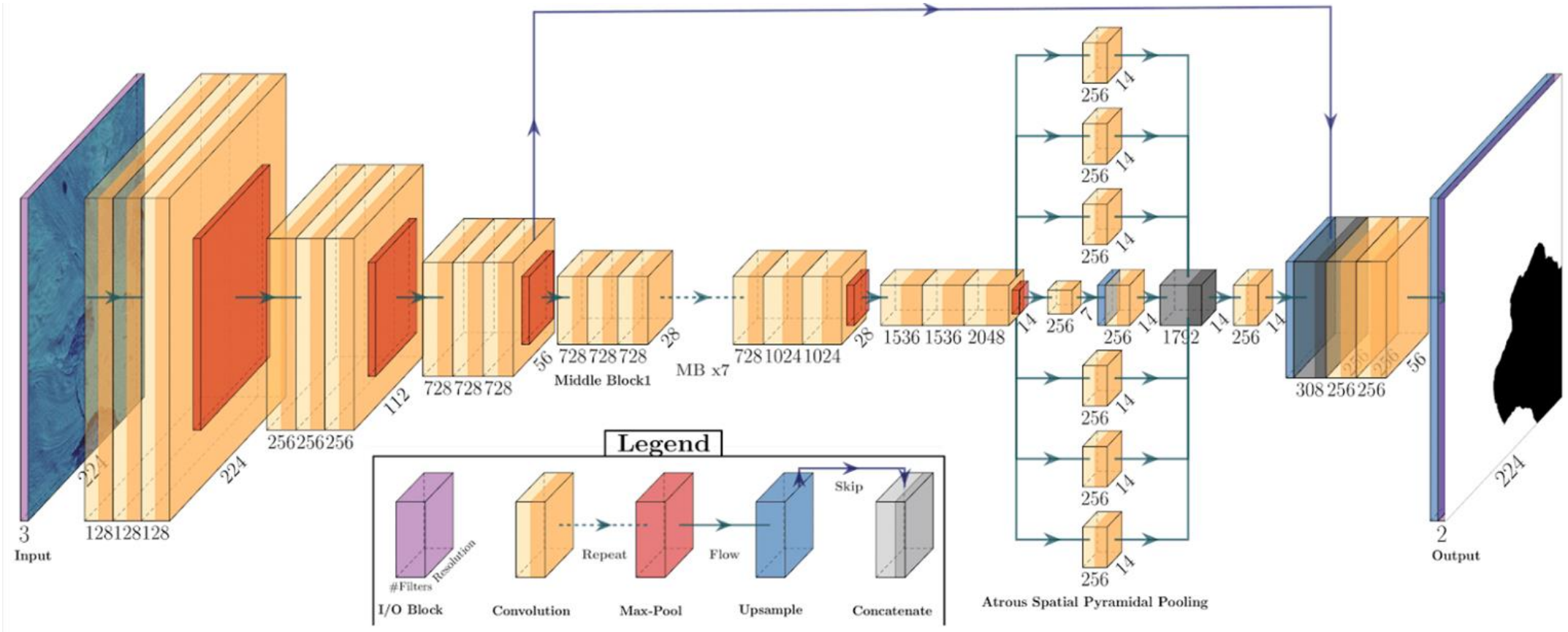
Convolutional Neural Networks



Slides adapted from:

CS231N: Fei Li, Andrej Karpathy & Justin Johnson

MIT6874: Dana Erlich, Param Vir Singh, David Gifford, Alexander Amini, Ava Soleimany

Daniel Cheng

Outline

Part 1

- **Classical machine vision foundations**
- **Convolutional Neural Networks (CNN) Foundations**
- **CNN Architectures**

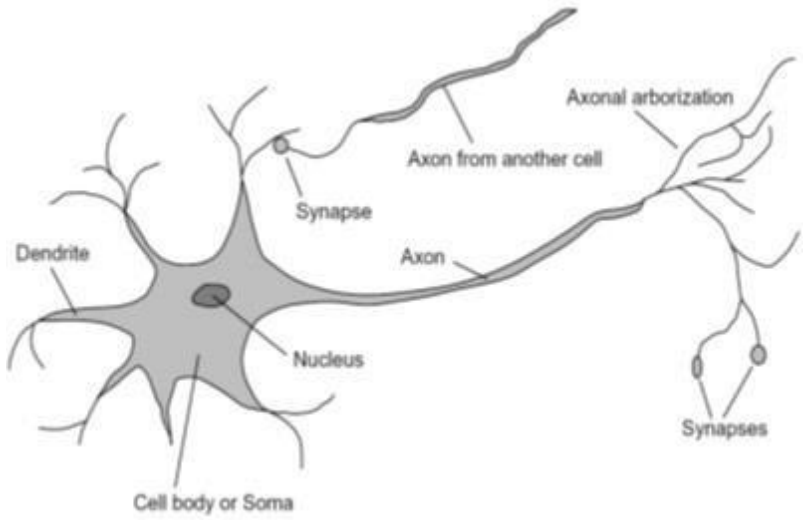
Part 2

- Training CNNs
- Understanding and Visualizing CNNs

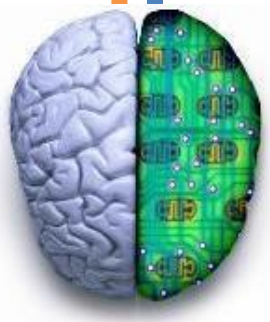
Part 3

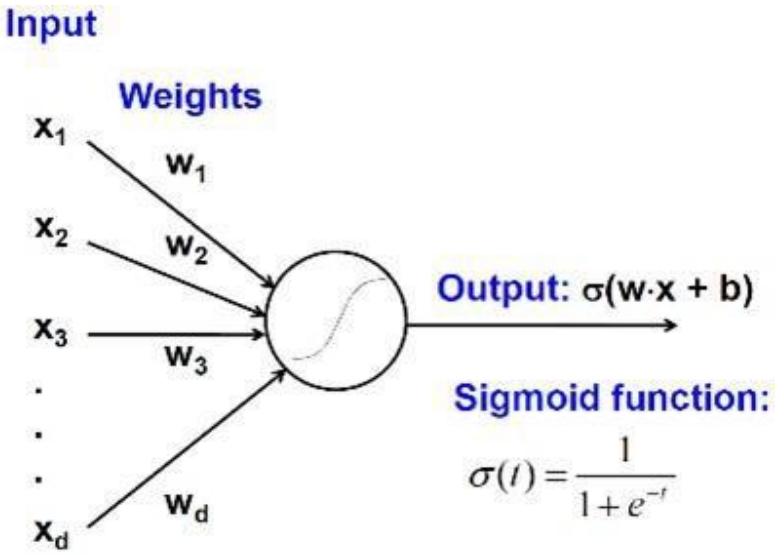
- Applications

Biological neuron and Perceptrons



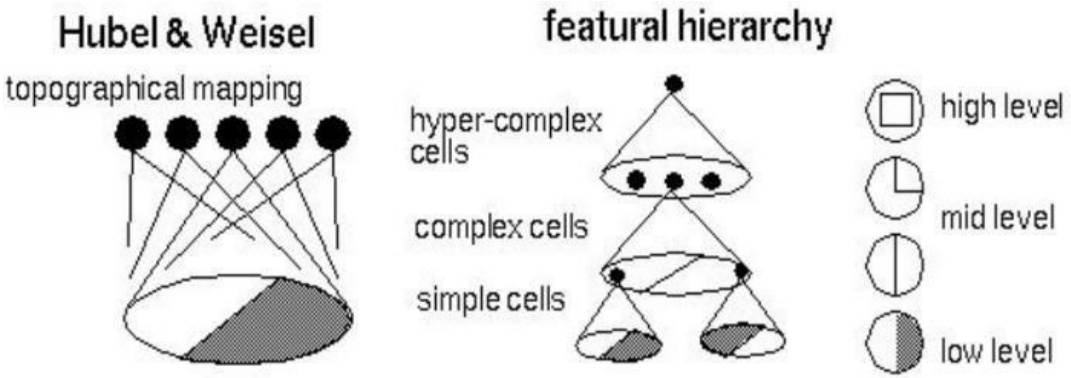
A biological neuron



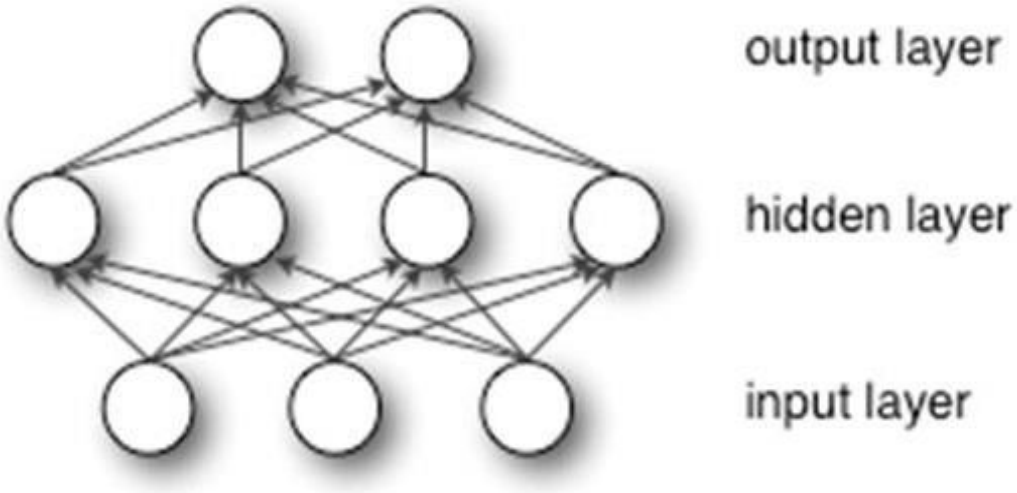


An artificial neuron (Perceptron)
- a linear classifier

Biologically Inspired Multi-layer Perceptrons



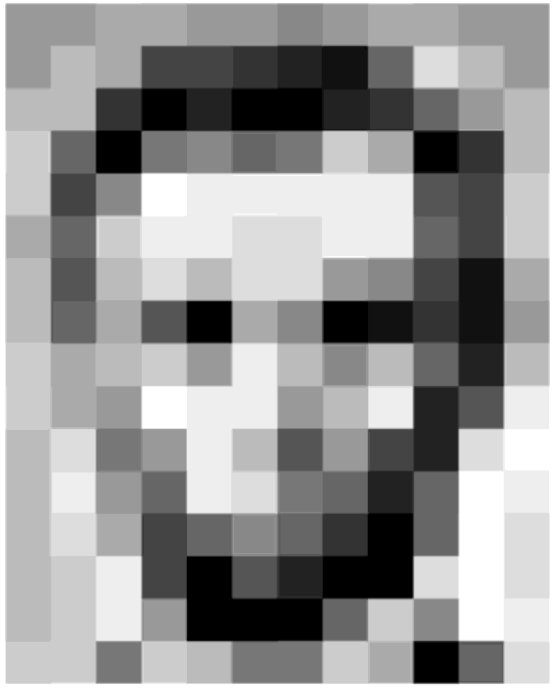
Biological Inspiration & Architecture



Multi-layer Perceptron
- A *non-linear* classifier

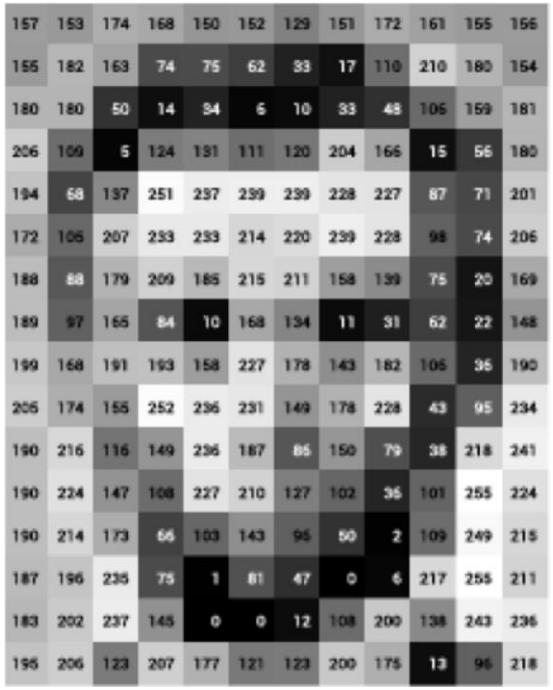
What computers ‘see’: Images as Numbers

What you see



Input Image

What you both see



Input Image + values

What the computer "sees"

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	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	155	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	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

Pixel intensity values
(“pix-el”=picture-element)

Levin *Image Processing & Computer Vision*

- An image is just a matrix of numbers [0,255].i.e.,1080x1080x3 for an RGB image.
- Question: is this Lincoln? Washington? Jefferson? Obama?
- MLPs/NNs allow the computer to perceive not just numbers, but features

Learning a Hierarchy of Feature Extractors

- Classical computer vision techniques such as edge detection, Support Vector Machines (SVMs)/K-nearest neighbors (KNNs), and texture analysis filters require expert knowledge

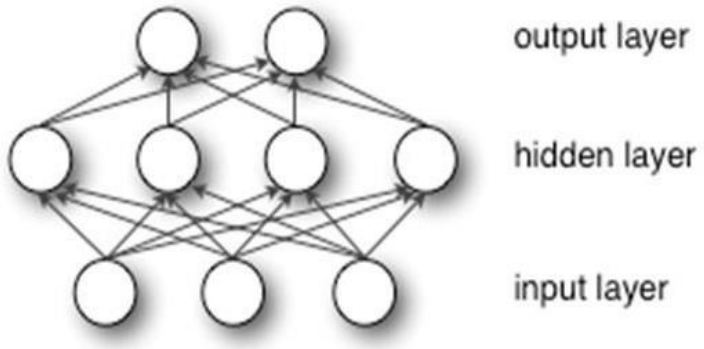
- **Key Innovation: CNNs and ML allow the network training to automatically learn the features**

Multi-layer Perceptrons, or Neural Networks

- Also known as MLPs, NNs, Artificial Neural Networks (ANNs)
- **Training:** find network weights **w** to minimize the error between true training labels y_i and estimated labels $f_w(\boldsymbol{x_i})$

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f_{\mathbf{w}}(\mathbf{x}_i)\right)^2$$

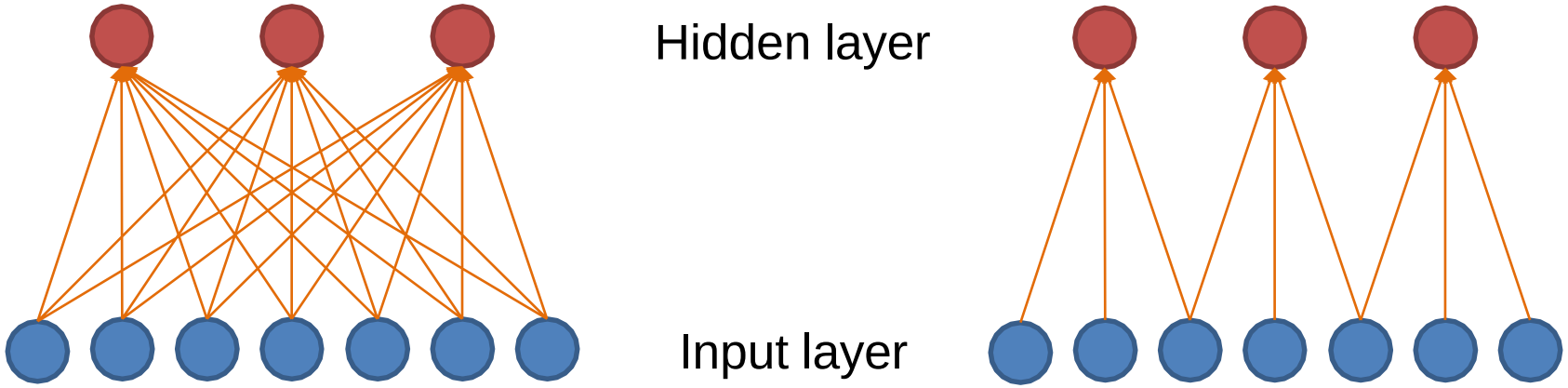
- Minimization can be done by gradient descent provided f is differentiable
- This training method is called **back-propagation**



Convolutional Neural Networks

- Also known as CNNs, ConvNets
- Subset of NNs with the following features:
  1. Local connectivity
  2. Weight sharing
  3. Typically, handle multiple input channels

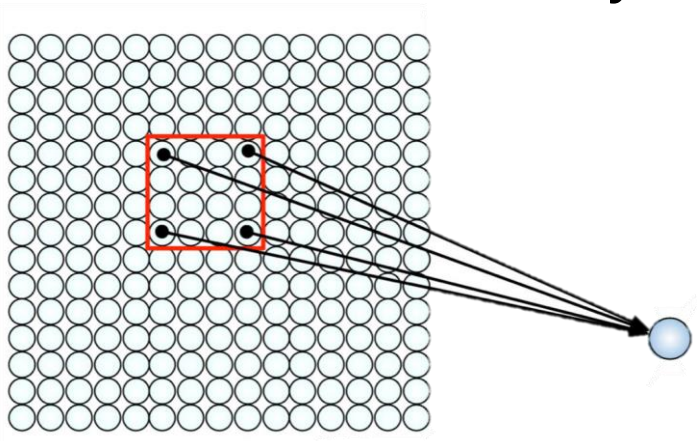
CNN: Local Connectivity



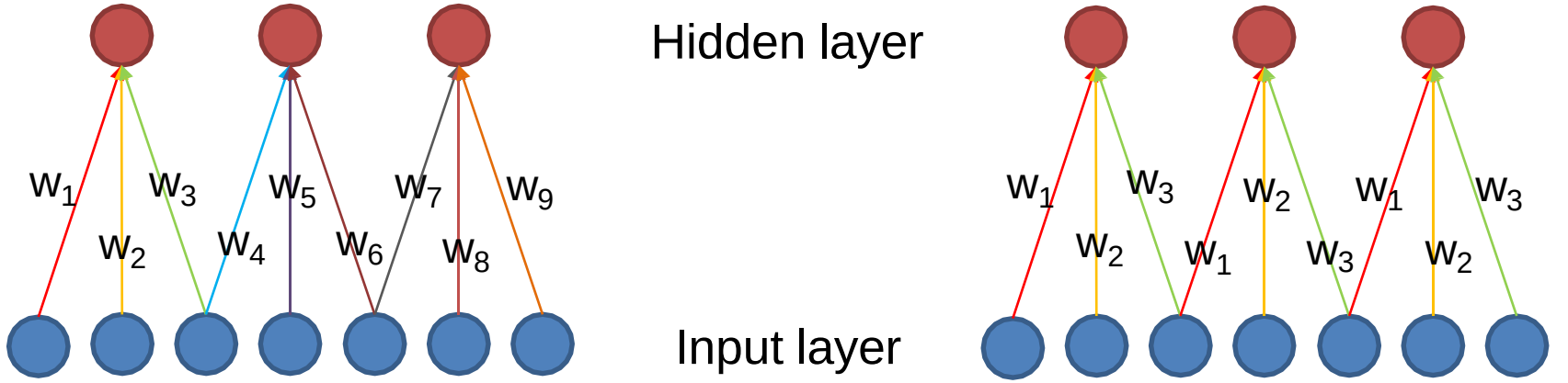
Global connectivity

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Global connectivity: 3 x 7 = 21
  - Local connectivity: 3 x 3 = 9

Local connectivity



CNN: Weight Sharing

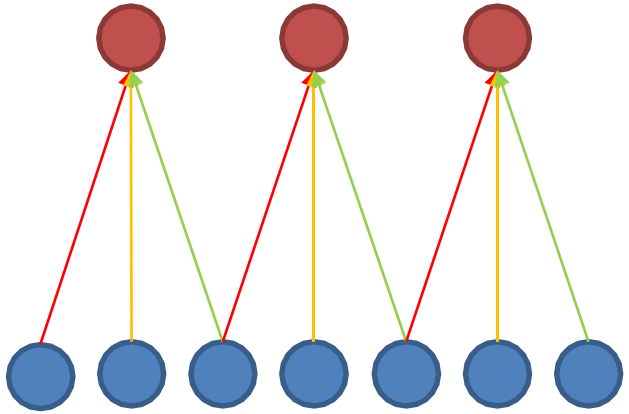


Without weight sharing

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters
  - Without weight sharing: 3 x 3 = 9
  - With weight sharing : 3 x 1 = 3

With weight sharing

CNN with multiple input channels

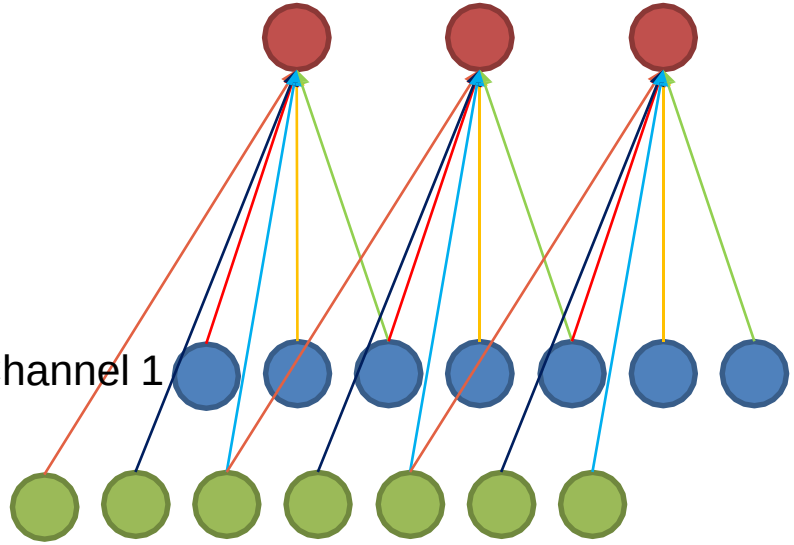


Hidden layer

Input layer

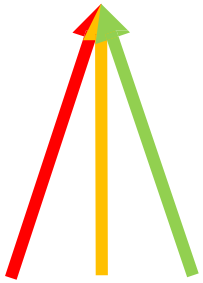
Channel 1

Channel 2

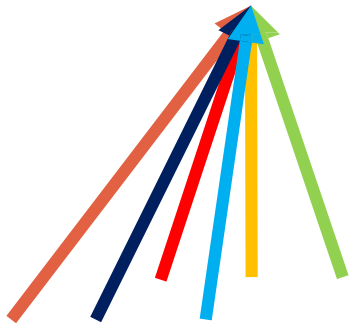


Single input channel

Multiple input channels



Filter weights



Filter weights

Putting them together

- Local connectivity
- Weight sharing
- Handling multiple input channels

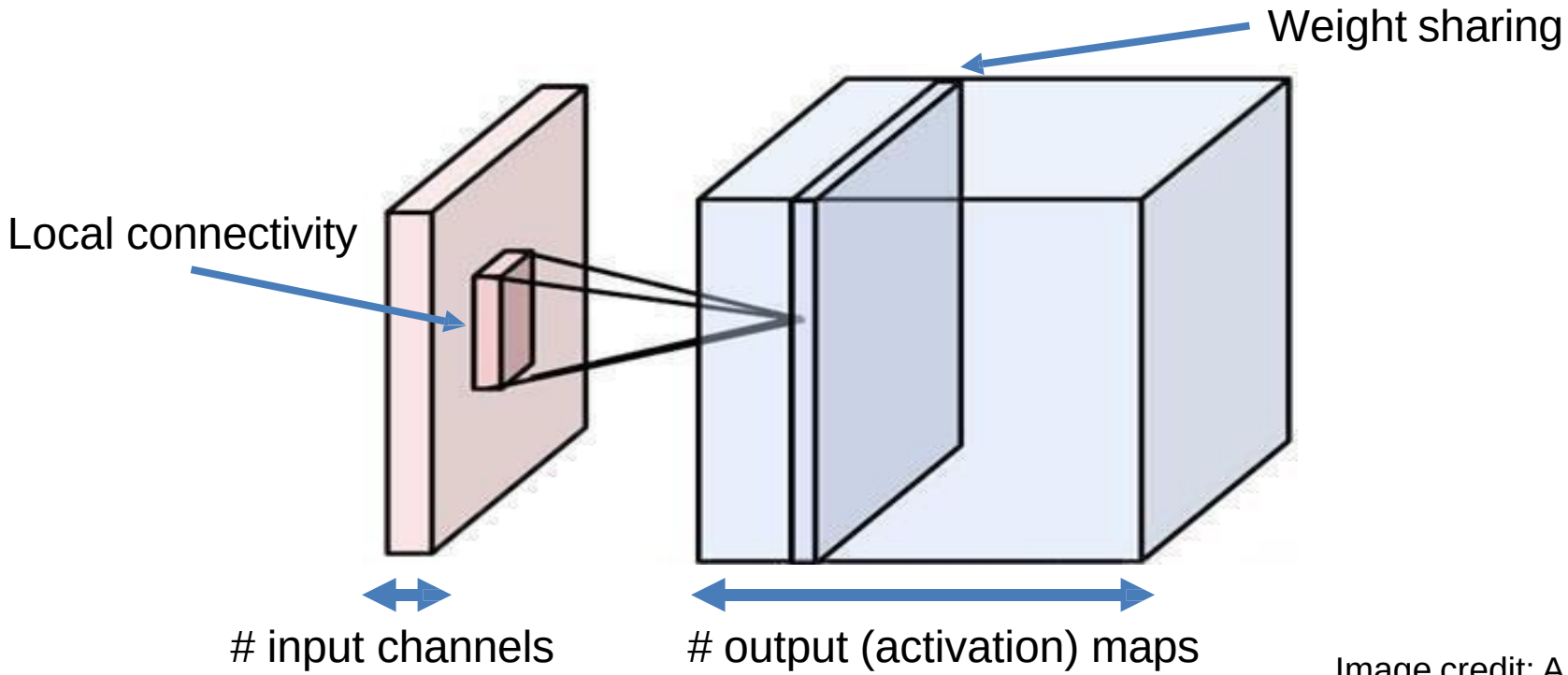
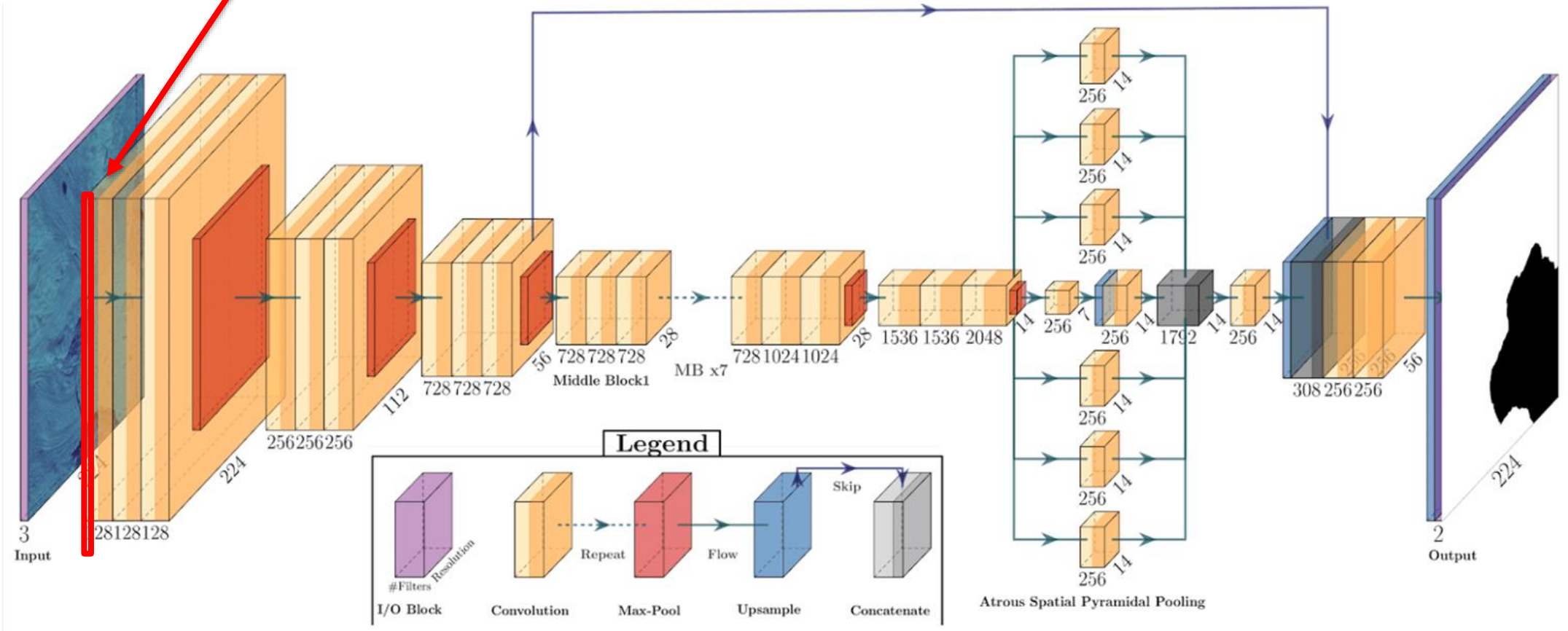


