UniGe | MaLGa

# Convolutional Neural Networks

**Nicoletta Noceti**
Nicoletta.noceti@unige.it

## Convolutional Neural Networks

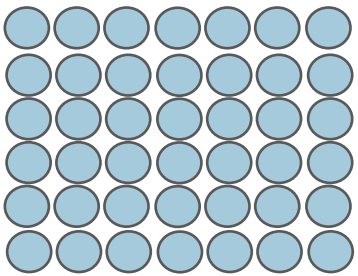
A specialized kind of neural network for processing data with a known grid-like topology

Examples:

- Time-series

1D grid

- Images



2D grid

## Motivations

CNNs leverage three important principles:

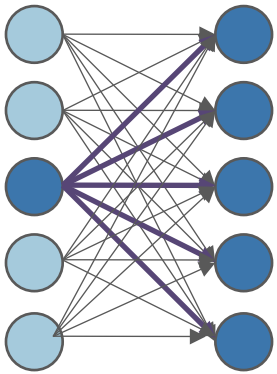
- **Sparse interaction**
- **Parameter sharing**
- **Equivariant representations** ← The layers show equivariance to translation. We will go back to this point later...

## Sparse interactions

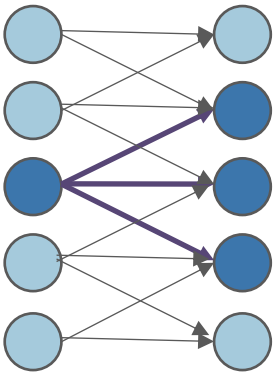
- In traditional DNNs every output unit interacts with every input unit
- In CNNs the “field of view” of each neuron is “limited” and the weights are re-used in different positions of the input
- The pros is that we have to learn fewer parameters, with improvements in
  - Memory requirements
  - Statistical efficiency: statistical strength for more samples per weight, reduced variance when estimating the parameters
  - Computations (less parameters/flops)

# Sparse interaction: intuitions

Focus on the input unit



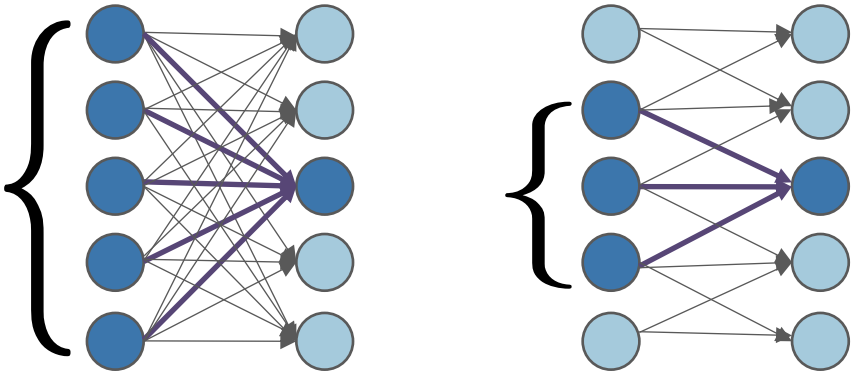
*DNN*



*CNN*

# Sparse interaction: intuitions

Focus on the output unit

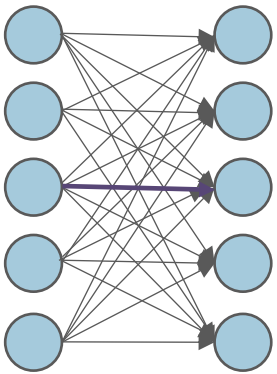


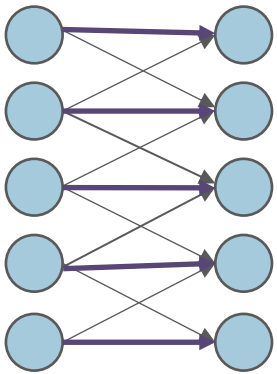
*DNN* *CNN*

*Receptive fields*

## Parameter sharing

- The same parameter is used by more than one function in the model
  - In traditional DNNs each weight is used exactly ones when computing the output
  - In CNNs each member of a kernel (a weight) is used at every position of the input





# A basic operation

## Convolution/Cross Correlation

$$I_{out}^{conv}(i,j) = (K * I)(i,j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} I(m,n)K(i-m, j-n) =$$

$$= \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} I(i-m, j-n)K(m,n)$$

$$I_{out}(i,j) = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} I(i+m, j+n)K(m,n)$$

# A basic operation

## Convolution/Cross Correlation

$$I_{out}(i,j) = \sum_{m=-W}^{+W} \sum_{n=-W}^{+W} I(i+m, j+n) K(m,n)$$



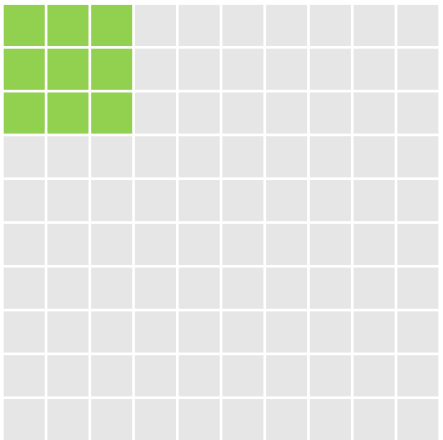
*An image of size 10x10*

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

# A basic operation

## Convolution/Cross Correlation

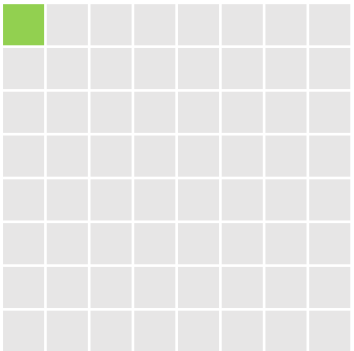
$$I_{out}(i,j) = \sum_{m=-W}^{+W} \sum_{n=-W}^{+W} I(i+m, j+n) K(m,n)$$



*INPUT: an image of size 10x10*

*A kernel (filter) of size 3x3*

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

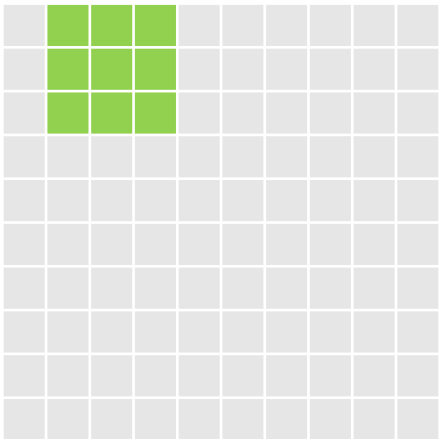


*OUTPUT: a structure of size 8x8*

# A basic operation

## Convolution/Cross Correlation

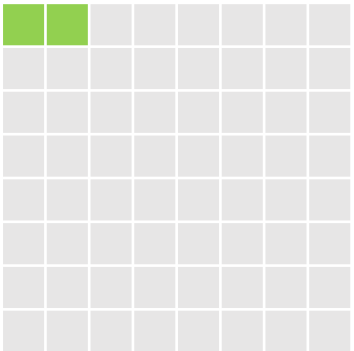
$$I_{out}(i,j) = \sum_{m=-W}^{+W} \sum_{n=-W}^{+W} I(i+m, j+n) K(m,n)$$



*INPUT: an image of size 10x10*

*A kernel (filter) of size 3x3*

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

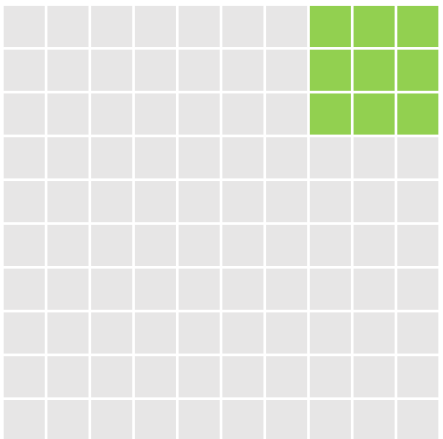


*OUTPUT: a structure of size 8x8*

# A basic operation

## Convolution/Cross Correlation

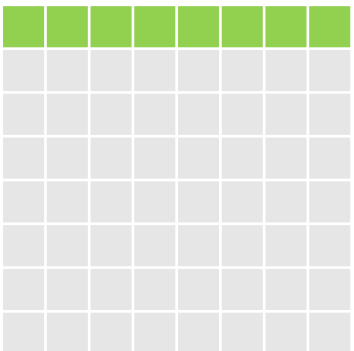
$$I_{out}(i,j) = \sum_{m=-W}^{+W} \sum_{n=-W}^{+W} I(i+m, j+n) K(m,n)$$



