Biologically Inspired Computation

Dr Marta Vallejo

m.vallejo@hw.ac.uk

Lecture 6

Deep Neural Networks Architectures

- Unsupervised Pretraining Networks
  - Deep Belief Networks
  - Autoencoders
  - Denoising Autoencoders
  - Stacked (Denoising) Autoencoders
  - Unsupervised Layer-wise pretraining process
- Convolutional Neural Networks
  - Cells in the visual cortex
  - Filters/Kernels
  - Pooling Layer
  - Visualisation of CNN
- Emergent Architectures
  - Deep Spatio-Temporal Neural Network
  - Multi-dimensional Recurrent Neural Network
  - Convolutional Autoencoder

Unsupervised Pretraining Networks

Unsupervised Pretraining Networks are topologies with directed and undirected edges, where layers are connected to each other, but units are not.

The training is executed in **two phases**:

- **Unsupervised pre-training**: the layers are stacked sequentially and trained in a layer-wise manner using unlabelled data. This step is used to initialise each layer and capture multiple levels of representation simultaneously (a new transformation is composed by previous learned transformations).

- **Supervised fine-tuning:** an output classifier layer is stacked, and the whole neural network is optimised by retraining with labelled data.

In general, in unsupervised learning, the deep layers only marginally improve the classification performance as opposed to supervised learning where the deeper layers are much more helpful.

Unsupervised Pretraining Networks

Advantages:

- The pre-trained weights are closer to the optimal than the randomly generated ones → speed-up the training
- Since these networks exploit unlabelled data, they can help avoid overfitting
- Overfitting is avoided even when labelled data is insufficient as is common in the real world problems.

Examples of these networks are:

- Stacked Autoencoder (SAE)
- Deep Belief Network (DBN)

Examples of pre-trained models commonly used:

- **Word2Vec** trained on Google News or Wikipedia
- **VGG-16**, **Resnet152**, etc. trained on ImageNet

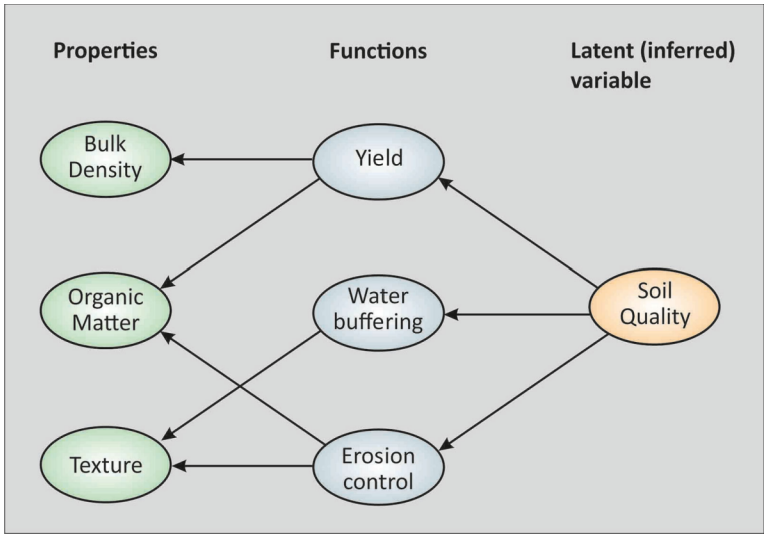
Deep Belief Networks

Deep Belief Networks are probabilistic models based on two concepts:

- Bayesian Belief Networks
- Restricted Boltzmann Machines

Bayesian Belief Networks:

Directed acyclic graphs composed by stochastic binary units. Edges represent relationships (causality) between features.



Each node *xi* has a conditional probability *p(xi|parents(xi))* showing the effect of the parents on the node.

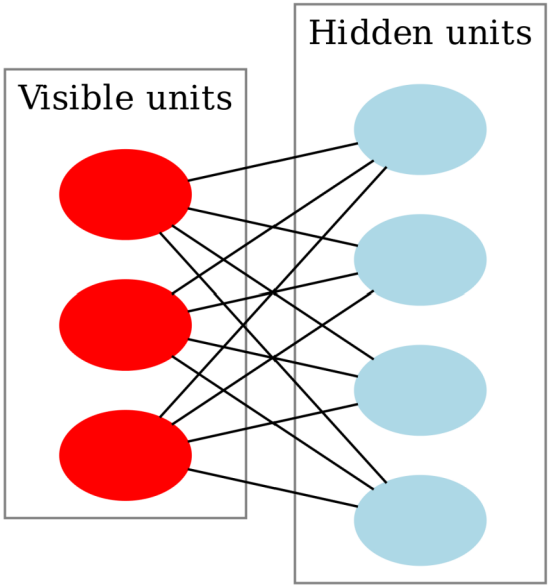
Deep Belief Networks

Restricted Boltzmann Machines

Undirected **energy based model** with only one layer of hidden units and one visible layer (binary states), with no connection between them.

It learns a probability distribution over a set of inputs.

$$p(v) = \sum_h p(h)\, p(v|h)$$



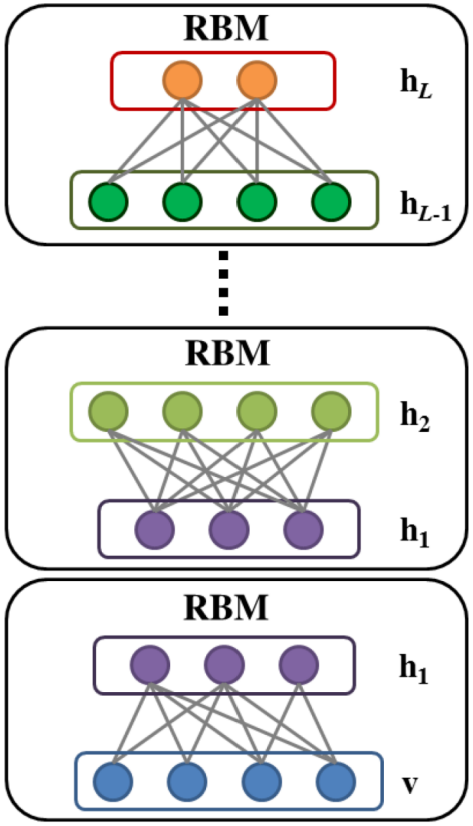
Constructive Divergence algorithm for training.

Hinton et al. (2006) A fast learning algorithm for deep belief nets

Deep Belief Networks

Deep belief network uses Restricted Boltzmann Machine as building blocks of the architecture forming a multilayer structure.

Unsupervised pre-training:



Payton et al. 2016

- They are trained as Restricted Boltzmann Machines
- The layers are added and trained sequentially from bottom (observed data) "greedy layer wise" to top (higher representation level)
- We do not need labelled data

When this layer-wise unsupervised training is finished, we have a network populated with good initialisation values.

After that, we **fine tune** the whole network in a "supervised way" using backpropagation or the wake-sleep algorithm (Hinton et al. A fast learning algorithm for deep belief networks)

Autoencoders

Autoencoder is an **unsupervised** deterministic mapping that try to learn an approximation to the identity function:

$$\hat{y}_{w,b}(x) \approx x$$

But, forcing the dimensions to reduce: then the model must learn the most **representational features.**

They allow the compression and decompression of data. This process is:

- Data-specific
- Lossy
- Learned automatically from examples rather than engineered by a human → it's easy to train

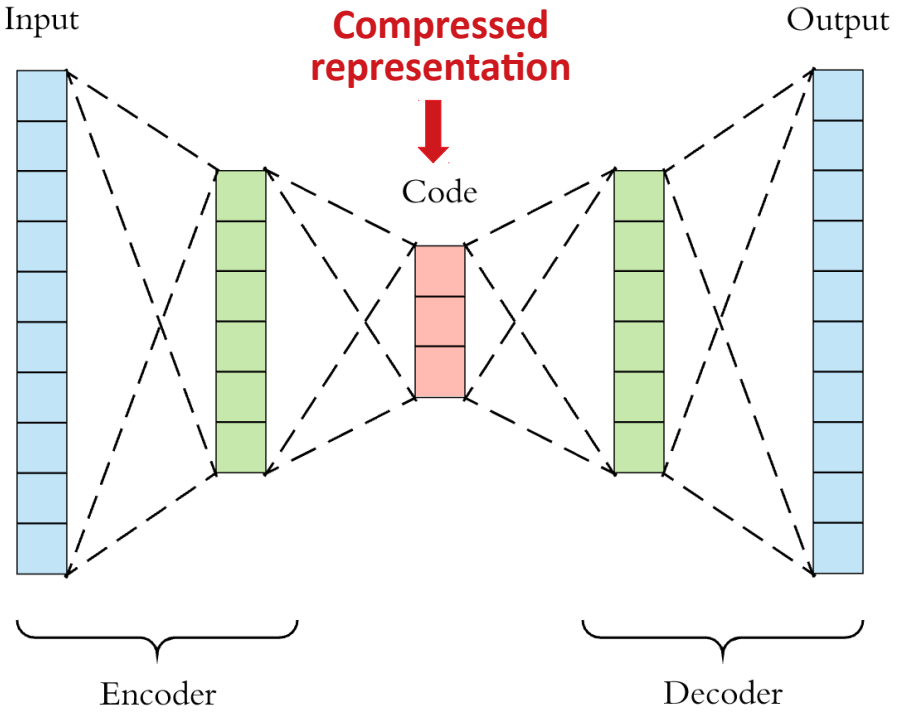
Concept: The compressed vector should be informative enough to get the original data back by decoding it.

Autoencoders: Structure

They require:

- an encoding function
- a decoding function
- the distance function that measures the amount of information loss (compressed-decompressed representation).

Objective: Minimise the error between the input and the output using a cost function. The function typically used is simply the distance from the original data.



