# Machine Learning L+Pr

Béla J. Szekeres, PhD

Lecture 10

ELTE Faculty of Informatics, Szombathely, Hungary

# Topics of this day

- Motivation of Convolutional networks
- Basic bulding blocks
- Putting together

## Motivation

**Problem definition:** MNIST `http://yann.lecun.com/exdb/mnist/`

| | | |
|---|---|---|
| **0** | $\longrightarrow$ | **(1 0 0 0 0 0 0 0 0 0)** |
| **1** | $\longrightarrow$ | **(0 1 0 0 0 0 0 0 0 0)** |
| **2** | $\longrightarrow$ | **(0 0 1 0 0 0 0 0 0 0)** |
| **3** | $\longrightarrow$ | **(0 0 0 1 0 0 0 0 0 0)** |
| **4** | $\longrightarrow$ | **(0 0 0 0 1 0 0 0 0 0)** |
| **5** | $\longrightarrow$ | **(0 0 0 0 0 1 0 0 0 0)** |
| **6** | $\longrightarrow$ | **(0 0 0 0 0 0 1 0 0 0)** |
| **7** | $\longrightarrow$ | **(0 0 0 0 0 0 0 1 0 0)** |
| **8** | $\longrightarrow$ | **(0 0 0 0 0 0 0 0 1 0)** |
| **9** | $\longrightarrow$ | **(0 0 0 0 0 0 0 0 0 1)** |

we're dealing

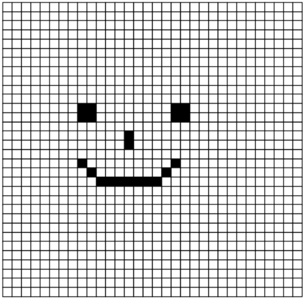
with handwritten digit classification

- we have 10 classes
- represent the classes in binary
- **One-hot encoding**
- 60K training samples - 10K test
- $28 \times 28$ pictures

## Motivation

Problem definition: CIFAR-10

https://www.cs.toronto.edu/~kriz/cifar.html



we're dealing with

a multiclass classification task again

- we have 10 classes
- 50K training samples - 10K test
- $32 \times 32 \times 3$ pictures

$\rightarrow$ we can

transform it to a vector with size 3072

## Problem with deep networks

- Dense networks are working fine, but...
- if there are 1000 neurons in each layer, the number of weights increase dramatically
- the training will be extremely slow
- in the previous example for example with 1 hidden layer which contains 1000 neurons we have $3072 \cdot 1000 + 1000 \cdot 10 + 10 + 1000 \approx 3M$ parameters
- and this was only a $32 \times 32 \times 3$ shape picture

### Solution - Convolutional networks

- We know the inputs are images
- so we can encode certain properties into the network architecture

# Motivation

What is the main problem in machine learning? **Feature selection**

What features to use to build our model?

What features to use in order to recognize faces?

- location of nose
- distance between eyes

What features to use to recognize a cheetah?

- shape of ears
- black pattern

# Building blocks

**We have 3 important steps**

- convolutional operation (filter)
- pooling
- flattening

# Kernels

## Feature detector/kernel/filter

Feture detectors are represented as matrices

$\rightarrow$ helps to detect the relevant features in a given image

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

This is the sharpen kernel

- increase the pixel intensity of the given one
- reduce it at the neighborhood

the original image

image after applying the sharpen kernel

## Kernels

### Feature detector/kernel/filter

Feture detectors are represented as matrices

$\rightarrow$ helps to detect the relevant features in a given image

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

This is the edge detection kernel

- decrease the pixel intensity of the given one
- don't change at the neighborhood

the original image

image after applying
the edge detector kernel

# Kernels

## Feature detector/kernel/filter

Feture detectors are represented as matrices
→ helps to detect the relevant features in a given image

| | | |
|---|---|---|
| **1** | **1** | **1** |
| **1** | **1** | **1** |
| **1** | **1** | **1** |

