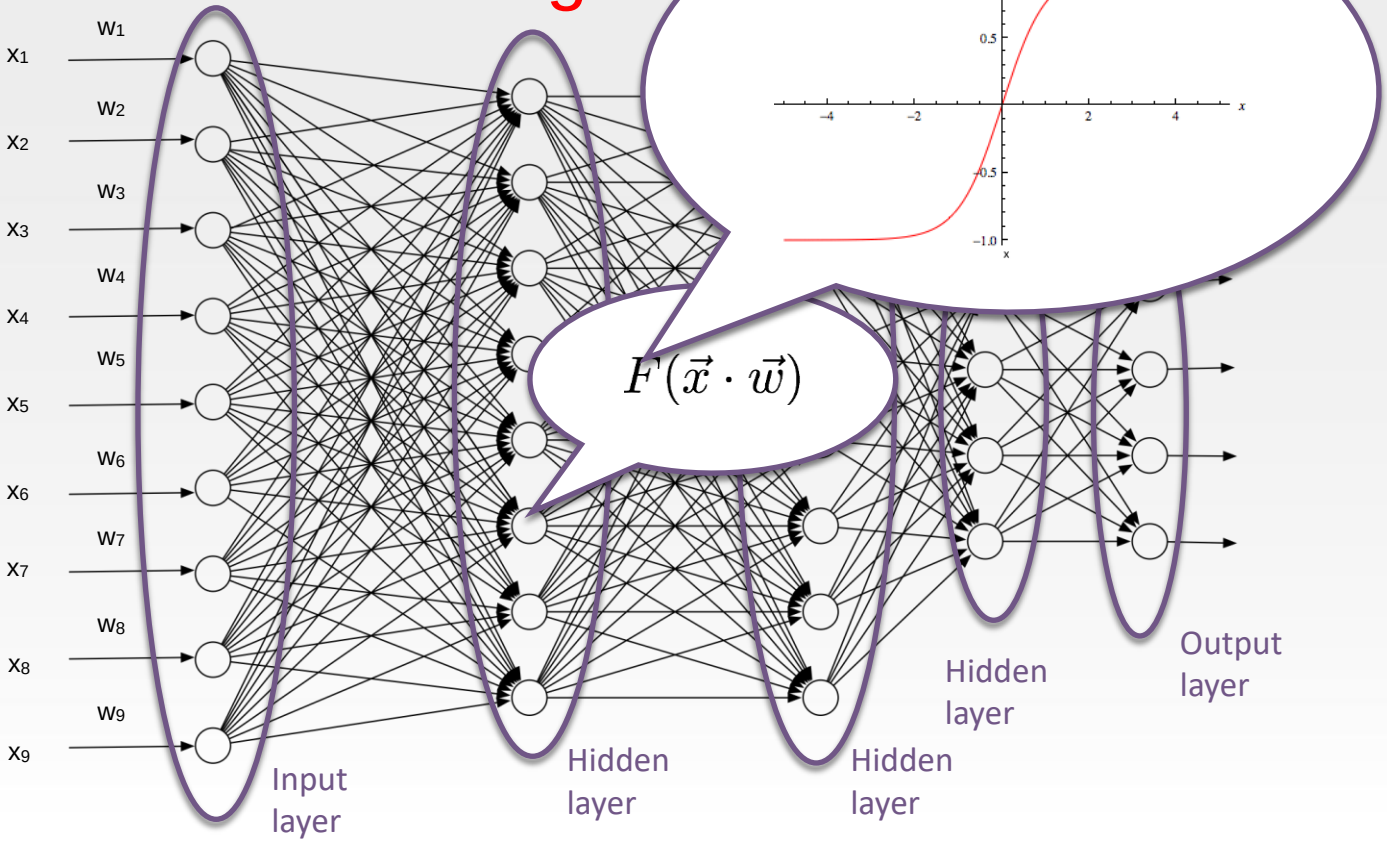
INFO 284 – Machine Learning

Convolutional Neural Networks

Marija Slavkovik & Bjørnar Tessem

UNIVERSITY OF BERGEN

Forward feeding ne

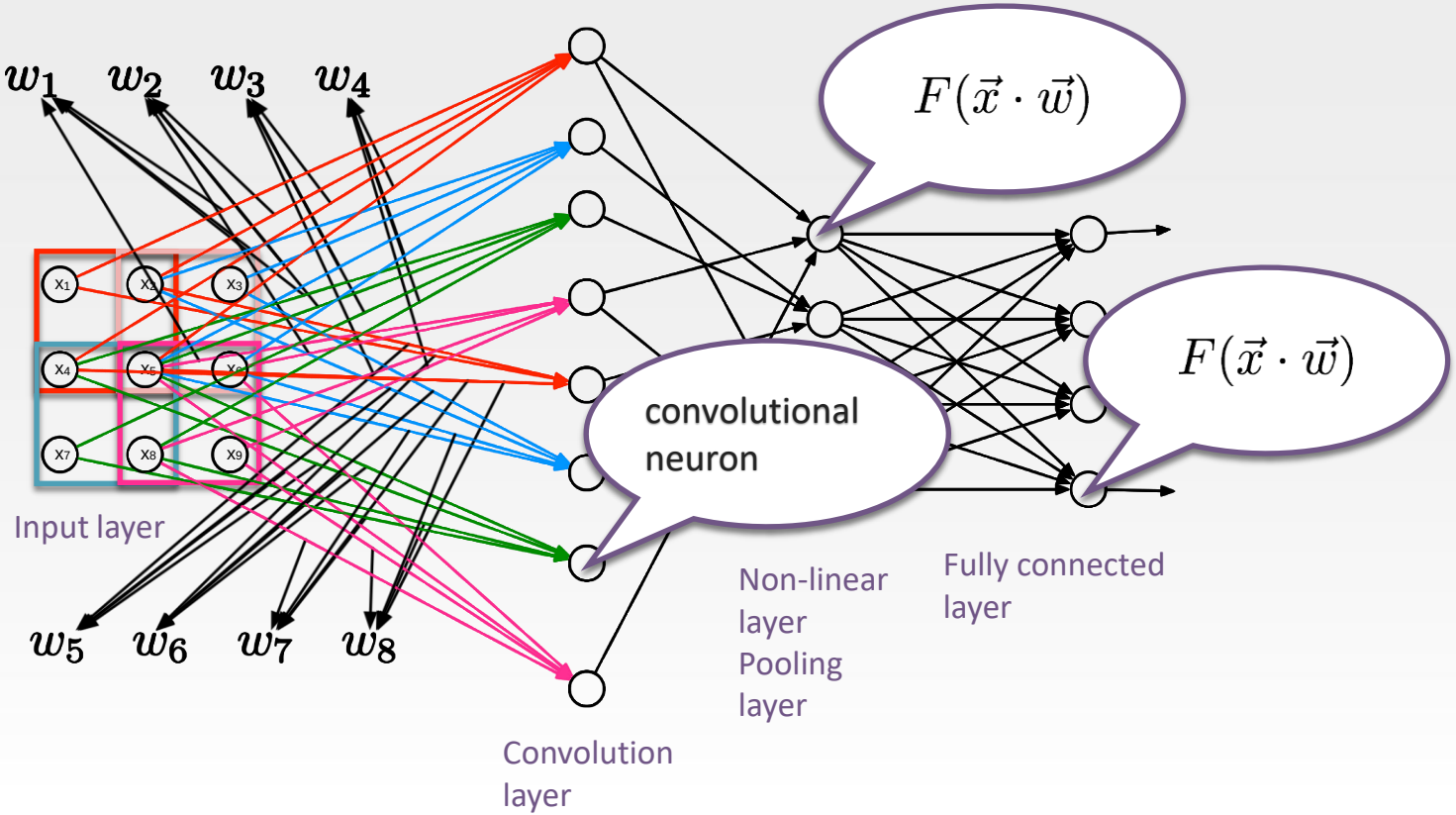


Convolutional networks

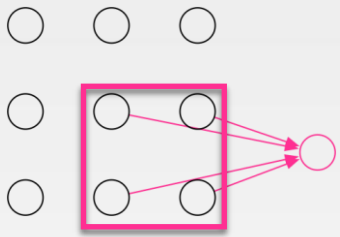
- Convolutional networks are a forward feeding neural networks type with a special kind of architecture, specially constructed for image classification
- Differences with forward feeding networks:
  - All neutrons in a layer is not always connected with all neurons in next layer
  - Hidden layers can be: convolutional, pooling, and fully connected, (and activation layers)
