4. Deep learning with ANN

INTRODUCTION

In the earlier sections, we focused our attention on supervised learning applied to the training of multi-layer FF artificial neural networks. The training, involving the learning of the network weights and thresholds/biases, enables FFNN to classify patterns, map variables across heterogeneous spaces, compute continuous models of arbitrary functions, just to mention some main prototypical abilities. However, in spite of its significant potentiality, the extent of the supervised learning to practical applications has been acknowledged inadequate, because of two main reasons, namely the extraction of good features from the data and the intrinsic weakness of training algorithms as the network complexity increase (vanishing gradient issue in the backpropagation. In the first case, the reconstruction of a function with high accuracy would requires sampling the input domain as much as possible, especially in the regions featuring high information content. Or in contrast, one should remove useless data to highlight only the relevant information. In visual recognition tasks, one should envisage some pre-processing to extract relevant clues such as intensity-varying regions as the boundaries among image areas. Such operations can be labor-intensive and cannot scale well in general. Further, it is unclear what the optimal features for a predefined task are and usually they can vary a lot throughout tasks. When using ANN, one would expect that the network is able by itself to discover the best representative features automatically. The ANN framework, which would like to emulate our vision system, should be able to directly deal with coarse data, as they are measured, without any manual processing. In vision recognition for example, it should be completely autonomous in processing the dense retinal image without any external pre-processing, with all the needed image sub-processing implemented aboard, alike the neural processing pathway from the retina to the visual cortex. For the second issue, supervised training becomes challenging as the complexity of the network increases. Modeling non-linear and multi-factorial systems requires many hidden layers, which make the backpropagation algorithm less performant, affecting the overall learning ability. As we have seen, the decrease of the training performance is due to the progressive reduction of the error, back propagated throughout the layers, which affects the weight learning in the earlier layers. As we described, this issue is traditionally termed “vanishing gradient problem in the backpropagation”.
From these premises, we present two classes of ANN, namely the convolutional and the autoencoder neural networks, which can address differently the same problem of data synthesis. Both convolutional and autoencoder networks are feedforward networks aiming at learning a compressed, distributed representation (encoding) of an input dataset (usually an image but in principle can be applied to any generic input pattern). Such networks do not learn a “mapping” between the training data and its labels (as in the traditional supervised learning), but attempt to learn instead the internal structure of the data itself. The neural structure forces the network to learn only the most significant features achieving a dimensionality reduction with respect the size of the input. Such networks have a strong connection with the biological approach to sensorial processing. The visual recognition for example is a typical data processing task encompassing distributed and progressively compact representations. Each compact representation is the results of an unsupervised learning procedure. Cascades of progressively compact (non-linear) and distributed representations are the fundamental core of the deep learning paradigm.

VISUAL RECOGNITION AS AN ACT OF DEEP LEARNING

Recognizing a simple item (e.g. an alphanumeric character) from our visual experience (e.g. while reading a book) appears to be a simple task from the human point of view but it is a very challenging computational task when performed by a computer system. We learnt during the primary school the alphabet and the numbers and now we are able to recognize any alphanumeric character irrespective of whether its many different and variant appearances (e.g. stretched, blurred, rotated, colored, incomplete …). From a psychophysical point of view, one can wonder what kind of visual model our brain is using to represent such an item, made available to the cognitive system that, in conjunction with the perceptual system, finally provides that item with a semantic meaning for recognition task. The visual model of one character could be for instance a set of lines or curves whose connections describe the whole item shape.

From a physiological point of view, we know that the vision system entails numerous processing steps that progressively transform and compress the visual signal, from the retina up to the visual cortex, along devoted neural pathways. Such a neural structure, result of the long-term evolution to optimize the retinal image processing, involves the extraction of significant static and dynamic features (visual content) and their encoding in the visual cortex. This brain region encompasses some 140 million neurons, with tens of billions of connections between them. And yet human vision includes the primary visual cortex V1, and the entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing. Specifically, the human visual cortex had evolved up to a distributed structure where specific neural cells are task (feature orientation)- and spatial (retinal location)- dependent, that is they activate when one specific visual feature (e.g. a line) is present in the retinal image at a particular position. The sensitivity to the direction of linear visual patterns for instance seems innate but the use (experience) specializes and refines single cell populations. As we know from Hubel and Wiesel (Nobel laureates, 1959), the visual cortical structure allows in general the encoding throughout several neural layers that gradually increase the description specificity of the visual features. This neural framework can be transposed formally to a representation in depth of the visual content, performed by a cascade of neural populations, where the population size is expected to decrease as the depth increase (data compression).
It can be easily argued that this kind of representation is not only specific to the visual cortex but it is the working principle of the overall brain model. This paradigm is usually termed “Learning in depth” or “Deep learning” mechanism. According to this principle, the sensorial experience, along with any piece of knowledge, is deconstructed into the brain throughout a chain of neural layers. It is assumed that such a chain, joining the sensation to semantics, consists of heterogeneous links, each of them representing one specific feature map (e.g. sensorial, geometric, perceptual, …, emotional, semantic), with the level of abstraction increasing when moving from the head to the tail of the chain. Stepping back to the visual recognition task, we can hypothesize that the scalar retinal representation (pixel intensity map) is the first link of such a chain and the signal pattern of the ganglion cells, representing the neural fingerprint of the retina, is the second. According to the physiological processing performed by lateral geniculate nucleus (edge detection) and to the encoding (location and orientation of detected edges) performed in the primary visual cortex, we can expect that the retinal image undergoes decomposition in the brain as a map of nodes, spatially distributed (retinotopic map), where each node encodes for a particular geometric visual feature. In vision science, this decomposition is part of a more complex physiological processing where a set of computational steps, namely image segmentation, image completion, spatiality modeling, object modeling and perception are mandatory to allow recognition (Fig. 1).

