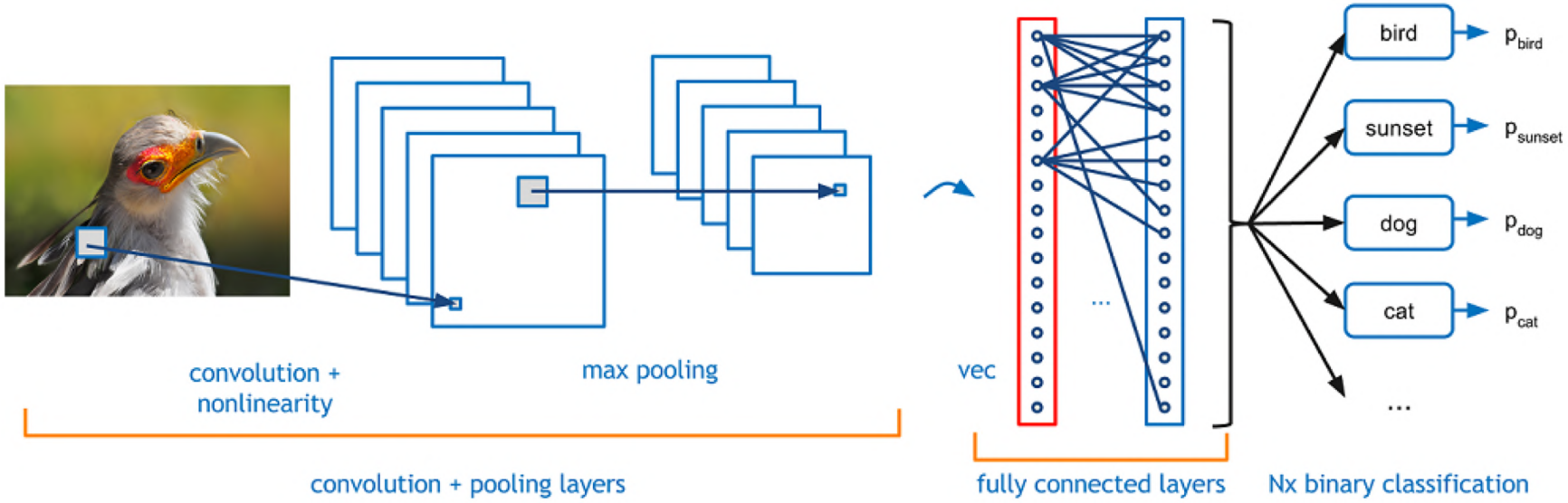
Convolutional Neural Networks



Many slides from: Stanford CS231 and Roger Grosse

Overview

What makes image analysis hard?

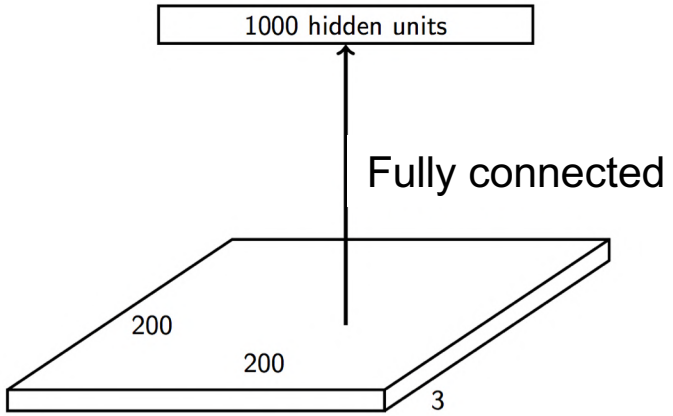
- Semantic understanding
- Vision needs to be robust to many types of transformations or distortions:
  - i. Change in pose/viewpoint
  - ii. Change in illumination
  - iii. Deformation
  - iv. Occlusion (some objects are hidden behind others)
  - v. Intra class variation (e.g. chairs)
  - vi. Contextual information

Source: Roger Grosse

Overview

Problem 1 with MLP and image analysis:

- Suppose we want to train a network that takes a 200×200 RGB image as input.



- What is the problem with having this as the first layer?

  Input size = 200 × 200 × 3 = 120K.

  Parameters = 120K×1000 = 120 million.

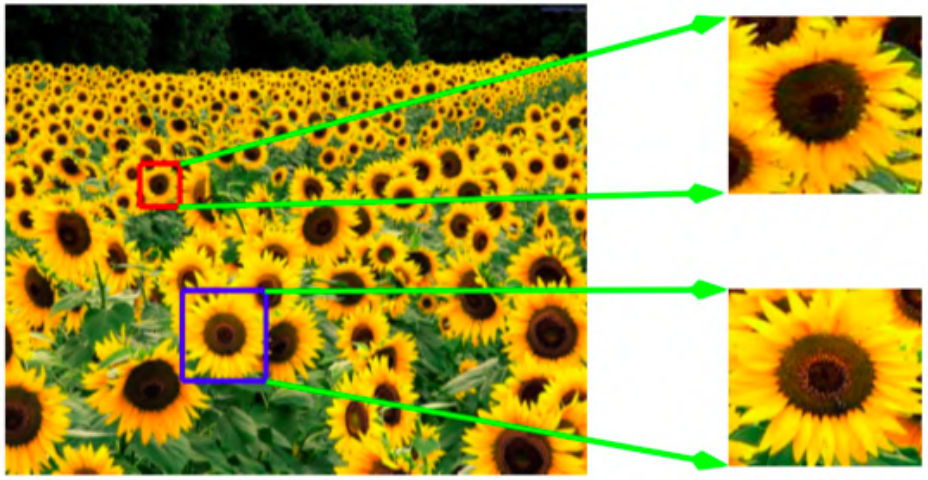
- Too many parameters for a single layer!!!

Source: Roger Grosse

Overview

Problem 2 with MLP and image analysis :

- Shift invariance: The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.
- E.g., edges, corners, contours, object parts.
- We want a neural net architecture that lets us learn a set of feature that are **shift** (**and scale**) **equivariant**.

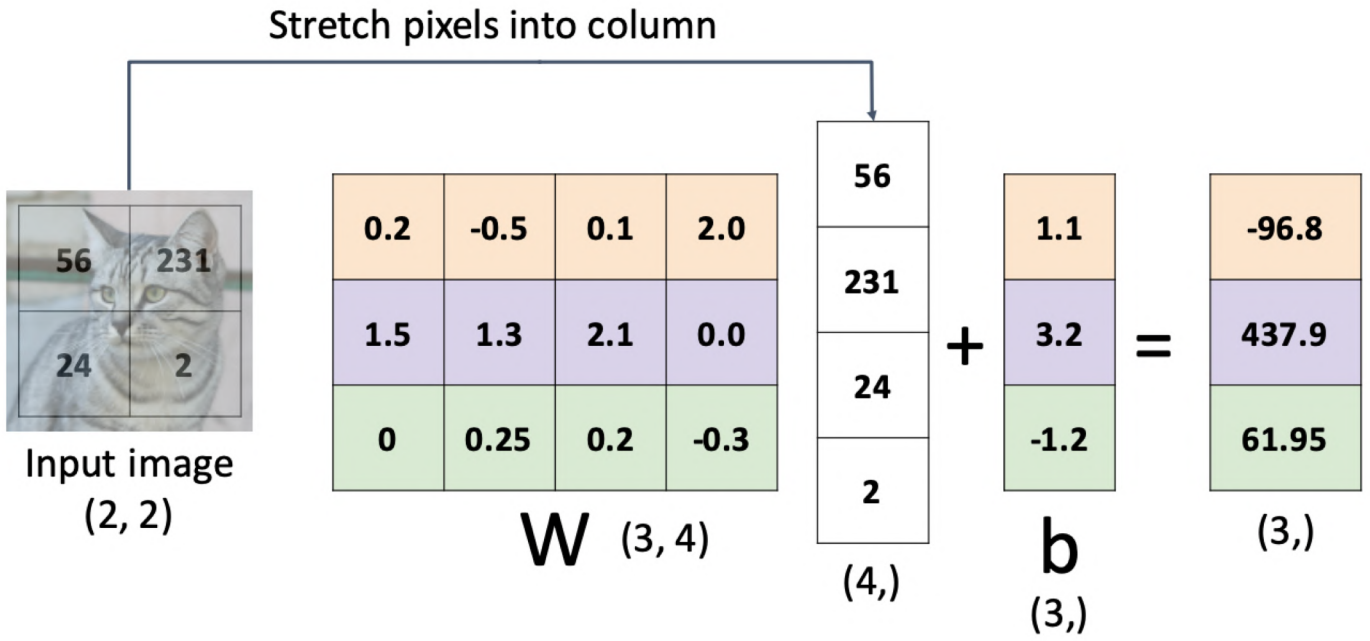


Source: Roger Grosse

Overview

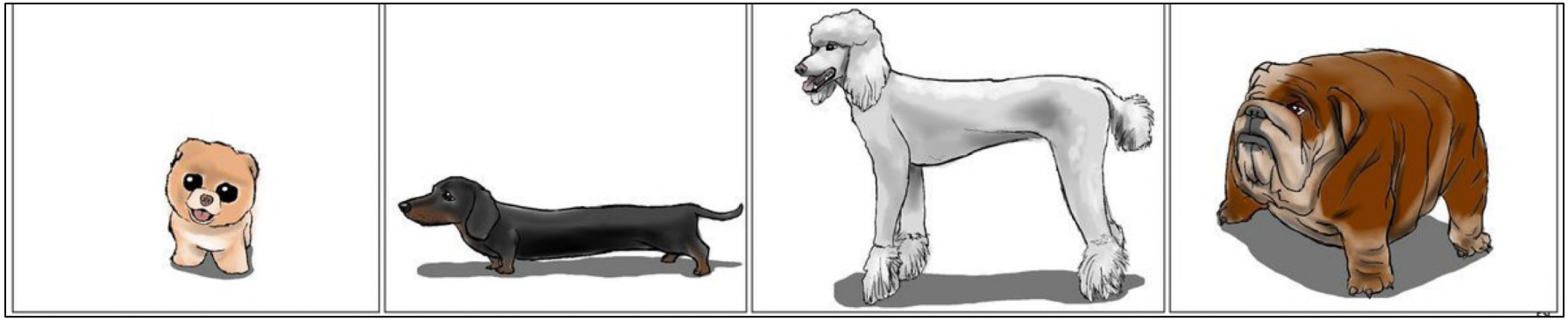
- So far, MLP does not respect the spatial structure of images:

$$f(x; W) = Wx + b$$



Source: Justin Johnson

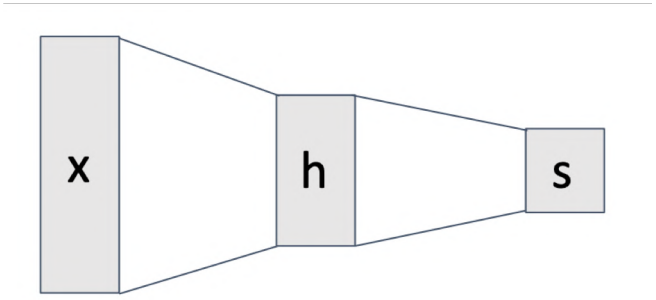
Extensions: Convolutional Networks



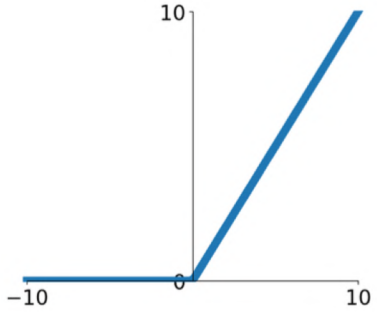
Perceptron Multiclass Multilayer Conv nets

Components of Fully Connected Network

Fully-Connected Layers



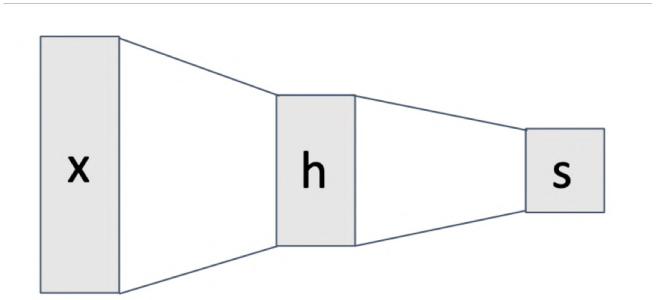
Activation Function



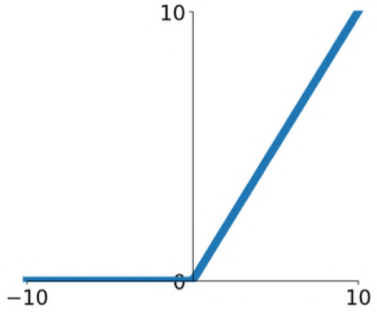
Source: Justin Johnson

Components of Convolutional Network

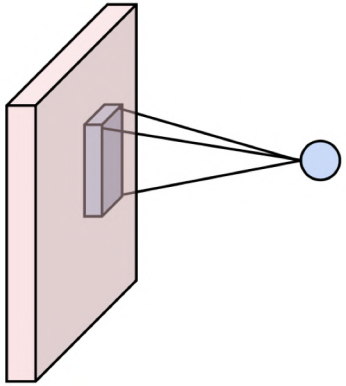
Fully-Connected Layers



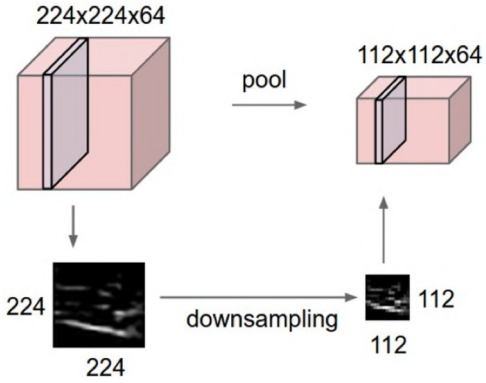
Activation Function



Convolutional Layers



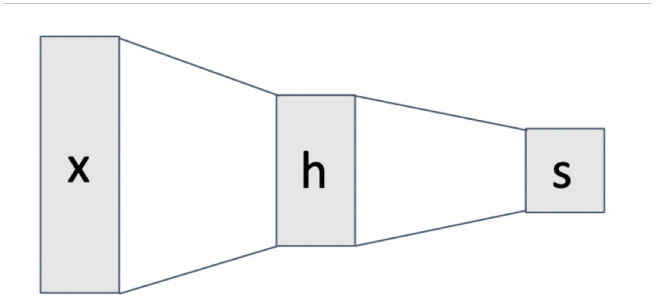
Pooling Layers



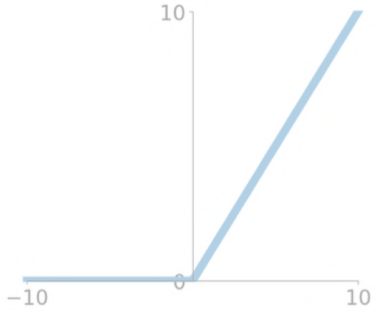
Source: Justin Johnson

Components of Convolutional Network

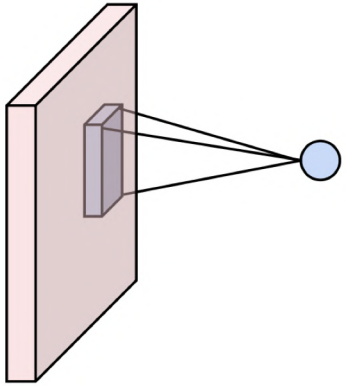
Fully-Connected Layers



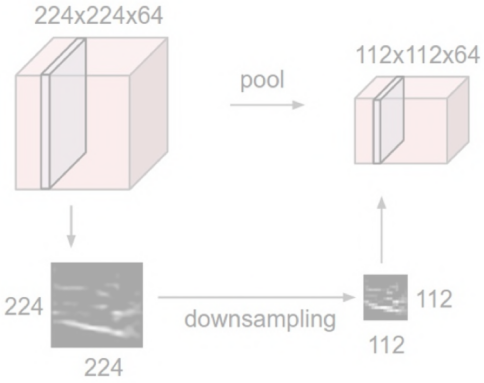
Activation Function



Convolutional Layers



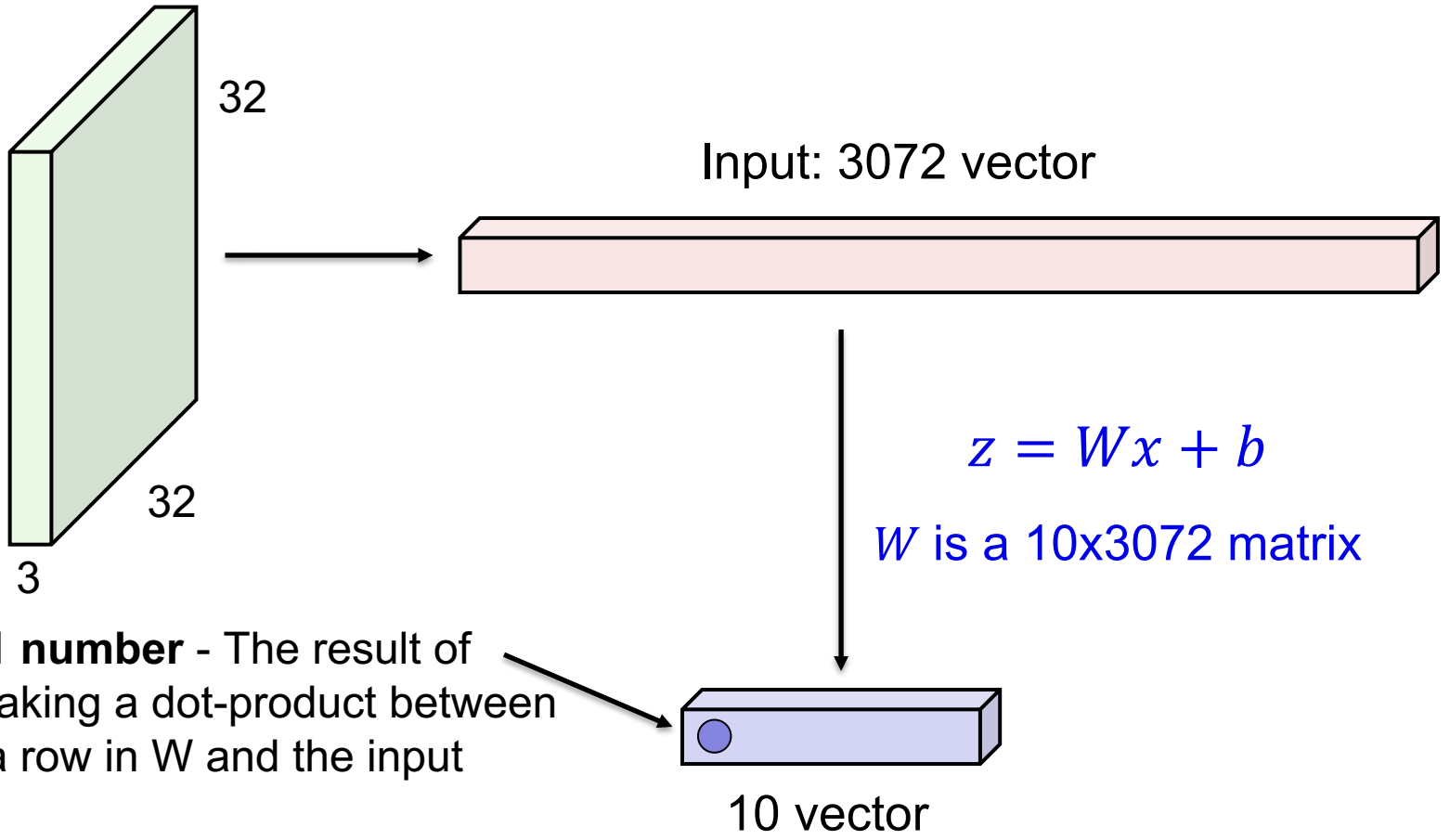
Pooling Layers



Source: Justin Johnson

Fully Connected Layer

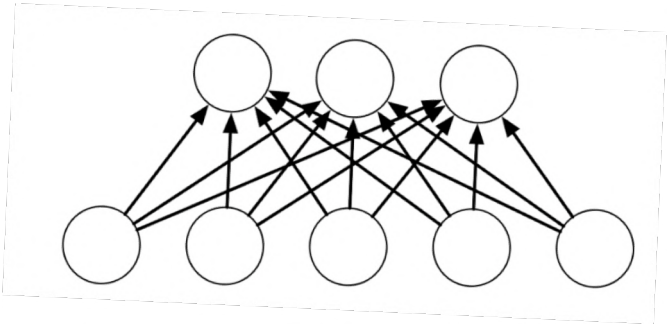
32x32x3 image $\Longrightarrow$ Flatten into 3072x1 vector

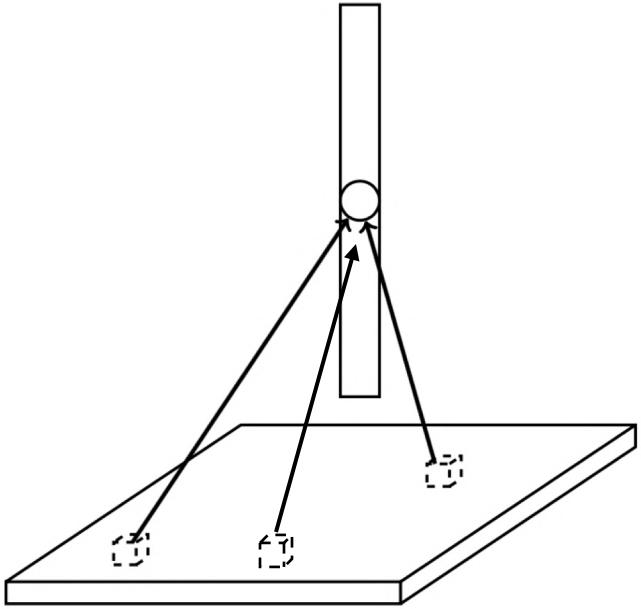


Source: Justin Johnson

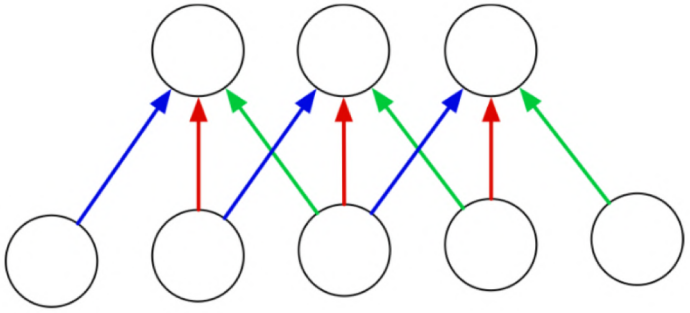
Fully Connected Layer

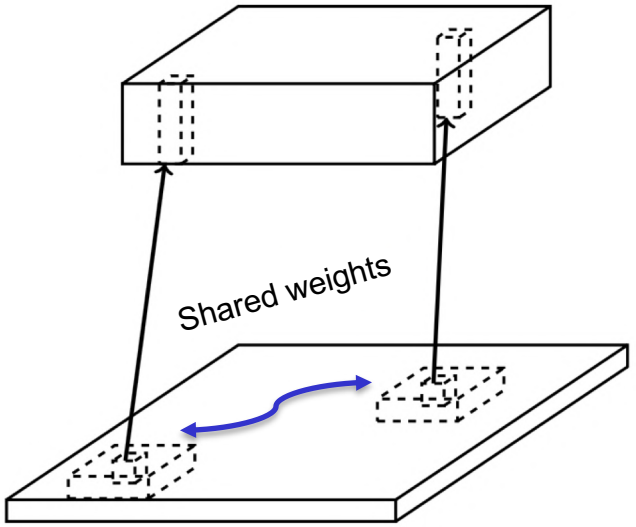
- Each hidden unit looks at the entire image.





Convolution Layers



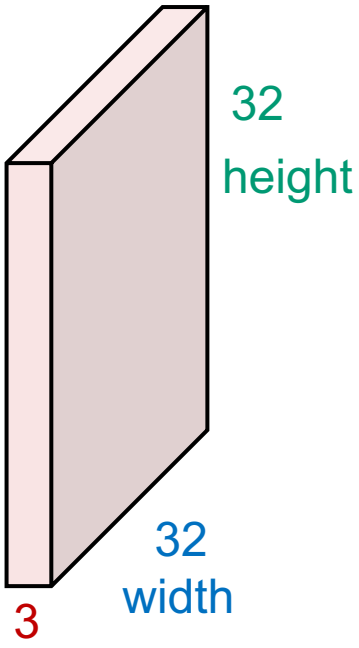


- Each column of hidden units looks at a small region of the image, and the weights are shared between all image locations.

Source: Roger Grosse

Convolution Layer

3x32x32 image

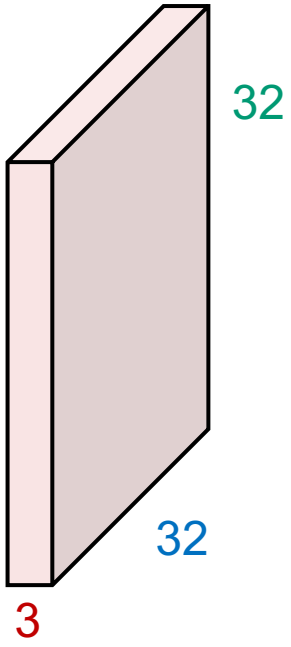


Depth/channels

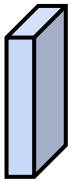
Source: Stanford CS231

Convolution Layer

3x32x32 image



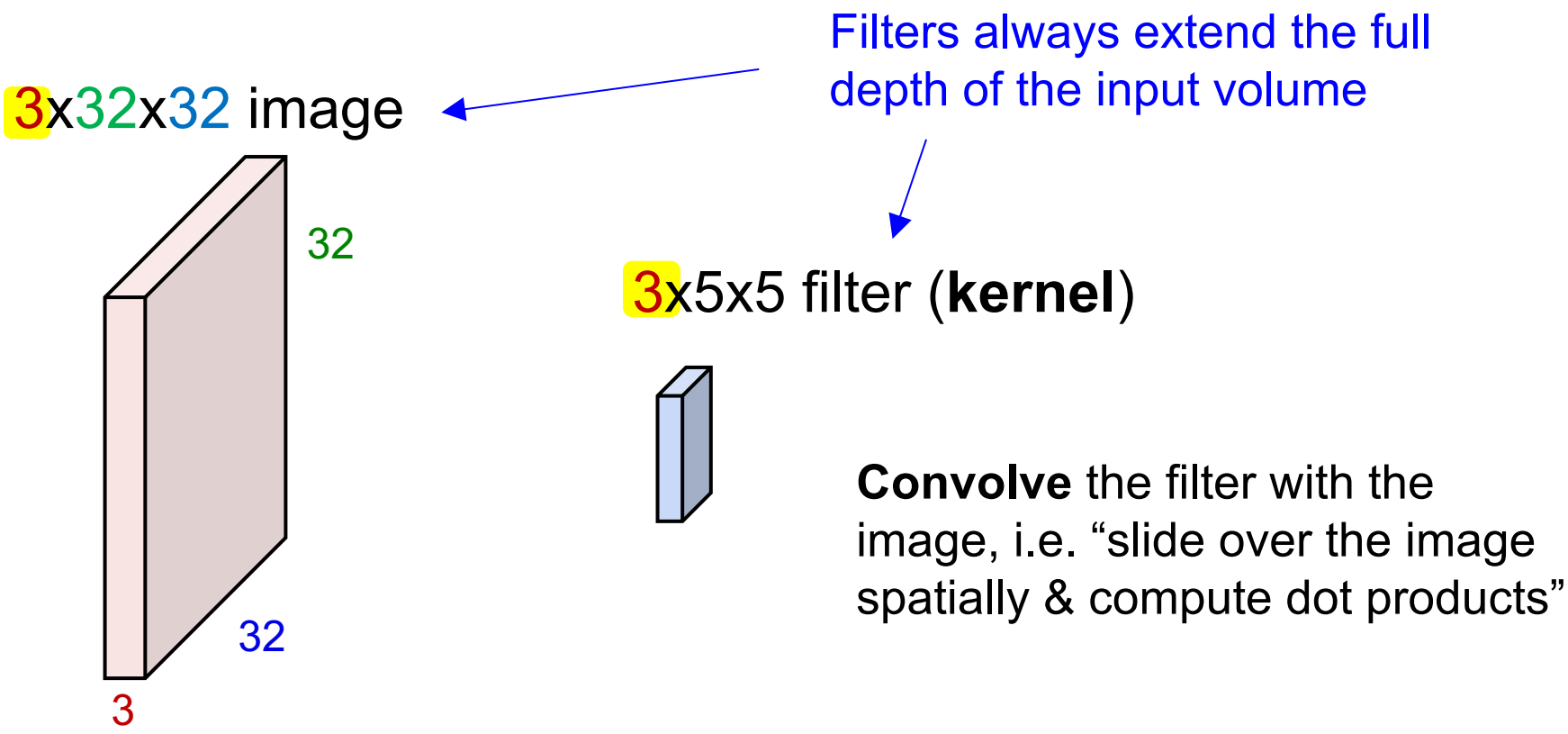
3x5x5 filter (**kernel**)



