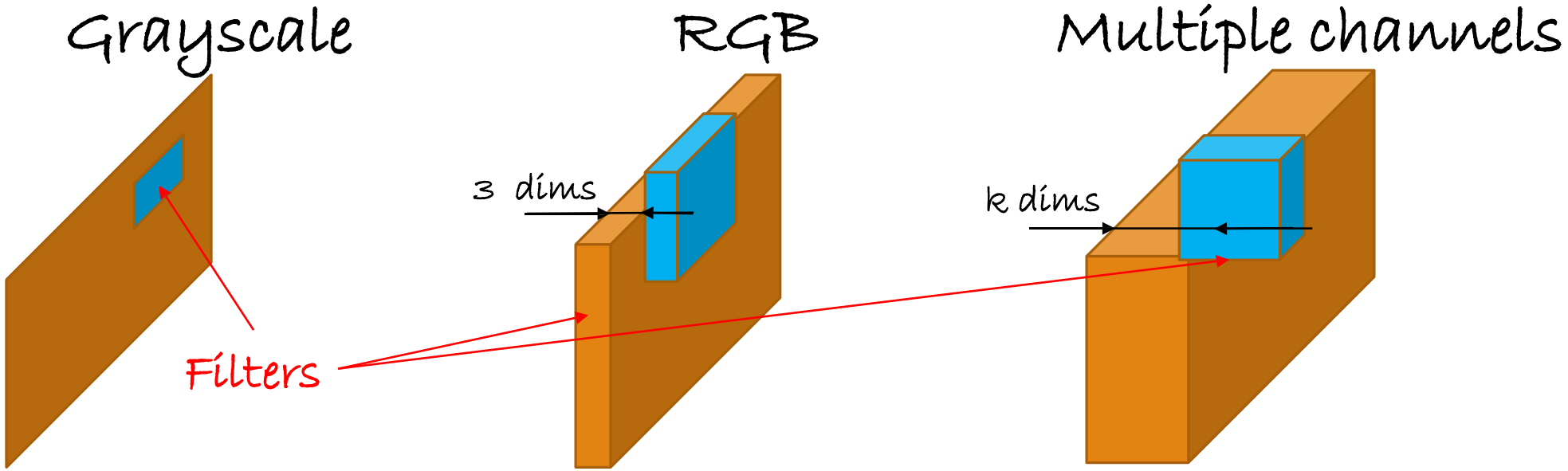
2-D filters (parameters)

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can "exploit" the spatial nature of images



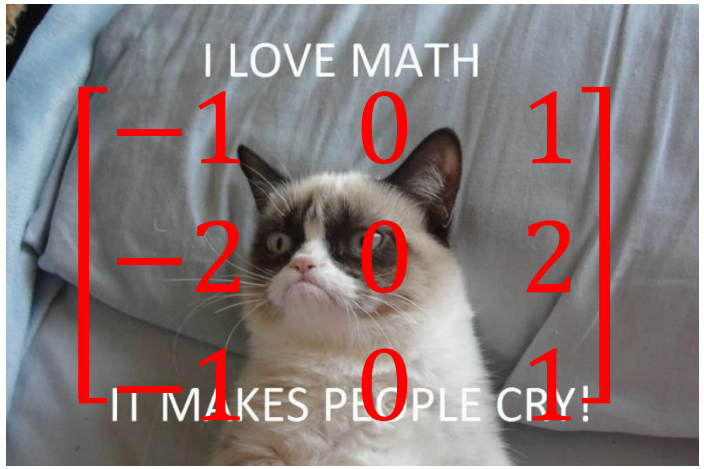
k-D filters (parameters)

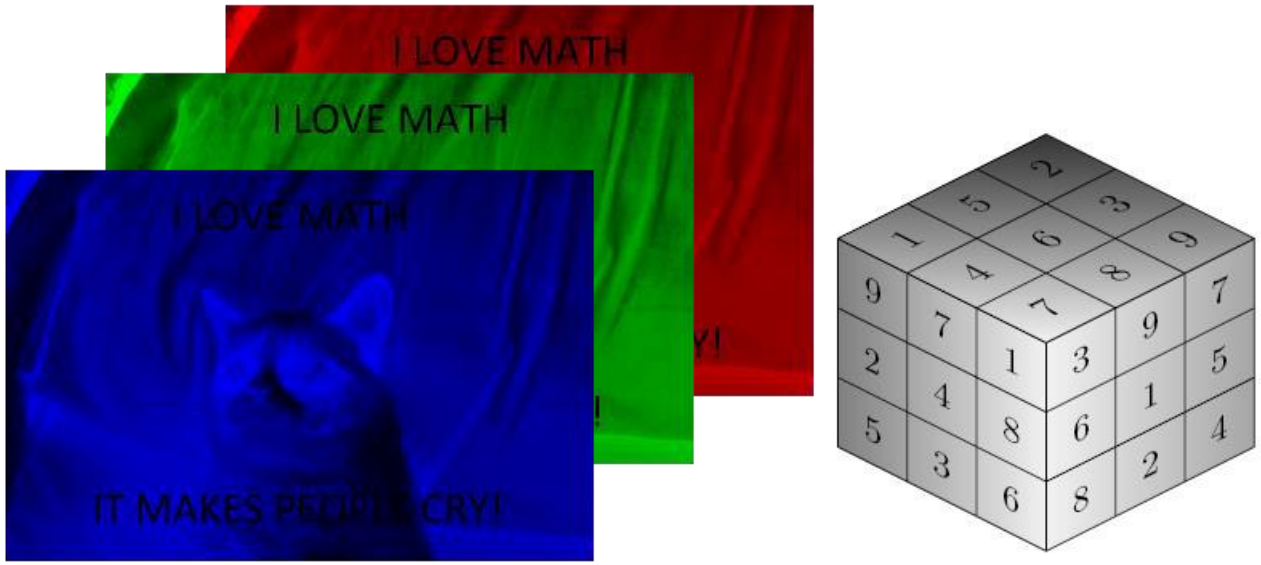
- Similarly, if images are k-D, parameters should also be k-D



What would a k-D filter look like?

e.g. Sobel 2-D filter

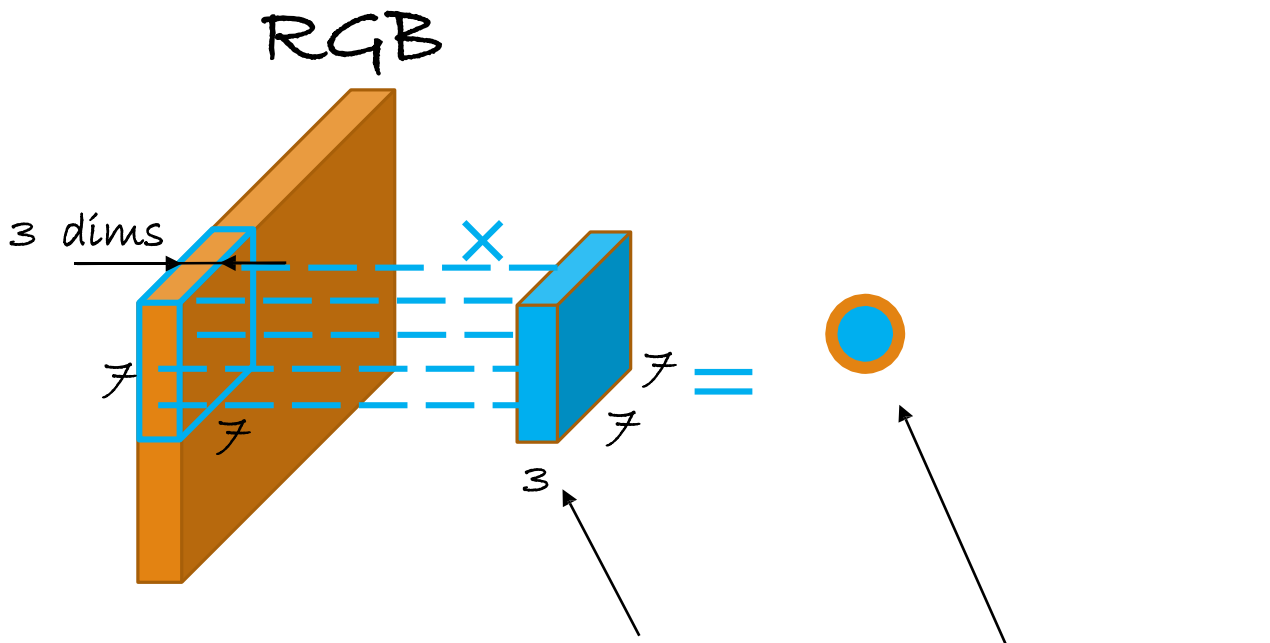




Source: http://uvadlc.github.io/

Think in space

filters (parameters)=weights

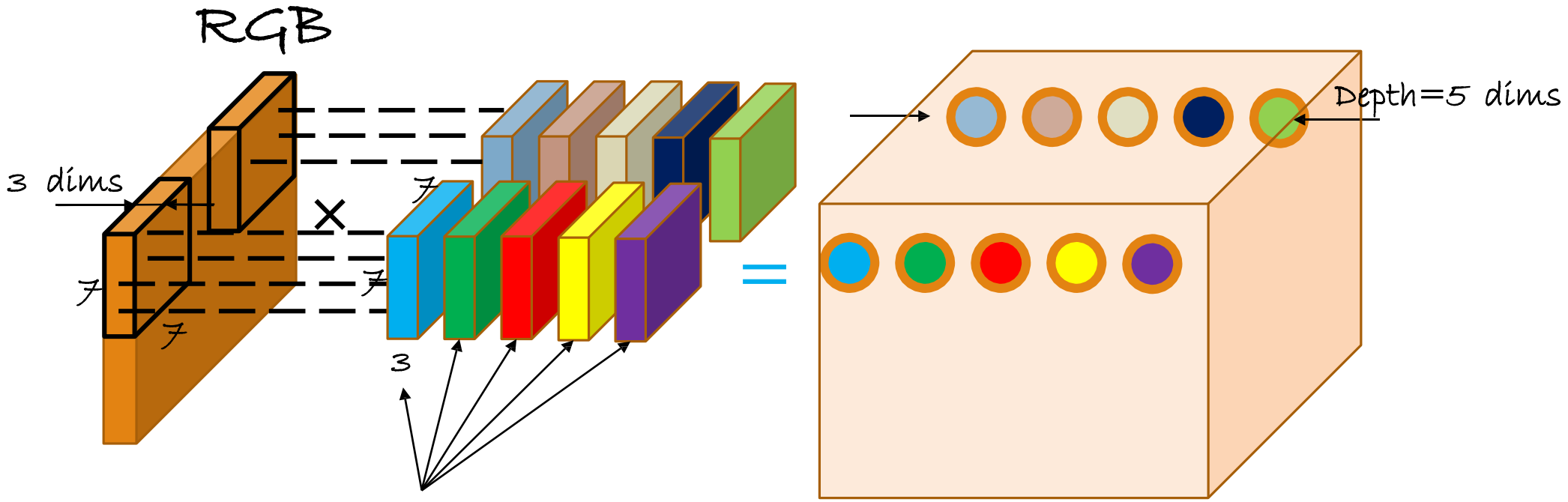


How many weights for this neuron?

$$7 \cdot 7 \cdot 3 = 147$$

Think in space

The activations of a hidden layer form a volume of neurons, not a 1-d "chain"
This volume has a depth 5, as we have 5 filters



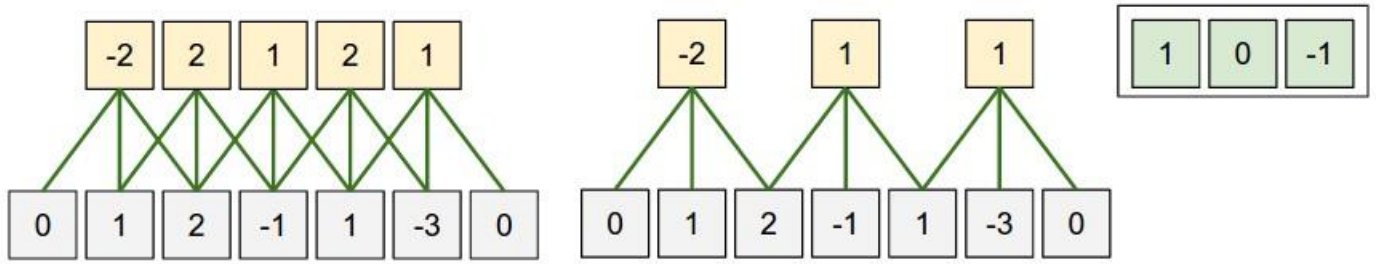
How many weights for these 5 neurons?

$a_{rc} \sim x_{region}^{T} W$

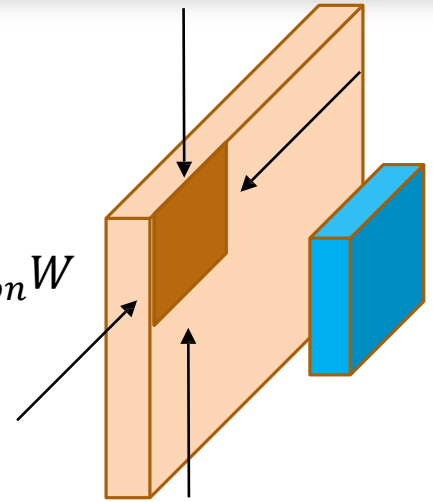
$5 \cdot 7 \cdot 7 \cdot 3 = 735$

Local connectivity

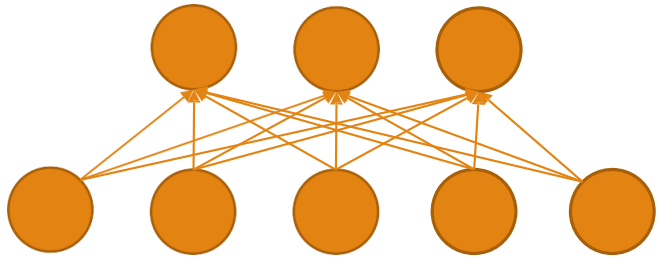
- The weight connections are surface-wise local!
  - Local connectivity

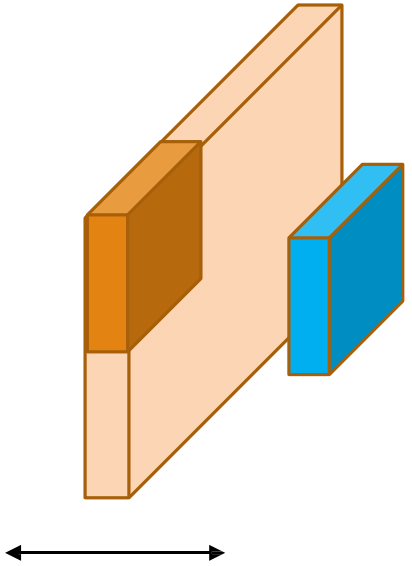


$$a_{rc} \sim x_{region}^{T} W$$



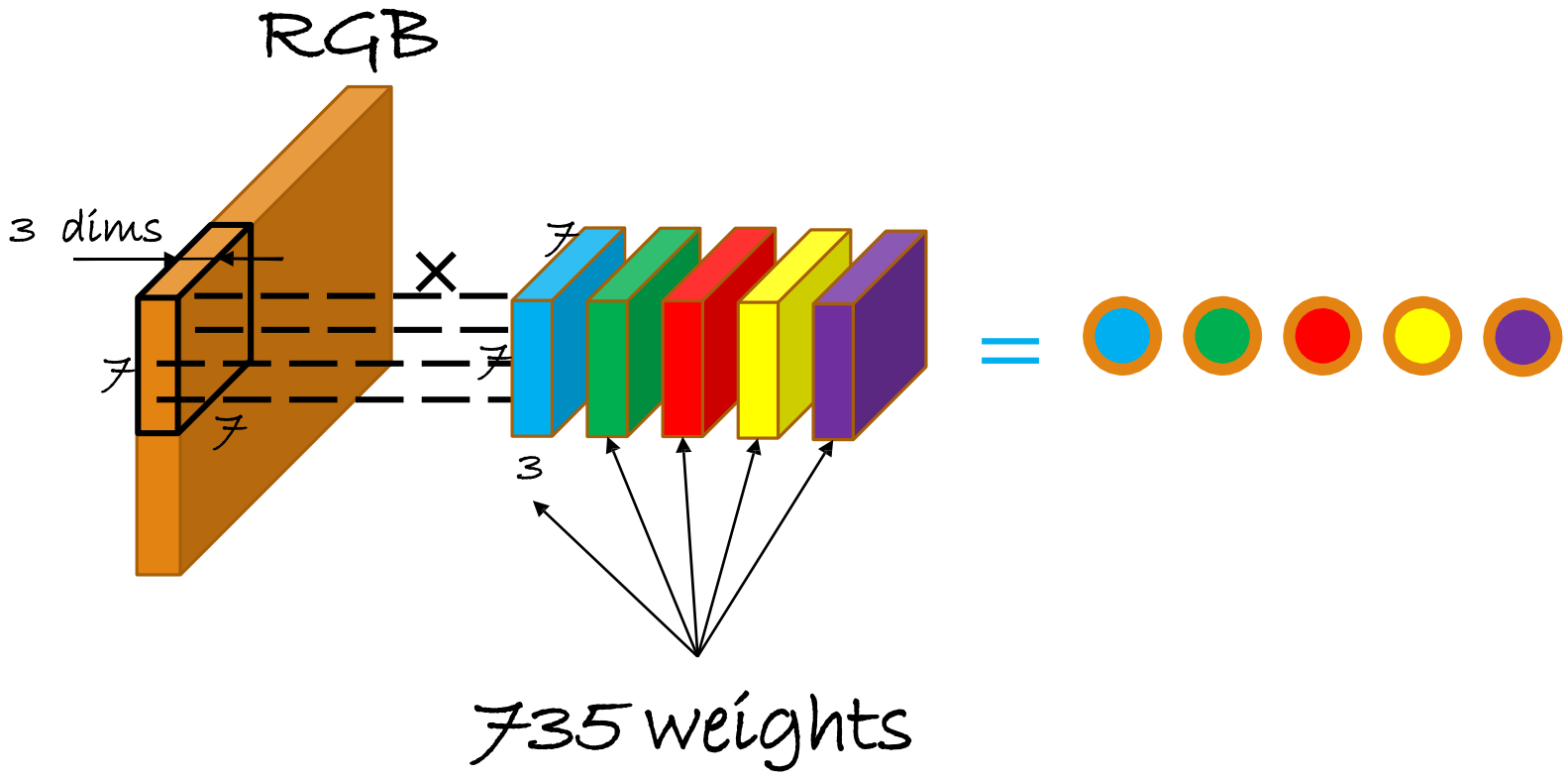
- The weights connections are depth-wise global
- For standard neurons no local connectivity
  - Everything is connected to everything





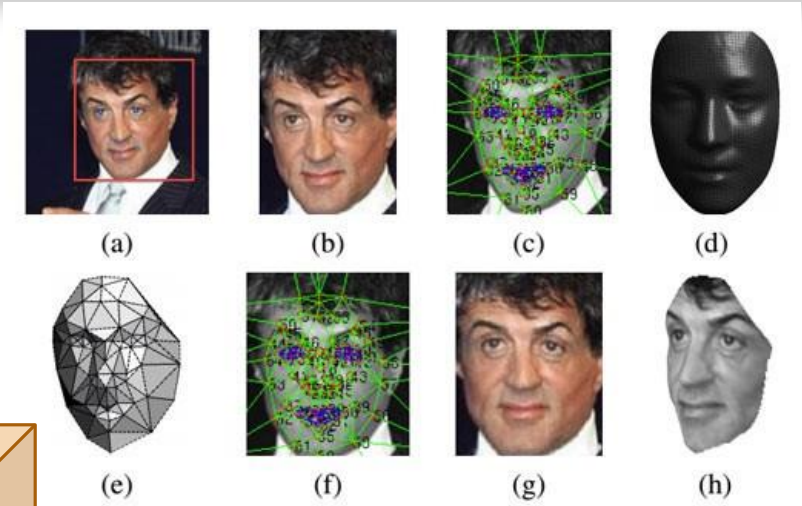
Source: http://uvadlc.github.io/

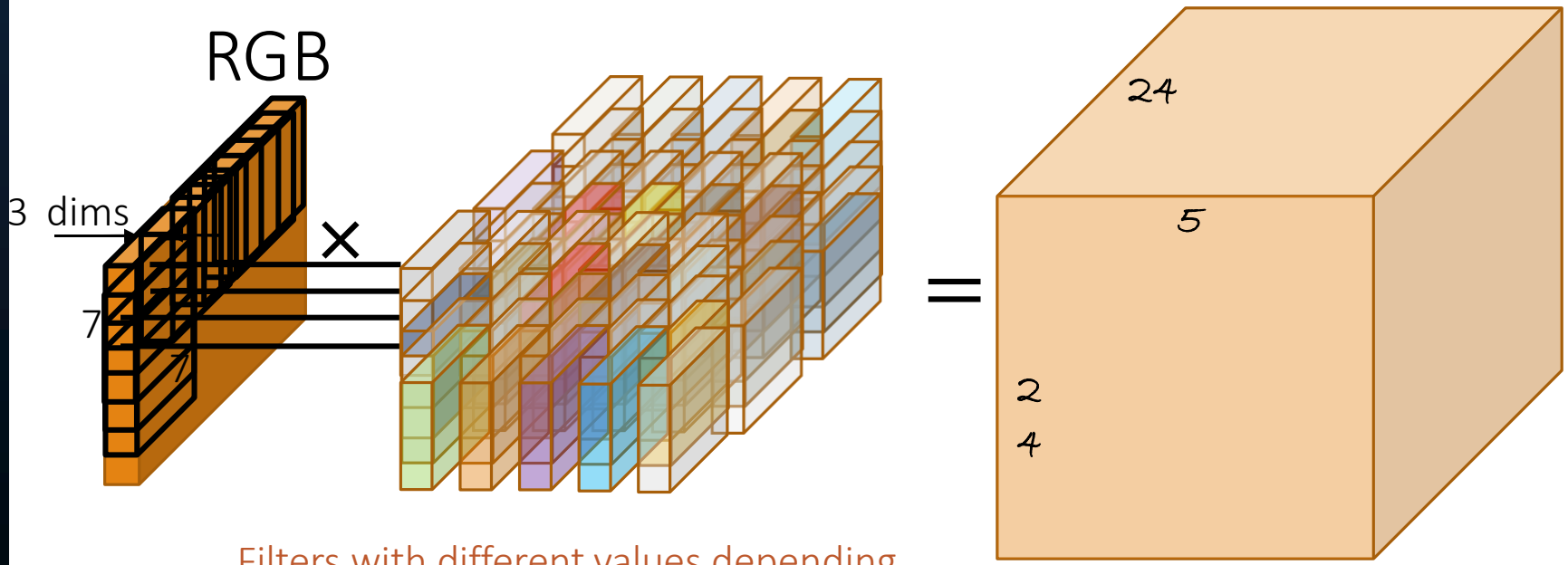
Again, think in space



Local connectivity ≠ Convolutional filters

- Local but non-shareable filters are possible
- Still useful for some applications





Filters with different values depending on image location and channel ID

Assume the image is 30x30x3.
1 filter every pixel (stride =1) H
ow many parameters in total?

24 filters along the x axis
24 filters along the y axis
Depth of 5
× $7 * 7 * 3$ parameters per filter

$423K$ parameters in total

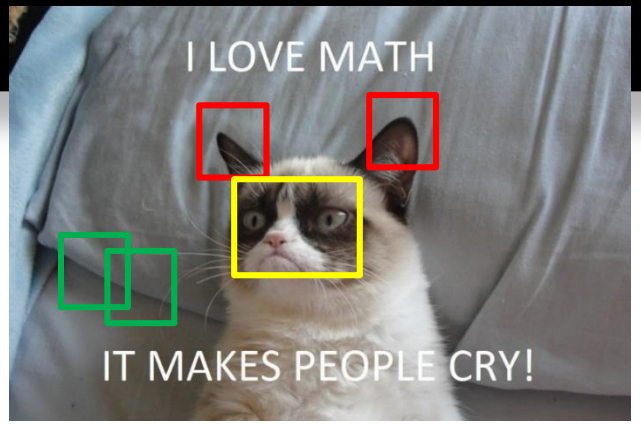
Source: http://uvadlc.github.io/

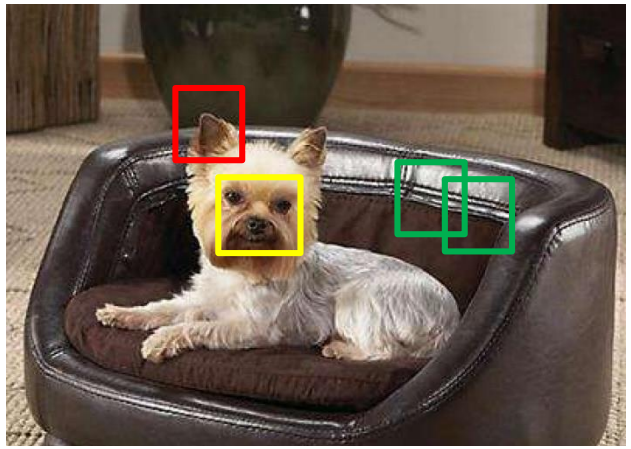
Problem!

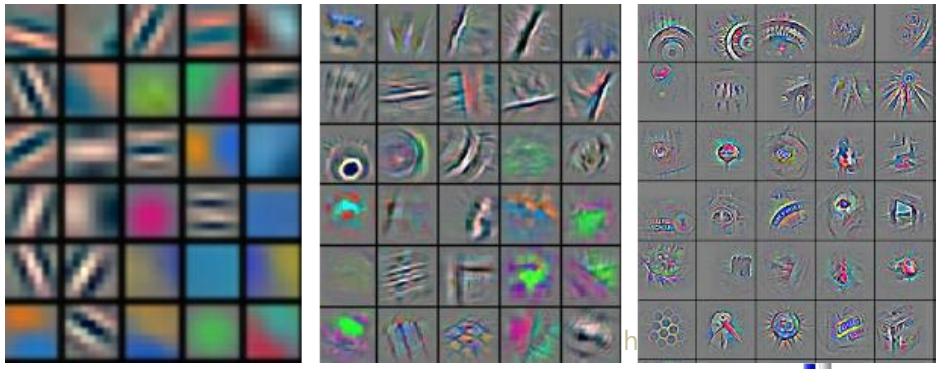
- Clearly, too many parameters
- With a only 30x30 pixels image and a single hidden layer of depth 5 we would need 85K parameters
  - With a 256x256 image we would need $46x10^6$ parameters
- Problems 1: Fitting a model with that many parameters is not easy
- Problems 2: Finding the data for such a model is not easy
- Problems 3: Are all these weights necessary?