- Convolutional neural networks were already used in the 1970s, but the modern version has been proposed by LeCun et al. 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.

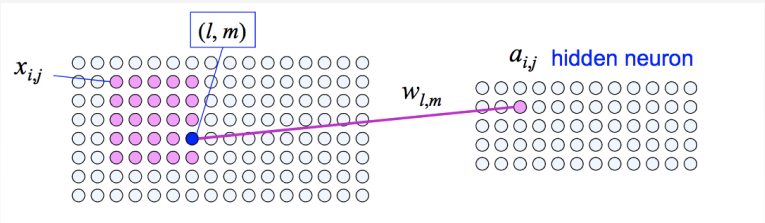
Convolutional neural network



Convolutional layer



Receptive field



- The name “convolutional” comes from the fact that the operation carried out by each neuron is also known as a convolution. The activation value of the convolutional neuron can be expressed as:

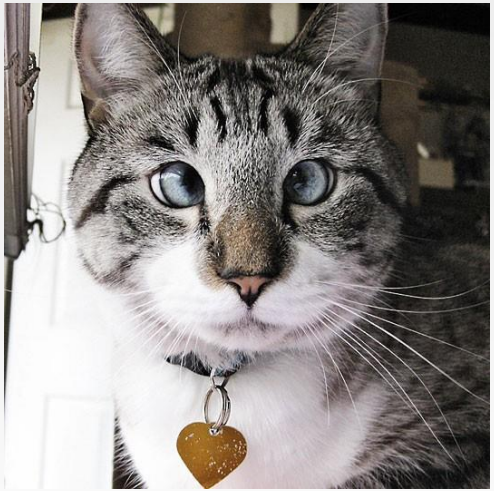
$$a_{i,j} = b + \sum_{l=0}^{r} \sum_{m=0}^{r} w_{l,m} x_{i+l,j+m}$$

