UniGe | MaLGa

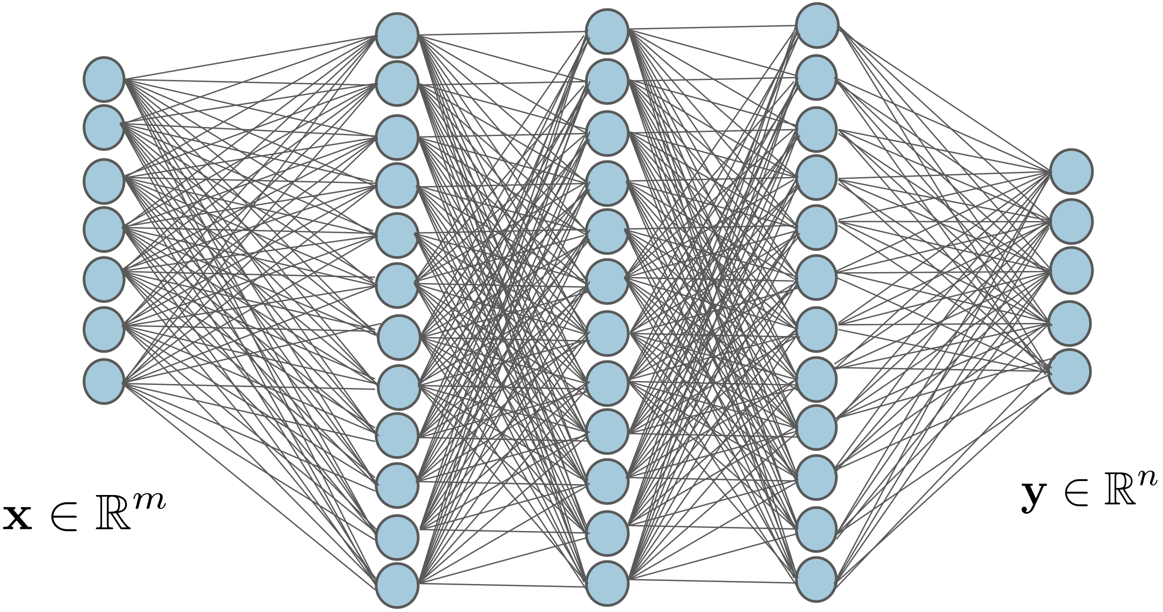
Convolutional Neural Network

Deep Learning

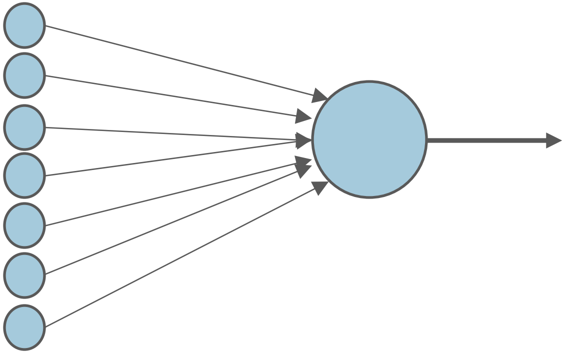
Nicoletta Noceti

A refresh

Deep Neural Network Recap

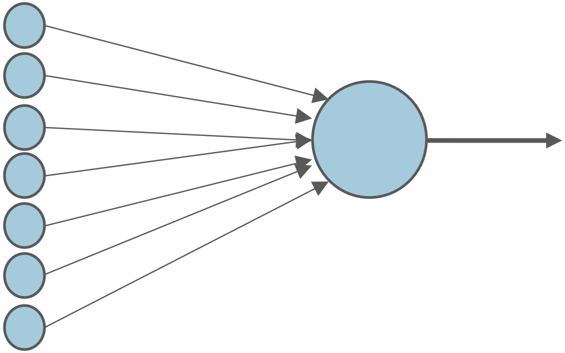


The role of a neuron



$$y = f(\mathbf{x}) = \sigma(\sum_i w_i x_i + b)$$

The role of a neuron



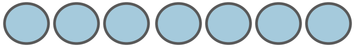
- Each neuron is connected to all the others
- Correlations between input are not taken into account
- As the size of the input and the depth of the architecture increase, the number of parameters increases dramatically

$$y = f(\mathbf{x}) = \sigma(\sum_i w_i x_i + b)$$

Convolutional Neural Networks

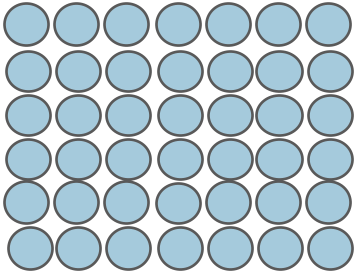
- A specialized kind of neural network for processing data with a known grid-like topology

- Examples:
  - Time-series



1D grid

  - Images



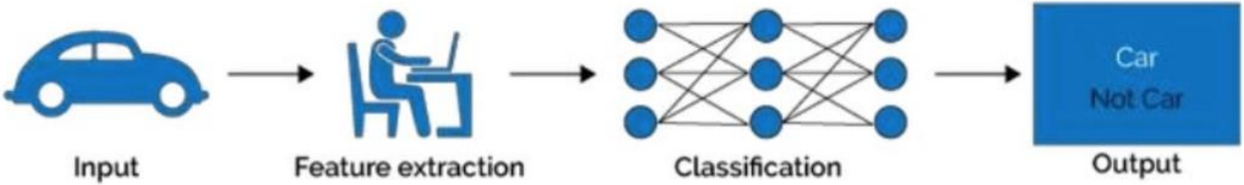
2D grid

NNs don’t scale to images!

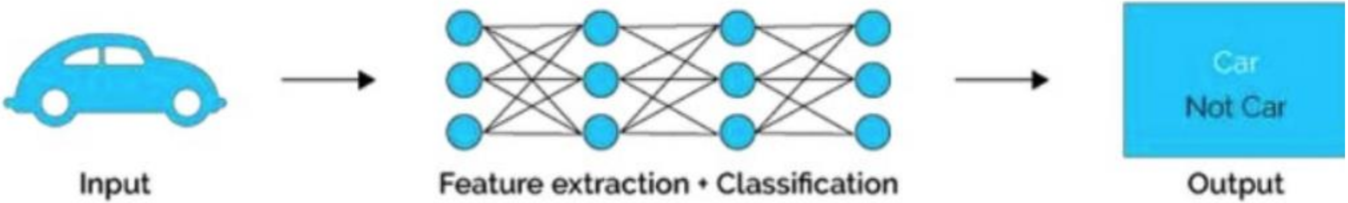
- Let us consider a fully connected network with a single unit
  - Tiny color images of size 32 x 32 x 3
    - Size of the input layer: 32 x 32 x 3 = 3072
    - Size of the weights: 3072
  - Small color images of size 200 x 200 x 3
    - Size of input layer and weights: 200 x 200 x 3 = 120000

