ARTIFICIAL INTELLIGENCE

LEI/3, LMA/3, MBE/1

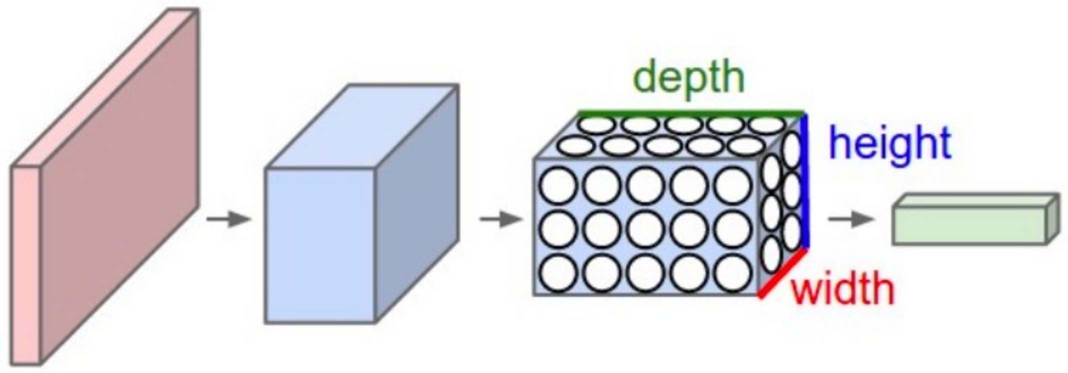
University of Beira Interior, Department of Informatics

Hugo Pedro Proença

hugomcp@di.ubi.pt, **2022/23**

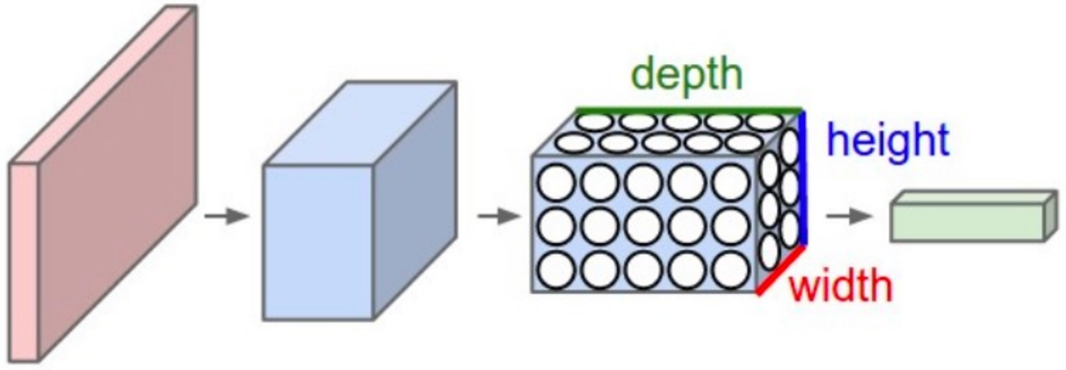
Convolutional Neural Networks (CNNs)

- CNNs are a type of Neural Networks that have been augmenting their popularity in most tasks related to Computer Vision
  - E.g., Image Segmentation, Classification.
- The property of **shift invariance** gives them the **biological inspiration** of the human visual system and keeps the number of weights relatively small, making learning a feasible task.
- In opposition to traditional Feed-forward nets, neurons in CNNs are arranged in **three dimensions**.



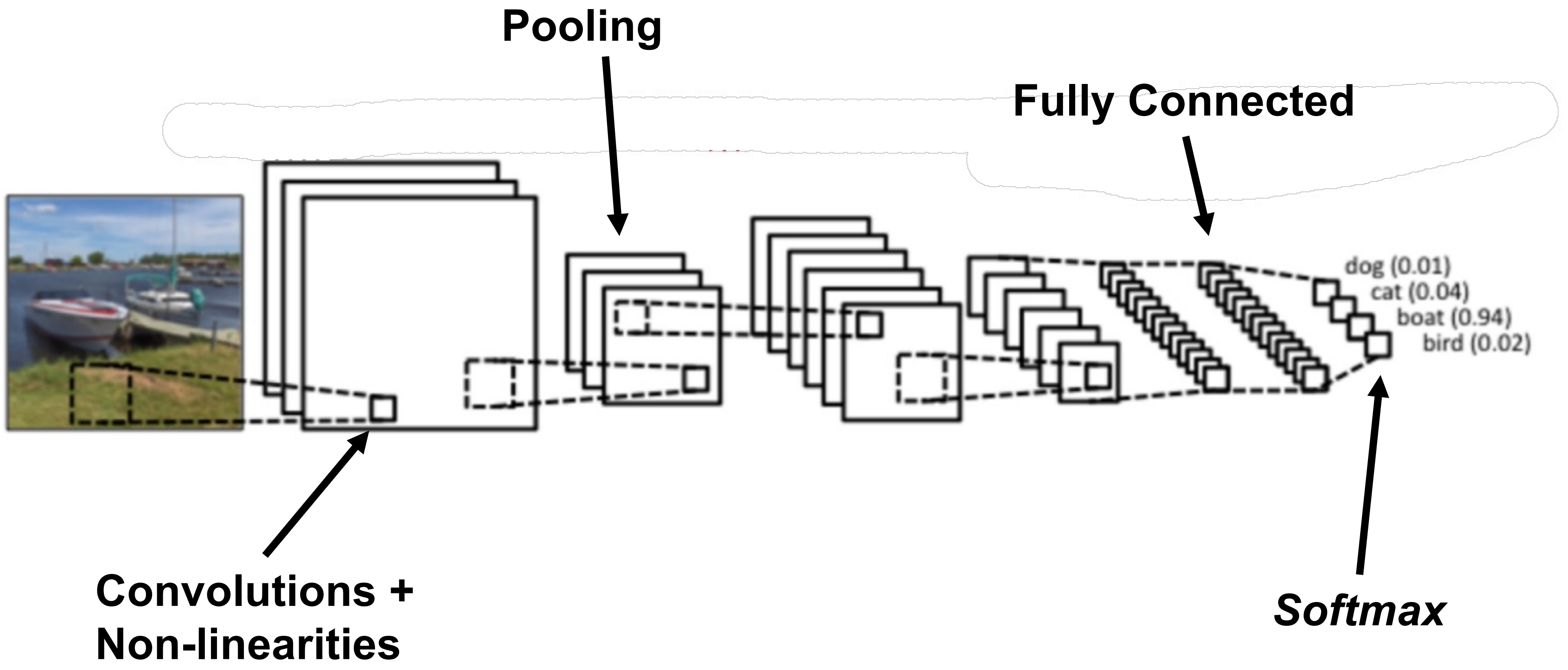
Convolutional Neural Networks (CNNs)

- Each layer of a CNN transforms a 3D input into a 3D output.
- This pioneering work in CNNs was due to Yann LeCun (LeNet5) after many previous successful iterations since 1988.
- Initially, the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits...
- The efficacy of CNNs in visual tasks is the main reason behind the popularity of deep learning. They are powering major advances in computer vision, with applications for robotics, security and medical diagnosis.



Convolutional Neural Networks (CNNs)

- The typical architecture of CNNs is as follows:



These operations are the basic building blocks of *most CNNs*, so understanding how these work is an important step to actually understand the functioning of these powerful models.