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

where the vector $x_{i,j}$ is the r×r receptive field matrix at location (i,j), w is the weight matrix (also r×r) and * is the convolution

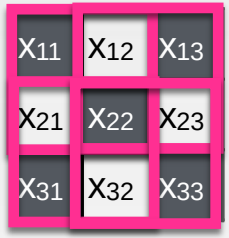
Why convolutional layers

28 x 28 x 1 = 728

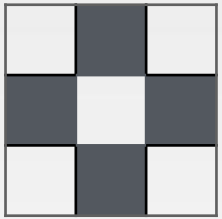


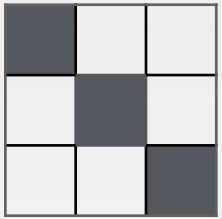
536 x 536 x 3 = 861,888

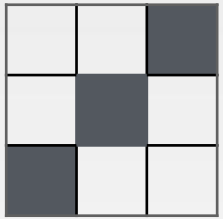
How does a convolution layer work?



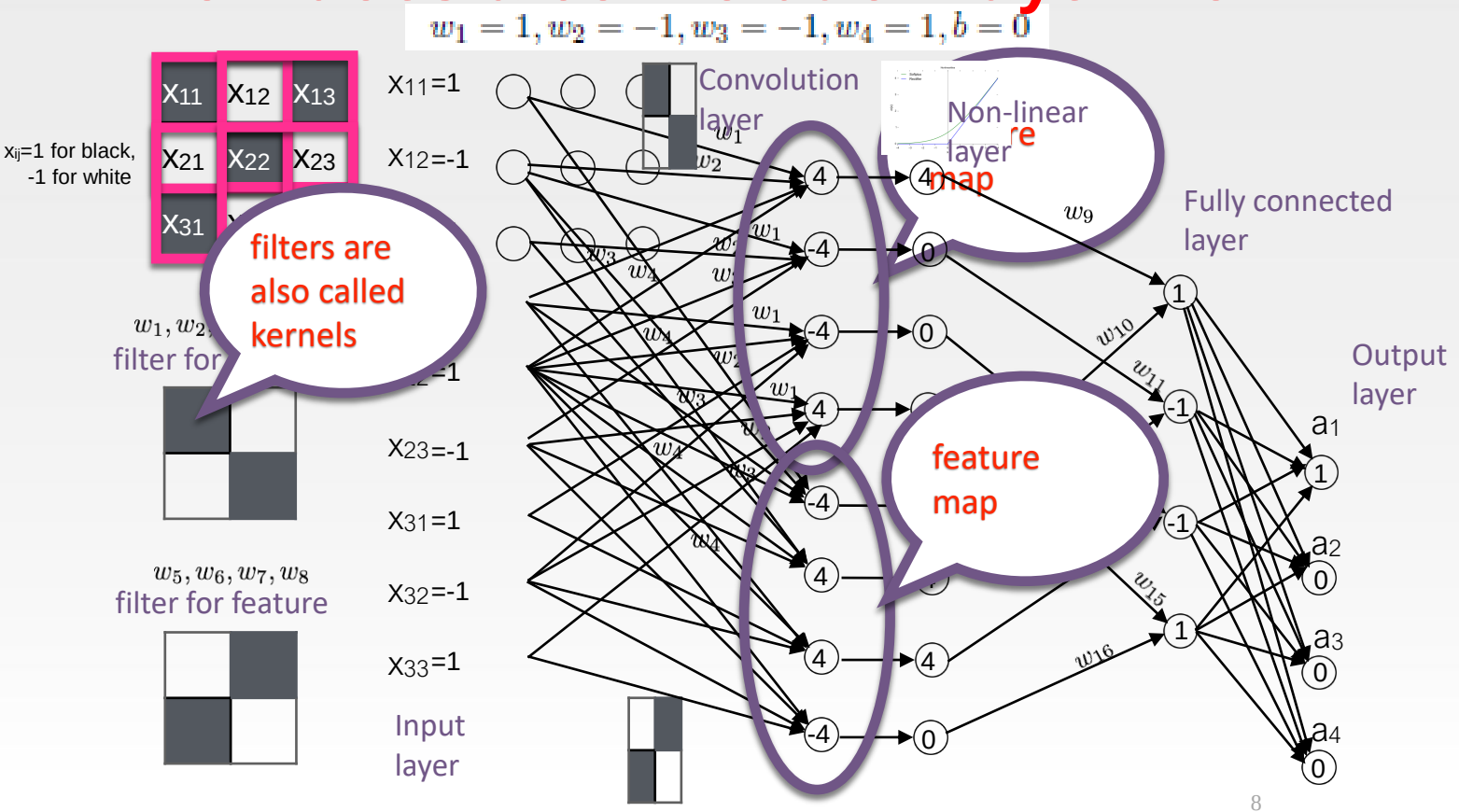
x_{ij}=1 for black,
-1 for white



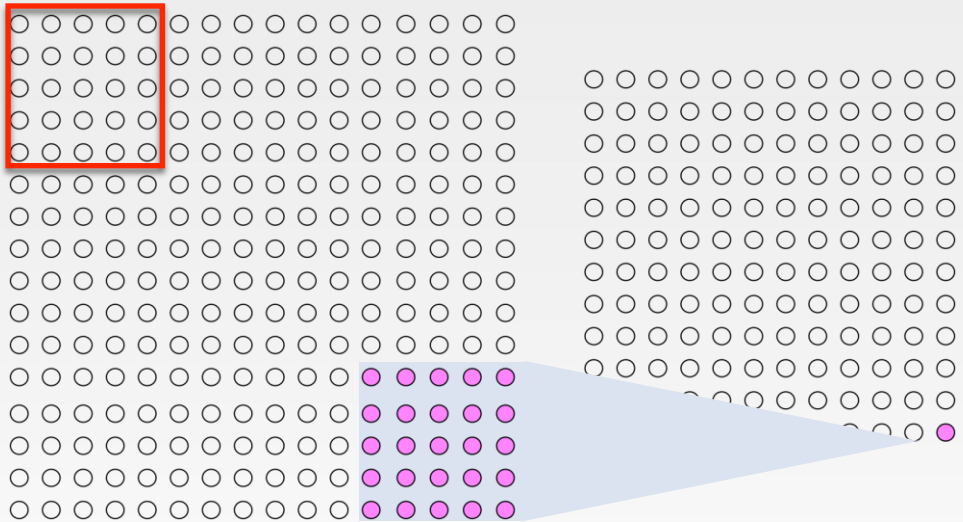




How does a convolution layer work?



Terminology



- Stride length - by how much is the receptive field moved

- If the input layer contains n_l x n_l pixels and the size of local receptive fields is r×r, then the convolutional layer will contain n_c × n_c neurones, where $n_c = n_l – r + 1$.

Hyperparameters