Hypothesis

- Imagine
  - With the right amount of data …
  - … and if we connect all inputs of layer l with all outputs of layer $l + 1$, …
  - … and if we would visualize the (2d) filters (local connectivity → 2d) …
  - … we would see very similar filters no matter their location
- Why?
  - Natural images are stationary
  - Visual features are common for different parts of one or multiple image



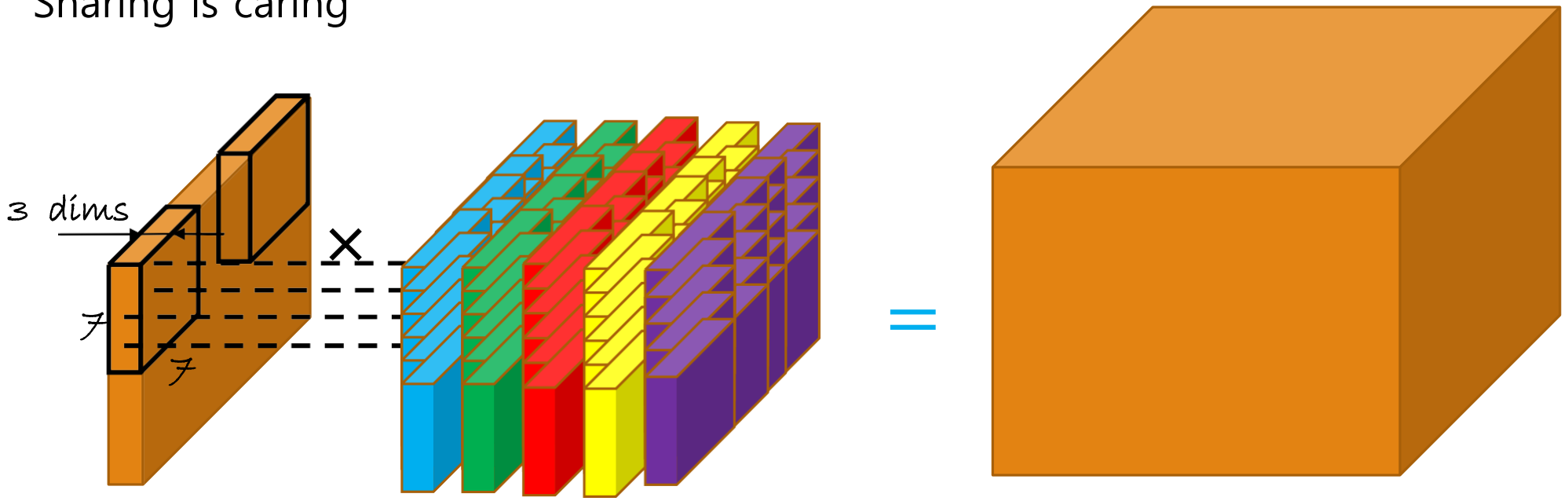




Source: http://uvadlc.github.io/

Solution? Share!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring



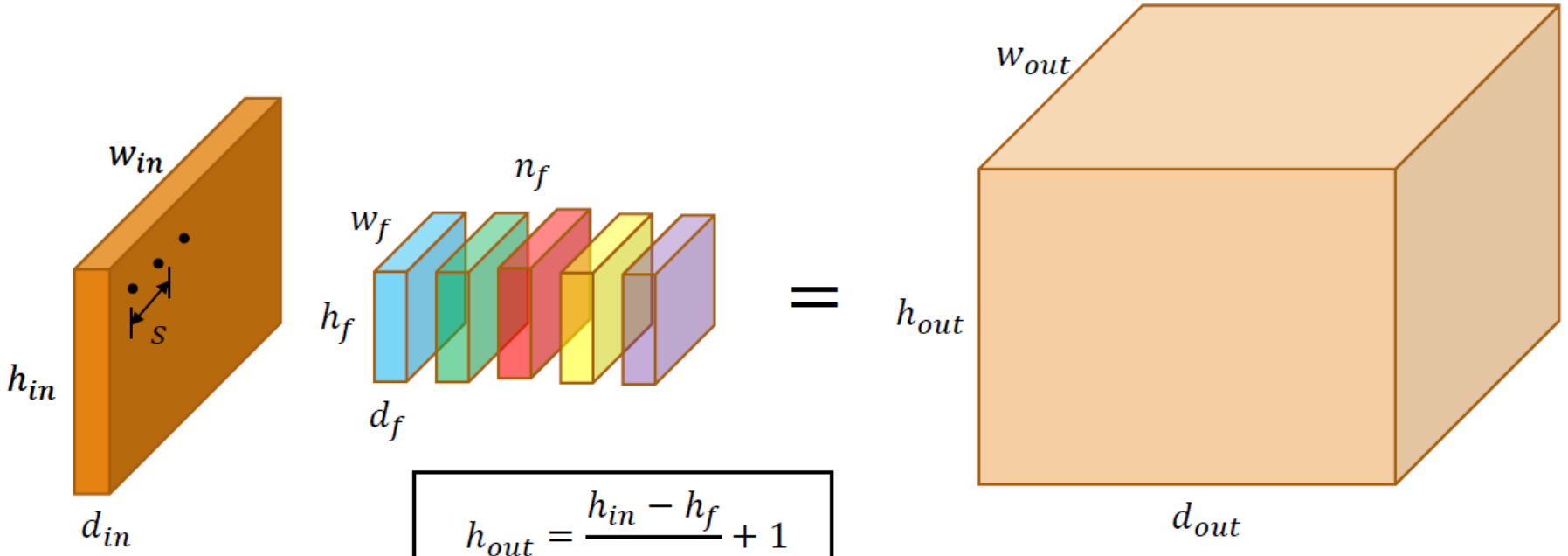
Assume the image is 30x30x3.
1 column of filters common across the image.
How many parameters in total?

Depth of 5
× 7 * 7 * 3 parameters per filter

735 parameters in total

Source: http://uvadlc.github.io/

Output dimensions?



$$h_{out} = \frac{h_{in} - h_f}{s} + 1$$

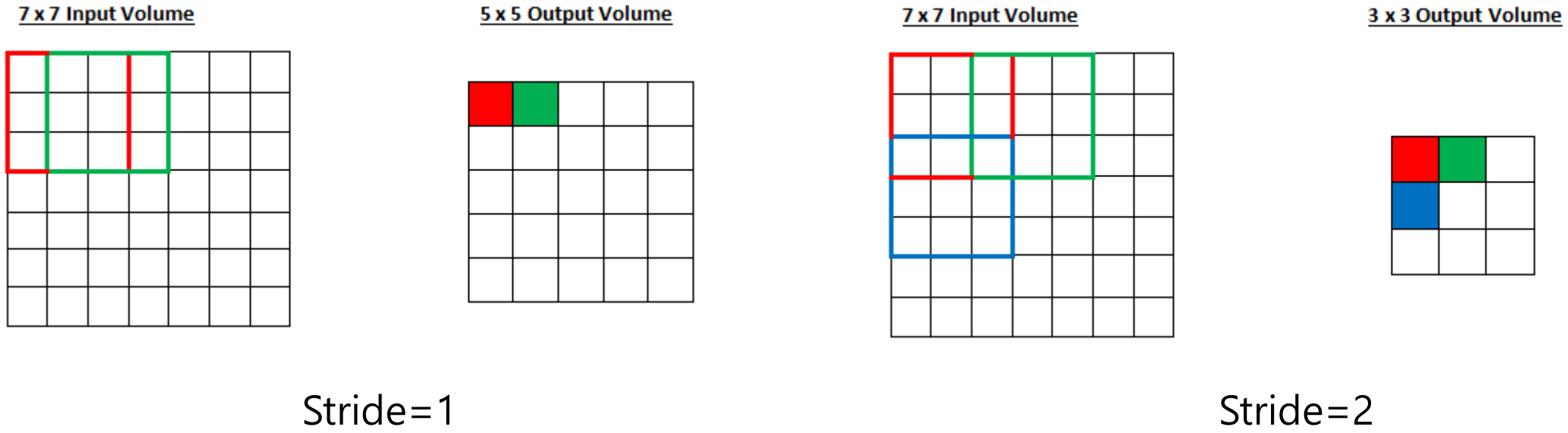
$$w_{out} = \frac{w_{in} - w_f}{s} + 1$$

$$d_{out} = n_f$$

$$d_f = d_{in}$$

Stride

- Stride
  - Stride controls how the filter convolves around the input volume. The amount by which the filter shifts is the stride. The stride was implicitly set at 1.
- Example

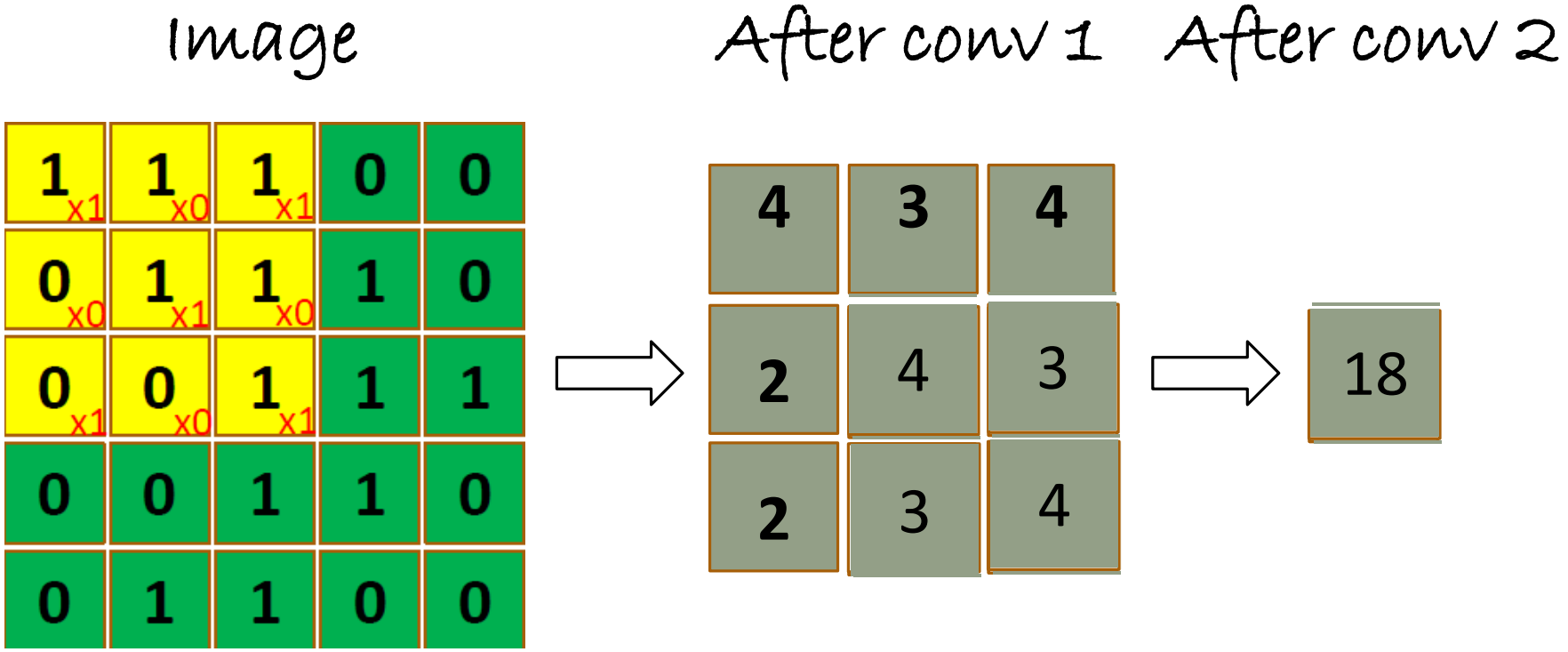


Stride=1

Stride=2

Problem, again :S

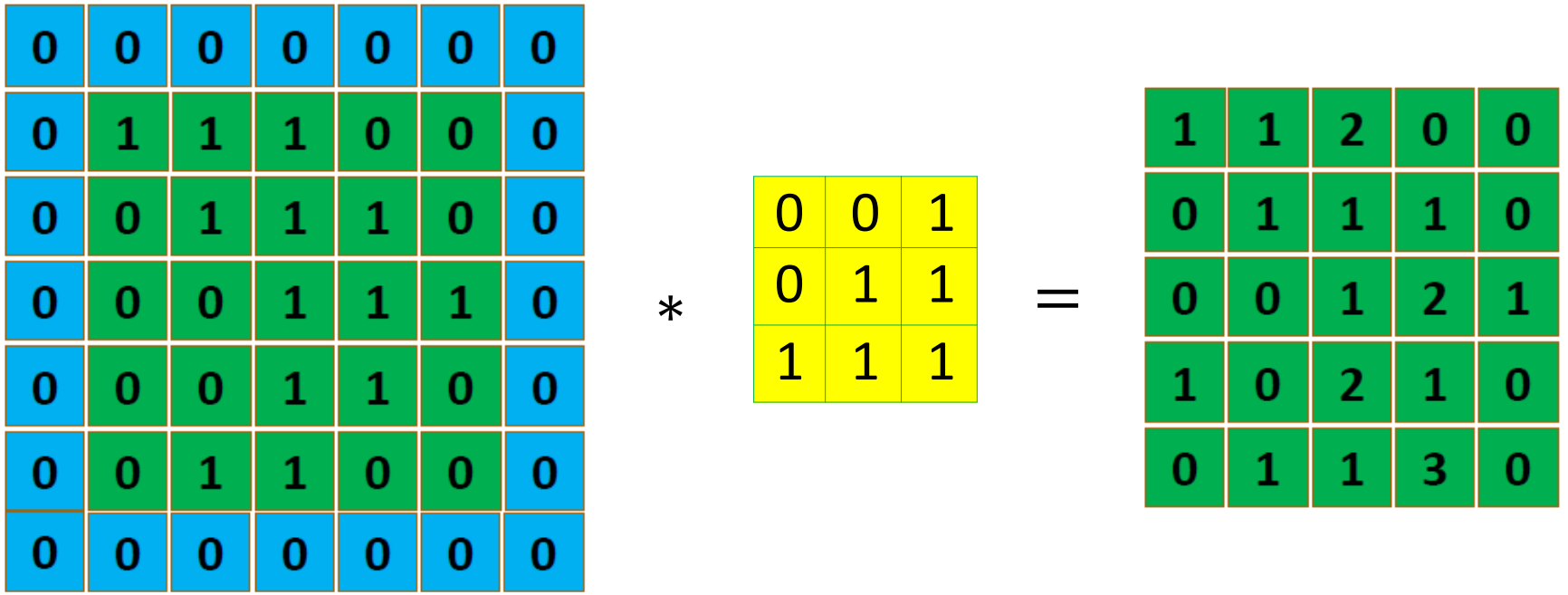
- Our images get smaller and smaller
- Not too deep architectures
- Details are lost → recognition accuracy drops



Source: http://uvadlc.github.io/

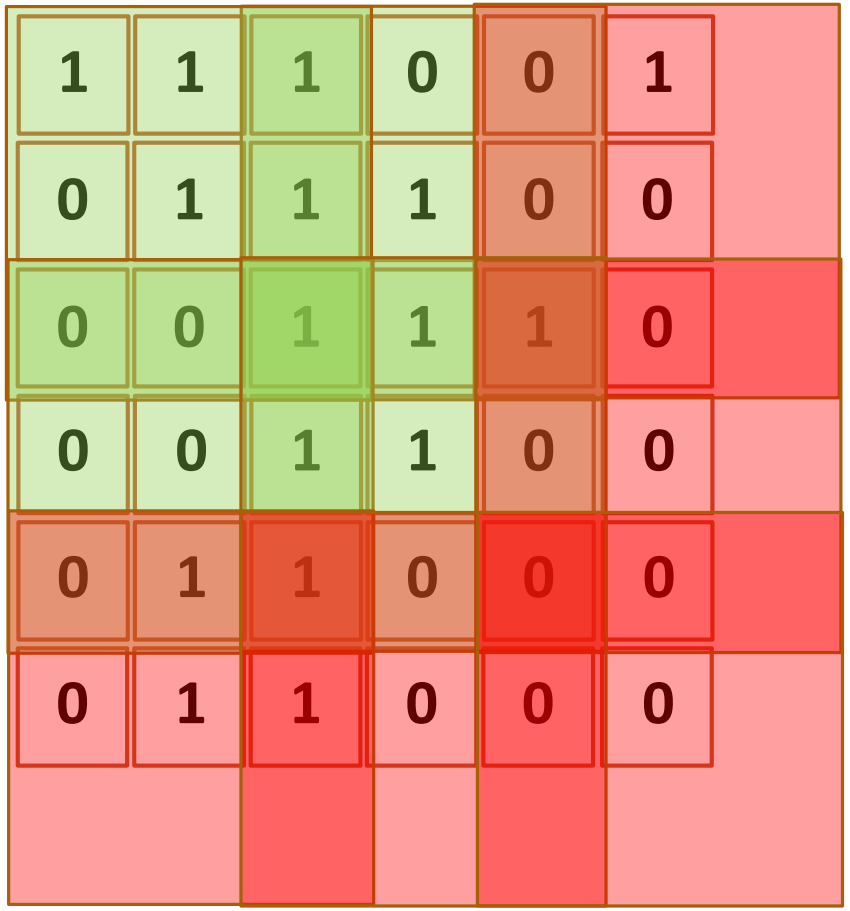
Solution? Zero-padding!

- For s=1, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0



Good practices

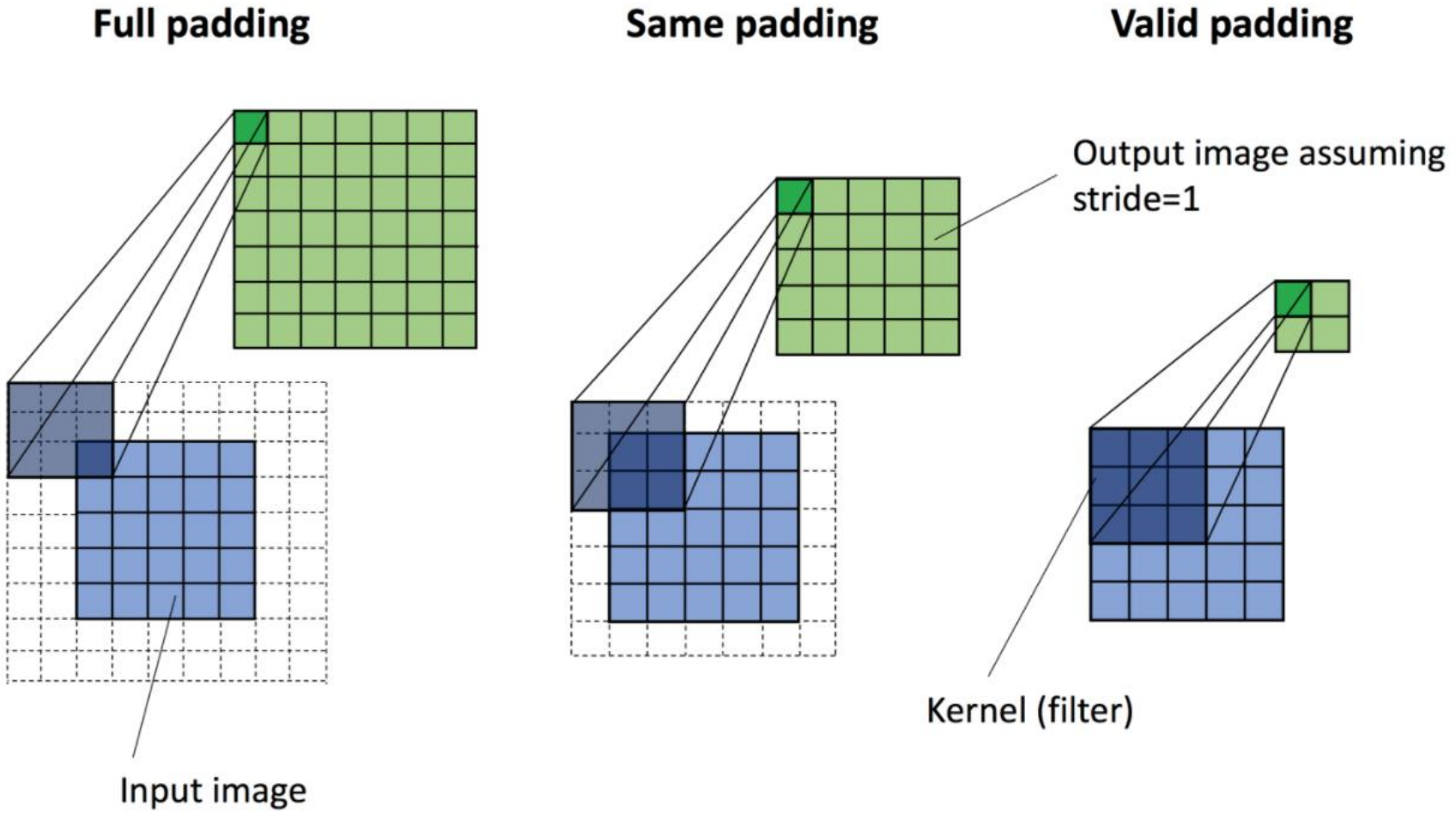
- Resize the image to have a size in the power of 2
  - Stride $s = 1$
  - A filter of $(h_f, w_f) = [3 \times 3]$ works quite alright with deep architectures
  - Add 1 layer of zero padding
- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 2$, $(h_f, w_f) = [3 \times 3]$
  - image size $(h_{in}, w_{in}) = [6 \times 6]$
  - $(h_{out}, w_{out}) = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored



Source: http://uvadlc.github.io/

The effect of zero-padding in a convolution

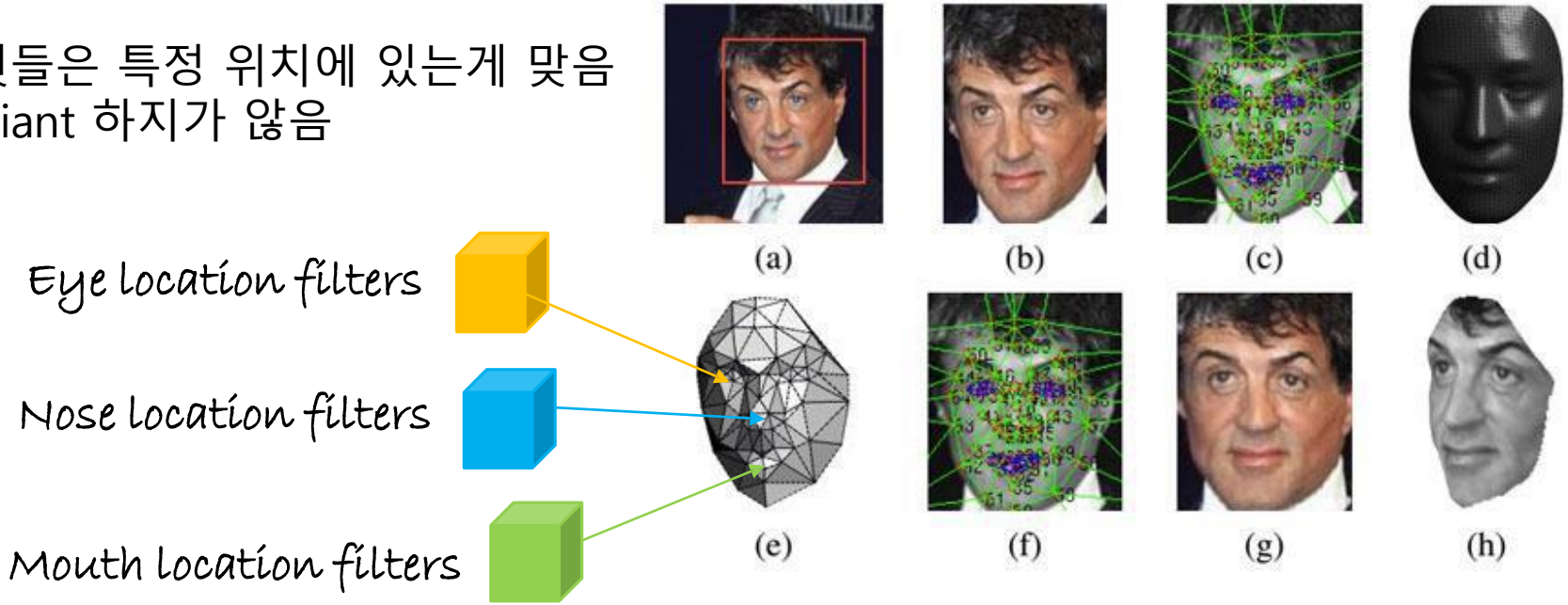
- Performing a discrete convolution in one dimension (feature map size: m)



Sometimes convolutional filters are not preferred

- When images are registered and each pixel has a particular significance
  - E.g. after face alignment specific pixels hold specific types of inputs, like eyes, nose, etc.
- In these cases maybe better every spatial filter to have different parameters
  - Network learns particular weights for particular image locations [Taigman 2014]

코나 눈 이런 것들은 특정 위치에 있는게 맞음
→ Spatial invariant 하지가 않음



Source: http://uvadlc.github.io/

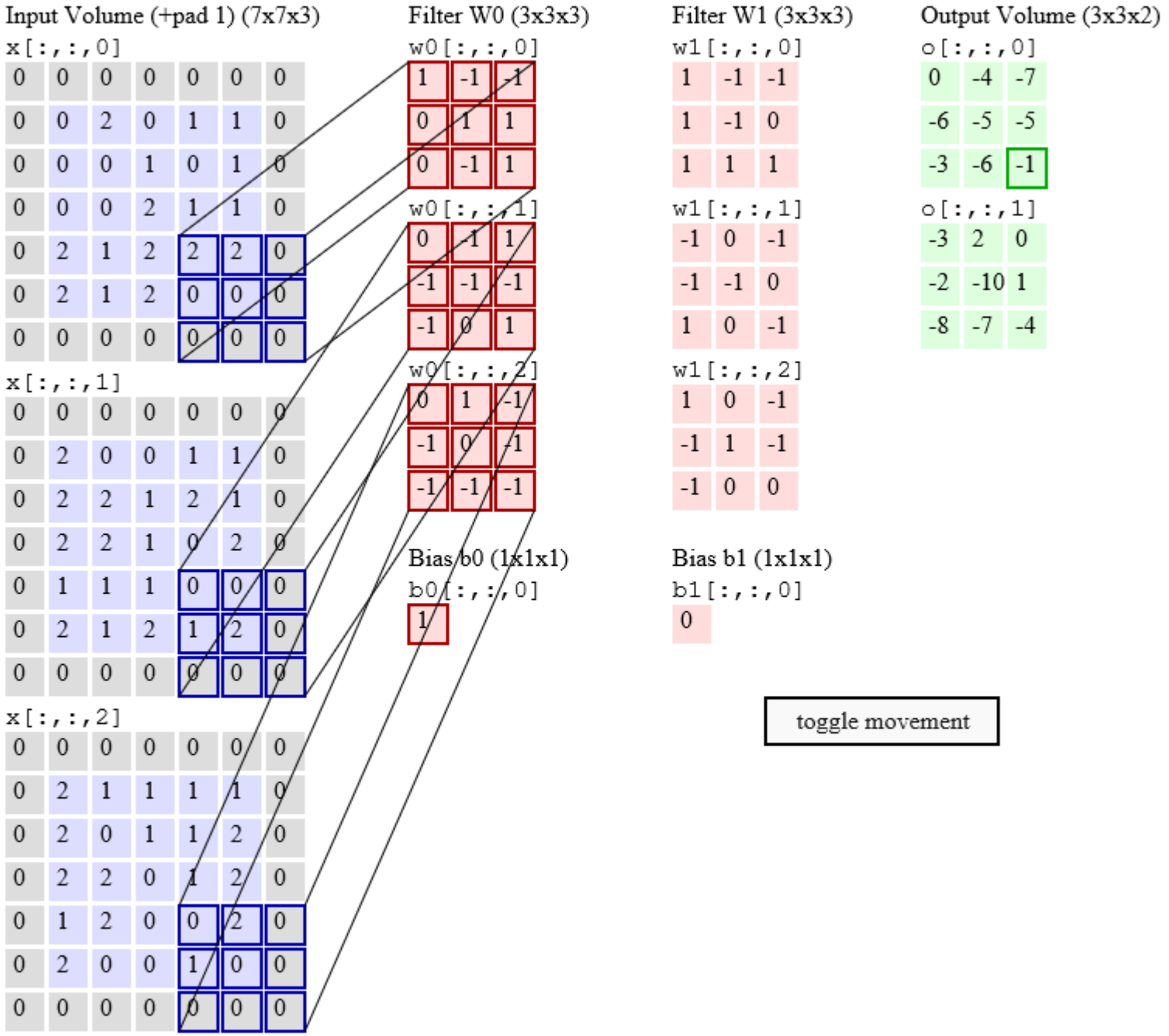
Convolution Demo

Accepts input volume of size
- W=5, H=5, D=3

CONV layer parameters are
- Number of filters K=2,
- their spatial extent F=3,
- the stride S=2,
- the amount of zero padding P=1