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

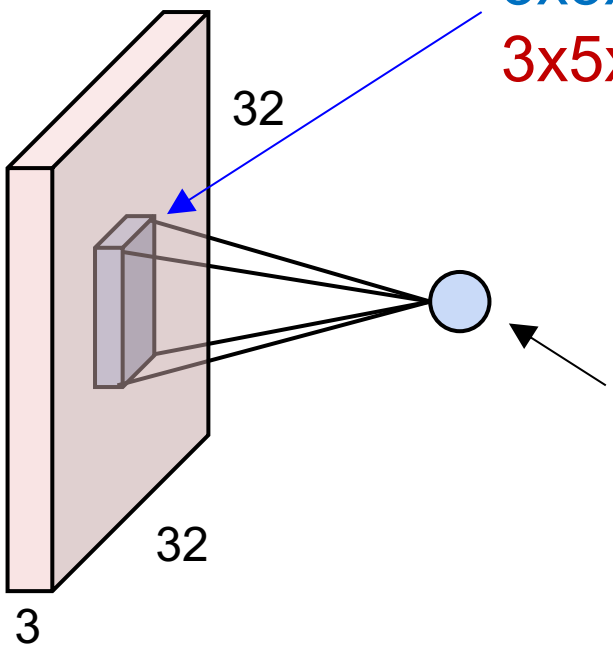
Source: Stanford CS231

Convolution Layer



Source: Stanford CS231

Convolution Layer

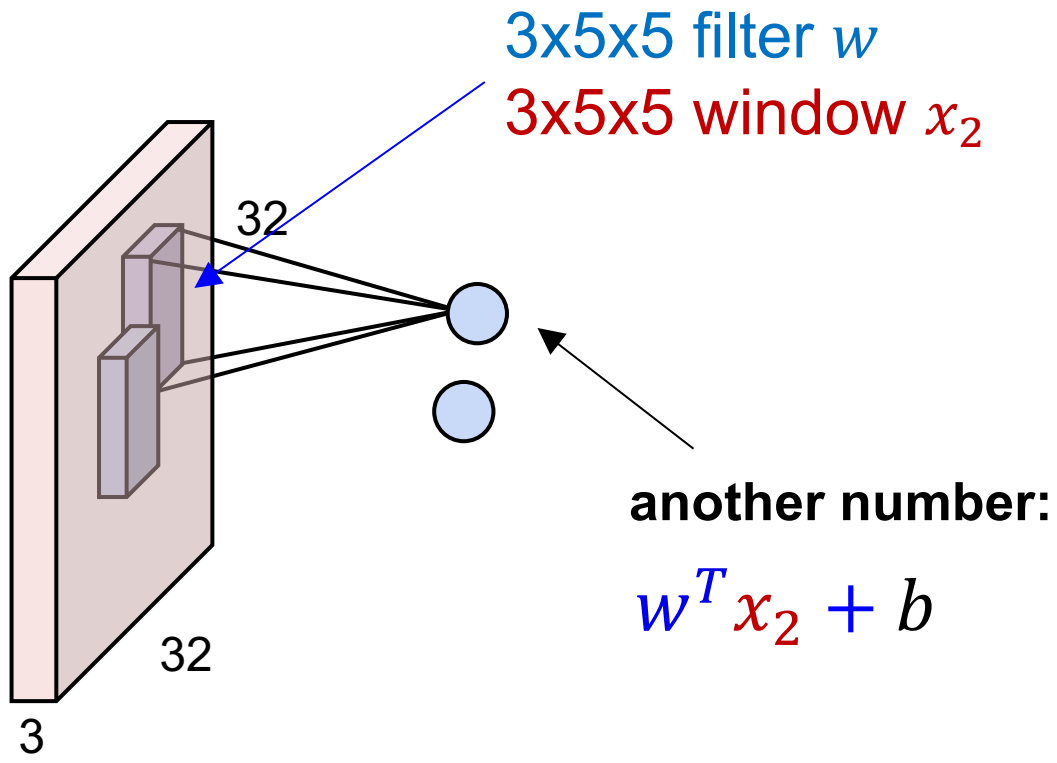


1 number:
the result of taking a dot product between the filter w and a small 3x5x5 window of the image x (i.e. 3*5*5 = 75-dimensional dot product + bias)

$$w^T x_1 + b$$

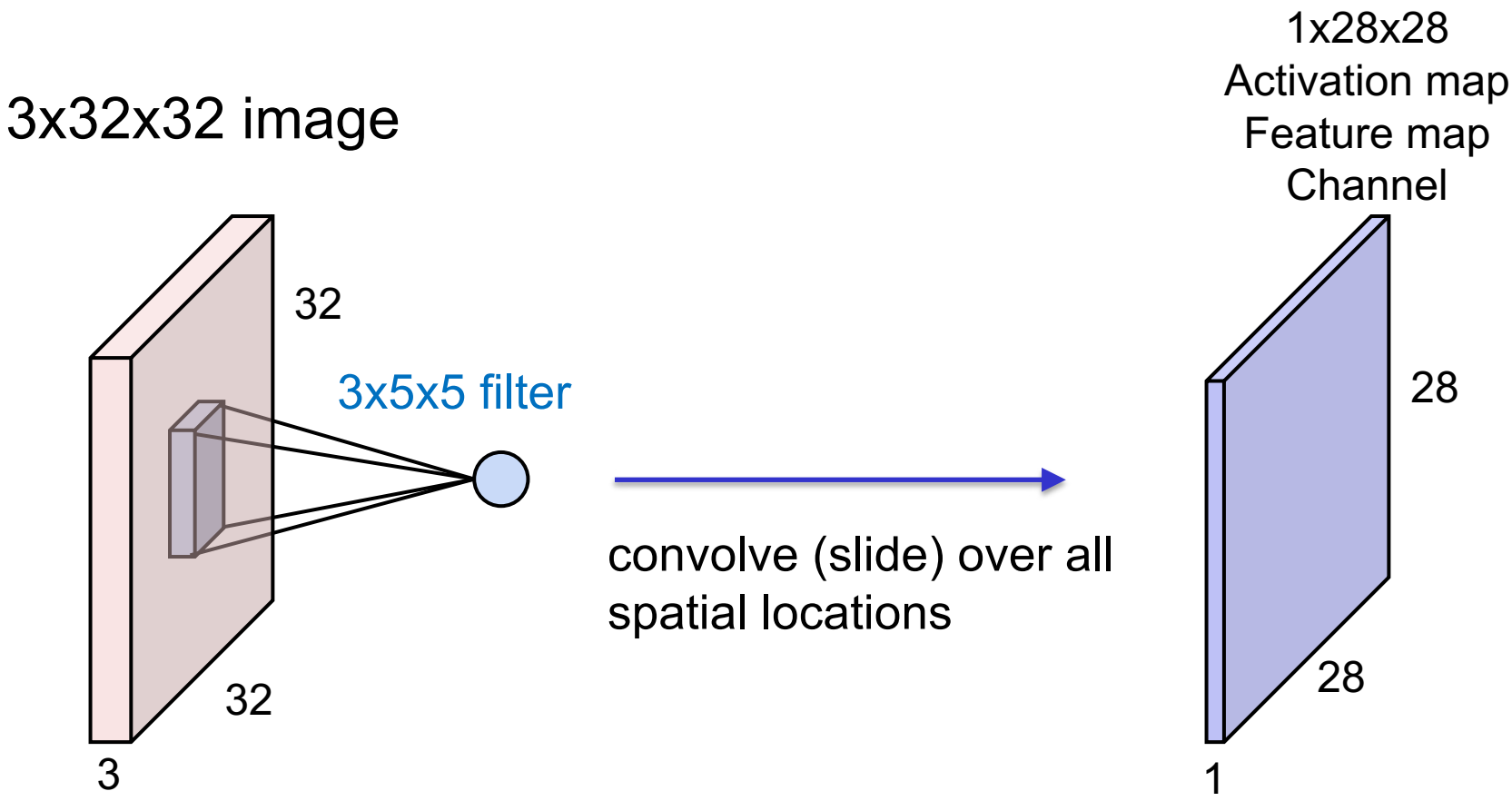
Source: Stanford CS231

Convolution Layer



Source: Stanford CS231

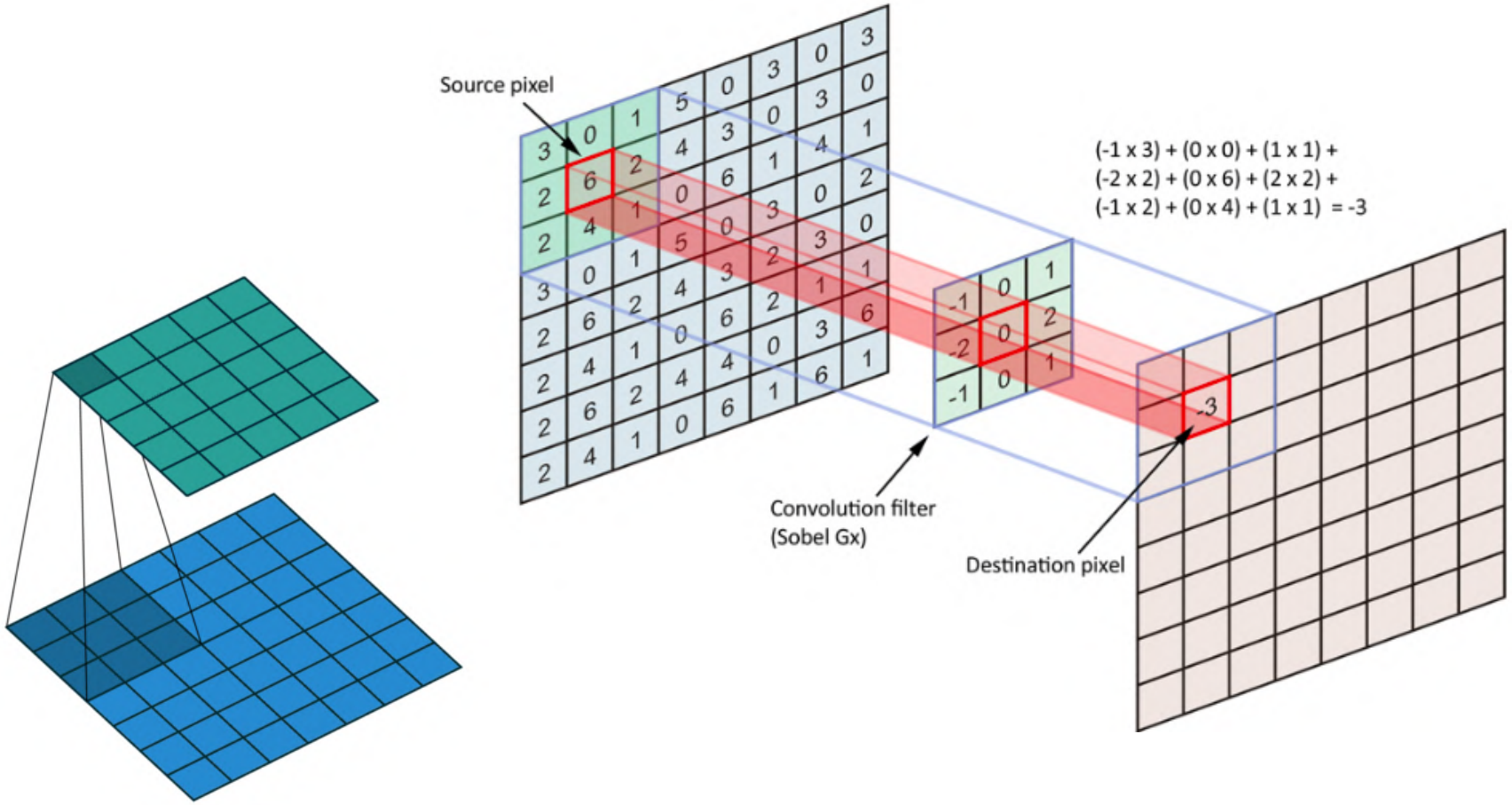
Convolution Layer



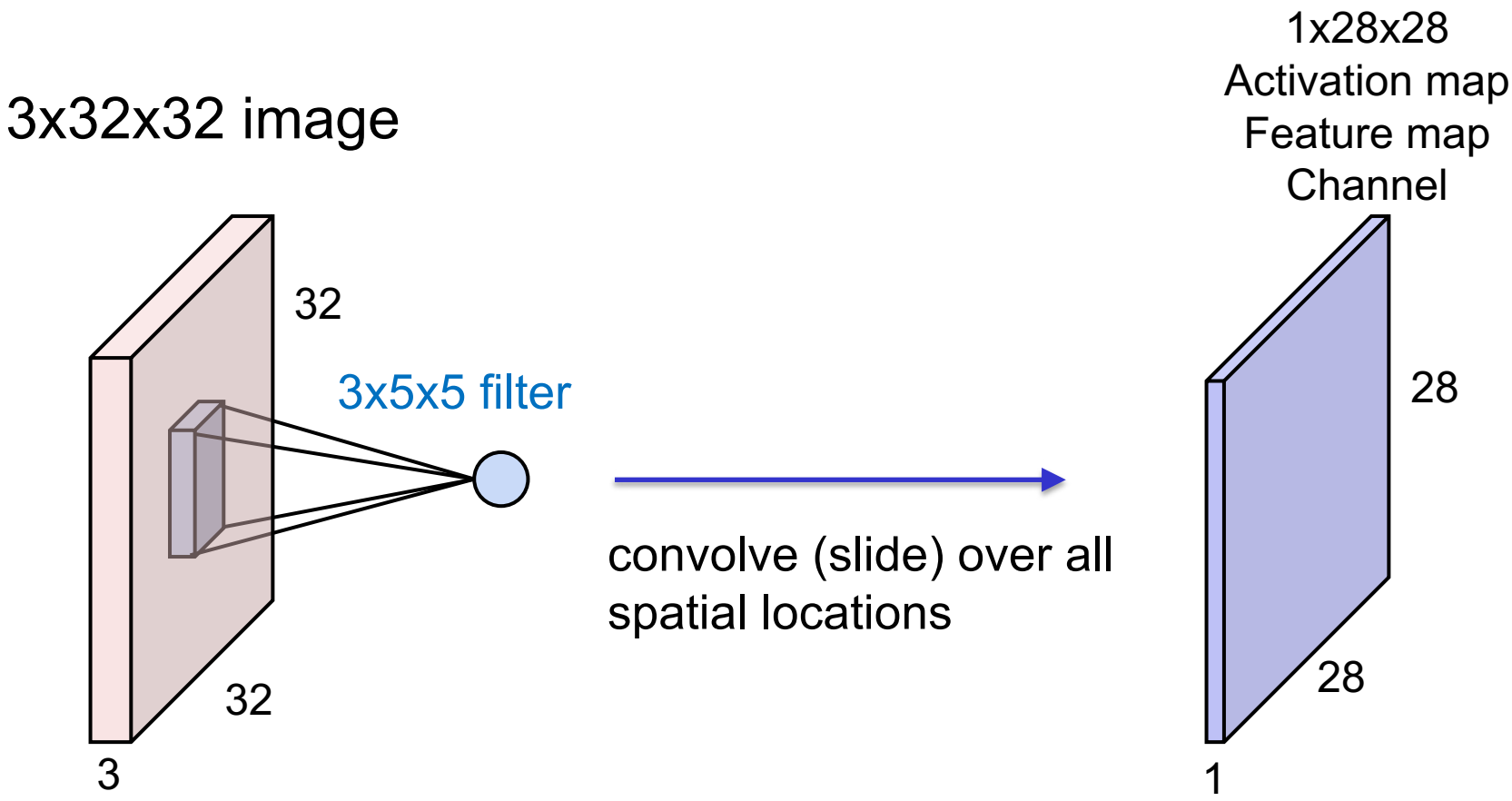
Dims of the activation map = N-F+1

Source: Stanford CS231

Convolution Layer



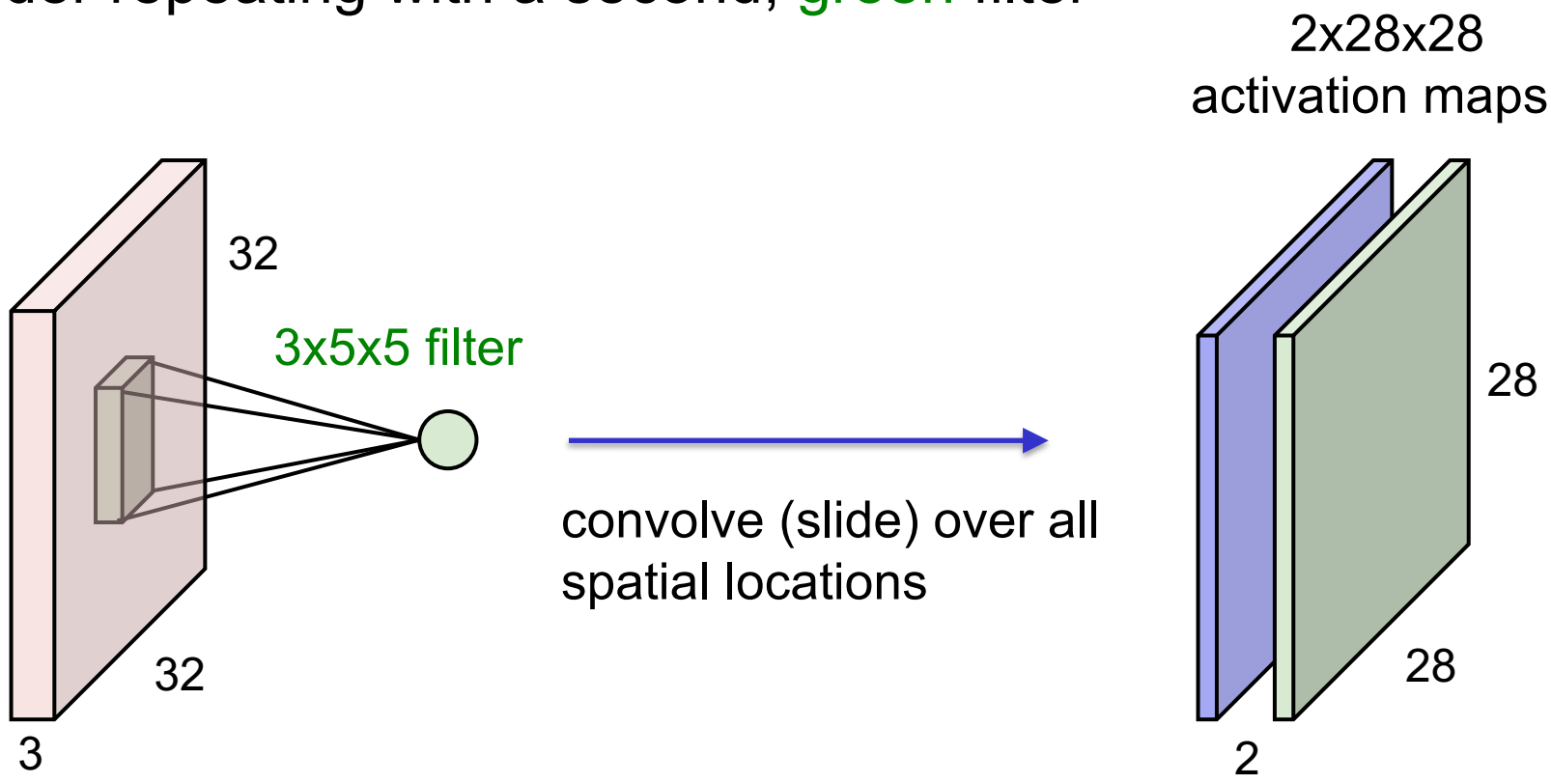
Convolution Layer



Source: Stanford CS231

Convolution Layer

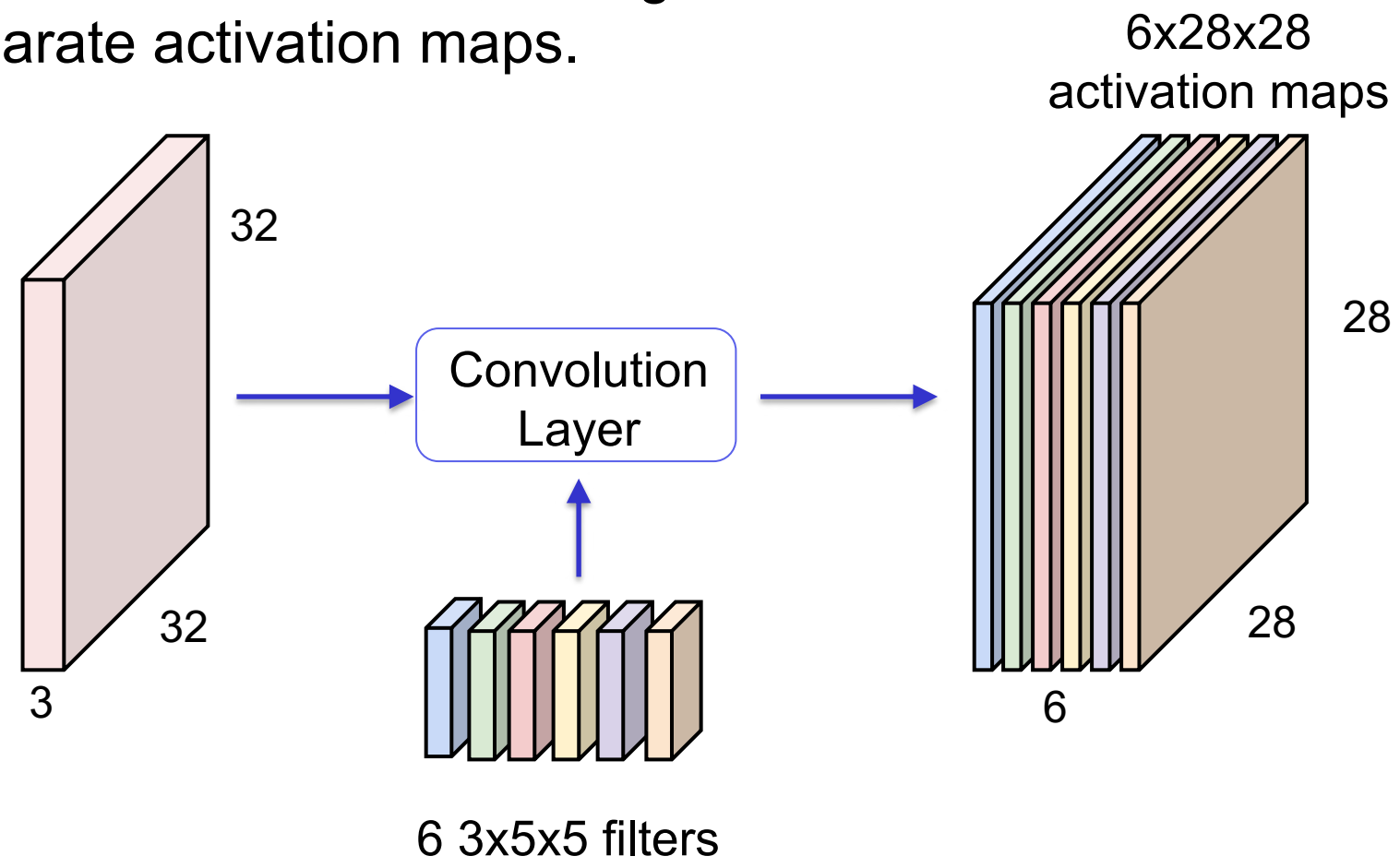
Consider repeating with a second, green filter



Source: Stanford CS231

Convolution Layer

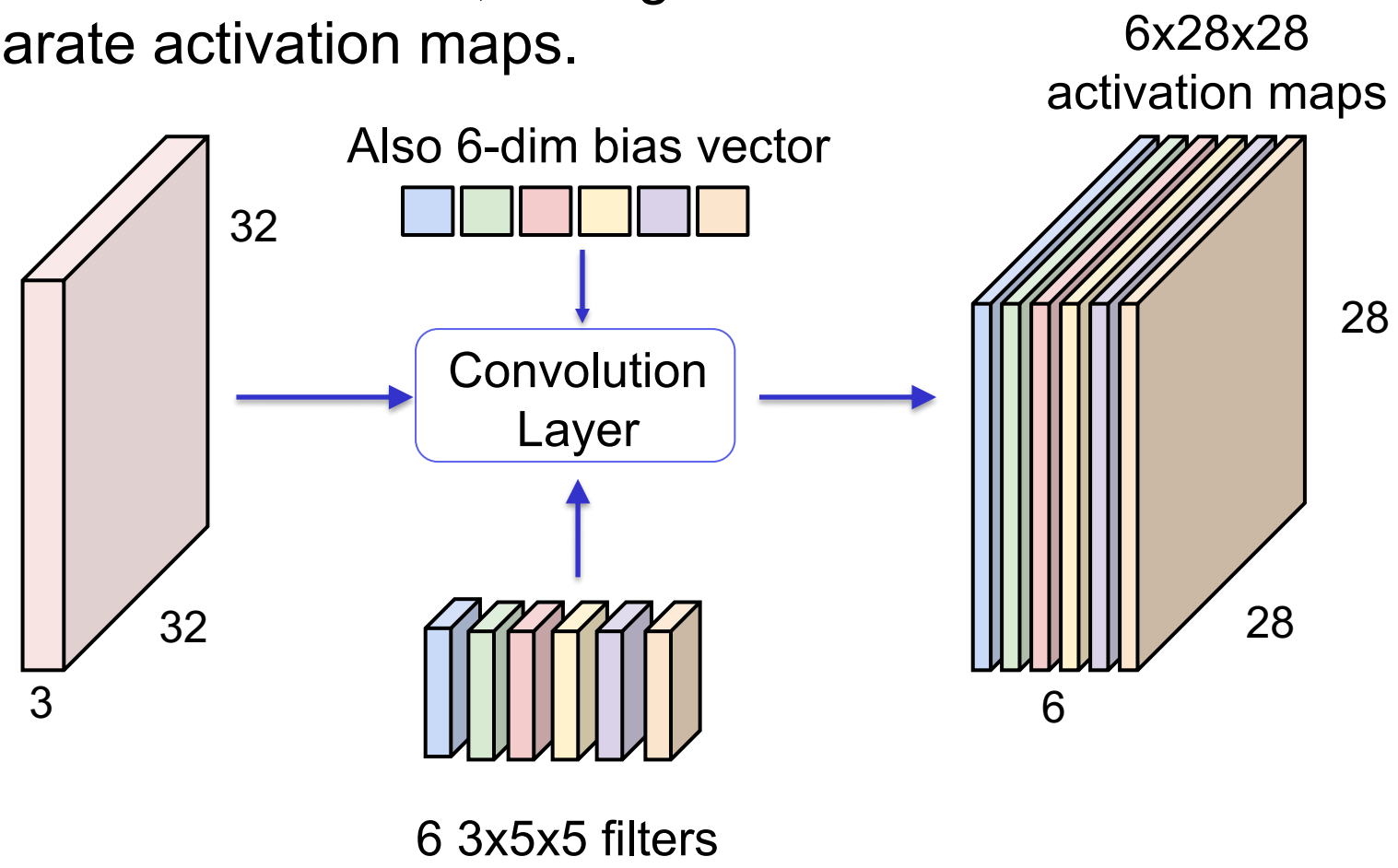
Consider 6 3x5x5 filters, we'll get 6 separate activation maps.



Source: Stanford CS231

Convolution Layer

Consider 6 3x5x5 filters, we'll get 6 separate activation maps.

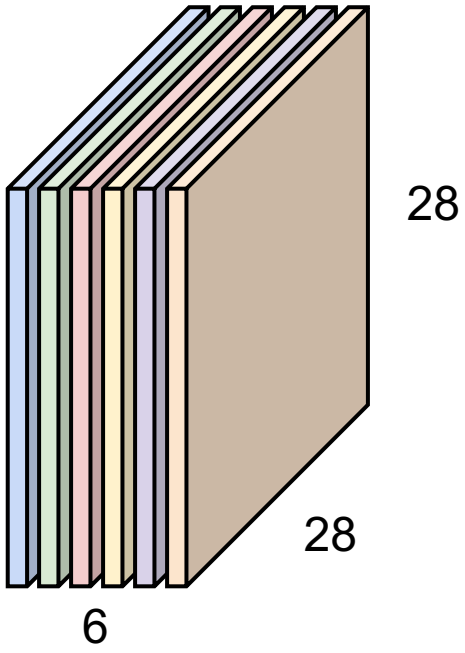


Source: Stanford CS231

Convolution Layer

Consider 6 3x5x5 filters, we'll get 6 separate activation maps.

Can be seen as 6 activation maps, each is a 28x28 matrix.

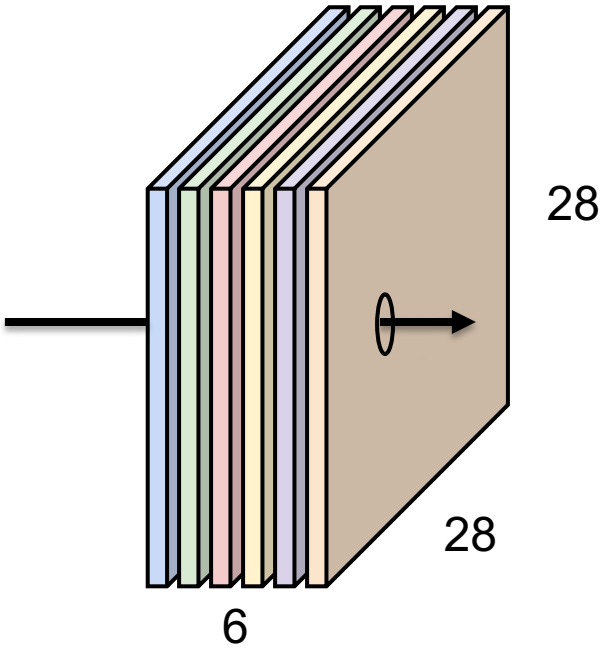


Source: Stanford CS231

Convolution Layer

Consider 6 3x5x5 filters, we'll get 6 separate activation maps.

Can be seen as 6x28x28 tensor,
at each point a 6-dim vector.