Autoencoders

Autoencoders can be used for **dimensionality reduction** problems:

- Feature selection: each feature is called **latent variables**
- More compact representation of our search space: enhance the performance
- Alternative method to Principal Component Analysis (PCA): only linear feature representation. Select the ones with more variance using eigenvectors.
- Autoencoders can learn non-linear feature representation

Disadvantages:

- Lack of **interpretability** of non-visual data
- Complicated calculations
- Slower than PCA
- Prone to overfitting

Types:

- Vanilla autoencoder
- Multilayer autoencoder
- Convolutional autoencoder
- Regularised autoencoder

To know more: https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Autoencoders

1. Define a feature-extraction function:

$$h = f_\theta(x)$$

h : feature vector
x : input
f_θ : encoder

2. Define a reconstruction function:

$$r = g_\theta(h)$$

feature space → input space
r : reconstruction function
g_θ : decoder

3. Goal: minimise the reconstruction error between x and r

$$\Im_{AE}(\theta) = min \sum_i L(x^{(i)}, g_\theta(f_\theta(x^{(i)})))$$ using stochastic gradient descent

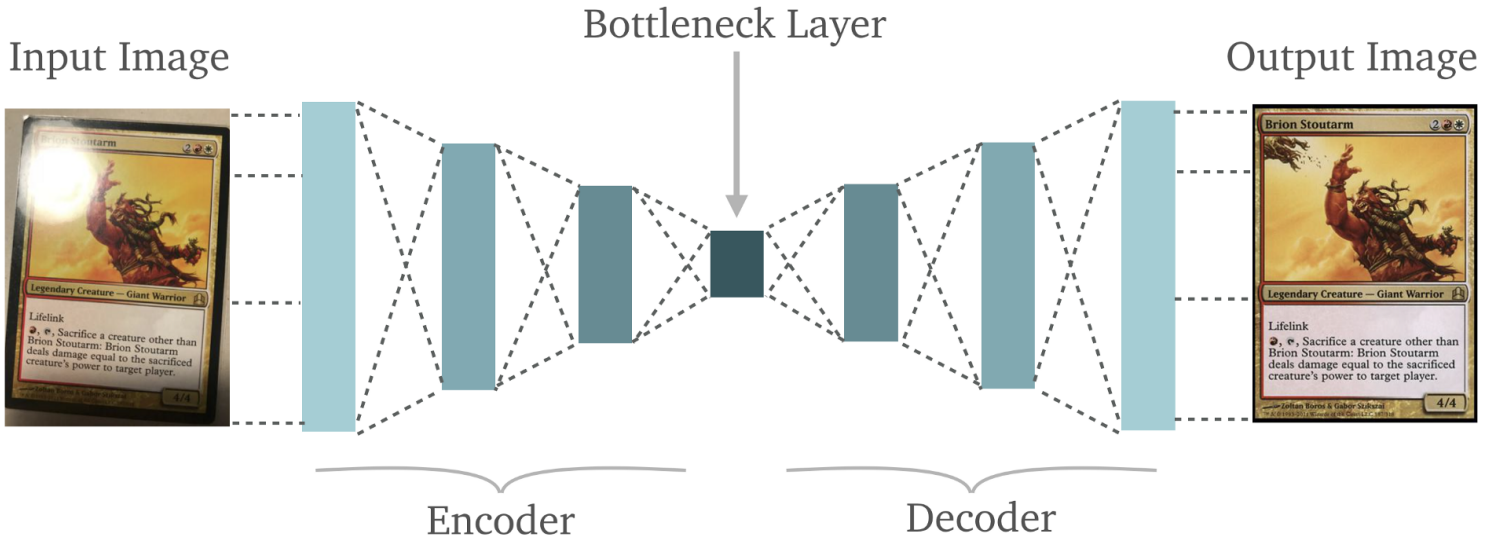
Stochastic gradient descent: online version of gradient descent where you deal with the training samples one by one instead processing all at a time

$$f_\theta(x) = S_f(Wx+b)$$
$$g_\theta(x) = S_g(W'x+b)$$

S_f and S_g : activation functions

Denoising Autoencoder

Almost the same than an autoencoder, but we add **noise** to the input data.
Doing that we are able to reconstruct data from a corrupted input data.
The hidden layer will learn only the more robust features, rather than just the identity (like in the normal autoencoder).



$$\Im_{AE}(\theta) = min \sum_i E_q(\bar{x}|x^{(i)})[L(x^{(i)}, g_\theta(f_\theta(x^{(i)})))]$$

Averages over corrupted samples $\bar{x}$

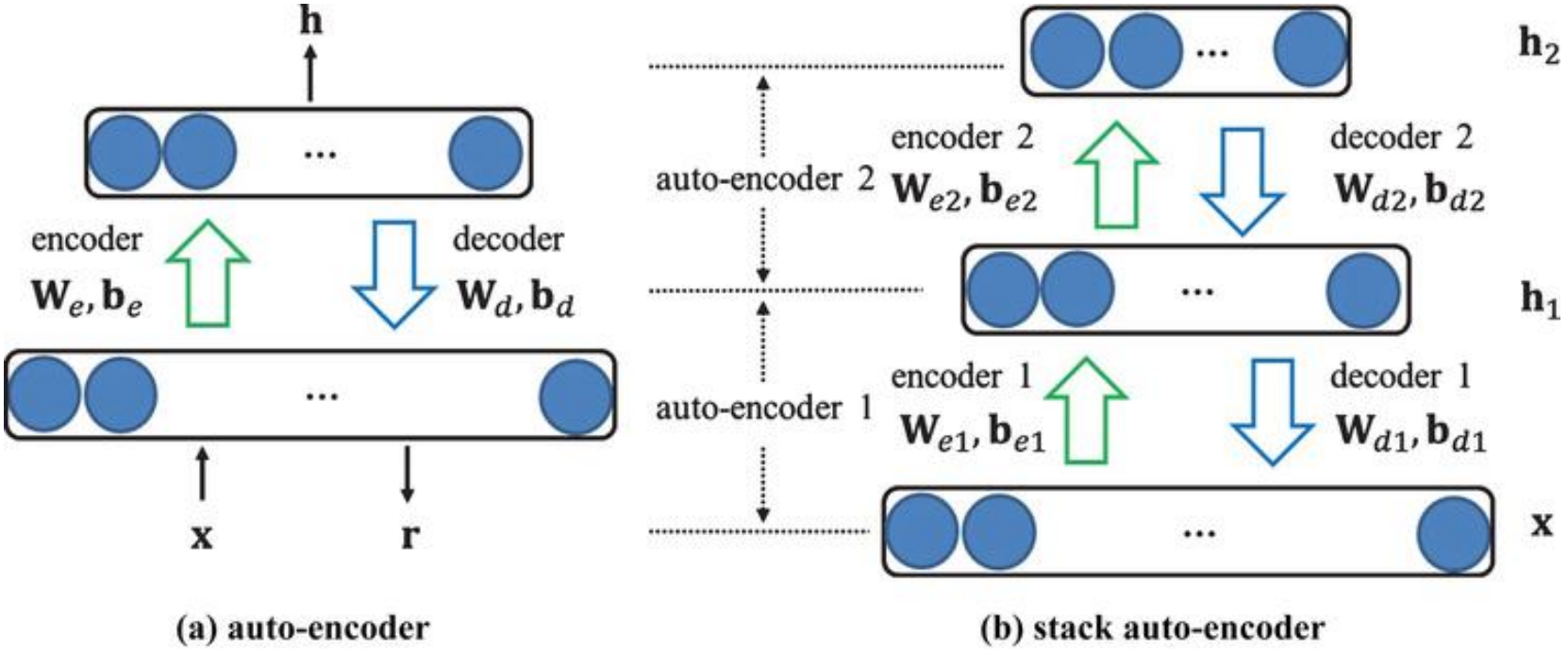
Training using stochastic gradient descent

Image: https://hackernoon.com/a-deep-convolutional-denoising-autoencoder-for-image-classification-26c777d3b88e

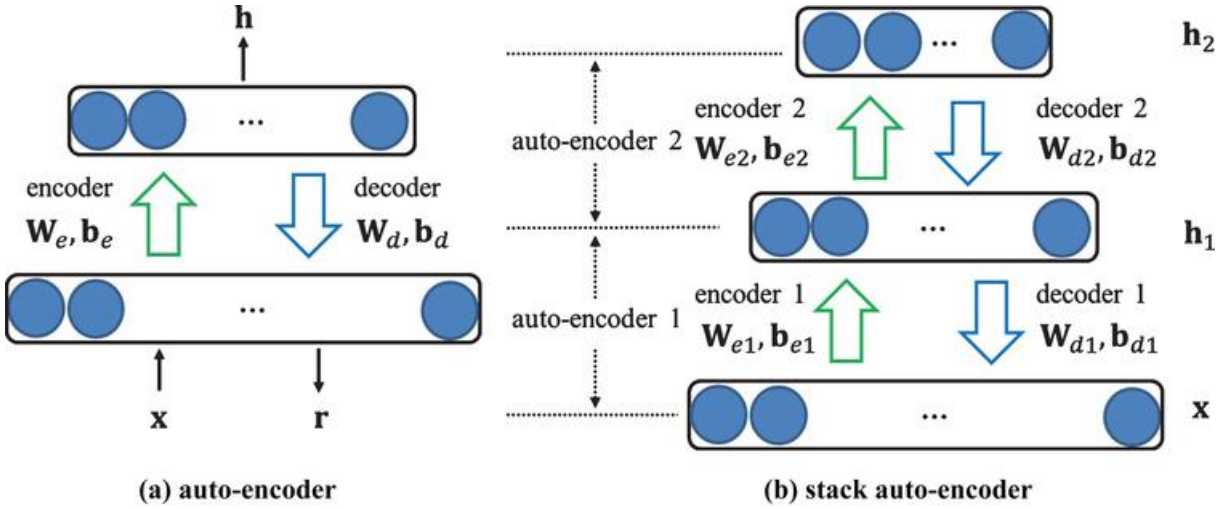
Stacked (Denoising) Autoencoder

Stacking layers of autoencoders (both normal and denoising), produces a deeper topology known as **Stacked** or **Deep (denoising) Autoencoders** It is trained layer by layer, by trying to minimise the error in the layer's reconstruction compared to it's input.

