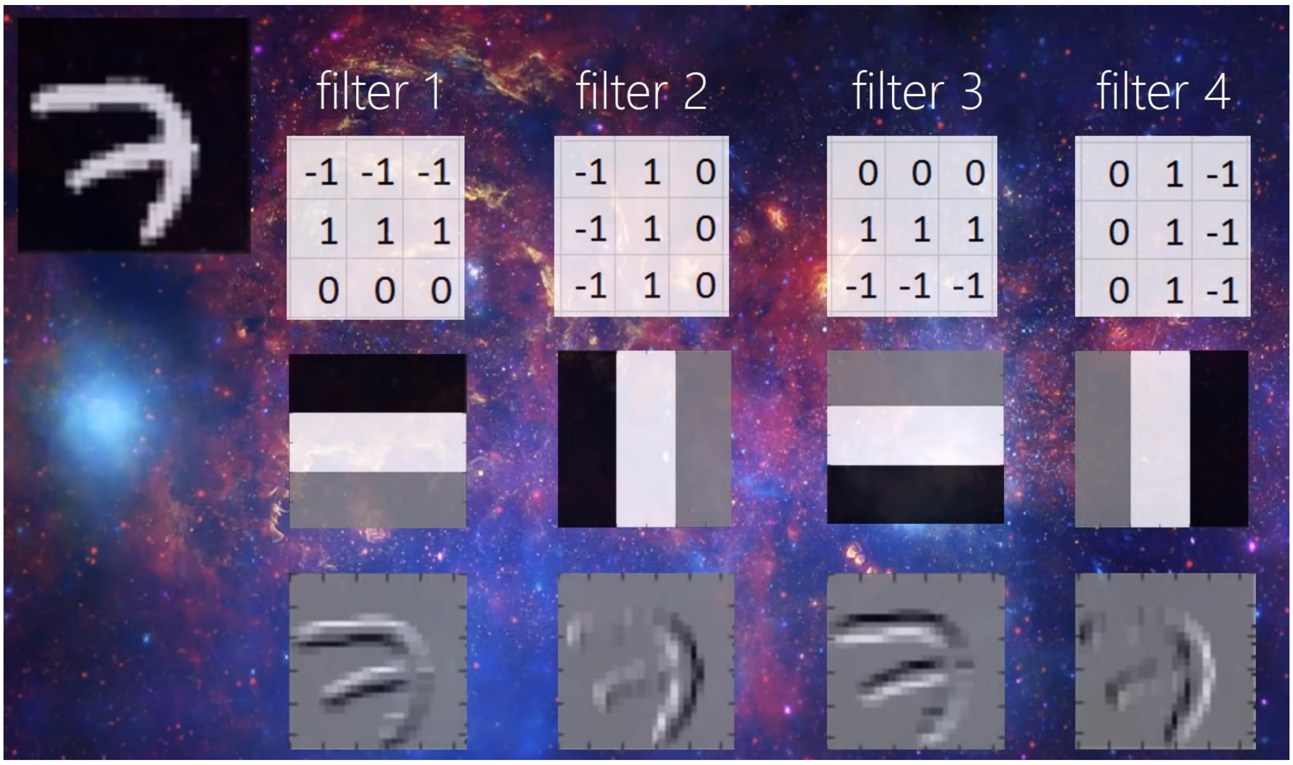
Video 1 - https://www.youtube.com/watch?v=QzY57FaENXg
CNN – deep learning technique focused on pattern recognition. It’s a part of neural network. There are filters that performs pattern recognition. Applied to an image, so a set of pixels, filters can understand what is an object. A filter is a block of pixels, for example a 3x3 block. We look at the image and we get a numeric score that represent how that part of the image is similar to the shape of it, until we map every 3x3 blocks. We get an array of numbers that tells us how much is present that shape. We can do that for a lot of filters. If we combine all the filters togheter, in a process called “pulling”, we understand what is inside the filter. This is the first layer, but we can do more layers. We can goo deeper.

Video 2 - https://www.youtube.com/watch?v=YRhxdVk_sIs
CNN is a neural network specialized in understanding patterns in an image. The basis is the convolutional layers. They apply a transformation and send it in another layer. They are basically filters that detects patterns. A pattern can be “edges”, “corners”, “squares”… At first they are simple, going deeper they are able to detect more sophisticated objects.
In MNIST dataset for example, the first conv_layer can have a filter of size 3x3. It slides into all 3x3 blocks in the image. This process is called convolving. We have a dot product between matrix filter and matrix of pixels of the image. The result is stored into a cell of another matrix. After convolving into all the image, we get another representation of the image. Then this is convolved by another filter. Example:



They detect different edges.

