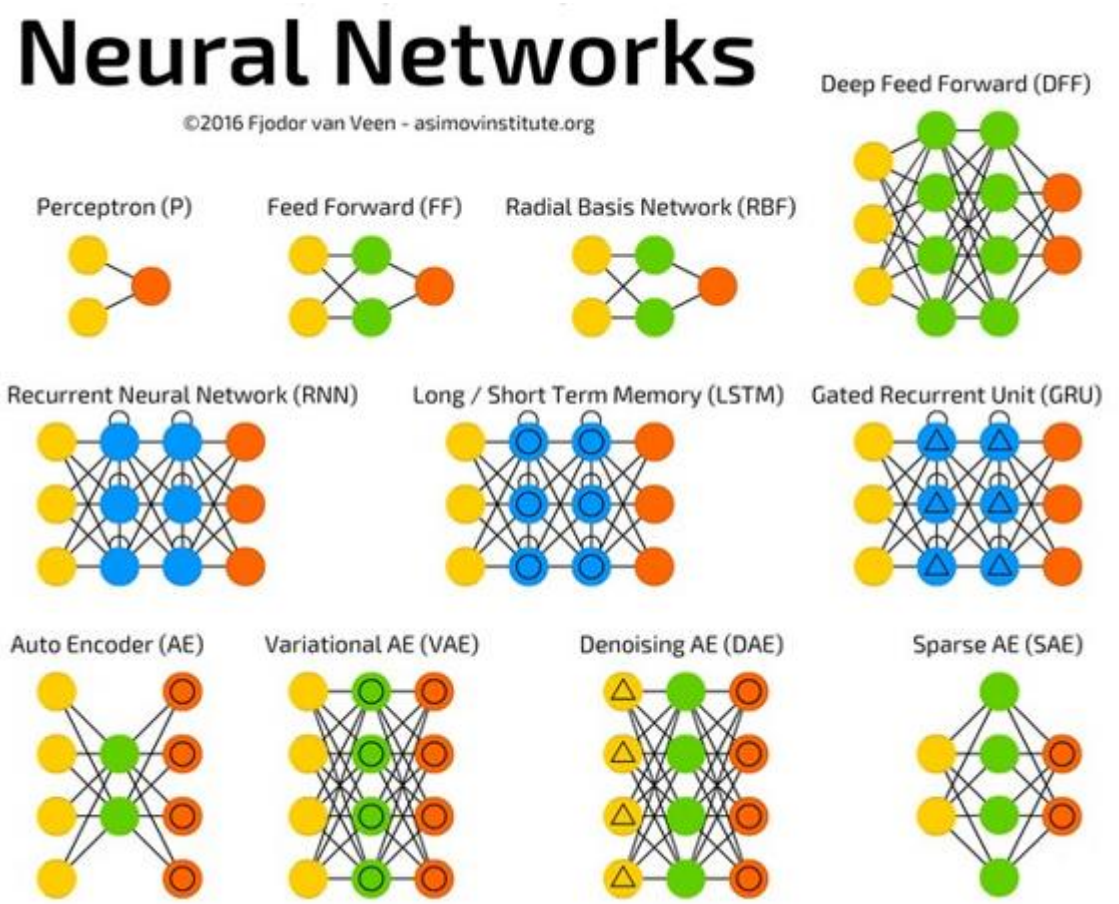
CSC 767 Neural Networks & Deep Learning

Lecture 4

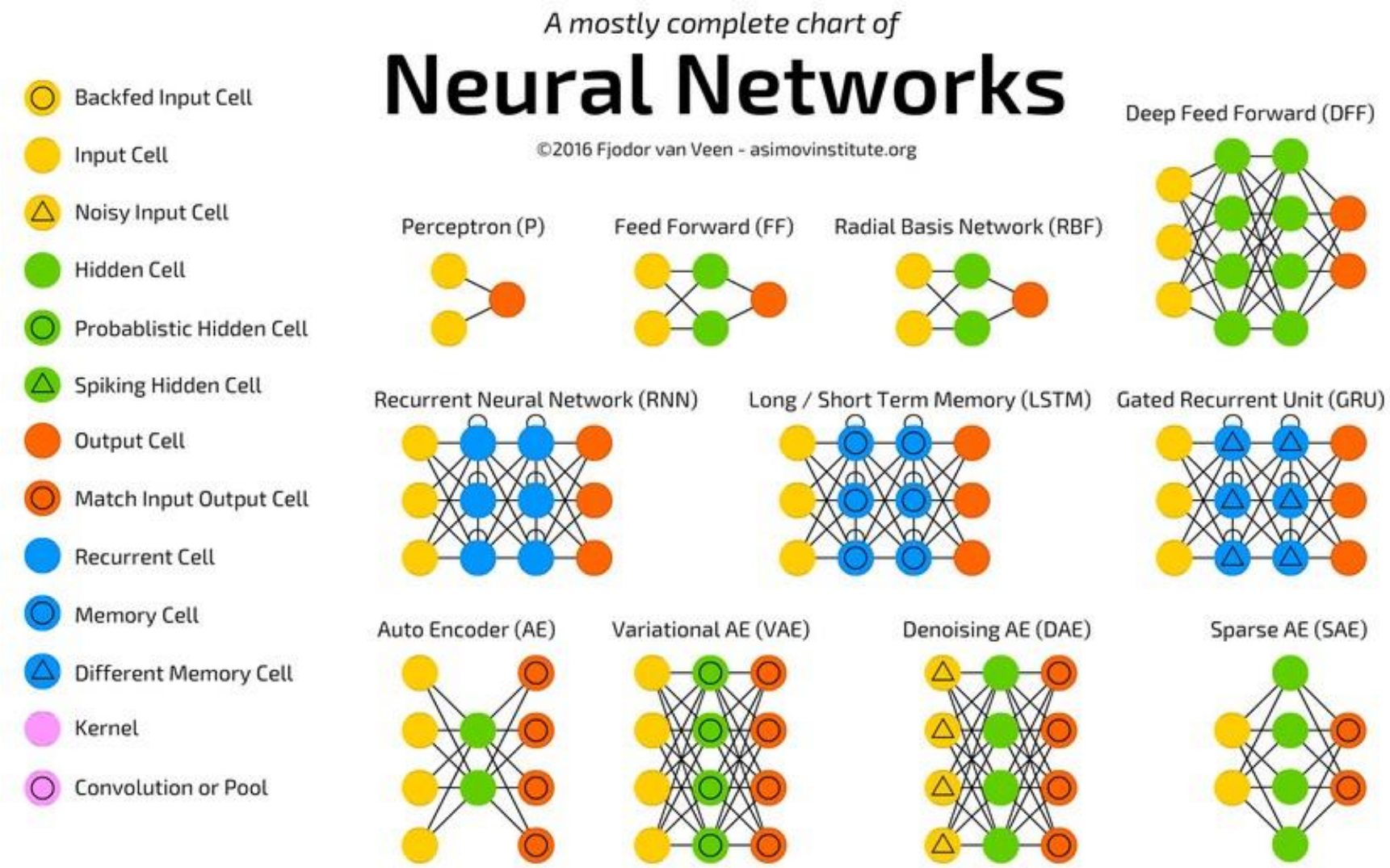
Convolutional Neural Networks

Architecture Designs for Deep Learning (DL). Image Classification. Convolutional Neural Networks (CNNs). Preserving Spatial Architecture. Understanding CNNs

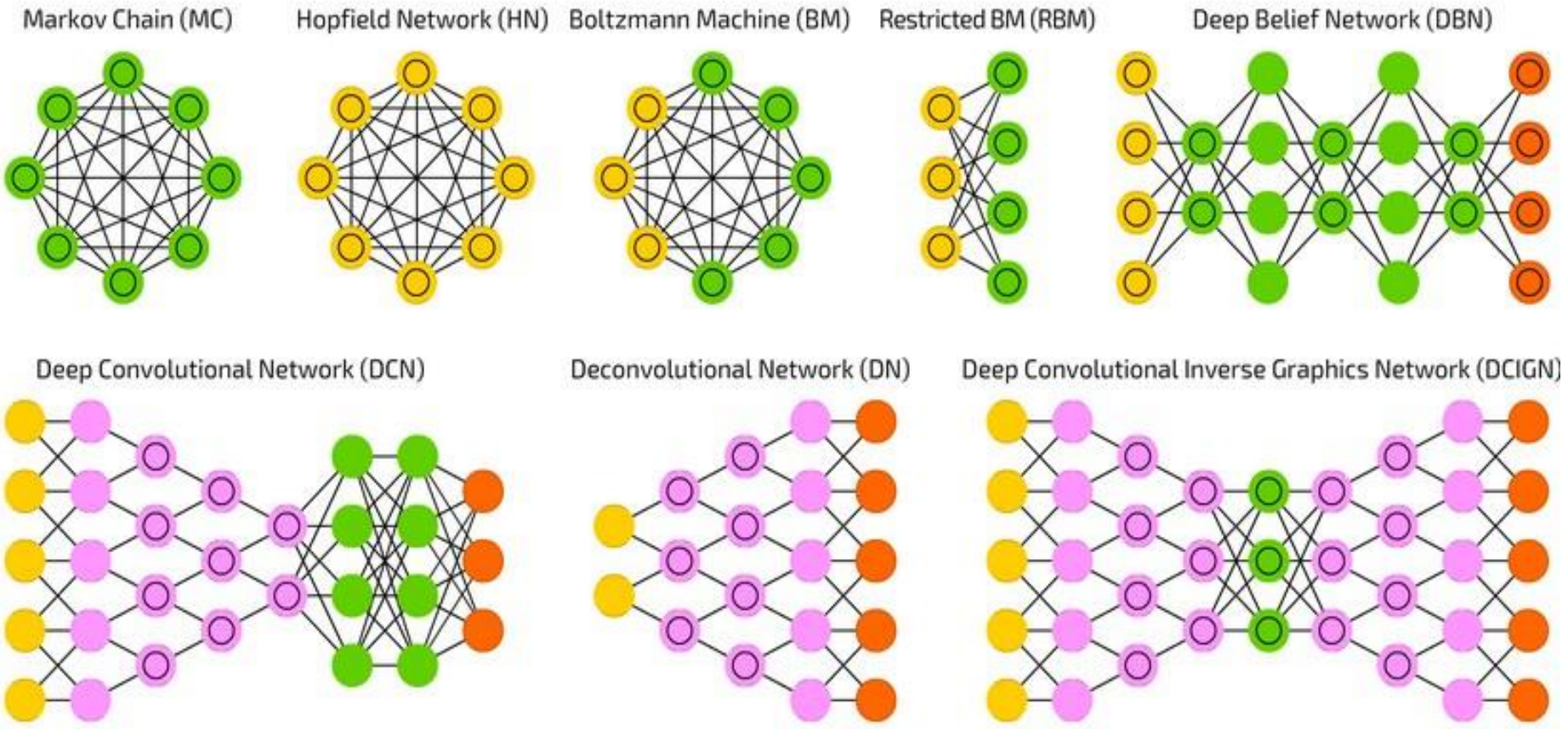
Architecture Designs for Deep Learning



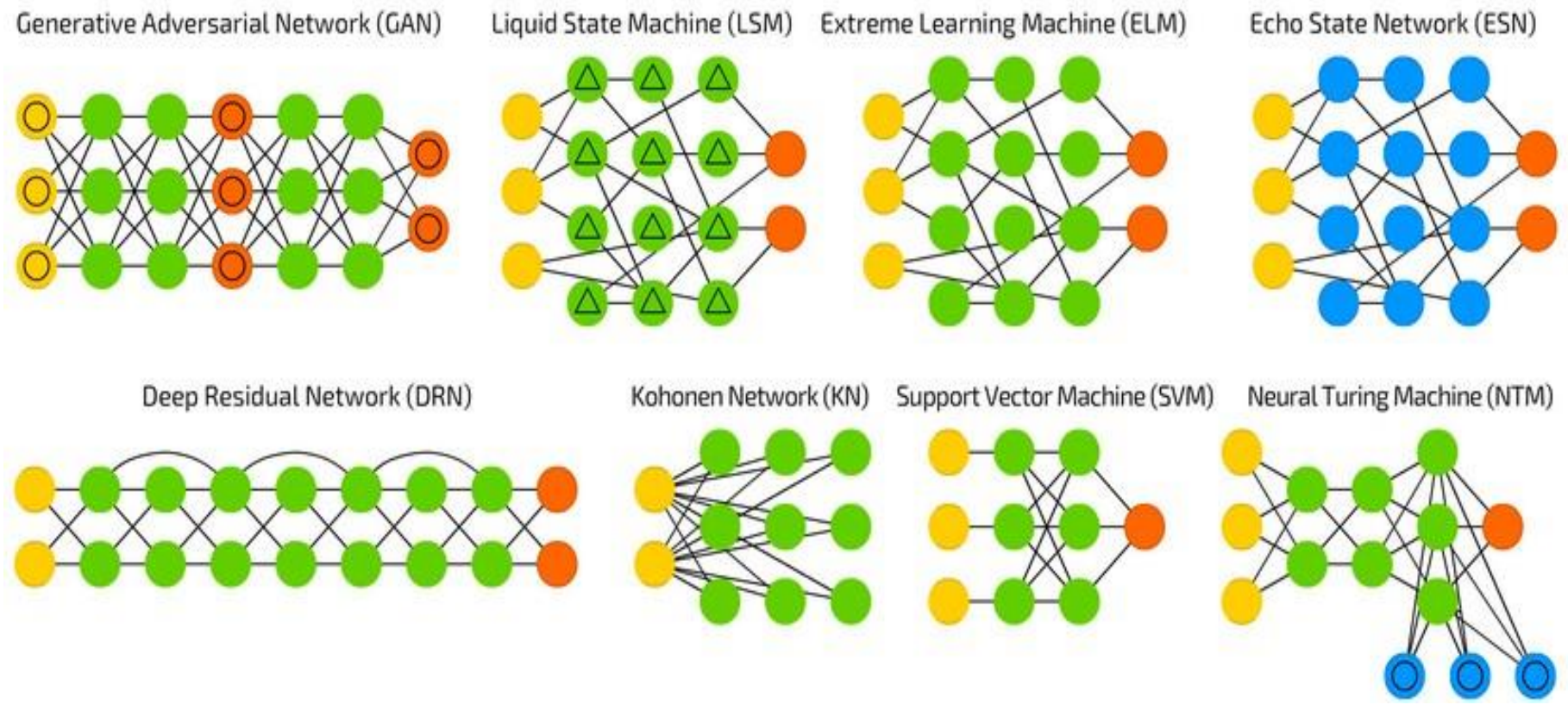
Generic Neural Architectures (1-11)



Generic Neural Architectures (12-19)



Generic Neural Architectures (20-27)

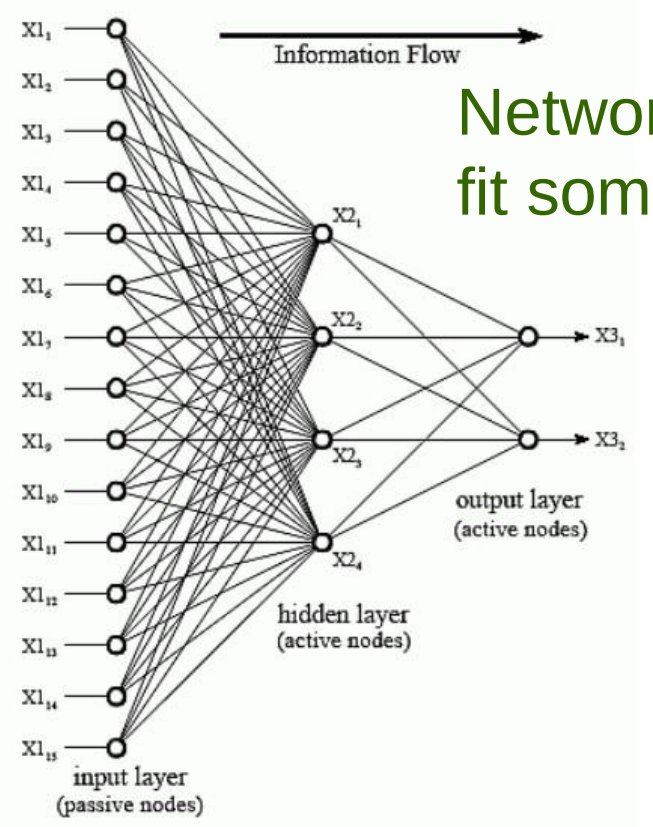


Architecture Terminology

- The word *architecture* refers to the overall structure of the network:
  - How many units should it have?
  - How the units should be connected to each other?
- Most neural networks are organized into groups of units called *layers*
  - Most neural network architectures arrange these layers in a chain structure
  - With each layer being a function of the layer that preceded it.

Main Architectural Considerations

1. Choice of depth of network : the number of hidden layers through which data must pass in a multistep process of pattern recognition.
2. Choice of width of each layer : the maximal number of nodes in a layer.



Network with even one hidden layer is sufficient to fit some training set

Advantage of Deeper Networks

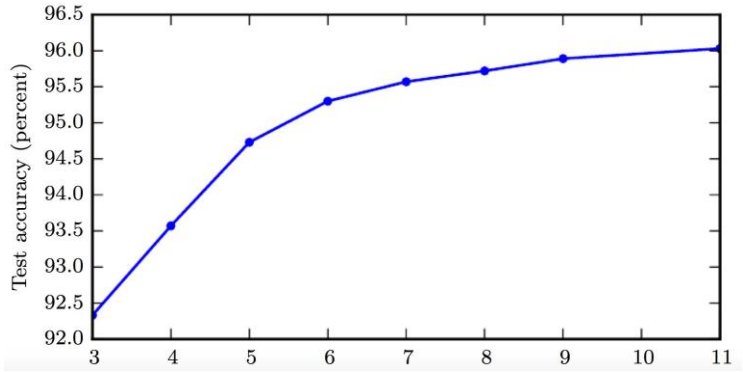
- Deeper networks have
  - Far fewer units in each layer
  - Far fewer parameters
  - Generalize well to the test set
  - But are often more difficult to optimize
- Ideal network architecture must be found via experimentation guided by validation set error

Summary

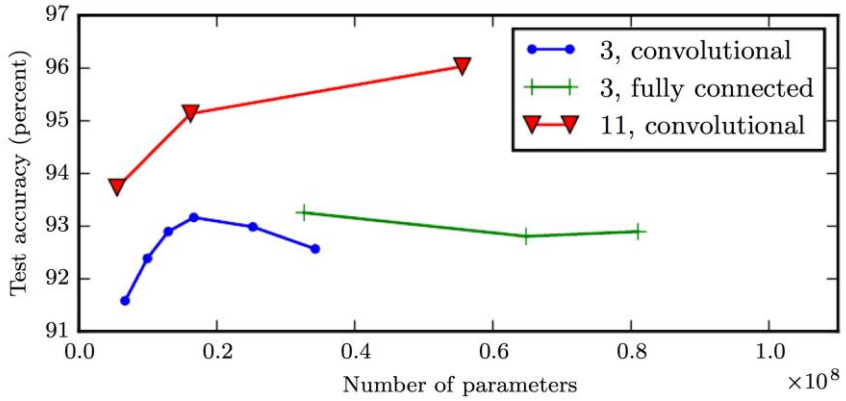
- A feedforward network with a single layer is sufficient to represent any function
- But the layer may be infeasibly large and may fail to generalize correctly
- Using deeper models can reduce number of units required and reduce generalization error

Empirical Results

- Deeper networks perform better



Test accuracy consistently increases with depth



Increasing parameters without increasing depth is not as effective

- Deep architectures indeed express a useful prior over the space of functions the model learns

Image Classification

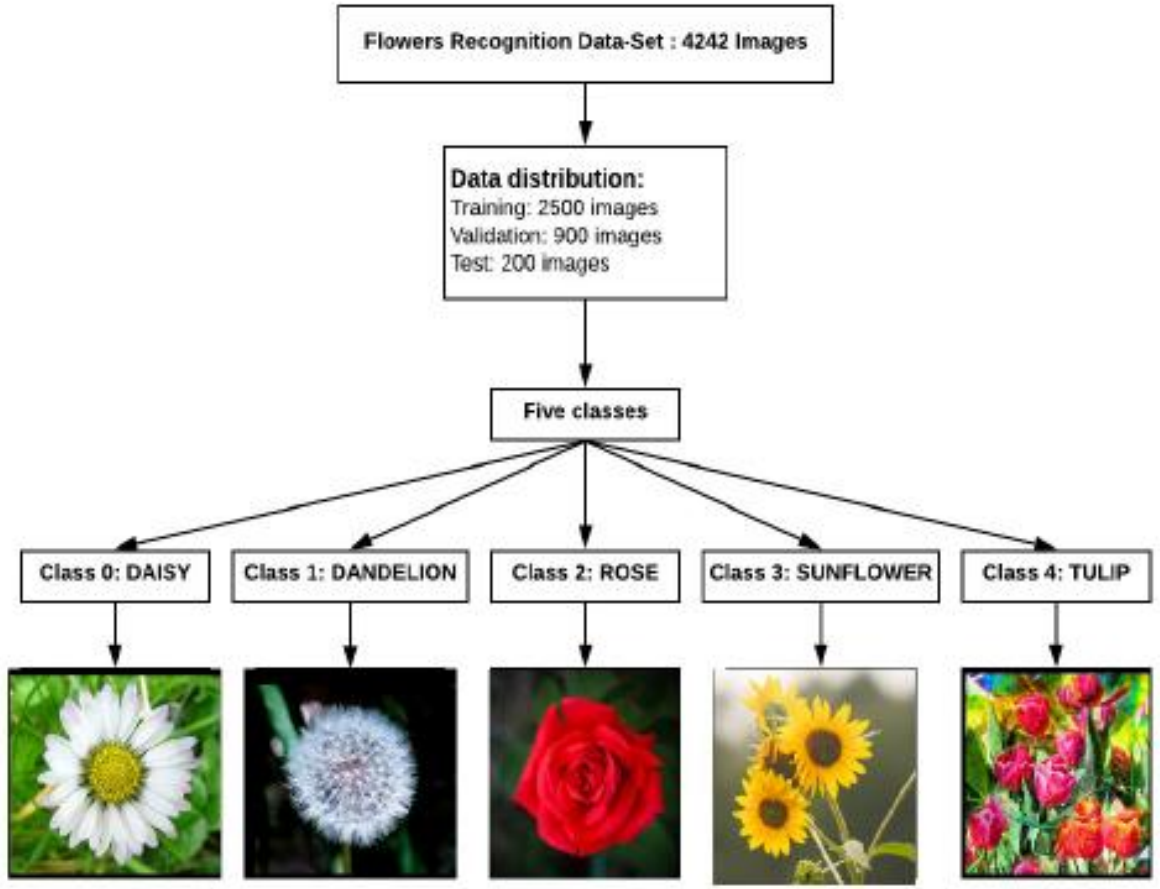


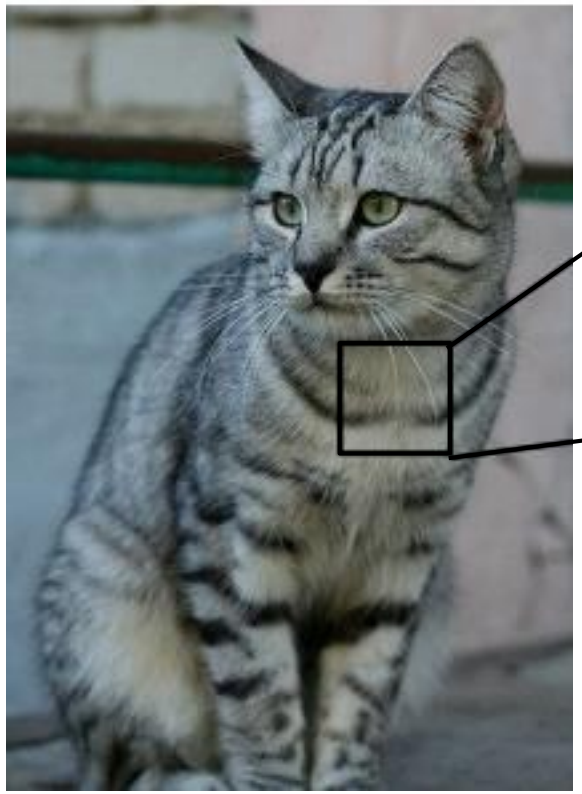
Image Classification: A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

The image classification model takes a single image and assigns probabilities to 4 labels, *{cat, dog, hat, mug}*. As shown in the image, keep in mind that to a computer an image is represented as one large 3-dimensional array of numbers. In this example, the cat image is 248 pixels wide, 400 pixels tall, and has three color channels Red,Green,Blue (RGB).
Therefore, the image consists of 248 x 400 x 3 numbers, or a total of 297,600 numbers. Each number is an integer that ranges from 0 (black) to 255 (white). Our task is to turn this quarter of a million numbers into a single label, such as *"cat"*.

The Problem: Semantic Gap




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  98 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  68  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 100 100  92  74  65  72  78]
 [ 89  93  98  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  78  82  99  94]
 [130 120 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [120 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  88 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

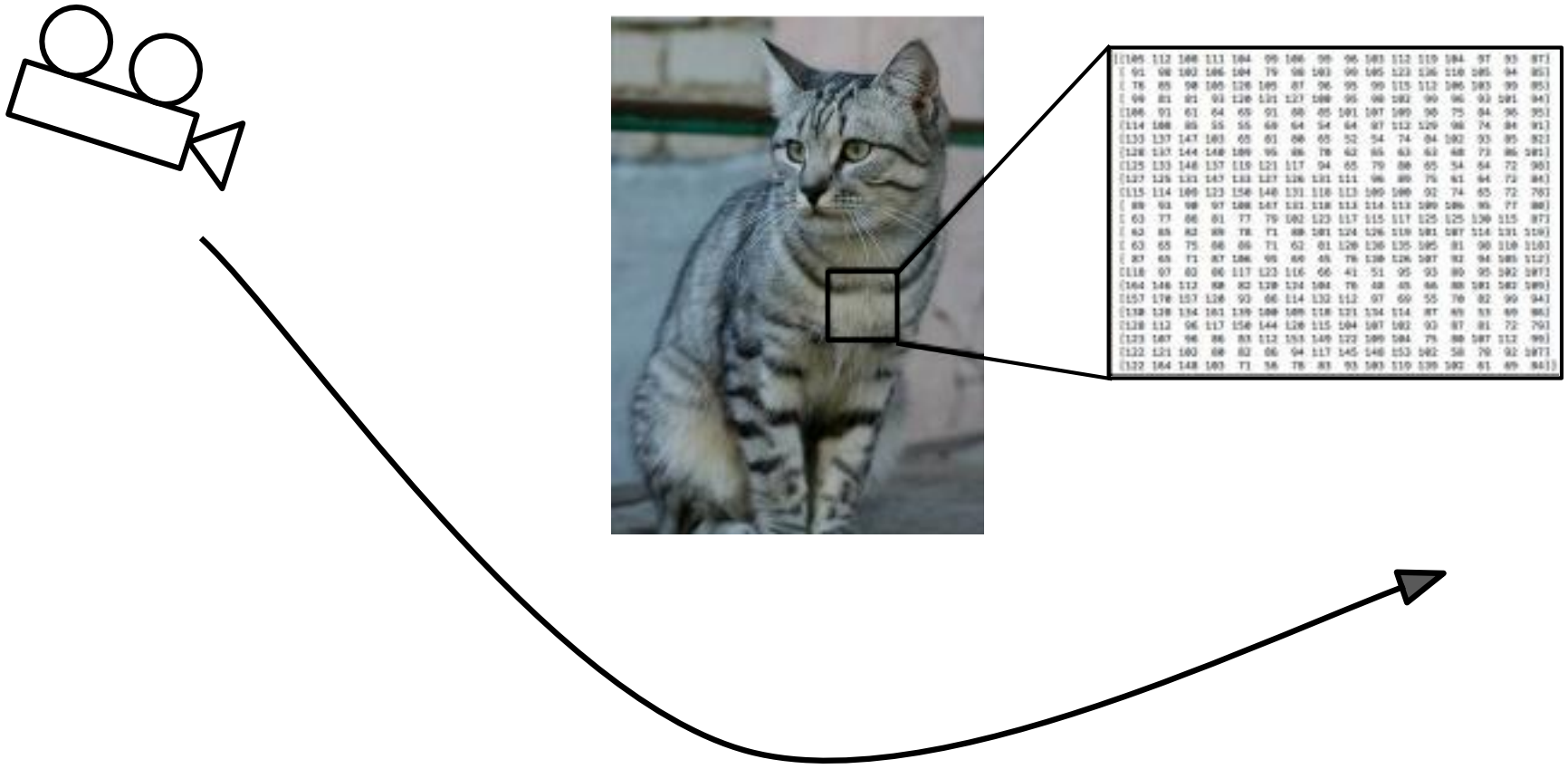
e.g. 800 x 600 x 3
(3 channels RGB)

Challenges

- **Viewpoint variation**. A single instance of an object can be oriented in many ways with respect to the camera.
- **Scale variation**. Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- **Deformation**. Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- **Occlusion**. The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- **Illumination conditions**. The effects of illumination are drastic on the pixel level.
- **Background clutter**. The objects of interest may *blend* into their environment, making them hard to identify.
- **Intra-class variation**. The classes of interest can often be relatively broad, such as *chair*. There are many different types of these objects, each with their own appearance.

A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations.

Challenges: Viewpoint variation



All pixels change when the camera moves!

Challenges: Background Clutter

This image is CC0 1.0 public domain

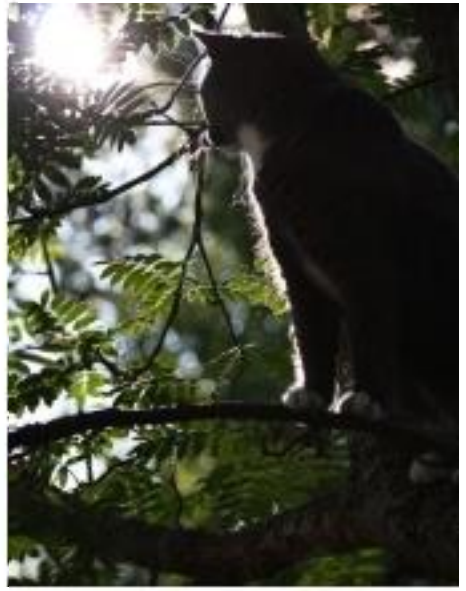
This image is CC0 1.0 public domain

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

Challenges: Deformation

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges: Intraclass variation

An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

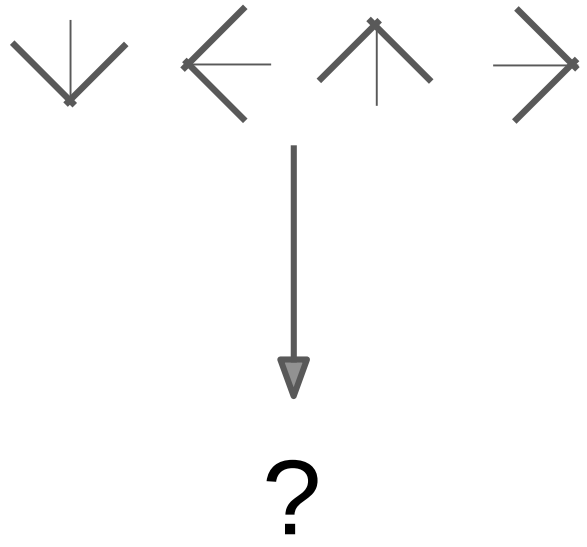
Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts have been made

Find edges

Find corners



?

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986

Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set

First classifier: **Nearest Neighbor**

```
def train(images, labels):
  # Machine learning!
  return model
```

Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Predict the label of the most similar training image

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

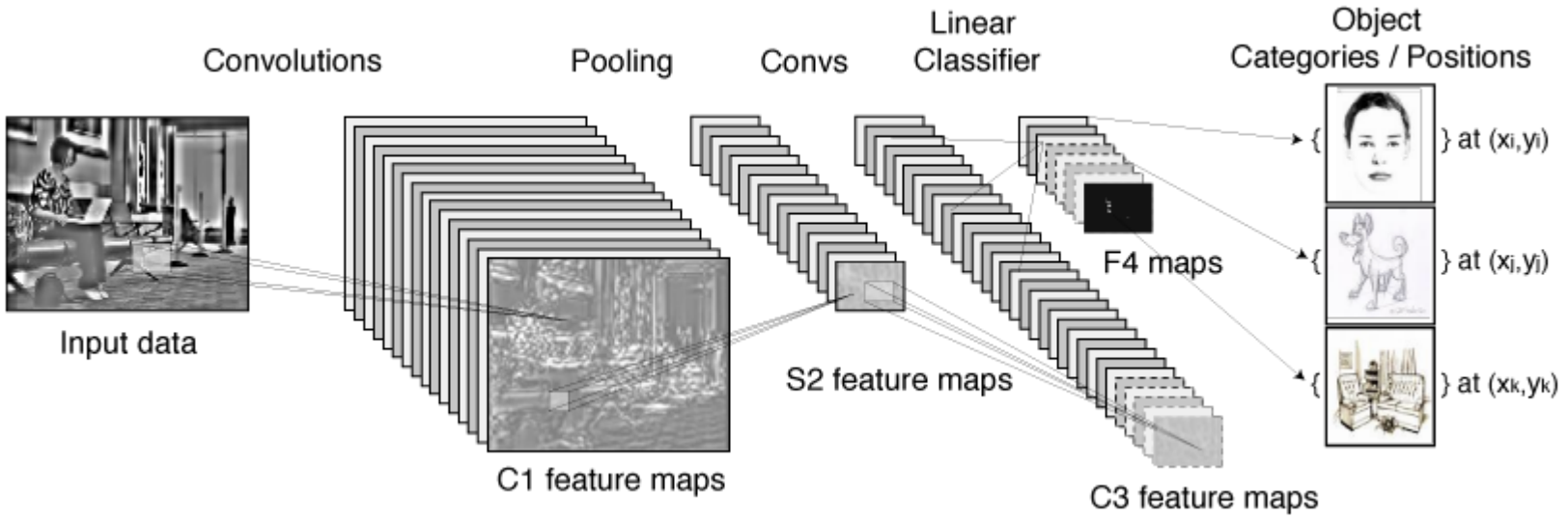
Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

Test images and nearest neighbors

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

Convolutional Neural Networks



Convolutional Neural Networks

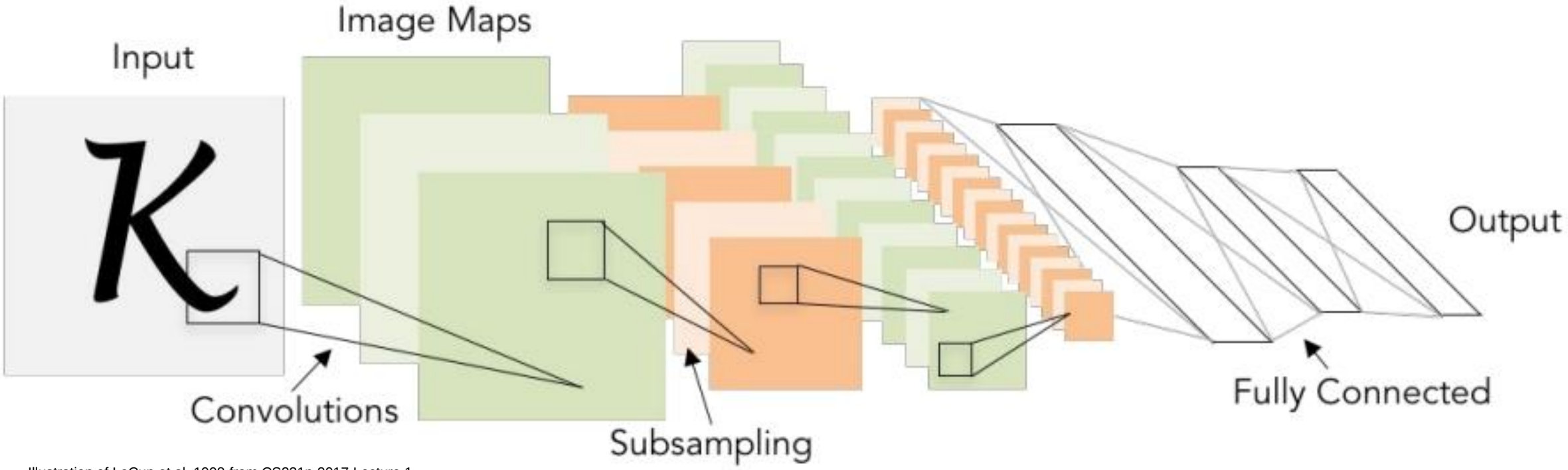
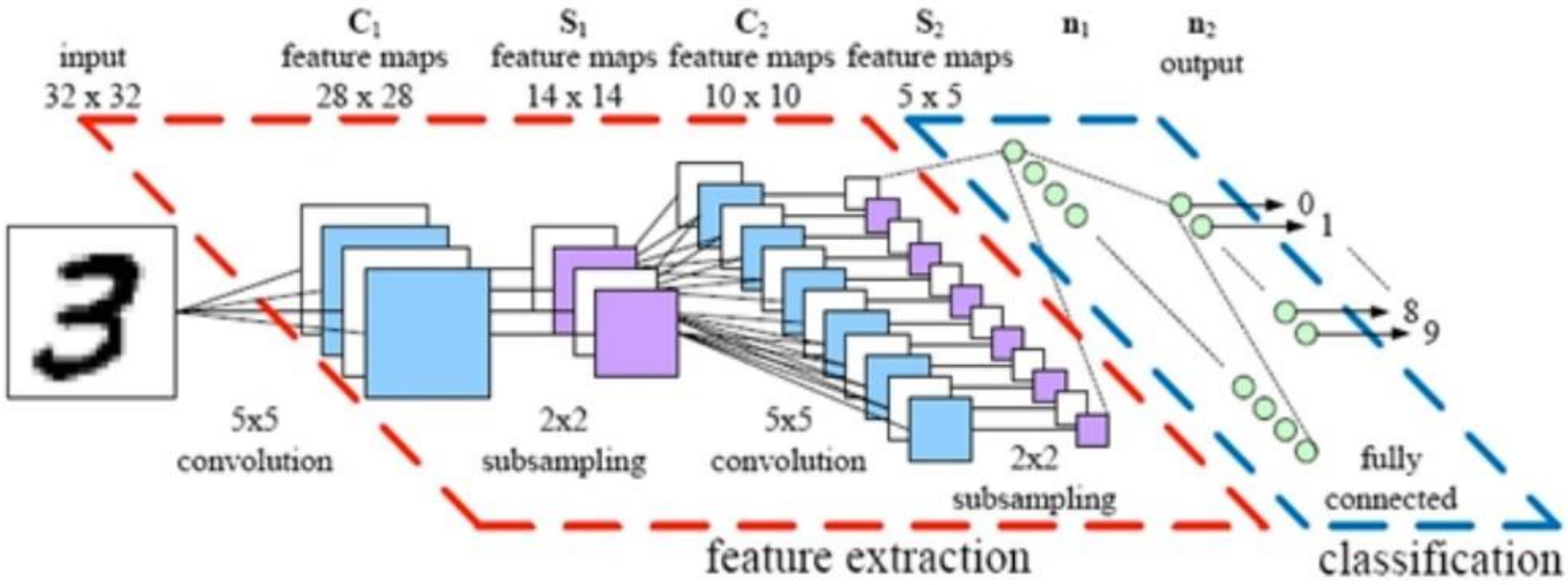
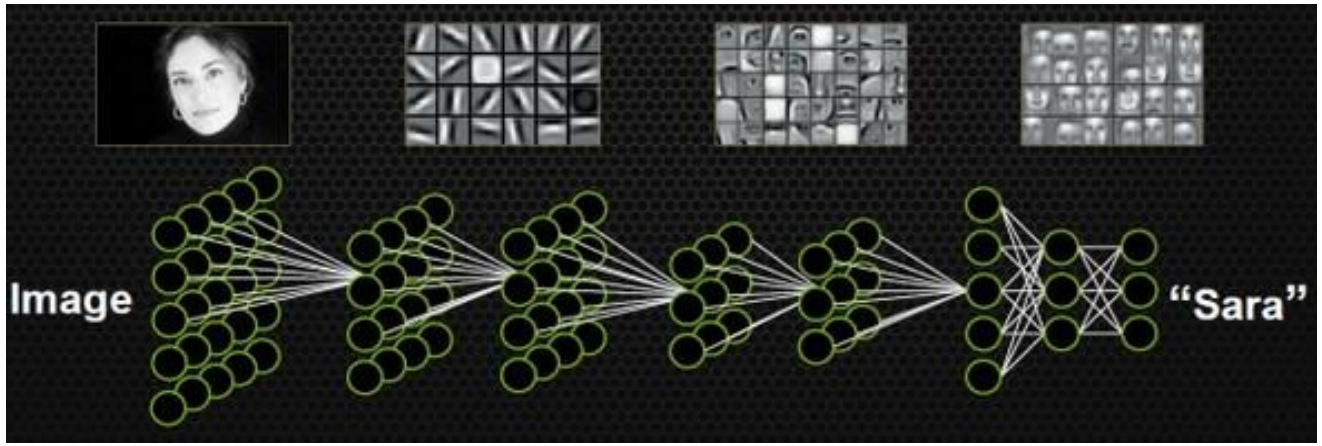


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

CNN Architectures

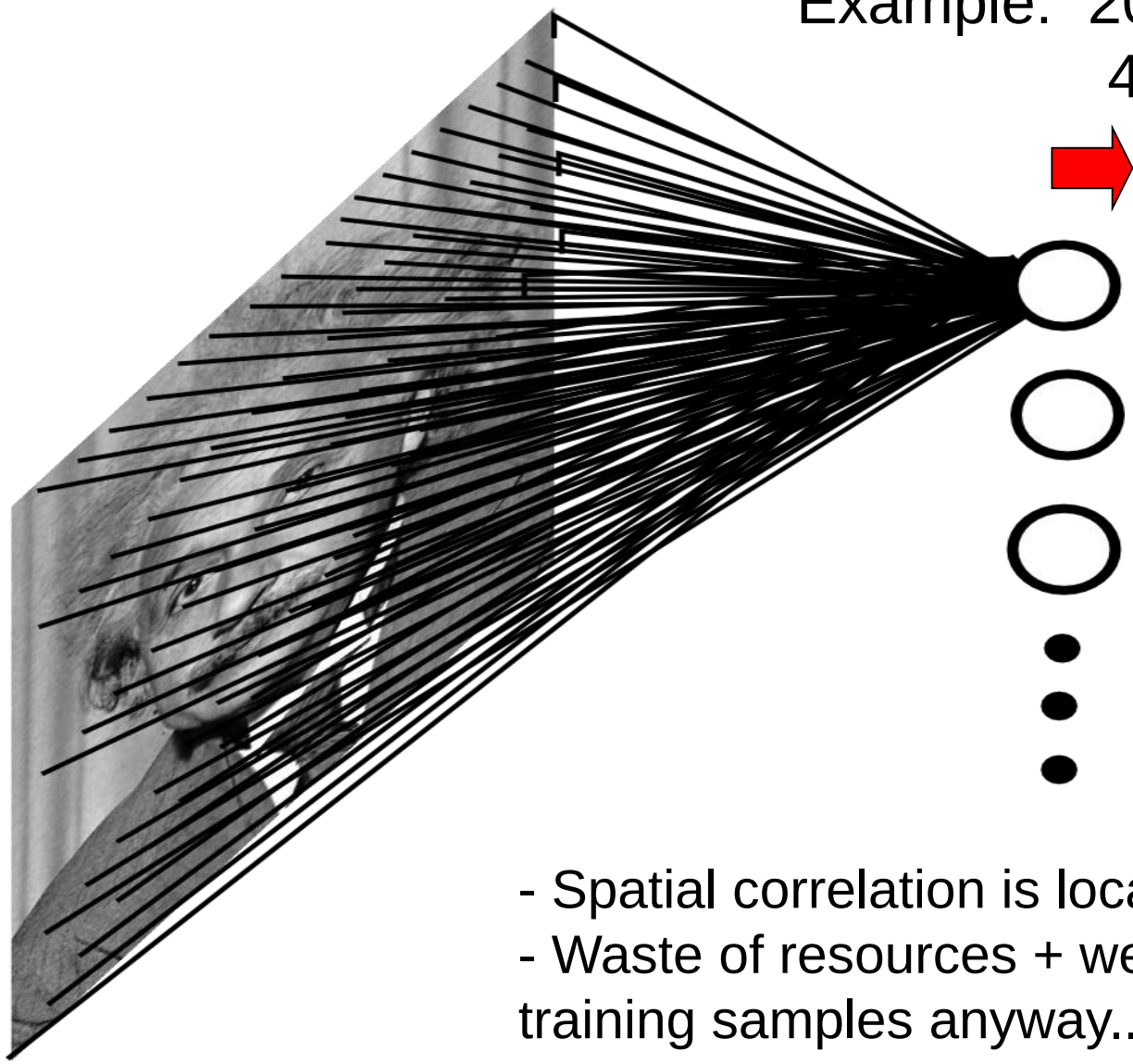




More complex features captured In deeper layers

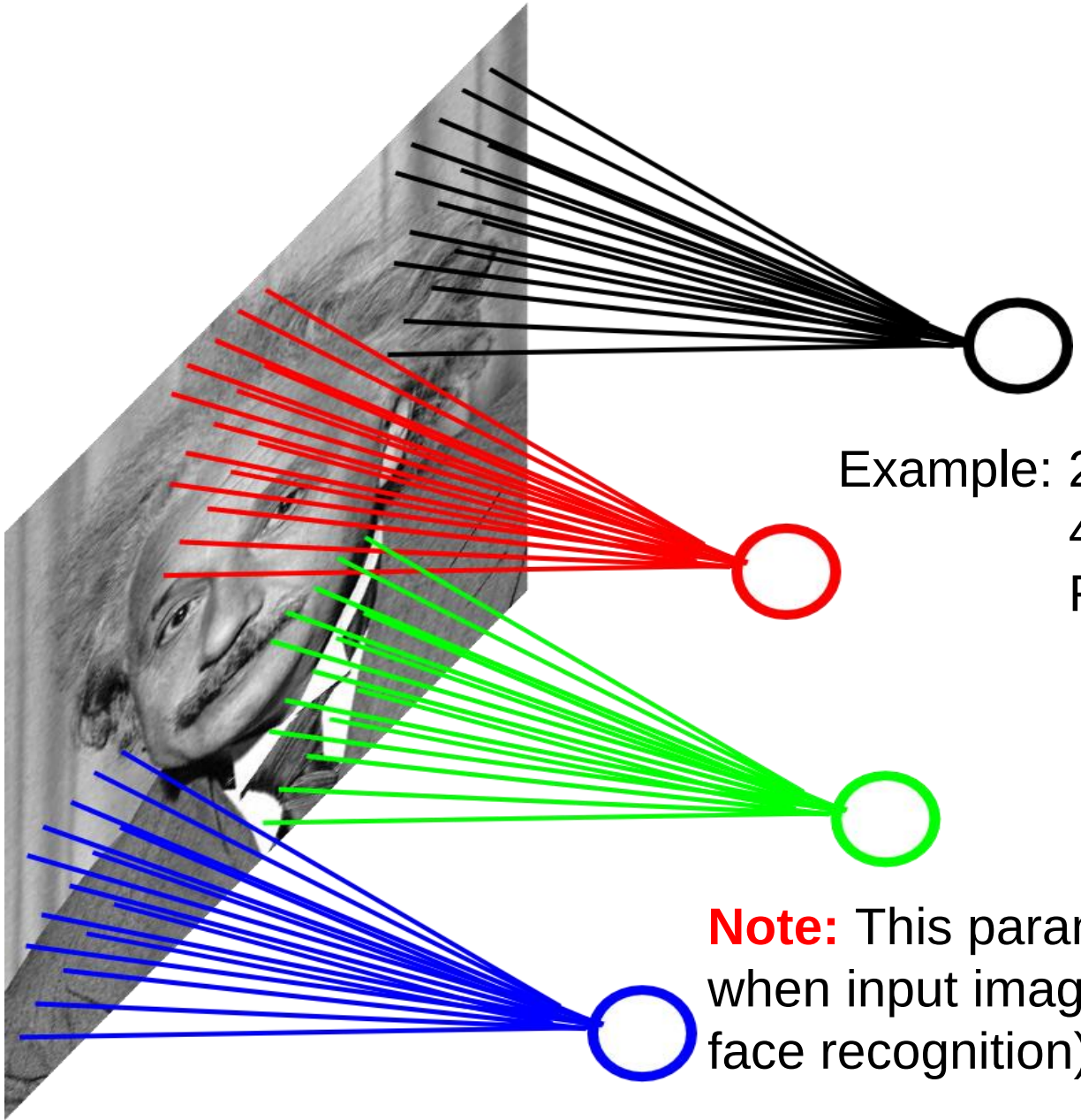
Fully Connected Layer

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



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

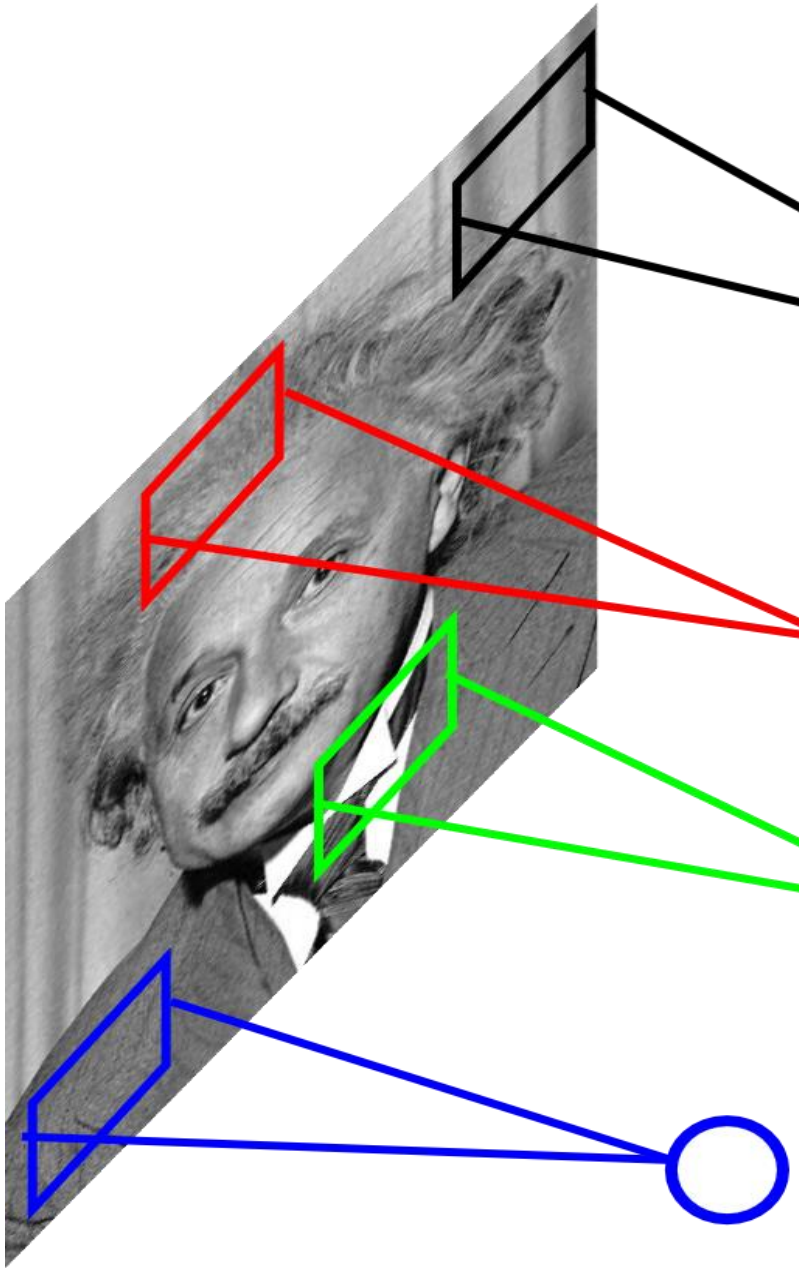
Locally Connected Layer



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

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

Locally Connected Layer

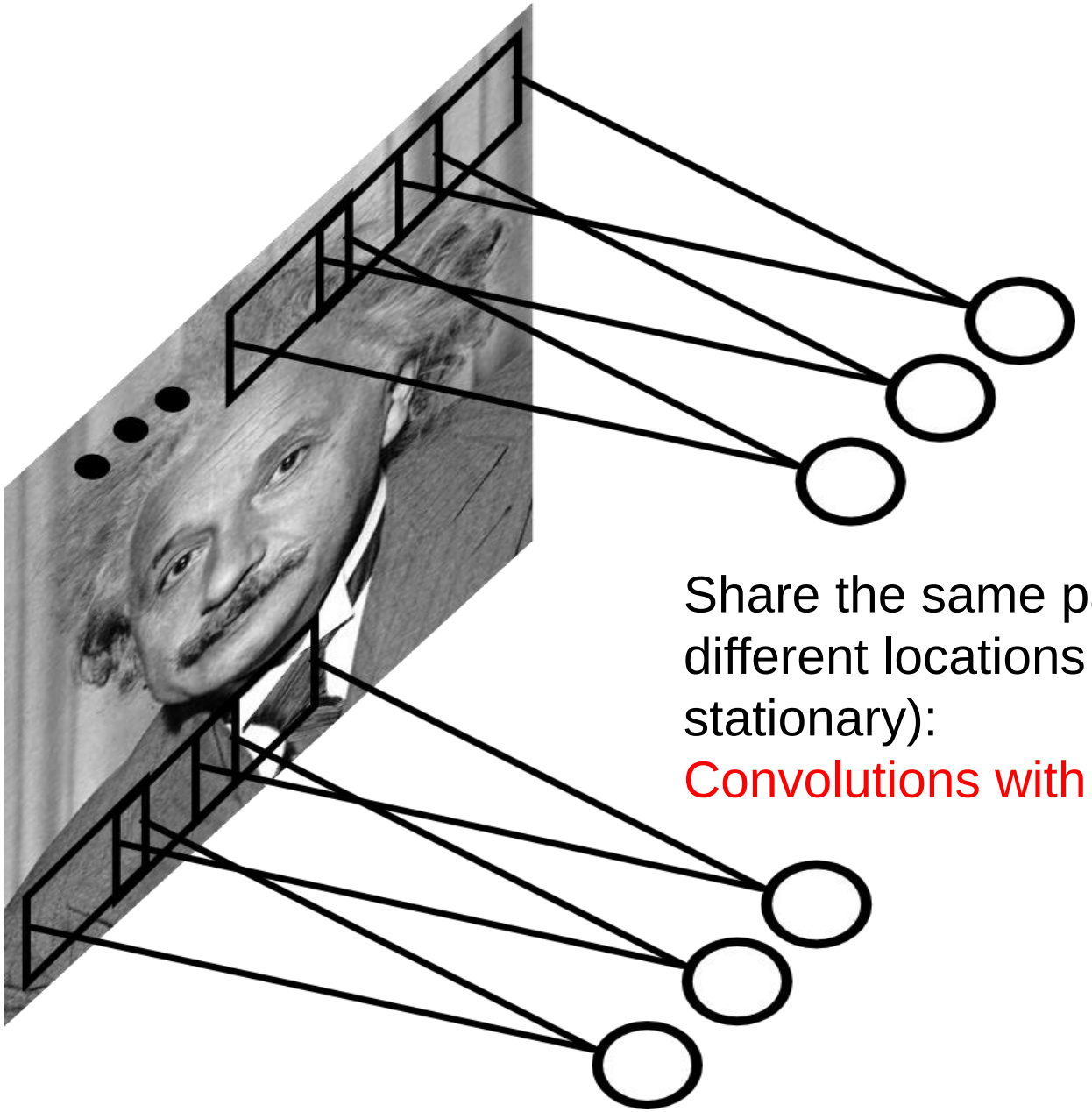


STATIONARITY? Statistics is similar at different locations

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

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

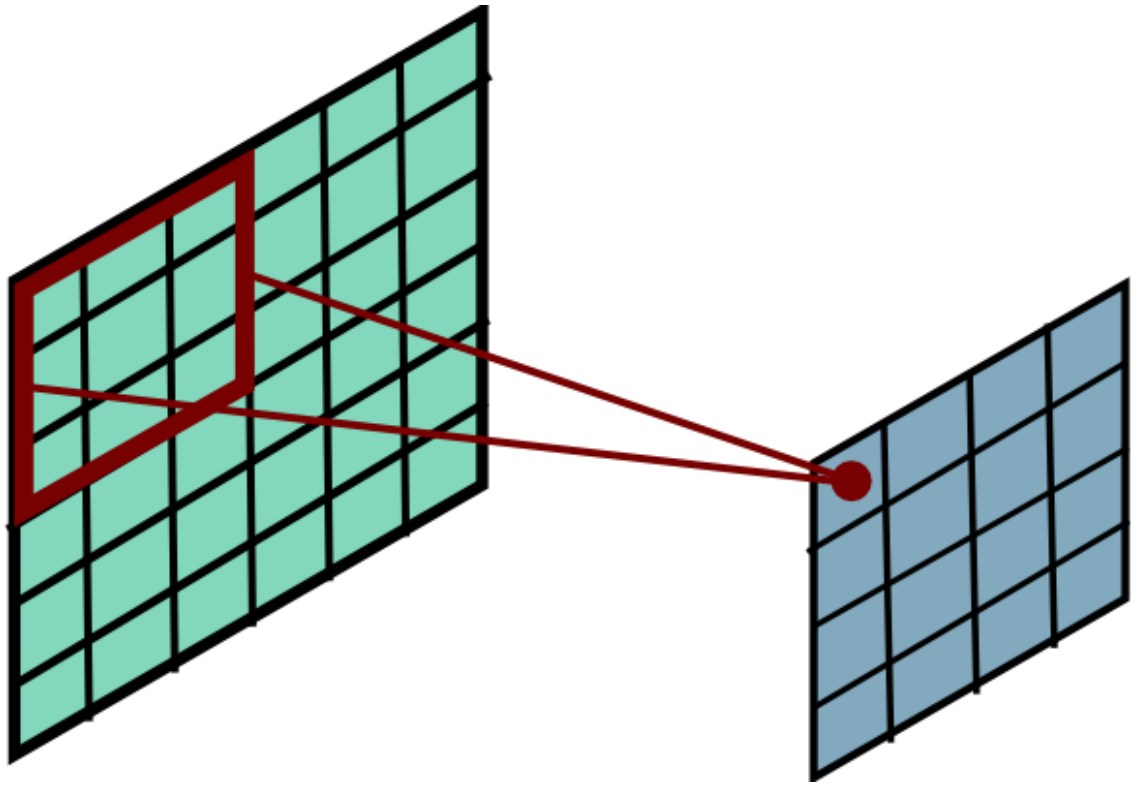
Convolutional Layer



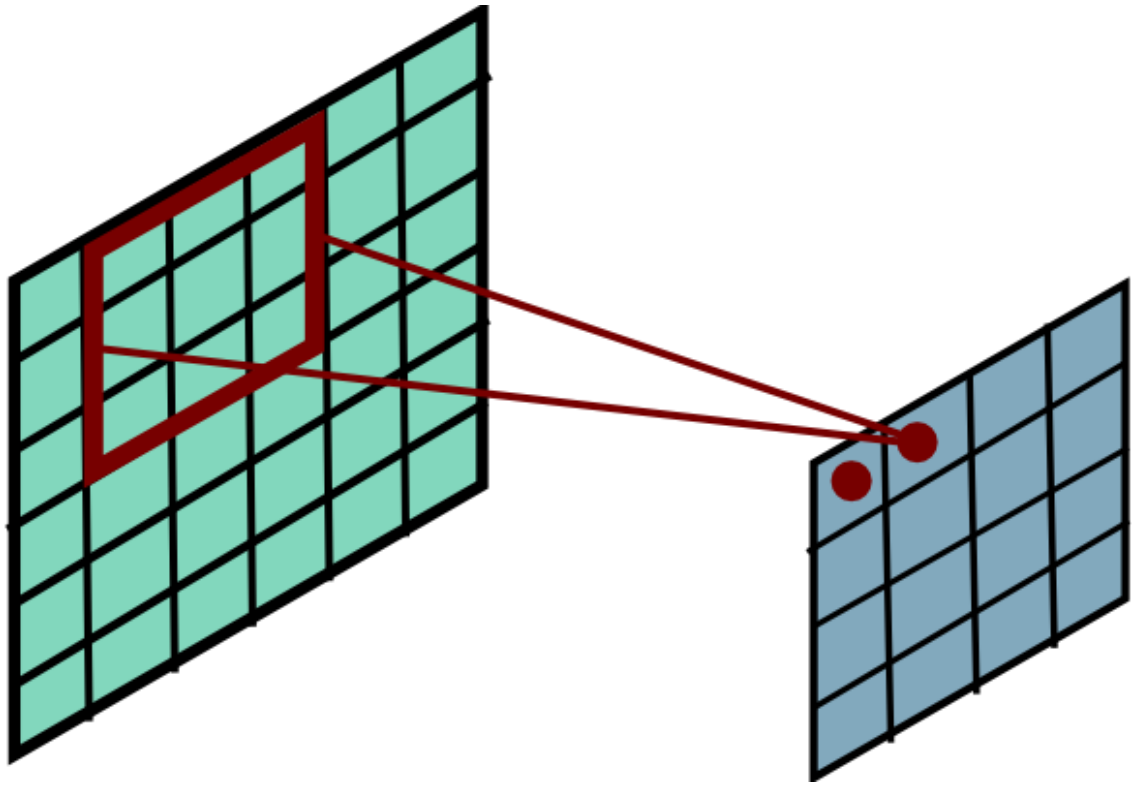
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