By using them in an unsupervised manner, we can learn starting weights prior to the main supervised training procedure with deep learning.



Pre-training process in SAE and DBN



- First, weight vector W1 is trained between input units x and hidden units h1 in the first hidden layer as an RBM or AE.
- After the W1 is trained, another hidden layer is stacked, and the obtained representations in h1 are used to train W2 between hidden units h1 and h2 as another RBM or AE.
- The process is repeated for the desired number of layers.
- Once the stacked auto encoder is trained, its output can be used as the input to a supervised learning algorithm

Convolutional Neural Networks

A Convolutional Neural Network (CNN or ConvNet) is a type of feed-forward neural network made up of neurons that have learnable weights and biases, very similar to ordinary multi-layer perceptron.

CNNs are designed to process different data types, especially two-dimensional images. They are directly inspired by the primary visual cortex of the brain.

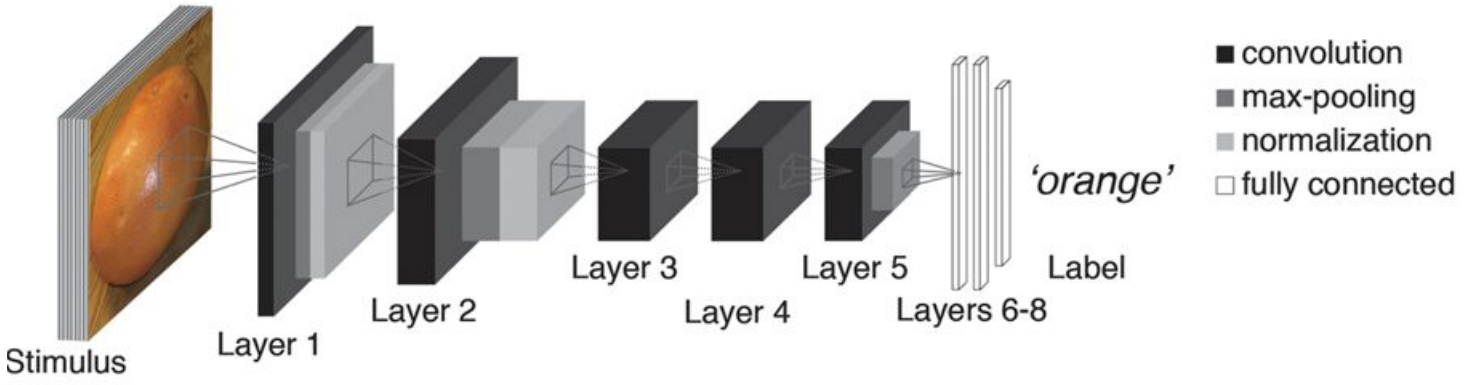
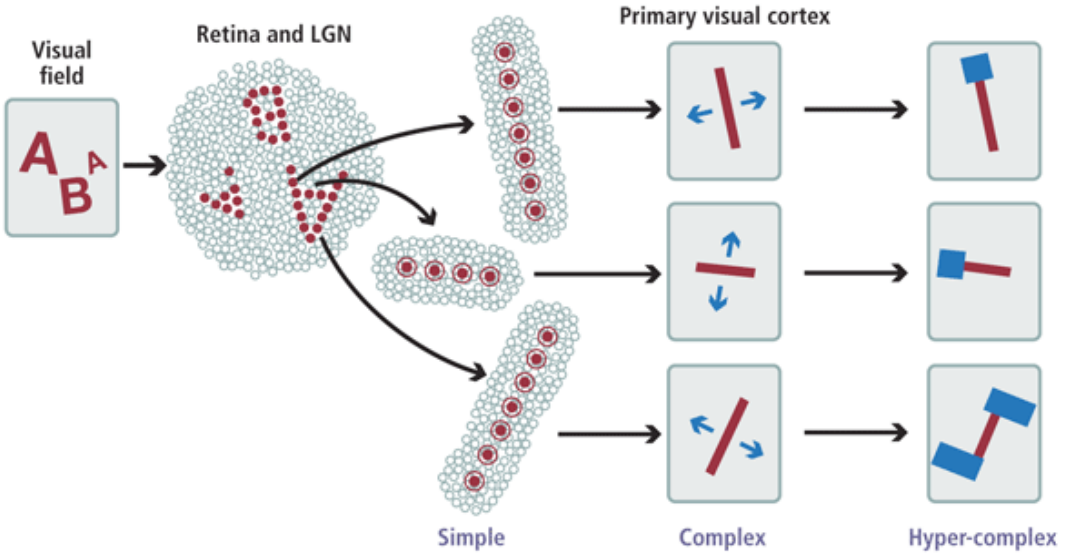


Image: https://www.nature.com/articles/srep27755/figures/1

Type of Cells in the Visual Cortex

In the primary visual cortex of the brain, cells can be divided into:

- **Simple Cells**: respond to points of light or bars of light in a particular orientation
- **Complex Cells**: respond to bars of light in a particular orientation moving in a specific direction.
- **Hypercomplex Cells**: respond to bars of light in a particular orientation, moving in a specific direction and of a specific line length.



Complex cells synthesise the information from simple cells to identify more intricate forms.

Image: KMichmizos-Brain-Morphism.pdf

Convolutional Neural Networks

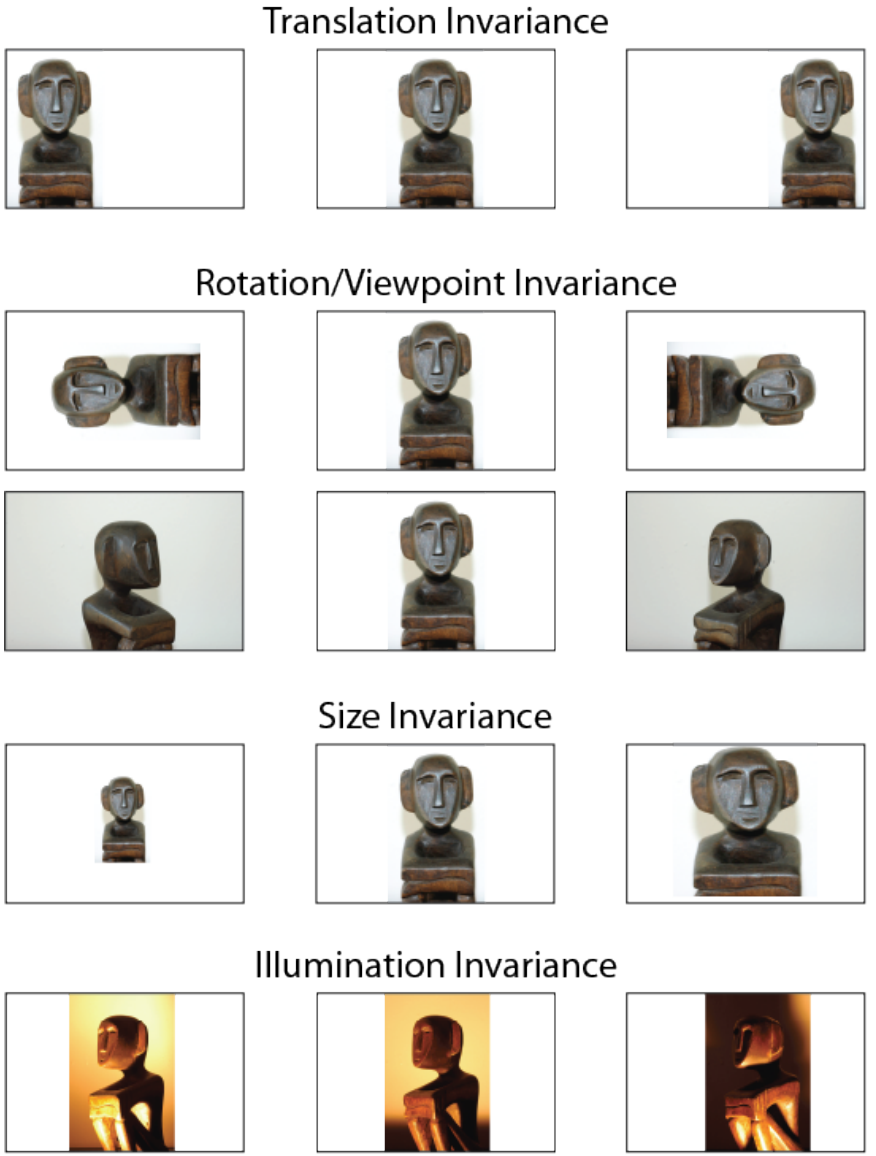
Popular deep CNNs used in computer vision:

- **AlexNet**
  Created by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton
  Won ILSVRC 2012
  8 layer net
  https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py
- **VGG**
  Built by Visual Geometry Group (University of Oxford)
  ILSVRC 2014
  19 Convolutional layer net
  https://www.cs.toronto.edu/~frossard/vgg16/vgg16_weights.npz
- **Inception GoogLeNet**
  Created by Google
  ILSVRC 2014
  22 Convolutional layer net
  https://github.com/conan7882/GoogLeNet-Inception-tf
- **ResNet (Residual Neural Network)**
  Created by Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun
  ILSVRC & COCO 2015 Competitions: 1st place in all main tracks
  152 Convolutional layer net
  https://github.com/KaimingHe/deep-residual-networks

Convolutional Neural Networks

CNNs have the following characteristics:

- **Local connectivity**: Each neuron only receives input from a small local group of pixels in the input image (these inputs are close to each other)
- **Translational Invariance**: it is able to recognised them with the help of pooling layers.
- **Rotation Invariance**: no directly
- **Size Invariance**: no directly
- **Illumination Invariance**: also achievable due to the pooling layers.