*INPUT: an image of size 10x10*

*A kernel (filter) of size 3x3*

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

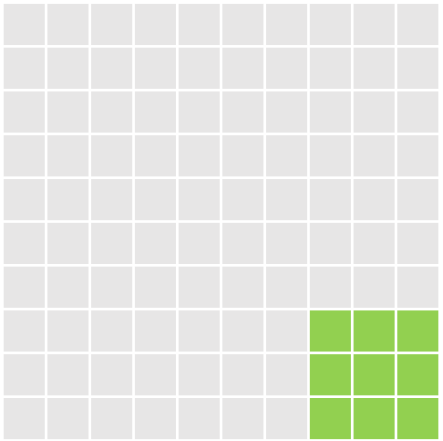


*OUTPUT: a structure of size 8x8*

# A basic operation

## Convolution/Cross Correlation

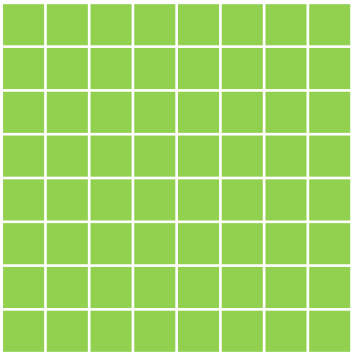
$$I_{out}(i,j) = \sum_{m=-W}^{+W} \sum_{n=-W}^{+W} I(i+m, j+n) K(m,n)$$



*INPUT: an image of size 10x10*

*A kernel (filter) of size 3x3*

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



*OUTPUT: a structure of size 8x8*

# A couple of observations

**Dense Neural Networks don't scale to images. Example: Let us consider a fully connected network with a single unit**

- With tiny color images of size 32 x 32 x 3 in input:
  - Size of the input layer: 32 x 32 x 3 = 3072
  - Size of the weights: 3072
- With small color images of size 200 x 200 x 3 in input:
  - Size of input layer and weights: 200 x 200 x 3 = 120000

**Dense Neural Networks don't fully exploit/capture the specificities of images**

UniGe | MaLGa

# CNNs

# A change of paradigm

From Feature engineering...

$$\Phi(\Psi(x))$$

$x$

*Data Representation* (***feature extraction***)

$\Psi(x)$

Classifier

$y$

# A change of paradigm

From Feature engineering...