Stride length - by how much is the receptive field moved

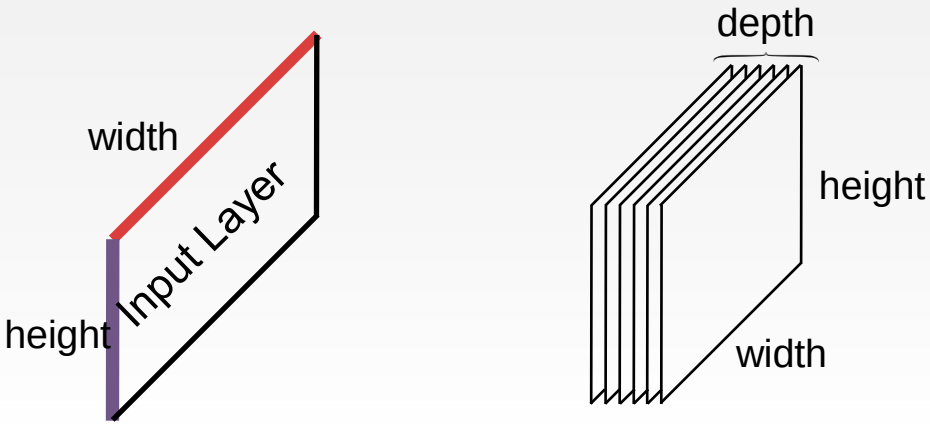
- If the input layer contains n_I x n_I pixels and the stride length is s(>1) while the receptive field is r×r, then the convolutional layer will contain $n_C \times n_C$ neurones, where $n_C = (n_I - r)/s + 1$.
- since n_C must be an integer, s cannot take any value. To alleviate this problem, the input size n_I can be increased by inserting a number of zeros around the border (zero-padding). If P denotes the amount of zero-padding, then

$$n_C = \frac{n_I + 2P - r}{s} + 1$$

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

3D Layers

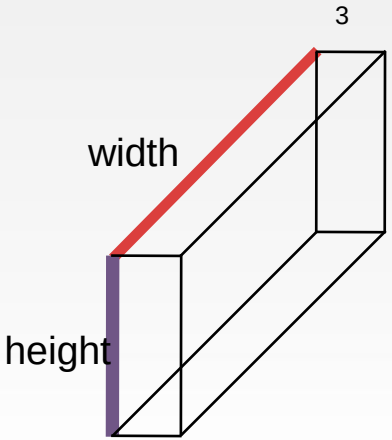
- Often a convolutional layer is represented as a neural volume (tensor):
  - (weight x height) is the size of each feature map
  - depth is the number of feature maps
- To recognise more features a convolutional layer must consists of several different feature maps