Convolutional Neural Networks (CNNs)

- **Convolution**
  - This block computes the convolution between an input map **x** with a bank of k multi-dimensional filters **f,** to obtain the results **y.**

$$\mathbf{x} \in \mathbb{R}^{H \times W \times D}, \quad \mathbf{f} \in \mathbb{R}^{H' \times W' \times D \times D''}, \quad \mathbf{y} \in \mathbb{R}^{H'' \times W'' \times D''}.$$

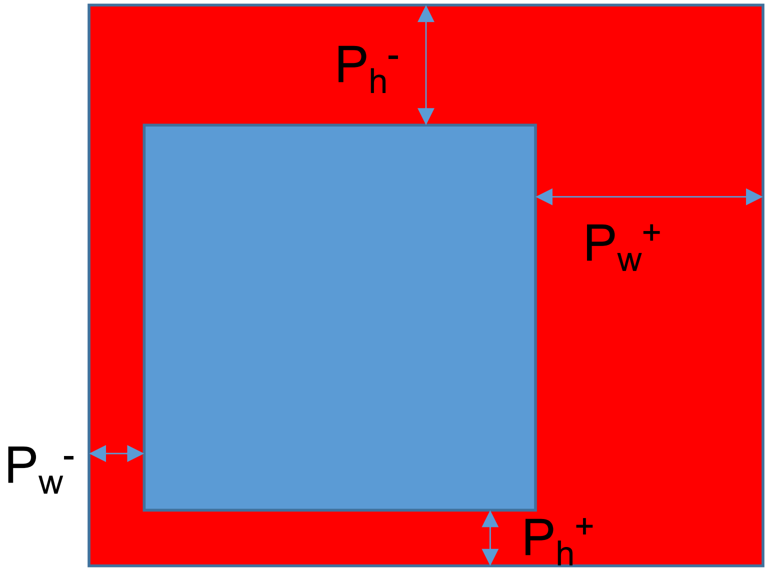
- Formally, the outputs **y** are given by:

$$y_{i''j''d''} = b_{d''} + \sum_{i'=1}^{H'} \sum_{j'=1}^{W'} \sum_{d'=1}^{D} f_{i'j'd} \times x_{i''+i'-1,j''+j'-1,d',d''}.$$

Convolutional Neural Networks (CNNs)

- Convolution (padding and stride)
  - Usually it is possible to specify top, bottom, left, right paddings (P_h^-, P_h^+, P_w^-, P_w^+) of the input array and subsampling strides (S_h,S_w) of the output array.

$$y_{i''j''d''} = b_{d''} + \sum_{i'=1}^{H'} \sum_{j'=1}^{W'} \sum_{d'=1}^{D} f_{i'j'd} \times x_{S_h(i''-1)+i'-P_h^-, S_w(j''-1)+j'-P_w^-, d', d''}.$$



The output size is given by:

$$H'' = 1 + \left\lfloor \frac{H - H' + P_h^- + P_h^+}{S_h} \right\rfloor.$$

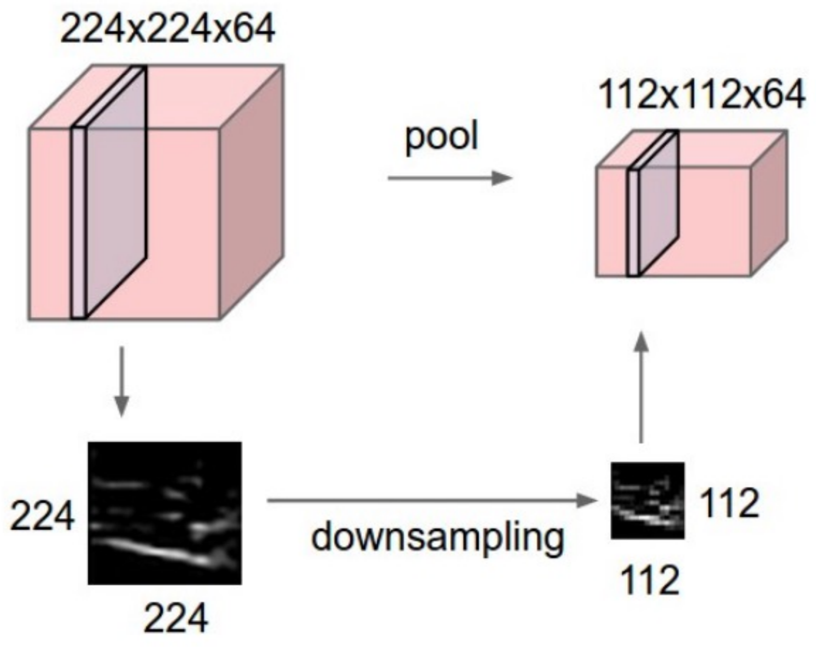
Convolutional Neural Networks (CNNs)

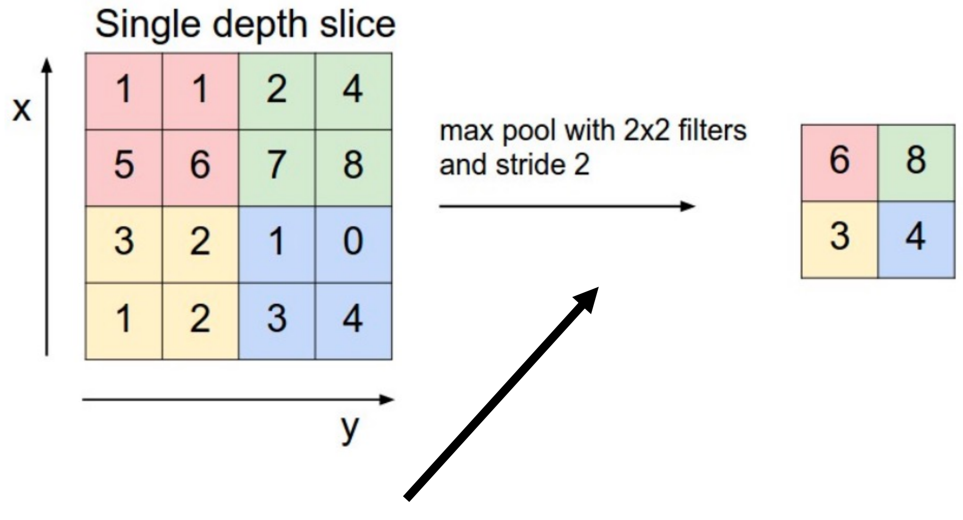
- **Spatial Pooling**
  - The typical blocks are the max and sum pooling, respectively computing the maximum and the summed response of each feature channel in a H’ x W’ patch.
- Pooling progressively reduces the spatial size of the input representation.
  - This reduces the number of parameters and, therefore, controls over fitting;
  - Also, it makes the network invariant to small transforms, distortions and translations in the input image (a small distortion in input will not change the output of pooling).

$$y_{i''j''d} = \max_{1\le i'\le H', 1\le j'\le W'} x_{i''+i'-1, j''+j'-1, d}. \qquad y_{i''j''d} = \frac{1}{W'H'} \sum_{1\le i'\le H', 1\le j'\le W'} x_{i''+i'-1, j''+j'-1, d}.$$

Convolutional Neural Networks (CNNs)

- **Pooling**
  - Note that Pooling down samples the input volume only spatially;
  - The input depth is equal to the output depth;
  - The pooling operation is often considered **deprecated**. To reduce the size of the representation, in is possible to use larger strides in the convolution layers.