$$\Phi(\Psi(x))$$

$x$

*Data Representation* ***(feature extraction)***

$\Psi(x)$

Classifier

$y$

... to Feature learning

*This part learns the representation*

*This part trains the classifier*

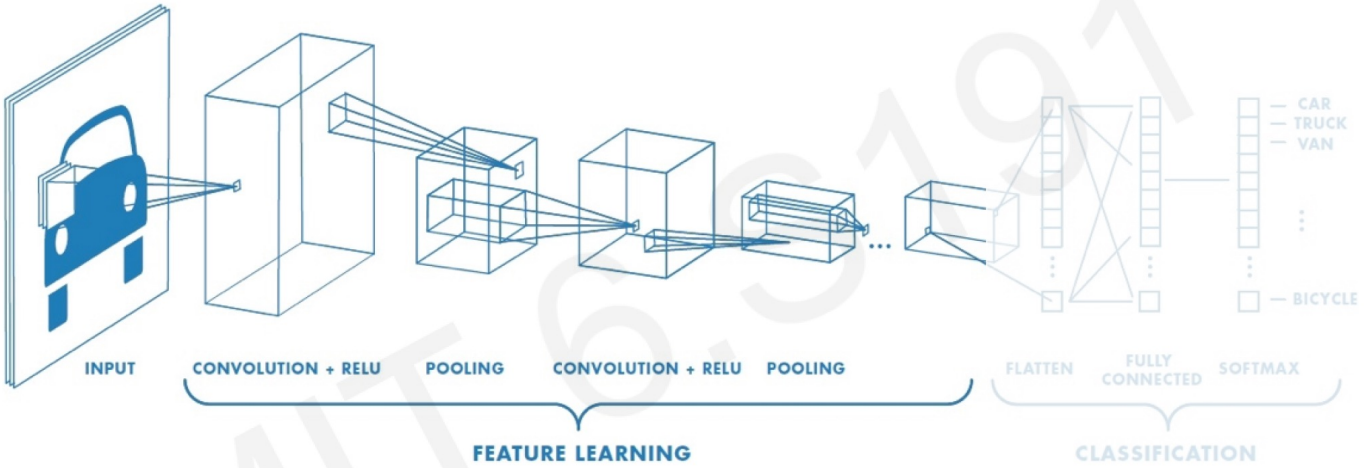
$x$

(Deep) NN

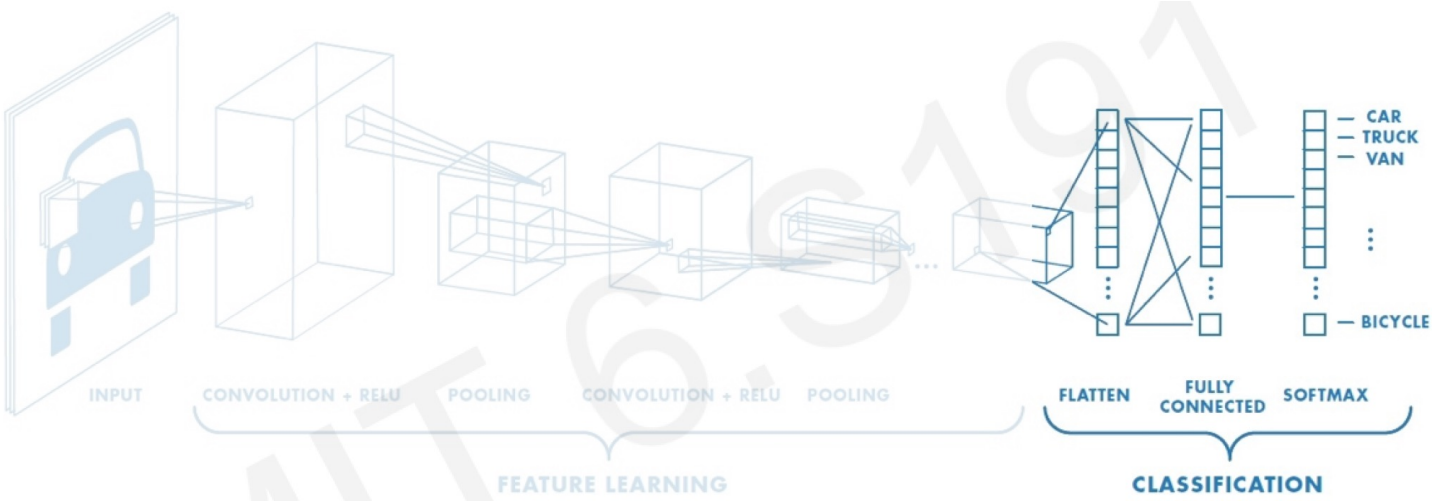
$y$

$f(x)$

# A typical CNN



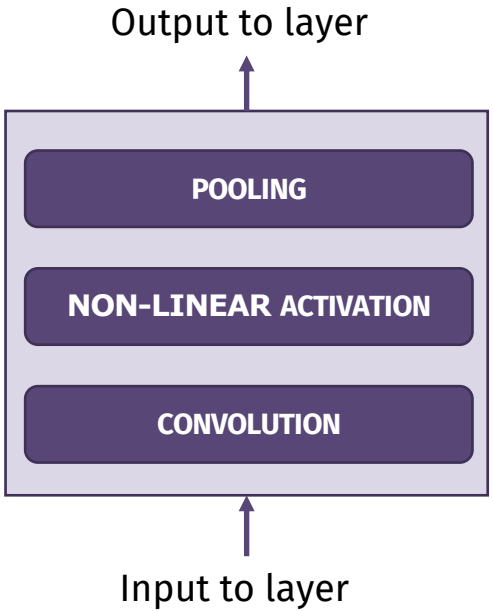
# A typical CNN



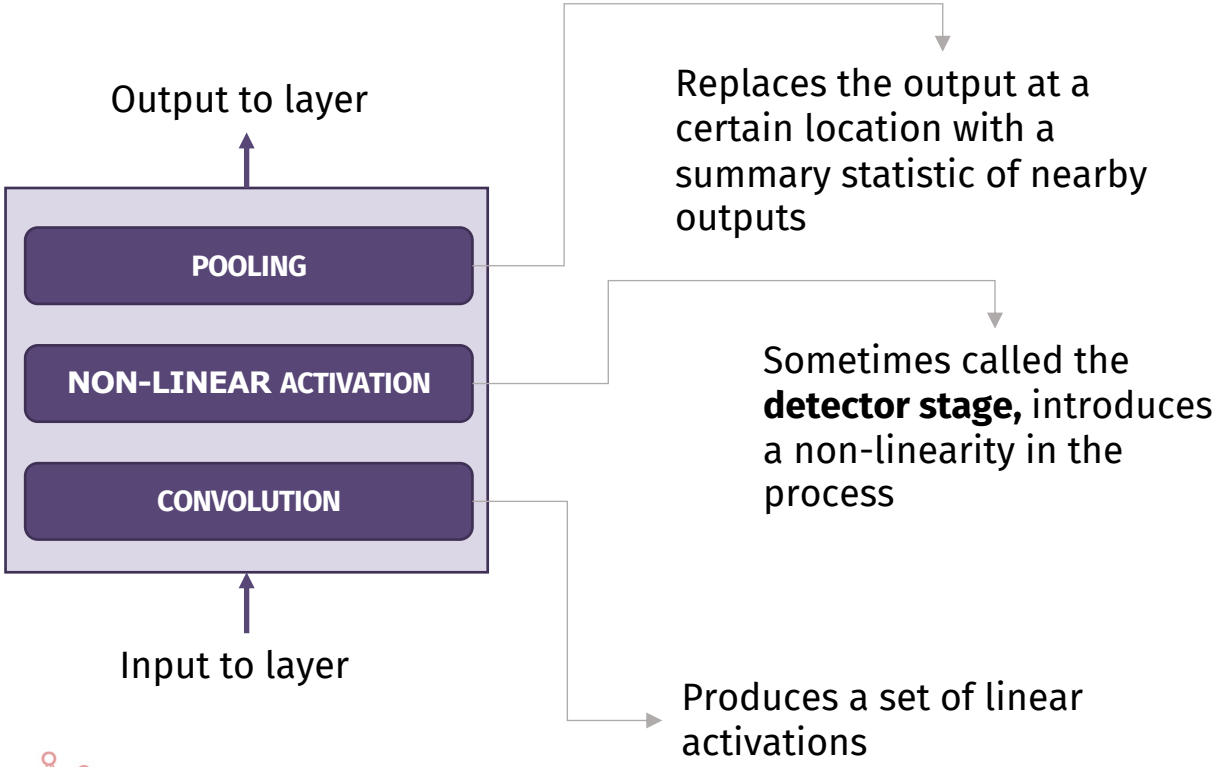

*Interlude:* $softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{NC} e^{y_j}}$

*[NC is the number of classes]*

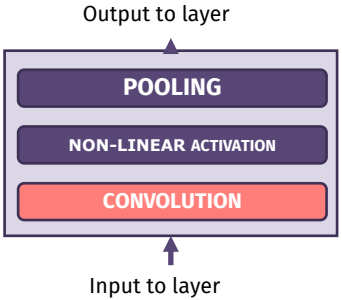
# A typical CNN layer



# A typical CNN layer



# The conv layer

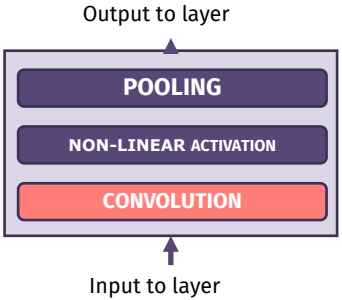



## A closer look to convolution and parameter sharing

As the kernel slides on the image**, it is able to capture the same property in different image regions**

Multiple kernels/filters are used → Multiple feature detectors can be used to capture different image properties

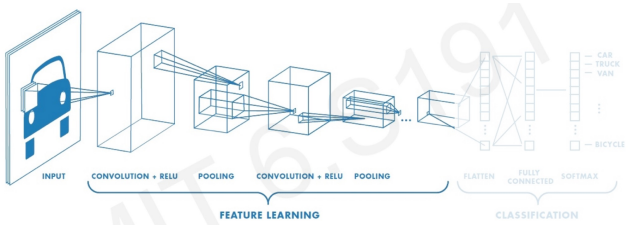
# The conv layer



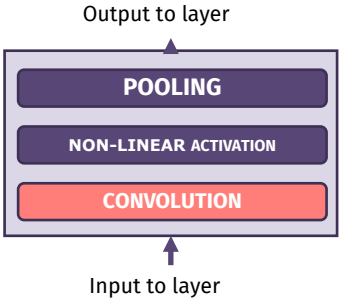

## A closer look to convolution and parameter sharing

As the kernel slides on the image**, it is able to capture the same property in different image regions**

Multiple kernels/filters are used → Multiple feature detectors can be used to capture different image properties



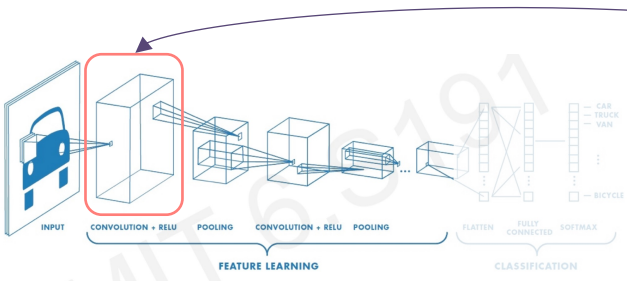
# The conv layer




## A closer look to convolution and parameter sharing

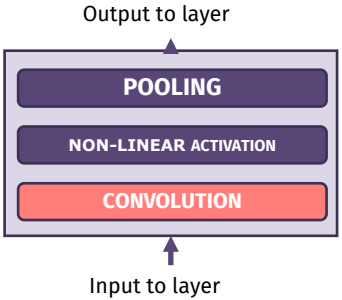
As the kernel slides on the image**, it is able to capture the same property in different image regions**

Multiple kernels/filters are used → Multiple feature detectors can be used to capture different image properties




$$O(:,:,j) = \mathbf{x} * F(:,:,j)$$

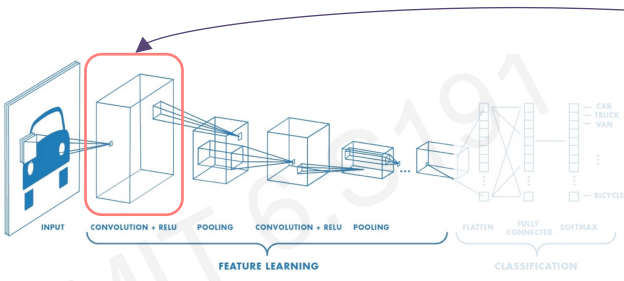
# The conv layer




## A closer look to convolution and parameter sharing

As the kernel slides on the image**, it is able to capture the same property in different image regions**

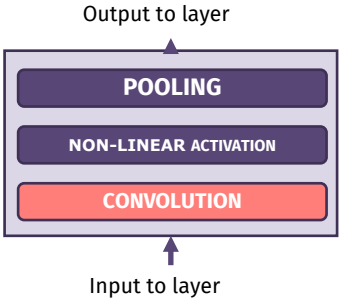
Multiple kernels/filters are used → Multiple feature detectors can be used to capture different image properties




$$O(:,:,j) = \mathbf{x} * F(:,:,j)$$

*Input image*

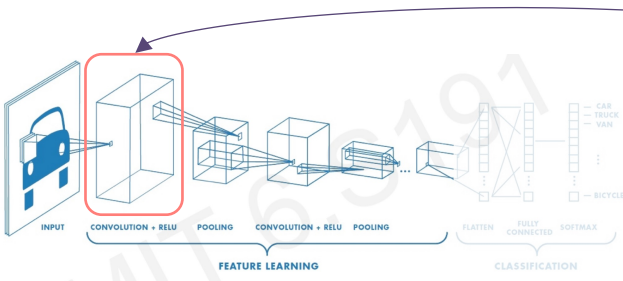
# The conv layer




## A closer look to convolution and parameter sharing

As the kernel slides on the image**, it is able to capture the same property in different image regions**