Image credit: A. Karpathy

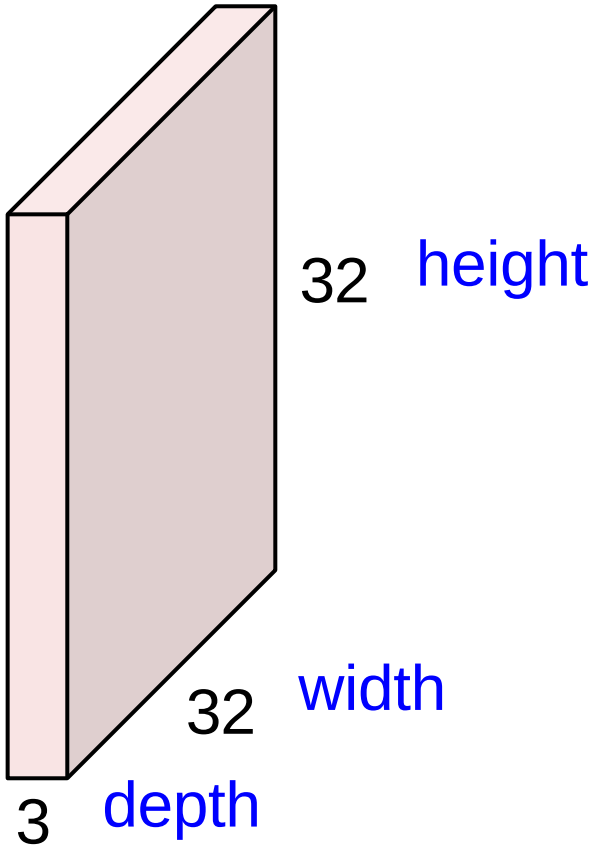
CNN Architecture

- Now, how to assemble a simple CNN?
- Let's begin single Convolutional layer



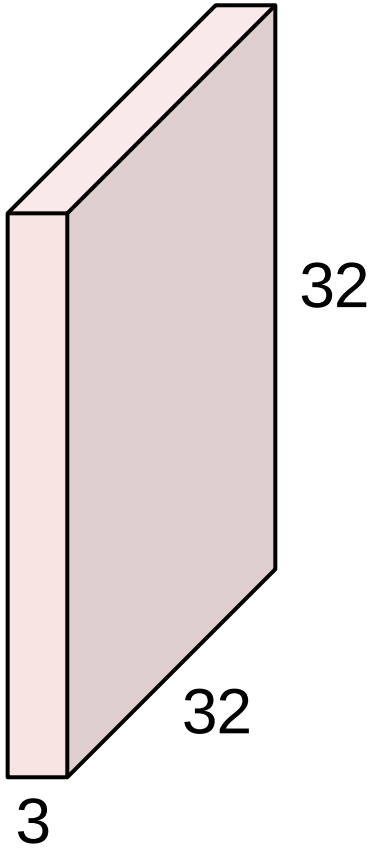
Convolution Layer

32x32x3 image

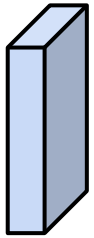


Convolution Layer

32x32x3 image



5x5x3 filter

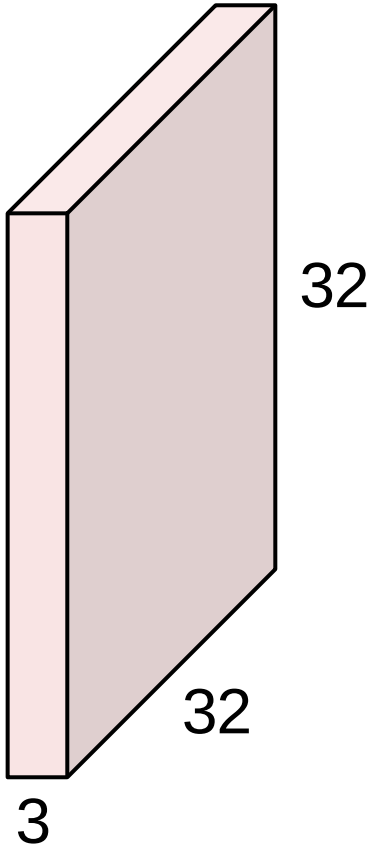


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

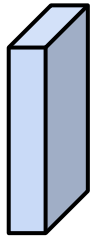
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

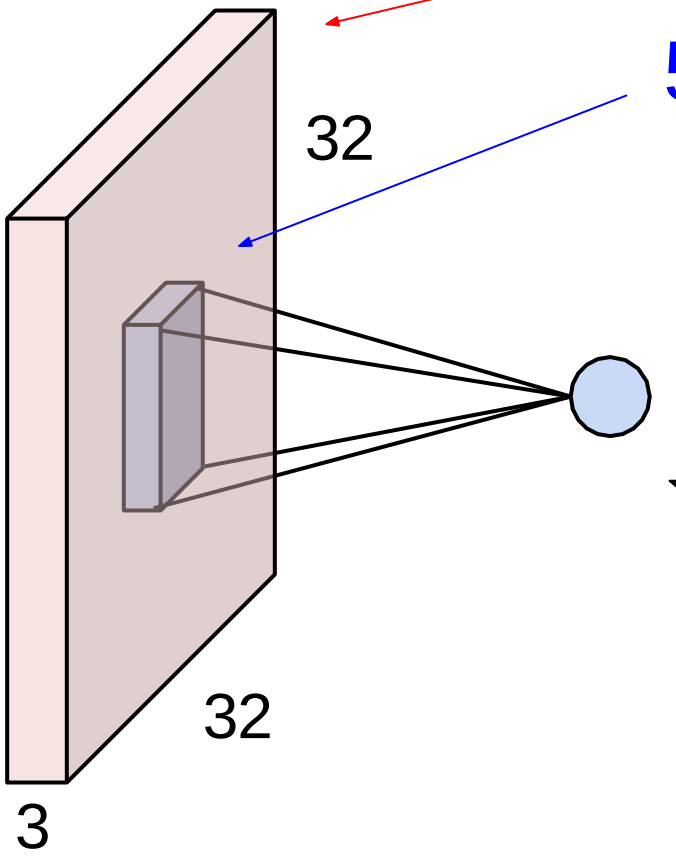


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



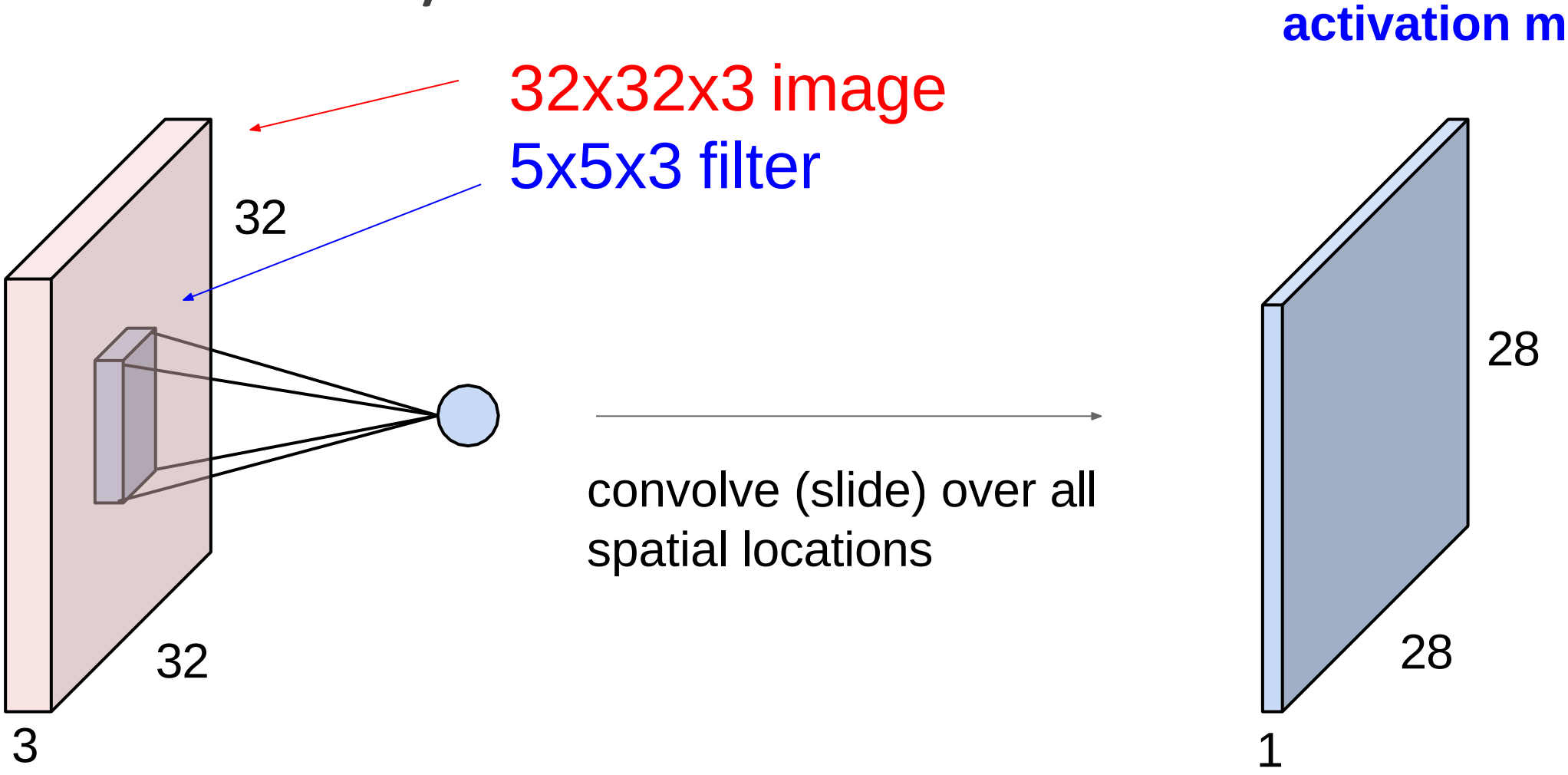
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

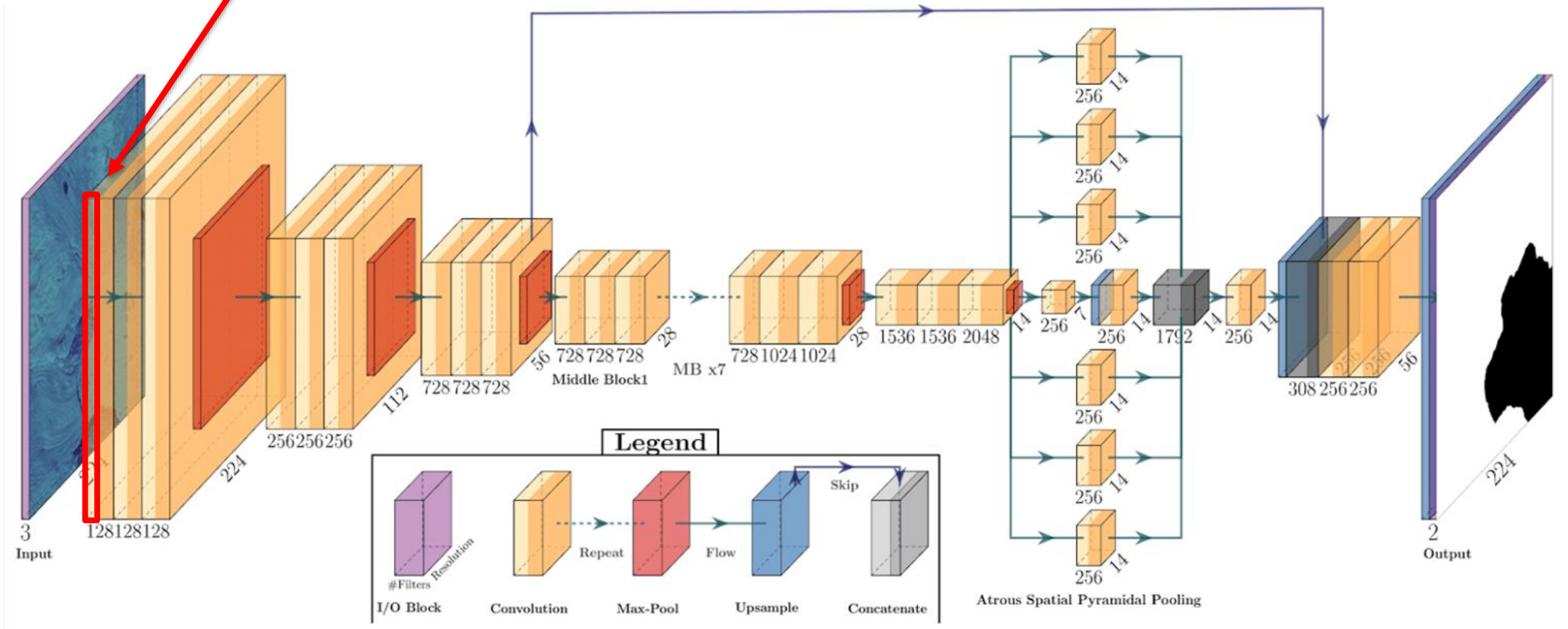
$$w^T x + b$$

Convolution Layer



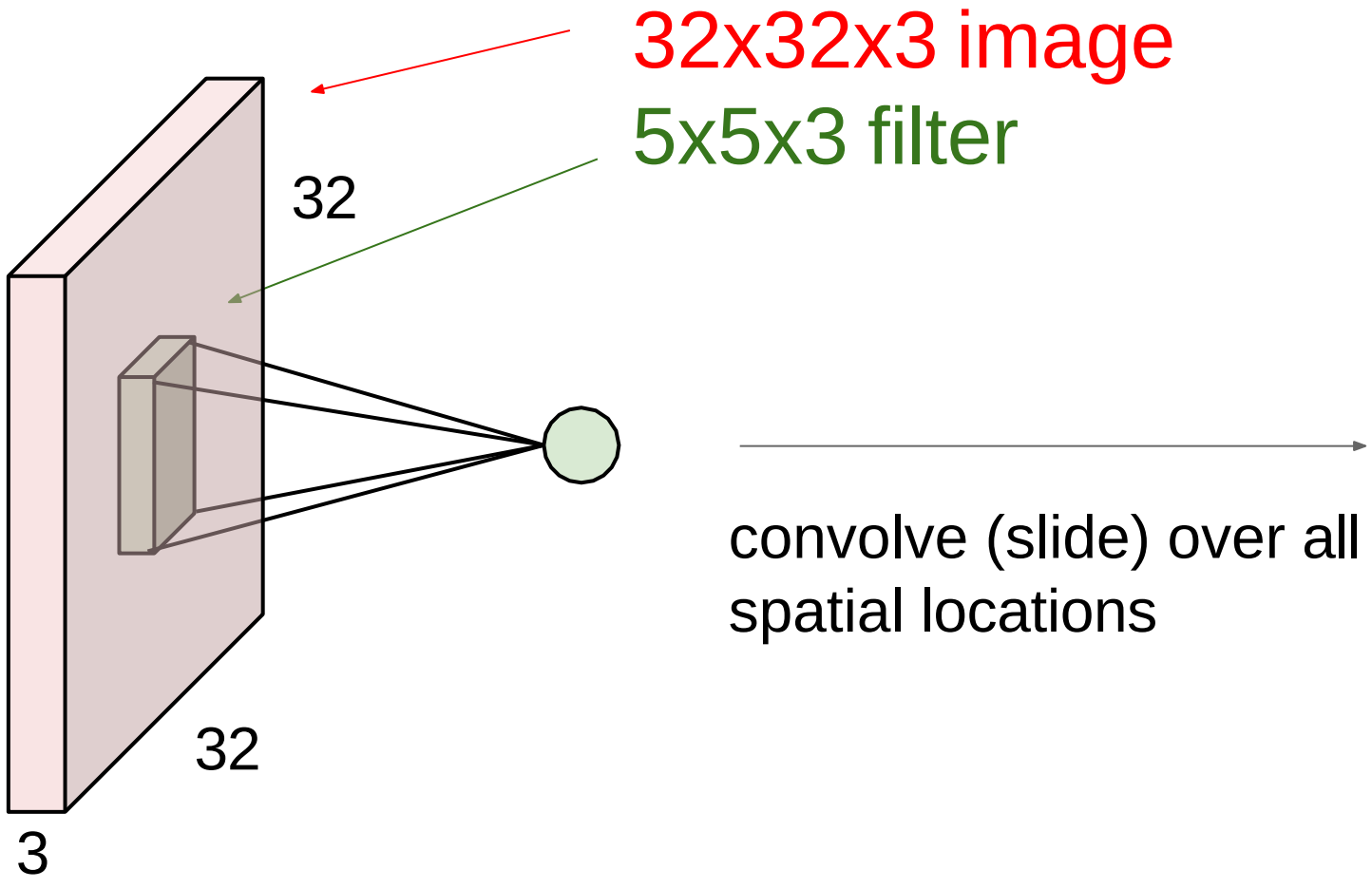
CNN Architecture

- We can create a single Convolutional layer
- Let's stack them together to get multiple feature/activation maps



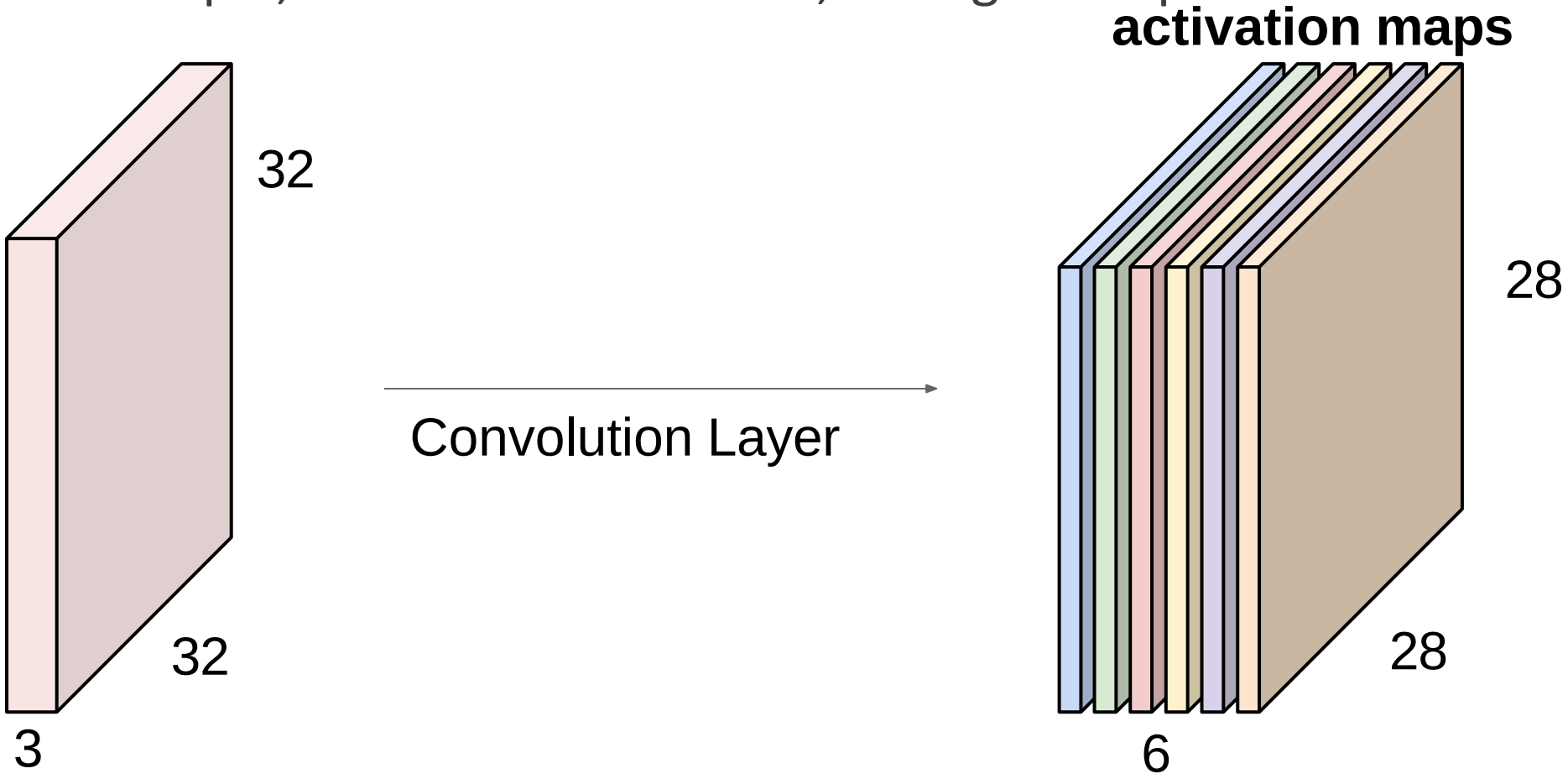
Convolution Layer

consider a second, green filter



Convolution Layer

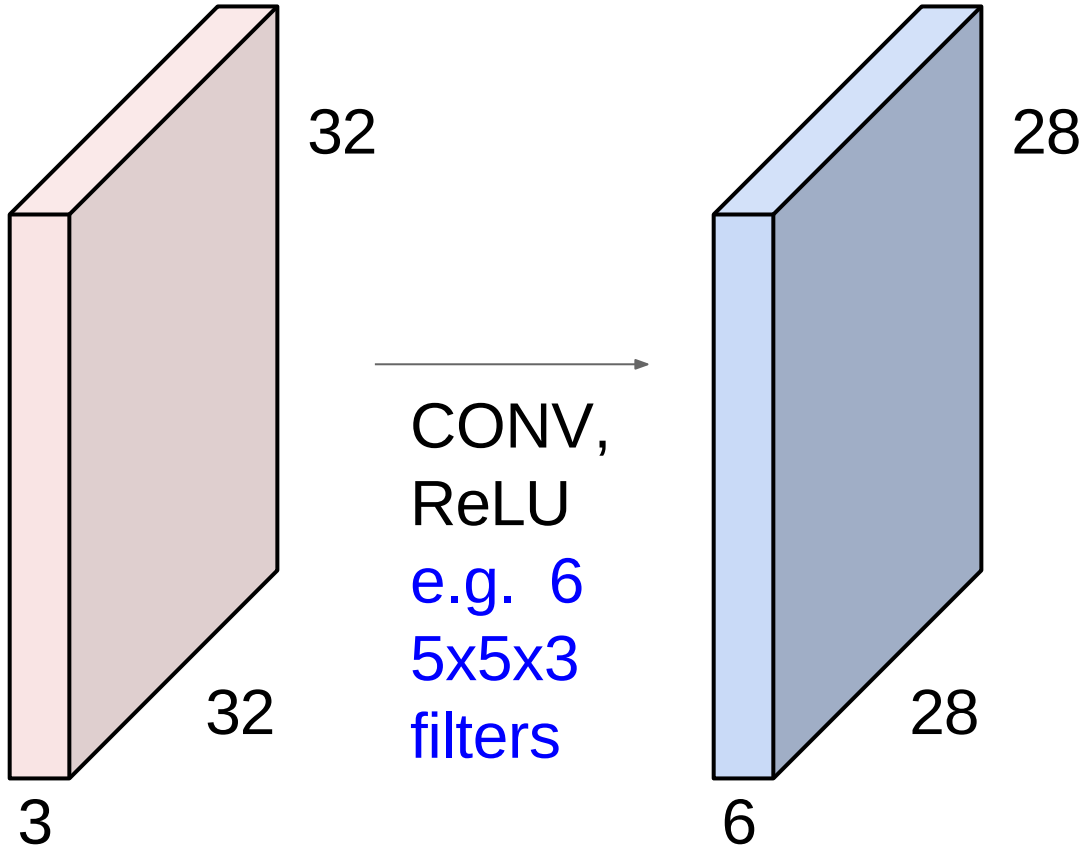
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

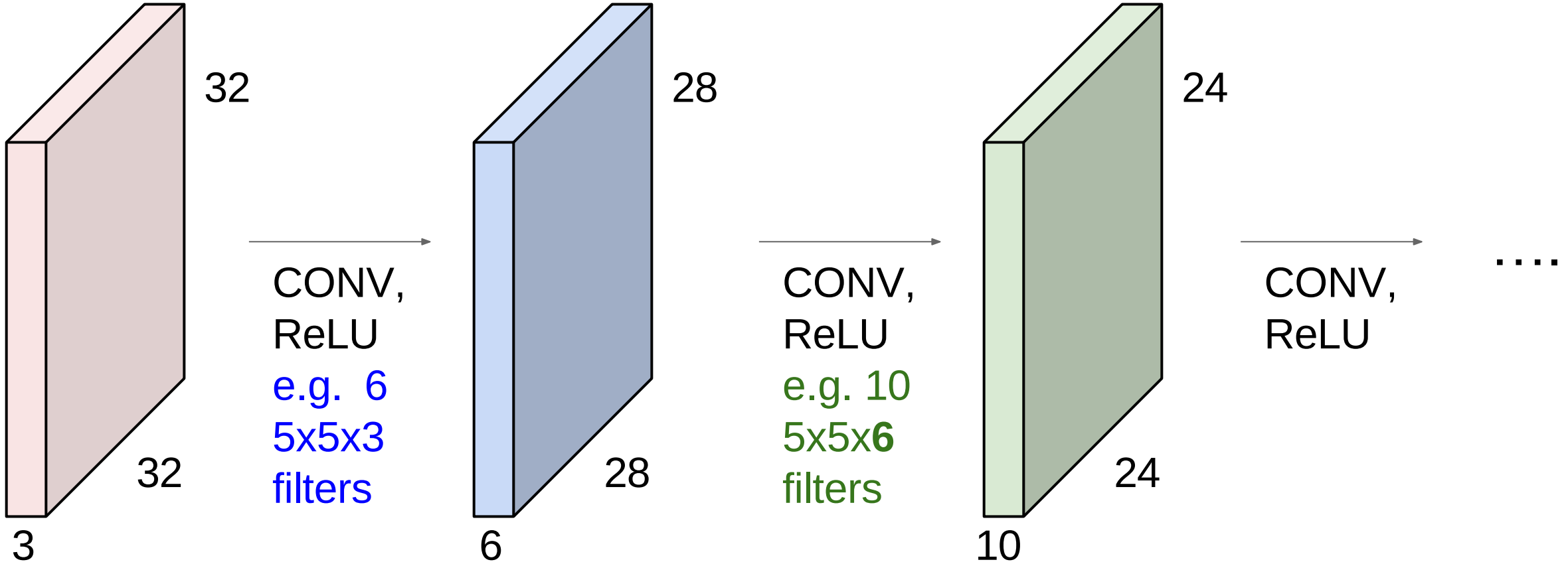
Convolution Layer

Note: CNNs are a sequence of Convolution Layers, **interspersed with activation functions**



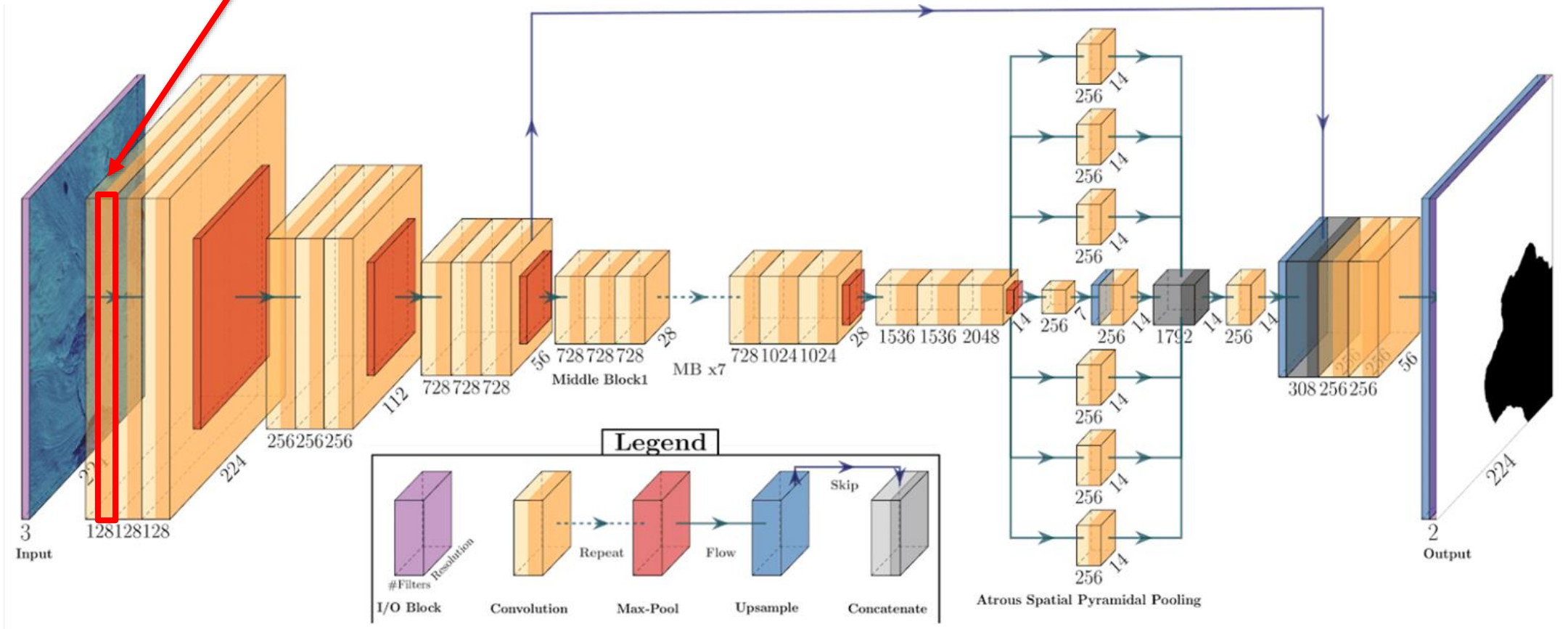
Convolution Layer

Note: CNNs are a sequence of Convolution Layers, **interspersed with activation functions**

