POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

CNN: Convolutional Neural Networks

Theoretical Project of
Numerical Analysis for Machine Learning

Author: **Giorgio Monaco**

Student ID: 277930
Advisor: Prof. Edie Miglio
Academic Year: 2024-25

Contents

Introduction

Deep learning has revolutionized modern machine learning by enabling neural networks to automatically extract useful representations from raw data. Among the architectures that made this progress possible, *Convolutional Neural Networks* (CNNs) stand out as one of the most influential. Originally developed for computer vision, CNNs have also achieved remarkable success in speech recognition, time-series forecasting and natural language processing.

Unlike traditional fully connected networks, CNNs are designed to exploit the *spatial and structural properties of data*. They achieve this by using the mathematical operation of convolution, which processes information locally and hierarchically. This principle reflects the idea that nearby elements of the input (such as adjacent pixels in an image) are highly correlated and should be analyzed together. As a result, CNNs learn models that are more efficient, require fewer parameters and generalize better than dense architectures.

Conceptually, CNNs bring together mathematical insights from signal processing and biological inspiration from neuroscience. The notion of local receptive fields in the human visual cortex, described in the pioneering experiments of Hubel and Wiesel, 1968 [4], directly influenced the design of convolutional layers.

In this work, we address the mathematical definition of convolution and the motivation for its use in neural networks, analyze the role of pooling and its modern alternatives and discuss several important variants of the convolution operator. We then review applications of convolution across different types of data, from one-dimensional signals to video and volumetric analysis and examine the neuroscientific principles that inspired CNNs.

1 | What is Convolution

In its most general form, **convolution** is an operation on two functions f and g that produces a third function $f * g$, graphically expressing how the *shape* of one function is modified by the other. The term *convolution* refers to both the resulting function and to the process of computing it. In the context of deep learning, convolution is the fundamental building block of Convolutional Neural Networks (CNNs), allowing them to efficiently extract and combine local features from structured data such as images, audio signals or sequences.

1.1. One-dimensional convolution

The convolution of f and g is written $f * g$, typically denoting the operator with the symbol $*$. It is defined as the integral of the product of the two functions after one is reflected about the y-axis and shifted (is commutative, so both of them could be shifted):

$$s(t) = (f * g)(t) = \int f(\tau)g(t-\tau)d\tau \tag{1.1}$$

The first argument (in this example, the function f) to the convolution is often referred to as the **input** the second argument (in this case g) as the **kernel** and the output as the **feature map**.

When we work on computer data, time will be usually discretized. Formally, the discrete convolution of two functions f and g is defined as:

$$s(t) = (f * g)(t) = \sum_{\tau=-\infty}^{+\infty} f(\tau)\, g(t-\tau). \tag{1.2}$$

However in practice, the sums are finite since we deal with signals of finite length.

The commutative property of convolution arises because we have **flipped** the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. The only reason to flip the kernel is to obtain the commutative property.

While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation. In machine learning, CNNs usually employ a slightly different operation called *cross-correlation*, defined as:

$$s(t) = (f * g)(t) = \sum_{\tau=-\infty}^{+\infty} f(\tau)\, g(t+\tau). \tag{1.3}$$

The difference between convolution and cross-correlation is the absence of the flip in the kernel. Despite this technical distinction, the term "convolution" is commonly used in the literature to refer to both operations [2].

1.2. Two-dimensional convolution

For two-dimensional data such as images, convolution generalizes naturally. Given an image $I : \mathbb{R}^2 \to \mathbb{R}$ and a kernel (or filter) $K : \mathbb{R}^2 \to \mathbb{R}$, the discrete convolution is defined as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n). \tag{1.4}$$

Analogously, the cross-correlation used in CNNs is given by:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n). \tag{1.5}$$

In practice, K is a small matrix that slides over the image, producing an output feature map. Each element of the feature map is a local weighted sum of the pixels in the receptive field defined by the kernel.

1.3. Convolution as matrix multiplication

Convolution can be expressed as multiplication by a matrix with a special structure. In one dimension, this matrix is the *Toeplitz Matrix*, meaning each row is a shifted version of the kernel. In two dimensions, the operation corresponds to a *doubly block circulant matrix*, where each block is itself Toeplitz.