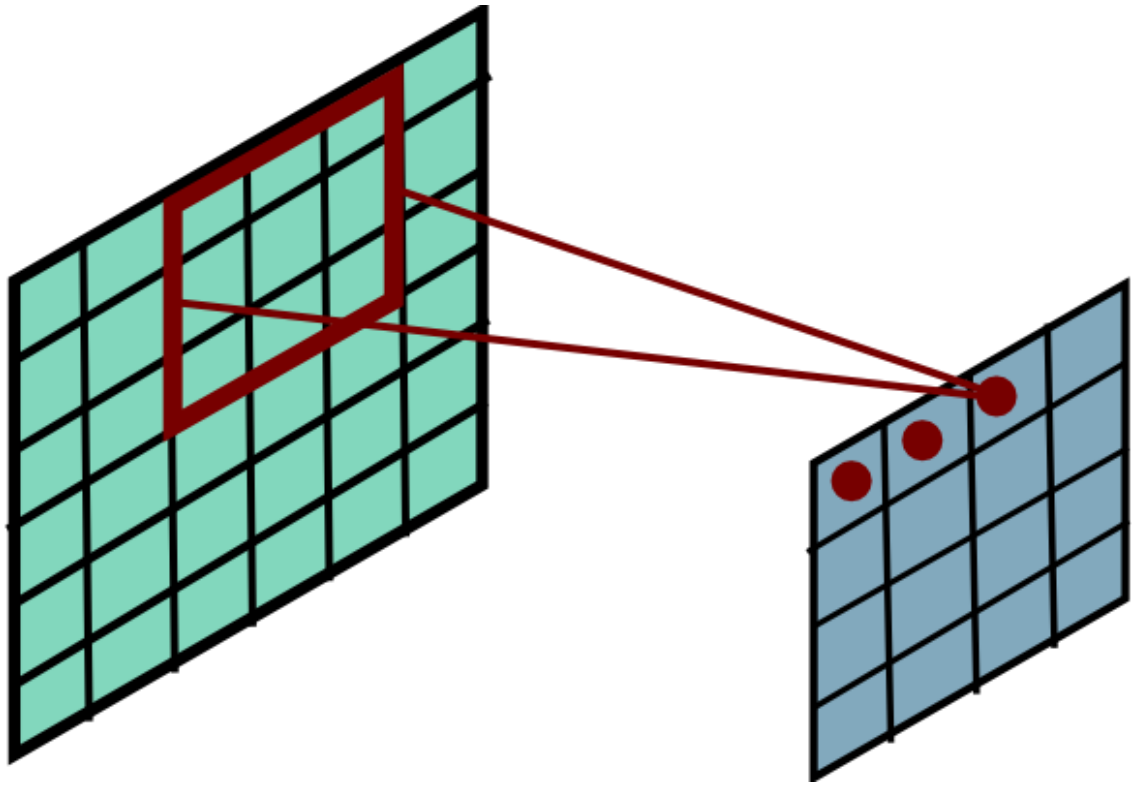
Convolutional Layer



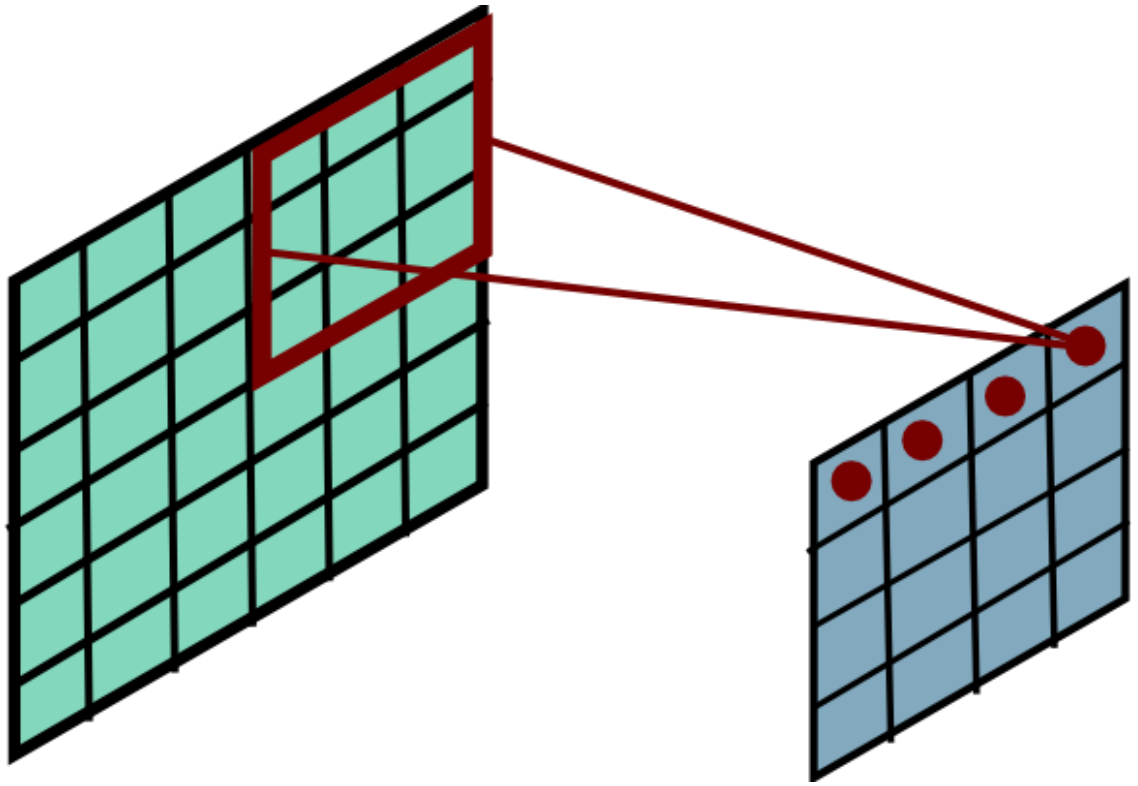
Convolutional Layer



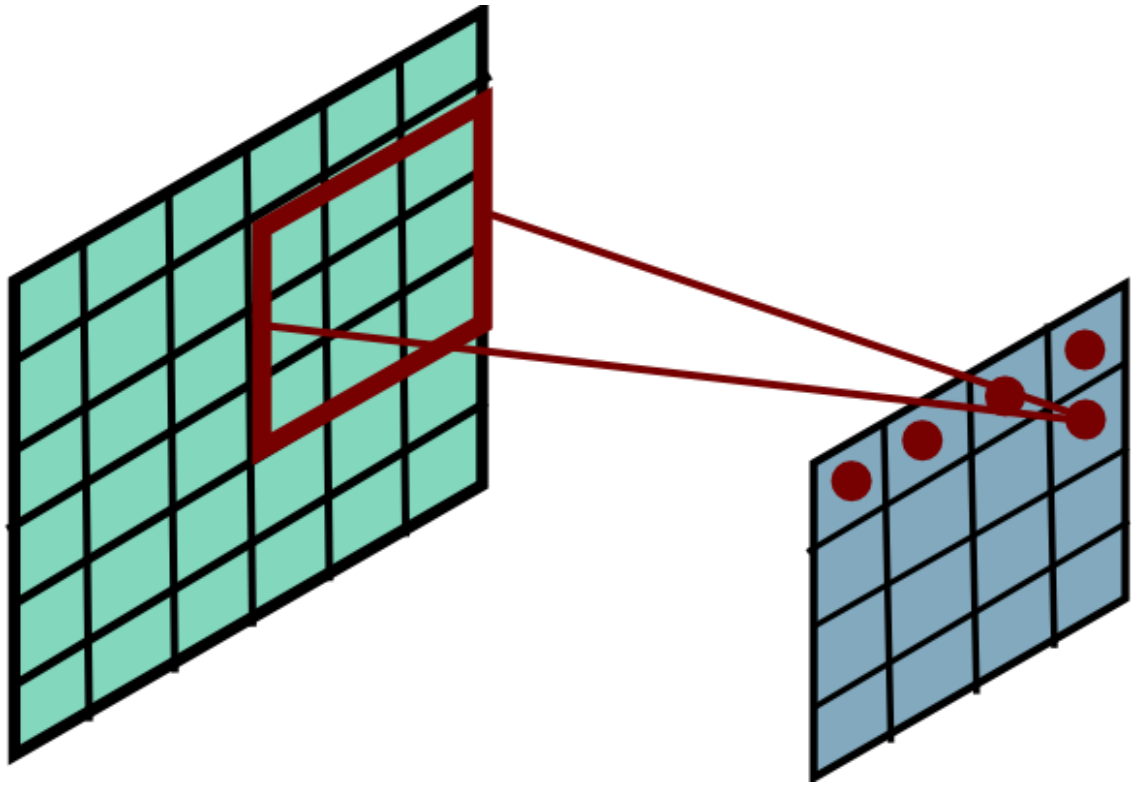
Convolutional Layer



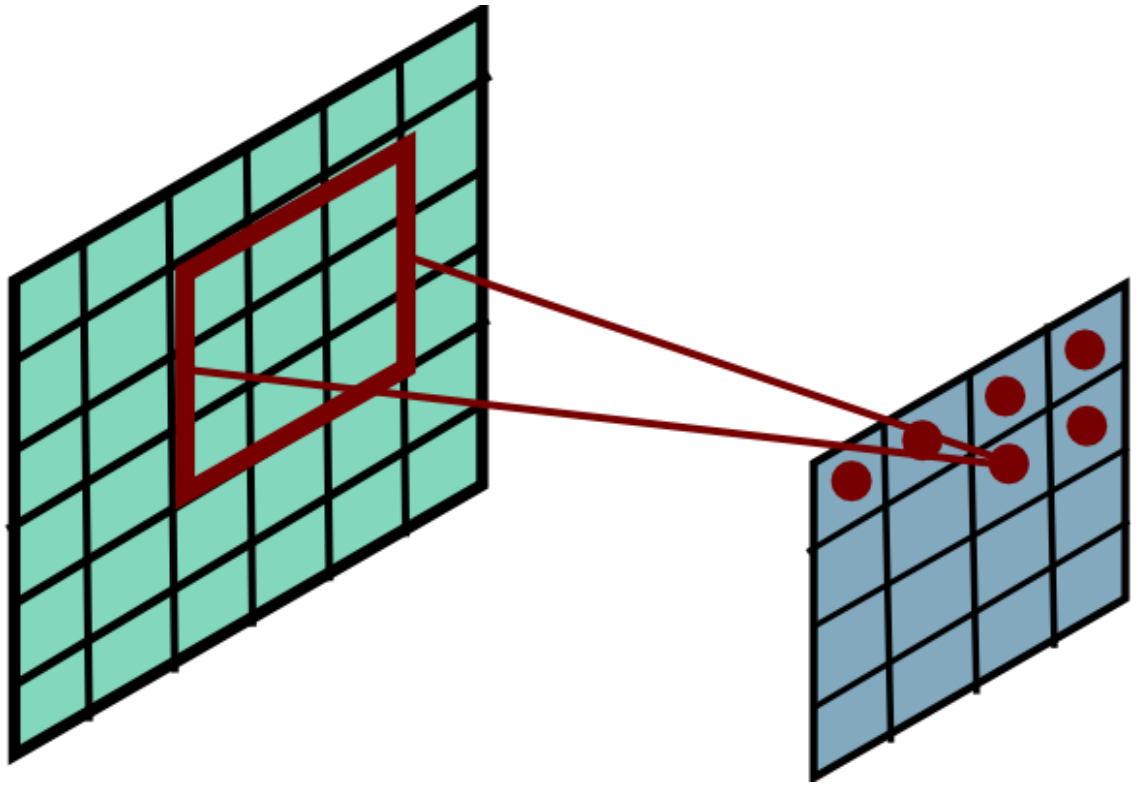
Convolutional Layer



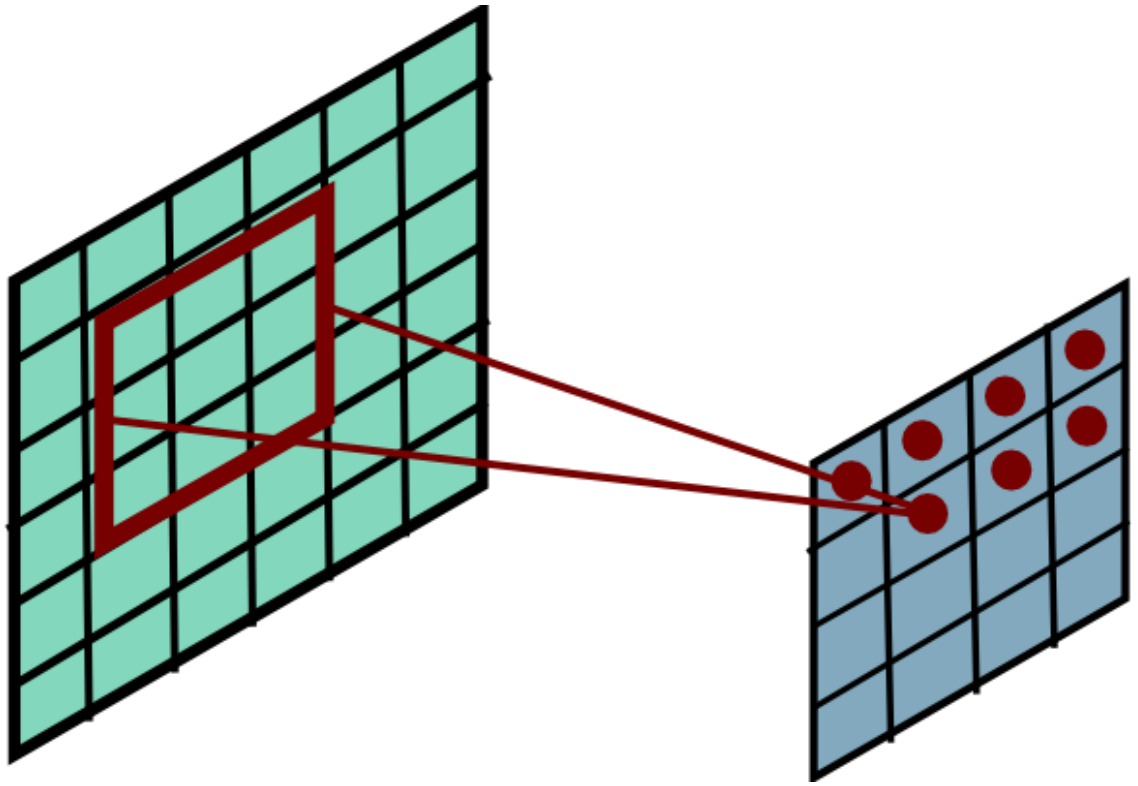
Convolutional Layer



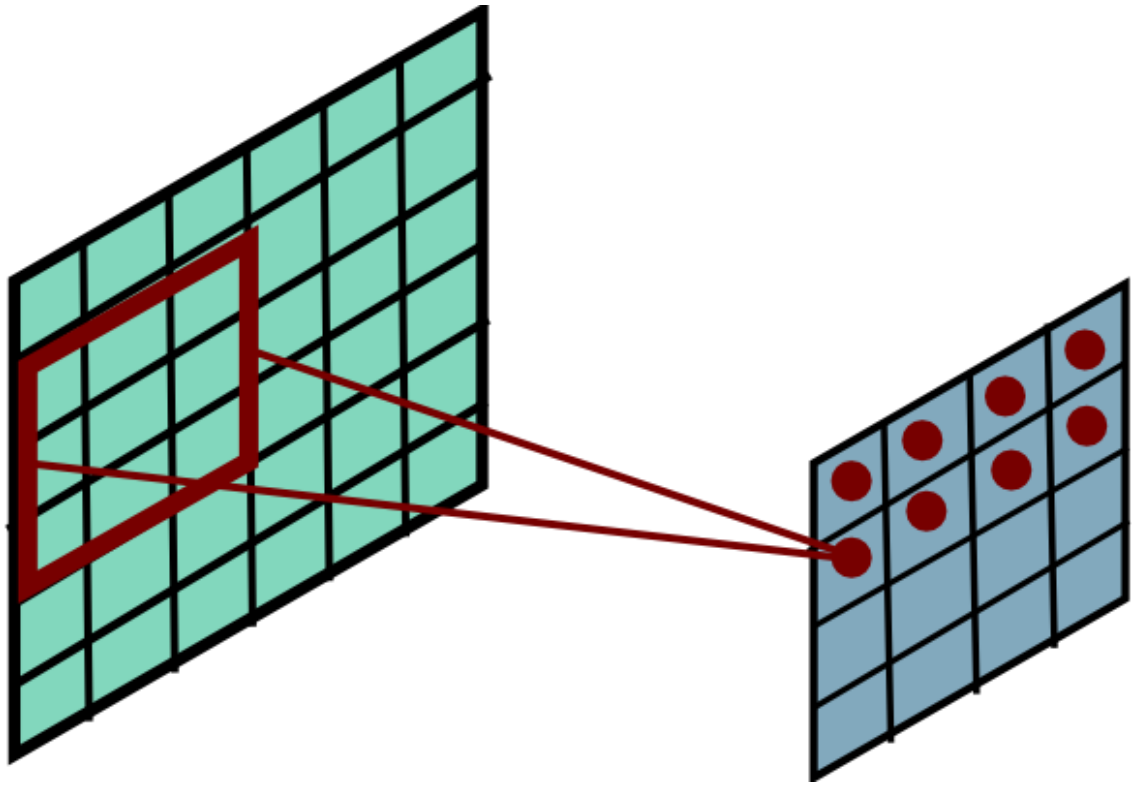
Convolutional Layer



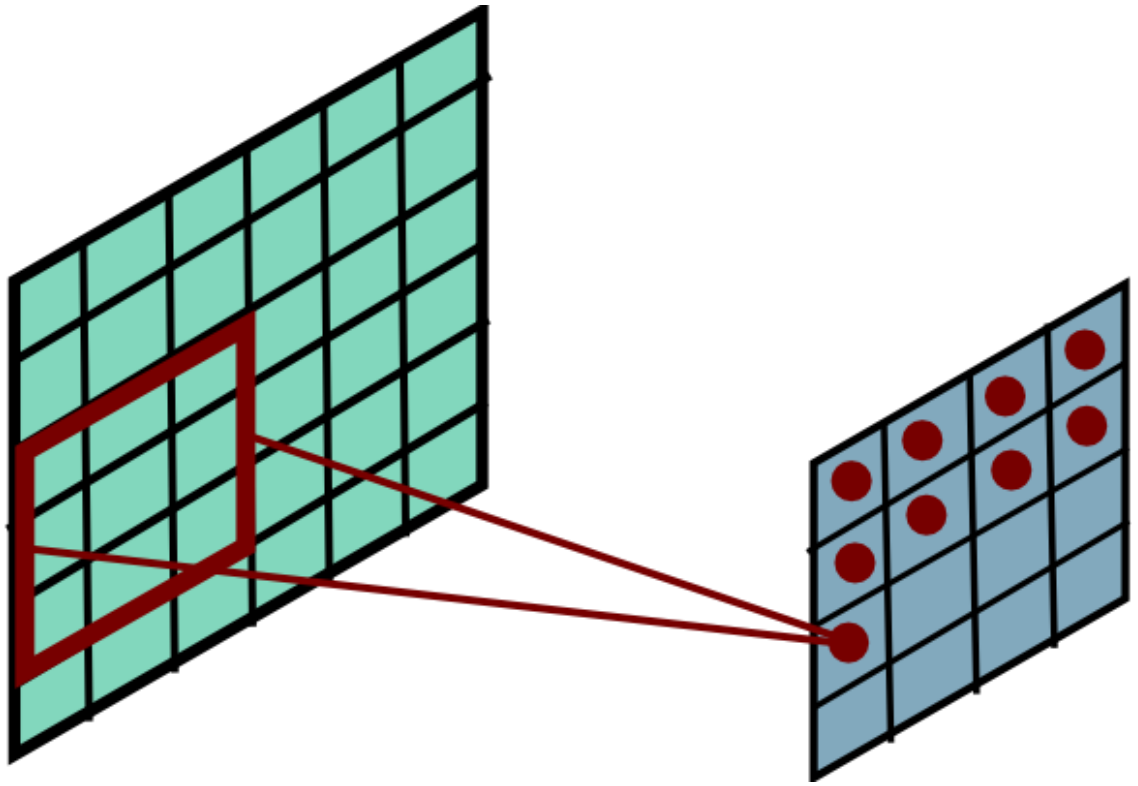
Convolutional Layer



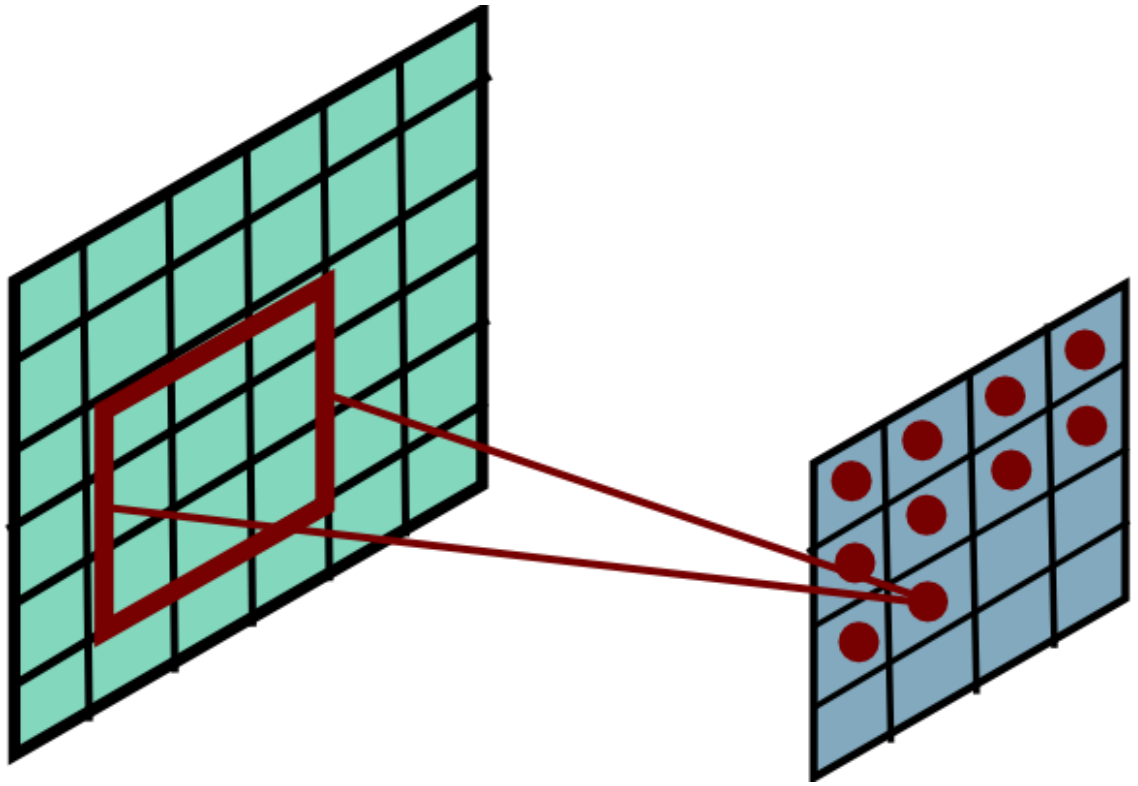
Convolutional Layer



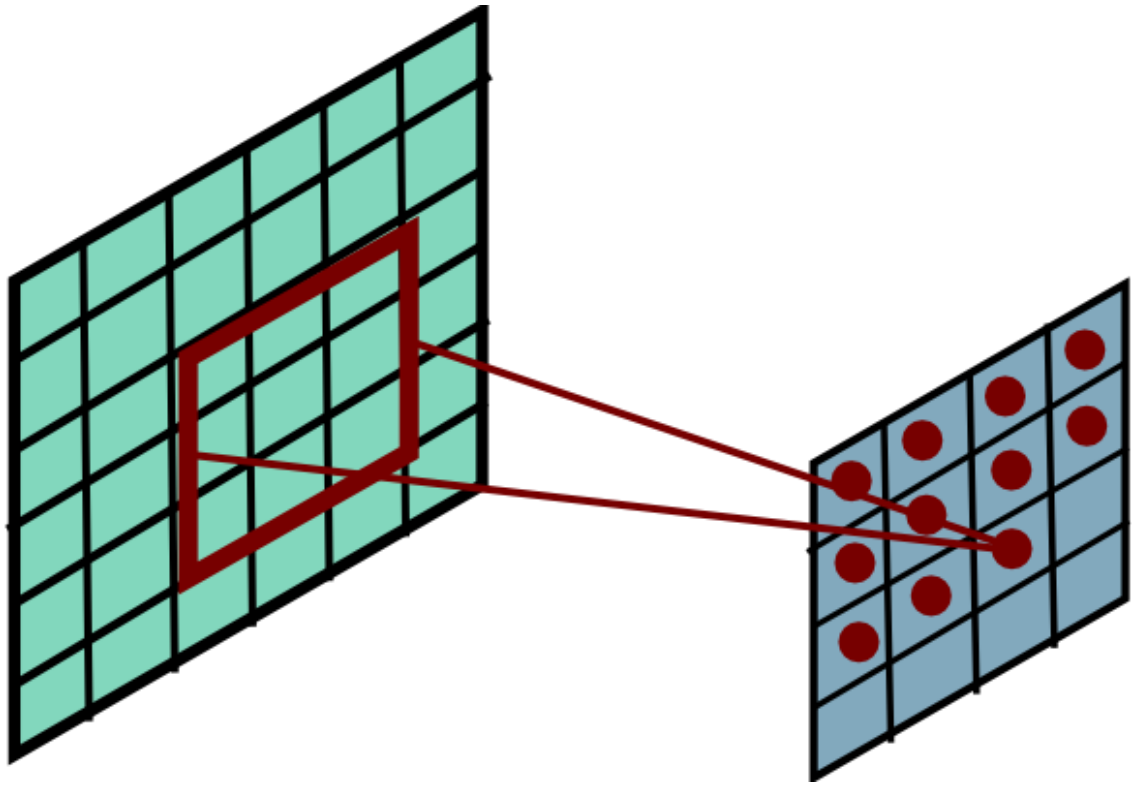
Convolutional Layer



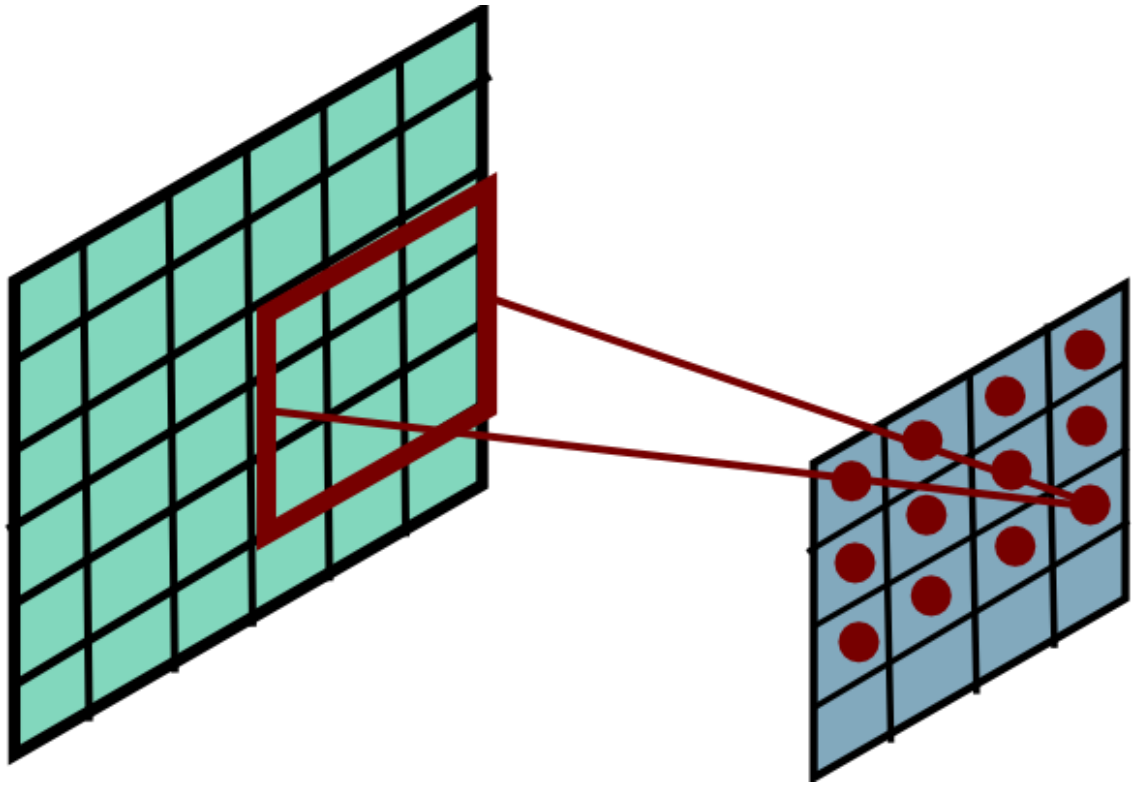
Convolutional Layer



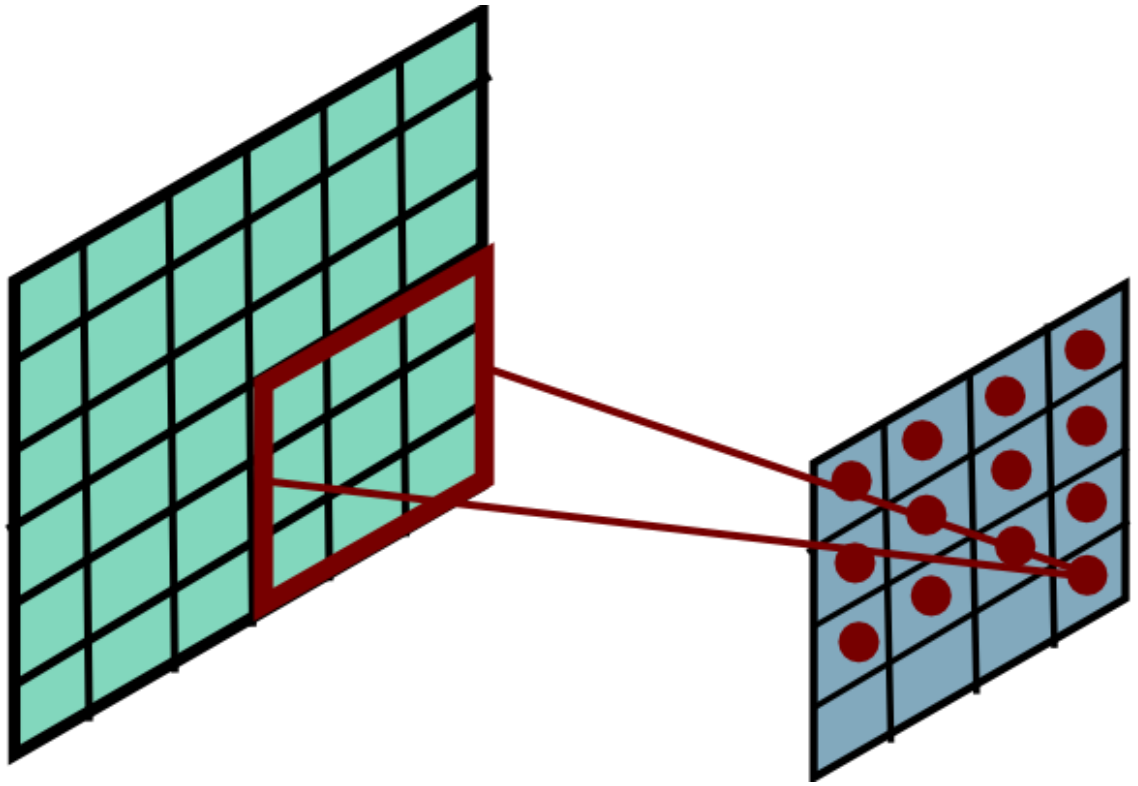
Convolutional Layer



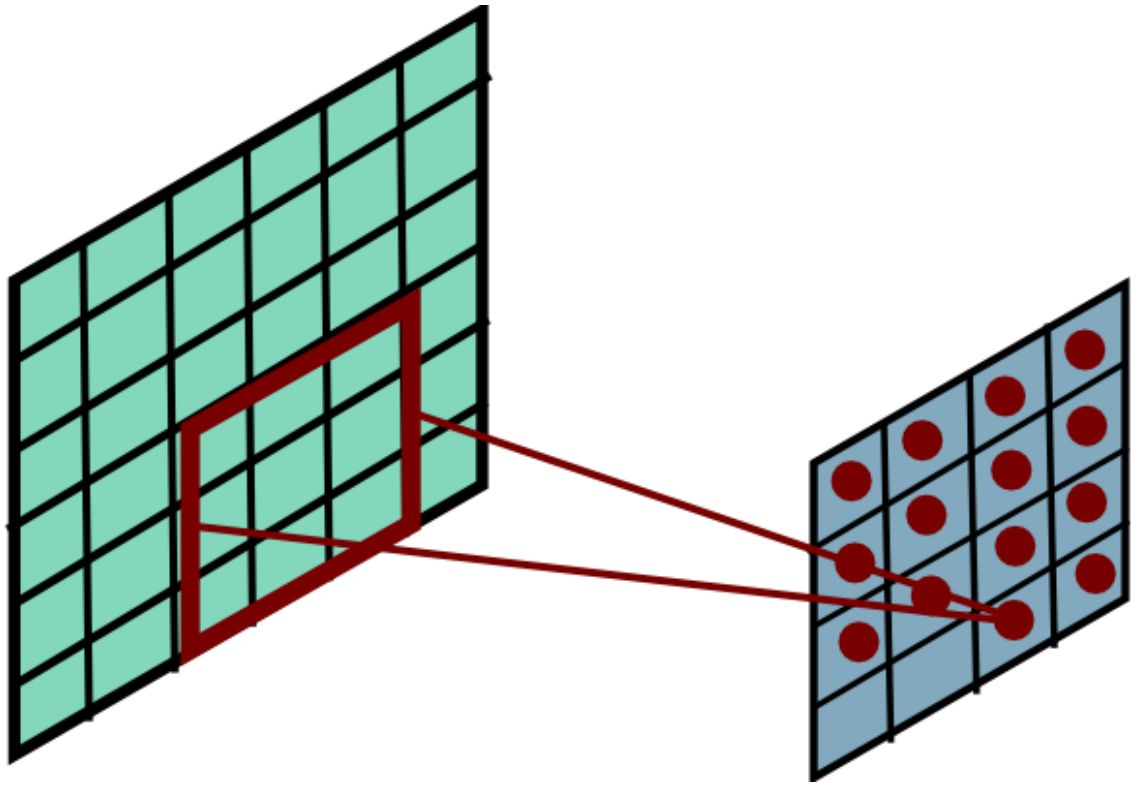
Convolutional Layer



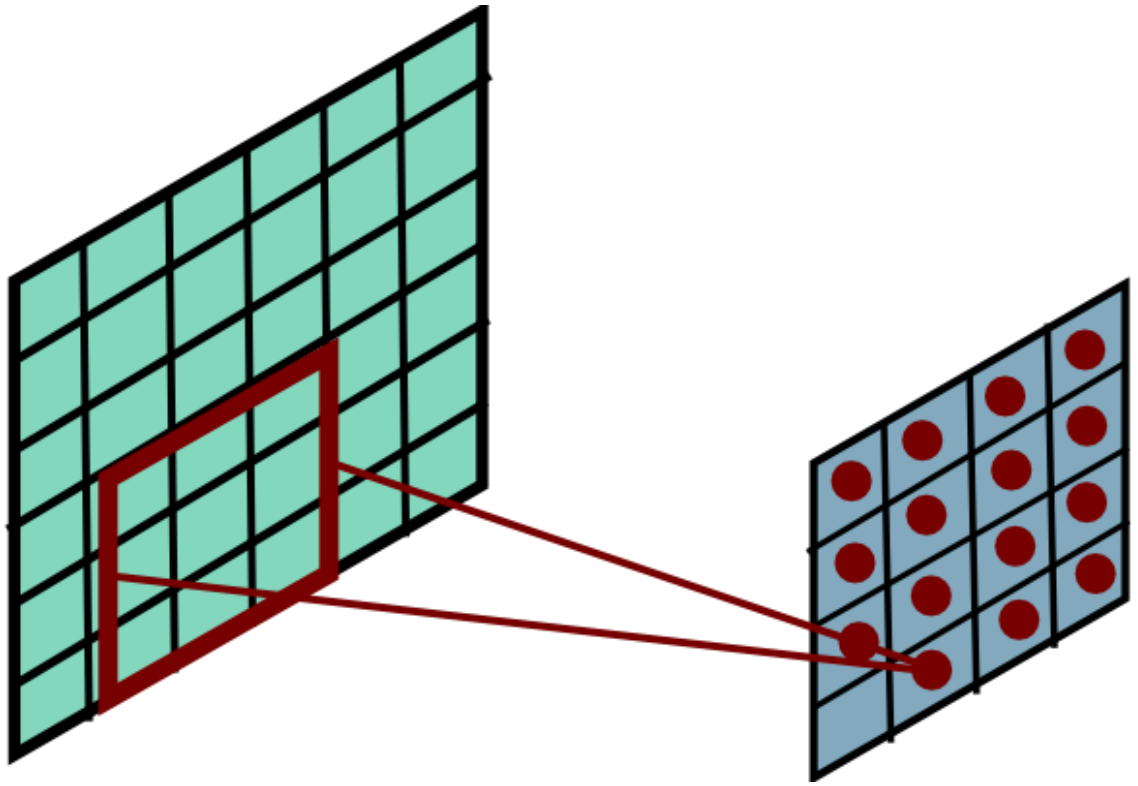
Convolutional Layer



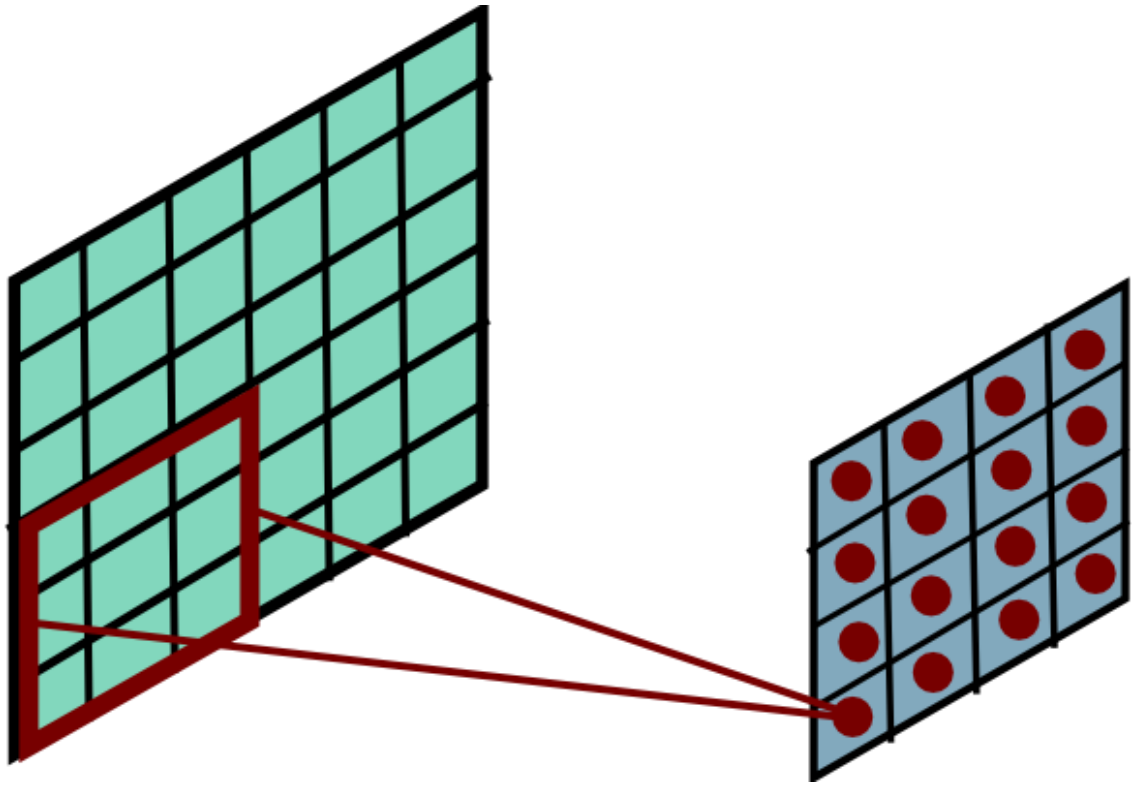
Convolutional Layer



Convolutional Layer



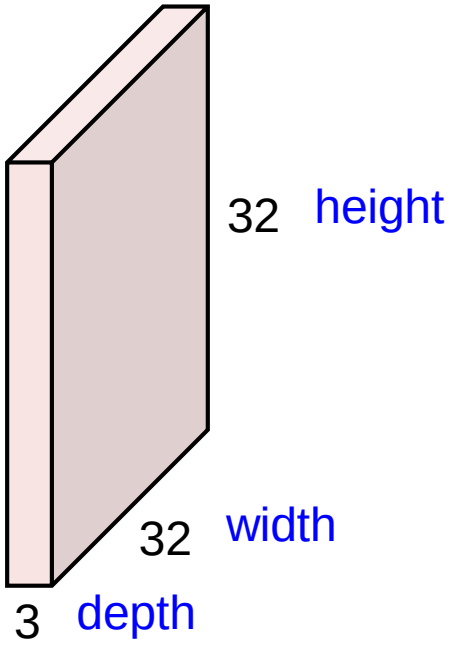
Convolutional Layer



Mathieu et al. “Fast training of CNNs through FFTs” ICLR 2014

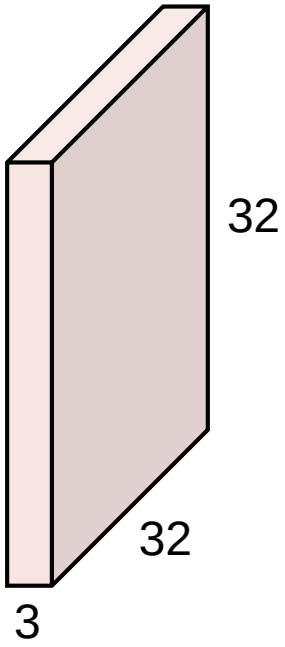
Convolution Layer

32x32x3 image -> preserve spatial structure

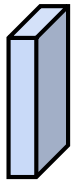


Convolution Layer

32x32x3 image



5x5x3 filter

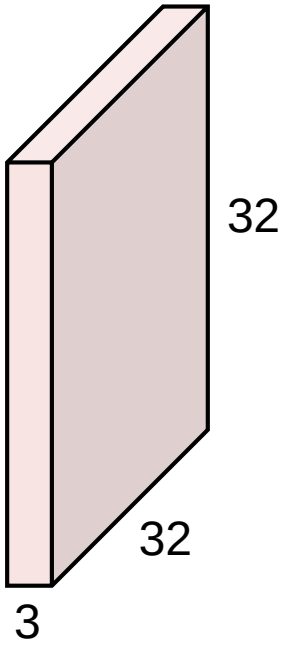


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

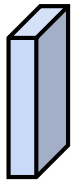
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

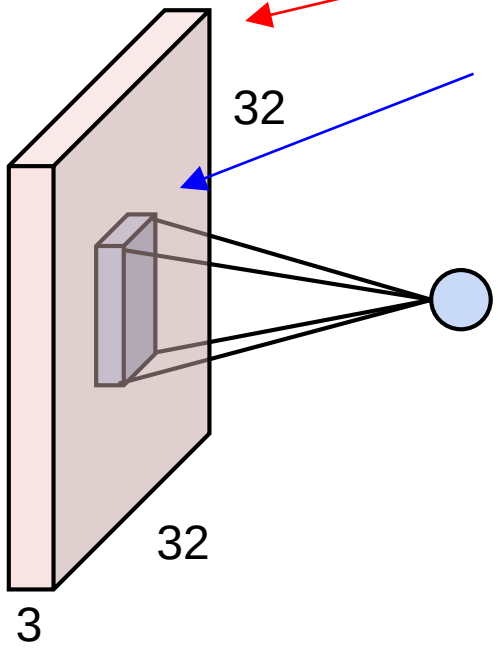


5x5x3 filter



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

Convolution Layer



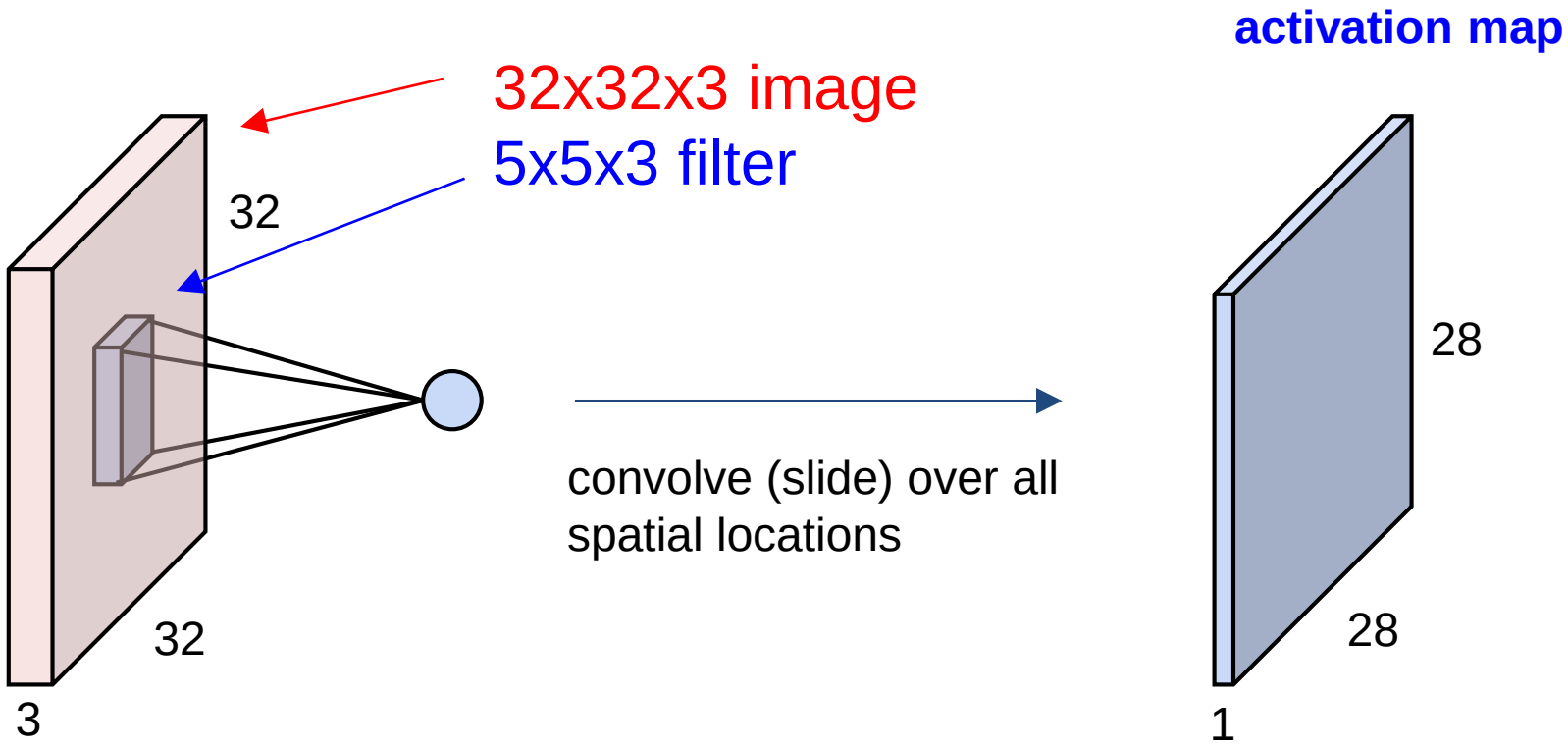
32x32x3 image

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)

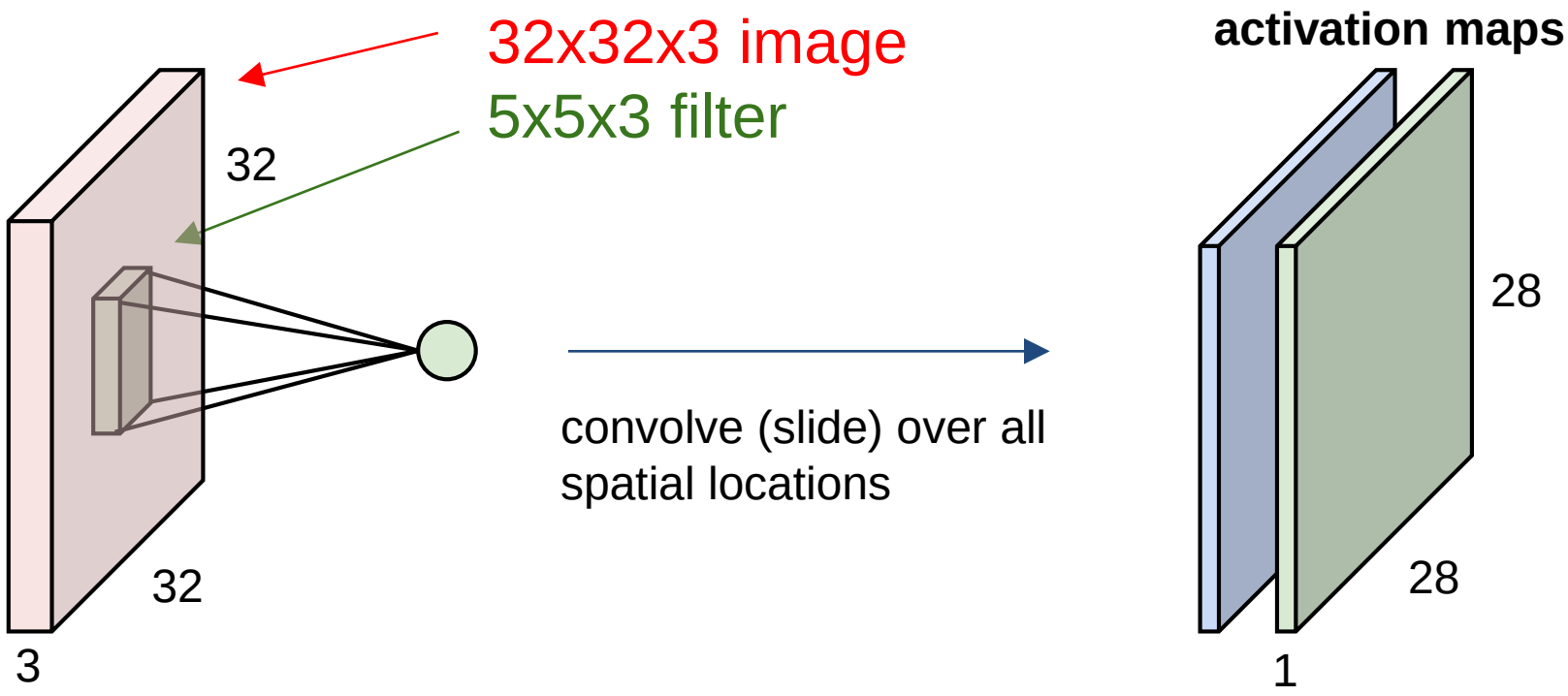
$$w^T x + b$$

Convolution Layer – red filter (kernel)

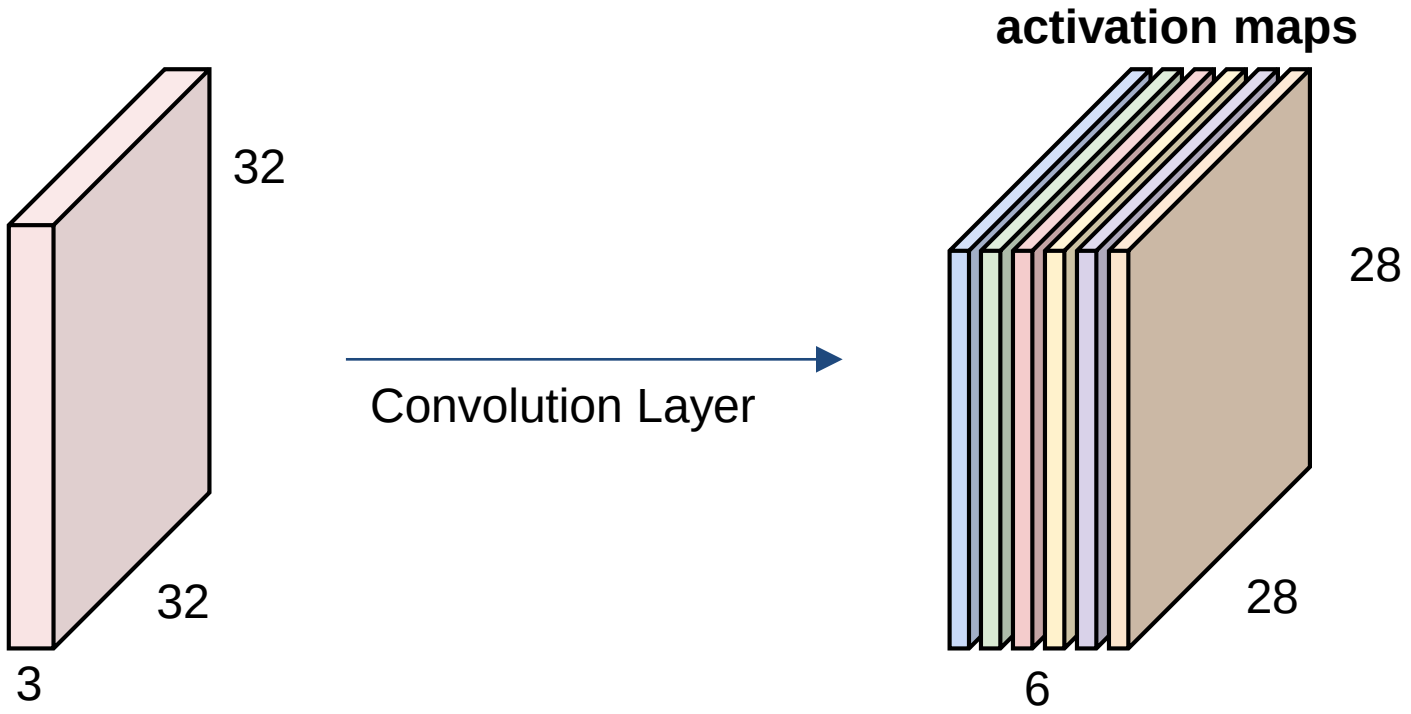


Convolution Layer

consider a second, green filter



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!

Power of Convolution

- Describe a “system” (or operation) with a very simple function (impulse response).
- Determine the output by convolving the input with the impulse response

Example Multi-Dimensional Convolution

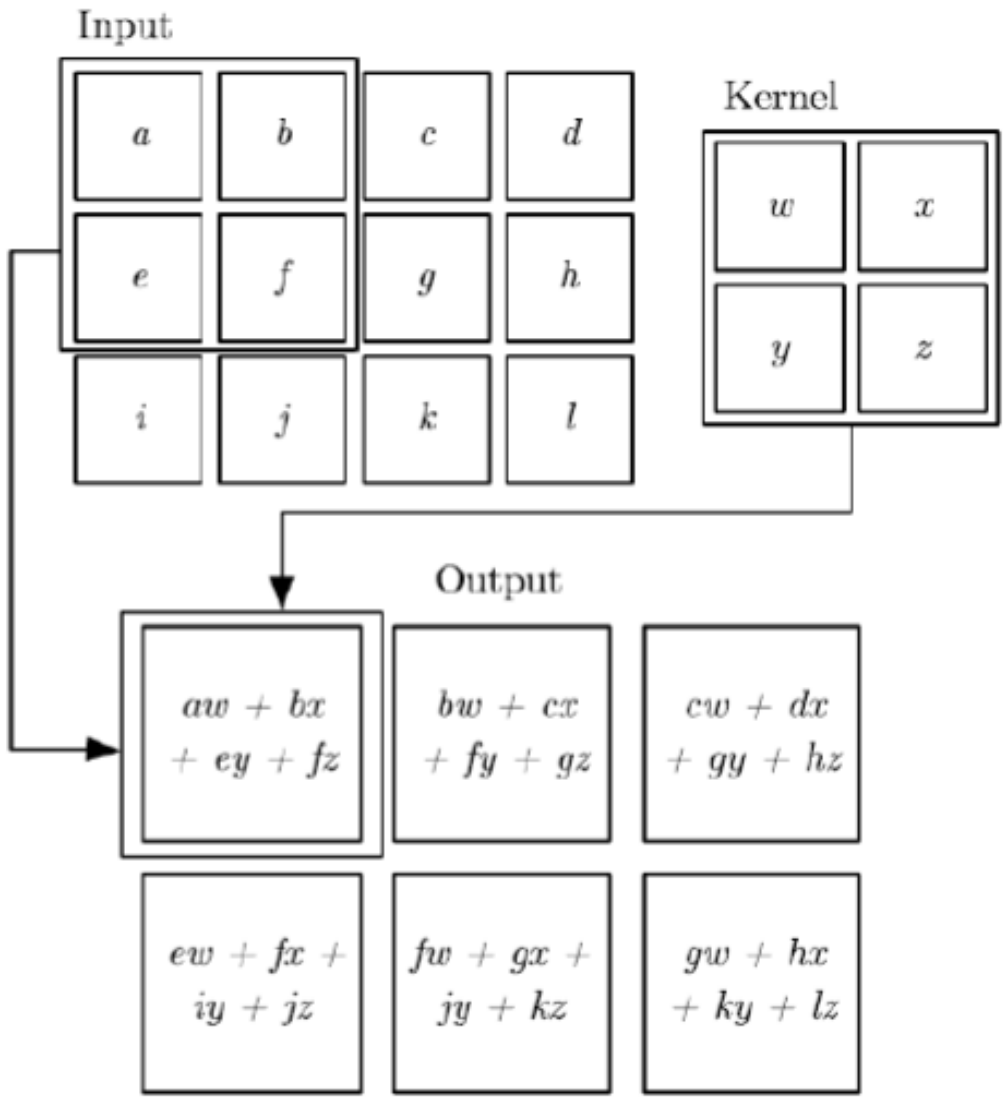
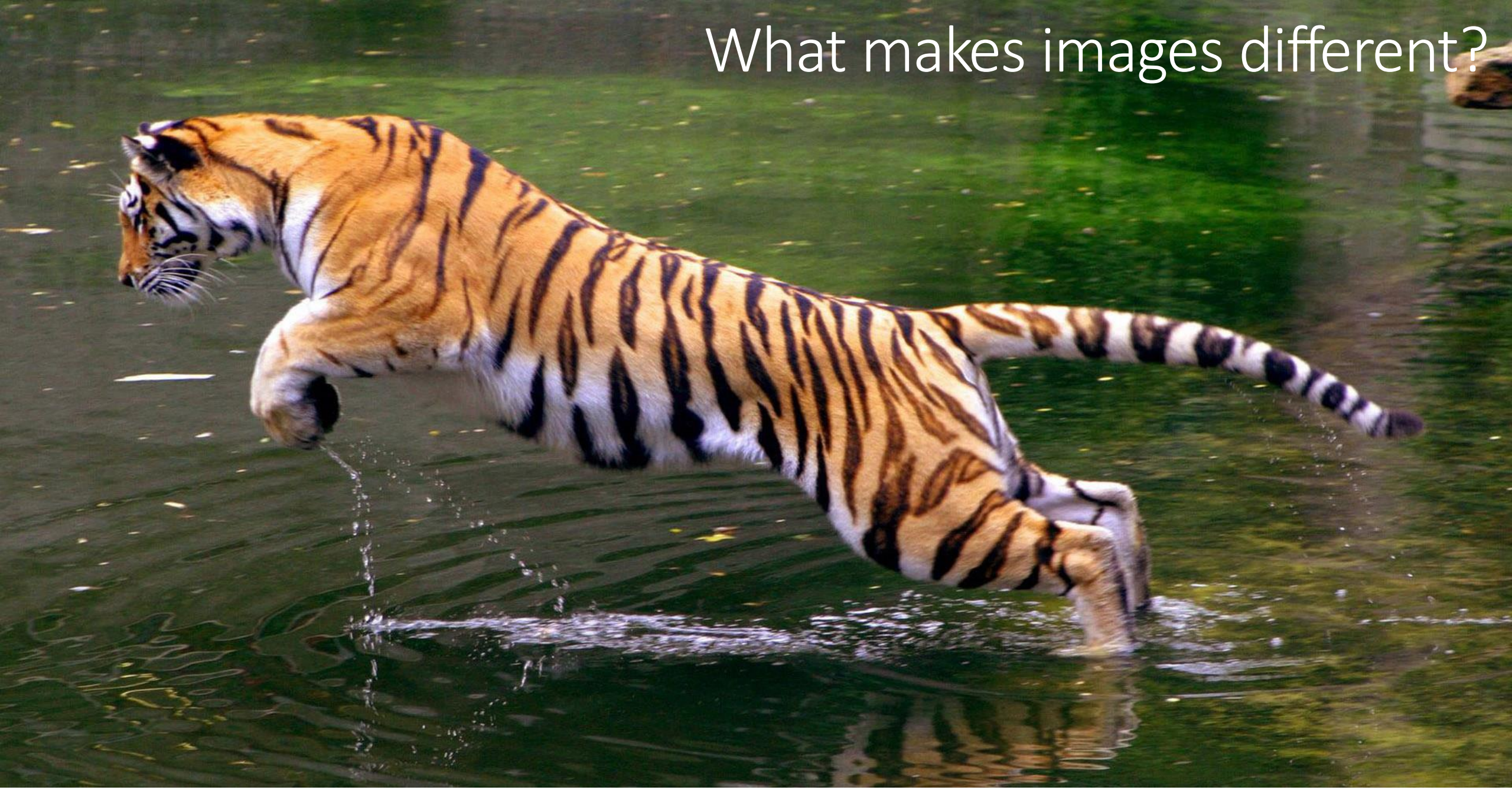
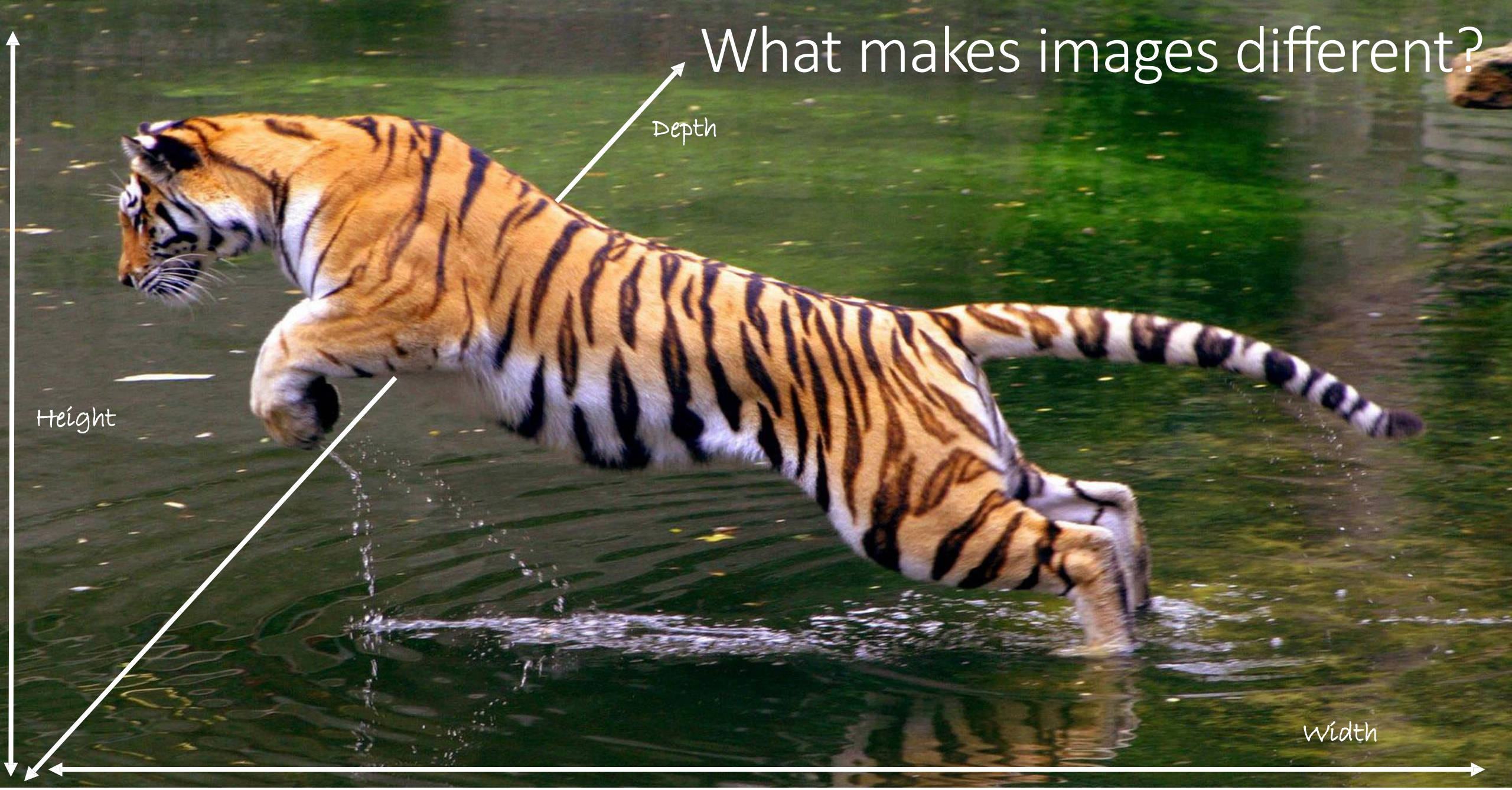


Figure: Bengio et al., "Deep Learning", MIT Press, in prep.

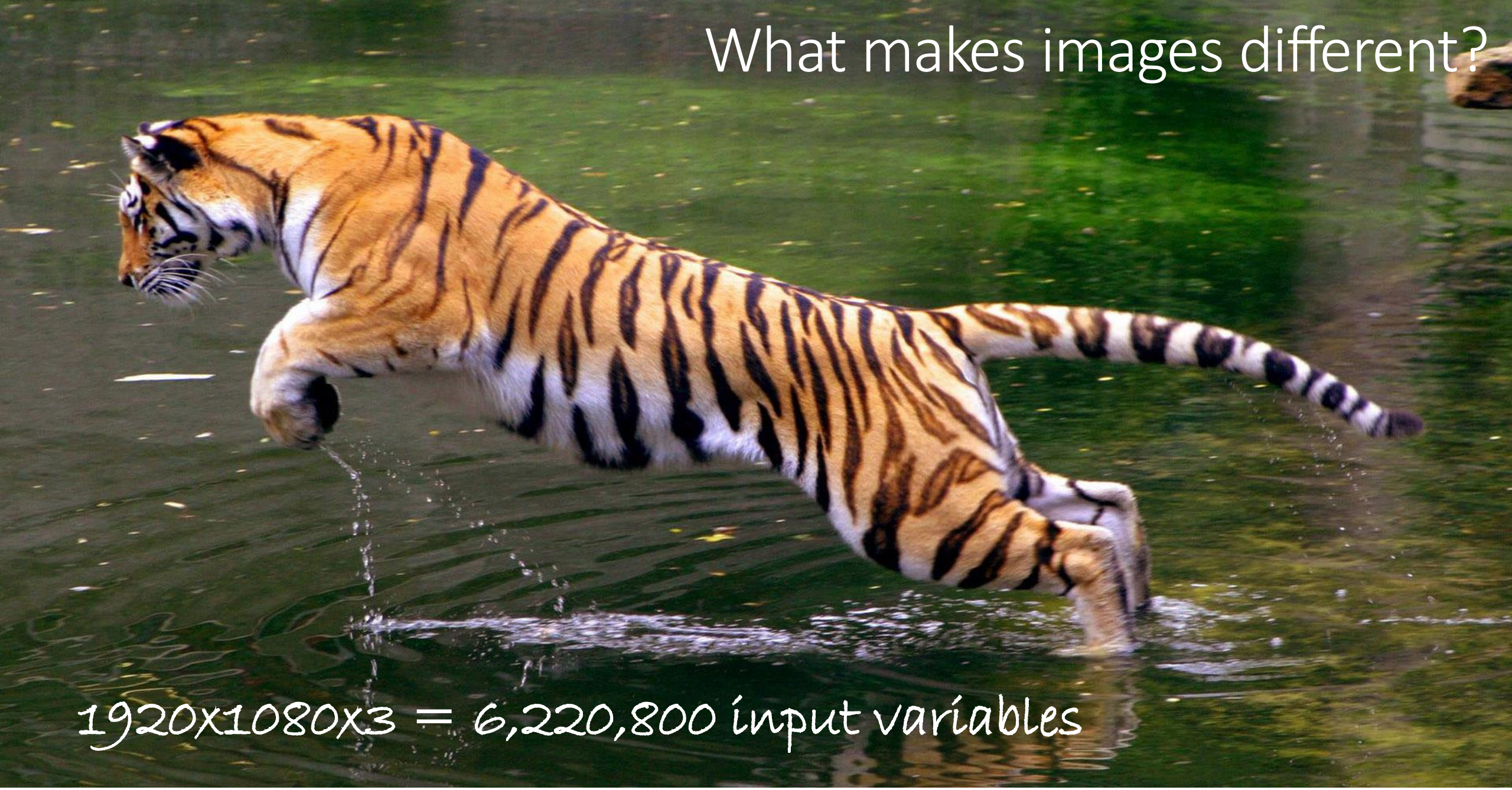
What makes images different?



What makes images different?

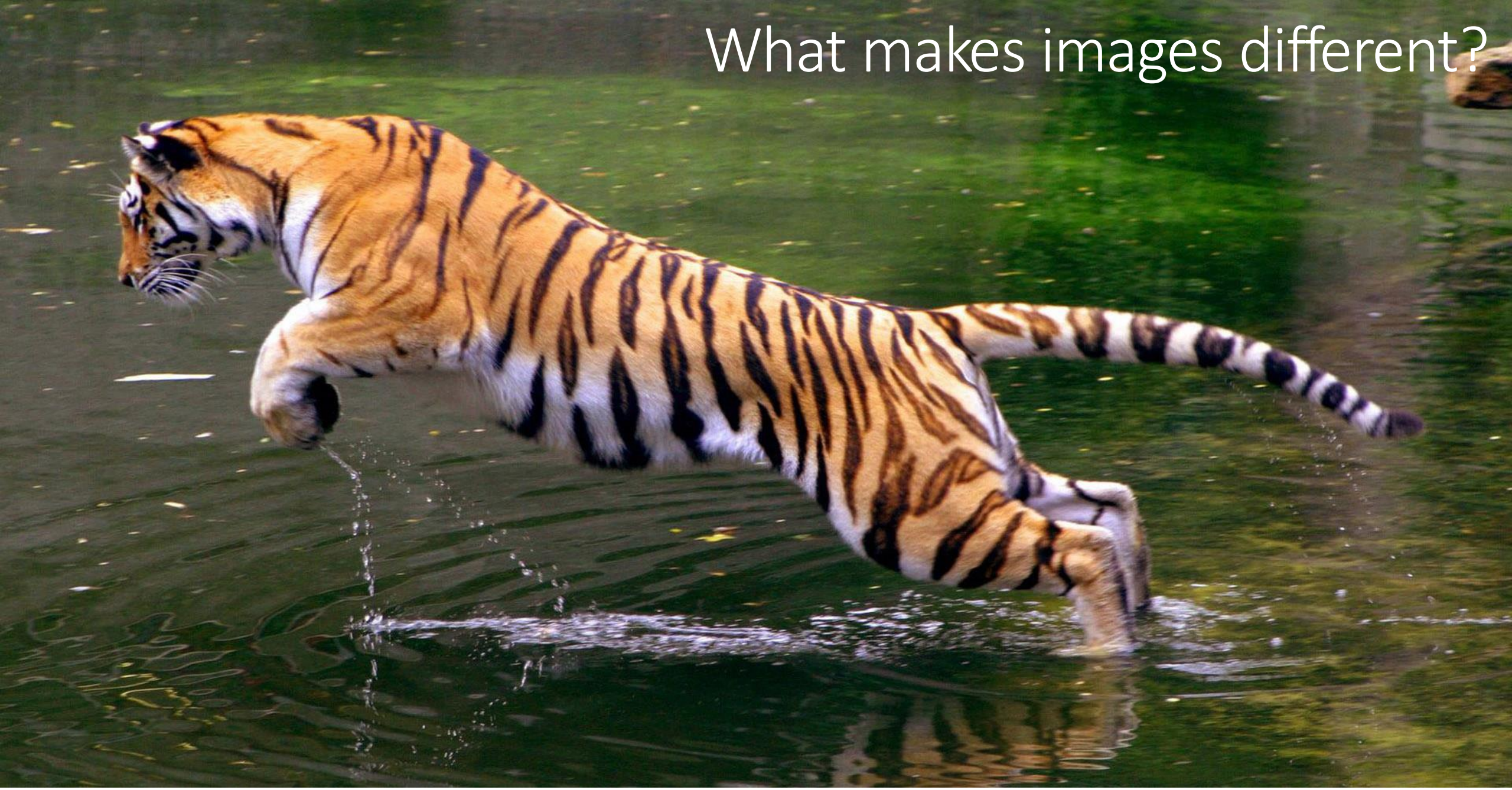


What makes images different?



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 image amounts to ~200K input variables
  - 1-layered NN with 1,000 neurons → 200 million parameters
- Images are stationary signals → 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

Convolutional Neural Networks

- Question: Spatial structure?
  - Answer: Convolutional filters
- Question: Huge input dimensionalities?
  - Answer: Parameters are shared between filters
- Question: Local variances?
  - Answer: Pooling

CNN Layers

- Stages of CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood
- Pooling of a rectangular region:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - …

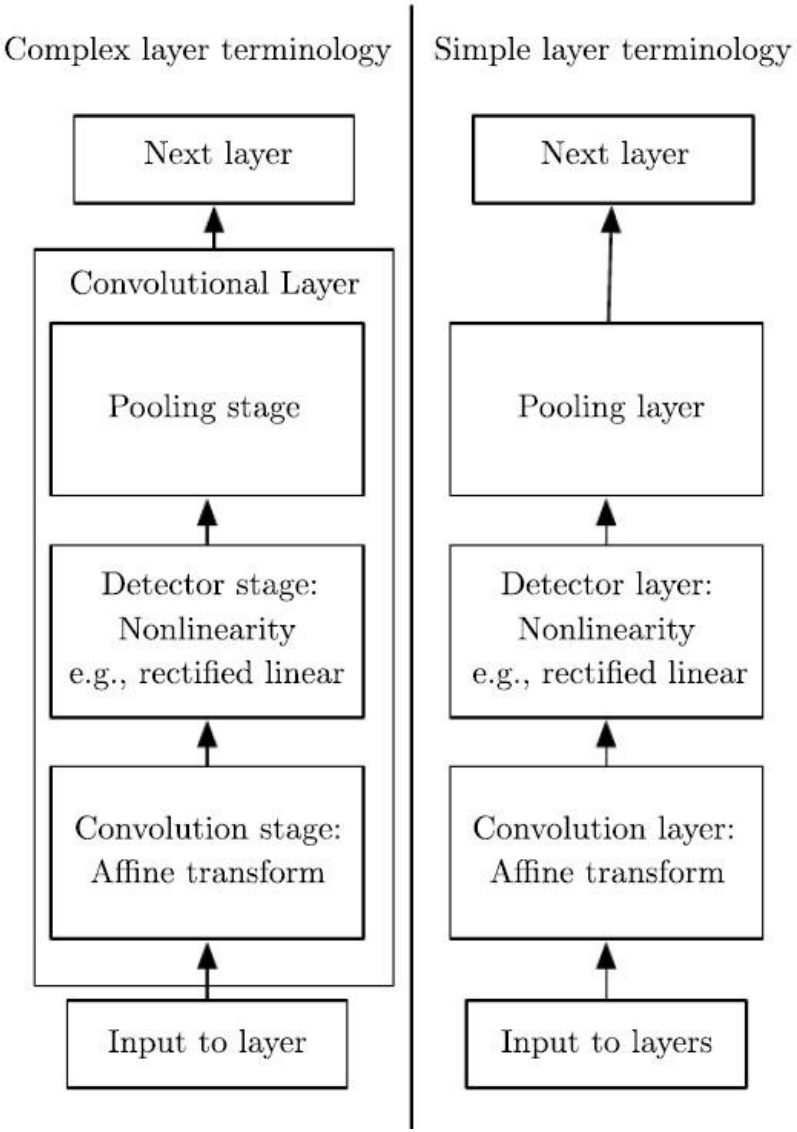
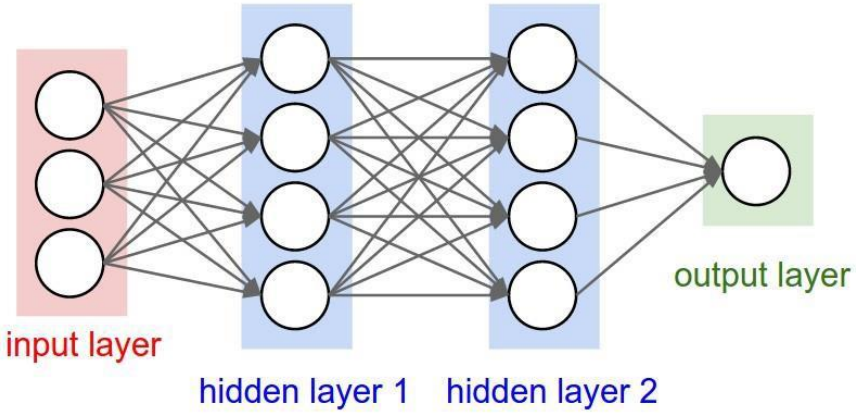
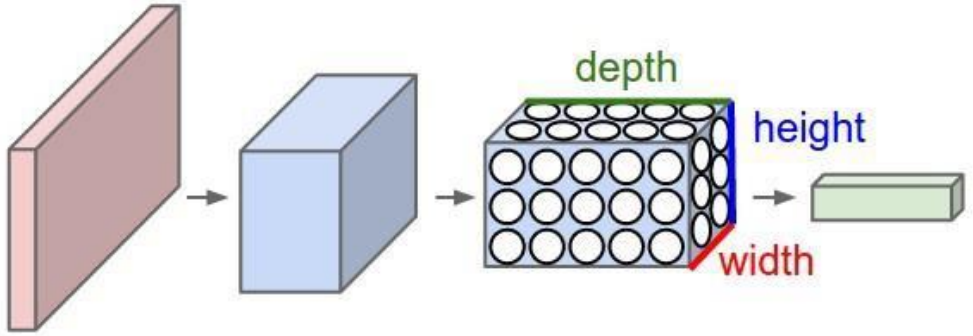


Figure: Bengio et al., "Deep Learning", MIT Press, in prep.