Multiple kernels/filters are used → Multiple feature detectors can be used to capture different image properties

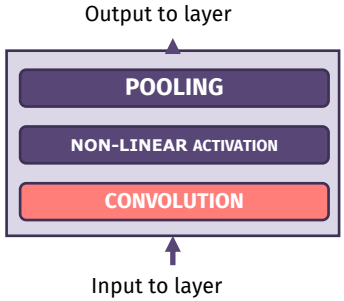


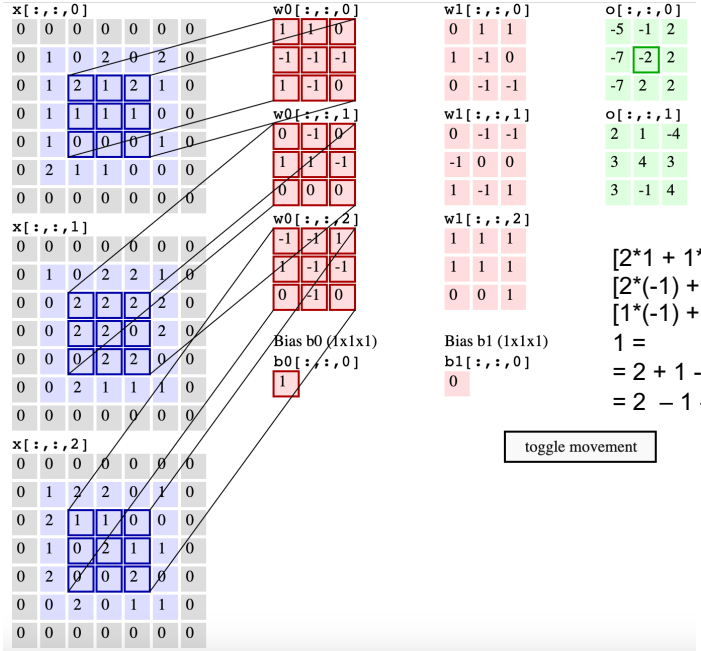

$$O(:,:,j) = \mathbf{x} * F(:,:,j)$$

*Input image (in the first conv layer)*

*A bank of C filters*

# An example from the animation

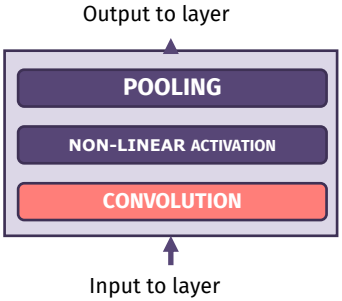


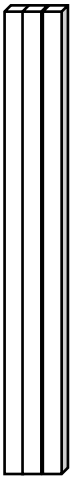

[2*1 + 1*1 + 1*(-1) + 1*(-1) + 1*(-1)]+
[2*(-1) + 2*1 + 2*1]+
[1*(-1) + 1*(-1) + 2*(-1) + 1*(-1)]+
1 =
= 2 + 1 - 1 – 1 - 1 - 2 + 2 + 2 – 1 – 1 – 2 – 1+1
= 2 – 1 - 1 – 1 – 1 = **-2**

See the demo here: https://cs231n.github.io/convolutional-networks

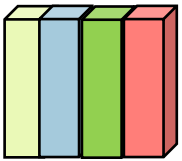
# Another animation to clearly understand




*W x H x 3*

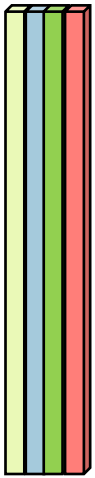


*K x K x 3 x 4*
*(C==4)*

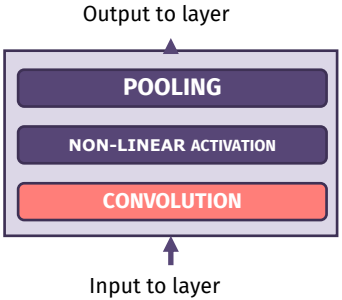


*C is the number of channels, a hyper-parameter of the conv layer*

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



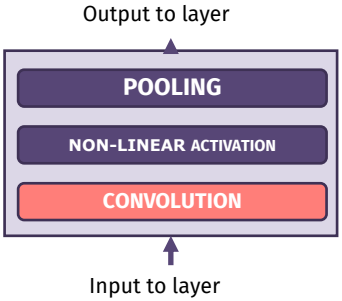
# Output feature size of conv layers




Three parameters control the size of the output of a layer

- **Depth**, 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 kernel application in each one of the (original) point