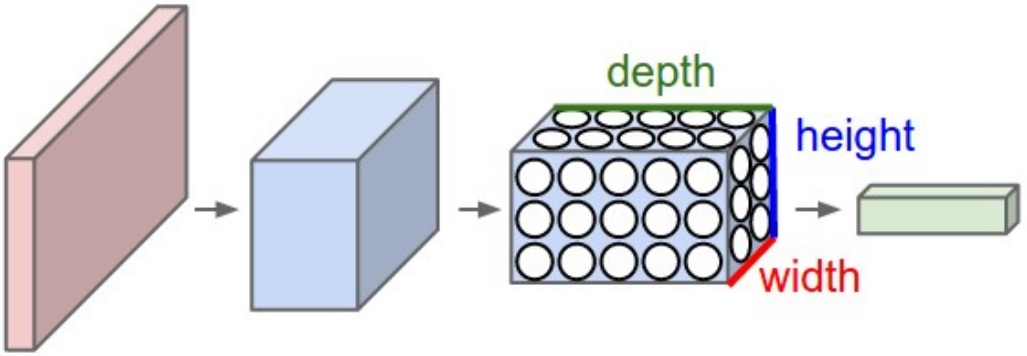
Rotation and size invariance can be achievable with large datasets with numerous variations or by applying preprocessing techniques to your data.

Image: An Enhanced Approach in Image Mosaicing using ORB Method with Alpha Blending Technique

Convolutional Neural Networks

Some considerations:

- Layers are organised in 3 dimensions: width, height and depth.
- The neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it.
- The final output will be reduced to a single vector of probability scores, organised along the depth dimension.



Basic Components:

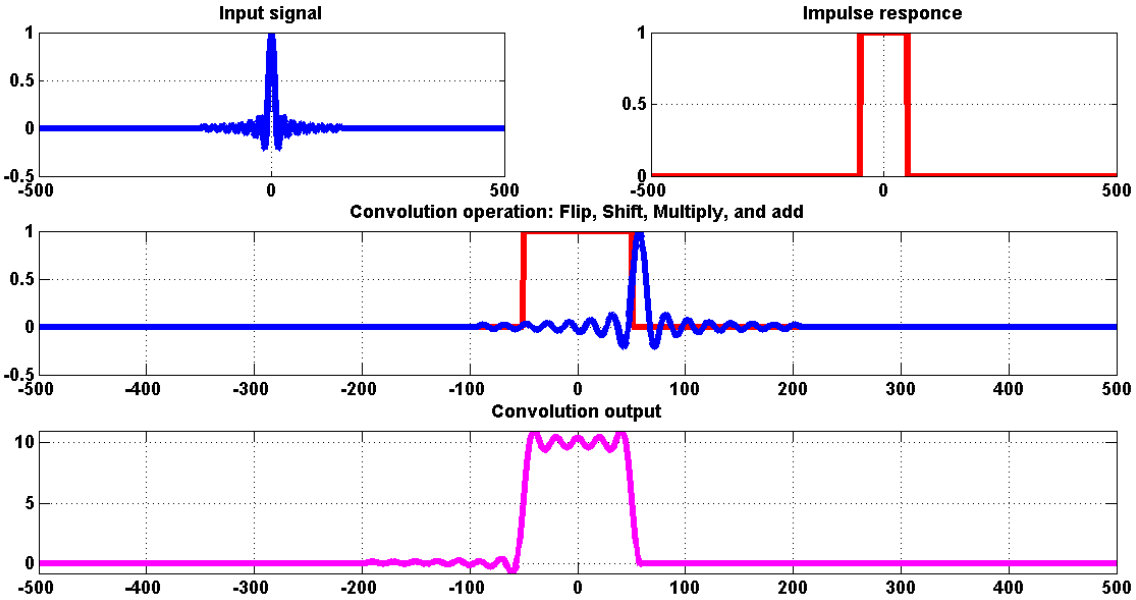
- **The hidden layers/Feature extraction**: Here the network performs a series of **convolutions** and **pooling** operations. Features are detected

- **The classification**: fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image.

Image: One Intelligent Agent to Rule Them All - Laurin Paech

Convolutional Neural Networks

Convolution refers to the mathematical combination of two functions to produce a third one that expresses how the shape of one is modified by the other.



Intuitively, we are going to take an image and a filter as input and produces a filtered output by applying convolution. In CNN terminology the filter is called **filter/kernel** and the produced filter output is called **feature map.**

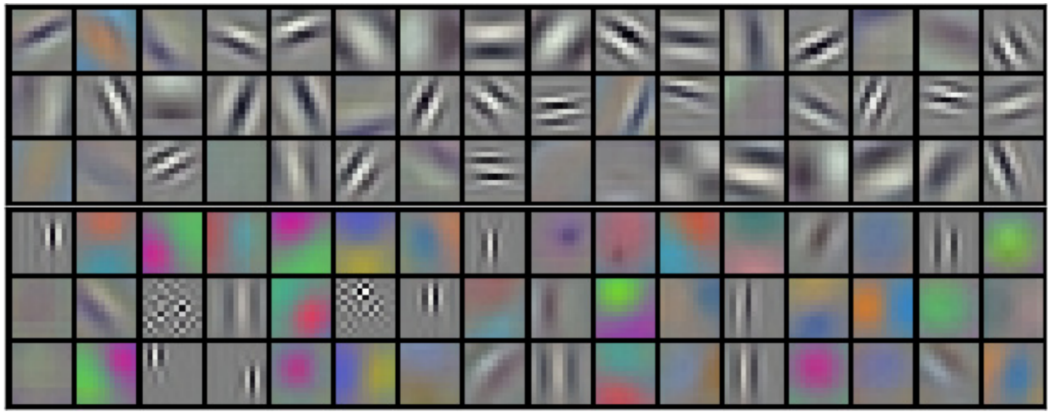
Image: https://uk.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/43642/versions/1/screenshot.png

CNN: Filters / Kernels

CNN uses filters to **extract features** of an image. **Feature maps** are groups of local weighted sums that represents highly correlated sub-regions of data.

Historically, these filters were often a set of weights modelled with mathematical functions like Gaussian, Laplacian or Canny filter.

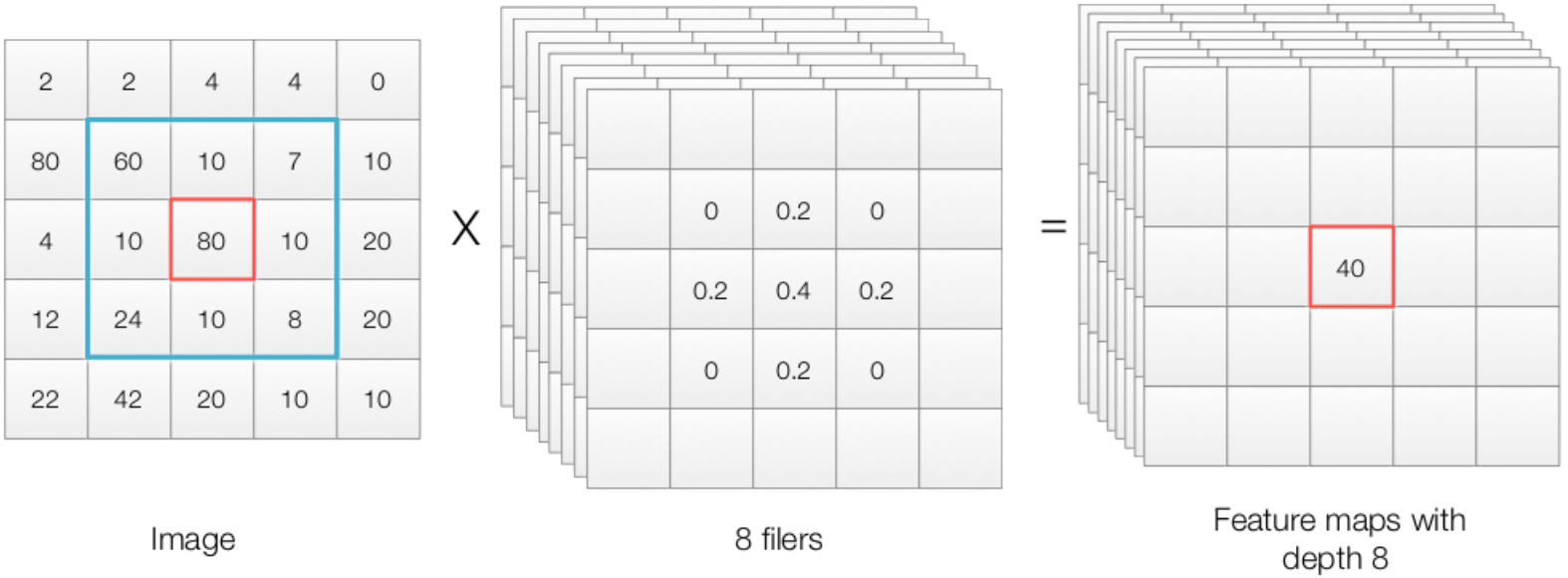
CNN provide a machinery to learn these filters from the data directly and have been found to be superior (in real world tasks) compared to historically crafted filters.



Here are the 96 filters learned in the first convolution layer in AlexNet. Many filters turn out to be edge detection filters common to human visual systems. (Source Krizhevsky et al. 2012)

CNN: Filters / Kernels

Since identical patterns can appear regardless of the location in the data (translational invariance), **multiple filters** are applied repeatedly across the entire dataset, which also improves training efficiency by reducing the number of parameters to learn.



