Computer vision and imaging

From *classical* to *deep learning-based* methods in Computer Vision

Week 5, Lecture 2

Aleš Leonardis

Office: UG 34

Outline

- From *classical* to machine **(*deep) learning-based*** **computer vision** approaches
- Make step-by-step progression towards an end-to-end deep neural network (convolutional network)
  - Design principles
  - Design choices
  - Advantages
  - Disadvantages
  - Conditions under which can be applied
- **Compare the two approaches**

Edge/Contour/Feature detection

- "Manual" design requires proficiency in various techniques (e.g., mathematical modelling, various types of optimisation, etc.).
- Researchers are required to develop a variety of custom methodologies

- Hand designed and
  Hand designed with some ML components
  - Design principles (mathematical models)
  - Design choices (edge models, corners, textons, noise models, first, second order derivatives, etc.; parameter settings)
  - Advantages (compact, robust and stable under assumed conditions, interpretable and quick to evaluate and train (if at all); low on data requirements, low on compute requirements)
  - Disadvantages (manually designed features; performance is lacking)
  - Conditions under which can be applied (clear understanding of the problem and modelling, assumptions)
- Let us remind ourselves: What was leading to better performance?

Better performance - Lessons learned

- Observations based on the performance evaluation graphs
  - Combining several different filters is beneficial
  - Using different modalities can further improve the score

- Machine learning can help (facilitated by the annotated datasets)
  - Learn how to select the best combination of different filters
  - Learn how to combine different filters across modalities

- What is missing?
  - **Learning features automatically**
  - **Employing highly nonlinear models to combine them**

Edge/Contour detection – progression

- Next step: Learn detectors in an end-to-end fashion

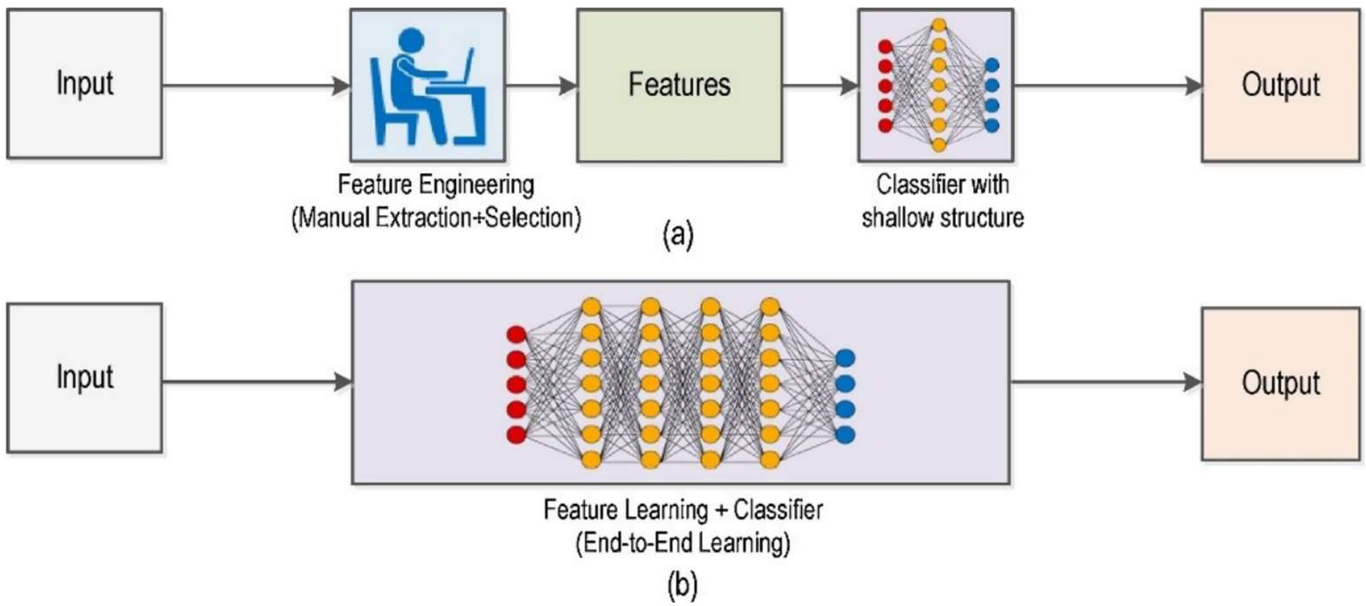


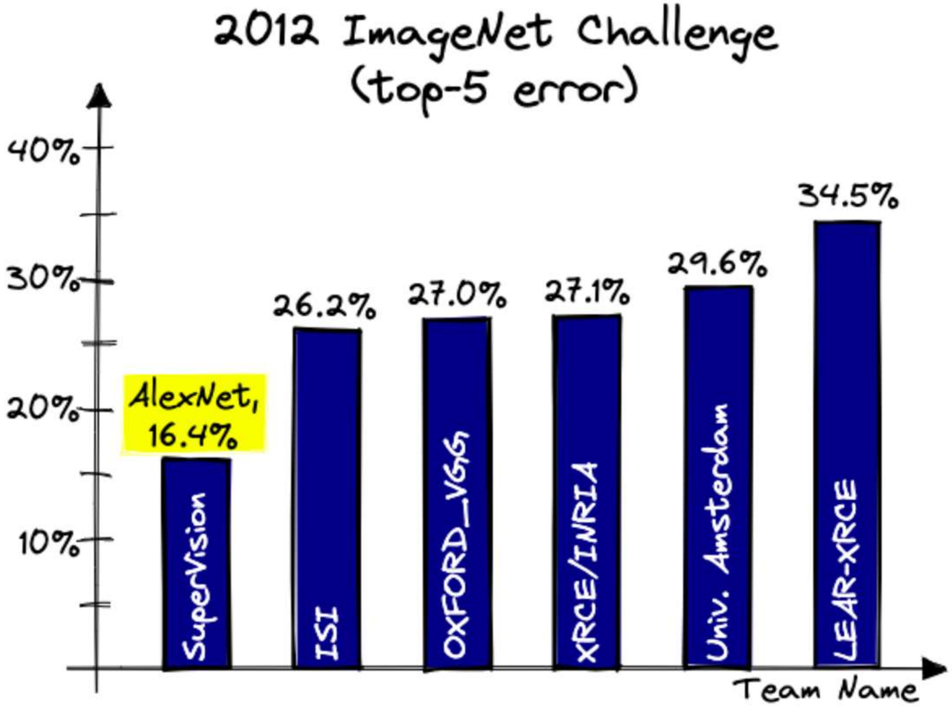
Fig. 1. (a) Traditional Computer Vision workflow vs. (b) Deep Learning workflow. Figure from [8].

Deep learning-based CV

- Neural networks are actually not new
- NN have been around (on and off) for quite some time (Perceptron was introduced in 1957), but the performance was mostly inferior.
- There were also theoretical considerations within the research community about their potential for success (inability to converge to a good solution, local minima, overfitting)
- Around 2012 deep learning paradigm proved its utility and strength (based on neural networks) – AlexNet
- Reasons: **Enough labelled data** and **sufficient computational power**

AlexNet (2012)

- Demonstrated excellent performance on the largest image dataset of the time, ImageNet
- **I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge (ILSVRC) 1000 label classification
- ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories.
- 1.2 million training images, 150,000 testing images.



Results of ILSVRC 2012 [11].

Plan for the rest of the lecture

- Following the objectives
  - Learn a variety of filters/features (many!)
  - Combine them
  - In a nonlinear way
  - Build a model that can be optimised for a specific task
- Make step-by-step progression towards an end-to-end deep neural network (convolutional)

Learning to detect edges

Learn the weights of a filter/kernel based on the data

1	0	-1
1	0	-1
1	0	-1

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

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

w1	w2	w3
w4	w5	w6
w7	w8	w9

=

Instead of defining the weights of the kernels manually, let us define them as parameters and learn them

The underlying operation is still convolution

Learning to detect edges

- How we can learn the weights of a filter/kernel based on the data?
- Instead of defining the weights of the kernels manually, let as define them as parameters.
- How can we design a model than can later be trained by an ML algorithm, i.e., backpropagation?
- The optimisation process will be treating these weights as parameters and they will be optimised to produce a desirable output.
- This may result in a more robust performance (than what researchers can design). The underlying operation is still convolution.