# Output features size of conv layers




Size BEFORE convolution

Kernel size

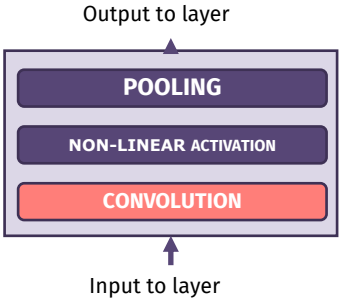
Padding

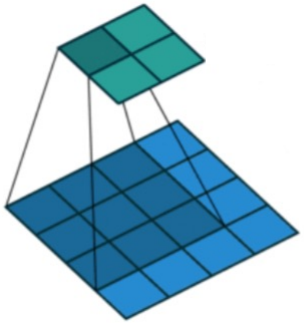
$$O = \frac{W - K + 2P}{S} + 1$$

Size AFTER convolution

Stride

# Output features size of conv layers
## Examples

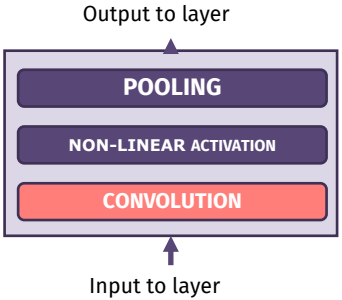


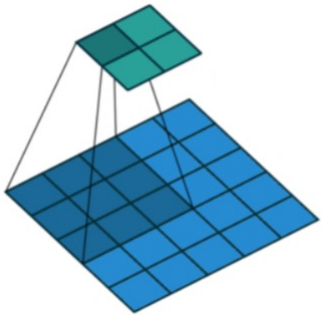



No padding, stride 1

$$O_2 = \frac{W - K + 2P}{S_1} + 1$$

# Output features size of conv layers
## Examples

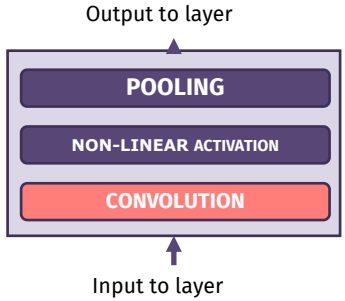


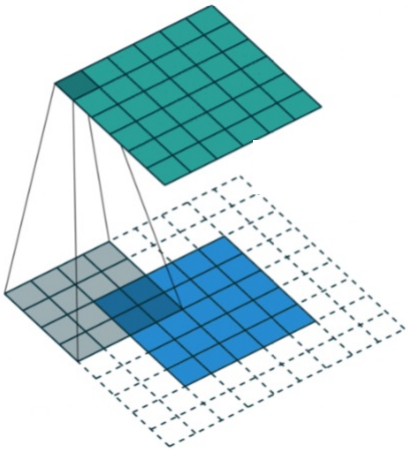



$$O_2 = \frac{W^5 - K^3 + 2P^0}{S_2} + 1$$

No padding, stride 2

# Output features size of conv layers
## Examples

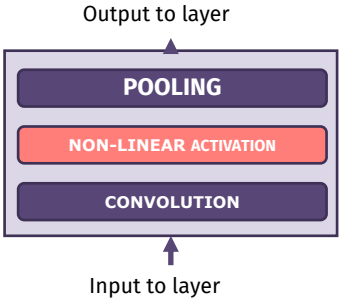


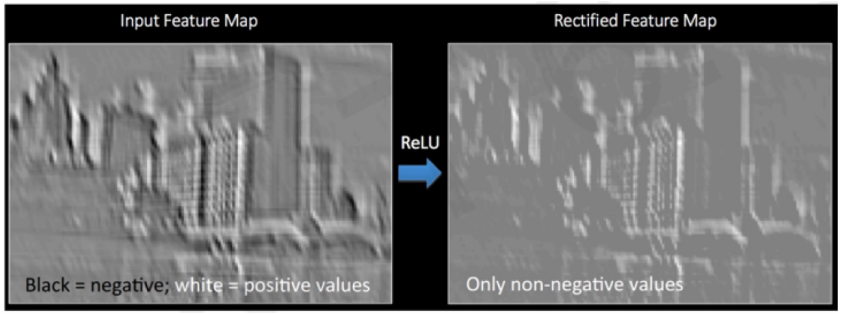
Padding 2, stride 1

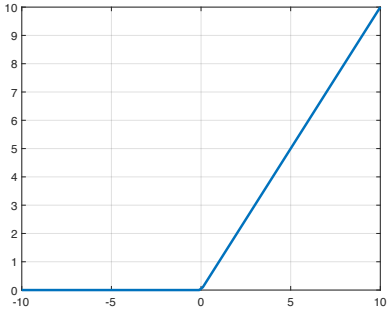
$$O = \frac{W - K + 2P}{S} + 1$$

# Introducing non-linearities



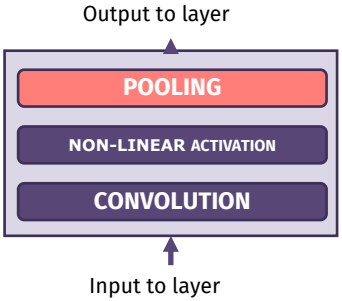

A typical choice is ReLU: it is applied after each convolutional layer and only preserves non-negative values



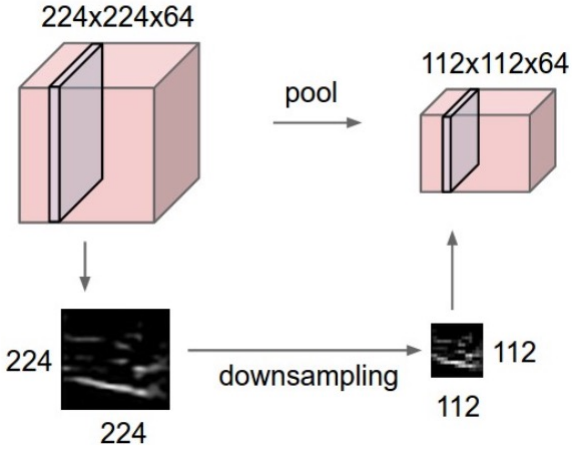




$$f(x) = max(0, x)$$

# Pooling and invariance

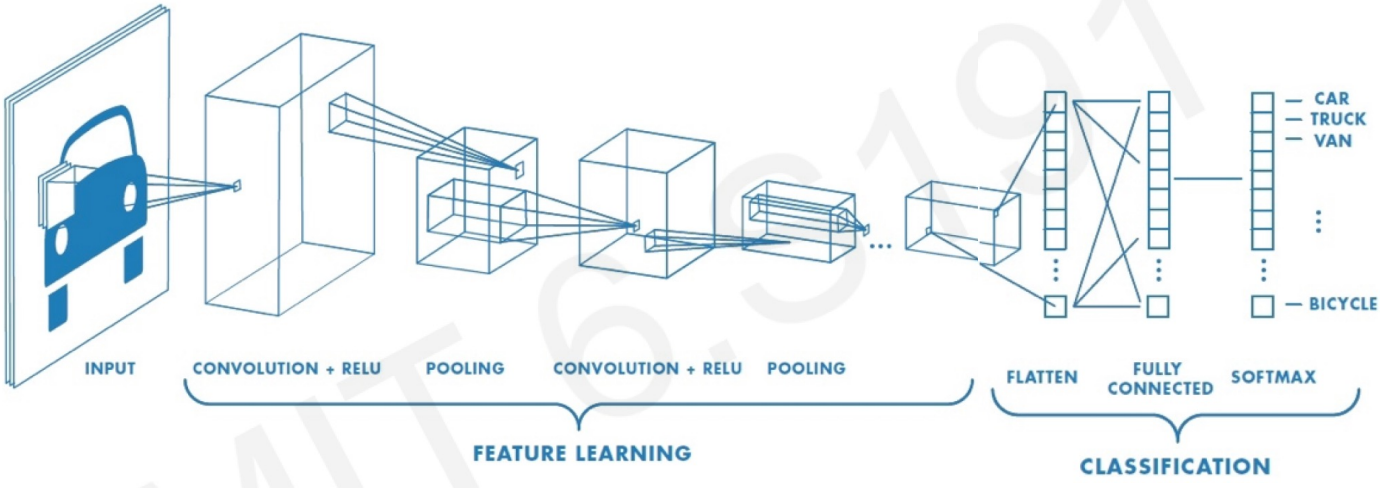



- **It provides invariance to small shifts of the inputs**
- Pooling functions:
  - **Average pooling**: average activation of the convolutional layer
  - **Max pooling**: max activation of the convolutional layer



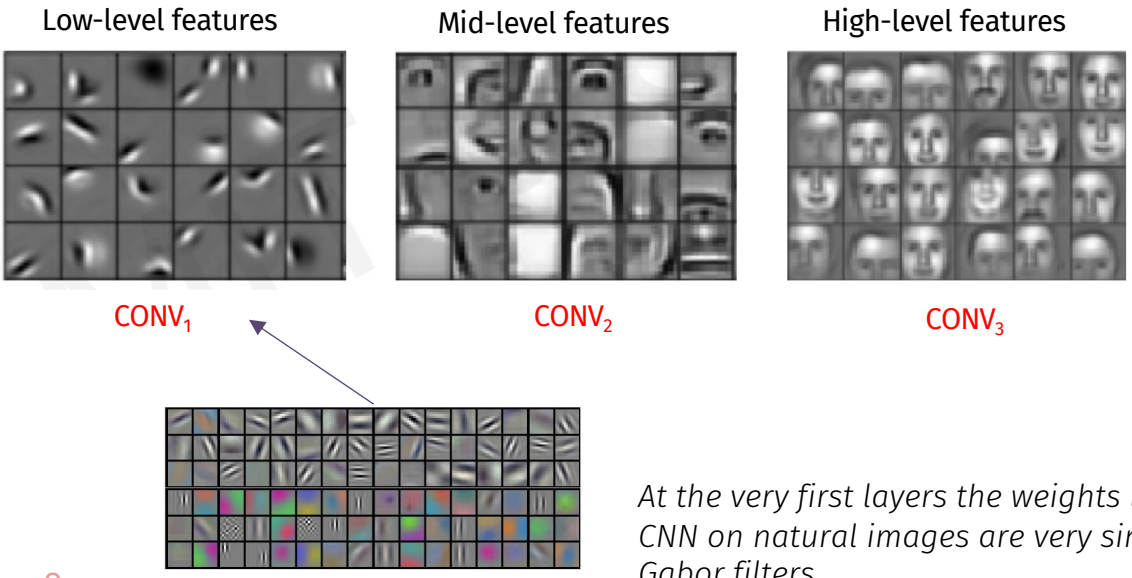

***No parameter to be estimated here!***

# A sequence of Conv layers



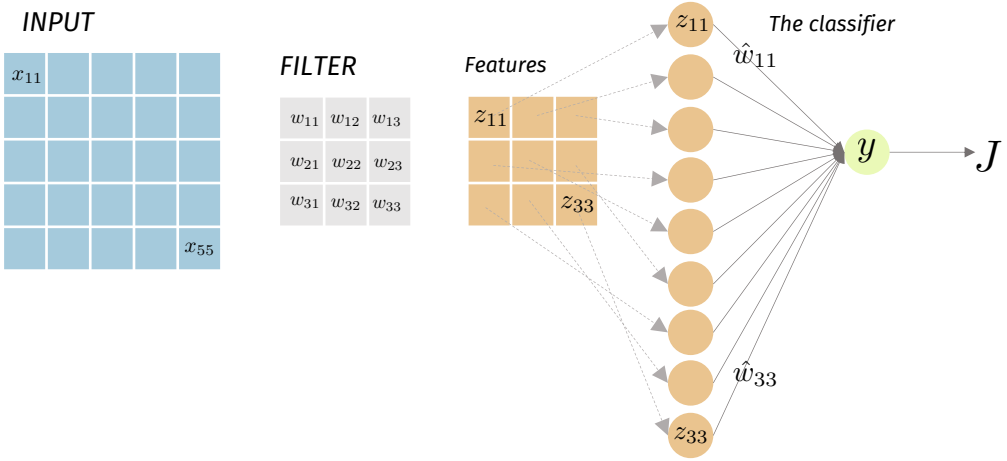
# Towards image classification

So... Is it possible to learn the most appropriate representation, possibly with a hierarchy, directly from the data?