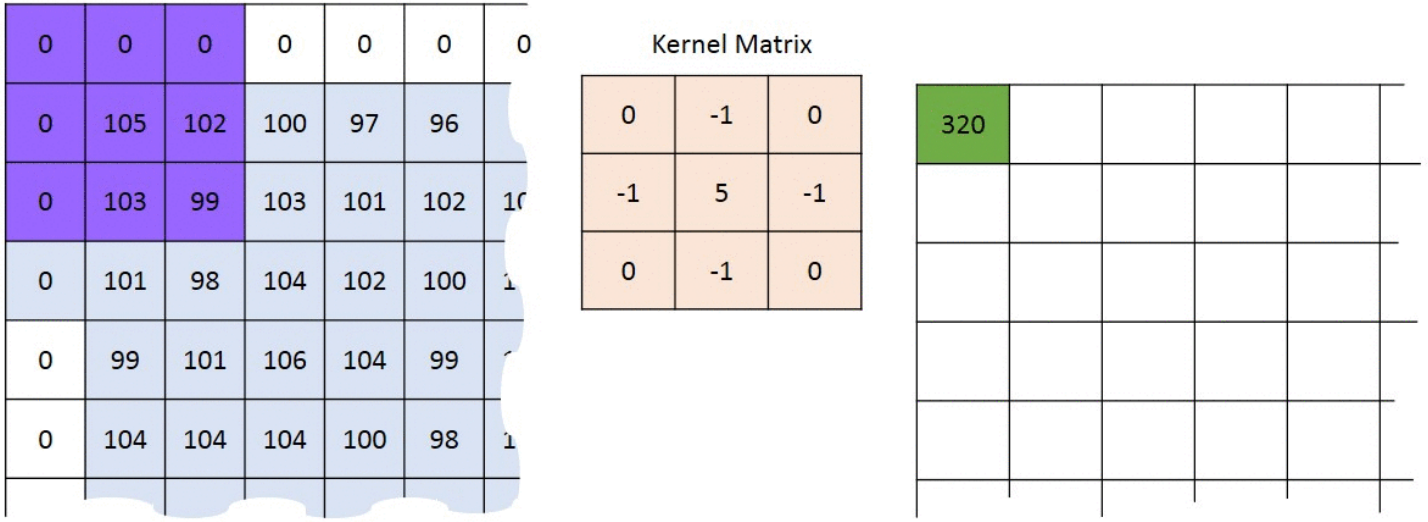
https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

If n-filters are applied, the feature map created has depth n. Then, this process results in different feature maps, detecting low-level filters.

The filters are initialised at random. In general smaller filters perform better

Convolutional Neural Networks

We execute a convolution by sliding the filter over the input. At every location, a matrix multiplication is performed, summing the result onto the feature map.



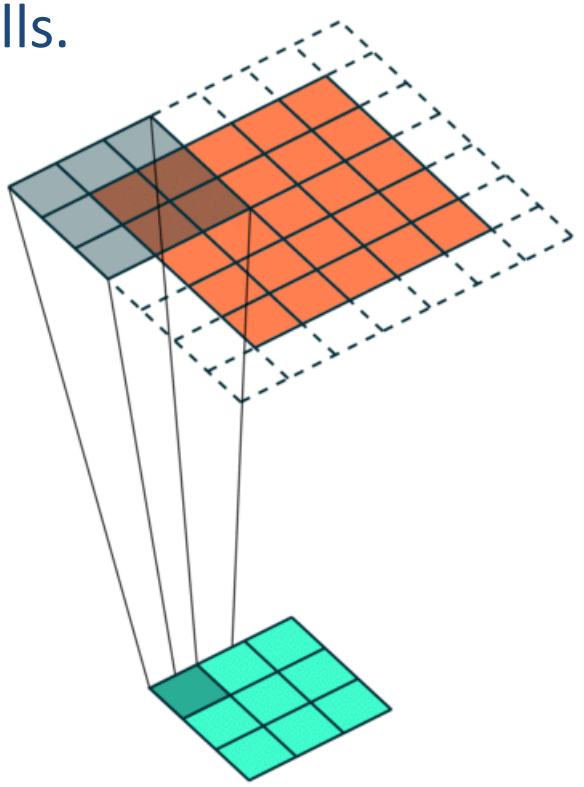
Convolution with horizontal and vertical strides = 2

Convolutional Neural Networks

Stride is the size of the step the convolution filter moves each time. A stride size is usually 1, meaning the filter slides pixel by pixel.

By increasing the stride size, your filter is sliding over the input with a larger interval and thus has less overlap between the cells.

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

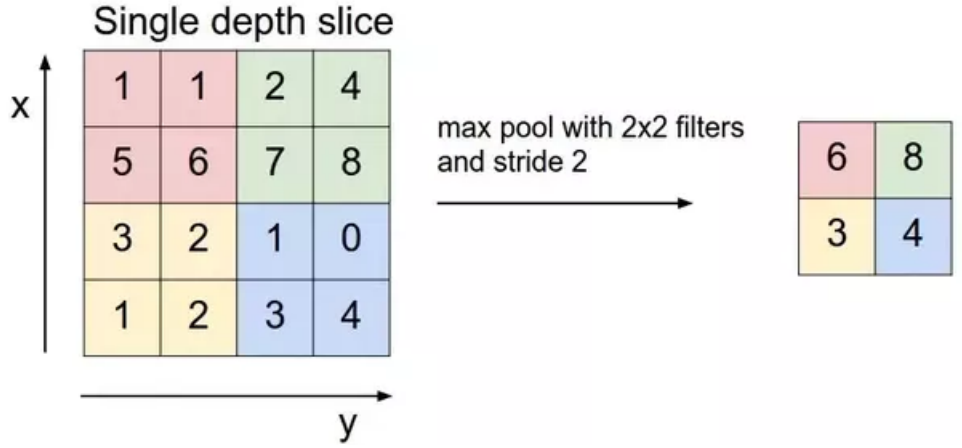


Because the size of the feature map is always smaller than the input, we have to do something to prevent our feature map from shrinking. This is where we use **padding**.

CNN: Pooling Layer

Finally we take all of these feature maps and put them together as the final output of the convolution layer.

The **pooling layer** combines them into higher-level features. Pooling layers reduce the size of our feature map.



At each pooling layer, the **maximum** activation of the units in a small region of the previous convolutional layer is computed.

This non-overlapping subsampling allows higher-layer representations to be invariant to small translations of the input (abstraction) and reduces computational cost

CNN: Batch Normalisation

After applying filters on the input, we apply a batch normalisation followed by a ReLU for adding non-linearity to our output by inserting the BatchNorm layer immediately after the fully connected/convolutional layers and before the non-linearities.

Batch normalisation:

- allows the model to converge much faster in training and therefore you can use higher learning rates.
- networks are significantly more robust to bad initialisation.
- achieve a more stable gradient propagation

However, a degrading problem can still prevent the network from achieving better results.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy

Transfer Learning Steps for CNNs

Encoding knowledge gained while solving one problem and applying it to a different but related problem. Since we are not retraining the entire model nor the convolutional layers, this is **incredibly fast** to run.

Steps to follow:

1. Load the pretrained model but exclude the top layers of the model (dense layers). Essentially we are just extracting the CNN layers.

2. Add the dense layers of your choice (final one should contain the number of classes you have to predict). We change this because their task was different from ours.

3. Train your new model again on your data. Make sure your input images are in the same format as the pretrained model. The reason it is fast is because it is already reasonably close to an optimum and does not have to find it from scratch.

Visualisation of CNN

Visualisation of the values of trainable filters, feature maps and hidden units from layer to layer plays a very important **diagnostic role**.

It allows us to inspect the function of intermediate layers and, therefore, better understand how data are converted/processed inside a deep model .

Several approaches for understanding and visualizing CNN have been developed in the literature:

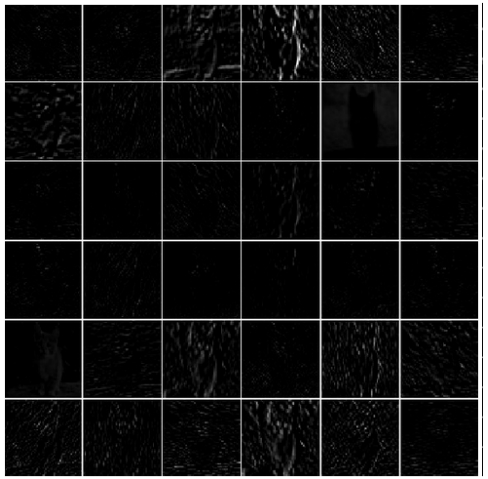
- Visualising the layer activations
- Visualising first-layer weights
- Retrieving images that maximally activate a neuron
- Embedding the codes with T-distributed Stochastic Neighbour Embedding (t-SNE)
- Occluding parts of the image

Try this interactive page for visualization of Convolutional Neural Networks (hand written digits)

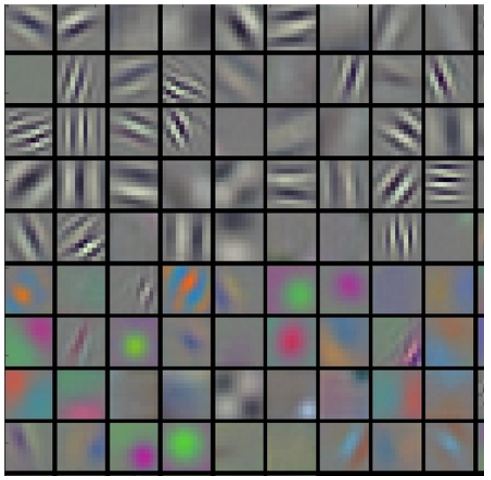
http://scs.ryerson.ca/~aharley/vis/conv/