Streamlining the process - Some prerequisites

To streamline the process we need to go over some prerequisites

- Padding
- Strided convolutions
- Volume convolutions
- One layer network
- Bias + nonlinearity
- A simple network
- Followed by Training (backpropagation – covered in later lectures by Jianbo)

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

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

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

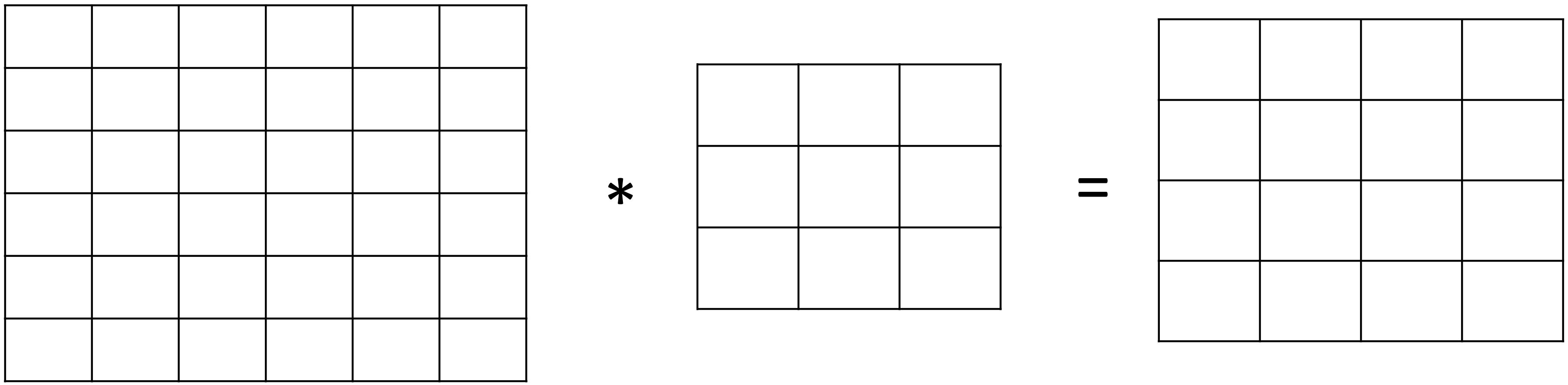
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

w1	w2	w3
w4	w5	w6
w7	w8	w9

=

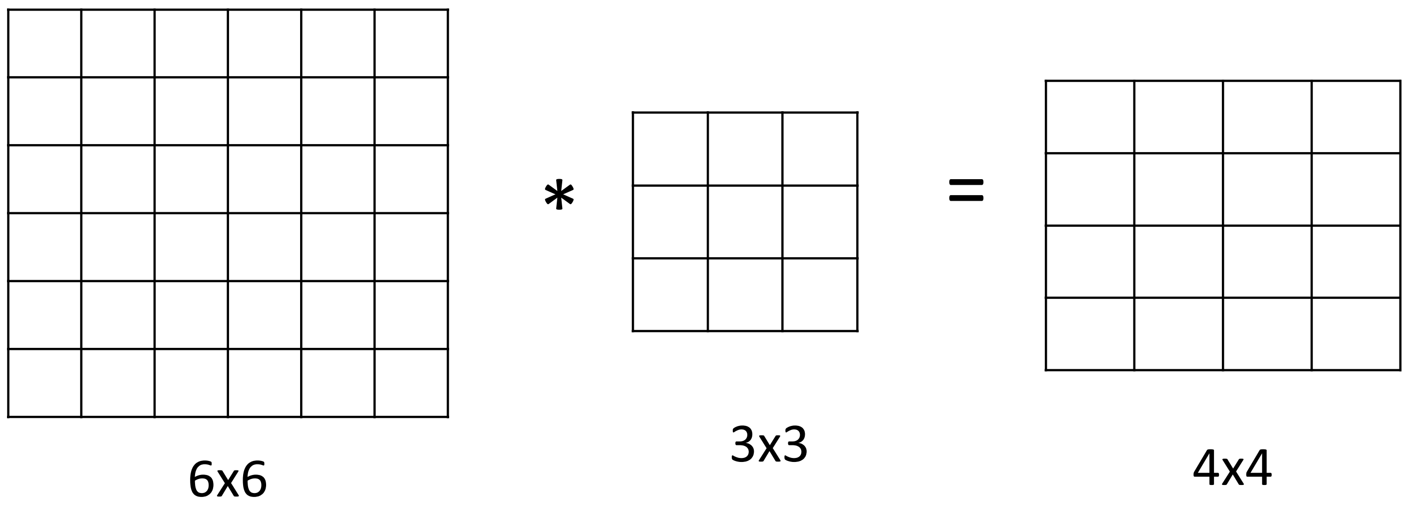
Padding



6x6 3x3 4x4

- In regular convolution operation, the size of the output decreases with respect to the input
- This may be undesirable if we repeatedly apply convolutions to the output of previous convolutions (e.g., multiple layers)
- Also, the pixels closer to the borders do not equally contribute to the output
- **Padding prevents that** (padding is extending the region around the image with additional pixels to preserve the original size of the output after operation

Padding



- By convention the padding is with zeros
- Padding can be one, two or more pixels

- [6x6] * [3x3] -> [4x4]
- Padding 1 -> [8x8] * [3x3] -> [6x6]

n – image size

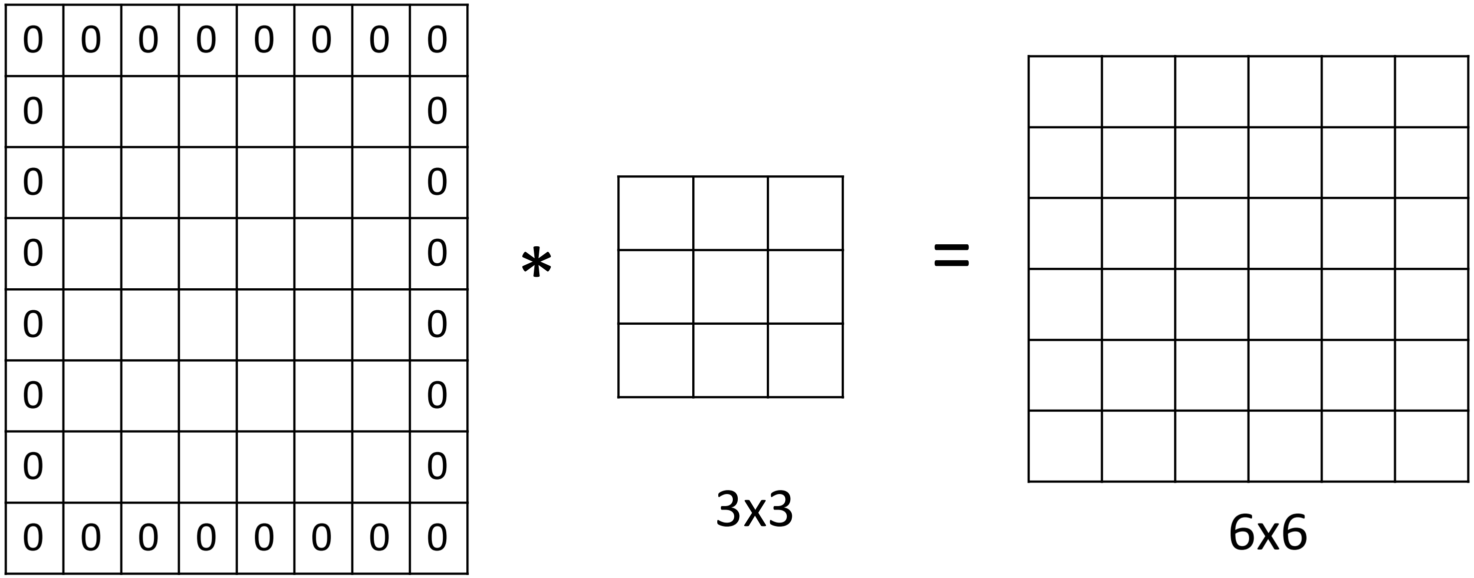
f – filter size

p - padding

o – output

o = [n+2p-f +1]x[n+2p-f +1]

Padding



3x3

6x6

(6+2)x(6+2)

- By convention the padding is with zeros
- Padding can be one, two or more pixels

- [6x6] * [3x3] -> [4x4]
- Padding 1 -> [8x8] * [3x3] -> [6x6]

n – image size

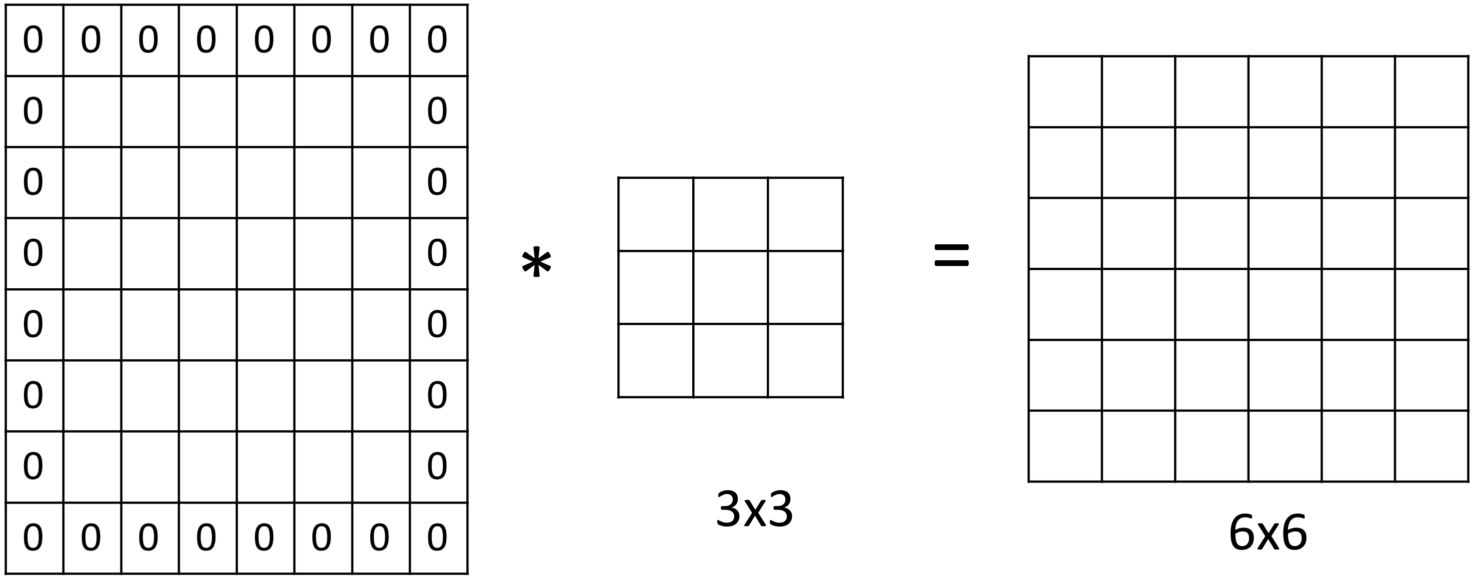
f – filter size

p - padding

o – output

o = [n+2p-f +1]x[n+2p-f +1]