From shallow to deep models

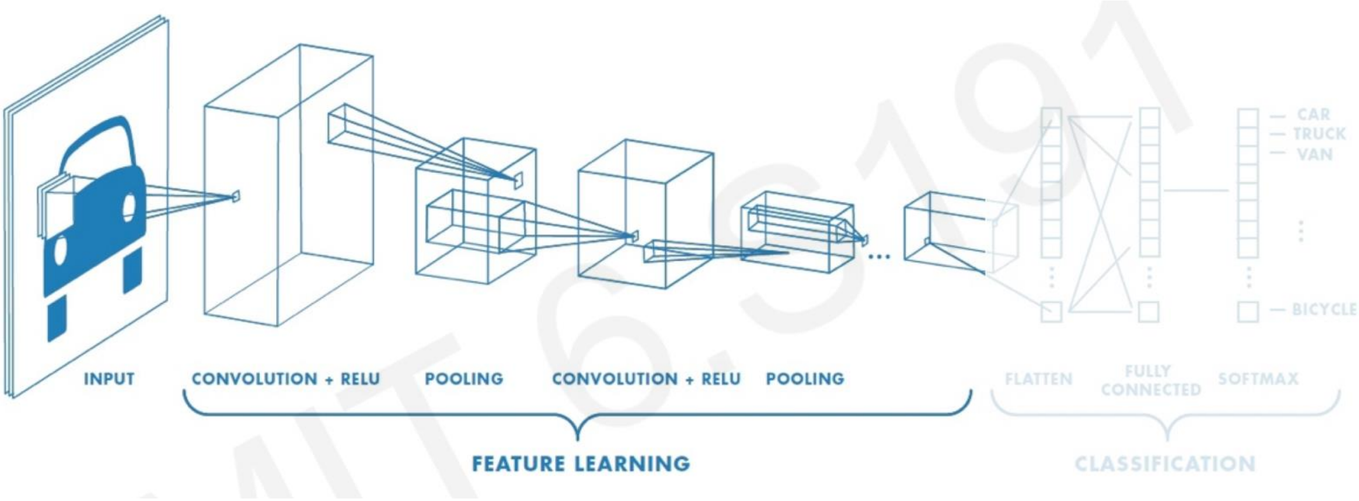
Shallow models



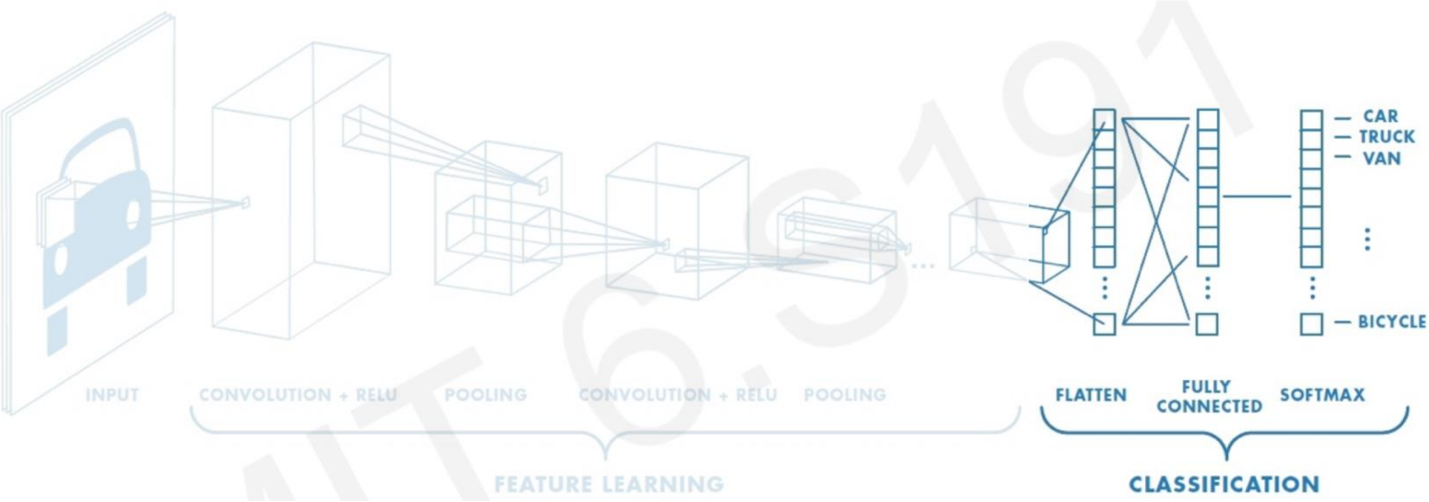
Deep models



A typical CNN



A typical CNN



$$softmax(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

Interlude: convolution

The convolution/cross-correlation operation

For 2D input arrays:

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) =$$

$$= \sum_m \sum_n I(i-m, j-n)K(m,n)$$

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Cross-correlation (with an example)

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

$$\begin{array}{|c|c|c|c|} \hline X_{11} & X_{12} & X_{13} & X_{14} \\ \hline X_{21} & X_{22} & X_{23} & X_{24} \\ \hline X_{31} & X_{32} & X_{33} & X_{34} \\ \hline X_{41} & X_{42} & X_{43} & X_{44} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|} \hline W_{11} & W_{12} \\ \hline W_{21} & W_{22} \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|} \hline Y_{11} & Y_{12} & Y_{13} \\ \hline Y_{21} & Y_{22} & Y_{23} \\ \hline Y_{31} & Y_{32} & Y_{33} \\ \hline \end{array}$$

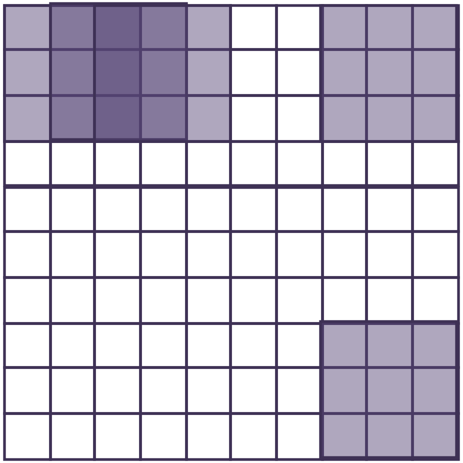
$Y_{11} = X_{11}W_{11} + X_{12}\,W_{12} + X_{21}\,W_{21} + X_{22}\,W_{22}$
$Y_{12} = X_{12}W_{11} + X_{13}\,W_{12} + X_{22}\,W_{21} + X_{23}\,W_{22}$
$Y_{13} = X_{13}W_{11} + X_{14}\,W_{12} + X_{23}\,W_{21} + X_{24}\,W_{22}$
....

UniGe | MaLGa

2D convolution

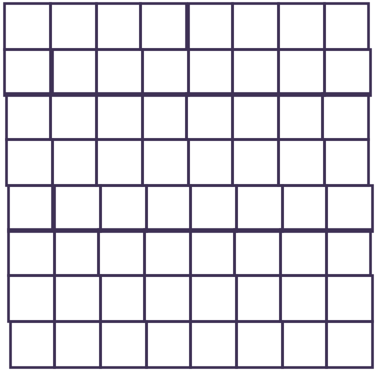
A "feature detector" (kernel) slides over the inputs to generate a feature map