Visualisation of CNN

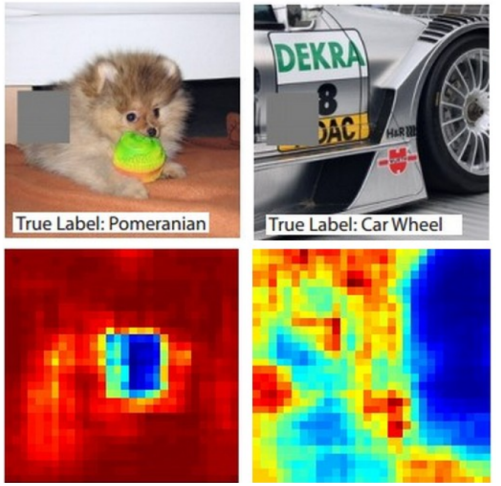
Layer activations



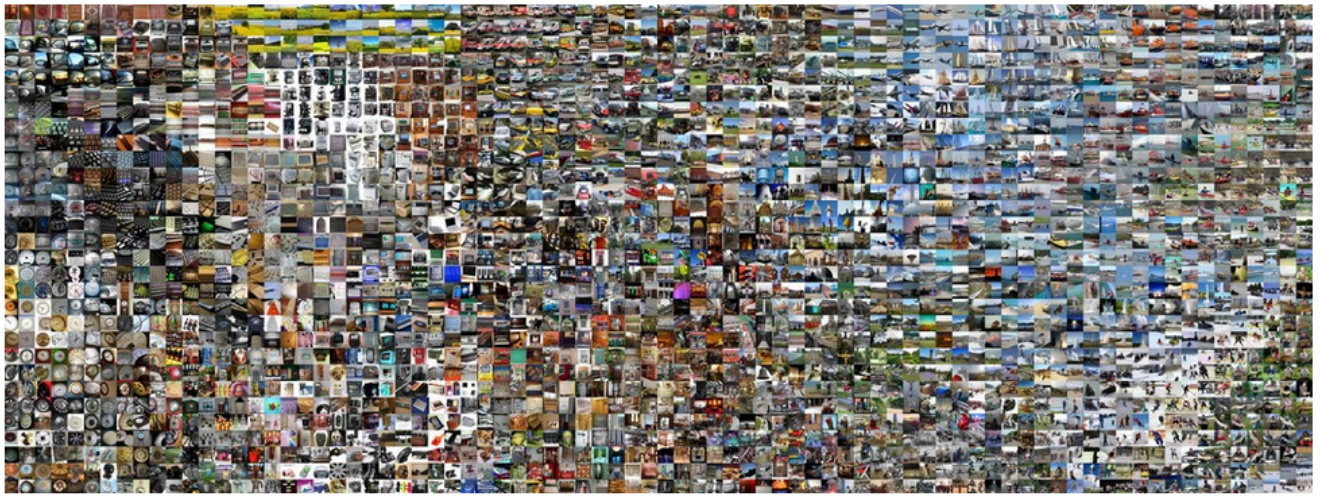
First-layer weights



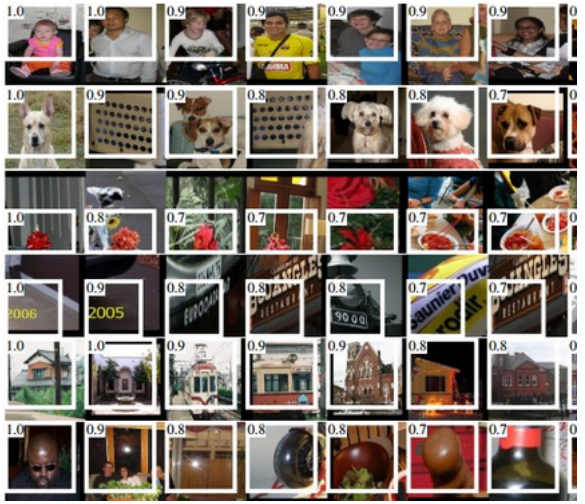
Occluding parts of the image



Embedding the codes with t-SNE



Maximally activate a neuron



Emergent Architectures

Emergent architectures: architectures besides DNNs, CNNs and RNNs. In general, new models consist of a composition of the traditional architectures. Here it will be introduced three of them:

- Deep Spatio-Temporal Neural Network (DST-NN)
- Multi-Dimensional Recurrent Neural Network (MD-RNN)
- Convolutional Autoencoder (CAE)

Deep Spatio-Temporal Neural Networks

Lacks in the deep architectures seen up to now:

- RNN: lack of spatial modelling
- CNN: lack of sequential/temporal modelling

Combining in a network the capability of modelling spatial data and sequential data at the same time allow us to approach the following problems:

- Human-object interaction understanding
- Human motion, emotion recognition
- Driver manoeuvrer anticipation

There is not a standard topology. The main idea is to build multiple networks and wire them together in order to capture some complex structure.

Attempts can be classified in different types:

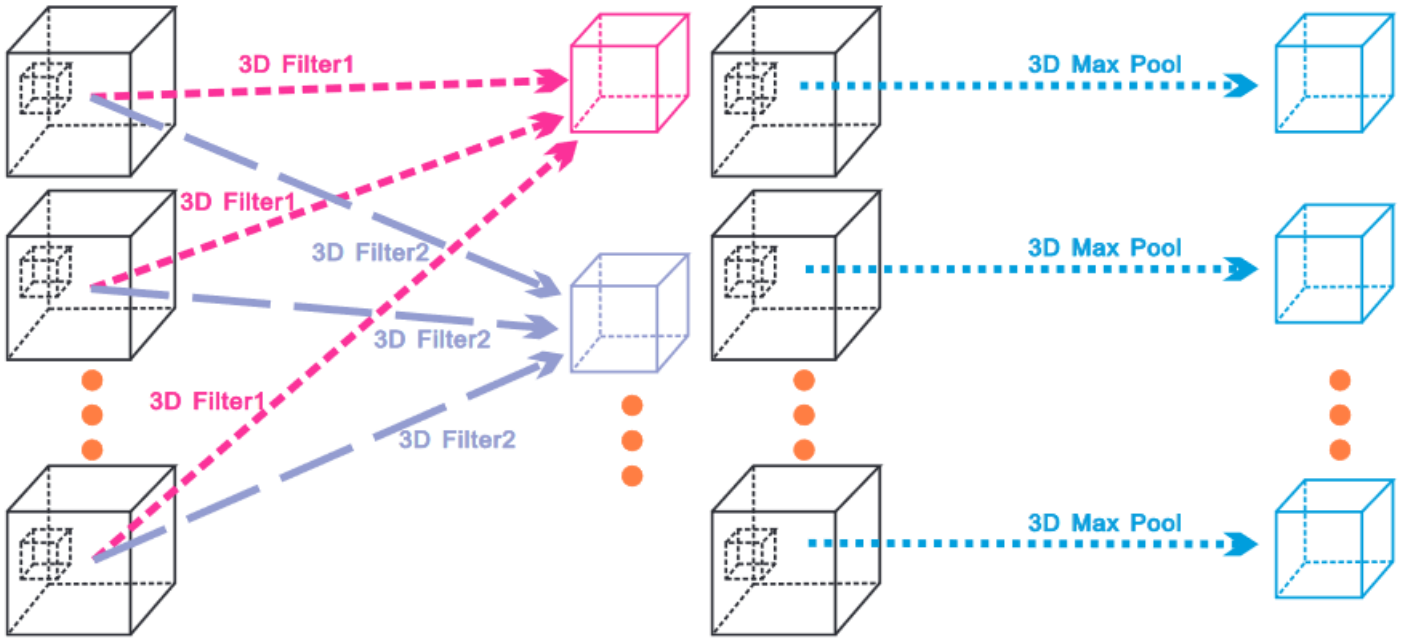
- Connecting a stack of CNN together: treating video frames as still images
- Connecting a fully CNN with a RNN (LSTM or Deep Belief Networks)
- Connecting multidimensional RNNs

Limit amount of available datasets

3-Dimensional Convolutional Neural Networks (C3D)

A depth channel as an additional channel, along with the RGB channels.

A 3-dimensional stack of CNN with the same topology. Straightforward, but it does not make full use of the geometric information in the data



(a) 3D convolution layer (b) 3D max pooling layer

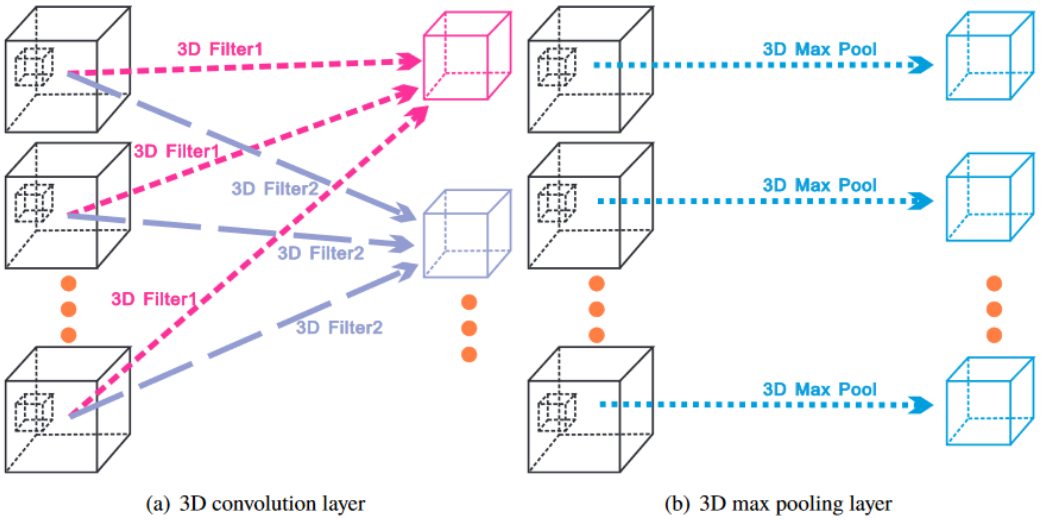
(Pengcheng et al. 2018)

Conventional CNNs are computationally intensive -> C3D push the computational requirements **into another level** (each computation depends on multiple images)

3-Dimensional Convolutional Neural Networks (C3D)

C3D can be defined as:

$$CNN_{i,j}^{k}$$



(Pengcheng et al. 2018)

where i and j are the usual spatial coordinates of the contact map, and k is a "temporal" index.