This is the blur kernel

- don't change the pixel intensity of the given one
- we use the neighborhood

the original image

image after applying the blur kernel

# Kernels

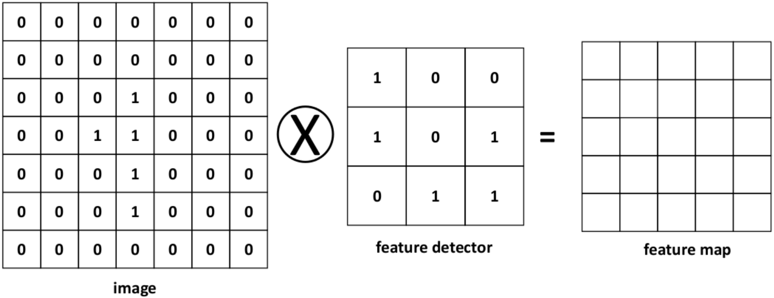
Every single kernel will use a specific feature of the image

When using edge detector, we assume the edges are important

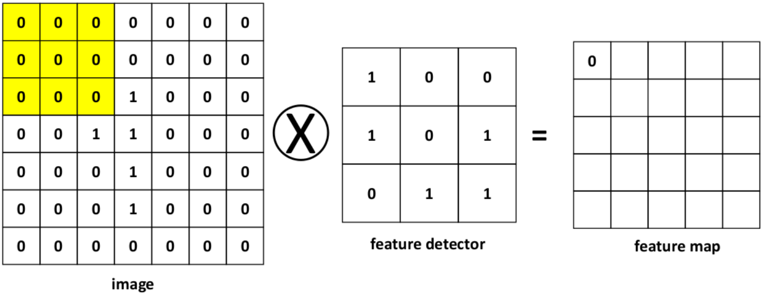
**How to decide what feature detector to use?**

- No need to decide in advance!
- convolutional networks use many kernels and during the training procedure it eventually select the best possible

## How does a kernel work?



# How does a kernel work?



# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

image

$\otimes$

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

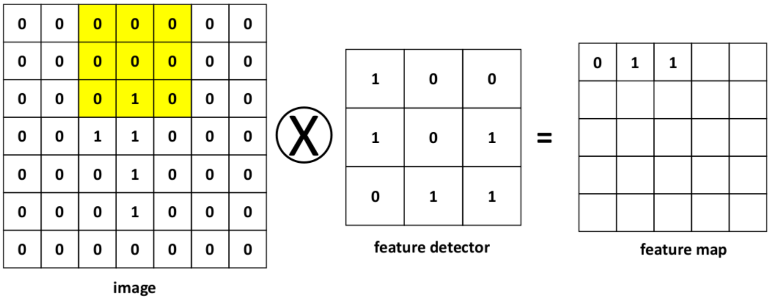
feature detector

=

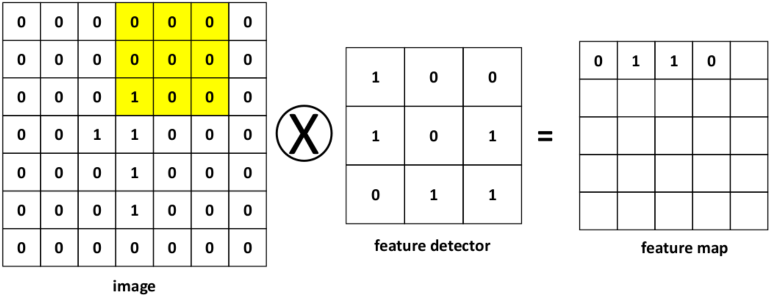
| | | | | |
|---|---|---|---|---|
| 0 | 1 | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

feature map

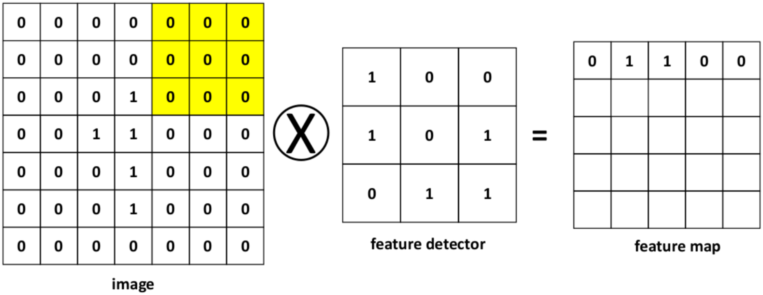
# How does a kernel work?