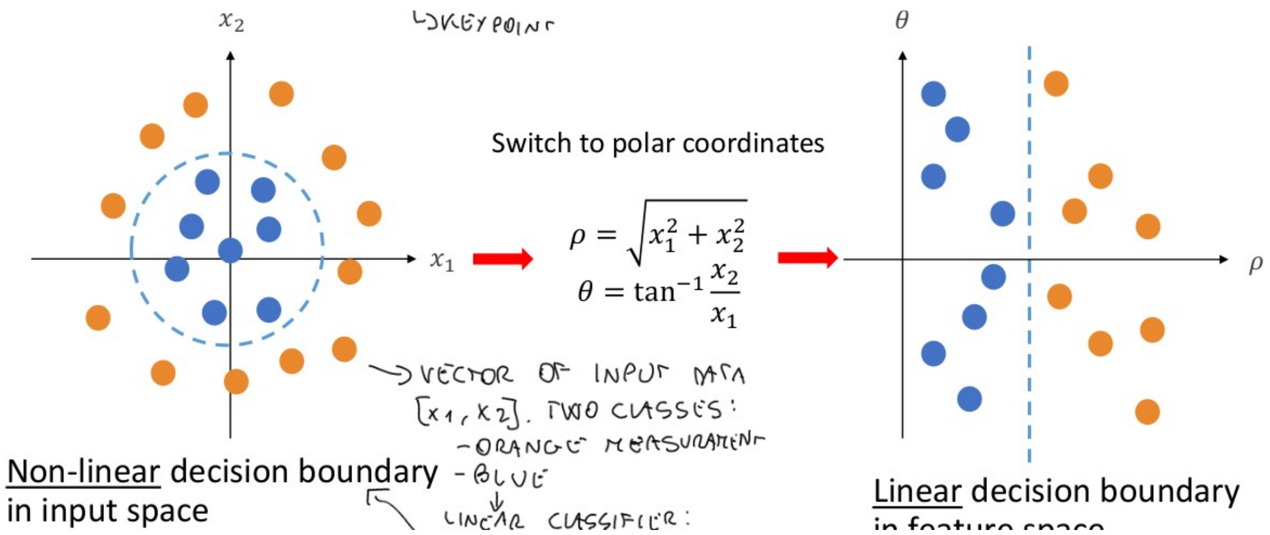
Slides Di Stefano
The goal is to find a function that taking as input an image, outputs a value that correspond to the label of an object. The parameters of this function are found by minimizing a loss function based on training data.

$$\boldsymbol{\theta}^* = \mathbf{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}\, \boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{D}^{\boldsymbol{train}}) \qquad \boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{D}^{\boldsymbol{train}}) = \frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i}} \underline{\boldsymbol{L}(\boldsymbol{\theta}, (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}))}$$

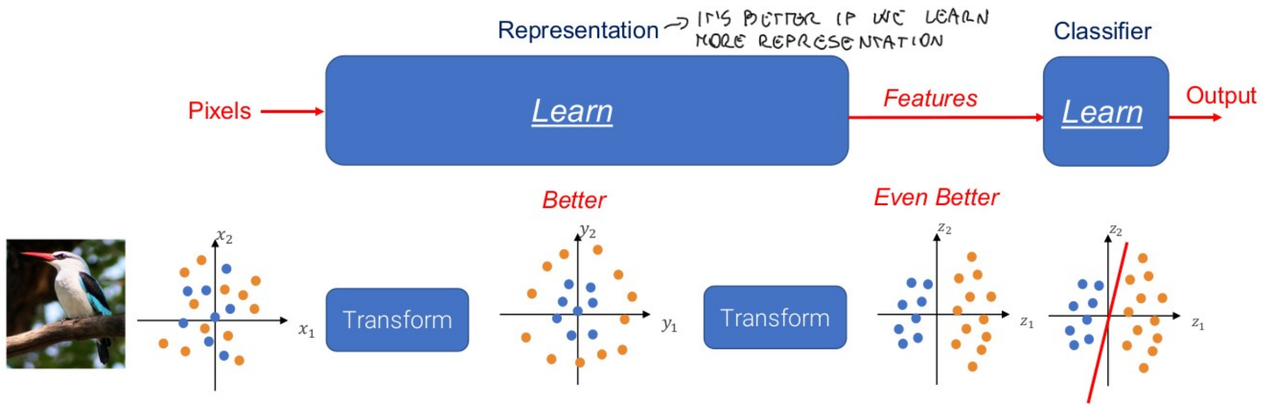
The used loss typically is the cross entropy loss. It measures the difference between two probability distributions: the predicted probability distribution outputted by the model and the true probability distribution (ground truth).