Padding – Valid - Same



(6+2)x(6+2)

- By convention the padding is with zeros
- Padding can be one, two or more pixels
- **Valid** (no padding) **[n-f +1]x[n-f +1]**
- **Same** (pad such that the input and output sizes are the same) **p=(f-1)/2**

n – image size

f – filter size

p - padding

o – output

o = [n+2p-f +1]x[n+2p-f +1]

Strided convolution

- Another building block towards building convolutional neural networks
- Perform convolutions with steps that may differ from step = 1

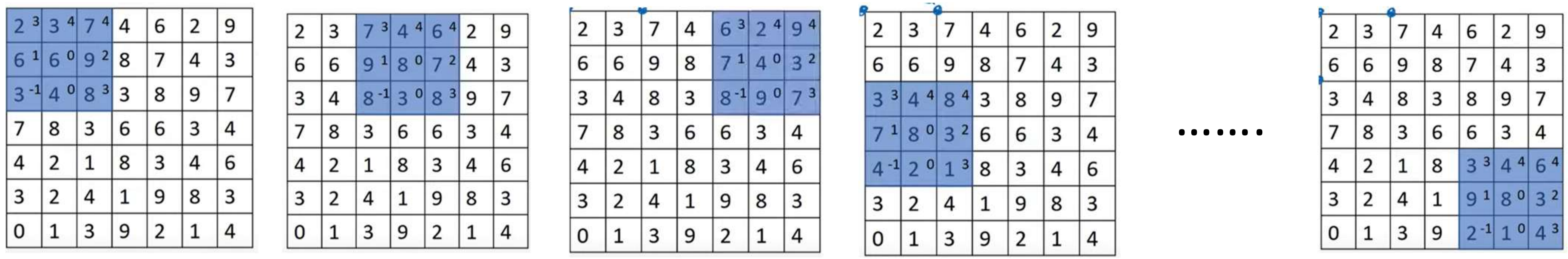
2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

*

3	4	4
1	0	2
-1	0	3

=

91	100	83
69	91	127
44	72	74



The input and output sizes governed by the formula **o=((n-f)/s +1) x (n-f)/2 +1);** Stride = 2; n=7, f=3, s=2;

Summary of convolutions (padding, stride)

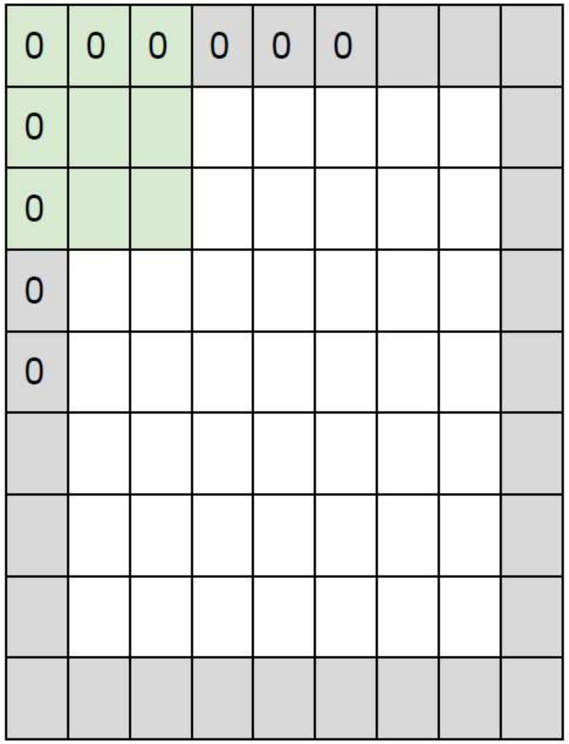
$n \times n$ image $\qquad f \times f$ filter

padding p $\qquad$ stride s

We usually pick the dimensions such that we do not need to discard parts of images.

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

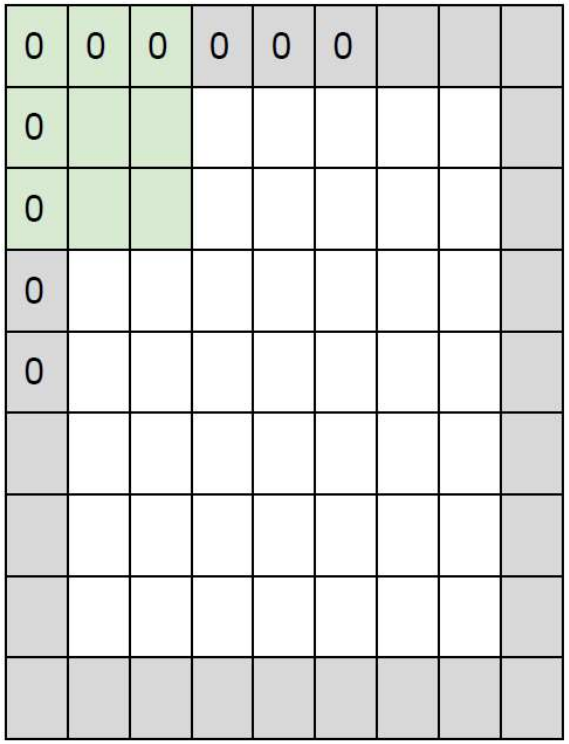
How to keep spatial size: padding



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

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

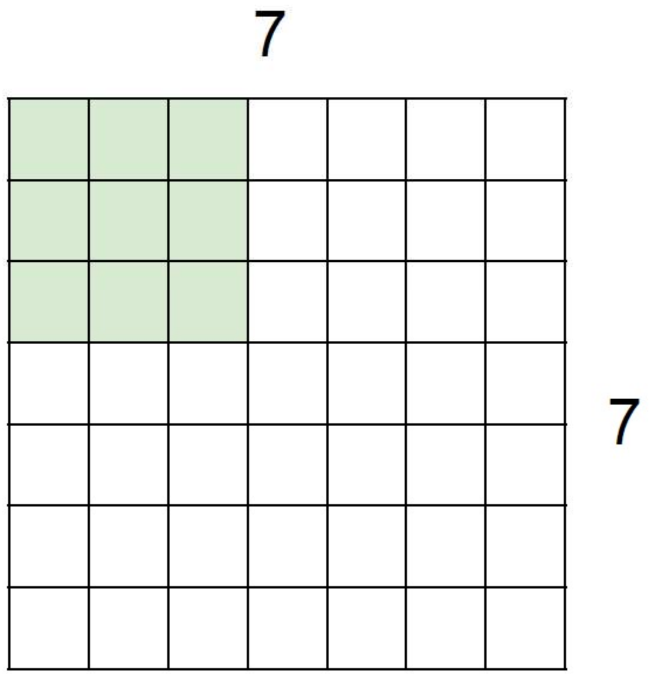
How to keep spatial size: padding



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

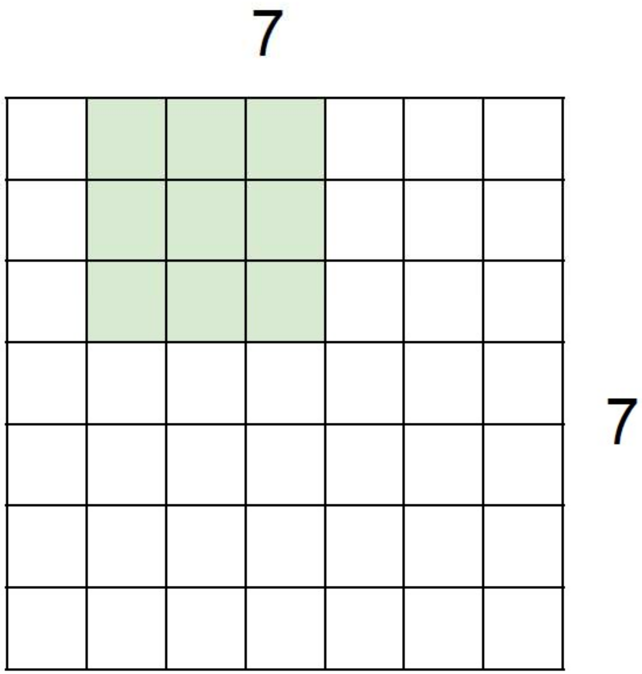
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

A closer look at spatial dimensions:



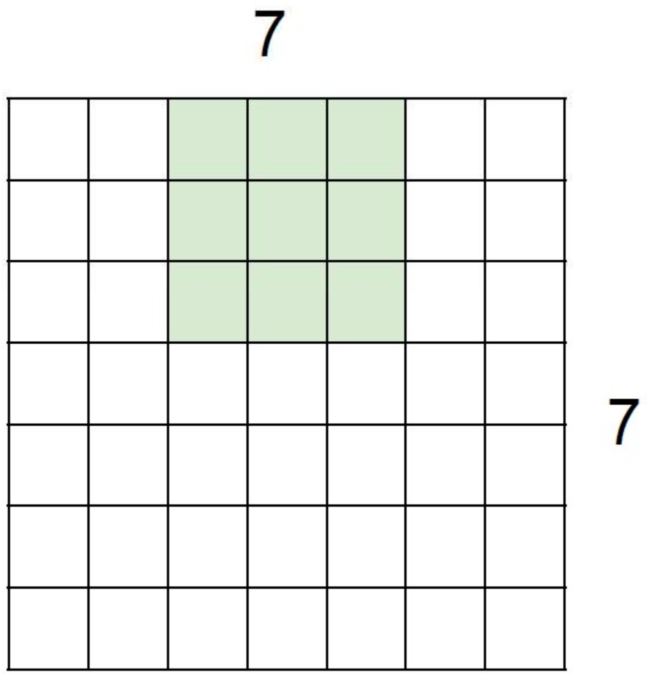
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



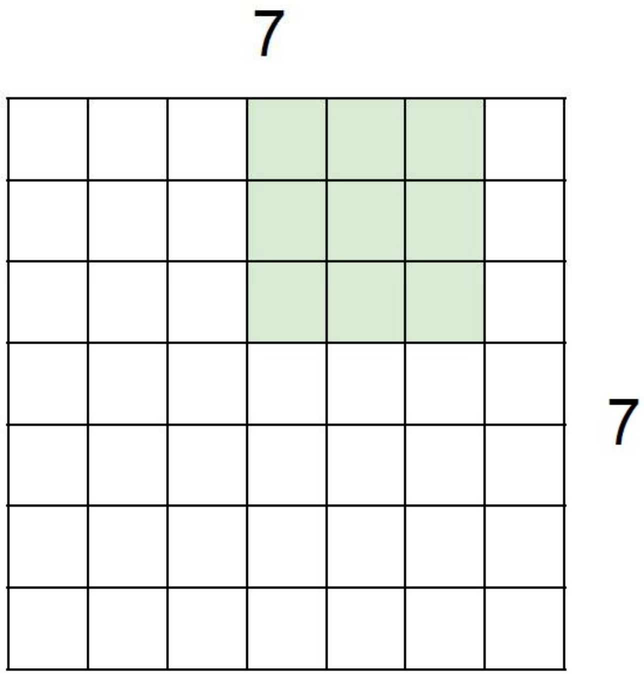
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



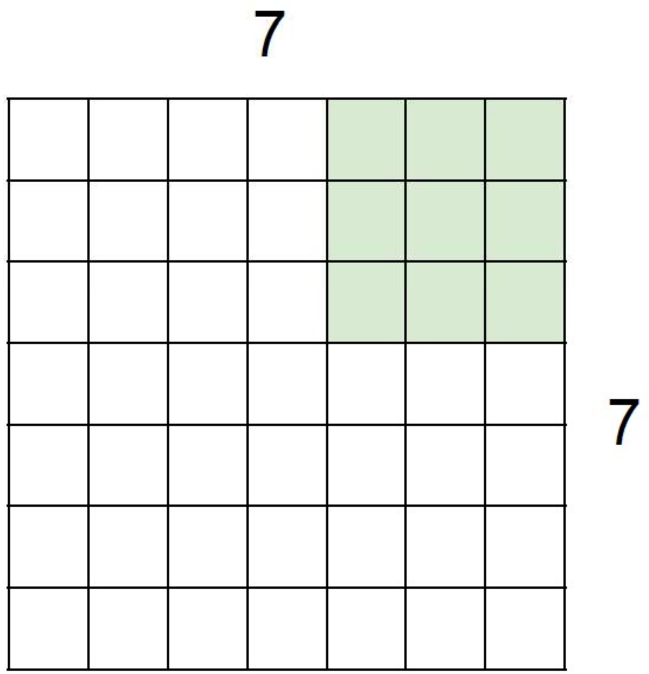
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

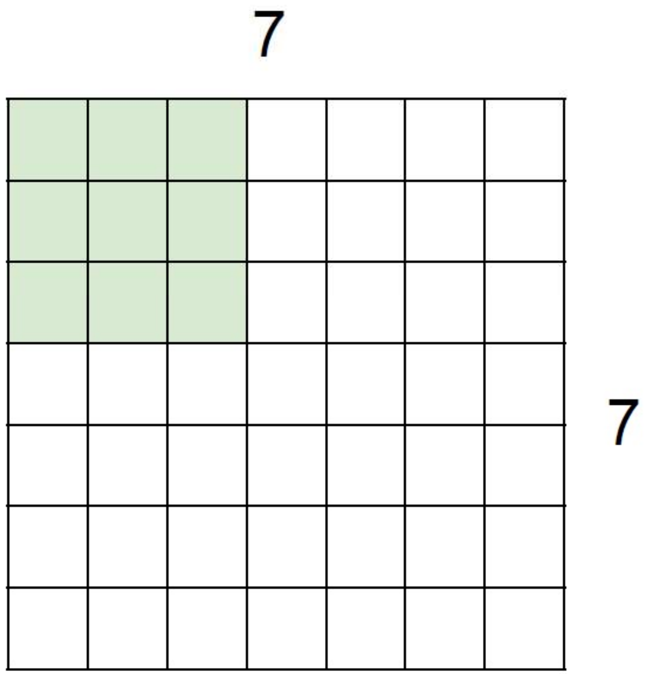
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

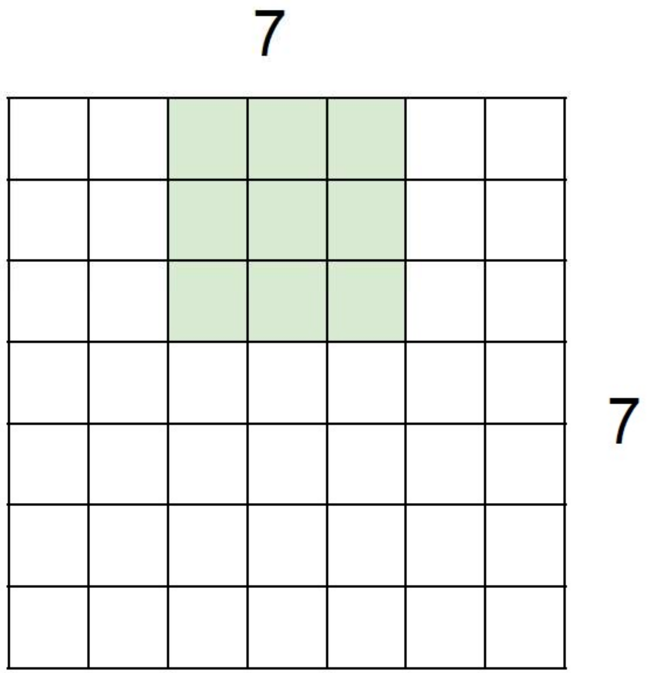
=> 5x5 output

A closer look at spatial dimensions:



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

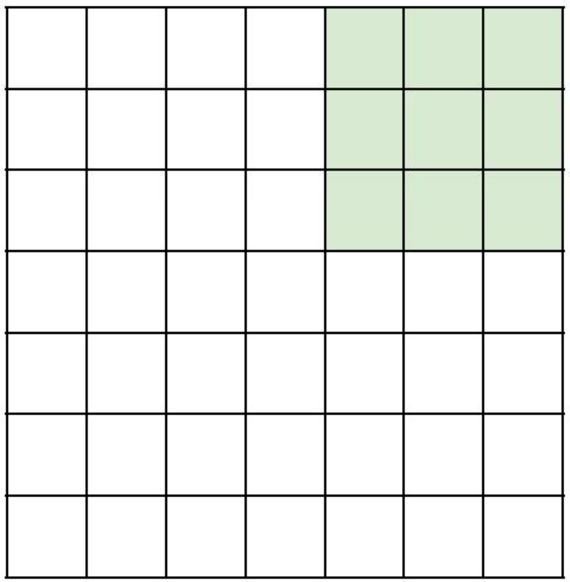
A closer look at spatial dimensions:



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

A closer look at spatial dimensions:

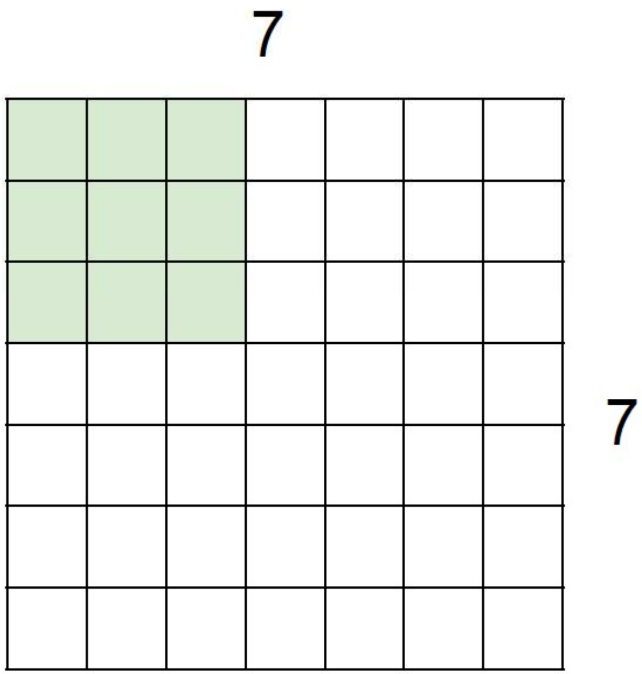
7



7

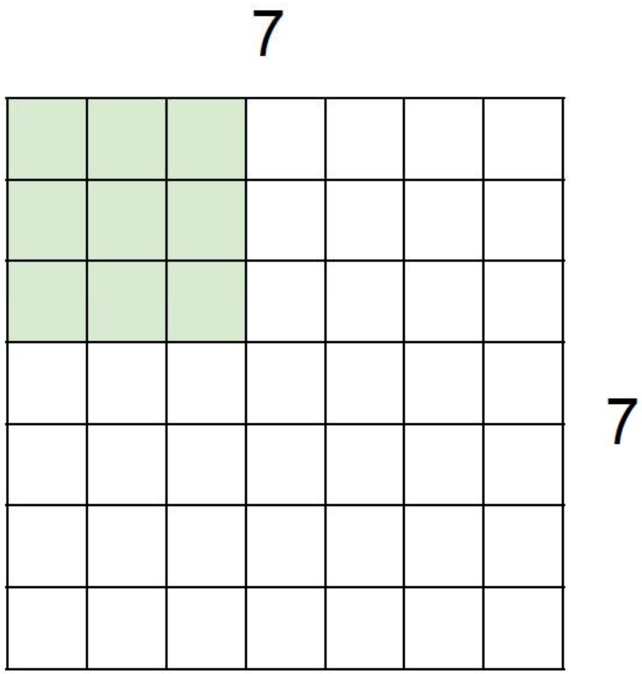
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

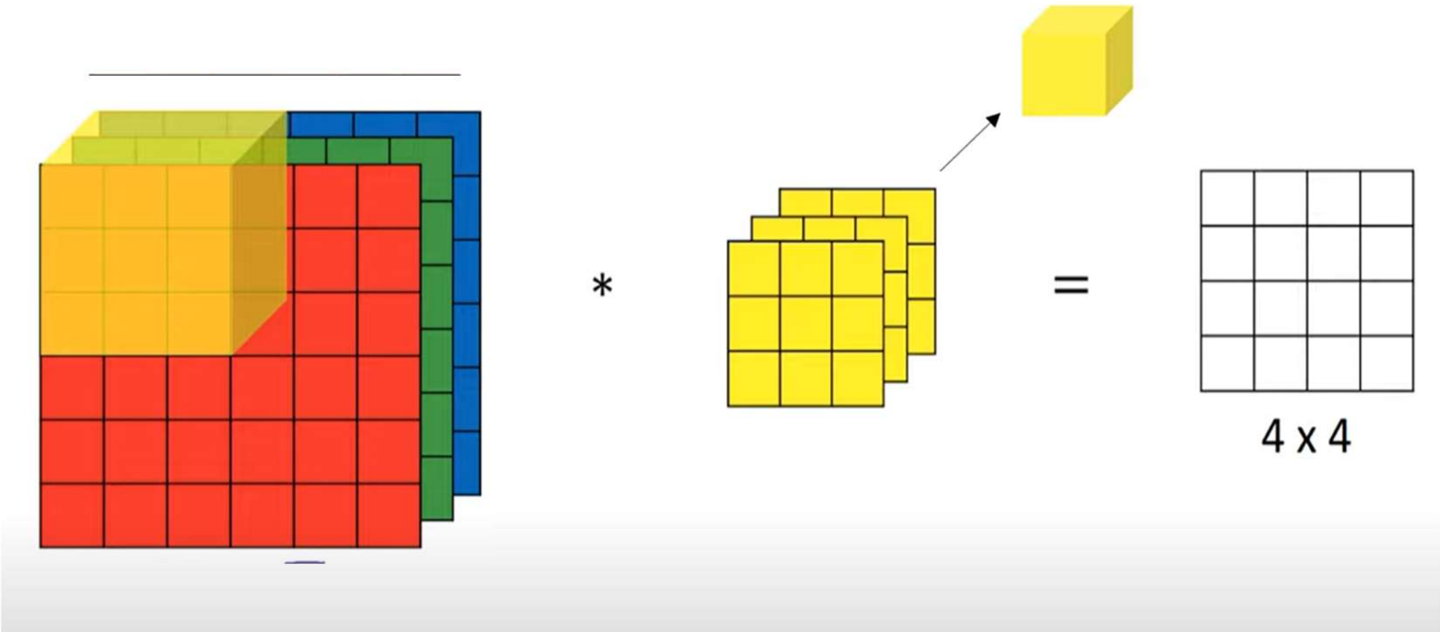
A closer look at spatial dimensions:



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

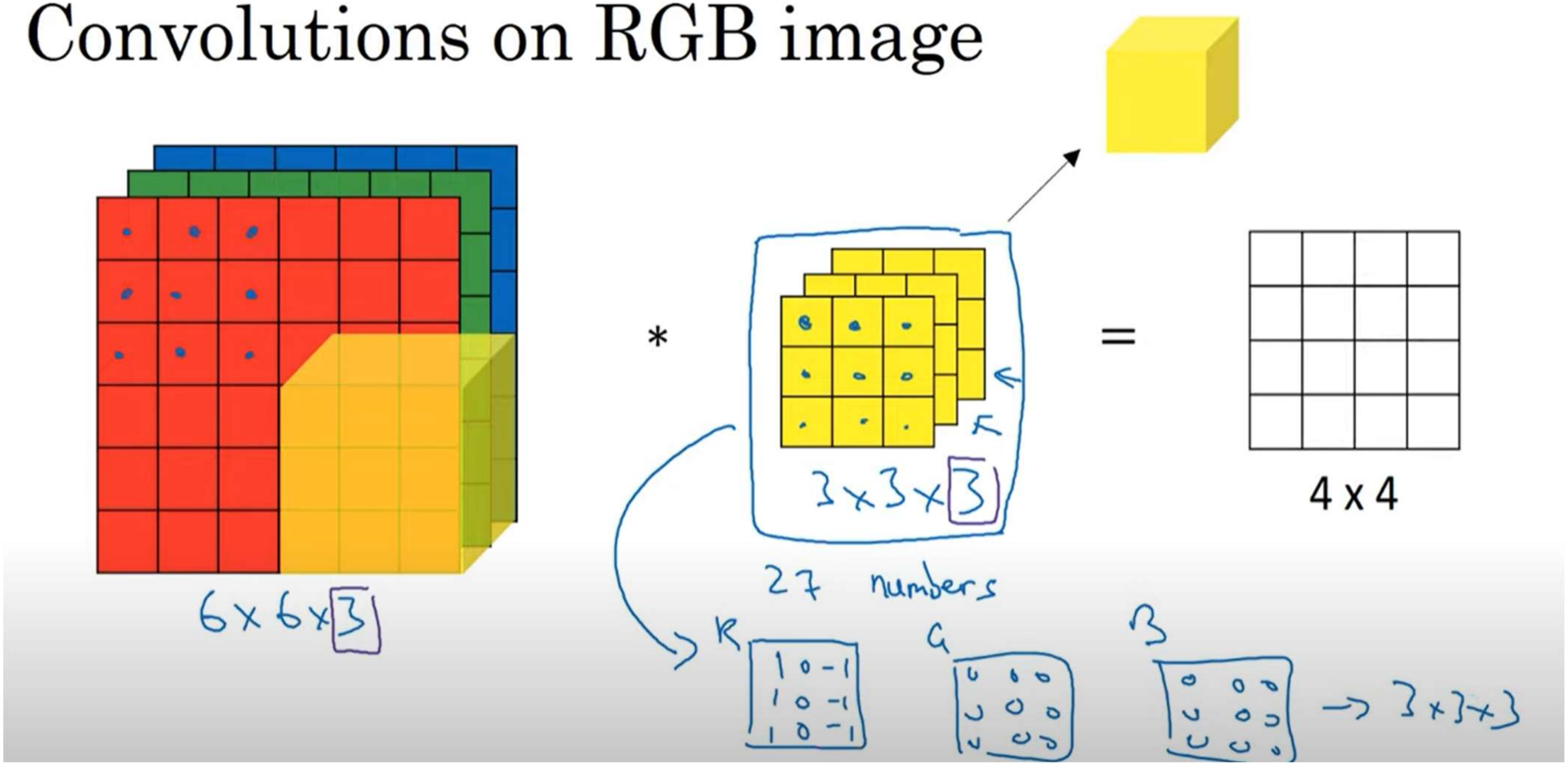
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolutions on RGB Image