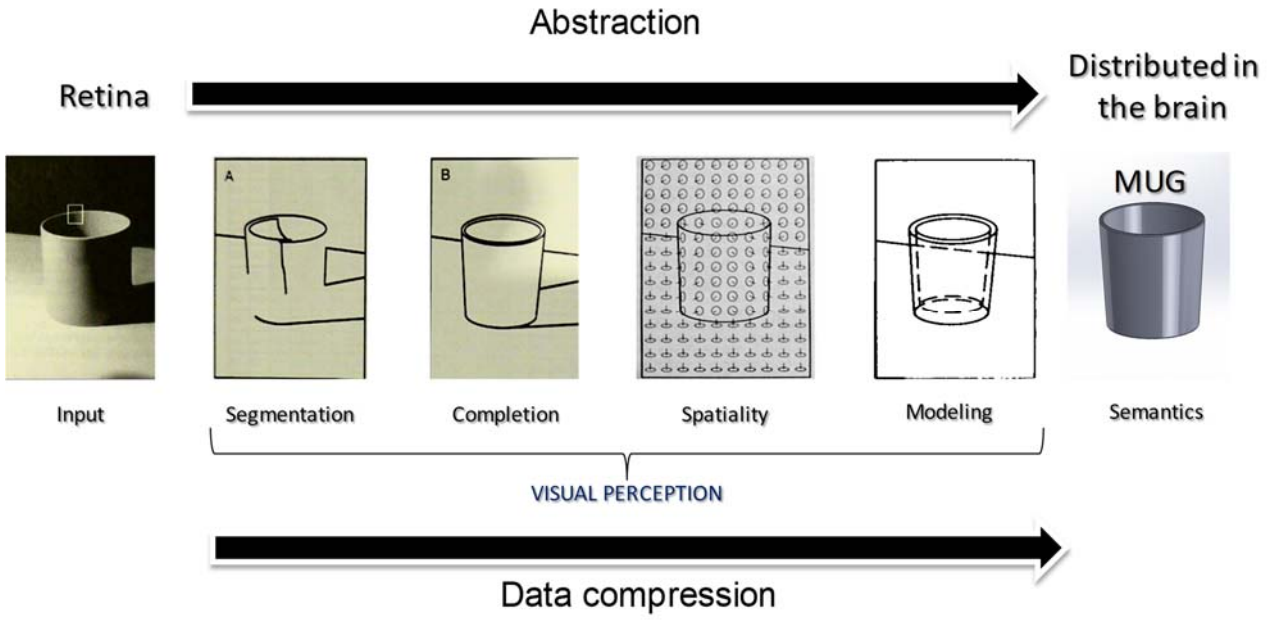


Figure 1. Simplified deconstruction model of the retinal image into following steps up to semantics.

Segmentation (process of extracting static and dynamic light contrast changes in the retinal image) for example is a complex computational step involving devoted neural processing and distributed encoding. Center-surround cells in the lateral geniculate nucleolus elaborate first the neural patterns coming from ganglion cells in the retina (representing retinal receptive fields) and their output patterns are then encoded by the simple cells in the V1 area of the visual cortex (Fig. 2). We know that the receptive field can be mapped by determining the response to small bright spots in each sub-region of the retina. Within the facilitator region, the stimulus forces an increase of the firing rate with respect to the activity in case of no stimulus. Within the inhibitory region, the stimulus forces a decrease of the firing rate.

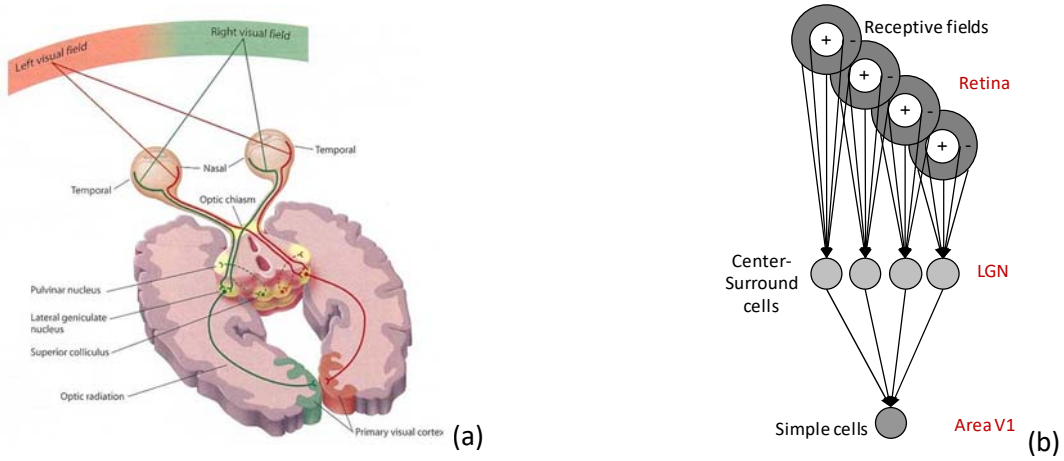


Figure 2. (a) Visual neural pathways from retina to primary visual cortex. (b) Connection schematics of receptive fields in the retina, center-surround cells in lateral geniculate nucleolus and simple cells in the V1 area.