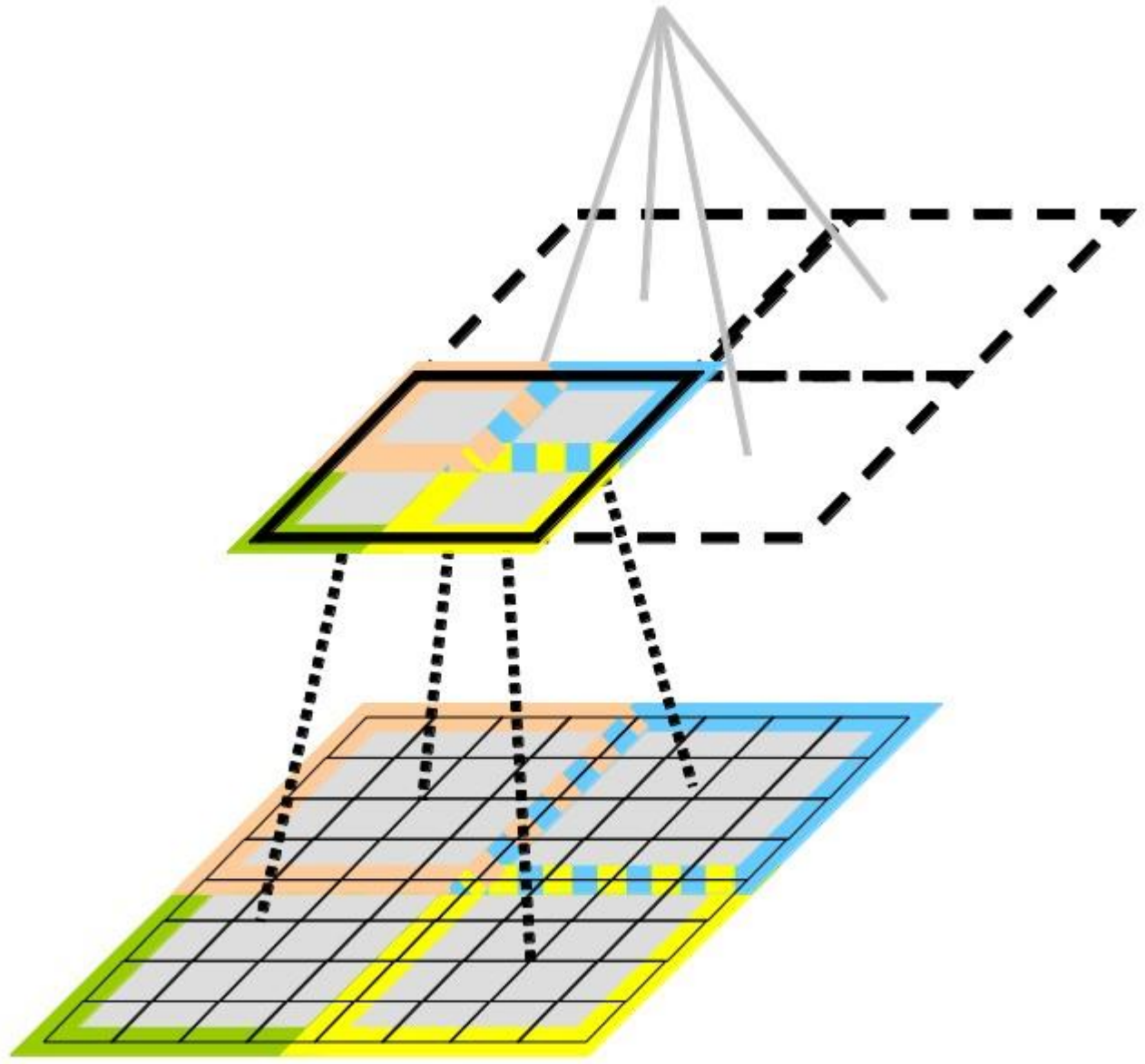
Produces a volume of size
- W′=(w-F+2P)/S+1
- H′=(H-F+2P)/S+1
- D′=K

```
V[0,0,0] = np.sum(X[:3,:3,:] * W0) + b0
V[1,0,0] = np.sum(X[2:5,:3,:] * W0) + b0
V[2,0,0] = np.sum(X[4:7,:3,:] * W0) + b0
.....................................
```



http://cs231n.github.io/convolutional-networks/

Pooling



Source: http://uvadlc.github.io/

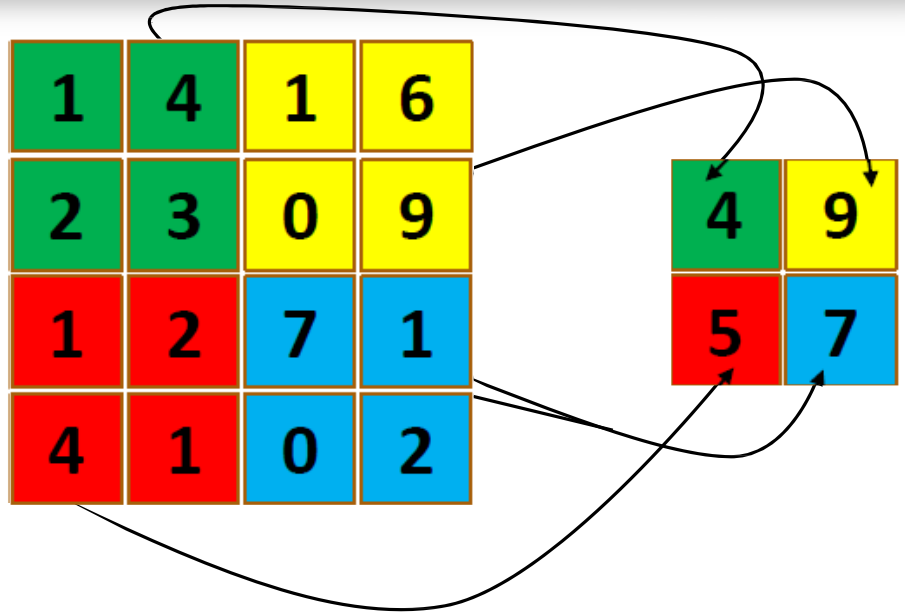
Pooling

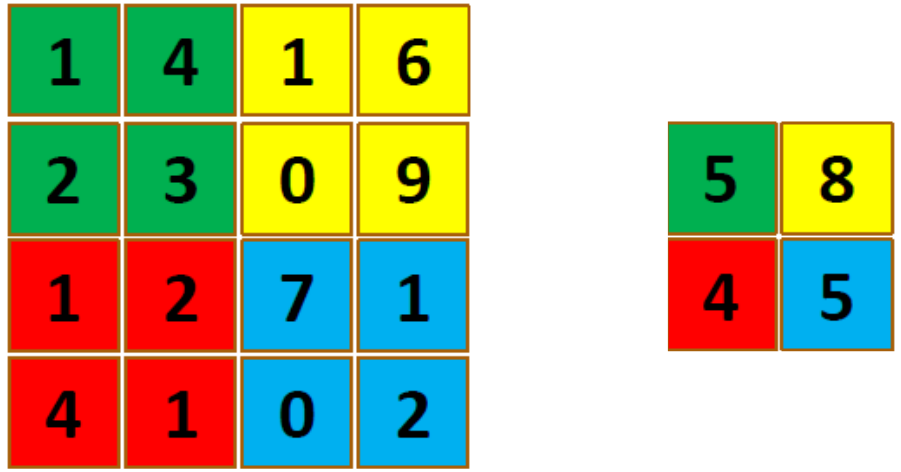
- Aggregate multiple values into a single value
  - Invariance to small transformations
  - Reduces the size of the layer output/input to next layer → Faster computations
  - Keeps most important information for the next layer
- Max pooling
- Average pooling

Source: http://uvadlc.github.io/

Max / Average pooling (New module!)

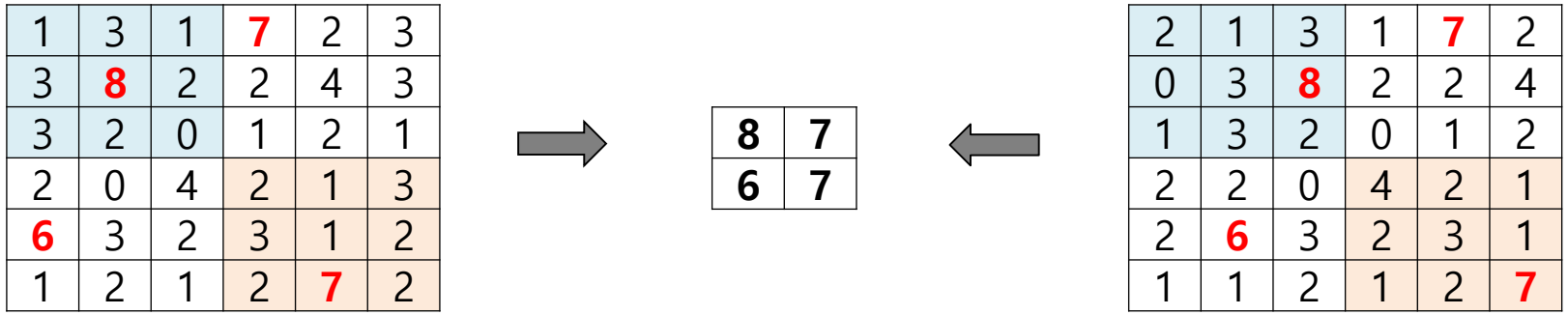
- Max pooling
  - Run a sliding window of size $[h_f, w_f]$
  - At each location keep the maximum value
  - Activation function: $i_{\max}, j_{\max} = \arg max_{i,j \in \Omega(r,c)} x_{ij}$
  - Gradient wrt input: $\frac{\partial a_{rc}}{\partial x_{ij}} = \delta_{i=i_{max}, j=j_{max}}$
  - The preferred choice of pooling
- Average pooling
  - Run a sliding window of size $[h_f, w_f]$
  - At each location keep the maximum value
  - Activation function: $a_{rc} = \frac{1}{rc} \sum_{i,j \in \Omega(r,c)} x_{ij}$
  - Gradient wrt input: $\frac{\partial a_{rc}}{\partial x_{ij}} = \frac{1}{rc}$





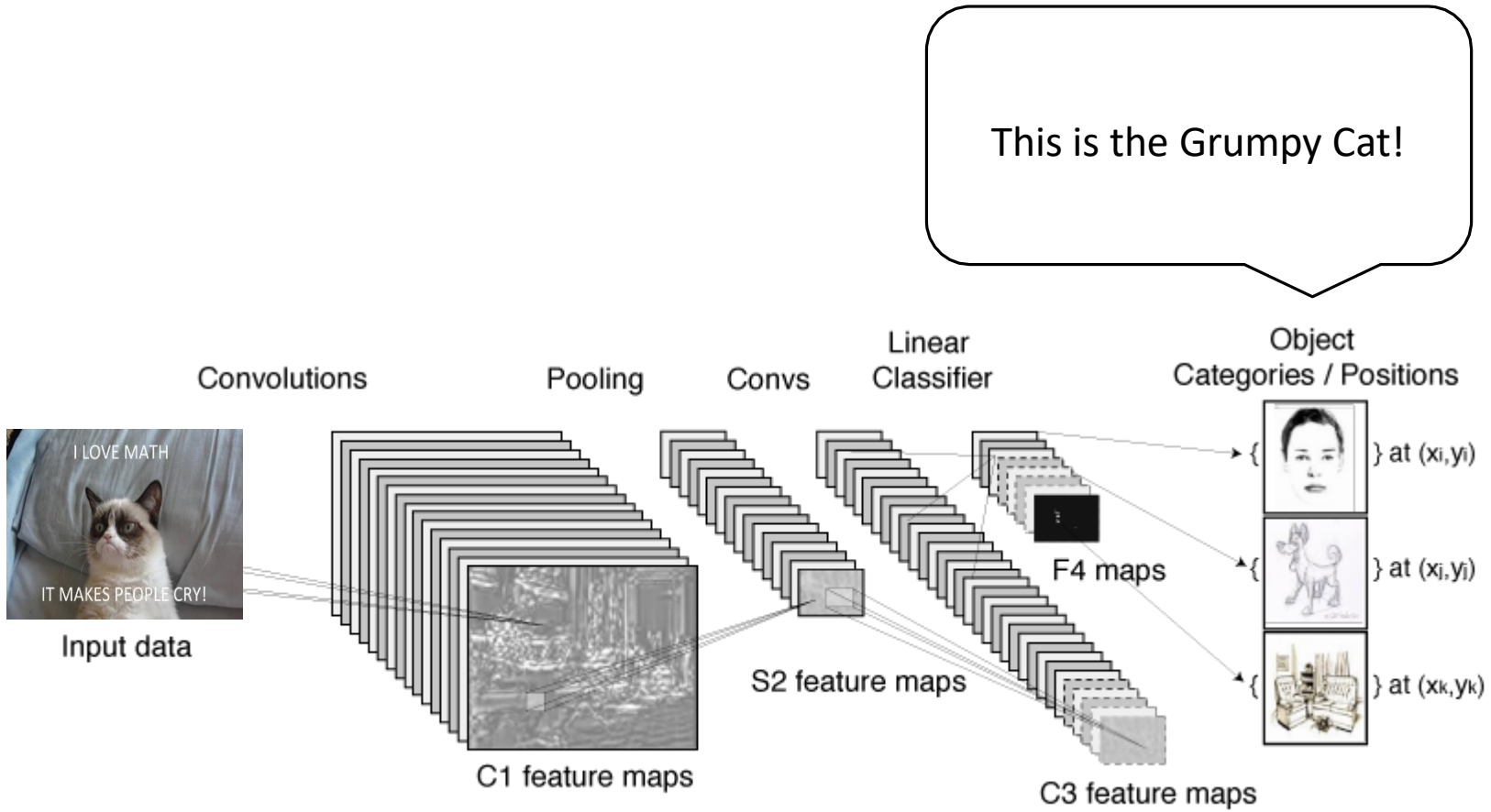
Pooling Layer

- Pooling Layer
  - No parameters to learn
  - No change in the number of channels
  - Less susceptible to changes in input (Robust)



Source: http://uvadlc.github.io/

Convnets for Object Recognition



Source: http://uvadlc.github.io/

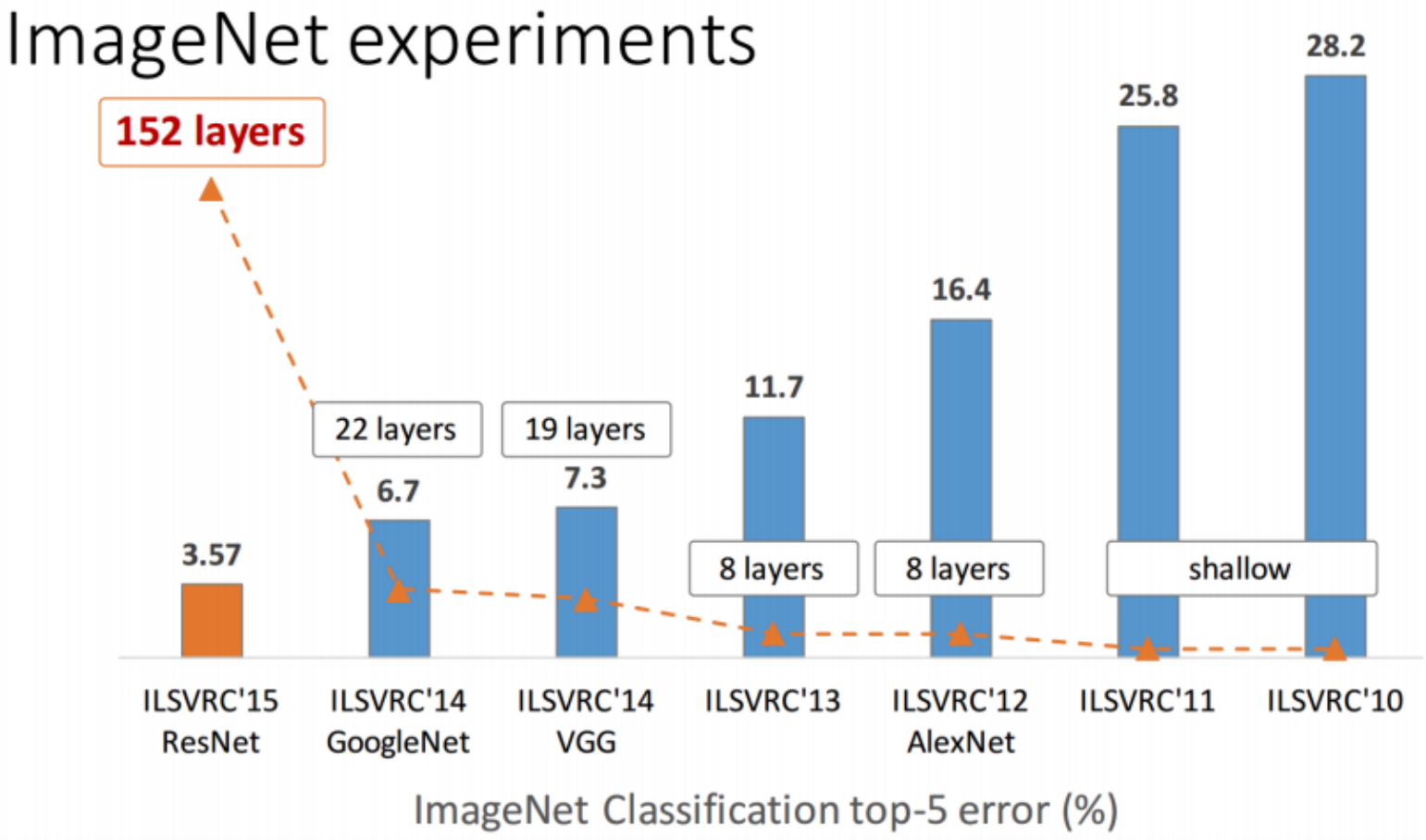
t-SNE embedding of a set of images based on their CNN codes

Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based

ImageNet

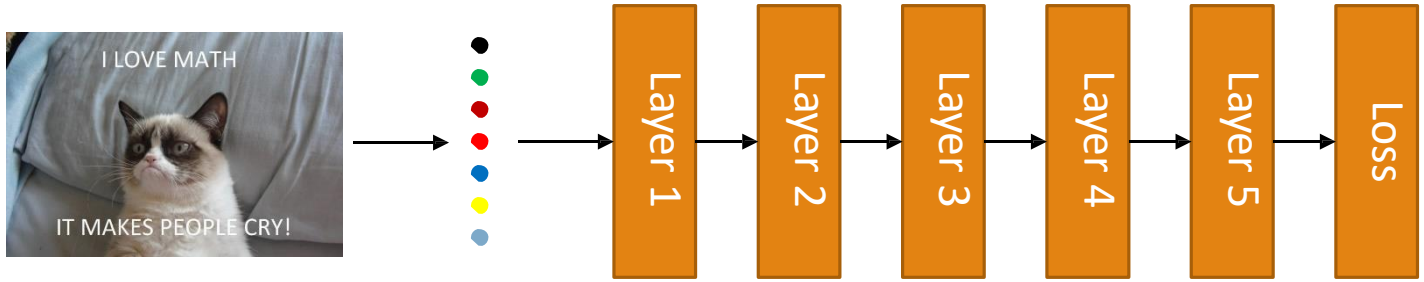
Performance of CNN

- ImageNet

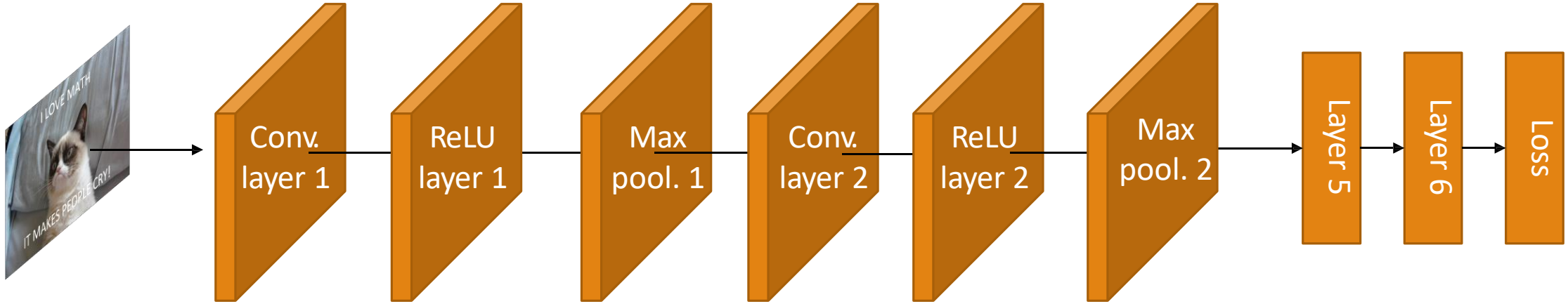


Standard Neural Network vs Convnets

Neural Network



Convolutional Neural Network



Convnets in practice

- Several convolutional layers
  - 5 or more
- After the convolutional layers non-linearities are added
  - The most popular one is the ReLU
- After the ReLU usually some pooling
  - Most often max pooling
- After 5 rounds of cascading, vectorize last convolutional layer and connect it to a fully connected layer
- Then proceed as in a usual neural network

CNN Case Study 1: Alexnet

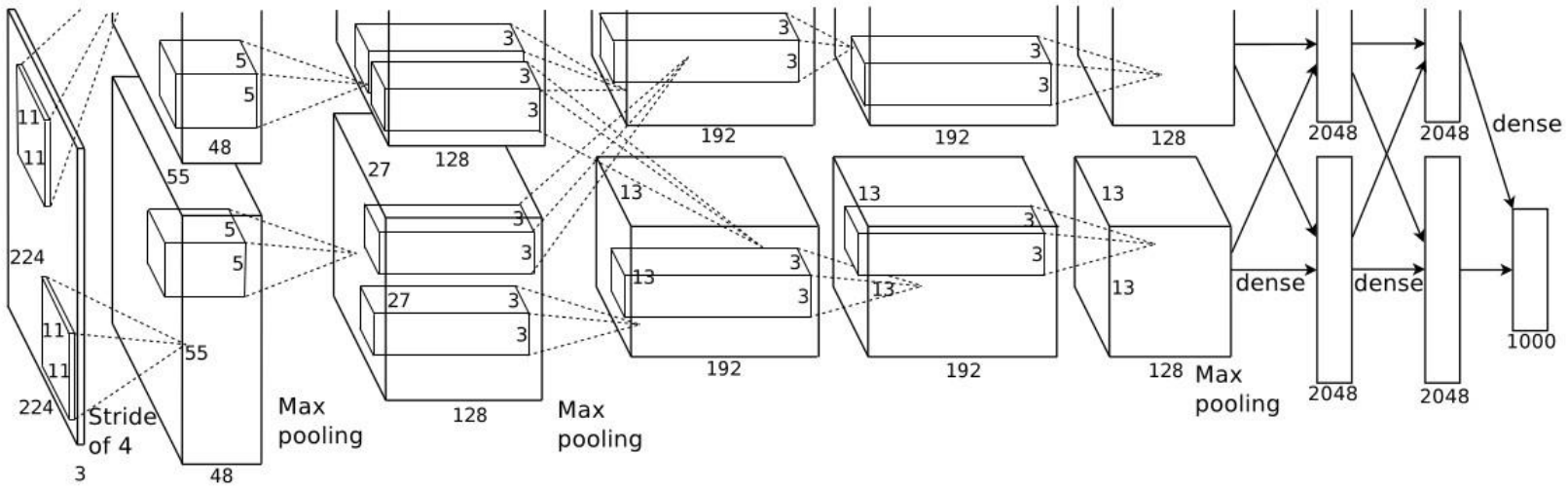
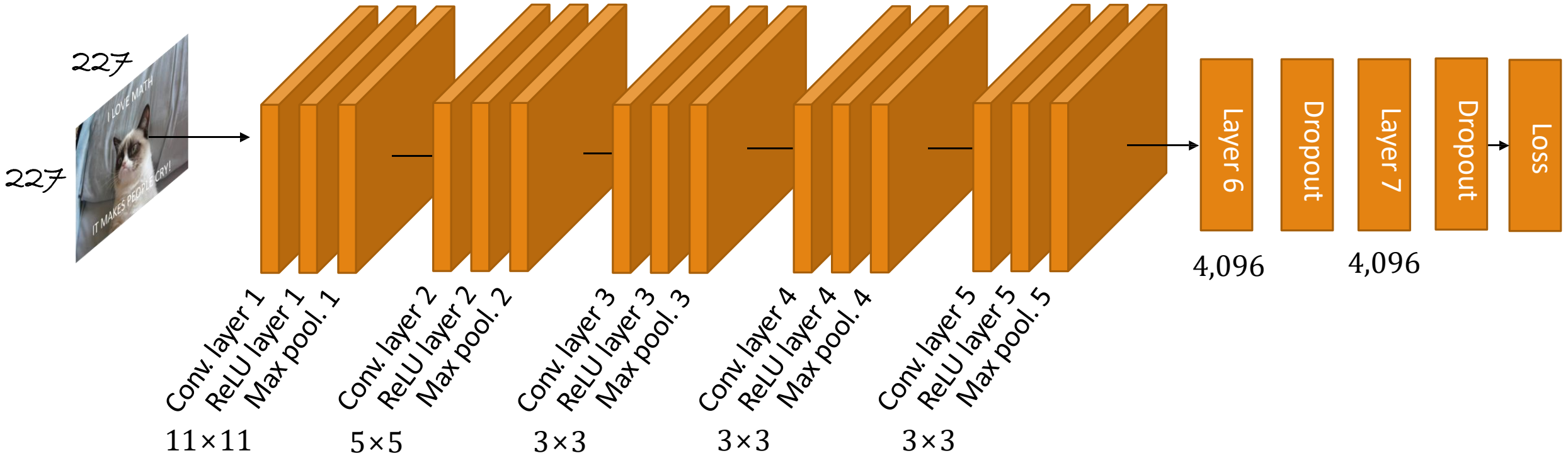