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$



Input tensor of size 10x10

Kernel of size 3x3

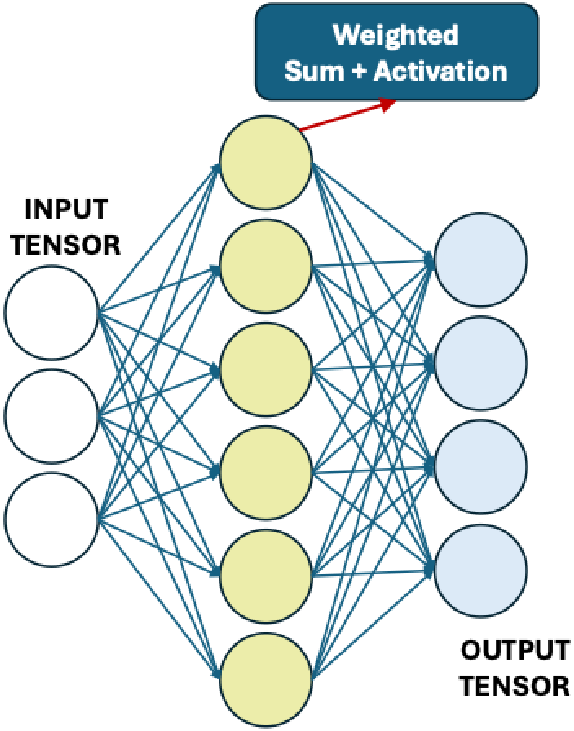


Output tensor of size 8x8

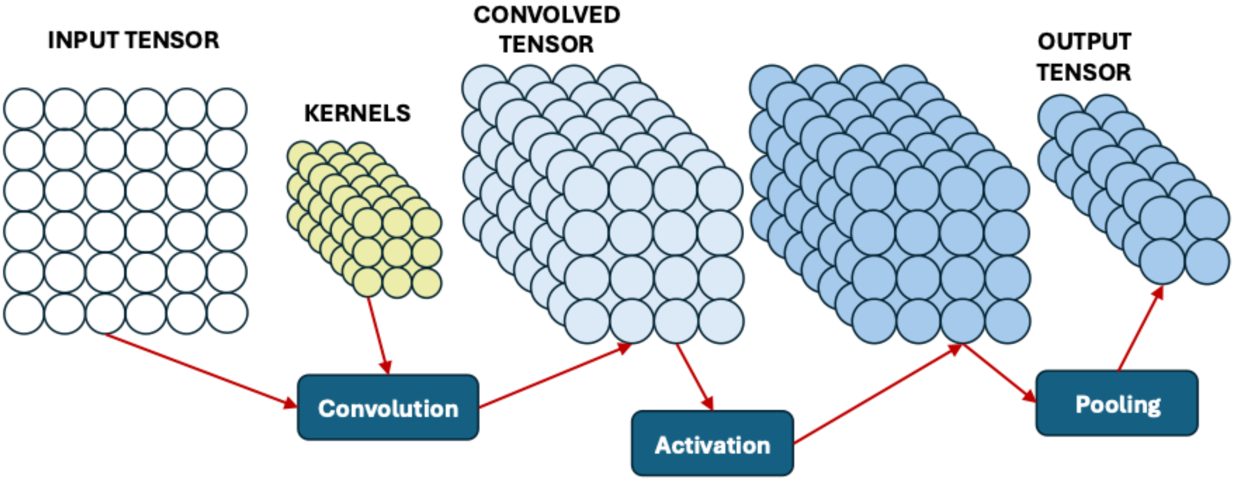
UniGe | MaLGa

Convolutional layer

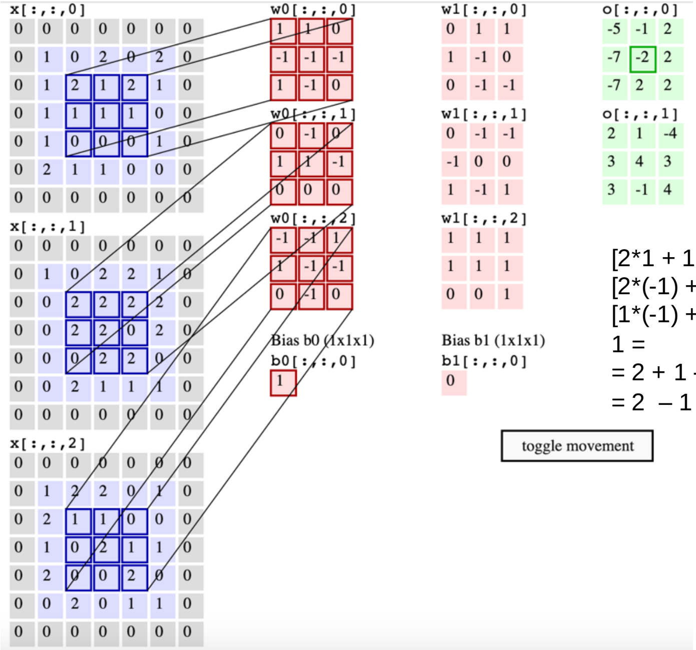
A sketch of a dense layer



A sketch of a convolutional layer



An example from https://cs231n.github.io/convolutional-networks/



x[:,:,0]

0	0	0	0	0	0	0
0	1	0	2	0	2	0
0	1	2	1	2	1	0
0	1	1	1	1	0	0
0	1	0	0	0	1	0
0	2	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	2	2	1	0
0	0	2	2	2	2	0
0	0	2	2	0	2	0
0	0	0	2	2	0	0
0	0	2	1	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	2	2	0	1	0
0	2	1	1	0	0	0
0	1	0	2	1	1	0
0	2	0	0	2	0	0
0	0	2	0	1	1	0
0	0	0	0	0	0	0

w0[:,:,0]

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

w0[:,:,1]

0	-1	0
1	1	-1
0	0	0

w0[:,:,2]

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

Bias b0 (1x1x1)
b0[:,:,0]

1

w1[:,:,0]

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

w1[:,:,1]

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

w1[:,:,2]

1	1	1
1	1	1
0	0	1

Bias b1 (1x1x1)
b1[:,:,0]

0

o[:,:,0]

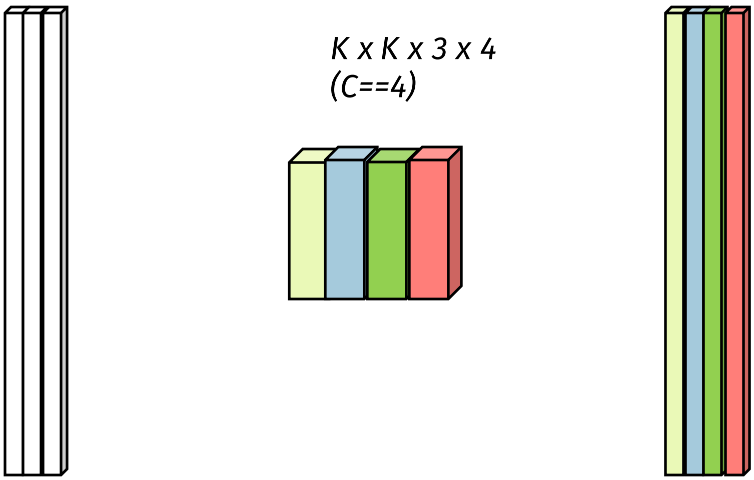
-5	-1	2
-7	-2	2
-7	2	2

o[:,:,1]

2	1	-4
3	4	3
3	-1	4

[2*1 + 1*1 + 1*(-1) + 1*(-1) + 1*(-1)]+
[2*(-1) + 2*1 + 2*1]+
[1*(-1) + 1*(-1) + 2*(-1) + 1*(-1)]+
1 =
= 2 + 1 - 1 – 1 - 1 - 2 + 2 + 2 – 1 – 1 – 2 – 1+1
= 2 – 1 - 1 – 1 – 1 = -2

Another animation to clearly understand

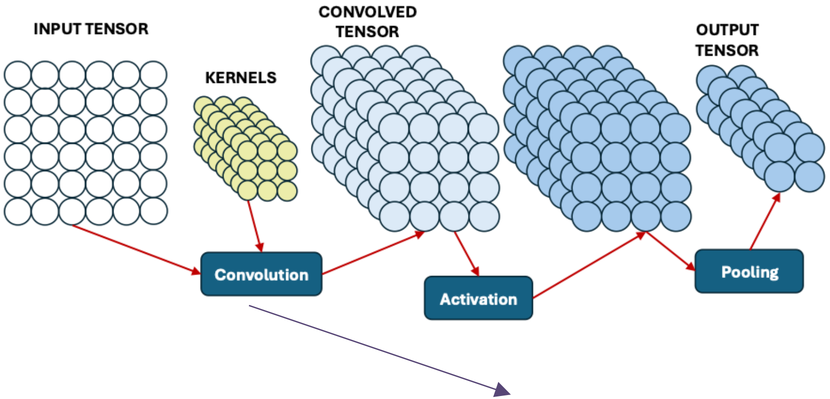


W x H x 3

K x K x 3 x 4
(C==4)

W^l x H^l x 4
(C==4)

A sketch of a convolutional layer

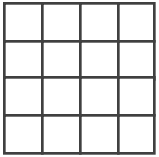


Also called the detector stage, it provides a set of linear activations

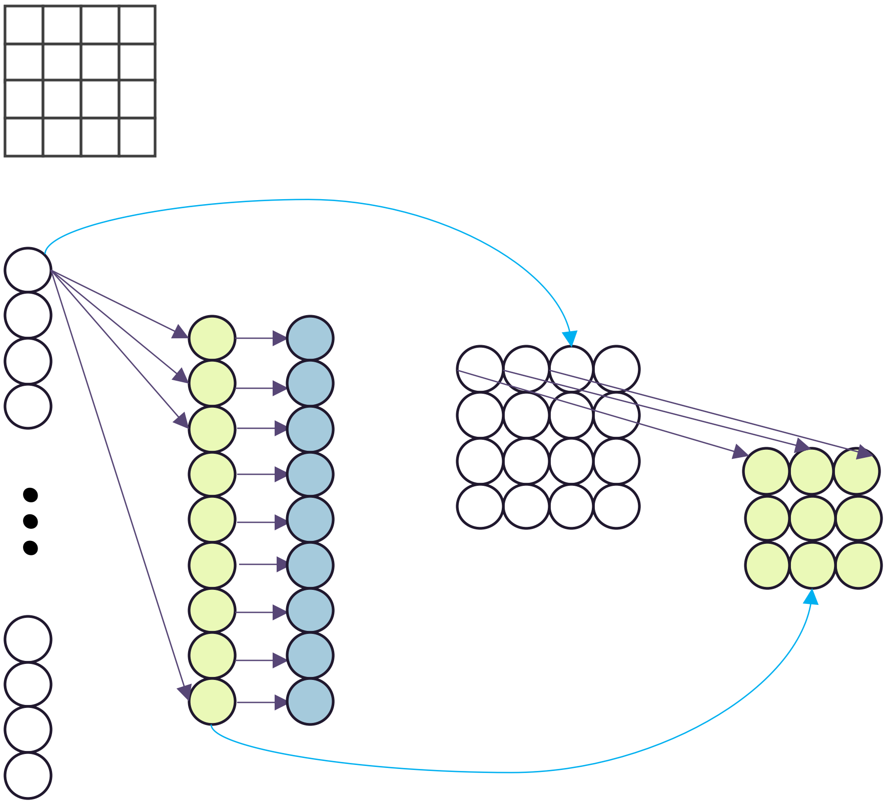
As the kernel slides on the image, it is able to capture the same property in different image regions → THERE IS PARAMETER SHARING