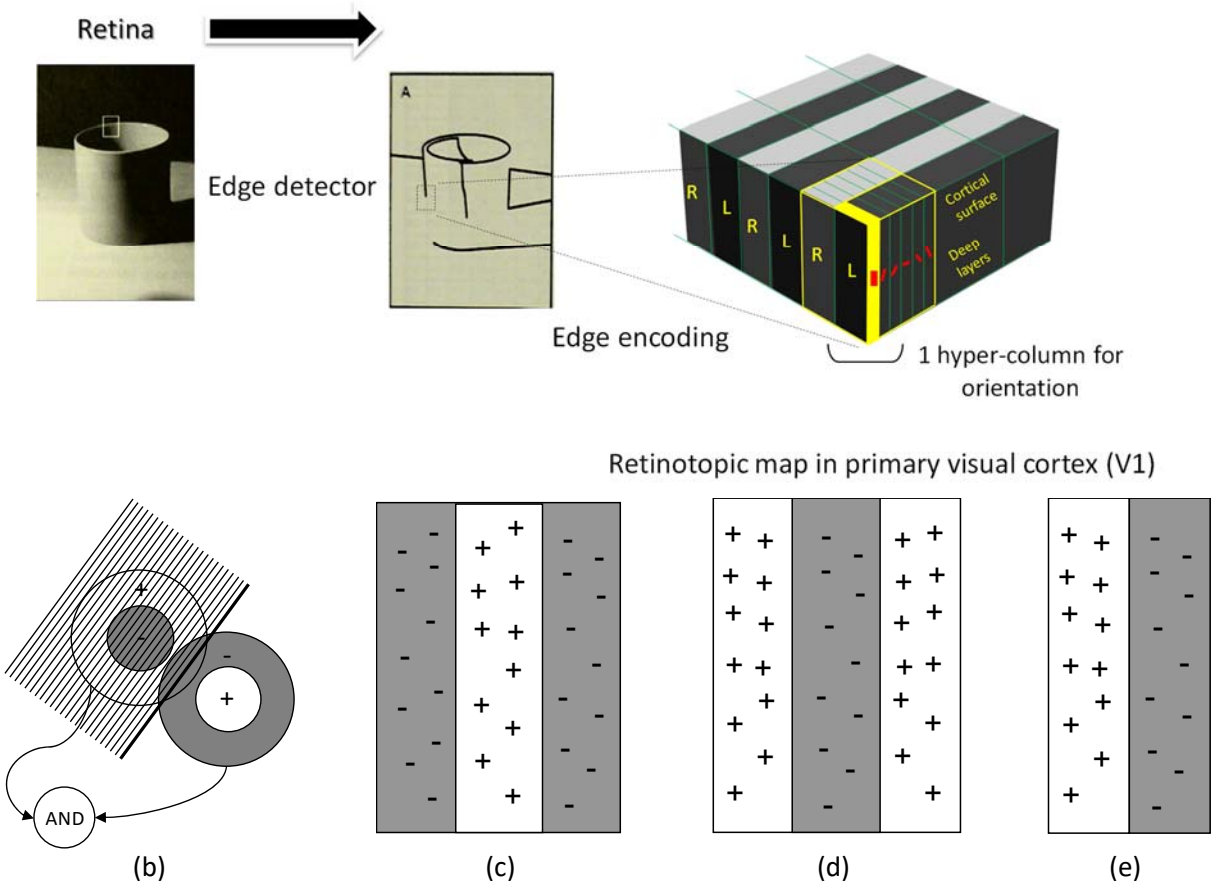
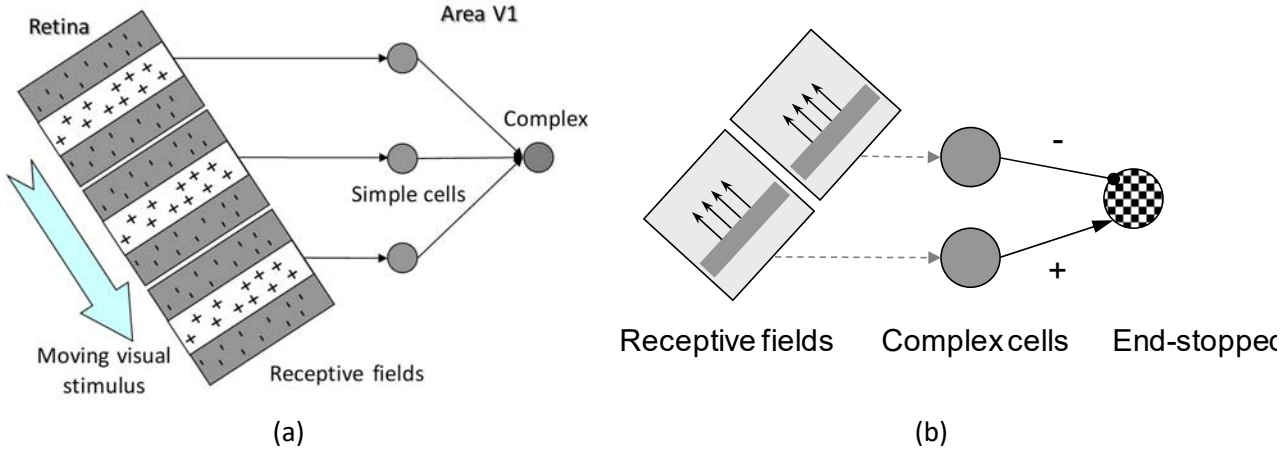


Figure 3. (a) Retinotopic map and the regular progression along a dimension that represents the cell tuning to orientation. (b) Composition in the LGN of two opposite receptive fields producing a dark-bright edge detector. (b)

Detector for bright vertical line (simple cell in V1). (c) Detector for dark vertical line (simple cell in V1). (d) Detector for dark-bright border (simple cell in V1).



(a) (b)

Figure 4. Receptive fields (and neural wiring) of complex (a) and hyper-complex (b) cells in the primary visual cortex.

The composition of several contiguous receptive fields produces elongated receptive fields, which are the basis for the readings of edges and lines. As earlier stated, simple cells in V1 area of the primary visual cortex activate retinotopically to encode static visual patterns (Fig. 3). The dynamics of the visual patterns is encoded by the so-called complex cells in the visual cortex that are thought to receive input from several simple cells whose receptive fields have the same orientation but different positions in the retina. Their response to stationary spots is very low whereas the response to moving spots is high. While simple cells are encoding both lines and edges with their own orientation, complex cells are encoding the direction of the motion of the line/border. This makes their receptive fields (in the retina) wider than the receptive fields of corresponding simple cells (Fig. 4a). A third type of cells, present both in the primary and other visual cortices, have more selective receptive fields than complex cells. They are called hyper-complex. The most interesting feature of these cells is that extending a line or edge beyond a certain length causes them to fire less strongly than they would do to a shorter line or edge. For this reason, they are also termed end-stopped cells (Fig. 4b). The identification of specific cells, devoted to encode lines and segments (with specific location and orientation), defined in the retinal coordinate system, allows arguing that any visual item is actually encoded into a visual feature map. This map can be regarded as a number of layers, each representative of specific geometric characteristic as diagonal, horizontal, and vertical lines, vertices, and ends, to cite the most relevant (Fig. 5a). In each layer, the corresponding geometric feature is encoded retinotopically. The idea that object shape might be represented as maps (rather than simple lists) of features has high degree of physiological plausibility because many areas of visual cortex appear to contain maps organized by retinal location as we discussed earlier. When associated to a learning task, the decomposition into such geometric building blocks, performed by the visual feature map, is associated to a structural description of the item. This further representation specifies the links between the visual features that serve at reconstructing the visual appearance of such an item. This composite representation (visual + structural) is memorized (in a distributed way) and made available to cognitive tasks when necessary. Structural description are representations that contain explicit information about parts and relations between parts. They are usually depicted as networks in which nodes represent the whole object, various parts of the object and labeled links (or arcs) between nodes denote the relations between parts. As an example, let us consider the structural description of the visual shape of an ‘A’ presented in figure 5b. The highest node represents the entire figure, and the lower nodes represent its geometric parts, namely the line segments and vertices. The arcs between nodes individuate obviously the spatial relations of the shape components. This basic structural description bears significant similarities to hierarchical schemes allowing relation information among the parts to be explicitly encoded. Structural descriptions can be extended to include intrinsic reference frame for the whole object and for each part. In principle, this means that simple translations and rotations of the shape can be embedded into the description, thus providing a modeling kernel that makes the representation invariant to rigid transformation.

