Convolutional Neural Networks

Enrique Romero

Computational Intelligence
Master in Artificial Intelligence

Soft Computing Group
Computer Science Department
Universitat Politècnica de Catalunya, Barcelona, Spain

Outline

Outline

A Bit of History

Convolutional Neural Networks (CNNs) are very old models:

- Neocognitron [Fukushima, 1980] incorporated most of the elements in modern CNNs, but with a different approach
- Time-Delay Neural Networks
  [Lang and Hinton, 1988, Lang et al., 1990] are one-dimensional versions of CNNs applied to time series, trained with back-propagation
- Convolutional (Neural) Networks are described for the first time in [LeCun et al., 1989]

A Bit of History

CNNs were some of the first neural networs to be used in commercial applications:

- AT&T developed a CNN for reading checks [LeCun et al., 1998], and by the end of the 1990s this system was reading over 10% of all checks in the USA
- Microsoft deployed several OCR and handwritting recognition systems based on CNNs [Simard et al., 2003]
- More information: [LeCun et al., 2010]

More recently, CNNs have attracted much attention because the winner models of several important contests were based on CNNs (the first one was the 2012 ImageNet object recognition challenge `http://www.image-net.org/challenges/LSVRC/2012/results.html`)

Since then they are widely used, mainly for image recognition tasks

Outline

Main Ideas and Motivation

A (simple) definition: **CNNs** are **Feed-forward Neural Networks** that compute, in at least one of the layers, a **convolution** instead of a matrix multiplication

The convolution operation implicitly leads to:

- **Sparse interactions** (sparse connectivity)
- **Weight sharing**

with the aim of obtaining **translation invariance**: be able to find a pattern in any place of the input

Typically, the convolution operation is combined with other operations, such as **non-linear transformations** and **pooling**

Outline

Definition of Convolution

An example of convolution:

Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to "combine" the results is to merge the convolution kernels. The new image convolution kernel looks like this:

-1	-1	-1
-1	8	-1
-1	-1	-1

Below result I got with edge detection:

More examples at
`https://www.aishack.in/tutorials/image-convolution-examples`

Definition of Convolution

Definition of **convolution** of two functions F, G:

$$C(t) = (F * G)(t) = \int_{-\infty}^{\infty} F(x)\ G(t-x)\ dx$$

We can see the convolution $(F * G)(t)$ as a weighted average (similar to the expectation) of a function reflected and shifted by t

Definition of **cross-correlation** of two functions F, G:

$$C(t) = (F * G)(t) = \int_{-\infty}^{\infty} F(x)\ G(t+x)\ dx$$

We can see the cross-correlation $(F * G)(t)$ as a measure of similarity of two series as a function of the displacement of one relative to the other

This is also known as a sliding inner-product, and it is commonly used for searching short, known feature (F) in a long signal (G)

Definition of Convolution

In CNNs we use the **discrete cross-correlation**

For example in 2D CNNs we will define:

$$C(n,m) = (K * I)(n,m) = \sum_i \sum_j K(i,j) I(n+i, m+j)$$

where

- I is the **input** (an image, for example)
- K is the **kernel**, **feature detector** or **filter** (the weights)
- C is the **feature map** (the output)
- The sum is defined over all valid values

Since the inner product is a measure of similarity, C **will larger in the positions where** K **and** I **are similar**

Definition of Convolution

Should we rename CNNs as "Cross-correlation NNs"? The convolution is the same as the cross-correlation with a flipped kernel, since (by applying simple change of variables)

$$\text{Conv}(F, G, t) = \int_{-\infty}^{\infty} F(x)\ G(t-x)\ dx = \int_{-\infty}^{\infty} F(t-y)\ G(y)\ dy$$

$$\text{CrossC}(F, G, t) = \int_{-\infty}^{\infty} F(x)\ G(t+x)\ dx = \int_{-\infty}^{\infty} F(y-t)\ G(y)\ dy$$

and therefore

$$\text{Conv}(F, G, t) = \text{CrossC}(\bar{F}, G, t)$$

with $\bar{F}(z) = F(-z)$

Outline

Viewing the Convolution in a Standard MLP

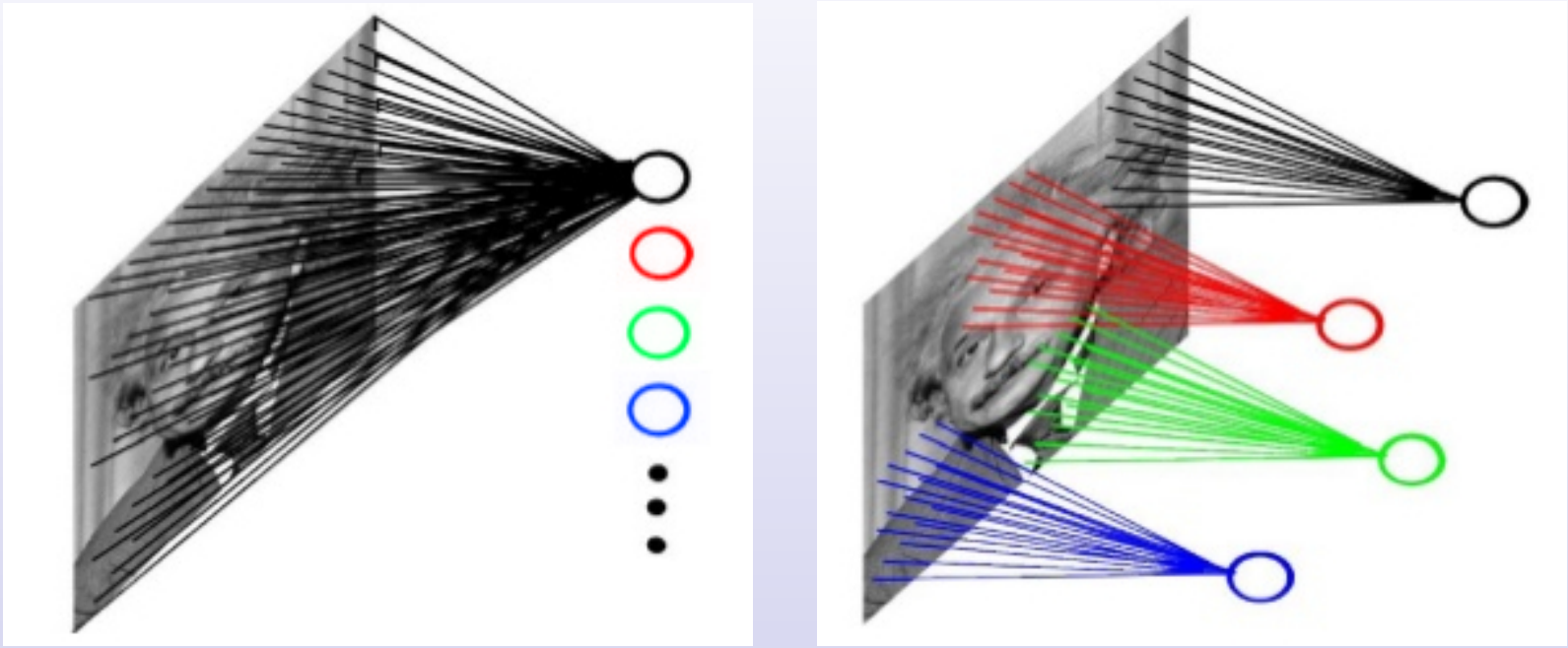


Figure 1: Representation of classical MLP (left) and CNN (right) hidden units

With a convolution, the **same weights** are used to compute output values in **different and small parts** of the input

Viewing the Convolution in a Standard MLP

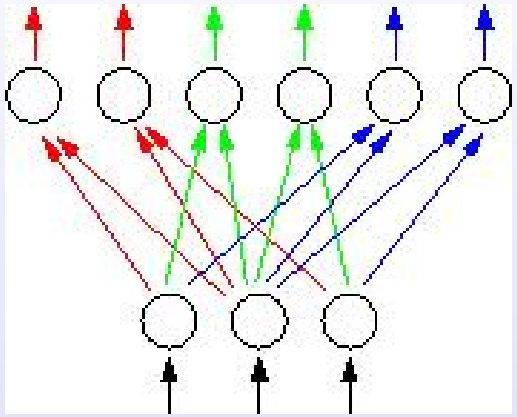


Figure 2: Viewing the convolution in a standard MLP

Note that

- Related to **translation invariance**, we have
  - **Sparse interactions** (sparse connectivity)
  - **Weight sharing**
- We can have **many convolutions in the same layer**
- There are **much less parameters** than a standard fully connected network with the same number of hidden units

Viewing the Convolution in a Standard MLP

A convolutional operation is similar to a **local receptive field** with shared weights

The previous figure shows that **discrete convolution can be seen as a multiplication by a matrix with restrictions**:

- Equal weights in different units
- Zeros for several weights

Although it is not used in practice (it would be very inefficient) it allows to understand it as a classical Multilayer Perceptrons (MLPs) and **apply standard techniques (back-propagation, for example) in a natural way**

Obviously, it needs to **reshape** the input data (e.g., an image) to a vector

Outline

Working with Channels

In real data sets, there are additional dimensions where we do not want to apply the convolution operation:

- 1D: Multi-dimensional time series (for example, animations of "skeletons", where the data is described by the angles of each joint in the skeleton)
- 2D: Color images (each pixel is a rgb vector)
- 3D: Color volumetric data (for example, medical rgb CT scans)

These additional dimensions are usually called **channels** (also referred to as the **depth** of the input)

How does the convolution operate with channels?

Working with Channels

For example, in 2D color images (3 channels), we have:

- The input is a $N \times M \times 3$ tensor
- Every filter K_f is a $A \times B \times 3$ tensor
- The output of the convolution of the input image I and the filter K_f is a $N \times M \times 1$ tensor (a matrix), where each component is (we sum up along the channels):

$$C_{K_f}(n,m) = \sum_{i=1}^{A} \sum_{j=1}^{B} \sum_{k=1}^{3} K_f(i,j,k) I(n+i, m+j, k)$$

- Every filter "outputs an image of just one channel": If we have F filters, the output of the convolutional layer is a $N \times M \times F$ tensor, where the third component has the convolutions of the image with every filter:

$$C(n,m,f) = C_{K_f}(n,m)$$

Working with Channels

Therefore, **the outputs of a convolutional layer, in turn, are also reshaped in a structure with channels**, so that it can be used as the input of a new convolutional layer:

- The number of dimensions of the original image ($N \times M$) is constant (approximately, see below) between adjacent convolutional layers
- The number of output channels of a convolutional layer is the number of filters of that layer (F)

This **allows to construct deep CNNs in a natural way**

Outline

Working with Bias

We can add some bias terms to the convolution

How do they operate?

Typically, we will have one bias per channel in the output and one bias per filter, shared across all locations in the convolution:

$$C_{K_f}(n,m) = \sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{3}[K_f(i,j,k)I(n+i,m+j,k) + b_f(k)]$$

Outline

Parameters of Convolutions

The first parameter is, obviously, **the size of the filter** $A \times B$

With the previous definition of convolution, the size of the output of the convolution of a $N \times M$ image with an $A \times B$ filter is exactly $(N - A + 1) \times (M - B + 1)$, since only valid positions can be used to compute the convolution

We may want:

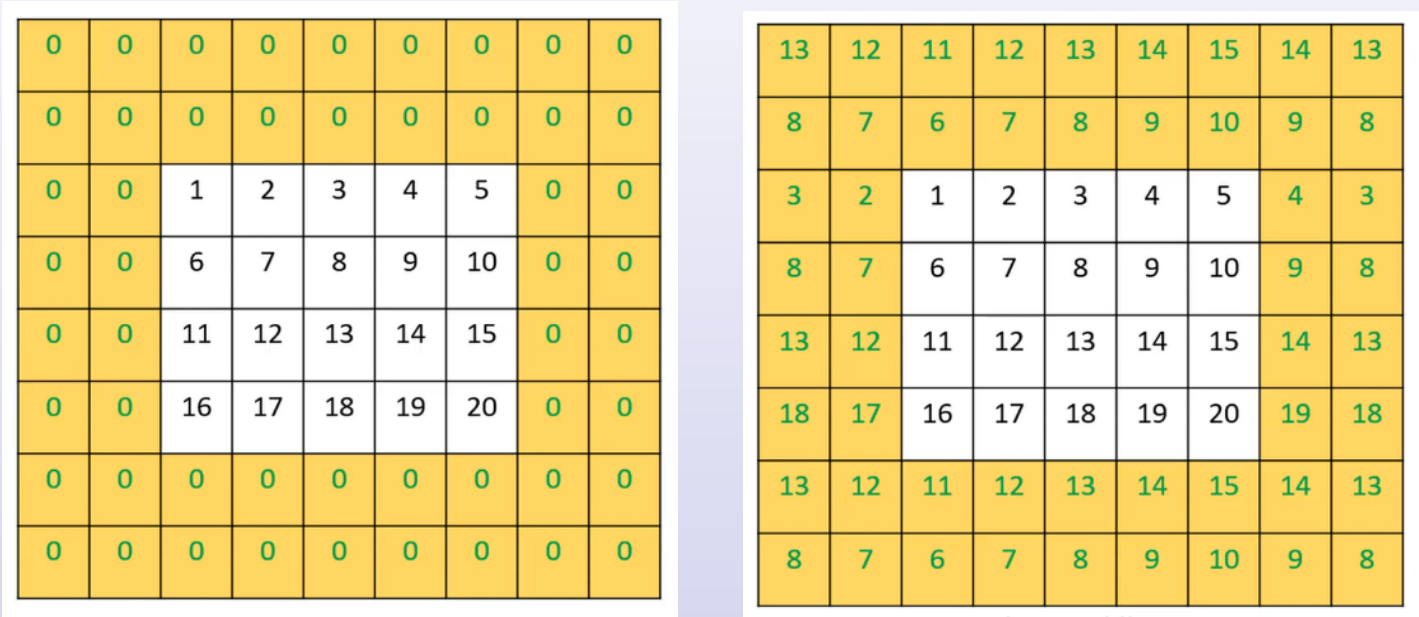
- On the one hand, to have the possibility to obtain outputs with the same size than the original one by adding extra values at the borders of the image (**Padding**)
- On the other hand, reduce the computational cost by removing (or not computing) some of the outputs of the convolution (**Stride**)

Parameters of Convolutions: Padding

The most common settings for Padding are

- When Padding = *valid*, the convolution is only allowed to visit positions where the entire filter is entirely within the image (and therefore reducing the size of the output)
- When Padding = *same*, extra values are added at the borders so that the size of the output is equal to the input (every pixel contributes to the same number of convolutions):
  - Zero-padding: adds zeros at the borders
  - Reflection-padding (Mirror-padding): values outside the borders are obtained by mirror-reflecting the image across the border

Parameters of Convolutions: Padding



0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8

Figure 3: Zero-padding (left) and Reflection-padding (right)

Parameters of Convolutions: Stride

The easiest and most common way to reduce the outputs of a convolution is to sample only every s pixels in each direction

It is a **downsampled convolution**:

$$C(n,m) = \sum_i \sum_j K(i,j) I(n + i \times s, m + j \times s)$$

The parameter s is the *stride* of the downsampled convolution

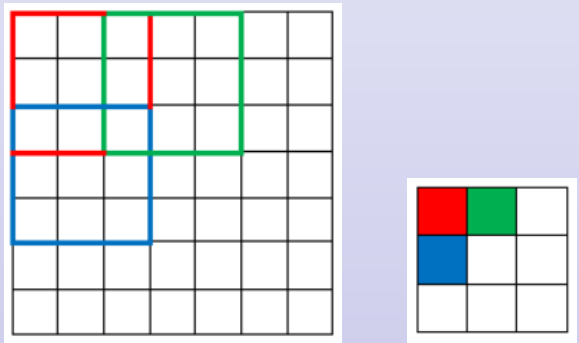


Figure 4: Convolution with Stride = 2

Some information is lost, but the computations are cheaper

Outline

Other Types of Convolutions

Apart from the standard convolution already explained, we can find other types of convolutions:

- Deformable convolution, more robust to geometric deformations
- Steerable convolution, more robust to linear transformations
- etc

More details at [Li et al., 2022]

Outline

Additional Operations and Layers in CNNs

Note that CNNs can handle **inputs of variable size**: applying each filter at different positions depending on the size of the input (similar to the stride parameter)

CNNs may have other types of operations and layers:

- Non-linearities: apply a non-linear transformation to the output of the convolution, such as
  - logistic, hyperbolic tangent
  - ReLU and its variations: leaky ReLU, PReLU, ELU, Swish, Mish,... (see [Li et al., 2022])
- **Pooling layers** (see next slide)
- Fully connected layers: standard layers (typically on top of the convolutional layers)

Additional Operations and Layers in CNNs: Pooling layers

Pooling layers **replace the output of the network at a certain rectangle with a summary statistic of its contents**

The most common pooling functions are:

- Max-pooling: the output is the maximum value
- Average-pooling: the output is the mean value

Parameters of the pooling operation:

- The size of the rectangle (typically 2×2)
- The stride (typically 2)

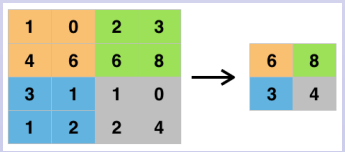


Figure 5: Max-pooling of size 2×2 and stride $= 2$

Additional Operations and Layers in CNNs: Pooling layers

Pooling layers are used to **reduce the spatial size of the representation and the number of parameters**, thus reducing the amount of computation in the network

For example, with a pooling of size of 2×2 and stride 2 the number of elements is reduced to 25% of its original size

Additionally, it makes the representation approximately invariant to small variations of the input (for example, if we translate the input by a small amount, the change in the values of the outputs will be small)

Outline

Typical Architectures of CNNs

In summary, in a CNN we can find:

- Convolutional layers
- Non-linearity layers (transformations of the convolution)
- Pooling layers
- Fully connected layers

Note that **we have separated the aggregation function (the convolution) and the activation function** of standard MLPs

Typically:

- A pooling layer is inserted after several Convolutional + Non-linearity layers (**deep feature learning/extraction**)
- Fully connected layers are the output layers of the network (typically for **classification/discrimination**)

Typical Architectures of CNNs

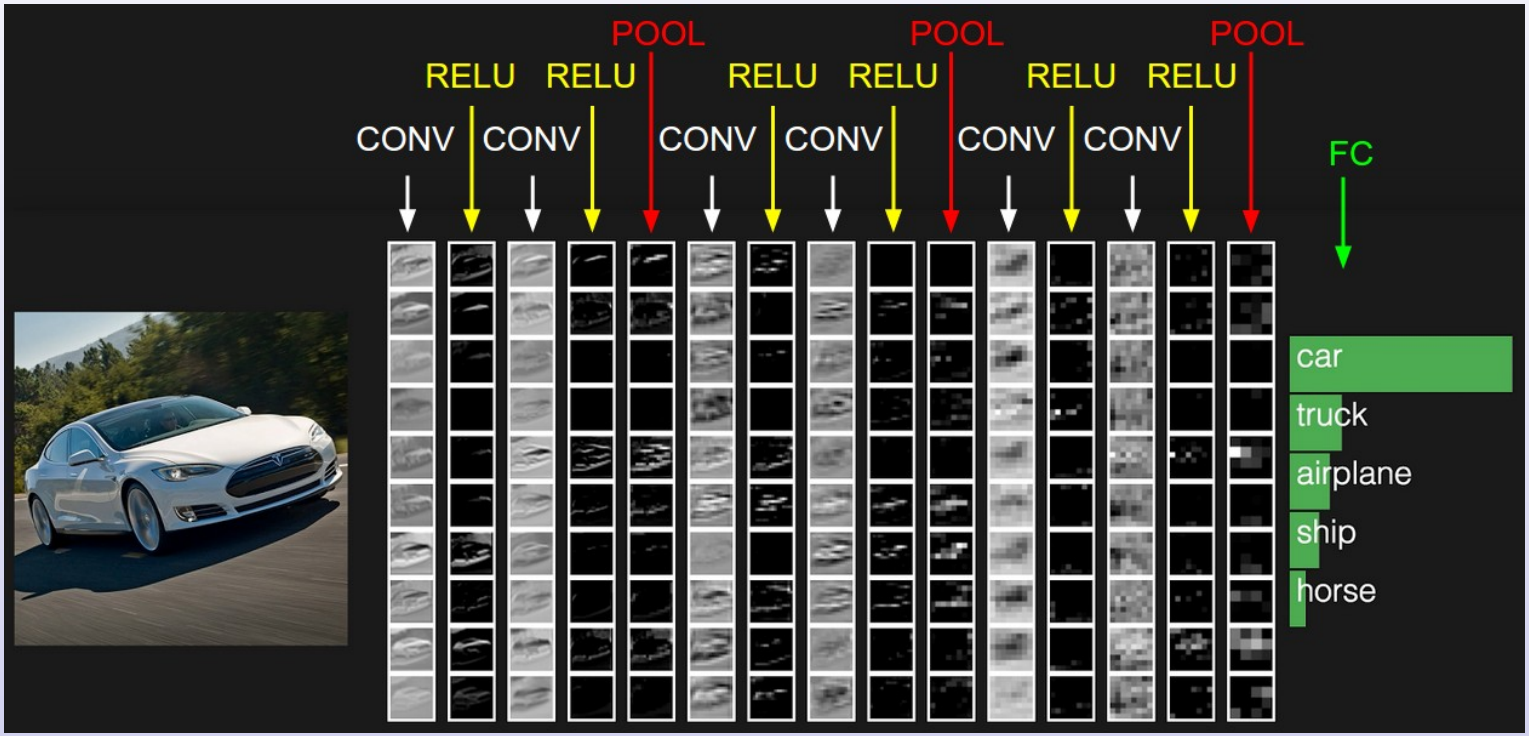


Figure 6: Typical architecture of a CNN (layers)

Typical Architectures of CNNs

What about the sizes? A common practice is to divide the size of the inputs by a factor and multiply the depth by another factor, forming a pyramid

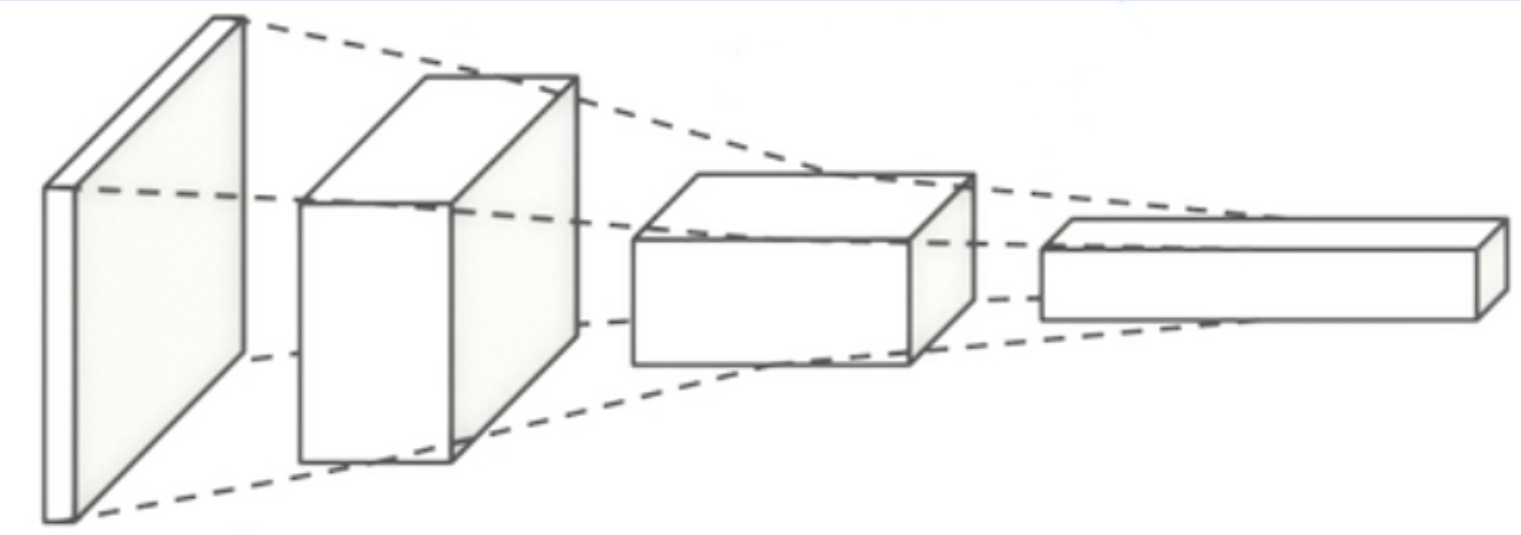


Figure 7: Typical architecture of a CNN (sizes)

Several Famous Architectures of CNNs

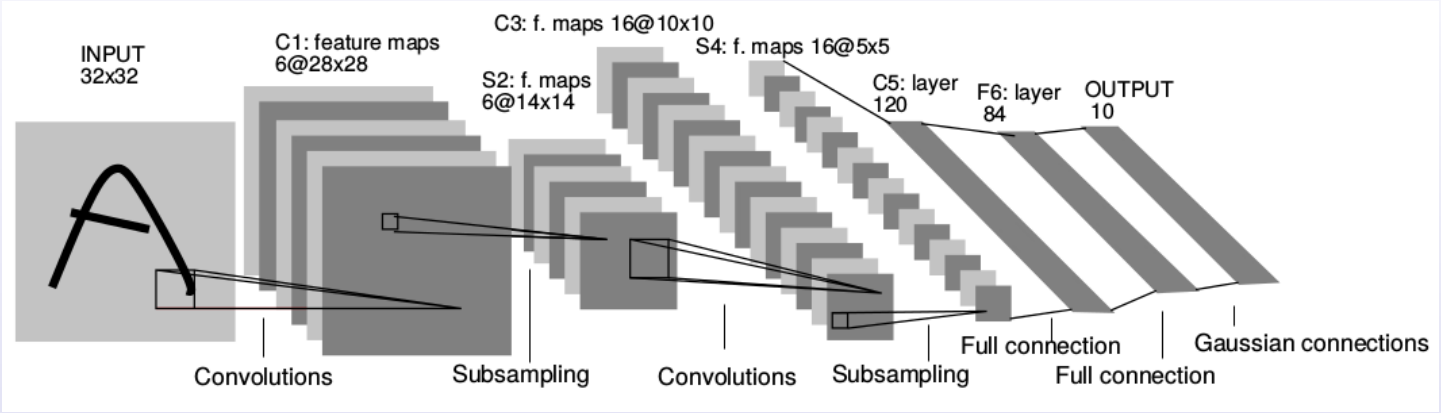


Figure 8: LeNet-5 architecture [LeCun et al., 1998]

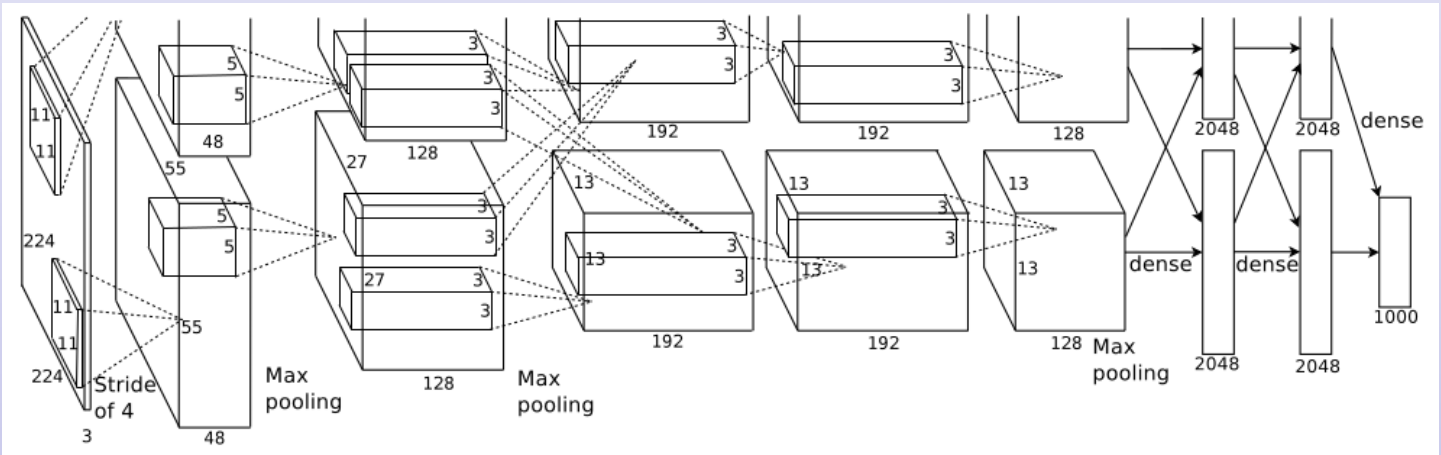


Figure 9: AlexNet architecture [Krizhevsky et al., 2012]

Several Famous Architectures of CNNs

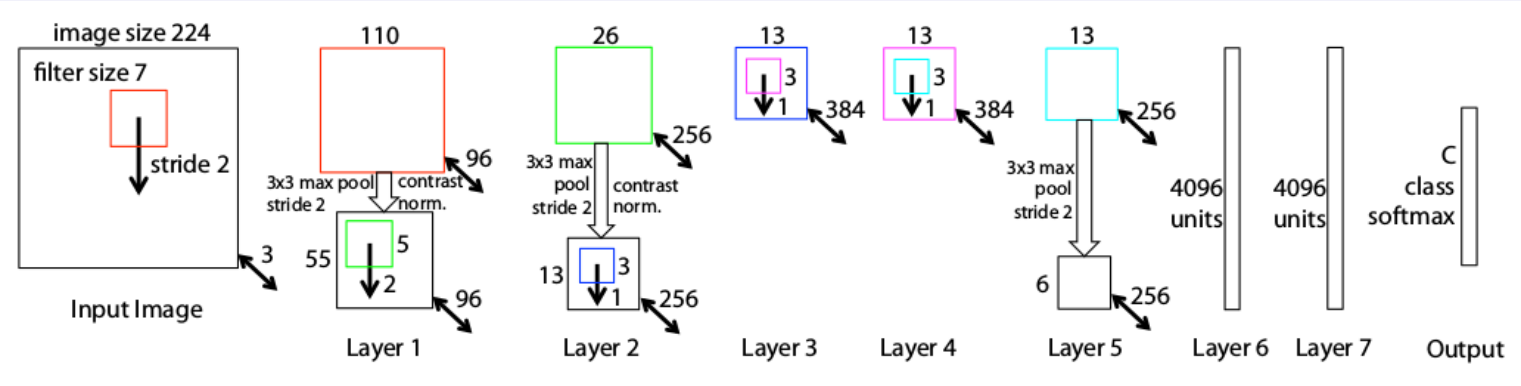


Figure 10: ZFNet architecture [Zeiler and Fergus, 2014]

More famous CNNs:
`http://cs231n.github.io/convolutional-networks`

And many more (only for Image recognition, until 2018!)

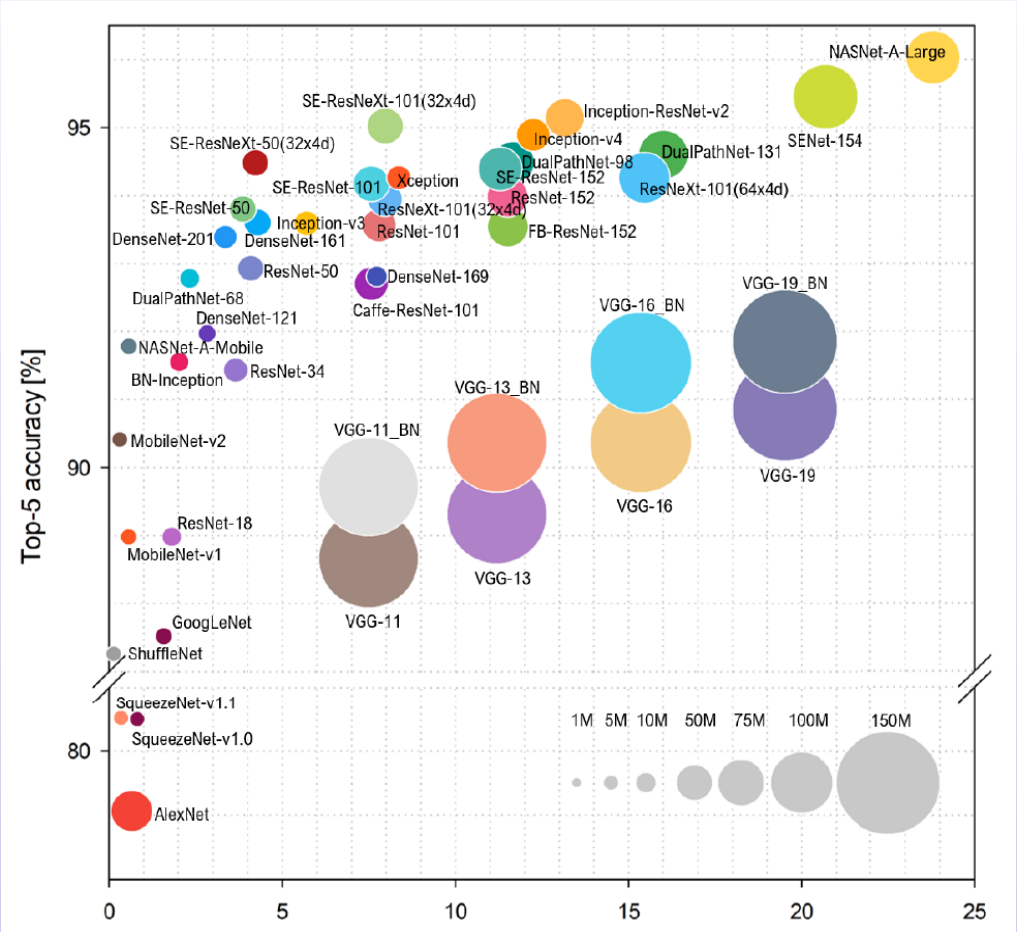


Figure 11: [Bianco et al., 2018]

Outline

Training CNNs with Back-Propagation

Since CNNs can be viewed as classical FNNs (see above), back-propagation can be easily applied to CNNs:

- For shared weights in convolutional layers, the derivative is the sum of the back-propagated derivatives, which can also (after a simple transformation) be seen also as a convolution
- For non-linearities, the derivative back-propagates as usual
- For pooling layers, the derivative is back-propagated according to the type of pooling and forward propagations (max-pooling only propagates the derivative of the maximum element, etc)
- For fully connected layers, the derivative back-propagates as usual

Outline

What Make CNN Work?

Basically, the joint interaction of a number of ingredients:

- Good initialization of the weights [Glorot and Bengio, 2010]
- Non-saturated activation functions, such as Rectified Linear Units (ReLUs) $f(x) = max(0, x)$ (or any of its variants) [Nair and Hinton, 2010, Glorot et al., 2011]
- Adaptive learning rates (Adam, RMSProp, Adagrad,...) [Kingma and Ba, 2015, Tieleman and Hinton, 2012, Duchi et al., 2011]
- Strong regularization techniques, such as dropout [Srivastava et al., 2014] or other tricks, such as batch normalization [Ioffe and Szegedy, 2015]
- High performance resources (GPUs, supercomputers)
- A large set of labeled examples
- Data where the convolution operation make sense
- etc

What Make CNN Work?

A comparative analysis of several CNN architectures from different points of view:

- Accuracy
- Number of parameters
- Memory usage
- Computational complexity
- Inference time

can be found in [Bianco et al., 2018]

That's it!

Bibliography

- Bianco, S., Cadene, R., Celona, L., and Napoletano, P. (2018). Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access*, 6:64270–64277.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159.
- Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36:193–202.
- Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256.
- Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.
- Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32th International Conference on Machine Learning*.
- Kingma, D. P. and Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 24, pages 1106–1114. MIT Press.
- Lang, K. J. and Hinton, G. E. (1988). The Development of the Time-delay Neural Network Architecture for Speech Recognition. Technical Report CMU-CS-88-152, Carnegie-Mellon University.
- Lang, K. J., Waibel, A. H., and Hinton, G. E. (1990). A Time-delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks*, 3(1):23–43.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

Bibliography

- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional Networks and Applications in Vision. In *International Symposium on Circuits and Systems*, pages 253–256.
- Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33:6999–7019.
- Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *27th International Conference on Machine Learning*, pages 807–814.
- Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *International Conference on Document Analysis and Recognition*, pages 958–963.
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.
- Tieleman, T. and Hinton, G. E. (2012). Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*.
- Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *European Conference on Computer Vision (Lecture Notes in Computer Science 8689)*, pages 818–833.