Example: max() pooling

Convolutional Neural Networks (CNNs)

- **Non-Linearity**
  - There are two basic non-linear activation functions used in CNNS: “ReLU” (Rectified Linear Units) and “Sigmoid”.

$$y_{ijd} = \max\{0, x_{ijd}\}.$$

$$y_{ijd} = \sigma(x_{ijd}) = \frac{1}{1 + e^{-x_{ijd}}}.$$

  - As advantages with respect to each other, Sigmoid is consider not to blow up activation, while ReLU **does not vanishes the gradient**
    - In the case of Sigmoid, when the input grows to infinitely large, the derivative tends to 0.
  - However, in the case of ReLU, there is no mechanism to constrain the output of the neuron, as the input is often the output)

Convolutional Neural Networks (CNNs)

- **Fully Connected layers**
  - Neurons in a fully connected layer have full connections to all activations in the previous layer, as in a regular feed-forward network.
  - In practical terms, these neurons resemble pretty much the neurons in ”Convolution” layers.
    - The only difference between fully connected and Convolution layers is that the neurons in the former layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters.
    - However, the neurons in both layers still compute dot products, so their functional form is identical.
  - For example, an FC layer with K=4096 that is looking at some input volume of size 7×7×512 can be expressed as a Convolution layer with F=7 x 7 x 4096 (padding 0, stride 1).
  - In other words, we are setting the filter size to be exactly the size of the input volume;
  - Hence the output will simply be 1×1×4096.

Convolutional Neural Networks (CNNs)

- **Softmax**
  - Can be seen as the combination of an activation function (exponential) and a normalization operator.
  - It is usually applied as the transfer function of the last layer of the CNN, where the idea is to push up the maximum value of the responses to “1”, and all the other values to “0”.
  - In practice, it simulates the probability of the input corresponding to each category, represented by a neuron in the output layer.

$$y_{ijk} = \frac{e^{x_{ijk}}}{\sum_{t=1}^{D} e^{x_{ijt}}}.$$

Convolutional Neural Networks (CNNs)

- Most of the memory used by CNNs is used in the early Convolutional layers, whereas most of the parameters of the network are in the fully connected layers.
  - Example VGGNet:

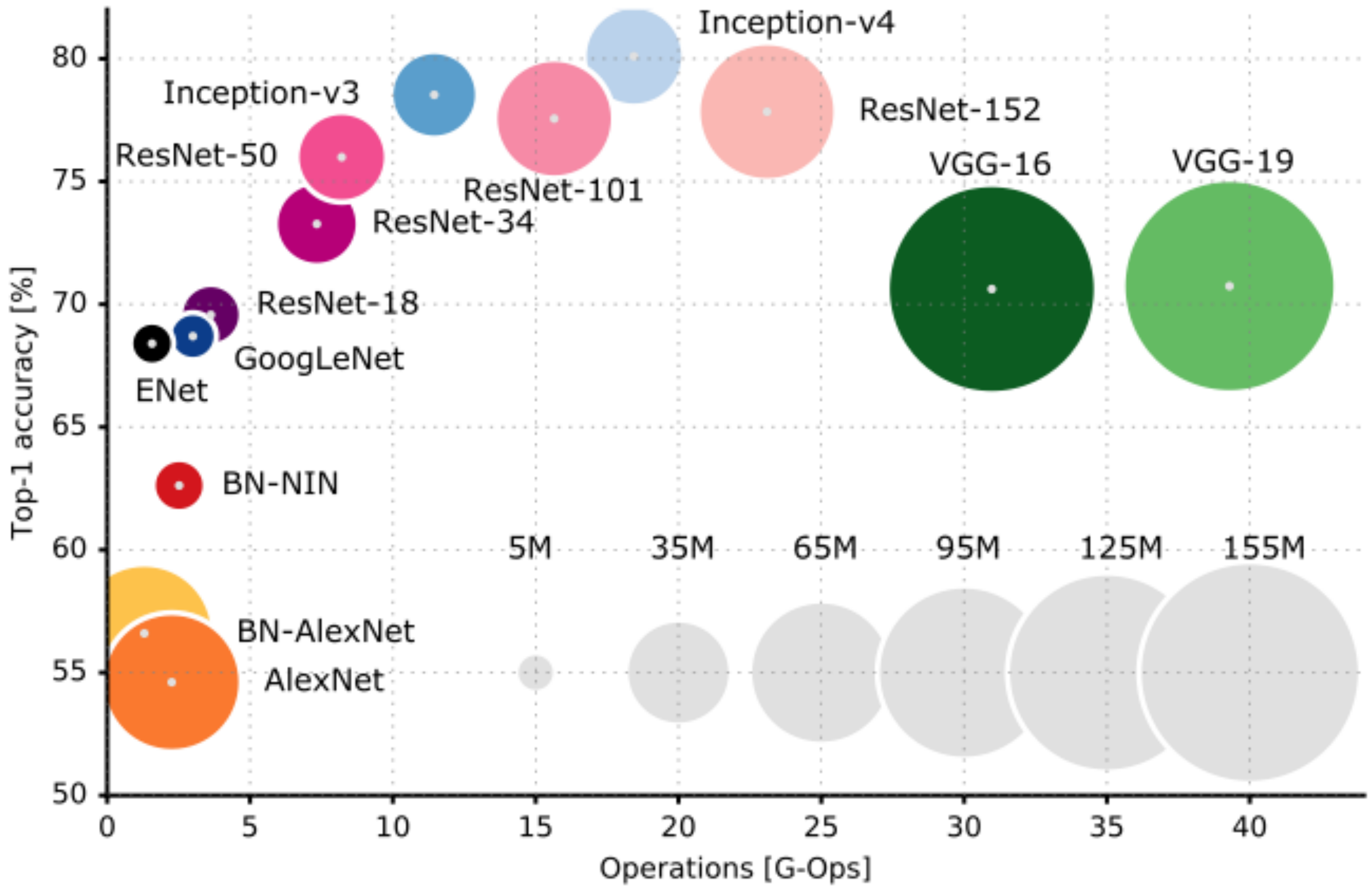
```
INPUT: [224x224x3] memory: 224*224*3=150K weights: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K weights: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M weights: (3*3*128)*128 = 147,456 POOL2: [56x56x128]
memory: 56*56*128=400K weights: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K weights: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K weights: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K weights: 0
FC: [1x1x4096] memory: 4096 weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 weights: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 weights: 4096*1000 = 4,096,000
```

Convolutional Neural Networks (CNNs)

- Example VGGNet
  - The total memory used is about 4 bytes * 24,000,000 = 93 MB
  - This is required only for the **forward step**
  - In practice, the backward step requires around the double memory;
  - The network has 138,000,000 parameters to be tuned by the back-propagation algorithm.
- It should be noted that the conventional paradigm of a linear list of layers has recently been challenged
  - Google’s Inception architectures and also Residual Networks from Microsoft Research Asia.
  - Both of these feature more intricate and different connectivity structures.