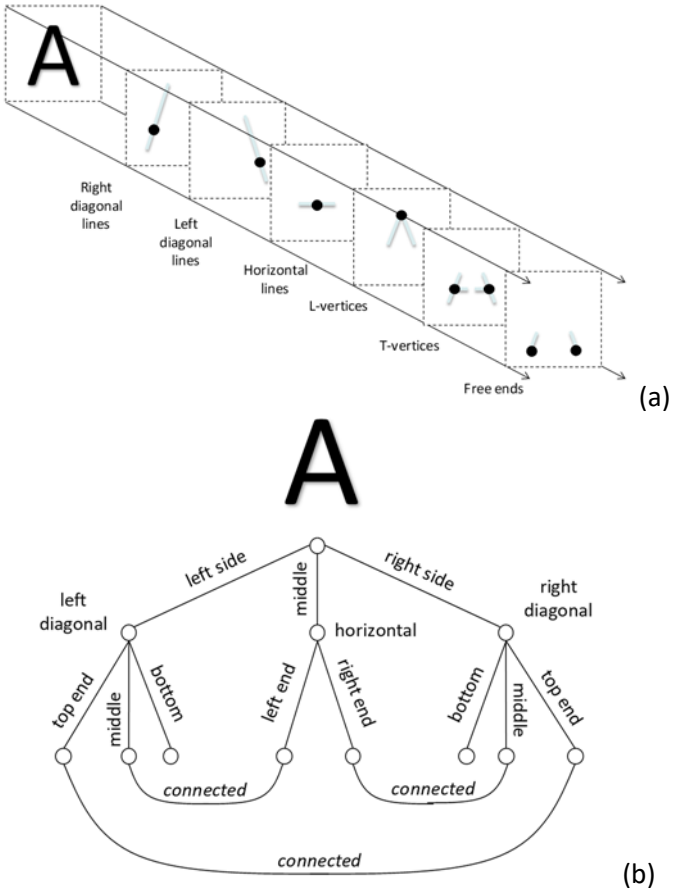


Figure 5. (a) Example of the decomposition of the A character into its geometric building blocks which describe the visual feature map of such an item. (b) Structural feature map for the same item, which exploits the visual feature maps to establish connection among single features that allows reconstructing the item visual shape.

CONVOLUTIONAL NEURAL NETWORK FOR IN-DEPTH FEATURE ENCODING

CONVOLUTION AND FEATURE MAP

We have seen that cells in the visual cortex are sensitive to small sub-regions of the visual field, called receptive fields. Such sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images. Convolutional neural networks (CNN) are FFNN designed to exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. In other words, the inputs of hidden units in layer **m** are from a subset of units in layer **m-1**, units that have spatially contiguous receptive fields. This wiring modality of the CNN differs therefore from the traditional fully connection we have seen so far in the FFNN. This paradigm is called sparse connectivity (Fig. 6).

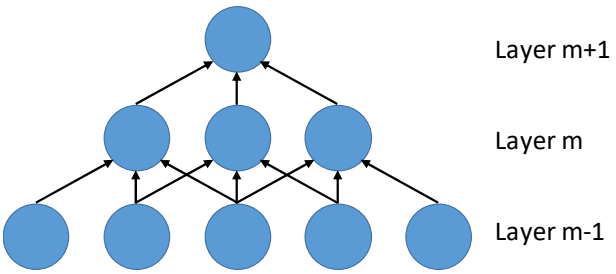


Figure 6. CNN wiring exploiting spatially-local correlation.

Let us consider that layer **m**-1 is the input retina. In figure 6, units in layer **m** have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer. Units in layer **m**+1 have a similar connectivity with the layer below. We say that their receptive field with respect to the layer below is also 3, but their receptive field with respect to the input is larger (5). Each unit is unresponsive to signals outside of its receptive field with respect to the retina. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. However, as shown above, stacking

many such layers leads to (non-linear) "filters" that become increasingly "global" (i.e. responsive to a larger region of pixel space). For example, the unit in hidden layer **m**+1 can encode a non-linear feature of width 5 (in terms of pixel space).
Convolutional neural networks use three basic ideas: *local receptive fields*, *shared weights*, and *pooling*. A local receptive field consists of a set of connections (which determines the local connectivity), along the corresponding weight set (which determines the filter property), that is replicated across the entire visual field. These replicated units share the same parameterization (weights and bias) and form a feature map. Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt (Fig. 7). The constraints on the model enable CNNs to achieve better generalization on vision problems.

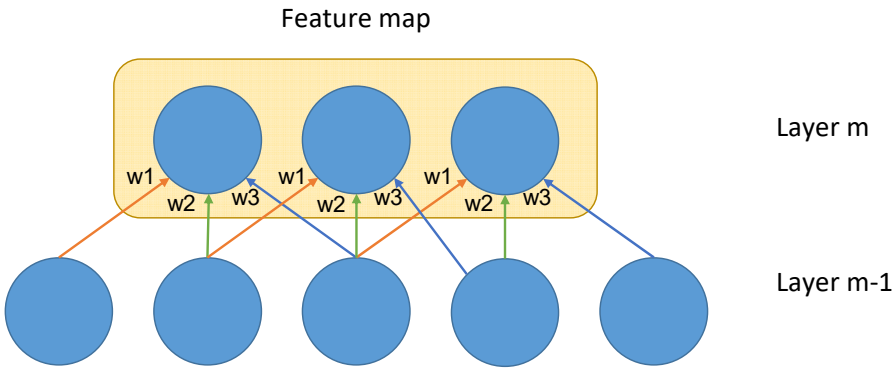


Figure 7. Three hidden units belonging to the same feature map. Weights of the same color are shared—constrained to be identical.