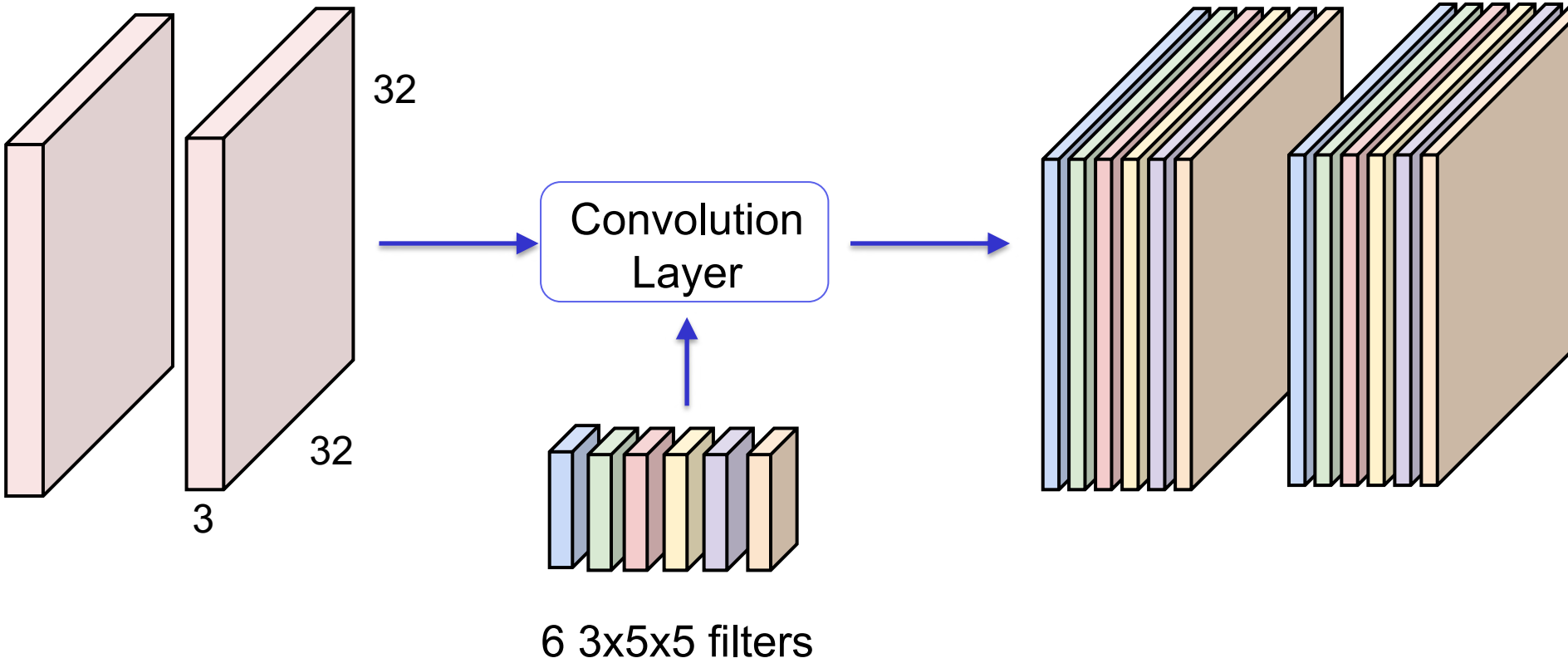
Source: Stanford CS231

Convolution Layer

Convolution with a **batch of images**

2x3x32x32 image

2x6x28x28
batch of outputs



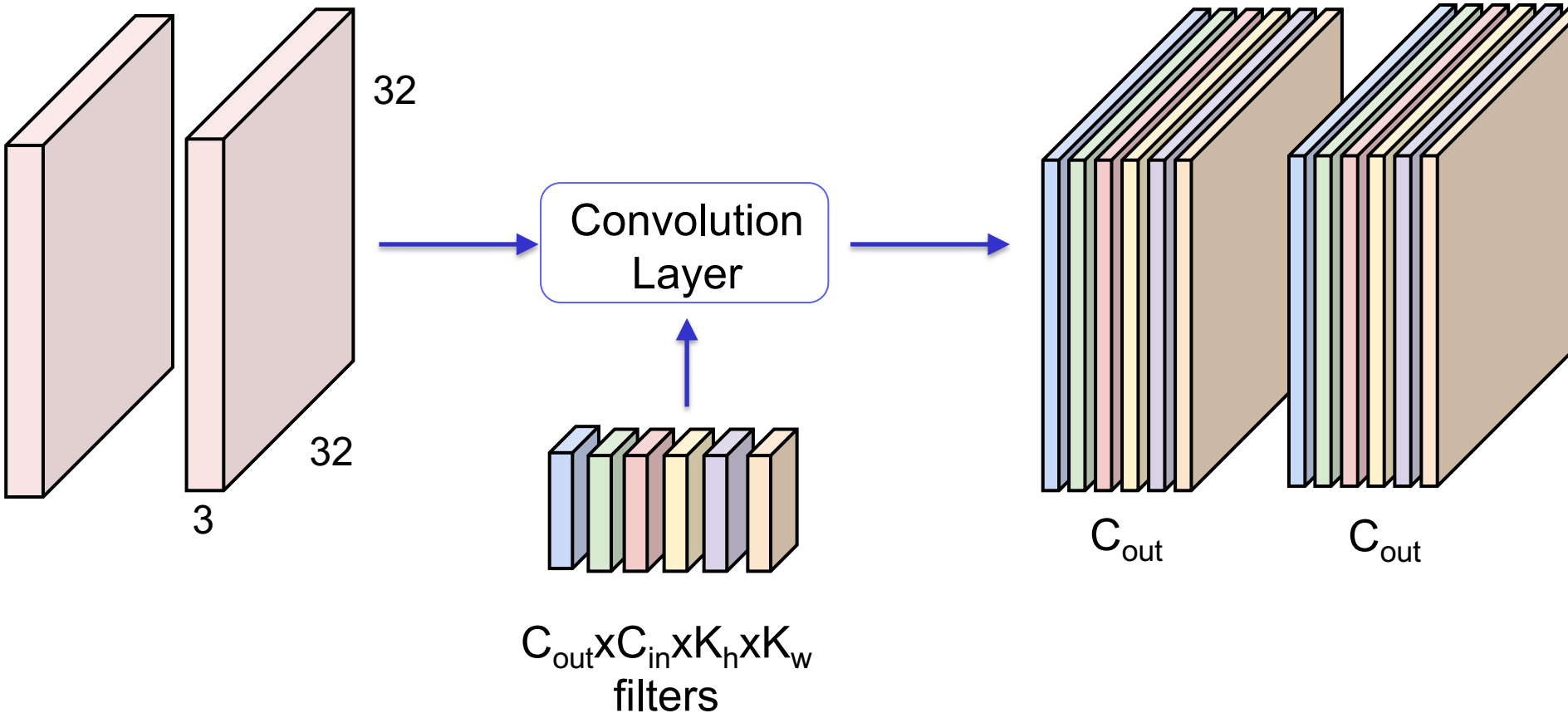
Source: Justin Johnson

Convolution Layer

Convolution with a **batch of images**

$N \times C_{in} \times H \times W$ image

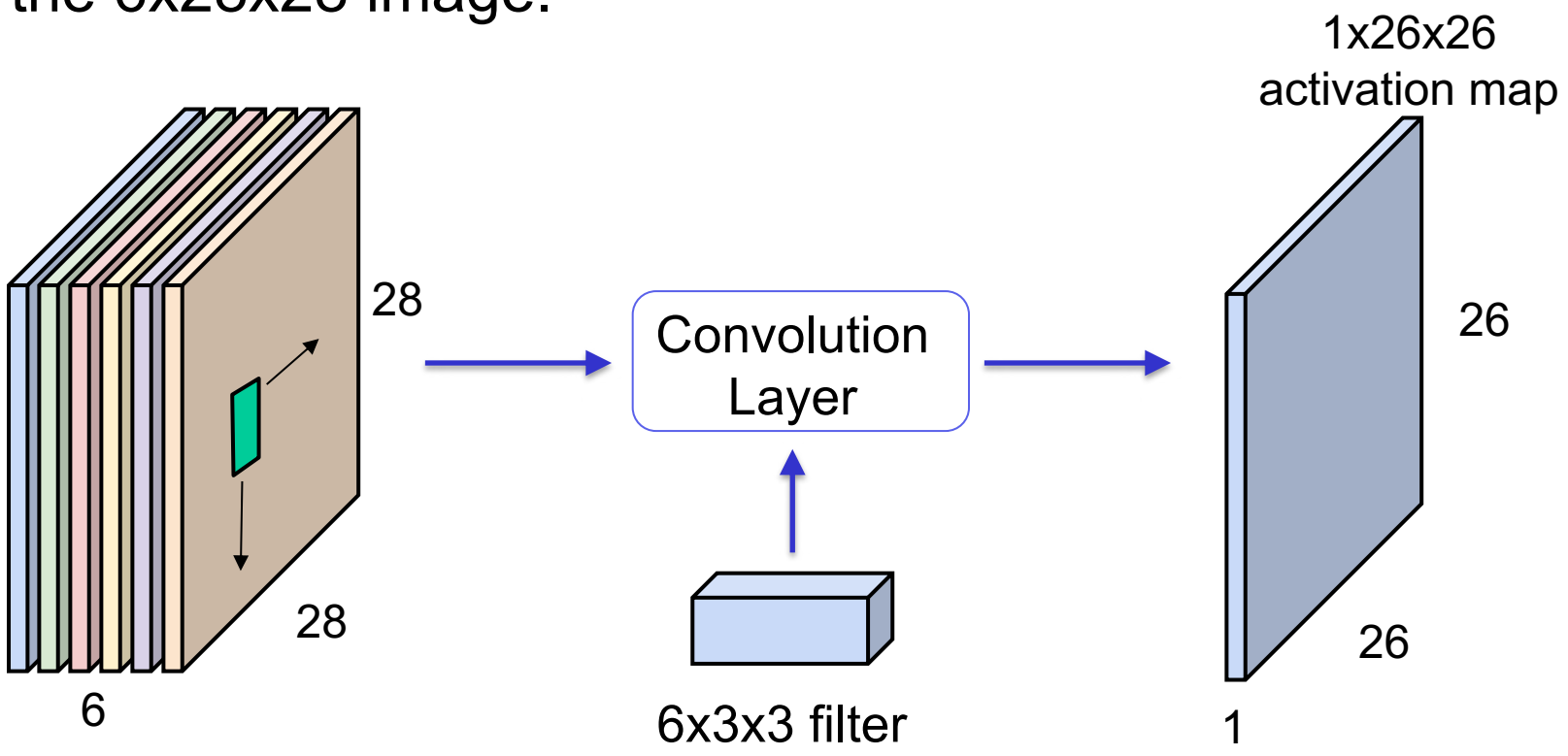
$N \times C_{out} \times H' \times W'$ batch of outputs



Source: Justin Johnson

Stacking Convolution

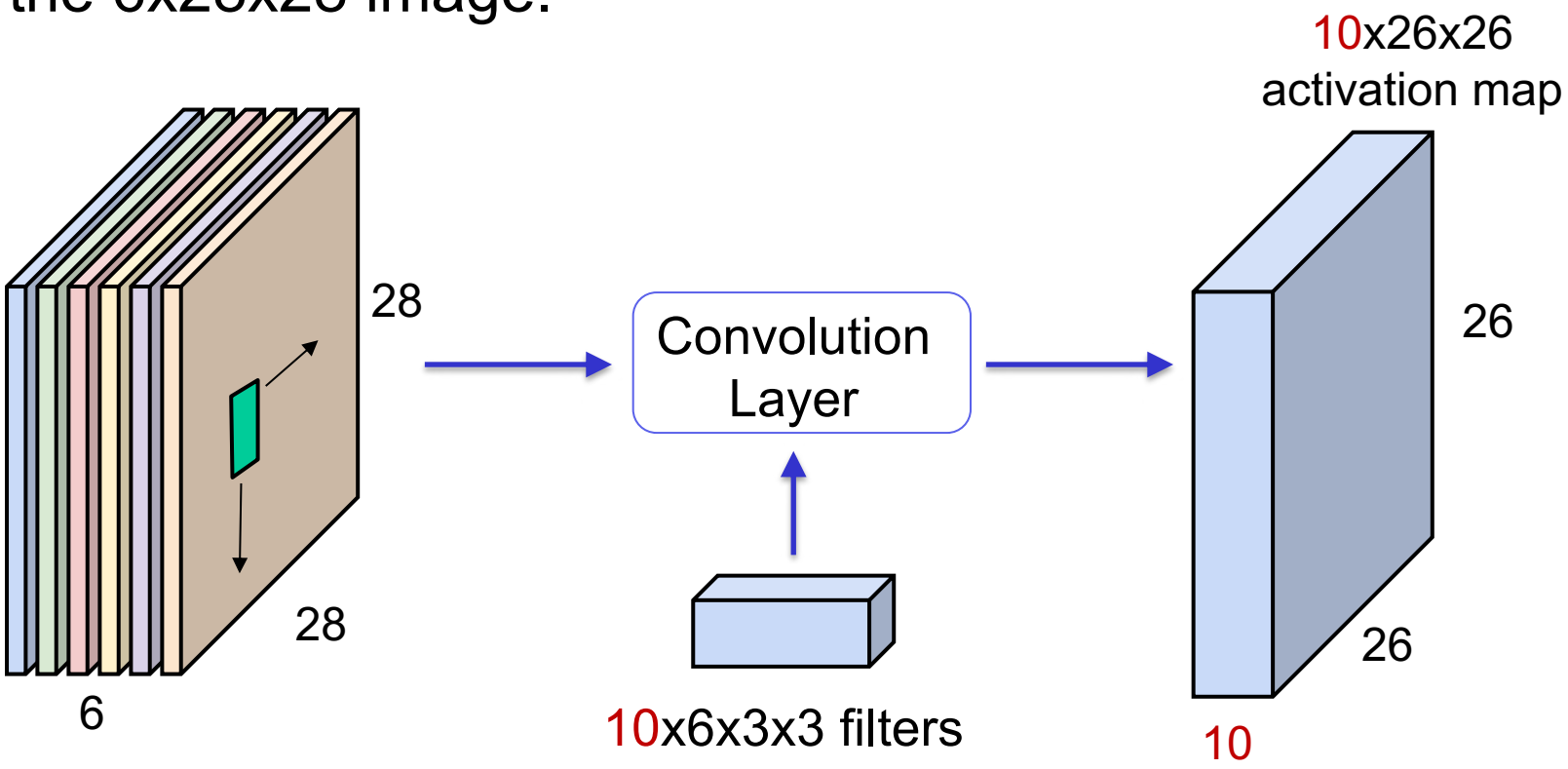
In the next convolution layer we can apply a 6x3x3 filter to the 6x28x28 image.



Source: Justin Johnson

Stacking Convolution

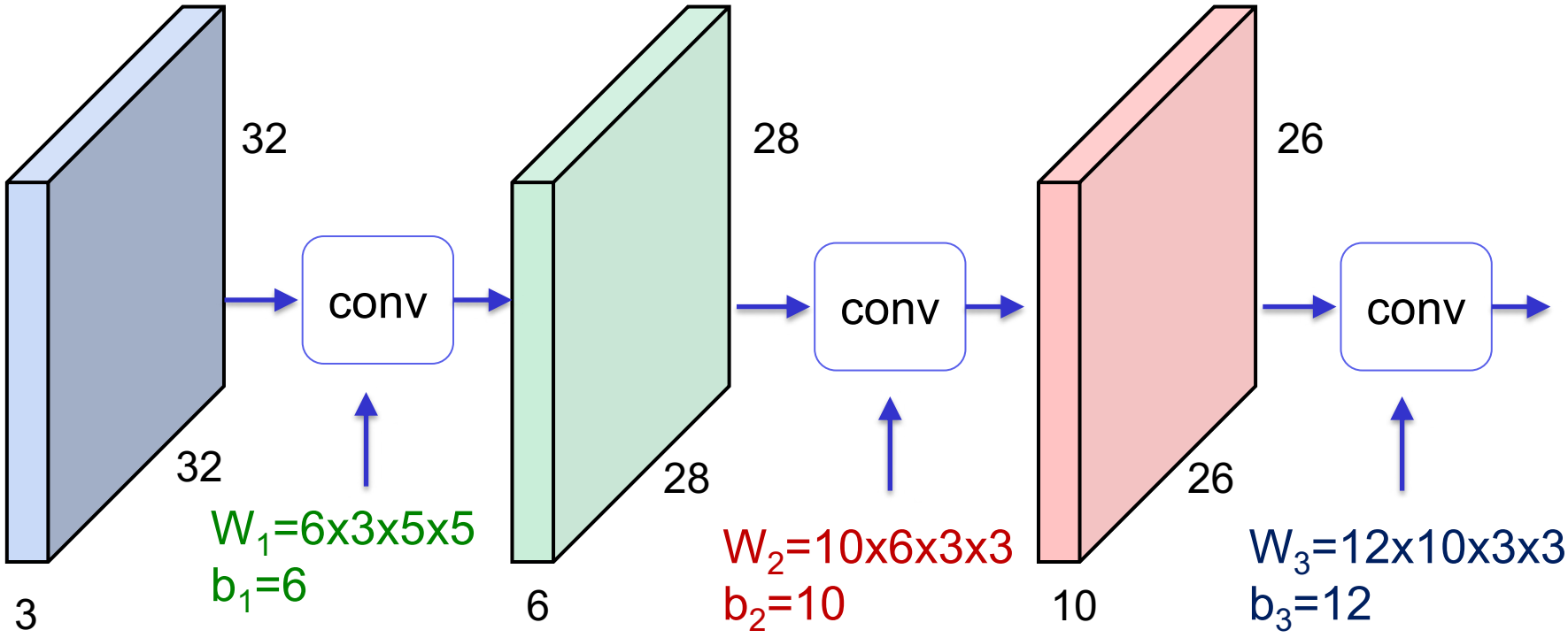
In the next convolution layer we can apply 10x6x3x3 filters to the 6x28x28 image.



Source: Justin Johnson

Stacking Convolution

Stacking 3 conv. layers



Input:
Nx3x32x32

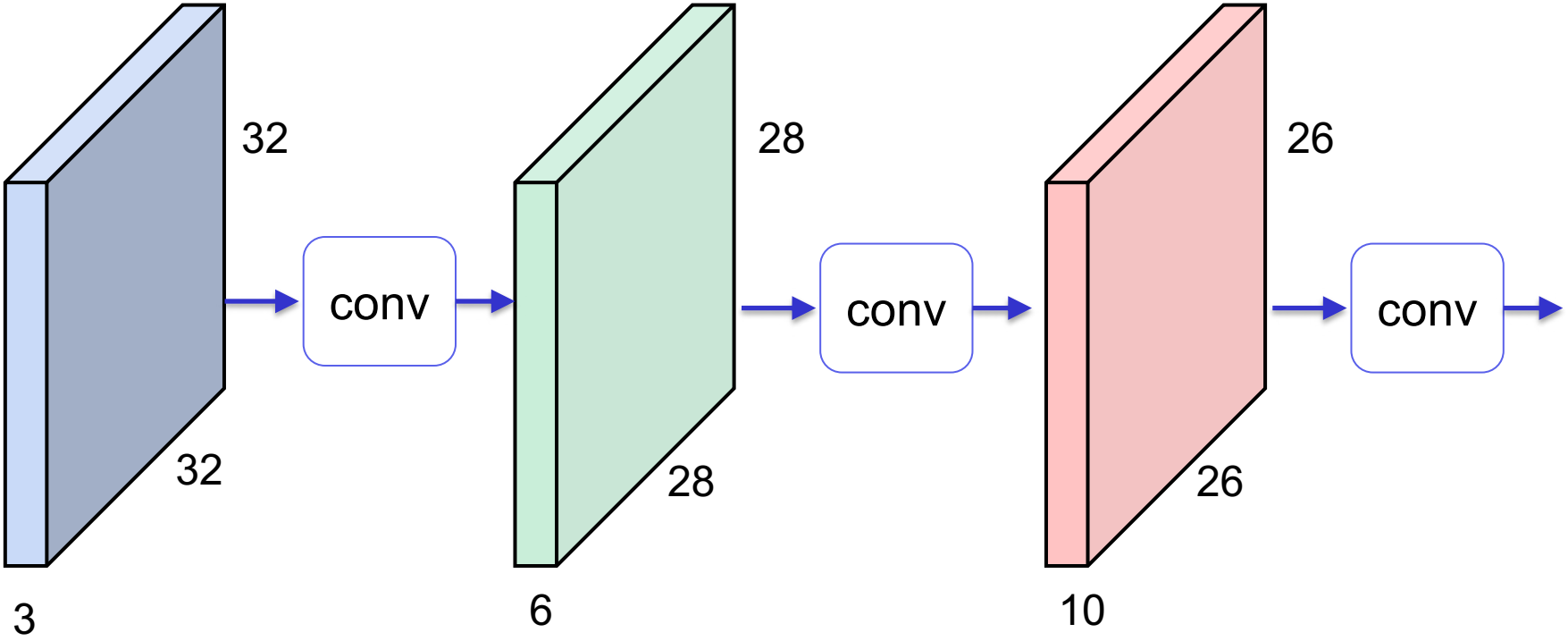
1st hidden layer:
Nx6x28x28

2nd hidden layer:
Nx10x26x26

Source: Justin Johnson

Stacking Convolution

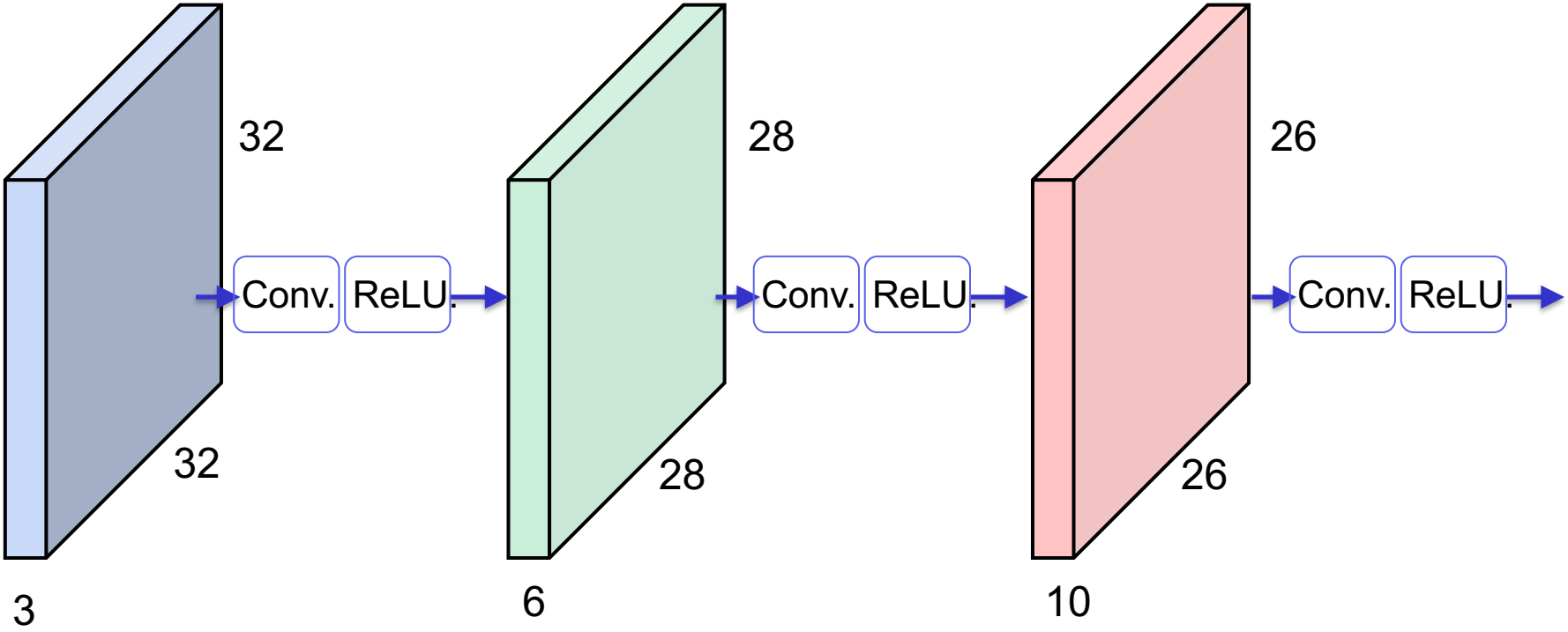
- **Question**: What happens if we stack few convolution layers?
- **Answer**: We get another convolution! (Note, conv. is a linear operation)



Source: Justin Johnson

Stacking Convolution

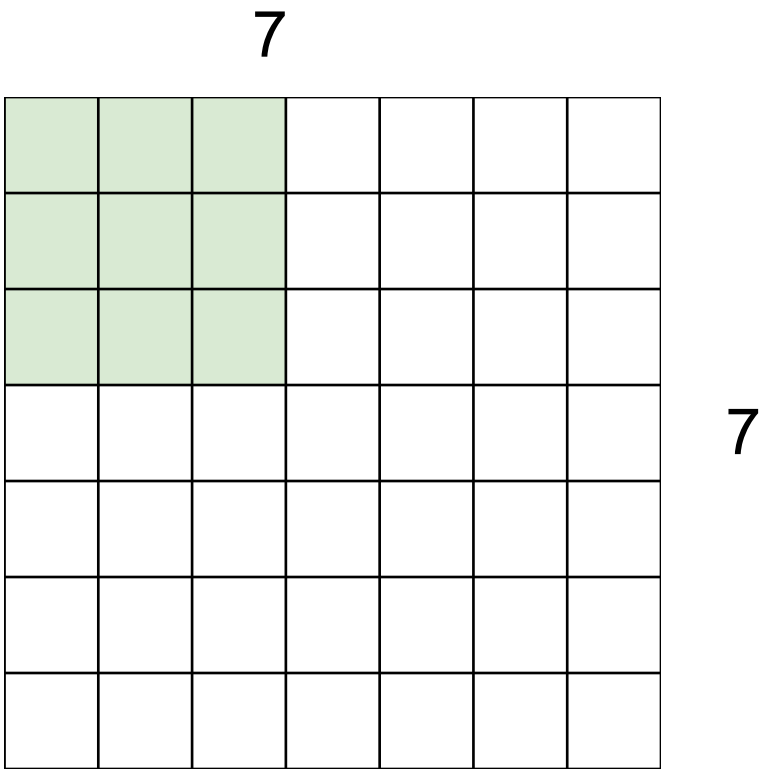
- **Solution**: Add a non-linear activation functions between conv. layers.



Source: Justin Johnson

A closer look at spatial dimensions

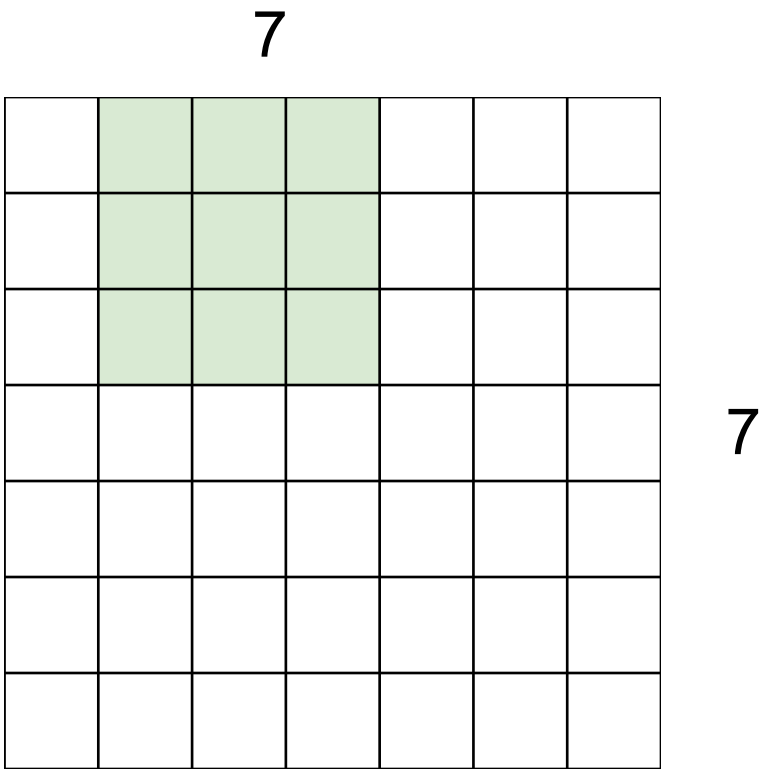
7x7 input (spatially)
assume 3x3 filter



Source: Stanford CS231

A closer look at spatial dimensions

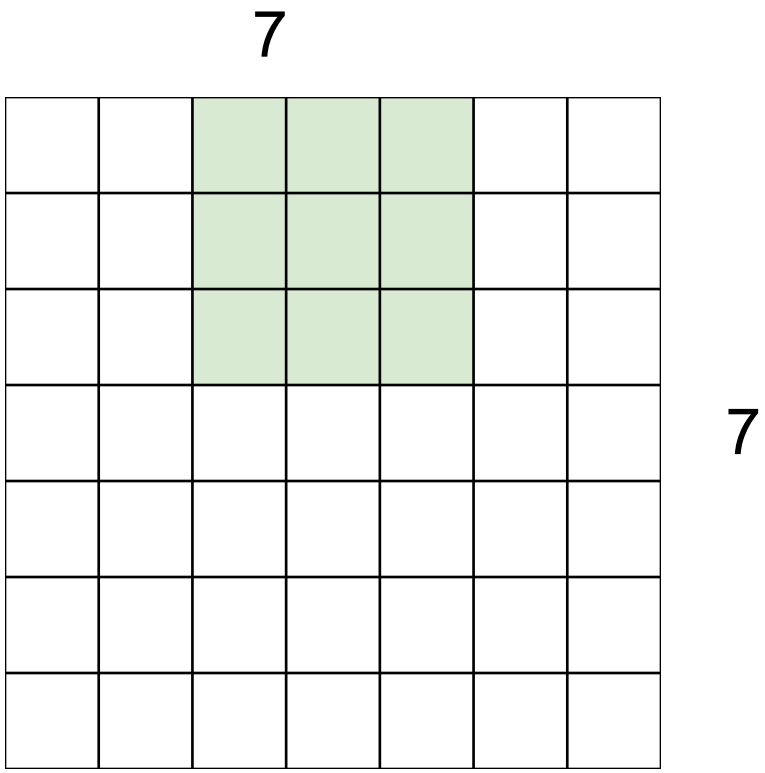
7x7 input (spatially)
assume 3x3 filter



Source: Stanford CS231

A closer look at spatial dimensions

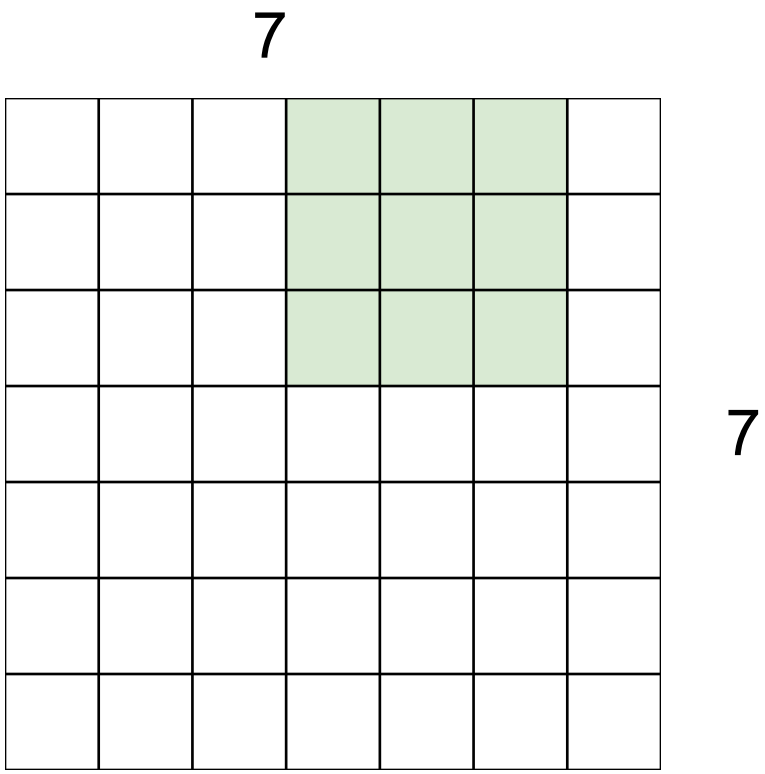
7x7 input (spatially)
assume 3x3 filter



Source: Stanford CS231

A closer look at spatial dimensions

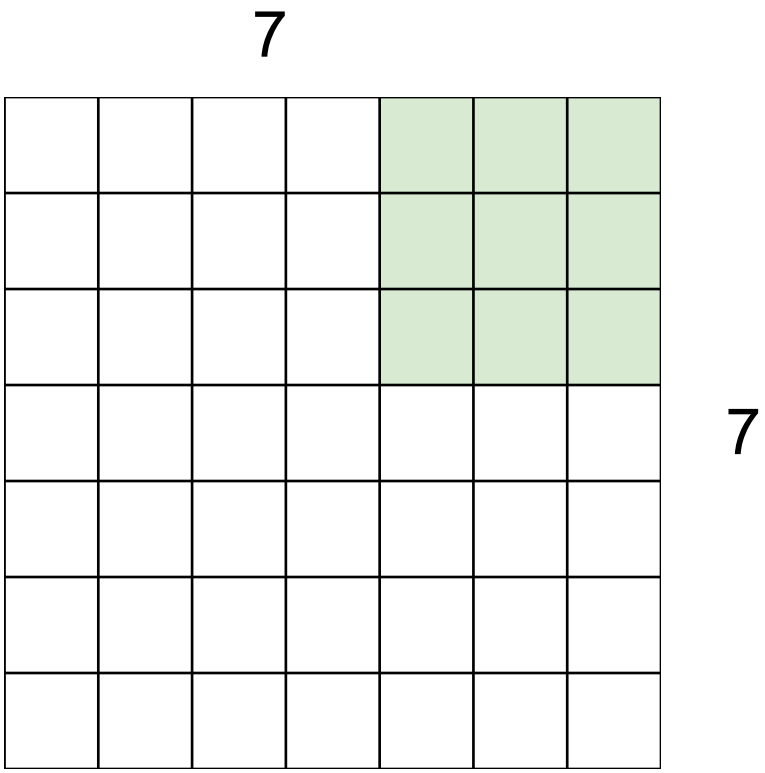
7x7 input (spatially)
assume 3x3 filter



Source: Stanford CS231

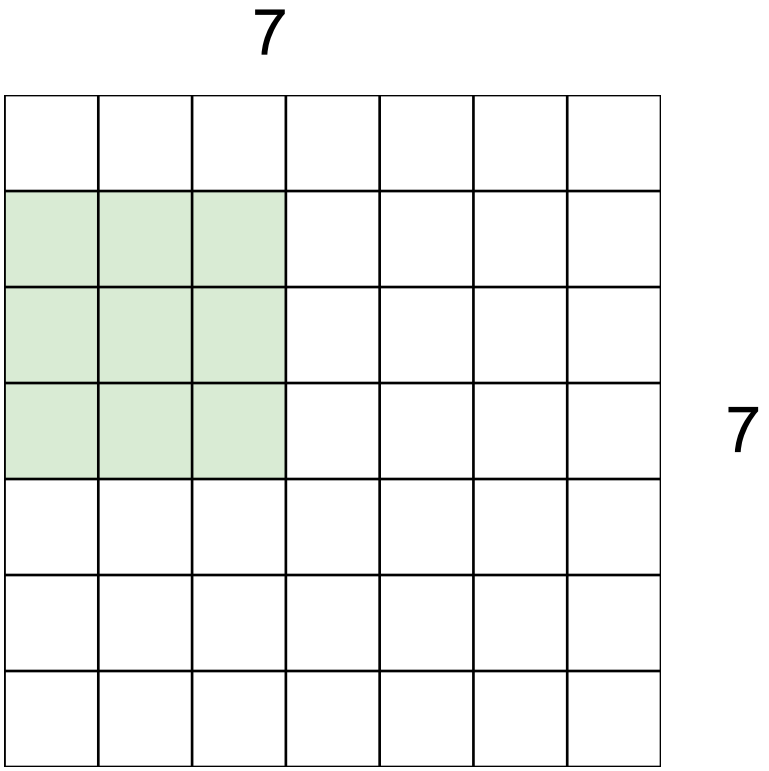
A closer look at spatial dimensions

7x7 input (spatially)
assume 3x3 filter



Source: Stanford CS231

A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

$\Rightarrow$ 5 x 5 output $(7 - 3 + 1)$

In general:
Input: N
Filter: F
Output: N-F+1

Problem: Feature maps "shrinks" with each layer.