Multiple feature detectors can be used to capture different image properties → Their number is called channels

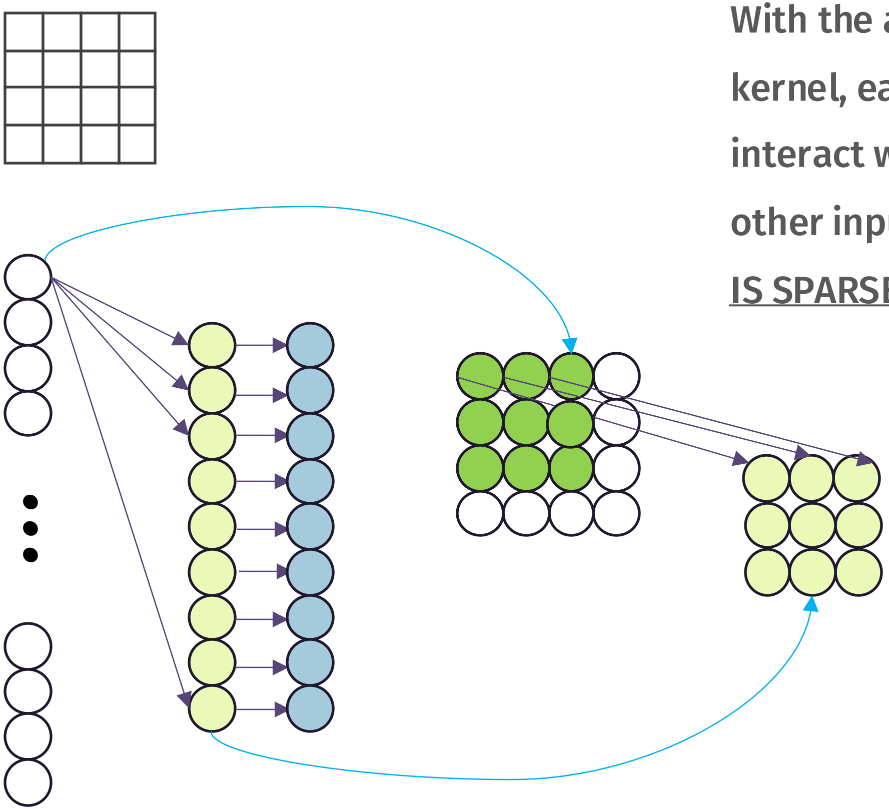
From dense to sparse interaction



From dense to sparse interaction



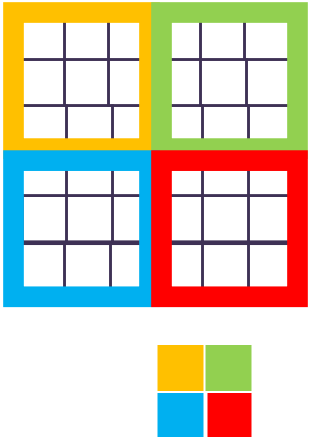
From dense to sparse interaction



With the application of the kernel, each input elements interact with only a subset of other input elements → THERE IS SPARSE INTERACTION

Output feature size of conv layers

- Three parameters control the size of the output of a layer
  - **Channles**, the number of filters (kernels) of the layer
  - **Stride,** the step used to slide the filter on the input
    - When stride > 1 we are down-sampling the input data
    - **Tiling** refers to the special case where stride = kernel span
  - **Padding** to enlarge the input and allow for kernels application in each one of the (original) point



Output features size of conv layers

Size BEFORE convolution

Kernel size

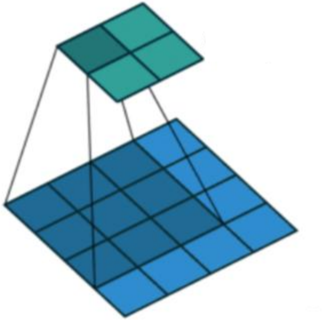
Padding

$$O = \frac{W - K + 2P}{S} + 1$$

Size AFTER convolution

Stride

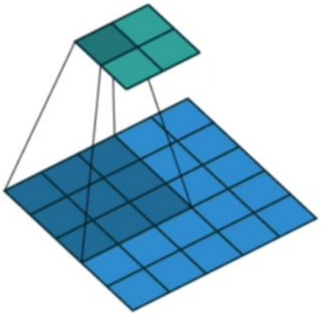
Output features size of conv layers
Examples



No padding, stride 1

$$\underset{2}{O} = \frac{\overset{4}{W} - \overset{3}{K} + 2\overset{0}{P}}{\underset{1}{S}} + 1$$

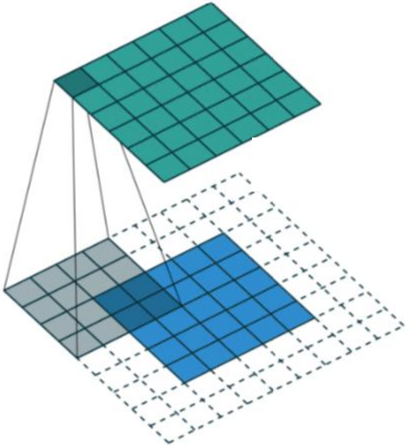
Output features size of conv layers
Examples



No padding, stride 2

$$\underset{2}{O} = \frac{\overset{5}{W} - \overset{3}{K} + 2\overset{0}{P}}{\underset{2}{S}} + 1$$

Output features size of conv layers
Examples



Padding 2, stride 1

$$\underset{6}{O} = \frac{\overset{5}{W} - \overset{4}{K} + 2\overset{2}{P}}{\underset{1}{S}} + 1$$

To sum up: output features size of conv layers

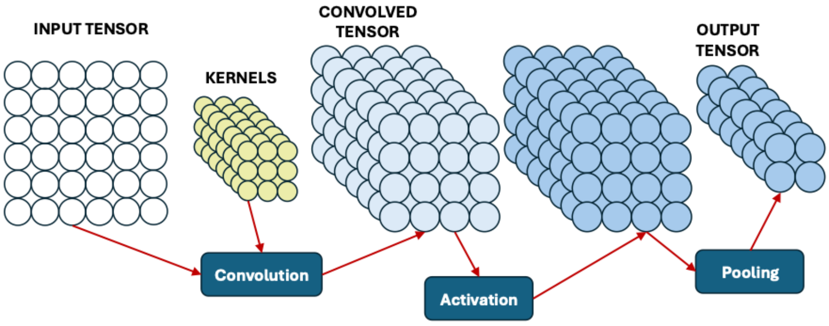
- Input size: W_1 x H_1 x D_1
- Parameters:
  - Number of kernels N
  - Kernel size K
  - Stride S
  - Padding P

- Output size: W_2 x H_2 x D_2

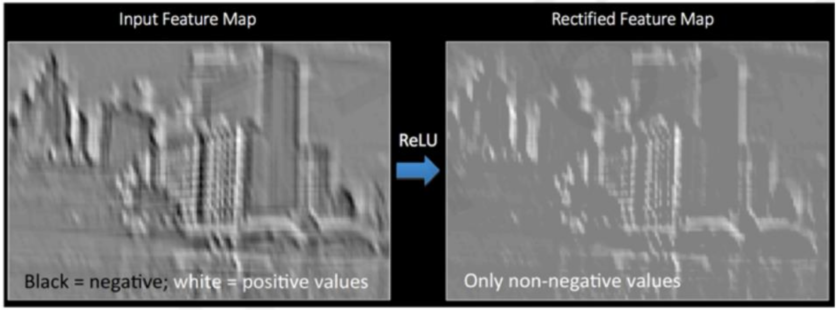
  where
- $W_2 = (W_1 - K +2P)/S + 1$
- $H_2 = (H_1 – K + 2P)/S + 1$
- $D_2 = N$