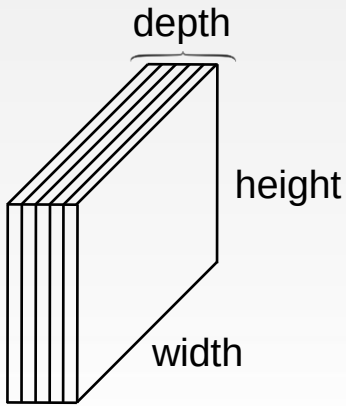
Convolutional Layer

3D Layers

- Colour images have an input volume with depth 3 (RGB), hence the receptive field has also a depth 3.
- Each neuron in the convolutional layer has 3 coordinates (i, j, k), $3r^2$ weights, and a bias. (r = size of receptive field)



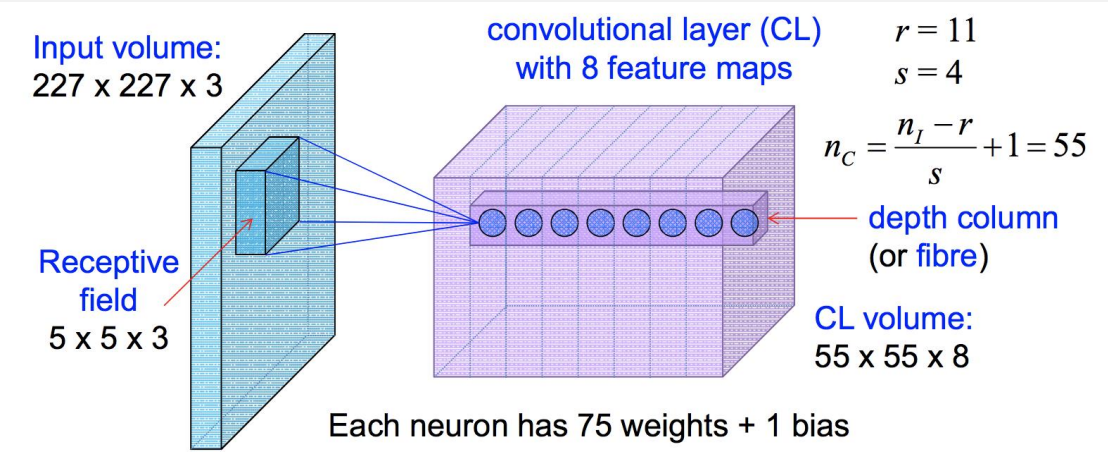
Input Layer



Convolutional Layer

3D Layers

- Each neuron in the Convolutional layer is connected to each pixel of 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

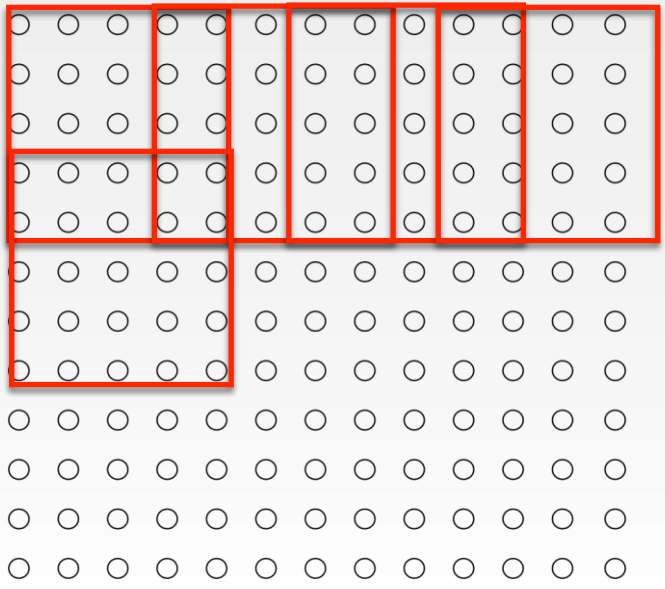
- For each feature map we need r x r + 1 variables. If we have N_F feature maps, a convolutional layer requires:

$$\#\text{variables} = (r^2 + 1)N_F$$

- For instance, if r = 5, N_F = 20 we need 26×20 = 520 variables. **This number does not depend on the size of the input image!**
- By comparison, a fully connected hidden layer with 30 neurones and an image size 28×28=784 would require 784×30 weights, plus 30 biases, for a total of 785×30 = 23,550 parameters (more than 40 times). And this is just for a small image size!
- Clearly, full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting

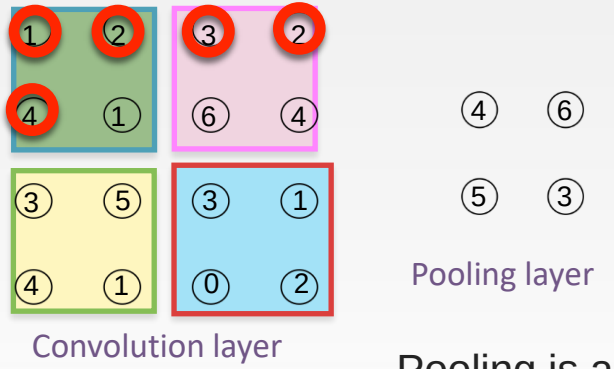
Pooling layers

- A Pooling layer is used after one or several convolutional layers to reduce the information size by merging a small r x r region