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Source: http://uvadlc.github.io/

Architetural detials

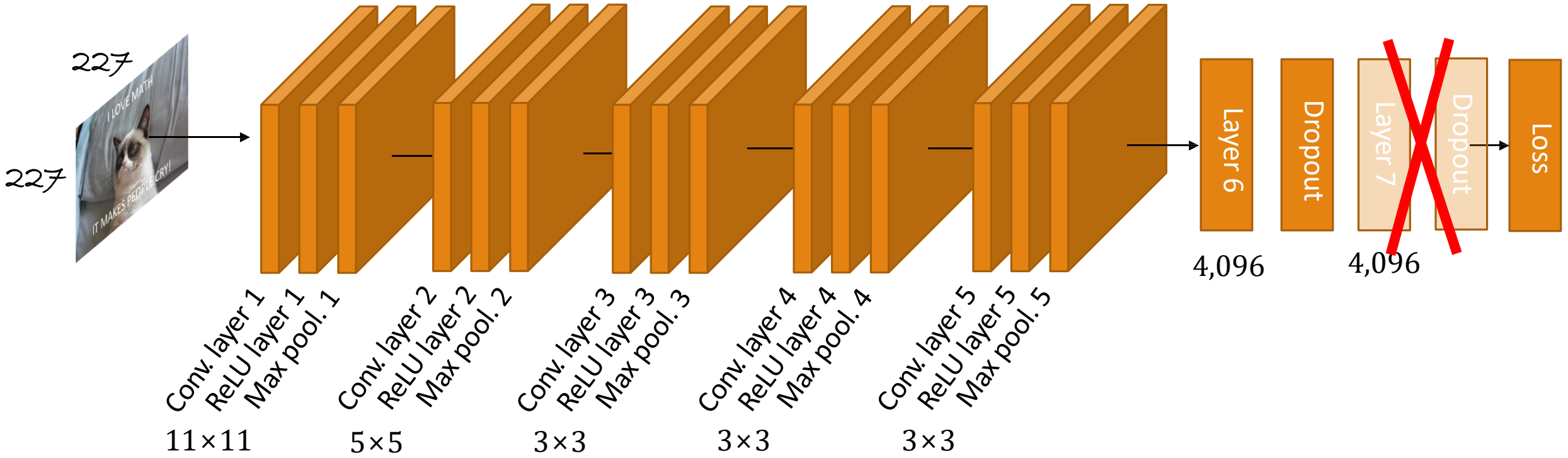
18.2% error in Imagenet



Source: http://uvadlc.github.io/

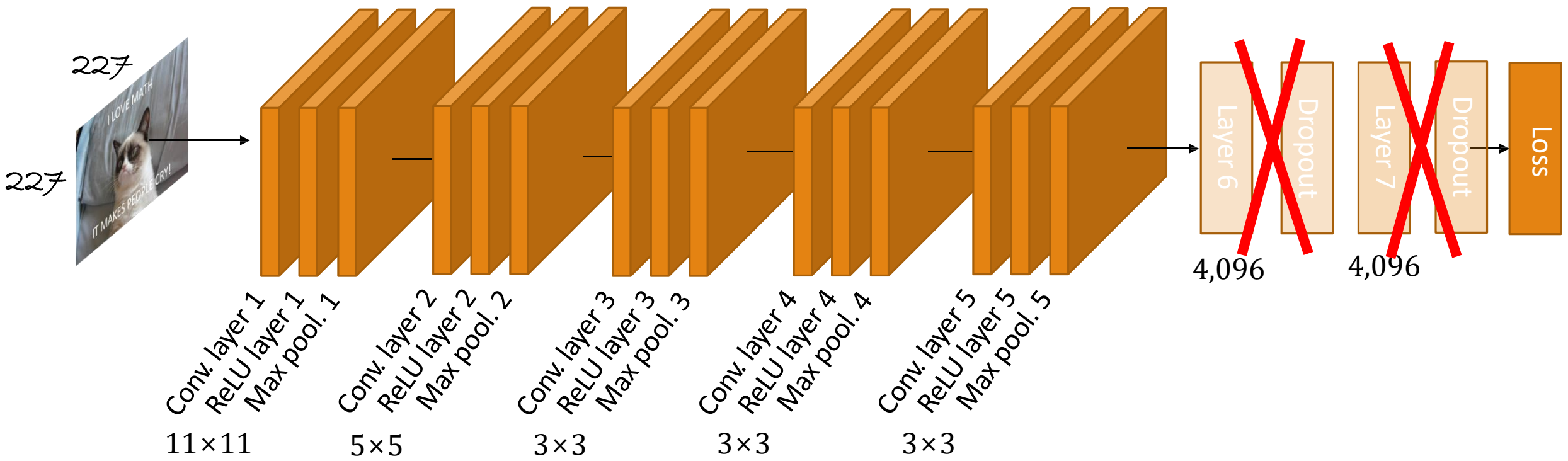
Removing layer 7

1.1% drop in performance, 16 million less parameters



Removing layer 6, 7

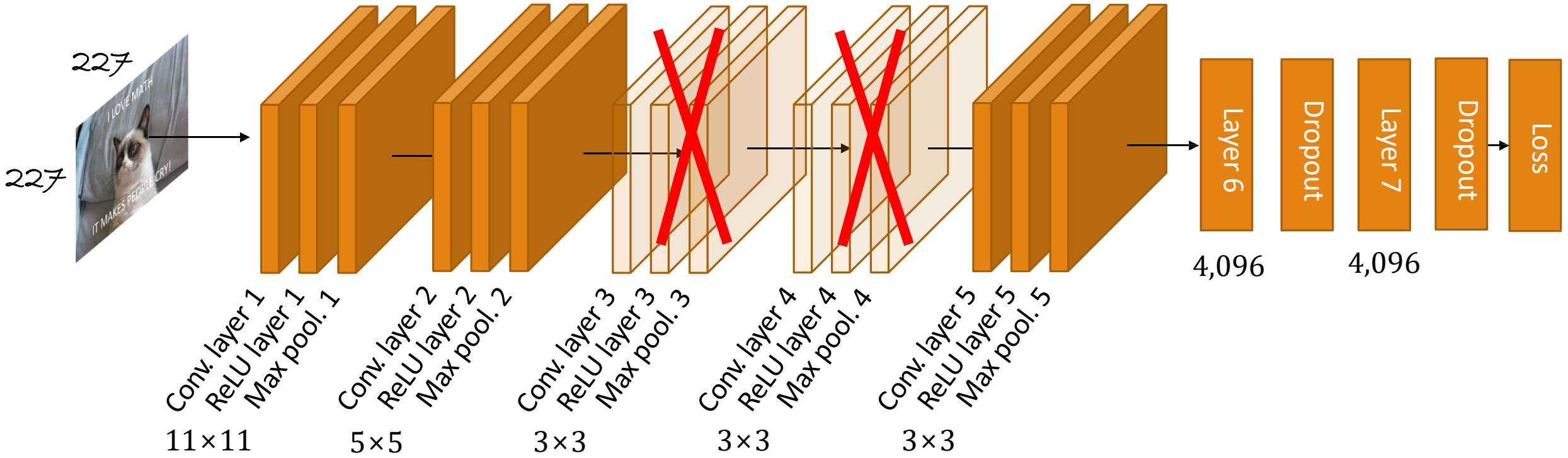
5.7% drop in performance, 50 million less parameters



Source: http://uvadlc.github.io/

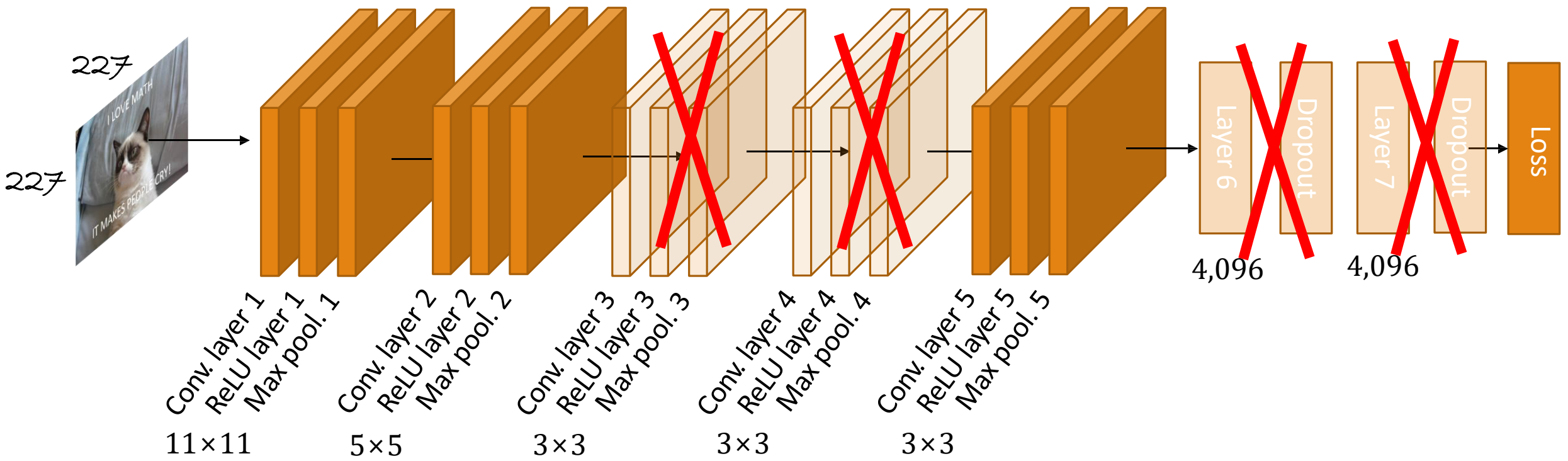
Removing layer 3, 4

3.0% drop in performance, 1 million less parameters. Why?



Removing layer 3, 4, 6, 7

33.5% drop in performance. Conclusion? Depth!



Source: http://uvadlc.github.io/

The effect of the architecture

Error %	Train Top-1	Val Top-1	Val Top-5
Our replication of Krizhevsky *et al.* [18], 1 convnet	35.1	40.5	18.1
Removed layers 3,4	41.8	45.4	22.1
Removed layer 7	27.4	40.0	18.4
Removed layers 6,7	27.4	44.8	22.4
Removed layer 3,4,6,7	71.1	71.3	50.1
Adjust layers 6,7: 2048 units	40.3	41.7	18.8
Adjust layers 6,7: 8192 units	26.8	40.0	18.1
Our Model (as per Fig. 3)	33.1	38.4	16.5
Adjust layers 6,7: 2048 units	38.2	40.2	17.6
Adjust layers 6,7: 8192 units	22.0	38.8	17.0
Adjust layers 3,4,5: 512,1024,512 maps	18.8	**37.5**	**16.0**
Adjust layers 6,7: 8192 units and Layers 3,4,5: 512,1024,512 maps	**10.0**	38.3	16.9

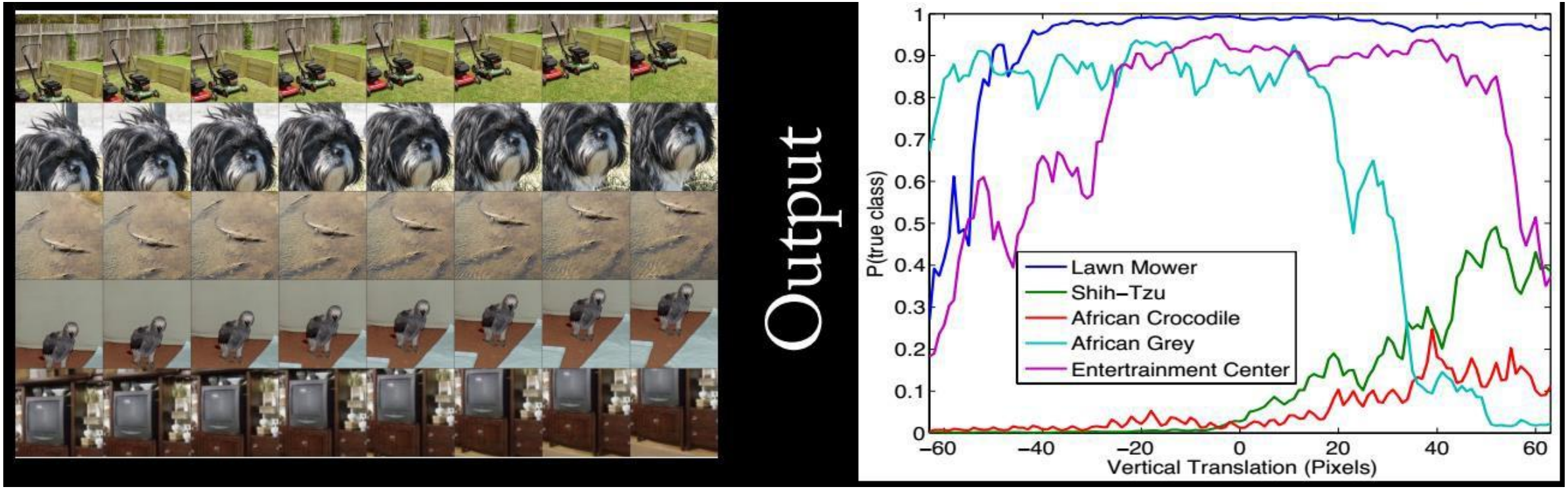
Depth is important

Early signs of overfitting

Overfitting

Source: http://uvadlc.github.io/

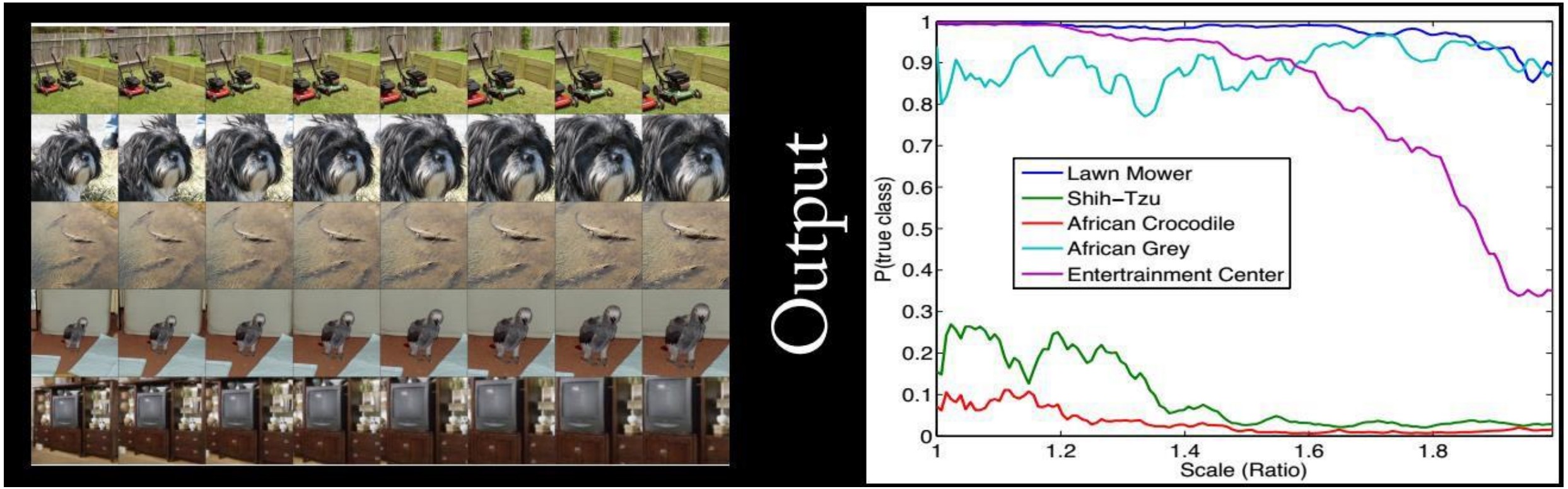
Translation invariance



AlexNet is translation invariant

Credit: R. Fergus slides in Deep Learning Summer School 2016

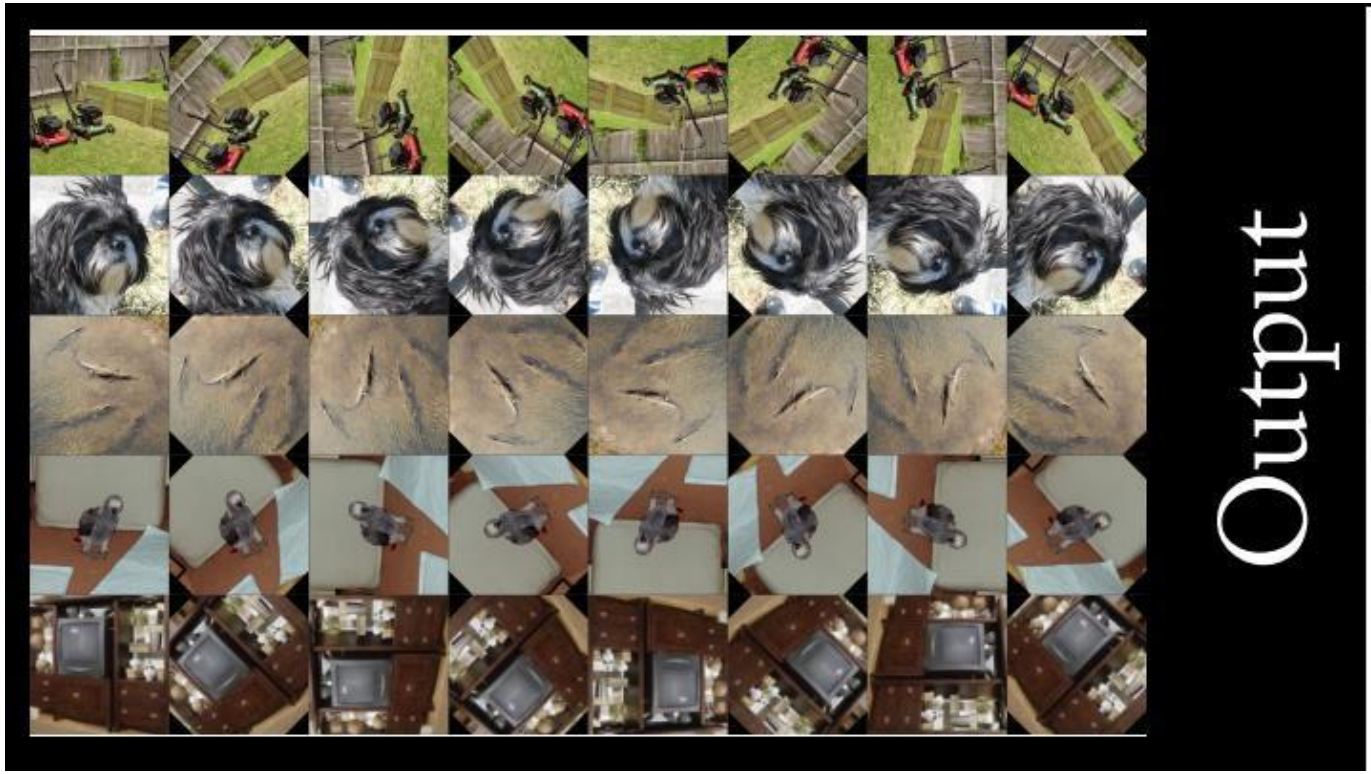
Scale invariance

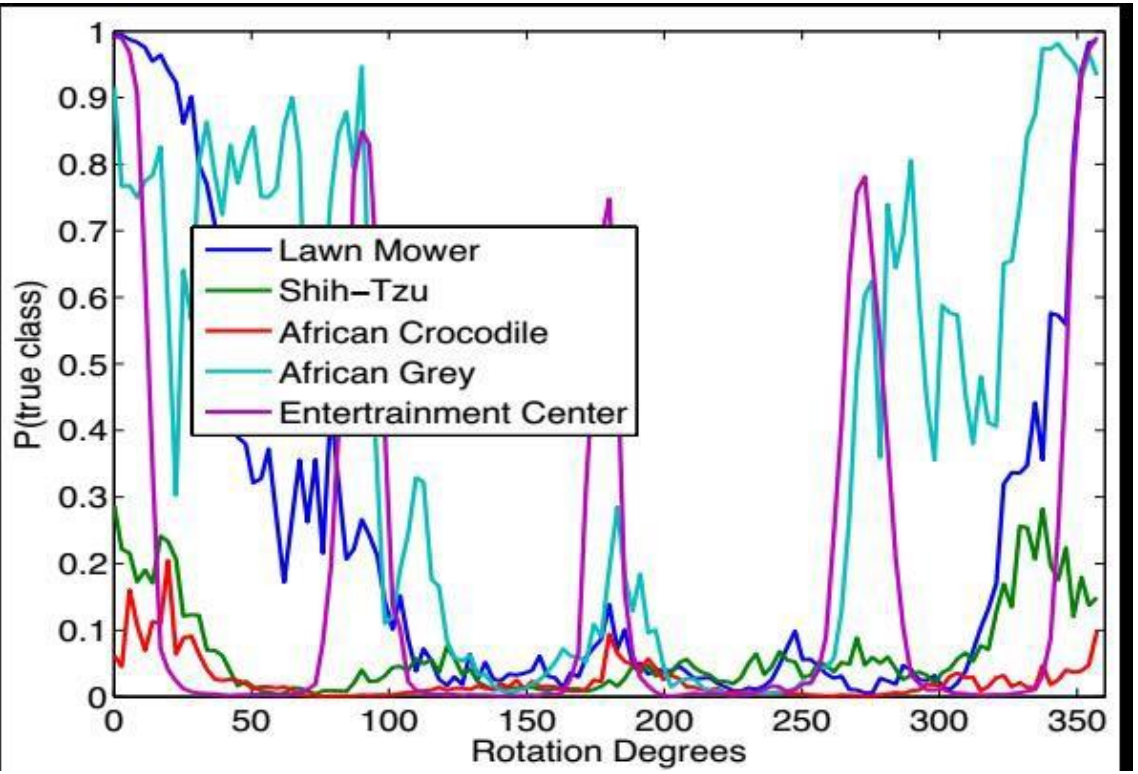


AlexNet is scale invariant in some cases

Credit: R. Fergus slides in Deep Learning Summer School 2016

Rotation invariance



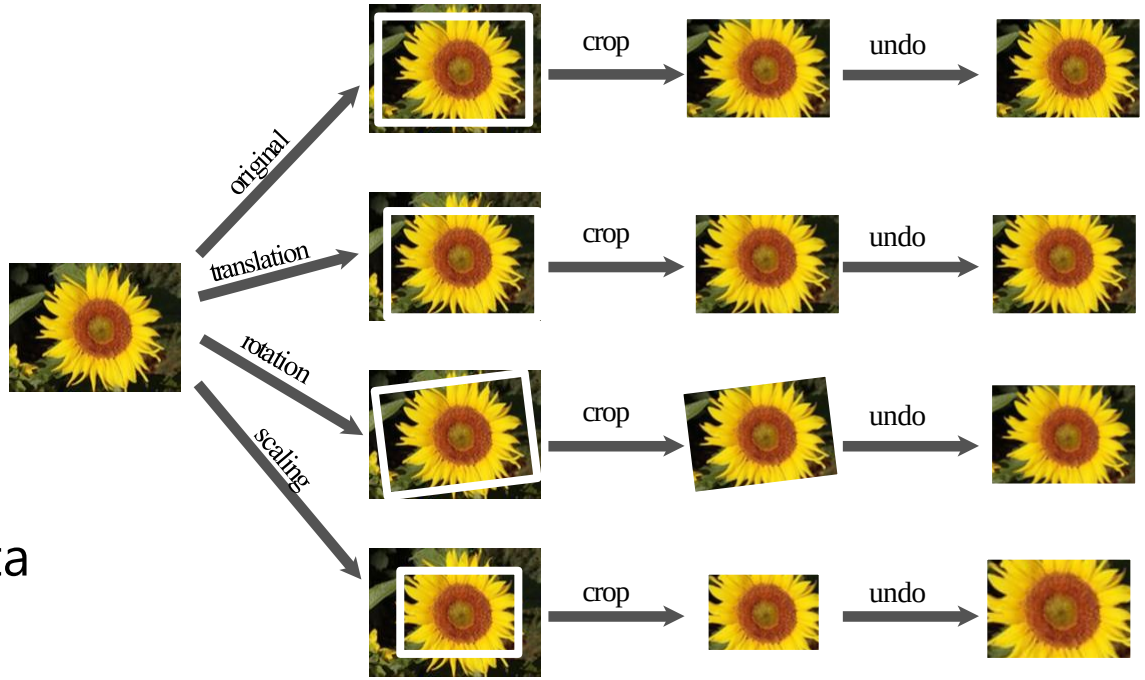


AlexNet is not rotation invariant

Credit: R. Fergus slides in Deep Learning Summer School 2016

INVARIANCE BY DATA SET EXPANSION

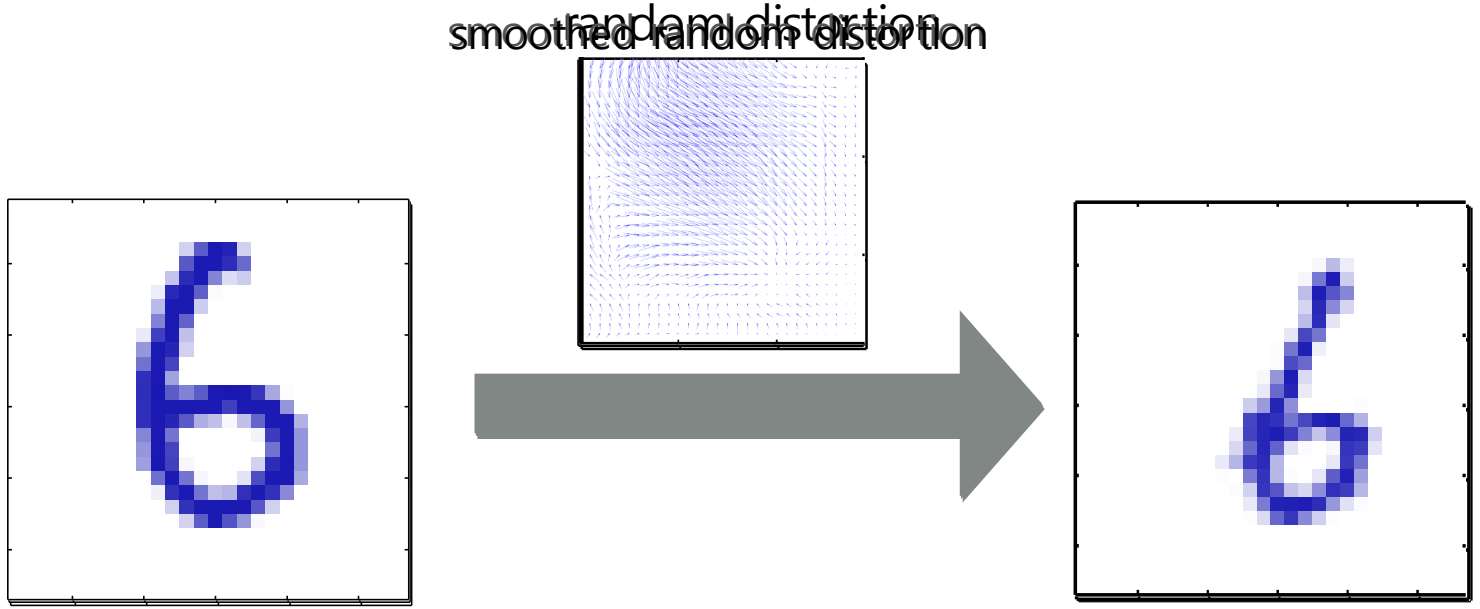
- Generating additional examples
  - Invariances built-in in convolutional network:
    - small translations: due to convolution and max pooling
    - small illumination changes: due to local contrast normalization
  - It is not invariant to other important variations such as rotations and scale changes
  - However, it's easy to artificially generate data with such transformations
    - could use such data as additional training data
    - neural network will learn to be invariant to such transformations



Source: http://uvadlc.github.io/

INVARIANCE BY DATA SET EXPANSION

- Generating additional examples, distortion field
  - Can add ''elastic'' deformations (useful in character recognition)
  - We do this by applying a ''distortion field'' to the image
    - a distortion field specifies where to displace each pixel value



See Simard et al. for more detail

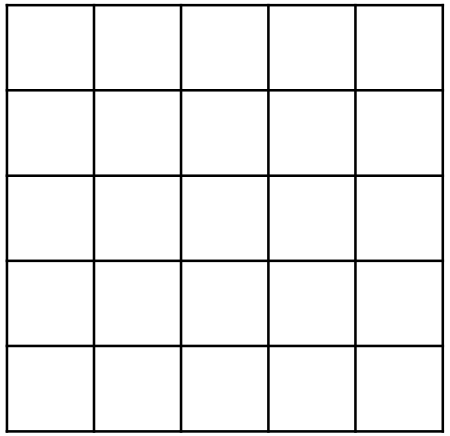
(from Bishop's book)

Understanding convnets

How large filters?