A feature map is attained by repetitive application of a function across sub-regions of the whole image, in other words, by *convolution* of the input image with a linear filter, adding a bias term and then applying the neural activation function. If we designate the *i*-th feature map at a given layer as $u^{(i)}$, whose filter is determined by the weight matrix $W^{(i)}$ and bias (neuron threshold) $b^{(i)}$, then the feature map $u^{(i)}$ at the specific location (j,k) in the layer reads as follows:

$$u_{j,k}^{(i)} = tanh\left(b^{(i)} + \sum_{l=1}^{f_x}\sum_{m=1}^{f_y} w_{l,m}^{(i)} a_{j-1+l,k-1+m}\right)$$

where f_x and f_y are the x and y sizes of the filter, and $a_{x,y}$ is the input activation at position *x*,*y*. When considering the first convolutional layer then input activation corresponds to the image pixel value. By convention, the bias b is equivalent to the traditional threshold multiplied by the virtual input -1. For example provided that we have a 40×40 pixel image and we set a local receptive field of 5×5, the output of the first neuron mapping the receptive field will be computed as:

$$u_{1,1} = tanh\left(b + \sum_{l=1}^{5}\sum_{m=1}^{5} w_{l,m} a_{0+l,0+m}\right)$$

requiring only 26 parameters (25 weights + 1 threshold).
Assuming that we adopt a zero-padding strategy (adding zeros to the image boundary) for the convolutional product, the output size (s_x, s_y) of each convolutional layer can be computed as:

$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1$$
$$s_y = \frac{(I_y + 2z_y - f_y)}{L_y} + 1$$

where I_x and I_y are the input image sizes in x and y directions, z_x and z_y are the zero-padding sizes, f_x and f_y are the filtering sizes, and L_x and L_y are the stride lengths of the filter. The stride accounts for quantification (in pixel) of the shift the filter is undergoing when convoluted with the image. A stride length of 1 will correspond to the filter shift of one 1 pixel. A stride length of 3 will correspond to the filter shift of one 3 pixels. Let us assume a 240×240 pixel images, a convolutional filter of 7×7 pixels with a stride length of 3 and a 2-pixel zero padding both in x and y directions. The output of the convolution will be then an image map of 80×80 pixels.

The network wiring, of a network processing an image, between the input and the first convolutional hidden layers can be seen in Fig. 8. As a consequence, the network is structurally not fully connected.

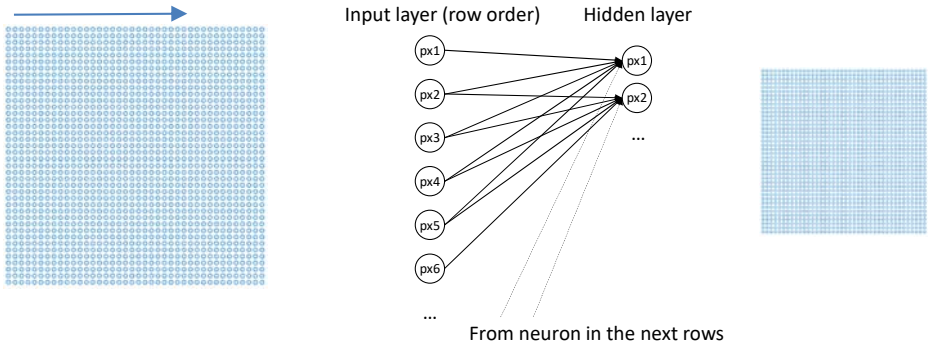


Figure 8. Network wiring.

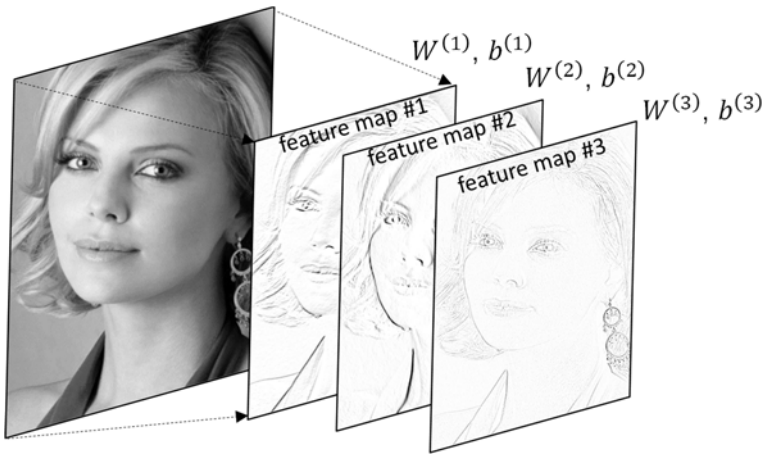


Figure 9. Multiple feature maps into a single layer.

Multiple feature maps can be applied which provide a richer synthesis of the data. This means to endow each hidden layer with multiple *feature* maps $\{u^{(i)}, i = 1 \dots N\}$. By stacking the weight matrices $\{W^{(i)}\}$ of a hidden layer into a 4D tensor provides a compact representation of the processing for the layer. Such a tensor contains entries for every combination of destination feature map, source feature map, source vertical position, and source horizontal position. Similarly, the biases $\{b^{(i)}\}$ can be represented as a vector containing one element for every destination feature map (Fig. 9). Taking a broad view, $W_{j,k}^{(i,g)}$ denotes the weight connecting each pixel of the *i*-th feature map at layer m, with the pixel at coordinates (j, k) of the *g*-th feature map of layer (m-1).

In case of color images, the convolutional network generalizes to feature maps processing all the three color channels in the image (Fig. 10). Assuming we have a 40x40 input image, and 5×5 local receptive fields, then there will be 36×36 neurons in the hidden layer. This is because we can only move the local receptive field 35 neurons across (or 35 neurons down), before colliding with the right-hand side (or bottom) of the input image.