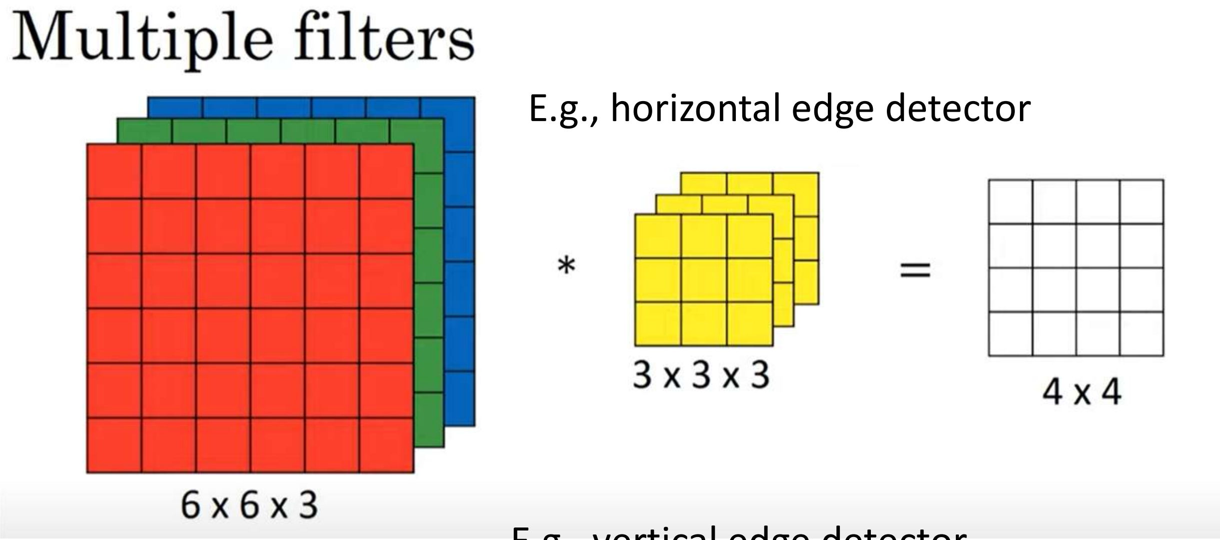
- We know how convolutions over 2D images are formulated
- Images are usually RGB; experiments showed how important colour is for downstream tasks
- Convolutions over 3D volumes. Image nxn x 3 colour channels
- The full filter will also have 3 channels. Height, width, number of channels.
- Filter – like a stack (3x3x3). The result is a 2D output.
- The number of channels (image, filter) MUST be equal.

Convolutions on RGB image



- Example: looking for only vertical edges in the red channel (how will the filter volume look like?)
- Different parameters/weights – different kernels – different features extracted from an RGB image
- Constructing the kernels manually would require careful considerations
- By convention, the image and the filter may have different height and width, but they have the SAME number of channels.

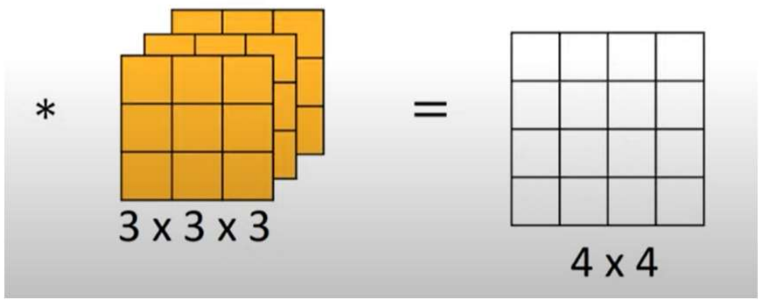
Convolutions over volumes – multiple features



E.g., horizontal edge detector

Horizontal edges

E.g., vertical edge detector



Vertical edges

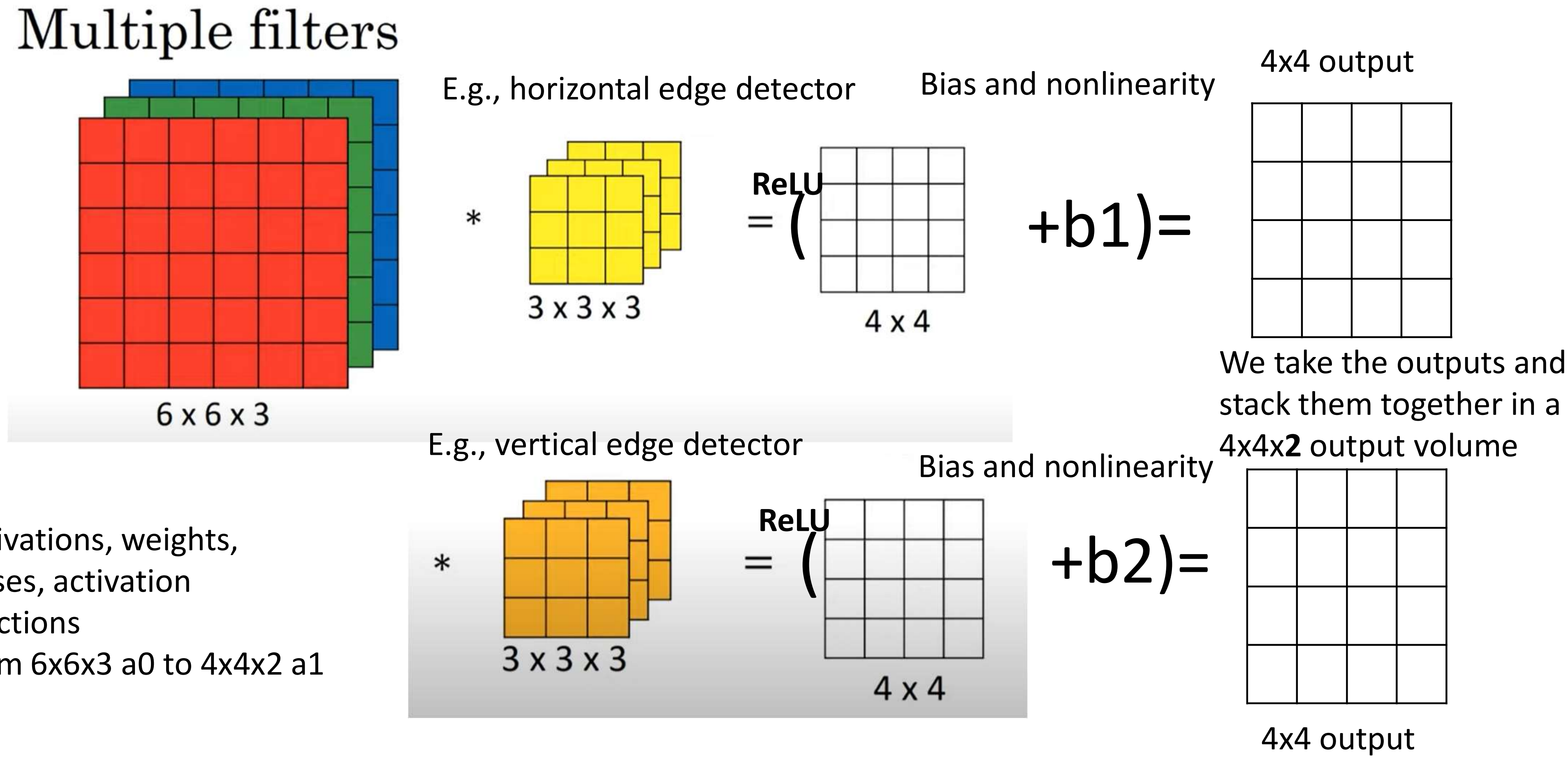
We take the outputs and stack them together in a 4x4x**2** output volume

Two channels come from using two different filters

(n-f+1)x(n-f+1) x no_of_filters
(stride=1; padding 0)

An arbitrary number of filters. Number of filters can be very large – an arbitrary number!

One layer of a convolutional network



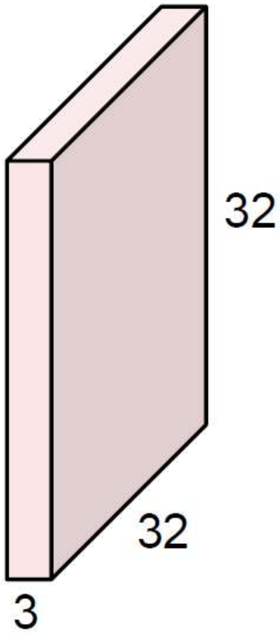
Activations, weights, biases, activation functions
From 6x6x3 a0 to 4x4x2 a1