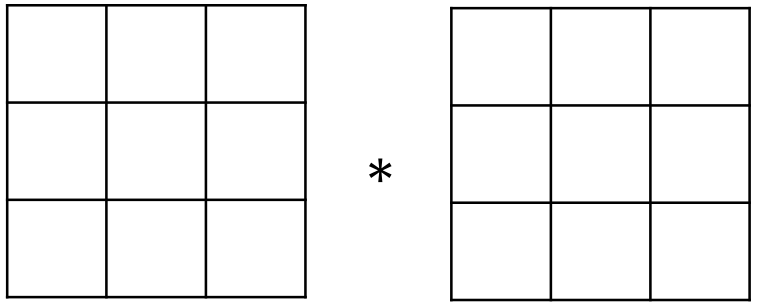
- Traditionally, medium sized filters (smaller than 11 × 11)
- Modern architectures prefer small filter sizes (e.g. 3 × 3)
- We lose frequency resolution
- Fewer parameters to train

$(2d+1)^2$



$vs.$

$(d+1)^2$ $(d+1)^2$



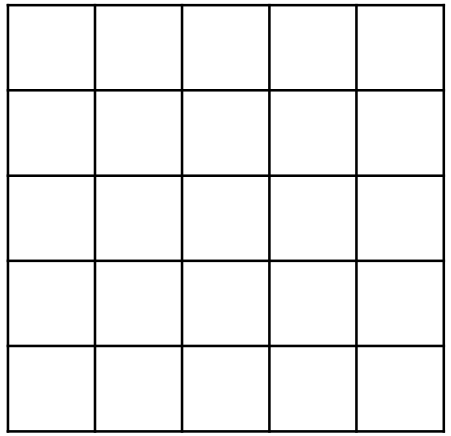
$(Layer\ l) * (Layer\ l+1)$

Source: http://uvadlc.github.io/

How large filters?

- Traditionally, medium sized filters (smaller than 11 × 11)
- Modern architectures prefer small filter sizes (e.g. 3 × 3)
- We lose frequency resolution
- Fewer parameters to train
- Deeper networks of cascade filters
  - Still, the same output dimensionalities

$(2d+1)^2$

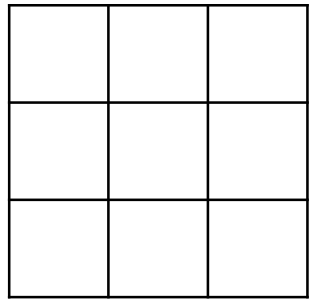


For stride 1 the first feature map has dimensionality

$$\frac{H-2d-1}{1}+1=H-2d$$

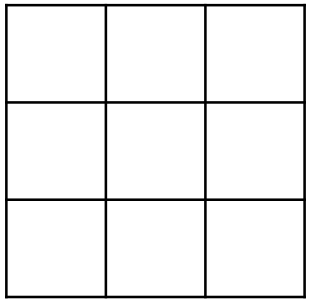
$vs.$

$(d+1)^2$



$*$

$(d+1)^2$



For stride 1 the first feature map has dimensionality $H-d$, the second image, $\frac{H-d-d-1}{1}+1=H-2d$

$(Layer\ l) * (Layer\ l+1)$

Source: http://uvadlc.github.io/

Several interesting questions

- What do the image activations in different layers look like?
- What types of images create the strongest activations?
- What are the activations for the class “ostrich”?
- Do the activations occur in meaningful positions?

Feature maps strength

- The convolution activations are also called feature maps
  - A deep network has several hierarchical layers
  - hence several feature maps
- Naturally, given an image some feature maps will be “stronger”
  - Contribute higher amount to the final classification score
- Why? What pixel structure excited these particular feature maps?
  - Generally, what part of the picture excites a certain feature map?

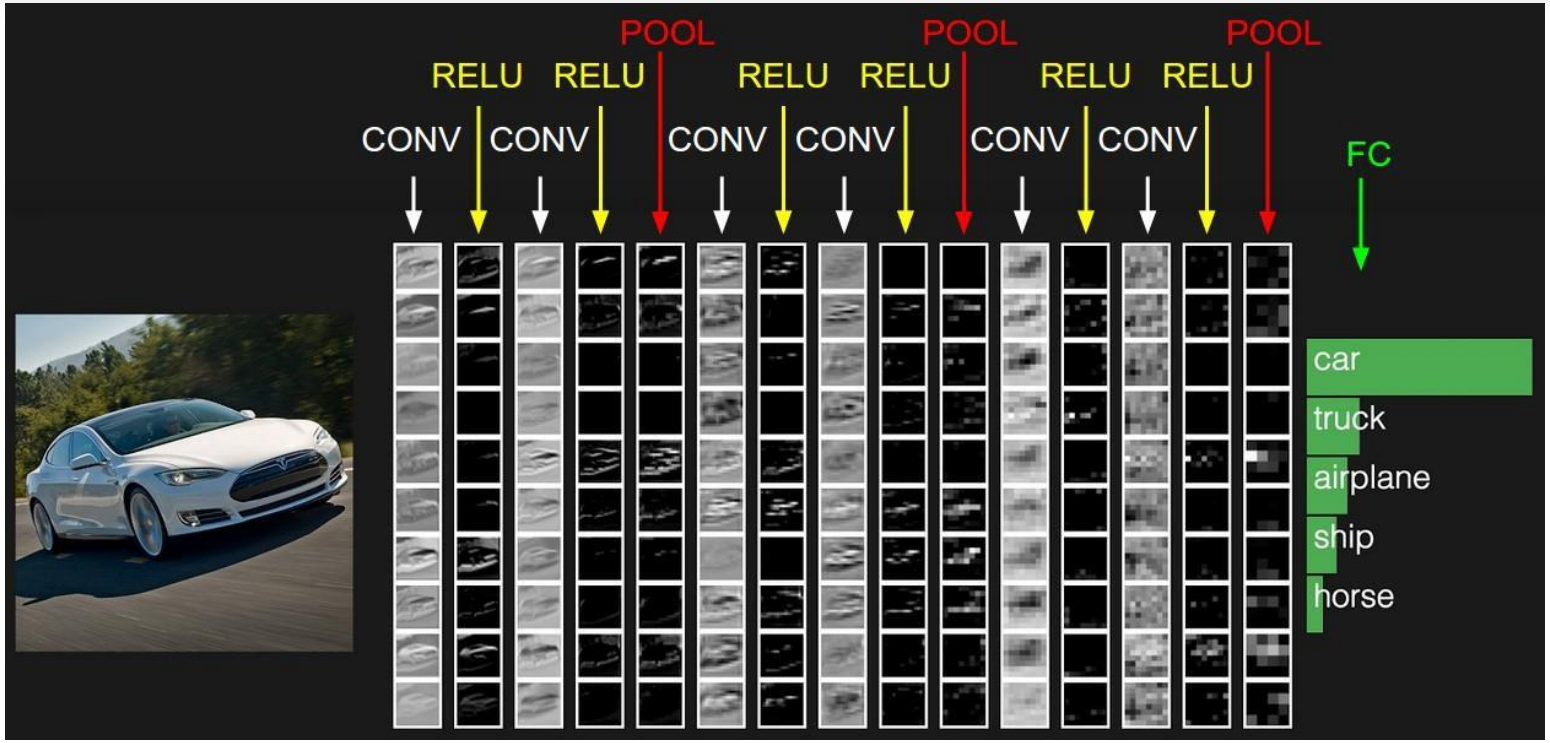
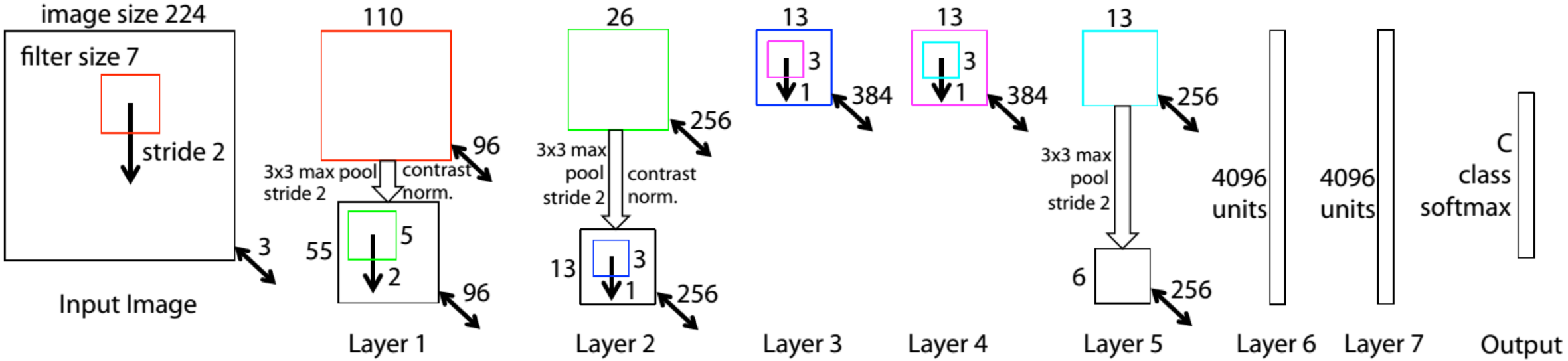


Image borrowed by A. Karpathy

Starting from a convnet [Zeiler2014]



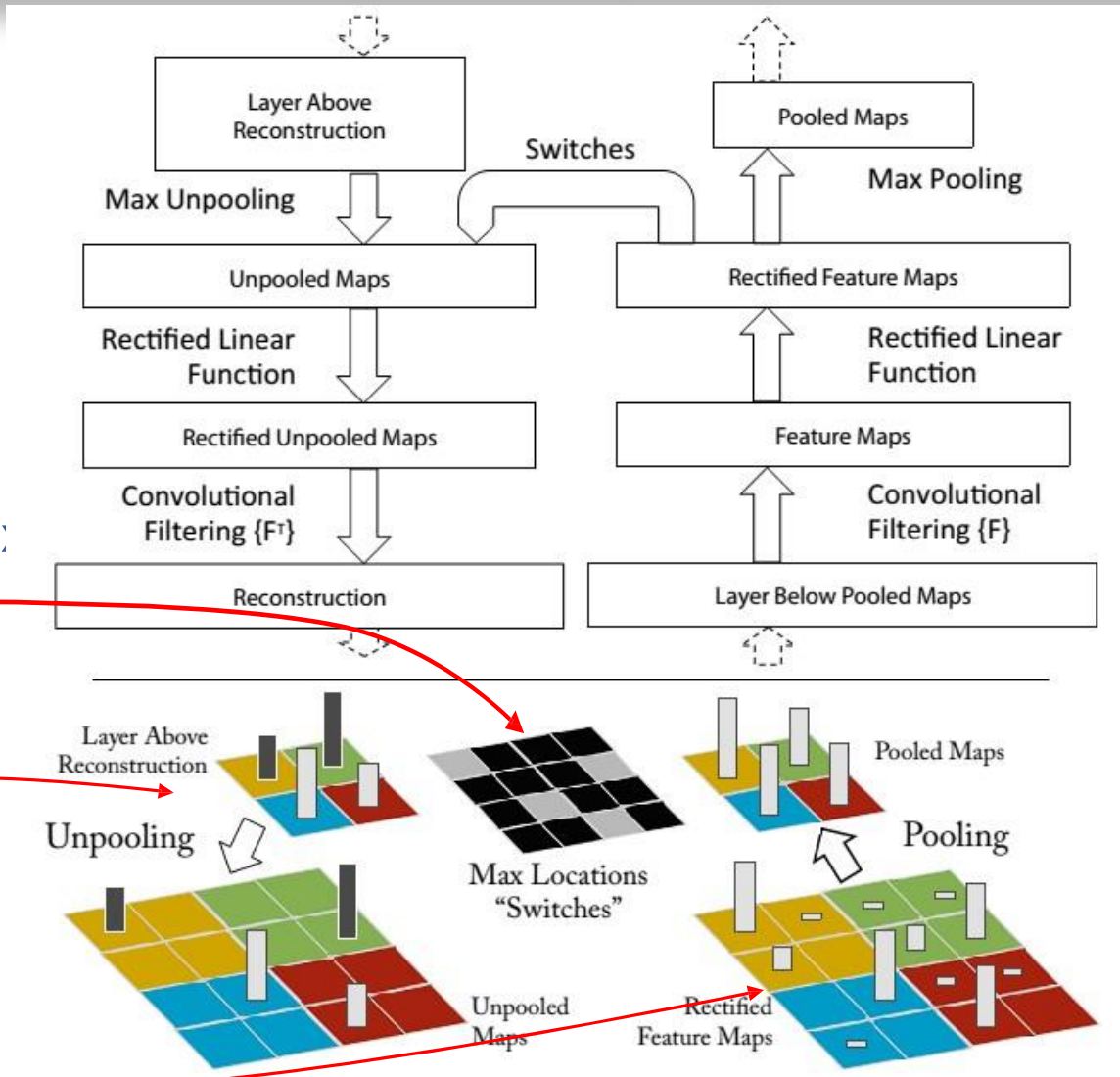
$$\left\lceil \frac{224-7}{2} \right\rceil + 1 = 110 \qquad \left\lceil \frac{110-3}{2} \right\rceil + 1 = 55 \qquad \left\lceil \frac{55-5}{2} \right\rceil + 1 = 26 \qquad \left\lceil \frac{26-3}{2} \right\rceil + 1 = 13$$

Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer fiters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form (6x6x 256 = 9216 dimensions). The final layer is a C-way softmax function, C being the number of classes. All fiters and feature maps are square in shape.

Visualization with a Deconvnet [Zeiler2014]

- Same as convent, but in reverse
  - Instead of mapping pixels to activations, mapping activations to pixels
  - showing what input pattern originally caused a given activation in the feature maps
- Attach a deconvnet layer on every convnet layer
- Unpooling via switches that remember where max pooling was activated
- Deconv filtering equals to "reverse conv filtering"

$$\mathrm{F} = \begin{bmatrix} 1 & 5 \\ 6 & 3 \end{bmatrix} \quad F^T = \begin{bmatrix} 3 & 6 \\ 5 & 1 \end{bmatrix}$$

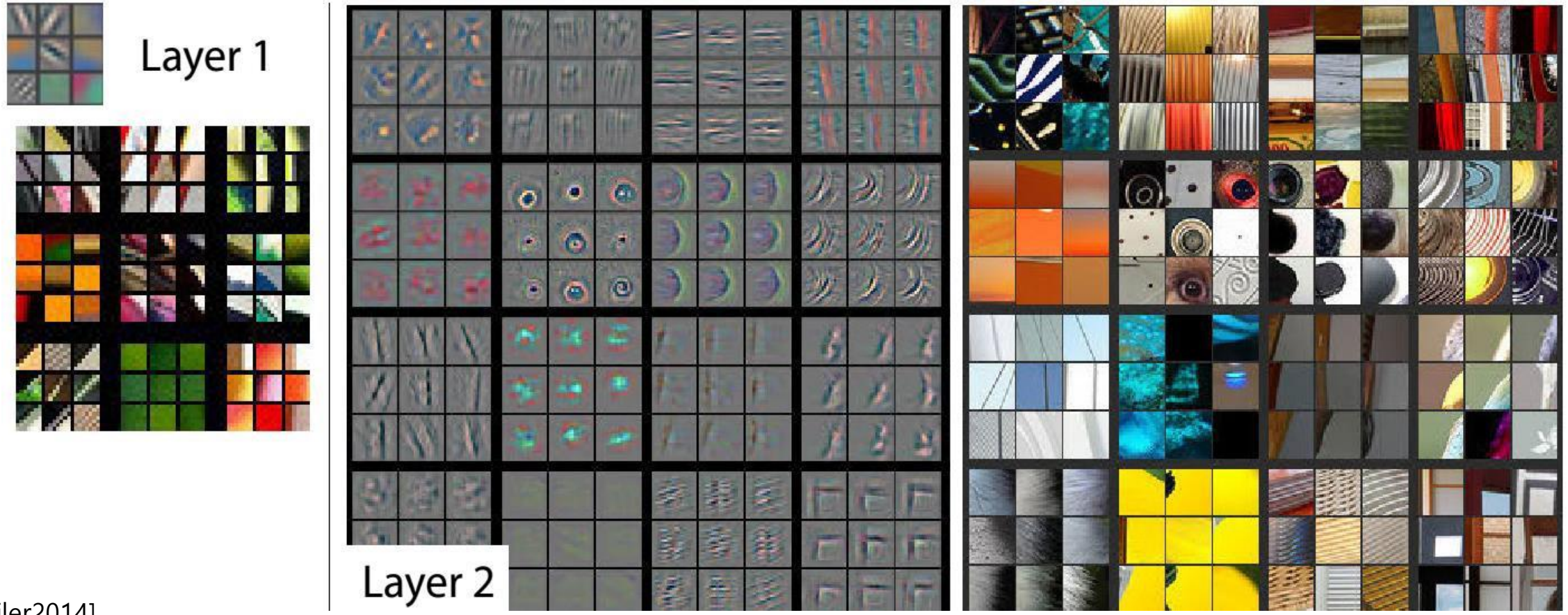


[Zeiler2014]

What excites feature maps?

- "Given a random feature map what are the top 9 activations?"

  To examine a given convnet activation, we set all other activations in the layer to zero and pass the feature maps as input to the attached deconvnet layer.

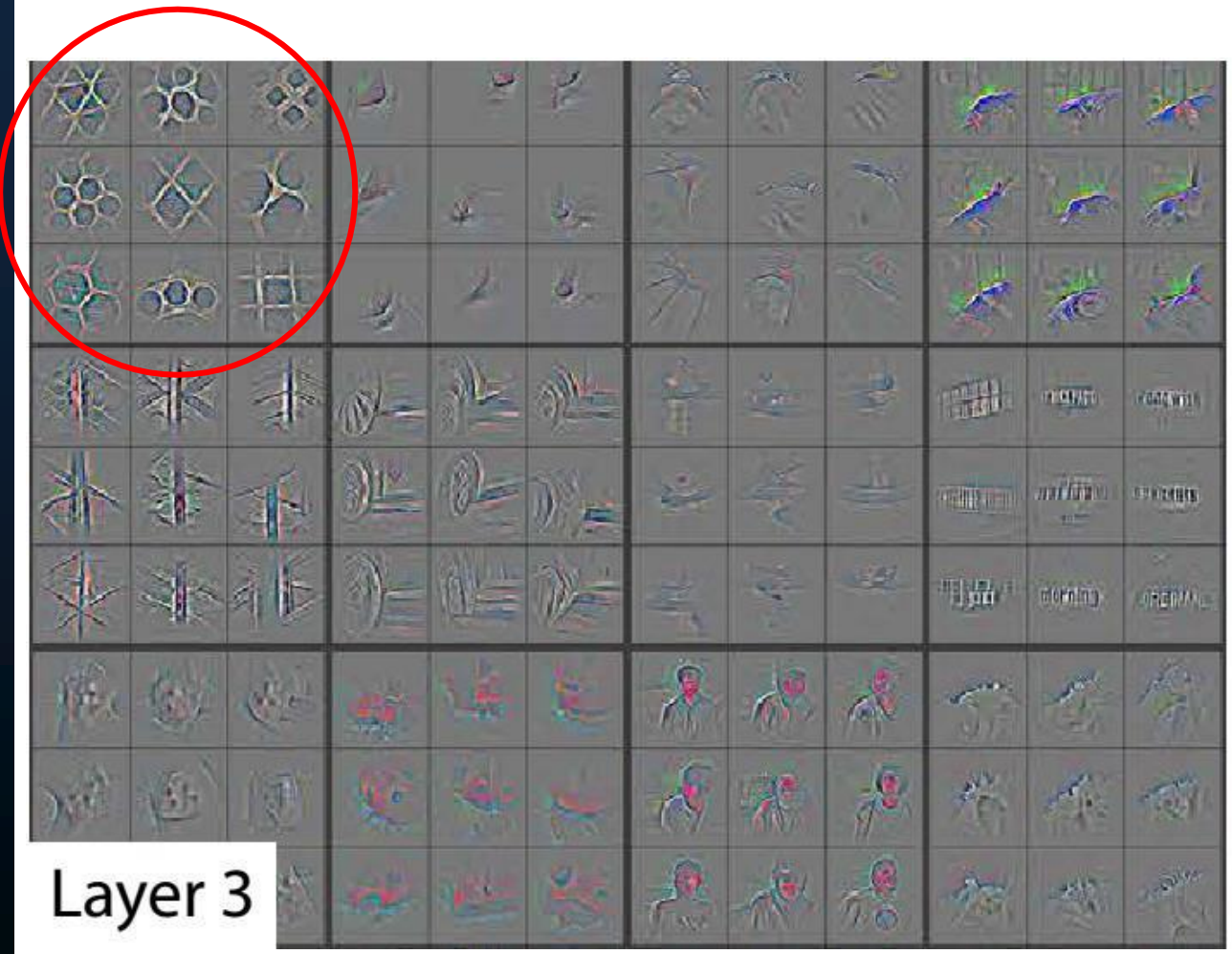


[Zeiler2014]

What excites feature maps?