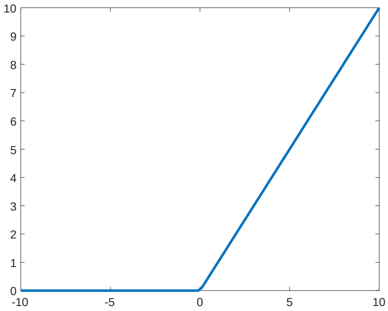
- Number of weights per filter: K x K x D_1
- Number of parameters in total:
  - K x K x D_1 x N weights
  - N biases

A sketch of a convolutional layer



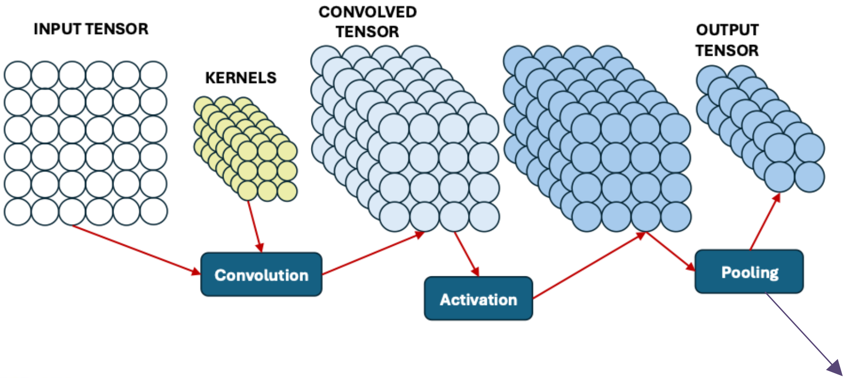
A typical choice: ReLU

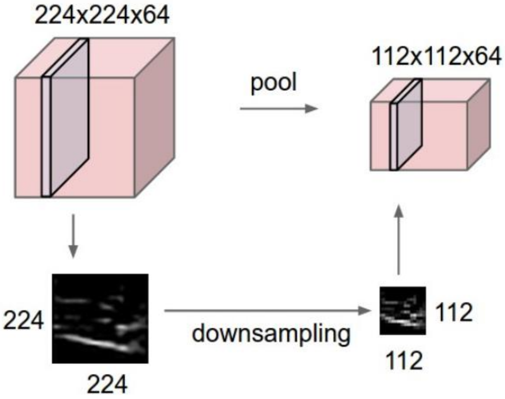




$$f(x) = max(0, x)$$

A sketch of a convolutional layer

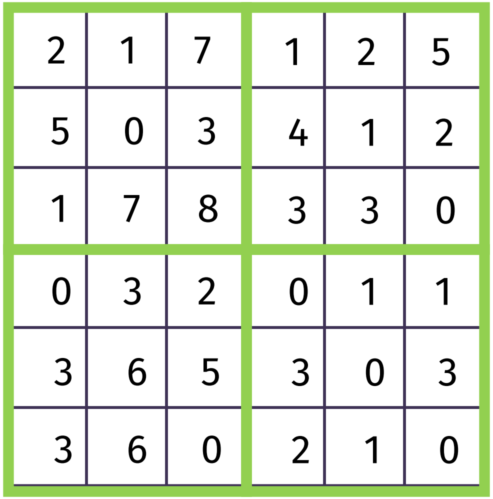




A way to further reduce the dimensionality of the representation while providing invariance to small shifts of the inputs

Common choices: average or max pooling

Pooling with an example



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

Max pooling

8	5
6	3

Average pooling

3.8	2.3
3	1.2

Pooling can help with local invariance although some information is lost

No parameter to be estimated here!

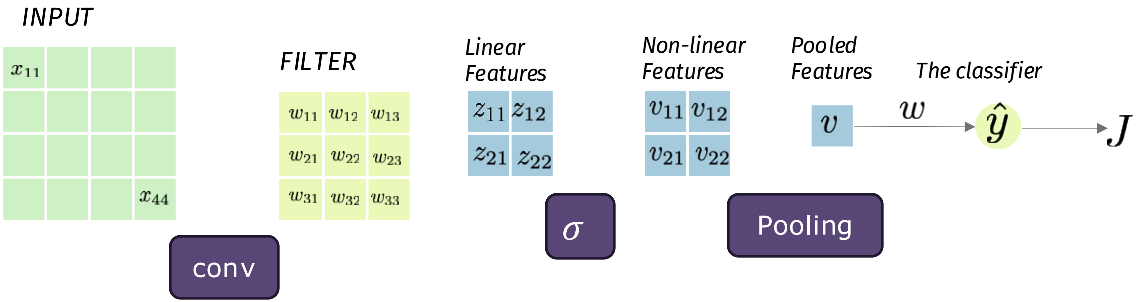
To sum up: output features size of pooling layer

- Input size: W_1 x H_1 x D_1
- Parameters:
  - Window size H
  - Stride S

- Output size: W_2 x H_2 x D_2 where
- $W_2 = (W_1 - H)/S + 1$
- $H_2 = (H_1 – H)/S + 1$
- $D_2 = D_1$
- Number of weights per filter: K x K x D_1
- Number of parameters in total:
  - K x K x D_1 x N weights
  - N biases

Backpropagation in CNNs

Backgropragation in CNNs (intuition)



INPUT

x_{11}			
			x_{44}

conv

FILTER

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Linear Features

z_{11}	z_{12}
z_{21}	z_{22}

σ

Non-linear Features

v_{11}	v_{12}
v_{21}	v_{22}

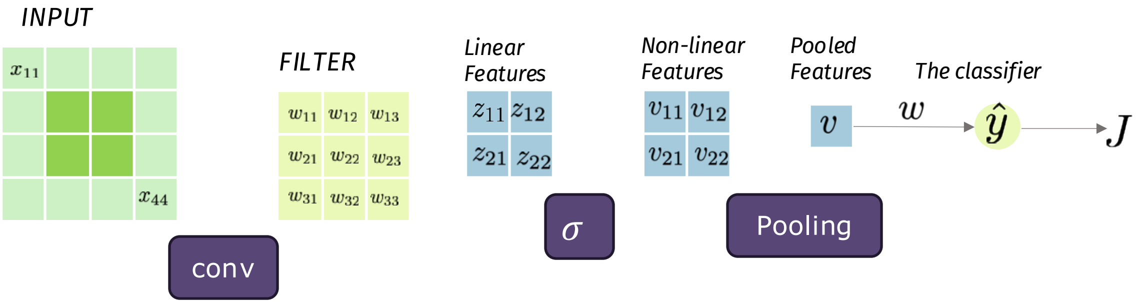
Pooling

Pooled Features

The classifier

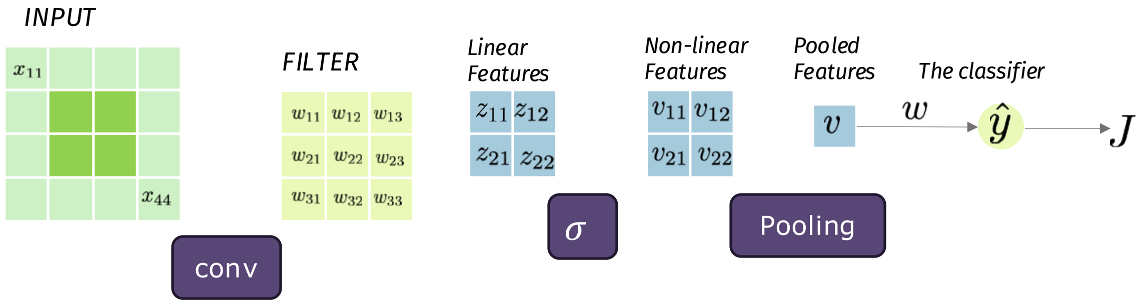
$$v \xrightarrow{w} \hat{y} \rightarrow J$$