Typically, the input layer is a 4-dimensional tensor (**depth**, height, width, channel). Then, the size of weights for each channel in each filter should be **m×n×k**

However, it only works if the video is a sequence of similar images

3-Dimensional Convolutional Neural Networks (C3D)

Procedure:

1. When we move the filter across the input layer, we do it in 3 directions, along three axes. A filter contains a set of weights and a bias, and applying them to the input layer will result in a new **3D cuboid**.

2. The filters will finally produce a new 4D tensor, which is ready for the next 3D convolution.

3. For 3D pooling, the principle is the same as in the 2D case. We reduce the size in the first three dimensions and leave the channel unchanged.

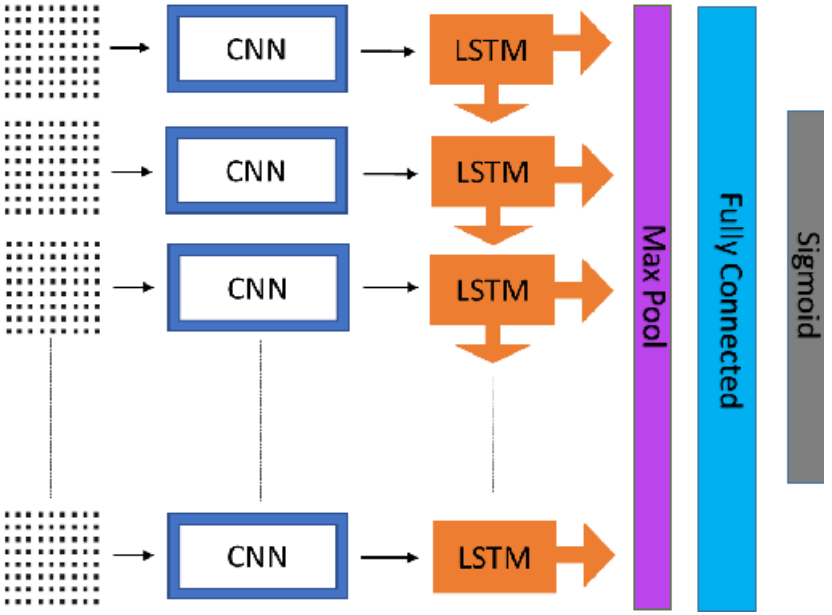
4. Finally, we average the predictions at the video level.

Spatio-Temporal: CNN + RNN

RNN networks operate **on frame-level** CNN activations, and can learn how to integrate information over time.

The spatio features are initially automatically learned using a CNN architecture for obtaining global video-level descriptors. These features are then treated as input to the RNN by **transforming a 3D output for each time step into a 1D output**

By sharing parameters through time, both architectures are able to maintain a constant number of parameters, while capturing a global description of the video's temporal evolution, which help discriminate among actions.



(Maddula et al. 2017)

Read more in: Beyond short snippets: Deep networks for video classification. Ng 2015

Two-stream CNN for spatial and temporal features

Using a stack of consecutive video frames means that the model should learn spatio-temporal motion-dependent features in the first layers → **Hard**

A video recognition architecture that is divided into **two streams** with softmax scores that are combined by late fusion.

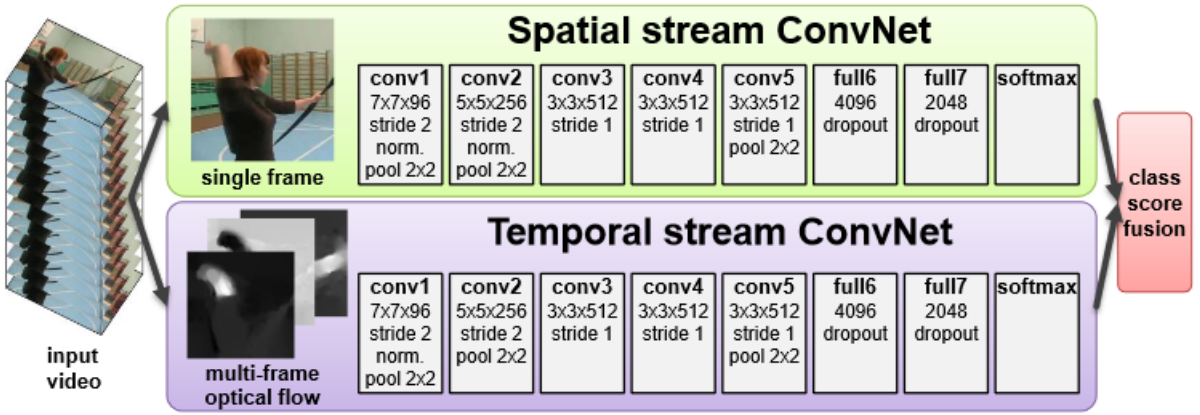


Figure 1: **Two-stream architecture for video classification.**

Simonyan et al. 2014

- **Spatial stream:** image classification architecture → action recognition from single still images.
- **Temporal stream:** input is a description of the motion between video frames (a stacking optical flow of displacement vector fields between several consecutive frames)

Two-stream CNN for spatial and temporal features

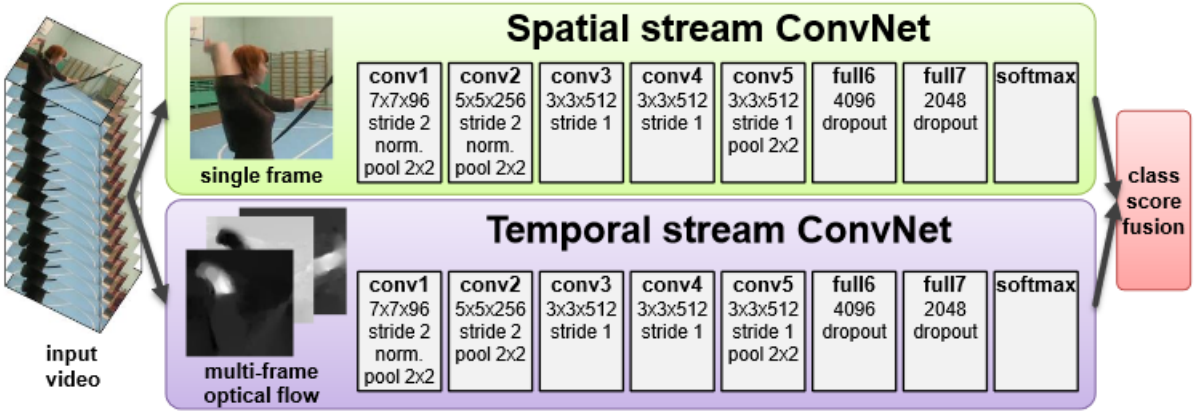


Figure 1: **Two-stream architecture for video classification.**

Configuration details:

Simonyan et al. 2014

- Use of ReLU in all hidden weight layers
- max-pooling performed over 3×3 spatial windows with stride 2
- normalisation
- The only difference between spatial and temporal CNN is that the second normalisation layer from the latter is removed to reduce memory consumption.
- Using of pretraining techniques
- Training is the same for both spatial and temporal nets: mini-batch stochastic gradient descent with momentum

To know more: Two-Stream Convolutional Networks for Action Recognition in Videos Simonyan 2014

Multi-Dimensional Recurrent Neural Network

Standard RNNs are inherently one dimensional → poorly suited to multidimensional data like images.

It assumes some ordering on the multidimensional data.

IDEA: replace the single recurrent connection with as many connections as there are spatio-temporal dimensions in the data. These connections allow the network to create a flexible internal representation of surrounding context, which is robust to localised distortions.

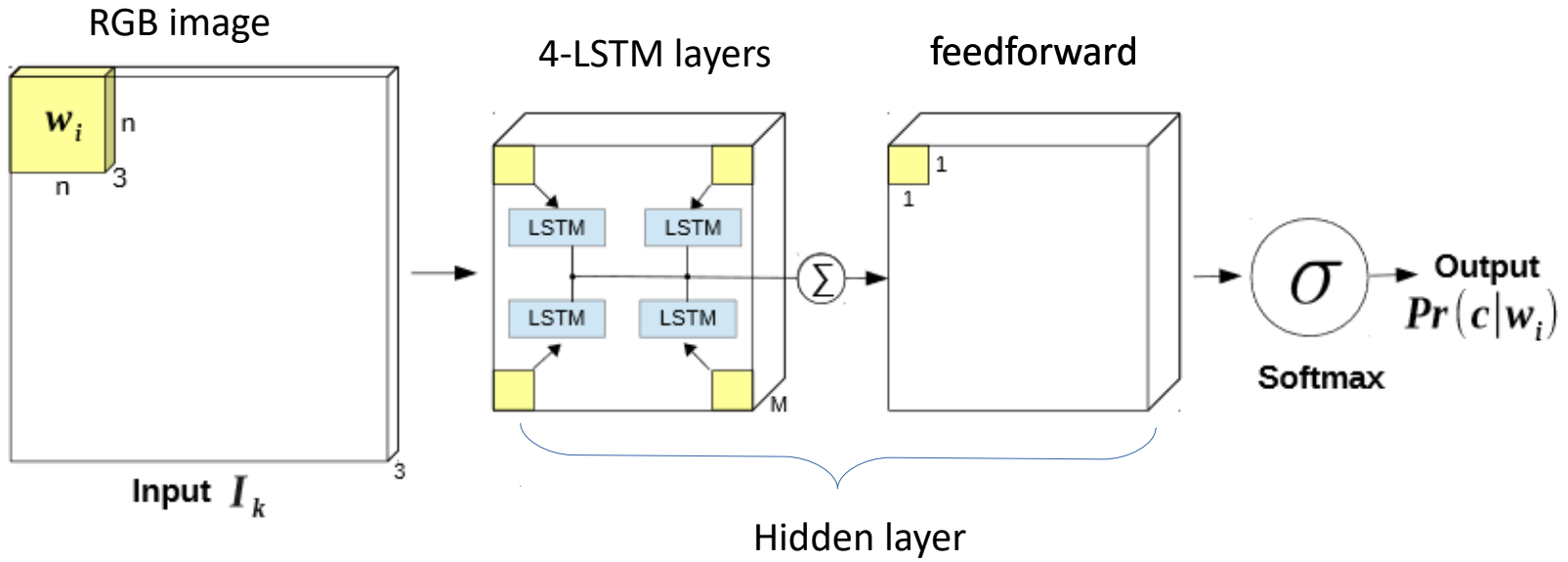
LSTM is one-dimensional:

Each cell contains a single recurrent connection -> activation controlled by a single forget gate.