Similar activations from lower level visual patterns

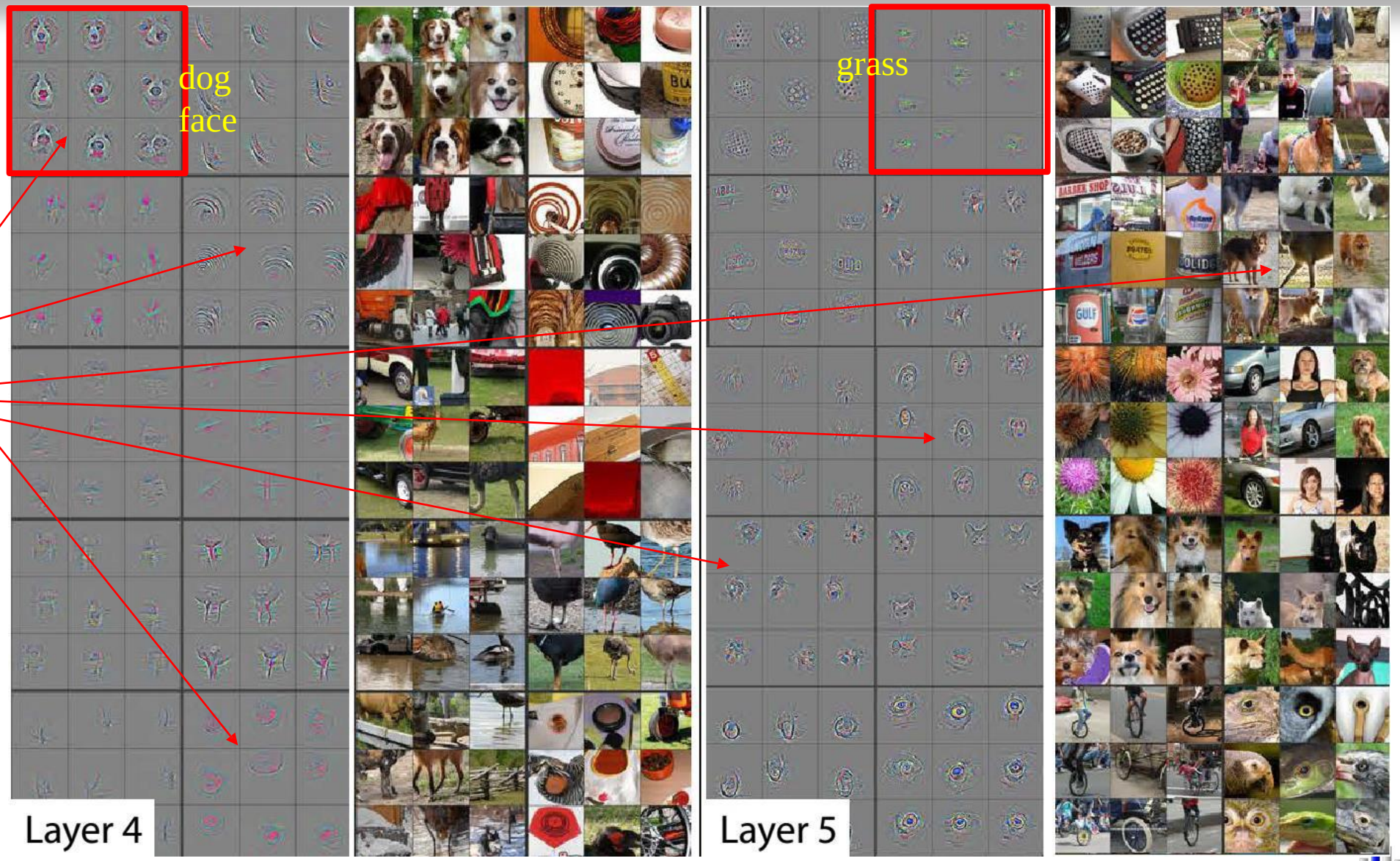
[Zeiler2014]



What excites feature maps? [Zeiler2014]

Similar activations from semantically similar pictures

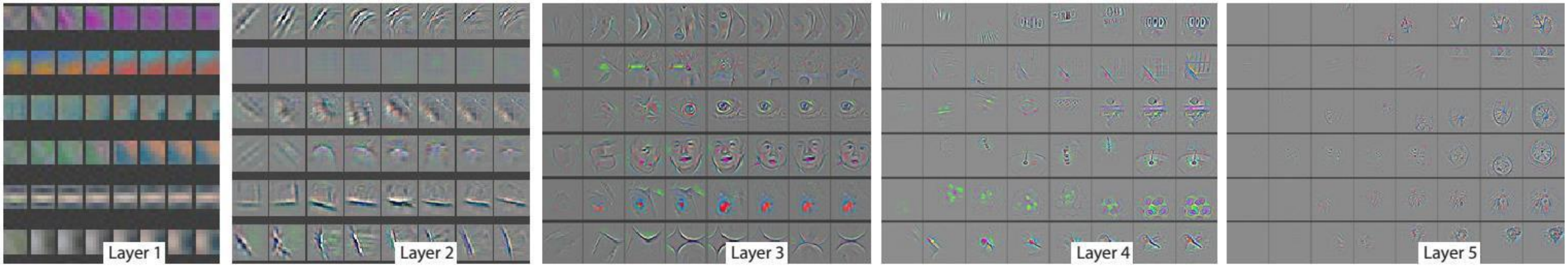
Visual patterns become more and more intricate and specific (greater invariance)



[Zeiler2014]

Feature evolution over training

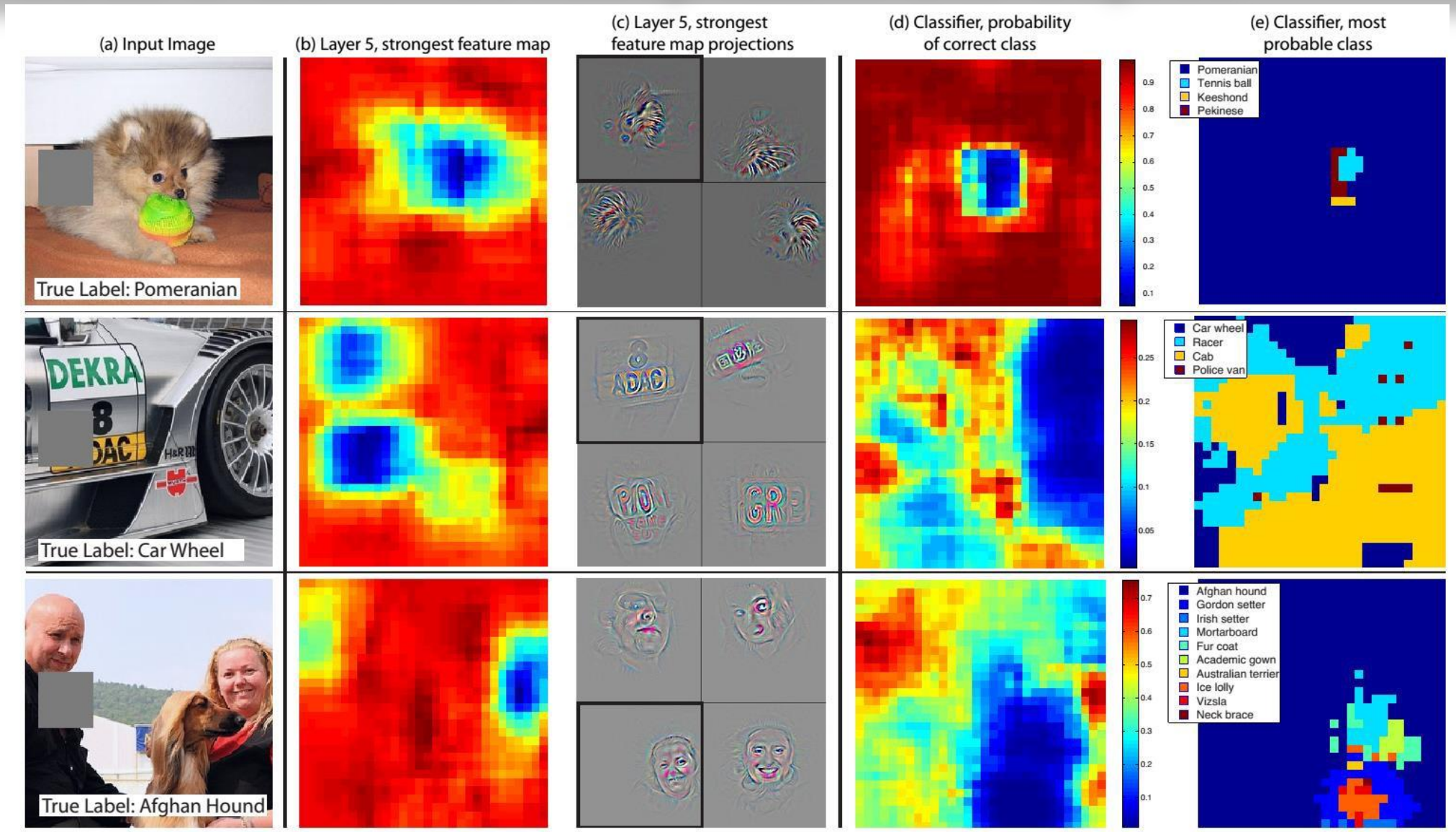
- For a particular neuron (that generates a feature map)
- Pick the strongest activation during training
- For epochs 1, 2, 5, 10, 20, 30, 40, 64



The lower layers of the model can be seen to converge within a few epochs. However, the upper layers only develop develop after a considerable number of epochs (40-50), demonstrating the need to let the models train until fully converged.

[Zeiler2014]

But does a Convnet really learn the object?



[Zeiler2014]

Correspondence Analysis

- Correspondence Analysis
  - implicit mechanism for establishing correspondence between specific object parts in different images (ex) eyes and nose & face
  - $\epsilon_i^l = x_i^l - \tilde{x}_i{}^l$: ($x_i^l$: feature vectors at layer l for $original\ image$, $\tilde{x}_i{}^l$: for occluded image)
  - $\Delta_l = \sum_{i,j=1,i\neq j}^{5} H(\mathrm{sign}(\epsilon^l{}_i), \mathrm{sign}(\epsilon^l{}_j))$ where H is hamming distance → Lower Δ_l means consistency

[Zeiler2014]

Occlusion Location	Mean Feature Sign Change Layer 5	Mean Feature Sign Change Layer 7
Right Eye	0.067 ± 0.007	0.069 ± 0.015
Left Eye	0.069 ± 0.007	0.068 ± 0.013
Nose	0.079 ± 0.017	0.069 ± 0.011
Random	0.107 ± 0.017	0.073 ± 0.014

The lower scores for the eyes and nose (compared to random object parts) show the model implicitly establishing some form of correspondence of parts at layer 5 in the model. At layer 7, the scores are more similar, perhaps due to upper layers trying to discriminate between the dierent breeds of dog.

What is a “Convnet dog”, however? [Simonyan2014]

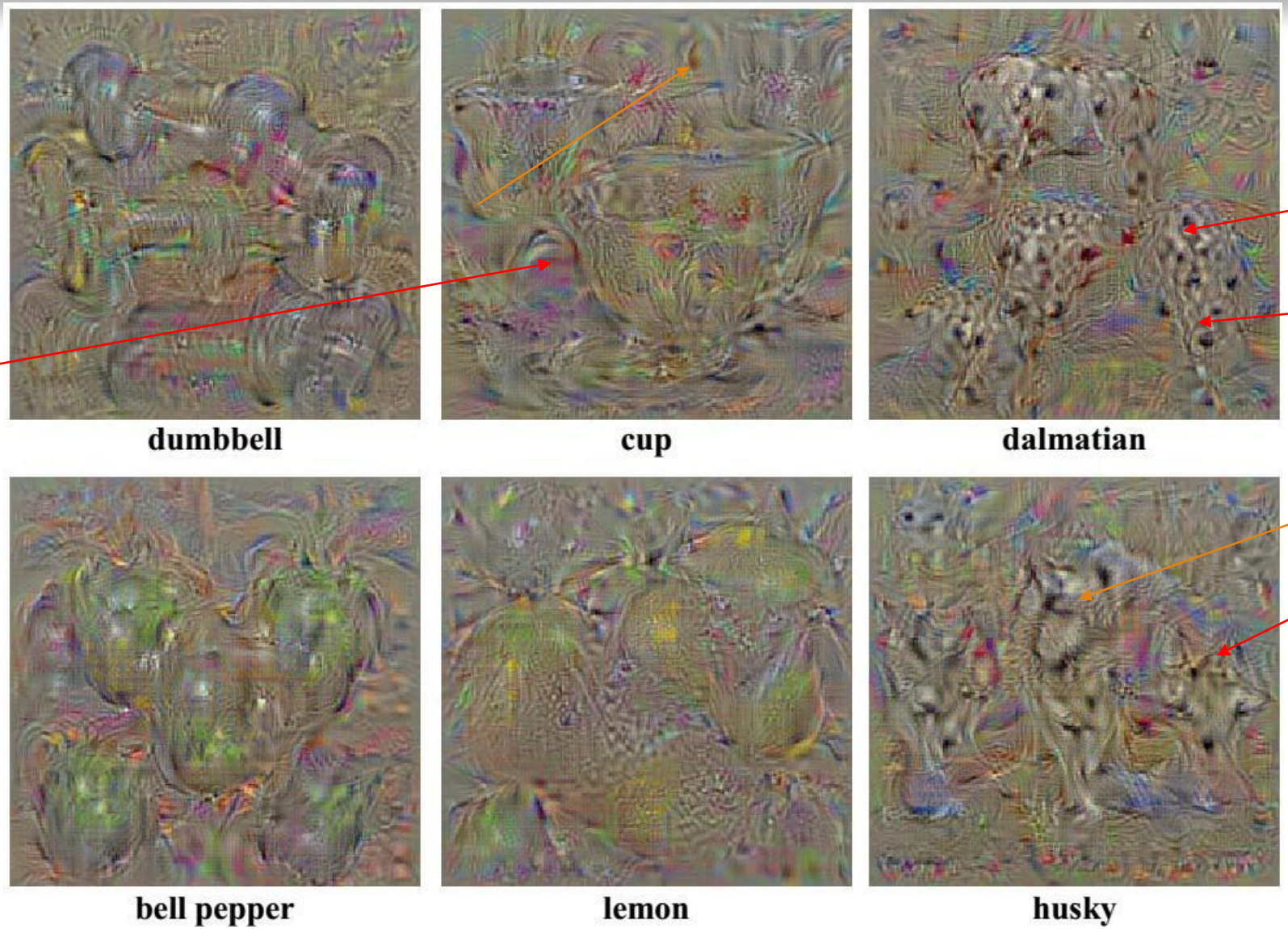
- What is the image with the highest “dogness score”

$$\text{argmax}_I \, S_c(I;\theta) - \lambda |I|^2$$

- The parameters θ are fixed during the training
- Optimization is done with respect to the image I
- Initialized with the “average training image”

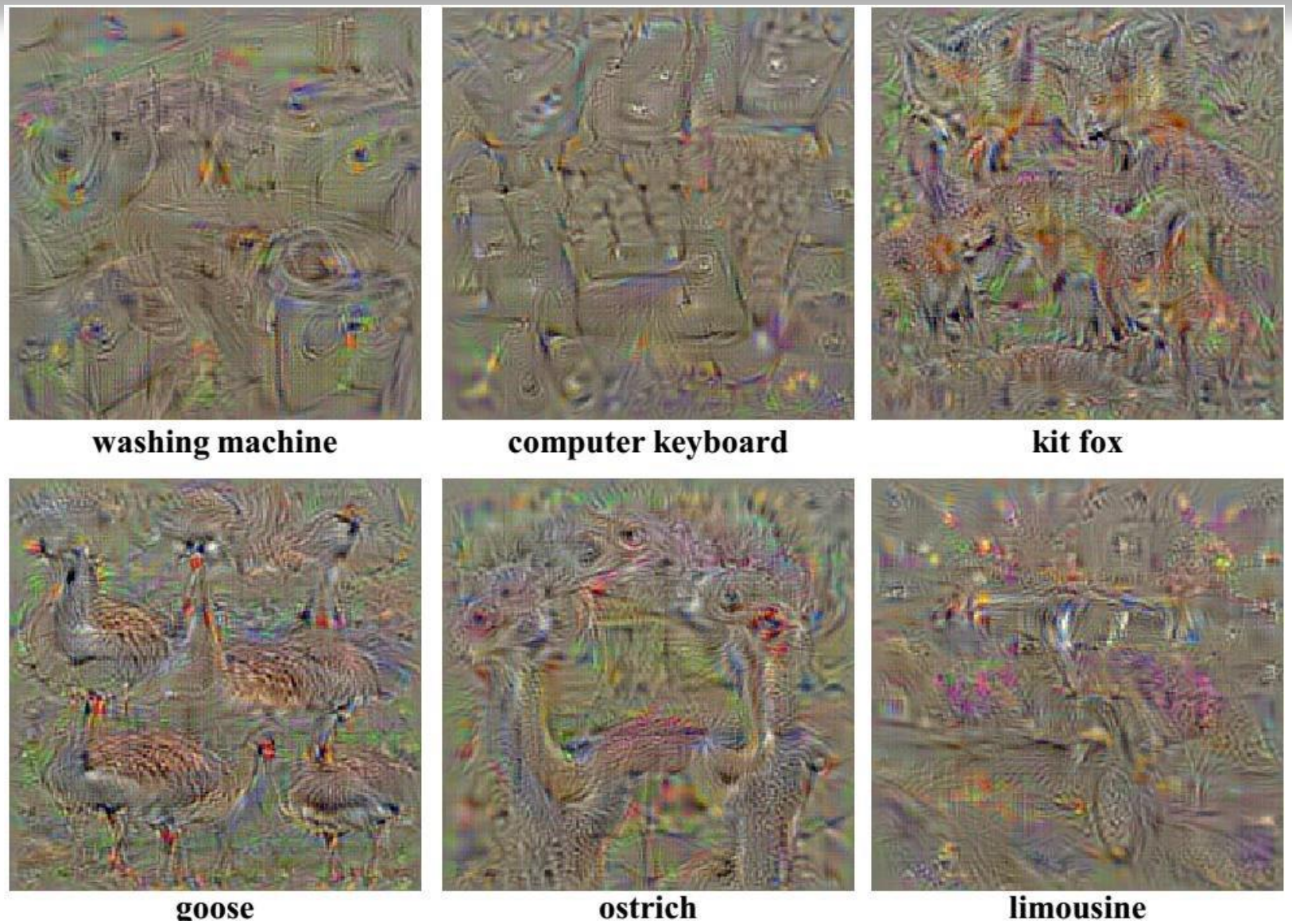
[Simonyan2014]

Maximum-scoring class image



[Simonyan2014]

Maximum-scoring class image



[Simonyan2014]

Class-specific image saliency

- Given the "monkey" class, what are the most "monkey-ish" parts in my image?
  - rank the pixels of I_0 based on their influence on the score $S_c\,(I_0)$.
- Approximate S_c around an initial point I_0 with the first order Taylor expansion
  - $S_c(I)|_{I_0} \approx w^T I + b, where\ w = \frac{\partial S_c}{\partial I}|_{I_0} from\ backpropagation$
  - the magnitude of elements of w defines the importance of the corresponding pixels of I for the class c
- Solution is locally optimal
- Weakly Supervised Object Localisation

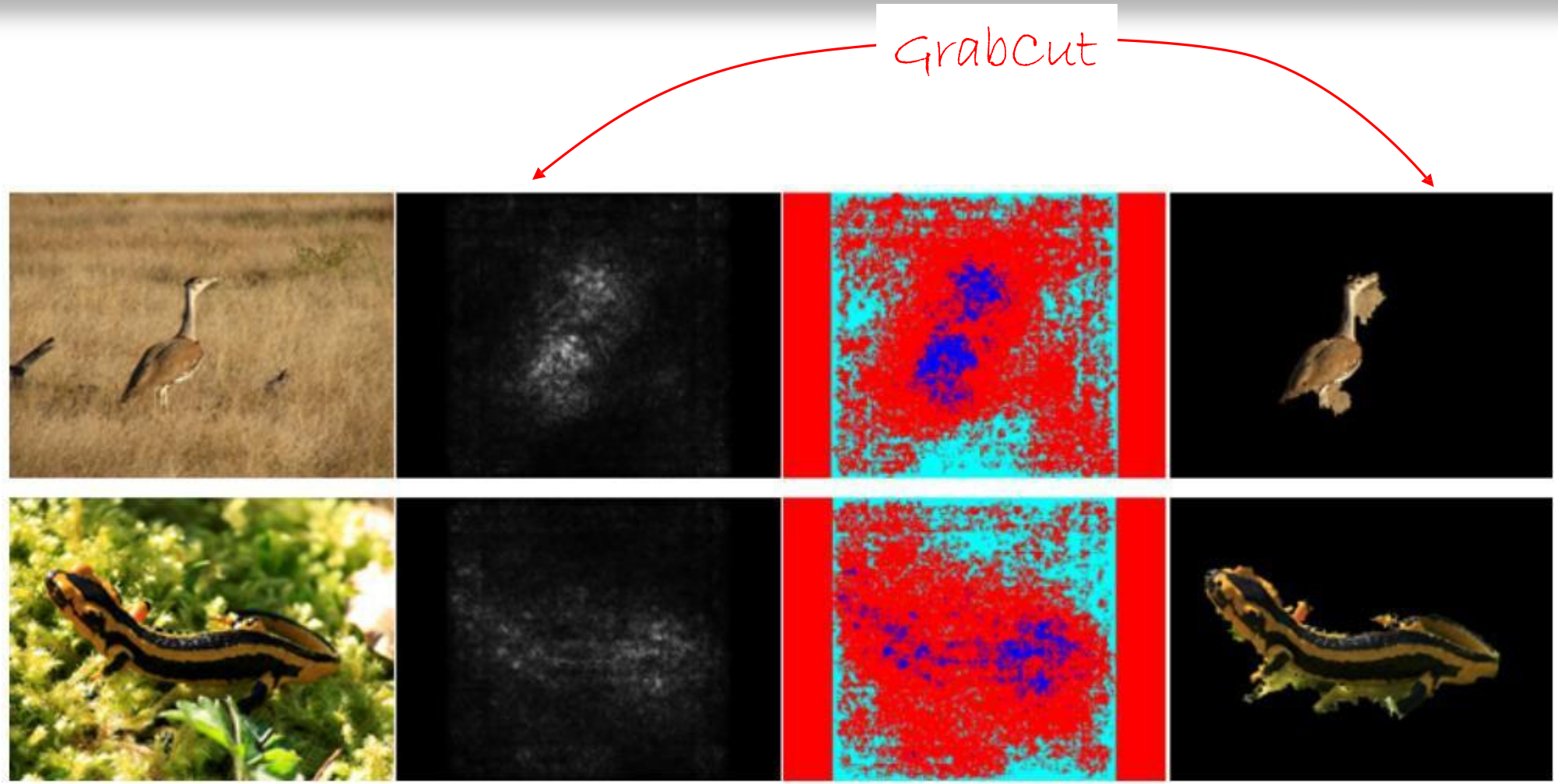
Examples: Class Saliency Extraction

[Simonyan2014]

Examples

[Simonyan2014]

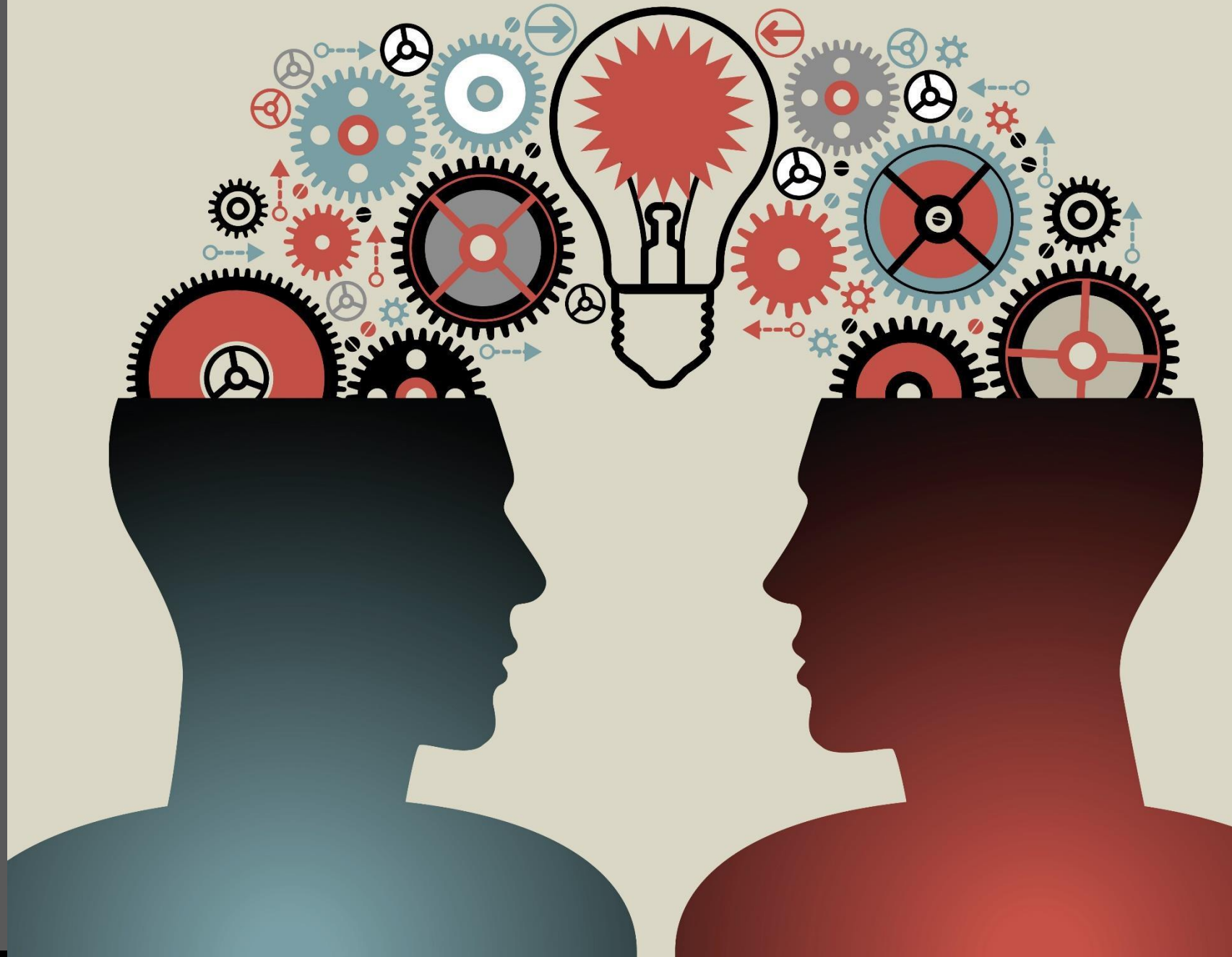
Object localization using class saliency



encode the location of the object of the given class in the given image, used for object localisation

Source: http://uvadlc.github.io/

Knowledge transfer



Transfer learning

- Assume two datasets, T and S
- Dataset S is
  - fully annotated, plenty of images
  - We can build a model h_s

Imagenet: 1million

h_A

- Dataset T is
  - Not as much annotated, or much fewer images
  - The annotations of T do not need to overlap with S
- We can use the model h_s to learn a better h_T
- This is called transfer learning
  - Convnets are good in transfer learning

My dataset: 1,000

h_B

Source: http://uvadlc.github.io/

SLCF What You Imagine is What You Get

Solution I: Fine-tune using h_T as h_s initialization

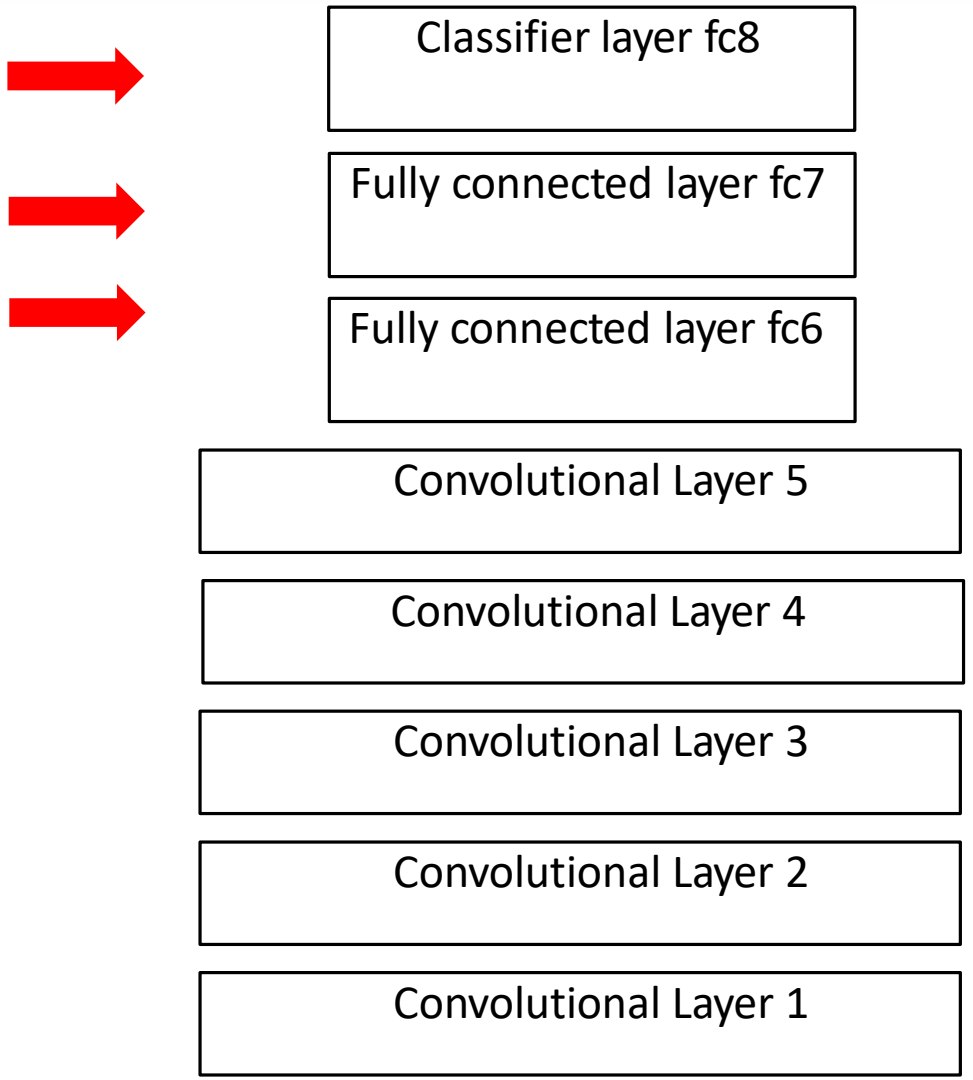
- Assume the parameters of S are already a good start near our final local optimum
- Use them as the initial parameters for our new CNN for the target dataset

$$\theta_{T,l}^{(t=0)} = \theta_{S,l}^{t=\infty} \; for \; some \; layer \; l = 1,2 \ldots$$

- This is a good solution when the dataset T is relatively big
  - E.g. for Imagenet S with approximately 1 million images
  - For a dataset T with more than a few thousand images should be ok
- What layers to initialize and how?