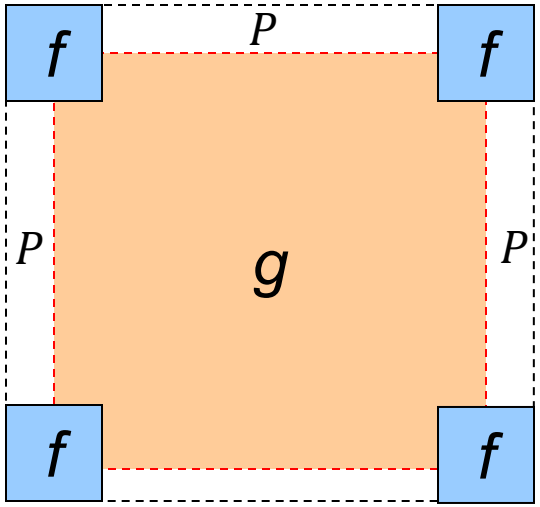
Solution: Zero Padding!

Source: Stanford CS231

Padding

- Assume Image size $N \times N$, filter size $F \times F$, and padding P
- Resulting convolution for **valid**:

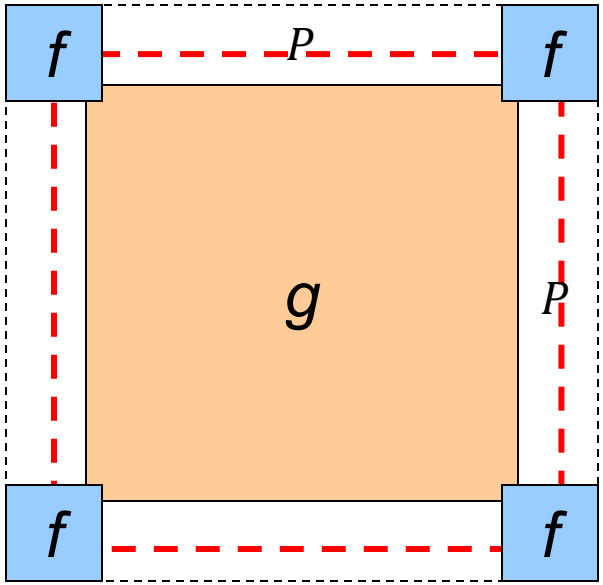
$$(N + 2P - F + 1) \times (N + 2P - F + 1)$$



Convolution Operation: Spatial Dimensions

- Three common methods for padding: **full**, **same**, **valid**.
- Recall $\mathrm{R} = (N + 2P - F + 1)$

full

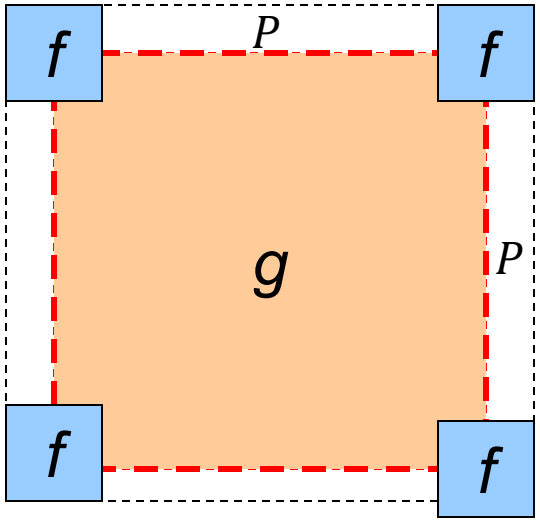


$(N + F - 1) \times (N + F - 1)$

$P = F - 1$

Full padding

same

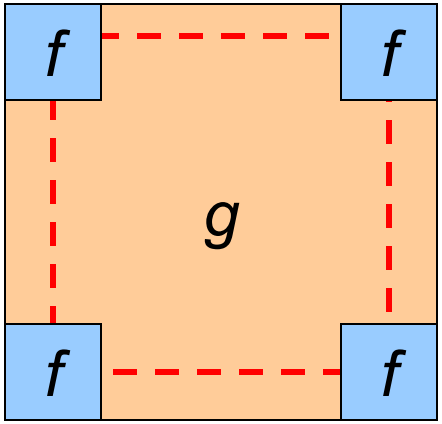


$N \times N$

$P = (F - 1)/2$

Half padding

valid



$(N - F + 1) \times (N - F + 1)$

$P = 0$

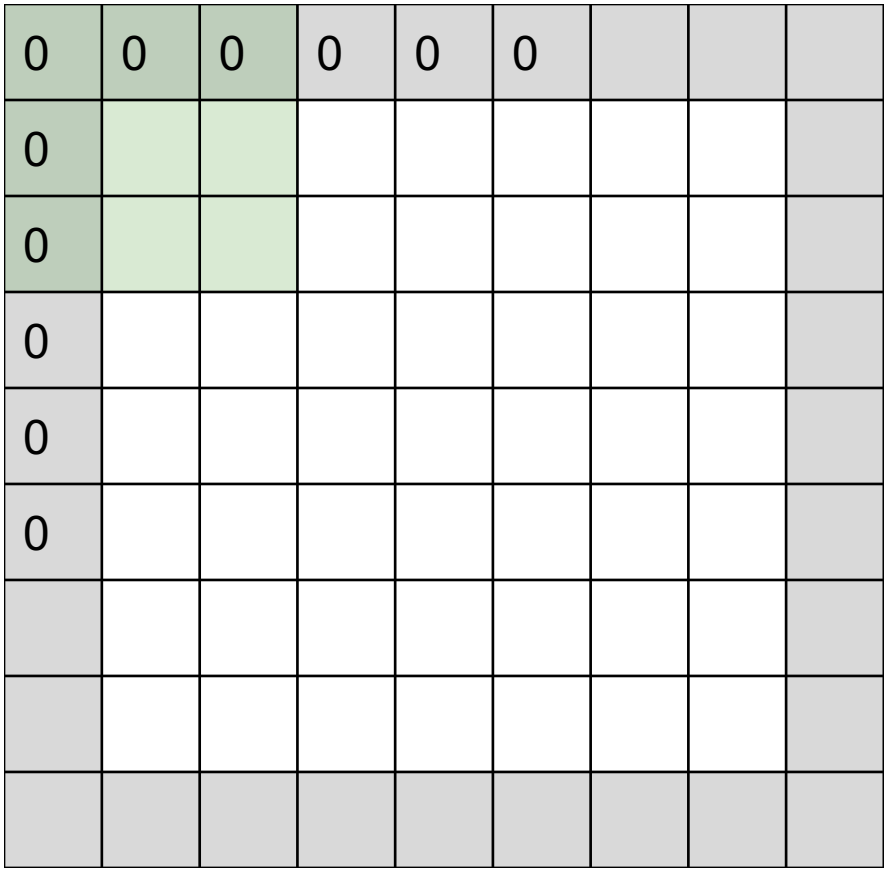
No padding

Padding

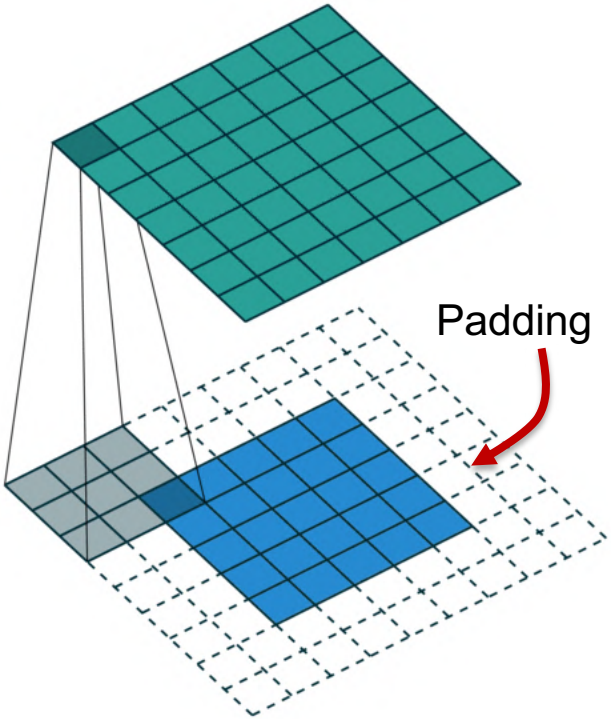
Example

- Input: 7x7
- Filter: 3x3
- Output: 5x5

- Padding: 1=(3-1)/2
- Output: 7x7



Padding



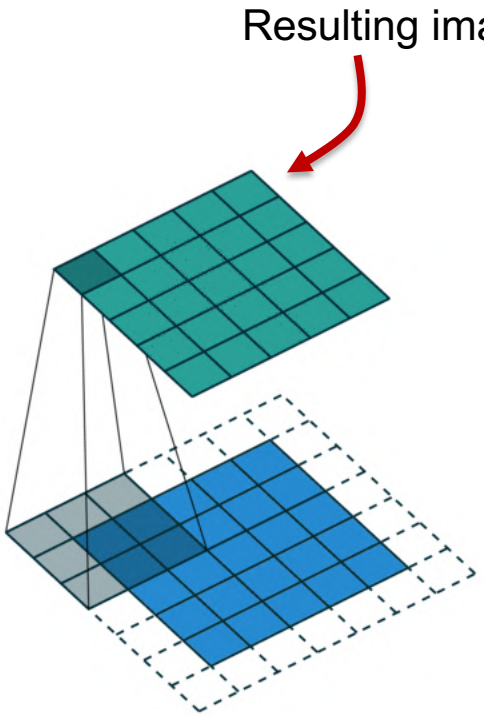
5x5 image, 3x3 filter

Full

(full padding)

Output = $(N + F - 1)$

Padding = $(F - 1)$



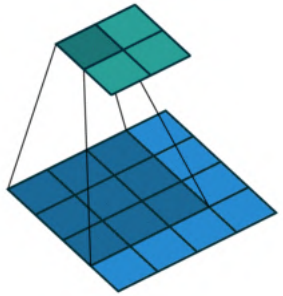
5x5 image, 3x3 filter

Same

(half padding)

Output = N

Padding = $(F - 1)/2$



4x4 image, 3x3 filter

Valid

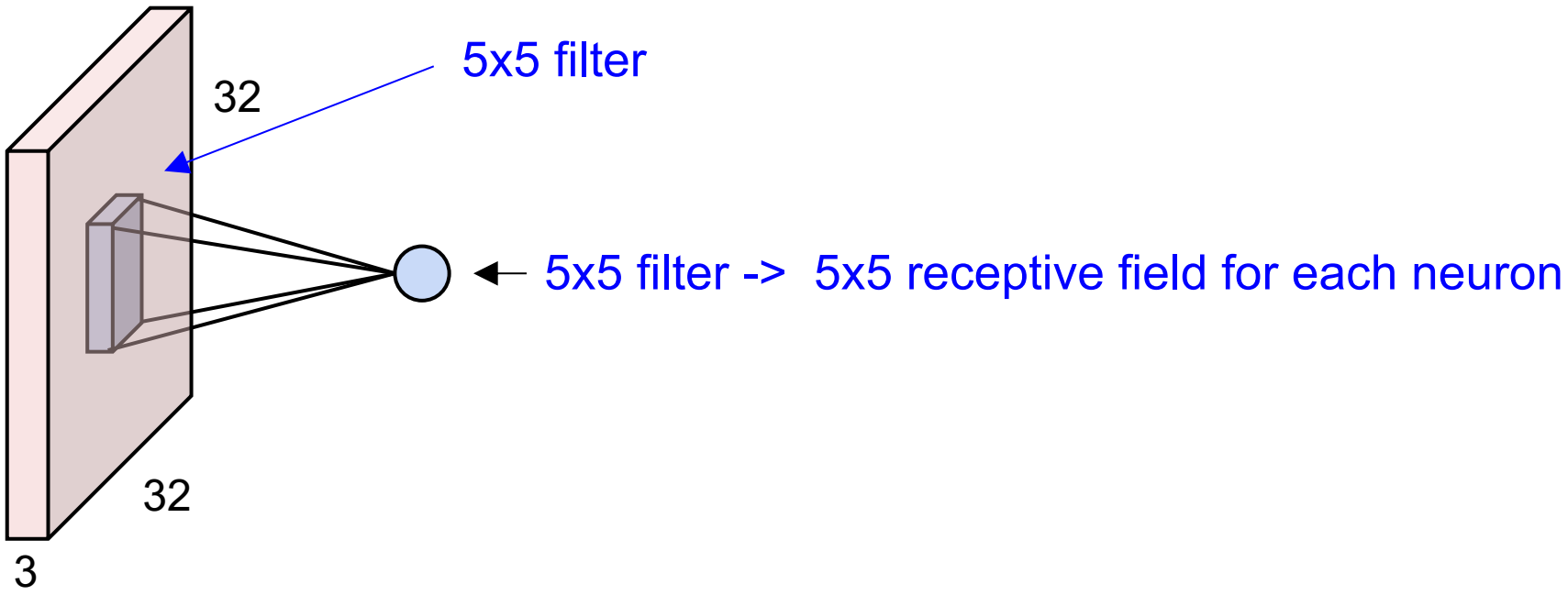
(no padding)

Output = $(N - F + 1)$

Padding = 0

Receptive field

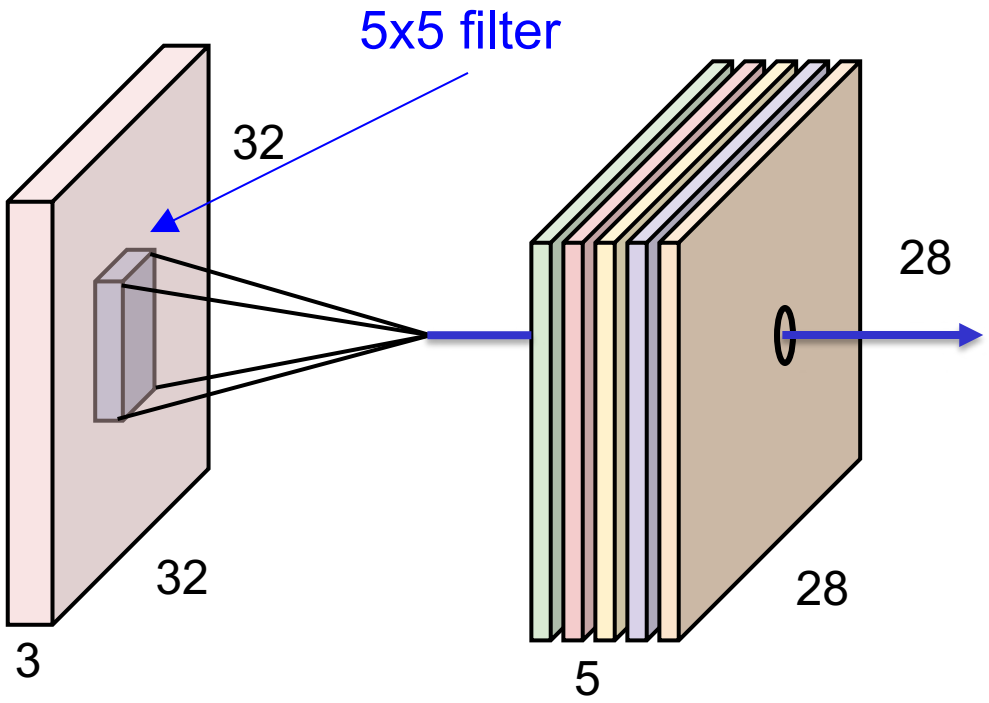
- The spatial region in the input image that influences a neuron respond is called “receptive field”.
- The deeper the layer is the larger the receptive field.



Source: Stanford CS231

Receptive field: multiple channels

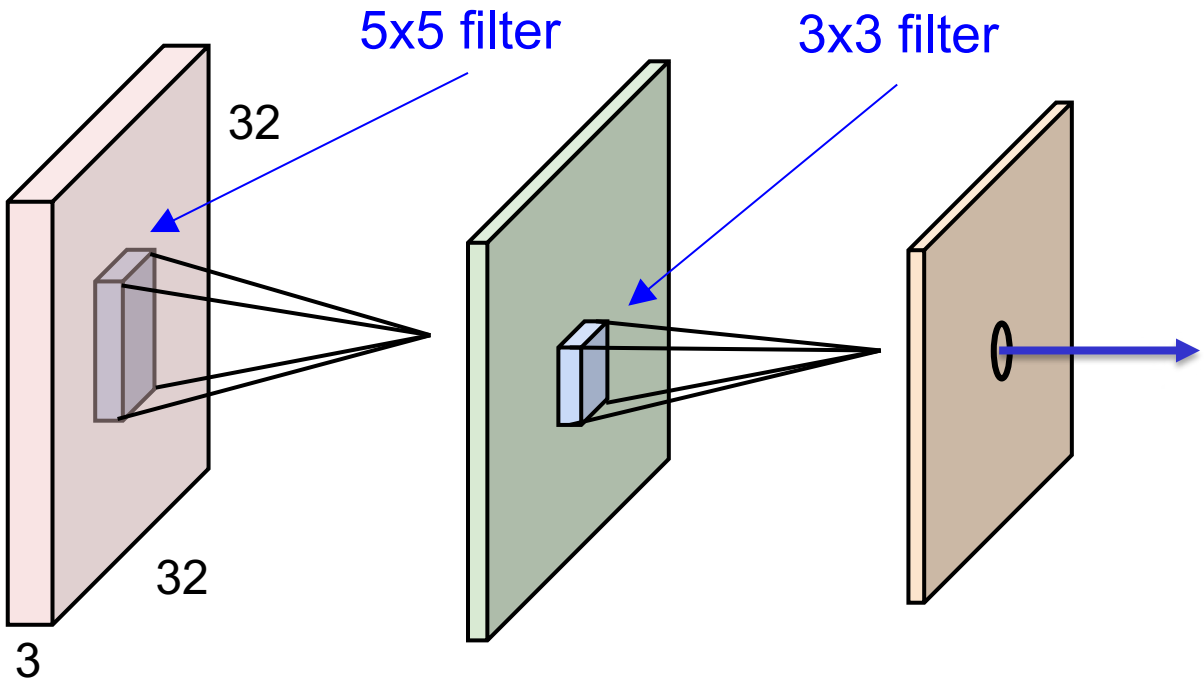
- If we have a multiple channel layer, there will be C different channels, all having the same receptive field.



Source: Stanford CS231

Receptive field: multiple layers

- What will be the receptive field of a neuron in the 2nd layer?
- **Answer**: Similar to the size of a "full" convolution of a 3x3 filter with a 5x5 image = (5+3-1)x(5+3-1) = 7x7

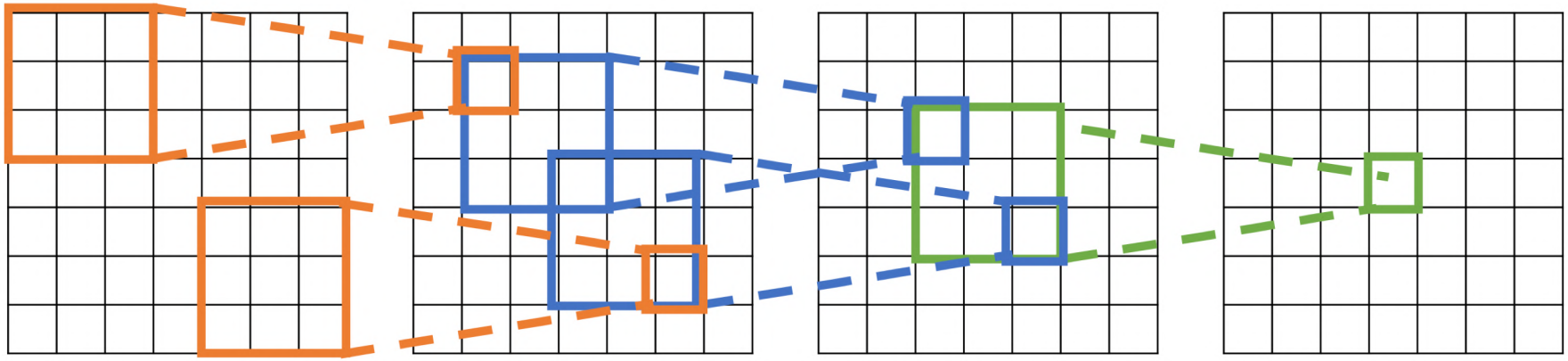


Source: Justin Johnson

Receptive field: multiple layers

- For L successive conv. with KxK filters, the RF is

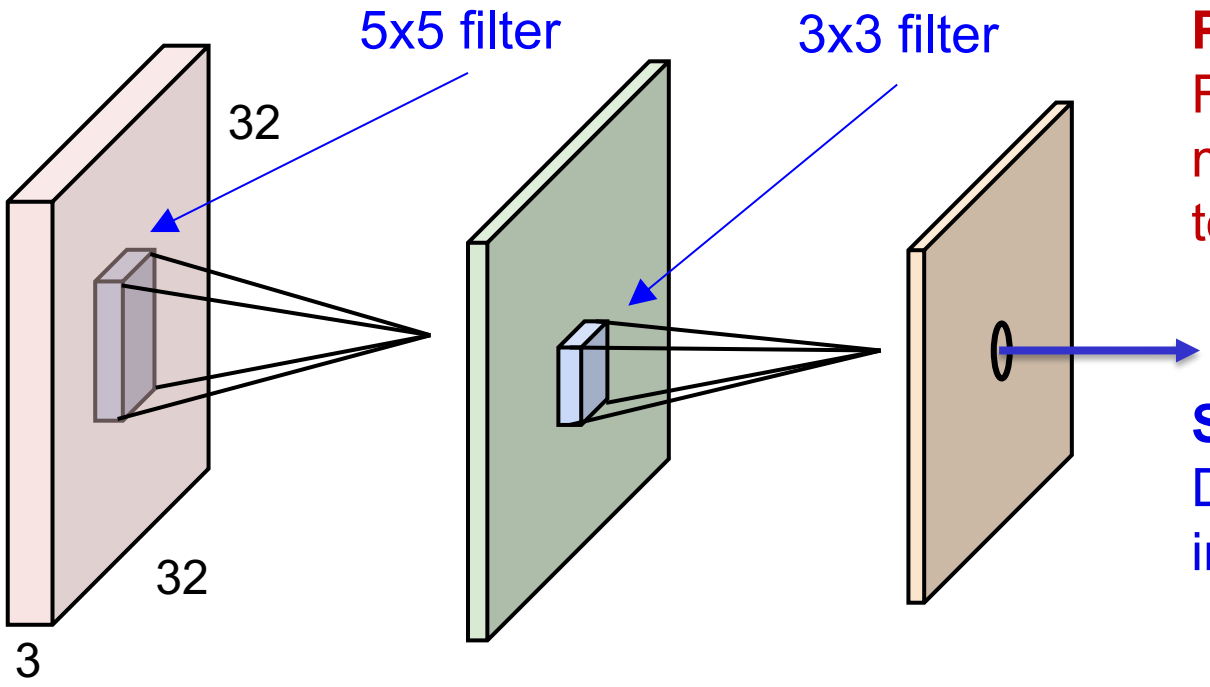
$$RF = 1 + L * (K - 1)$$



Source: Justin Johnson

Receptive field: multiple layers

- What will be the receptive field of a neuron in the 2nd layer?
- **Answer**: Similar to the size of a "full" convolution of a 3x3 filter with a 5x5 image = (5+3-1)x(5+3-1) = 7x7



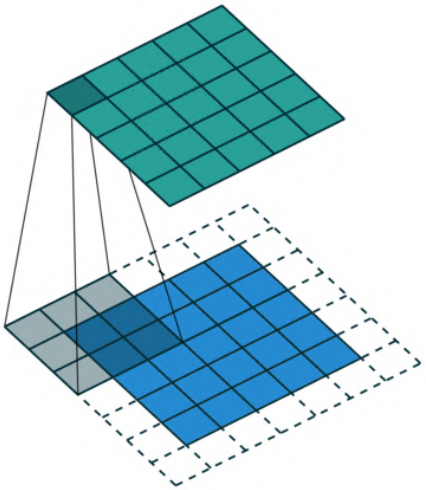
Problem:
For large image we need many layers for an output to "see" the whole image.

Solution:
Downsample the image inside the network!

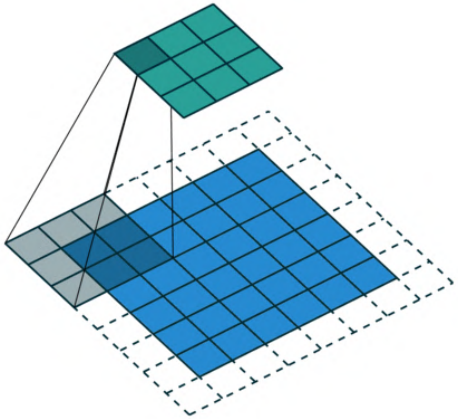
Source: Justin Johnson

Convolutional layer: Stride

- Sometimes we want to scale down the resulting convolution (in the spatial domain).
- It can be implement by down-sampling the results.
- This operation is called "**Striding**".