Extension: n-dimensions by using n-recurrent connections with n-forget gates. Then, the 2D-LSTM is able to learn the neighbouring context information of every pixel

Multi-Dimensional Recurrent Neural Network

Architecture:

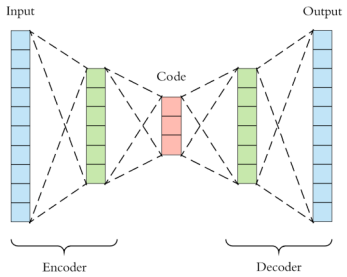


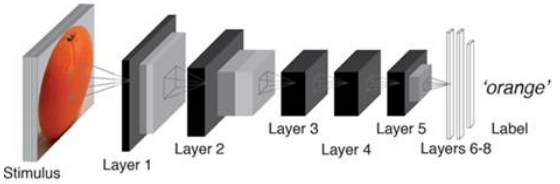
- An input image is divided into non-overlapping windows (nxn) pixels
- Each window with RGB channels is fed into four separate LSTM memory blocks.
- Each block is connected to its surrounding directions: left-top, left-bottom, right-top and right-bottom
- The feedforward layer sums all directions and squash it by the Hyperbolic tangent (tanh).
- The output is sent sent to the softmax layer

Image & source: Scene Labeling with LSTM Recurrent Neural Networks – Byeon 2015

Convolutional Autoencoder

Convolutional Autoencoders are designed to utilise the advantages of both autoencoders and CNN:



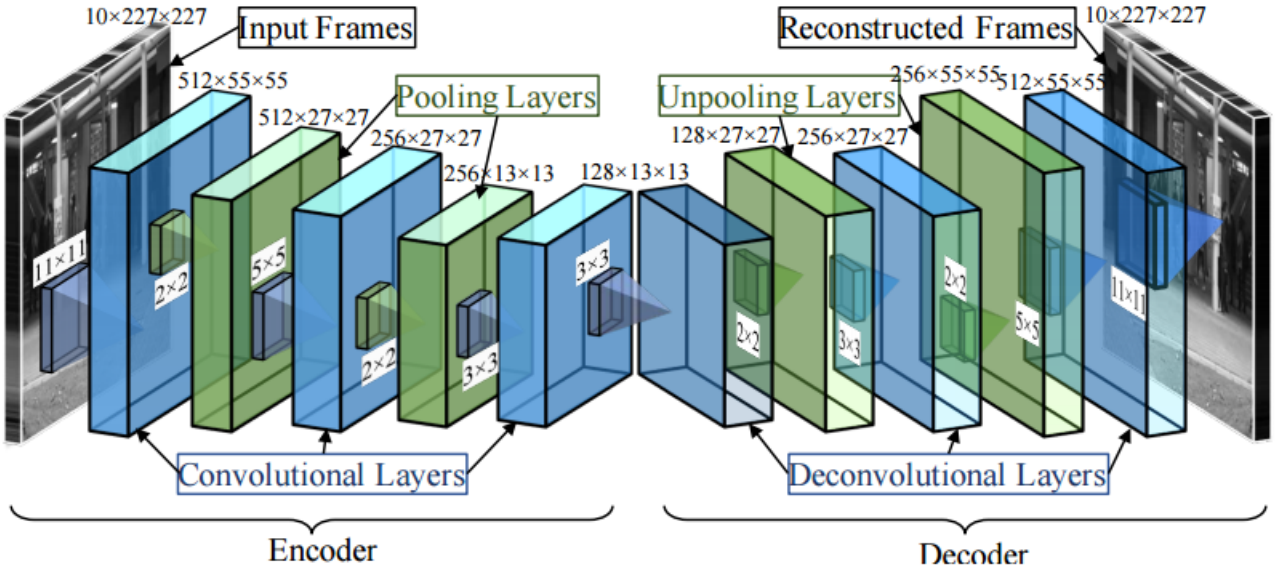


The basic idea is similar to a normal autoencoder, which learns hierarchical representations through reconstructing its input data, but CAE additionally utilises spatial information by integrating convolutions.

Advantages:

- Instead of manually engineer convolutional filters we let the model learn the optimal filters that minimize the reconstruction error.
- These filters can then be used in any other computer vision task.
- Learn good hierarchical representations of spatial data
- Can be well regularised by unsupervised training

Convolutional Autoencoder



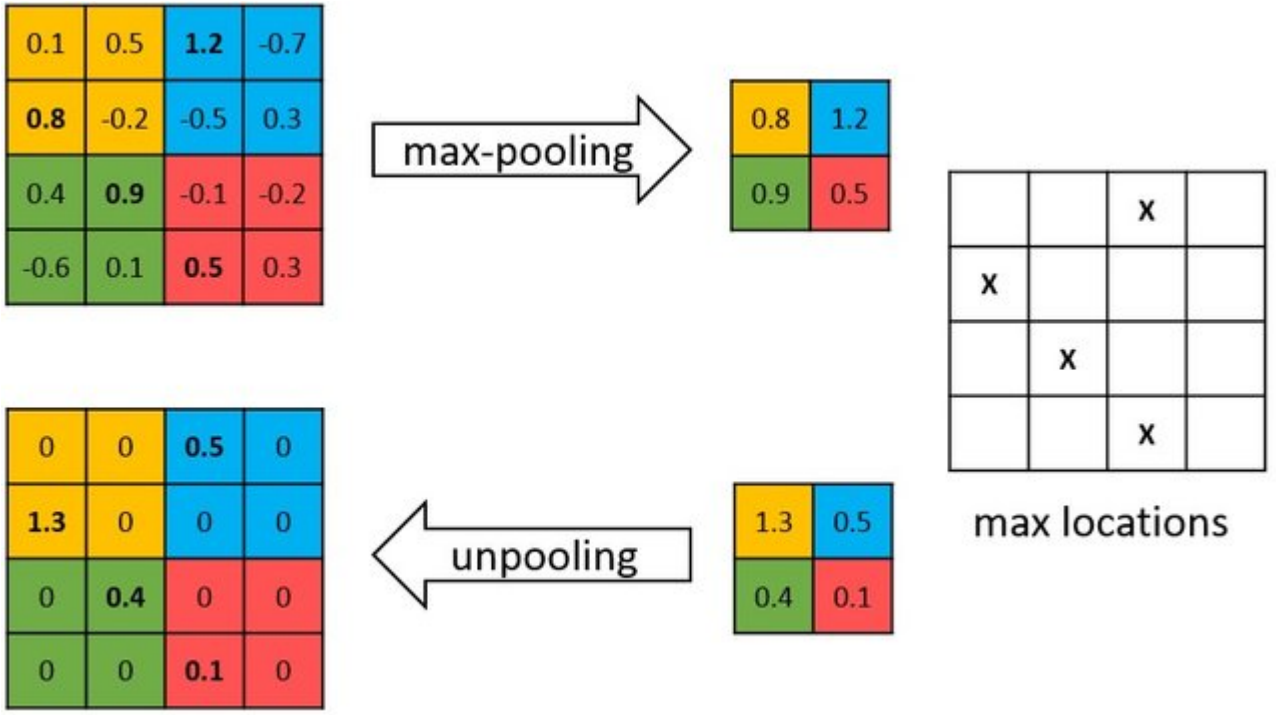
Basic structure of CAEs. Instead having only fully connected layers:

- **Encoder**: convolution layers and a pooling layers
- **Decoder**: deconvolution layers and unpooling layers

Plausible filters only emerge once a CAE is trained with a max-pooling layer.

Image: http://bioserver.cpgei.ct.utfpr.edu.br/disciplinas/eeica/apresenta%C3%A7%C3%B5es/Deep%20Learning%20-%20Aula%203%20-%20Autoencoders.pdf

Convolutional Autoencoder



Max-pooling layers result in loss of information, and so an unpooling layer should try to approximately restore the original values.

To do that, for each pooling layer, the max locations are stored. These locations are then used in the unpooling layer.

Convolutional Autoencoder

Training:

The CAE is trained in the same manner as a normal autoencoder (firstly done individually for each layer (same amount of epochs) and after that, the whole network is trained in a fine tuning using labelled data)

After training a CAE, the unpooling and deconvolution components are removed. Then, the convolution and pooling components are used to initialise a supervised CNN, by adding a fully connected layer followed by a classification layer.

In training the reconstruction error is minimised (normally by using stochastic gradient descent), extracting feature vectors from input data and recreate the data from the feature vectors.

Since deep layers only marginally improve the classification performance, it is crucial to have many more dimensions (feature maps) in the deeper layers than the shallow layers.