$$\boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{D}^{train}) = \sum_{i=1}^{N} -\log p_{model}\left(y = y^{(i)} \middle| \boldsymbol{x} = \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

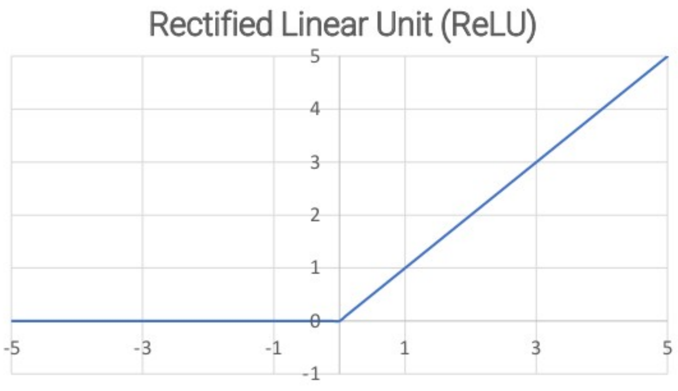
There are 3 sets: train, validation and test. We need to avoid overfitting and underfitting on training set. To do that regularization can be applied to loss function. Then this ccan be minnimized by gradient descent algorithms. Learning rate must be tuned. To be more efficient, SGD (Stochastig gradient descent) can be applied. It consists in updating the parameters based on the gradient computed on each per sample loss, so after processing each individual training sample. The update step of the parameters can be done after computing the loss for a series of mini-batch training samples (SGD with mini-batches). The number of iterations are called epochs. Training time is an hyperparameter.
Representation is important: transforming data into a better representation can make classification easier:


x_2
↳KEYPOINT
x_1
Switch to polar coordinates
$\rho = \sqrt{x_1^2 + x_2^2}$
$\theta = \tan^{-1}\frac{x_2}{x_1}$
θ
ρ
→ VECTOR OF INPUT DATA [x1, x2]. TWO CLASSES:
- ORANGE MEASUREMENT
- BLUE
LINEAR CLASSIFIER:
Non-linear decision boundary in input space
Linear decision boundary

Deep learning can be seen as a sort of representation learning:

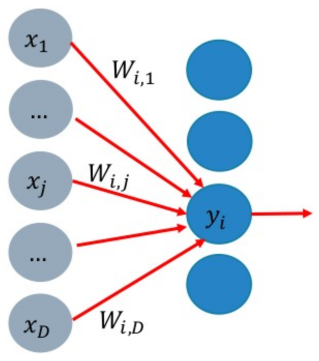


since a chain of linear layers is equivalent to a single linear layer, to realize representation learning effectively, we need to introduce non linear computations. We apply a chain of linear layers and non linear activations. This is called Neural Network. The representation layer is called hidden layer and its size is an hyperparameter. A non-linear function can be the Rectified Linear Unit (ReLU).



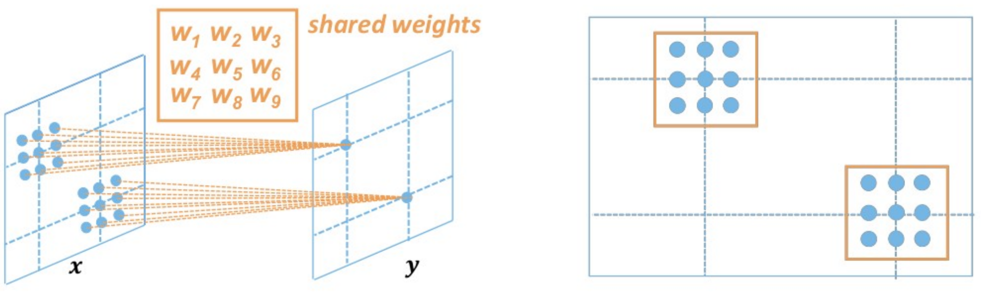
If each input is connected to each output in this way:

$$\phi(a) = \max(0, a)$$



We have a fully connected layer (FC). Those can be applied to detect local features. However, they require too many parameters and Flops (Floating point operations) to compute simple local features. For this reason we apply convolution.
In deep learning we deploy convolutional layers to detect features annd patterns based on filters learnt by minimizing a loss function.
In this conv-layers, each output unit is connected only to a –small- set of neighbouring input units. This realizes a so called local receptive field.

