

# Convolutional neural networks

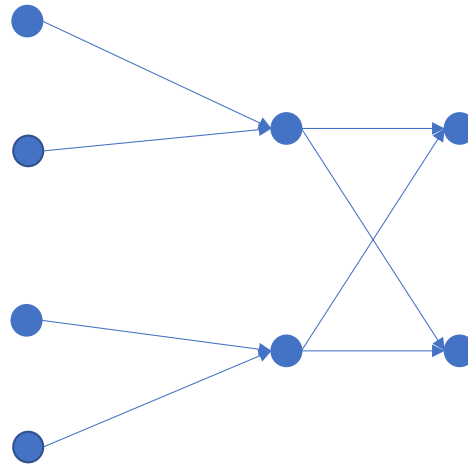
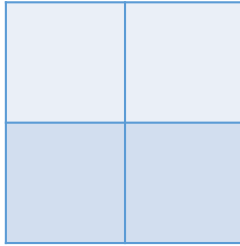
Bjarte Johansen  
bjajoh@equinor.com

# Convolutional neural networks

- A regularized kind of feed-forward neural network
- Specially constructed for image classification
- Not all neurons in the previous layer is connected to the next layer
- Can contain layers that are convolutional, pooling, and fully connected
- First proposed in the 1970s, but the modern version is by LeCun et al. in 1998:

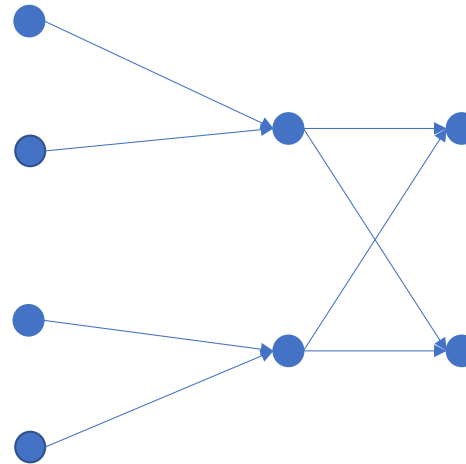
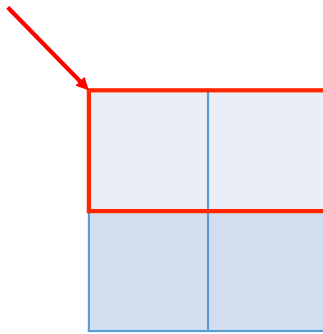
*Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", Proc. of the IEEE, 86(11): 2278-2324, November 1998.*