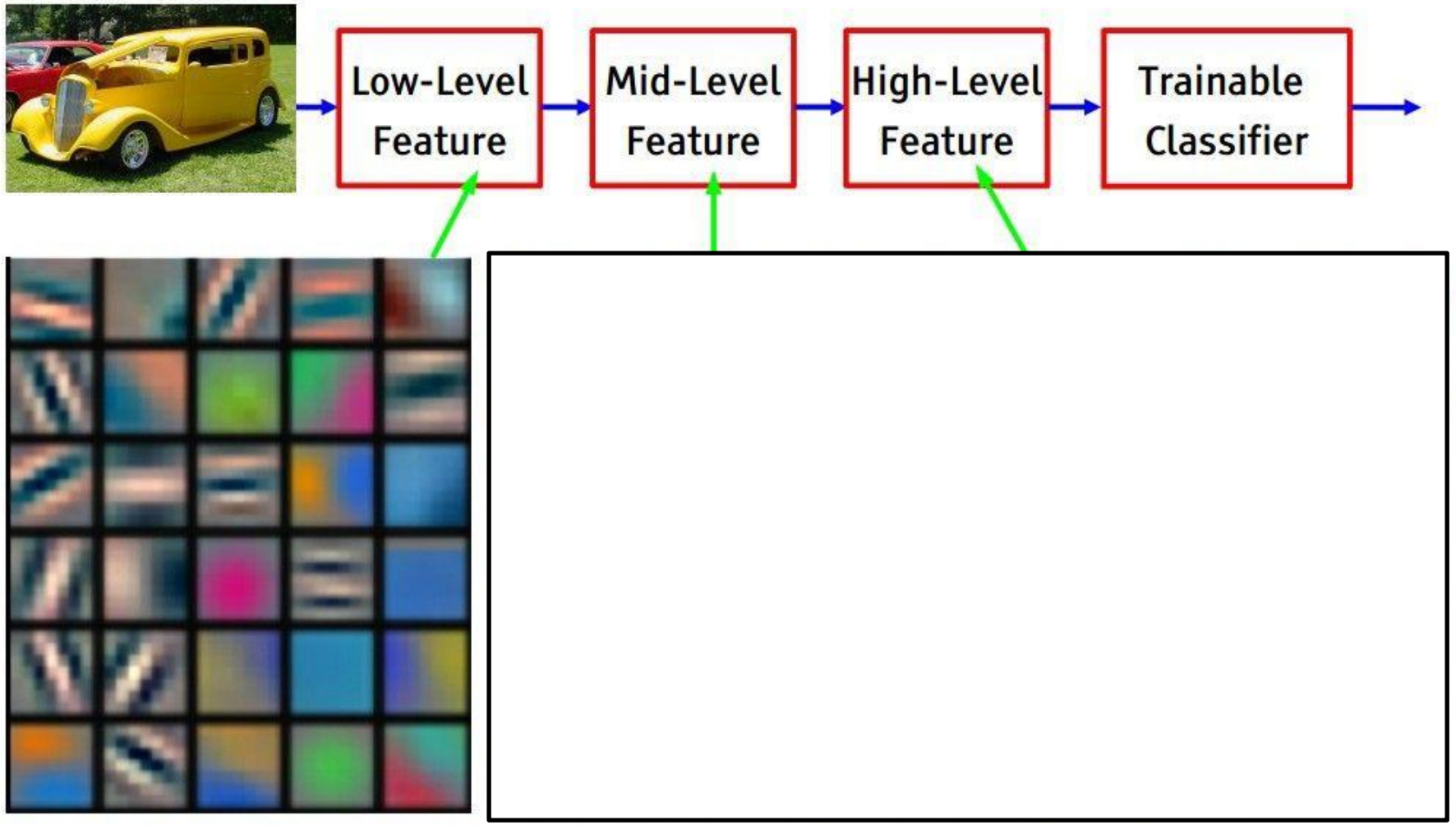


CNN Architecture

- We can get multiple feature/activation maps from previous layers
- What do those features look like?



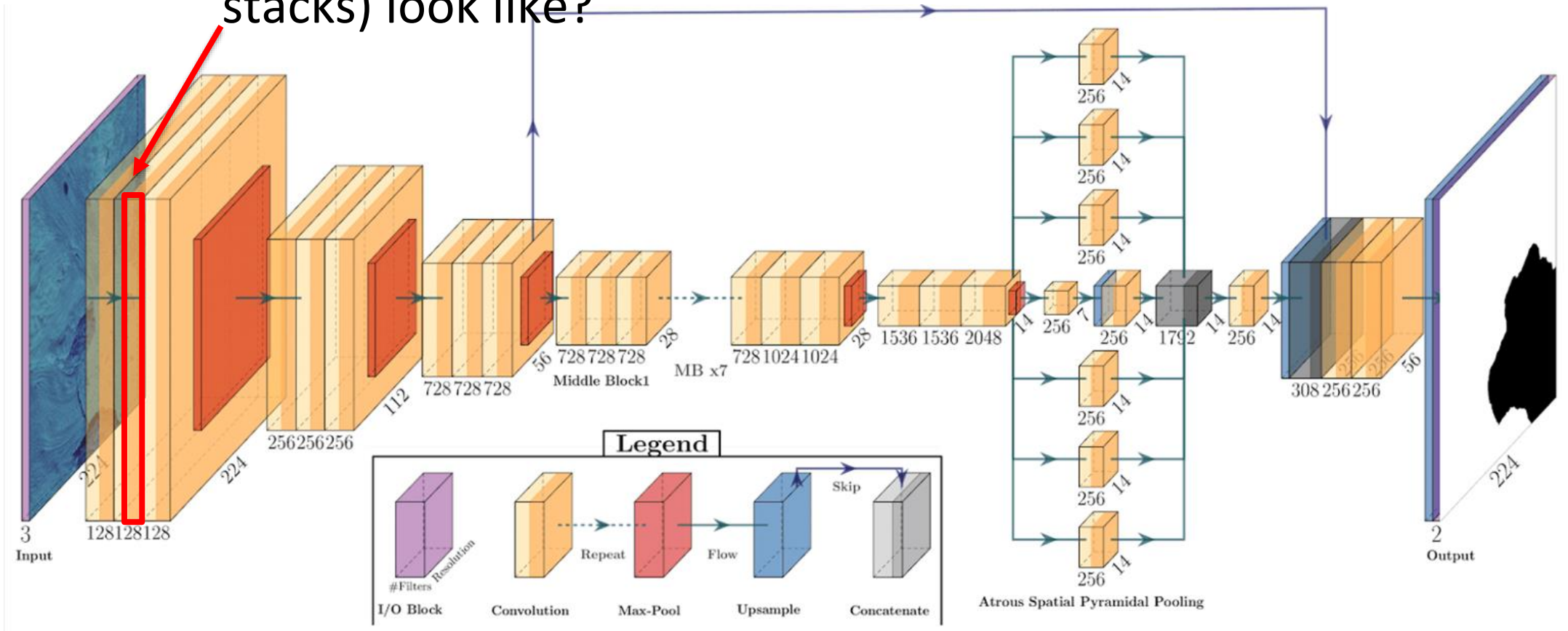
[From recent Yann LeCun slides]



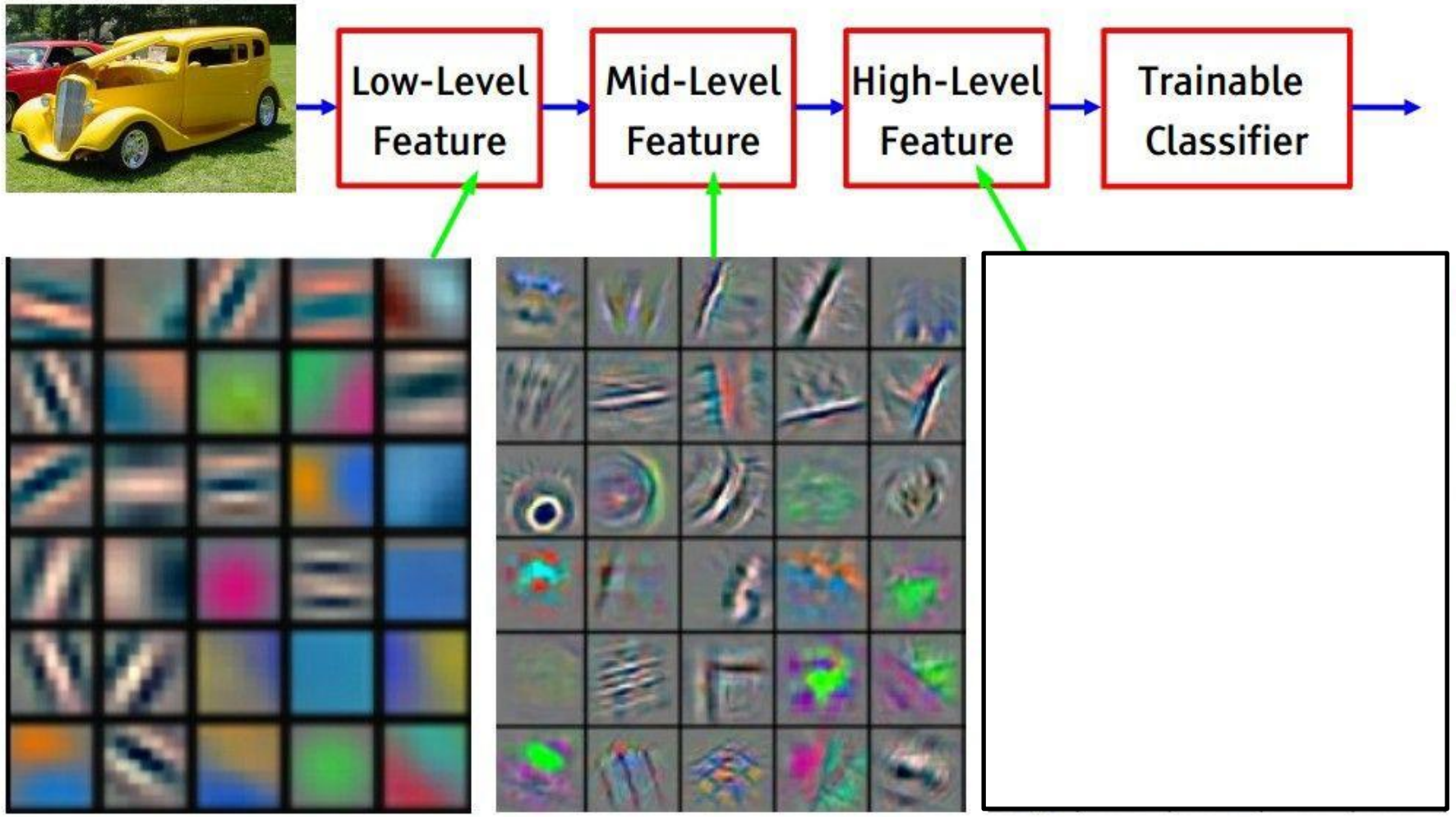
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN Architecture

- We can get simple features (lines, edges) from our input layers
- What do 2nd order features (from additional convolutional layer stacks) look like?



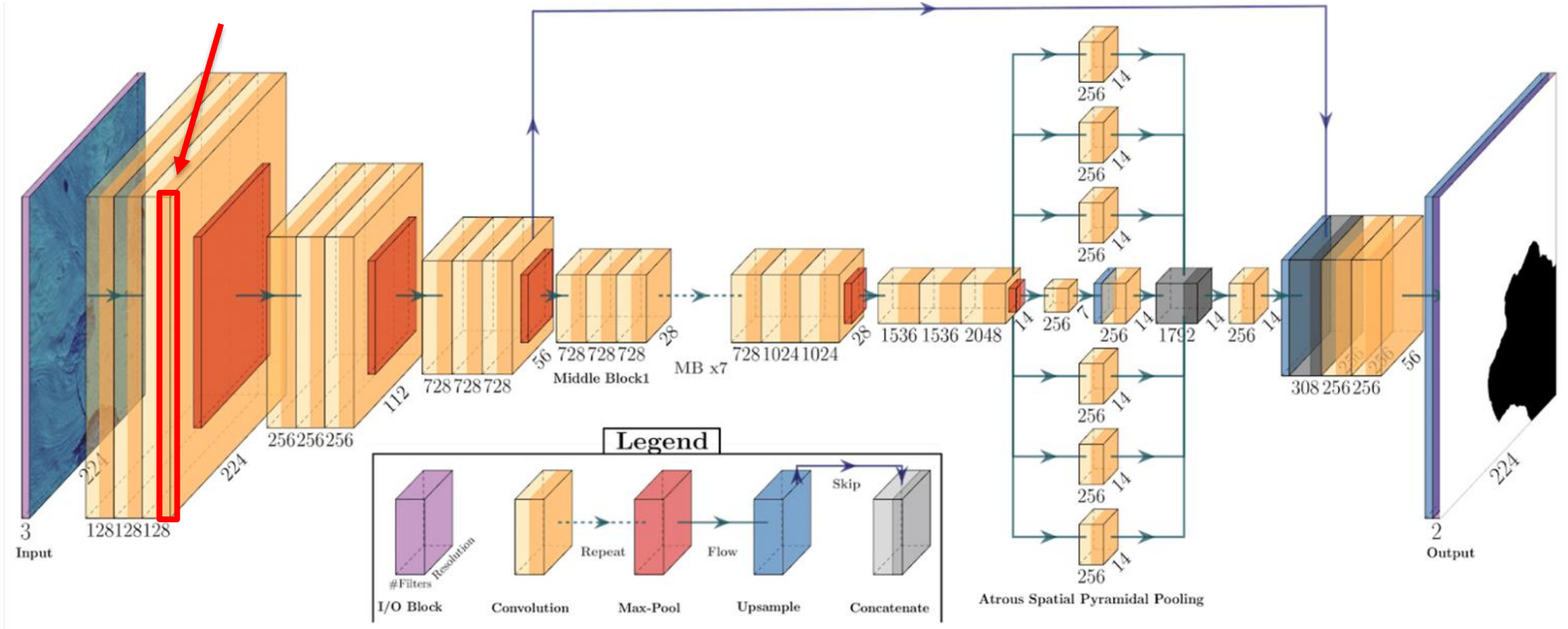
[From recent Yann LeCun slides]



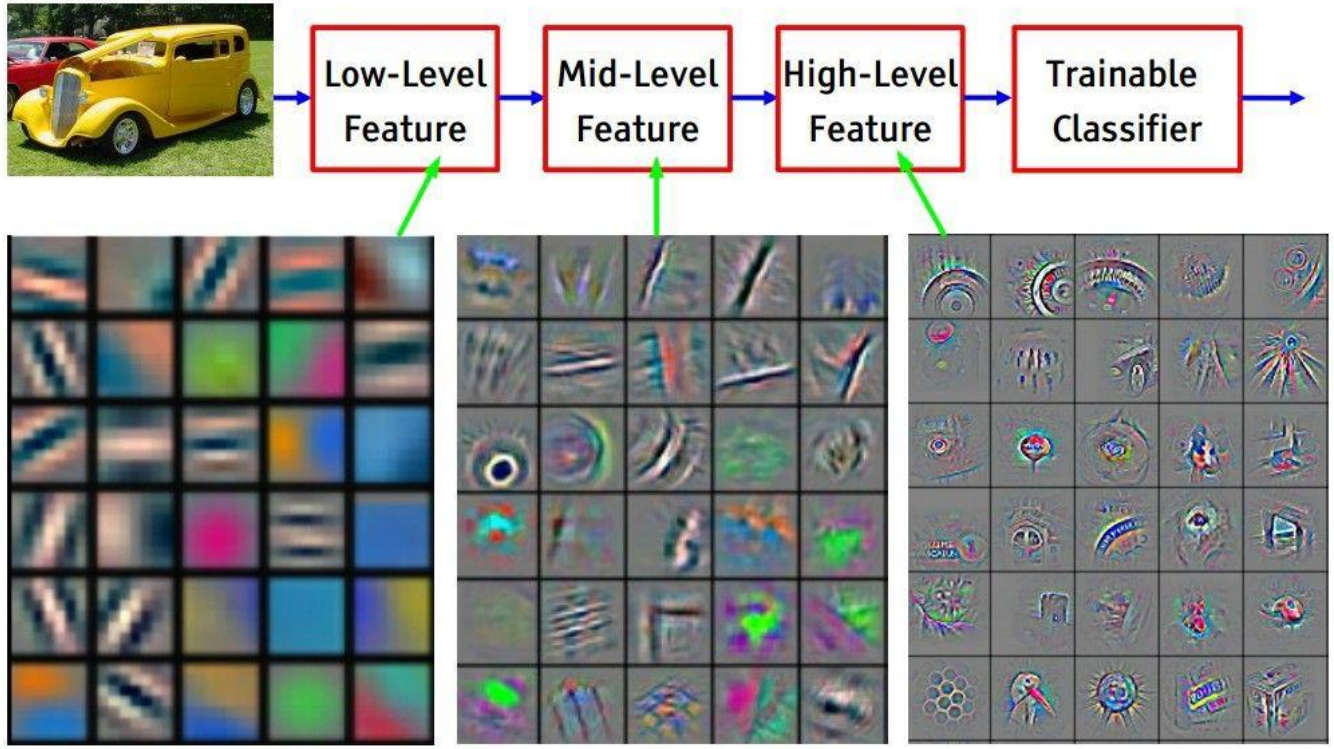
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN Architecture

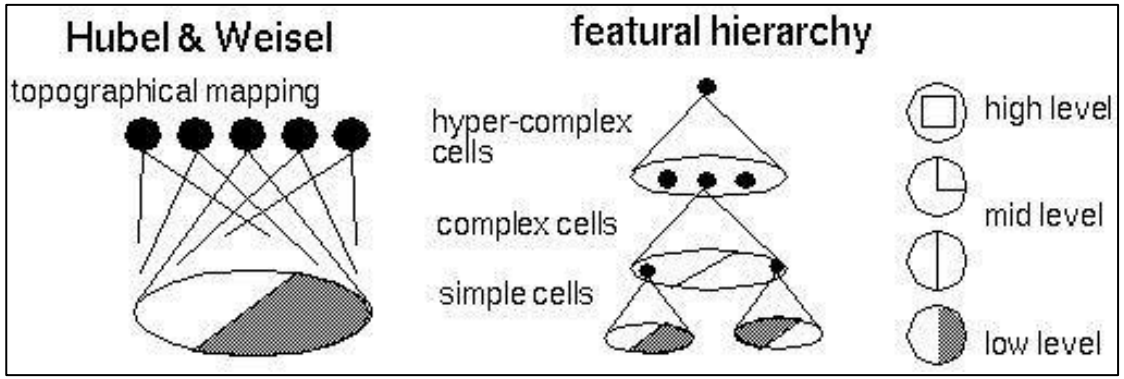
- We can get more complex features (shapes, corners, crosses) from our input layers
- What do higher order features (from additional convolutional layer stacks) look like?



[From recent Yann LeCun slides]

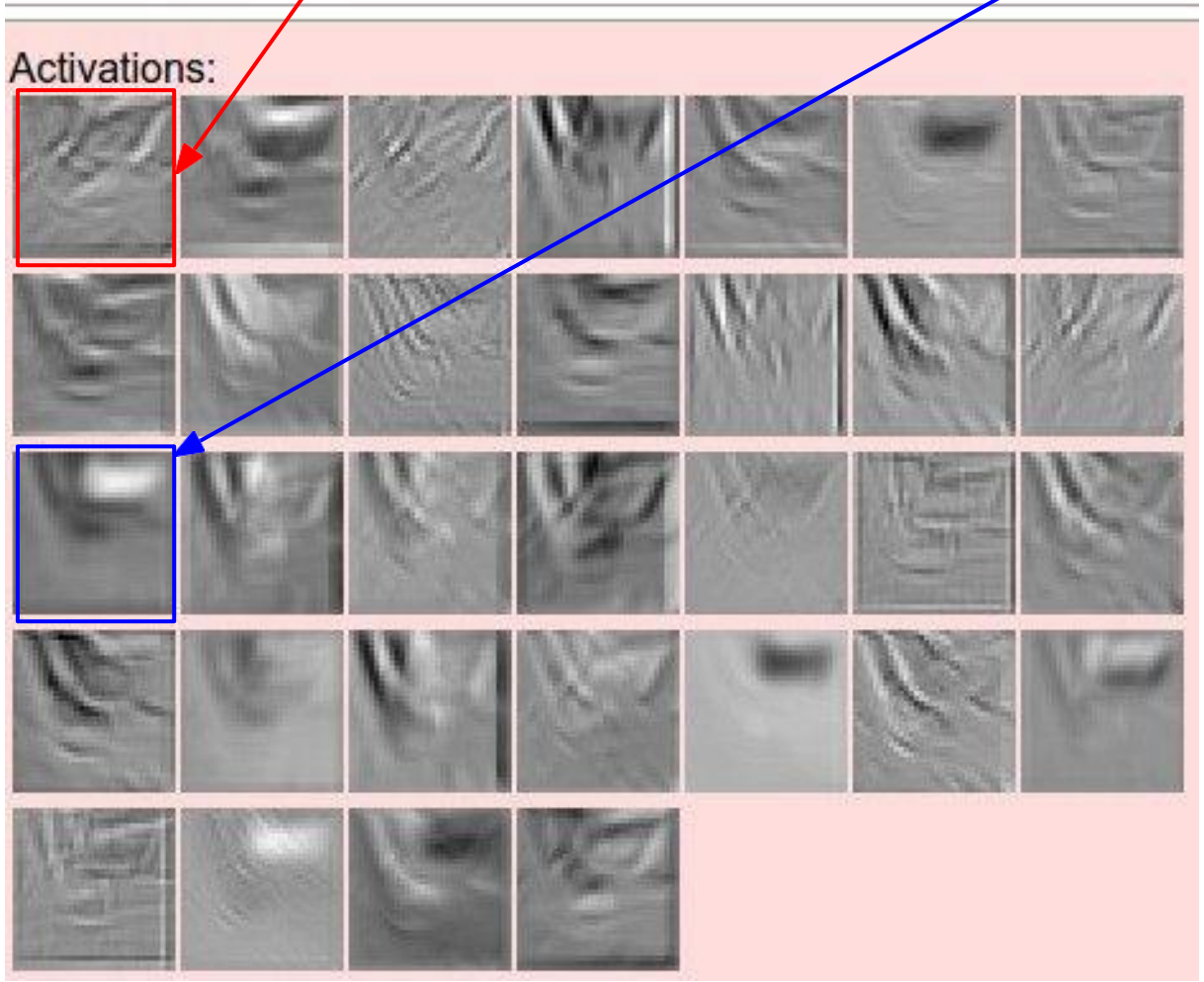


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



one filter =>

one activation map



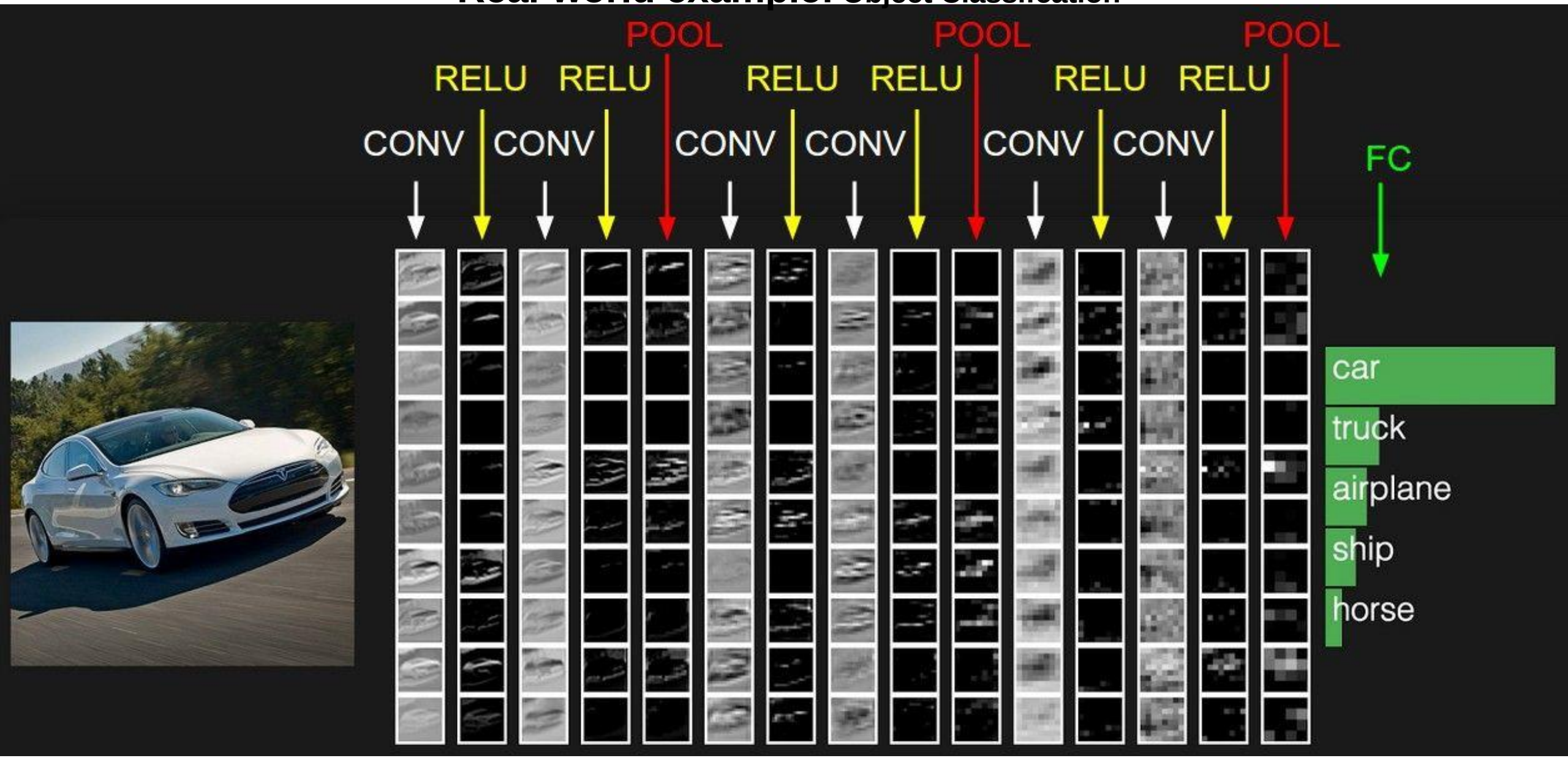
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

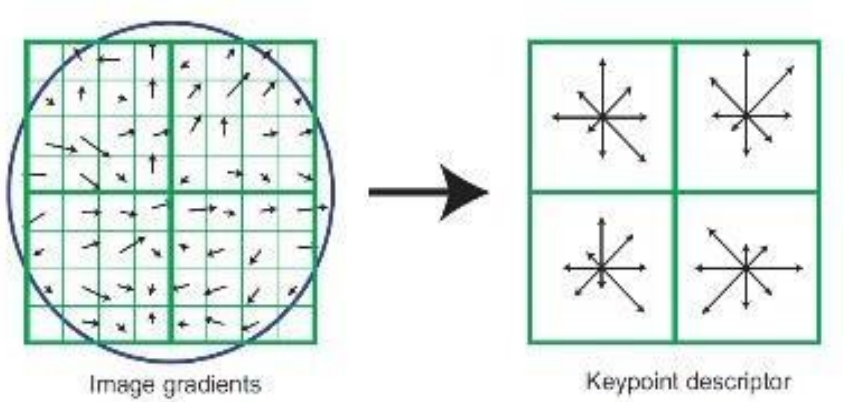
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