Stride=1

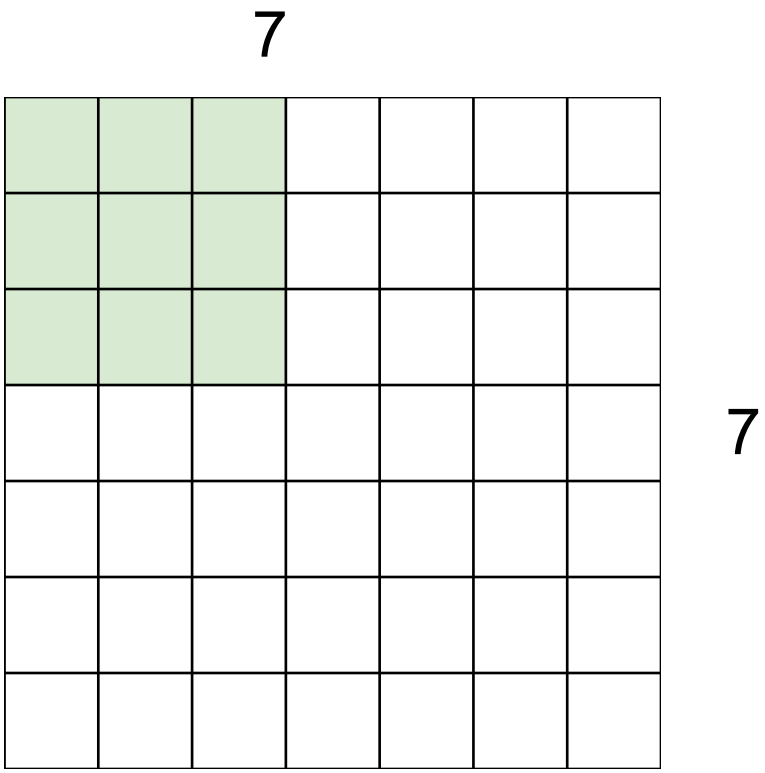


Stride=2

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1

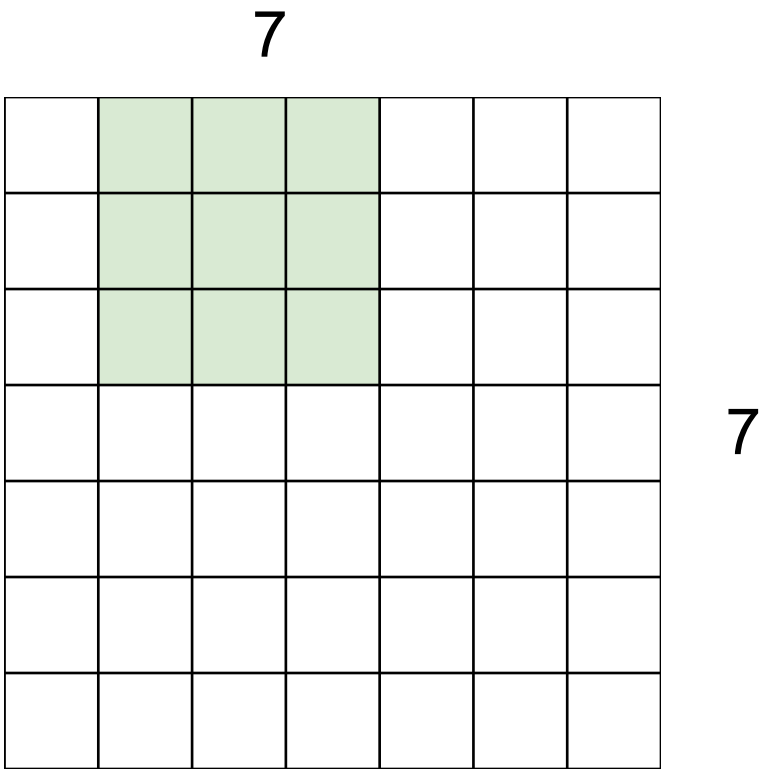


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1

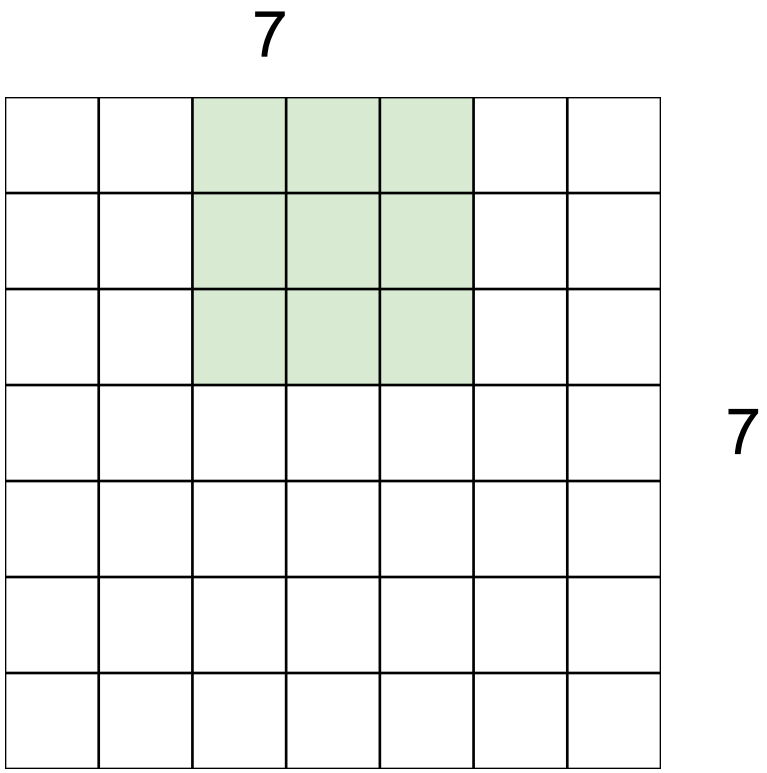


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1

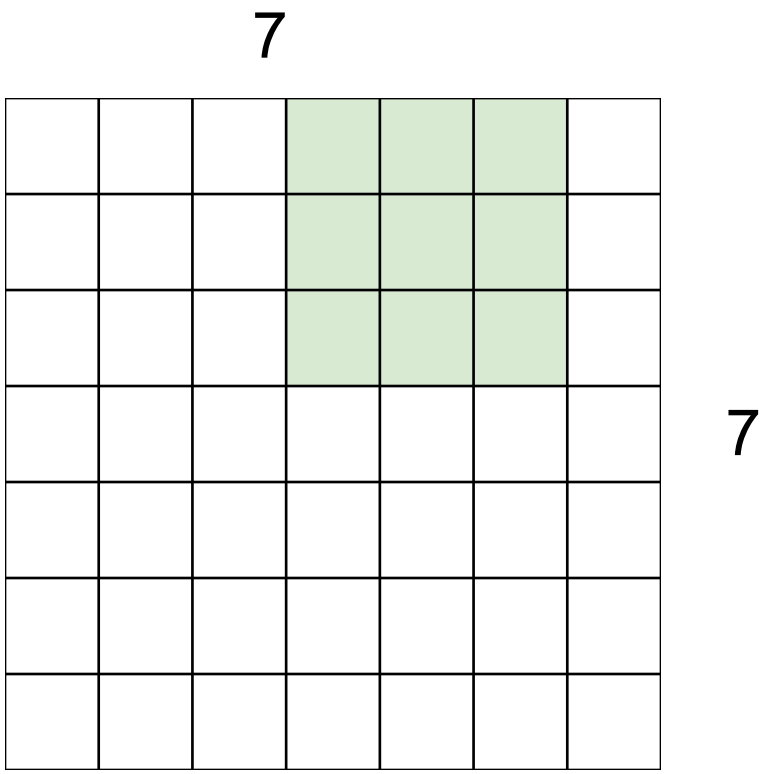


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1

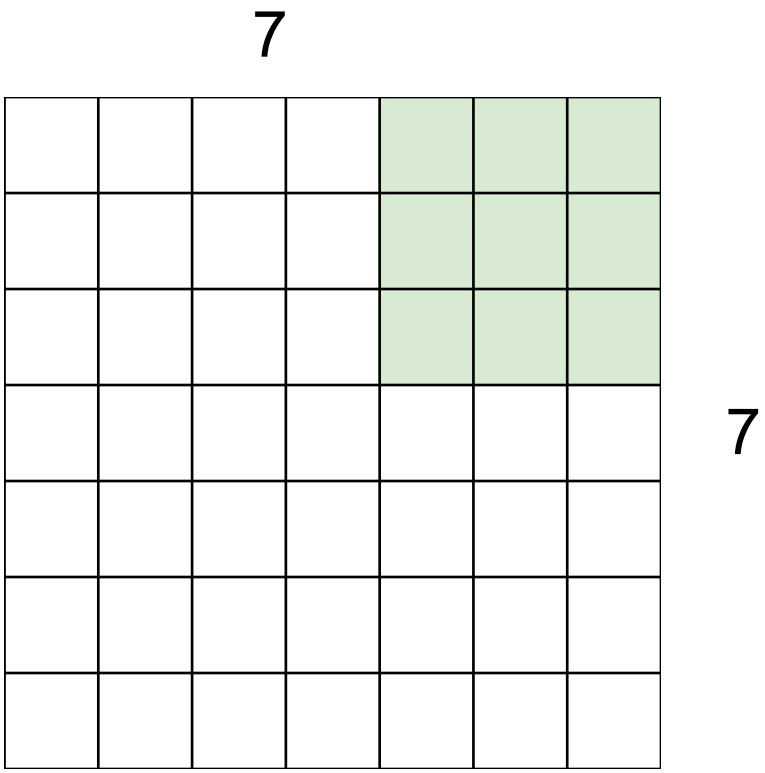


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1

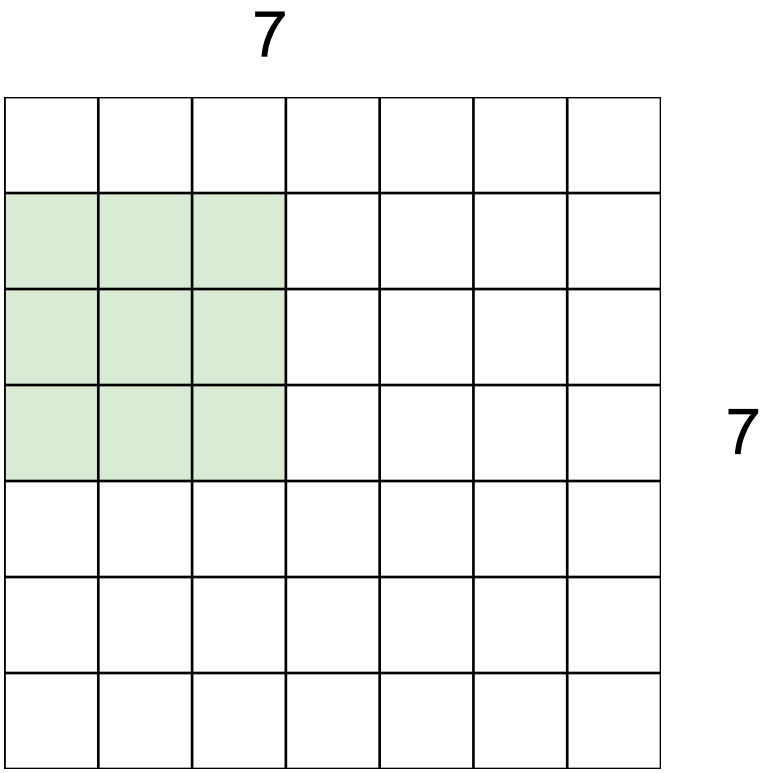


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Standard sliding: stride=1



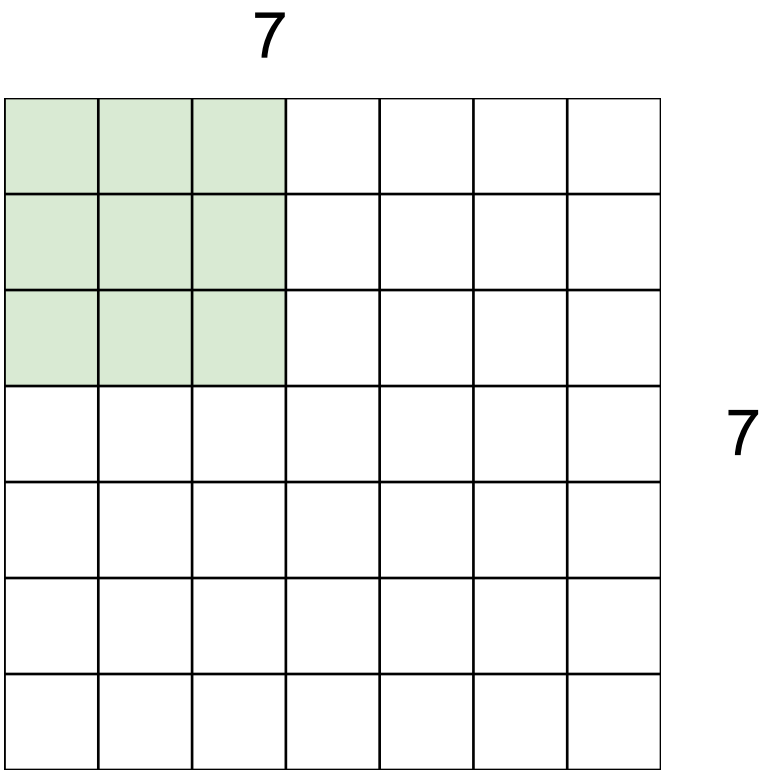
7x7 input (spatially)
assume 3x3 filter

$\Rightarrow$ 5 x 5 output $(7 - 3 + 1)$

Source: Stanford CS231

Convolutional layer: Stride

Now with stride=2

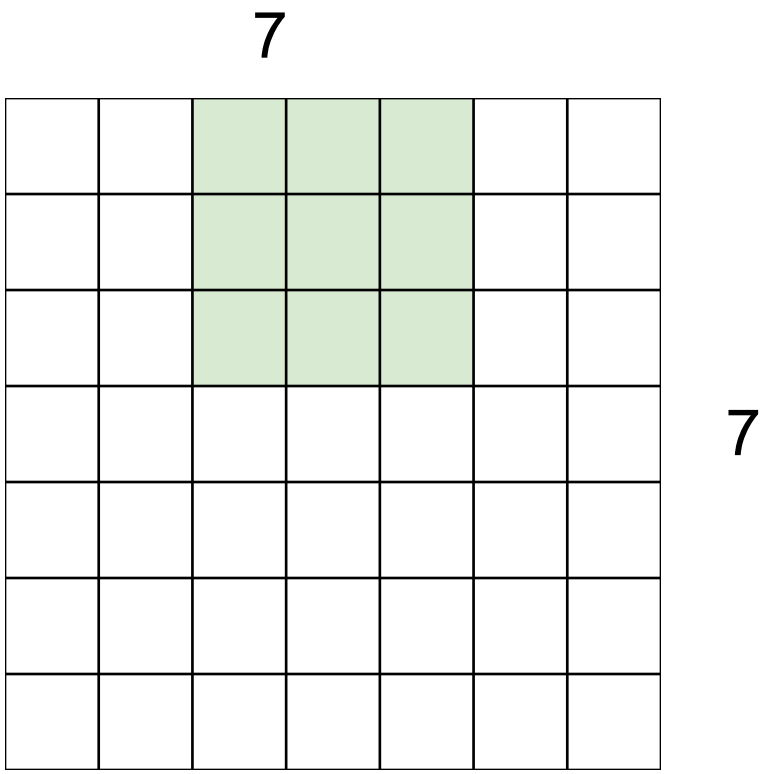


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Now with stride=2

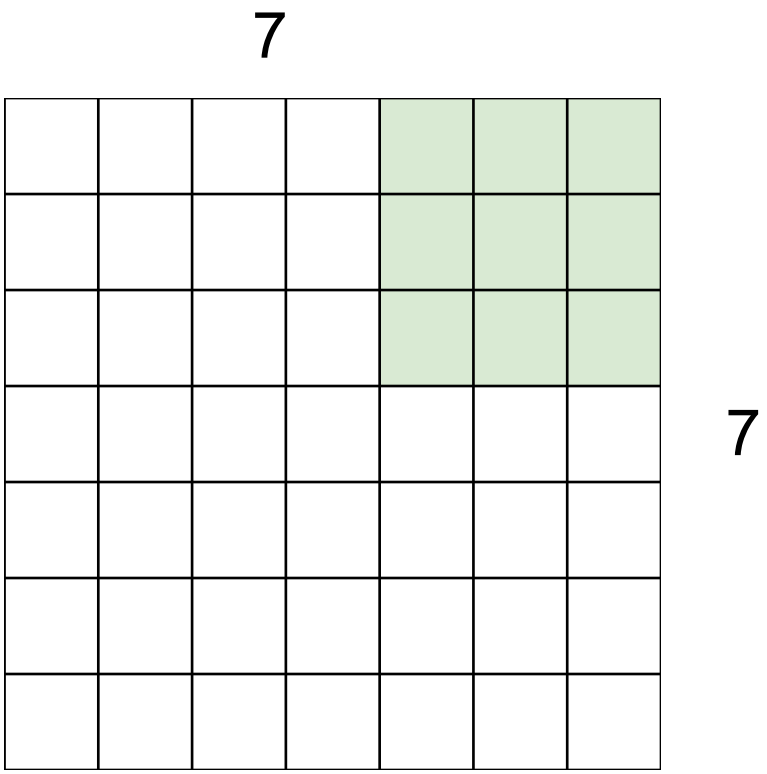


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Now with stride=2

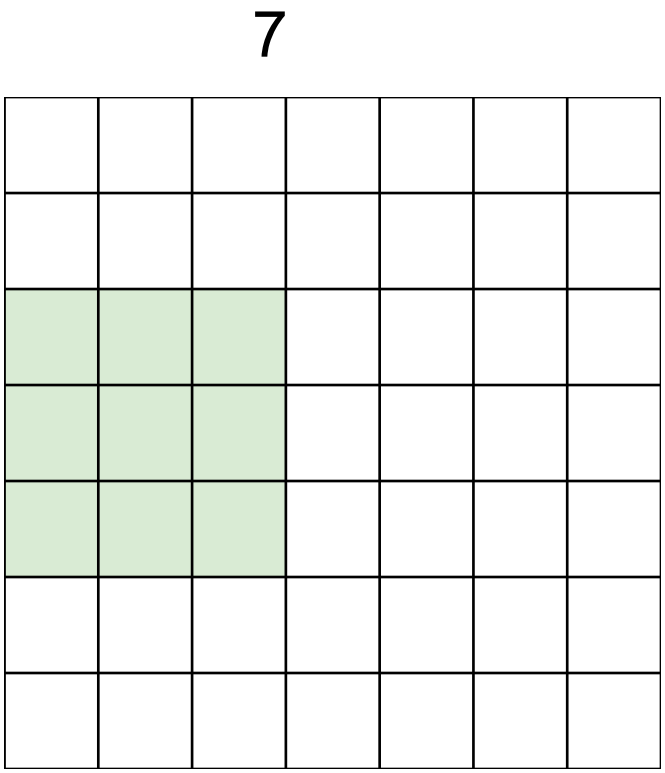


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Now with stride=2



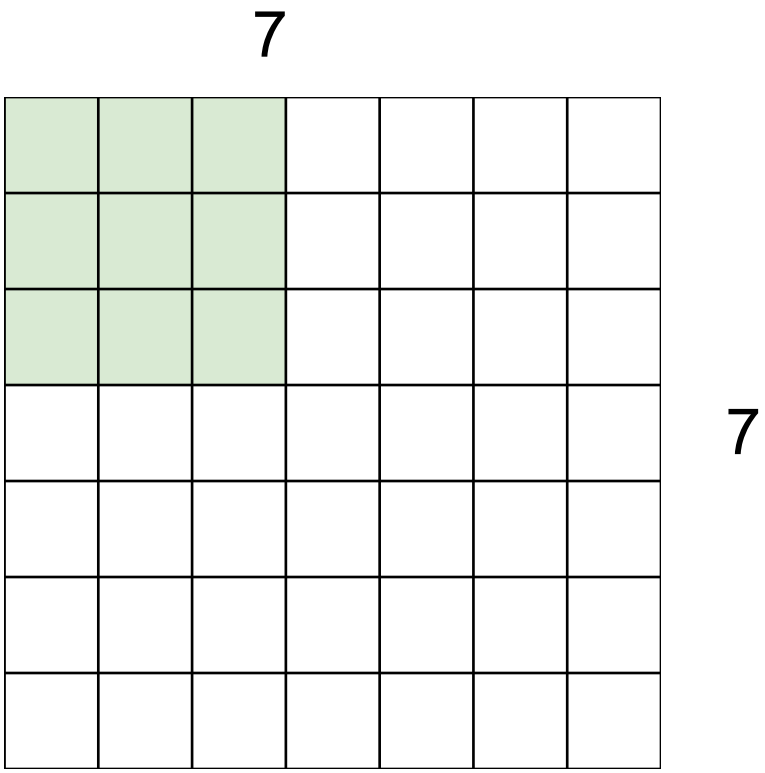
7x7 input (spatially)
assume 3x3 filter

$\Rightarrow$ 3 x 3 output $(\frac{7-3}{2}+1)$

Source: Stanford CS231

Convolutional layer: Stride

Let's try stride=3

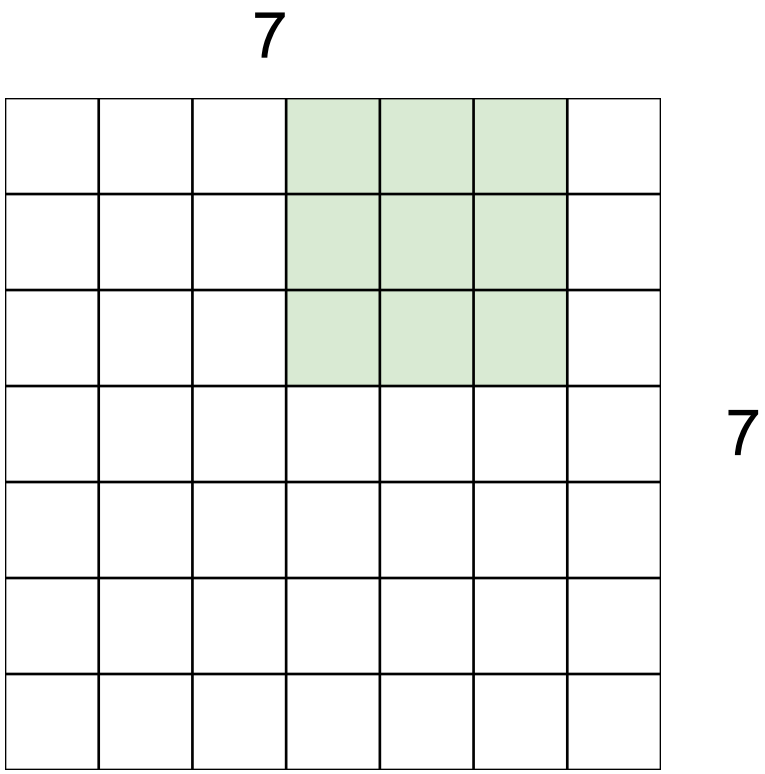


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Let's try stride=3

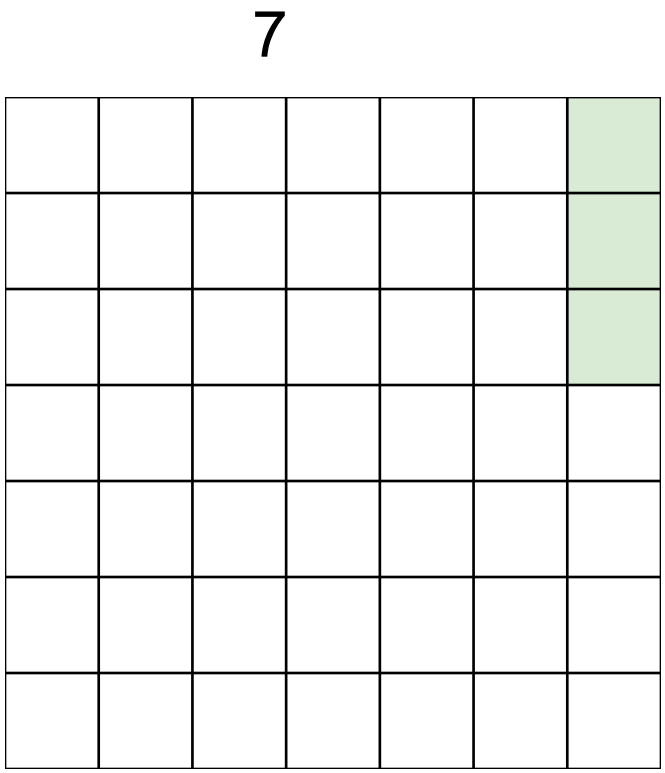


7x7 input (spatially)
assume 3x3 filter

Source: Stanford CS231

Convolutional layer: Stride

Let's try stride=3

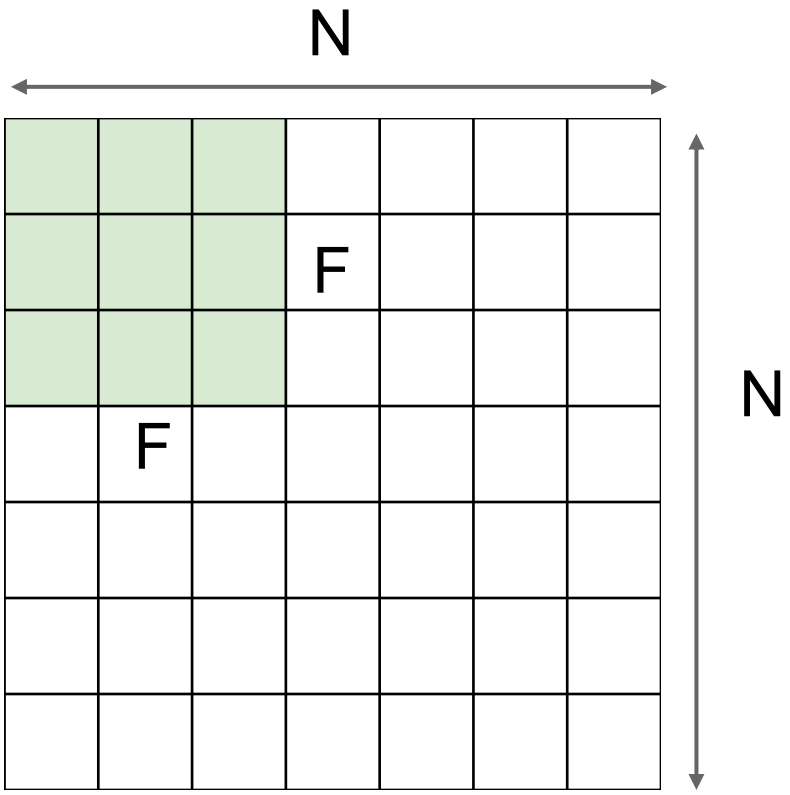


7x7 input (spatially)
assume 3x3 filter

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

Source: Stanford CS231

Convolutional layer: Stride



In the general case, output size:

$$\frac{N - F}{stride} + 1$$

N = 7, F = 3:

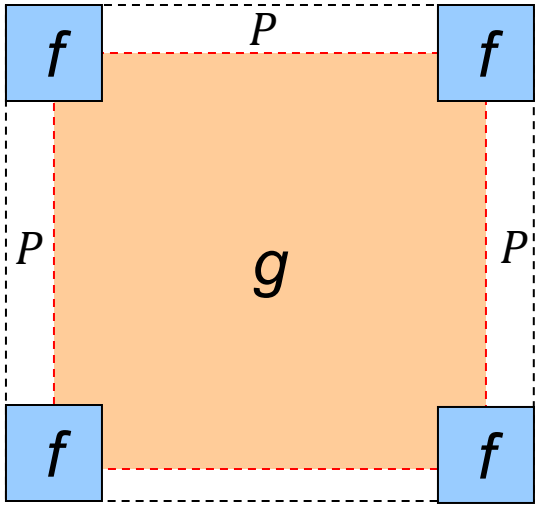
- stride 1 => (7 - 3)/1 + 1 = 5
- stride 2 => (7 - 3)/2 + 1 = 3
- stride 3 => (7 - 3)/3 + 1 = 2.33 ☹

Solution?: Use zero padding!!

Source: Stanford CS231

Convolutional layer: Stride

A closer look at spatial dimensions:



In the general case, output size:

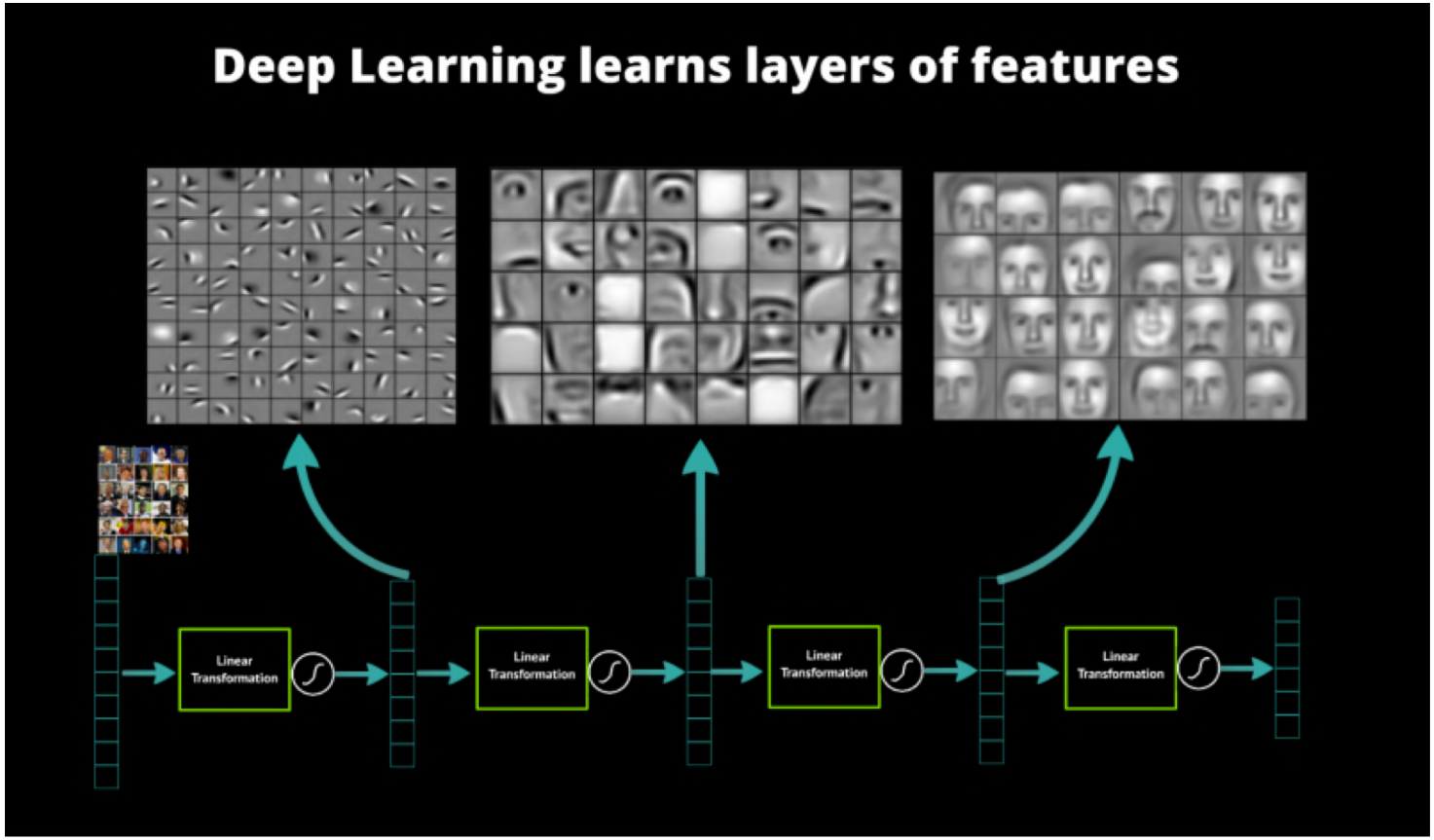
$$\frac{N - F + 2P}{stride} + 1$$

N = 7, F = 3, P = 1:

- stride 3 => $(7 - 3 + 2)/3 + 1 = 3$ ☺

Source: Stanford CS231

Feature Extraction by Convolution



Convolution layers for feature extraction

What does this convolution kernel do?

$*$

0	0	0
0	8	0
0	0	0

$=$

- What will be the average value of the resulting image?

Source: Roger Grosse

Convolution layers for feature extraction

What does this convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0

$=$

- What will be the average value of the resulting image?

Source: Roger Grosse

Convolution layers for feature extraction

What does this convolution kernel do?

$*$

0	-1	0
-1	4	-1
0	-1	0

$=$

- What will be the average value of the resulting image?

Source: Roger Grosse

Convolution layers for feature extraction

What does this convolution kernel do?

$*$

0	-1	0
-1	8	-1
0	-1	0

$=$

- What will be the average value of the resulting image?

Source: Roger Grosse

Convolution layers for feature extraction

What does this convolution kernel do?