Convolutional Neural Networks (CNNs)

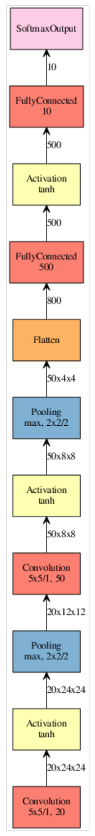
- Accuracy vs. Number of operations for a single forward step. Circumference radii corresponds to the number of parameters



Source: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

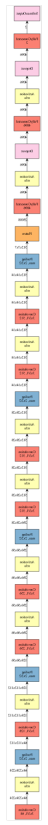
Convolutional Neural Networks (CNNs)

- An illustration of the most popular deep learning architectures is provided in http://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/

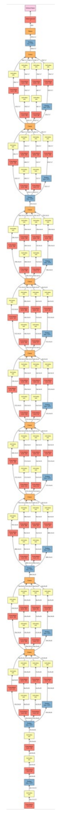


LeNet

AlexNet



VGG



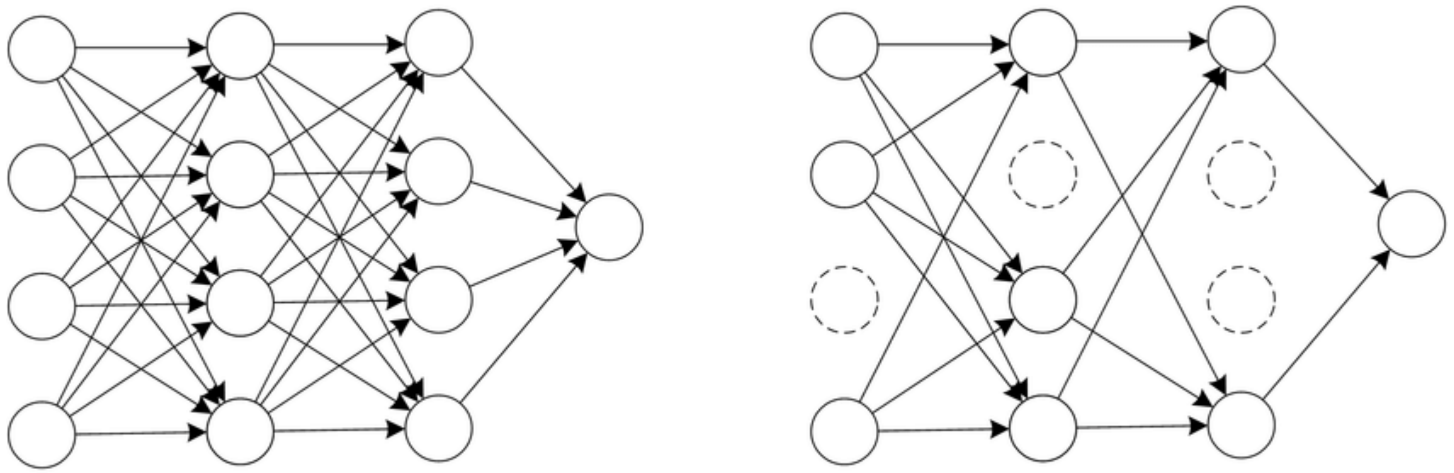
GoogLeNet

Inception

Resnet

CNNs: Other Layers

- **Dropout** Layers. This kinds of layers drops out units of a neural network during the learning phase.
  - Typically, a proportion (0, 1) of neurons is randomly chosen and not considered for a particular "**forward**/**backward**" pass.
  - Dropout is an approach to regularization in neural networks which helps to avoid interdependent learning amongst the neurons.
  - Recall that regularization is way to **prevent over-fitting**, by adding a penalty to the loss function.
  - It is applied exclusively to the fully connected layers of a CNN model.

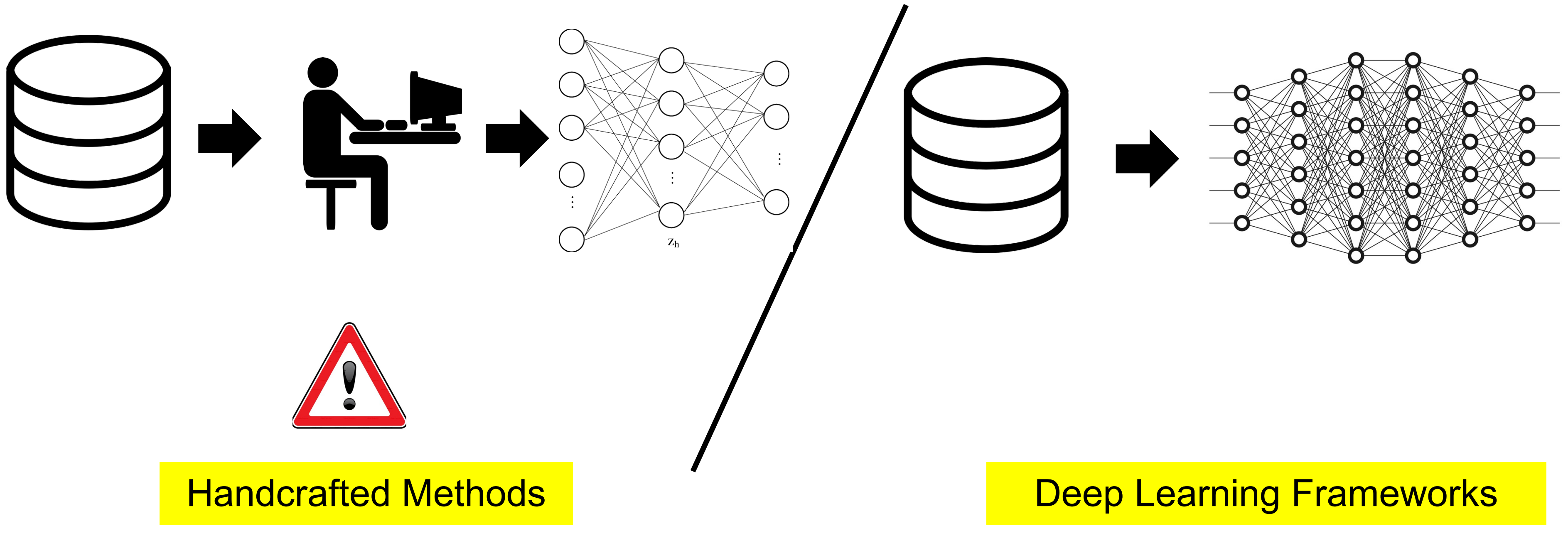


CNNs: Other Layers

- **Batch Normalization** Layers. To increase the stability of a neural network, this kind of layers normalizes the output of a previous layer by **subtracting** the batch mean and **dividing** by the batch standard deviation.
- This kind of layer can be added both after fully connected layers, but also after convolutional layers.
- Typically, using batch normalisation: 1) allows **higher learning rates;** 2) makes weights **easier to initialise**, helping to reduce the sensitivity to the initial starting weights.
- As the activations of one layer are the inputs of the next one, each layer in the neural network receives – at each iteration – diferente input distributions. This is problematic because it forces each layer to continuously adapt to its changing inputs.
- Using Batch Normalization allows the layer to learn on a more stable distribution of inputs (close to a standardized Gaussian distribution) and accelerates the training of the network.