Backgropragation in CNNs (intuition)



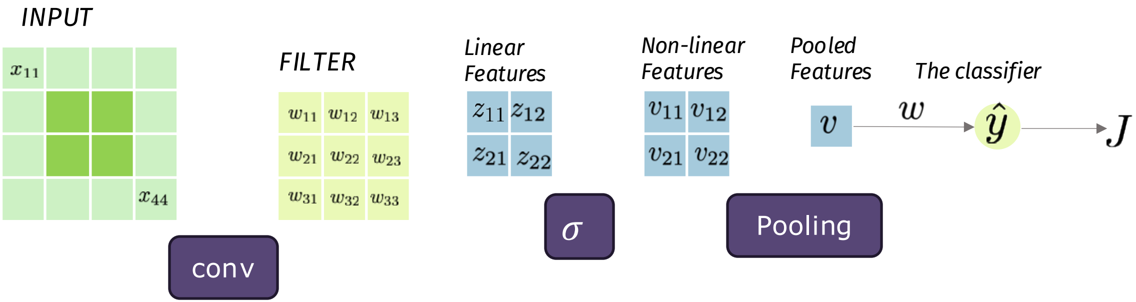
UniGe | MaLGa

Backgropragation in CNNs (intuition)



$$z_{r,c} = \sum_{i=1}^{3} \sum_{j=1}^{3} w_{i,j} x_{r+i-1,c+j-1}$$

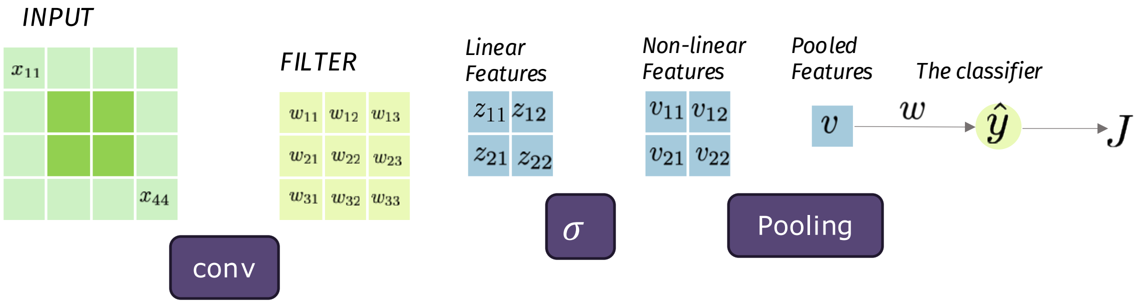
Backgropragation in CNNs (intuition)



$$z_{r,c} = \sum_{i=1}^{3} \sum_{j=1}^{3} w_{i,j} x_{r+i-1,c+j-1}$$

$$v_{s,t} = f_{\sigma}(z_{s,t})$$

Backgropragation in CNNs (intuition)

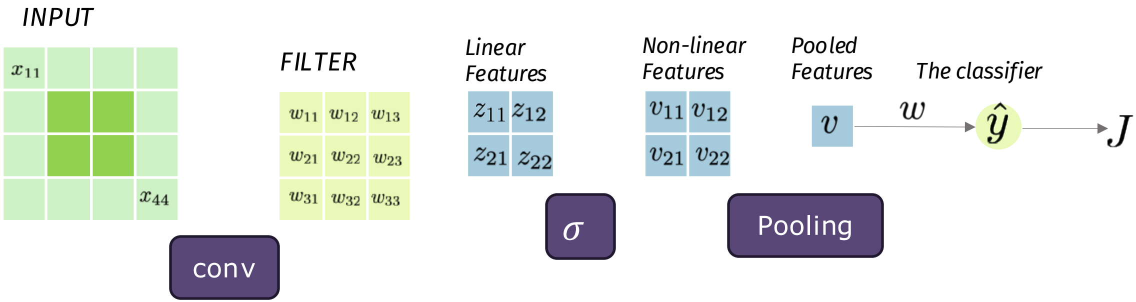


$$z_{r,c} = \sum_{i=1}^{3} \sum_{j=1}^{3} w_{i,j} x_{r+i-1,c+j-1}$$

$$v_{s,t} = f_{\sigma}(z_{s,t})$$

$$v = \frac{1}{4} \sum_{s=1}^{2} \sum_{t=1}^{2} v_{s,t}$$

Backgropragation in CNNs (intuition)



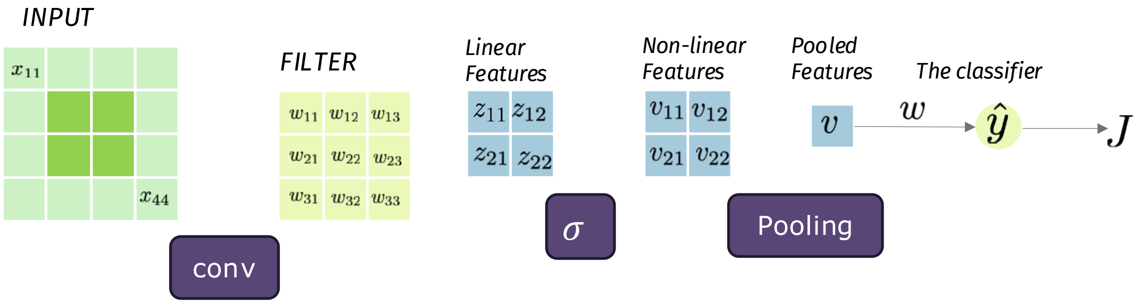
$$z_{r,c} = \sum_{i=1}^{3} \sum_{j=1}^{3} w_{i,j} x_{r+i-1,c+j-1}$$

$$\hat{y} = wv$$

$$v_{s,t} = f_\sigma(z_{s,t})$$

$$v = \frac{1}{4} \sum_{s=1}^{2} \sum_{t=1}^{2} v_{s,t}$$

Backgropragation in CNNs (intuition)



INPUT

$$\begin{array}{cccc} x_{11} & & & \\ & & & \\ & & & \\ & & & x_{44} \end{array}$$

FILTER

$$\begin{array}{ccc} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{array}$$

Linear Features

$$\begin{array}{cc} z_{11} & z_{12} \\ z_{21} & z_{22} \end{array}$$

Non-linear Features

$$\begin{array}{cc} v_{11} & v_{12} \\ v_{21} & v_{22} \end{array}$$

Pooled Features

v

The classifier

$v \xrightarrow{w} \hat{y} \rightarrow J$

conv

σ

Pooling

$$z_{r,c} = \sum_{i=1}^{3}\sum_{j=1}^{3} w_{i,j} x_{r+i-1,c+j-1}$$

$$v_{s,t} = f_\sigma(z_{s,t})$$

$$v = \frac{1}{4}\sum_{s=1}^{2}\sum_{t=1}^{2} v_{s,t}$$

$$\hat{y} = wv$$

$$J(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

Backpropagation

$$J(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

$$J_k(\mathbf{w}) = (\hat{y}_k - y_k)^2$$

$$\nabla(J_k(\mathbf{w})) = \begin{bmatrix} \frac{\partial J_k(\mathbf{w})}{\partial w} \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{1,1}} \\ \dots \\ \dots \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{3,3}} \end{bmatrix}$$

Backpropagation

$$J(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

$$J_k(\mathbf{w}) = (\hat{y}_k - y_k)^2$$

$$\boxed{\frac{\partial J_k(\mathbf{w})}{\partial w}}$$

$$\nabla(J_k(\mathbf{w})) = \begin{bmatrix} \frac{\partial J_k(\mathbf{w})}{\partial w_{1,1}} \\ \cdots \\ \cdots \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{3,3}} \end{bmatrix}$$

$$\frac{\partial J_k(\mathbf{w})}{\partial w} = \frac{\partial J_k(\mathbf{w})}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial v}$$