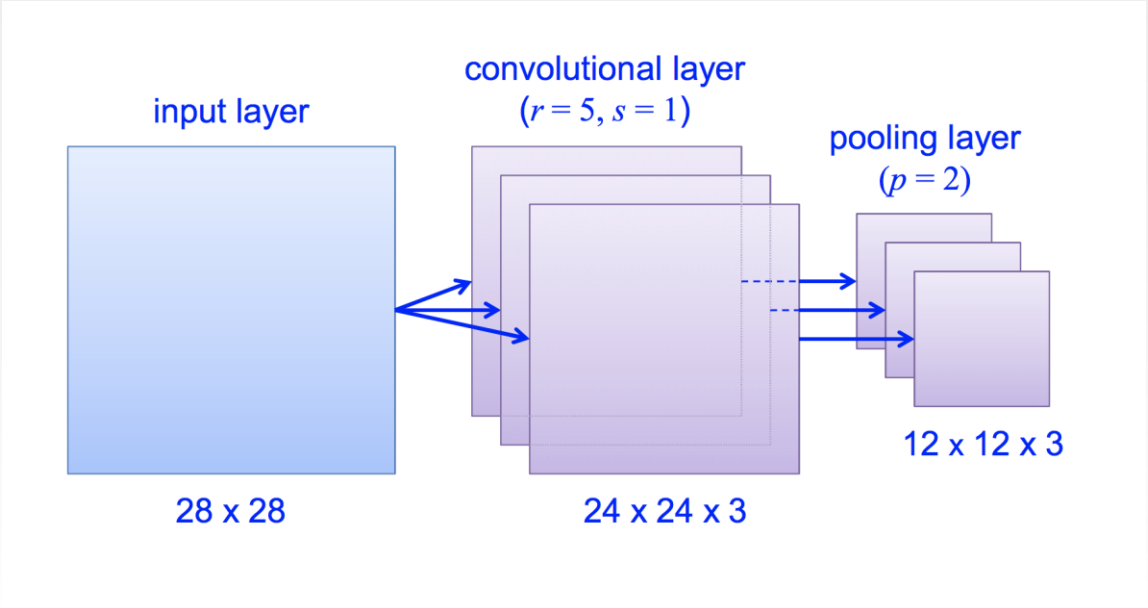
Input layer

Example: Maxpooling takes the maximum from separate regions



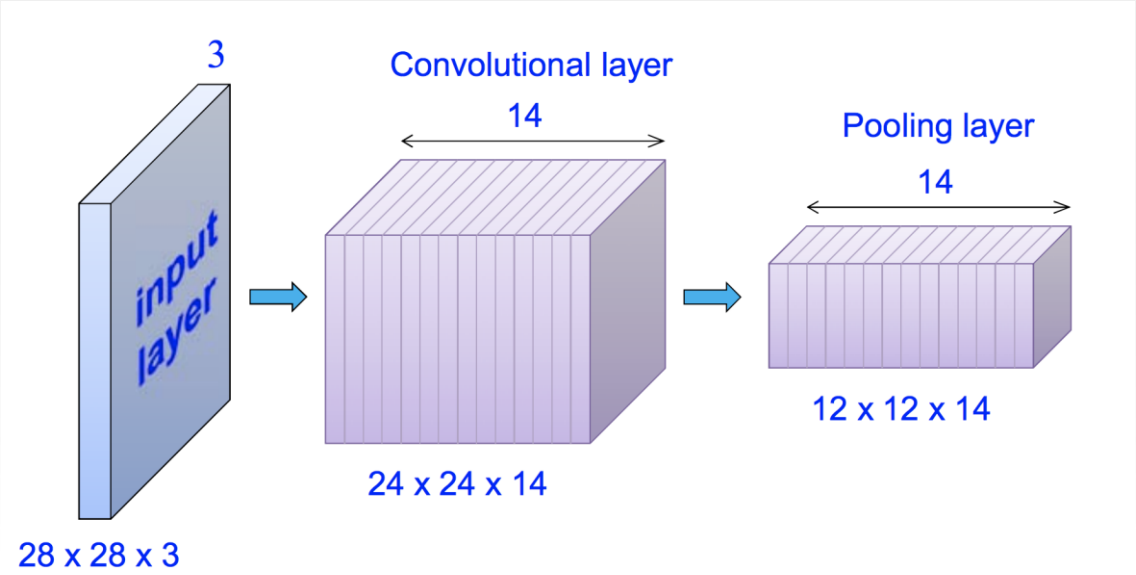
- Pooling is applied to each feature map of the convolutional layer separately