Deep Learning Multilayered Perceptron

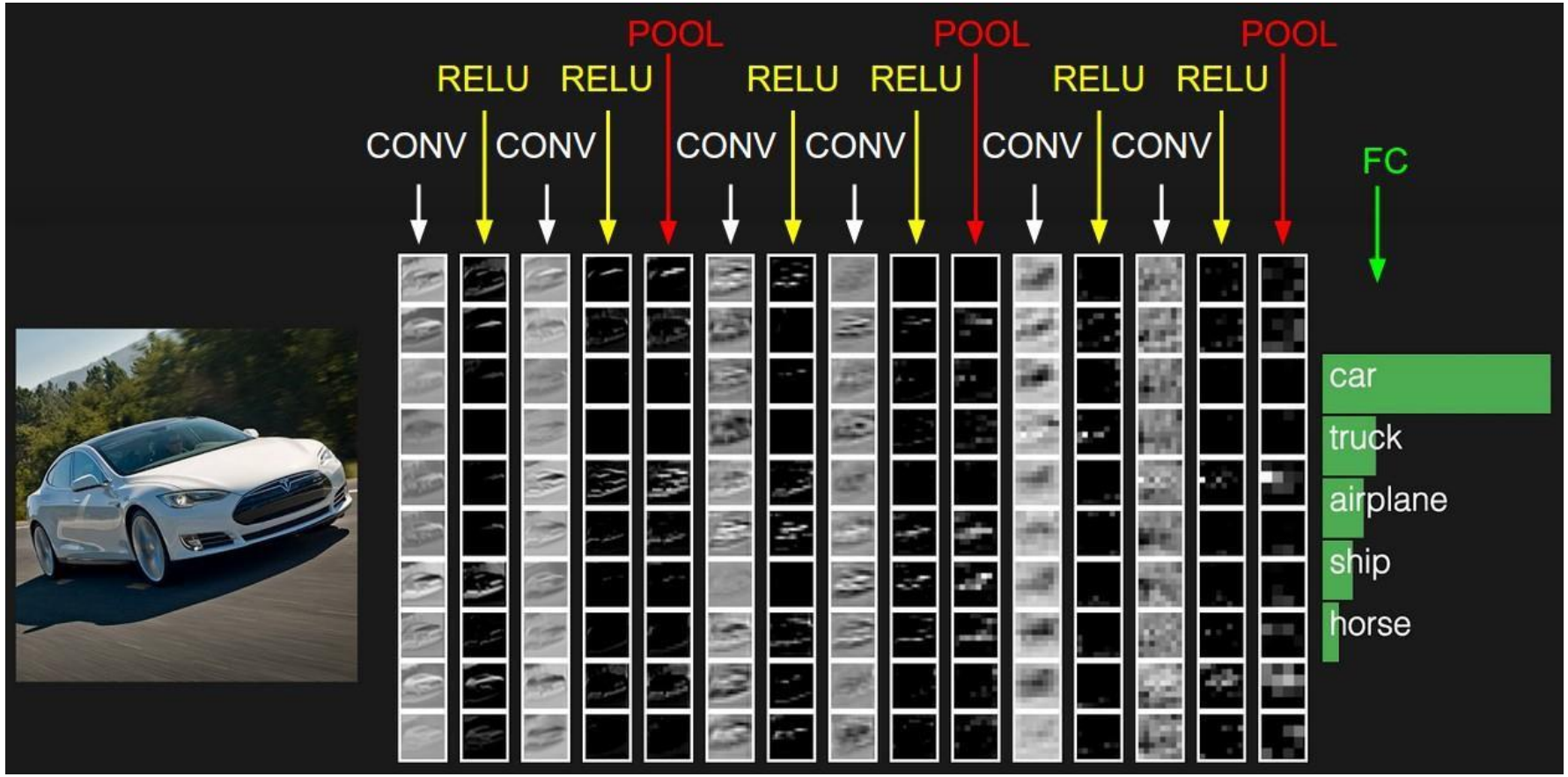


CNN



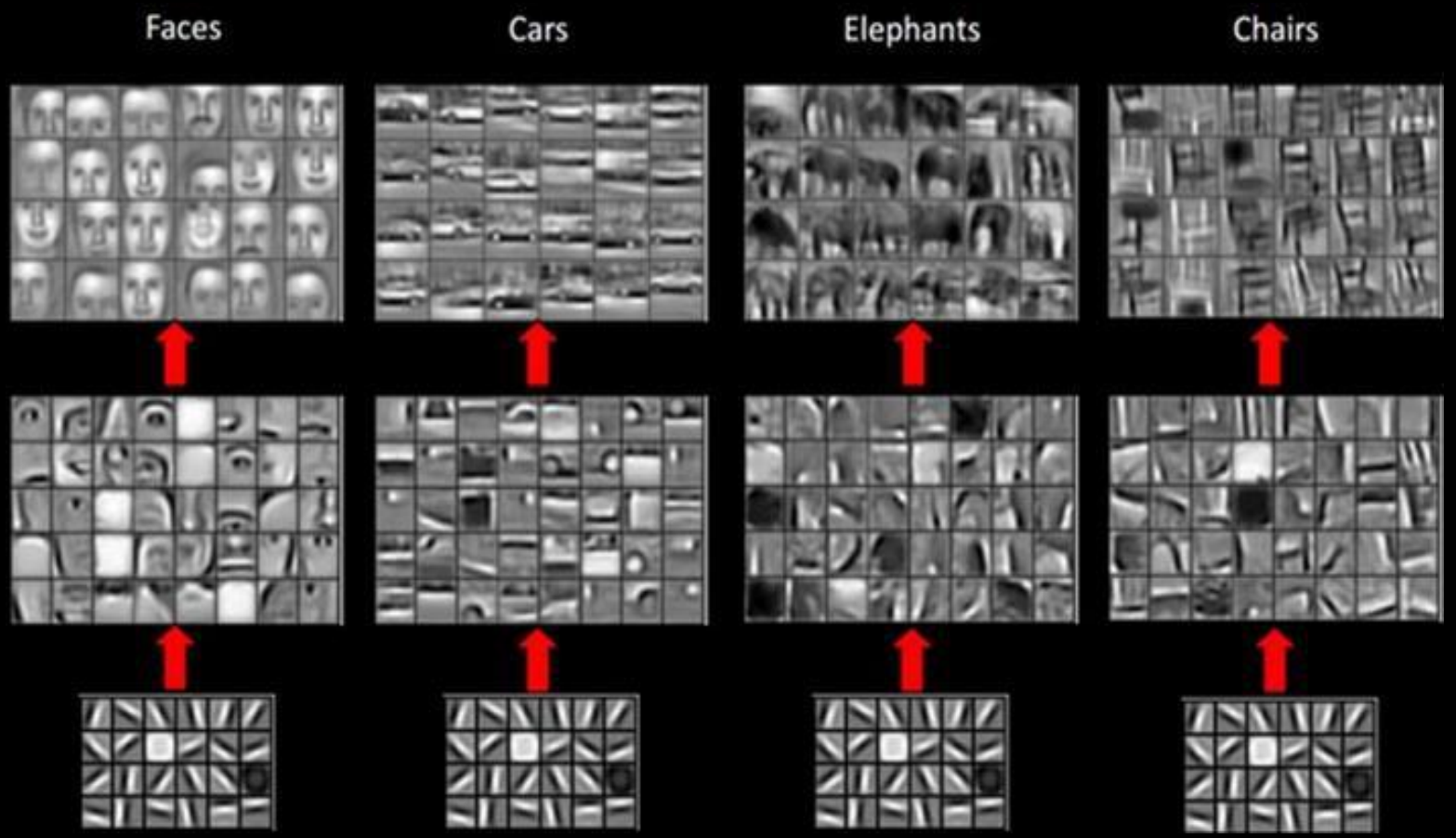
http://cs231n.github.io/convolutional-networks/

DL Architecture with Rectified Linear Unit (ReLU) Activation Function, Pooling and Fully Connected (FC) Layers



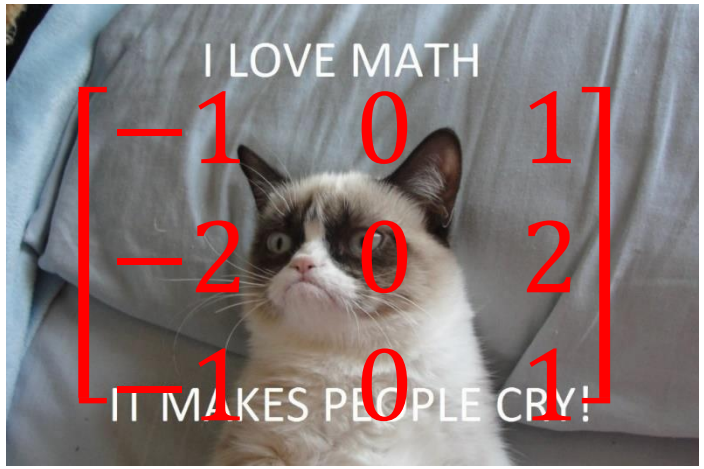
http://cs231n.github.io/convolutional-networks/

Preserving Spatial Structure

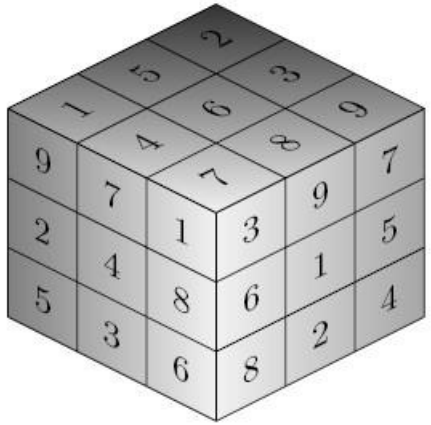


What Would a k-D Filter Look Like?

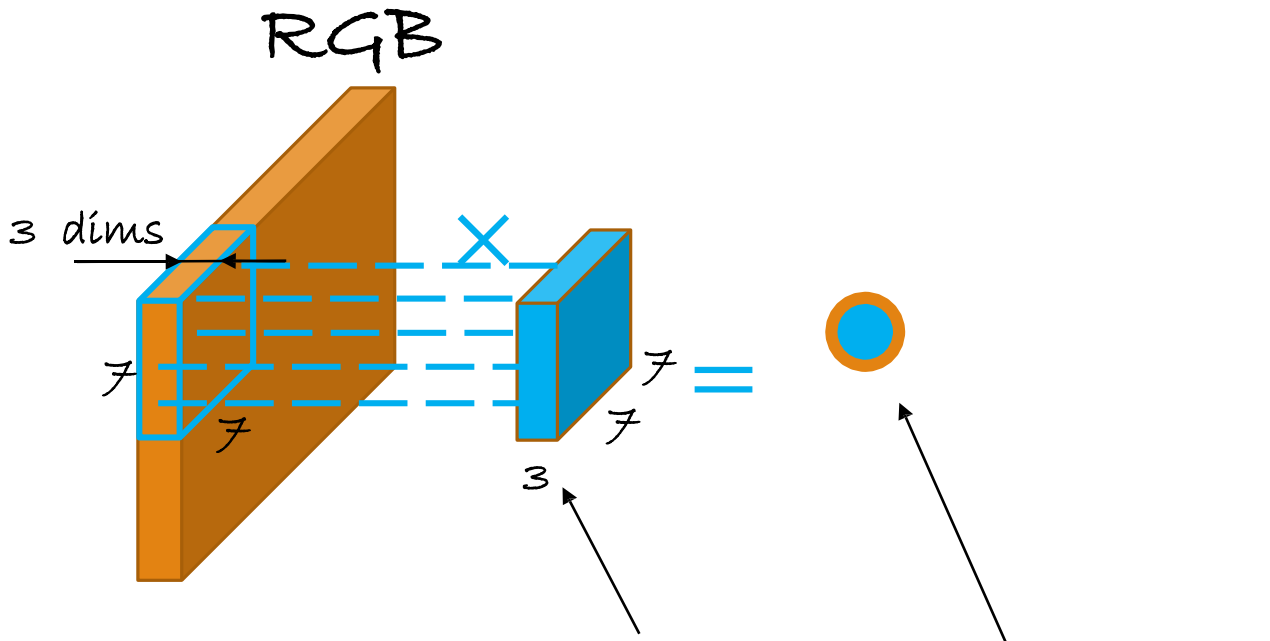
e.g. Sobel 2-D filter



$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Think in Space (1)

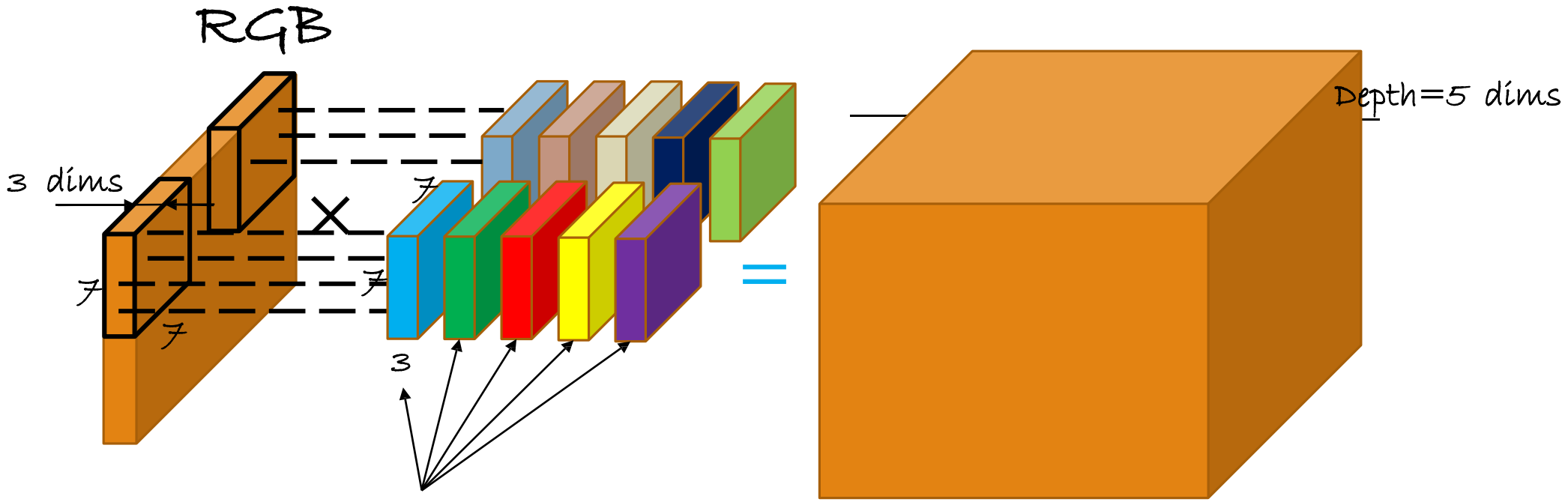


How many weights for this neuron?

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

Think in Space (2)

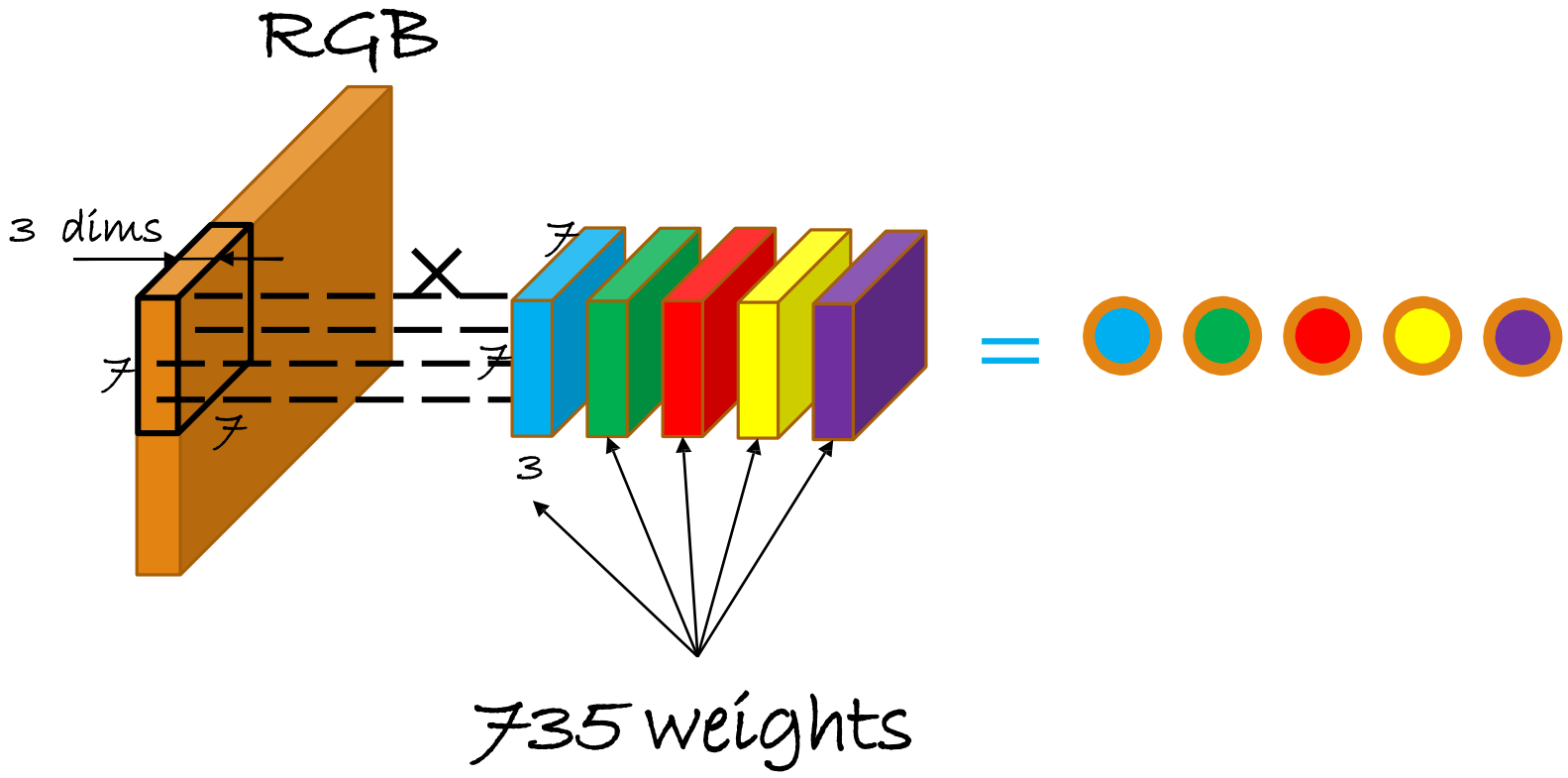
The activations of a hidden layer form a volume of neurons, not a 1-d "chain"
This volume has a depth 5, as we have 5 filters



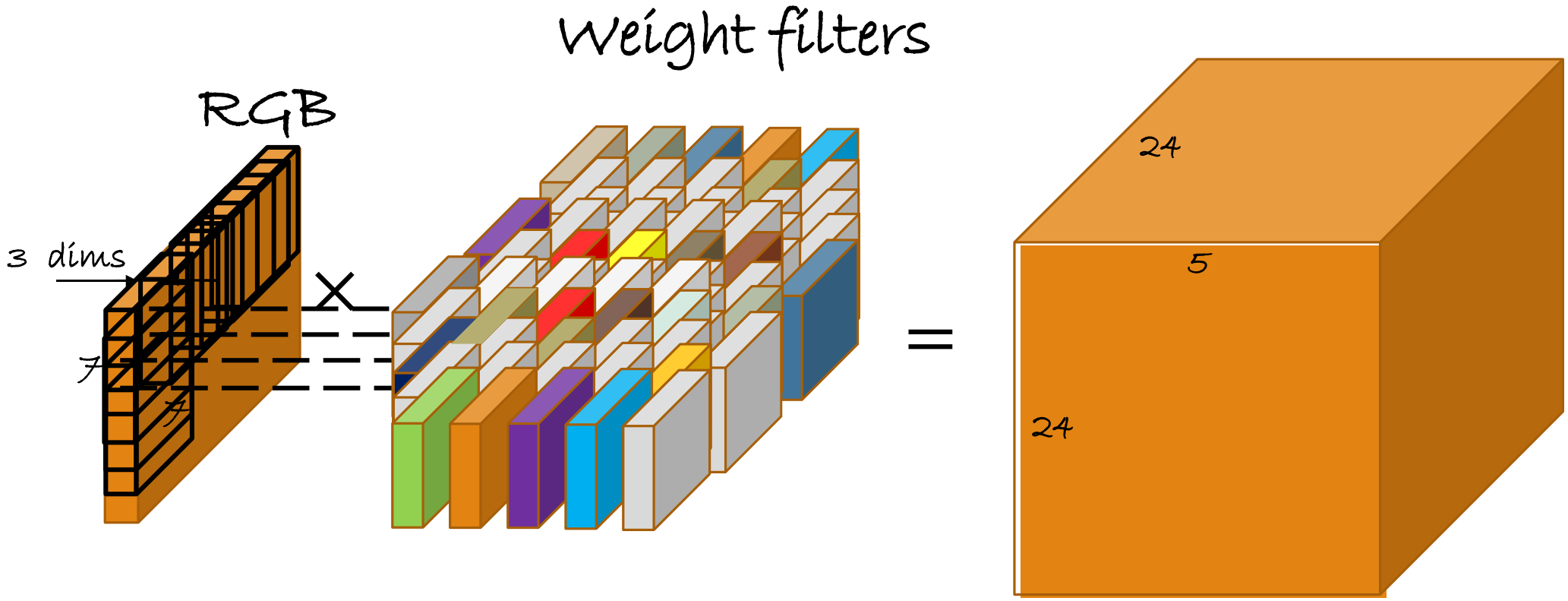
How many weights for these 5 neurons?

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

Again, Think in Space (147 x 5) = 735 weights



What About Cover the Full Image with Filters?



Assume the image is 30x30x3.

1 filter every pixel (stride =1)

How many parameters in total?

24 filters along the x axis

24 filters along the y axis

Depth of 5

$\times$ $7 * 7 * 3$ parameters per filter

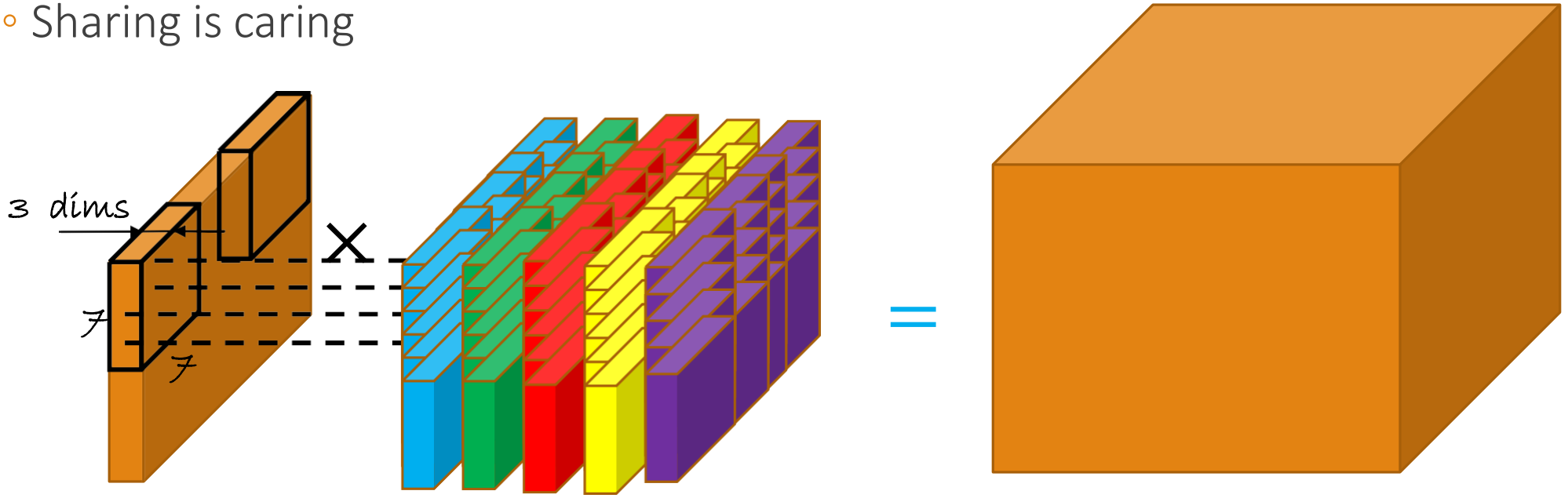
$423K$ parameters in total

Problem!

- Clearly, too many parameters
- With a only 30×30 pixels image and a single hidden layer of depth 5 we would need 85K parameters
  - With a 256×256 image we would need $46 \cdot 10^6$ parameters
- *Problem 1: Fitting a model with that many parameters is not easy*
- *Problem 2: Finding the data for such a model is not easy*
- *Problem 3: Are all these weights necessary?*

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
$\times$ $7 * 7 * 3$ parameters per filter

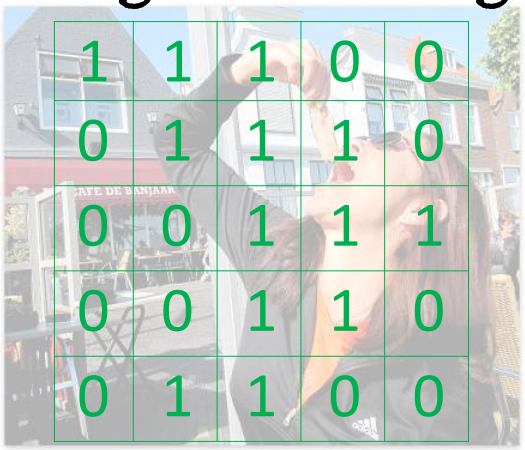
735 parameters in total

Shared 2-D Filters → Convolutions

Original image

Shared 2-D Filters → Convolutions

Original image



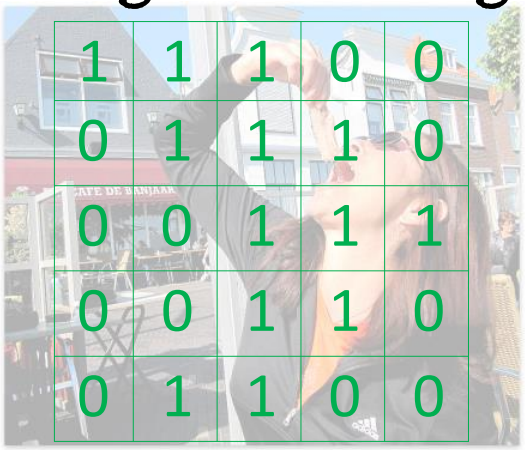
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

1	0	1
0	1	0
1	0	1

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

Result

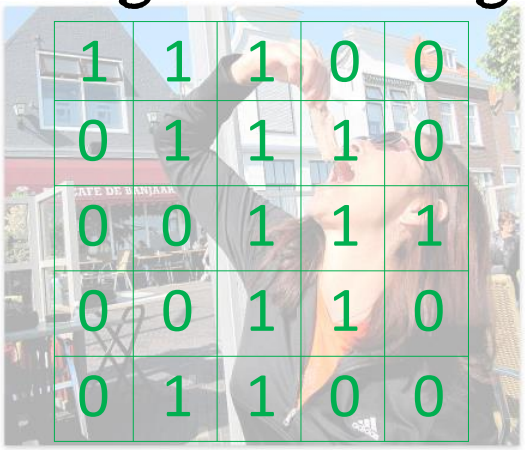
4		

Inner product

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

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

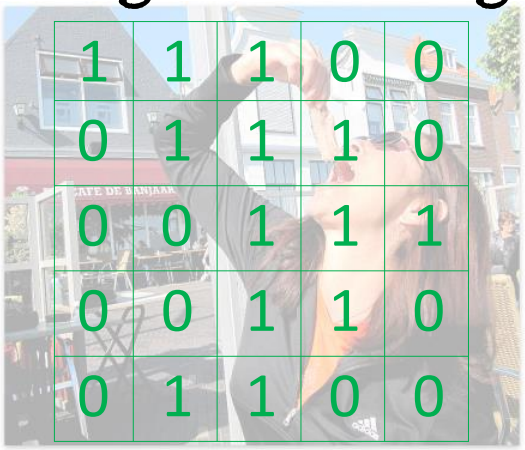
4	3	

Inner product

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

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

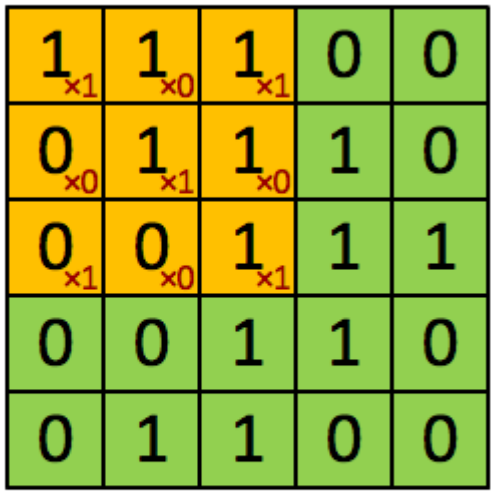
Result

4	3	4
2	4	3
2	3	4

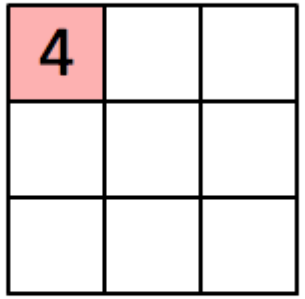
Inner product

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

Demo



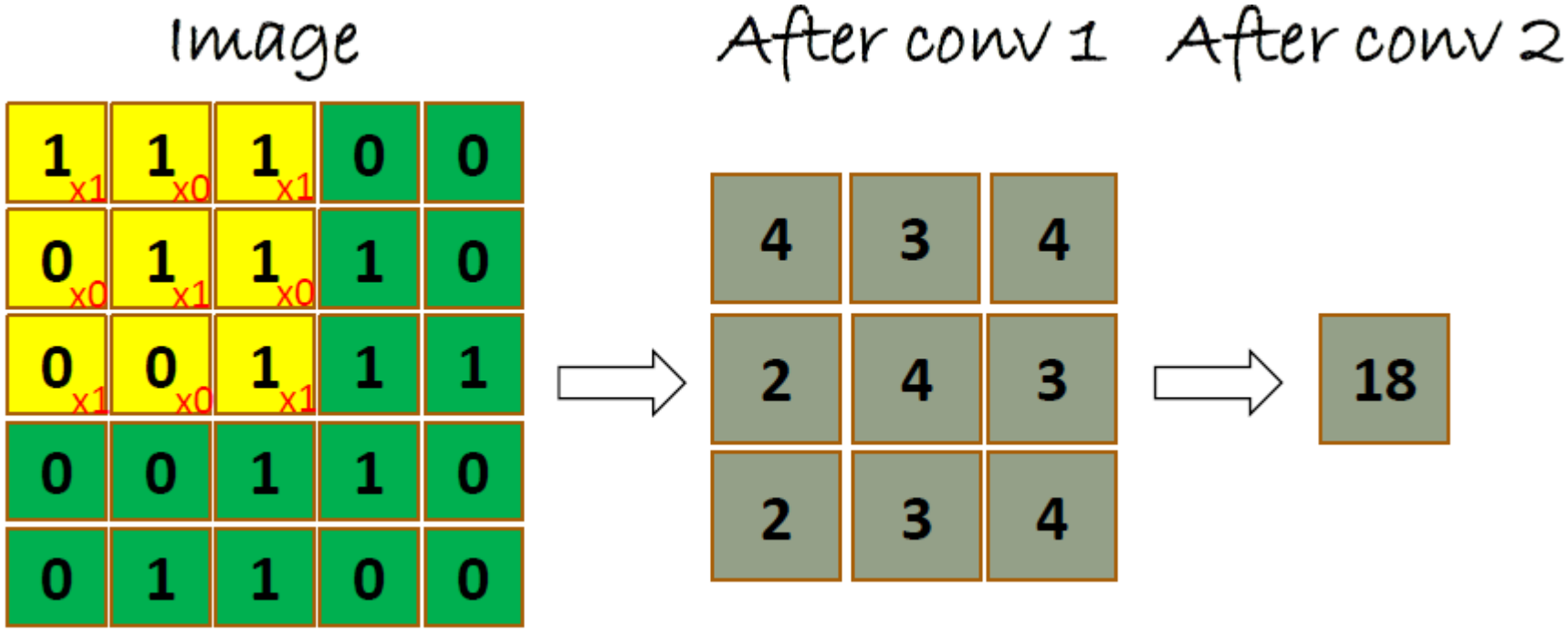
Image



Convolved Feature

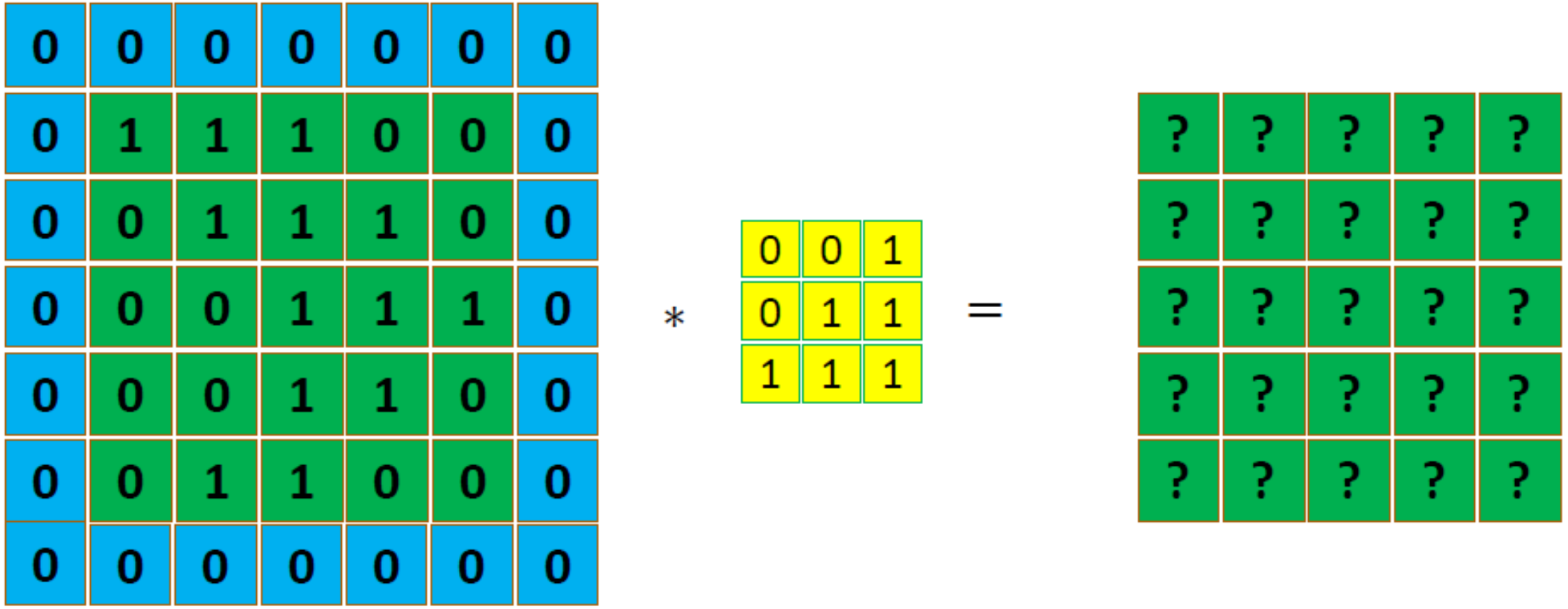
Problem, Again : S

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost

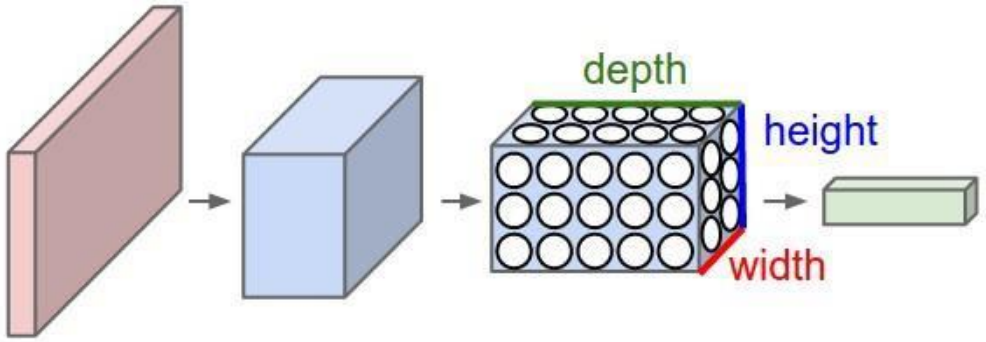


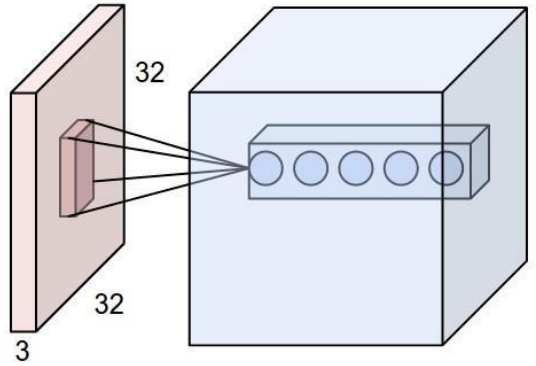
Solution? Zero-padding!

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



Understanding Convolutional NN





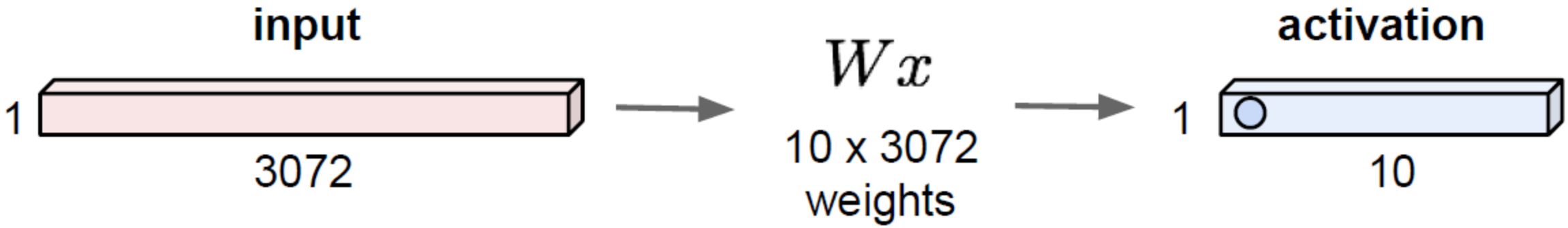
- Unlike a regular Neural Network, the layers of a CNN have neurons arranged in 3 dimensions: **width**, **height**, **depth**.
- (Note that the word *depth* here refers to the third dimension with regard of the activation volume

CNN vs. Regular Neural Networks

- Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.
- The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.
- And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer.
- **So what are the changes?** CNN architectures make the explicit assumption that the inputs are images, which allow us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

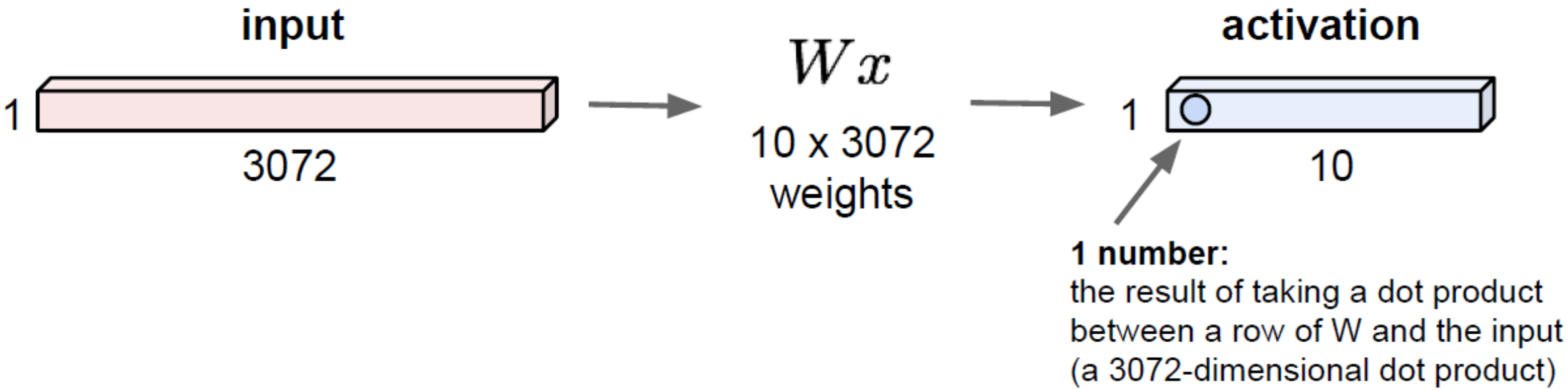
Fully Connected Layer (1)

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer (2)

32x32x3 image -> stretch to 3072 x 1



How to Initialize the Weights?

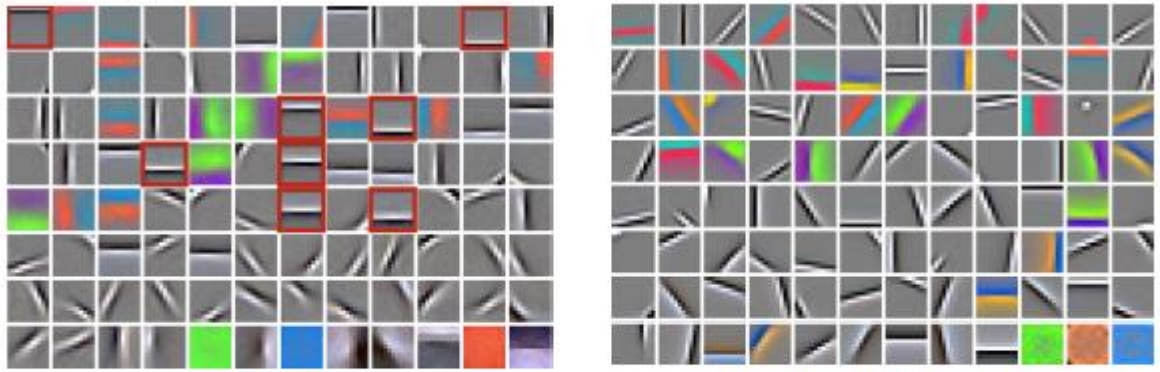
- Option 1: randomly
  - This has been shown to work nicely in the literature
- Option 2:
  - Train/obtain the “filters” elsewhere and use them as the weights
  - Unsupervised pre-training using image patches (windows)
  - Avoids full feedforward and backward pass, allows the search to start from a better position
  - You may even skip training the convolutional layers

How to Initialize the Weights? – Option 2

Under review as a conference paper at ICLR 2016

CONVOLUTIONAL CLUSTERING FOR UNSUPERVISED LEARNING

Aysegul Dundar, Jonghoon Jin, and Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{adundar,jhjin,euge}@purdue.edu



(a) k-means (b) convolutional k-means

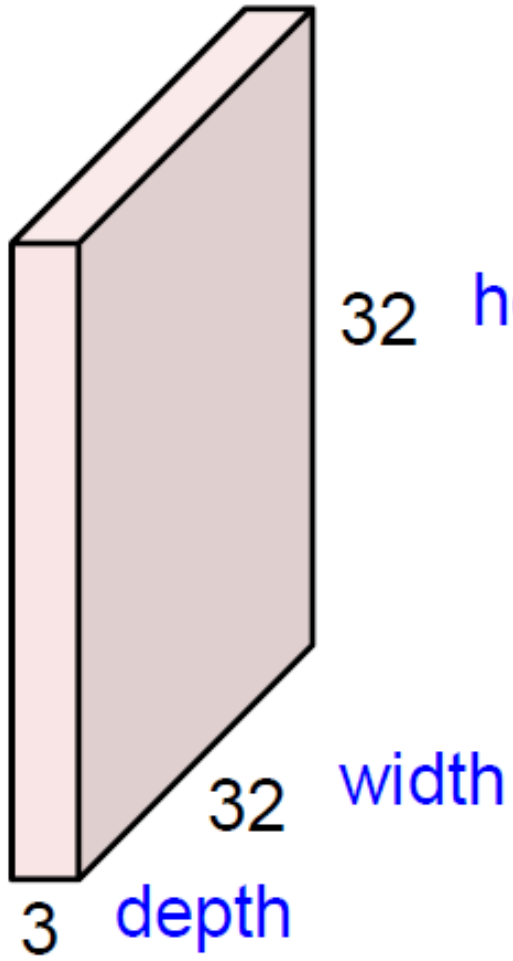
Figure 1: Filters trained on the STL-10 dataset with k-means and convolutional k-means. Filters are sorted by variance in descending order. While convolutional k-means learns unique features, the k-means algorithm introduces redundancy in filters. The duplicated features for horizontal edges are highlighted in red.

ABSTRACT

The task of labeling data for training deep neural networks is daunting and tedious, requiring millions of labels to achieve the current state-of-the-art results. Such reliance on large amounts of labeled data can be relaxed by exploiting hierarchical features via unsupervised learning techniques. In this work, we propose to train a deep convolutional network based on an enhanced version of the k-means clustering algorithm, which reduces the number of correlated parameters in the form of similar filters, and thus increases test categorization accuracy. We call our algorithm *convolutional k-means clustering*. We further show that learning the connection between the layers of a deep convolutional neural network improves its ability to be trained on a smaller amount of labeled data. Our experiments show that the proposed algorithm outperforms other techniques that learn filters unsupervised. Specifically, we obtained a test accuracy of 74.1% on STL-10 and a test error of 1.4% on MNIST.