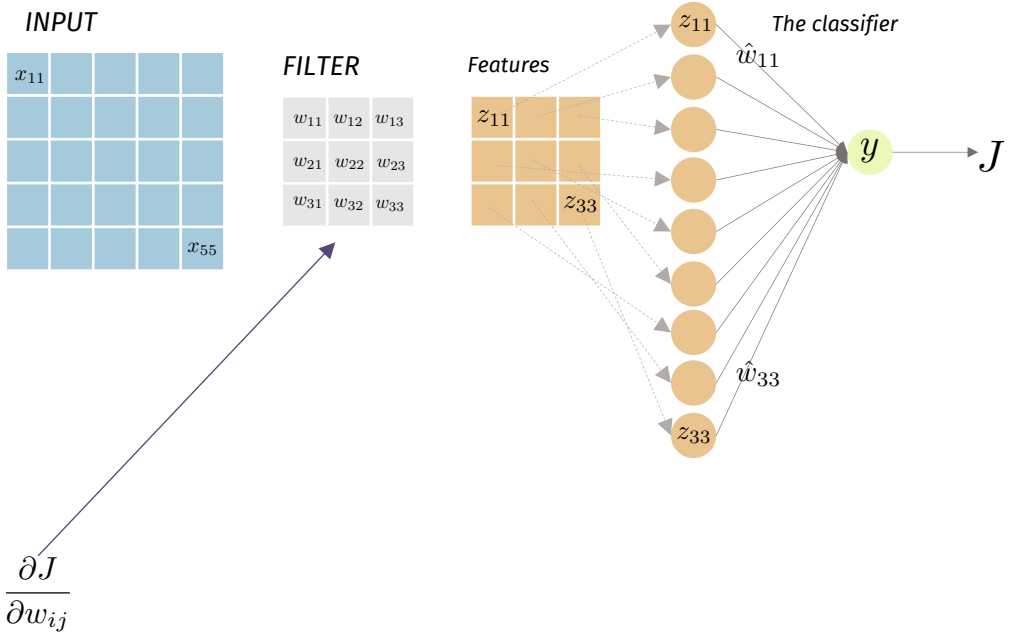



*At the very first layers the weights learnt by a CNN on natural images are very similar to Gabor filters*

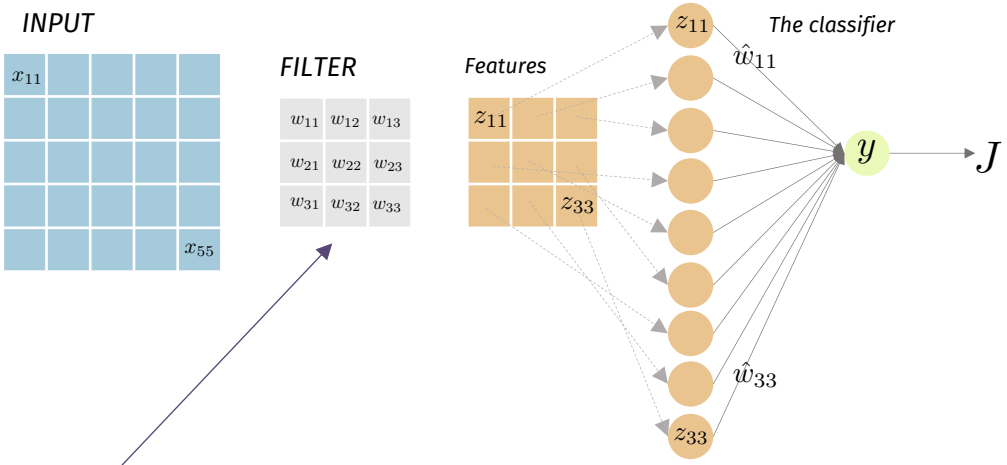
# Backgropragation in CNNs (intuition)



# Backgropragation in CNNs (intuition)

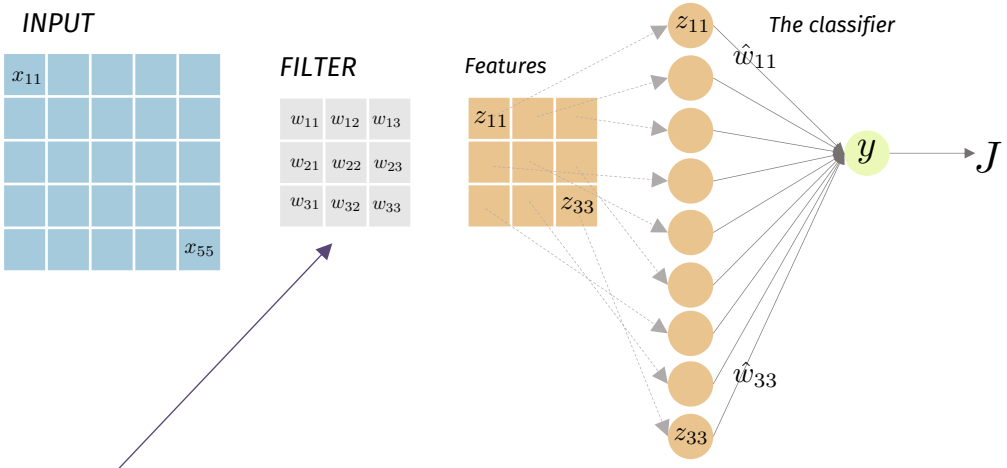


# Backgropragation in CNNs (intuition)




$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})} \frac{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})}{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}}$$

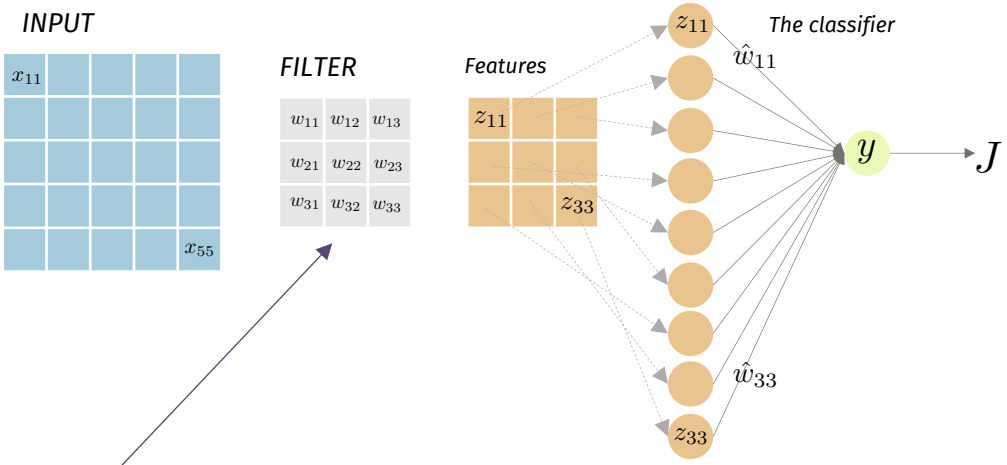
# Backgropragation in CNNs (intuition)