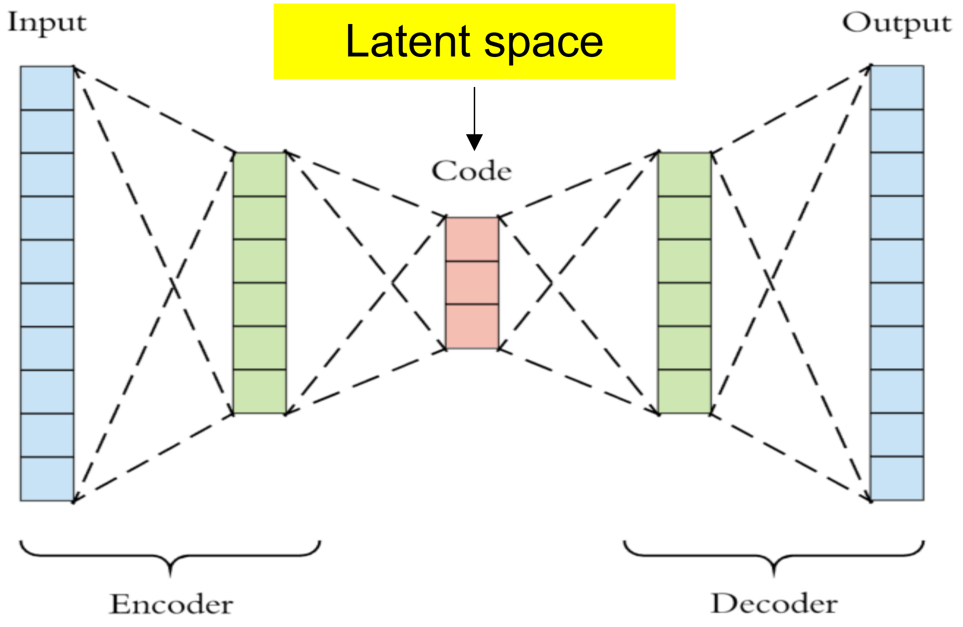
Deep Learning Architectures

- Deep Learning architectures are now “in the eye of the hurricane”, and have been advancing the state-of-the-art in multiple Machine Learning problems (if not all…)
- Recall that the main advantage of Deep Learning-based solutions with respect to handcrafted approaches, is that this new generation of models also carries out the feature extraction phase in an automatic way.



Auto-Encoders

- Autoencoders are a class of Neural Networks that try to reconstruct the input itself. They are unsupervised in nature.
- Typically, the general structure of an auto-encoder has two parts:
  - The **Encoder** sub-network, that receives the original data and obtains a “latent space representation”;
  - The **Decoder** sub-network, that receives the latent code and attempts to reproduce the original data.

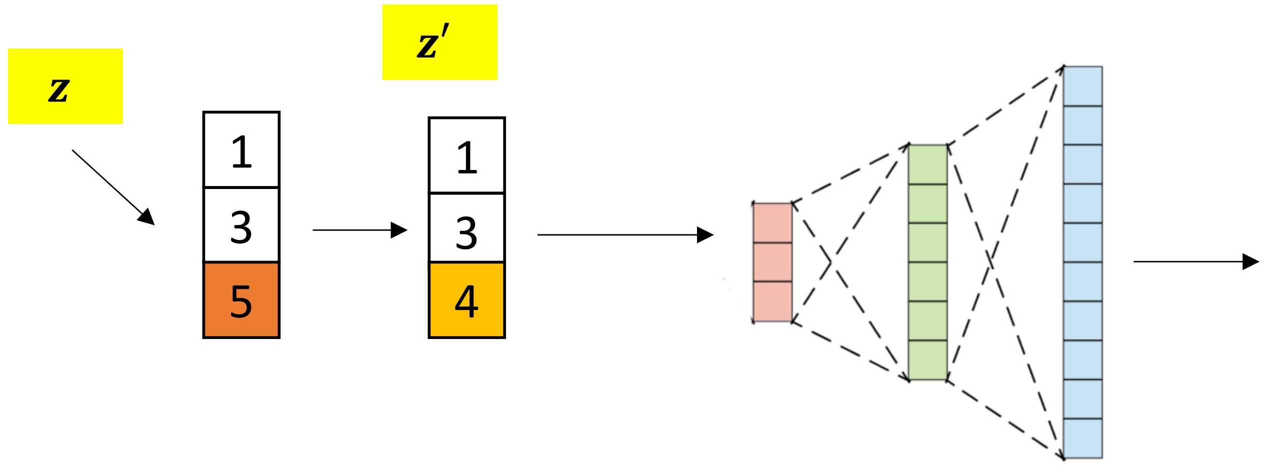


Auto-Encoders

- The first obvious application of auto-encoders is "**Data Storage and Transmission**"
  - Starting from a high-volume amount of information (size m), the latent code $\boldsymbol{z} \in \mathbb{R}^n$ is able to reconstruct the original data only with minor differences;
  - Obviously, n << m
- A second obvious application of using auto-encoders is to obtain a **compact feature representations** that can be used by Machine Learning models, for classification, regression or clustering purposes.
  - For such, it is assumed that a similarity between $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ (e.g., in terms of Euclidean/Co-sine distances) corresponds directly to the similarity of the corresponding original data

Auto-Encoders

- Subsequently, another ingenious application for auto-encoders was to "**Generate Data**"
  - There is a "**Generative**" paradigm of Machine Learning/Pattern Recognition models that attempts to model the phenomena to be handled
    - i.e., obtain an approximation of **p(C,I),** with "I" representing the input data and C the corresponding desired response.
  - This is in opposition to the "**Discriminative**" family of methods, which typically attempt to infer **p(C|I)**
- The idea in auto-encoders was to change some components in the latent code, to perceive the corresponding changes in the reconstructed data.

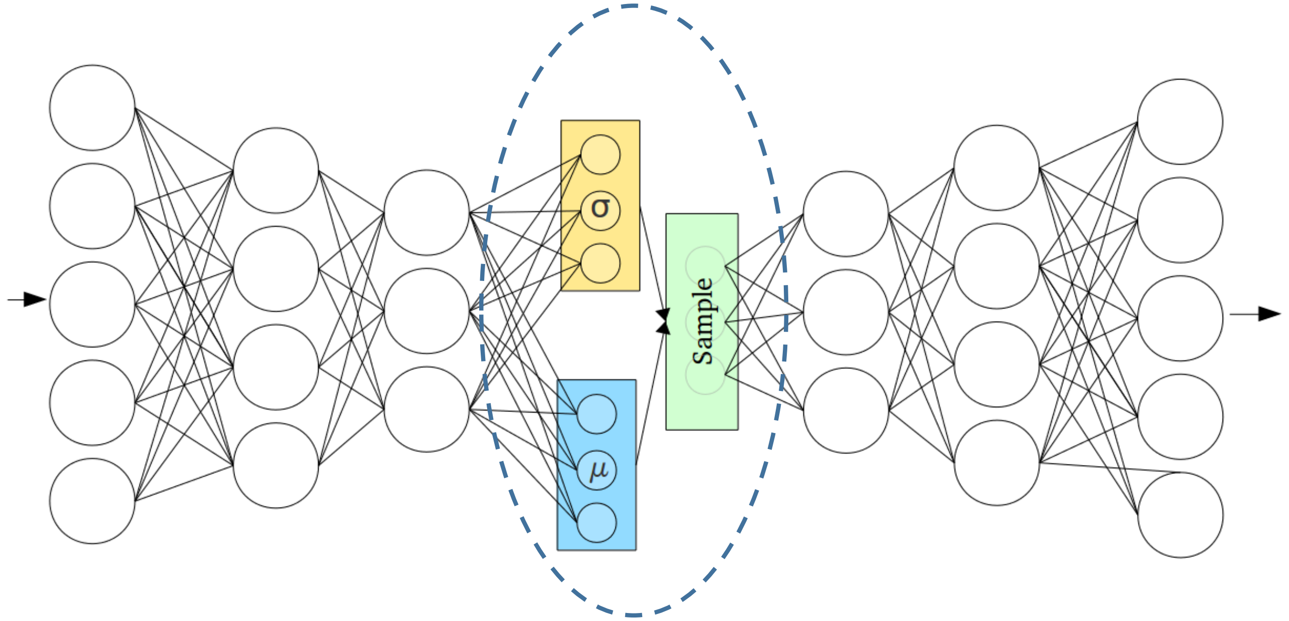


How do/much the objects change?