Reduction of complexity



Pooling layer in colour images

- In colour images, we have a 3D view. The pooling layer in the end performs a spatial downsampling preserving the depth of the previous layer:



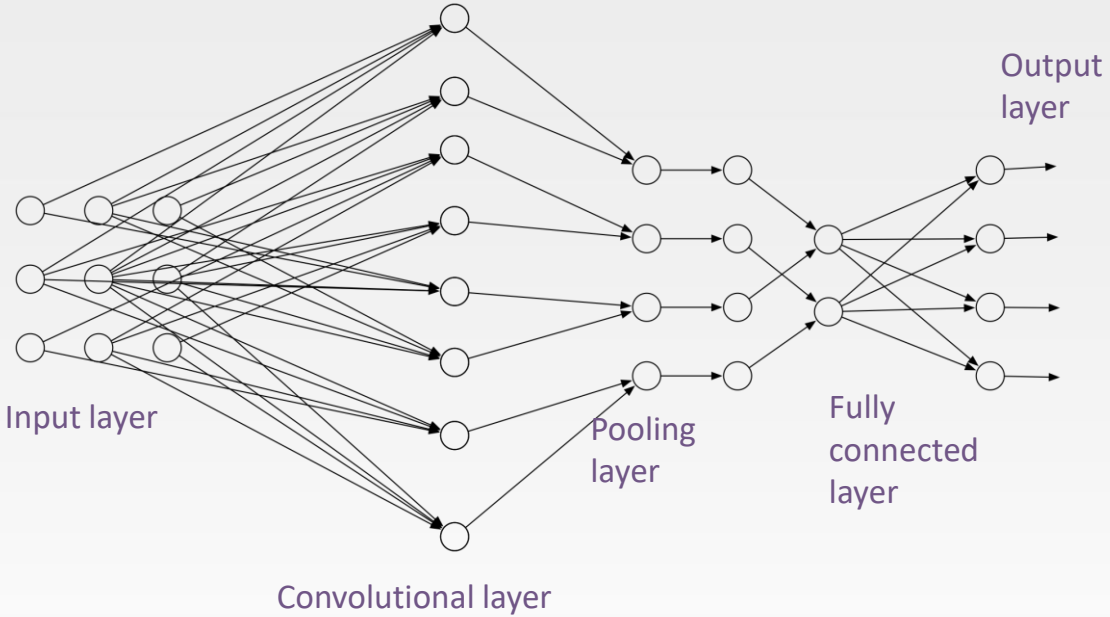
Pooling layers

- Pooling is a way for the network to say whether a given feature is found anywhere in a region of the 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, in which each neurone computes the square root of the sum of the squares in the p×p region:

$$y_{j,k} = \sqrt{\sum_{a=0}^{p-1} \sum_{b=0}^{p-1} x_{j+a,k+b}^2}$$

  - Average polling was used before, but not any more, since both max-pooling and L2-pooling have better performance.

The complete network



Learning?

As for traditional feed forward networks:

- Variant of backpropagation to get weight errors
- Gradient descent optimization

- One or more fully-connected hidden layers can be inserted between the pooling layer and the output layer

Loss functions

- Cross-entropy is used for loss in multi-class classification
  - Output y_i is a number in [0,1]
  - Convert to probability distribution (softmax)

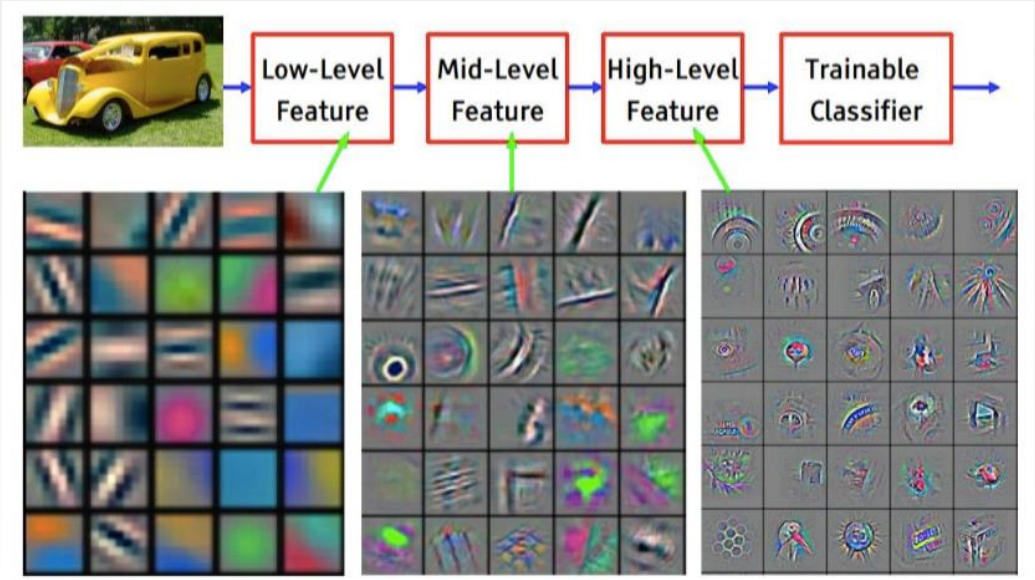
$$p(Class = i) = p_i = \frac{e^{y_i}}{\sum_{j=0}^{n} e^{y_j}}$$