What a conv layer does compute ?

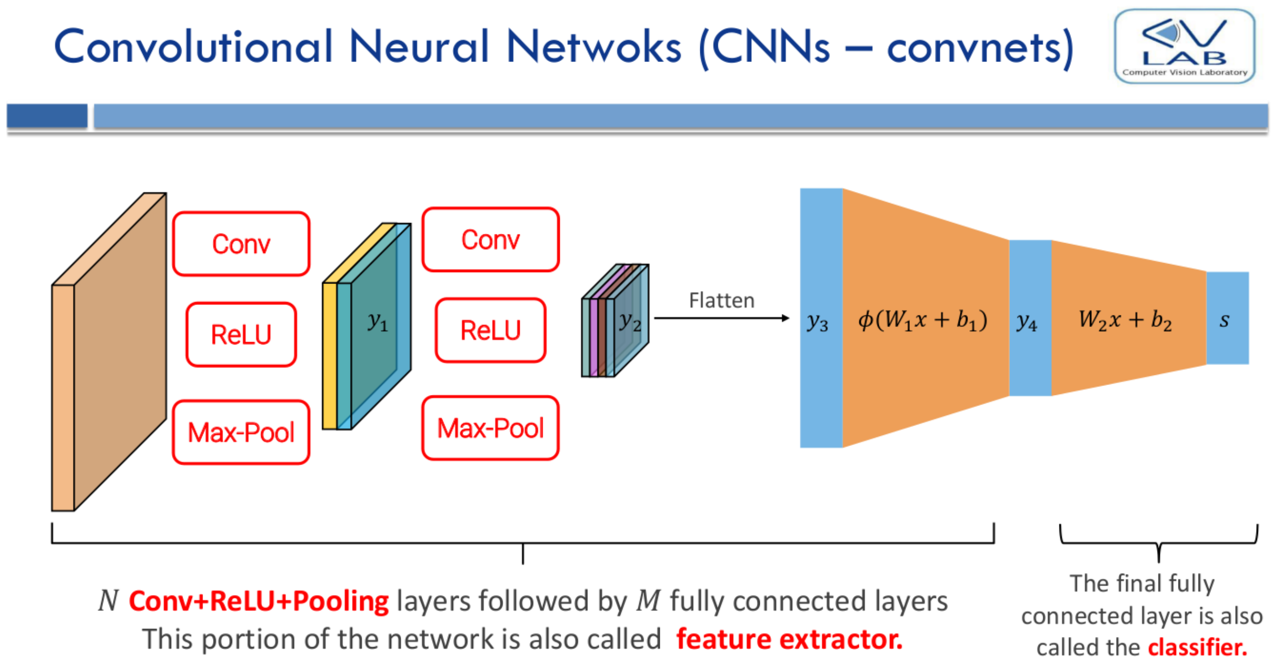


$$y(i,j) = w_1 x(i-1,j-1) + w_2 x(i-1,j) + w_3 x(i-1,j+1) + w_4 x(i,j-1) + w_5 x(i,j) + w_5 x(i,j+1) + w_7 x(i+1,j-1) + w_8 x(i+1,j) + w_9 x(i+1,j+1)$$

$$w = \begin{bmatrix} w(-1,-1) & w(-1,0) & w(-1,1) \\ w(0,-1) & w(0,0) & w(0,1) \\ w(1,-1) & w(1,0) & w(1,1) \end{bmatrix} \Rightarrow y(i,j) = \sum_{m=-1}^{m=1} \sum_{l=-1}^{l\_1} w(m,l)\,x(i-m,j-l)$$

Correlation !

RGB images have 3 channels, so convolution kernels must be 3-dimensional tensors of size $3 \times HK \times WK$.
By sliding the filter over the image, we get a single-channel output image called output activation of the conv layer. They are also called feature maps, as the learnt filters tend to specialize in detecting specific features.
As we traverse the chain of conv layers the receptive field gets larger and large, so we detect features dealing with larger and larger image regions (from local to global features).
We can apply strided convolution (filter doesn’t applied everywhere) and pooling layers (Aggregate neighbouring values into a single output by a specific hand-crafted function) or Max Pooling to get a down-sampled output.


Convolutional Neural Netwoks (CNNs – convnets)

N **Conv+ReLU+Pooling** layers followed by M fully connected layers
This portion of the network is also called **feature extractor.**

The final fully connected layer is also called the **classifier.**