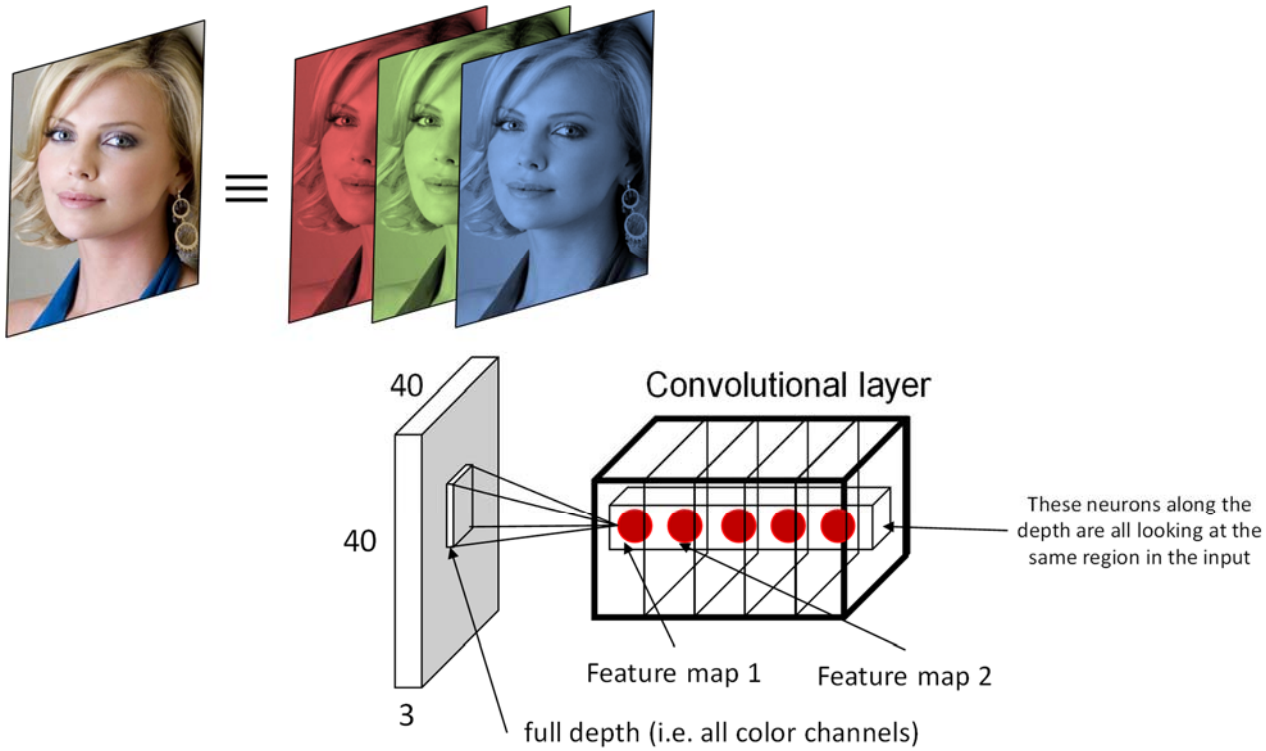


Figure 10. Input "volume" (e.g. 40×40×3) and the volume of neurons in the first Convolutional layer. Each neuron is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input.

An example of RGB input volume, where each neuron in the convolutional layer is connected only to a local region in the input volume spatially distributed across the three channels, is shown in Fig. 11.

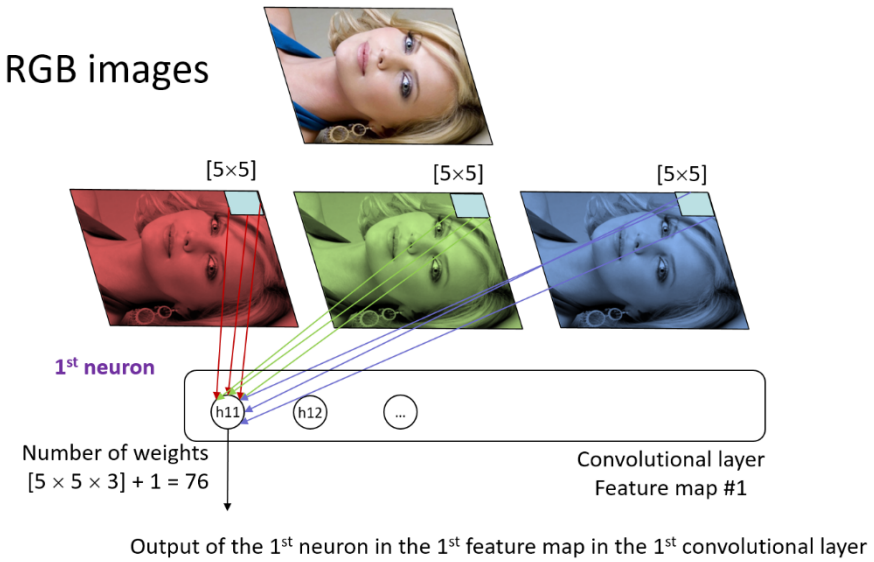
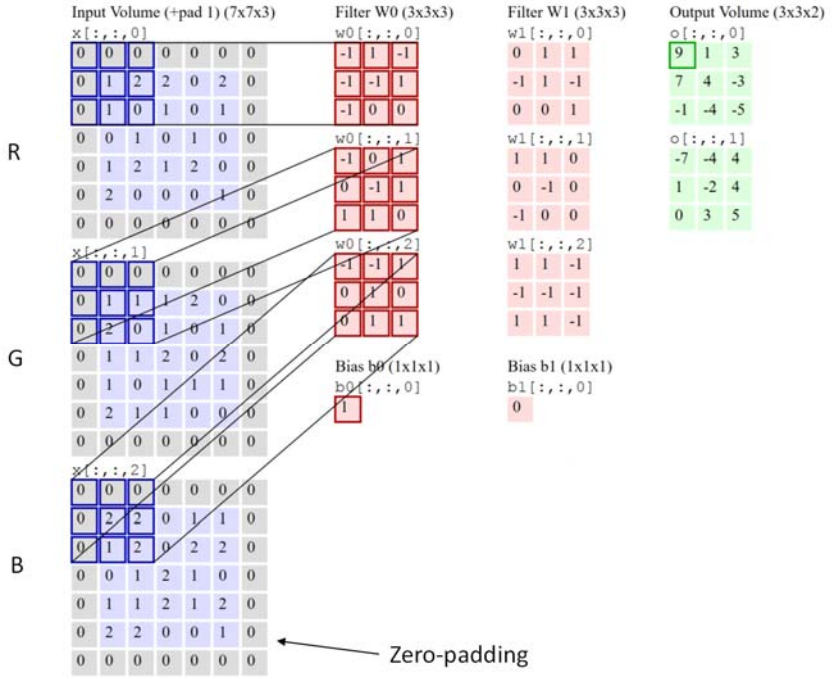
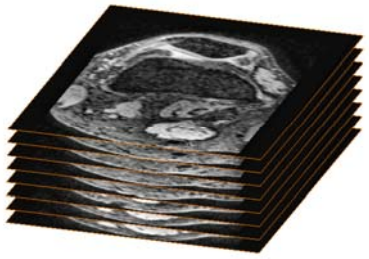


Figure 11. One neuron in the convolutional layer of one feature map is connected only to a local region in the input volume spatially distributed across the three channels.