Initializing h_T with h_s

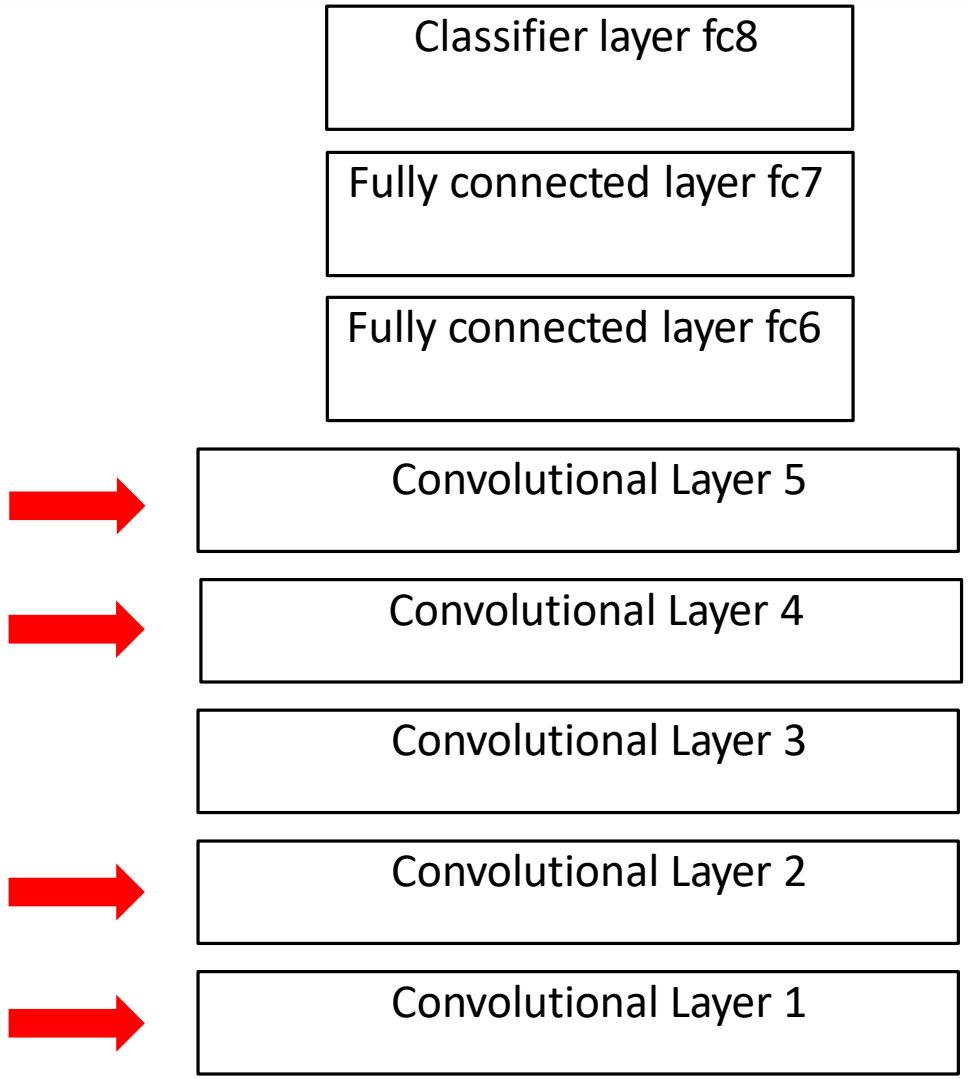
- Classifier layer to loss
  - The loss layer essentially is the "classifier"
  - Same labels -> keep the weights from h_s
  - Different labels -> delete the layer and start over
  - When too few data, fine - tune only this layer
- Fully connected layers
  - Very important for fine-tuning
  - Sometimes you need to completely delete the last before the classification layer if datasets are very different
  - Capture more semantic, "specific" information
  - Always try first when fine-tuning
  - If you have more data, fine-tune also these layers



Source: http://uvadlc.github.io/

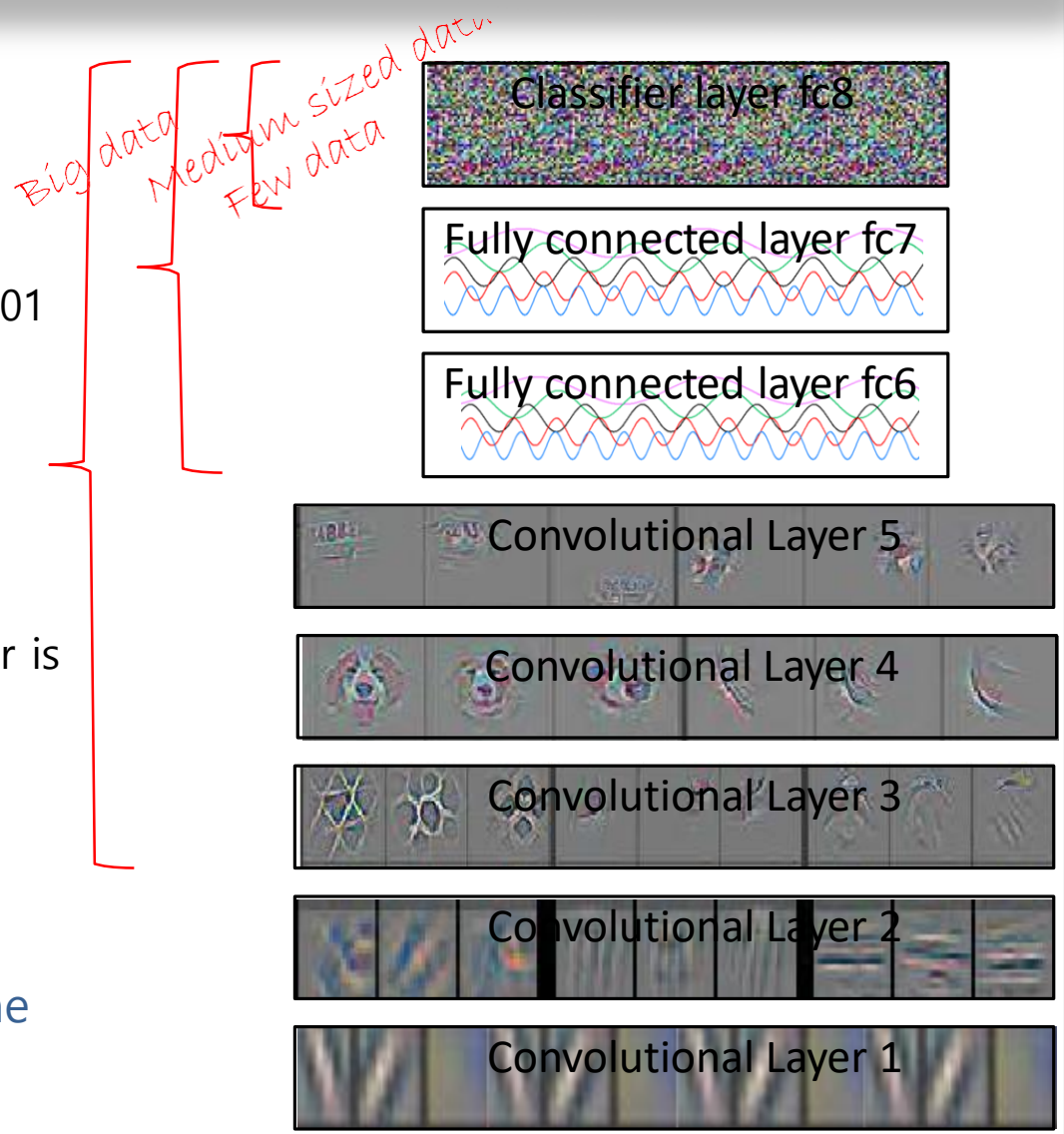
Initializing h_T with h_s

- Upper convolutional layers (conv4, conv5)
  - Mid-level spatial features (face, wheel detectors …)
  - Can be different from dataset to dataset
  - Capture more generic information
  - Fine-tuning pays off
  - Fine-tune if dataset is big enough
- Lower convolutional layers (conv1,conv2)
  - They capture low level information
  - This information does not change usually
  - Probably, no need to fine-tune but no harm trying



How to fine-tune?

- For layers initialized from h_s use a mild learning rate
  - Remember: your network is already close to a near optimum
  - If too aggressive, learning might diverge
  - A learning rate of 0.001 is a good starting choice (assuming 0.01 was the original learning rate)
- For completely new layers (e.g. loss) use aggressive learning rate
  - If too small, the training will converge very slowly
  - Remember: the rest of the network is near a solution, this layer is very far from one
  - A learning rate of 0.01 is a good starting choice
- If datasets are very similar, fine-tune only fully connected layers
- If datasets are different and you have enough data, fine-tune all layers



Source: http://uvadlc.github.io/

Solution II: Use h_S as a feature extractor for h_T

- Essentially similar to a case of solution I
  - but train only the loss layer
- Essentially use the network as a pretrained feature extractor
- This is a good solution if the dataset T is small
  - Any fine-tuning of layer might cause overfitting
  - Or when we don't have the resources to train a deep net
  - Or when we don't care for the best possible accuracy

Which layer?

Table 6. Analysis of the discriminative information contained in each layer of feature maps within our ImageNet-pretrained convnet. We train either a linear SVM or softmax on features from different layers (as indicated in brackets) from the convnet. Higher layers generally produce more discriminative features.

	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

Lower layer features capture more basic information (texture, etc). Good for image-to-image comparisons, image retrieval

Higher layer features are capture more semantic information. Good for higher level classification

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

CNN Case Study 2: VGGNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

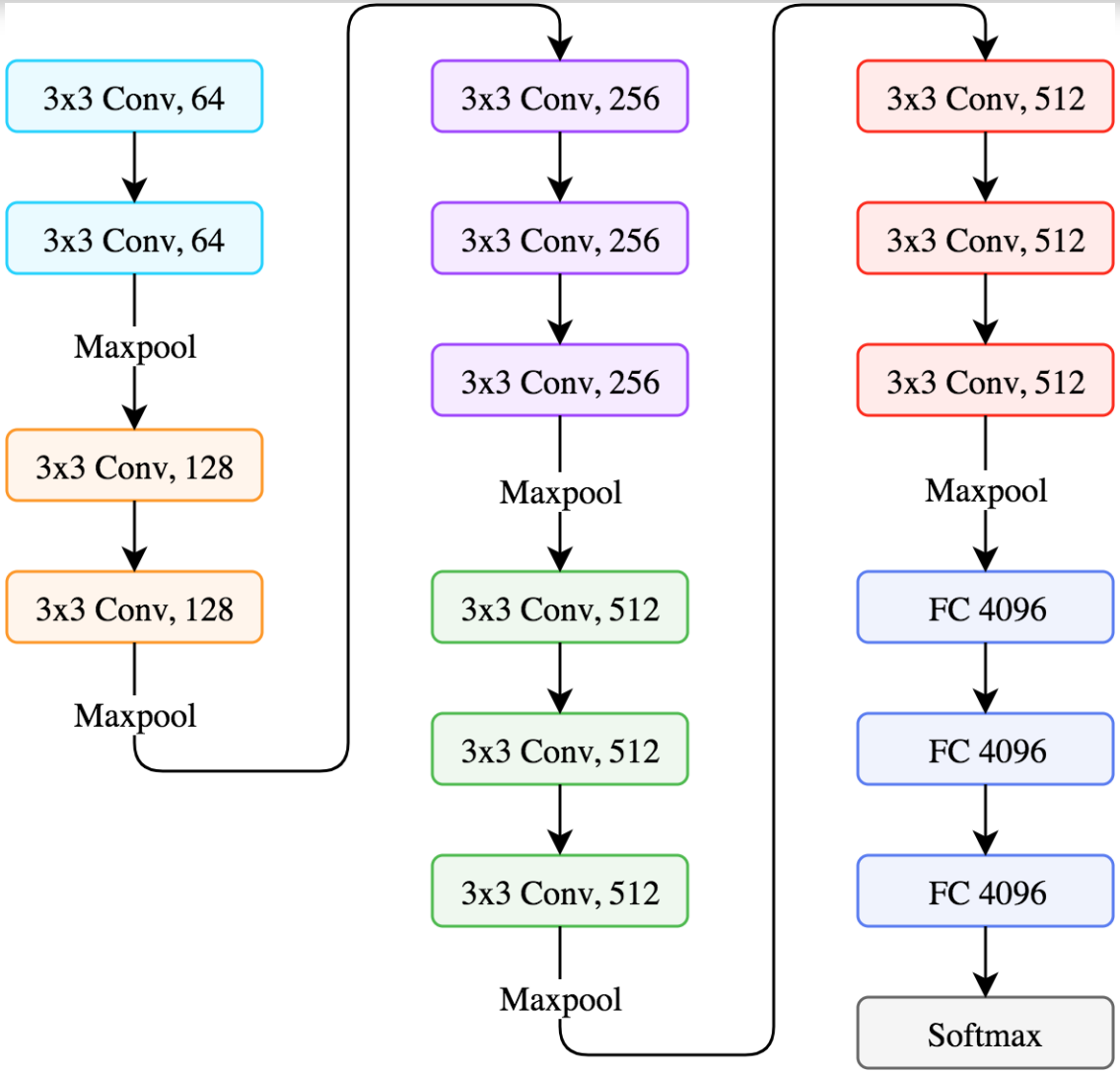
Source: http://uvadlc.github.io/

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG16: Differences from Alexnet

- Much more accurate
  - 7.3% vs 18.2% top-5 error
- About twice as many layers
  - 16 vs 7 layers
- Filters are much smaller
  - 3x3 vs 7x7 filters
- Harder/slower to train



Source: http://uvadlc.github.io/

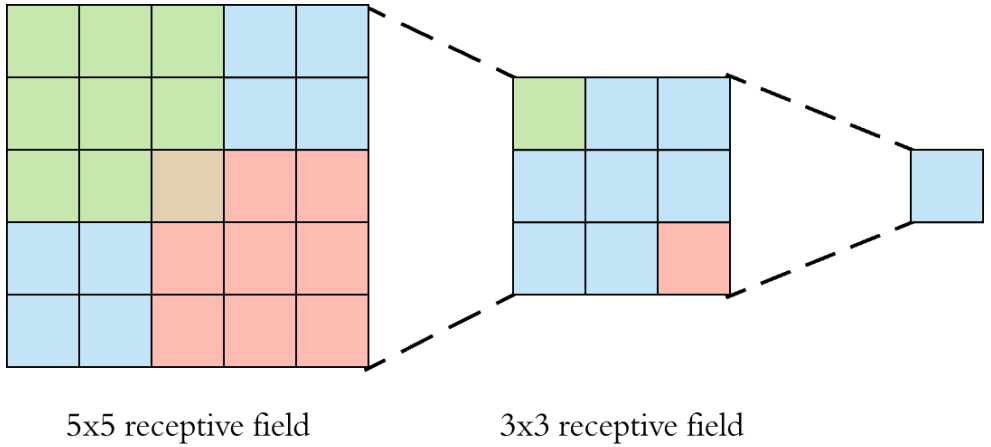
Characteristics

- Input size: 224 × 224
- Filter sizes: 3 × 3
- Convolution stride: 1
  - Spatial resolution preserved
- Padding: 1
- Max pooling: 2 × 2 with a stride of 2
- ReLU activations
- No fancy input normalizations
  - No Local Response Normalizations
- Although deeper, number of weights is not exploding

Why 3 × 3 filters?

- The smallest possible filter to captures the “up”, “down”, “left”, “right”
- Two 3 × 3 filters have the receptive field of one 5 × 5
- Three stacked 3 × 3 filters have the receptive field of one 7 × 7
  - 1 large filter can be replaced by

    a deeper stack of successive smaller filters



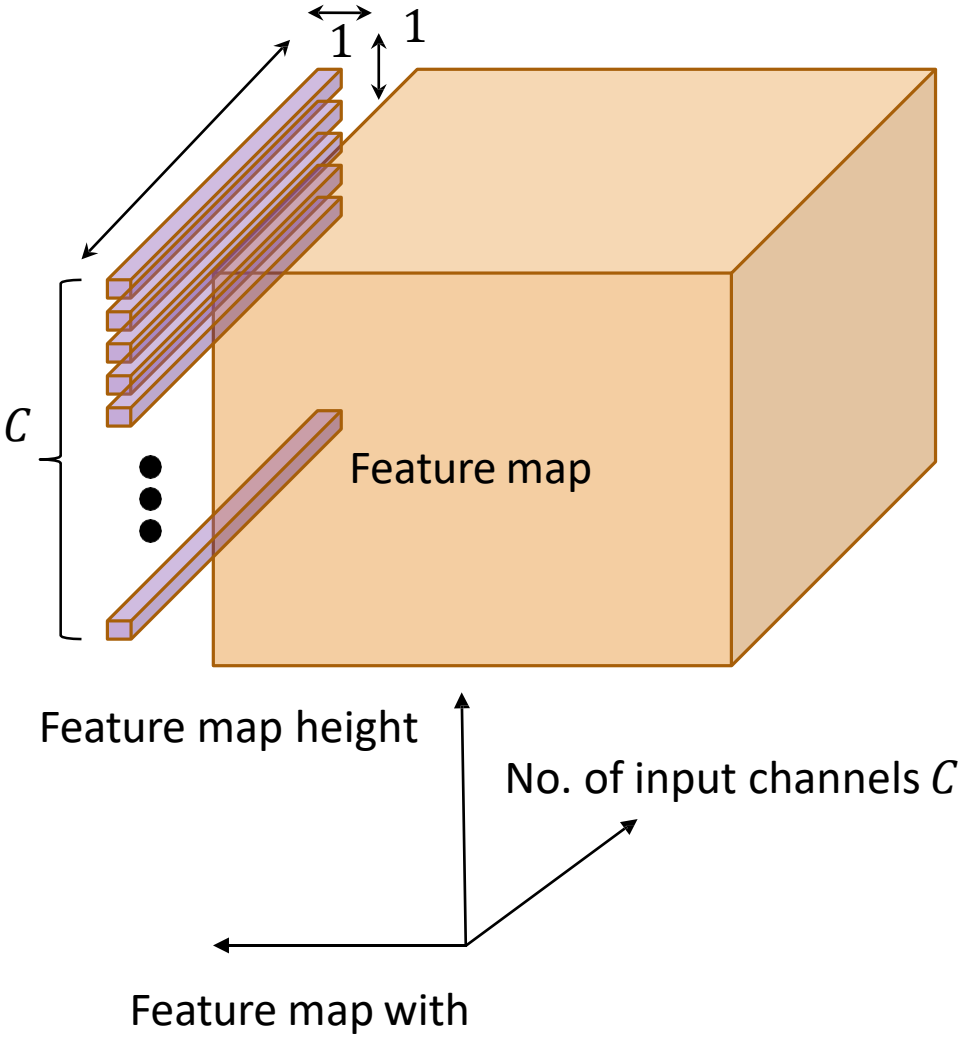
- Benefit?
  - Three more nonlinearities for the same “size” of pattern learning
  - Also fewer parameters and regularization
  - $(3 \times 3 \times C) \times 3 = 27 \cdot C$, $7 \times 7 \times C \times 1 = 49 \cdot C$

Picture credit: Arden Dertat

- Conclusion: 1 large filter can be replaced by a deeper, potentially more powerful, stack of successive smaller filters

Even smaller filters?

- Also $1x1$ filters are used
- Followed by a nonlinearity
- Why?
- Increasing nonlinearities without affecting receptive field sizes
  - Linear transformation of the input channels



CNN Case Study 3: Inception

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

Source: http://uvadlc.github.io/

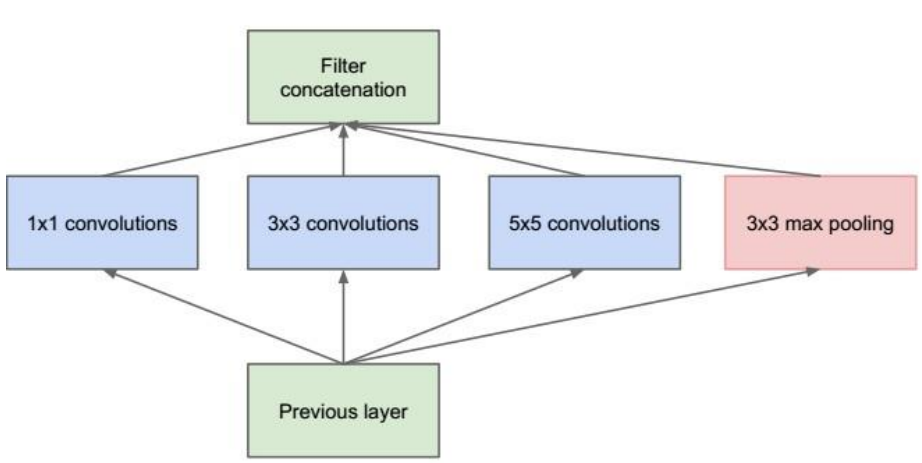
Basic idea

- Salient parts have great variation in sizes
- Hence, the receptive fields should vary in size accordingly
- Naively stacking convolutional operations is expensive
- Very deep nets are prone to overfitting

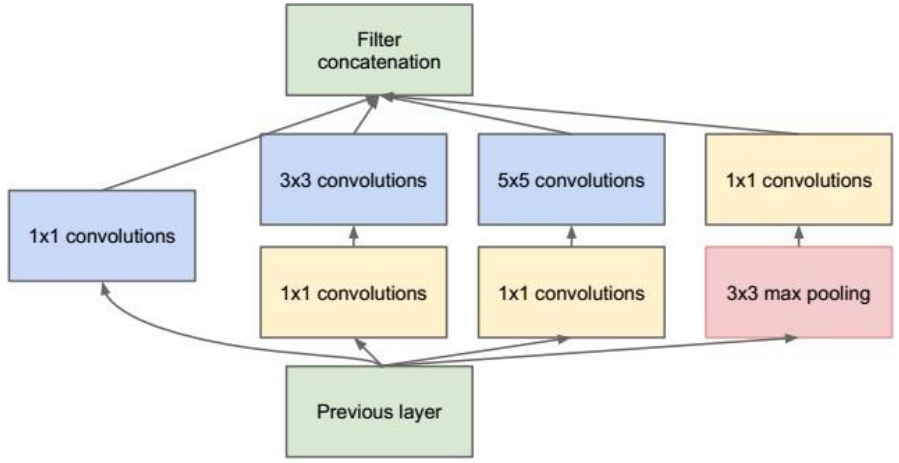
Picture credit: Bharath Raj

Inception module

- Multiple kernel filters of different sizes (1 × 1, 3 × 3, 5 × 5)
  - Naïve version
- Problem?
  - Very expensive!
- Add intermediate 1 × 1 convolutions



(a) Inception module, naïve version



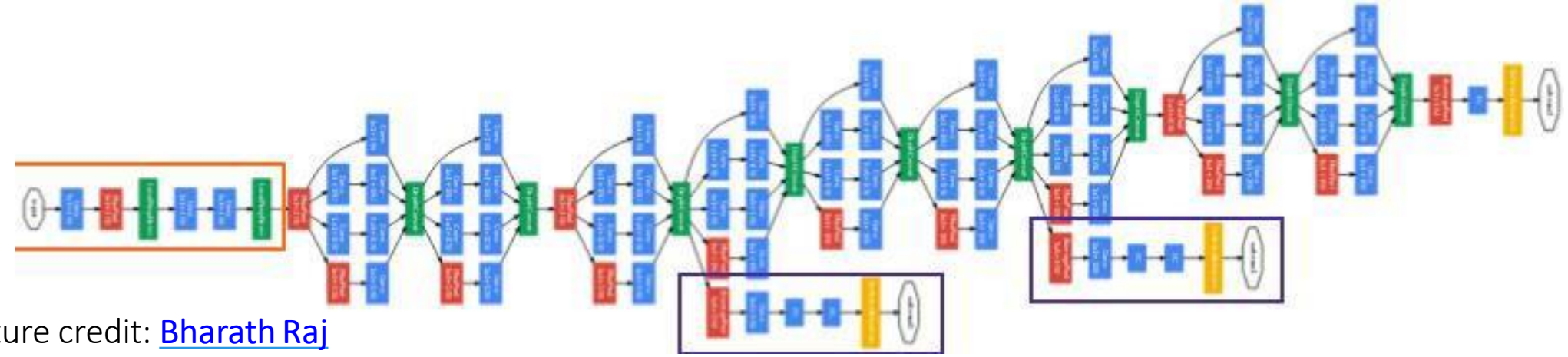
(b) Inception module with dimension reductions

Picture credit: Bharath Raj

Source: http://uvadlc.github.io/

Architecture

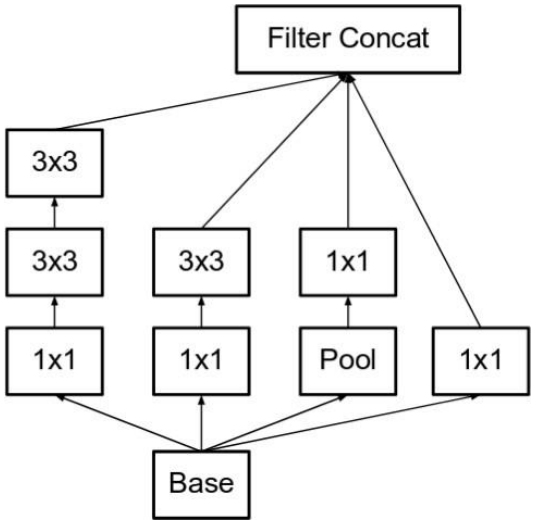
- 9 Inception Modules
- 22 layers deep (27 with the pooling layers)
- Global average pooling at the end of last Inception Module
- 6.67% Imagenet error, compared to 18.2% of Alexnet
- Because of the increased depth → Vanishing gradients
- Inception solution to vanishing gradients: intermediate classifiers
- Intermediate classifiers removed after training

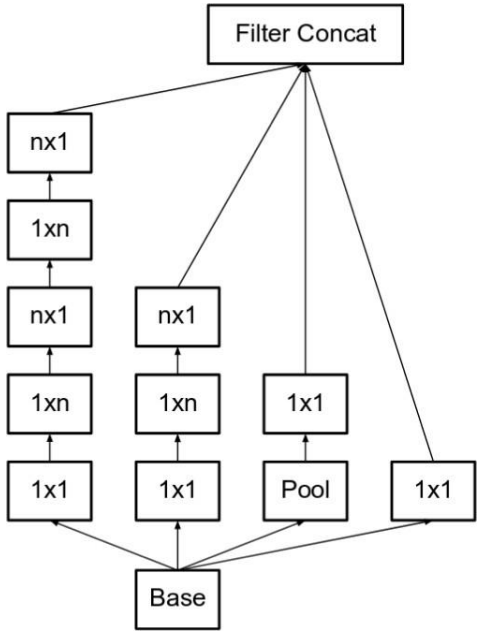


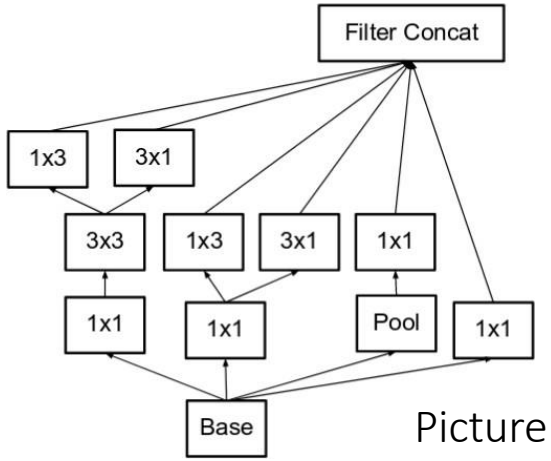
Picture credit: Bharath Raj

Inceptions v2, v3, v4, ….