These matrices are highly structured, because weights are shared across positions, and sparse, since kernels are usually much smaller than the input. This perspective shows that convolution is formally a kind of matrix multiplication, although in practice deep learning frameworks use optimized implementations that exploit sparsity and weight sharing rather than explicitly building these large matrices.

2 | Motivation for Convolution

Convolution introduces three key principles: *sparse interactions*, *parameter sharing* and *equivariance to translation* [2].

2.1. Sparse interactions

In a traditional neural network layer, every output unit interacts with every input unit. For high-dimensional data, such as images, this leads to an extremely large number of parameters. Convolution addresses this by connecting each output only to a local neighborhood of the input, known as its *receptive field*.

If the kernel has size $k \times k$, then each output depends on only k^2 inputs, rather than the entire input dimension. It is often possible to obtain very good performance on a ML task while keeping k several orders of magnitude smaller than the inputs.

This sparsity not only reduces the number of parameters, but also encodes the prior knowledge that local groups of variables are often more strongly correlated than distant ones.

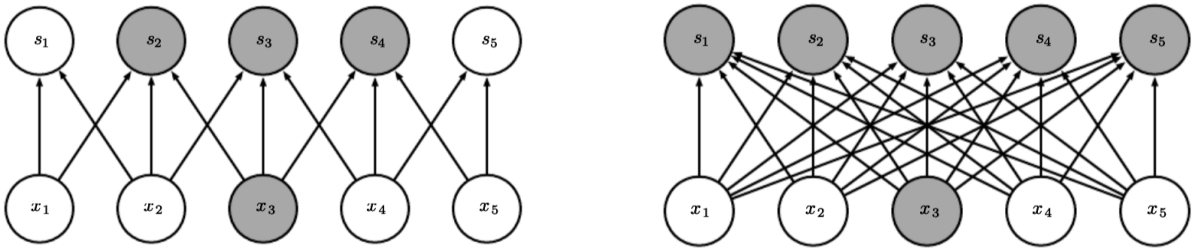

Figure 2.1: Sparse connectivity: when s is formed by convolution (left), and when s is formed by matrix multiplication (right).

In a deep convolutional network, stacking multiple convolutional layers allows units in deeper layers to have larger receptive fields, enabling them to capture more abstract and global features by efficiently constructing complex interactions from simple local ones.

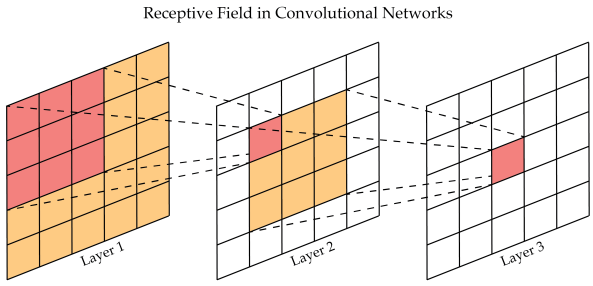

Figure 2.2: Visualization of growing receptive fields across CNN layers.

2.2. Parameter sharing

Another advantage of convolution is that the same kernel is applied across all spatial locations. We can say that the network has *tied weights*, because the values of the weights are the same everywhere. This means that instead of learning a separate set of weights for each location, the model learns only one kernel that is reused.

For example, a kernel that detects vertical edges will detect them anywhere in the image. This does not affect the runtime of forward propagation, still $O(k \times n)$ where k is the kernel size and the input is $m \times n$.

This greatly reduces the number of learnable parameters and improves generalization, since the same features are useful at multiple positions.

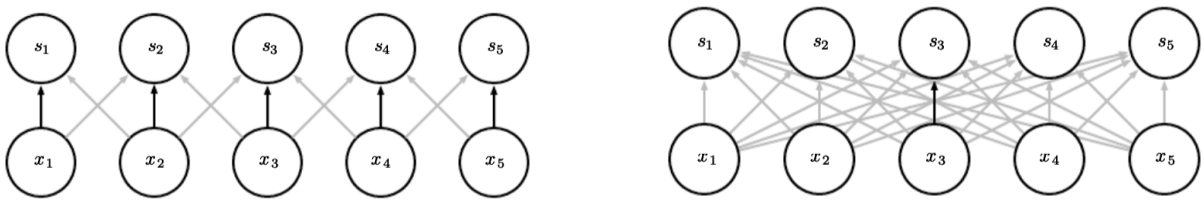

Figure 2.3: Parameter sharing: because of parameter sharing, the single parameter is used everywhere (left); in the other hand the parameter is used only once (right).

2.3. Equivariance to translation

The already discussed parameter sharing causes in convolution a property called *equivariance* to translation. Formally, a function f is equivariant to an operation T if, applying T to the input and then f, gives the same result as applying f first and then T to the output.

Convolution satisfies this property with respect to translations:

$$(K * I)(x + \Delta) = T_\Delta\big((K * I)(x)\big),$$

where T_Δ denotes a translation by Δ.

This means that if the input image is shifted, the output feature map shifts in the same way. In images, convolution produces a 2-D feature map where detected patterns shift consistently with the input, enabling parameter sharing across locations. This is useful for features like edges, which appear throughout an image, though in some cases, such as face recognition, different regions may require learning distinct features.

3 | Pooling

A standard layer in a convolutional network is typically composed of three stages. First, multiple convolutions are applied in parallel to generate linear activations. Next, these activations are passed through a nonlinear activation function, a step sometimes referred to as the *detector stage*. Finally, a *pooling operation* is applied to further transform and condense the output.

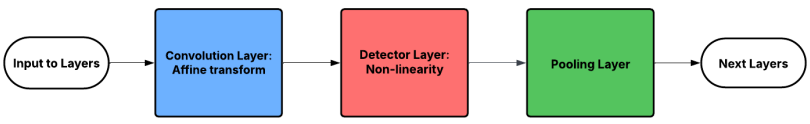

Figure 3.1: Data flow through a typical CNN block.

Pooling is a subsampling operation that has traditionally played a central role in CNNs. Its purpose is to reduce the spatial resolution of feature maps while retaining the most important information. The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network [2]. However, when the assumption does not hold, convolution and pooling may actually harm performance. Like any prior, these operations are only beneficial when their underlying assumptions are accurate: if a task requires preserving precise spatial information, excessive pooling can increase training error and lead to underfitting. For this reason, some architectures apply pooling selectively, on certain channels but not others, to balance the extraction of invariant features with the preservation of fine-grained spatial details.

3.1. Definition

Given an input feature map $X \in \mathbb{R}^{m \times n}$, pooling partitions X into non-overlapping regions and summarizes each region by a statistic. The two most common pooling operations are:

- **Max pooling**, which reports the maximum value within a rectangular neighborhood:

$$Y[i,j] = \max_{(p,q) \in \mathcal{R}_{ij}} X[p,q], \tag{3.1}$$

- **Average pooling**, which computes the mean value:

$$Y[i,j] = \frac{1}{|\mathcal{R}_{ij}|} \sum_{(p,q) \in \mathcal{R}_{ij}} X[p,q], \tag{3.2}$$

where $\mathcal{R}_{ij}$ denotes the receptive field corresponding to output element (i,j).

3.2. Motivation and role in CNNs

The central benefit introduced by pooling lies in the approximately **invariance to small translations** of the input.

Both max pooling and average pooling summarize local neighbourhoods in a way that makes the representation less sensitive to small shifts in the input. This invariance is particularly valuable in visual recognition tasks, when we care more about whether some feature is present than exactly where it is. For example, when recognizing whether an image contains a car, the network does not need to know the exact pixel-level positions of the wheels; it is enough to detect that there are four wheels, with two in the front and two in the back, regardless of small variations in their precise location.

It's also possible, if pooling over the output of separately parametrized convolutions, to make features able to learn *which* transformations to become invariant to.

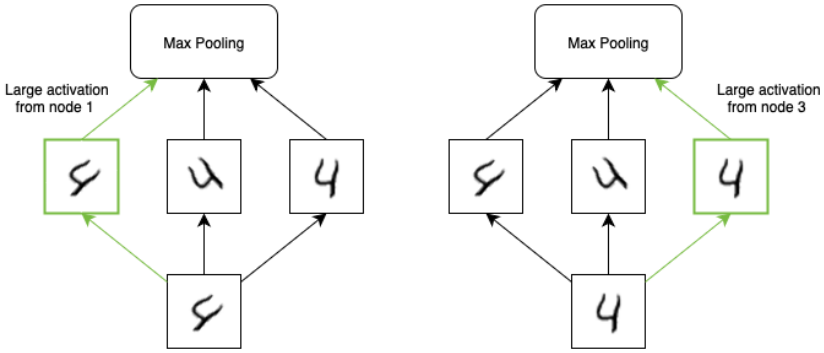

Figure 3.2: Example of filters that learn to become invariant to rotation.

Pooling also introduces several important benefits: reduces the spatial resolution of feature maps, which decreases the computational cost of subsequent layers, lowers memory requirements and improves the statistical efficiency of the network by enforcing more compact and robust representations.

Within the hierarchy of a CNN, pooling therefore plays a key role: early convolutional layers detect local features such as edges, and pooling combines them into progressively more abstract and stable patterns, supporting the network in building higher-level representations [2].

3.3. Modern alternatives

While max and average pooling have been widely used, more recent research has explored alternatives that either replace pooling entirely or refine it.

Springenberg et al., 2015 [6] investigated whether pooling is truly necessary for state of the art performance. They proposed an extremely simple architecture consisting only of convolutional layers, with dimensionality reduction achieved exclusively by **strided convolutions** (stride of 2). Despite the absence of pooling, normalization layers or complex activation functions, this homogeneous architecture was able to achieve competitive, and in some cases state of the art, performance on benchmark datasets such as CIFAR-10, CIFAR-100, ImageNet. Their findings demonstrated that small convolutional layers with stride can be sufficient to replace traditional pooling operators and raises important questions about the necessity of pooling in CNNs.

Traditional pooling operations either discard information (max pooling) or average it uniformly (average pooling), both of which may lead to a loss of useful detail. **SoftPool** (Stergiou et al., 2021 [7]) addresses this limitation by computing a weighted average, where the weights are proportional to the magnitude of the activations within each pooling region. This mechanism allows more information to be preserved, while still reducing the spatial resolution of the feature maps. Experimental evaluations have shown that SoftPool

improves classification accuracy on benchmarks such as ImageNet, CIFAR, and Kinetics, while introducing only minimal computational overhead. These results suggest that more informative pooling operators can provide benefits over max and average pooling and highlight the potential of adaptive downsampling methods in modern CNNs.

4 | Variants of the Basic Convolution Function

In the context of neural networks, we do not usually apply the standard convolution operation, as presented in Chapter 1. The convolution operation at the core of CNNs admits several useful modifications that expand its flexibility and efficiency. The variants of the basic convolution functions can differ slightly in how parameters are shared, how receptive fields are enlarged or how the computation is factorized.

4.1. Strided convolution

When dealing with images, software implementations usually work in batch mode, so they use as input and output of the convolution some 4-D tensors, with the first index into the different channels, the following two indices into the spatial coordinates of each channel and the fourth axis indexing different examples in the batch.

Assume we have 4-D kernel tensor K with element $K_{i,j,k,l}$ giving the connection strenght between a unit channel in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit. Assume our input consists of observed data v with element $V_{i,j,k}$ giving the value of the input within channel i at row j and column k. Assume our output consists of Z with the same format as V. If Z is produced by convolving K across V without flipping K, then:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n} \tag{4.1}$$

where the summation over l, m and n is over all values for which the tensor indexing operations inside the summation are valid.

In standard convolution, the kernel is applied at every possible location in the input. A simple modification consists of applying the kernel only at locations separated by a fixed step size, known as the *stride*. Formally, for stride s, the output of the convolution is defined only at positions spaced s units apart:

$$Z_{i,j,k} = c(K, V, s)_{i,j,k} = \sum_{l,m,n} [V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n}] \tag{4.2}$$

This reduces the spatial resolution of the output and combines the effects of convolution and subsampling. We already saw that *strided convolution* is often used as an alternative to pooling, section 3.3.

4.2. Zero-padding

Another common variant is related to the ability of *zero-padding*, where the input is padded with zeros around its border before applying convolution, to make it wider. Zero-padding allows control over the size of the output feature maps. Without padding, the spatial dimensions shrink after each convolution; with appropriate padding, the output can be kept at the same size as the input.

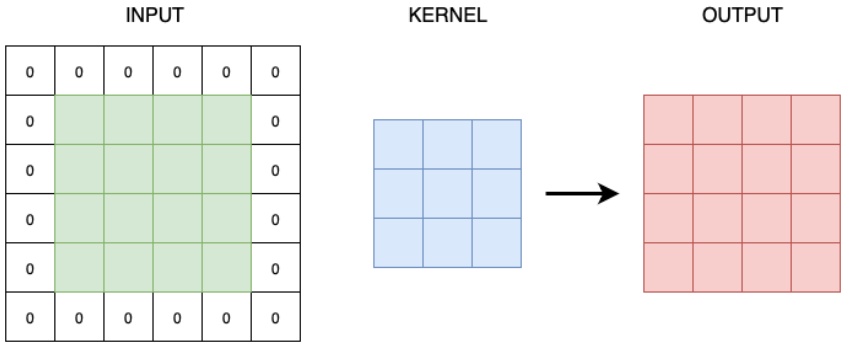

Figure 4.1: Example of zero padding equal to one.

The most common conventions are: **valid convolution** in which no padding is applied, so for a one-dimensional input of length m convolved with a kernel of size k the output length is $m - k + 1$; **same convolution** where padding is chosen so that the output has the same size as the input, i.e. m; **full convolution**, here padding is added so that the kernel can be applied at every possible shift, producing an output of size $m + k - 1$.

4.3. Unshared and Tiled Convolution

In a standard convolutional layer, the same kernel is applied across all spatial locations. This weight sharing greatly reduces the number of parameters. However, we can relax this assumption, leading to the concept of *locally connected layers*.

The most general case is the **unshared convolution** where each output unit has its own distinct set of weights rather than sharing the same kernel across all positions.

Formally, let $x \in \mathbb{R}^n$ be a one-dimensional input and suppose we use windows of size k to produce an output $y \in \mathbb{R}^m$. In a standard convolution, the same kernel $w \in \mathbb{R}^k$ is used at all locations. In the unshared case, each position i has its own kernel $w^{(i)} \in \mathbb{R}^k$, yielding:

$$y_i = \sum_{j=1}^{k} w_j^{(i)} \, x_{i+j-1}, \quad i = 1, \ldots, m. \tag{4.3}$$

This increases the number of parameters from k to $m \times k$, and in higher dimensions from k^2 to $m \times n \times k^2$, making the model far less efficient. Locally connected layers are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space.

Between the two extremes of full parameter sharing (*standard convolution*) and no sharing (*unshared convolution*), there exists an intermediate approach known as **tiled convolution**. In this variant, weights are shared, but only among a restricted set of positions. For example, one might specify a tiling factor t, such that every t consecutive locations reuse the same kernel, after which a new set of weights is introduced.

Formally, let $w^{(1)}, \ldots, w^{(t)}$ denote t different kernels. At position i, the kernel used is determined by $w^{(i \bmod t)}$, i.e.

$$y_i = \sum_{j=1}^{k} w_j^{(i \bmod T)} \, x_{i+j-1}. \tag{4.4}$$

This design relaxes the strong prior of full translation equivariance while still controlling the number of parameters compared to the fully unshared case. It allows the model to learn features that vary smoothly across space, without requiring each location to have an entirely independent set of weights.

Locally connected and tiled convolutional layers exhibit a particular interaction with max pooling: their detector units rely on distinct filters. When these filters capture different transformed versions of the same feature, the subsequent max pooling operation produces units that are invariant to such transformations. (Figure 3.2).

4.4. Separable convolution

A kernel can sometimes be factorized into simpler components. For example, a 2D convolution with a $k \times k$ kernel can be replaced by two 1D convolutions ($k \times 1$ followed by $1 \times k$). This is known as **spatially separable convolution** and it reduces computation when the factorization is exact or a good approximation.

An important extension is the *depthwise separable convolution*, introduced in MobileNets, 2017 [3]. This factorizes convolution into two steps:

1. a *depthwise convolution*, where each filter is applied independently to a single input channel;
2. a *pointwise convolution*, using 1×1 kernels to combine information across channels.

This decomposition greatly reduces the number of parameters and multiplications, enabling efficient CNNs suitable for mobile and embedded devices.

5 | Applications of Convolution to Different Data Types

Convolutional networks are not limited to images. The same mathematical operation can be applied to different data types, depending on their structure and dimensionality. In this section we review the most common cases.

5.1. One and Two dimensional data

The simplest applications of convolution are to one-dimensional and two-dimensional arrays. In the one-dimensional case, typical of time series or audio signals, convolutional filters slide along the temporal axis, detecting short-term dependencies, periodicities or characteristic frequency components. This makes 1D convolution suitable for speech recognition, time-series forecasting and other tasks where local temporal context is important.

In the two-dimensional case, characteristic of image data, the input is arranged in a grid of pixels (with additional channels for color). Here, convolutional kernels learn to capture local spatial features such as edges, textures and corners. By stacking multiple layers, the network progressively builds more complex and abstract representations, forming the foundation of modern computer vision systems.

5.2. Higher-dimensional data: video and volumetric analysis

Convolution naturally extends to three or more dimensions. A key application is in video understanding, where the input has two spatial dimensions and one temporal dimension. In this case, 3D convolutional kernels are used to jointly model motion and appearance, enabling the detection of spatio-temporal patterns such as moving objects, gestures, or human actions.

Ji et al. [5] proposed one of the first effective 3D CNN architectures for human action recognition. Their model performs convolutions over both spatial and temporal dimensions by applying 3D kernels to small cubes of consecutive video frames. This design captures motion features that are not accessible to standard 2D CNNs operating on individual frames.

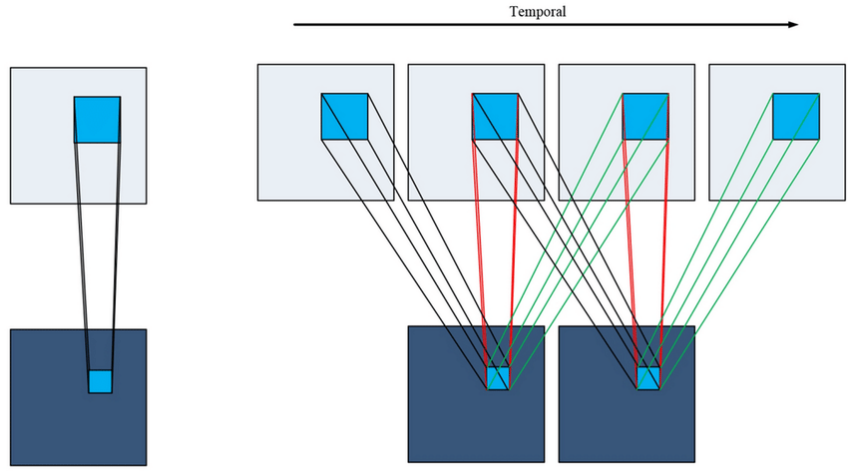

Figure 5.1: Comparison between 2D and 3D CNN

Formally, the value at position (x, y, z) on the jth feature map in the ith layer is given by

$$v_{ij}^{xyz} = \tanh\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr}\, v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right), \tag{5.1}$$

where (P, Q, R) are the spatial and temporal dimensions of the kernel, w_{ijm}^{pqr} are the kernel weights, and b_{ij} is the bias.

The proposed architecture was evaluated on large-scale datasets such as TRECVID surveillance videos and the KTH human action dataset. The results showed that 3D CNNs outperform traditional frame-based (2D CNNs) methods and handcrafted spatio-temporal features, particularly in real-world environments with cluttered backgrounds and viewpoint variations. Interestingly, the gains were more significant when the number of positive training examples was limited, highlighting the statistical efficiency of learned spatio-temporal features.

6 | Neuroscientific Principles for CNNs

Convolutional networks are perhaps the greatest success story of biologically inspired artificial intelligence. Though convolutional networks have been guided by many other fields, some of the key design principles of neural networks were drawn from neuroscience [2].

6.1. Receptive fields in the visual cortex

Hubel and Wiesel [4] conducted pioneering experiments on the visual cortex of cats and monkeys. They showed that neurons in early visual system responded selectively to specific regions of the visual field. These regions, known as *receptive fields*, cover only a small portion of the visual input. Moreover, many neurons respond preferentially to simple patterns such as oriented edges or bars of light.

In the perspective of deep learning, we can consider a very simplified view of brain function. According to this view, that early visual system identified is called **V1**, also known as the *primary visual cortex*, located in the back of the head.

This suggested a hierarchical organization: lower-level neurons detect simple visual features, while higher-level neurons combine them to represent more complex structures.

6.2. Translation to convolutional models

Convolutional networks reproduce several key properties observed in the primary visual cortex. V1 is organized as a **spatial map**: neurons respond to specific regions of the visual field, preserving the topographic organization of the retina. CNNs can capture this property by defining features on two-dimensional maps, where each unit corresponds to a localized region of the input.

