Neural Networks for Images and Spatial Information

Danilo Comminiello

Neural Networks 2023/2024

October 25, 2023

Sapienza
Università di Roma

Table of contents

1 Going Beyond Multiplayer Perceptron

Modeling Complex Systems
Drawbacks of Multilayer Networks
Need for Deeper Architectures

Modeling complex systems

Neural networks (NNs) are gaining a lot of attention due to their favorable capabilities, making them one of the hottest topics in the machine learning field.

The basic concept behind NNs has been built on hypotheses and models of physical and natural behavior in order to solve complex problems.

The modeling of such complex systems requires the *efficient* training of a big amount of data by NNs with many layers.

In this lecture, we are going to learn the basic concepts that lead to deal with advanced NN architectures that are particularly well suited for spatial information, such as images.

Looking for deeper representation I

The solution of complex tasks usually requires an inherently hierarchical representation, which is at the basis of modern deep learning.

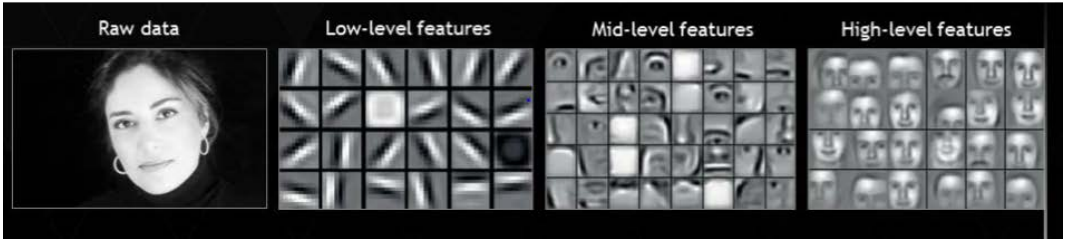


Figure 1: "Deep" learning implies learning features of features of features...
Image source: Analytics Vidhya.

Looking for deeper representation II

On one hand, there might exist a representation of our data that would take into account the relevant interactions among our features, on top of which a linear model would be suitable.

However, on the other hand, we simply do not know *a priori* how to calculate it by hand.

We need neural networks that are able to use observational data to jointly learn both a representation (via hidden layers) and a linear predictor that acts upon that representation.

Unfortunately, multilayer perceptrons (MLPs) may show some issues when modeling complex systems.

Drawbacks of multilayer networks I

Major drawbacks of multilayer neural networks are that:

- their training can become difficult,
- backpropagation-related algorithms may often stuck in local minima.

Some improvements can be obtained by trying different practical *tricks*, such as *multiple training* using random initialization.

However, even with tricks, generalization performance of multilayer NNs still may not be competitive to other methods.

Drawbacks of multilayer networks II

These drawbacks become even more severe if more than two hidden layers are used.

The more layers one uses:

- the more difficult the training becomes,
- and the probability to recover solutions corresponding to poor local minima is increased.

As a matter of fact, efforts to use more than two hidden layers were soon abandoned, until we had to face problems involving large amounts of data.

Issues to be addressed

From now on, we are going to focus and address on the following two issues:

1. Is there any need for networks with more than two or three layers?
2. Is there a training scheme, beyond or complimentary to the backpropagation algorithm, to assist the optimization process to settle in “good” local minima, by extracting and exploiting more information from the input data?

We will provide answers to both these points, starting form the first one.

Need for deeper architectures

In particular, we want to know whether one can obtain an equivalent input-output representation via a relatively simple functional formulation, e.g., via *shallow* networks with less than three layers of neurons, maybe at the expense of more elements per layer.

The answer is **yes**, as long as the input-output dependence relation is simple enough.

However, for more complex tasks, where more complex concepts have to be learned, e.g., recognition of a scene in a video recording, language and speech recognition, the underlying functional dependence is of very a complex nature that we are unable to express it analytically in a simple way.

Need for better generalization I

The answer to the second point, concerning networks, lies in what is known as compactness of representation.

We say that a network, realizing an input-output functional dependence, is compact if it consists of relatively few free parameters (few computational elements) to be learned and tuned during the training phase.

Thus, for a given number of training points, we expect compact representations to result in better generalization performance.

Using networks with more layers can lead to more compact representations of the input-output relation.

Need for better generalization II

Results from the **theory of logic circuits** of Boolean functions suggest that:

- a function, which can compactly be realized by k layers of logic elements, may need an exponentially large number of elements if it is realized via $k-1$ layers.

Since the number of computational elements one can afford depends on the number of training examples available, the consequences are not just *computational* but also *statistical*: poor generalization may be expected when using NNs with few layers for representing some functions.

Some of these results have been generalized and are valid for learning algorithms in some special cases.

2 From Dense Layers to Convolutions

Designing Structured Networks

Invariances

Designing structured networks

Sometimes we truly may not have any knowledge to guide the construction of structured architectures.

In these cases, a multilayer perceptron is often the best that we can do.

However, once we start dealing with high-dimensional perceptual data, these *structure-less* networks can grow unwieldy.

Images exhibit rich structure that is typically exploited by humans and machine learning models alike.

Leveraging pixels correlation

Until now, we have dealt with 2D images simply flattening each image into a 1D vector, and fed it into a fully-connected network.

This network is invariant to the order of their inputs.

However, if we consider the processing of high-quality images, learning the parameters of this network may turn out to be impossible.

Ideally, we would leverage the prior knowledge that nearby pixels are more related to each other.

Convolutional neural networks (CNNs) are a powerful family of neural networks that were designed precisely for this purpose.

Vectorization of an image

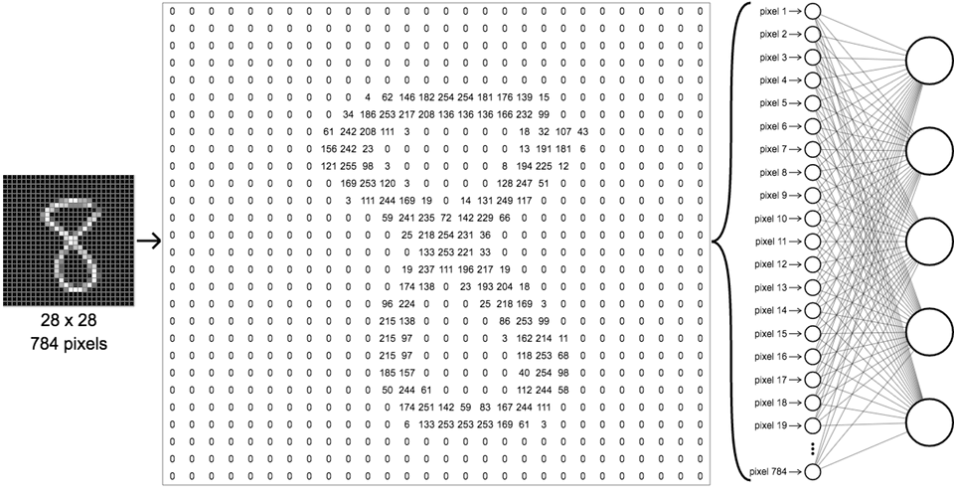


Figure 2: 1D vector flattening of a 2D image, the digit 8 of the MNIST dataset. Source: TowardsDataScience.com.

Detecting an object in an image I

A method for the detection of an object in an image should not overly rely on the precise *location* of the object.

Ideally, a detection method would somehow exploit this knowledge.

As an example, pigs usually do not fly and planes usually do not swim.

This ideas is taken to an extreme in the children's game "Where's Waldo", an example is shown in the next figure.

Detecting an object in an image II

Figure 3: The game “Where’s Waldo” consists of a number of chaotic scenes bursting with activity and Waldo shows up somewhere in each (typically lurking in some unlikely location). The reader’s goal is to locate him. Despite his characteristic outfit, this can be surprisingly difficult, due to the large number of confounders [1].

Exploiting spatial structure

We can derive some properties for a good object detector method using the spatial arrangement of pixels on an image:

- **Translation invariance**: the same object can appear equivalently on any part of the image.
- **Locality**: an object should depend only on a *local region* of the overall image, without regard for what else is happening in the image at greater distances.

These assumptions bring forth immediately the idea of filters and image convolutions.

3 Review of the Convolution Operation

Continuous- and Discrete-Time Convolution
Multidimensional Discrete-Time Convolution
Image Convolution with Kernels

Definition of the continuous-time convolution

Let's briefly review the operation of **convolution** and its application to spatial information.

In *mathematics*, the continuous-time convolution between two functions, say $f, g : \mathbb{R}^d \to R$ is defined as:

$$[f \circledast g](t) =< f(\tau), g(t-\tau) >= \int_{\mathbb{R}^d} f(\tau)g(t-\tau)d\tau.$$

That is, we measure the overlap between f and g when both functions are shifted by τ and "flipped".