# How does a kernel work?



# How does a kernel work?



# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**image**

$\otimes$

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

**feature detector**

=

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | | | | |
| | | | | |
| | | | | |
| | | | | |

**feature map**

# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**image**

$\otimes$

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

**feature detector**

=

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 3 | | | |
| | | | | |
| | | | | |
| | | | | |

**feature map**

# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

image

$\otimes$

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

feature detector

=

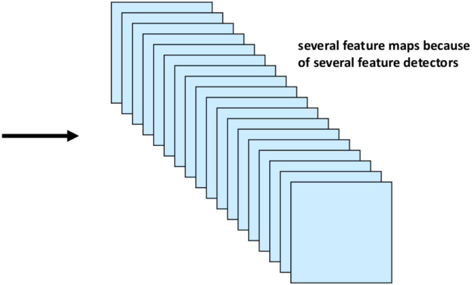
| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 3 | 1 | 1 | 0 |
| 1 | 2 | 2 | 2 | 0 |
| 0 | 2 | 2 | 2 | 0 |
| 0 | 1 | 0 | 2 | 0 |

feature map

# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

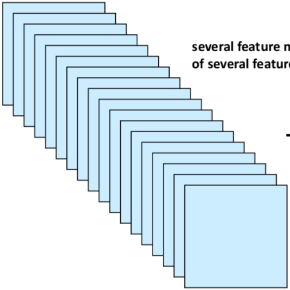
image



# How does a kernel work?

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

image



several feature maps because of several feature detectors

+ ReLU

# How does a kernel work?

## Spatial invariance

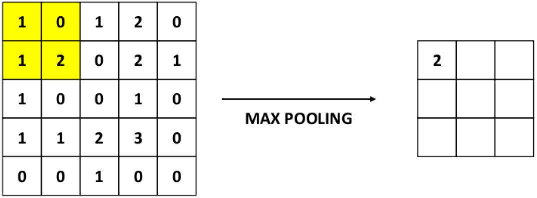
We would like to make sure to detect the same object no matter where is it located on the image or whether it is rotated/transformed

it is a cat: the location on the image does not matter or whether it is rotated or transformed

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.



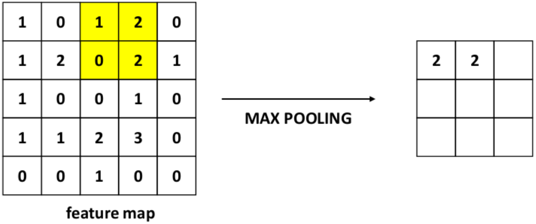

feature map

- we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise
- we reduce number of parameters: reduce overfitting

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.

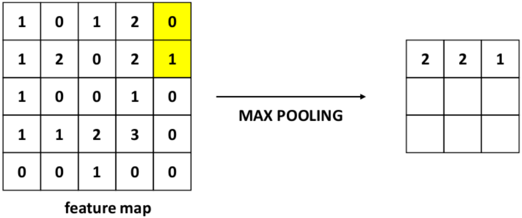



- we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise
- we reduce number of parameters: reduce overfitting

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.

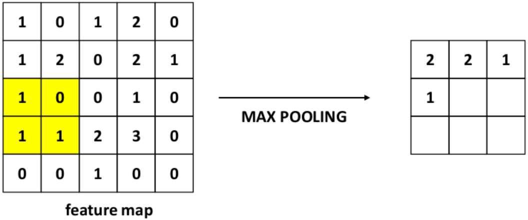



- we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise
- we reduce number of parameters: reduce overfitting

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.

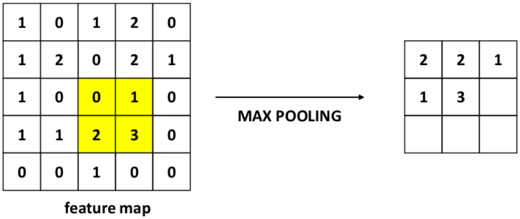



- we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise
- we reduce number of parameters: reduce overfitting

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.