# Convolutional neural network

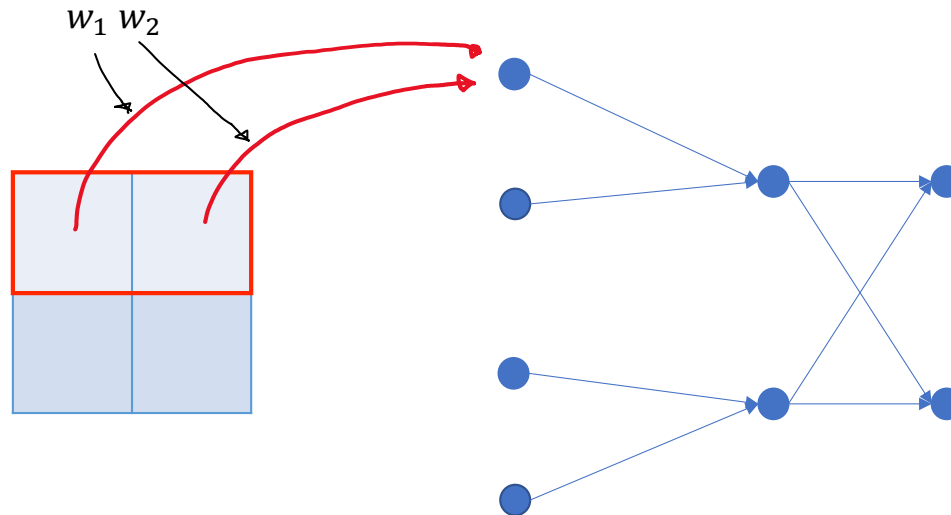


# Convolutional neural network

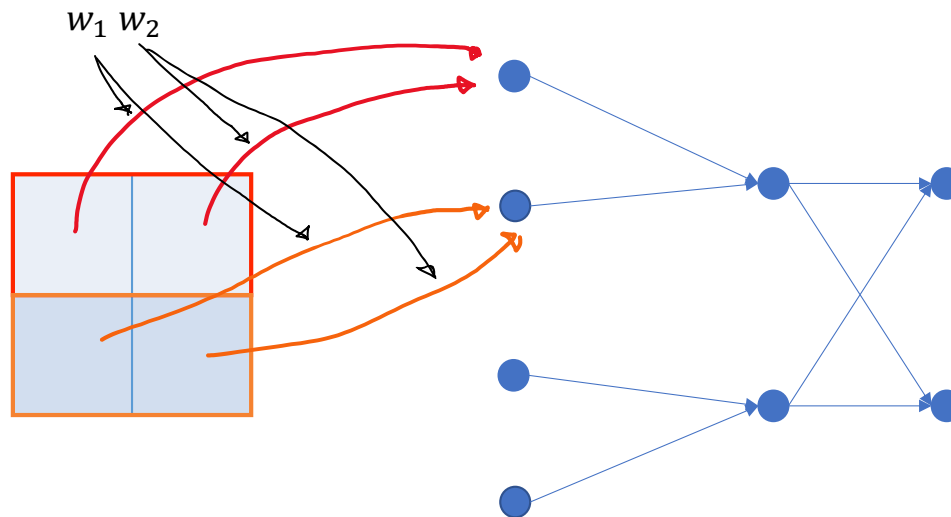
Receptive field



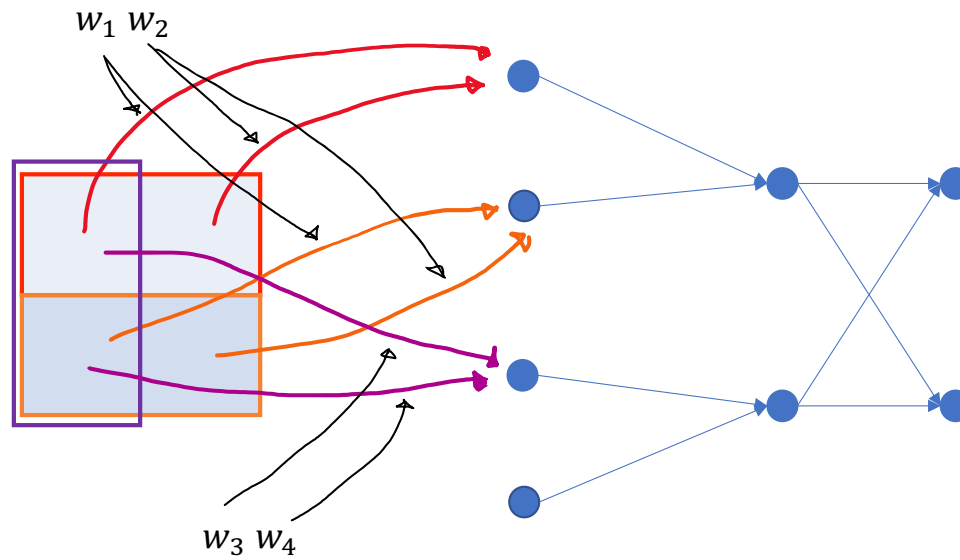
# Convolutional neural network



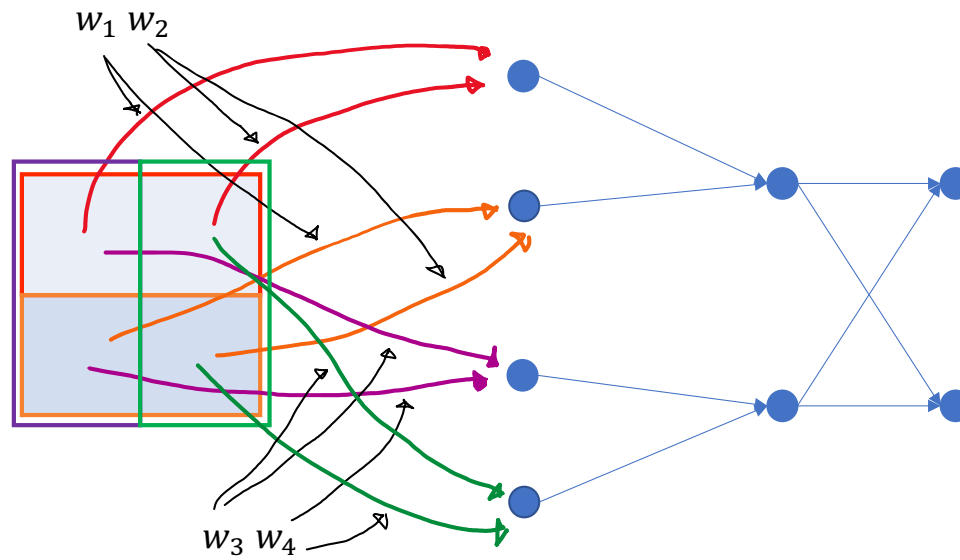
# Convolutional neural network



# Convolutional neural network

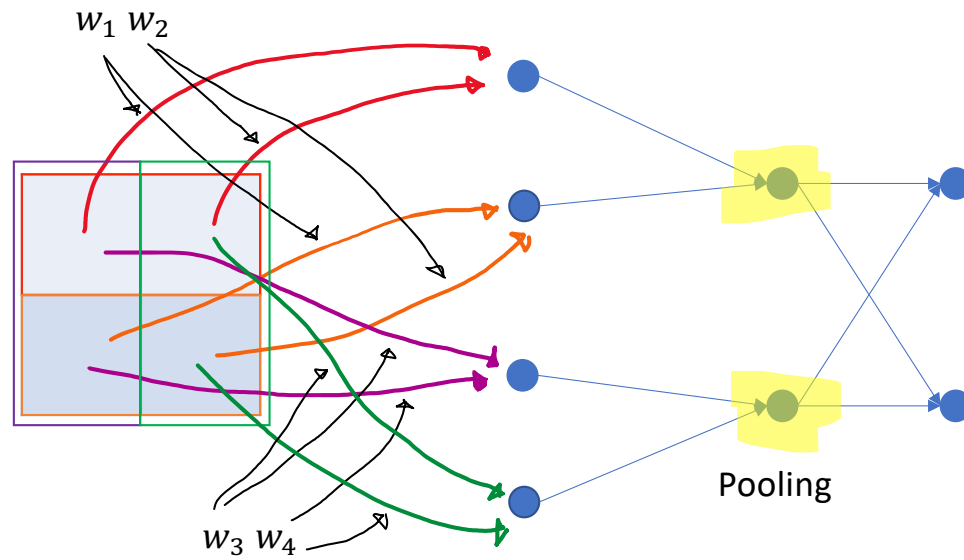


# Convolutional neural network





# Convolutional neural network




# What is a convolution?

$$(f \otimes g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

To convolve a kernel with an input signal: flip the signal, move to the desired time, and accumulate every interaction with the kernel.

# What is a convolution?

$$(f \otimes g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$


I am using this in place of  $*$  here just so we don't mix them up,  
but it is “just” fancy multiplication of functions (or signals)

To **convolve** a **kernel** with an **input signal**: **flip the signal**, **move to the desired time**, and **accumulate every interaction with the kernel**.

# What is a convolution?

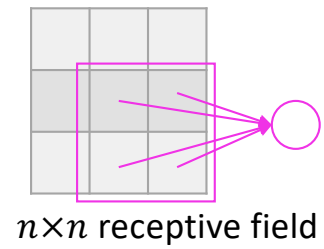
$$(f \otimes g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

To convolve a kernel with an input signal: flip the signal, move to the desired time, and accumulate every interaction with the kernel.

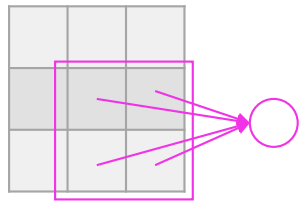
The kernel can also be describe as a window, filter, feature detector ...

<https://betterexplained.com/articles/intuitive-convolution/>

# What is a convolutional layer?



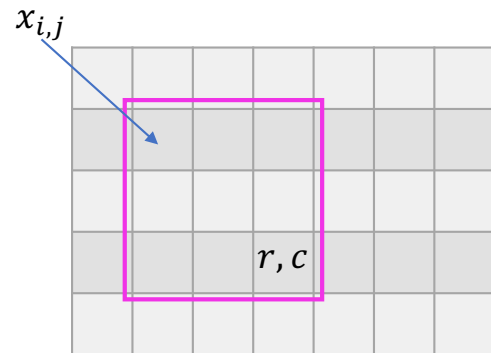
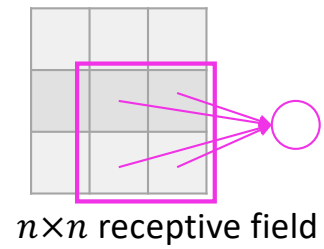
# What is a convolutional layer?