$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})} \frac{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})}{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}} \sum_{k=1}^{3} \sum_{h=1}^{3} \frac{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}}{\partial z_{kh}}$$

$$z_{kh} = \sum_{m=-1}^{+1} \sum_{n=-1}^{+1} I(k+m, h+n) K(m,n)$$

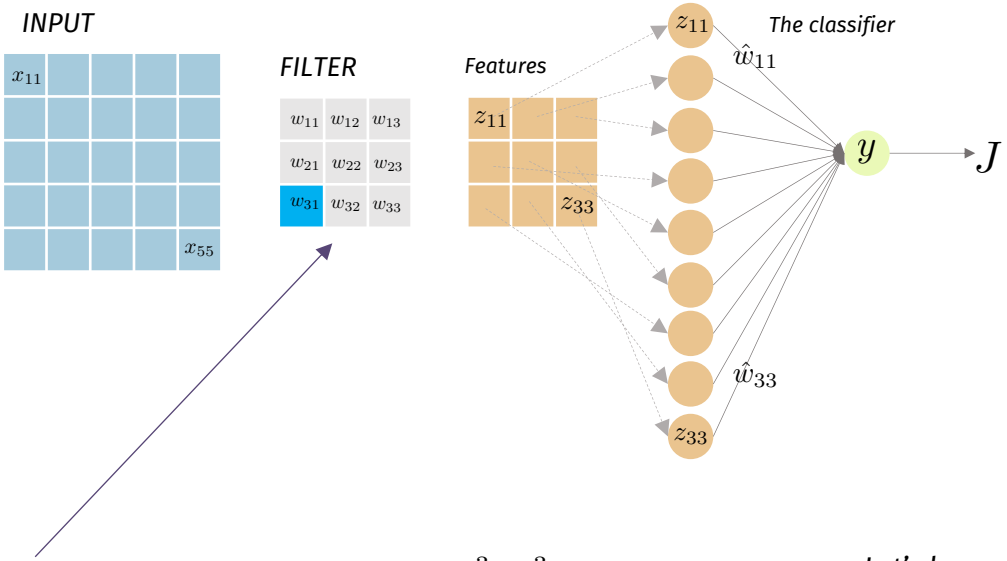
# Backgropragation in CNNs (intuition)




$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})} \frac{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})}{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}} \sum_{k=1}^{3} \sum_{h=1}^{3} \frac{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}}{\partial z_{kh}} \frac{\partial z_{kh}}{\partial w_{ij}}$$

$$z_{kh} = \sum_{m=-1}^{+1} \sum_{n=-1}^{+1} I(k+m, h+n) K(m,n)$$

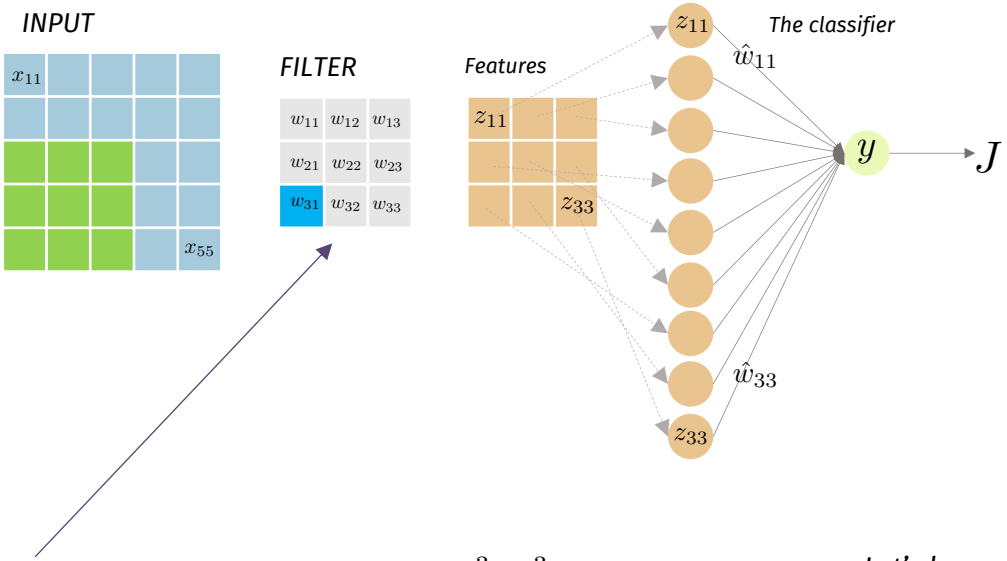
# Backgropragation in CNNs (intuition)




$$\frac{\partial J}{\partial w_{31}} = \frac{\partial J}{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})} \frac{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})}{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}} \sum_{k=1}^{3} \sum_{h=1}^{3} \frac{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}}{\partial z_{kh}} \frac{\partial z_{kh}}{\partial w_{31}}$$

*Let's be more specific*

# Backgropragation in CNNs (intuition)




$$\frac{\partial J}{\partial w_{31}} = \frac{\partial J}{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})} \frac{\partial f_\sigma(\hat{\mathbf{w}}^\intercal \mathbf{z})}{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}} \sum_{k=1}^{3} \sum_{h=1}^{3} \frac{\partial \hat{\mathbf{w}}^\intercal \mathbf{z}}{\partial z_{kh}} \frac{\partial z_{kh}}{\partial w_{31}}$$

*Let's be more specific*

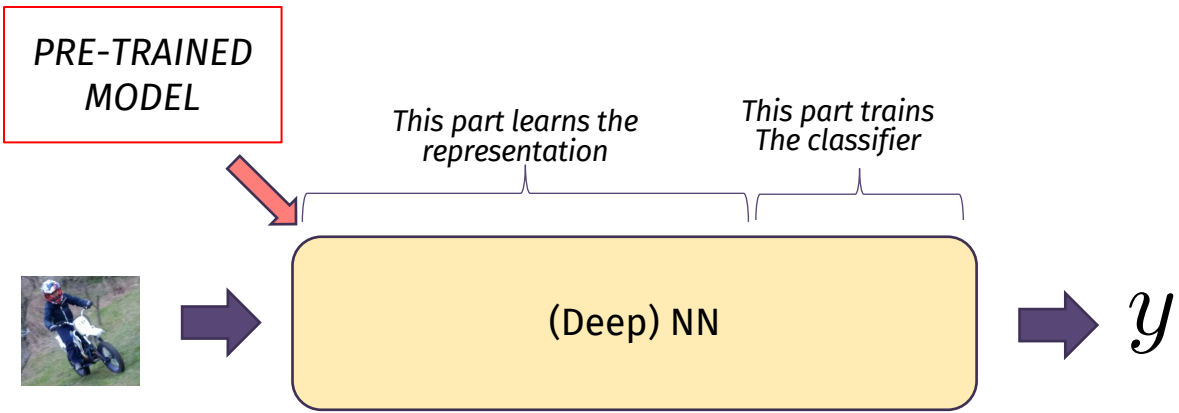
*... this is still a convolution/correlation*

UniGe | MaLGa

# Some further discussions

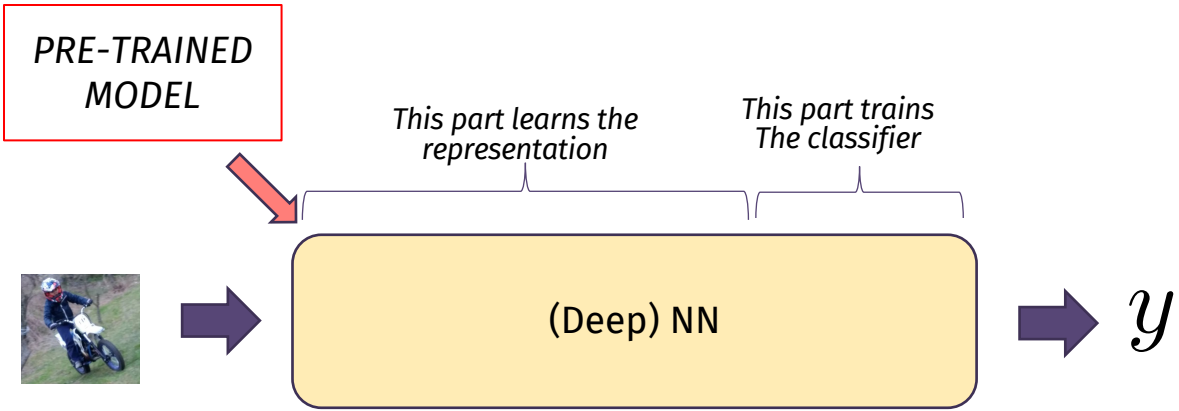
# About knowledge reuse

- Efficiency/sustainability are becoming hot keywords...
- Also, in some cases we might not have enough data
- It may be convenient to exploit already existing knowledge!