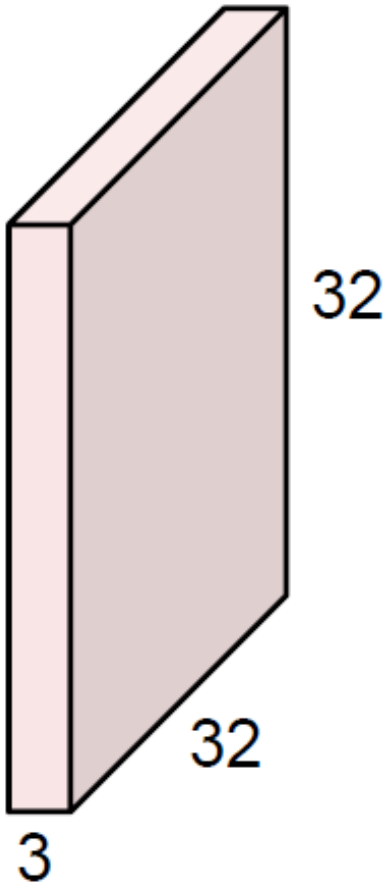
Convolution Layer (1)

32x32x3 image -> preserve spatial structure

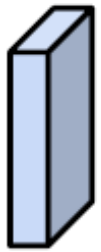


Convolution Layer (2)

32x32x3 image



5x5x3 filter

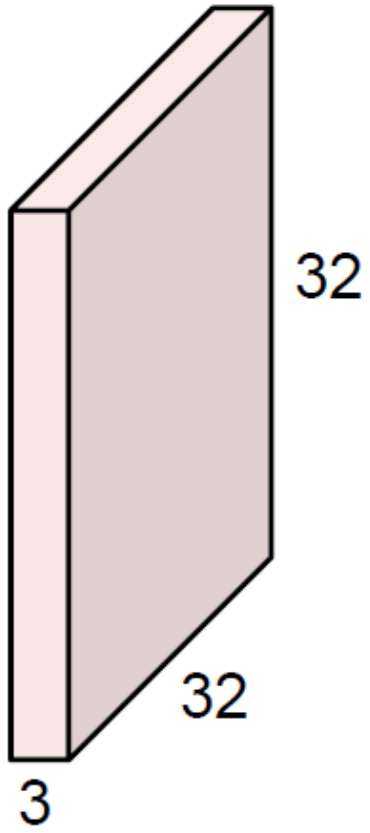


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

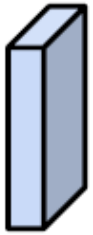
Convolution Layer (3)

Filters always extend the full depth of the input volume

32x32x3 image

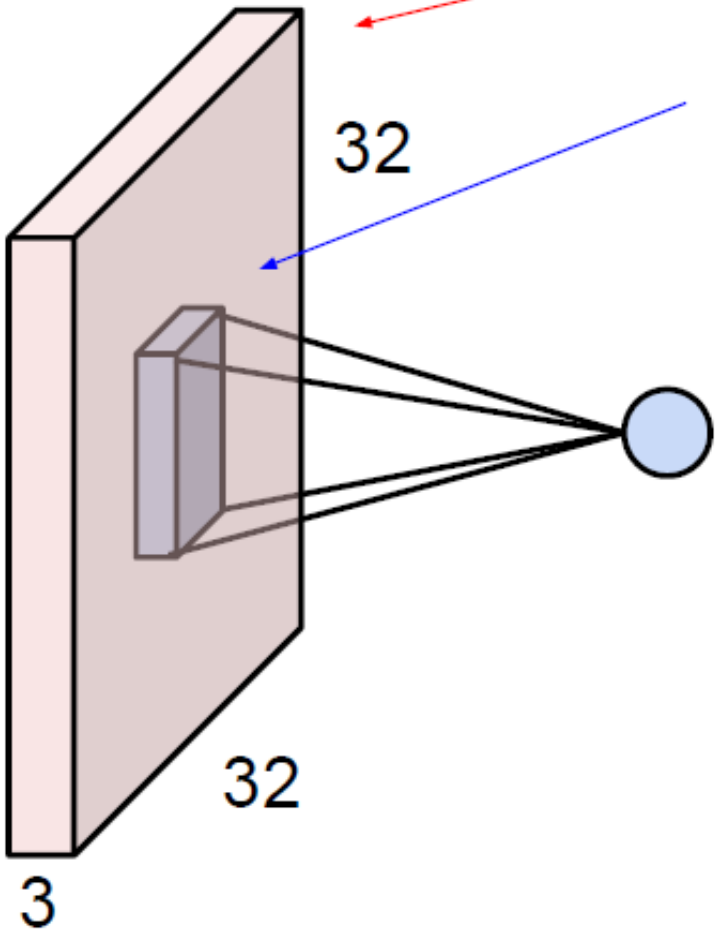


5x5x3 filter



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

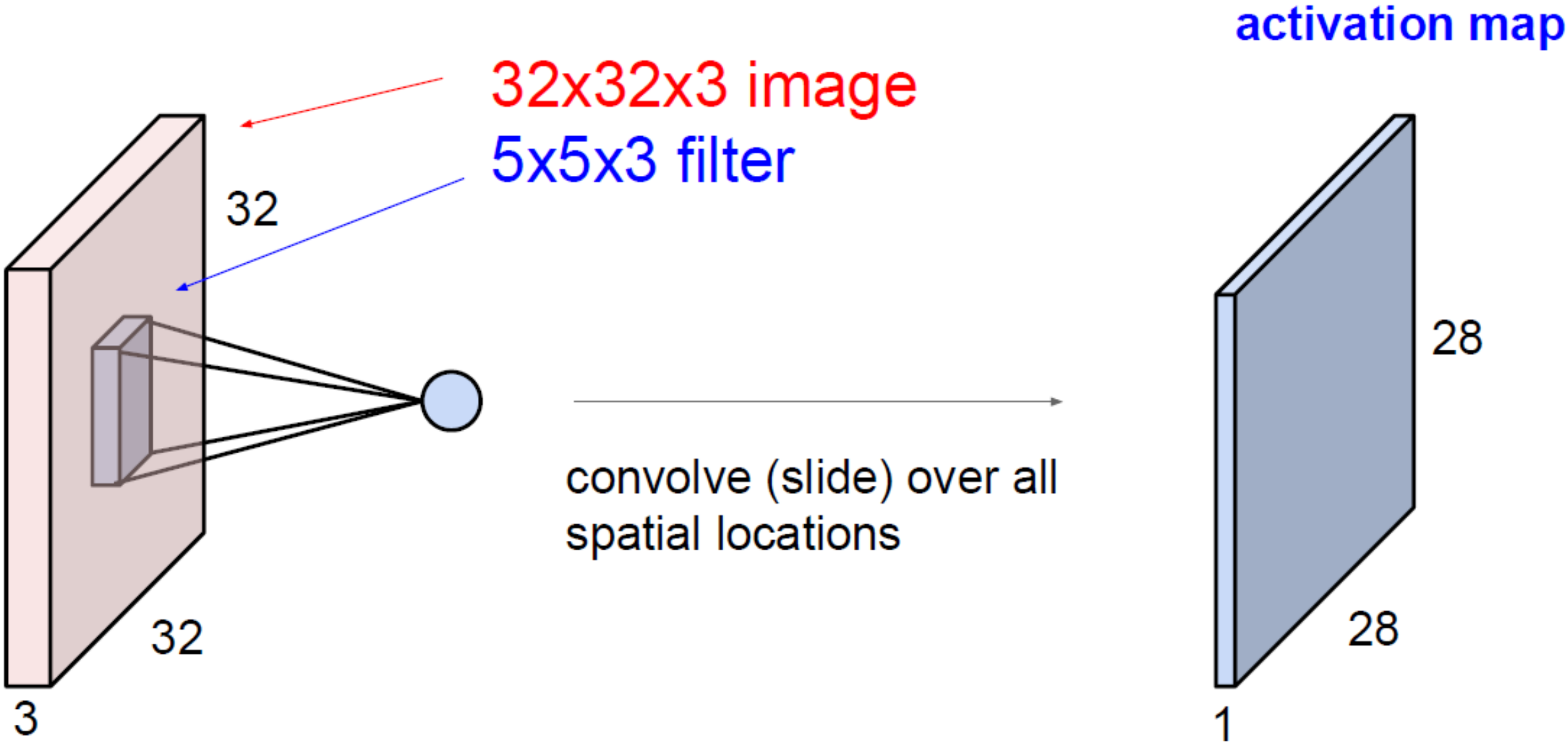
Convolution Layer (4)



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)

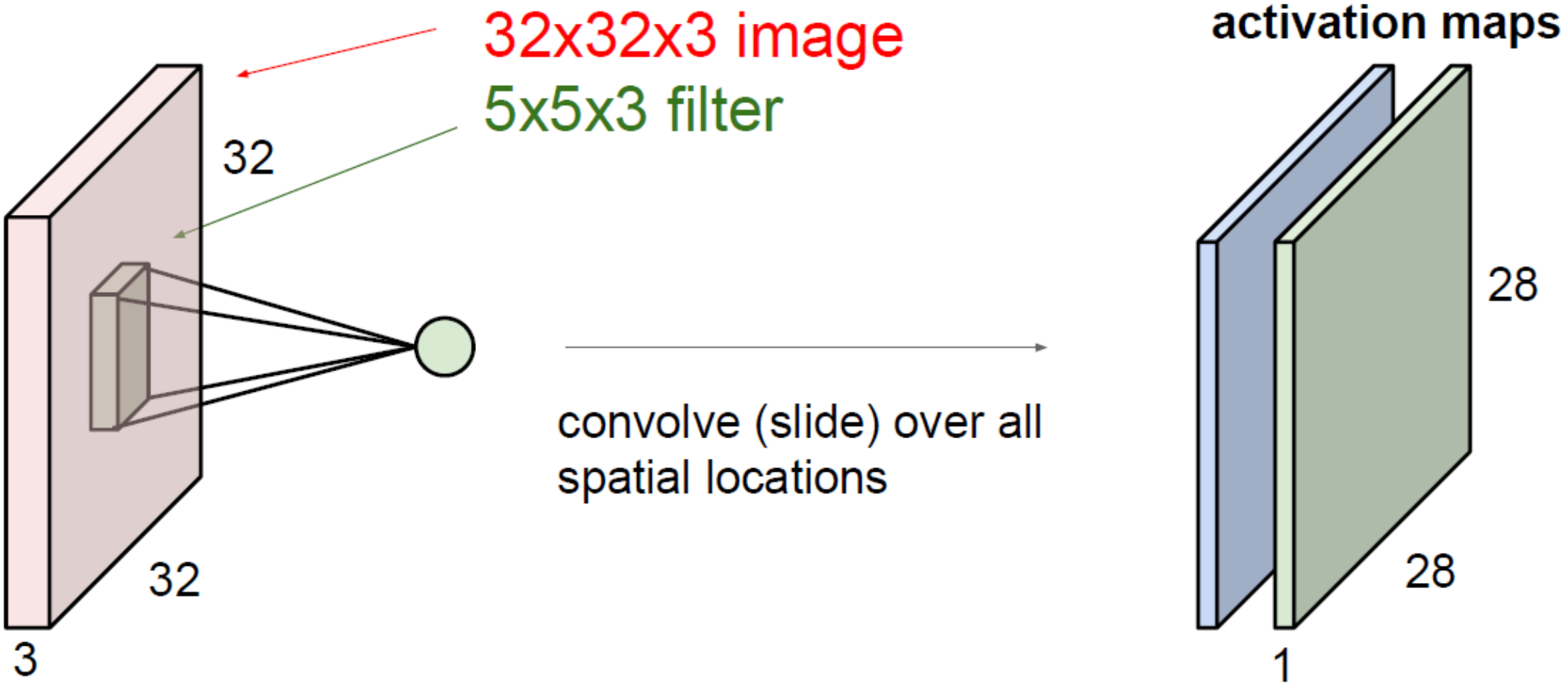
$$w^T x + b$$

Convolution Layer (5)

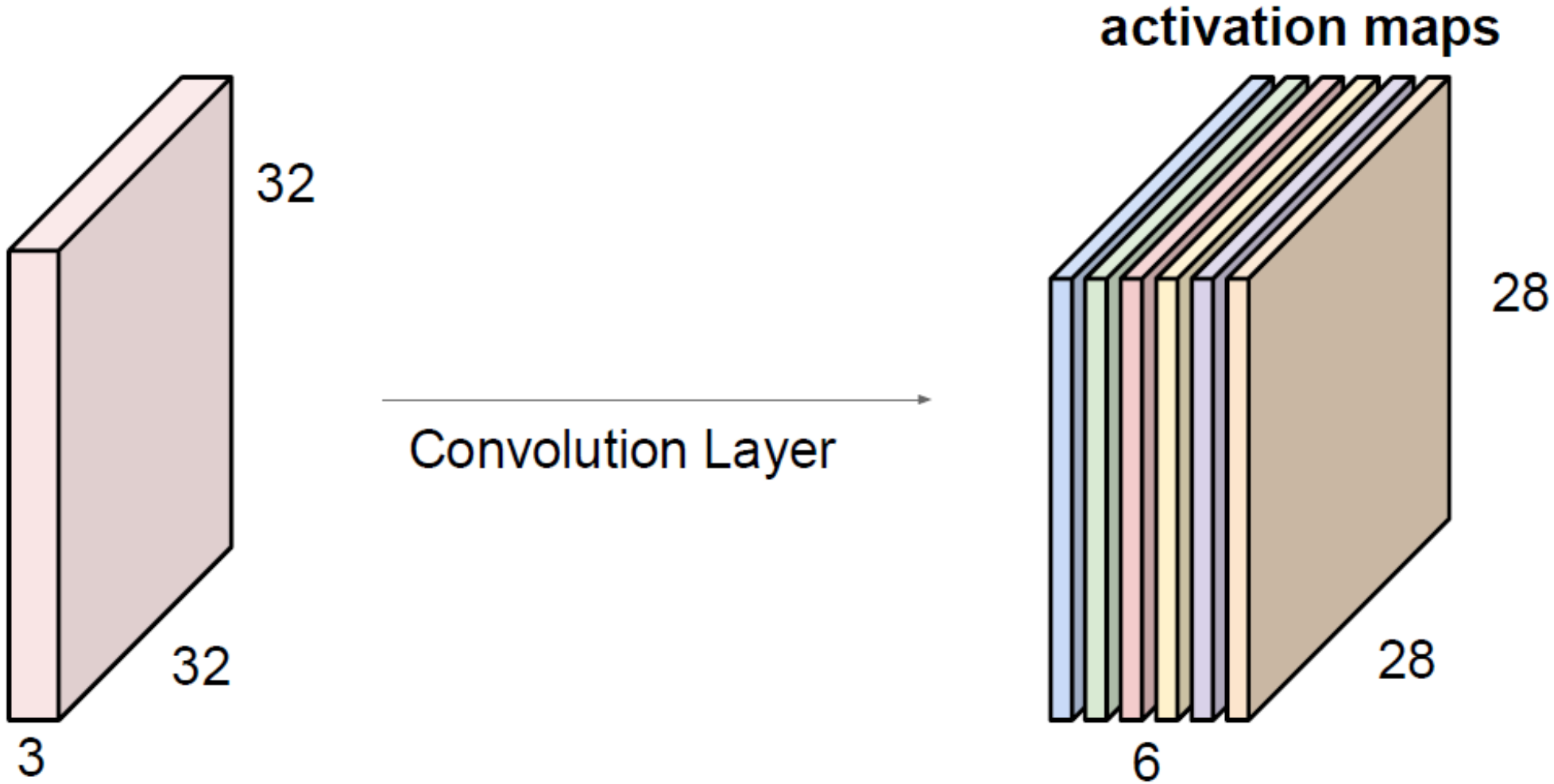


Convolution Layer (6)

consider a second, green filter

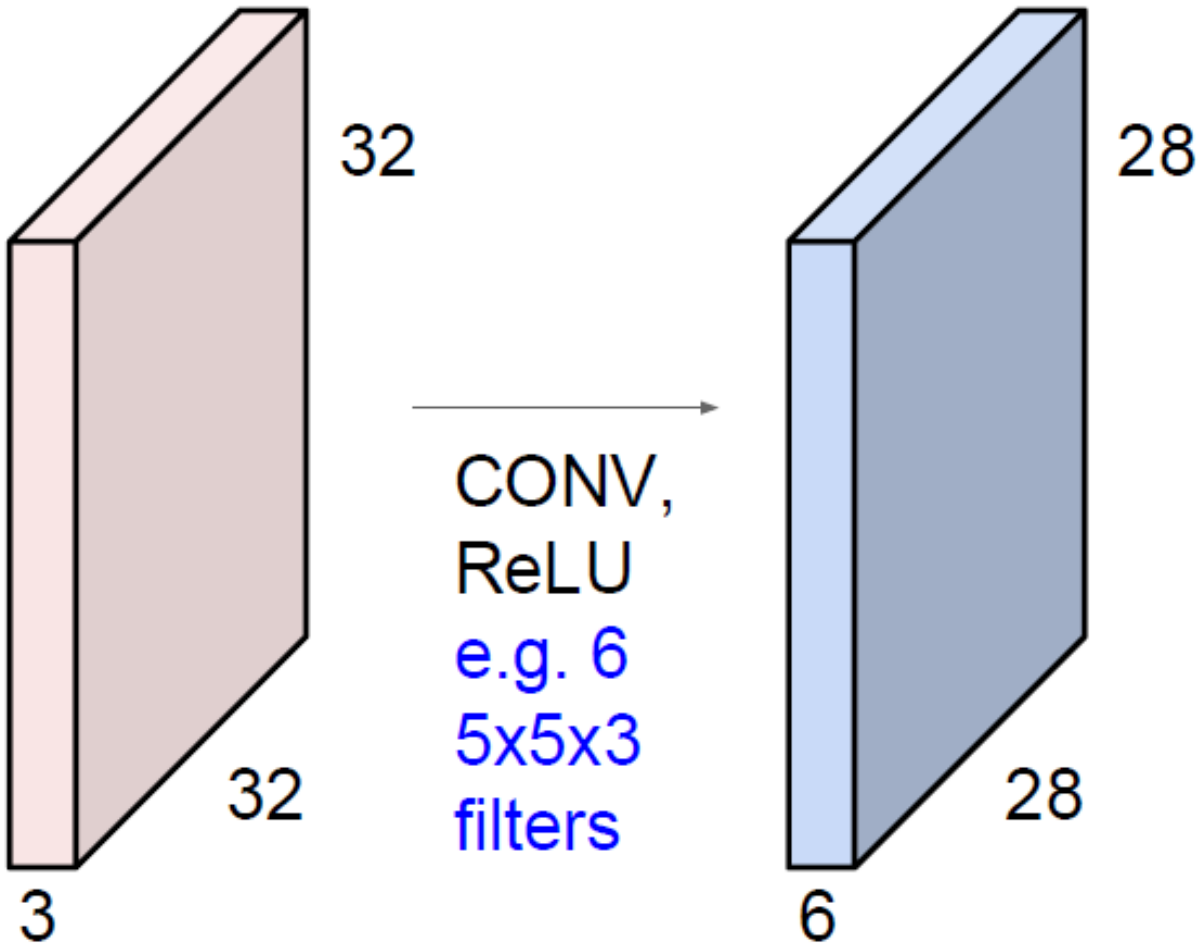


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

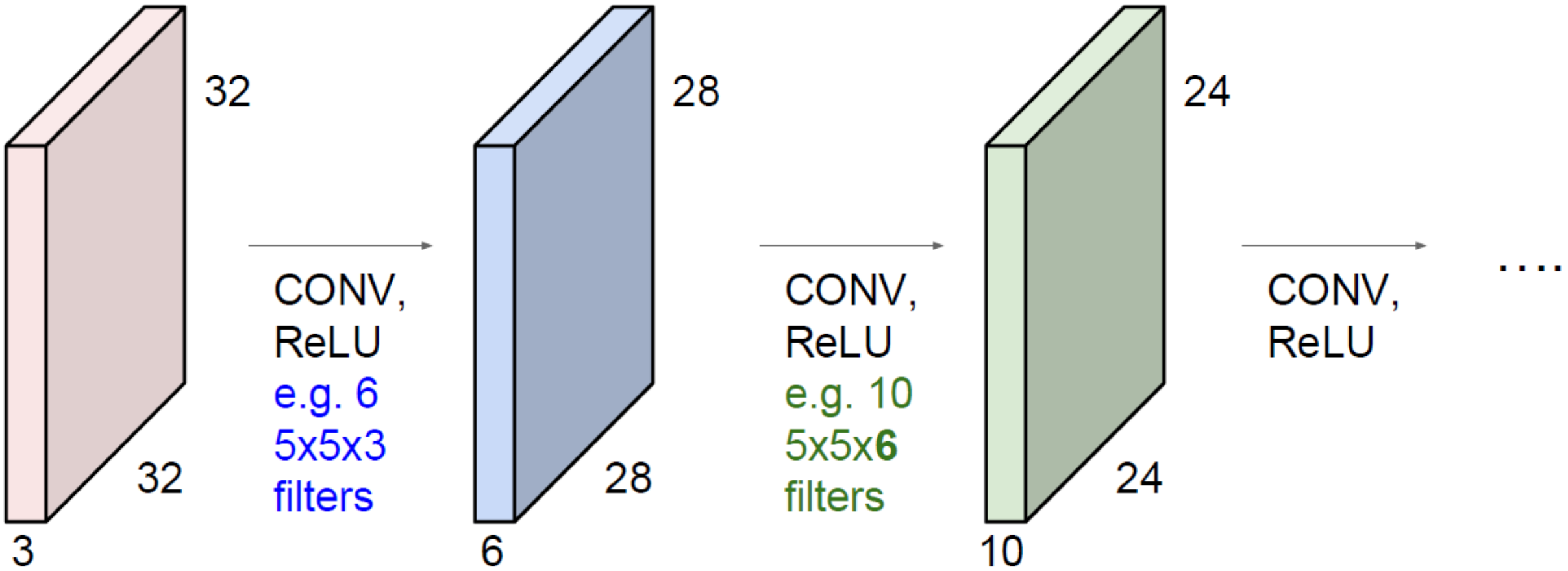


We stack these up to get a "new image" of size 28x28x6!

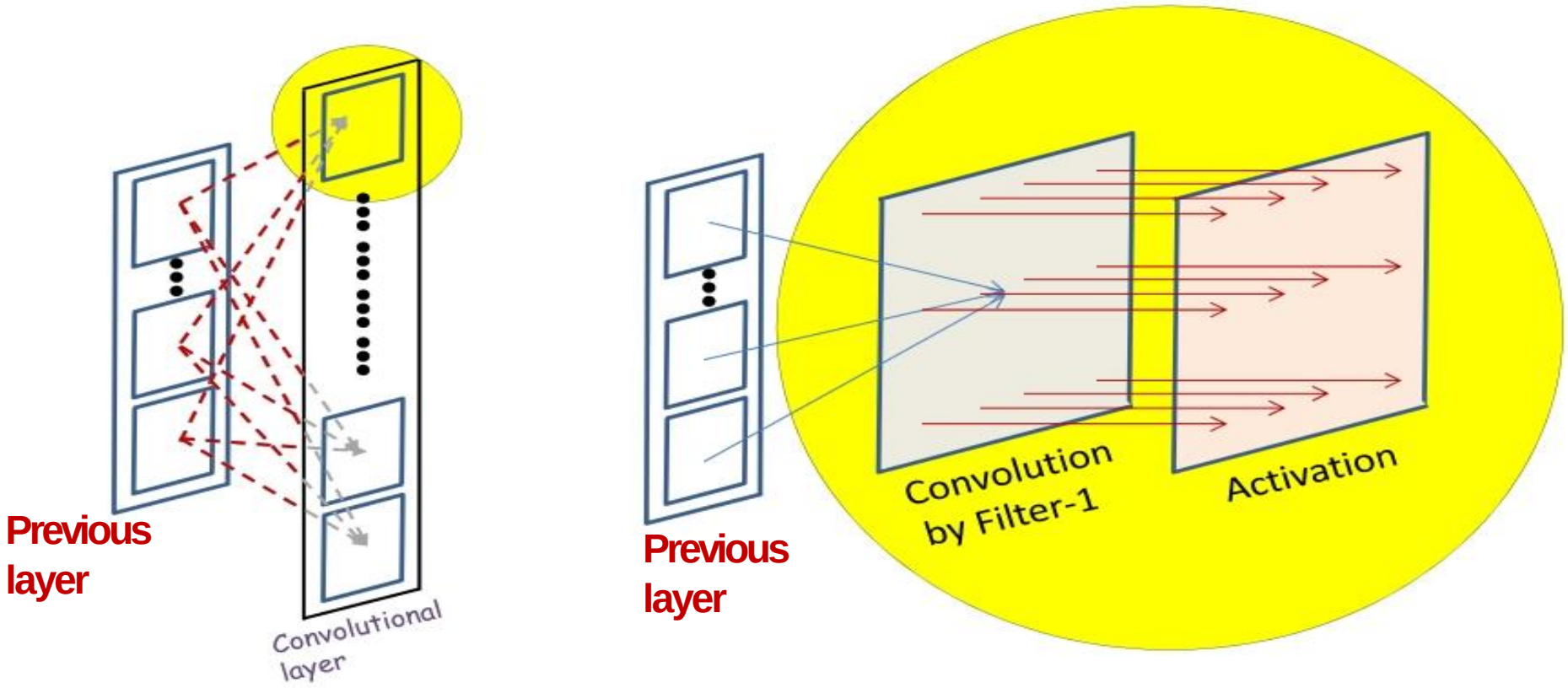
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

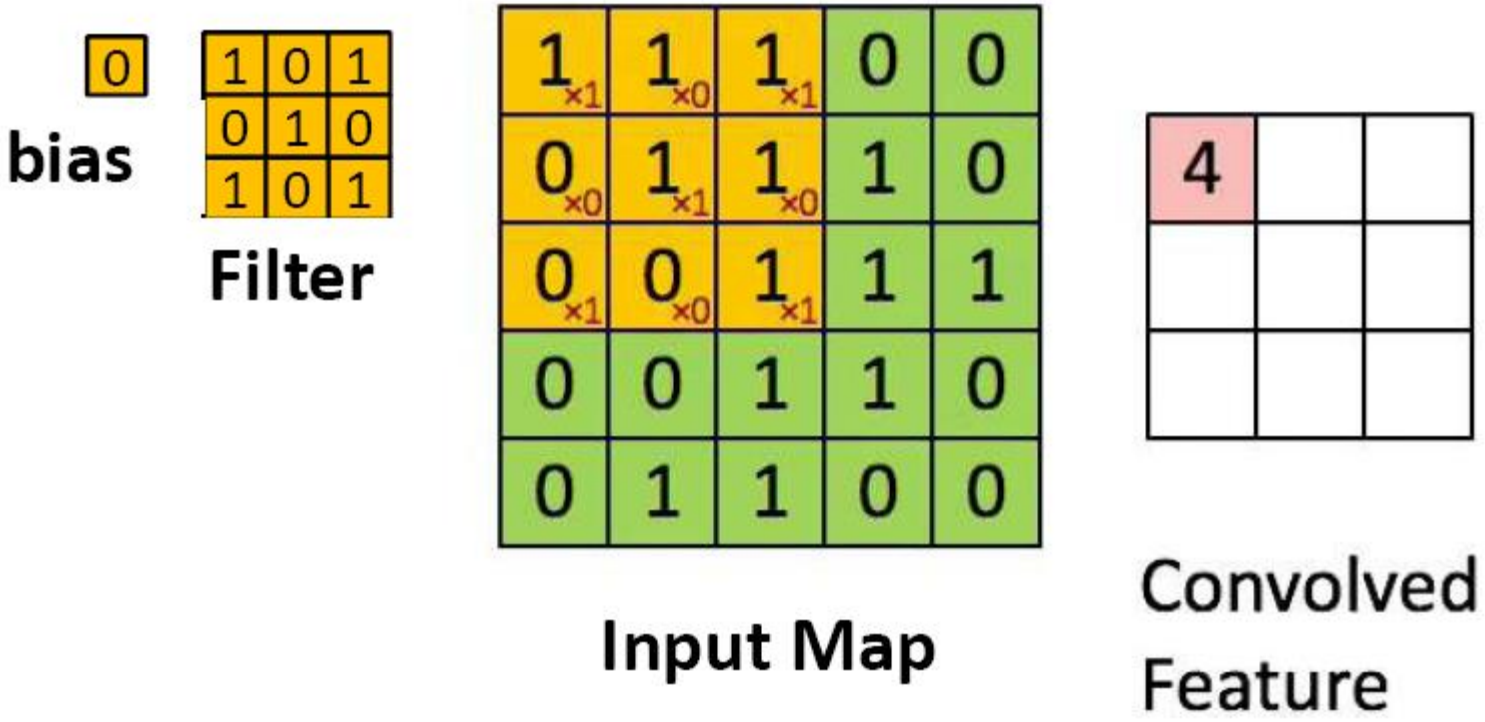


A convolutional layer



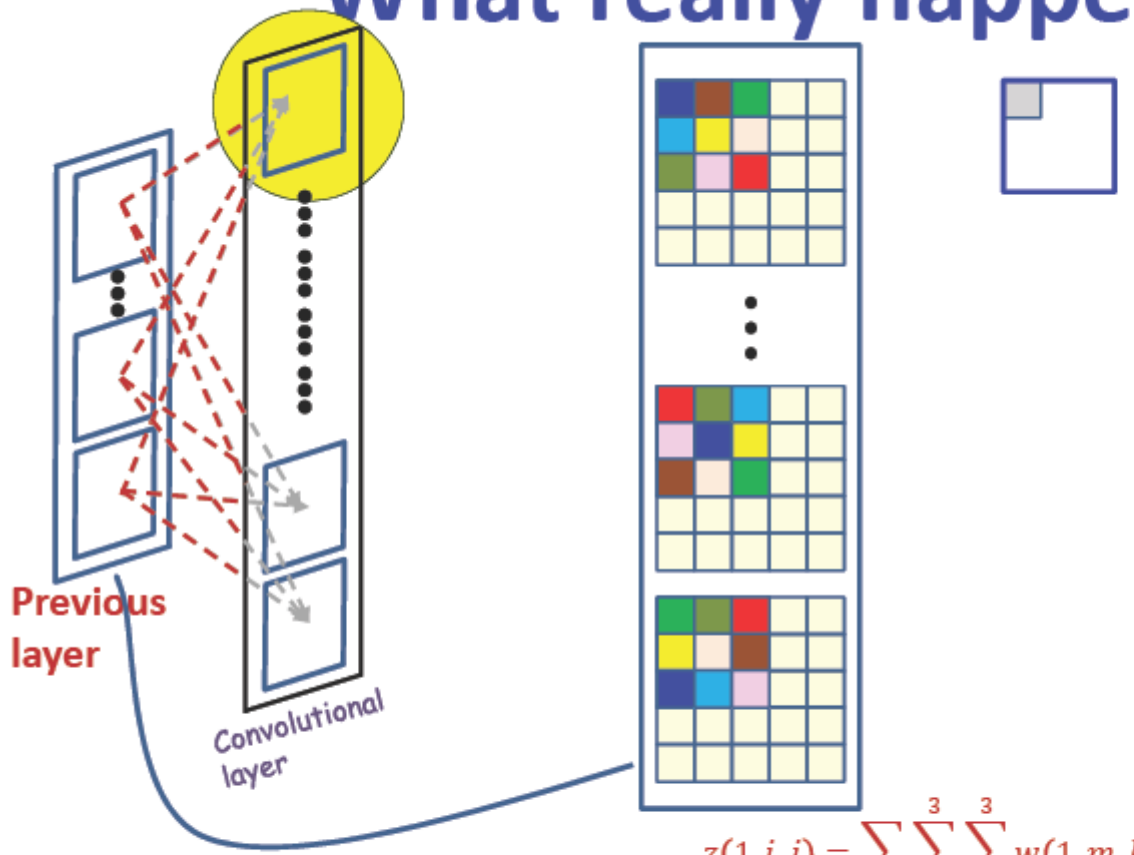
- Each activation map in the convolutional layer has two components – A *linear* map, obtained by ***convolution*** over maps in the previous layer
  - Each linear map has, associated with it, a ***learnable filter***
  - An ***activation*** that operates on the output of the convolution

What is a Convolution



- Scanning an image with a "filter"
  - At each location, the filter and the underlying map values are multiplied component wise, and the products are added along with the bias

What really happens

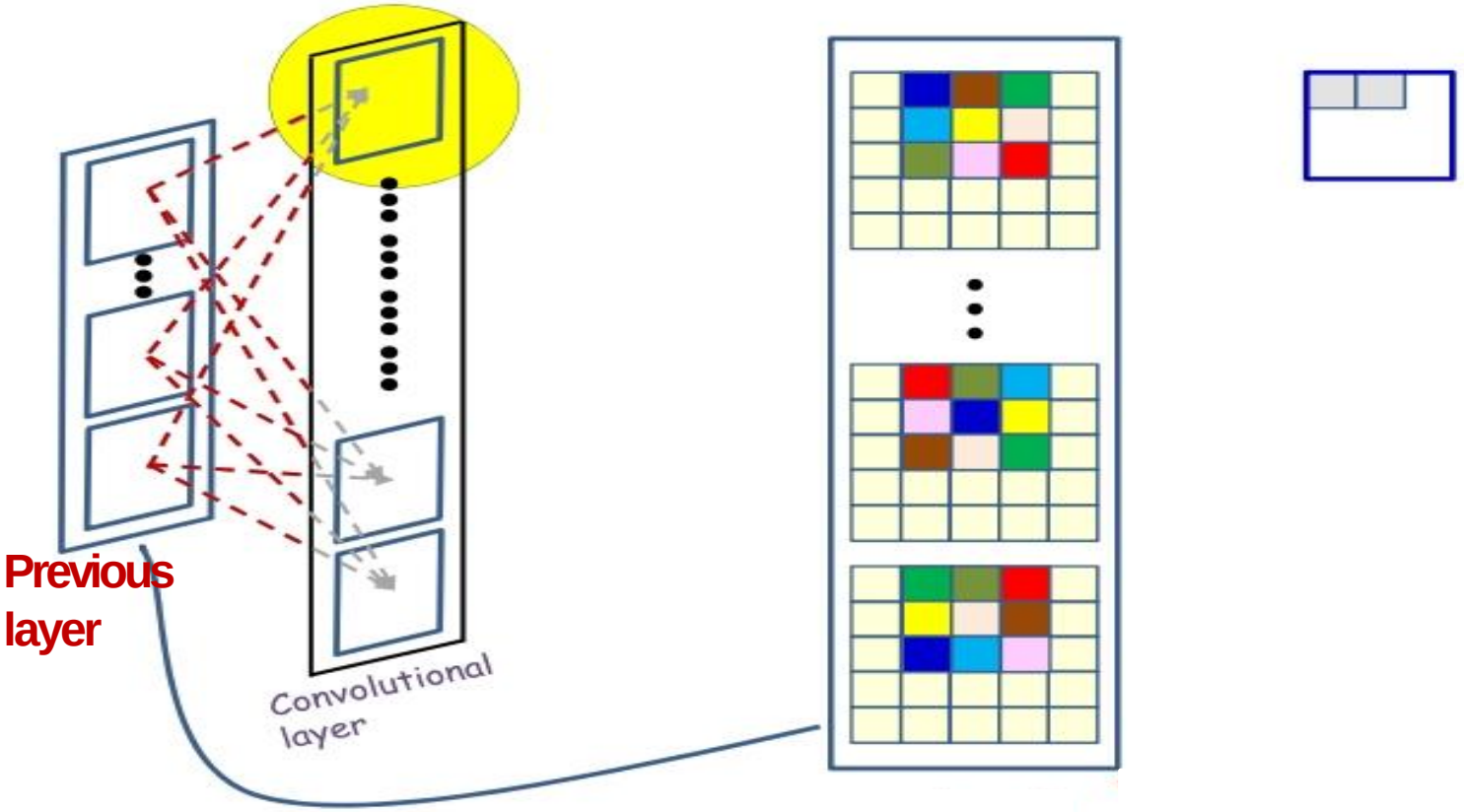


$$z(1,i,j) = \sum_{m} \sum_{k=1}^{3} \sum_{l=1}^{3} w(1,m,k,l) I(m, i+l-1, j+k-1) + b$$

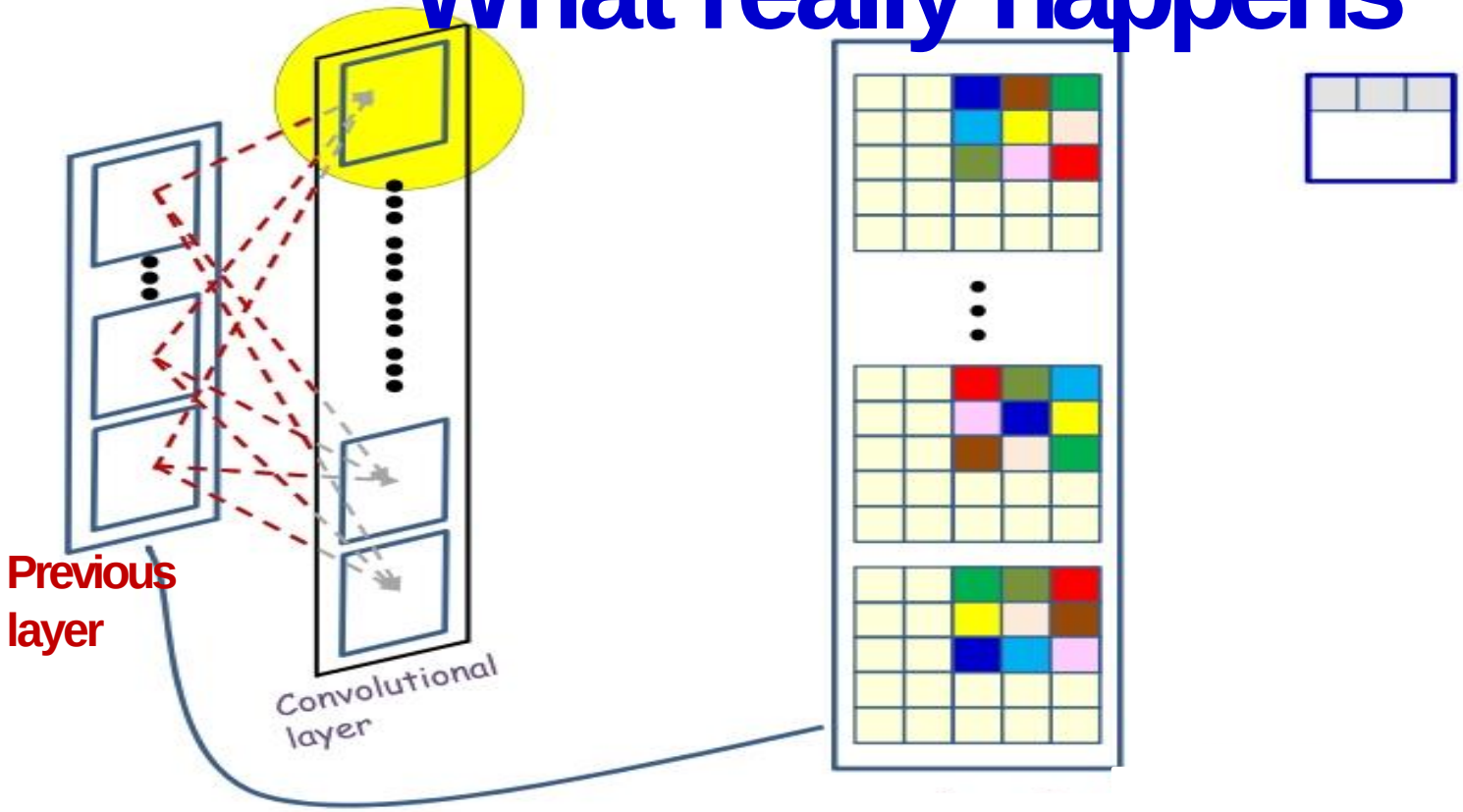
- Each output is computed from multiple maps simultaneously
- There are as many weights (for each output map) as