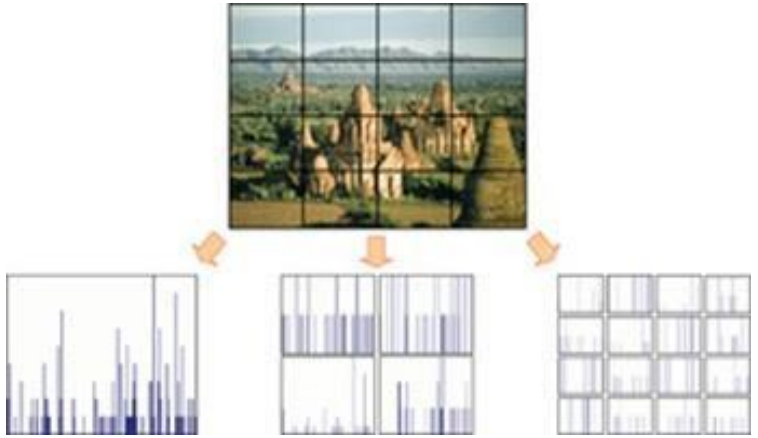
Real-world example: Object Classifcation



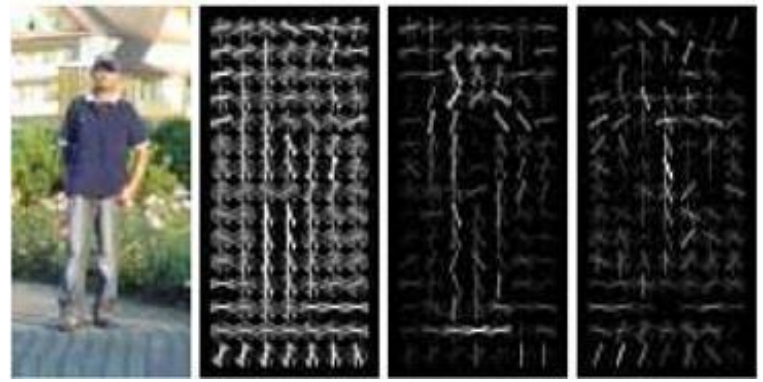
Context: Classical Computer Vision Features



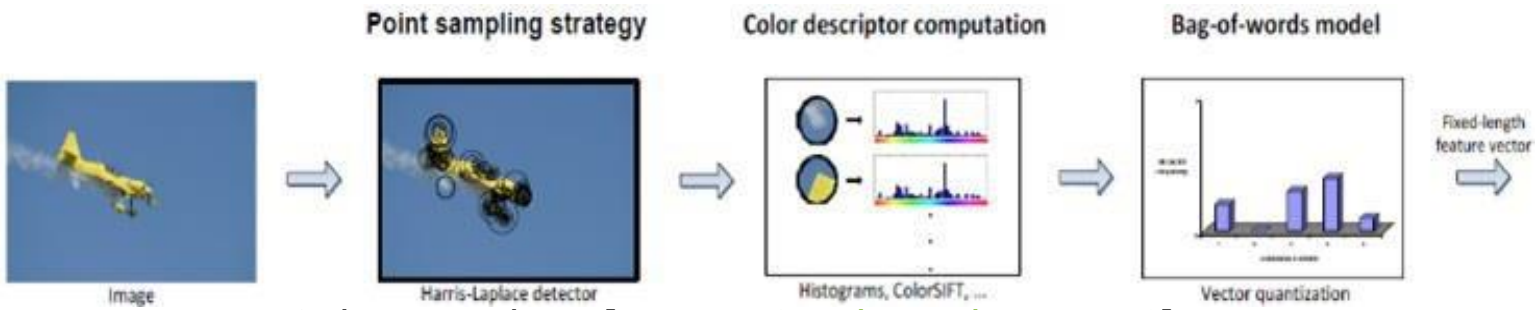
SIFT [Loewe IJCV 04]



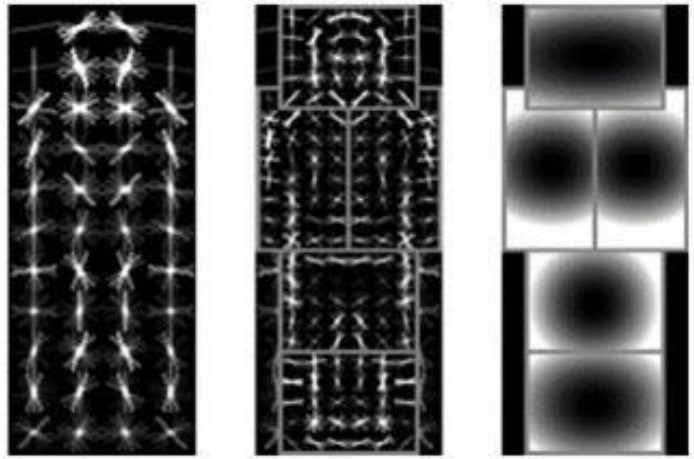
SPM [Lazebnik et al. CVPR 06]



HOG [Dalal and Triggs CVPR 05]



Color Descriptor [Van De Sande et al. PAMI 10]



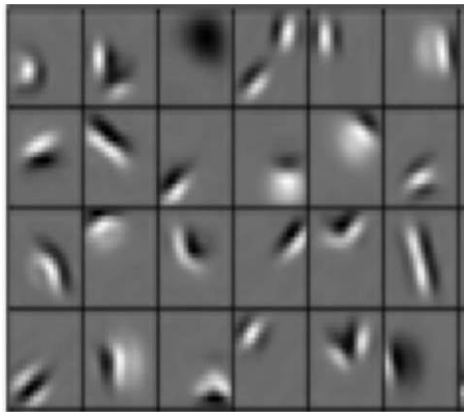
DPM [Felzenszwalb et al. PAMI 10]

Context: CNN Features

Key idea:

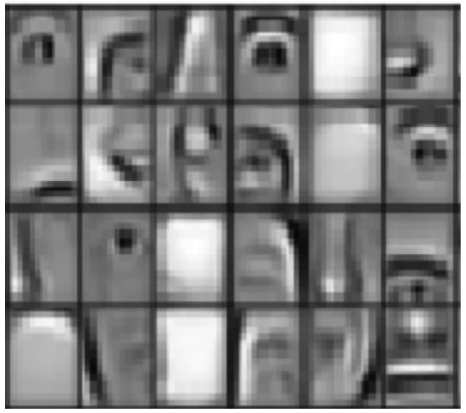
Learn hierarchy of features **directly** from the data

Low level features



Edges, dark spots

Mid level features



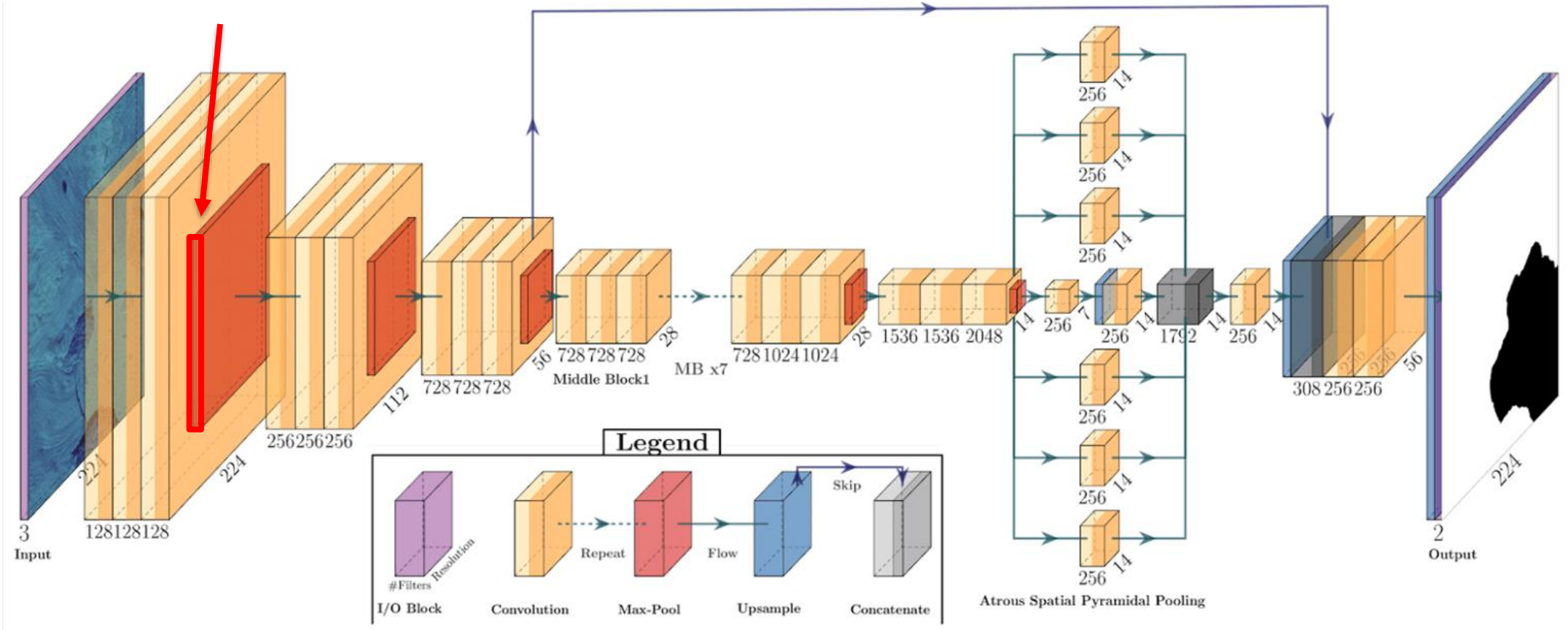
Eyes, ears,nose

High level features

Facial structure

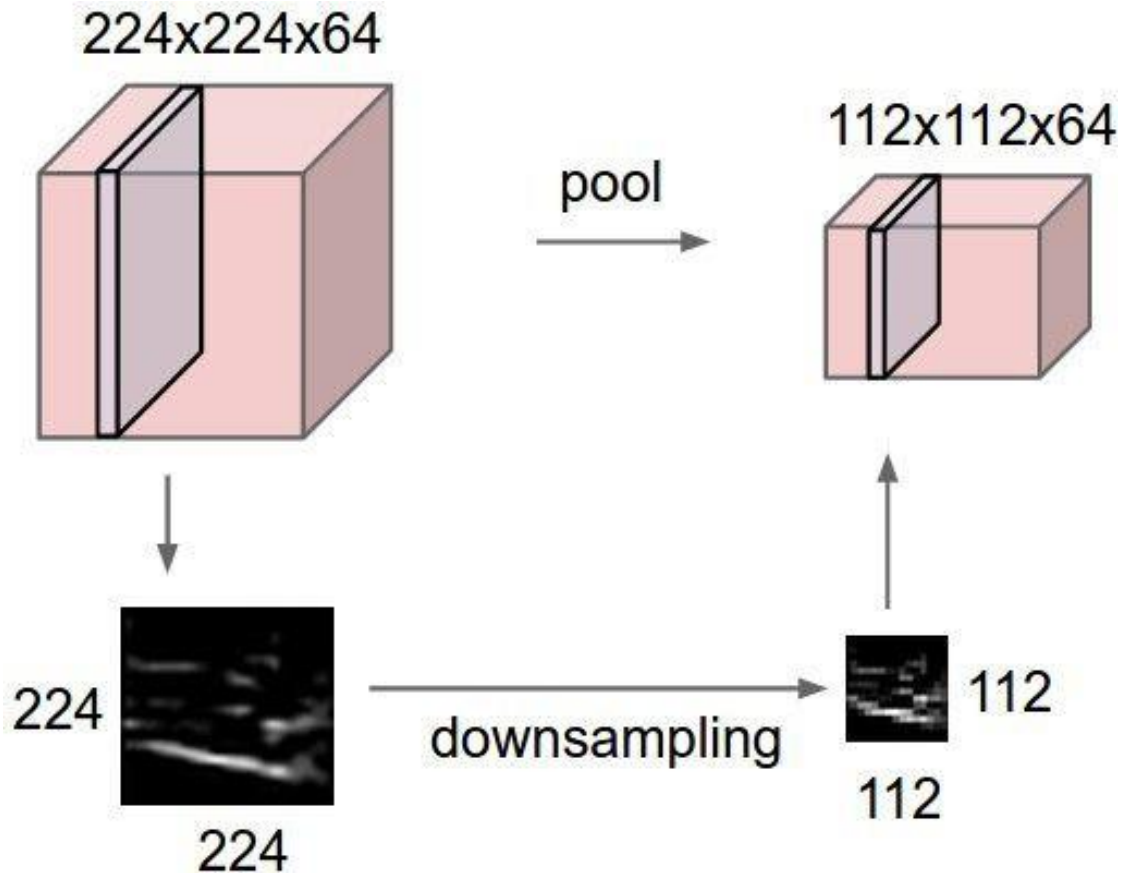
CNN Architecture

- We can get very complex features (textures, faces, objects) from our layers
- How do we balance the increasing layer complexity computationally?



Pooling layer

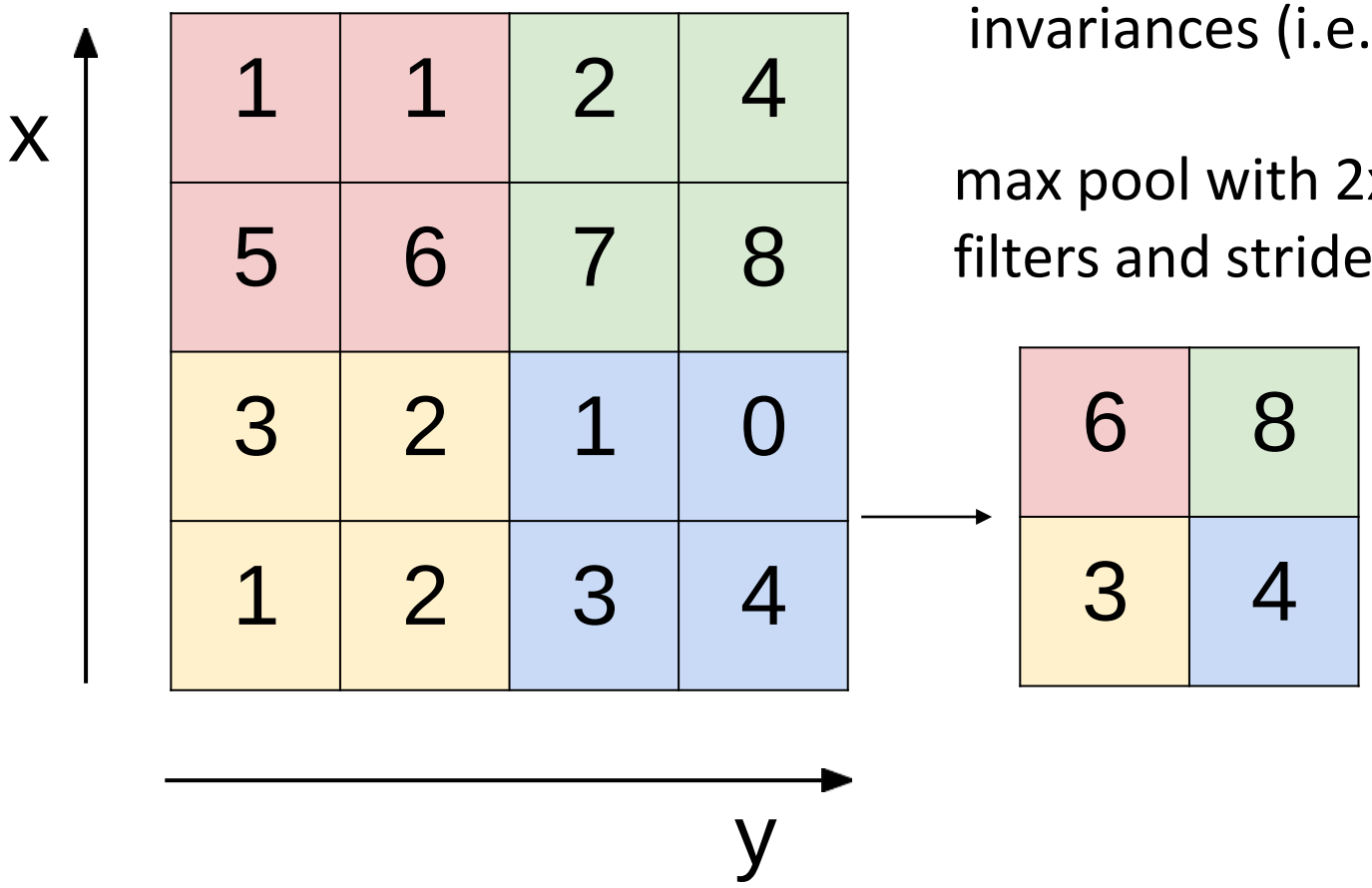
- Makes the representations smaller and more manageable
- Operates over each activation map independently:



Pooling layer

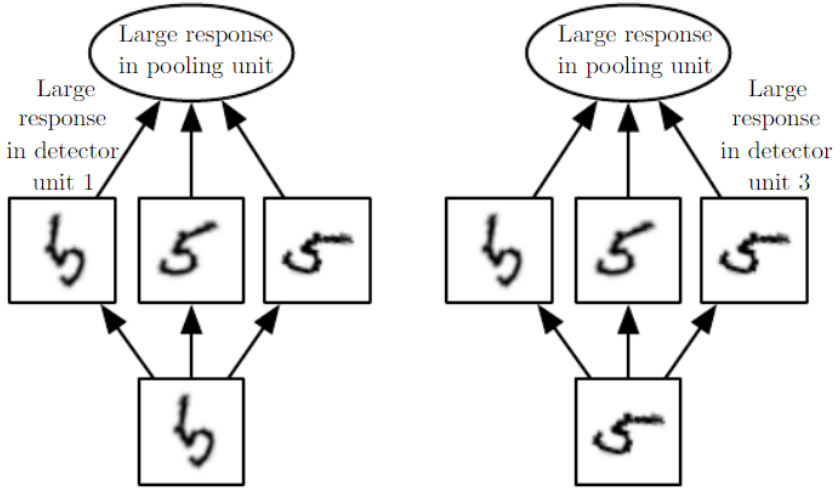
- Max pooling is most common pooling operation
- Allows for “OR”/”any” detection behavior and learned invariances (i.e., rotation invariance)

Single depth slice



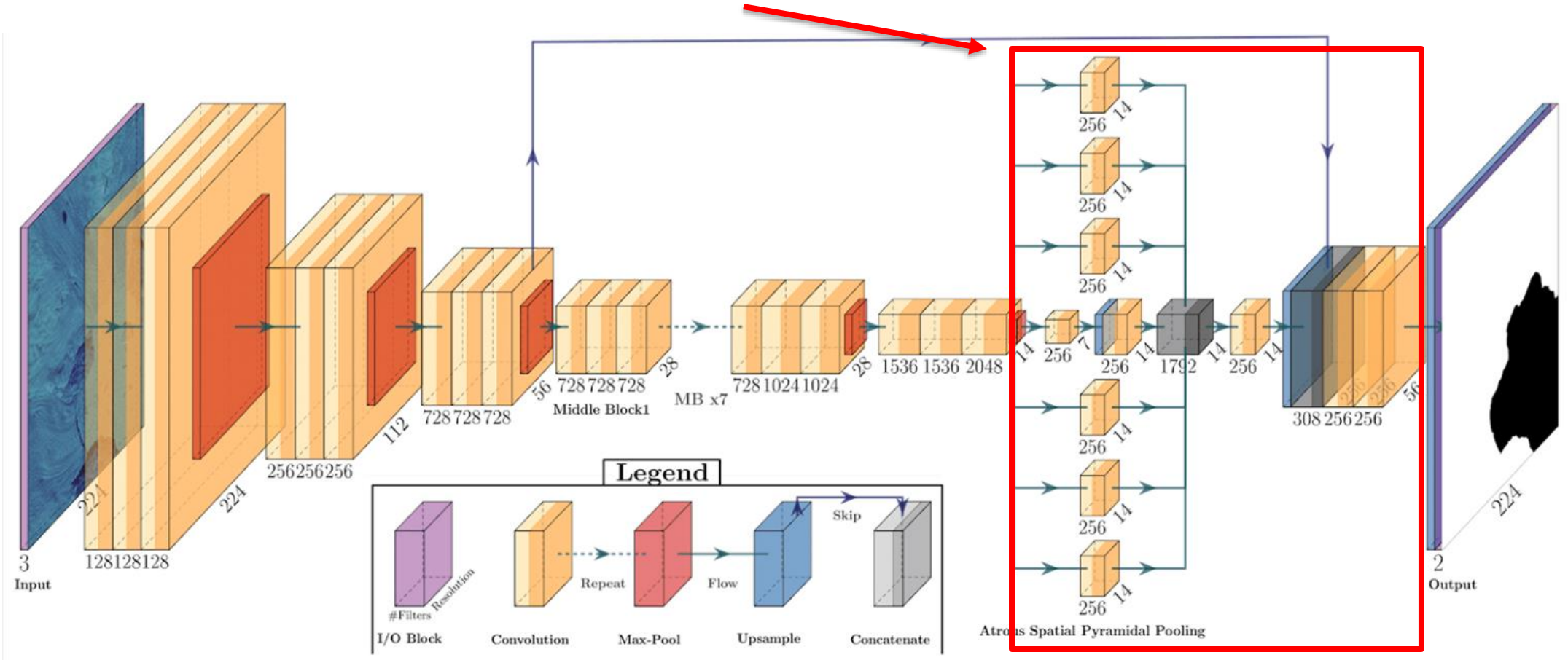
max pool with 2x2 filters and stride 2

6	8
3	4



CNN Architecture

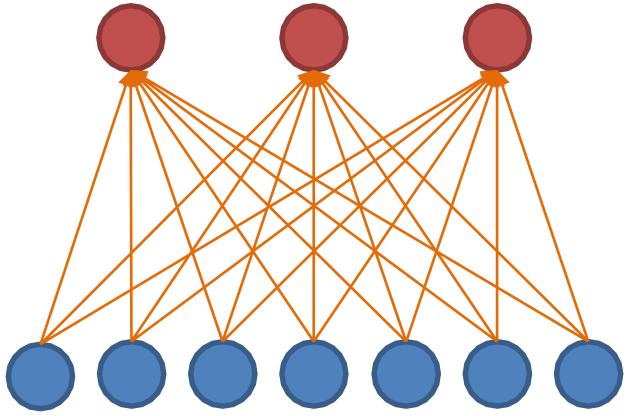
- We have many higher order features.
- How do we classify them?



Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary NNs
- Typically used as final or last stage classifications (expensive to use throughout)
- Spatial Pooling used in conjunction to aid in scale invariant classification

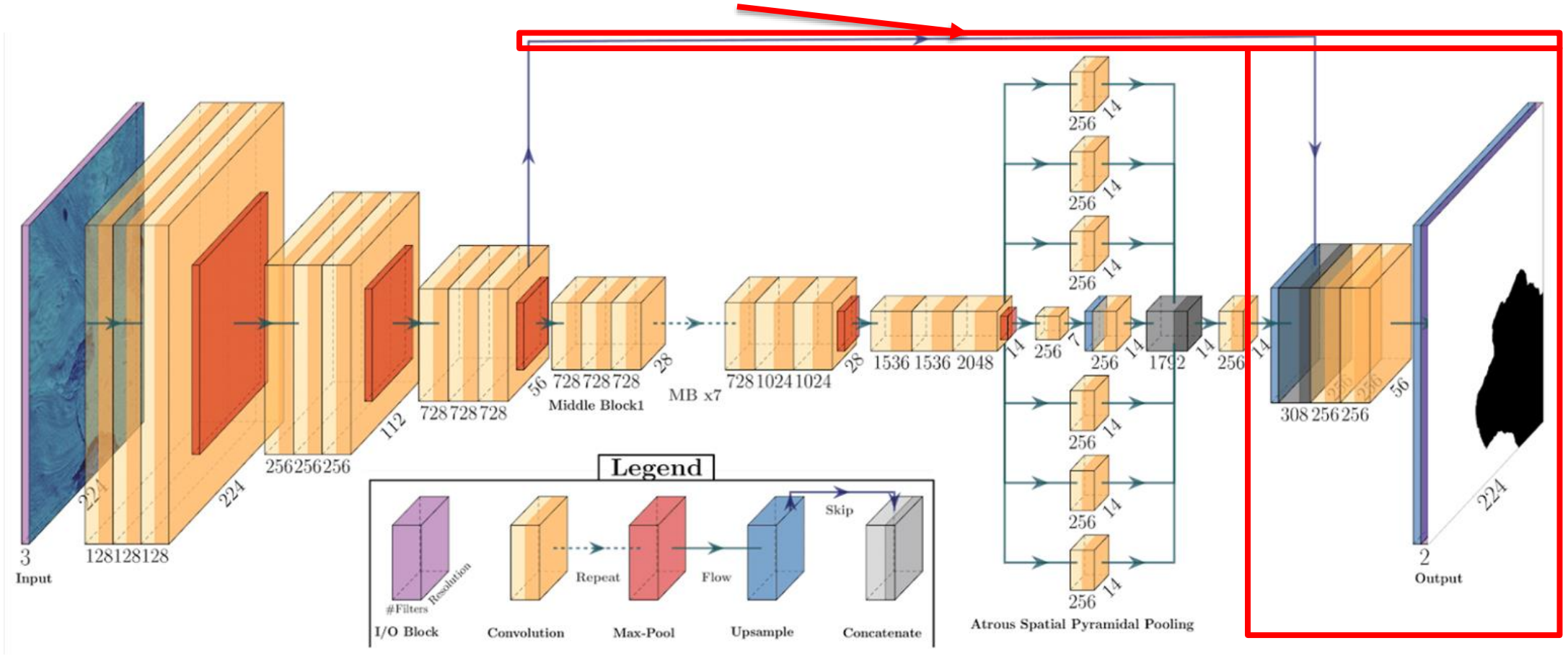
Classifier Score		
Dog	Cat	Glacier
0.1	0.2	0.9



High Level Features						
Fur	Face	Icy	Cold	Ugly	Color	Ugly
0.1	0.2	0.9	1.0	0.2	0.2	0.5

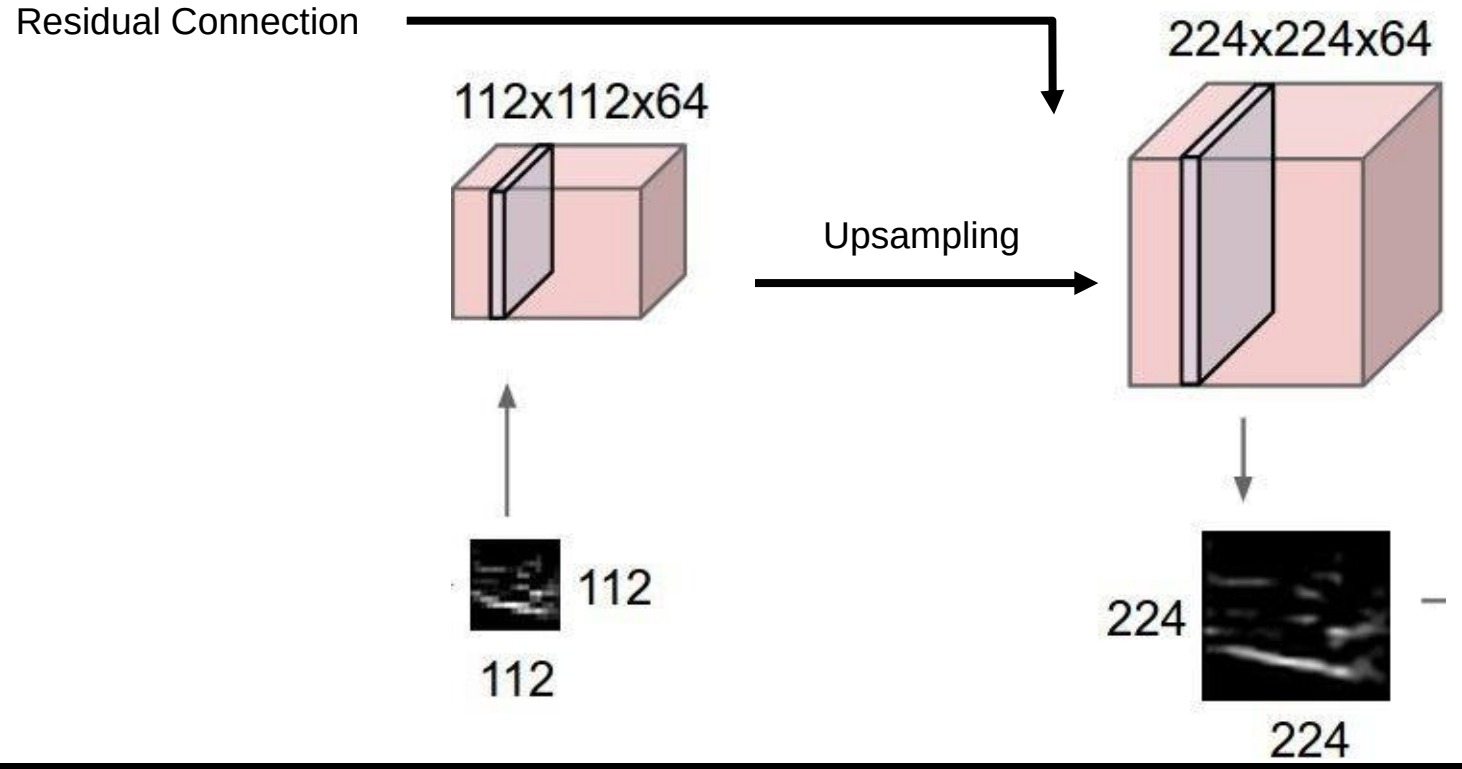
CNN Architecture

- We have many pixel-wise classifications.
- How do we upscale them to their original resolutions?



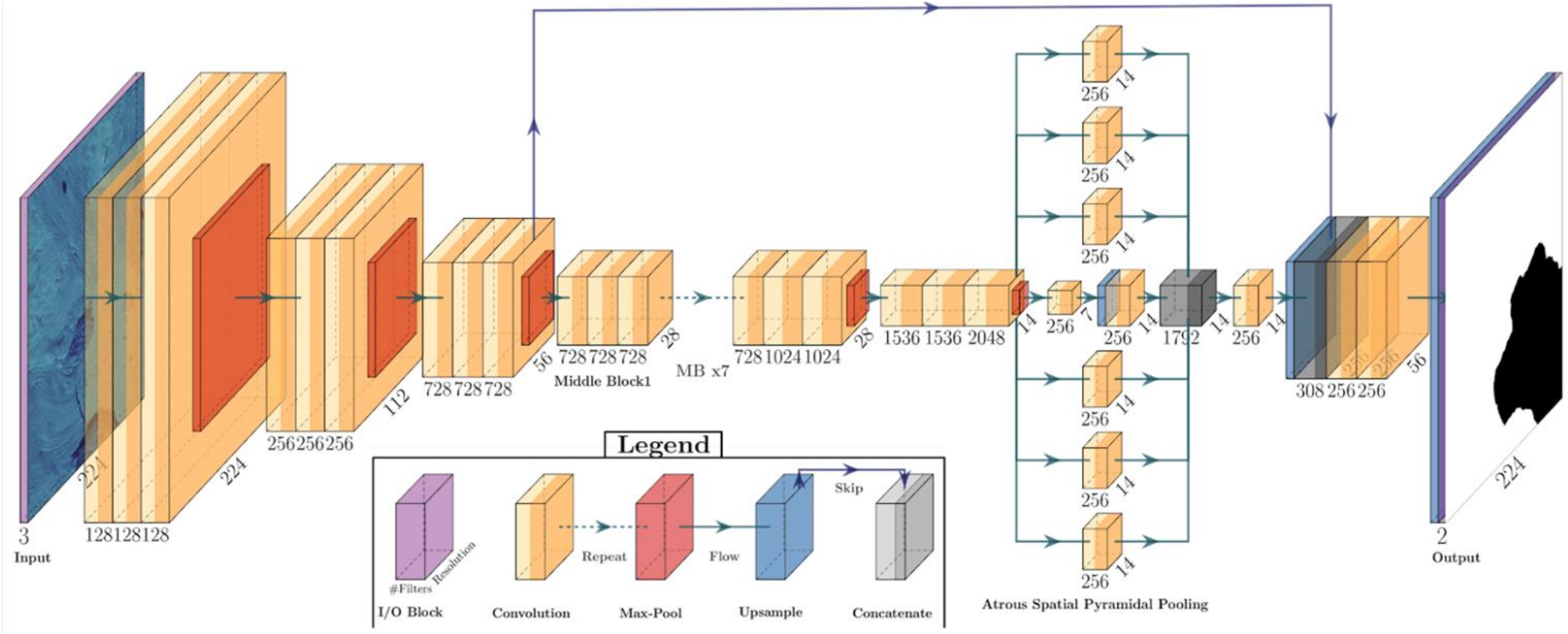
Upsampling + Residual Connections

- Reverse of pooling layer using bilinear interpolation (or similar)
- Residual connections from earlier in the CNNs aid with class localization
- Residuals also reduce vanishing gradients in deep networks
- FC layers can be used to aid in reconstruction, but are not required



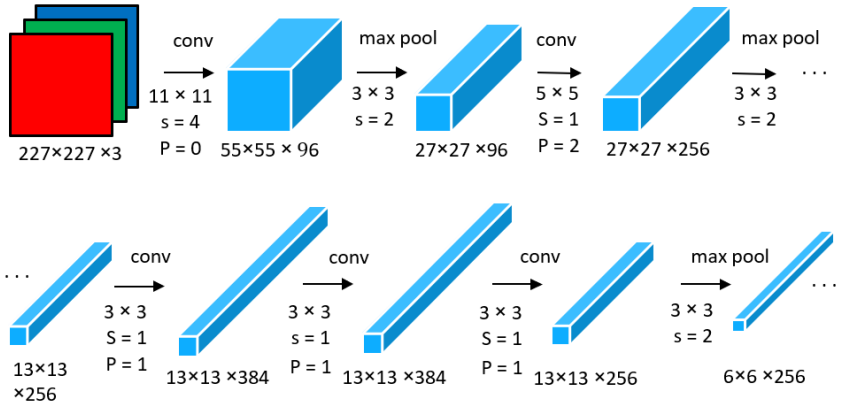
CNN Architecture

- We now have an end-to-end pixel-wise CNN classifier architecture! ☺



CNN Architectures

- We've learned 1 CNN architecture, but what about other CNN architectures, specifically, the feature extractor arhcitectures?
- What are their pros/cons for specific use cases?



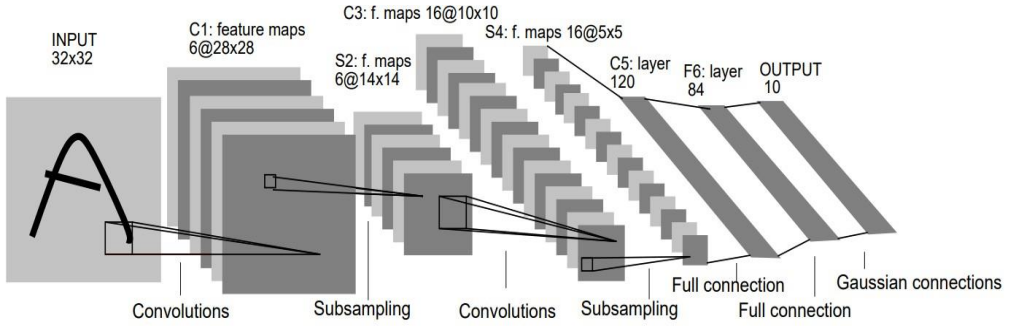
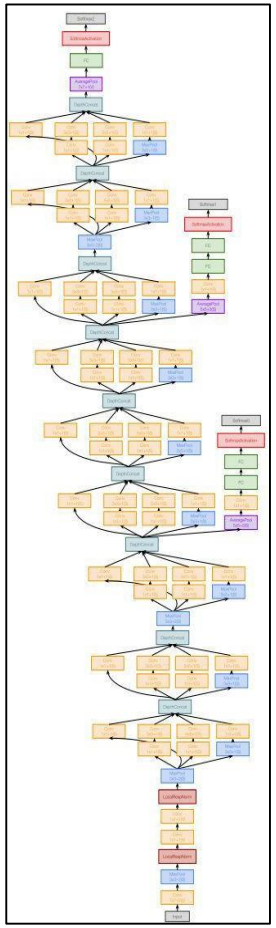
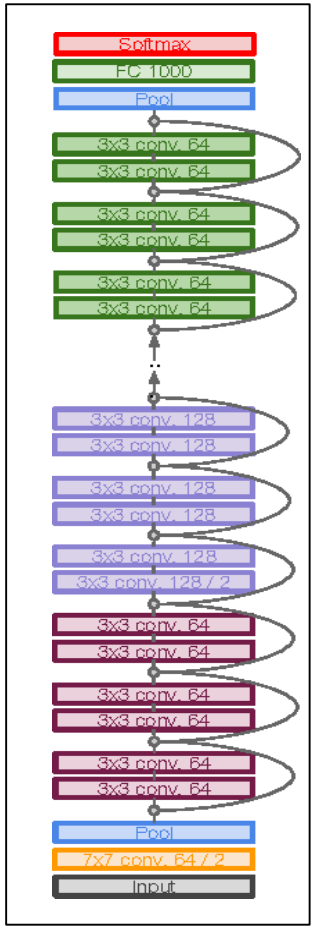


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.





LeNet-5

- *Gradient Based Learning Applied To Document Recognition - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998*
- Helped establish how we use CNNs today
- Replaced manual feature extraction

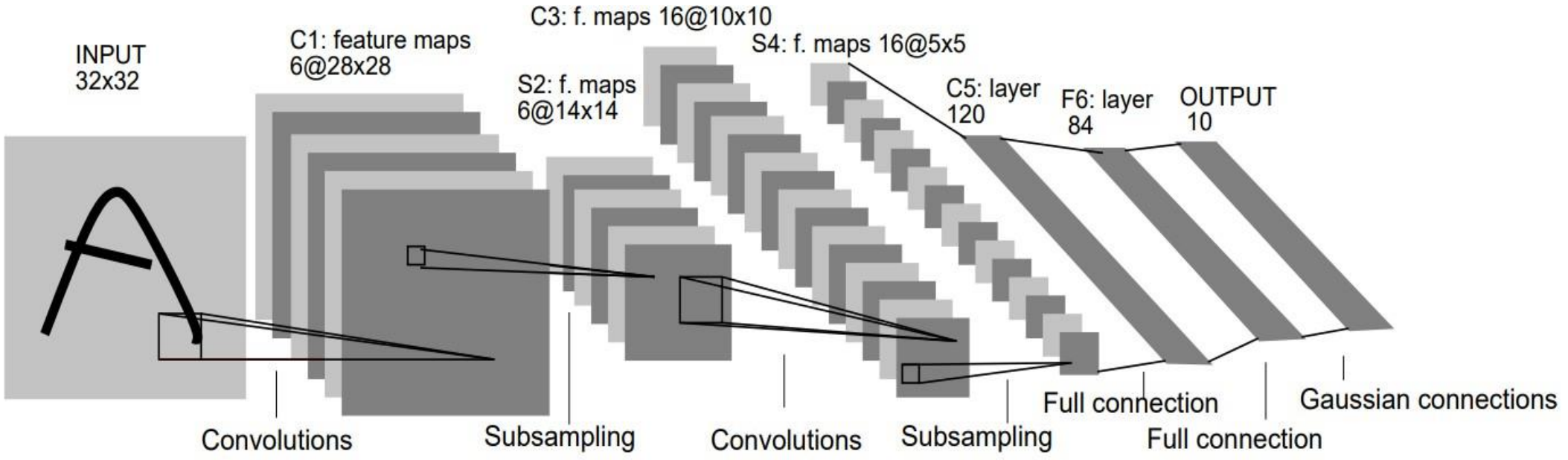
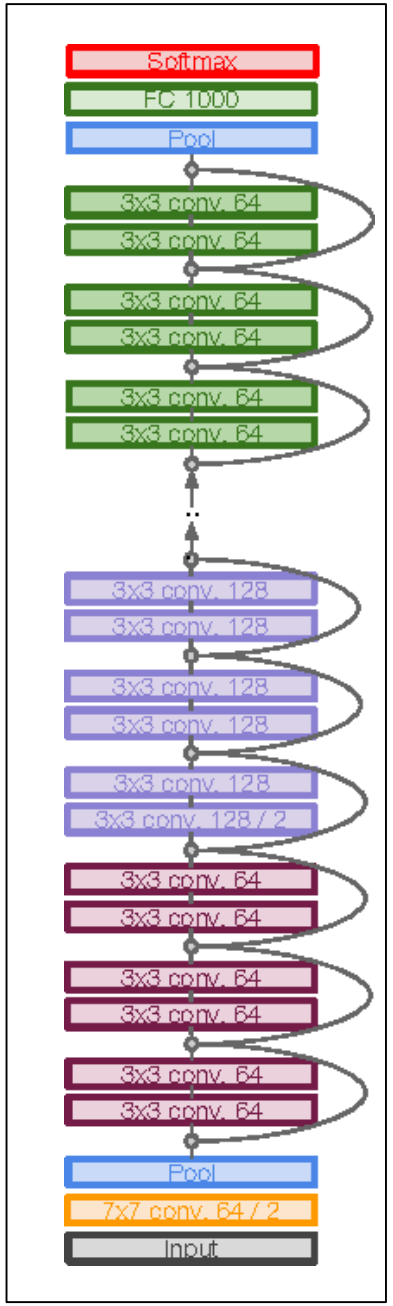


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

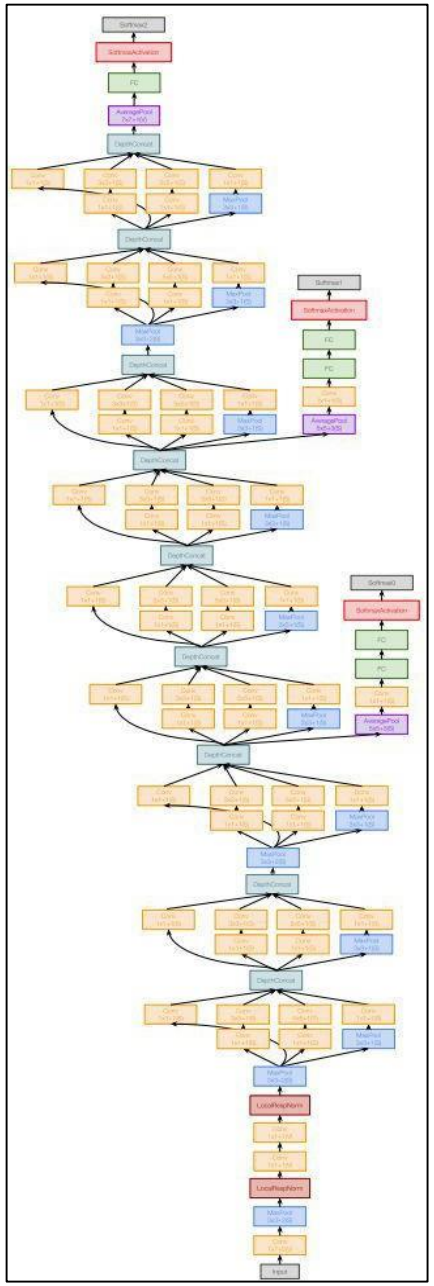
ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
- Makes use of/popularized Residual connections to enable deep neural networks and reduce vanishing gradient problem



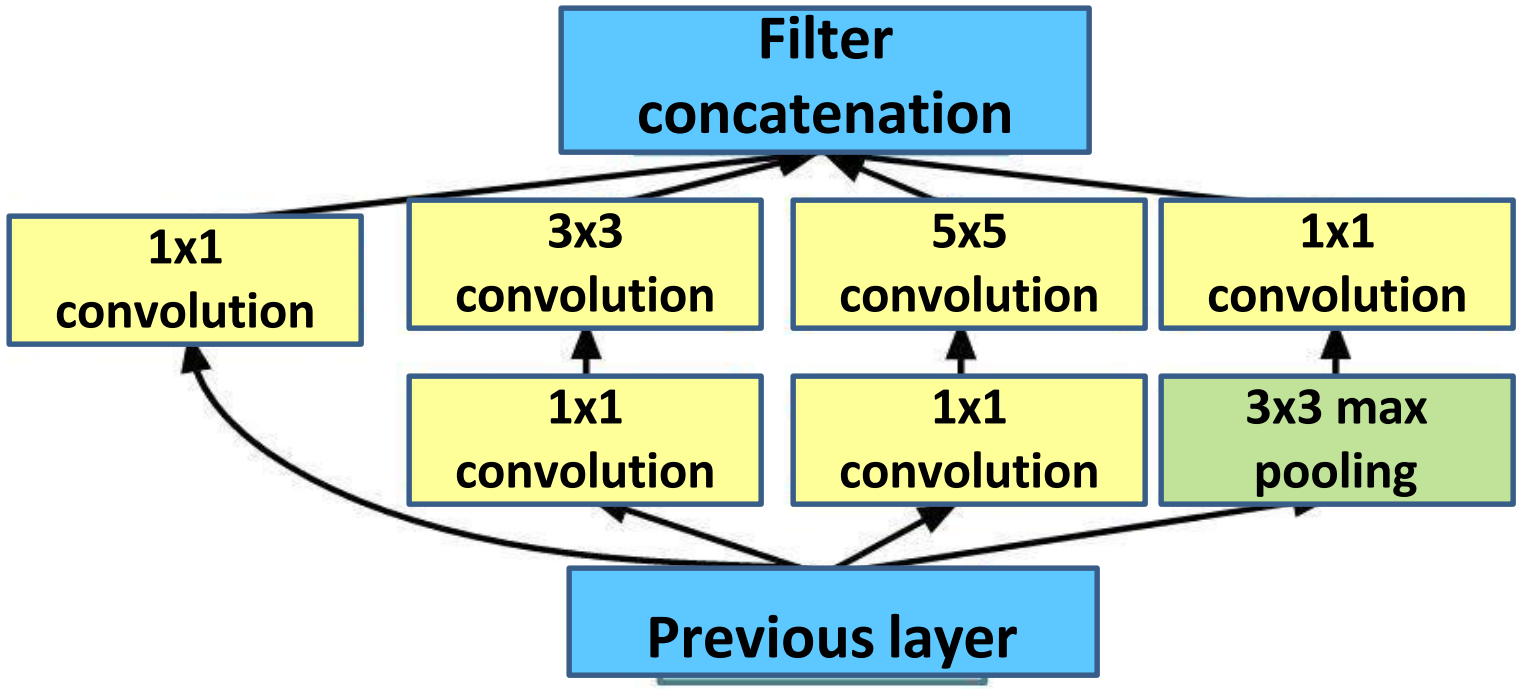
GoogleNet

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



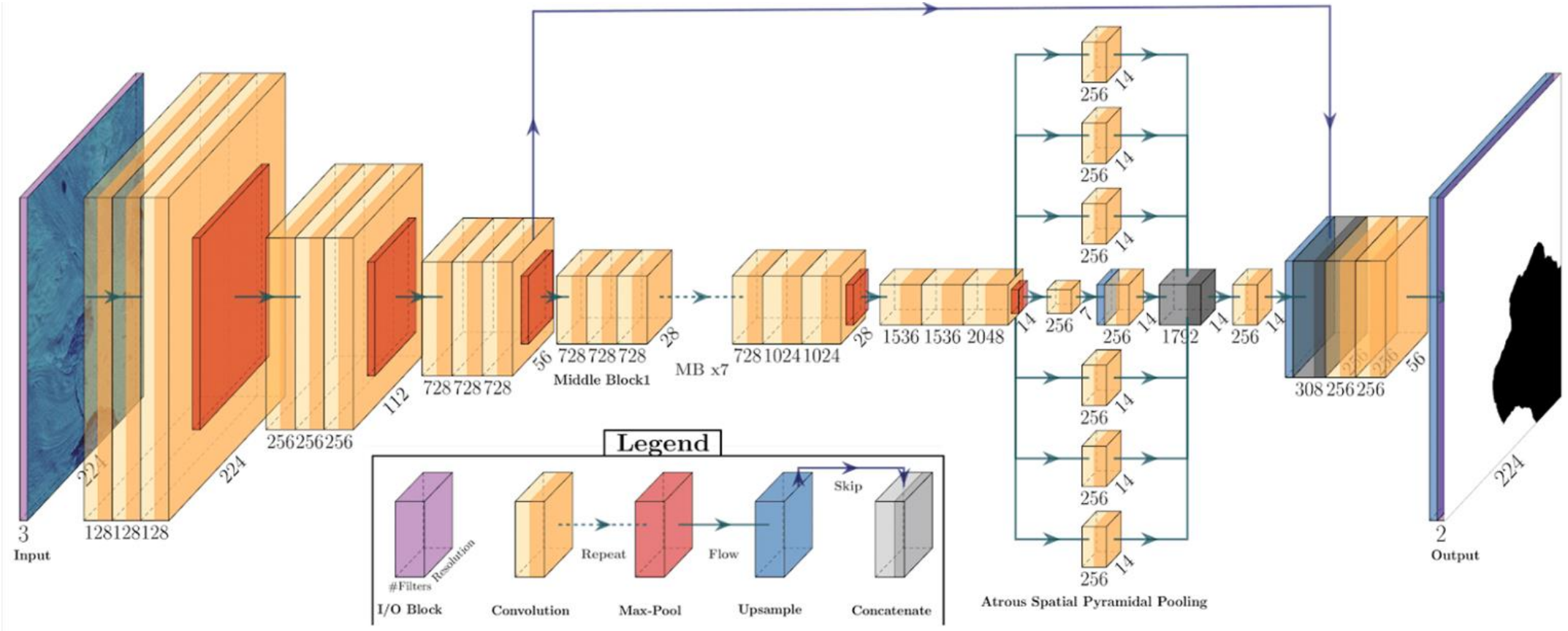
GoogleNet

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



UNet Family

- Well established and useful architecture
- Can use any feature extractor “head”, and outputs classifications per pixel



Outline

Part 1

- Classical machine vision foundations
- Convolutional Neural Networks (CNN) Foundations
- CNN Architectures

Part 2

- **Training CNNs**
- **Understanding and Visualizing CNNs**

Part 3

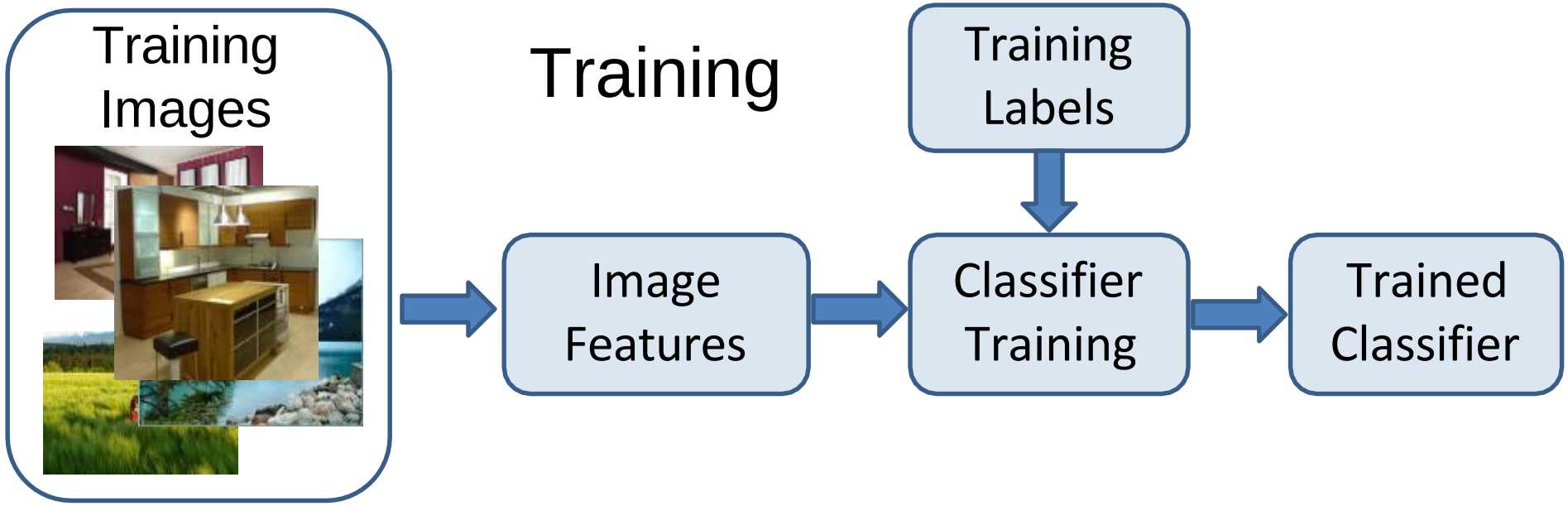
- Applications

Training Neural Networks

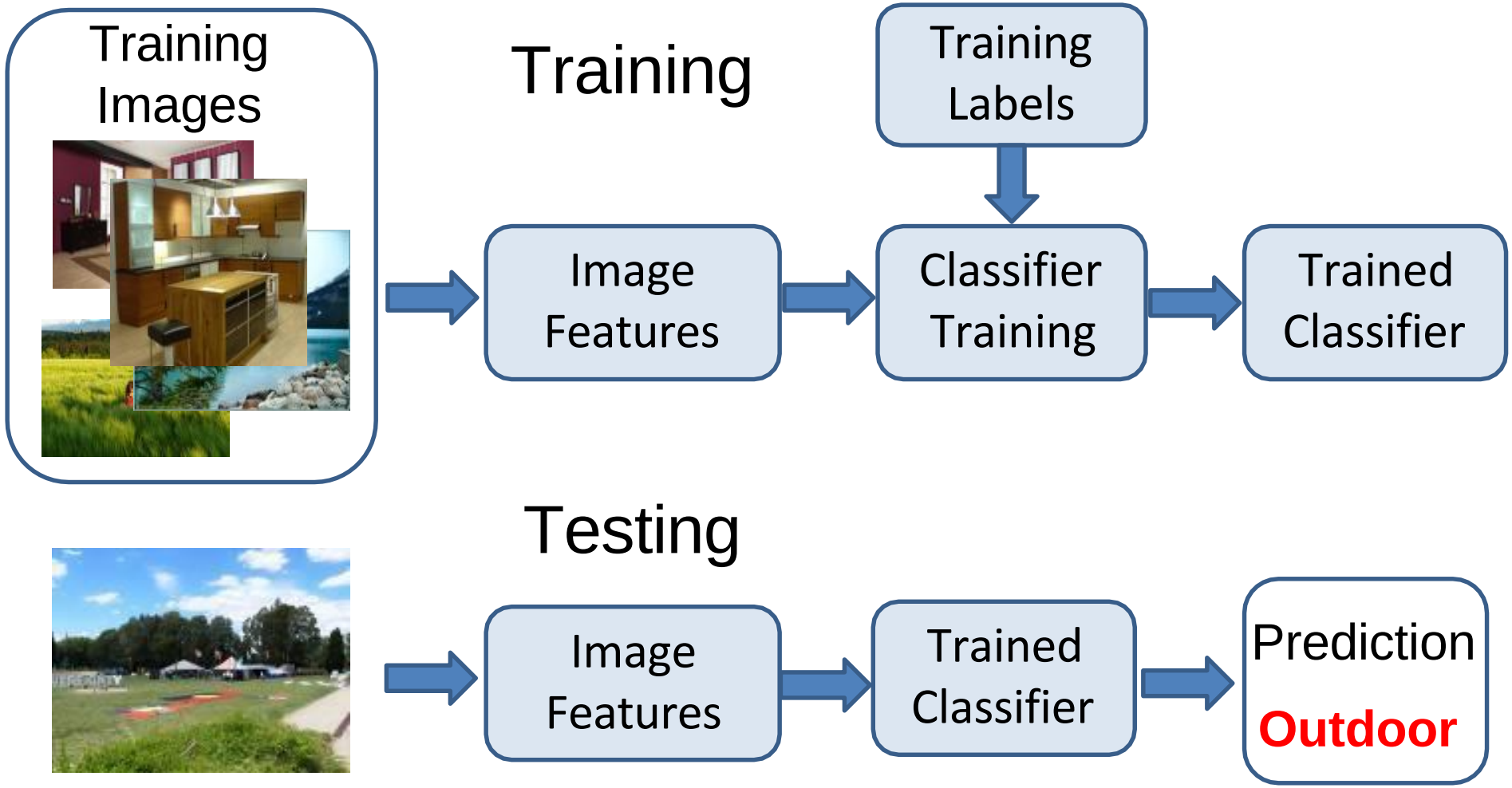
- We have architectures!
- How do we train them?

- Training (& Testing) Workflow
- Backpropagation + stochastic gradient descent with momentum
  - Neural Networks: Tricks of the Trade
- Dropout
- Data augmentation
- Batch normalization
- Initialization
  - Transfer learning

Training Workflow

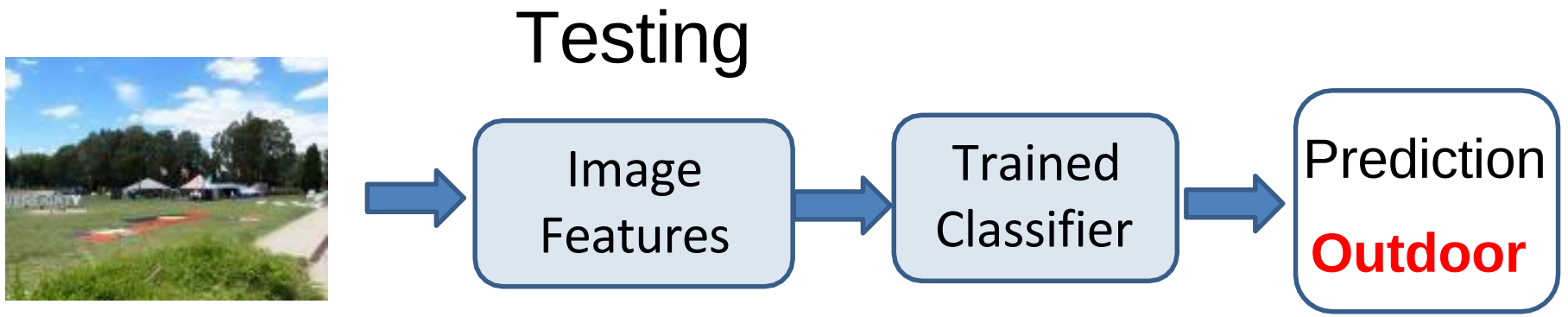


Training (& Testing) Workflow



Training (& Testing) Workflow

- Testing ensures we are not overfitting on the training data/ learned function
- i.e., ensure we can generalize to new unseen data
- Testing is usually also done during training
- This is called validation, and is performed on 10-20% of reserved training data to ensure generalization while training



Training CNN with gradient descent

- A CNN as composition of functions

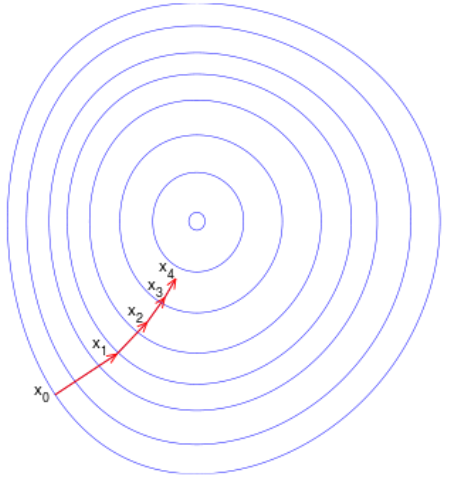
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = f_L(\dots\ (f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2)\ \dots; \boldsymbol{w}_L)$$

- Parameters

$$\boldsymbol{w} = (\boldsymbol{w_1}, \boldsymbol{w_2}, \dots \boldsymbol{w_L})$$

- Empirical loss function

$$L(\boldsymbol{w}) = \frac{1}{n} \sum_i l(z_i, f_{\boldsymbol{w}}(\boldsymbol{x_i}))$$



- Gradient descent (w/ optimizers such as Adam)

$$\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta_t \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}(\boldsymbol{w^t})$$

New weight

Old weight

Learning rate

Gradient

An Illustrative example

$$f(x, y) = xy, \qquad \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x$$

Example:

$$x = 4, y = -3 \Rightarrow f(x, y) = -12$$

Partial derivatives

$$\frac{\partial f}{\partial x} = -3, \qquad \frac{\partial f}{\partial y} = 4$$

Gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

$$f(x, y, z) = (x + y) \quad z = qz$$

$q = x + y$

$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$

$f = qz$

$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$

Goal: compute the gradient

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]$$

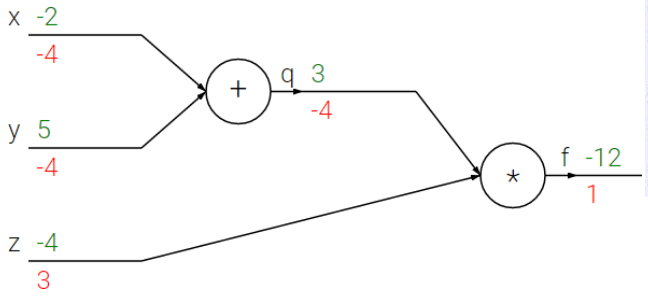
$$f(x, y, z) = (x + y) \quad z = qz$$

$$q = x + y$$
$$\frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz$$
$$\frac{\partial f}{\partial q} = z, \qquad \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