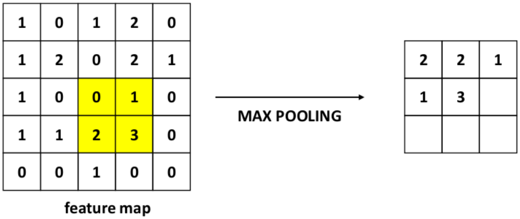



- we can reduce the dimension of the image in order to end with a dataset containing the important pixel values without the unnecessary noise
- we reduce number of parameters: reduce overfitting

# Pooling

## Max pooling

With **max pooling** we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features.



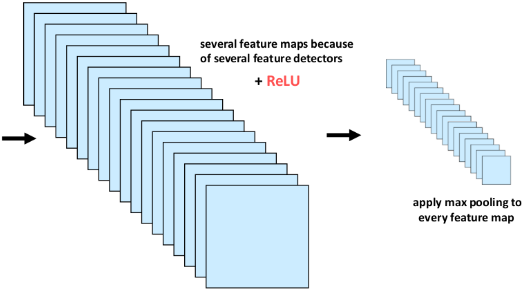

- There are other techniques: **average pooling** is popular as well
- instead of choosing the maximum value we calculate the average of the values present in the subset

# Pooling

With max pooling we select the most relevant features: this is how we deal with spatial invariance. We just care about the most relevant features

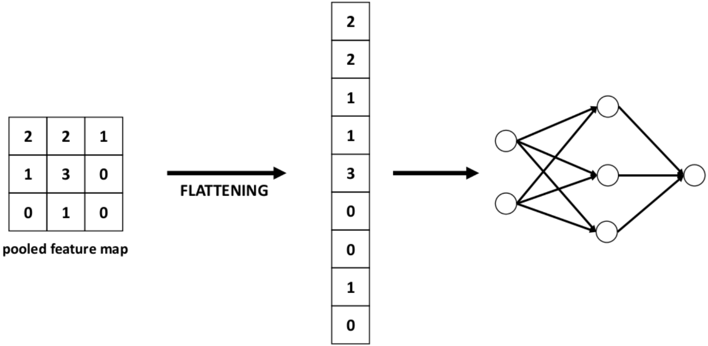
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

image

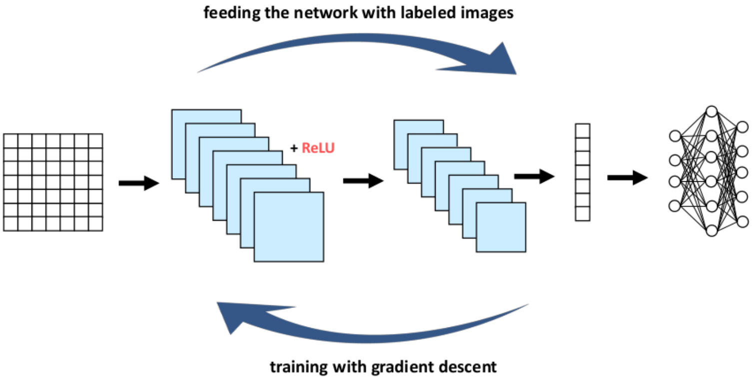


# Flattening

The last operation we have to make is the flattening procedure: we transform the matrix into a one-dimensional vector: this is the input of a standard densely connected neural network