$*$

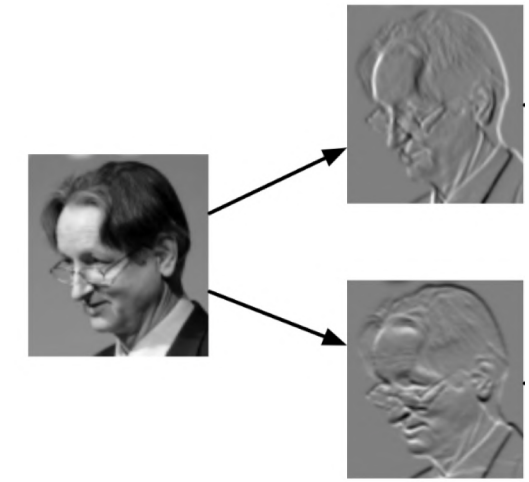
1	0	-1
2	0	-2
1	0	-1

$=$

Source: Roger Grosse

Convolution layers for feature extraction

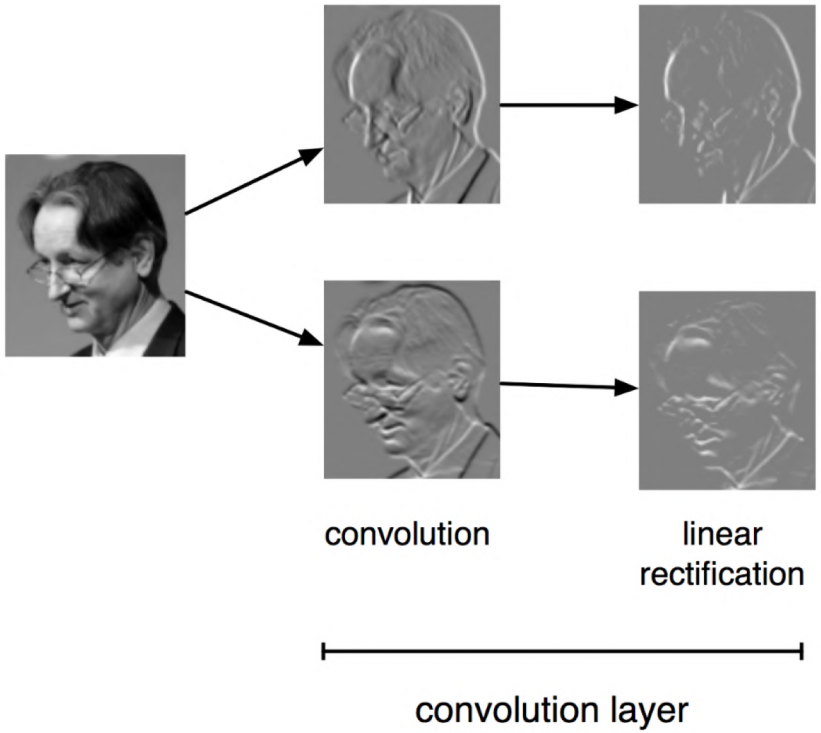
- The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



Source: Roger Grosse

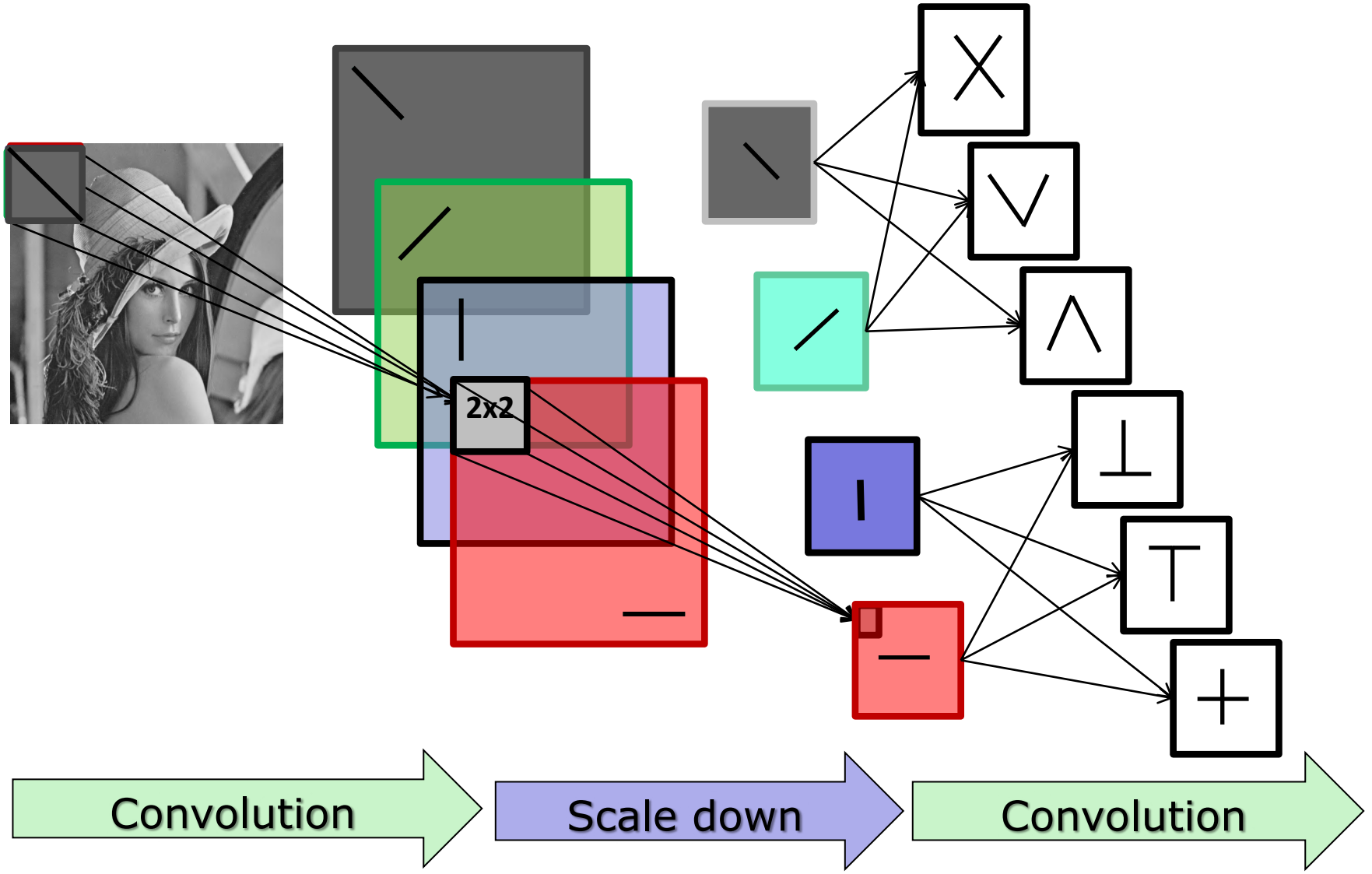
Convolution layers for feature extraction

- The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.
- It followed by a non-linear activation function e.g. ReLU:



Source: Roger Grosse

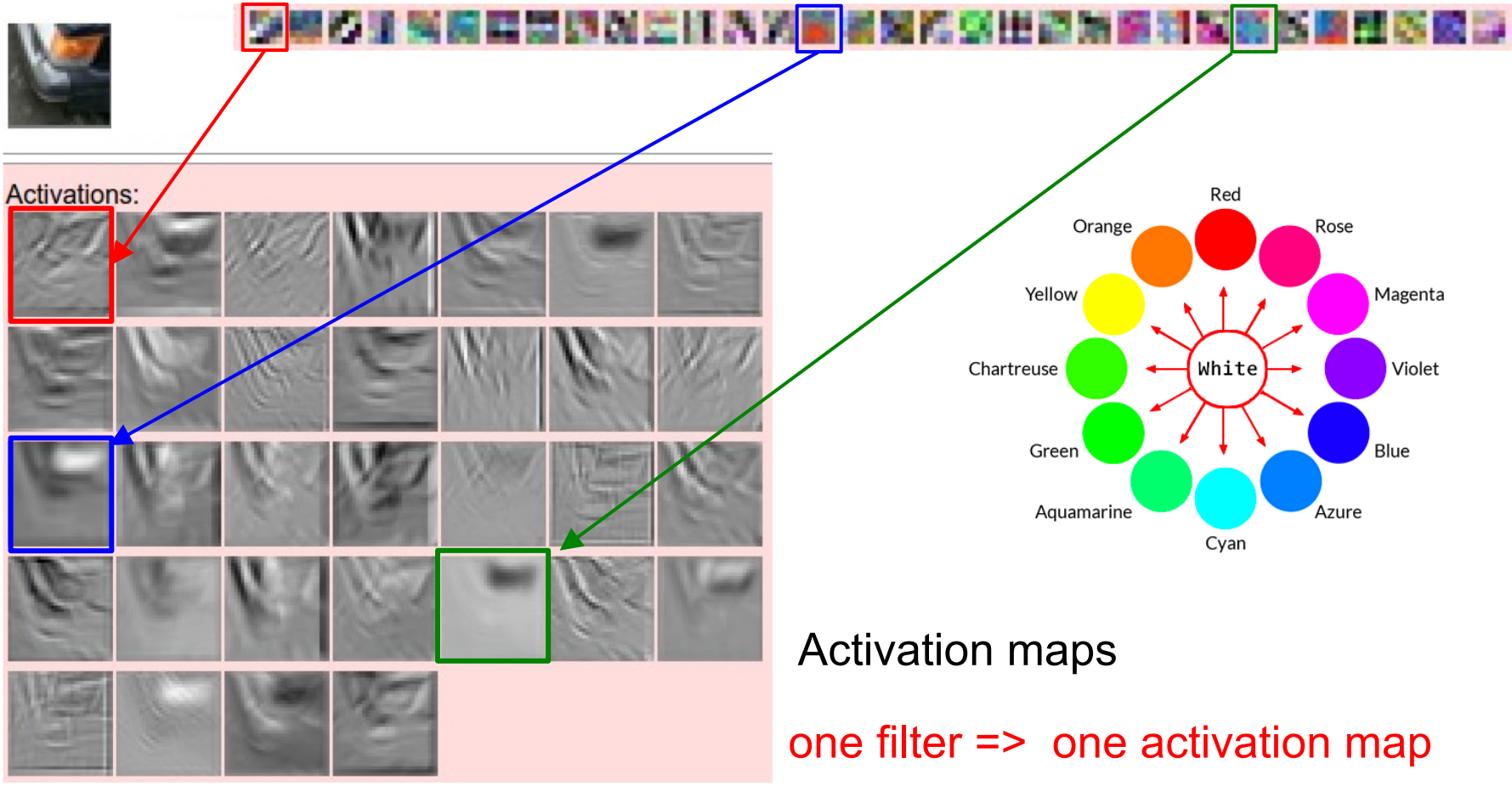
Feature extraction by composition



Source: Boris Ginzburg

Convolution layers for feature extraction

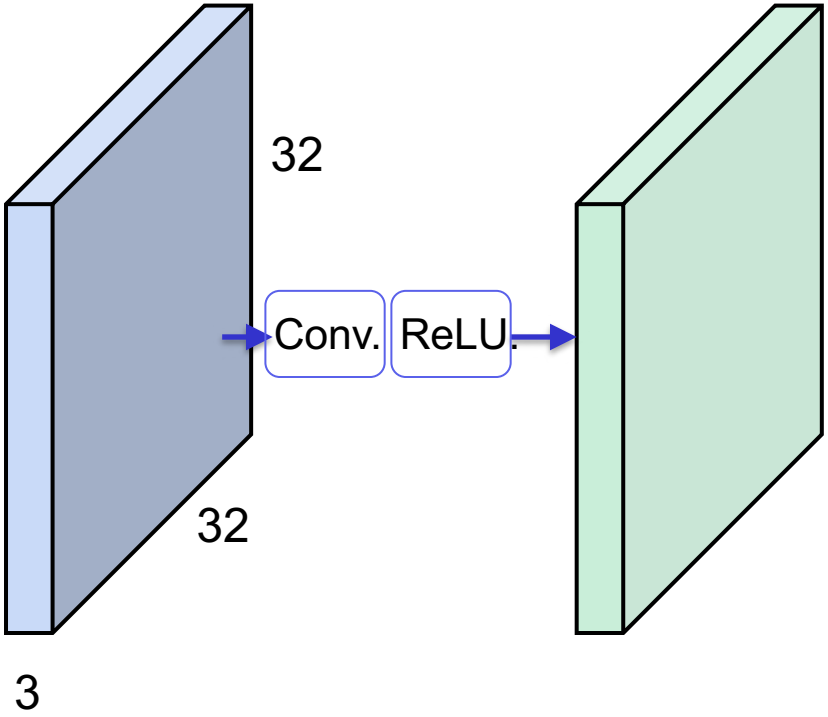
Example of a layer with 32 filters 5x5 each



Activation maps

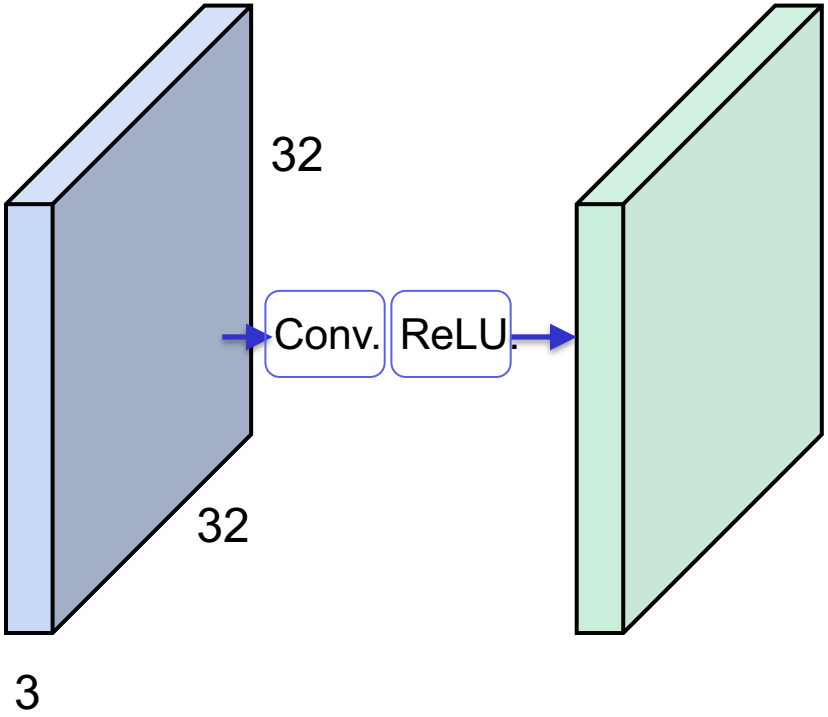
one filter => one activation map

What do Convolution filters learn?



Source: Justin Johnson

What do Convolution filters learn?

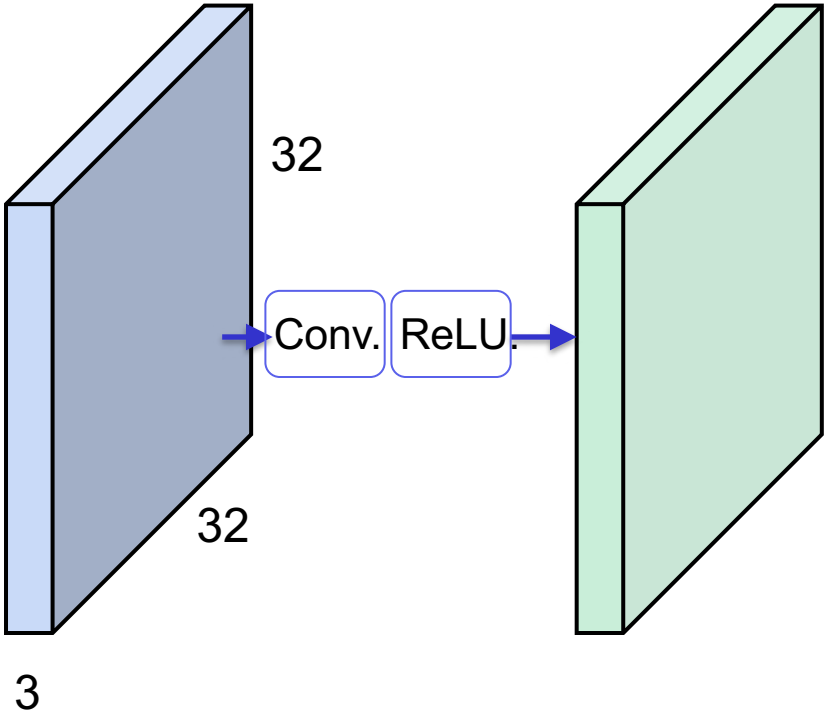


Linear Classifier: One template per class

Source: Justin Johnson

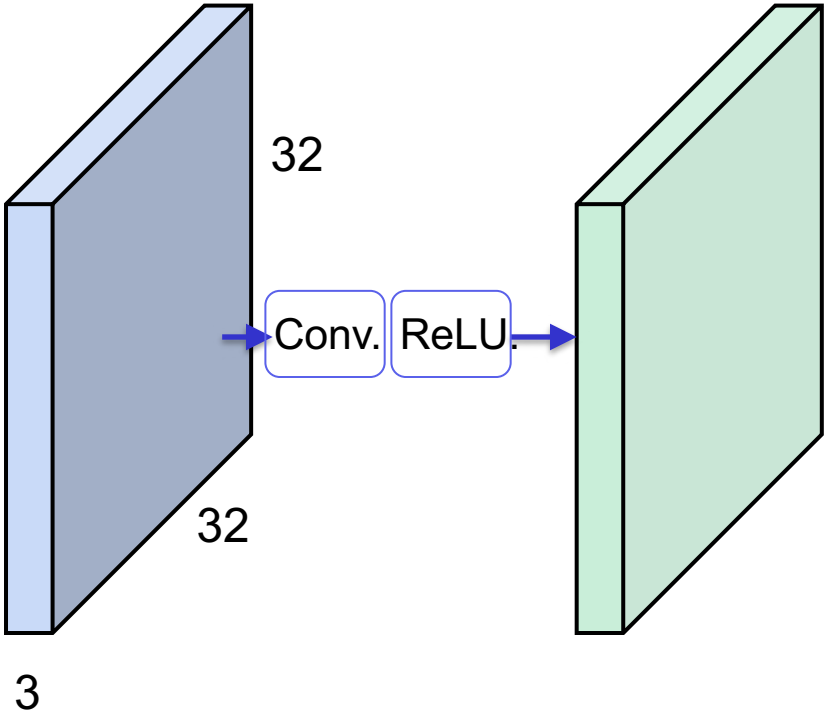
What do Convolution filters learn?

MLP: A bank of whole image component

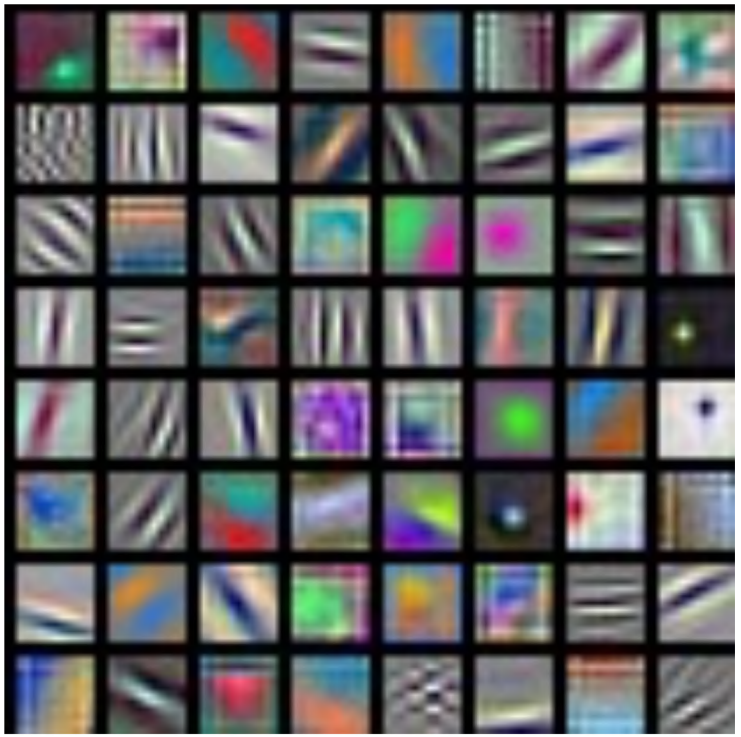


Source: Justin Johnson

What do Convolution filters learn?



1st layer Conv. filters: local image templates, oriented edge, opposing colors, high freq. -> grayscale filters



AlexNet: 64 filters, each 3x11x11

Source: Justin Johnson

Pytorch Convolutions layer

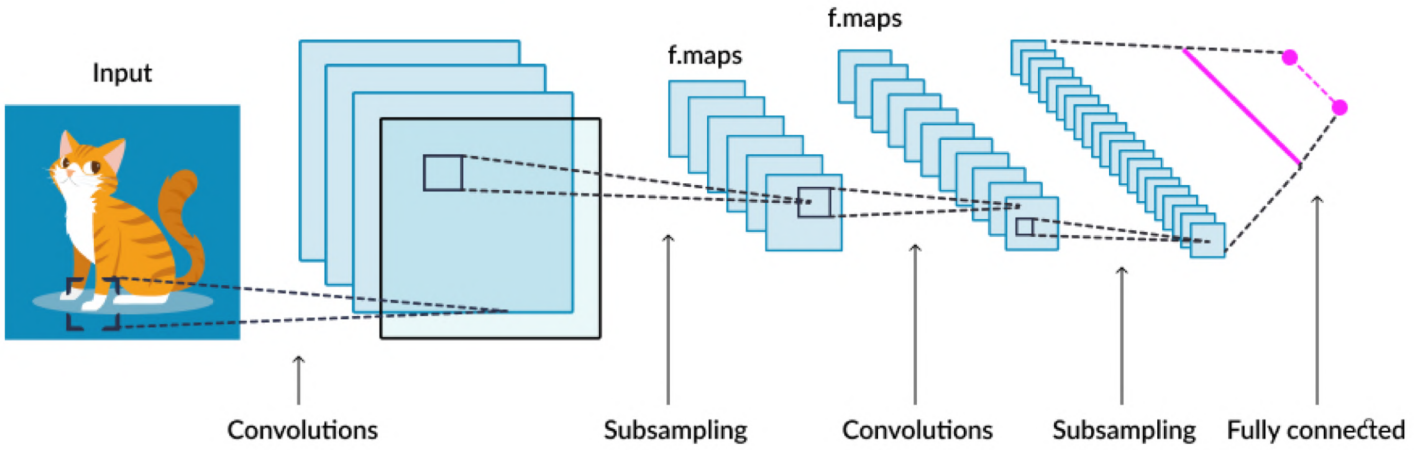
torch.nn.functional.conv2d(*input*, *weight*, *bias=None*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*) → Tensor

Applies a 2D convolution over an input image composed of several input planes.

- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iH, iW)$
- **weight** – filters of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, kH, kW)$
- **bias** – optional bias tensor of shape (out_channels). Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a tuple (*sH, sW*). Default: 1
- **padding** –

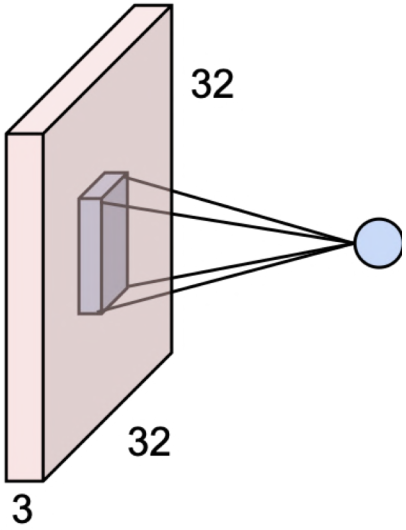
  implicit paddings on both sides of the input. Can be a string {'valid', 'same'}, single number or a tuple (*padH, padW*). Default: 0 `padding='valid'` is the same as no padding. `padding='same'` pads the input so the output has the same shape as the input. However, this mode doesn't support any stride values other than 1.

More on Convolutions

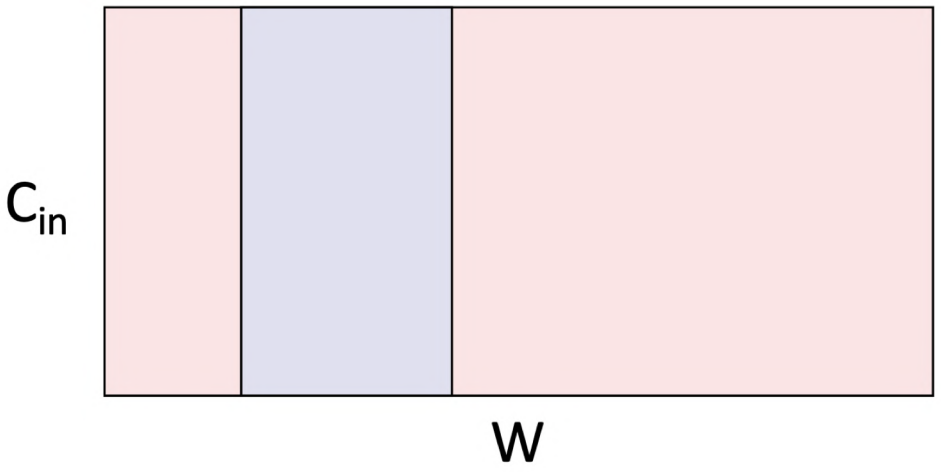


Other Types of Convolutions

So far, 2D convolutions



1D convolutions (e.g. for audio)



Input: $C_{in}xW$

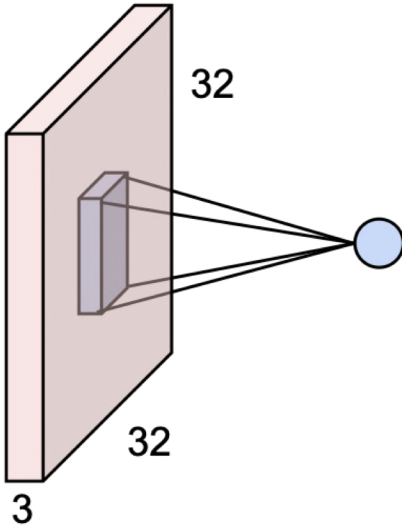
Filters: $C_{out}xC_{in}xF$

Output: $C_{out}xW$

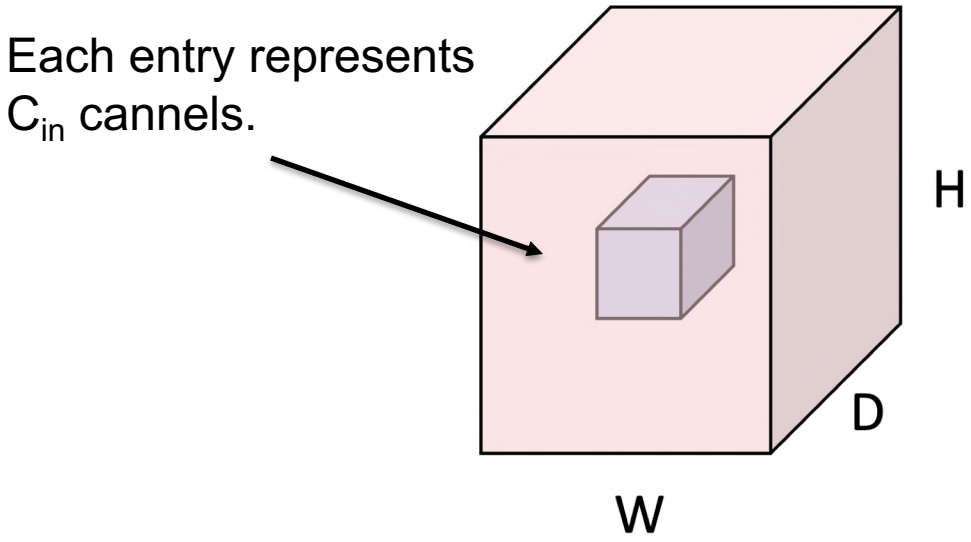
Source: Justin Johnson

Other Types of Convolutions

So far, 2D convolutions



3D convolutions (e.g. for voxels)



Input: C_{in}xHxWxD

Filters: C_{out}xC_{in}xFxFxF

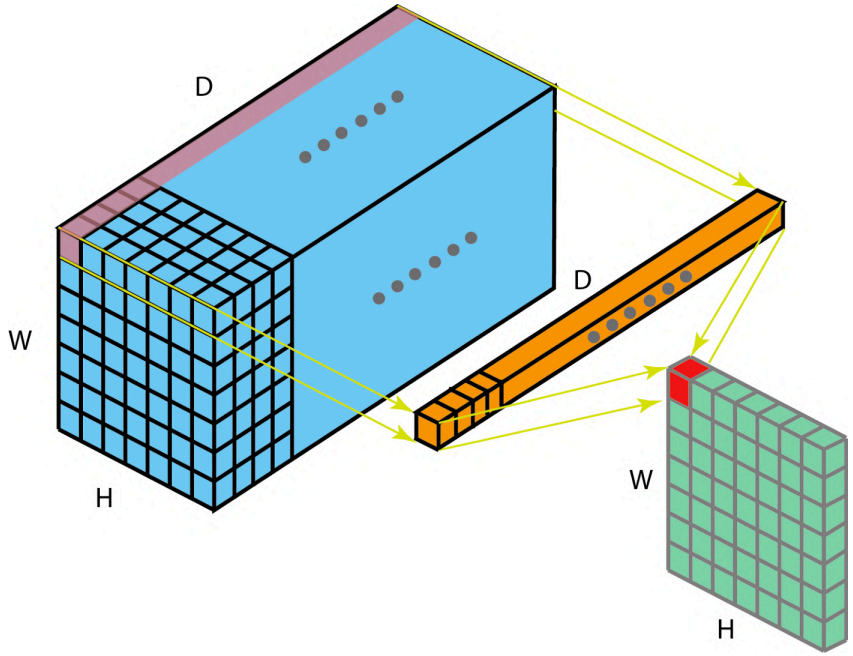
Output: C_{out}xHxWxD

Source: Justin Johnson

Special Convolutions

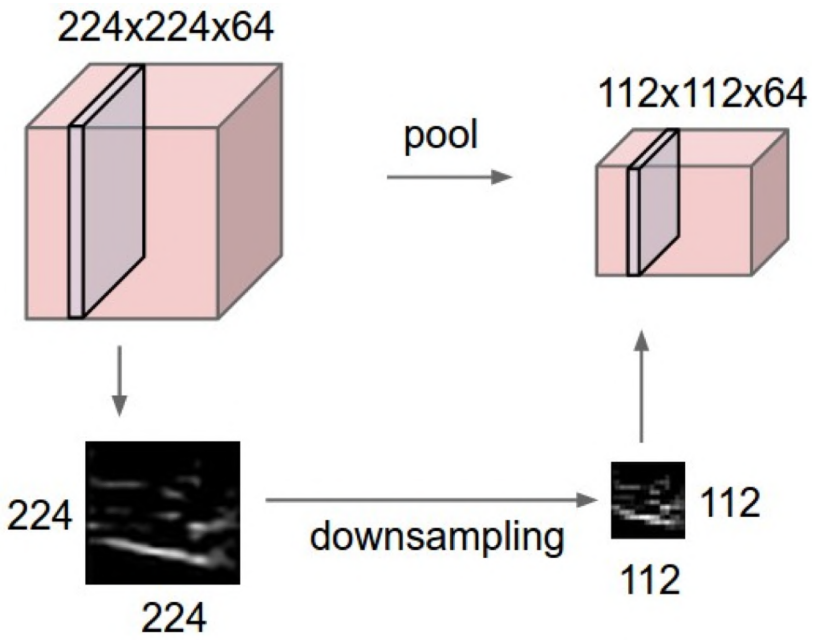
1x1 Convolution

- A 1x1 convolution layers make perfect sense
- It is used for dimensionality reduction for efficient computations or for efficient low dimensional embedding/representation
- 1 x 1 convolutions are used to compute reductions before expensive 3 x 3 and 5 x 5 convolutions.



Pooling layer

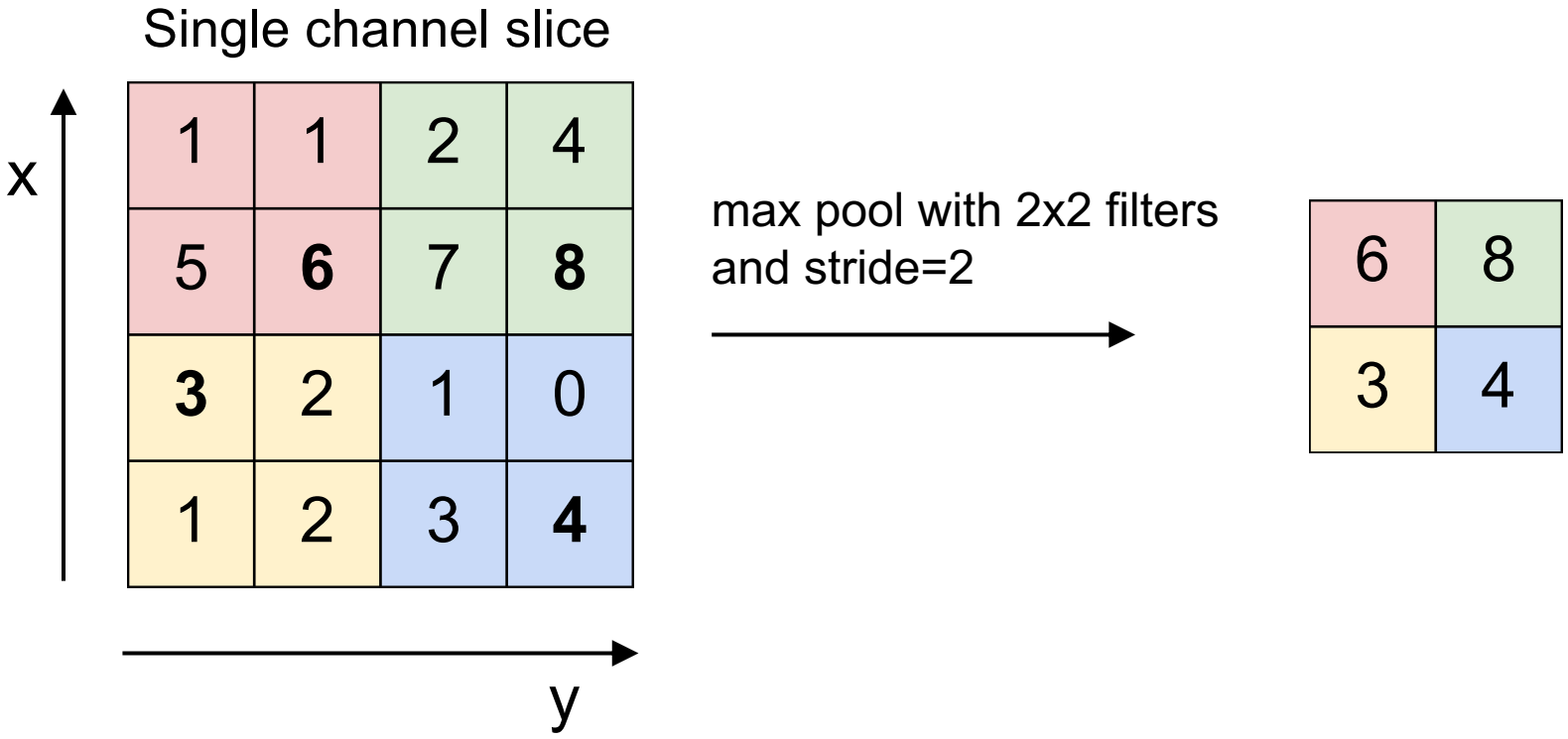
- Like convolution, the **polling layer** extracts a single value for each neighborhood (max pooling / average pooling).
- Commonly it followed by subsampling (stride) so it reduces the spatial size of the layer.
- In contrast to regular filtering, it operates over each activation map independently:



F x F pooling filter, stride S

Usually: F=2 or 3 , S=2

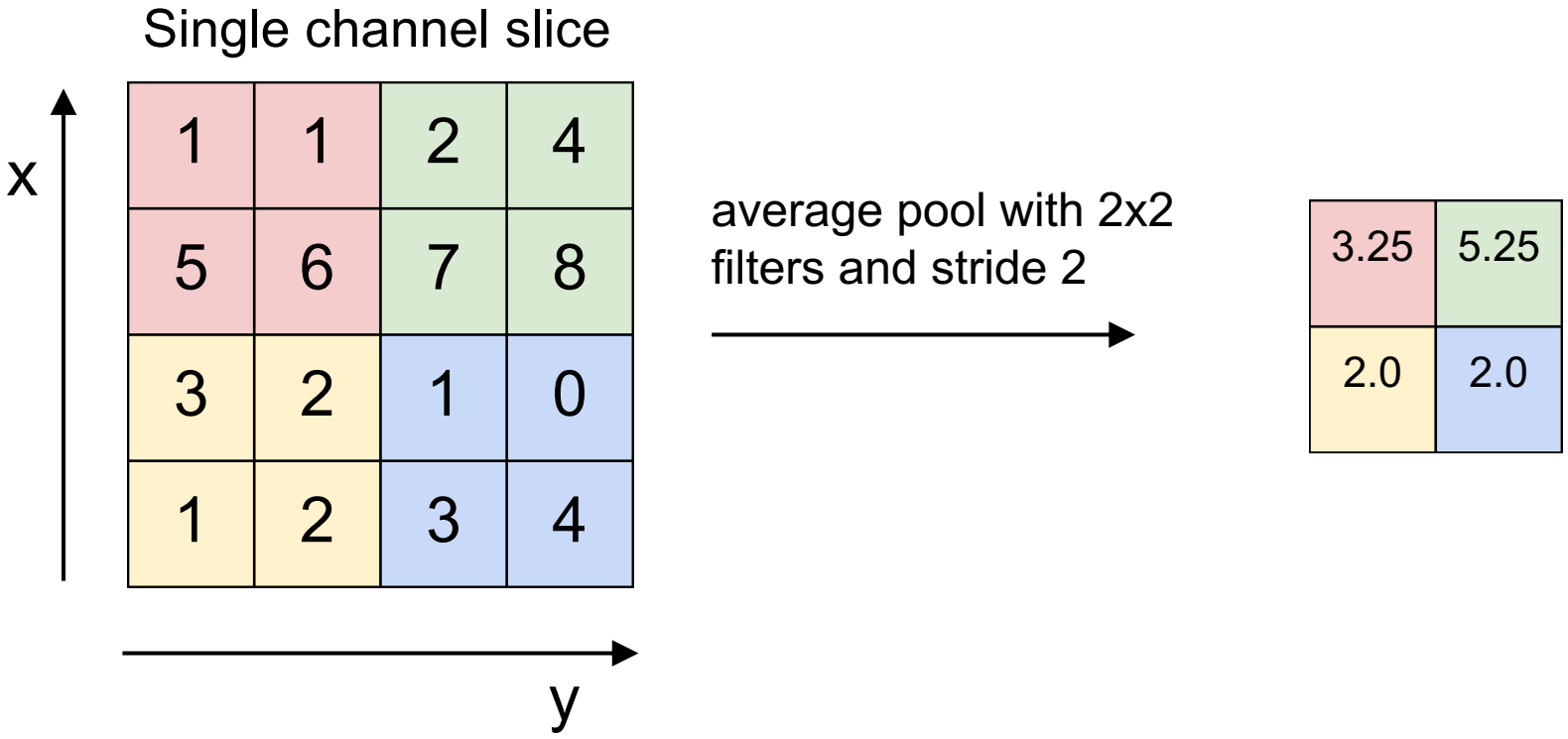
Max Pooling



- Backward pass: gradient from next layer is passed back only to the unit with max value
- Requires keeping the max indices for the backprop.

Source: Stanford CS231

Average Pooling



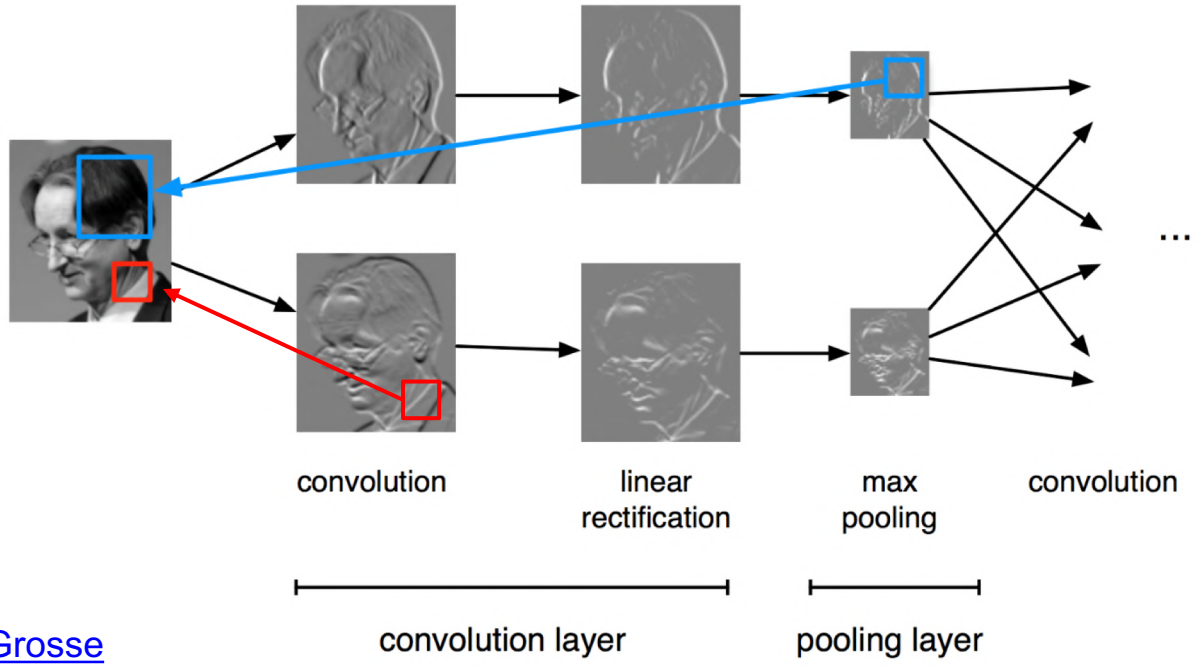
- Max/Average pooling are common to apply with stride=2, but in practice can be applied with any stride.

Source: Stanford CS231

Pooling layer

Why using pooling?

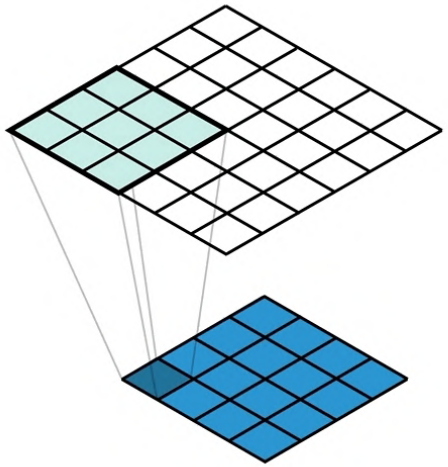
- Reduces the spatial size of the feature maps and accordingly reduces the amount of parameters and network computation
- Makes the network invariant to small geometric perturbations
- Higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.



Source image: Roger Grosse

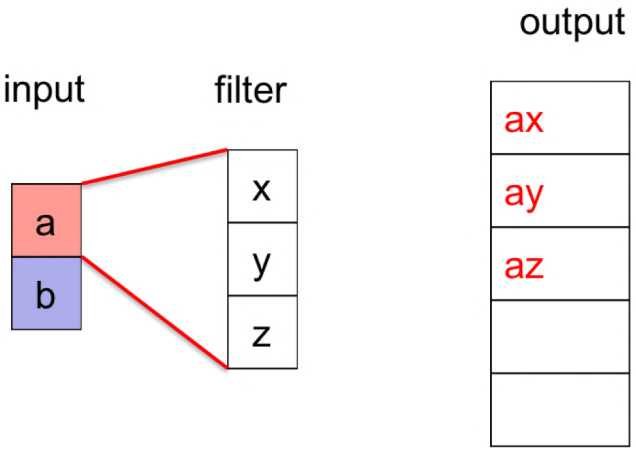
Transpose Convolution

- We saw that Image **down-sampling** can be performed by convolution + striding or by pooling.
- Often, we want to do transformations in the opposite direction i.e. we'd like to perform **up-sampling.**
- To achieve that, we use the **transposed convolution** (also called **deconvolution** or fractionally **strided convolution**).



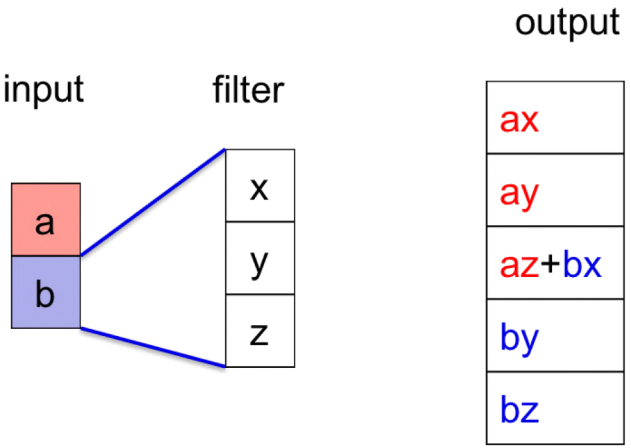
Methods for Upsampling:

- **Transpose convolution**: The most common method for up-sampling. It applies the opposite direction of a common convolution.
- *Typical convolution*: Take a dot product between the filter weights and an image window.
- *Transpose convolution*: Take a single value from the image and multiply the filter weights. Project those weighted values into the output feature map.



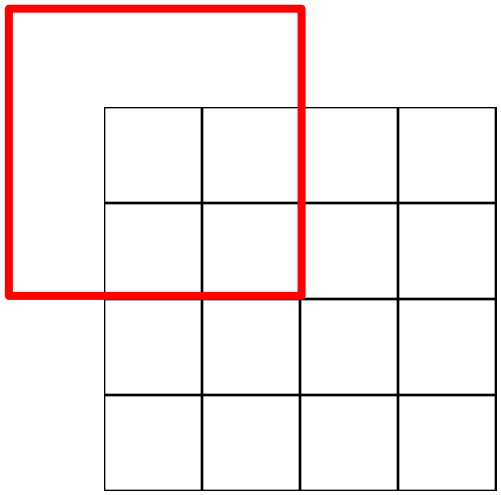
Methods for Upsampling:

- **Transpose convolution**: The most common method for up-sampling. It applies the opposite direction of a common convolution.
- *Typical convolution*: Take a dot product between the filter weights and an image window.
- *Transpose convolution*: Take a single value from the image and multiply the filter weights. Project those weighted values into the output feature map.



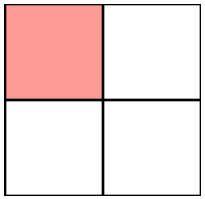
Learnable Upsampling: Conv-transpose

Common 3 x 3 convolution, **stride 2** pad 1



Input: 4 x 4

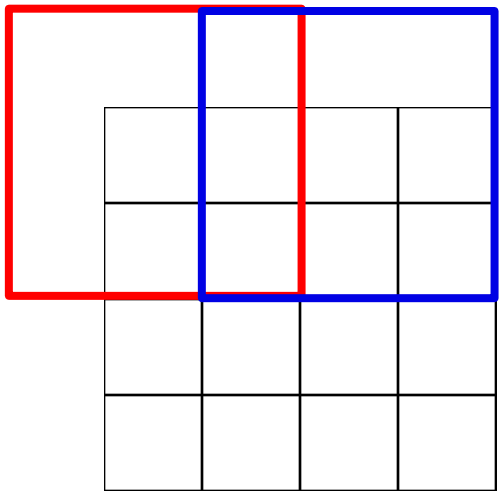
Dot product between filter and input



Output: 2 x 2

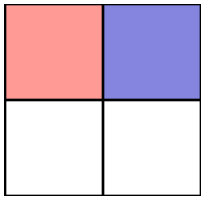
Learnable Upsampling: Conv-transpose

Common 3 x 3 convolution, **stride 2** pad 1



Input: 4 x 4

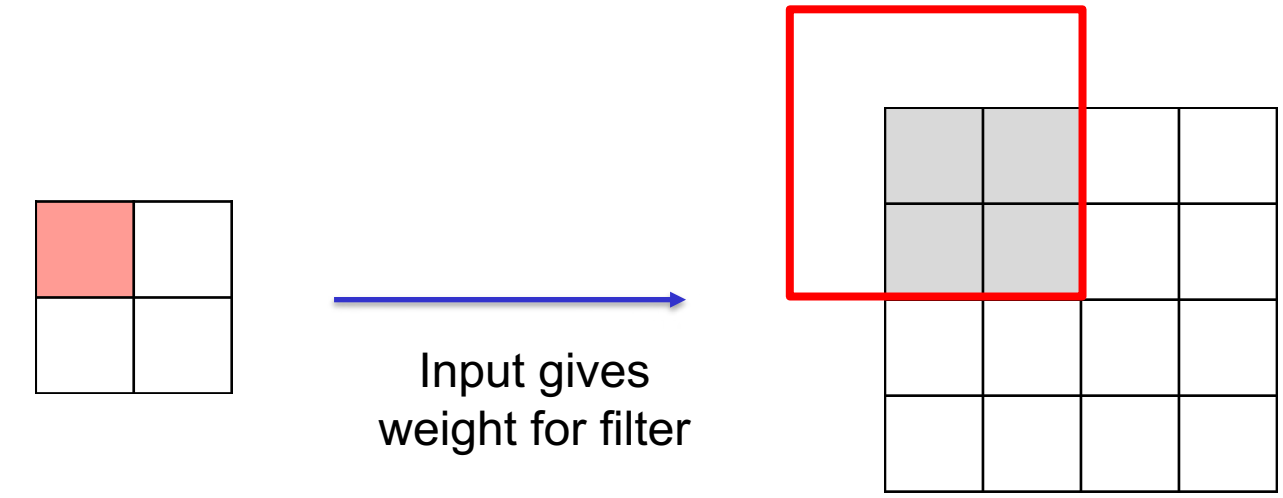
Dot product between filter and input



Output: 2 x 2

Learnable Upsampling: Conv-transpose

3 x 3 **conv-transpose**, **stride 2** pad 1

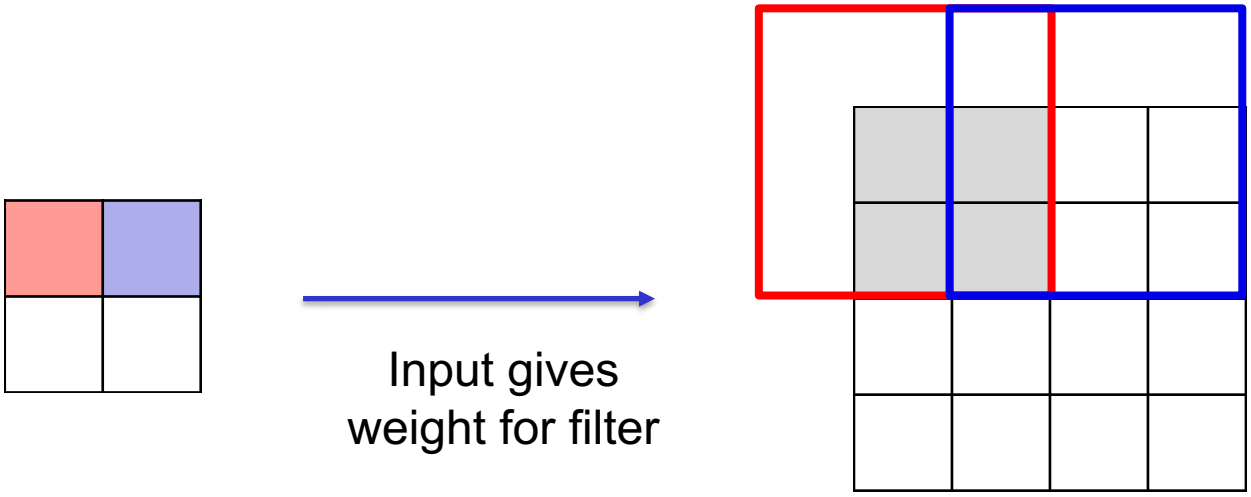


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Conv-transpose

3 x 3 **conv-transpose**, **stride 2** pad 1

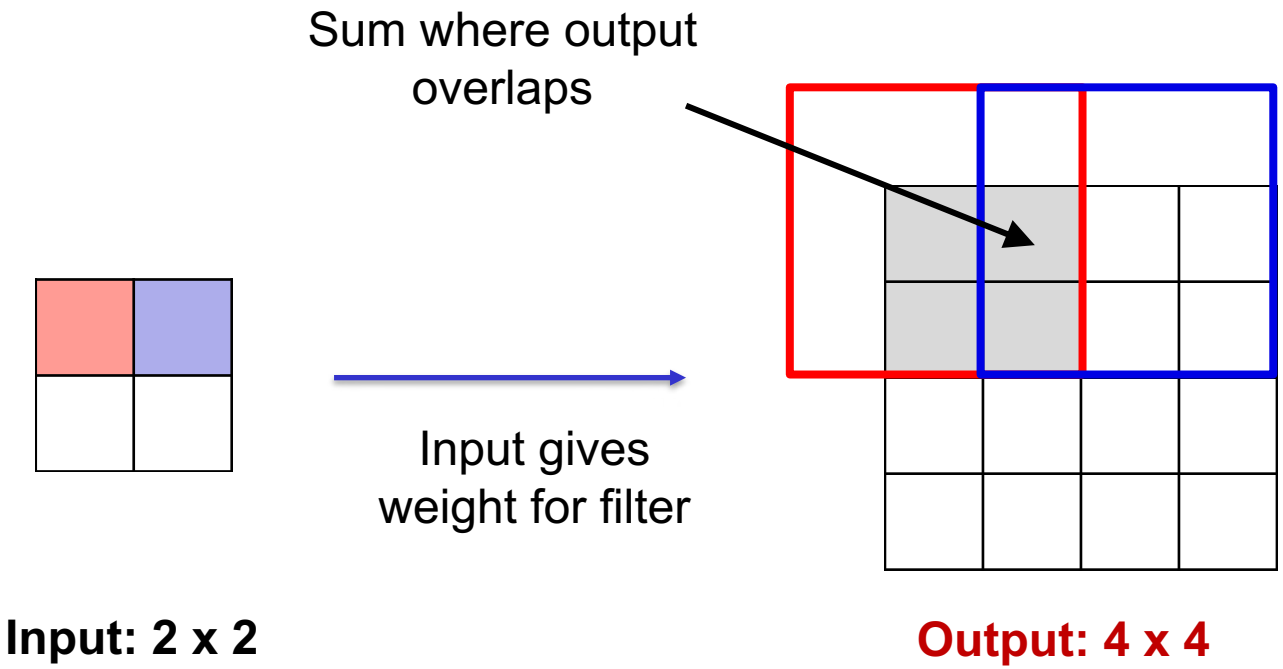


Input: 2 x 2

Output: 4 x 4

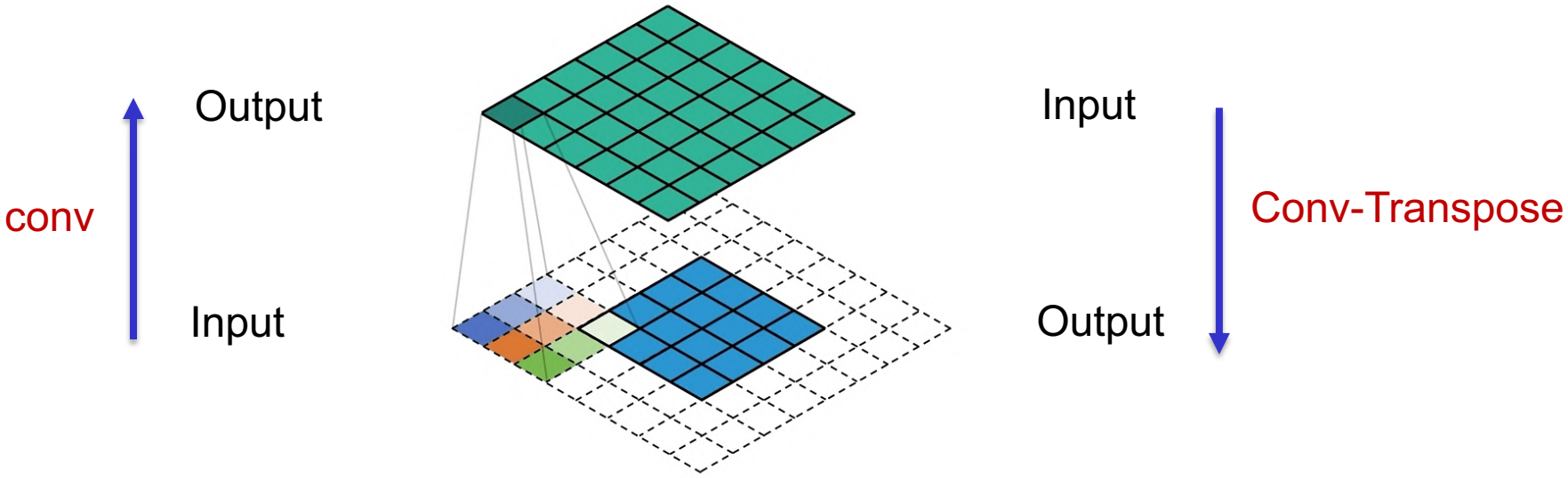
Learnable Upsampling: Conv-transpose

3 x 3 **conv-transpose**, **stride 2** pad 1



- Conv-transpose is similar to the back-prop pass for normal convolution
- The pack-prop operation of conv-transpose is the normal conv.

Learnable Upsampling: Conv-transpose



Source gif images

Learnable Upsampling: Conv-transpose

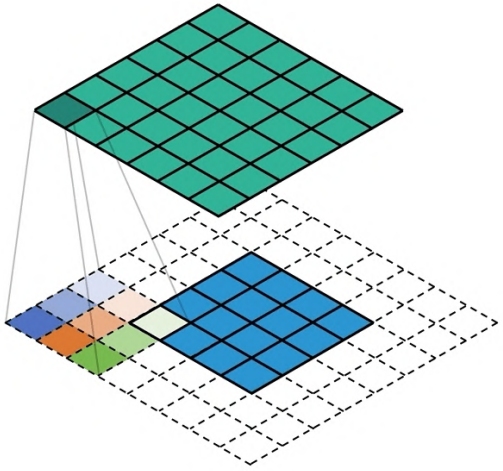
- In regular convolution:

$$outputSize = \frac{inputSize - F + 2P}{stride} + 1$$

- In conv-Transpose:

$$inputSize = \frac{OutputSize - F + 2P}{stride} + 1$$

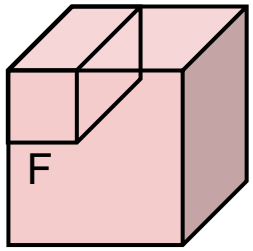
$$outputSize = (inputSize - 1) * stride + F - 2P + output\_padding$$



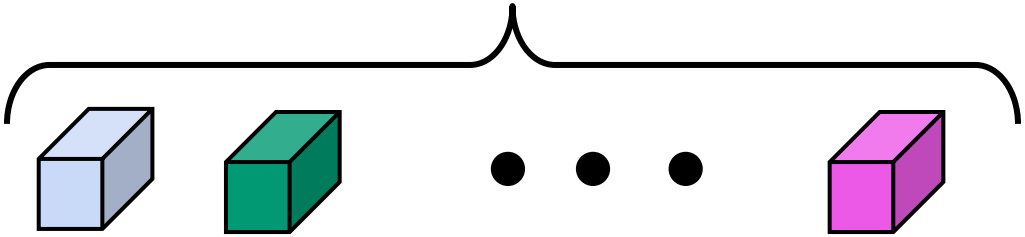
Implementing Convolutions: im2col

Efficient implementation of 2D convolution using GPU:
Reshape all image neighborhoods into columns (im2col operation), do matrix-vector multiplication

Feature map: C x H x W

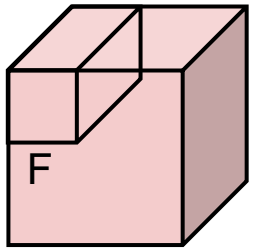


Conv weights: D filters, each C x F x F

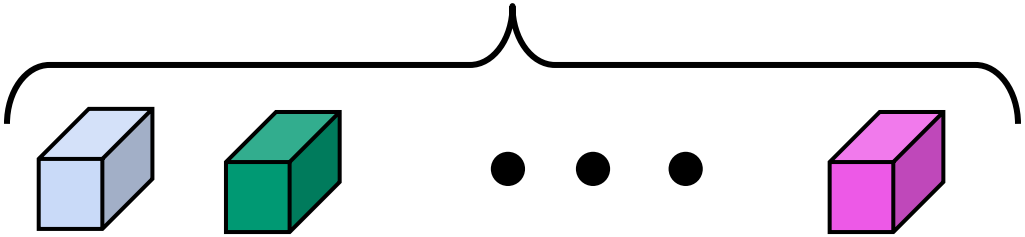


Implementing Convolutions: im2col

Feature map: C x H x W

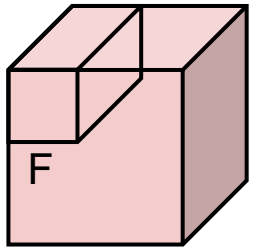


Conv weights: D filters, each C x F x F

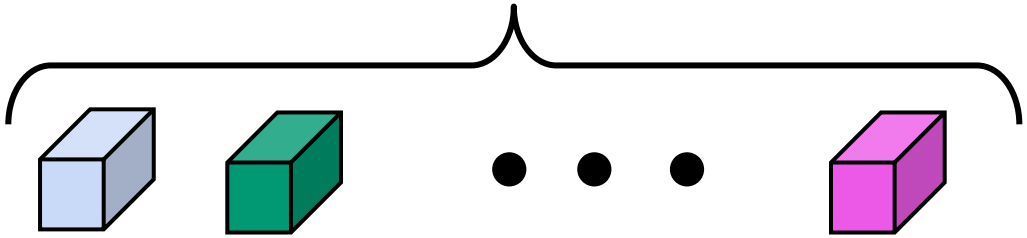


Implementing Convolutions: im2col

Feature map: C x H x W

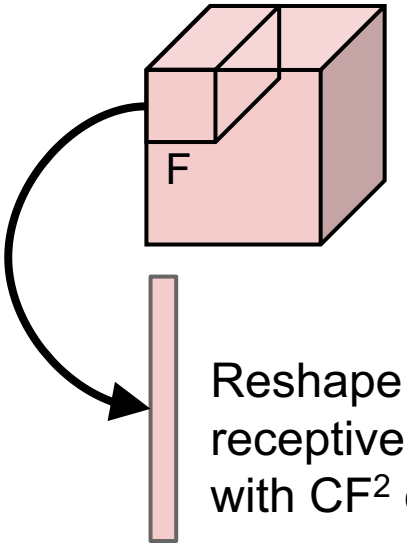


Conv weights: D filters, each C x F x F



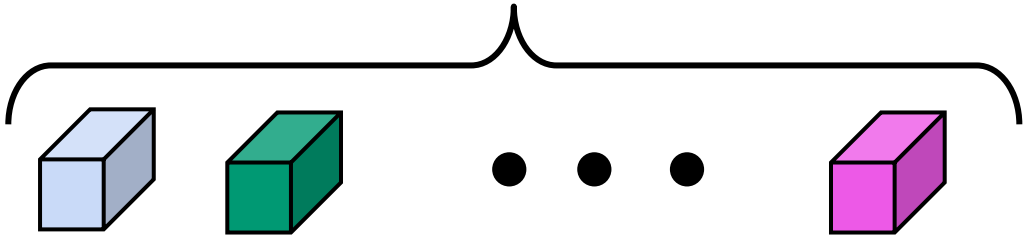
Implementing Convolutions: im2col

Feature map: C x H x W



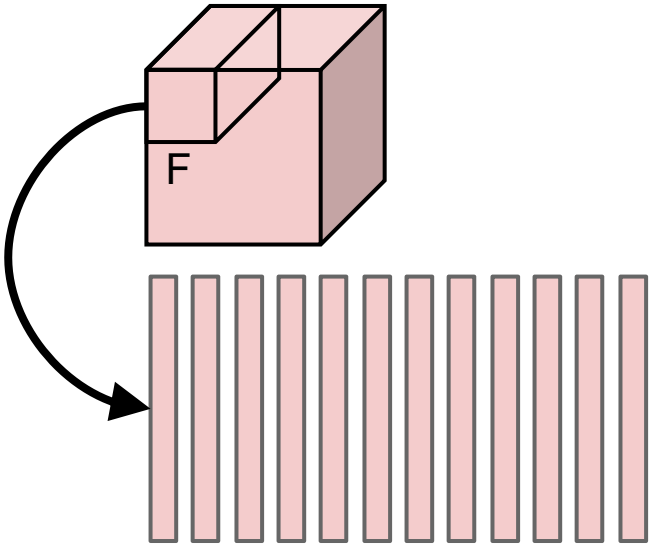
Reshape C x F x F receptive field to column with CF^2 elements

Conv weights: D filters, each C x F x F

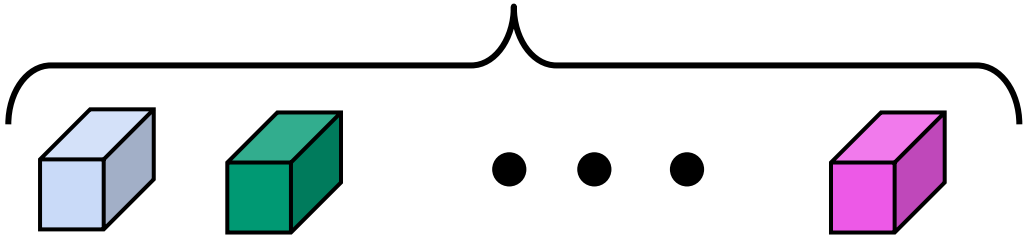


Implementing Convolutions: im2col

Feature map: C x H x W



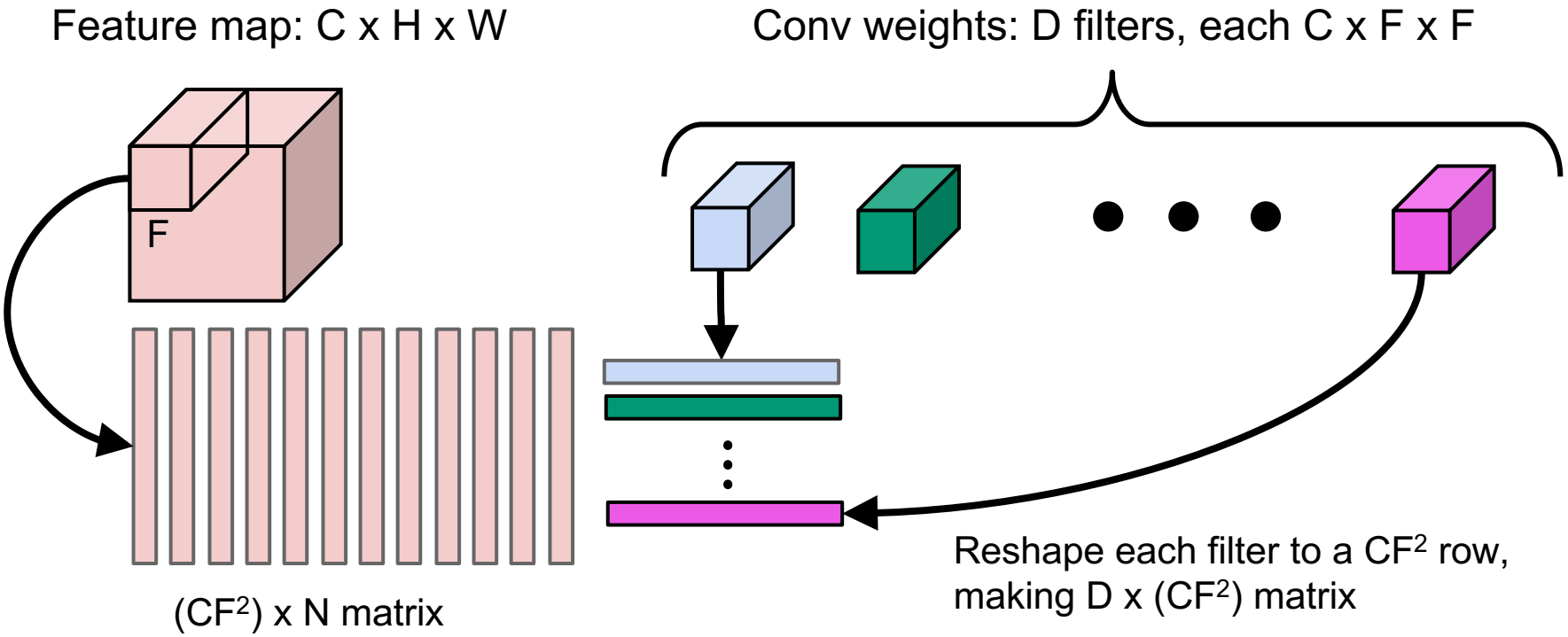
Conv weights: D filters, each C x F x F



Repeat for all columns to get (CF^2) x N matrix (N receptive field locations)

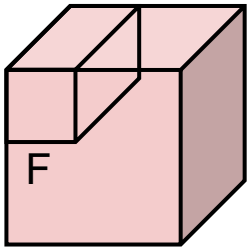
Elements appearing in overlapping receptive fields are duplicated; this uses a lot of memory

Implementing Convolutions: im2col

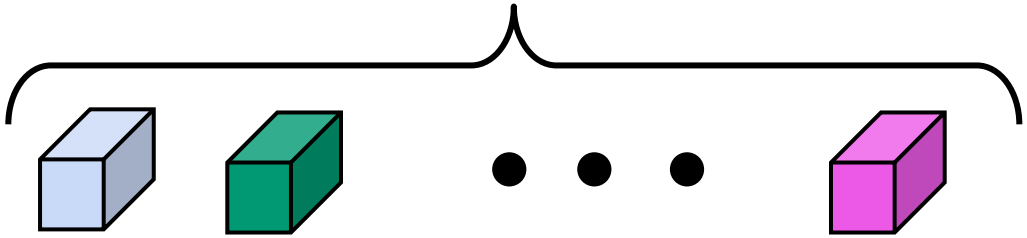


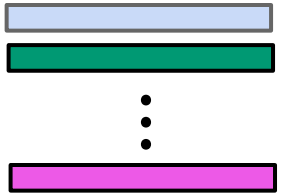
Implementing Convolutions: im2col

Feature map: C x H x W

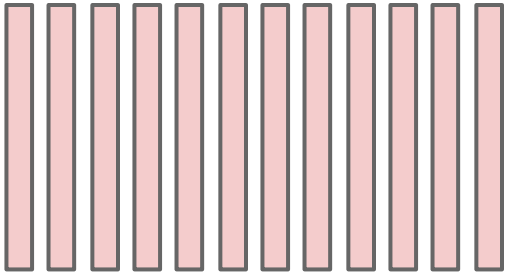


Conv weights: D filters, each C x F x F





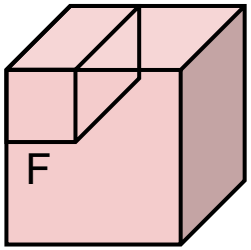
D x (CF^2) matrix



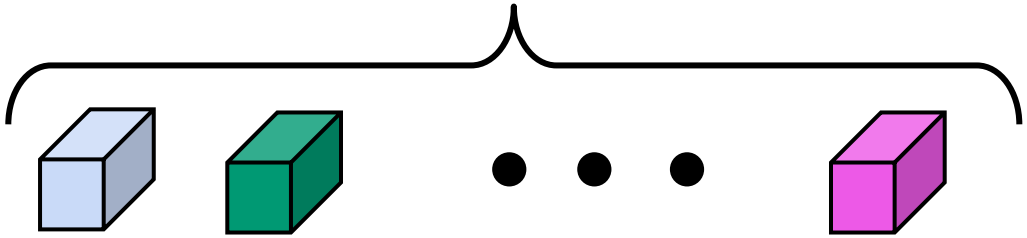
(CF^2) x N matrix

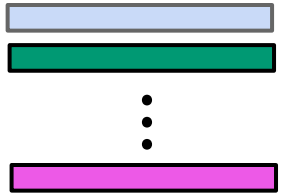
Implementing Convolutions: im2col

Feature map: C x H x W

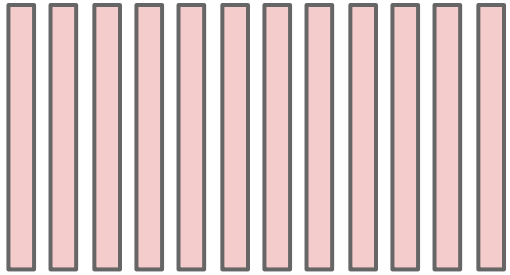


Conv weights: D filters, each C x F x F



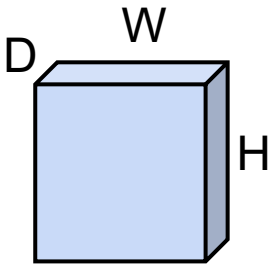


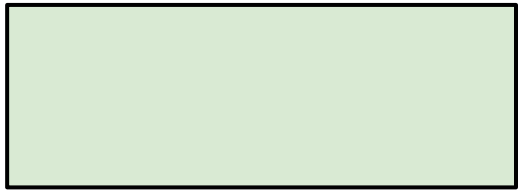
D x (CF^2) matrix



(CF^2) x N matrix

Matrix multiply

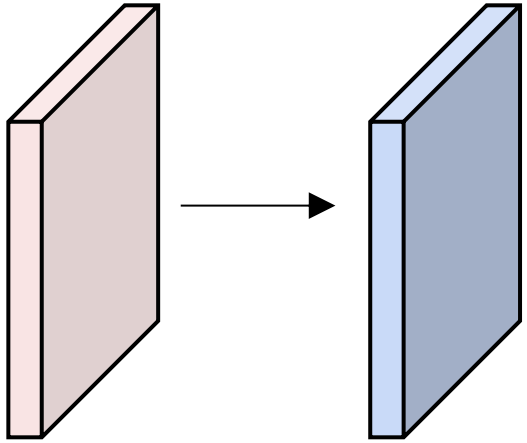




D x N result;
reshape to output tensor

Convolutional layer: # parameters

- Assume we have a **32x32 RGB image** and **10 filters** of size 5x5x3 each.



- What will be the size of resulting image with **stride=1** and **padding=2**?
- 32 spatially, so we end up with 32x32x10

$$\frac{32 - 5 + 2 * 2}{1} + 1 = 32$$

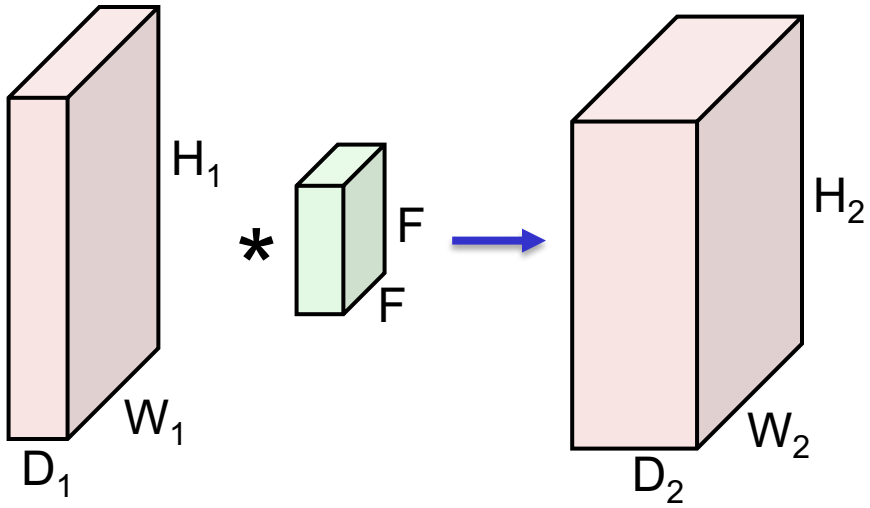
- Number of parameters in this layer?
- Each filter has 5*5*3+1=76 params ==> 76*10=760 params.
- How many mul-add operations? 32*32*10*(5*5*3+1)=778,240 ops.
- What will be the number of params for a FC layer?
- $(32^2 * 3) * (32^2 * 10) + 32^2 * 10 \ = 31{,}467{,}520$

Source: Stanford CS231

Convolutional layer: Parameters

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.



Source: Stanford CS231

Convolutional layer: Parameters

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.

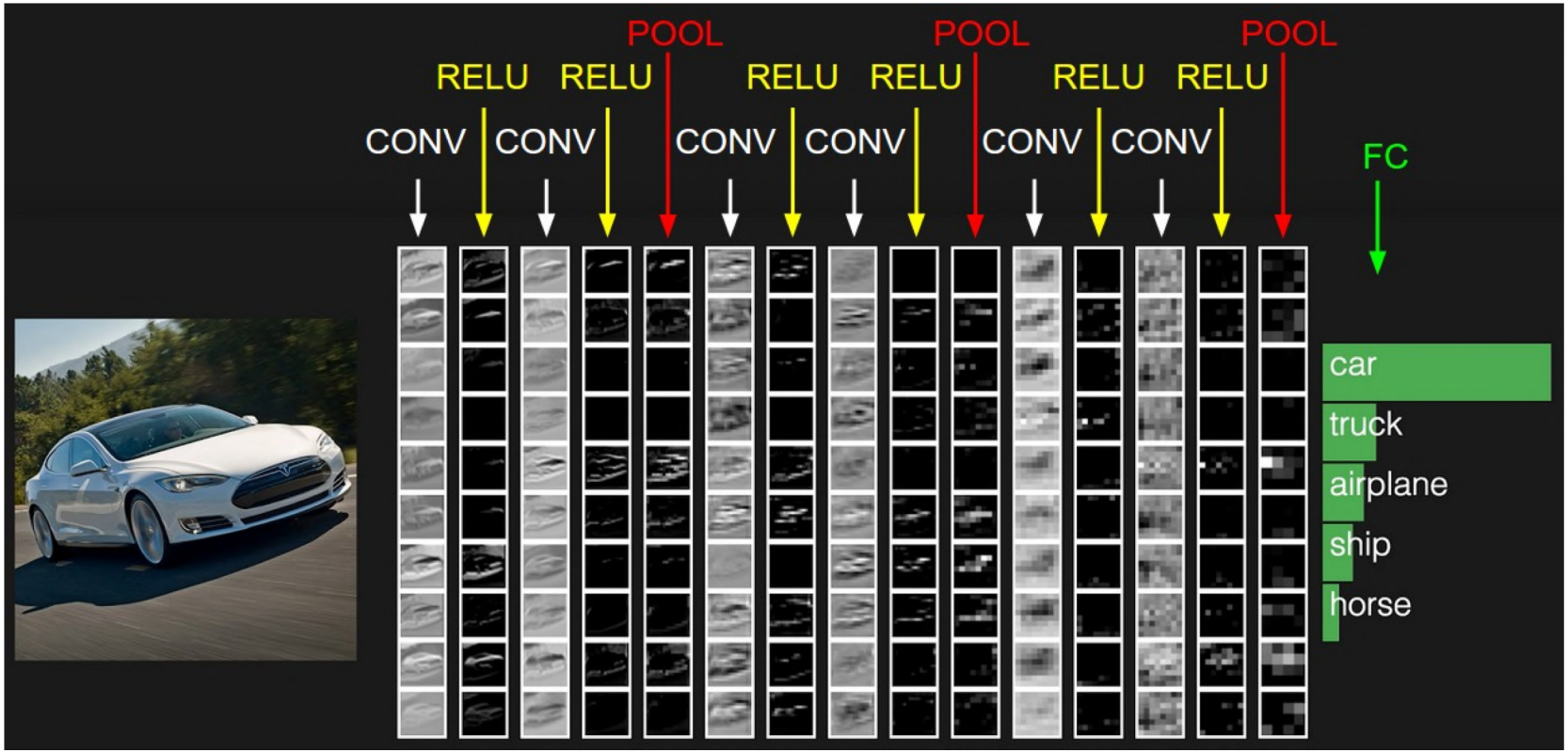
Typical settings:

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

Source: Stanford CS231

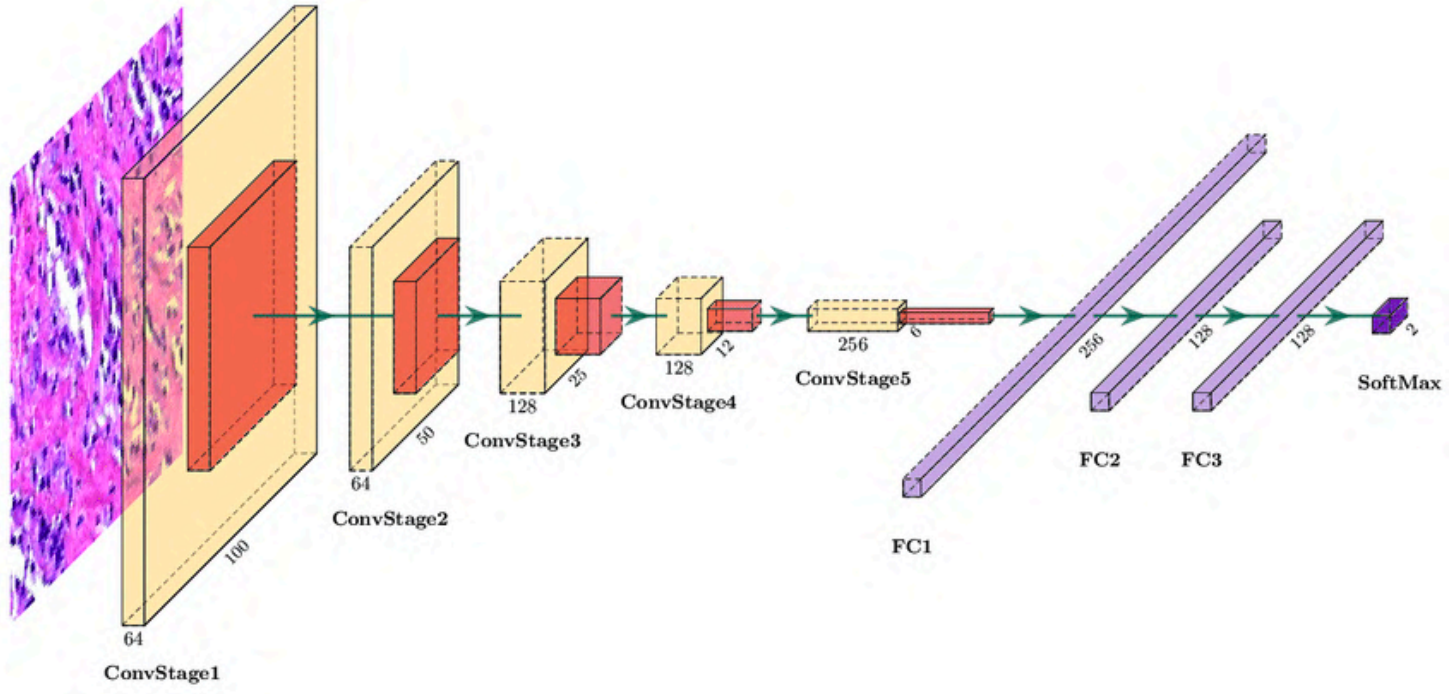
ConvNet: Hierarchy of layers

- ConvNets composed of a cascade of conv layers + pooling layers
- Commonly, the last layers are fully connected.



ConvNet: Hierarchy of layers

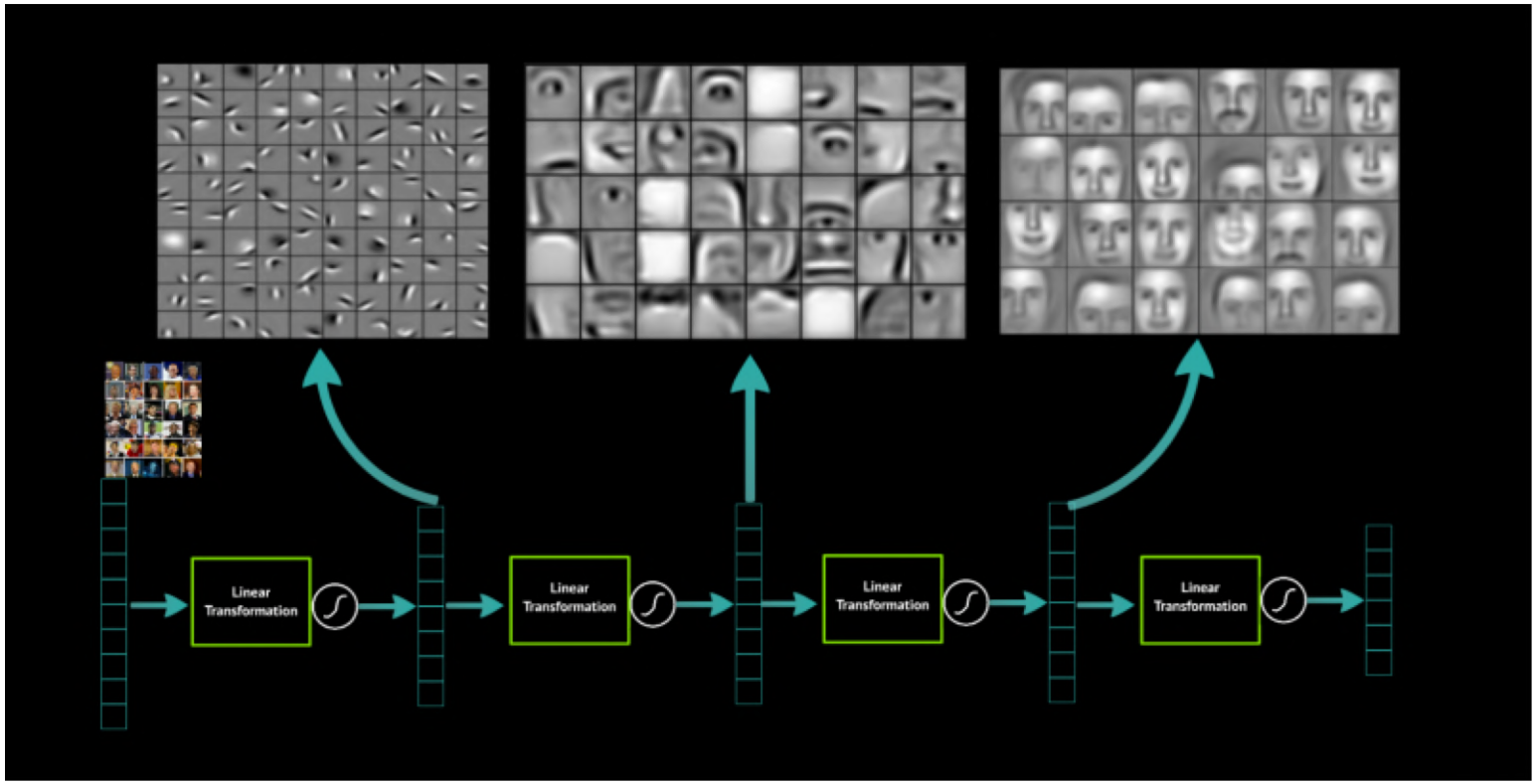
- Typically, as we go through the network:
  - Spatial sizes decreases (using pooling of strided conv.)
  - Number of channels increases.
- The CNN is terminated with few FC layers.



CNN architecture

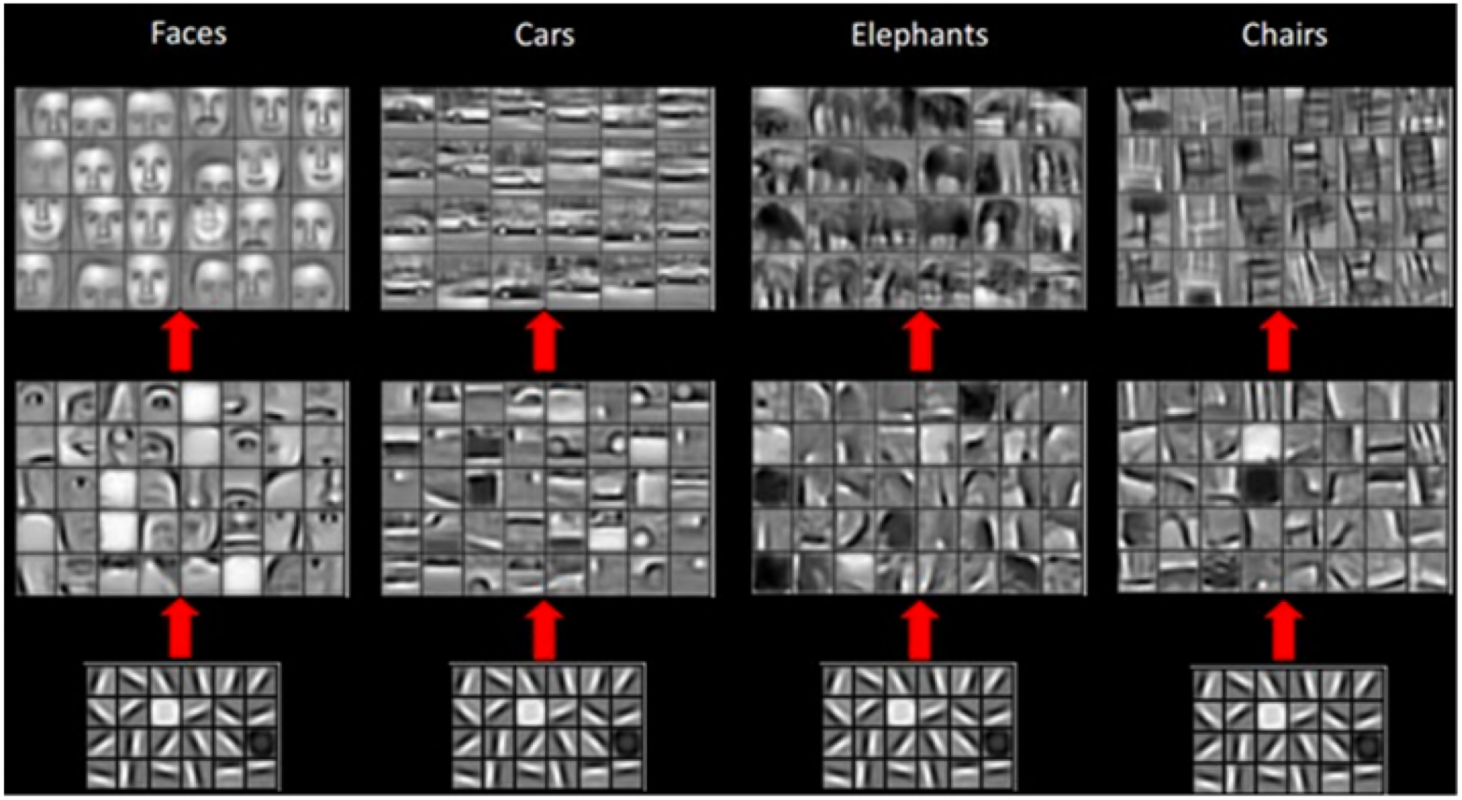
Convolution layers for feature extraction

- Resulting filters from multiple layers resemble cell responses in mammal's visual cortex.
- Lower features are more general while higher features are more specific.



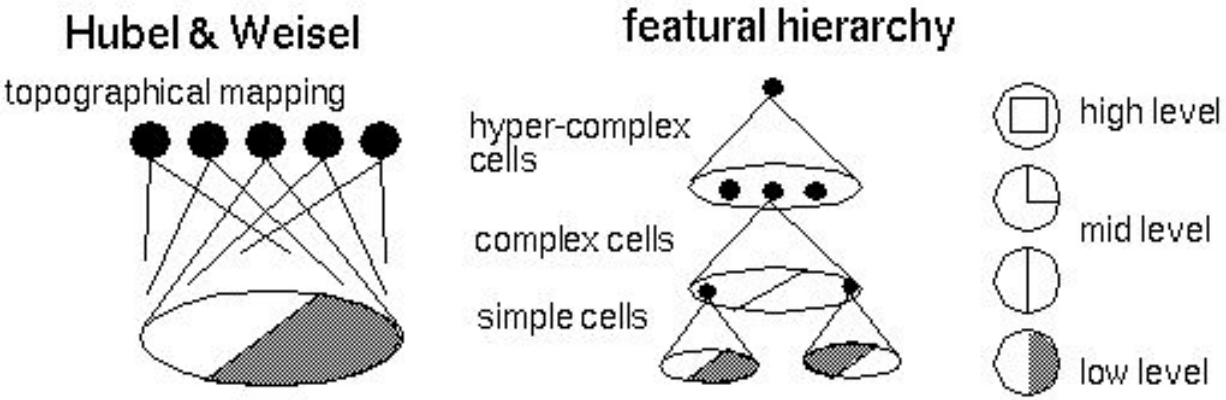
Convolution layers for feature extraction

- Resulting filters from multiple layers resemble cell responses in mammal's visual cortex.
- Lower features are more general while higher features are more specific.



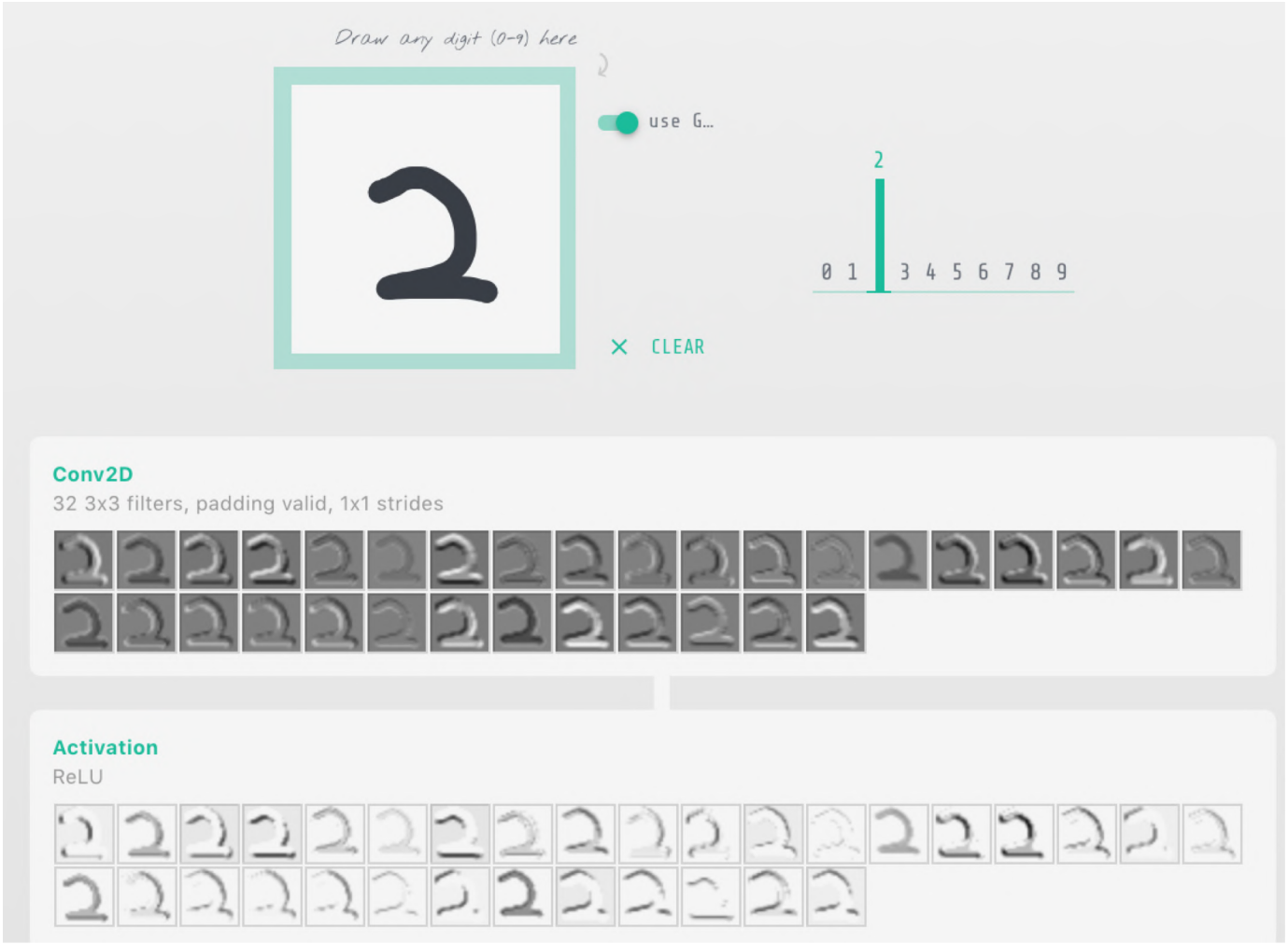
Inspiration: Biological visual system

- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
- Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells



Source

Demo: ConvNet for MNIST



Demo: https://transcranial.github.io/keras-js/#/mnist-cnn

THE END