  - Loss:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log p_i + (1 - y_i) \cdot \log(1 - p_i)$$

Learning CNN

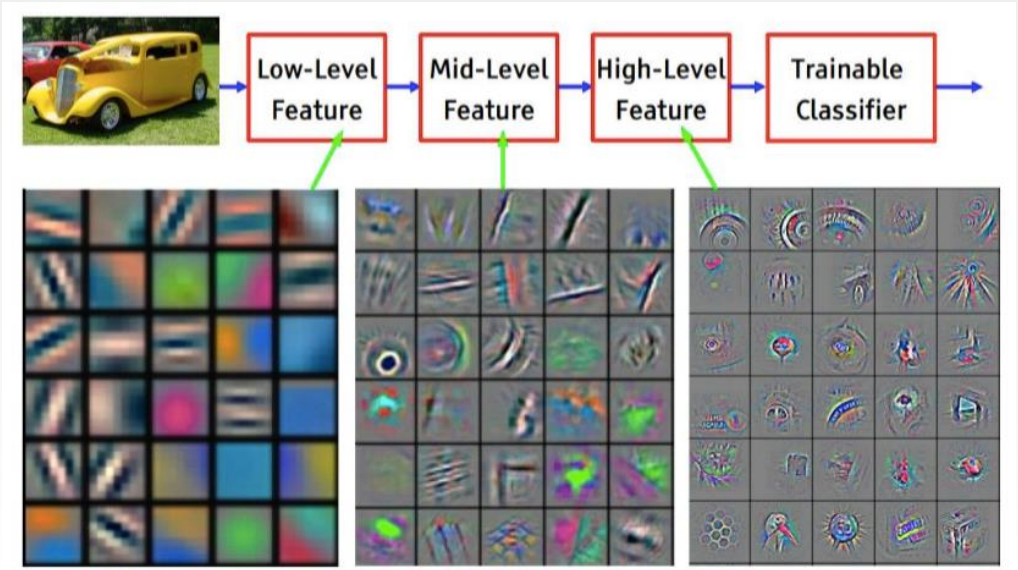
- Convolutional neural networks are trained using stochastic gradient descent and Backpropagation, but some modifications are needed to the Backpropagation algorithm, because it was derived for networks with fully-connected layers.
- In a CNN convolutional and pooling layers learn **local spatial structure** in the input training images;
- Later fully-connected layers learn at more **abstract levels**, integrating global information from across the entire image.



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

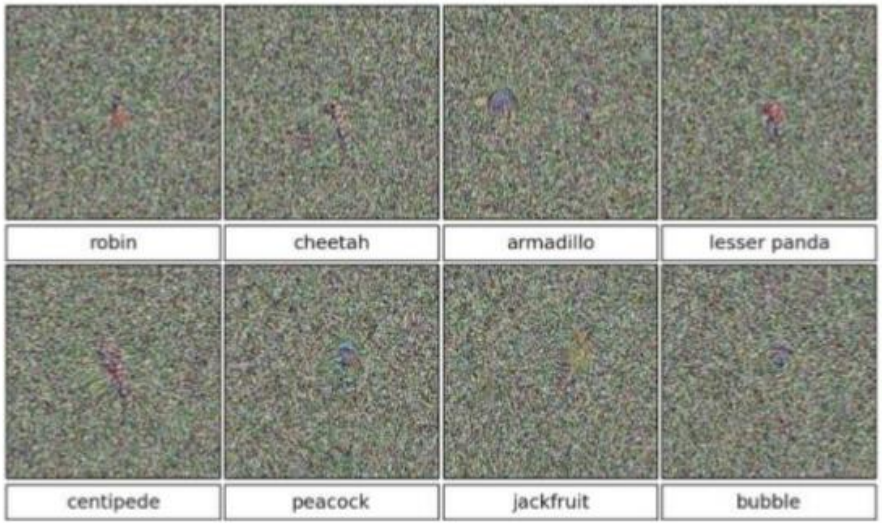
Visualising feature maps

- In practice, convolutional networks use tens of feature maps
- To see the feature maps learned by layer by layer, the corresponding weights can be visualised.
- Feature maps clearly indicate that the network is learning a spatial structure, but it is not easy to see what they are learning. Today, a lot of work is done to better understand such features.



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

Fooling the network



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

CNN in Python

- Many software frameworks are available
- Examples
  - TensorFlow: Google, C++ https://www.tensorflow.org/
  - Theano: Univ of Montreal, Python, https://github.com/Theano/Theano
  - Torch: C++, Lua, Python http://pytorch.org/
  - Tiny-Dnn: C++ https://github.com/tiny-dnn
  - Start with: Pytorch https://pytorch.org/

uib.no