Equivalently, it holds:

$$[f \circledast g](t) =< f(t-\tau), g(\tau) >= \int_{\mathbb{R}^d} f(t-\tau)g(\tau)d\tau.$$

Discrete-time convolution

Whenever we have discrete objects, the integral turns into a sum:

$$< f[n], g[n-k] >= \sum_{\mathbb{R}^d} f[k]g[n-k],$$

or equivalently:

$$< f[n-k], g[n] >= \sum_{\mathbb{R}^d} f[n-k]g[k].$$

The convolution can be easily seen as an *inner product* operator for each translation index.

Multidimensional discrete-time convolution

For 2D arrays (e.g., gray-scale images), we have a corresponding sum with indices (i, j) for f and $(i - \tau_1, j - \tau_2)$ for g respectively.

Specifically, in case of a discrete-time system, the 2D convolution can be defined, in terms of discrete translation indices k_1, k_2, as:

$$< f[i,j], g[i-k_1, j-k_2] >= \sum_{k_1 \in \mathbb{R}^d} \sum_{k_2 \in \mathbb{R}^d} f[k_1, k_2] g[i-k_1, j-k_2] \tag{1}$$

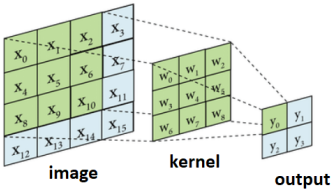


Figure 4: Representation of the 2D convolution output. Image source: Medium.

Designing kernels in image processing

A kernel can be designed to extract a specific information from an image.

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45° oblique lines

2	-1	-1
-1	2	-1
-1	-1	2

-45° oblique lines

Figure 5: Here is an example of kernels designed to extract particular tracts of an image, related to edges.

Convolution of designed kernels with an image

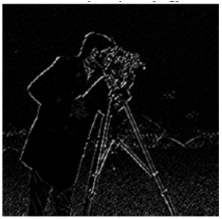
Original image

Low-pass filter

High-pass filter

Maximally to horizontal

Maximally to vertical



Maximally to 45° oblique

Figure 6: Results of the convolution of the original cameraman image with different kernels.

Convolution with a kernel to apply image transformations

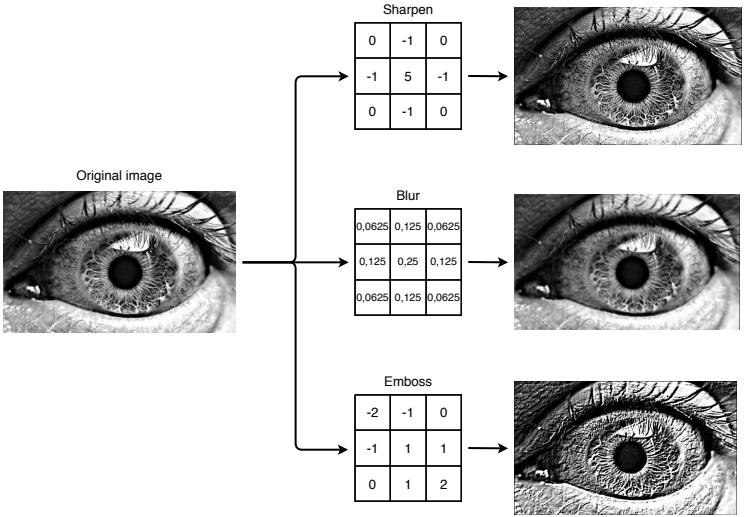


Figure 7: For an interactive visualization, see `http://setosa.io/ev/image-kernels/`.

4 Convolutions in Neural Networks To Leverage Spatial Information

Constraining the MLP

Convolutional Layers

Constraining the MLP to leverage spatial structure

Consider what an MLP would look like with 2D input images and hidden representations with same size $H \times W$.

Let $x[i, j]$ and $h[i, j]$ denote an image and its hidden representation, respectively, in the pixel location (i, j).

To allow each of the HW hidden nodes receiving input from each of the HW inputs, we would switch from using weight matrices (as in MLPs) to representing our parameters as four-dimensional weight tensors:

$$h[i, j] = u[i, j] + \sum_{l_1, l_2} W_{\text{MLP}}[i, j, l_1, l_2] \cdot x[l_1, l_2] = u[i, j] + \sum_{k_1, k_2} W[i, j, k_1, k_2] \cdot x[i + k_1, j + k_2].$$

The one-to-one correspondence between coefficients in both tensors is obtained just by setting $l_1 = i + k_1$ and $l_2 = j + k_2$.

Application of translation invariance

Now let's invoke the translation invariance principle: a shift in the inputs x should simply lead to a shift in the activations h.

This is only possible if W and u do not actually depend on (i, j), thus:

$$h[i, j] = u + \sum_{k_1, k_2} W[k_1, k_2] \cdot x[i + k_1, j + k_2].$$

This is a **convolution**! We are effectively weighting pixels $(i + k_1, j + k_2)$ in the vicinity of (i, j) with coefficients $W[k_1, k_2]$ to obtain the value $h[i, j]$.

Note that $W[k_1, k_2]$ needs many fewer coefficients than $W[i, j, k_1, k_2]$.

Application of locality

Now let's invoke the locality principle: it is not necessary to look very far away from (i, j) in order to glean relevant information to assess what is going on at $h[i, j]$.

This means that outside some range $|k_1|, |k_2| > \Delta$, we set $W[k_1, k_2] = 0$, thus:

$$h[i, j] = u + \sum_{k_1=-\Delta}^{\Delta} \sum_{k_2=-\Delta}^{\Delta} W[k_1, k_2] \cdot x[i + k_1, j + k_2]. \tag{2}$$

In a nutshell:

- $W[k_1, k_2]$ is referred to as a convolution kernel, or *filter*, or simply *weight matrix*;
- $h[i, j]$ in (2) represents the **convolutional layer**.

Cross-correlation... not convolution

Equation (2) looks similar to the 2D convolution definition in (1), with one major difference: rather than using $[i + k_1, j + k_2]$, we are using the difference instead, e.g., $[i - k_1, j - k_2]$.

However, it is worth noting that the definition in (2) is actually a **cross correlation**, thus *convolutional layers* are a slight misnomer.

Advantages and drawbacks of convolutional layers

When the local region, also called a *receptive field*, is *small*, the difference as compared to a fully-connected network can be dramatic.

While an MLP dense layer might have required billions of parameters, a convolutional layer typically needs just a few hundreds.

The price that we pay for this drastic modification is that our features will be translation invariant and that our layer can only take local information into account.

All learning depends on imposing inductive bias: when that bias agrees with reality, the models generalize well to unseen data.

Convolutional layer for colored images

In reality, images are rather as a 3^{rd} order tensor, e.g., with shape $1024 \times 1024 \times 3$ pixels.

We could think of the hidden representation as comprising a number of 2D grids stacked on top of each other.

These are sometimes called *channels* or *feature maps*.

Thus, we can generalize the definition of the convolutional layer as:

$$h[i, j, k] = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} W[a, b, c, k] \cdot x[i + a, j + b, c].$$

Spatial arrangement of output volume

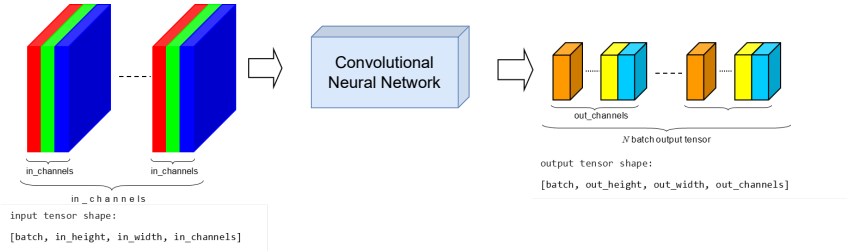


Figure 8: Multiple input channels are concatenated in batches. The output volume is arranged in output batches with lower dimensionality.

Waldo revisited

Here is what this looks like if we want to build an improved Waldo detector with convolutional layers.

Figure 9: The convolutional layer picks windows of a given size and weighs intensities according to the mask V. We expect that wherever the "waldoness" is highest, we will also find a peak in the hidden layer activations [1].

Next lecture

- We will introduce and define convolutional layers for structured signals to leverage the higher correlation between nearby values.
- We will walk through the basic operations that comprise the backbone of convolutional neural networks.
- Some of the most popular modern *ConvNets* will be presented.

References

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive Into Deep Learning*. 0.7.1 ed., 2020.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Boston, MA), pp. 1–9, June 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV), pp. 770–778, June 2016.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] Y. Bengio, "Deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, Nov. 2009.

Neural Networks for Images and Spatial Information

Neural Networks 2023/2024

Danilo Comminiello

https://sites.google.com/uniroma1.it/neuralnetworks2023

{danilo.comminiello, simone.scardapane}@uniroma1.it