# Using pre-trained models

- Instead of training the network from scratch, one can start from weights learnt on other (possibly very big) datasets, and then fine-tune on the target task



# Again on knowledge reuse: the role of data supervision

## From unsupervised to self-supervised learning

Sometimes data are available but only few of them have labels

# Again on knowledge reuse: the role of data supervision

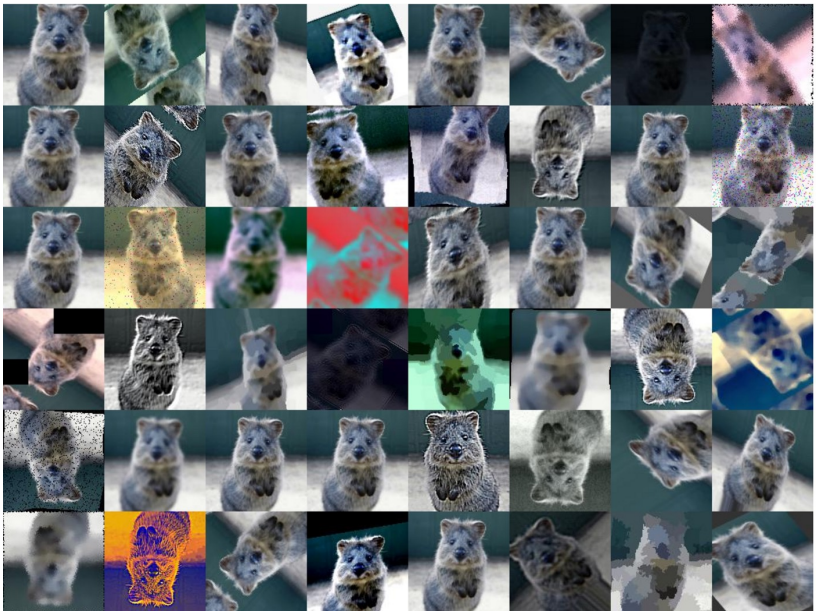
## From unsupervised to self-supervised learning


*Use the unlabeled data to train a self-supervised problem → The supervision is derived by the data*

*Fine-tune using the labeled data on the target supevised task*

# Data Augmentation

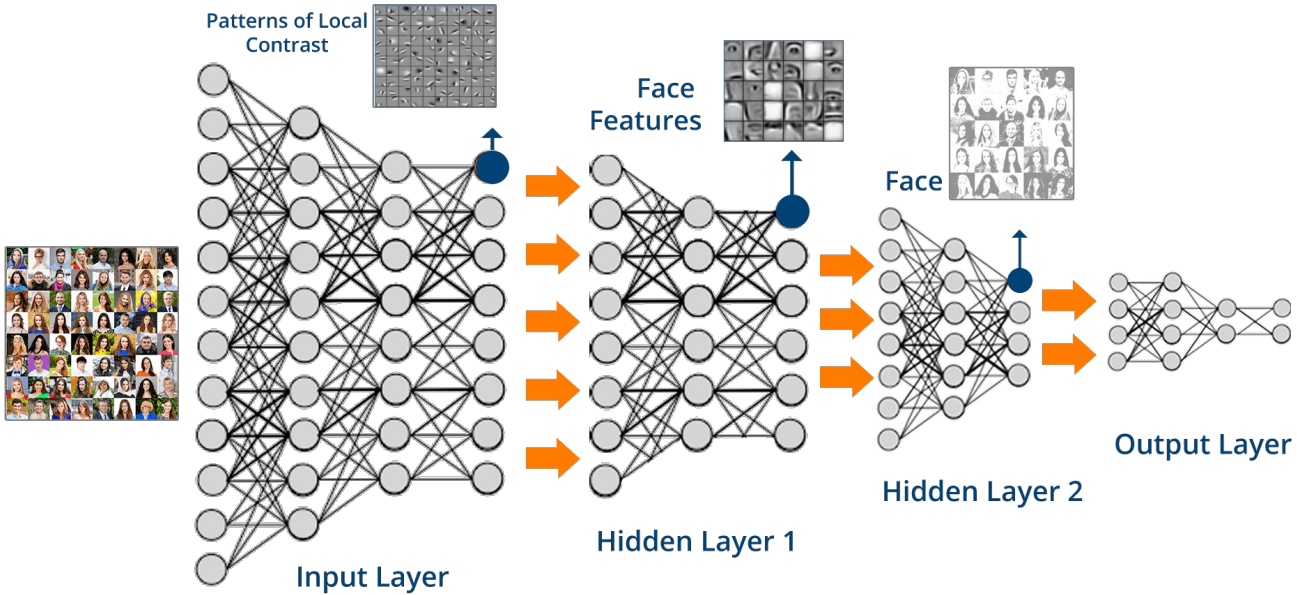
## An alternative to enrich the dataset



*Is it always possible?*

*From https://towardsdatascience.com/image-augmentation-for-deep-learning-using-keras-and-histogram-equalization-9329f6ae5085*

# How to interpret a deep architecture?



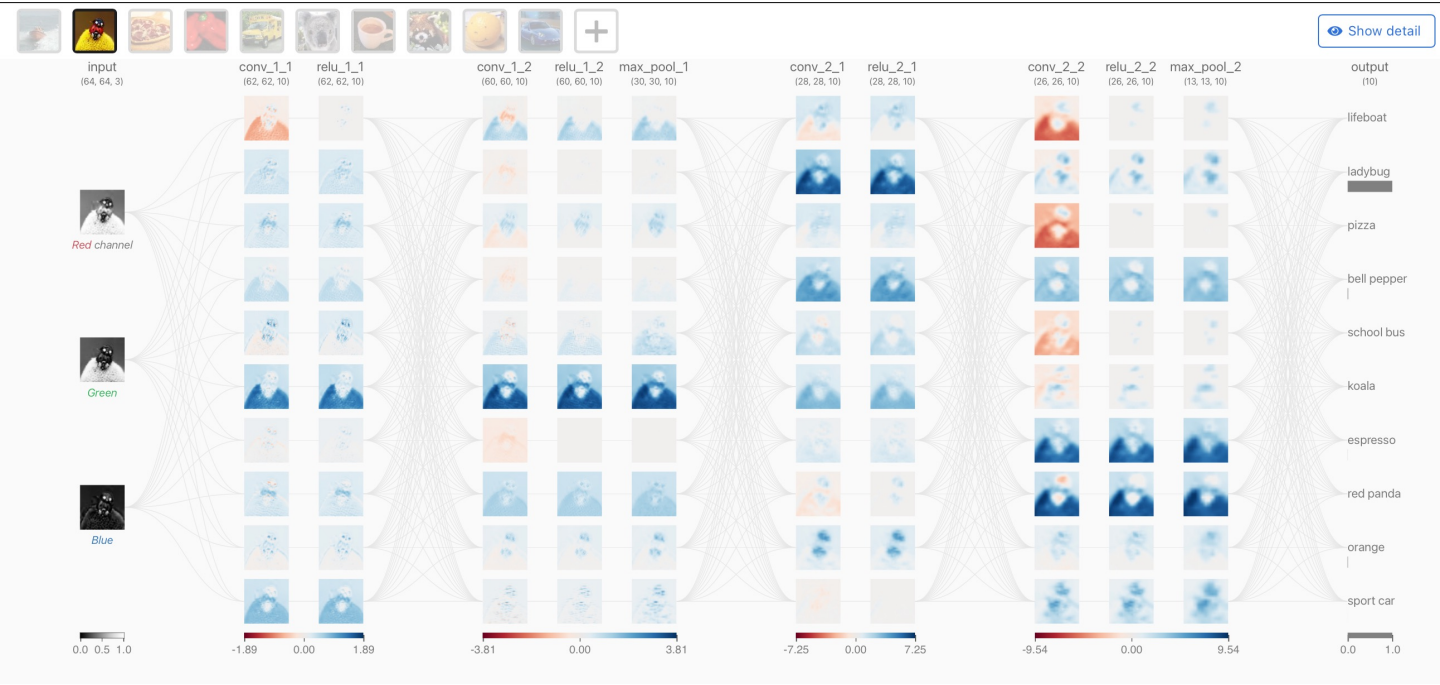
# Intepretable models or interpretable data?

*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*

# A nice visualization

https://poloclub.github.io/cnn-explainer/




UniGe | MaLGa

UniGe