# Putting all together

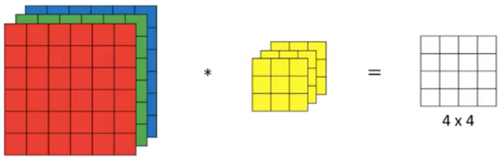


# Convolutions on a color image - over a volume

## Tutorials

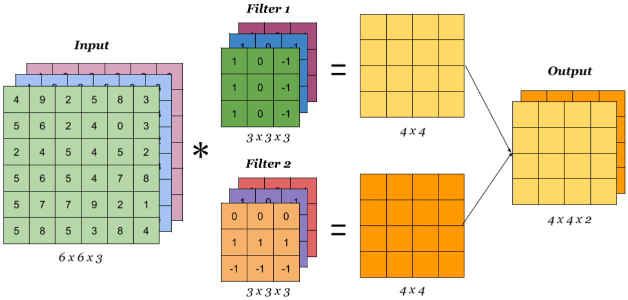
- https://www.youtube.com/watch?v=KTB_OFoAQcc
- https://gaussian37.github.io/dl-concept-cnn/
- https://guandi1995.github.io/One-Layer-of-a-Convolutional-Networks/

Convolutions on RGB image



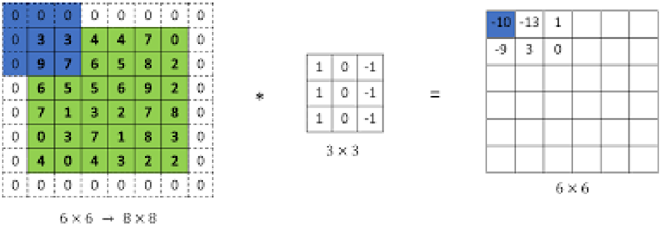
# Convolutions on a color image - over a volume

Convolution = Scalar product over the volume



# Padding

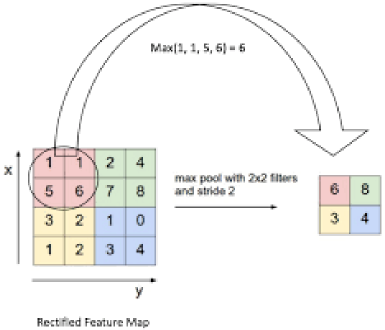
Padding with zeros by $P$ layers (containing 0s).



# Stride

## Pooling

Skipping $S$ pixels.



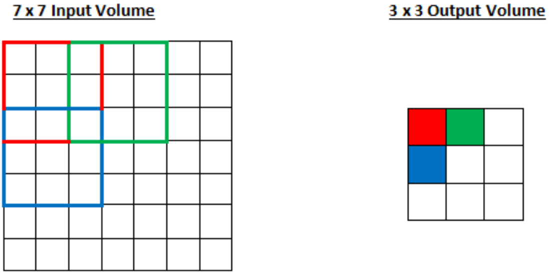

## Pooling

You have to do the pooling by layers separately in spite of convolution.

# Stride

## Convolution

Skipping $S$ pixels.



# Shape calculations

## Notations

- $f^{[l]}$: filter size
- $p^{[l]}$: filter size
- $s^{[l]}$: stride size
- $n_c^{[l-1]} = c$: number of input channels
- $n_c^{[l]} = n_f$: number of output channels = number of filters

## Input size

$$a^{[l-1]} = n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]} = n_H^{[l-1]} \times n_W^{[l-1]} \times c$$

## Filter size

$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} = f^{[l]} \times f^{[l]} \times c$$

# Shape calculations

**Input size**

$$a^{[l-1]} = n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]} = n_H^{[l-1]} \times n_W^{[l-1]} \times c$$

**Filter size**

$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} = f^{[l]} \times f^{[l]} \times c$$

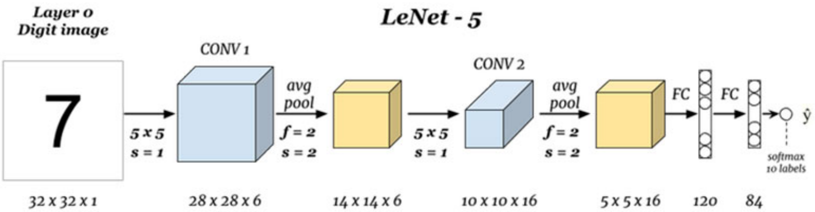
**Output size**

- $a^{[l]} = n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]} = n_H^{[l]} \times n_W^{[l]} \times n_f$
- $n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$ and $n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

Kernel sizes and paddings can be different in each dimensions:

`https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`

# Example: LeNet-5



## Other example