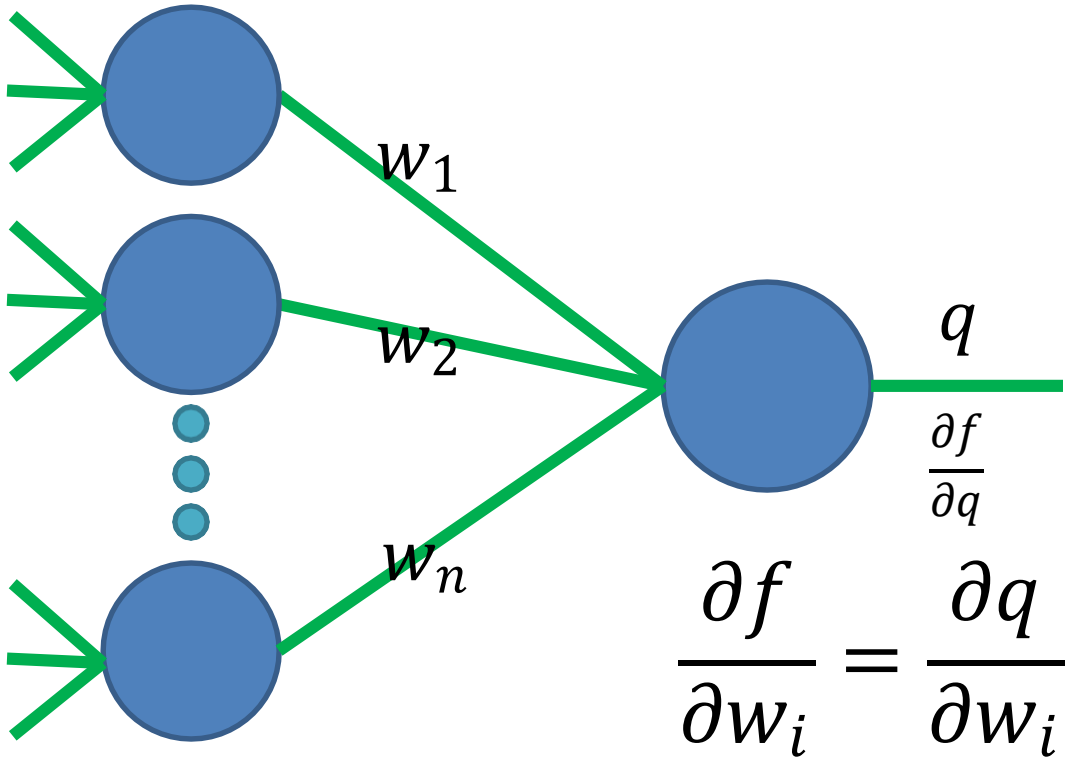



```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

Backpropagation (recursive chain rule)



$$\frac{\partial f}{\partial w_i} = \frac{\partial q}{\partial w_i}\frac{\partial f}{\partial q}$$

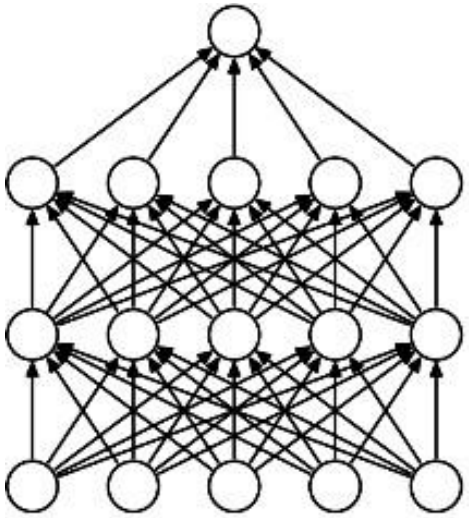
Local gradient

Gate gradient

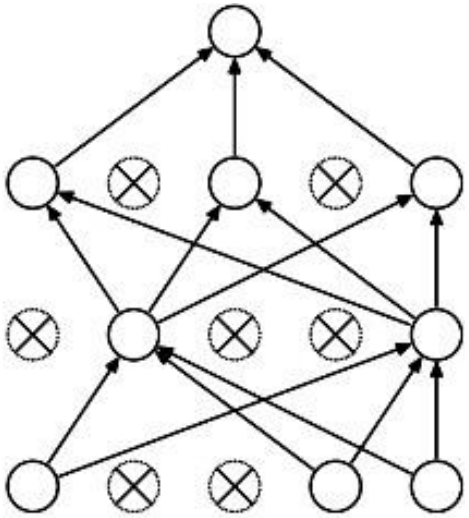
Can be computed during forward pass

The gate receives this during backprop

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

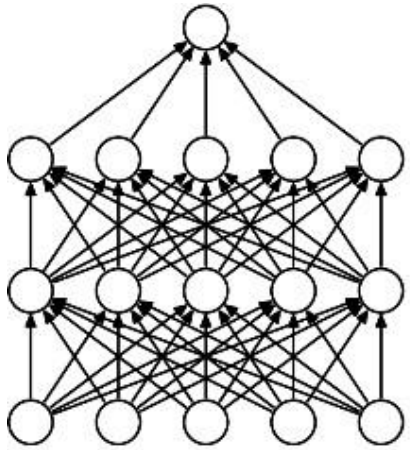
Intuition: successful conspiracies

- 50 people planning a conspiracy

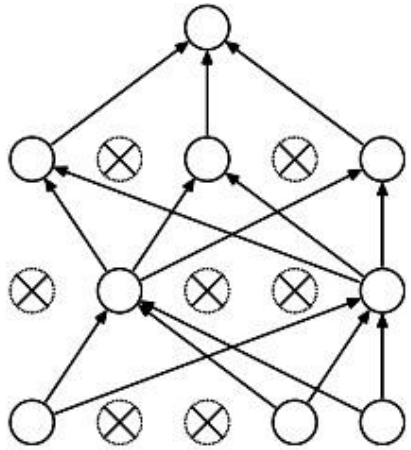
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.

- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

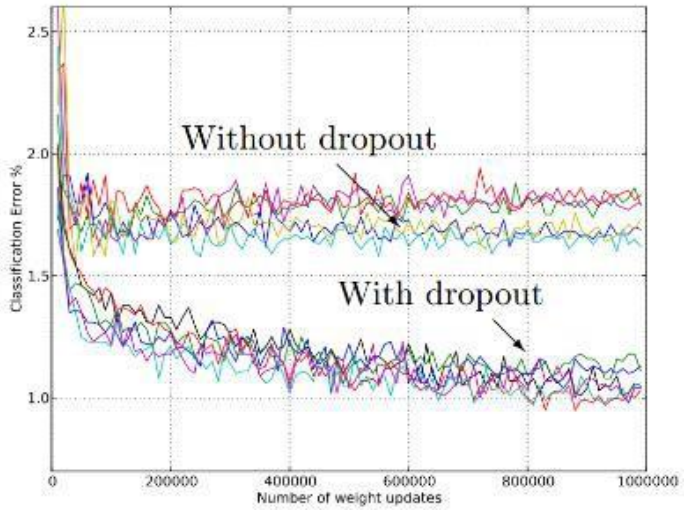
Dropout

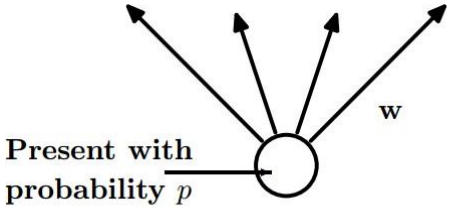


(a) Standard Neural Net

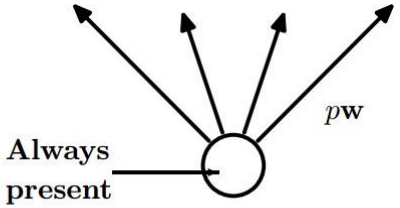


(b) After applying dropout.





(a) At training time



(b) At test time

Main Idea: approximately combining exponentially many different neural network architectures efficiently

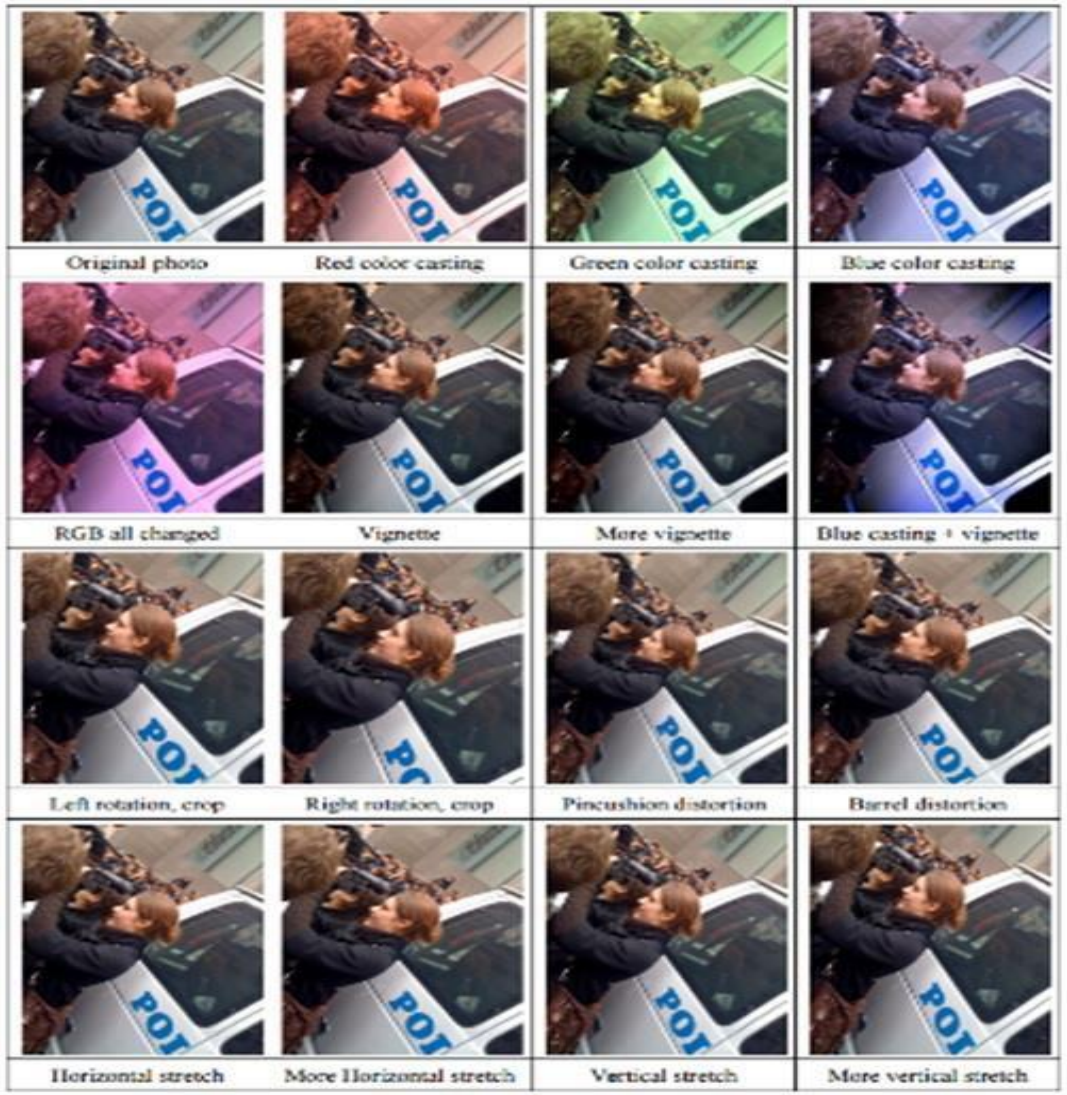
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

Data Augmentation

- Manually created training data is expensive/overfitting
- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Contrast/Brightness enhancement
  - Geometric distortion*



Deep Image [Wu et al. 2015]

Batch Normalization

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

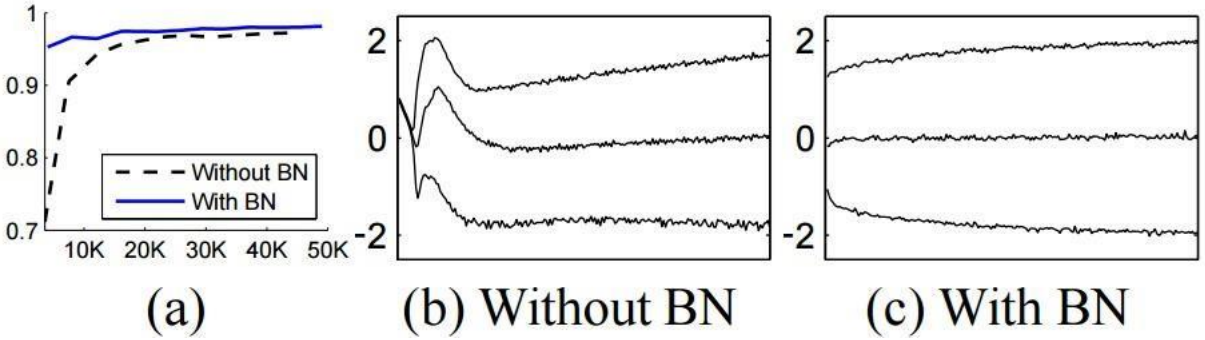
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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



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

Main Idea: Allow for stochastic gradient descent within computational limits, since most GPUs can handle more than one but less than all of the training data (i.e., m ~= 2-32)

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

Understanding and Visualizing CNNs

- We've trained our CNN
- How do we know what's going on inside?

- Find images that maximize some class scores
- Individual neuron activation

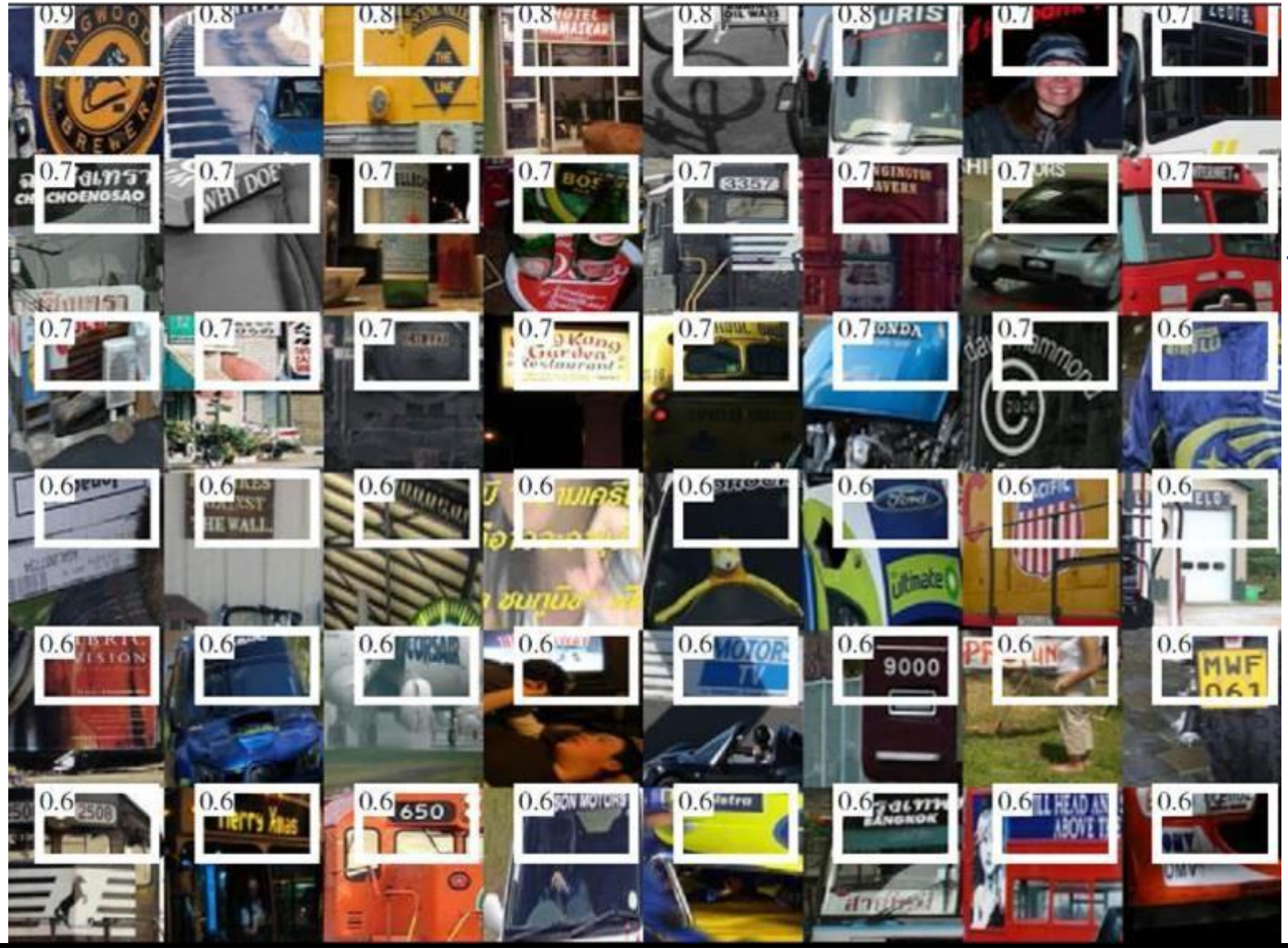
Individual Neuron Activation

RCNN [Girshick et al. CVPR 2014]

Individual Neuron Activation

RCNN [Girshick et al. CVPR 2014]

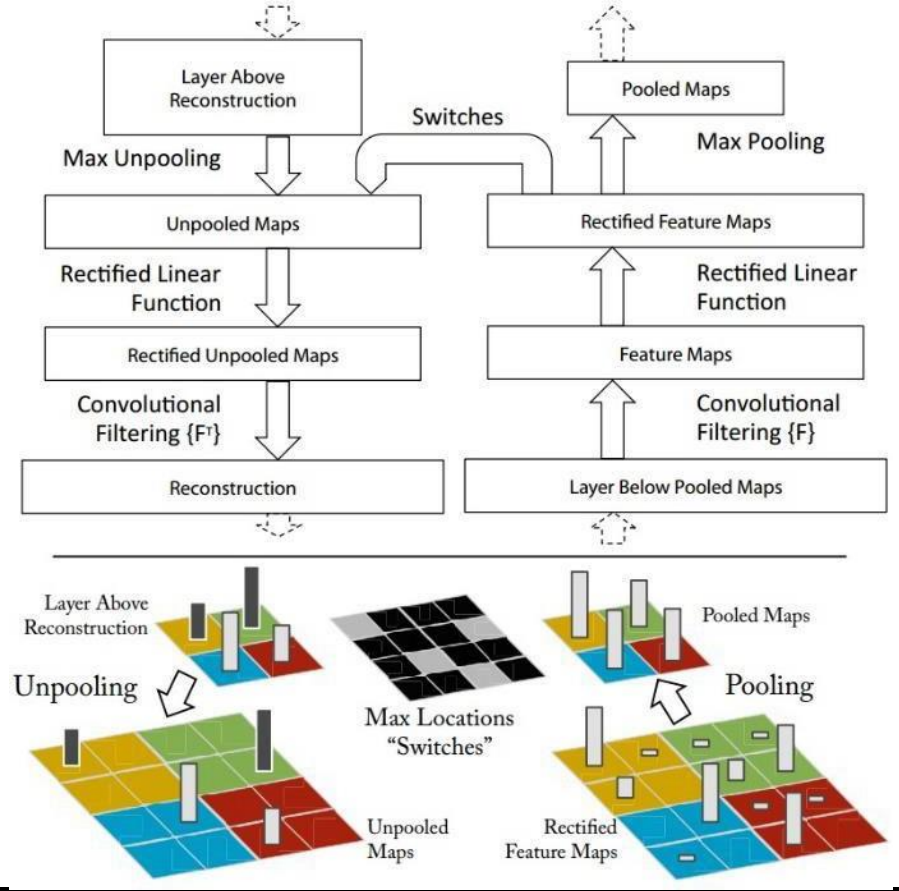
Individual Neuron Activation



RCNN [Girshick et al. CVPR 2014]

Map activation back to the input pixel space

- What input pattern originally caused a given activation in the feature maps?

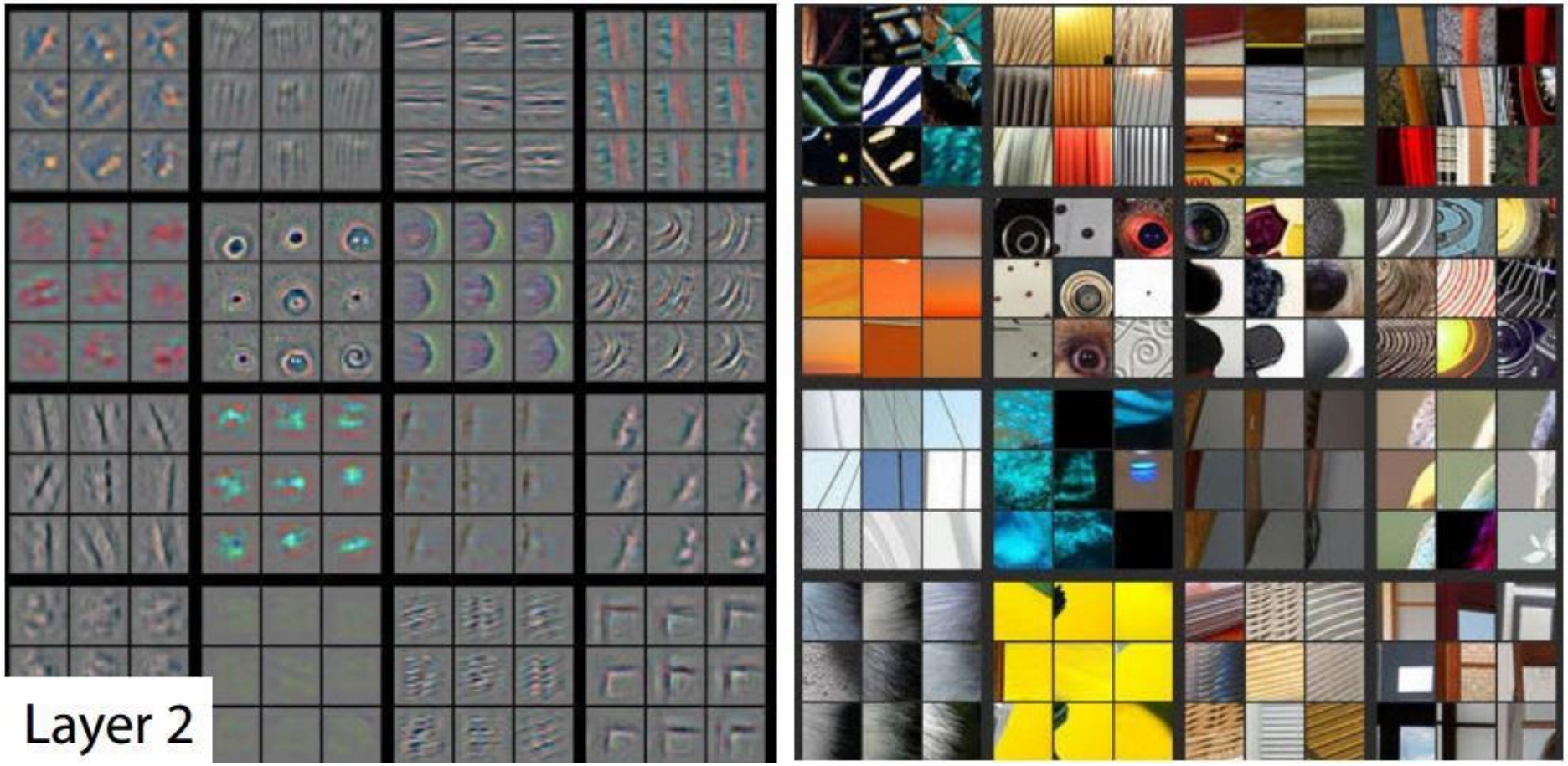


Zeiler and Fergus, ECCV 2014

Layer 1

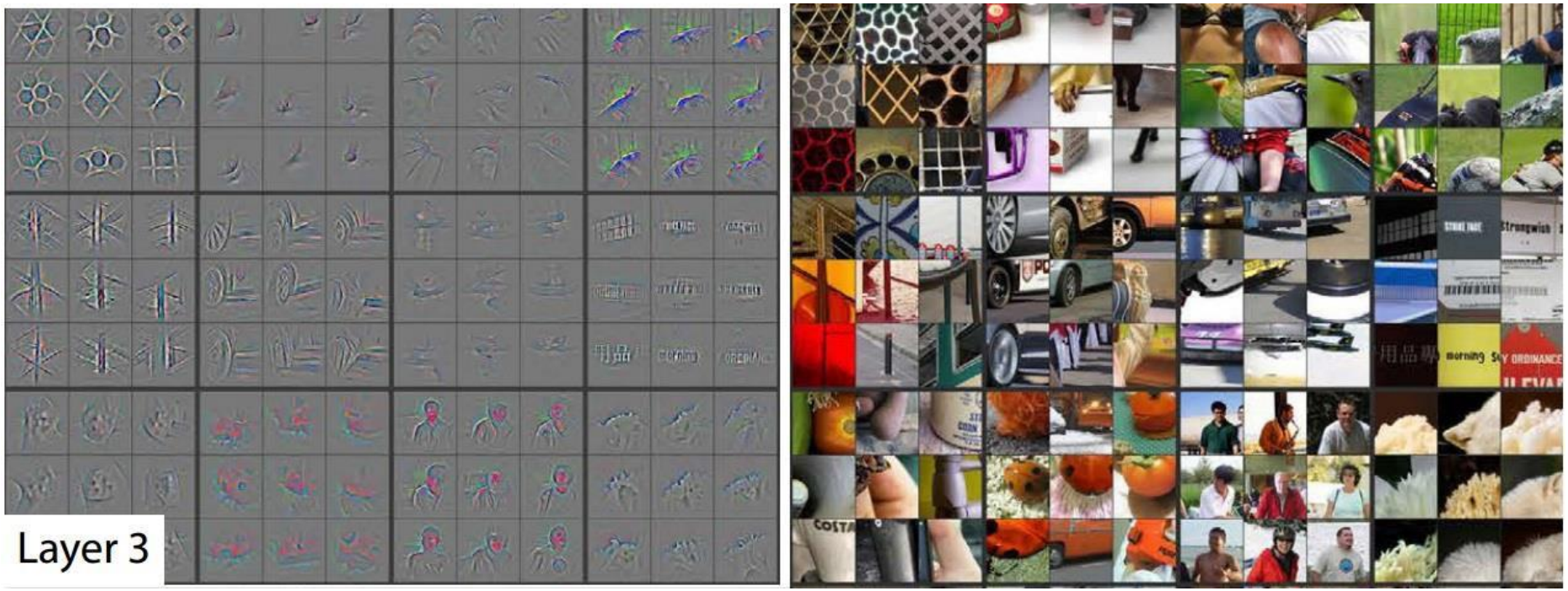
Zeiler and Fergus, ECCV 2014

Layer 2



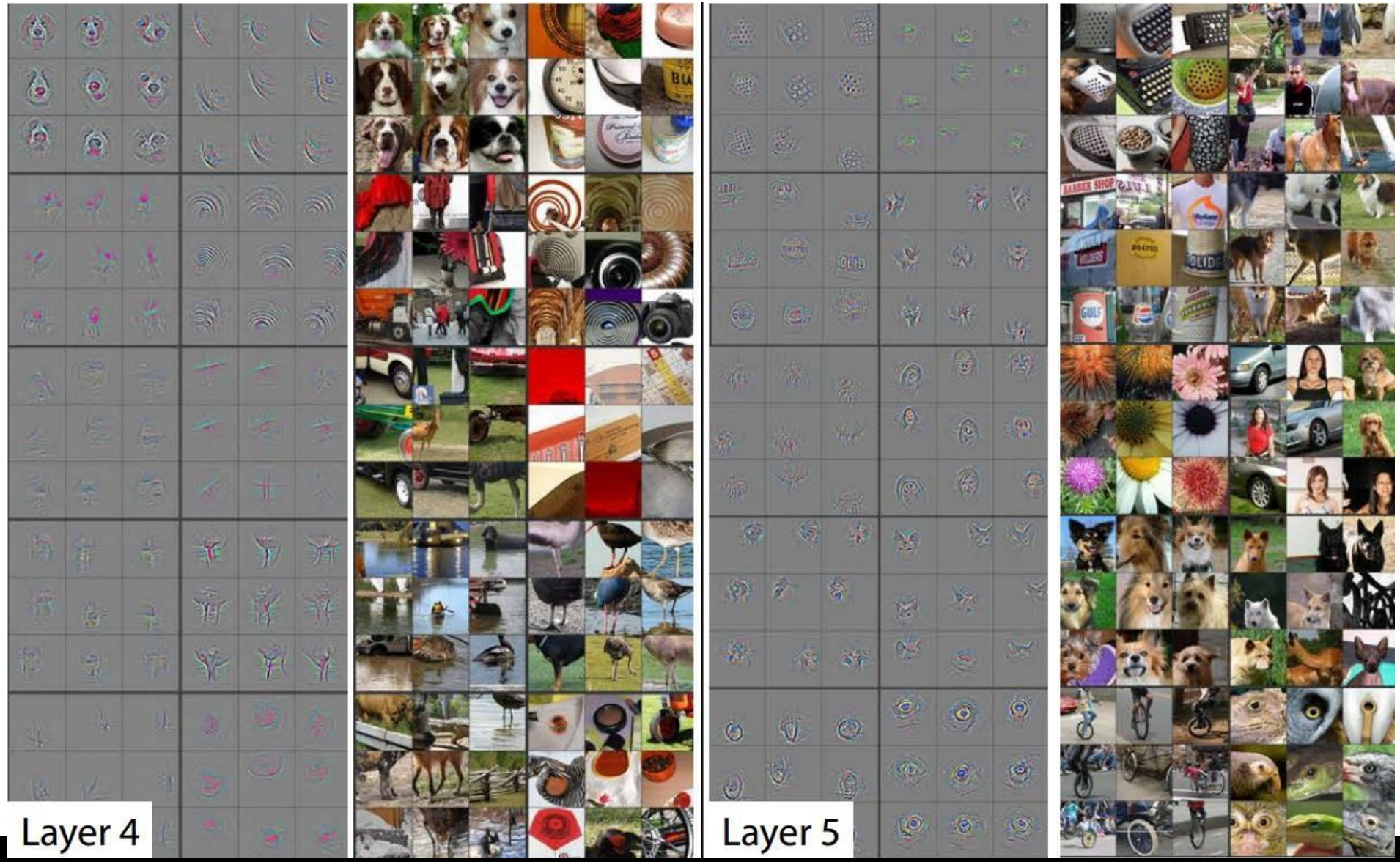
Zeiler and Fergus, ECCV 2014

Layer 3



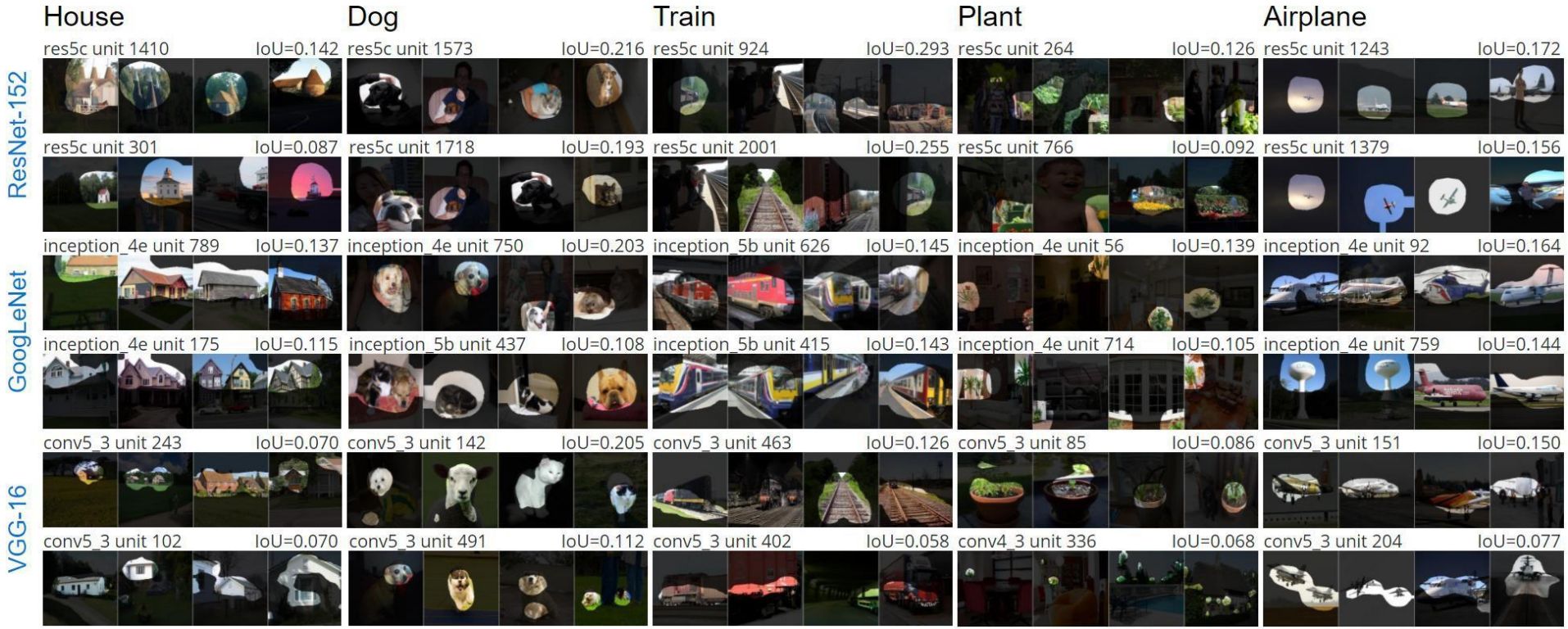
Zeiler and Fergus, ECCV 2014

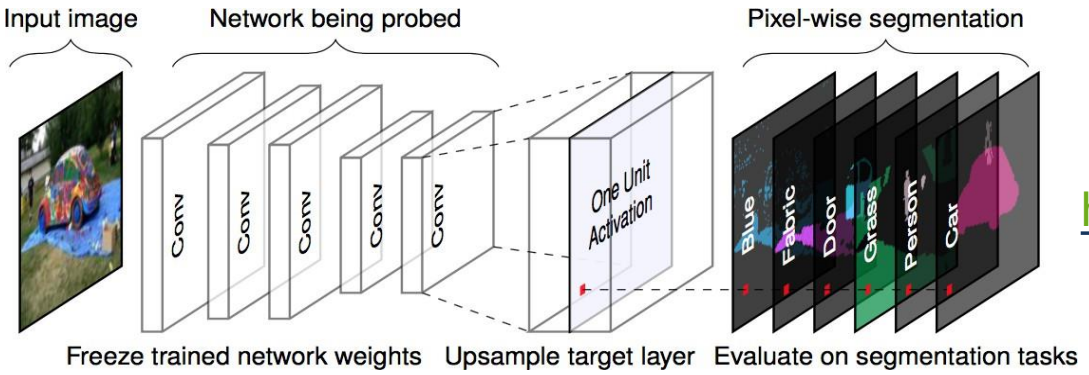
Layer 4 and 5