In this paradigm, one processing neuron in the hidden layer, belonging to one feature map, encodes therefore a receptive field which span across the three image channels (red, green and blue) as explicitly depicted in the next figure.



So far, the described convolution operation has been considered bi-dimensional (2D) even though in case of RGB images the convolution filter of a generic feature map span the three channels. Actually, the three channels can be regarded as a volume. However, there is no real 3D span in the third dimension of the 3D image.

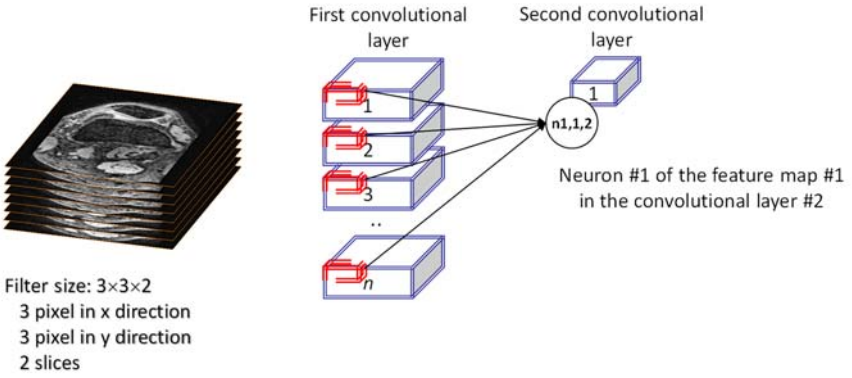


When dealing with volumetric image as the case of scans (MRI, CT, …) conversely the third dimension is explicit as the data is a representation of a physical characteristic of an images body (magnetic spin relation, x-ray absorbance, ..) sampled into a 3D space. Typically, 3D biomedical images are described in terms of number of pixels in x and y direction, along with the number of slices taken in the z direction. Applying a convolution filter to such data means to define not only the x, y size but even the z size which specifies how many slices will be processed simultaneously. Likewise, in this 3D convolution that is determined by the corresponding 3D feature map, we can compute the size of output of the convolutional processing as:

$$s_x = \frac{(I_x + 2z_x - f_x)}{L_x} + 1$$
$$s_y = \frac{(I_y + 2z_y - f_y)}{L_y} + 1$$
$$s_z = \frac{(n_s + 2z_z - f_z)}{L_z} + 1$$

where n_s is the number of slices of the scan, f_z is the number of slices addressed by the filter, L_z is the stride of the filter in the z direction. Considering an input volume of 65×65×8, composed say by 33800 voxels, a 5×5×2 filter with no padding and stride equal to 2 in all the three direction, the feature map will endow 3844 neurons, all featuring 51 parameters (50 weights and 1 threshold). This feature map can be regarded as a 3D distribution of neurons regularly distributed on a 3D grid of 31×31×4. Assuming that the first convolutional

layer encompasses *n* different features maps, one neuron in the second layer, belonging to the first feature map will be spanning all the feature maps in the previous layer. Provided that the filter size is 3×3×2 with no padding and stride of 2 along all the three directions, then the complete feature map will consist of 15×15×2 neurons, i.e. 450 neurons in total. Each neuron therefore will have (3×3×2) ×n +1 free parameters.



Another important concept of CNNs is max-pooling, which is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value. Max-pooling is useful in vision for two reasons: 1) by eliminating non-maximal values, it reduces computation for upper layers; 2) it provides a form of translation invariance. Imagine cascading a max-pooling layer with a convolutional layer (Fig. 12). There are 8 directions in which one can translate the input image by a single pixel. If max-pooling is done over a 2x2 region, 3 out of these 8 possible configurations will produce exactly the same output at the convolutional layer. For max-pooling over a 3x3 window, this jumps to 5/8. Since it provides additional robustness to position, max-pooling is a "smart" way of reducing the dimensionality of intermediate representations.

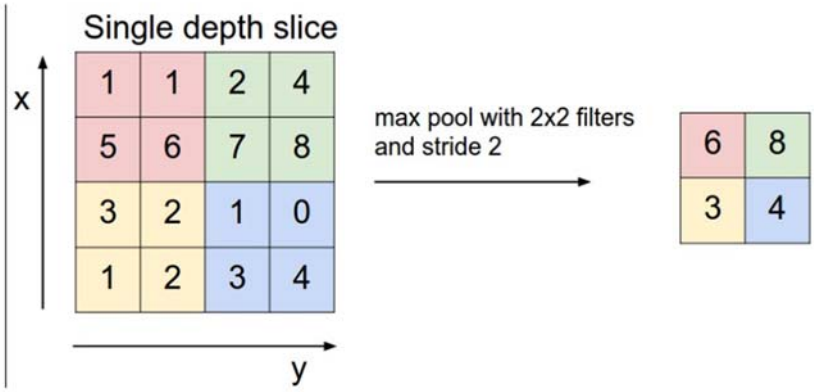
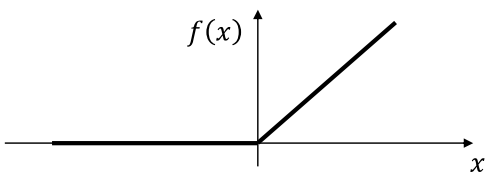


Figure 12. Max pooling with 2x2 filters and stride 2.

Another processing layer utilized in CCN is represented by the rectified linear unit layer (ReLU). It is chained to the convolutional layer to ensure that all the feature maps have positive signals (remember that we are using processing and activation function tanh with potential negative outputs).