- Factorize 5×5 in two 3×3 filters
- Factorize $n \times n$ in two $n \times 1$ and $1 \times n$ filters (quite a lot cheaper)
- Make nets wider
- RMSprop, BatchNorms, …

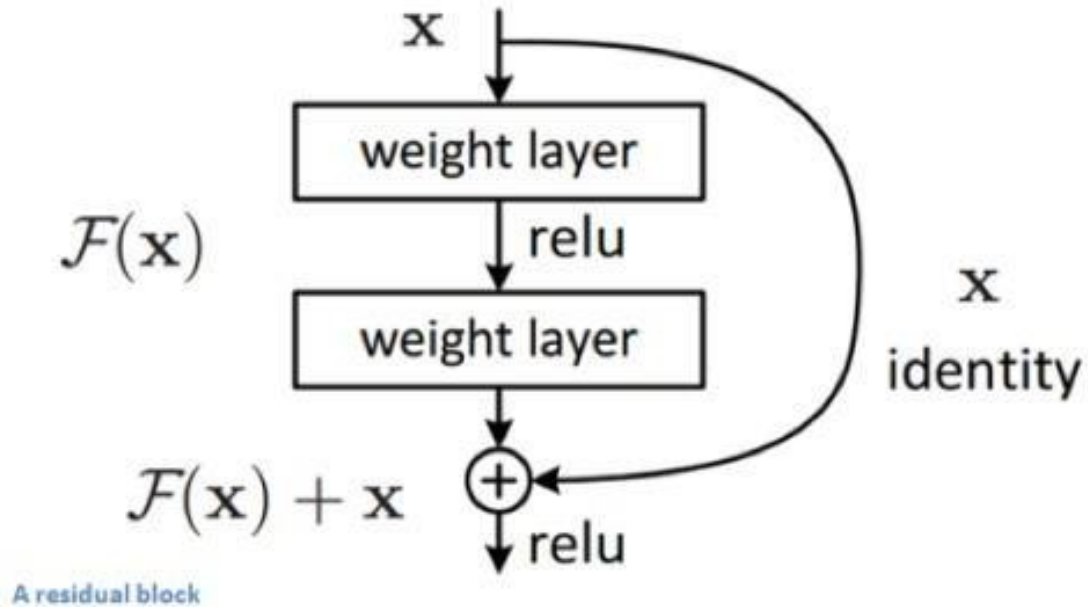






Picture credit: Bharath Raj

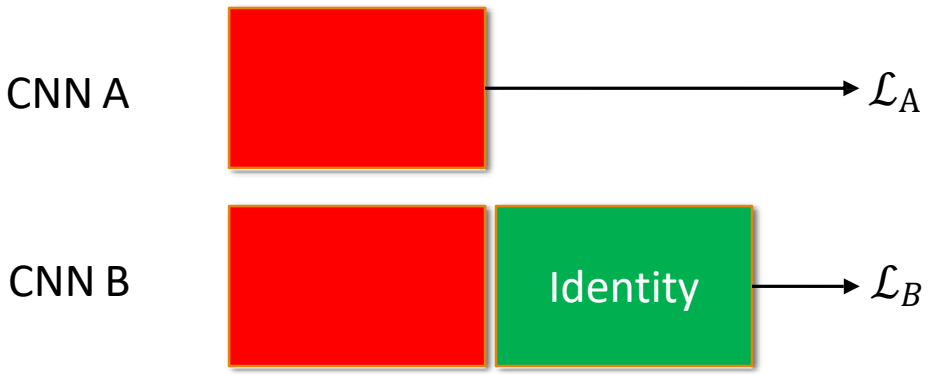
CNN Case Study 4: ResNets DenseNets HighwayNets



Source: http://uvadlc.github.io/

Hypothesis

- Hypothesis: Is it possible to have a very deep network at least as accurate as averagely deep networks?
- Thought experiment: Let's assume two Convnets A, B. They are almost identical, in that B is the same as A, with extra "identity" layers. Since identity layers pass the information unchanged, the errors of the two networks should be similar. Thus, there is a Convnet B, which is at least as good as Convnet A w.r.t. training error



Hypothesis

- We add a few extra layers on top of the N layers
- These few extra layers are identify mappings

$$y = x$$

- Quiz: What looks weird?

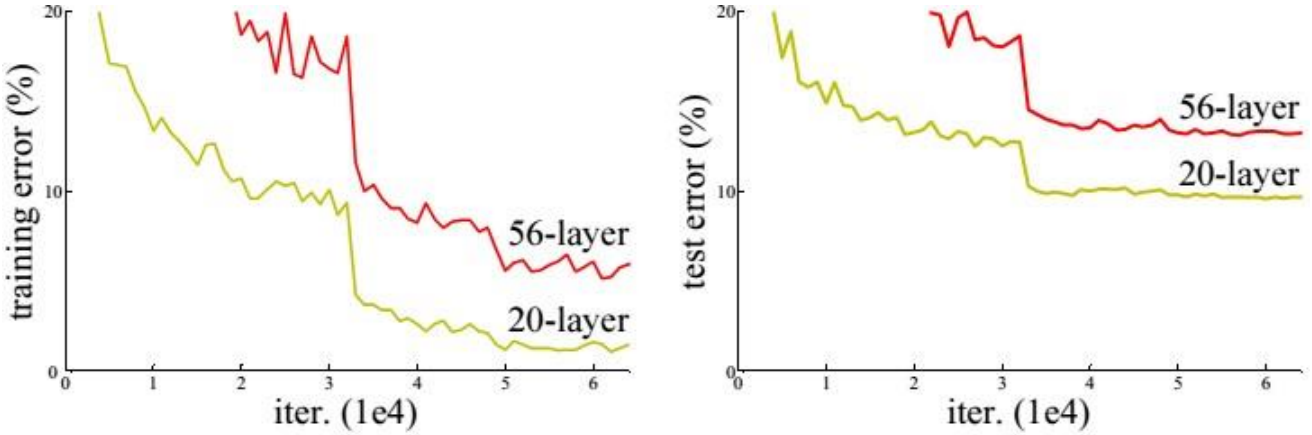
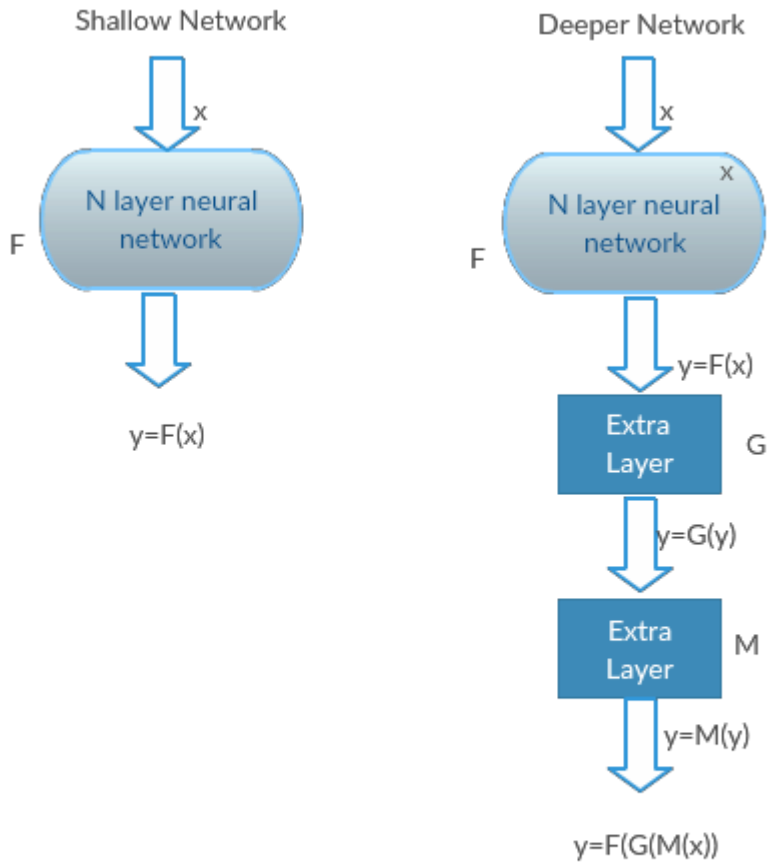


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

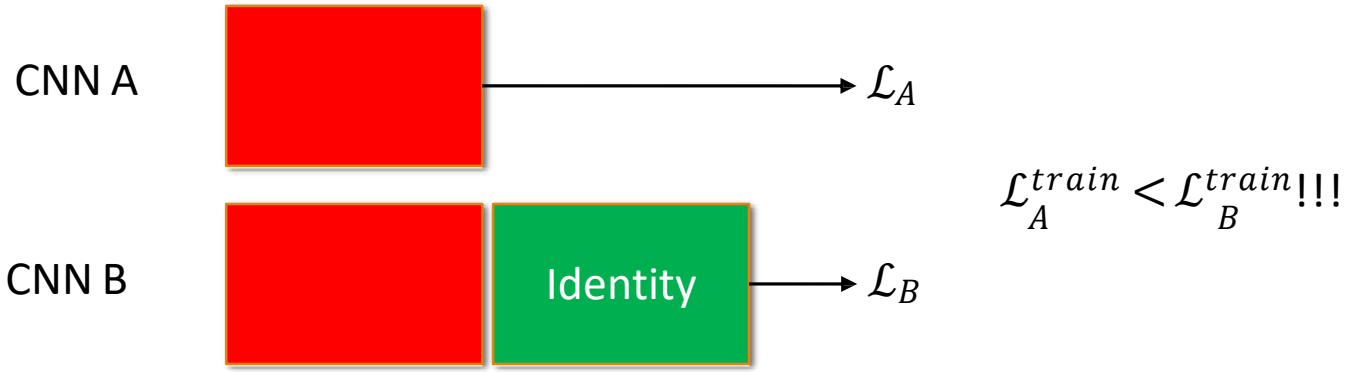


G and M act as Identity Functions. Both the Networks Give same output

Source: http://uvadlc.github.io/

Testing the hypothesis

- Adding identity layers increases training error!!
  - Training error, not testing error
- Performance degradation not caused by overfitting
  - Just the optimization task is harder
- Assuming optimizers are doing their job fine, it appears that not all networks are the same as easy to optimize



The “residual idea”, intuitively

- Observation
  - Very deep networks stop learning after a bit
    - An accuracy is reached, then the network saturates and starts unlearning
  - Signal gets lost through so many layers
  - Models start failing
  - Let’s say we have the neural network nonlinearity a = $F(x)$
  - Perhaps easier to learn a function a = $F(x)$ to model differences a~δy than to model absolutes a~y
    - Otherwise, you may need to model both the magnitude as well as the direction of activations
    - Think of it like in input normalization → you normalize around 0
    - Think of it like in regression → you model differences around the mean value
  - “Residual idea”: Let neural networks explicitly model difference mappings

$$F(x)=H(x)-x \Rightarrow H(x)=F(x)+x$$

    - $F(x)$ are the stacked nonlinearities
    - x is the input to the nonlinear layer

ResNet block

- $H(x) = F(x)+x$
- If dimensions don't match
  - Either zero padding
  - Or a projection layer to match dimensions

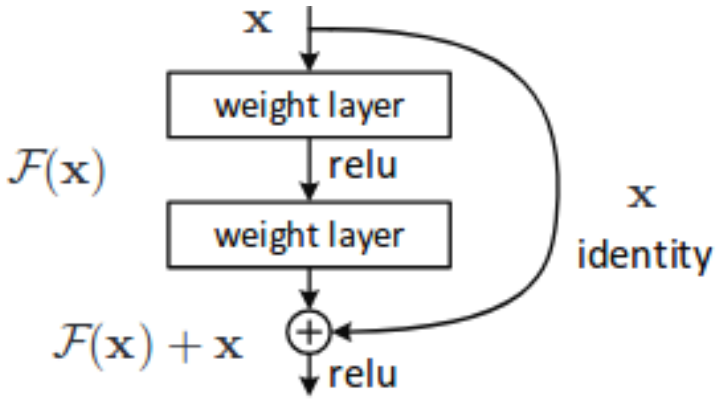
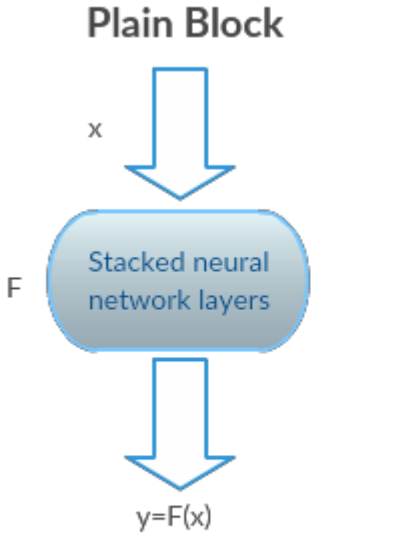
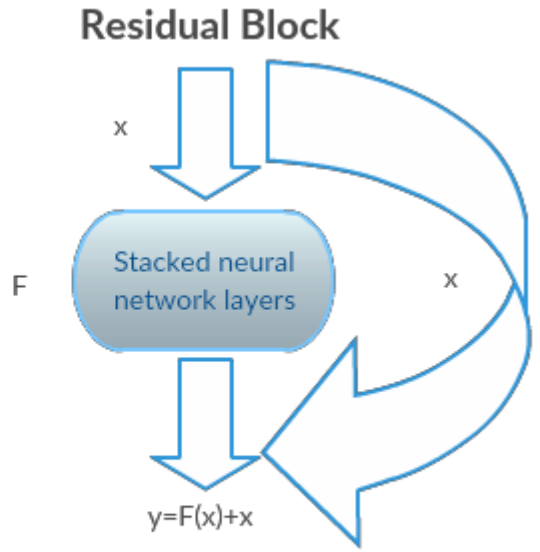


Figure 2. Residual learning: a building block.



Hard to get F(x)=x and make y=x an identity mapping



Easy to get F(x)=0 and make y=x an identity mapping

No degradation anymore

- Without residual connections deeper networks are untrainable
- Up to 1000 layers ResNets trained

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

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

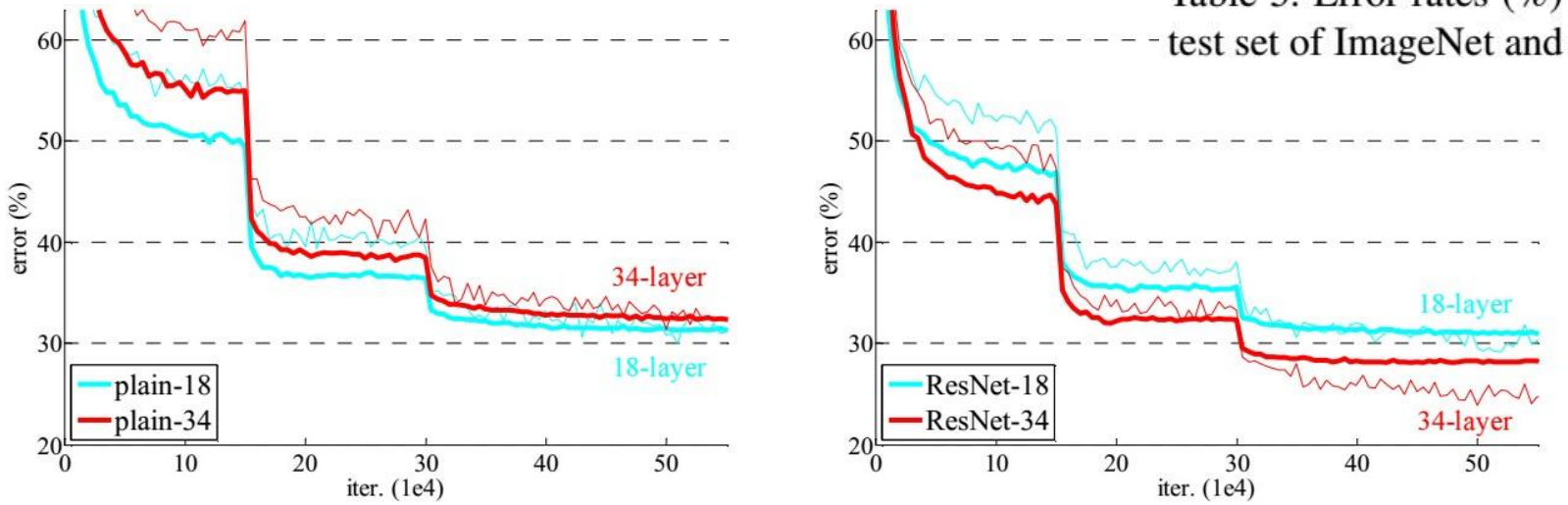


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

HighwayNet

- Similar to ResNets, only introducing a gate with learnable parameters on the importance of each skip connection

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

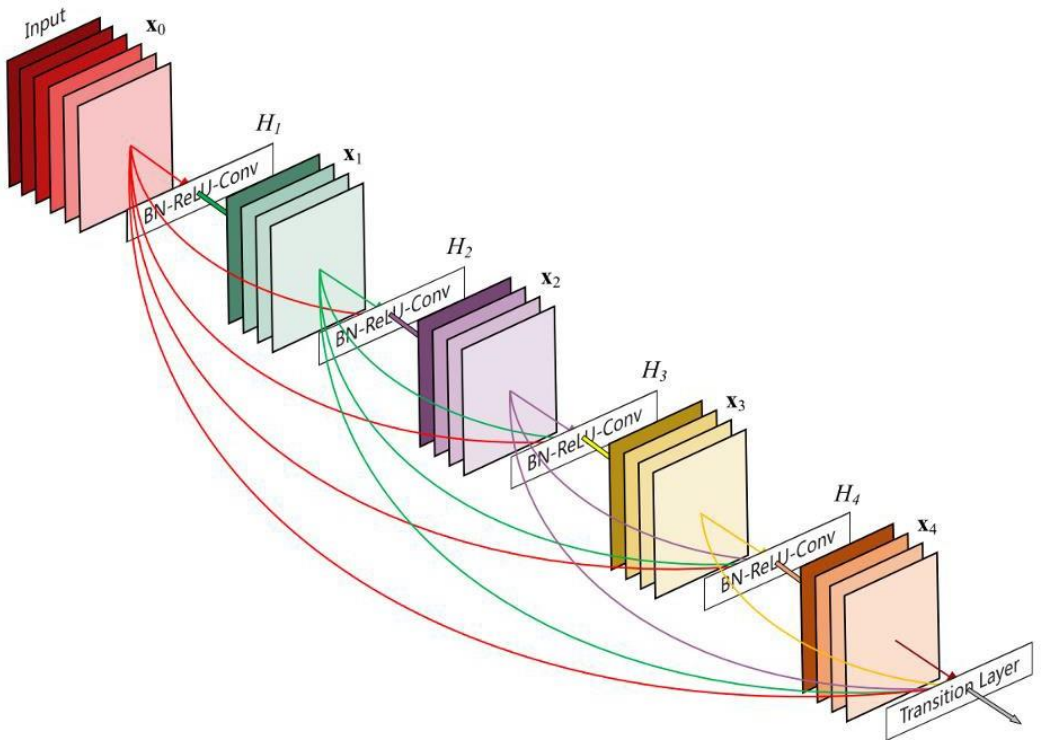
- Similar to … LSTMs as we will say later

DenseNet

- Add skip connections to multiple forward layers

$$y = \mathrm{h}(x_l, x_{l-1}, \ldots, x_{l-n})$$

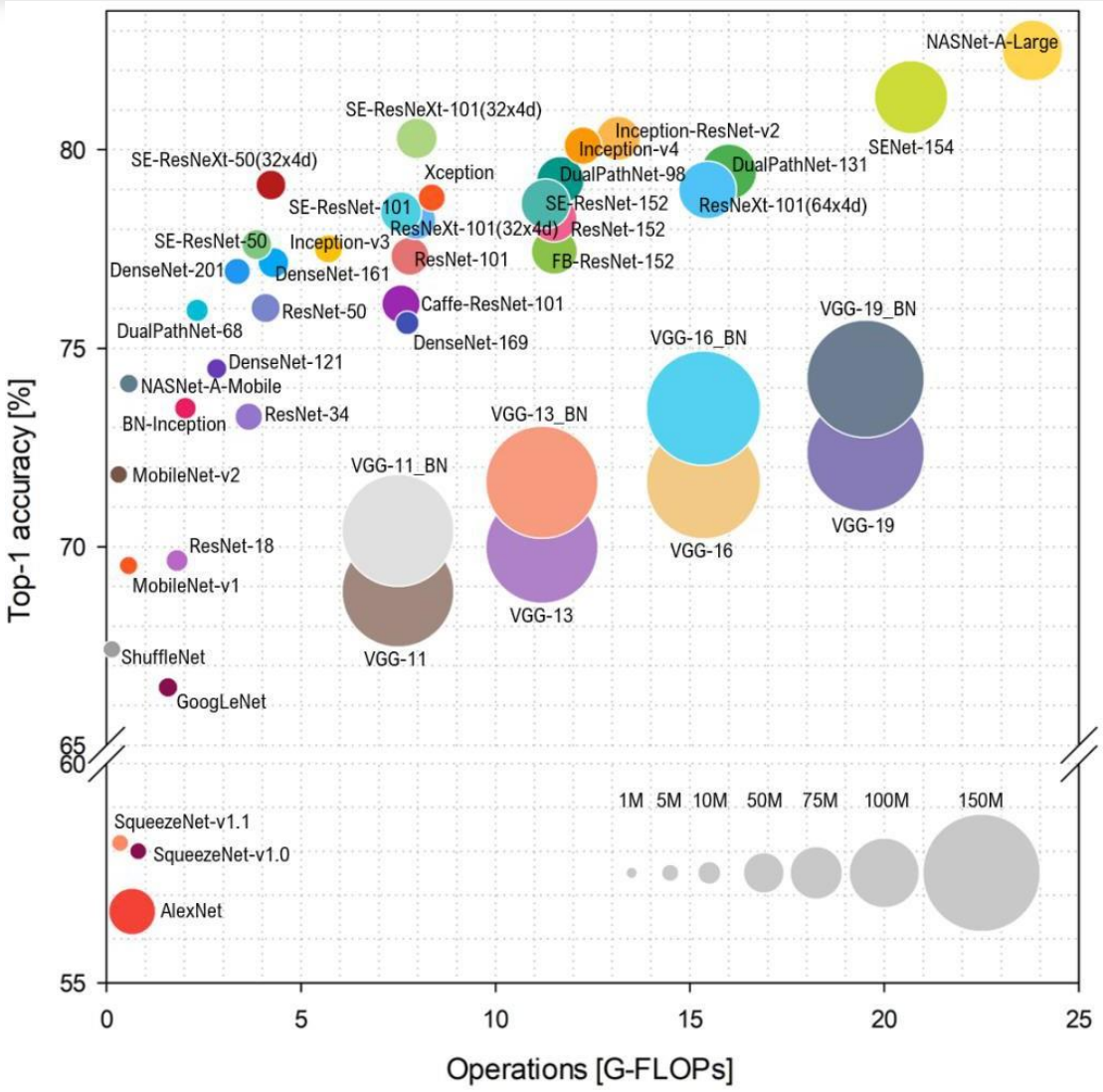
- Assume layer 1 captures edges, while layer 5 captures faces (and other stuff)
- Why not have a layer that combines both faces and edges (e.g. to model a scarred face)
- Standard ConvNets do not allow for this
- Layer 6 combines only layer 5 patterns, not lower



Neural Architecture Search

- It is also possible to learn the neural architecture
- Problem?
  - Architectures/graphs are discrete structures → Backprop?
  - Still, some very interesting workarounds have been proposed in practice
  - Will it work for you? If you are Facebook or Google, yes!
  - Competitive to recent state-of-the-art specifically designed for NAS

State-of-the-Art



Source: http://uvadlc.github.io/

Summary

- Convolutions
- Shared filters through local connectivity
- Convolutional Neural Networks
- Alexnet case study
- Visualizing ConvNets
- Transfer learning

Source: http://uvadlc.github.io/

Reading material & references

- http://www.deeplearningbook.org/
  - Part II: Chapter 9
  - [He2016] He, Zhang, Ren, Sun. Deep Residual Learning for Image Recognition, CVPR, 2016
  - [Simonyan2014] Simonyan, Zisserman/ Very Deep Convolutional, Networks for Large-Scale Image Recognition, arXiv, 2014
  - [Taigman2014] Taigman, Yang, Ranzato, Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification, CVPR, 2014
  - [Zeiler2014] Zeiler, Fergus. Visualizing and Understanding Convolutional Networks, ECCV, 2014
  - [Krizhevsky2012] Krizhevsky, Hinton. ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012
  - [LeCun1998] LeCun, Bottou, Bengio, Haffner. Gradient-Based Learning Applied to Document Recognition, IEEE, 1998

Source: http://uvadlc.github.io/