Second, V1 contains many **simple cells**, whose activity can be approximated as a linear function over a small receptive field. The detector units in CNNs are designed to emulate these cells, applying learned linear filters to local regions of the input.

Finally, V1 includes also **complex cells** which are insensitive to small shifts in the position of features and sometimes to changes such as lighting conditions. This biological mechanism inspires the use of pooling in CNNs.

It is generally believed that the same basic principles of V1 apply to other areas of the visual system. As far as we are concerned, the main role of these other anatomical regions is just to carry the signal to V1. When viewing an object, information flows from the retina, through a brain region called the *lateral geniculate nucleus* (LGN), to V1, then onward to V2, then V4, then to the last layer, the *inferotemporal cortex* (IT). In our vision, as we move deeper into the brain, a basic strategy of detection followed by pooling is repeated.

This alternation of detection and pooling observed across the multiple areas of the visual system (LGN, V1, V2, V4, IT) is mirrored in the layered structure of CNNs. As deeper layers combine the outputs of earlier ones, the network builds progressively more abstract features, similar to how the IT encodes complex objects and shows invariance to many transformations. Empirical studies even demonstrate that CNN activations can predict firing rates in IT and achieve performance comparable to humans on rapid object recognition tasks (DiCarlo, 2013 [1]).

6.3. Limitations of the analogy

Although CNNs were inspired by the described visual system, there are many important differences. Some of these differences are well established in neuroscience, while others remain open questions, as many aspects of biological vision are not yet fully understood.

A major difference concerns the input itself. Human vision is not uniformly high resolution: most of the retina provides only coarse information, while a small central region called the **fovea** captures fine detail. To perceive a full scene in high resolution, the brain integrates successive glimpses obtained through rapid eye movements known as **saccades**. Convolutional networks instead, usually process entire images at full resolution in a single pass.

Another difference is that the human visual system operates in conjunction with other senses such as hearing and is influenced by factors like mood, attention and prior knowledge. CNNs instead are purely visual systems that lack such multimodal integration. Moreover, the biological visual system does much more than object recognition. It interprets complex scenes, reasons about the relationships between objects, and processes 3-D geometric information necessary for action and navigation. Convolutional networks have been applied to some of these tasks, as described in section 5.2, but such applications are still in their infancy.

Feedback is another critical distinction: even in early cortical areas such as V1, activity is strongly shaped by top-down signals from higher visual areas. Although feedback connections have been studied in artificial neural networks, they have not yet produced consistent advantages comparable to those observed in biology. Finally, while firing rates in IT resemble the features extracted by deep convolutional layers, the intermediate computations may differ substantially. The brain probably uses more complex activation and pooling mechanisms than simple linear filters.

7 | Conclusion and Outlook

Convolutional Neural Networks have transformed machine learning by combining mathematical structure, computational efficiency and inspiration from neuroscience. Through convolution, they exploit local connectivity, weight sharing and translation equivariance, while pooling introduces useful invariances and reduces complexity. Variants of the basic convolution function extend their flexibility and modern alternatives to pooling demonstrate the adaptability of these models to evolving needs. Applications across one, two and three-dimensional data, highlight the generality of convolution as a tool for representation learning. The neuroscientific parallels further reinforce the conceptual foundation of CNNs, even though many differences remain between artificial models and biological vision.

Looking forward, convolutional networks are likely to remain central in deep learning, though their dominance is being challenged by newer architectures such as attention based models and transformers. Future directions may involve hybrid models that combine convolution with attention or other mechanisms, better approximations of biological processes such as foveation and feedback and broader applications beyond vision, including multimodal learning and scientific data analysis.

In summary, CNNs illustrate how mathematical principles, engineering solutions and biological insights can converge into models that not only achieve state of the art performance but also continue to inspire new generations of machine learning research.

Bibliography

[1] J. J. DiCarlo, D. Zoccolan, and N. C. Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415–434, 2012. doi: 10.1016/j.neuron.2012.01.010.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. URL `https://arxiv.org/abs/1704.04861`.

[4] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

[5] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1): 221–231, 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.59.

[6] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2015. URL `https://arxiv.org/abs/1412.6806`.

[7] A. Stergiou, R. Poppe, and G. Kalliatakis. Refining activation downsampling with softpool. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10371–10380, 2021. doi: 10.1109/ICCV48922.2021.01022. URL `https://arxiv.org/abs/2101.00440`.