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



The basic idea of building deep learning with CNN leads to bind together blocks constituted by the three elements above (Convolutional layer, ReLU, Max Pooling) with decreasing number of units up to a final stage of feature classification. Usually this consists of a fully connected FFNN shaped as a cluster network (e.g. softmax) whose multi-dimensional output refers to the number of predefined features or classes (Fig. 13). Softmax network maps a N-dimensional vector of arbitrary real values to a N-dimensional vector of real values in the range (0, 1) that add up to 1. This is obtained using normalized exponential activation as: $z_i = e^{P_i} / \sum e^{P_j}$. The CNN can be trained using the traditional algorithm of backpropagation. ReLU and Pooling layers do not require learning.

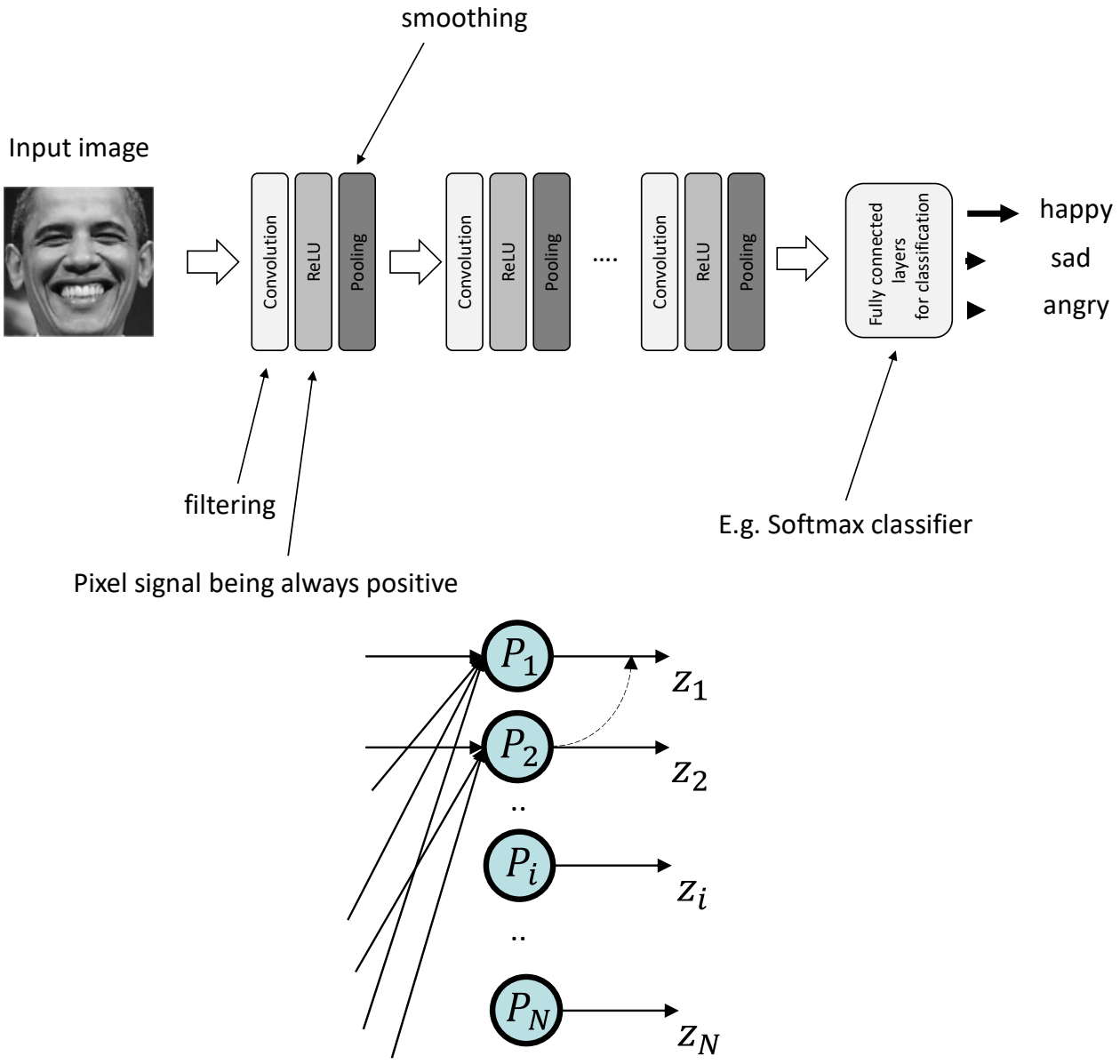


Figure 13. (Upper panel) Example of a classifier built using multi-layer convolutional neural network. The last layer consists of a fully connected FFNN shaped as a cluster network (e.g. softmax). (Lower panel) Softmax connectivity. It "maps" a *N*-dimensional vector of arbitrary real values to a *N*-dimensional vector of real values in the range (0, 1) that add up to 1. This is obtained using normalized exponential activation as: $z_i = e^{P_i} / \sum e^{P_j}$.

Let us consider a convolutional neural network aimed at classifying image content. The network is to be tailored for processing RGB color 120×120 pixel images. It has to be assumed that the convolutional processor in the first layer (convolution, RELU and max-pooling and) should detect 3 different features, while the second layer should detect 2 different features. Both layers have a 2×2 pixel size for max pooling. Let us

consider a 1-pixel zero padding, both in x and y directions. In the first layer, we take a 4×4 filter with stride length being equal to 2. The size of the output of the second layer is 8×8 pixels.
According to this setup, one can say that the output of the first convolutional layer will count 3 maps whereas the second will count 2 maps. The size of one activation map before max-pooling in the first layer will be 60×60 pixels whereas after the max-pooling the size will be 30×30. If one assumes that the stride length in the second layer is 2 than, in order to ensure the output size of 8×8, as asserted above, the filter size should be 2×2pixels.
As far as training is concerned, there are two main differences between multi-layer perceptron (MLP) backpropagation (fully connected layer) and convolutional nets:

1) the influence of weights is localized, so first figure out how to do backprop for, say a 3×3 filter convolved with a small 3x3 area of an input image, mapping to a single point in the result image.

2) the weights of convolutional filters are shared for spatial invariance. What this means in practice is that in the forward pass the same 3×3 filter with the same weights is dragged through the entire image with the same weights for forward computation to yield the output image (for that particular filter). What this means for backprop is that the backprop gradients for each point in the source image are summed over the entire range that we dragged that filter during the forward pass. Note that there are also different gradients of loss wrt x, w and bias since dE/dx needs to be backpropagated, and dE/dw is how we update the weights. w and bias are independent inputs in the computation DAG (there are no prior inputs), so there's no need to do backpropagation on those.