  size of the filter x *no. of maps in previous layer*

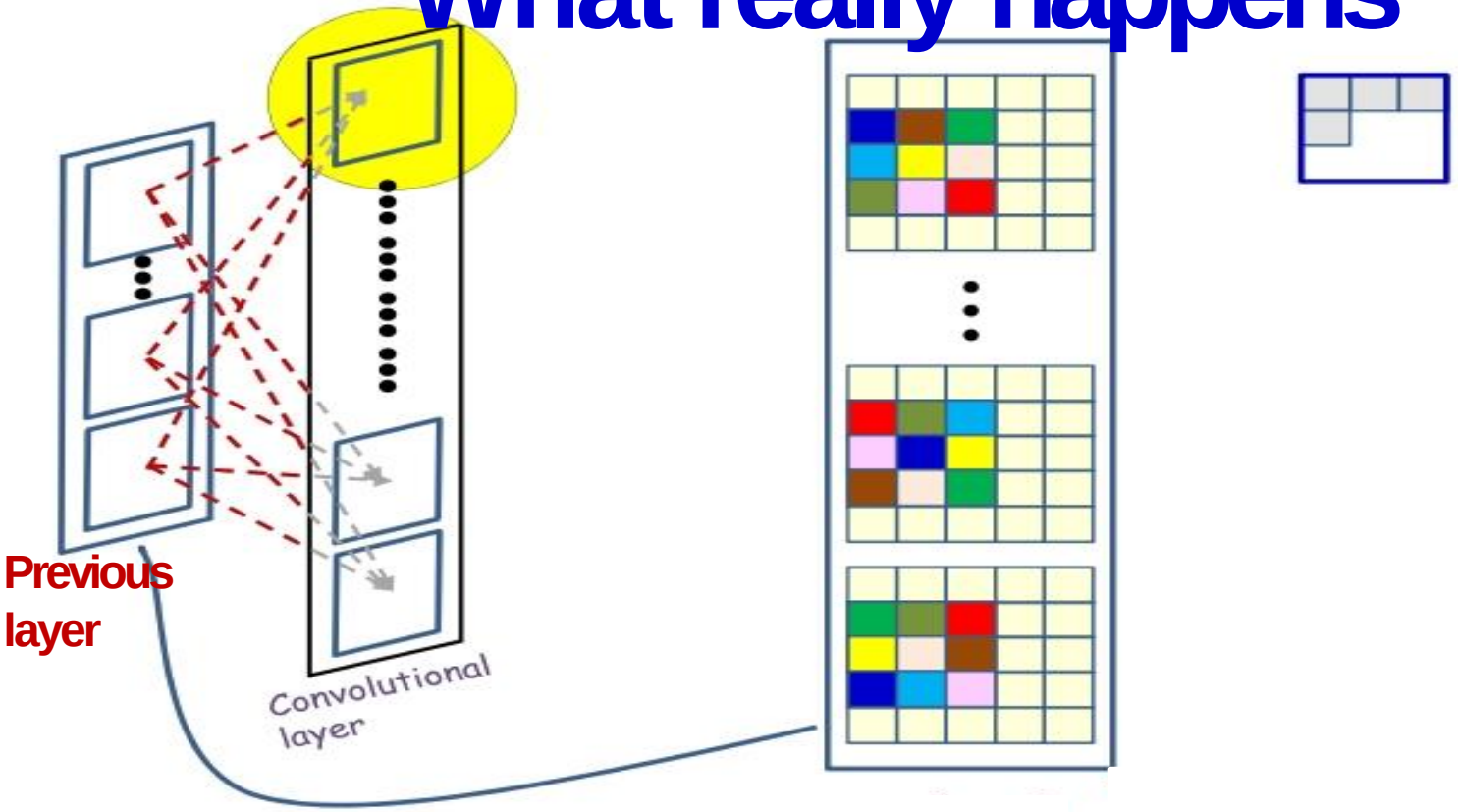
What really happens



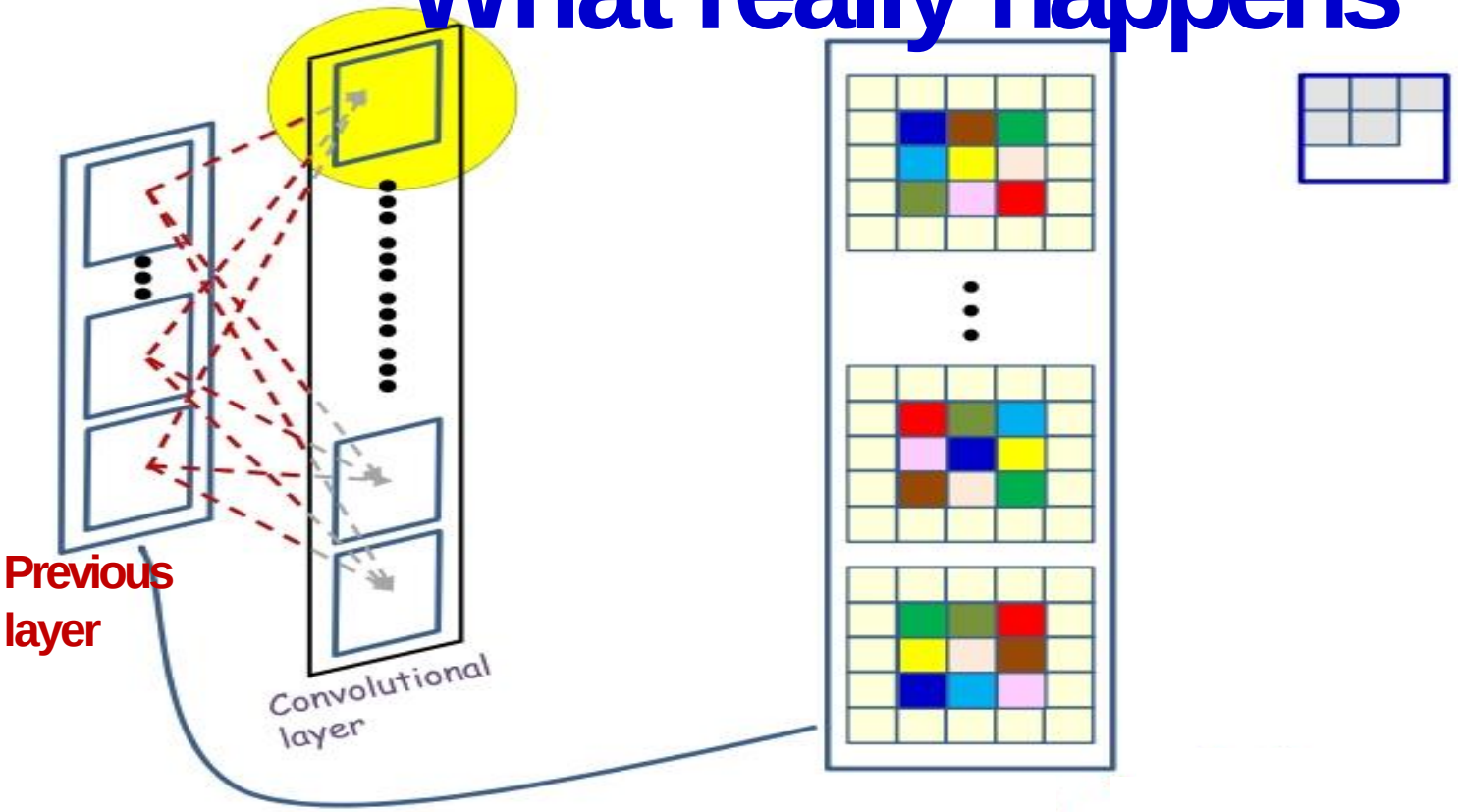
What really happens



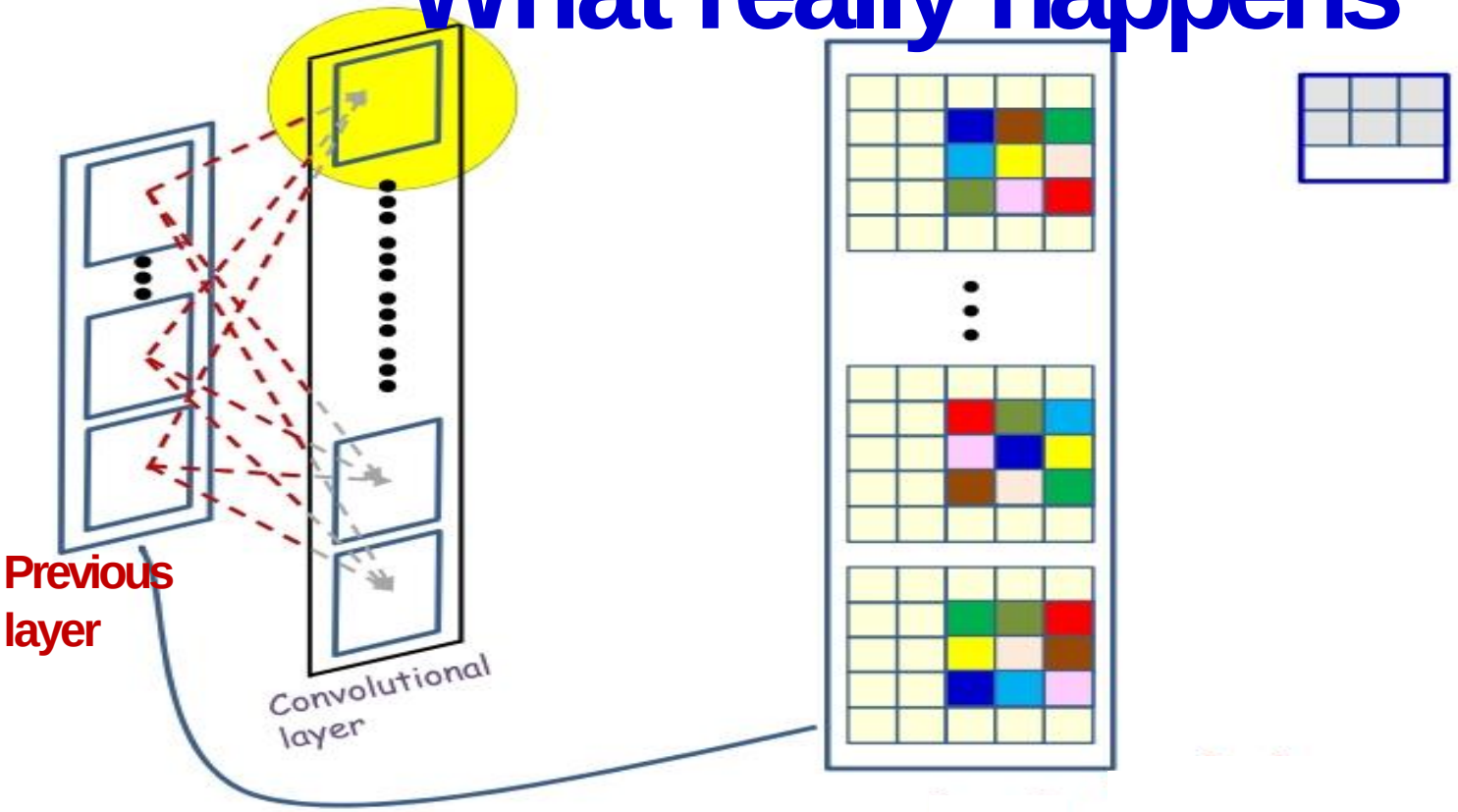
What really happens



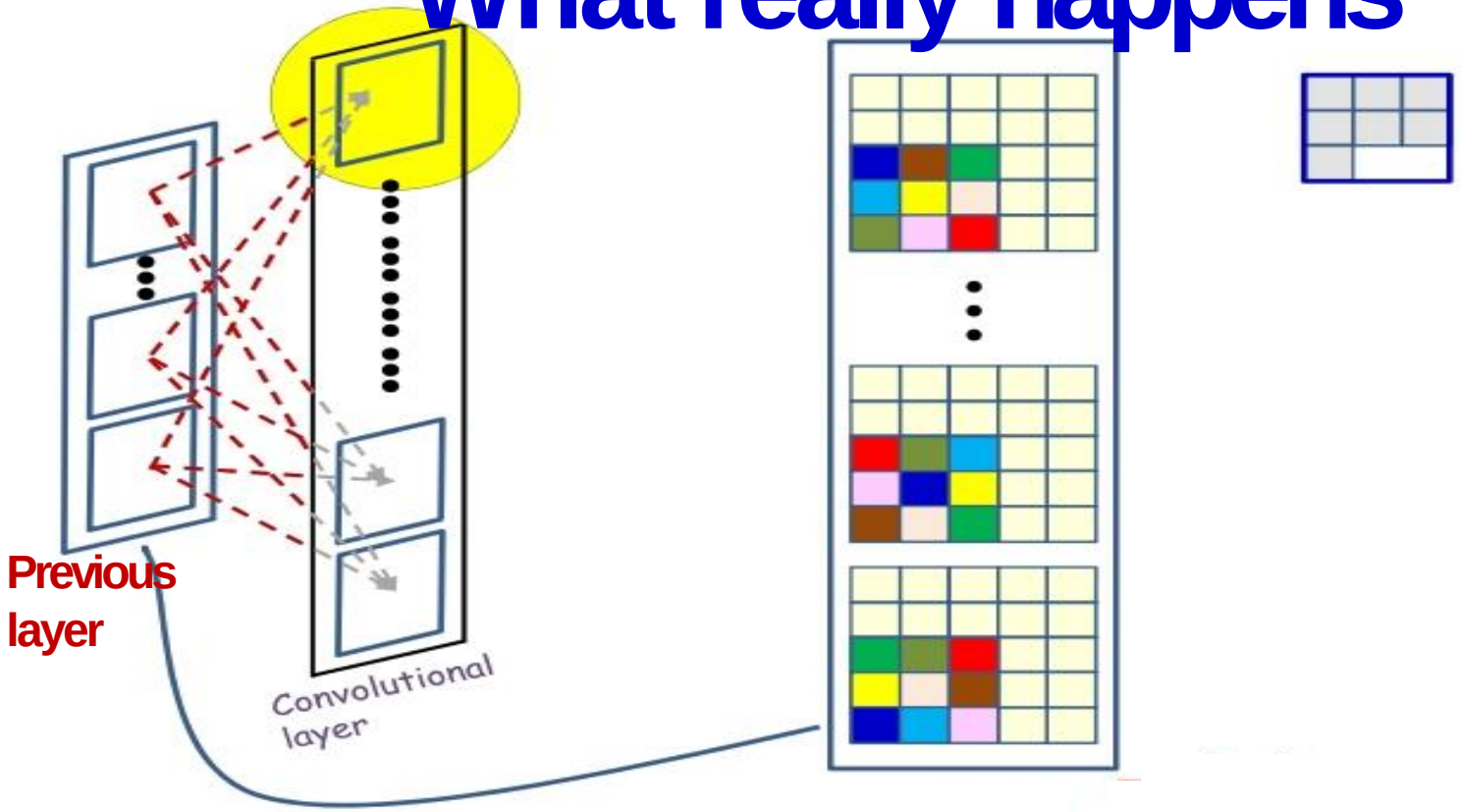
What really happens



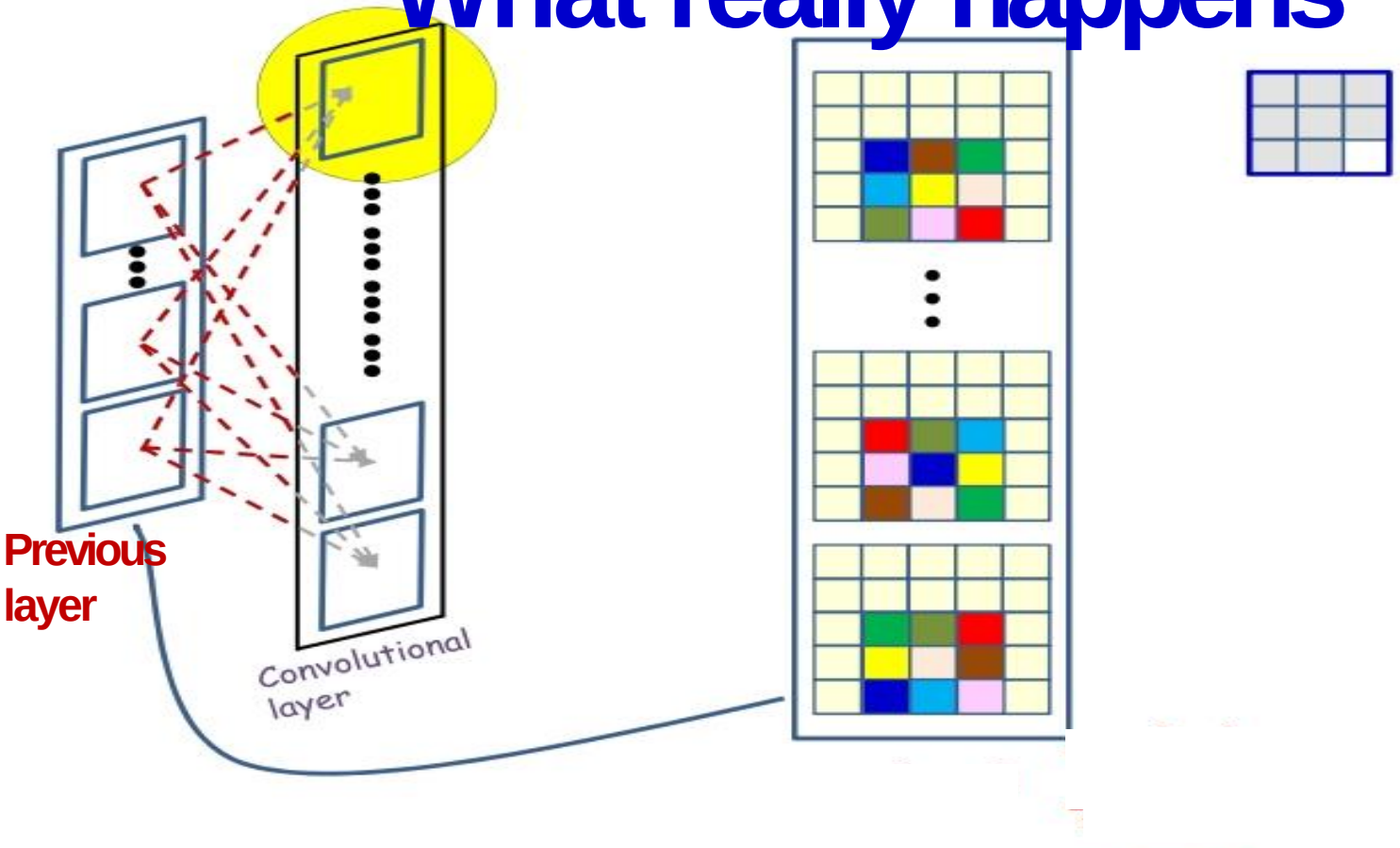
What really happens



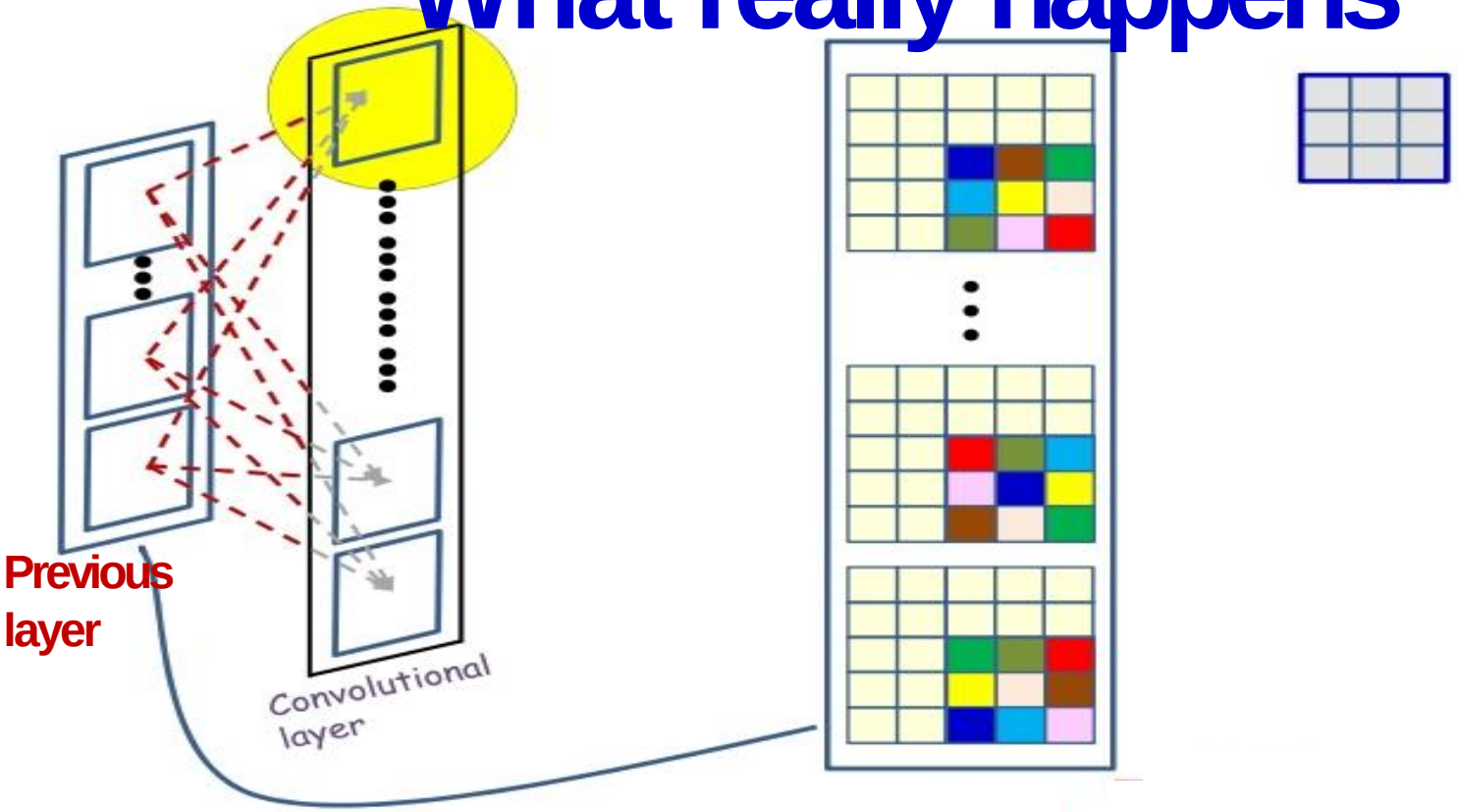
What really happens



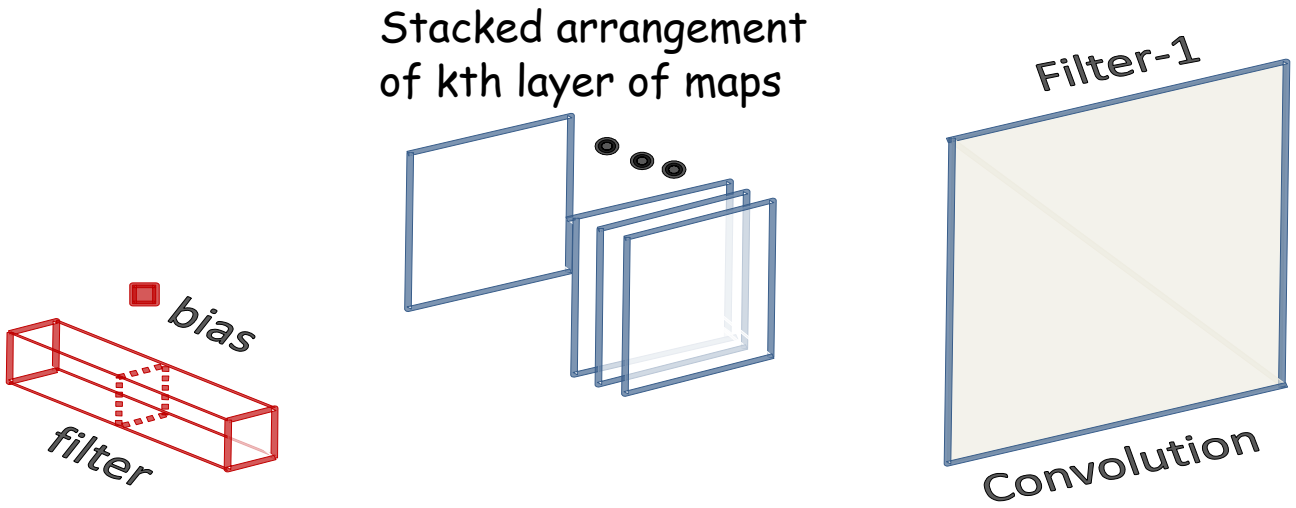
What really happens



What really happens



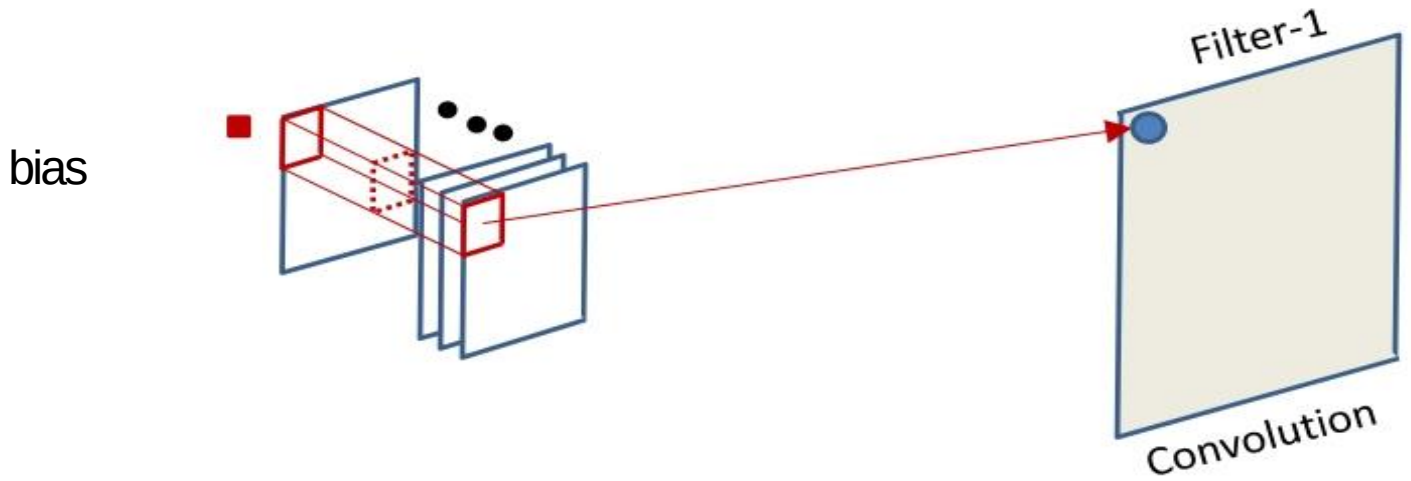
Abetter representation



Filter applied to kth layer of maps
(convolutive component plus bias)

- ..A *stacked* arrangement of planes
- We can view the joint processing of the various maps as processing the stack using a three- dimensional filter

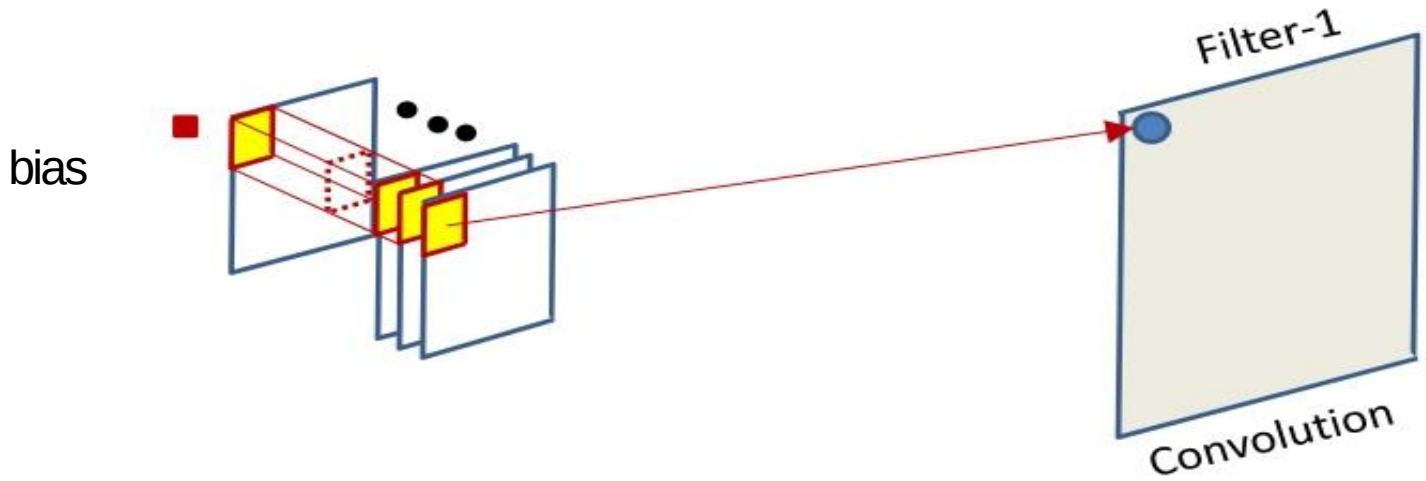
Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

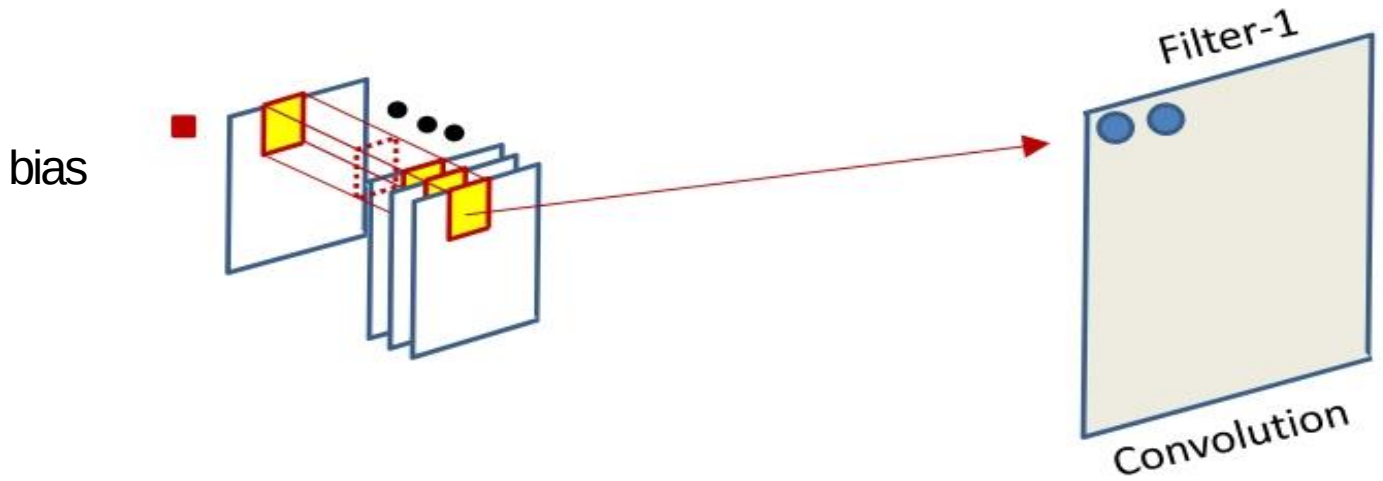
Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

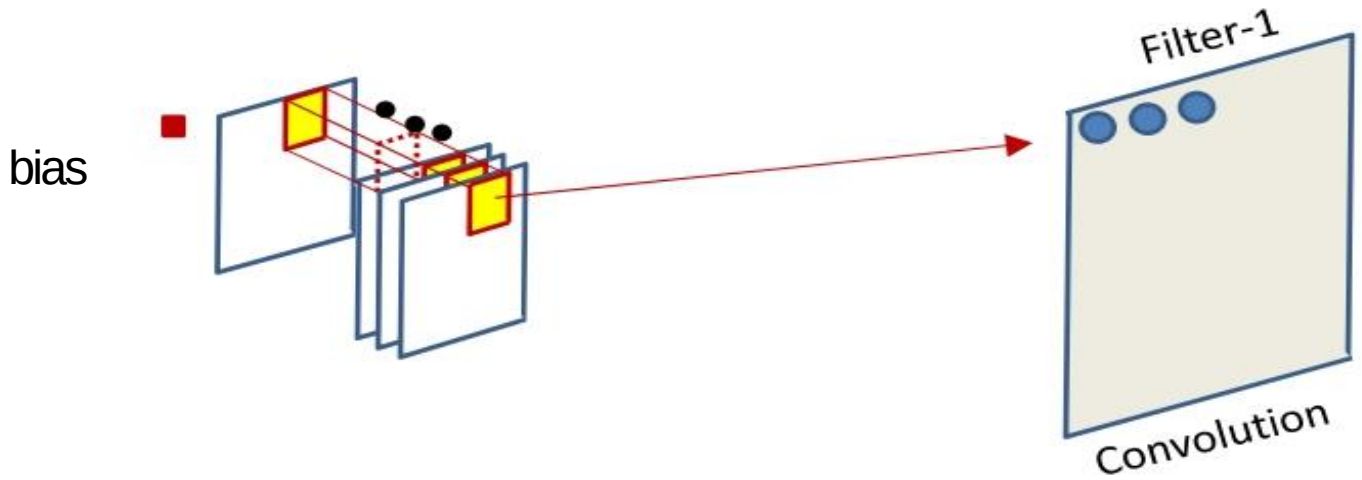
Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

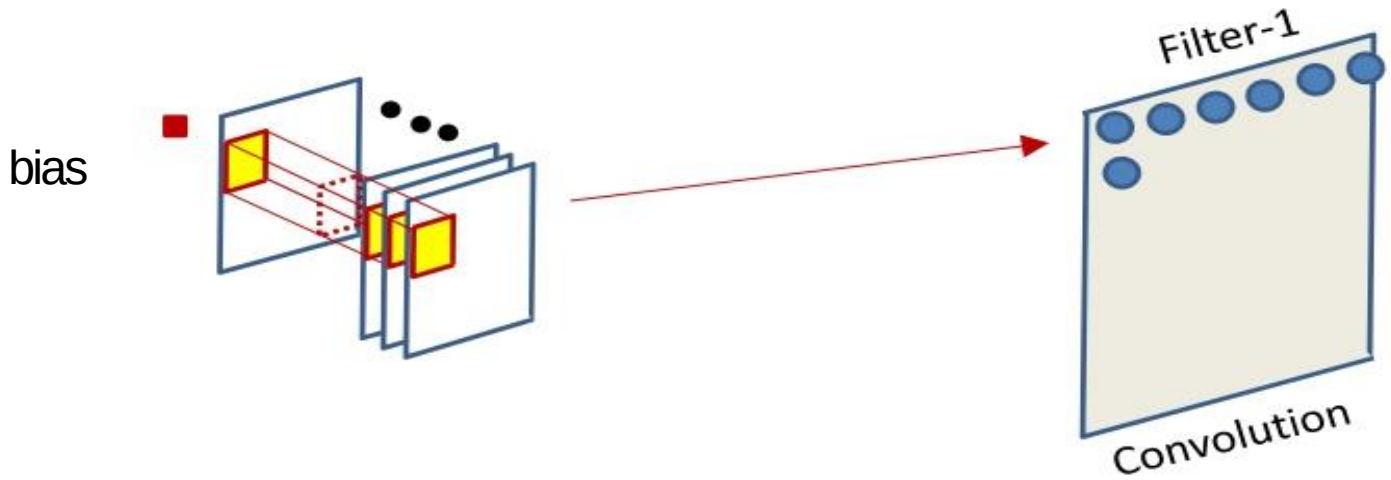
Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

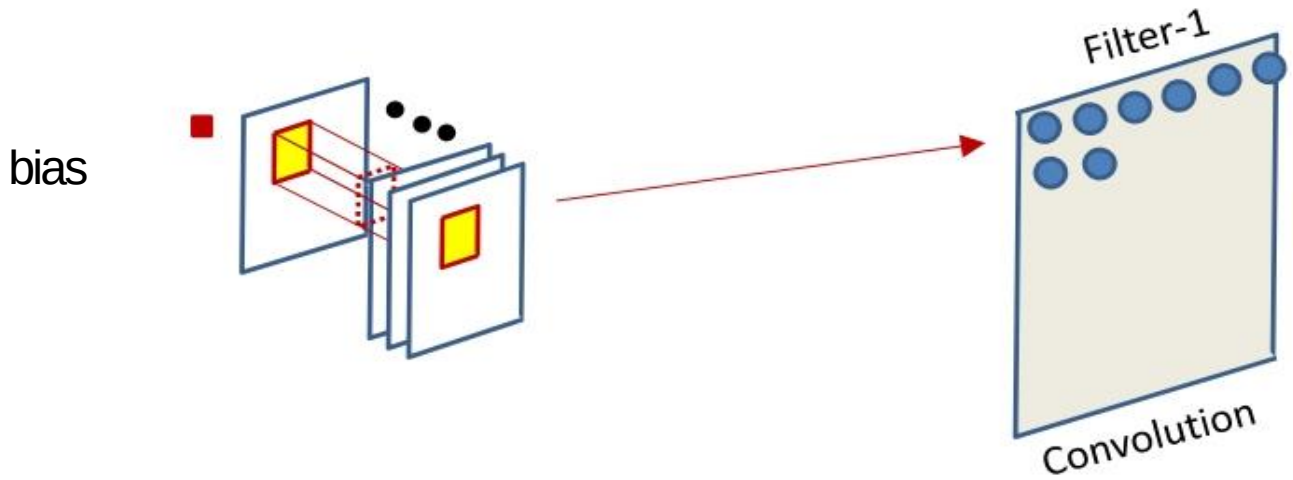
Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

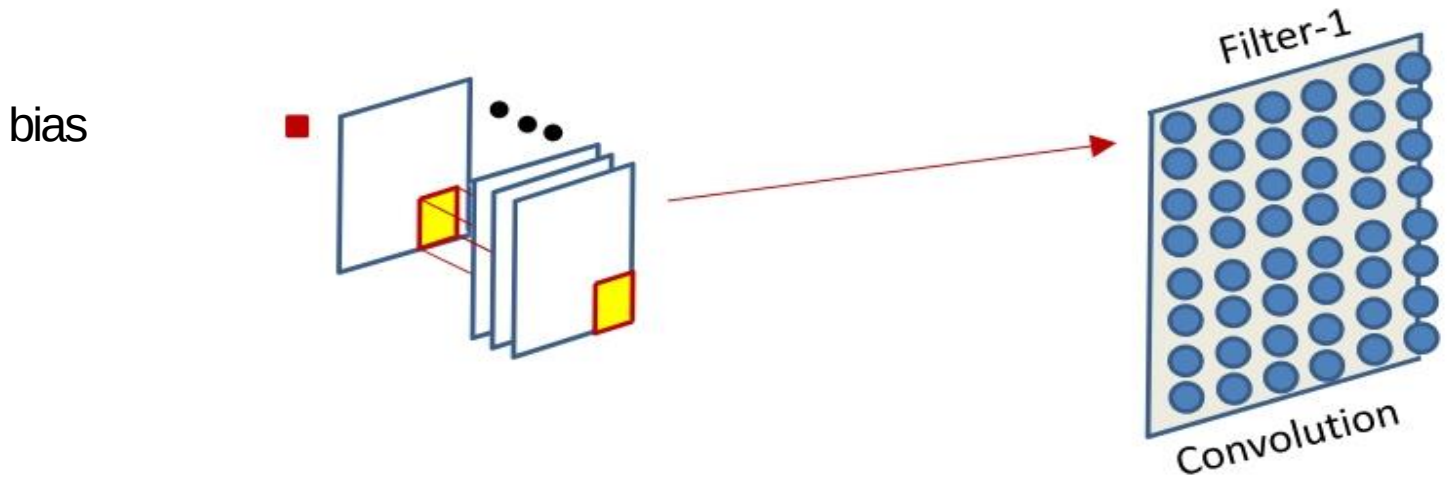
- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

Abetter representation



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

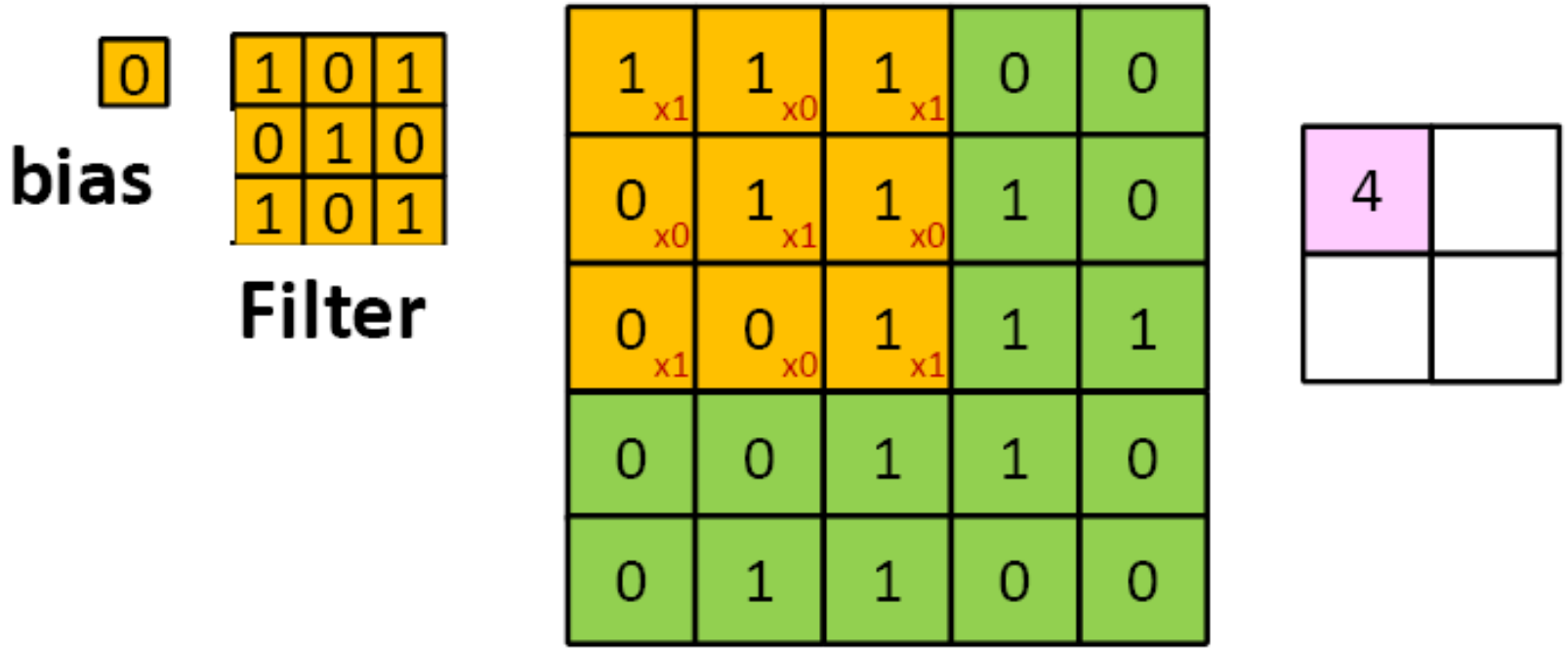
- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*



$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

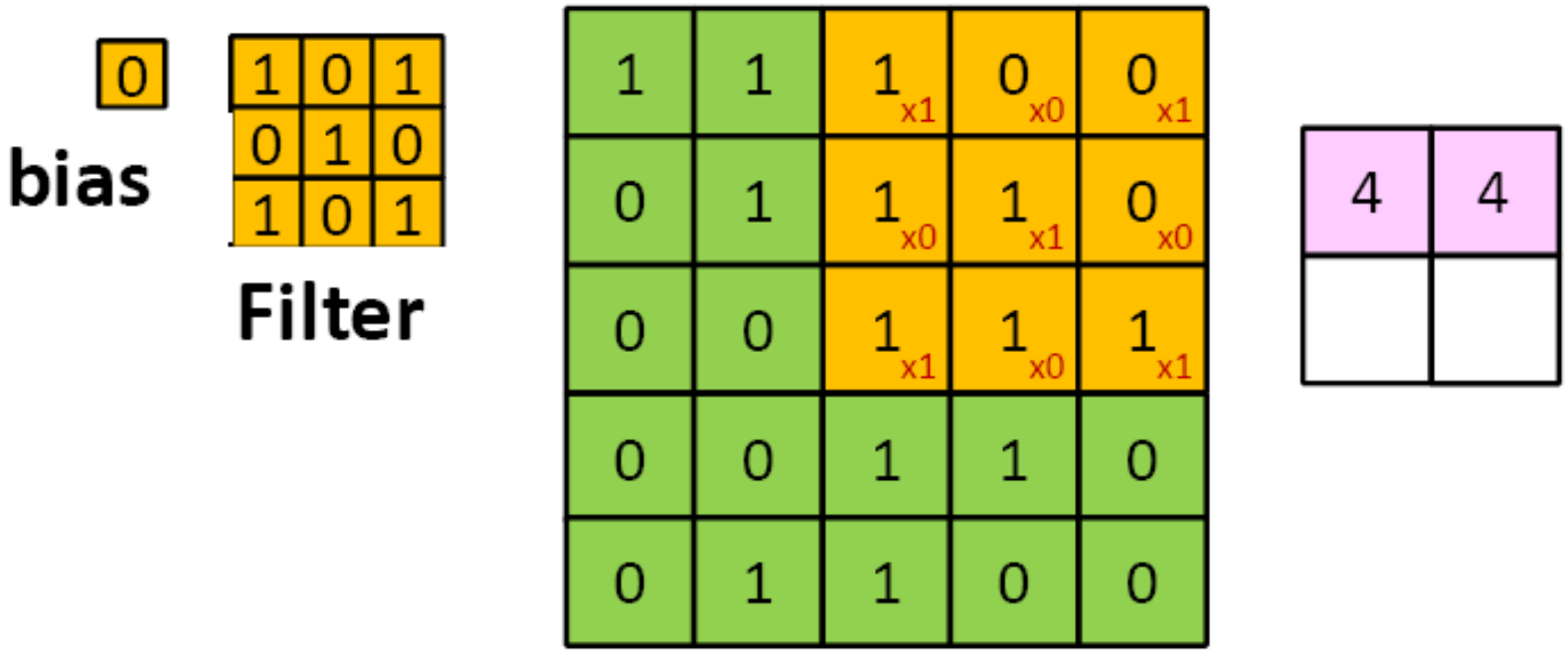
- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

Convolution can shrink a map by using strides greater than 1



- Scanning an image with a "filter"
  - The filter may proceed by *more* than 1 pixel at a time
  - E.g. with a "stride" of *two* pixels per shift

Convolution can shrink a map by using strides greater than 1

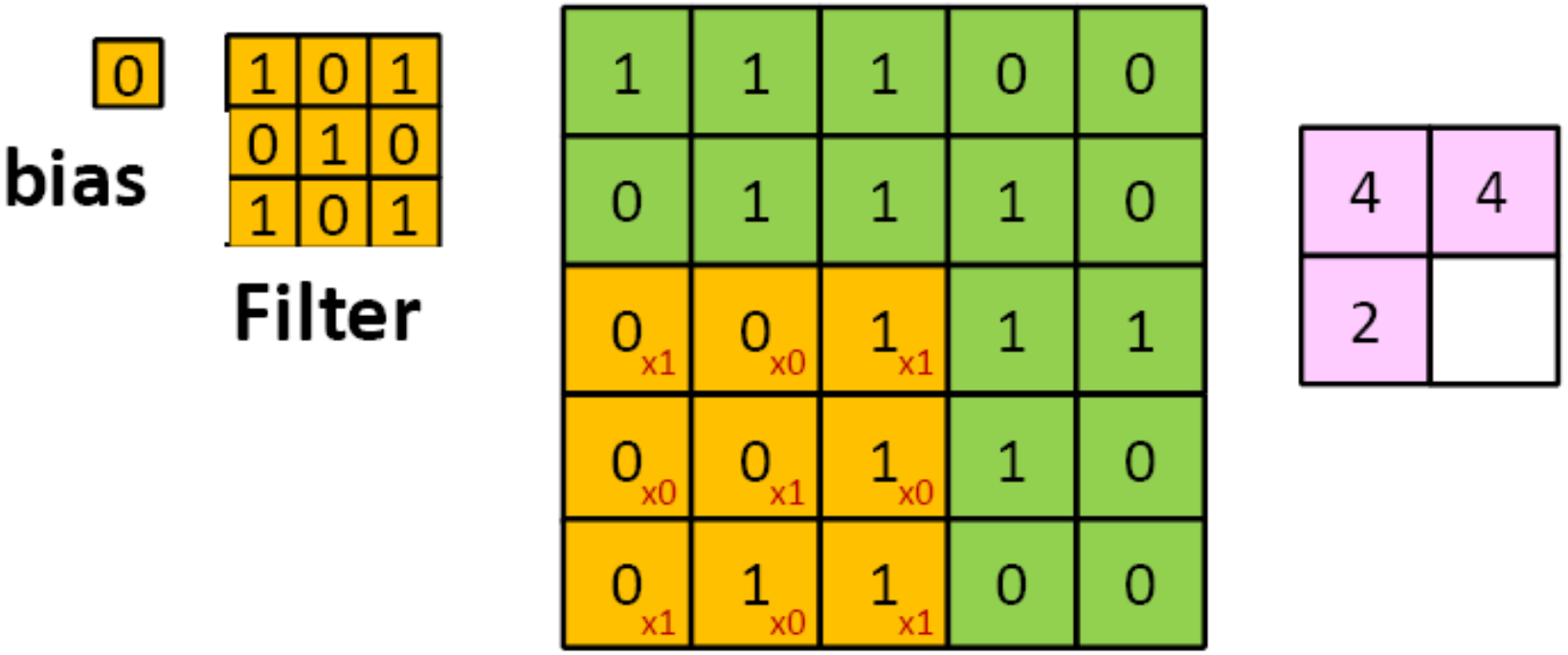


Scanning an image with a “filter”

– The filter may proceed by more than 1 pixel at a time

– E.g. with a “stride” of two pixels per shift

Convolution can shrink a map by using strides greater than 1

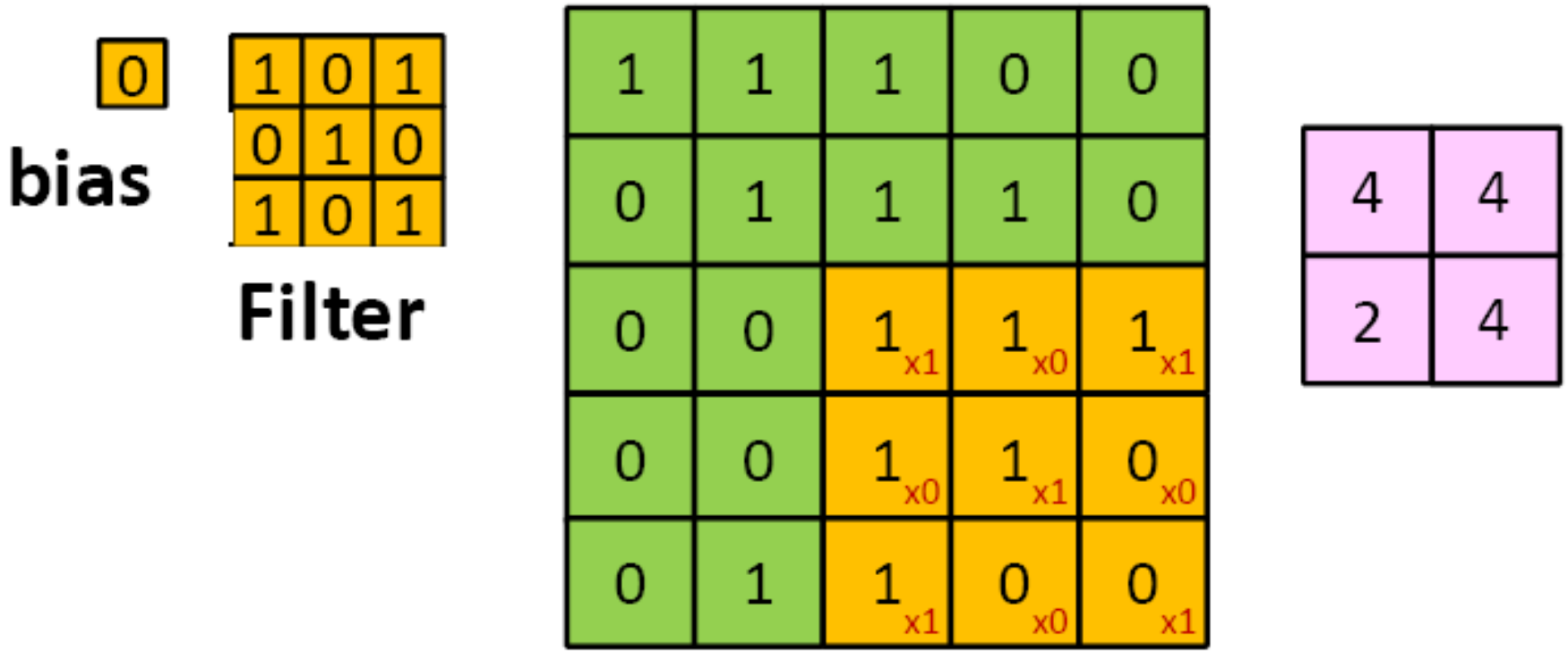


Scanning an image with a “filter”

– The filter may proceed by more than 1 pixel at a time

– E.g. with a “stride” of two pixels per shift

Convolution can shrink a map by using strides greater than 1



Scanning an image with a “filter”

– The filter may proceed by more than 1 pixel at a time

– E.g. with a “stride” of two pixels per shift

Preview

[Zeiler and Fergus 2013]

