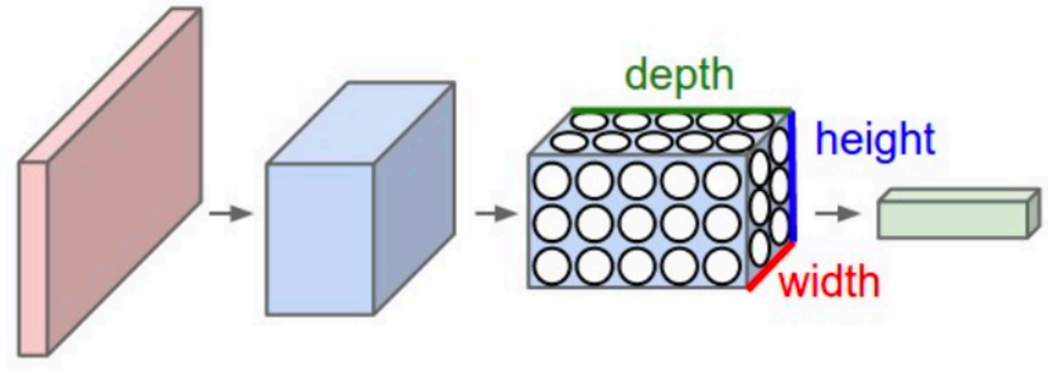
MACHINE LEARNING
MEI/1

University of Beira Interior, Department of Informatics

Hugo Pedro Proença, hugomcp@di.ubi.pt, 2019/2020

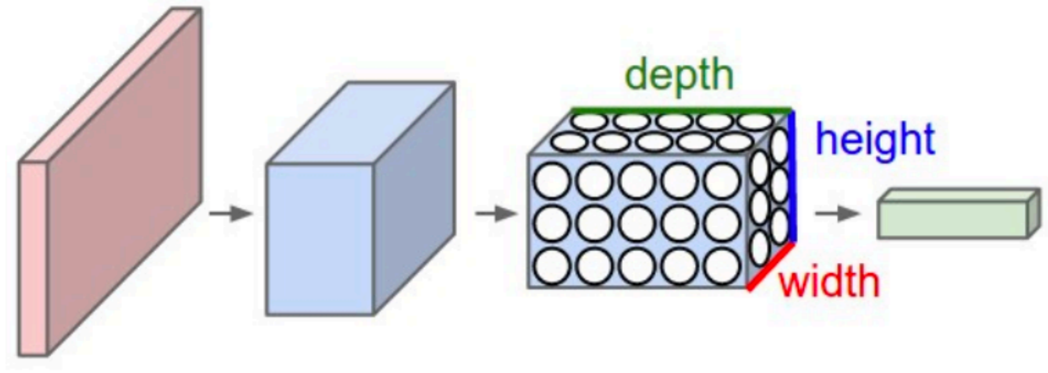
Convolutional Neural Networks (CNNs)

- 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**.



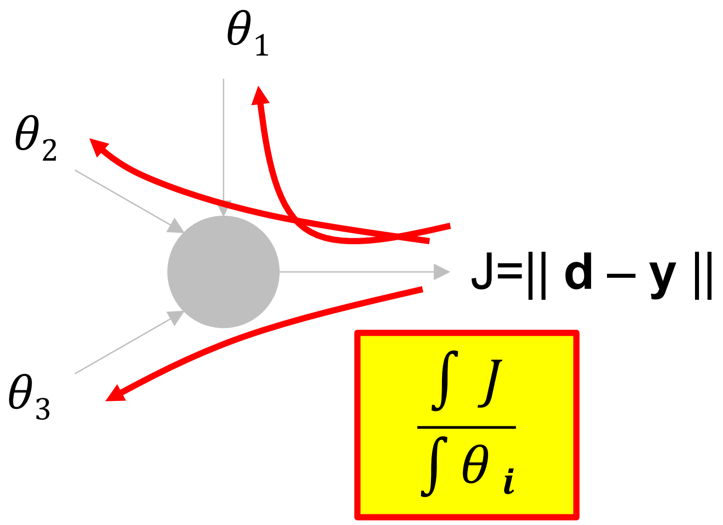
Convolutional Neural Networks (CNNs)

- 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.

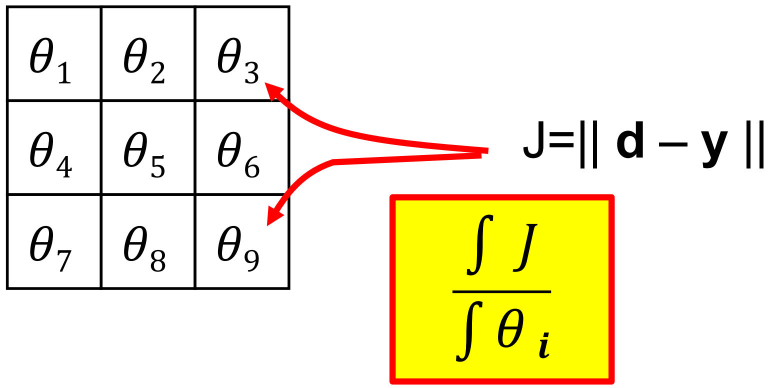


Convolutional Neural Networks (CNNs)

- The breakthrough advance of **Deep Learning** solutions is to perform in a "single-shot" the feature encoding and classification/regression (decision) phases.
- But, how exactly are such kind of frameworks extracting the ideal feature sets, for a specific problem?



Each **red arrow** adjusts the values of one network parameter, depending of its importance in the observed error (during backpropagation)

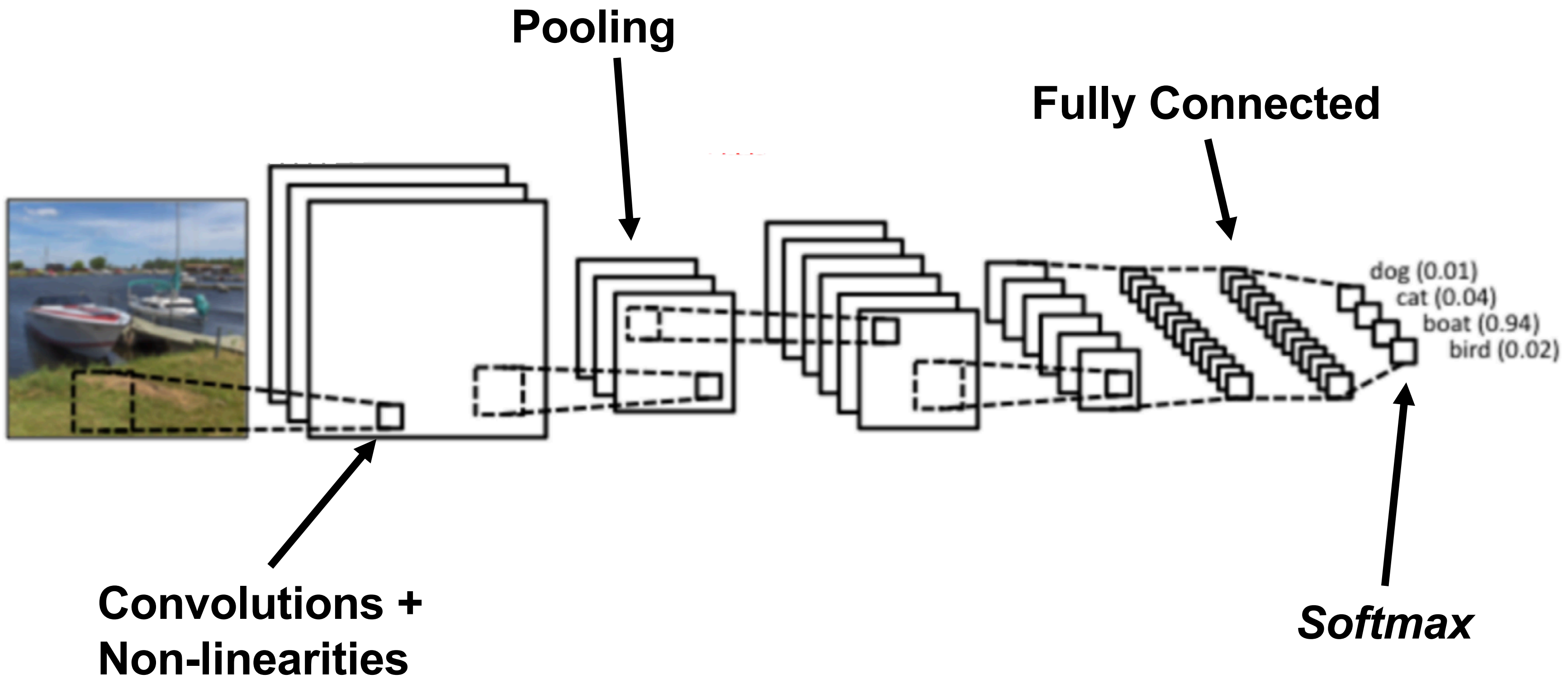


θ_1 θ_2 θ_3
θ_4 θ_5 θ_6
θ_7 θ_8 θ_9
J=|| **d** – **y** ||
$\frac{\int J}{\int \theta_i}$

Each **red arrow** adjusts the values of one component of a convolution kernel, depending of its importance in the observed error (during backpropagation)

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.**

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

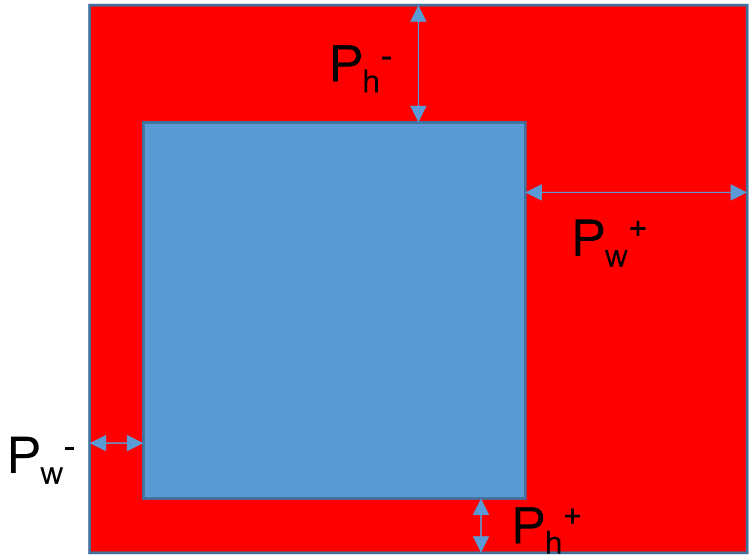
- Formally, the outputs **y** are given by:

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

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:

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

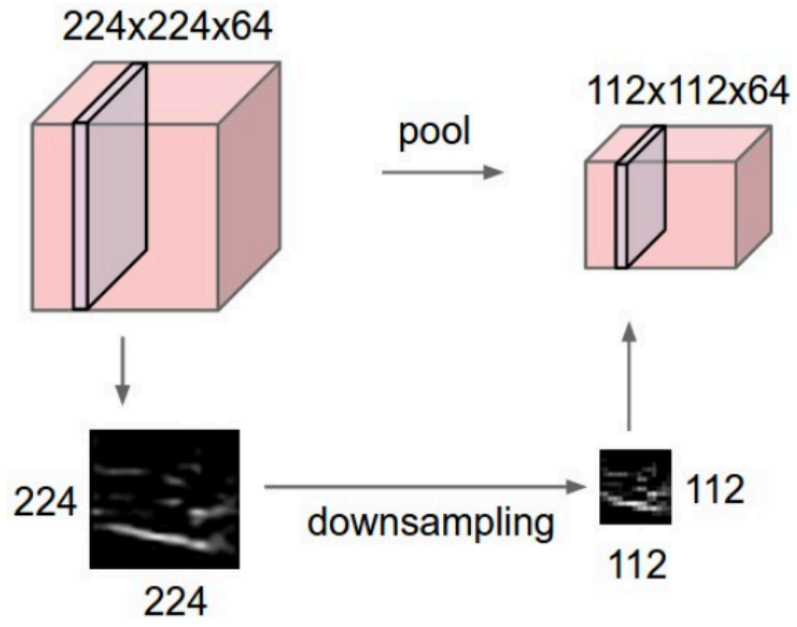
Convolutional Neural Networks (CNNs)