...(unfortunatelly, in a chaotic way)

Variational Auto-Encoders

- This kind of models have arisen upon the difficulties in controlling the appearance/features of the reconstructed data .
  - Standard autoencoders can obtain compact representations $\boldsymbol{z}$ and reconstruct their inputs well.
  - However, the main problem, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, **may not be continuous**, or **allow easy interpolation**.
- The key novelty in variational auto-encoders is a layer that explicitly encodes **means** and standard deviations of the latent representations, which are sampled to generate a reconstructed sample.

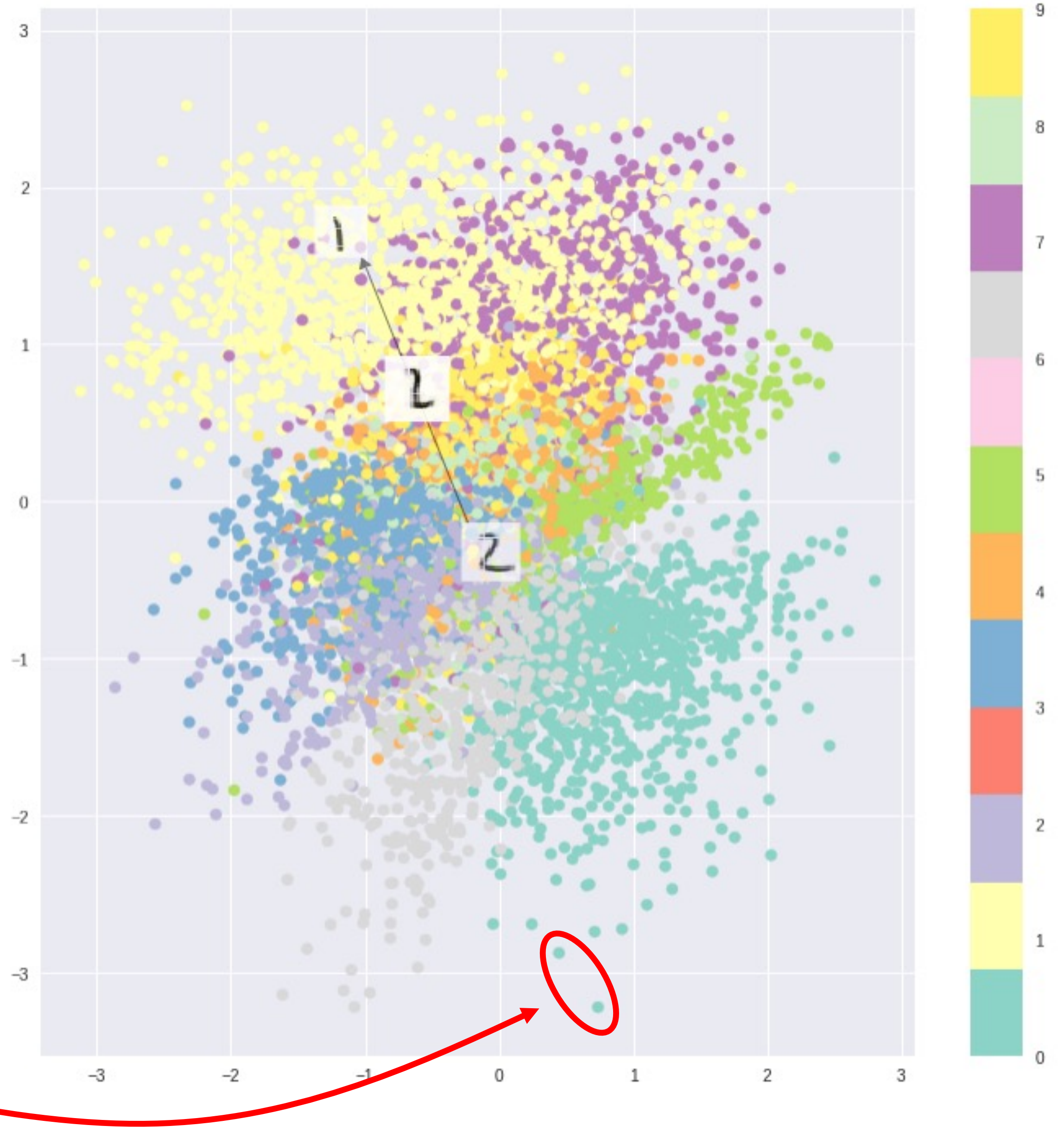


$\mu = [0.5, 1.2]$
$\sigma = [0.1, 0.2]$
Sample two values from:
$[X_1 \sim N(0.5, 0.1),\ X_2 \sim N(1.2, 0.2)]$

Variational Auto-Encoders

- The $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ values allow a continuity in the latent space, that can be used to generate synthetic elements according to some **pre-defined properties** and appearance features

In practice terms, it is assured that neighbor elements in the latent space correspond to similar instances in the image space