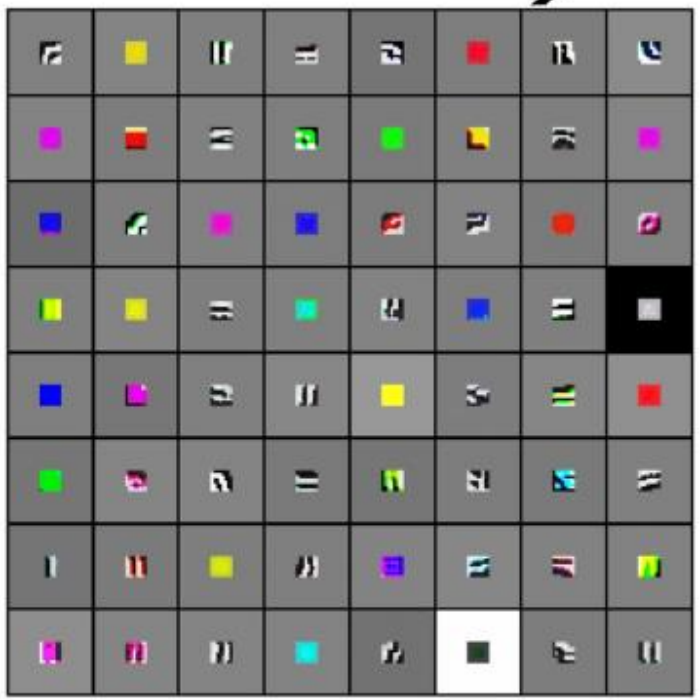
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

Low-level features

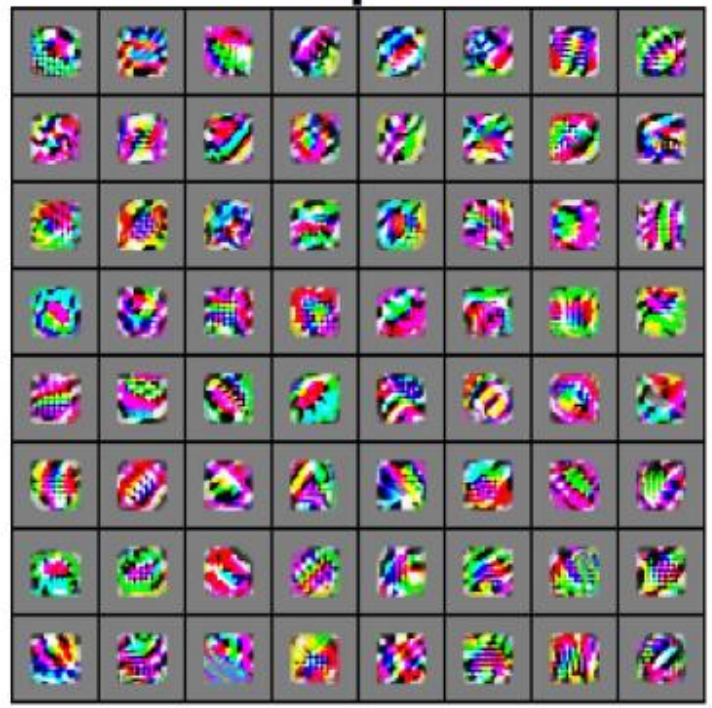
Mid-level features

High-level features

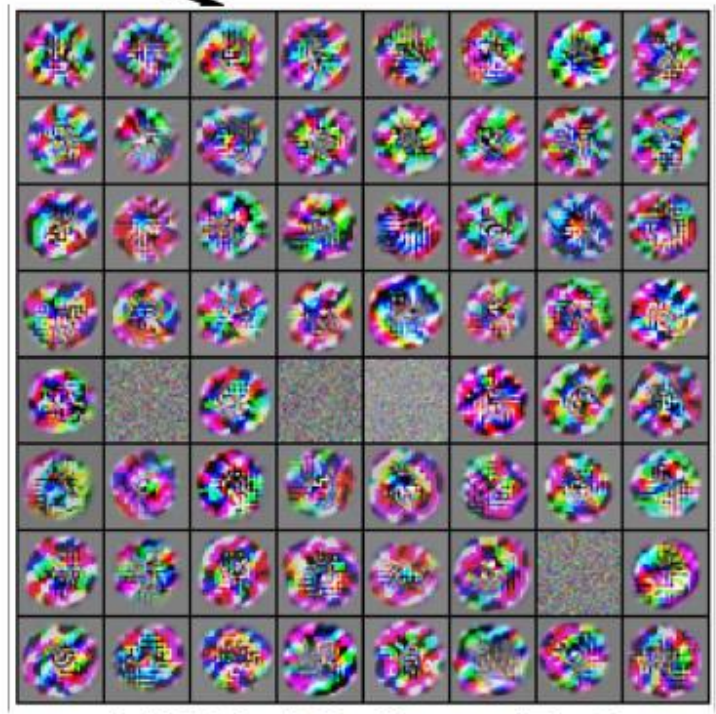
Linearly separable classifier



VGG-16 Conv1_1



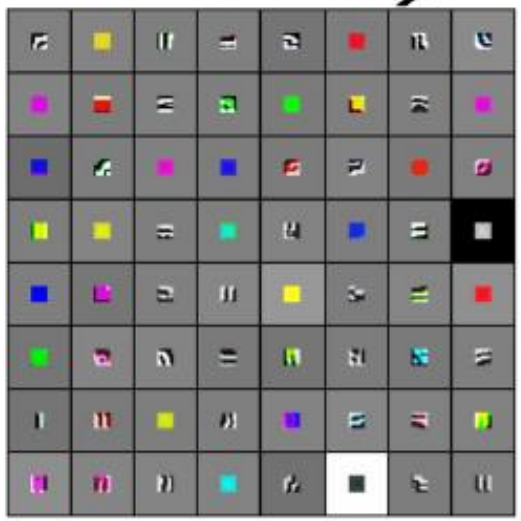
VGG-16 Conv3_2



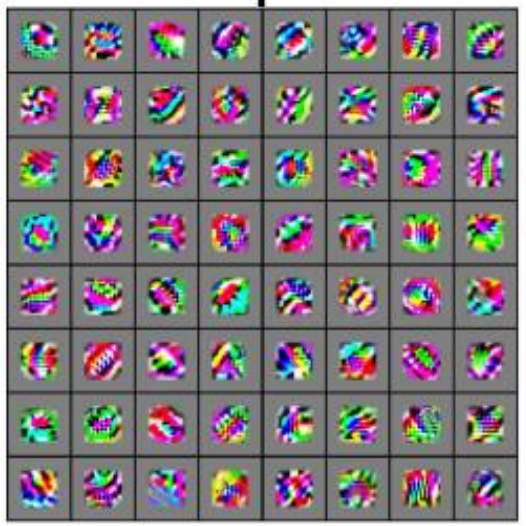
VGG-16 Conv5_3

Preview

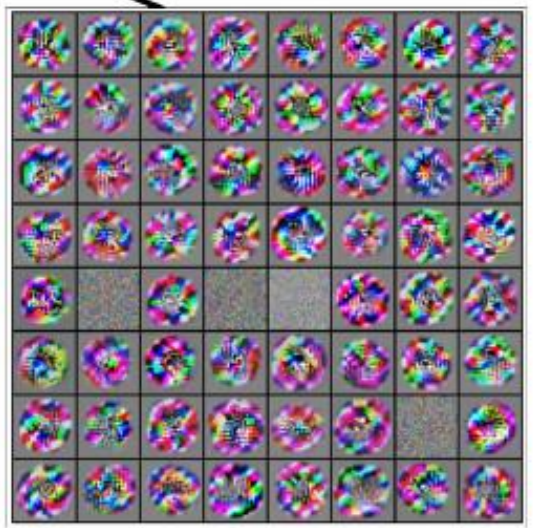
Low-level features → Mid-level features → High-level features → Linearly separable classifier



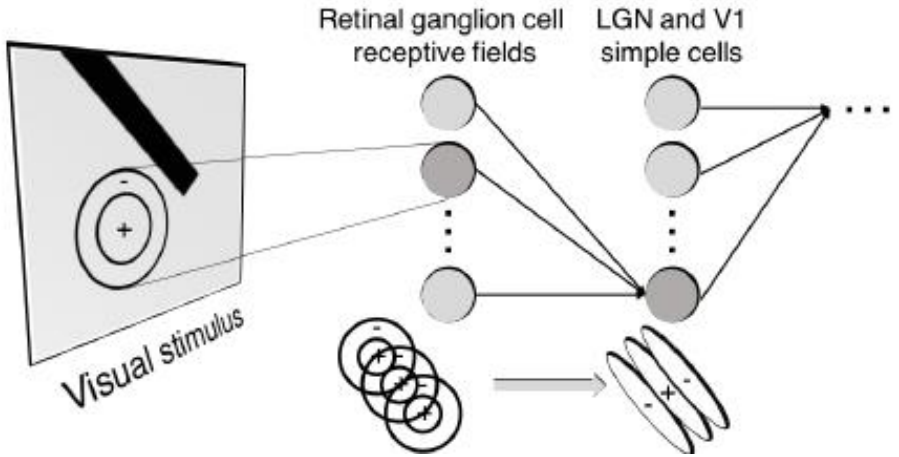
VGG-16 Conv1_1



VGG-16 Conv3_2



VGG-16 Conv5_3



Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

Example 1. For example, suppose that the input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter). Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Example 2. Suppose an input volume had size [16x16x20]. Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3*3*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth (20).

one filter =>
one activation map

example 5x5 filters
(32 total)

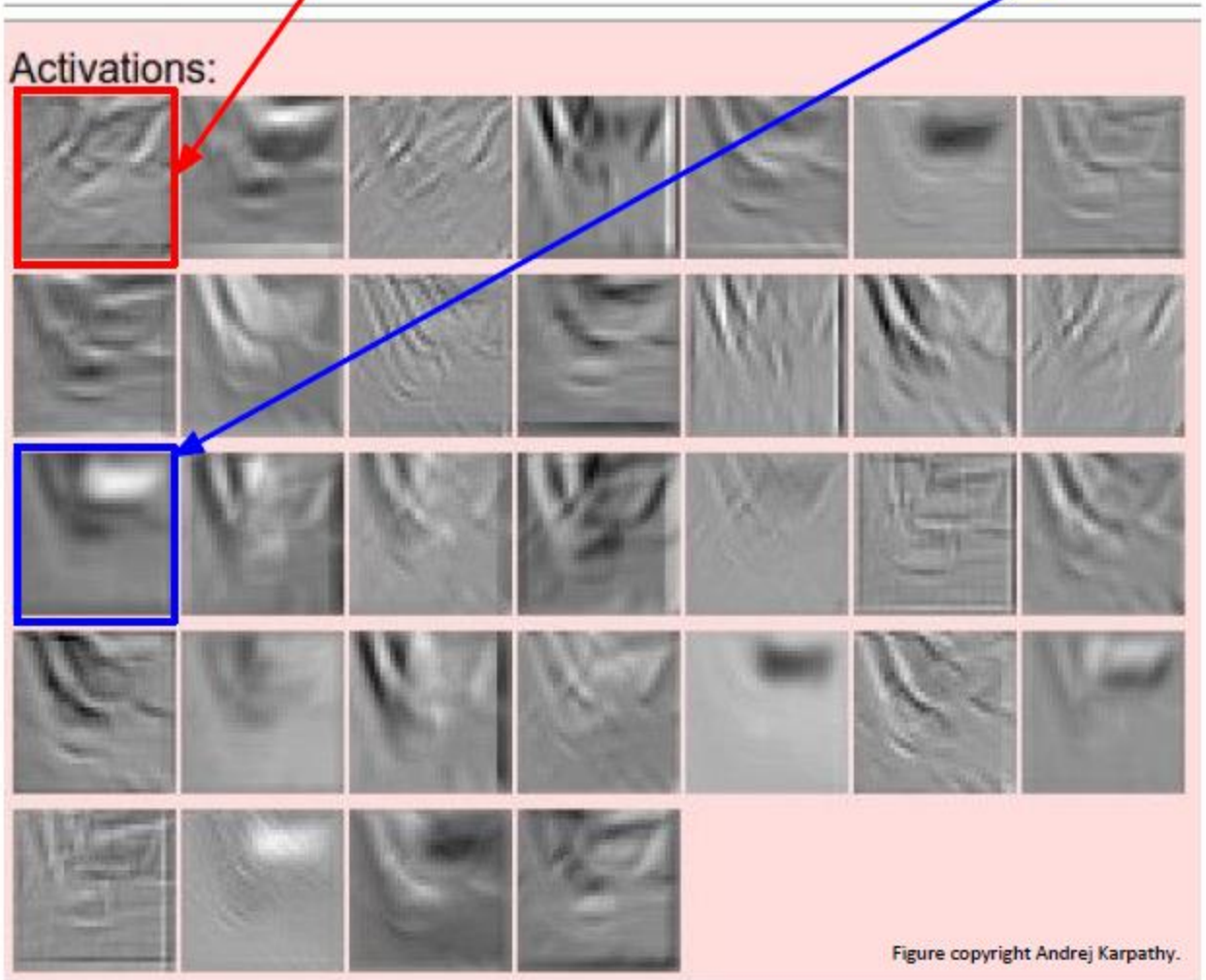


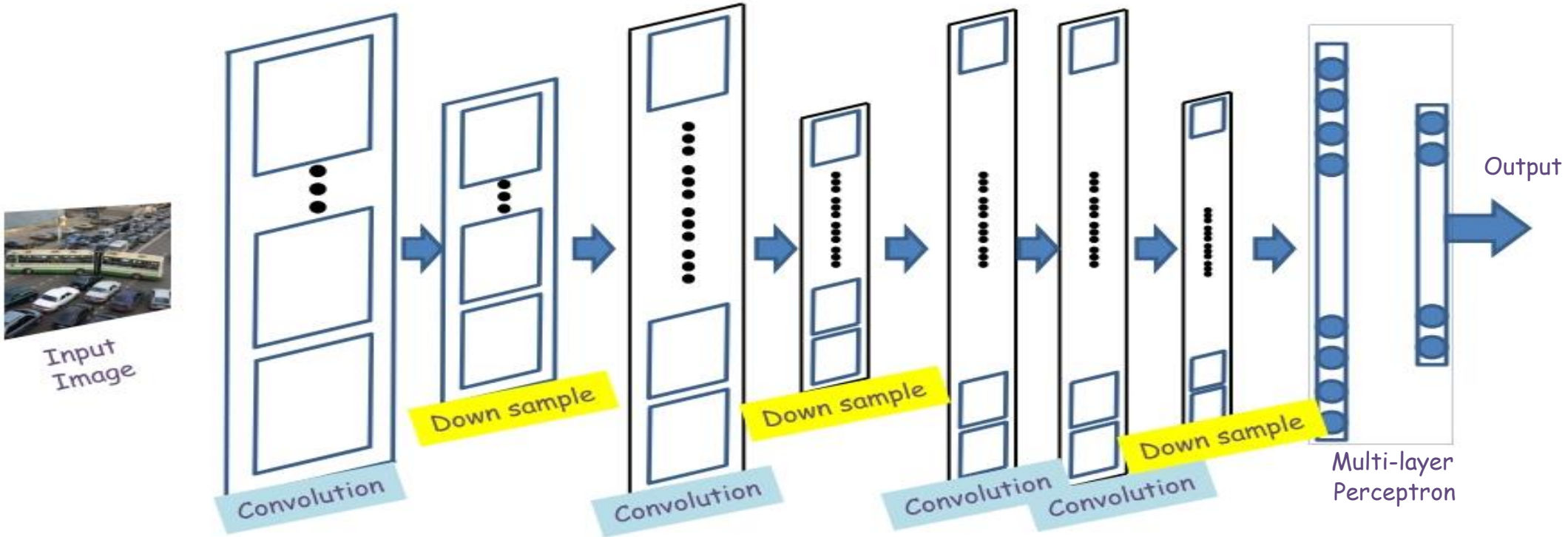
Figure copyright Andrej Karpathy.

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

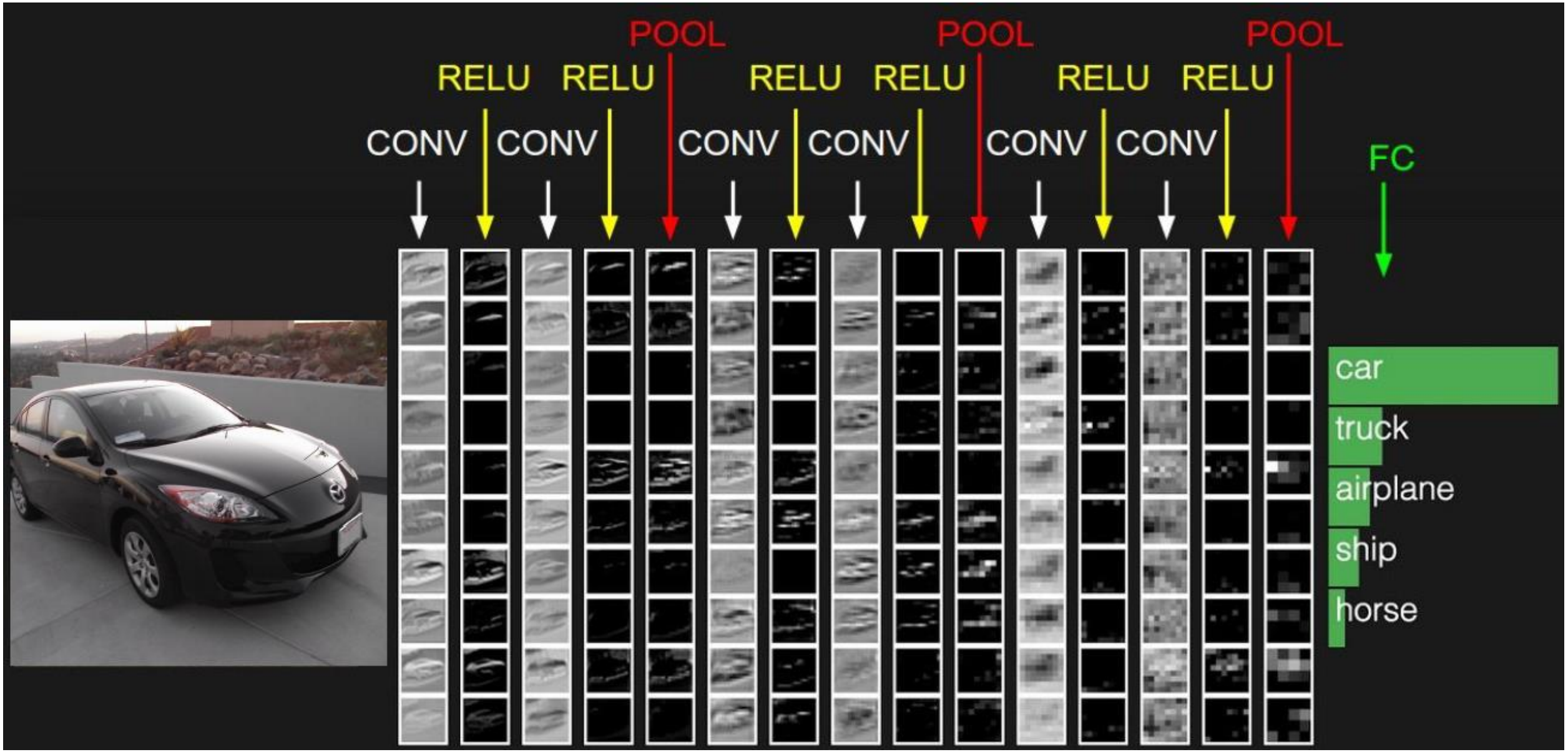
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

The general architecture of a convolutional neural network

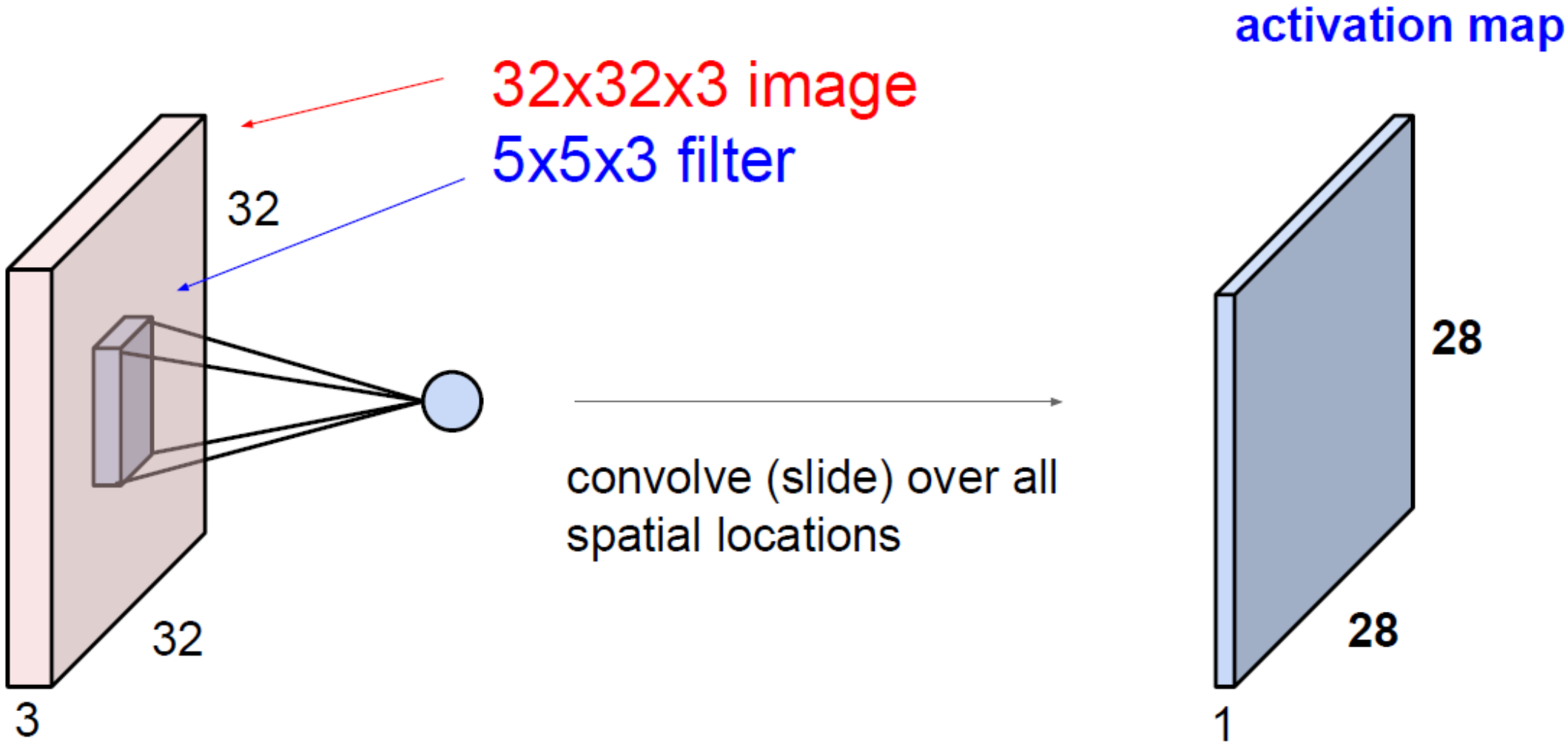


- A convolutional neural network comprises of “convolutional” and optional “downsampling” layers
- Followed by an MLP with one or more layers


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

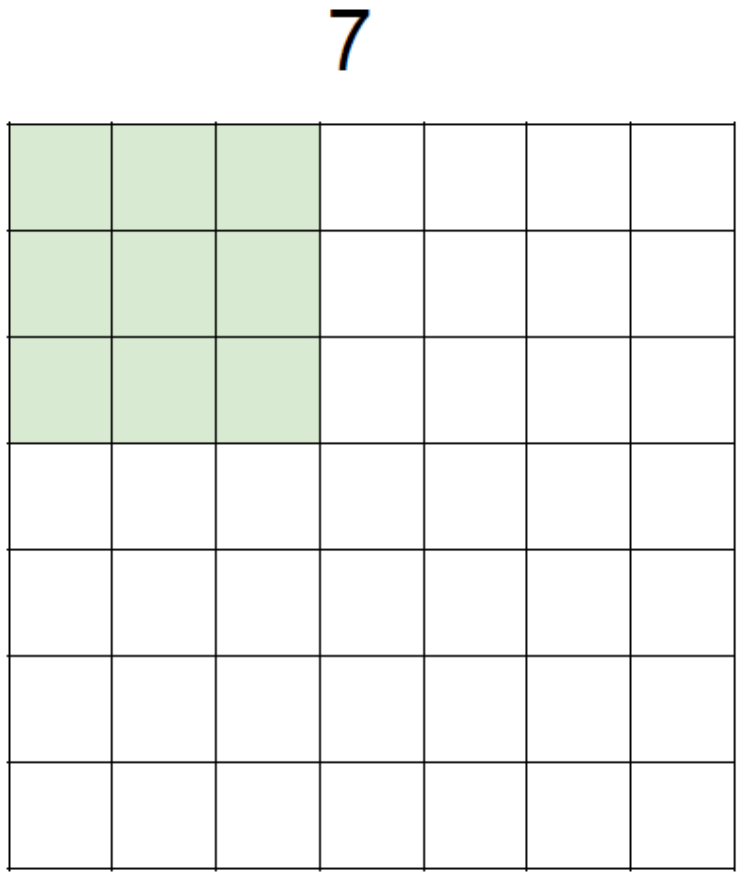
Spatial Dimensions (1)

A closer look at spatial dimensions:



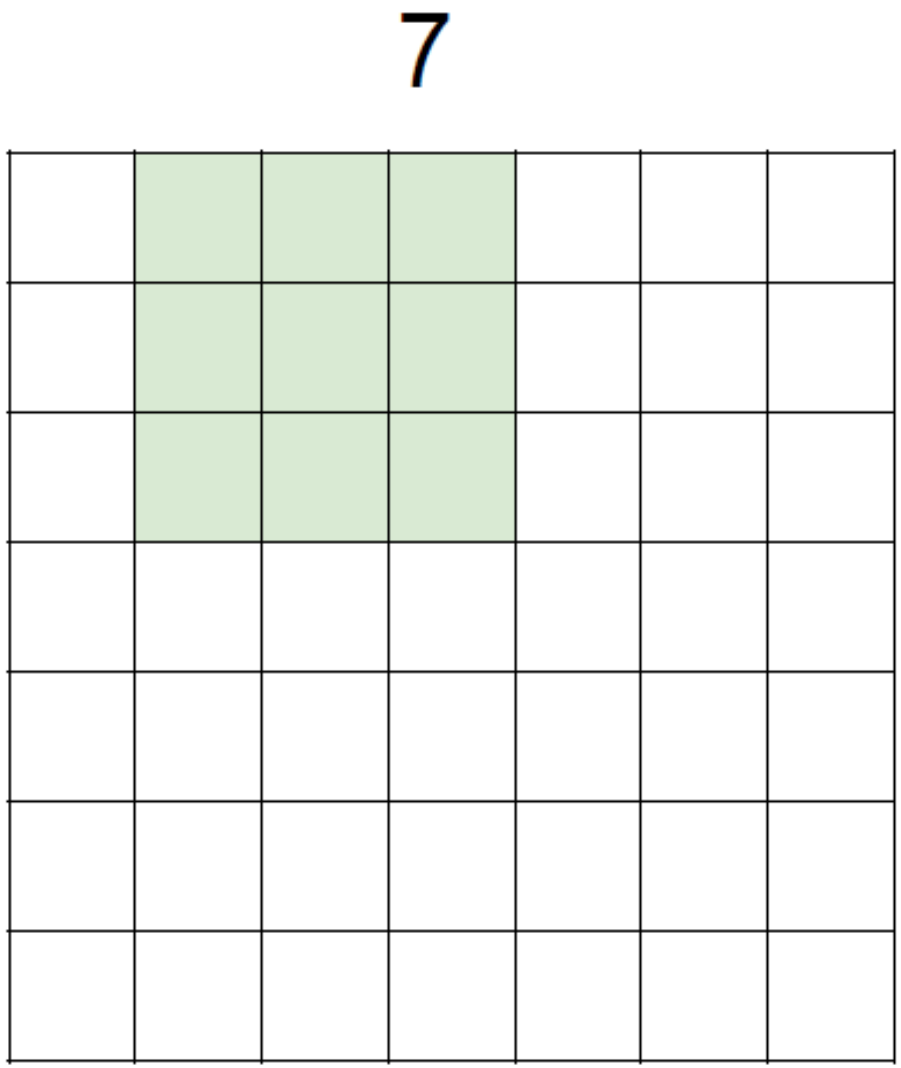
Spatial Dimensions (2)

A closer look at spatial dimensions:



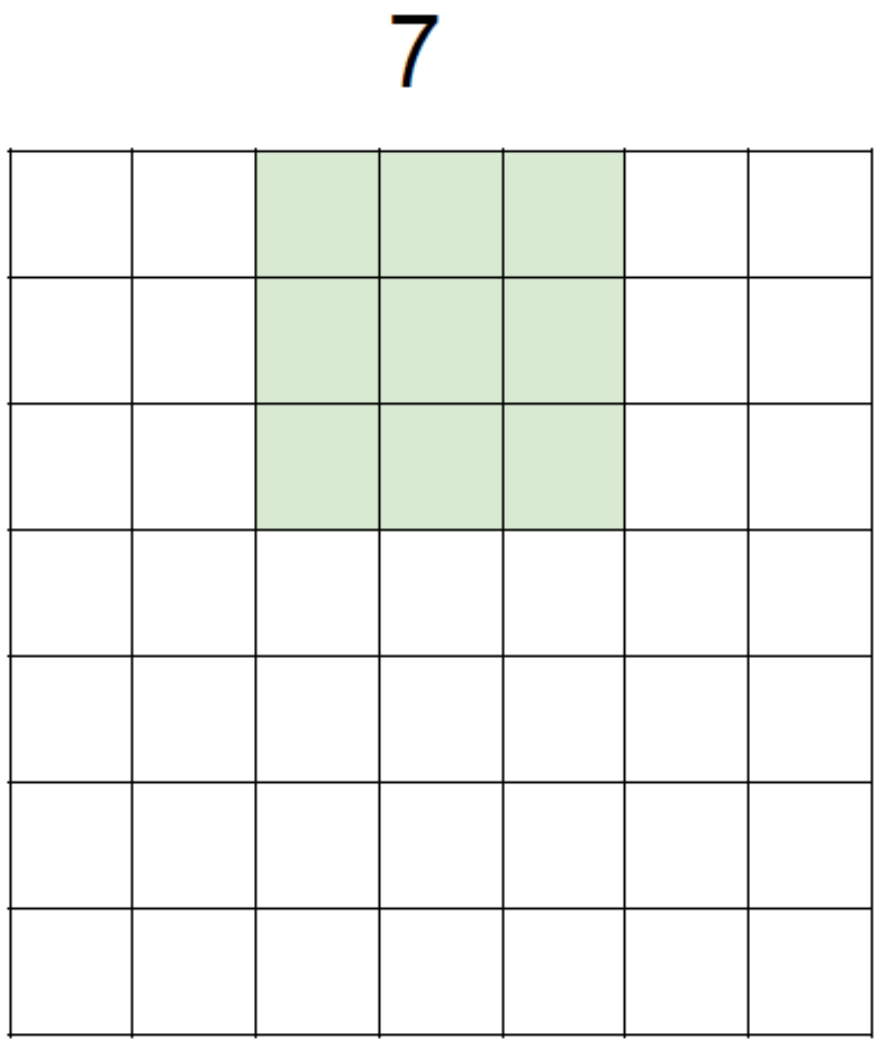
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



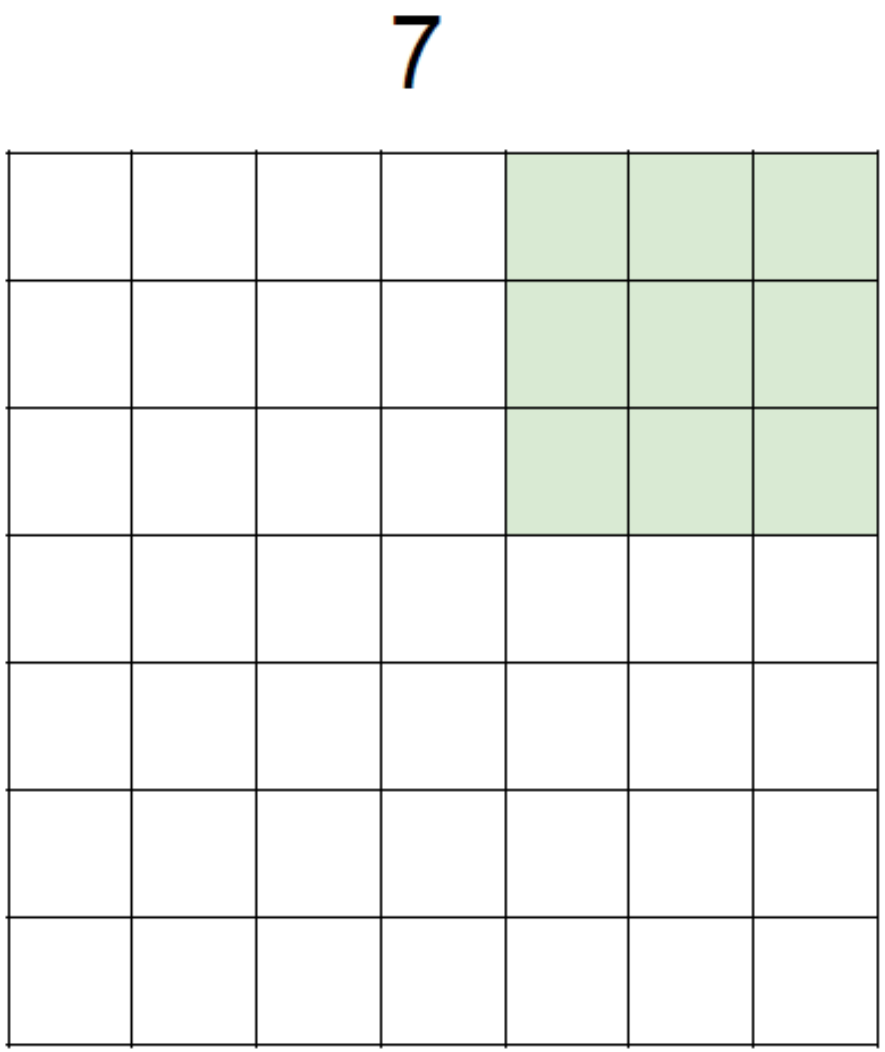
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

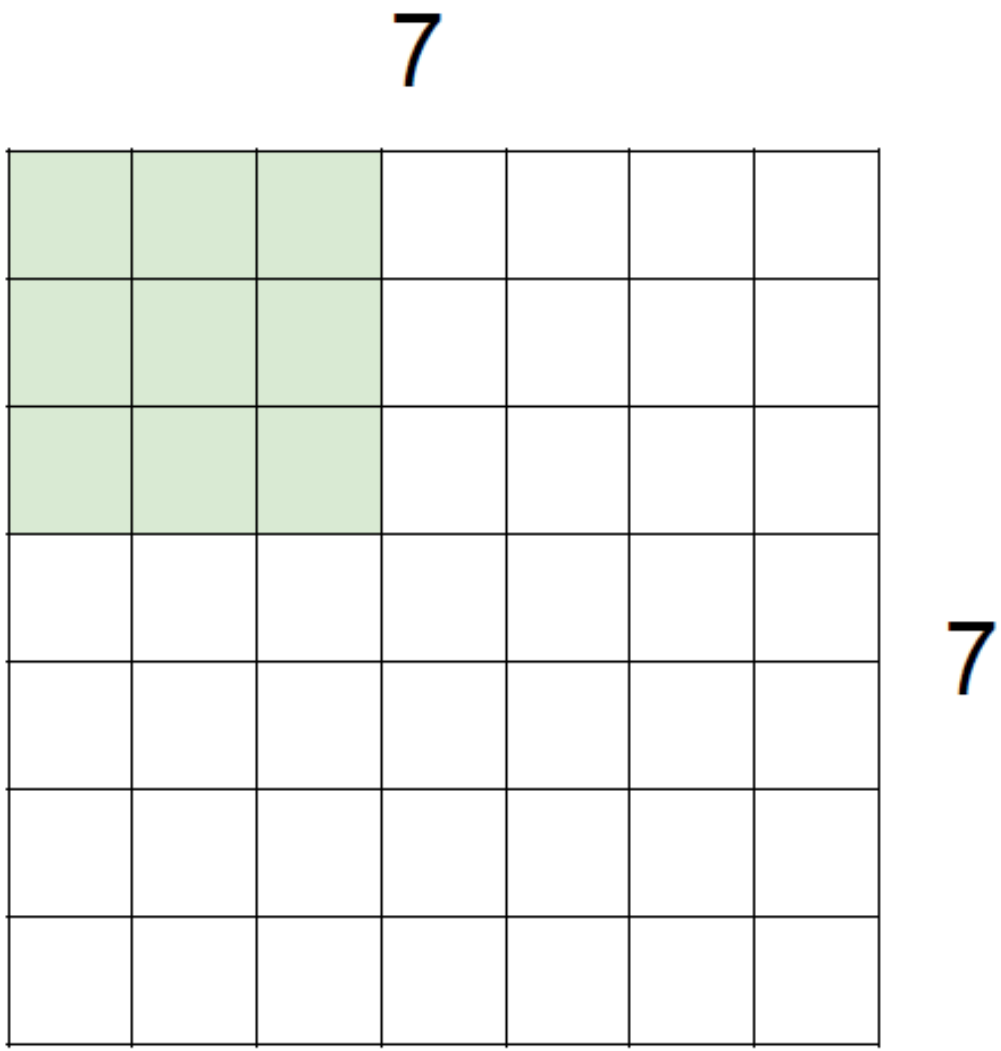
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

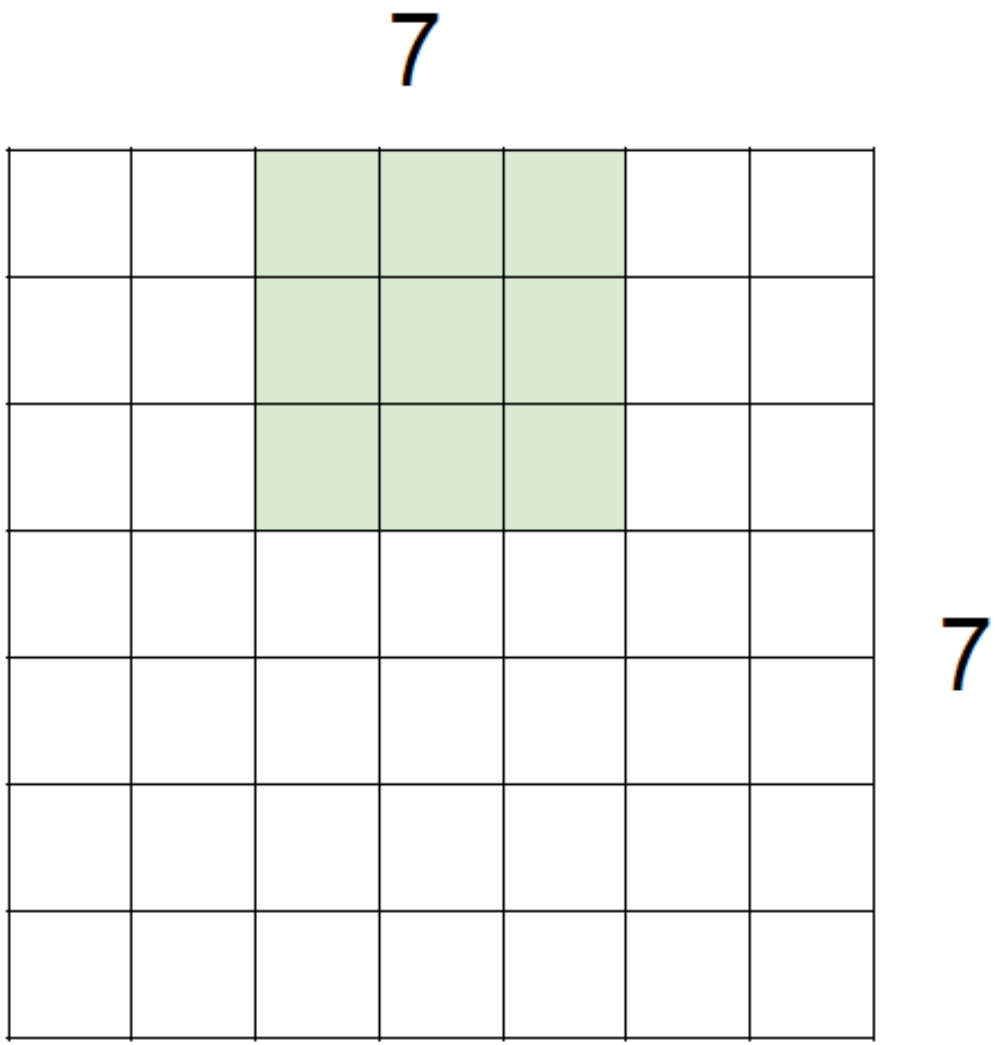
=> 5x5 output

A closer look at spatial dimensions:



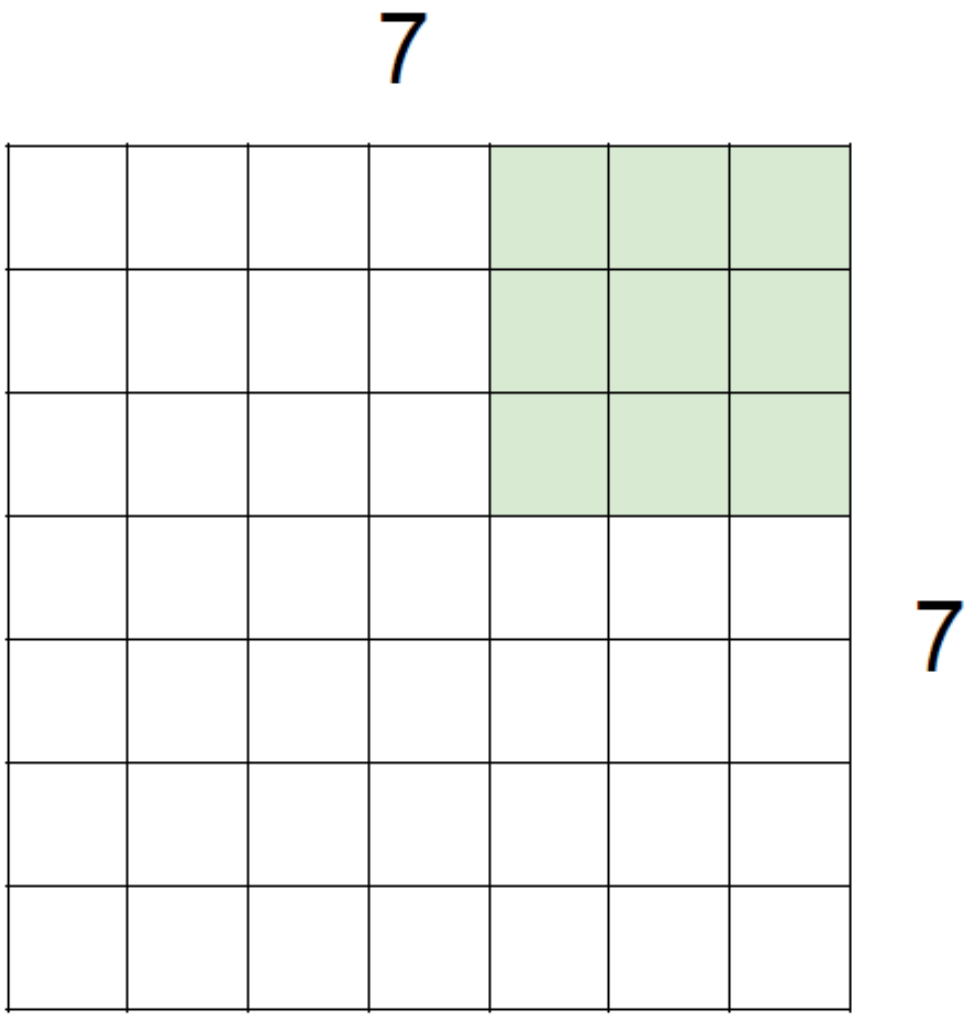
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



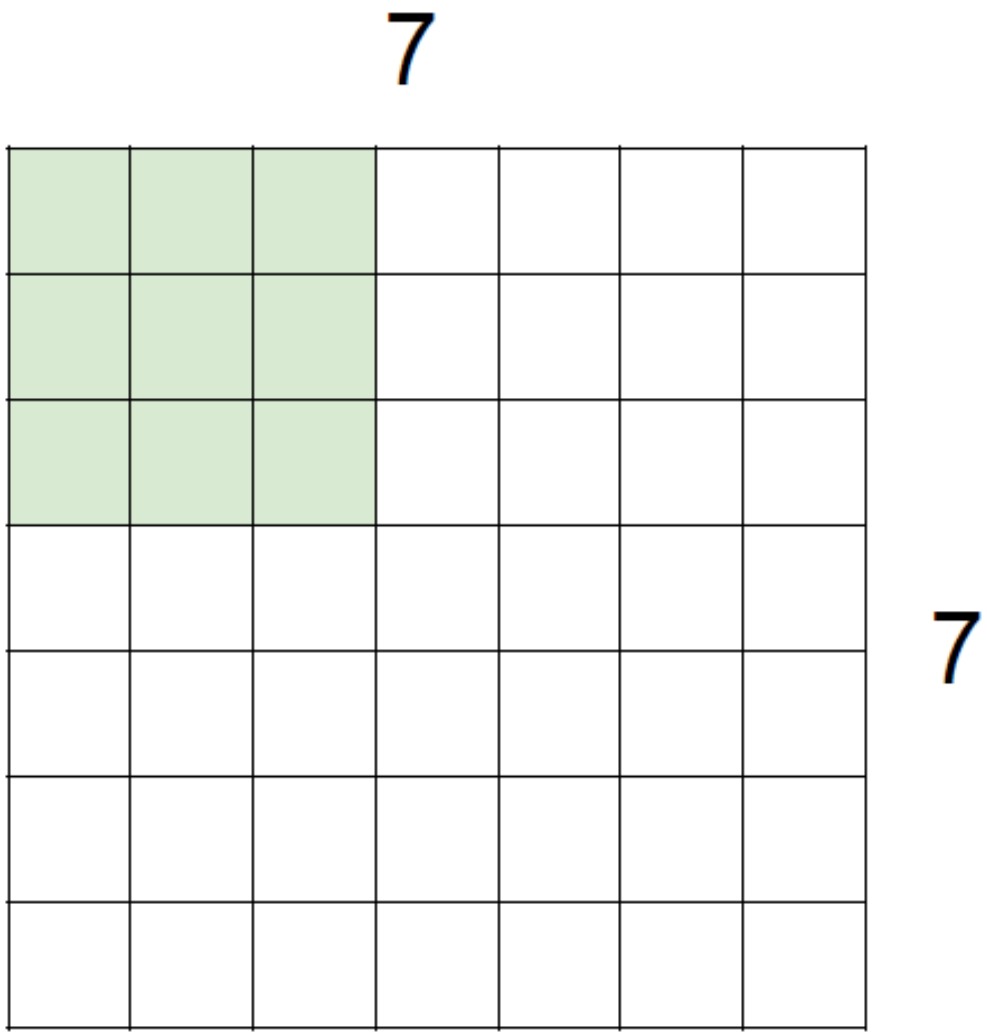
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