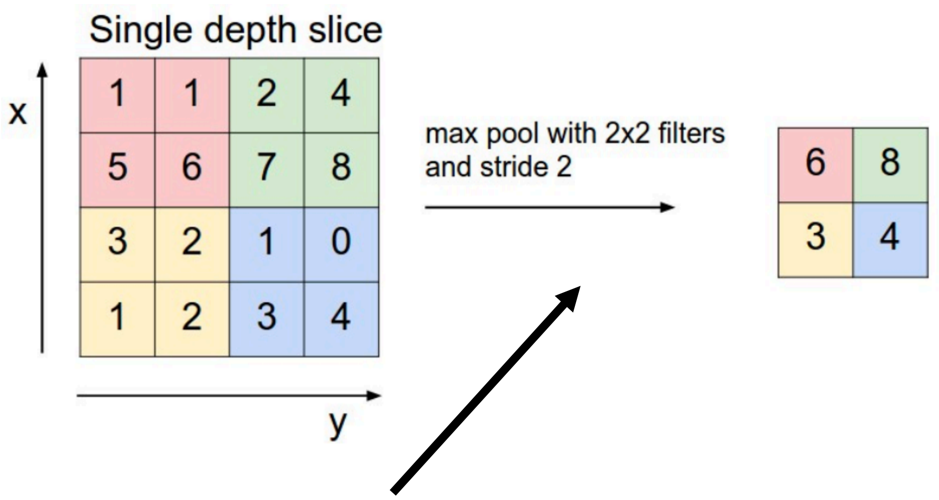
- **Spatial Pooling**
  - The typical blocks are the max and sum pooling, respectively computing the maximum and the summed response of each feature channel in a H' x W' patch.
- Pooling progressively reduces the spatial size of the input representation.
  - This reduces the number of parameters and, 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).

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

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:

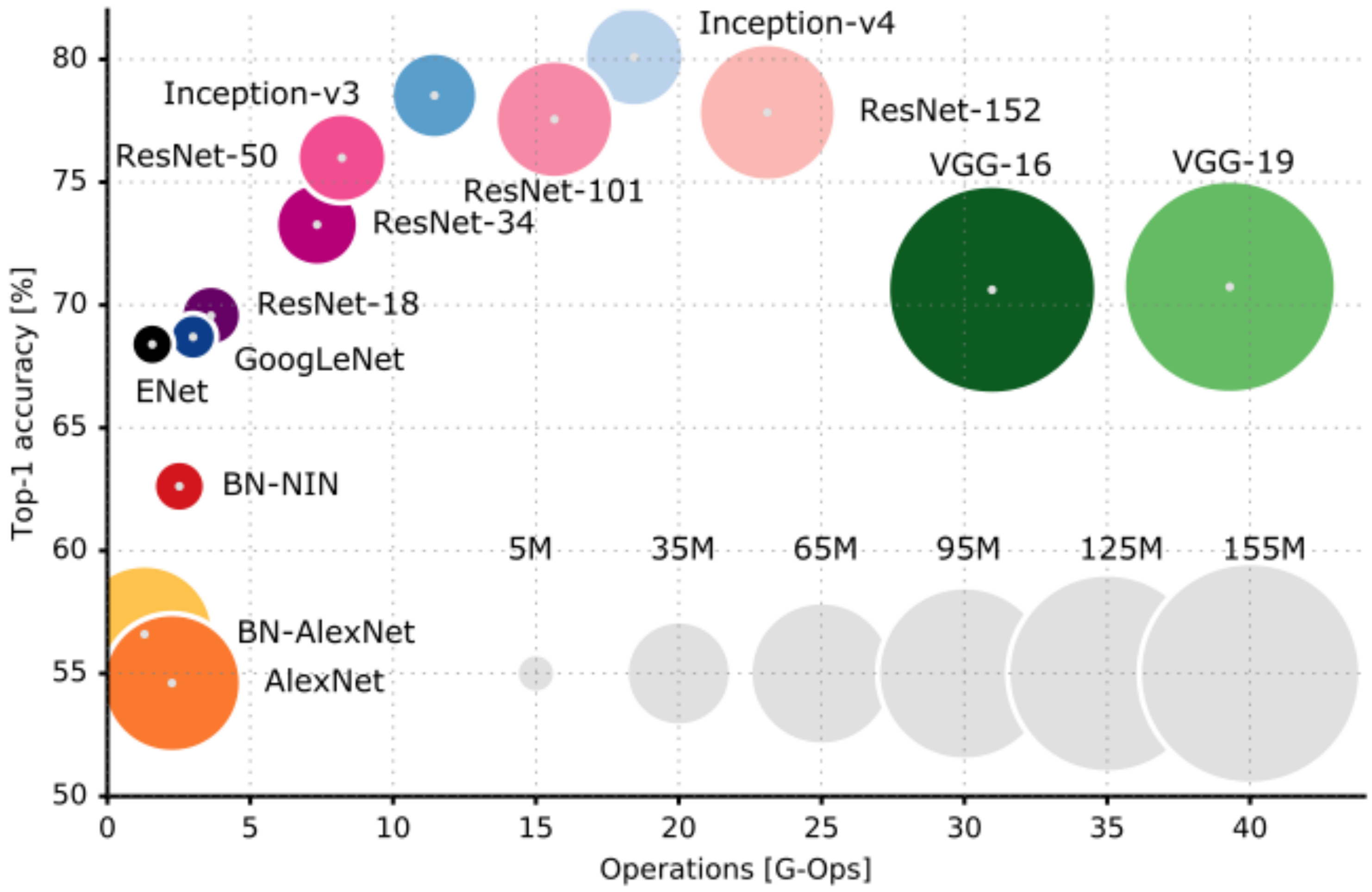
```
INPUT: [224x224x3] memory: 224*224*3=150K weights: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K weights: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M 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)

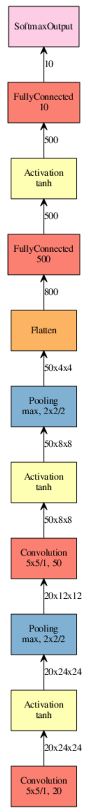
- Accuracy vs. Number of operations for a single forward step. Circumference radii corresponds to the number of parameters



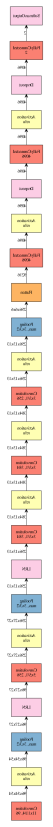
Source: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

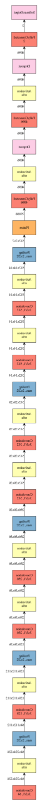
Convolutional Neural Networks (CNNs)

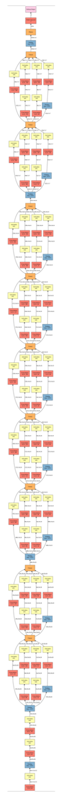
- An illustration of the most popular deep learning architectures is provided in http://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/

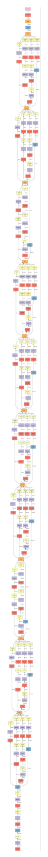


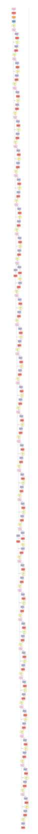
LeNet


AlexNet


VGG


GoogLeNet


Inception


Resnet