Convolutions on RGB images

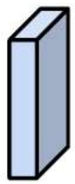
Convolution Layer

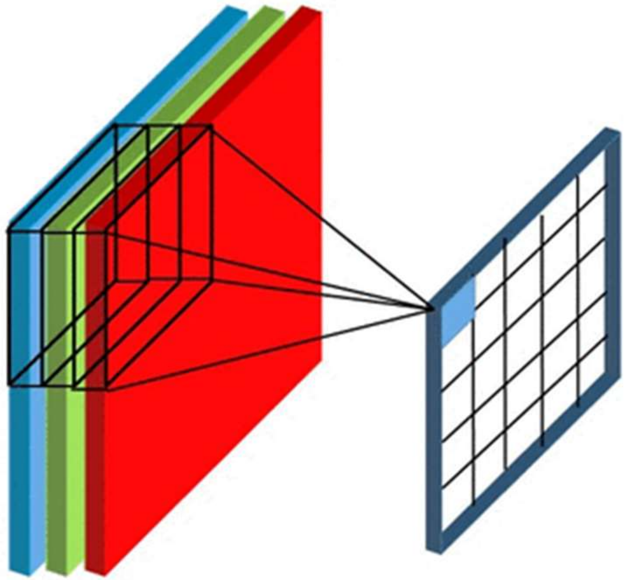
Filters always e
depth of the inp

32x32x3 image



5x5x3 filter



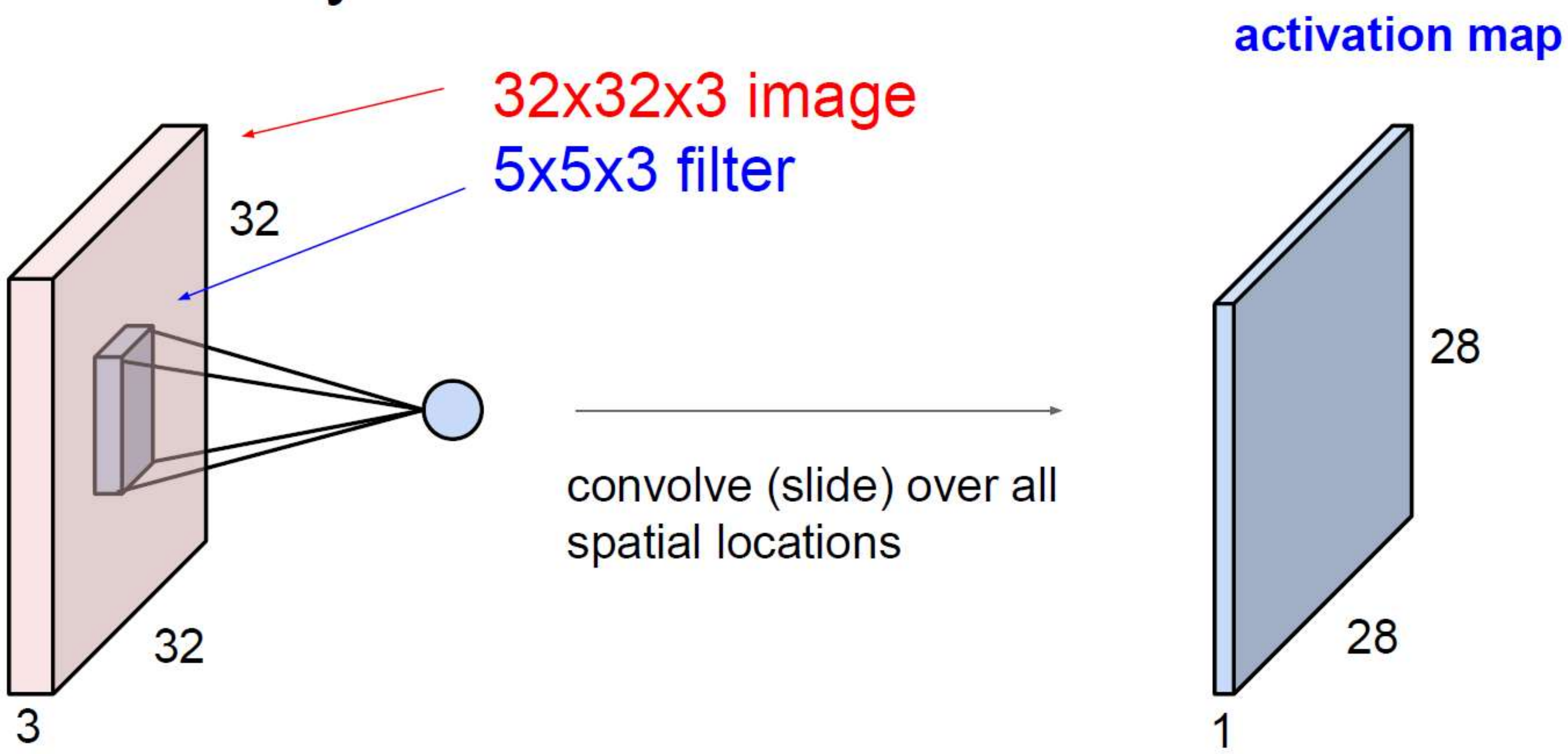


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolutions over colour images (height x width x channels);
Channels are often referred to as depth

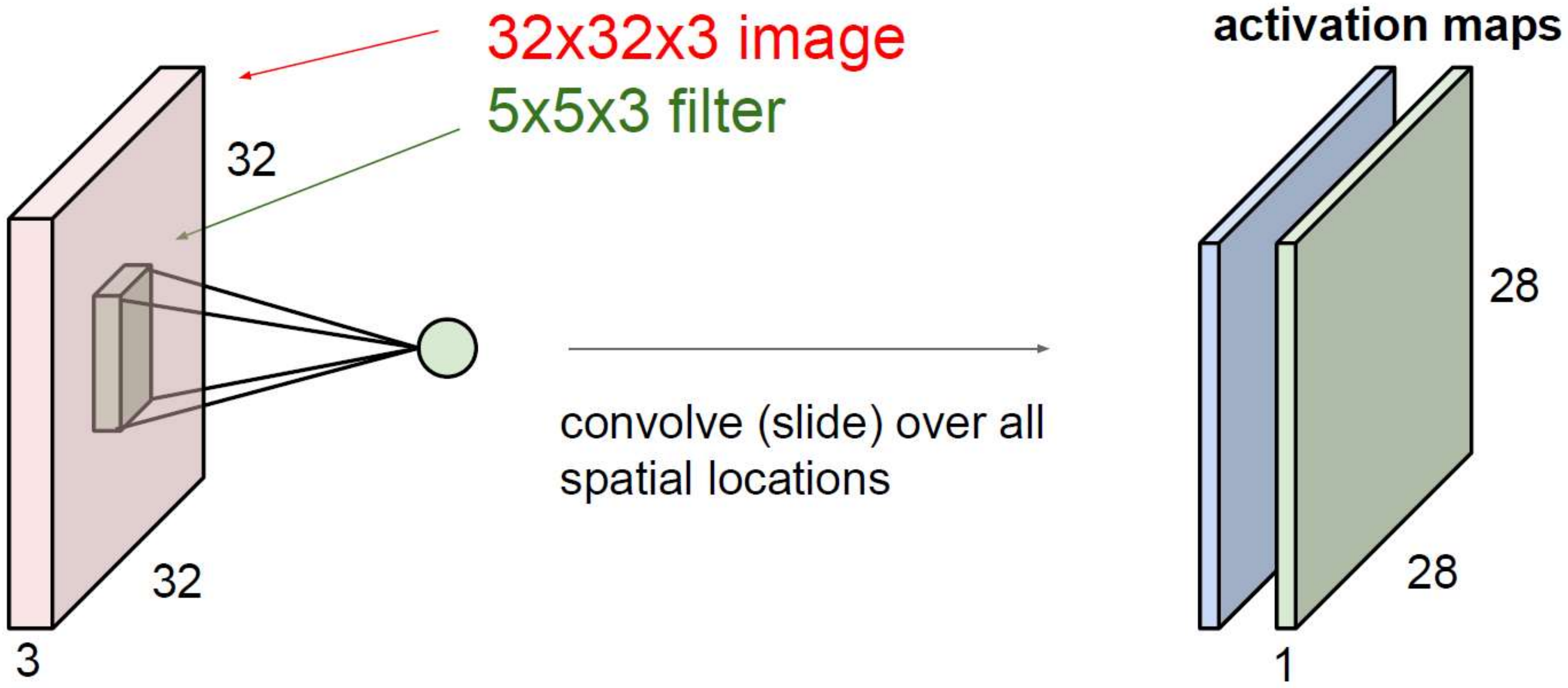
The output is a 2D image

Convolution Layer

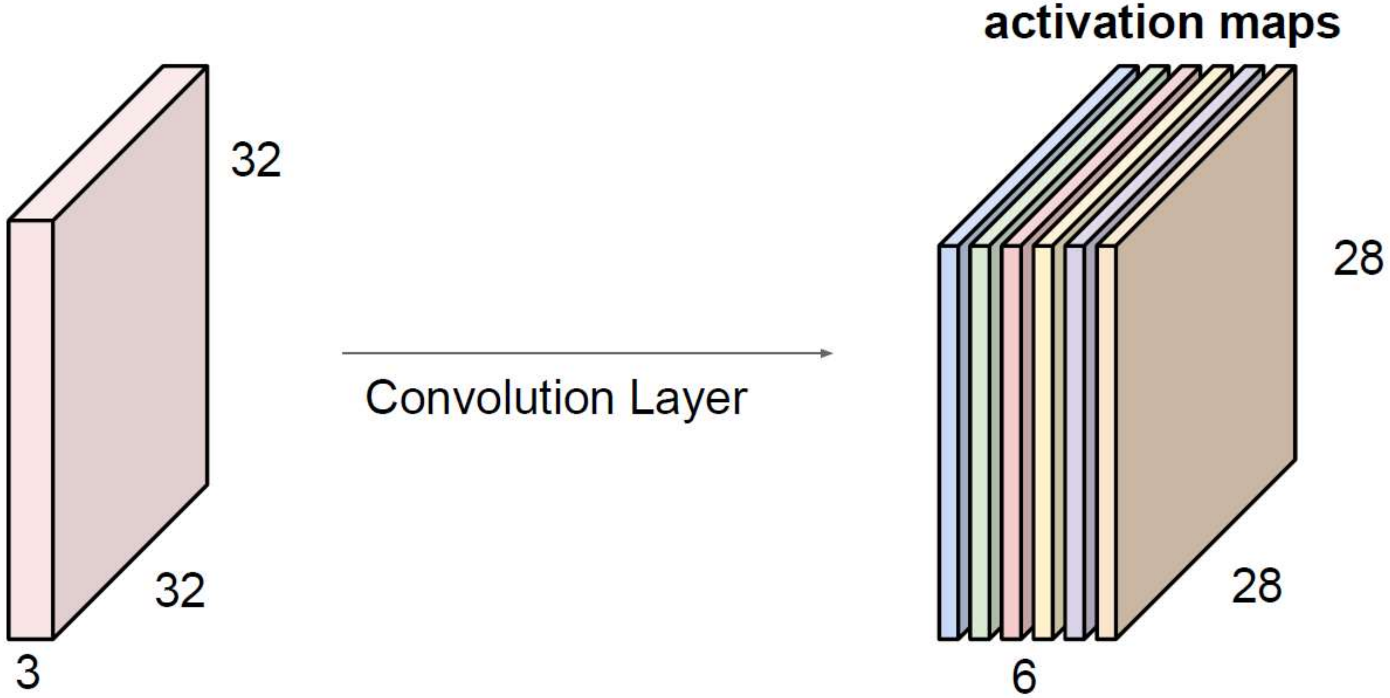


Convolution Layer

consider a second, green filter



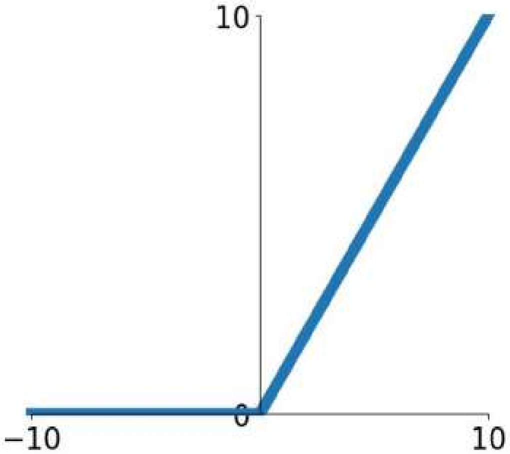
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



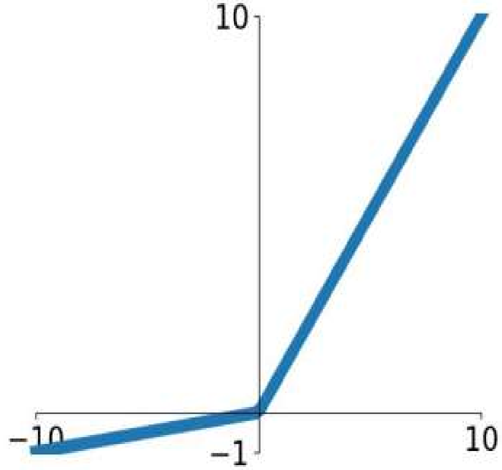
We stack these up to get a "new image" of size 28x28x6!

Activation Functions – ReLU (Rectified Linear Unit)

ReLu is an activation function that introduces property of non-linearity to a deep learning model.



$$f(x) = max(0, x)$$



$$f(x) = max(0.01x, x)$$

Leaky ReLU

Converges much faster than some other nonlinear activation functions, e.g., sigmoid/tanh

One layer of a convolutional network

Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

Summary of notation

Let layer l is a convolutional layer (superscript indicates the current layer):

(l-1) indicates a previous layer

$f^{[l]}$ = filter size
$p^{[l]}$ = padding (valid or same)
$s^{[l]}$ = stride
$n_c^{[l]}$ = number of filters

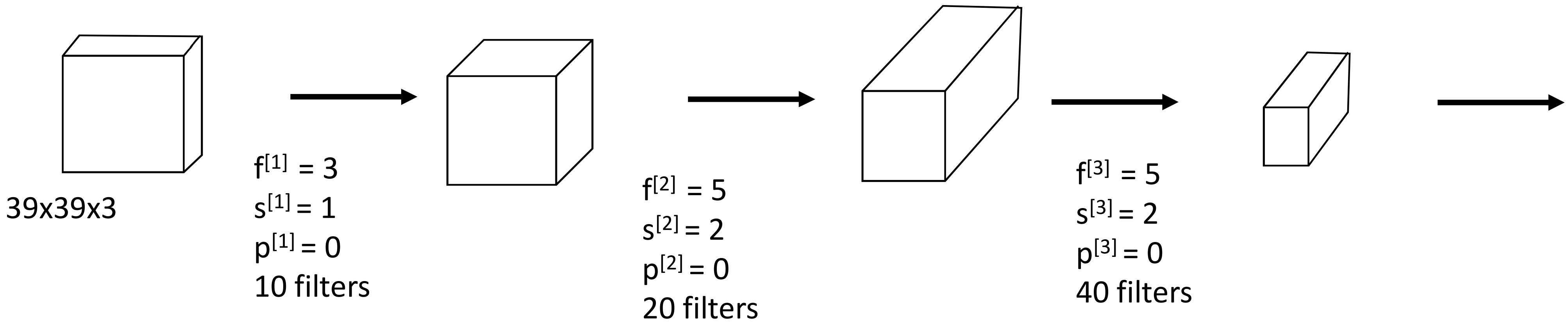
Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n^{[l]} = \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
Bias: $n_c^{[l]}$

Where does the number of channels come from?
  Number of channels in the output volume equals the number of filters
How about the size of each filter?
  Number of channels must be equal to the number of channels in the input layer

A simple convolution network example



Designing a network

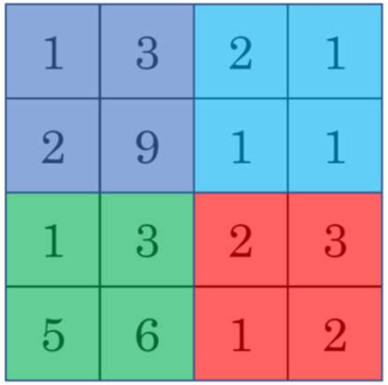
- There are numerous design choices and parameters that we have to deal with (as we also have to in traditional CV, but quite different)
- How do we find the right ones? Defining hyperparameters (e.g. filter sizes, number of layers, strides, paddings, …) is not straightforward. One should follow some established examples (empirical science)
- Some general principles: intermediate volumes shrink in W and H, but increase in the number of channels.

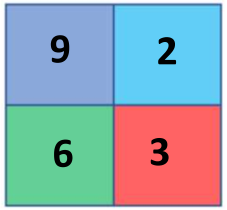
Types of layers in a convolutional network

- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)

Pooling

Pooling layer: Max pooling





Hyperparameters:
f=2
s=2
No parameters to learn

Andrew Ng

- Reduce the size of the representation and (speed up) the computation
- May also make features a bit more robust
- It works well in practice, precise reasons are not really well understood
- This is an operation where there is nothing to learn, there are no weights

Max Pooling – another example

1	3	2	1	3
2	9	1	1	5
1	3	2	3	2
8	3	5	1	0
5	6	1	2	9

(n+2p-f)/s + 1

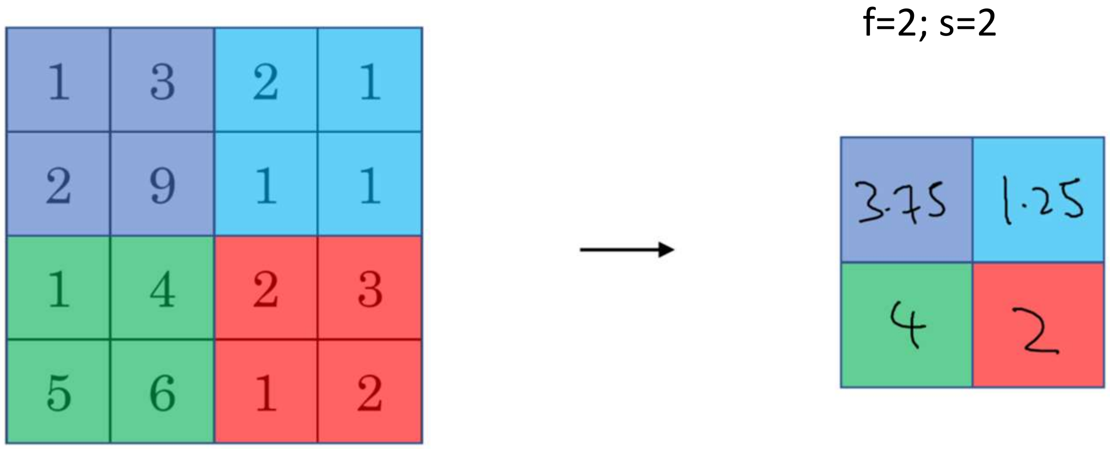
Hyperparameters:
f=3
s=1
No parameters to learn

The formula developed for the convolution also works well for Max Pooling

If you have more channels, the output will have the same number of channels. Pulling is performed on each channel independently.

Pooling

Pooling layer: Average pooling



Andrew Ng

Summary of Pooling

Hyperparameters:

f : filter size

s : stride

Max or average pooling

$n_H \times n_W \times n_c$

$$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor$$

No parameters to learn

What have we learned today

- We have achieved our main objectives set out at the beginning of this lecture
- Starting from the requirements
  - Learn a variety of filters/features (many!)
  - Combine them in a unified manner
  - In a nonlinear way
  - Build a model that can be optimised for a specific task
- We made step-by-step progression towards an end-to-end deep neural network
- We now know the building blocks of convolutional neural networks
- We can pull everything together and build an example (next lecture)

What is coming next (Week 6)?

- We will build an example of a convolutional neural network (activation maps, weights, biases)
- We will look at the main properties of the convolutional neural networks and what makes them distinctive
- We will briefly revisit the classical computer vision models for segmentation and show a few deep learning models
- We will revisit PCA method and show how it can be related to deep learning model of autoencoders
- We will summarise the overall insights and contrast the two methodological approaches

Computer vision and imaging

From *classical* to *deep learning-based* methods in Computer Vision

Week 6, Lecture 1

Aleš Leonardis

Office: UG 34