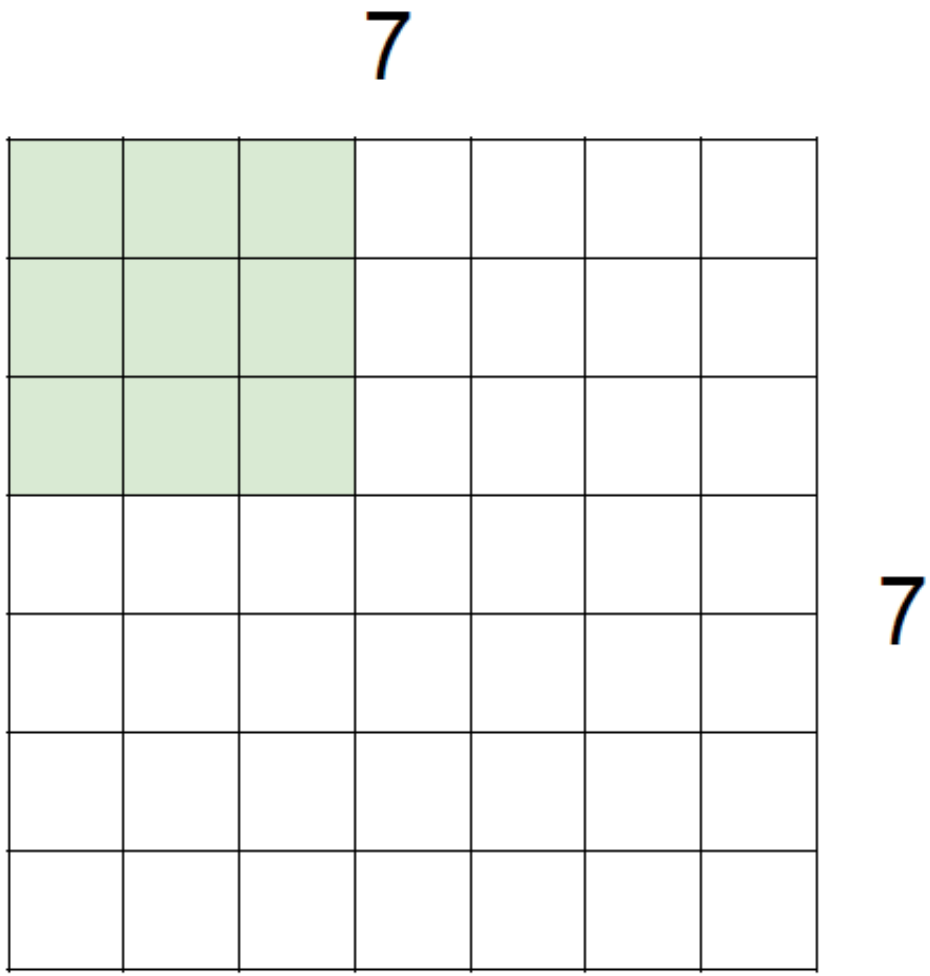
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



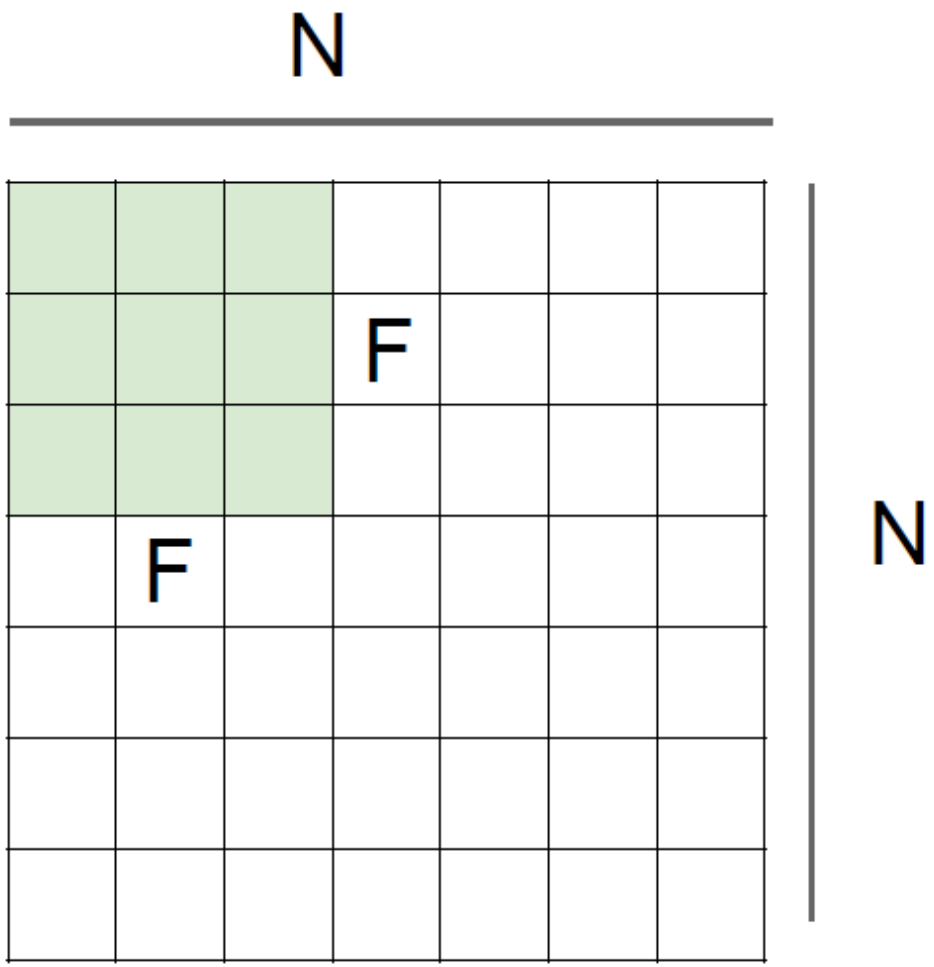
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

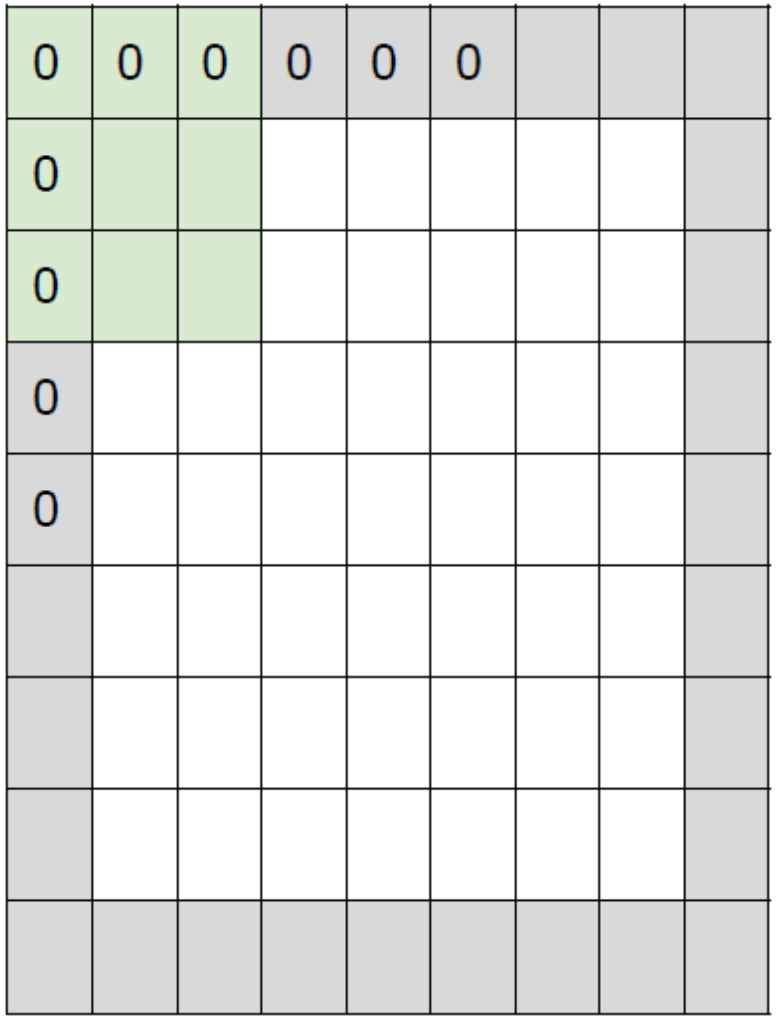
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

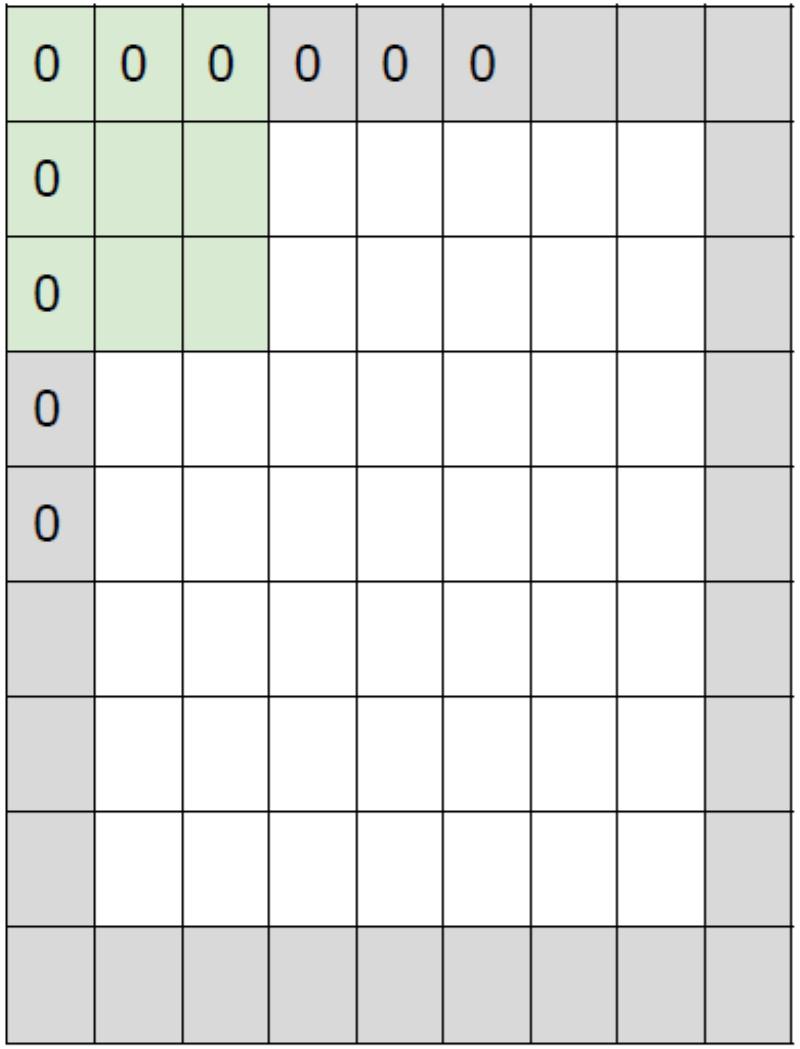
In Practice: Common to Zero Pad the Border (1)



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

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

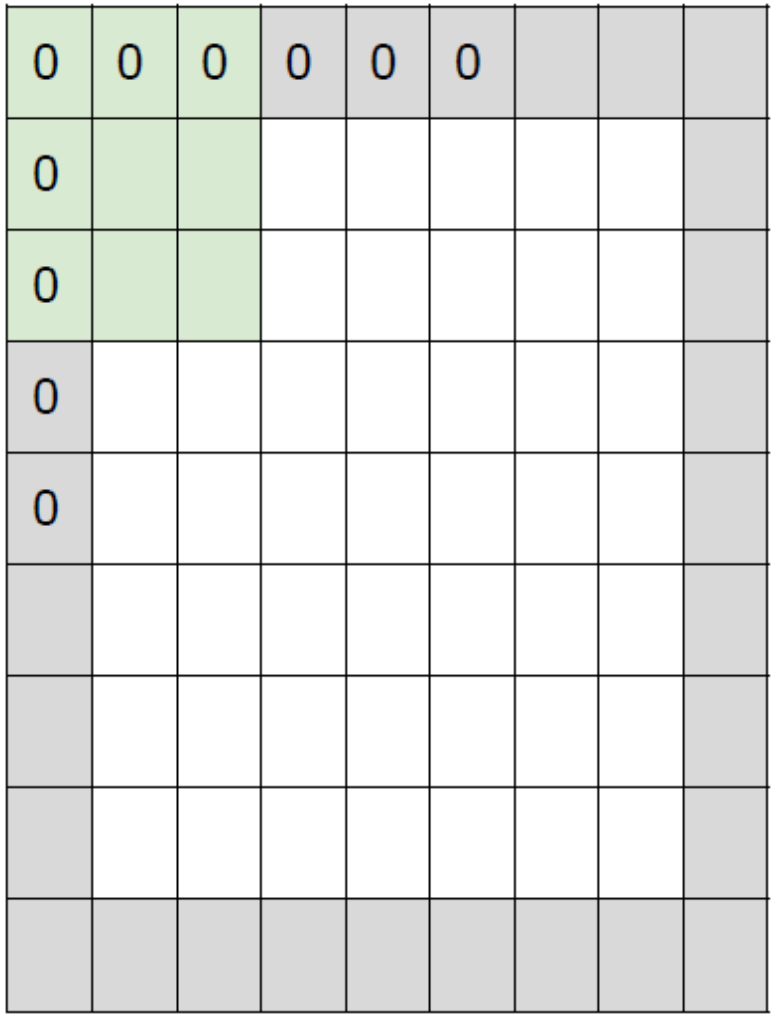
In Practice: Common to Zero Pad the Border (2)



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 (3)

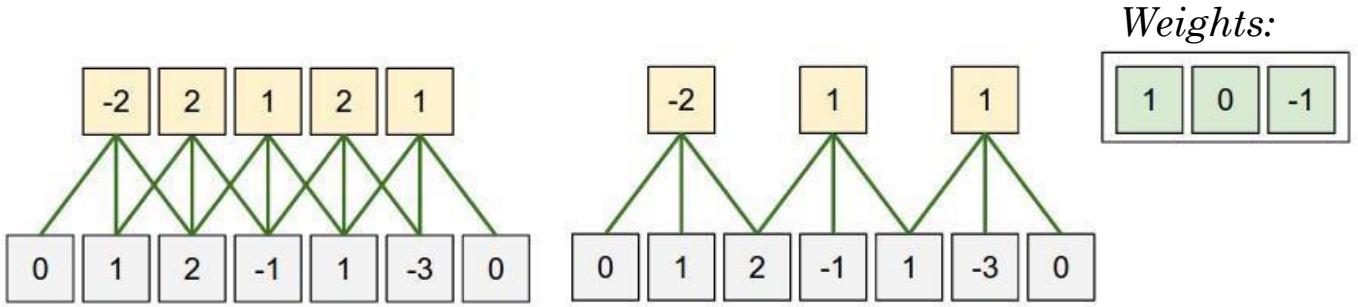


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Important Parameters

- Depth (number of channels)
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signal in the same receptive field
- Stride
  - The amount of space between neighboring receptive fields
  - If it is small, RFs overlap more
  - It it is big, RFs overlap less
- How to handle the boundaries?
  i. Option 1: Don't process the boundaries. Only process pixels on which convolution window can be placed fully.
  ii. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.



Padding Illustration

- Only convolution layers are shown.
- Top: no padding ➔ layers shrink in size.
- Bottom: zero padding ➔ layers keep their size fixed.

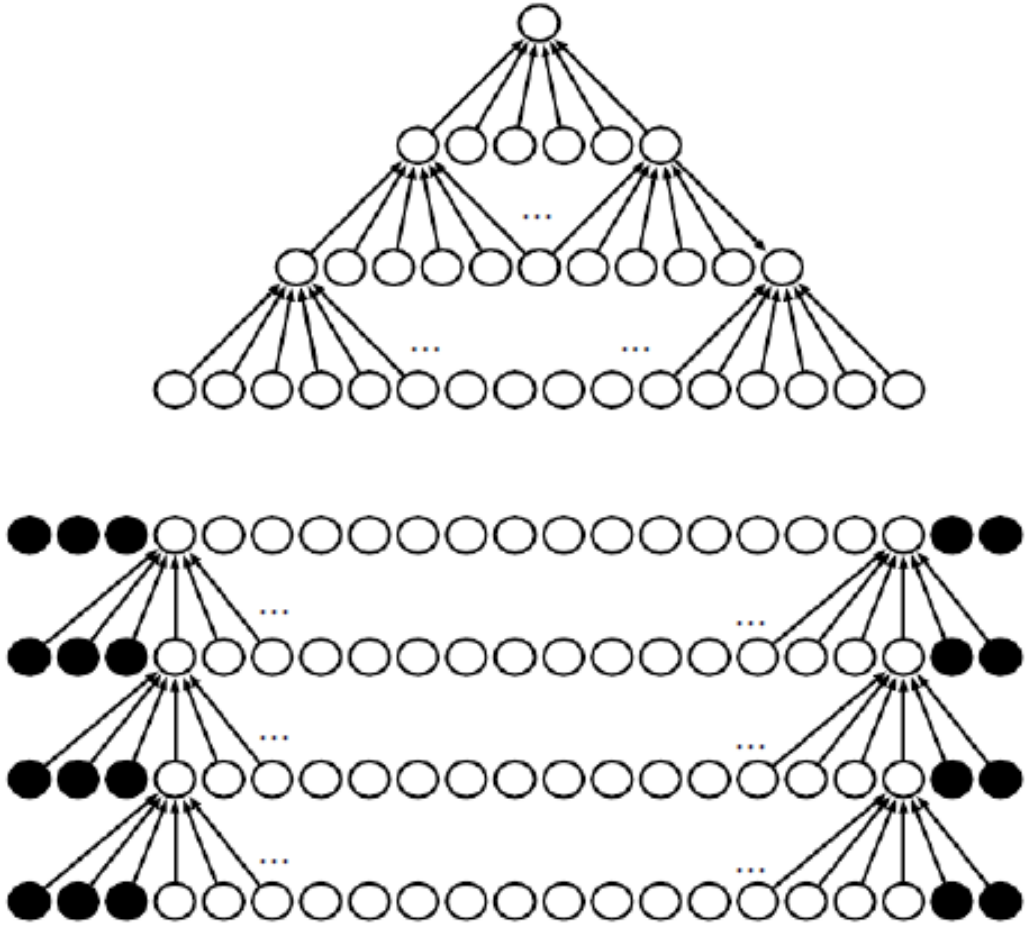


Figure 9.13: *The effect of zero padding on network size*: Consider a convolutional network with a kernel of width six at every layer. In this example, we do not use any pooling, so only the convolution operation itself shrinks the network size. *(Top)* In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. *(Bottom)* By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

Size of the Network (1)

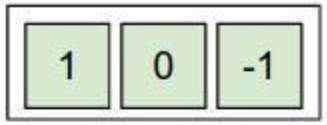
- Along a dimension:
  - W: Size of the input
  - F: Size of the receptive field (filter size)
  - S: Stride
  - P: Amount of zero-padding
- Then: the number of neurons in the output:

$$\frac{W - F + 2P}{S} + 1$$

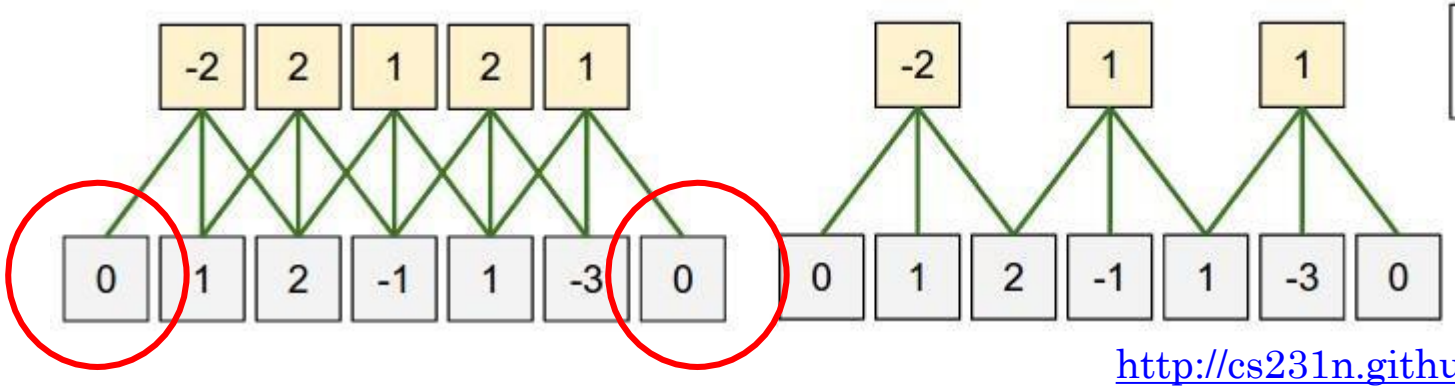
- If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume

Illustration of spatial arrangement at the bottom: In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1.
Left: The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5.
Right: The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3.
Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3.
The neuron weights are in this example [1,0,-1] (shown on right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Weights:



Zero padding



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

Size of the Network (2)

We can compute the spatial size of the output volume as a function of the input volume size ($\boldsymbol{W}$), the receptive field size of the Conv Layer neurons ($\boldsymbol{F}$), the stride with which they are applied ($\boldsymbol{S}$), and the amount of zero padding used ($\boldsymbol{P}$) on the border. You can convince yourself that the correct formula for calculating how many neurons "fit" is given by $(\boldsymbol{W} - \boldsymbol{F} + \boldsymbol{2P})/\boldsymbol{S} + \boldsymbol{1}$. For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output. With stride 2 we would get a 3x3 output. Lets also see one more graphical example:

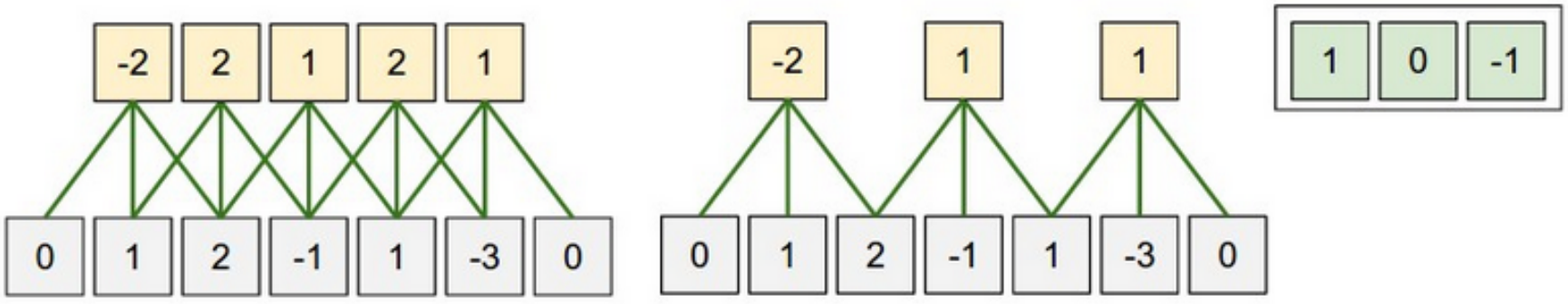


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. **Left:** The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. **Right:** The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3. Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3.

The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Size of the Network (3)

- Arranging these hyperparameters can be problematic
- Example:
- If W=10, P=0, and F=3, then

$$\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 0}{S} + 1 = \frac{7}{S} + 1$$

  i.e., S cannot be an integer other than 1 or 7.

- Zero-padding is your friend here.

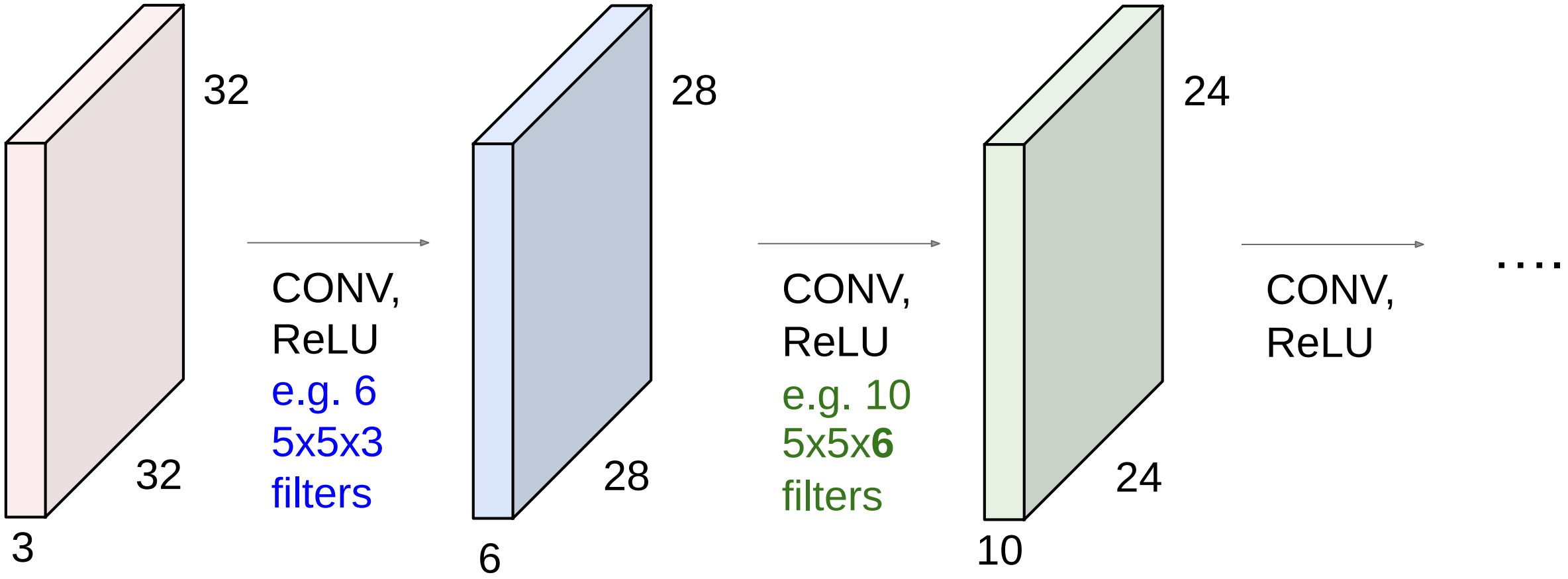
Spatial Arrangement

Three hyperparameters control the size of the output volume:
the **depth, stride** and **zero-padding**.

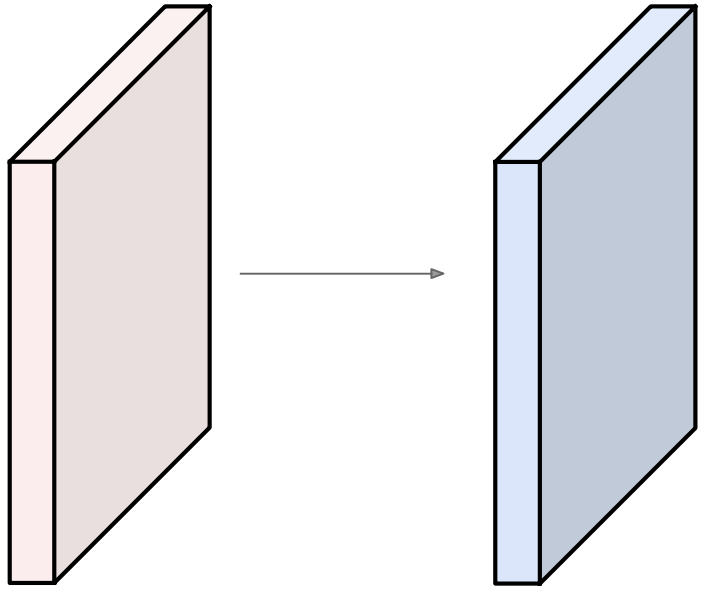
1. First, the **depth** of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We will refer to a set of neurons that are all looking at the same region of the input as a **depth column** (some people also prefer the term *fibre*).
2. Second, we must specify the **stride** with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
3. As we will soon see, sometimes it will be convenient to pad the input volume with zeros around the border. The size of this **zero-padding** is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly as we'll see soon we will use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

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.



Examples time:

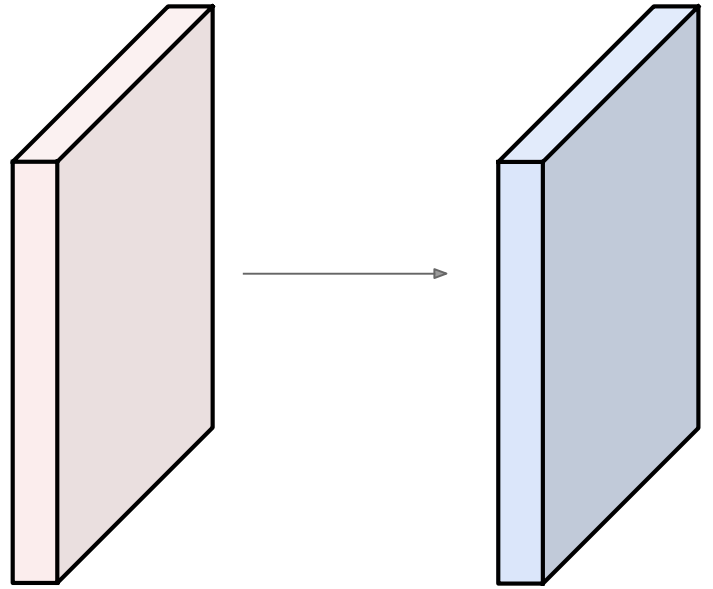


Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?

Examples time:



Input volume: **32x32x3**

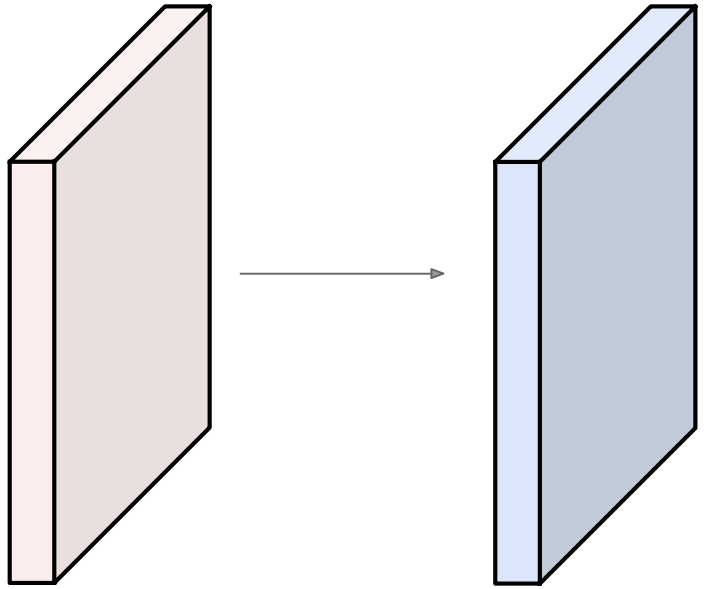
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

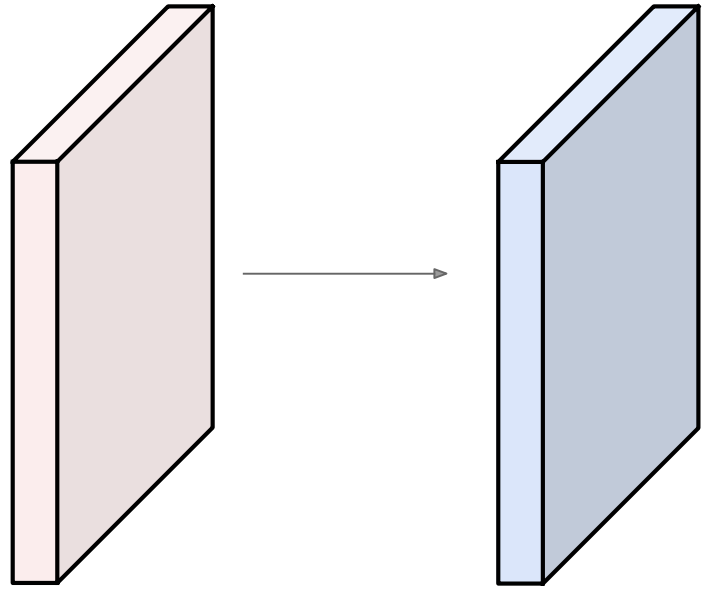
Examples time:



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

Number of parameters in this layer?

Examples 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**

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.

Summary. To summarize, the Conv Layer:

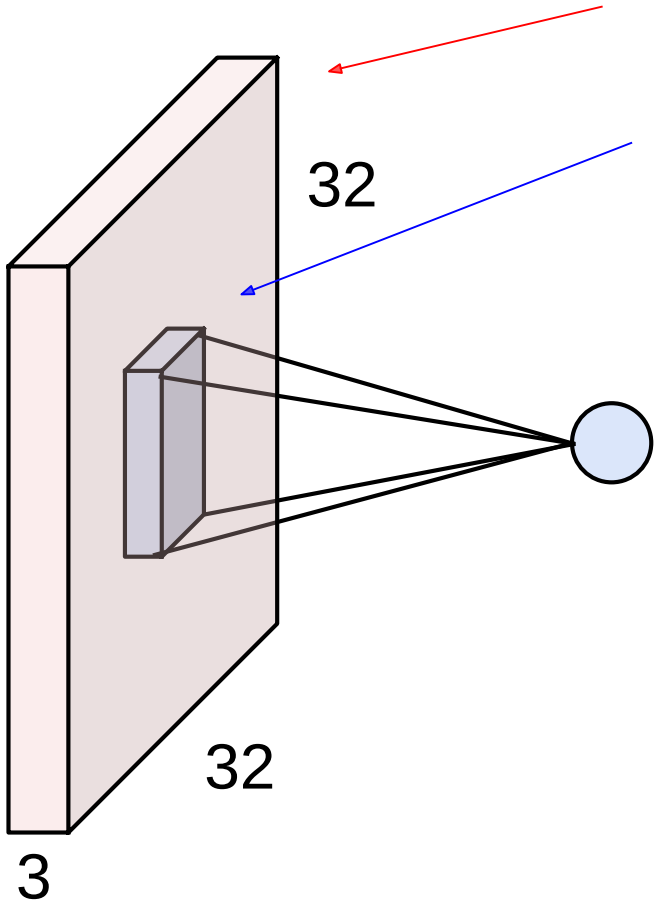
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (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 = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

The brain/neuron view of CONV Layer

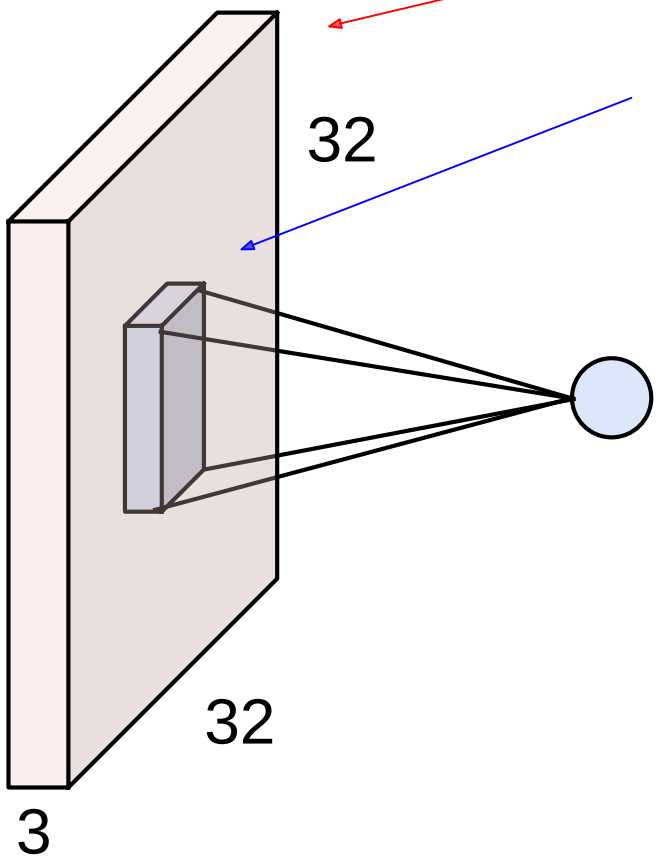


32x32x3 image

5x5x3 filter

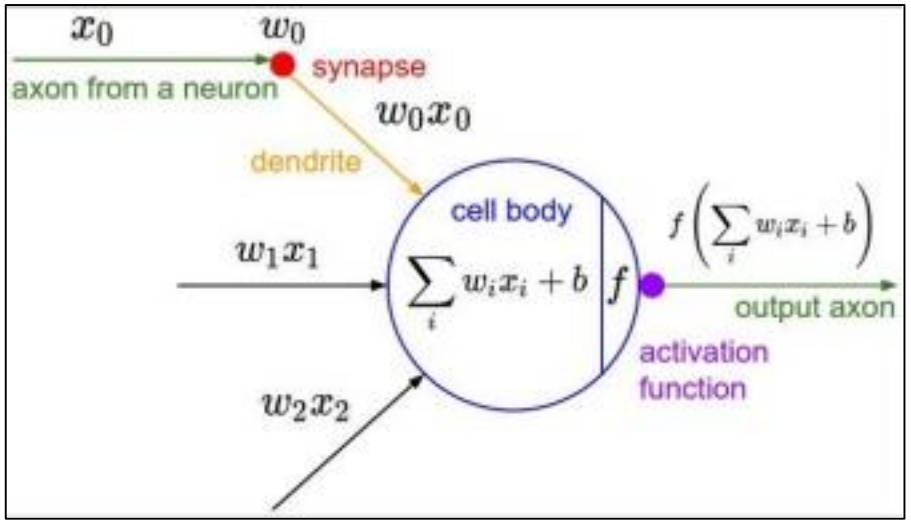
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



32x32x3 image

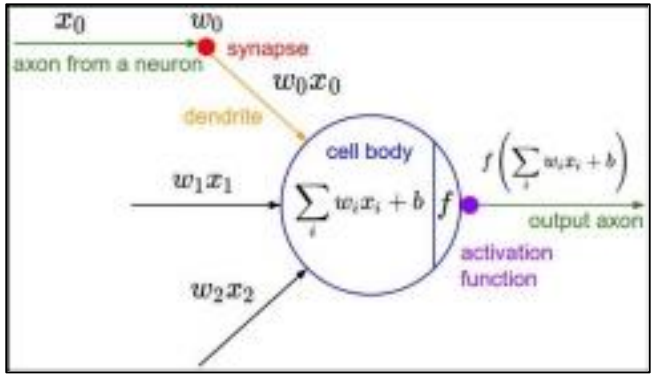
5x5x3 filter

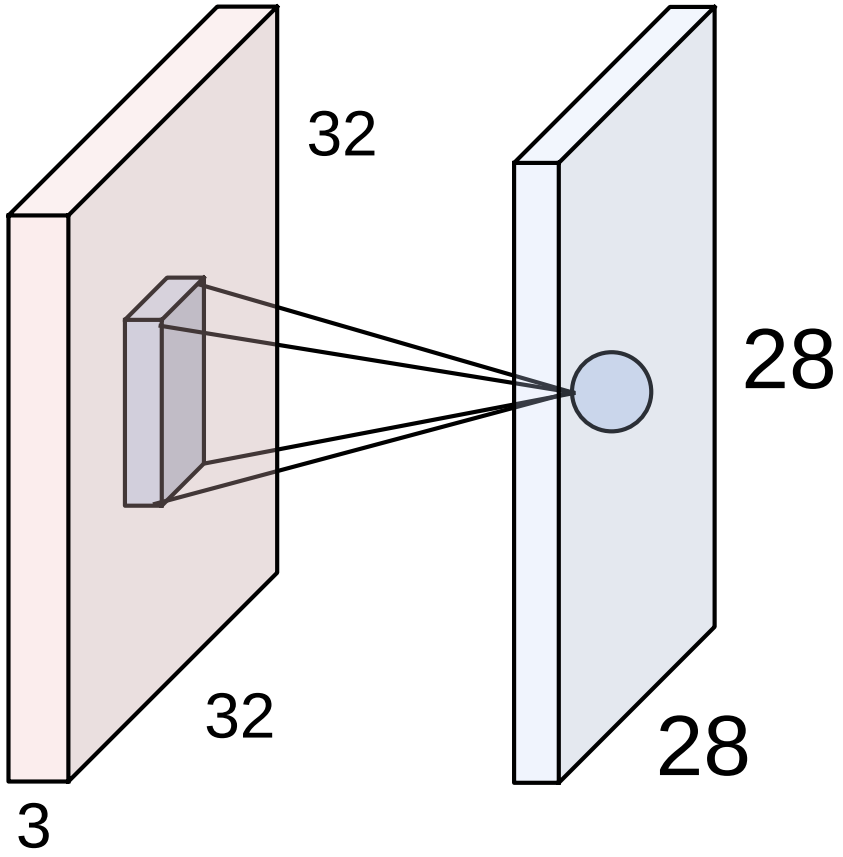


It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



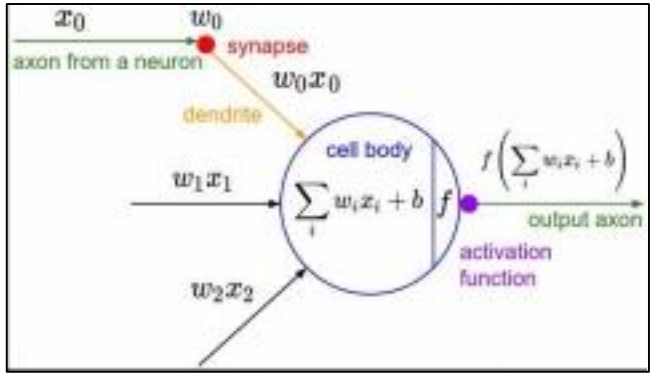


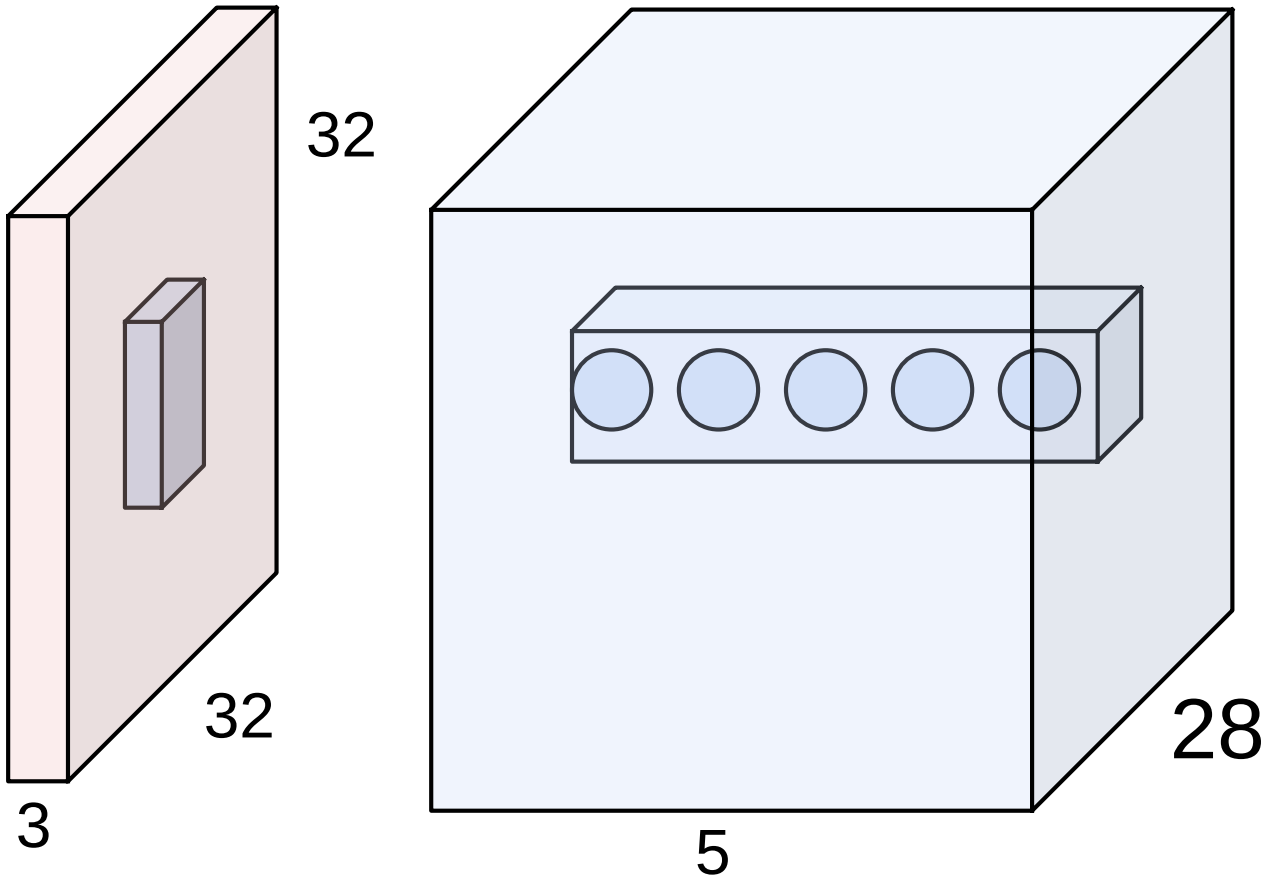
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

The brain/neuron view of CONV Layer





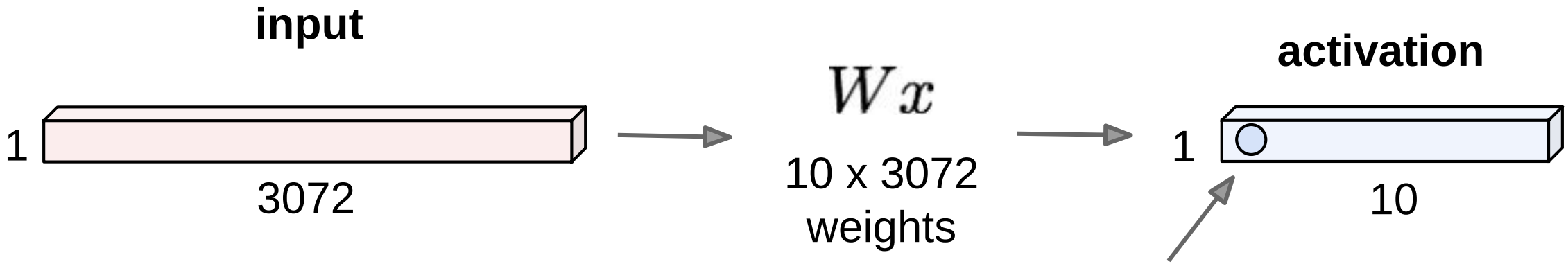
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Two More Layers to Cover: POOL and FC