Zeiler and Fergus, ECCV 2014

Network Dissection





http://netdissect.csail.mit.edu/

Things to remember

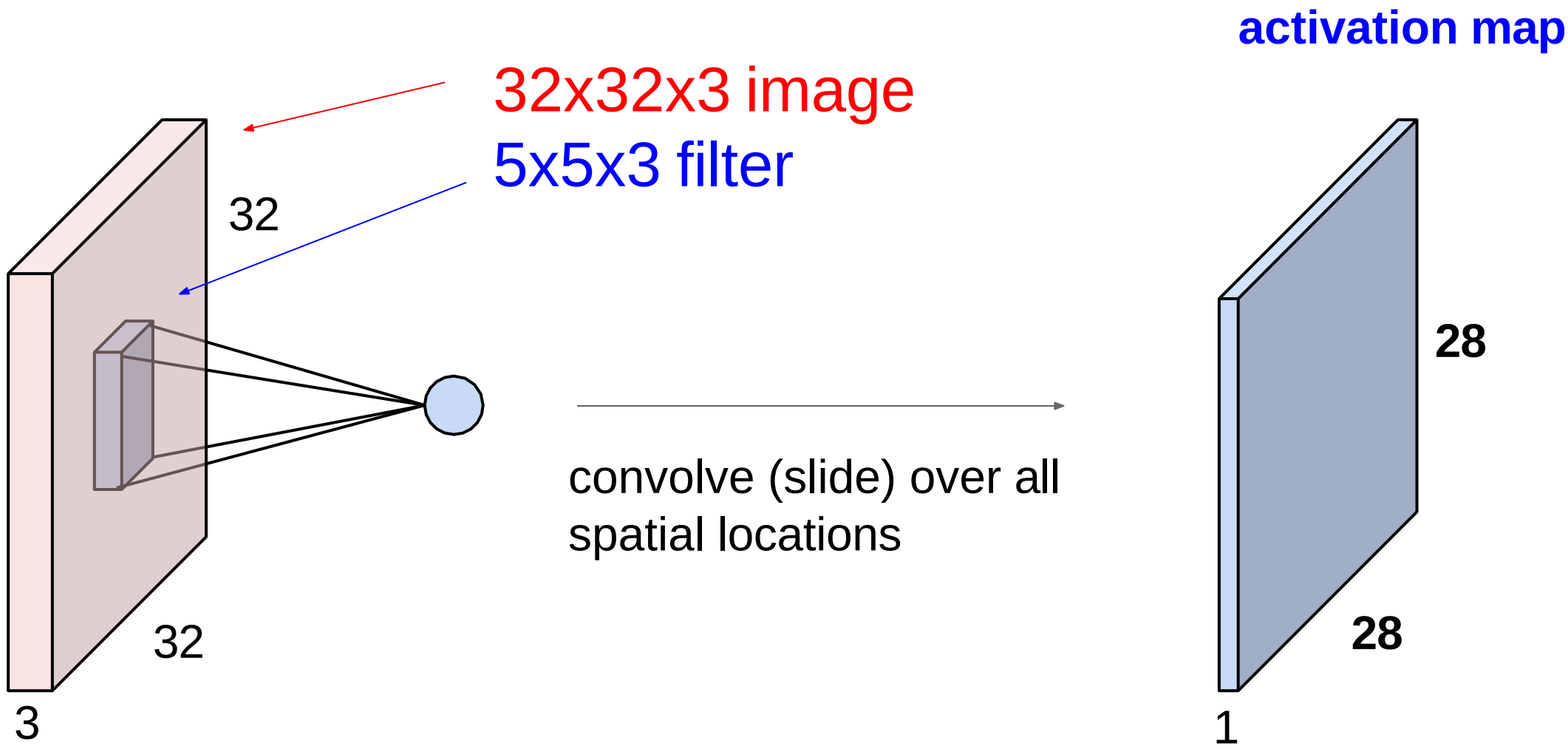
- Convolutional neural networks
  - A cascade of conv + ReLU + pool
  - Automatic feature learning
  - Numerous architectures
  - Tricks for training CNNs
- Visualizing CNNs
  - Activation mapping
  - Dissection

Resources

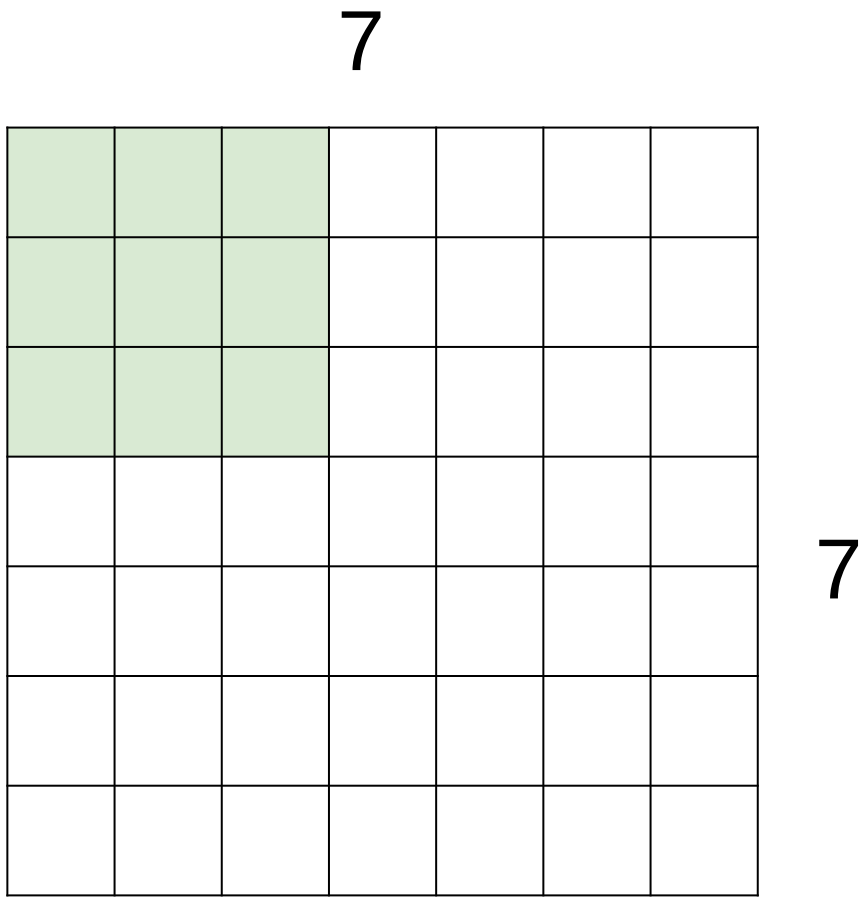
- http://deeplearning.net/
  - Hub to many other deep learning resources
- https://github.com/ChristosChristofidis/awesome-deep-learning
  - A resource collection deep learning
- https://github.com/kjw0612/awesome-deep-vision
  - A resource collection deep learning for computer vision
- http://cs231n.stanford.edu/syllabus.html
  - Nice course on CNN for visual recognition

Additional Slides
Implementation Details/Errata

A closer look at spatial dimensions:



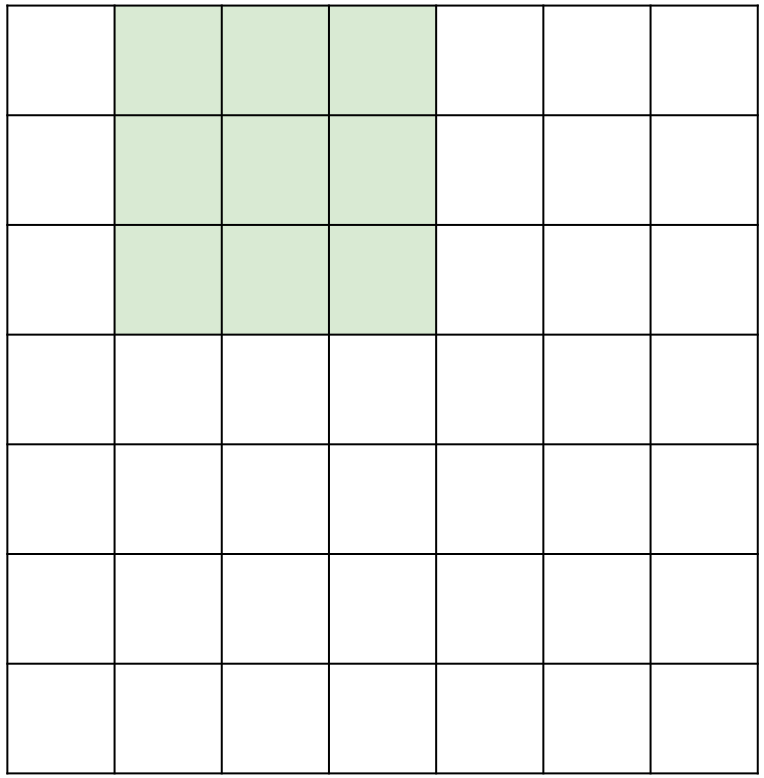
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

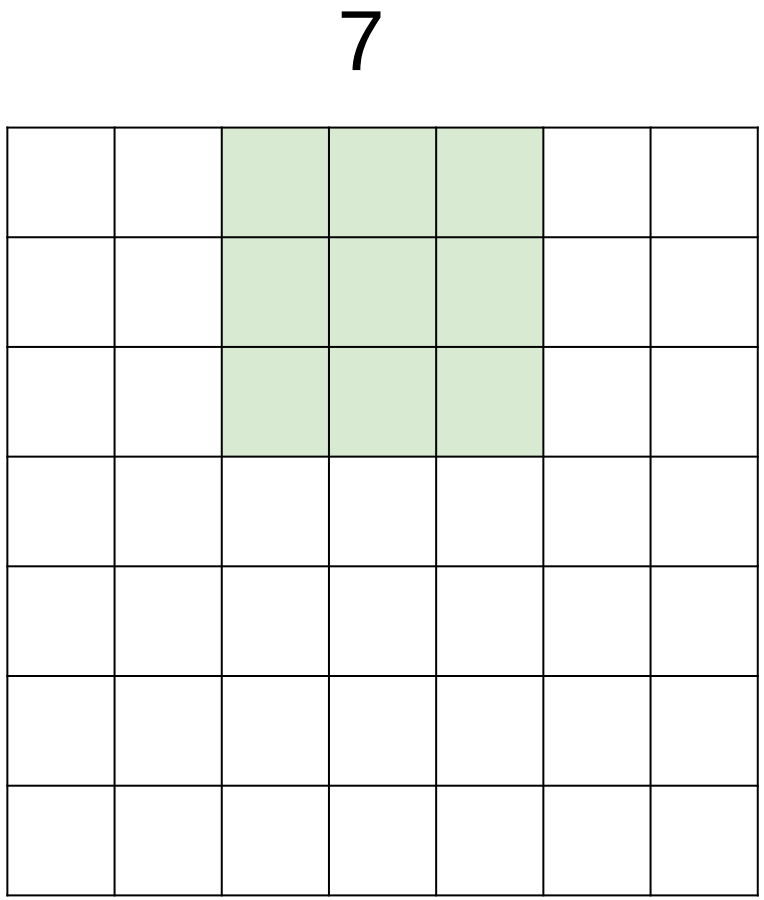
7



7

7x7 input (spatially)
assume 3x3 filter

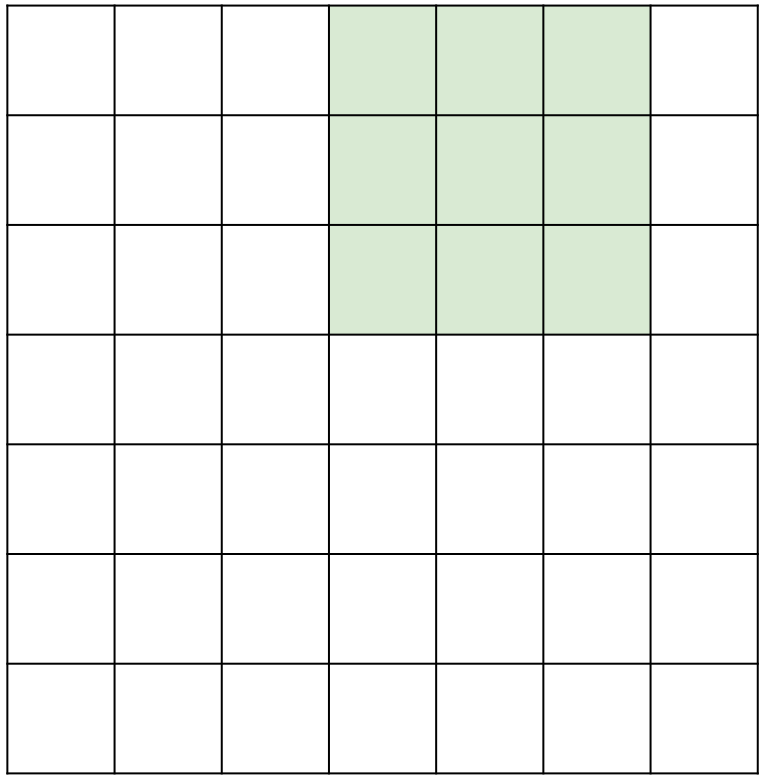
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:

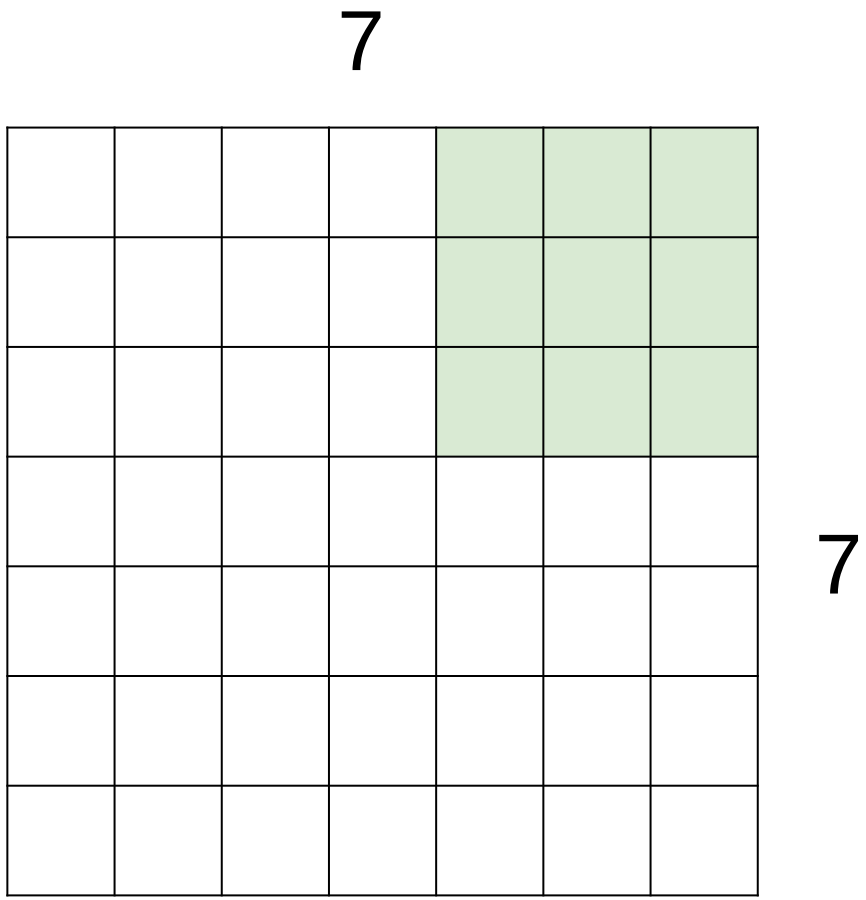
7



7

7x7 input (spatially)
assume 3x3 filter

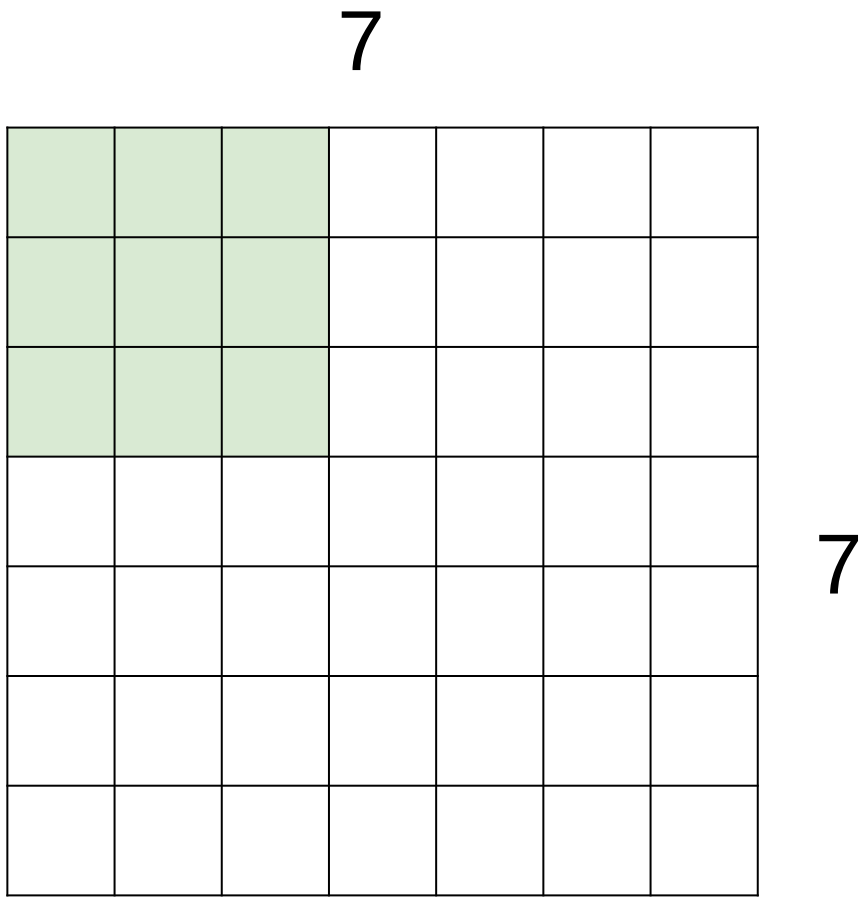
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

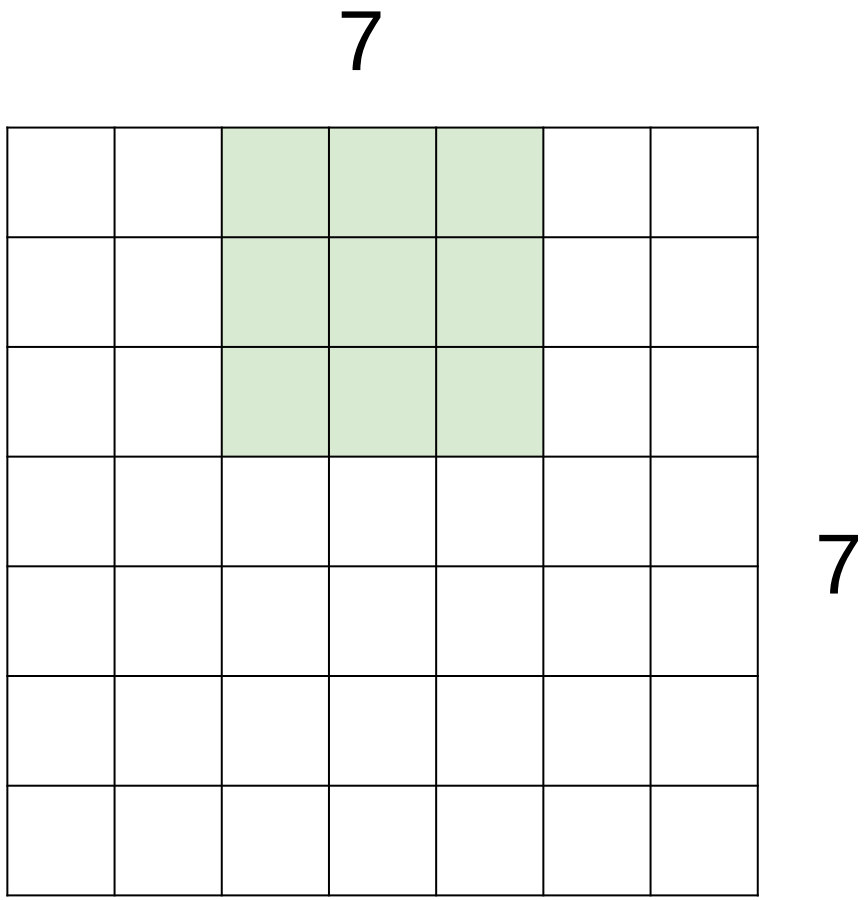
=> 5x5 output

A closer look at spatial dimensions:



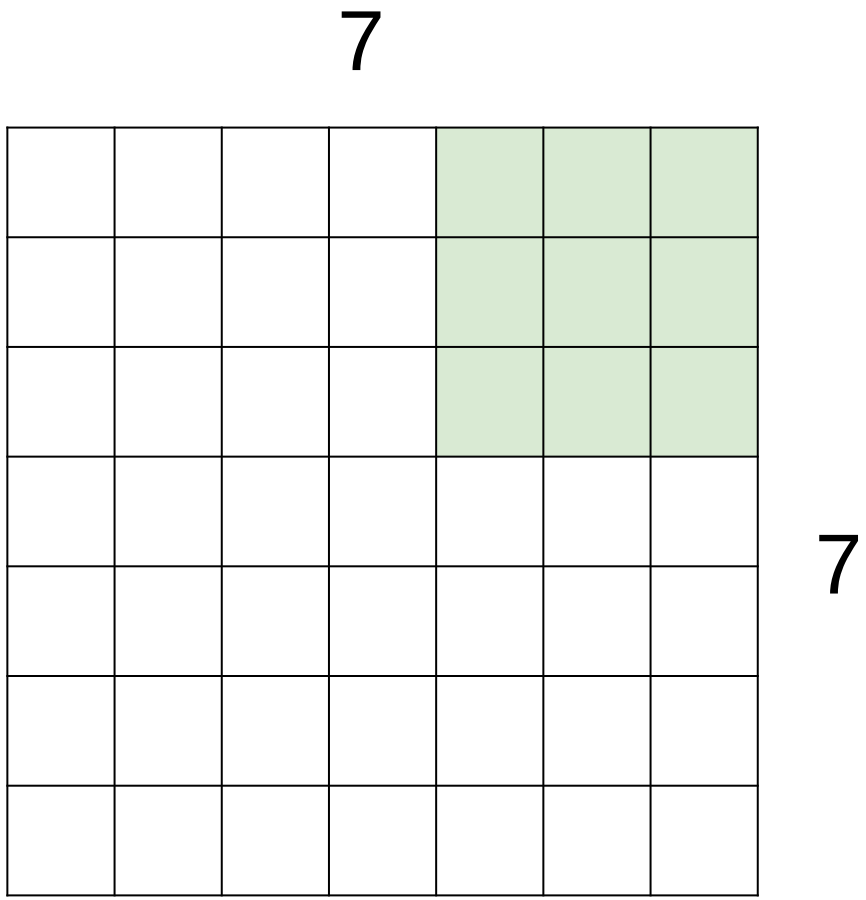
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



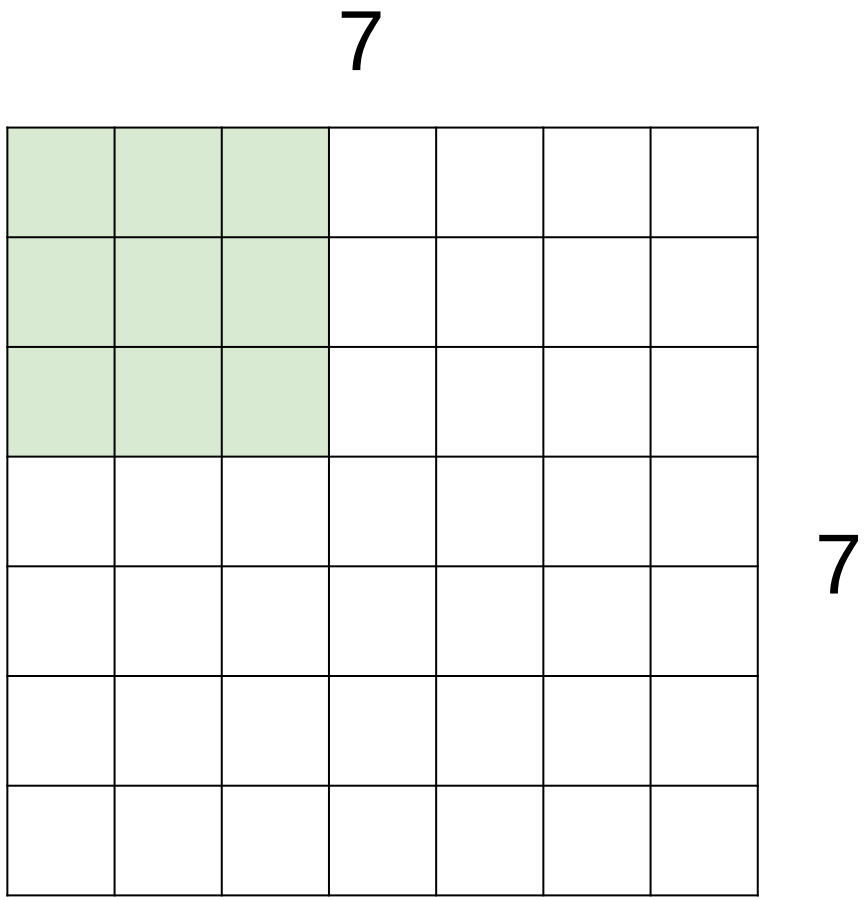
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



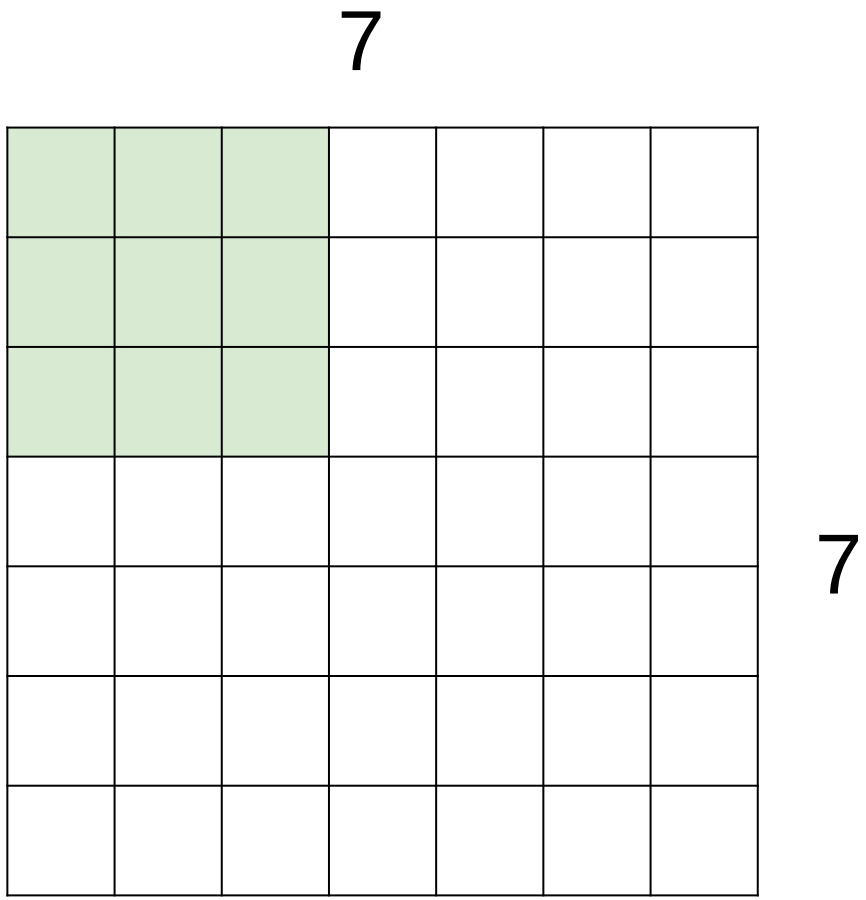
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



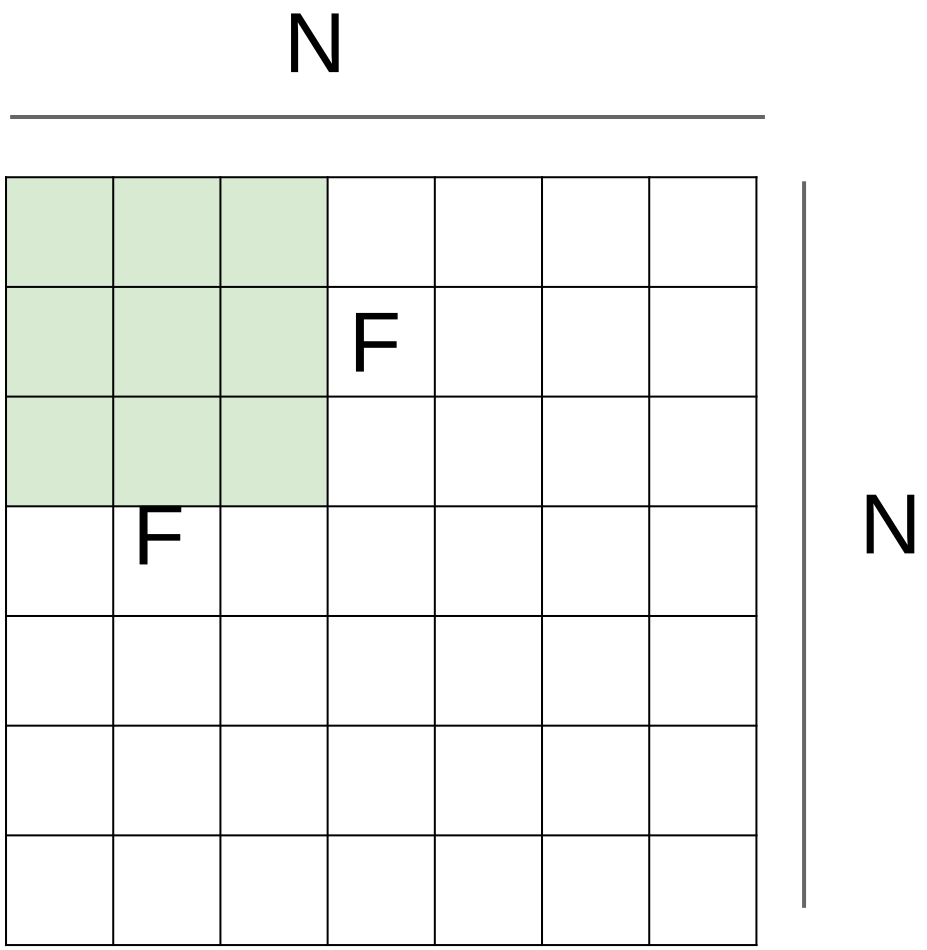
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

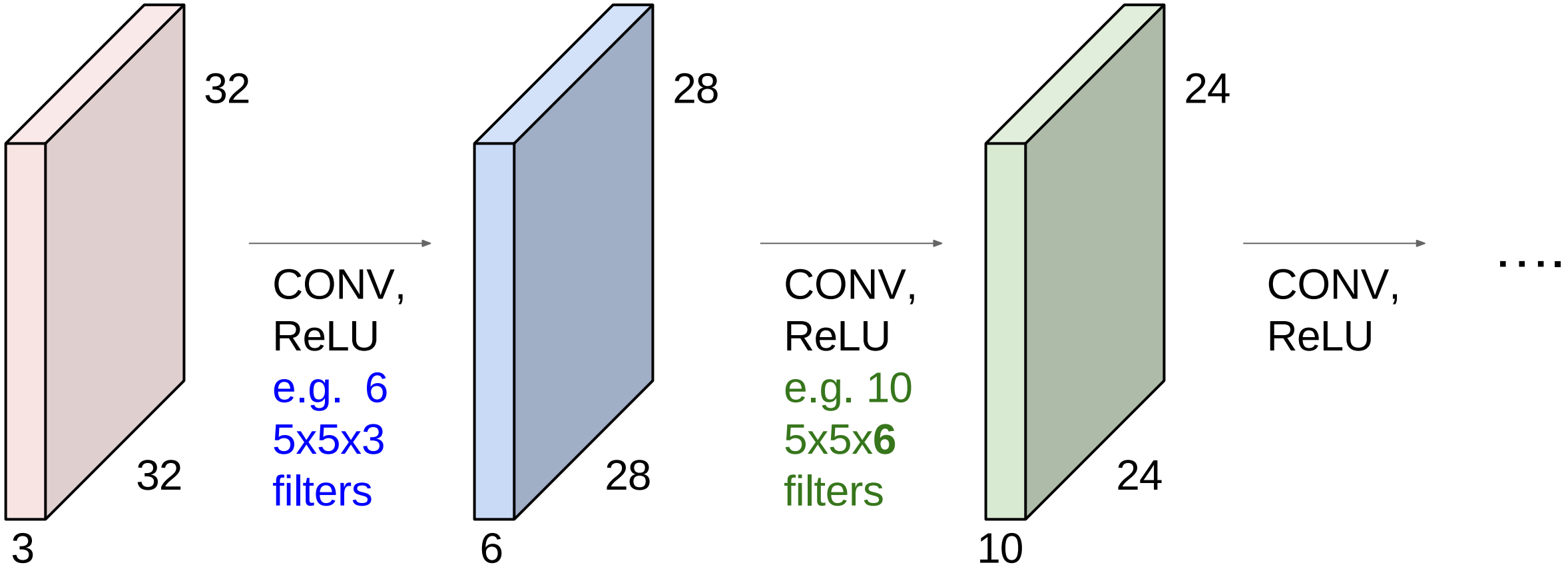


Output size:
(N - F) / stride + 1

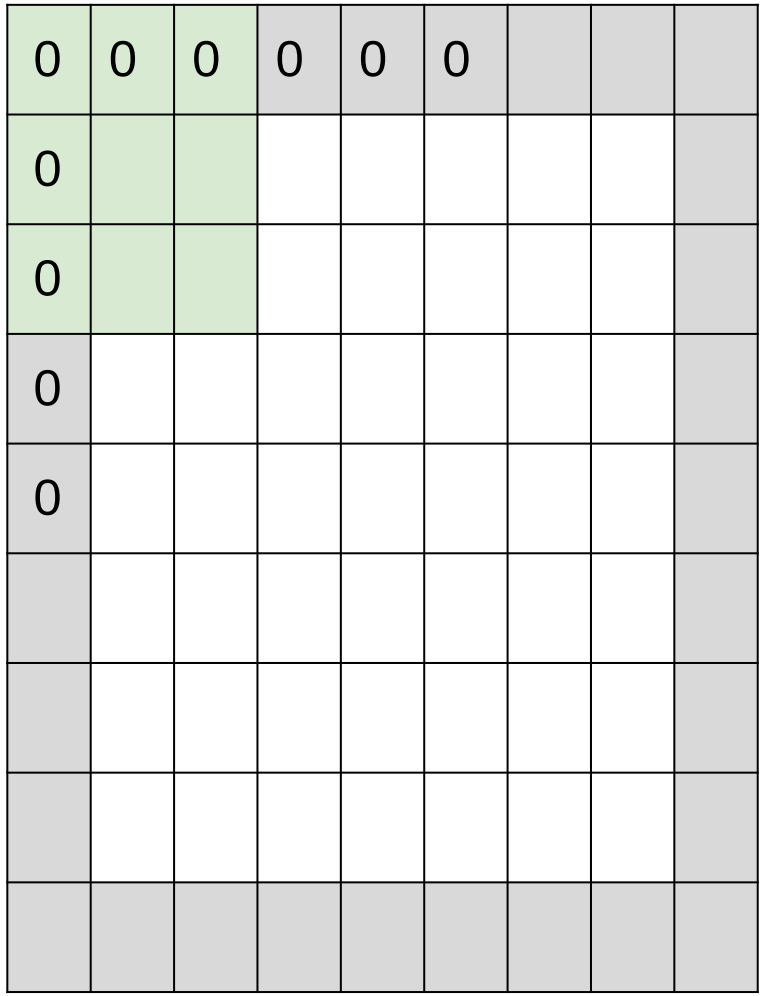
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 ☹

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



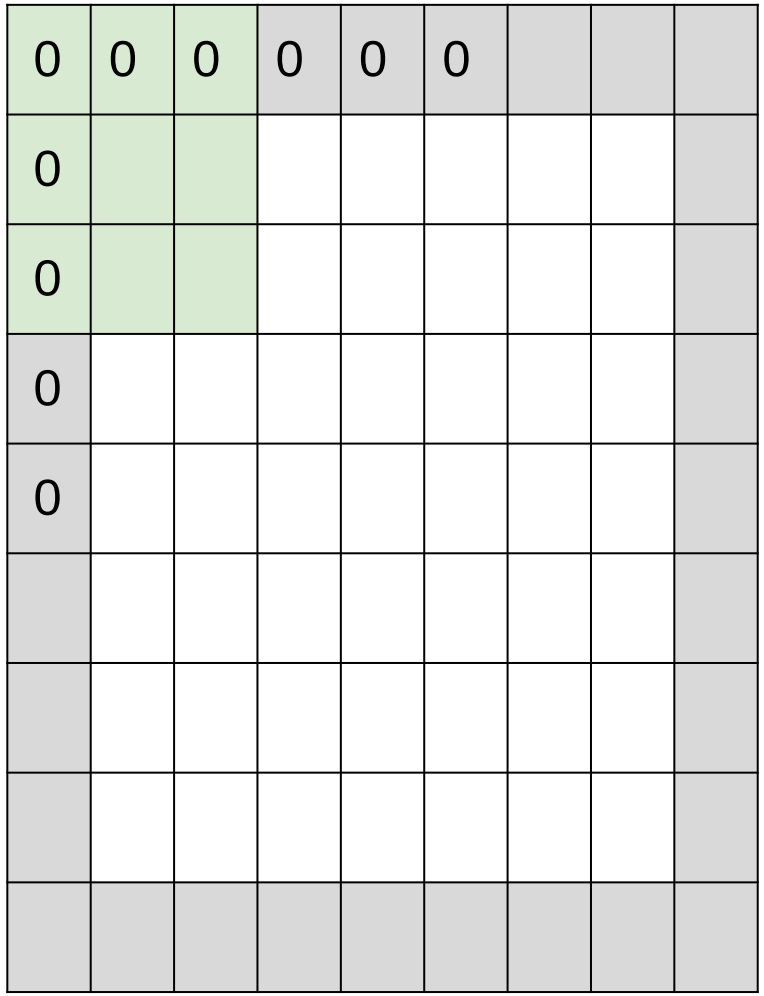
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

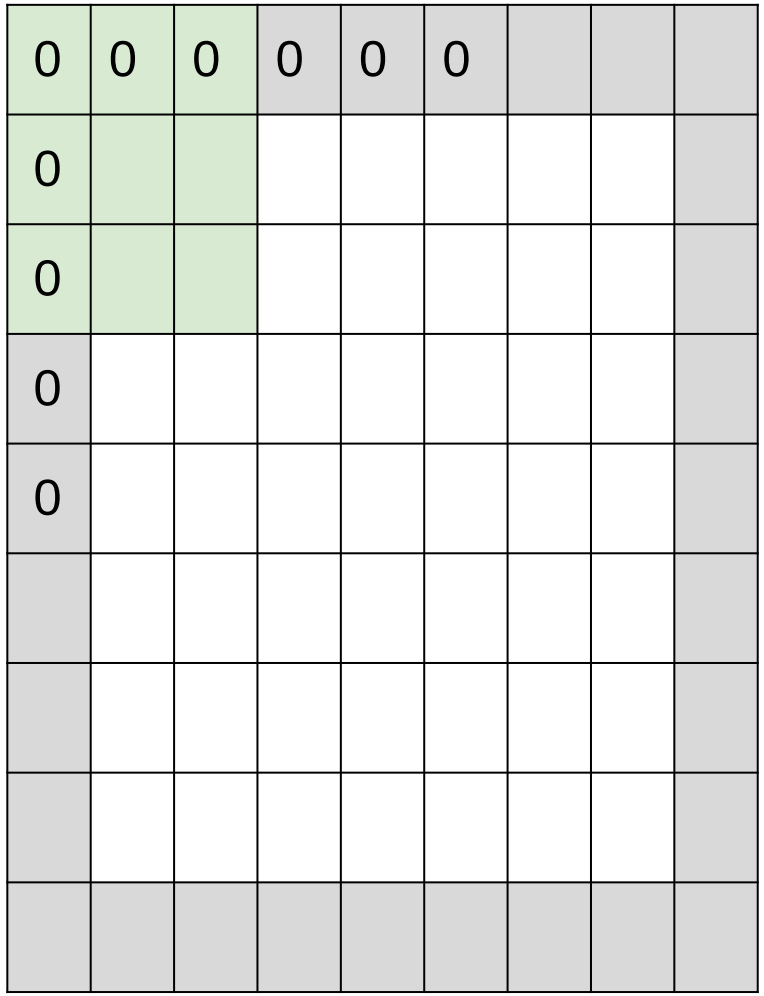
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

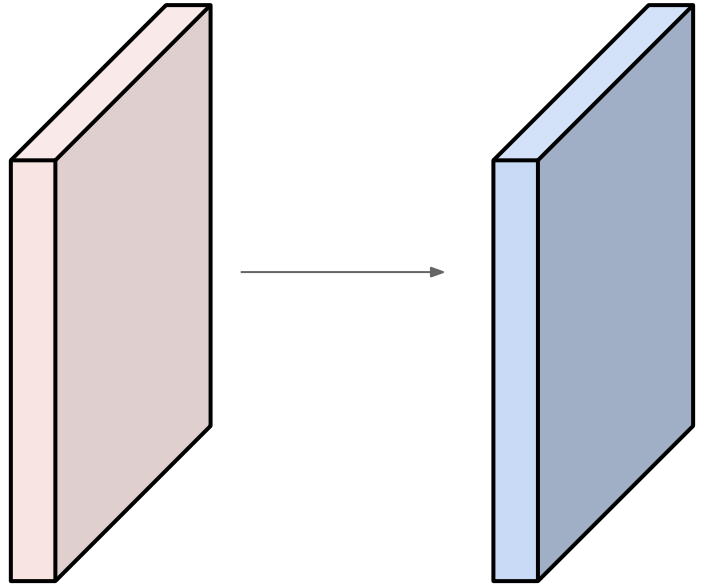
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 2, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

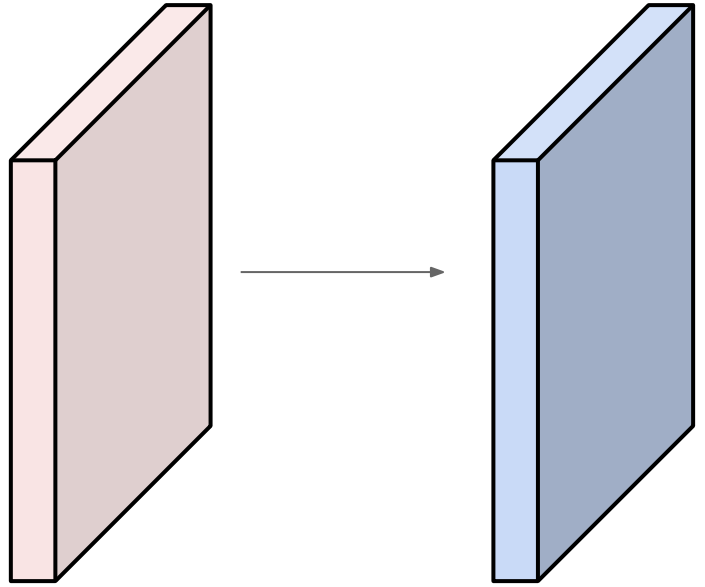
Example time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

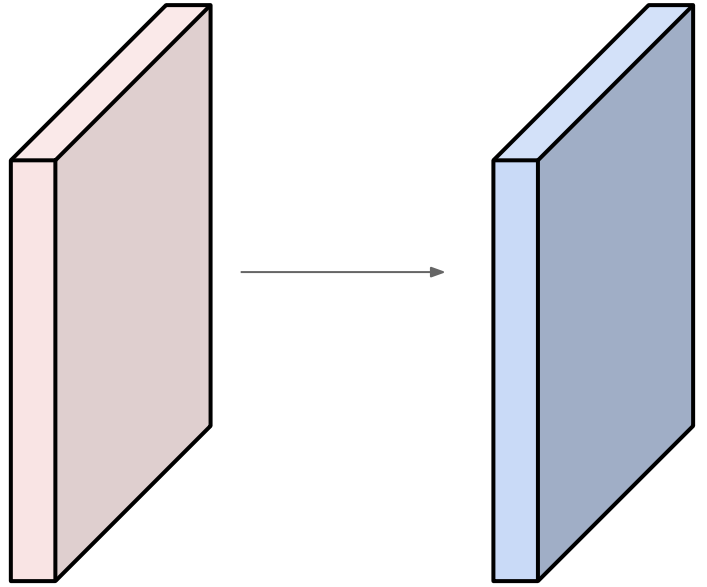
Example time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
=> 32x32x10 = **10240**

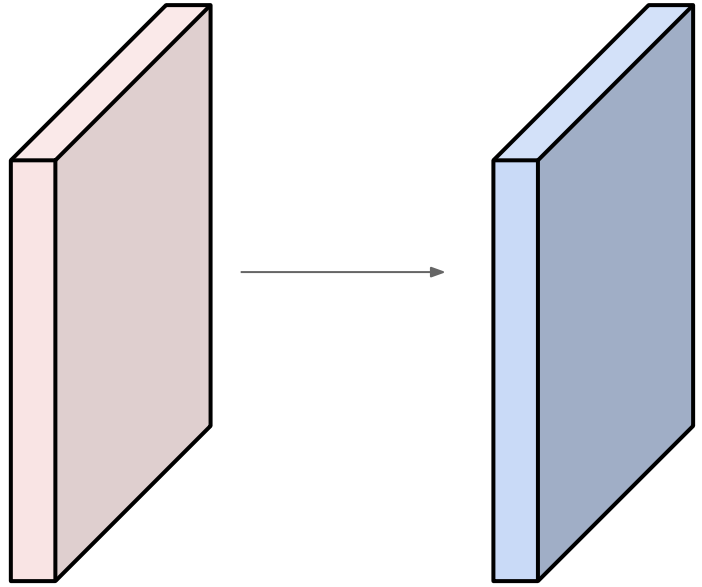
Example time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Example time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Example: CONV layer in Torch

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2