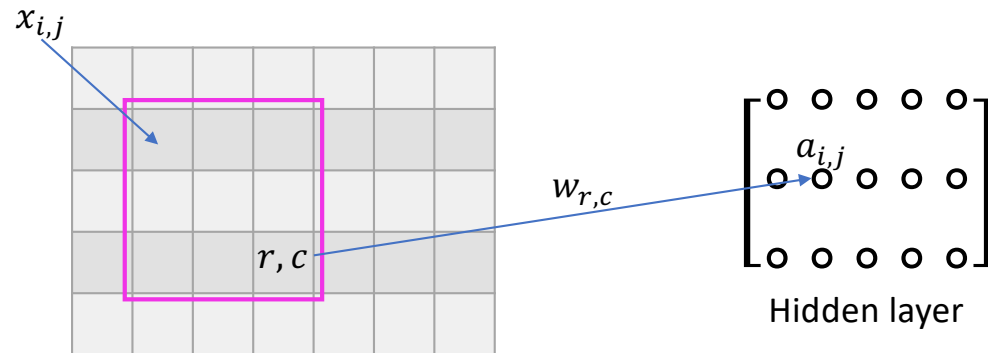
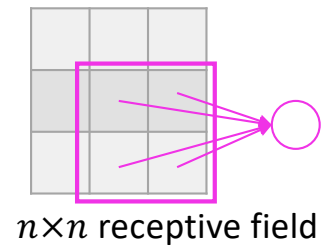
$n \times n$  receptive field



# What is a convolutional layer?

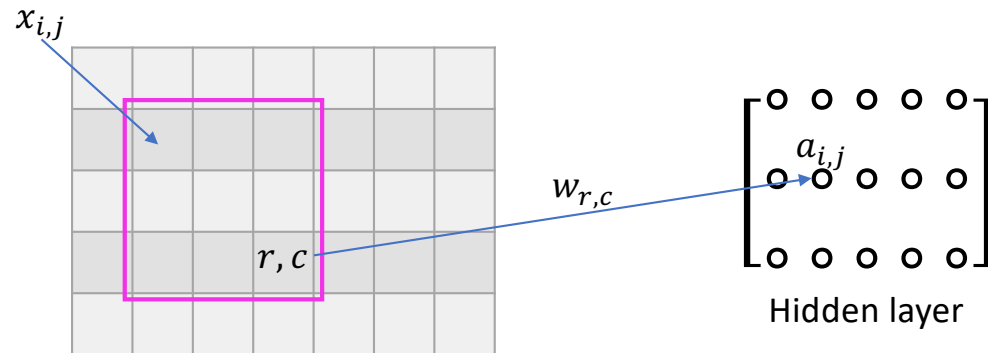
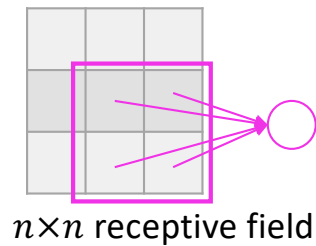


# What is a convolutional layer?





# What is a convolutional layer?



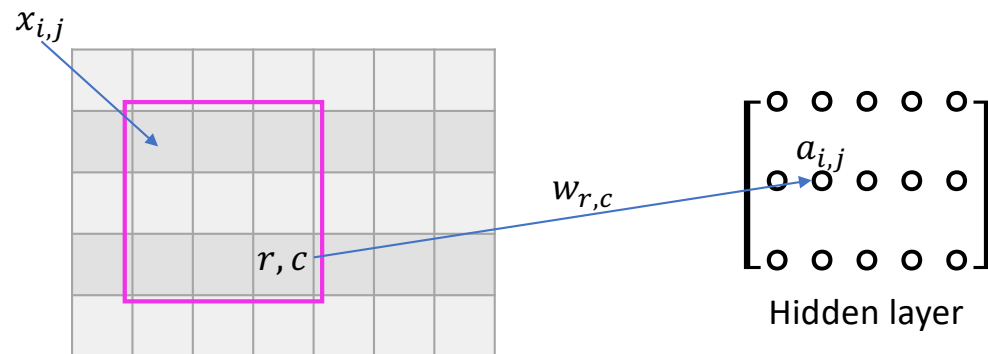
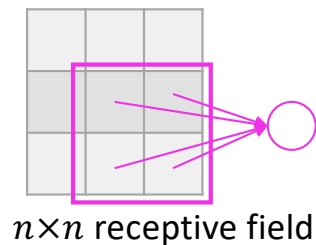
We can express the activation of a convolutional neuron as

$$a_{i,j} = b + \sum_{r=0}^n \sum_{c=0}^n w_{r,c} x_{i+r,j+c}$$

$$a_{i,j} = b + \vec{w} \otimes \vec{x}_{i,j}$$

Where the vector  $x_{i,j}$  is the  $n \times n$  receptive field matrix at location  $r, c$  in the input and  $w$  is a weight matrix.

# What is a convolutional layer?



We can express the activation of a convolutional neuron as

$$a_{i,j} = b + \sum_{r=0}^n \sum_{c=0}^n w_{r,c} x_{i+r,j+c}$$

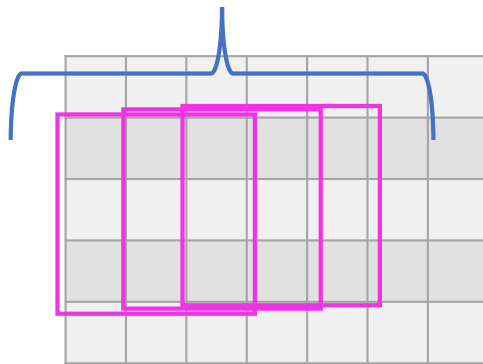
$$a_{i,j} = b + \vec{w} \otimes \vec{x}_{i,j}$$

Where the vector  $x_{i,j}$  is the  $n \times n$  receptive field matrix at location  $r, c$  in the input and  $w$  is a weight matrix.

If the input layer contains  $n \times n$  pixels and the size of the local receptive field is  $r \times r$ , then the convolutional layer will contain  $c \times c$  neurons, where  $c = n - r + 1$ .

# Hyperparameters

The distance the receptive field is moved for each sampling



Here, the stride length is 1

If the input layer contains  $n \times n$  pixels, the stride length  $s > 1$ , and the receptive field is  $r \times r$ , the convolutional layer will contain  $c \times c$  neurons where  $c = \frac{n-r}{s} + 1$ .

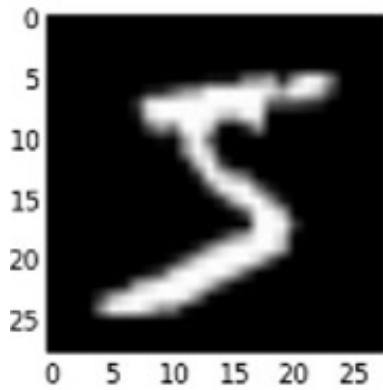
Since  $c$  must be an integer,  $s$  cannot take just any value. To solve this, we can increase the size of the input by padding our image around the border. If  $p$  denotes the amount of padding, then

$$c = \frac{n + 2p - r}{s} + 1$$

Normally we use 0 as our padding value.

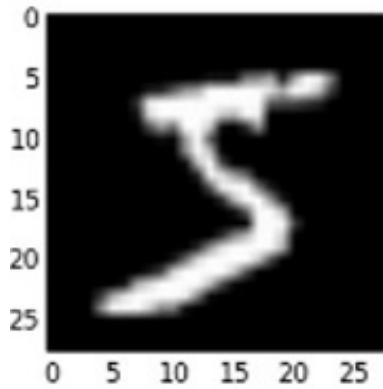
In general, larger local receptive fields tend to be helpful for larger input images.

# Why convolutional layers



$$28 \times 28 \times 1 = 728$$

# Why convolutional layers

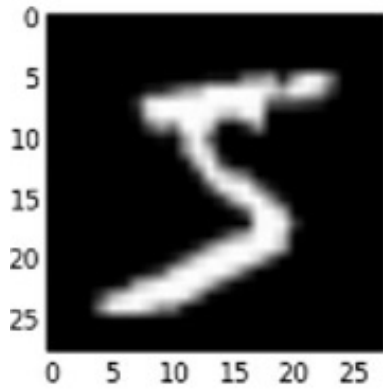


$$28 \times 28 \times 1 = 728$$



$$536 \times 536 \times 3 = 861,888$$

# Why convolutional layers



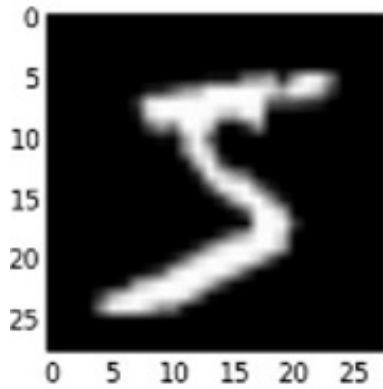
$$28 \times 28 \times 1 = 728$$



$$536 \times 536 \times 3 = 861,888$$

Why?

# Why convolutional layers



$$28 \times 28 \times 1 = 728$$



$$536 \times 536 \times 3 = 861,888$$

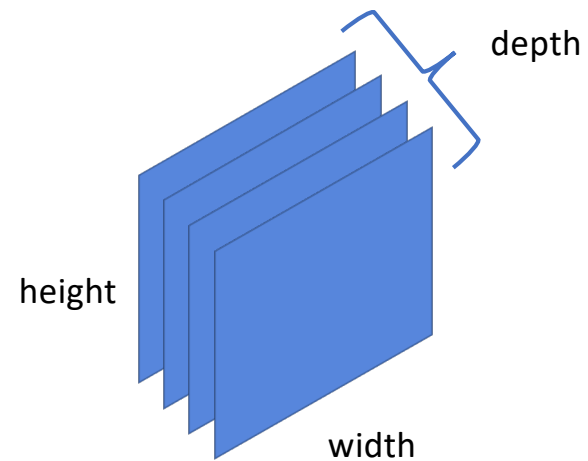
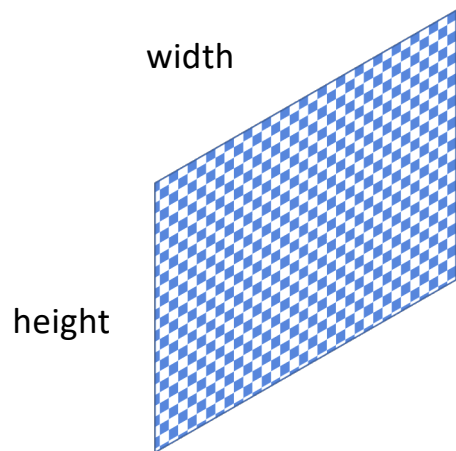


# 3D Layers

A convolutional layer is often represented as a volume, also referred to as a tensor:

- $weight \times height$  is the size of each feature map
- $depth$  is the number of feature maps

To recognize more features, a convolutional layer must consist of several different feature maps

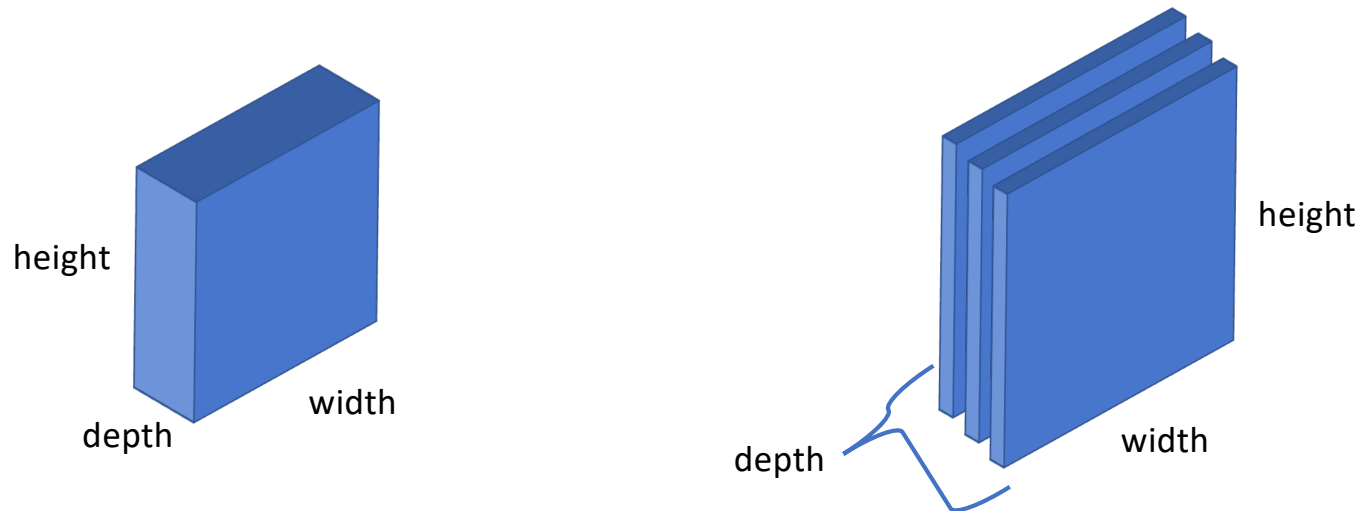




# 3D layers

Coloured images have an input volume with depth 3 (RGB). To capture all this information, the receptive field also needs to have a depth of 3.

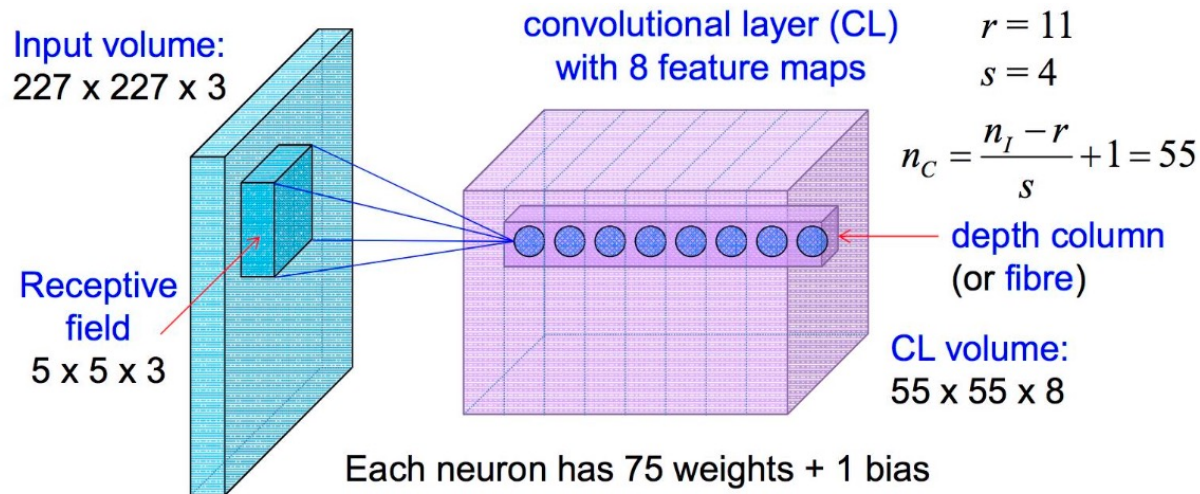
Each neuron in the convolutional layer therefore has 3 coordinates  $(i, j, k)$ ,  $3r^2$  weights, and a bias. ( $r$  = size of receptive field).



# 3D layers

Each neuron in the convolutional neural network is connected to each pixel in its local receptive field to the full depth (i.e. all colour channels)

All neurons along the depth (depth column or fibre) look at the same local region.



# Memory use

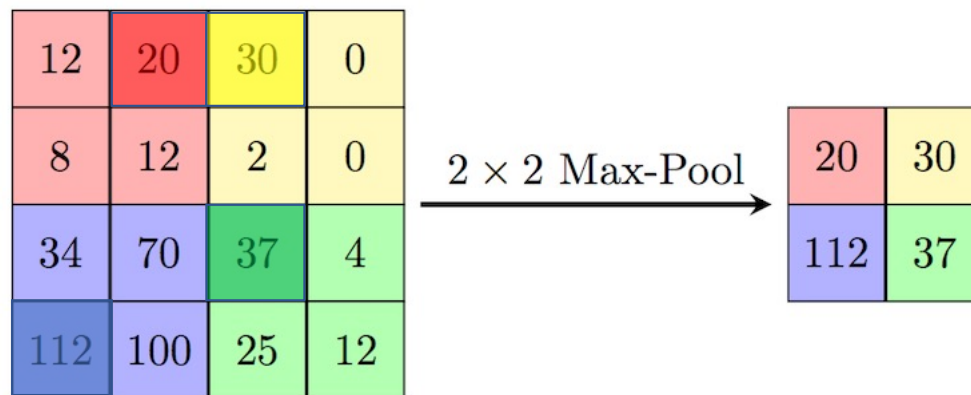
- For each feature map we need  $r \times r + 1$  parameters. If we have  $n$  feature maps, a convolutional layer requires:

$$\#parameters = (r^2 + 1)n$$

- If  $r = 5, n = 20$ , we need  $26 \times 20 = 520$  variables
  - This number does not depend on the input image
- A fully connected layer with 30 neurons and an image size of  $28 \times 28 = 784$ , would require  $784 \times 30 = 25,550$  parameters.

# Pooling layers

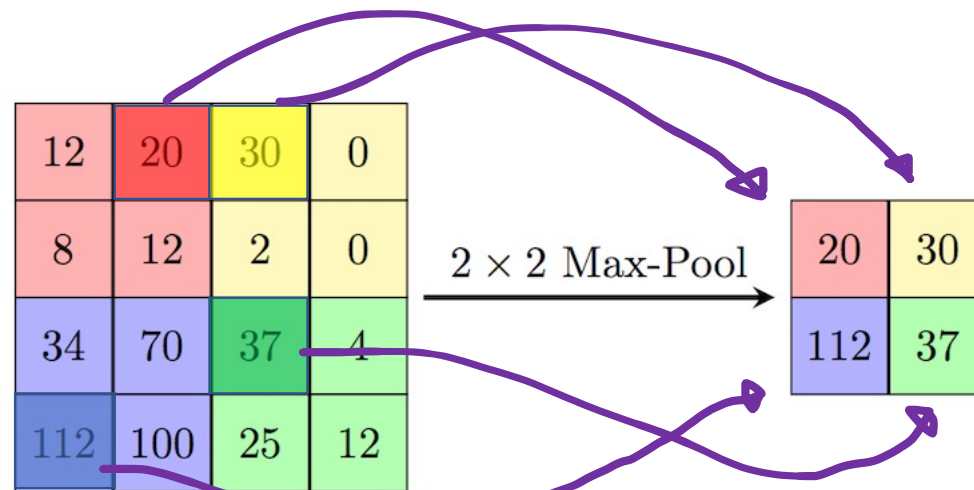
Often used after one or several conv. layers to aggregate the information and reduce the dimensionality by merging receptive fields.



[https://computersciencewiki.org/index.php/Max-pooling/\\_Pooling](https://computersciencewiki.org/index.php/Max-pooling/_Pooling)

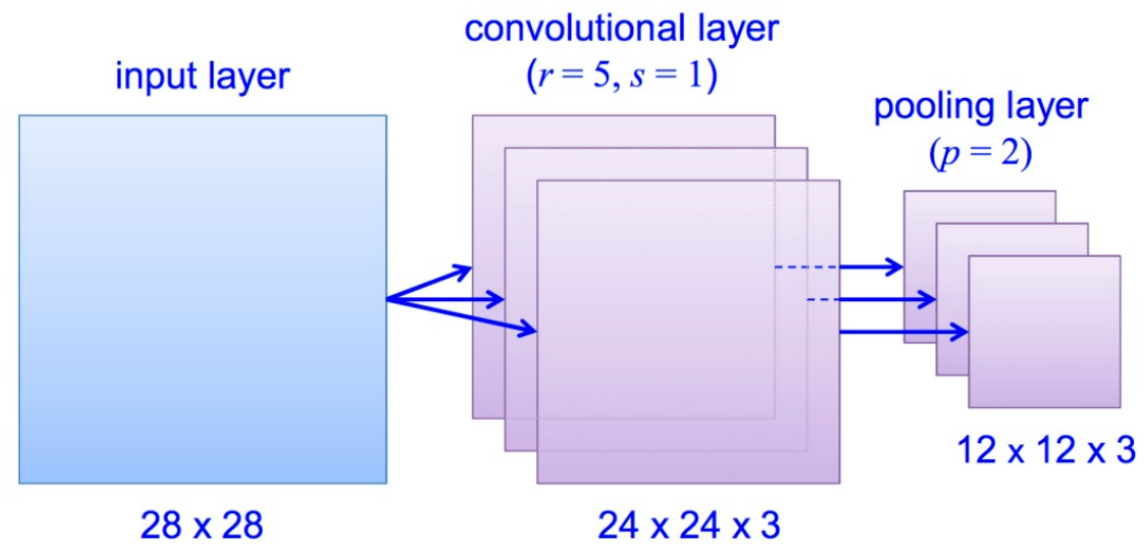
# Pooling layers

Often used after one or several conv. layers to aggregate the information and reduce the dimensionality by merging receptive fields.



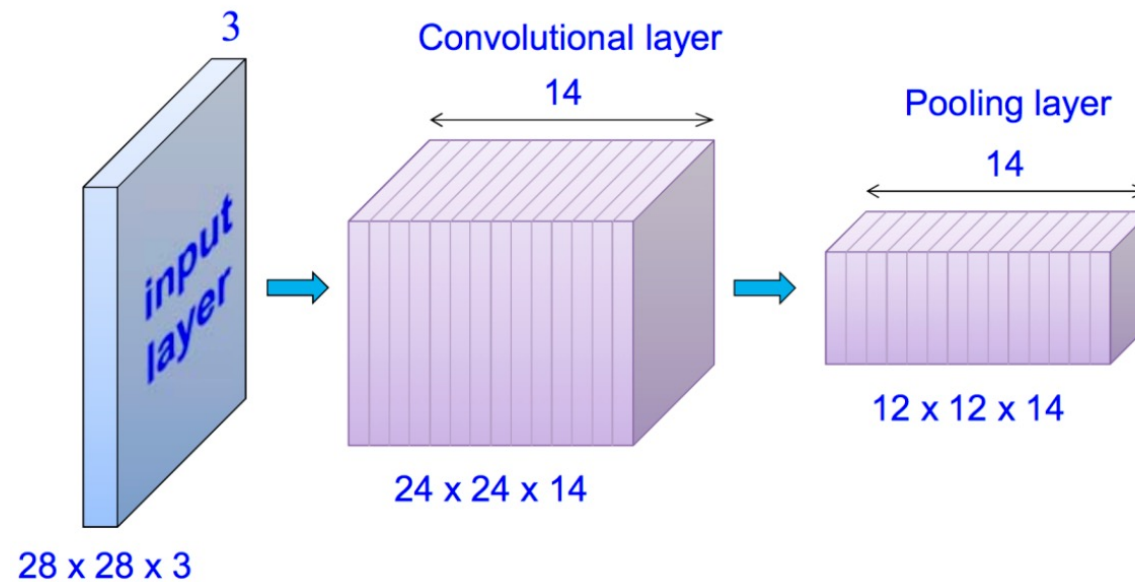
[https://computersciencewiki.org/index.php/Max-pooling/\\_Pooling](https://computersciencewiki.org/index.php/Max-pooling/_Pooling)

# Reduction of complexity



# Pooling layer in colour images

Colour images is represented in 3 dimensions. The pooling layers therefore performs a spatial downsampling and preserves the depth of the previous layer.



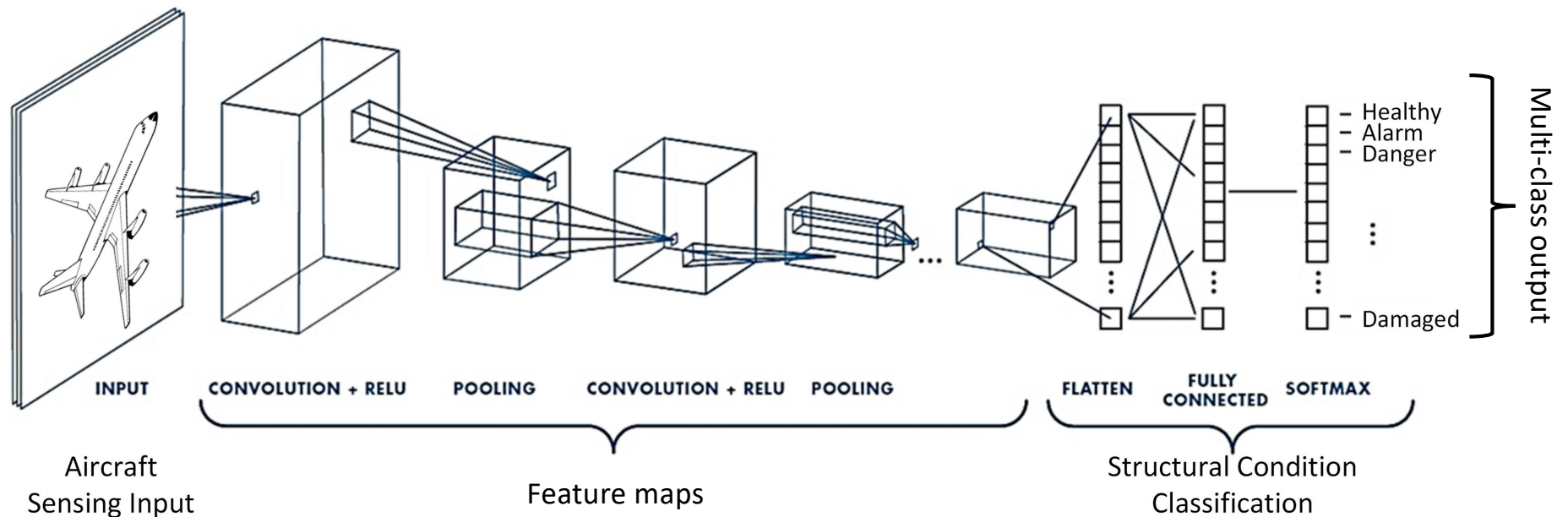
# Pooling layers

- Intuitively, pooling layers becomes a way for the network to say whether a given feature is present in a region of of an image.
- The heuristic is that, once a feature is found, its exact location is not as important as its rough location relative to other features.
- Another common pooling operation is L2-pooling; each neuron computes the L2 norm of the  $r \times r$  perceptive field

$$y_{j,k} = \sqrt{\sum_a^{r-1} \sum_b^{r-1} x_{j+a,k+b}^2}$$



# What does a complete CNN look like?



<https://www.mdpi.com/1424-8220/19/22/4933>

(not necessary to read, I just liked this image and this is where I found it)

# Loss functions

Cross entropy loss is often used for multi-class classification

- Output  $\hat{y}_i$  is a number in  $[0,1]$
- Convert output to a probability distribution (softmax)

$$p(\text{class} = c) = p_c = \frac{e^{\hat{y}_i}}{\sum_{i=0}^N e^{\hat{y}_i}}$$

The cross-entropy loss is then

$$\ell_{CE}(y, p) = - \sum_{c=1}^C y_c \log(p_c)$$

# Loss functions

Cross entropy loss is often used for multi-class classification

- Output  $\hat{y}_i$  is a number in  $[0,1]$
- Convert output to a probability distribution (softmax)

$$p(\text{class} = c) = p_c = \frac{e^{\hat{y}_i}}{\sum_{i=0}^N e^{\hat{y}_i}}$$

The cross-entropy loss is then

$$\ell_{CE}(y, p) = - \sum_{c=1}^C y_c \log(p_c)$$

For each class

# Loss functions

Cross entropy loss is often used for multi-class classification

- Output  $\hat{y}_i$  is a number in  $[0,1]$
- Convert output to a probability distribution (softmax)

$$p(\text{class} = c) = p_c = \frac{e^{\hat{y}_i}}{\sum_{i=0}^N e^{\hat{y}_i}}$$

The cross-entropy loss is then

$$\ell_{CE}(y, p) = - \sum_{c=1}^C y_c \log(p_c)$$

For each class

Calculate the expected value

# Loss functions

Cross entropy loss is often used for multi-class classification

- Output  $\hat{y}_i$  is a number in  $[0,1]$
- Convert output to a probability distribution (softmax)

$$p(\text{class} = c) = p_c = \frac{e^{\hat{y}_i}}{\sum_{i=0}^N e^{\hat{y}_i}}$$

The cross-entropy loss is then

For each class

$$\ell_{CE}(y, p) = - \sum_{c=1}^C y_c \log(p_c)$$

Calculate the expected value

The negative sum shows how similar the two distributions are

# Loss functions

Cross entropy loss is often used for multi-class classification

- Output  $\hat{y}_i$  is a number in  $[0,1]$
- Convert output to a probability distribution (softmax)

$$p(\text{class} = c) = p_c = \frac{e^{\hat{y}_i}}{\sum_{i=0}^N e^{\hat{y}_i}}$$

$$y = [0.0, 0.0, 1.0]$$
$$\hat{y} = [0.1, 0.2, 0.7]$$

The cross-entropy loss is then

For each class

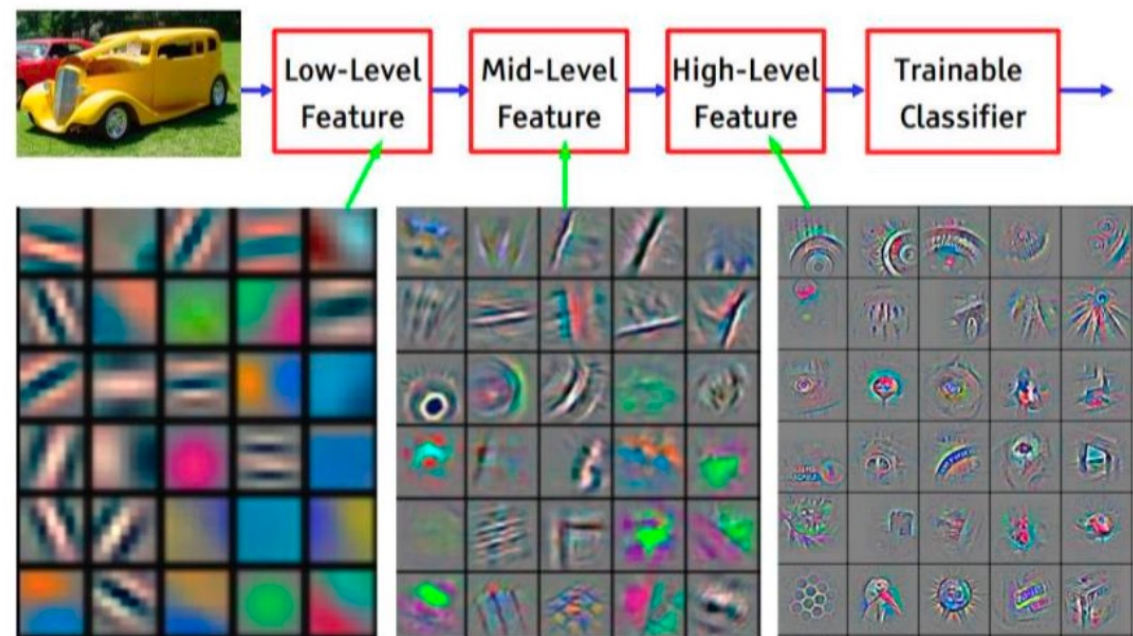
$$\ell_{CE}(y, p) = - \sum_{c=1}^C y_c \log(p_c)$$

Calculate the expected value

The negative sum  
shows how similar the  
two distributions are

# Learning CNN

- Use backprop and gradient descent, but...
- Pooling layers need to keep track of the index they were picked from.
- Weight updates need to only follow the receptive field that potentially activates a neuron in the CNN layer.
- Convolutional and pooling layers learn local spatial features;
- Later, fully connected layers integrates information across the features.

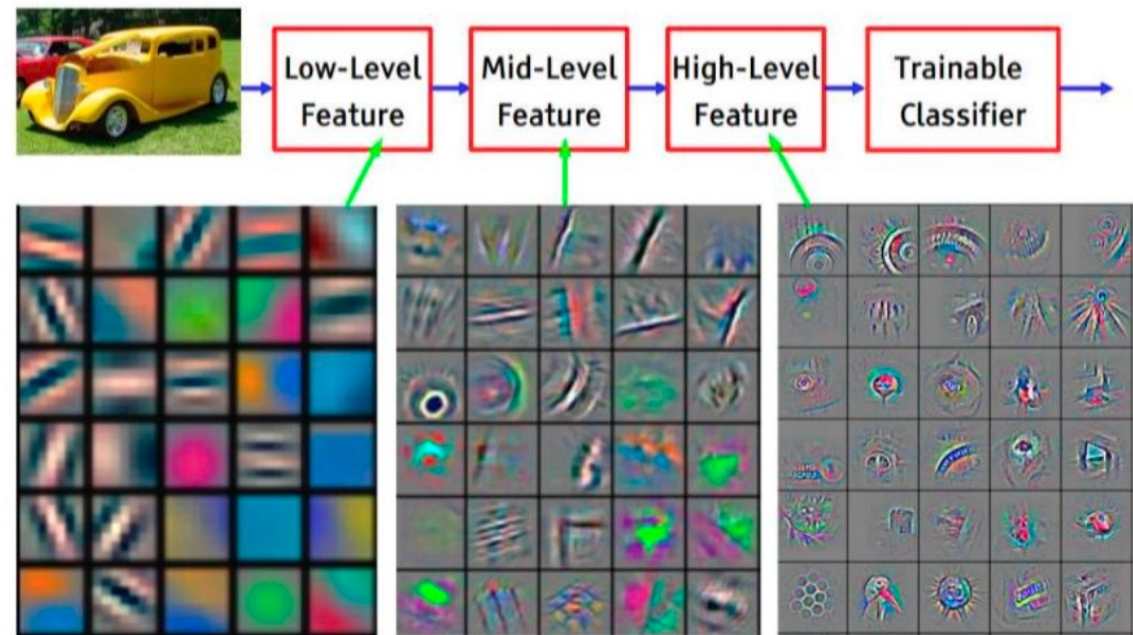


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[https://link.springer.com/content/pdf/10.1007/978-3-319-10590-1\\_53.pdf](https://link.springer.com/content/pdf/10.1007/978-3-319-10590-1_53.pdf)

# Visualising feature maps

- In practice, CNNs use tens of feature maps
- To see the features learned layer by layer, we can visualize the weights.
- We can see that the feature maps learn spatial structures.
- It is not easy to see exactly what they learned beyond that.



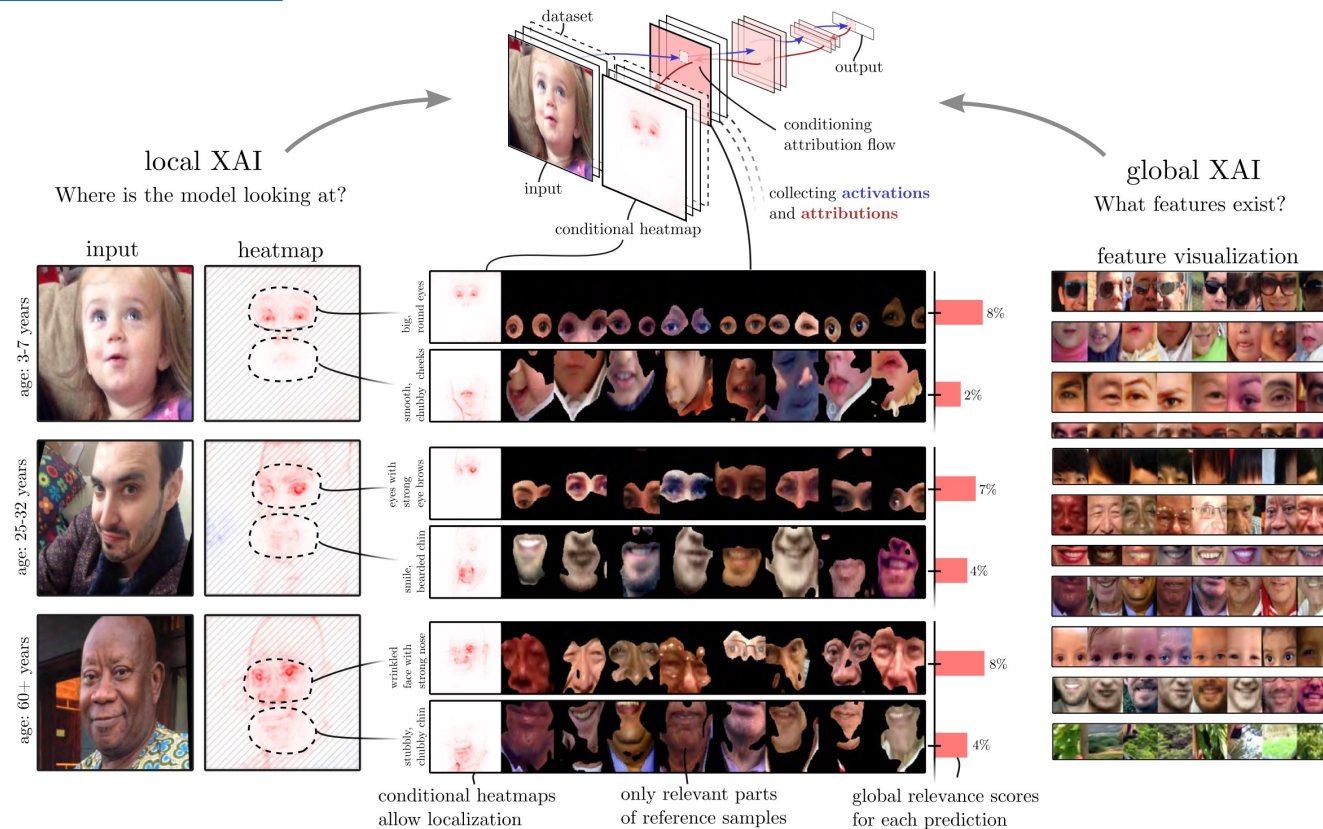
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



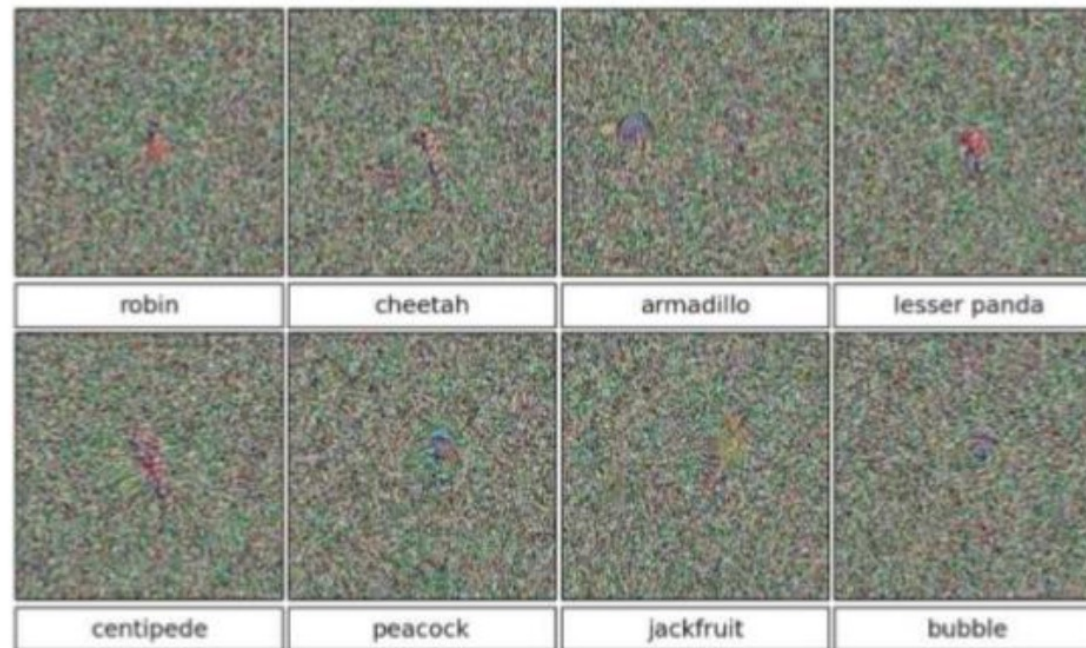
I don't expect you to read this, but...

# Some optional reading on explaining features

<https://arxiv.org/abs/2206.03208>



# Fooling the network



Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen et al. CVPR 2015]

# Fooling the network



The pathological liar ChatGPT

ChatGPT is trained as a pathological liar, it tries to invent things that could have been true.

The only thing language models like ChatGPT are capable of is to produce text that seems probable. They have a complete view of all the common clichés that we often fall for, writes the author of the chronicle.

# CNNs in Python

Many software frameworks are available

- Tensorflow, <https://www.tensorflow.org>
- Theano, <https://github.com/Theano/Theano>
- Torch, <https://pytorch.org>
- Tiny-dnn, <https://github.com/tiny-dnn/tiny-dnn>
- JAX, <https://jax.readthedocs.io>

To make it easy: Start with pytorch <https://pytorch.org>

# Next time

- Recurrent Neural Networks
- Long short-term memory
- Transfer learning
- Pretraining