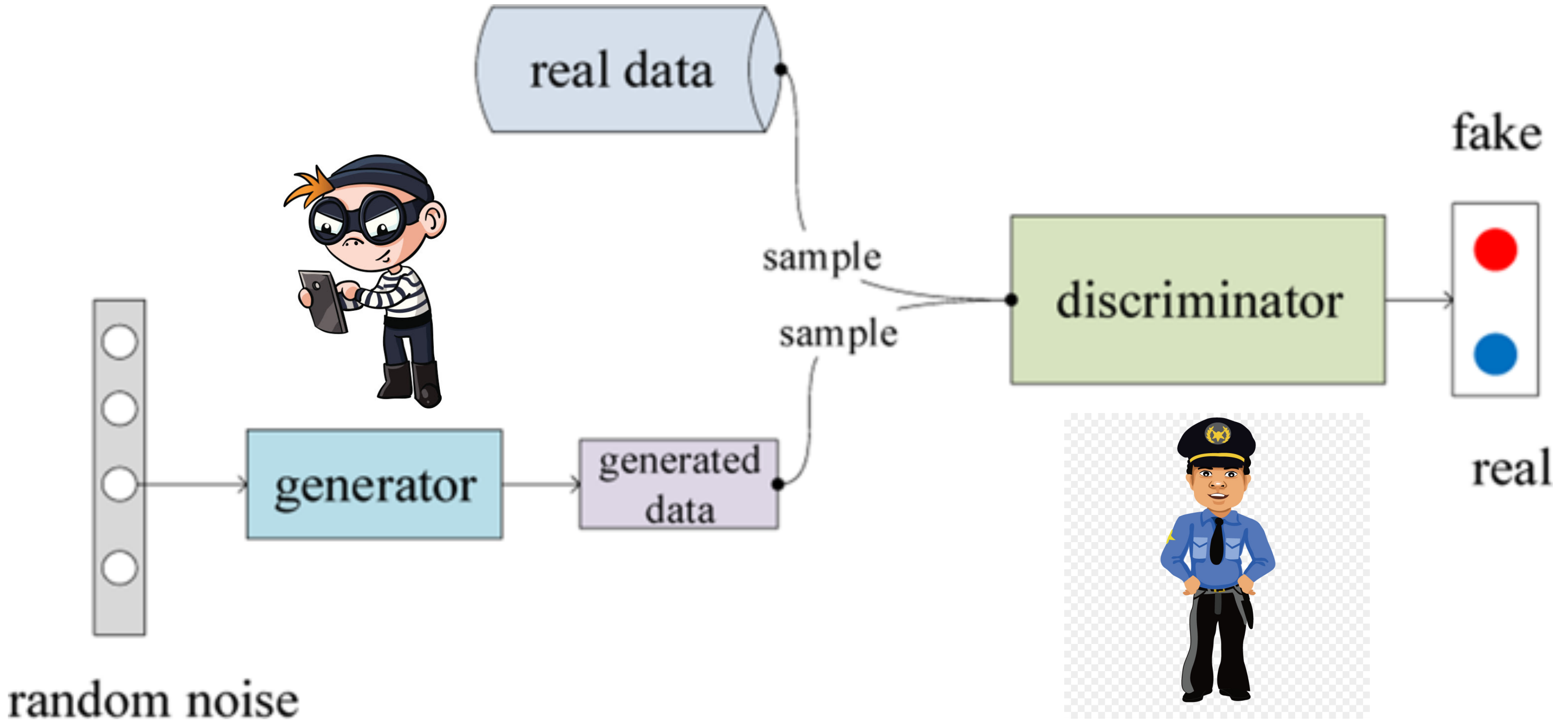


Adversarial Learning

- Facebook's AI research director **Yann LeCun** called adversarial training “*the most interesting idea in the last 10 years in Machine Learning*”.
- **Generative Adversarial Networks** (GANs) are architectures that use two neural networks, competing one against the other (thus the “adversarial”) in order to generate new, synthetic instances of data that can pass for real data.
  - GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal, including Yoshua Bengio, in 2014.
- GANs’ potential for both good and evil is huge, because they learn to mimic **any distribution of data**.
- GANs can be taught to create worlds eerily like our own in almost any domain: images, music, speech, prose…

GANs

- The basic idea in GANs is to have one network (**Generator**) trying to fool the other one, while the later (**Discriminator**) tries not to be fooled.
- This can be seen as a **Police Officer**←→ **Thief** game that, according to **Nash Game Theory**, typically converges into an equilibrium state.



GAN

- The **Discriminator** network is a typical binary classification CNN, that learns to distinguish between fake and real data.
- The **Generator** network receives one latent code (randomly generated, i.e., white noise) and produces one instance.
- The overall cost function is given by a two-player min-max game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]$$

- That can be decomposed into:

Discriminator

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{“recognize genuine”}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{\text{“recognize fakes”}}$$

Generator

$$\min_G V(G) = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}_{\text{“fool” } D}$$

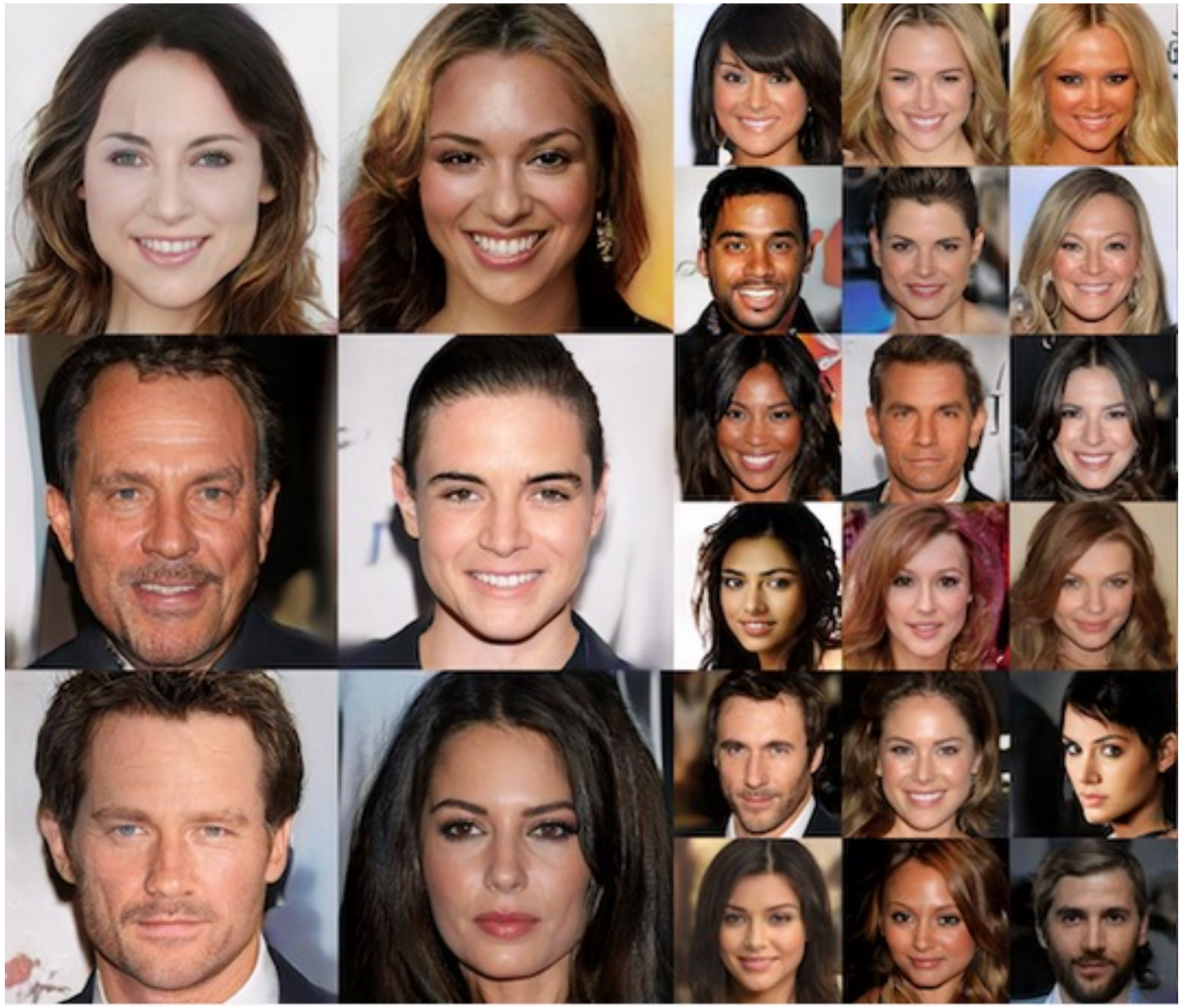
GAN

- GANs are trained in an iterative way:

1. Generate a set of Fake data **F**
2. Train the Discriminator (with Real data **R** **(labelled 0)** and Fake Data **F** **(labelled 1)**) //Learns to distinguish R from F
3. Set Discriminator.trainable =FALSE
4. Train the GAN (with Fake Data F (**labelled 0**)) //Learns to fool D
5. Move to Step 1.