Backpropagation

$$J(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

$$J_k(\mathbf{w}) = (\hat{y}_k - y_k)^2$$

$$\nabla(J_k(\mathbf{w})) = \begin{bmatrix} \frac{\partial J_k(\mathbf{w})}{\partial} \\ \boxed{\begin{matrix} \frac{\partial J_k(\mathbf{w})}{\partial w_{1,1}} \\ \cdots \\ \cdots \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{3,3}} \end{matrix}} \end{bmatrix}$$

$$\frac{\partial J_k(\mathbf{w})}{\partial w_{r,c}} = \frac{\partial J_k(\mathbf{w})}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial v}[\sum_{s=1}^{2}\sum_{t=1}^{2}\frac{\partial v}{\partial v_{s,t}}\frac{\partial v_{s,t}}{\partial f_\sigma(z_{s,t})}\frac{\partial f_\sigma(z_{s,t})}{\partial z_{s,t}}\frac{\partial z_{s,t}}{\partial w_{r,c}}]$$

Backpropagation

$$J(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

$$J_k(\mathbf{w}) = (\hat{y}_k - y_k)^2$$

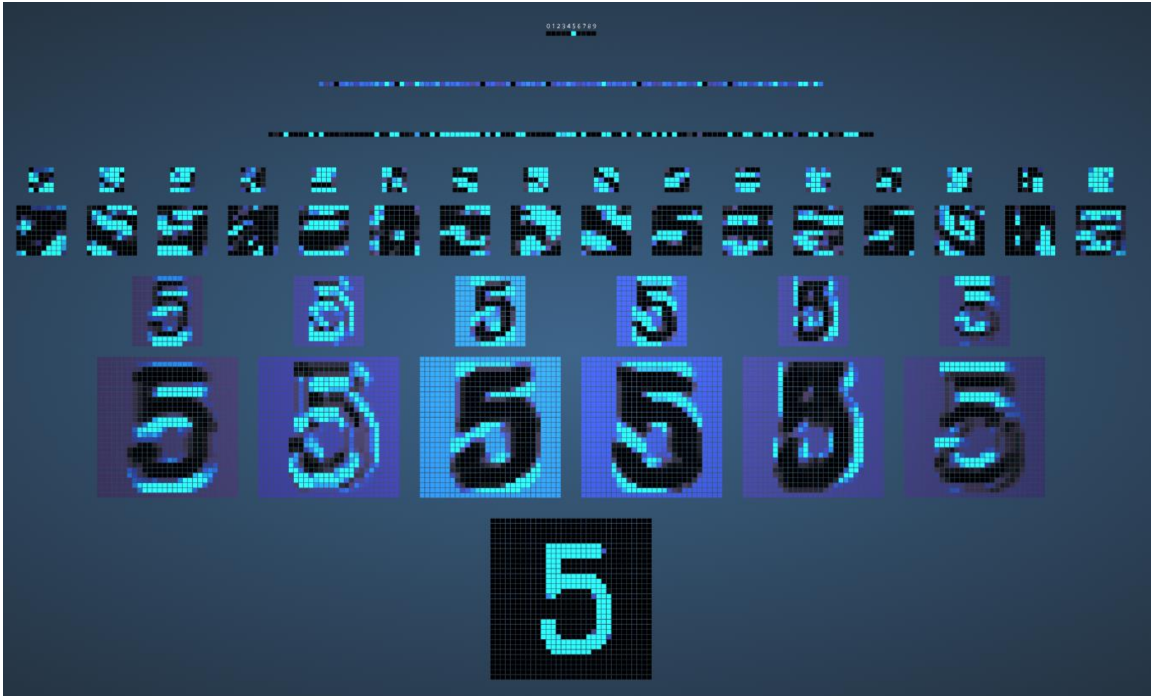
$$\nabla(J_k(\mathbf{w})) = \begin{bmatrix} \frac{\partial J_k(\mathbf{w})}{\partial} \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{1,1}} \\ \cdots \\ \cdots \\ \frac{\partial J_k(\mathbf{w})}{\partial w_{3,3}} \end{bmatrix}$$

$$\frac{\partial J_k(\mathbf{w})}{\partial w_{r,c}} = \frac{\partial J_k(\mathbf{w})}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial v}[\sum_{s=1}^{2}\sum_{t=1}^{2}\frac{\partial v}{\partial v_{s,t}}\frac{\partial v_{s,t}}{\partial f_\sigma(z_{s,t})}\frac{\partial f_\sigma(z_{s,t})}{\partial z_{s,t}}\frac{\partial z_{s,t}}{\partial w_{r,c}}]$$

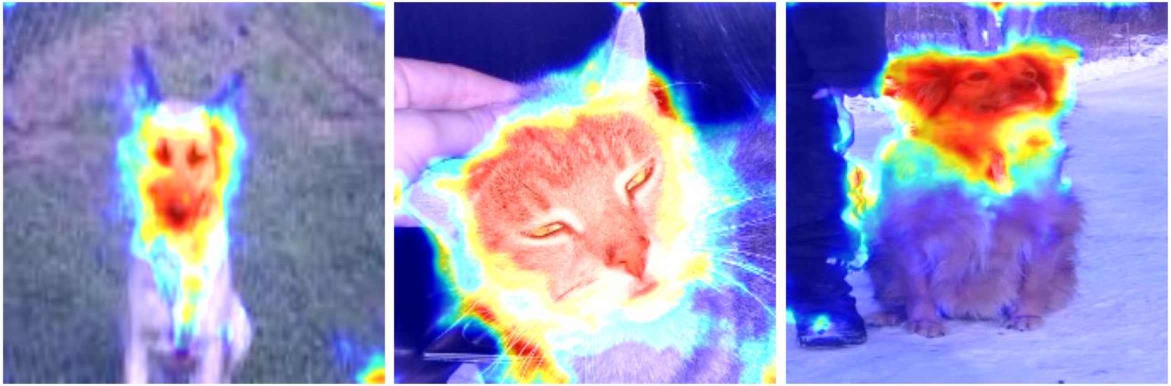
To further discuss...

A nice visualization

https://adamharley.com/nn_vis/cnn/3d.html



Intepretable models or interpretable data?



Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept

From https://towardsdatascience.com/understand-your-algorithm-with-grad-cam-d3b62fce353

Neuroscientific basis for convolutional networks

- Some of the design principles of Neural Networks have been drawn from neuroscience

- We now briefly discuss some of the connections between CNNs and a simplified version of the brain functions

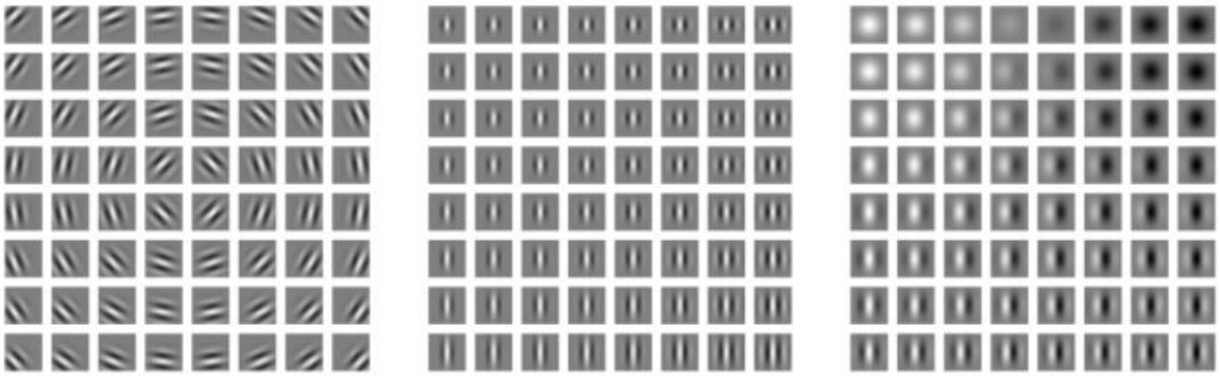
- We refer to the primary visual cortex (V1 area), the first one in the brain performing some significantly advanced processing of visual input

V1 area & CNNs

- V1 is arranged in a spatial map
- V1 contains simple cells , that an to some extent be characterized by a linear function (as for the detection step in CNNs)
- V1 also contains complex cells, that show some level of invariance to some changes in the visual input
- It is generally believed that the same basic principles apply to other areas in the visual stream, repeatedly

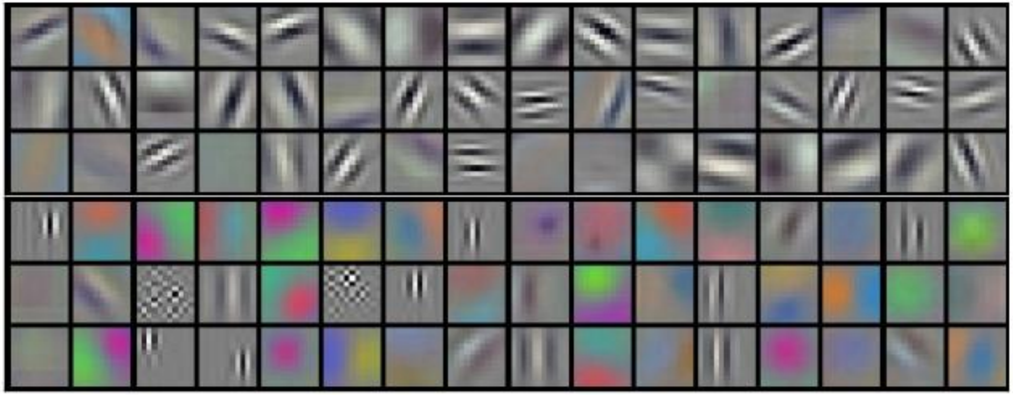
Again on V1 cells

- Experiments showed that most V1 cells have weights that can be described by Gabor functions



What about CNN weights?

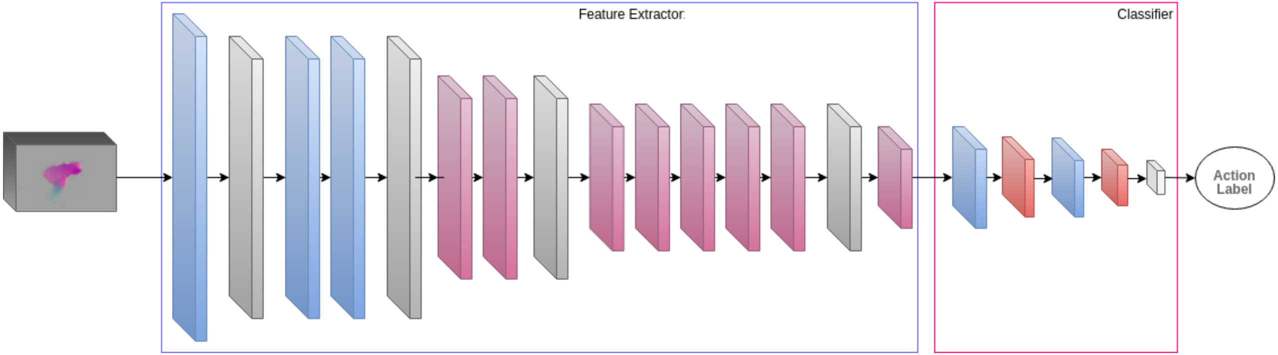
- At the very first layers the weights learnt by a CNN on natural images are very similar to Gabor filters



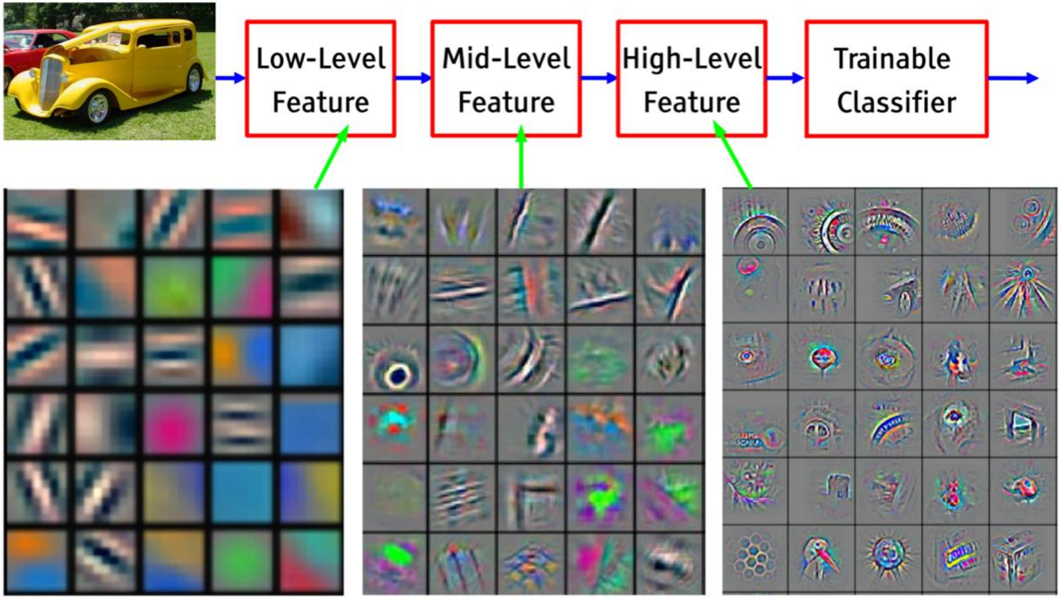
On the properties of weights learnt by convolutional layers

Generic features

Specific (to the data and the task) features

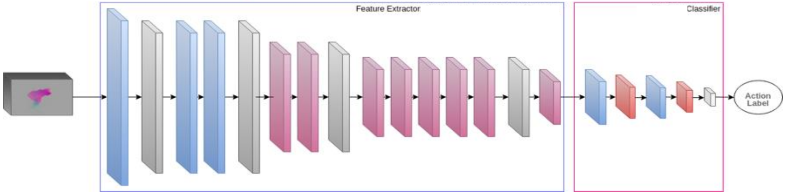


A hierarchical representation



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

Transfer learning



- It refers to the possibility of exploiting knowledge in terms of pre-trained models that can be used on different data and tasks (with some constraint)

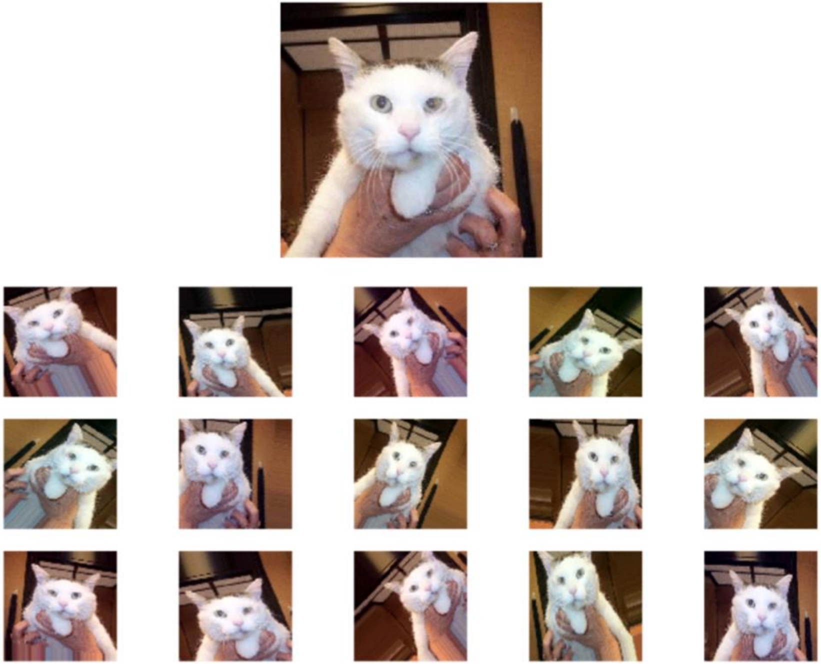
- Fine-tuning is a well-assessed procedure in which the weights are somehow adapted to the new problem/data starting from the pre-trained model

- This may imply a domain shift (also known as covariate shift) problem, due to the fact that the data distribution may change as you change the problem/data

CNN training

- Very data hungry and computationally intensive
- One of the trick for coping with data lack us data augmentation
  - The idea is to generate more data by applying some transformation to the image

Data augmentation



https://m2dsupsdlclass.github.io/lectures-labs/slides/04_conv_nets/index.html#82

CNN training

- Very data hungry and computationally intensive
- One of the trick for coping with data lack us data augmentation
  - The idea is to generate more data by applying some transformation to the image
- An alternative is to use synthetic data, but the model may be affected by domain shift issue (and thus it would need a specific domain adaptation strategy)

UniGe

MaLGa