GANs Applications

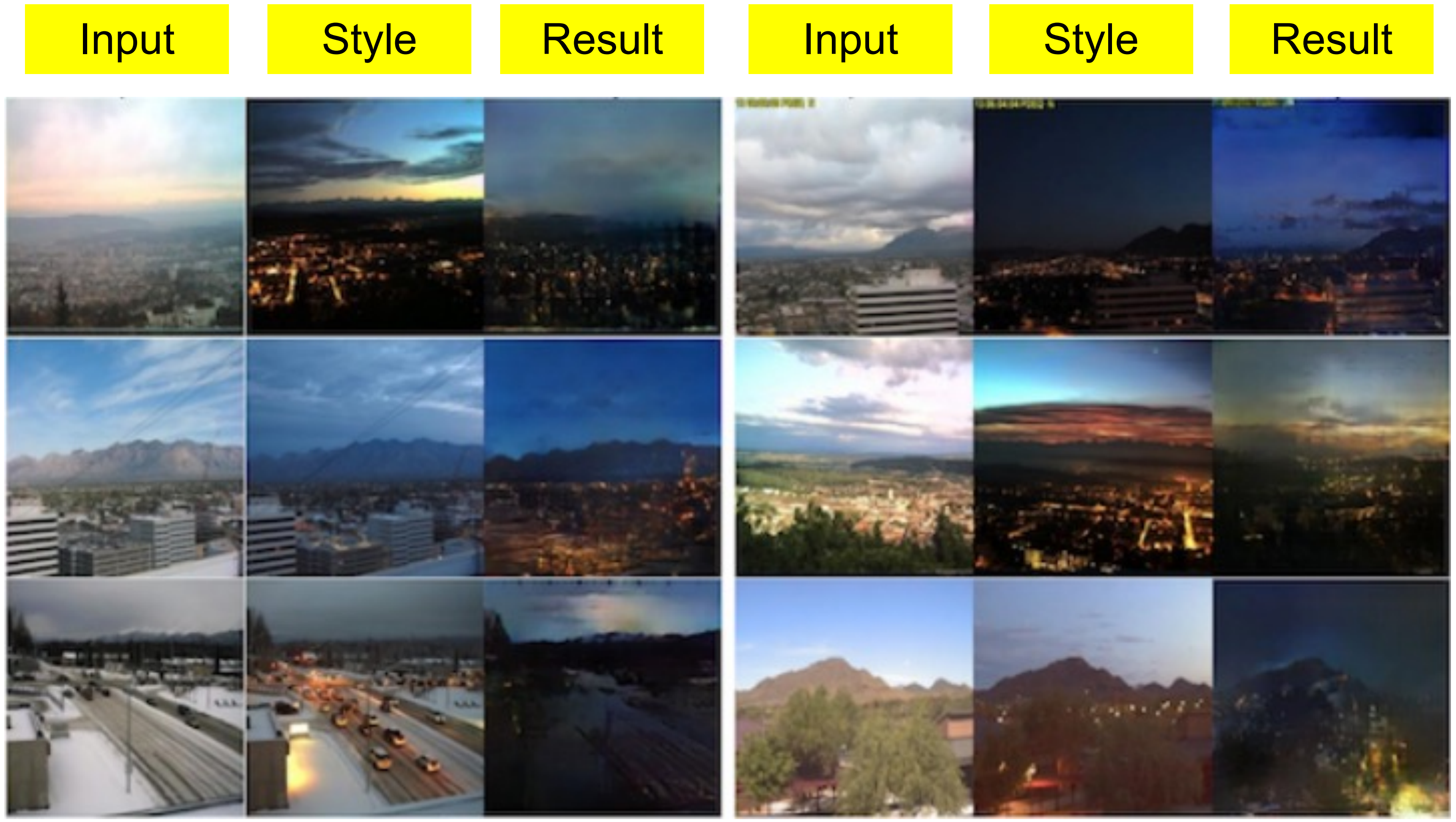
- E.g., plausible realistic photographs of human faces:



These persons don't exist!!

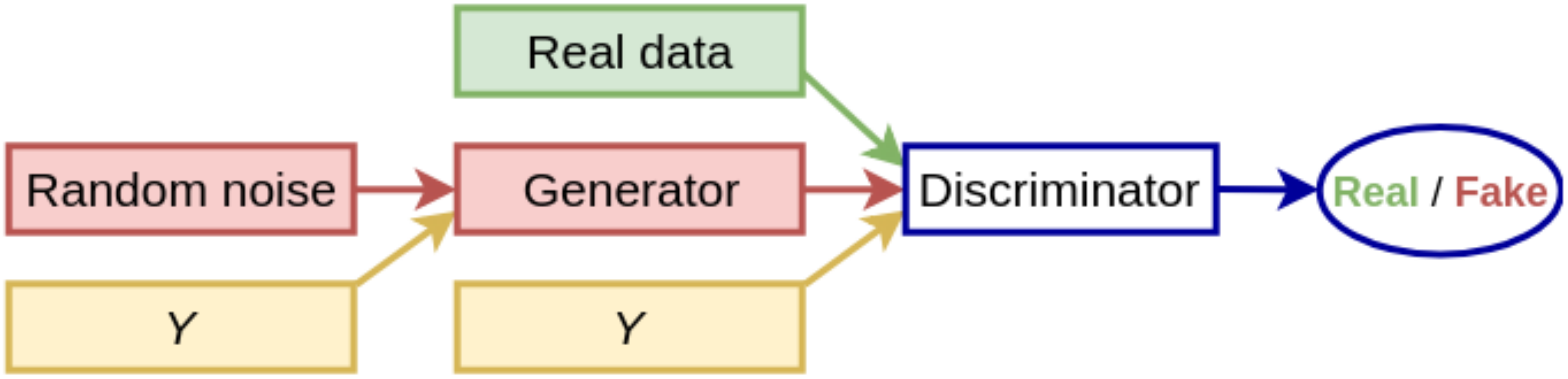
GANs Applications

- Image to Image Translation:



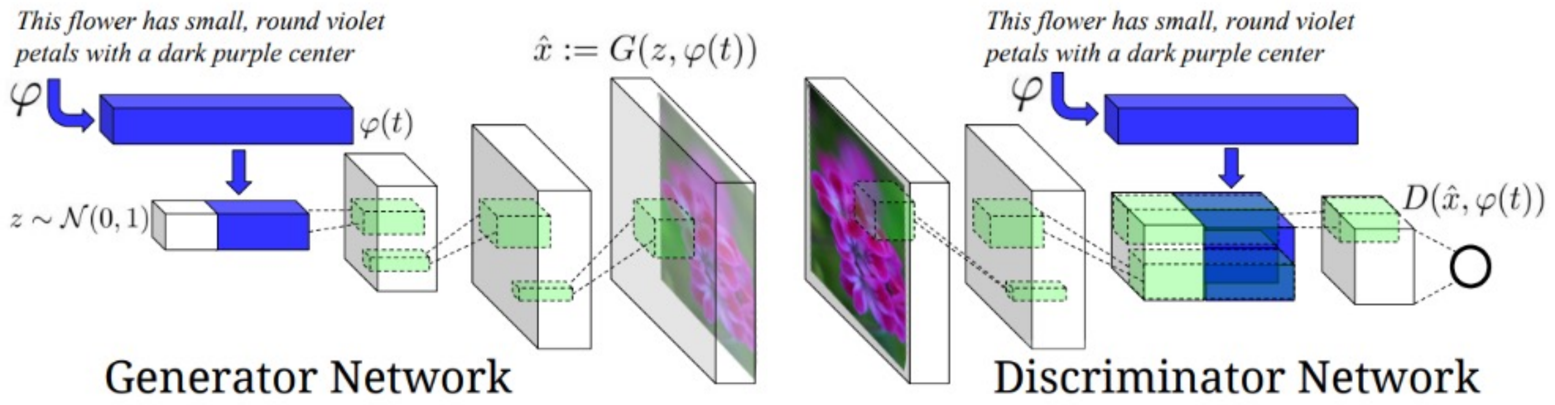
Conditional GANs

- Despite the remarkable effectiveness of GANs in generating synthetic (artificial) instances of one specific phenomenon, they provide a limited control over the specific features of the output.
  - Recall that the input is a random noise vector.
- Class-Conditional GANs (**cGANs**) introduce the label information to the learning architecture, enabling to produce instances of a specific class.
  - The Discriminator reports “1” only for genuine images with correct labels, and “0” for all other cases (genuine images with bad labels, and fake images with any label).



Conditional GANs (Applications)

- "*Text-To-Image Synthesis*": This is the problem of asking to a network, to generate images with specific features:


This flower has small, round violet petals with a dark purple center
$\hat{x} := G(z, \varphi(t))$
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network

- "Style Transfer": Transferring style between different kinds of objects: