

Convolutional Neural Networks (CNNs) : A Complete Guide

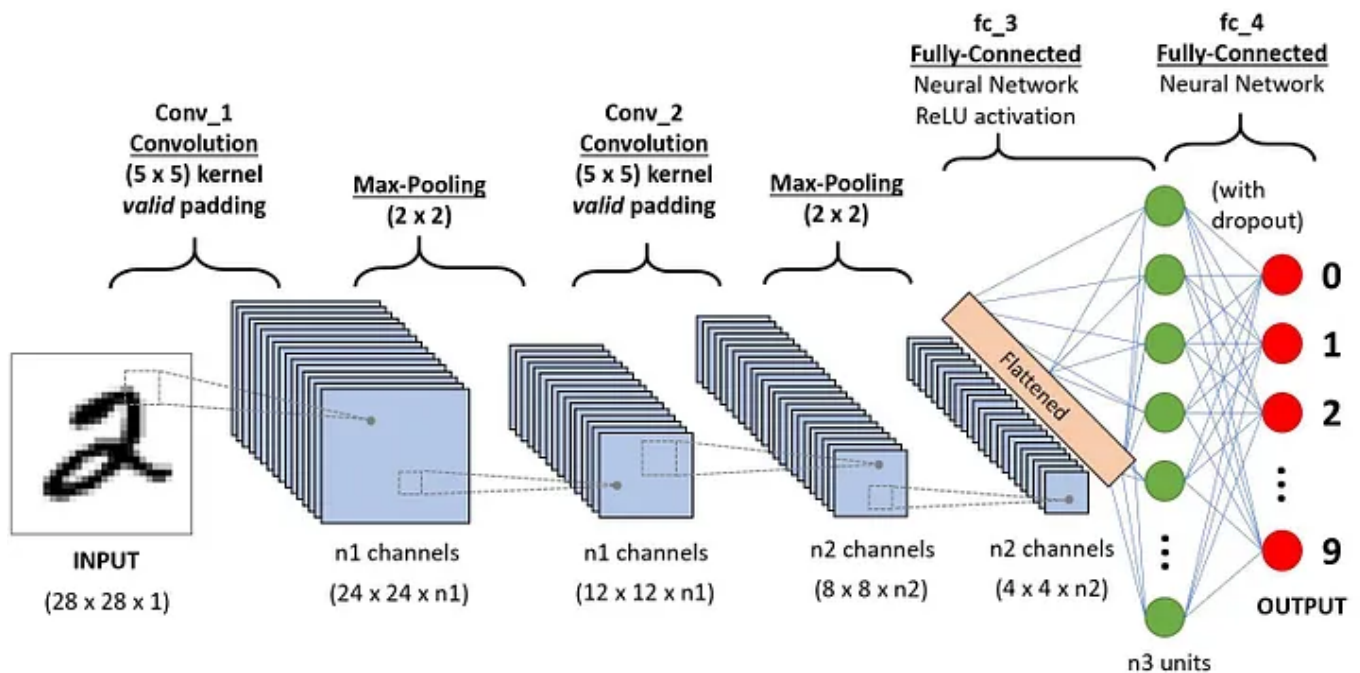


Alejandro Ito Aramendia · Follow

8 min read · Jan 1, 2024



133



A typical CNN architecture.

Contents

- [Introduction](#)

- Convolution Layer
- Pooling Layer
- Fully Connected (FC) Layer
- Summary.

Introduction

Convolution neural networks are fundamental for image analysis.

They are crucial for interpreting visual data, whether it's for **image classification**, **object detection** or even **medical image analysis**.

Following a unique architectural design, CNNs are a special type of neural network composed of **three** primary layers: the convolutional layer, the pooling layer and the fully connected layer.

- **Convolutional layer:** Using filters or kernels, this layer finds local patterns and features from the input image.
- **Pooling layer:** This layer downsamples the spatial dimensions of the input, reducing the network's computational complexity and increasing its ability to recognise patterns regardless of scale and position.
- **Fully Connected layer:** This layer integrates previously extracted features into a traditional neural network, in efforts to learn global relationships and generate task-specific outputs.

By implementing multiple convolutional and pooling layers, the network is able to learn deeper and more abstract features in regards to the input

image. Following these layers with a fully connected layer, the network is able to learn global dependencies, as well as complex patterns through non-linear activation functions.

Convolution Layer

As summarised earlier, the convolutional layer uses a **kernel** or **filter** (they are the same thing) to find and extract **local** features of an image. In modern CNNs, it is common to use multiple convolutional layers with distinct filters in order to capture a wider range of features and patterns.

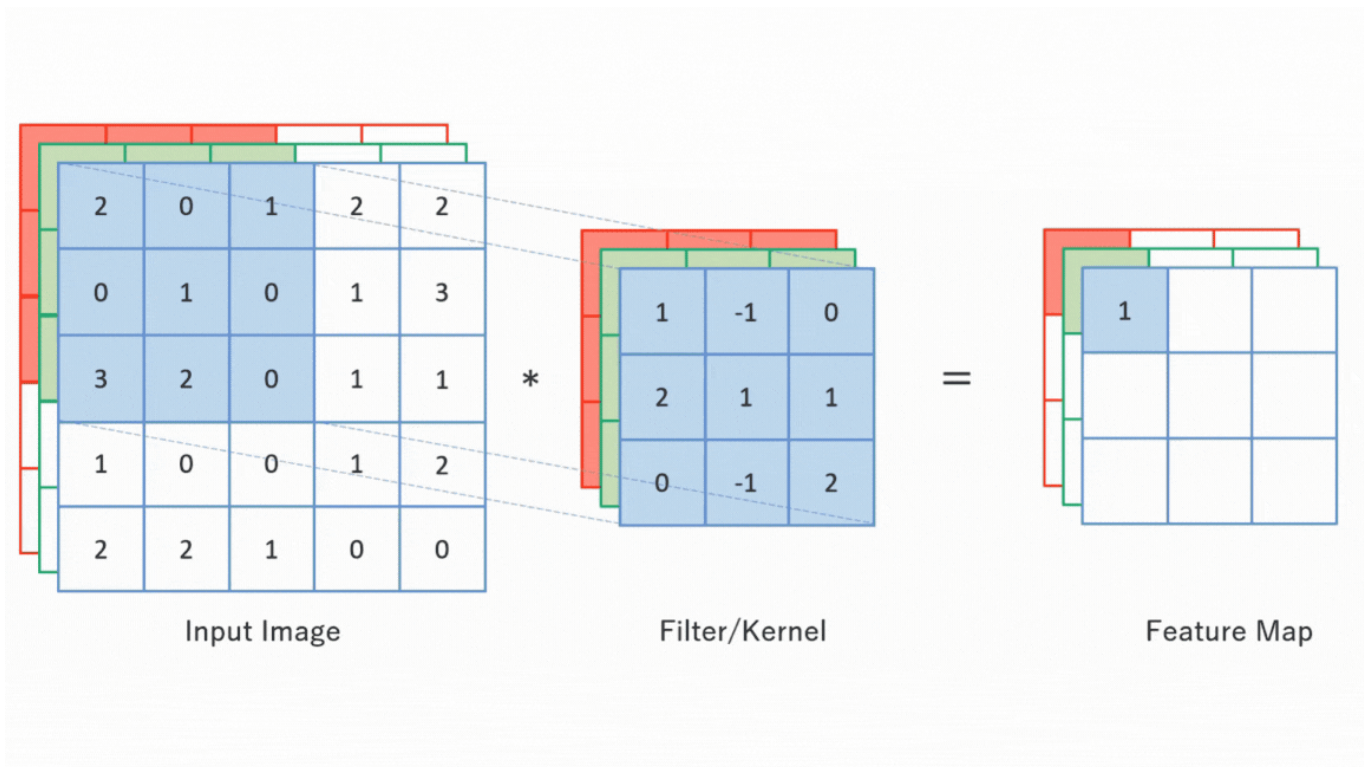
So anyway, what is a filter?

Filter

A filter is a small-sized matrix used to **extract** features from an image. It is applied on the input matrix through a convolution operation, specifically, **dot product**. In other words, the filter matrix is multiplied **element-wise** with the corresponding input values and then **summed** together into one value.

This filter is then **slid** across the input image until the whole input image has been convoluted. Bare in mind that since most images are composed of **RGB** channels, the same filter would be applied onto all three channels. The outputs of each of these channels would be **summed** to make a **single feature map**.

This information can be difficult to comprehend with just words, so please take a minute to absorb all this information using the animation below.

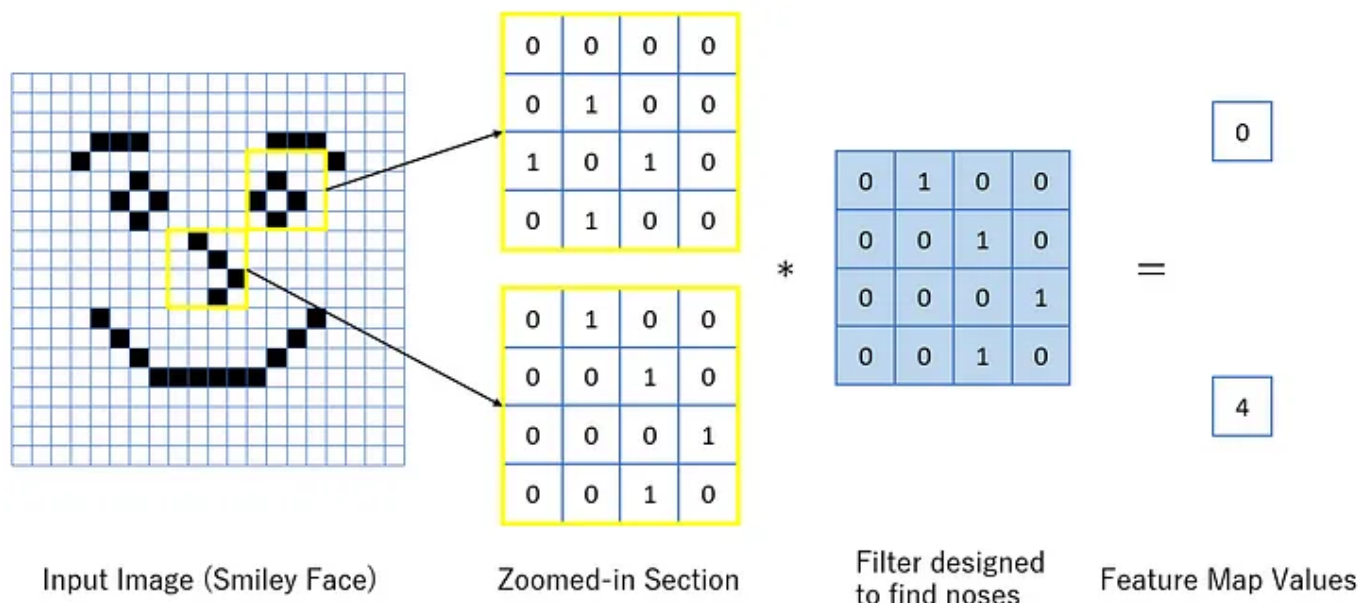


A convolutional layer demonstrating a filter of stride one sliding across the input image and creating a feature map through corresponding dot product operations.

***Note:** To **prevent** confusion, please be aware that the **blue filter** applies a dot product on **all three RGB channels**, not just the blue channel and **sums** their results into one output value on the blue feature map. This is then **repeated** with the green and red filters, producing **three** output feature maps. To simplify calculations, pixel values of 0 were assigned to both the green and red channel.*

After understanding this, it's now time to ask, how **exactly** do filters learn features?

Let's look and at a simple example for intuition purposes.



Two sections of a smiley face image being filtered with the corresponding output values shown.

The input image is a smiley face composed of white and black pixels taking values of 0 and 1, respectively. The pixel values of the zoomed-in image can be seen in the **yellow** boxes.

Additionally, the filter in this example was designed **specifically** to find noses within an image. Please bear in mind that in reality, the values of the filter matrix are **not a hyperparameter**, but in fact learnt during training and can take any value, negative or positive.

The filter is then applied onto the zoomed-in pixel maps with their dot product representing the values of the feature map. The reasoning behind naming the output, feature map, is that the **feature map** effectively portrays the **distribution** of a particular feature.

In our example, the yellow matrix representing the smiley face's nose **aligns perfectly** with our nose finding filter, therefore, producing a **high** feature map value of 4. On the other hand, the yellow matrix representing the smiley face's eye, unsurprisingly, does not align with the filter and as a result,

produces a dot product of **0**. Using this, the model learns that a nose is most likely located at the grid position which produced the value of **4**, rather than **0**.

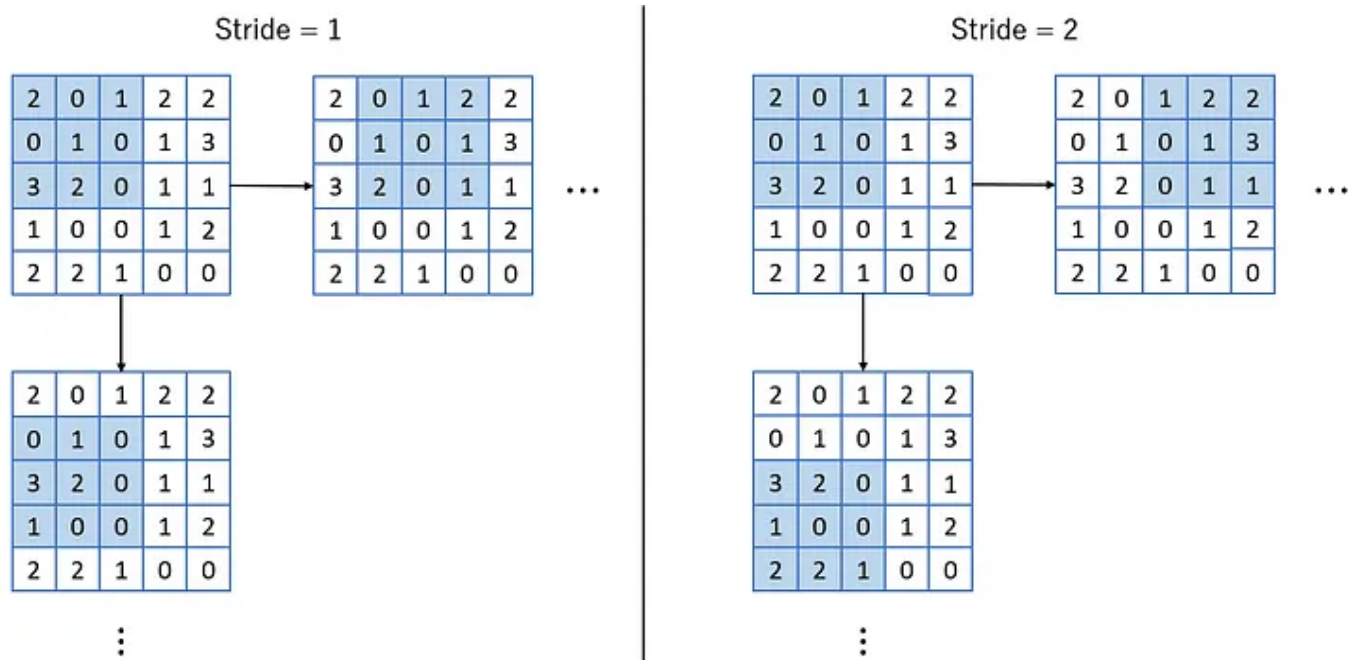
Now, this was only the first convolution layer. The output feature map is then further fed as inputs into the next convolution layer. As you indulge into deeper layers, more abstract and complex features can be learnt, many of which, are undetectable to the human eye.

Now that you have learnt how filters function, it is time to learn how they move across an image.

Stride

In the convolution layer, the term stride refers to the speed in which a filter translates across an image. A stride of **1** would mean that the filter travels one pixel at a time and with a stride of **2**, two pixels at a time etc.

Simple as that.



A 3×3 filter of stride 1 (left) and of stride 2 (right) applied on a 5×5 input matrix.

Why does this matter?

Well, take a look at the **stride = 1** image, the feature map will clearly be a 3×3 matrix. Now look at **stride = 2**, the feature map is clearly a 2×2 image.

Therefore, higher stride values are often used for **dimension reduction**, thus **lowering** computational complexity. However, be weary that if you assign too high of a stride value, the network may lose vital information.

Consequently, have you noticed anything interesting about feature maps?

They are **always** of smaller size than the input matrix. This leads us onto the next section, padding.

Padding

When applying a filter over an input matrix, which pixels experience the least filtering?

Evidently, the **borders**, with **corners** undergoing the least filtering (filtered only once).

This fact poses an issue. While central pixels are involved in numerous convolutions, the borders are not. This is called the **border effect**, which essentially is a loss of information at the borders since they contribute less towards the output feature map.

How do we fix this? PADDING.

0	0	0	0	0	0	0
0	2	0	1	2	2	0
0	0	1	0	1	3	0
0	3	2	0	1	1	0
0	1	0	0	1	2	0
0	2	2	1	0	0	0
0	0	0	0	0	0	0

With Padding

2	0	1	2	2
0	1	0	1	3
3	2	0	1	1
1	0	0	1	2
2	2	1	0	0

Without Padding

An input image with one layer of zero padding (left) and the same input image with no padding (right). These are the states of the images during the first filter convolution.

Paddings are additional layers, usually 0s, which are added around the outsides of an input image, mitigating **border effect** and also increasing the spatial dimensions of the resulting feature map.

Pooling Layer

Following the convolution layer is the pooling layer. Besides downsampling, this layer also provides additional advantages.

Just like the convolution layer, the pooling layer involves using a filter that slides over the input (feature map) and then extracts a **single** output value.



An illustration of pooling using a 2×2 pooling filter with a stride of 2, both vertically and horizontally. Each colour represents a pooled region, note that pooling windows can overlap, for example, if stride = (1, 1).

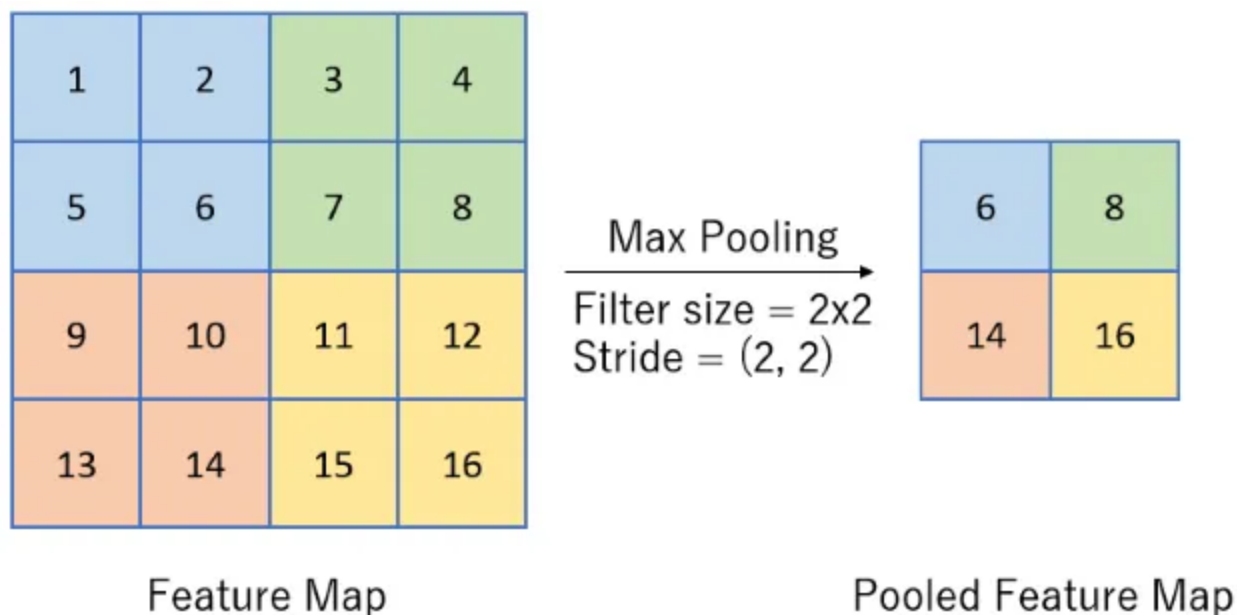
The main objectives of pooling are:

- **Spatial dimension reduction:** Pooling extracts only one value from the filtered area, therefore, reducing the matrix size and also the network's computational complexity.
- **Feature generalisation:** Small changes and shifts in the input will likely not affect the pooled feature map. This allows the network to be invariant to small variations.
- **Increased receptive field:** By pooling over a particular window of neighbouring pixels, the network is able to capture patterns from a larger area and increase its receptive field.

Generally, two types of pooling are used, **max pooling** and **average pooling**, however, take note that other variations do still exist. Each of these pooling techniques serve different purposes and will be discussed below.

Max Pooling

During max pooling, the most **dominant feature** (maximum value) within each pooling window is selected as the output value. Let's take a look at how max pooling is applied on the earlier displayed feature map.



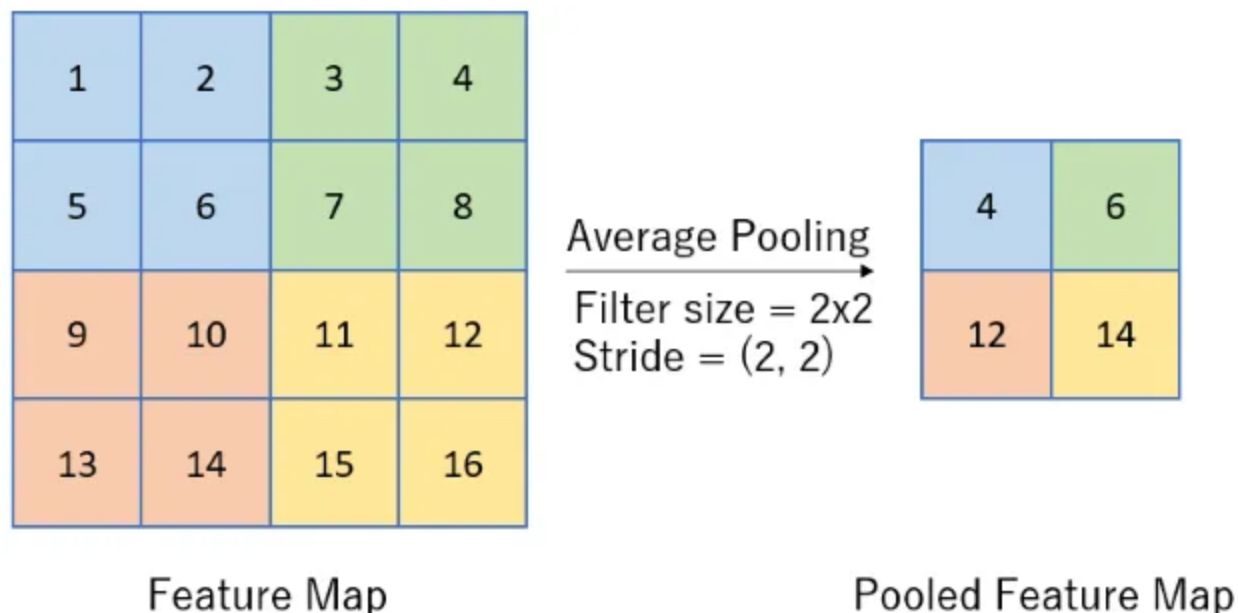
A max pooling operation.

Why perform max pooling?

By retaining the most dominant features in each local region, the network is able to ignore potential **noise** and focus on the most important features. In addition, by max pooling, the network also becomes **less** sensitive to small translations or shifts in the input.

Average Pooling

Similarly, during average pooling, the **average value** of each **pooling window** is used as the corresponding output value. The application of average pooling on the previous feature map can be seen below.



An average pooling operation.

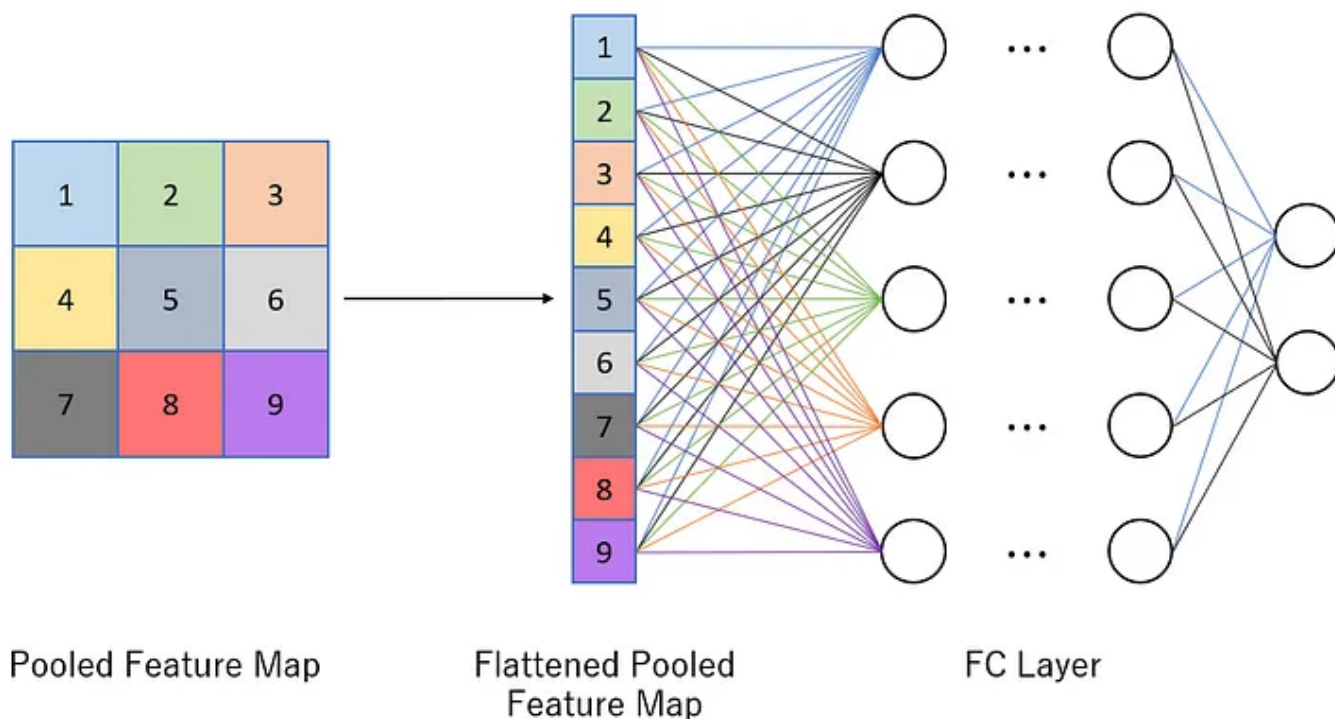
Why perform average pooling?

By computing the average value within each local region, a **smoothing** effect occurs and as a result reduces noise. Furthermore, just like in max pooling, the network becomes less sensitive to variations and shifts within the input data, making the model further robust.

Fully Connected Layer

The next and final layer in a convolutional network is the **fully connected (FC) layer**, also known as the **dense layer**. This layer follows a **traditional** neural network structure and aims to **aggregate** all the extracted features from previous layers so that a better understanding of global relationships is learnt. This particular layer is crucial for tasks such as image classification.

Take this feature map for instance.



A pooled feature map being flattened and inputted into a fully connected (FC) network layer.

Each value of the pooled feature map represents a **different** set of features, while some overlap may be present, many values will have been derived from different parts of the image, take feature 1 and 9 for example.

Next, the pooled feature map is **flattened**. This transforms the matrix into a **column vector** so that it is able to be fed into the fully connected network.

Inside the fully connected network, each layer connects **all** its nodes to **each and every** node in the next layer, hence the name 'fully connected'. This is a **critical** characteristic of the fully connected layer.

This is because, by having all extracted features contribute to every individual node, the network is able to combine data and generate further relationships between them as well as capture **global** context and patterns.

Along with the use of **non-linear** activation functions such as **ReLU** at each node within the fully connected layer, the network is able to learn complex **non-linear** relationships between different features, something that wasn't possible in previous layers.

The importance of FC layers can be seen in image classification. For instance, when feeding the FC layer's outputs through a **softmax function**, a probability distribution corresponding to the possible classes can be outputted.

Summary

In this article we have discussed the functionality of conventional neural networks (CNNs) and its architecture. The first **convolution layer** finds and extracts features from an input image. Then, the **pooling layer** takes the output feature map and downsamples it through either max or average pooling. By keeping the dominant features and reducing the noise, the network can focus on learning only the important details. Finally, the output is **flattened** into a column vector and then fed into a **fully connected (FC) layer**, which combines features and learns corresponding patterns in regard to the image's global context.

Just a note on the CNN architectural design, commonly used CNNs will use a mixture of **alternating** pooling and convolution layers such as those in the first image of this article (yes, scroll up, a lot) for higher efficiency. However, the CNN will **most often** end in a fully connected layer.

I hope that you have understood how basic CNNs work. If you have any questions **please** do not hesitate to comment them down below.

Machine Learning

Cnns

Neural Networks

Image Analysis

AI

Open in app ↗



Search

Write



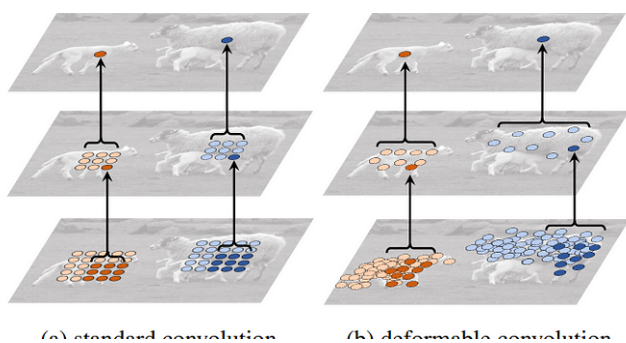
Written by Alejandro Ito Aramendia

58 Followers

Learning and writing.

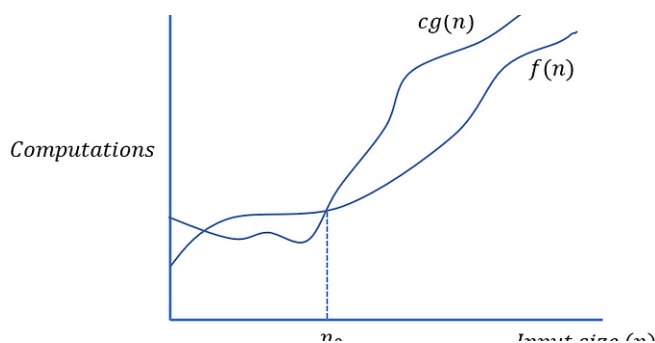
Follow

More from Alejandro Ito Aramendia



(a) standard convolution

(b) deformable convolution



Alejandro Ito Aramendia

Deformable Convolutional Networks (DCNs) : A Complete...

Everything You Need to Know

8 min read · Feb 1, 2024



Alejandro Ito Aramendia

A Guide to Understanding Big-O, Big-Ω and Big-Θ Notation

Time Complexity is critical when it comes to programming. It is a key decider on whether...

5 min read · Oct 1, 2023



177



2



...



61



1



...



Alejandro Ito Aramendia

A Guide to Dijkstra's Algorithm | All You Need

Picture this, you are on holiday in a foreign country and you are lost. The area is...

8 min read · Oct 31, 2023



53



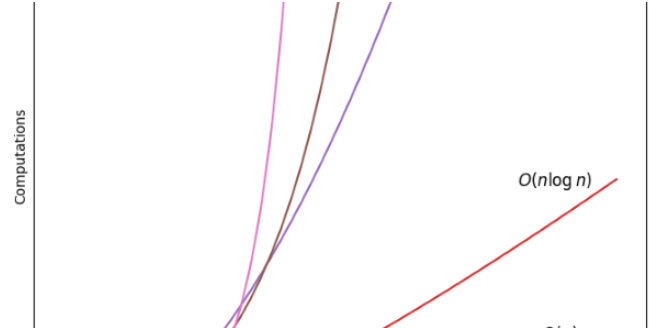
...



1



...



Alejandro Ito Aramendia

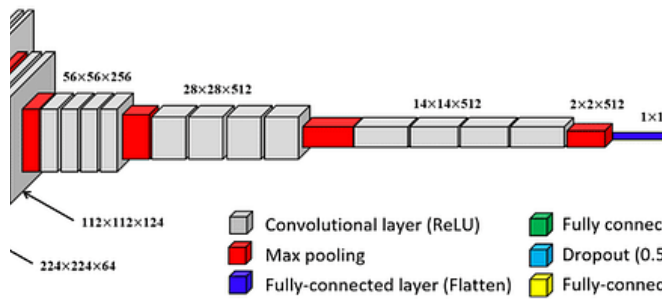
What Exactly is Time Complexity?

Have you ever been programming and had multiple algorithms to choose from? Which...

5 min read · Oct 1, 2023

See all from Alejandro Ito Aramendia

Recommended from Medium



D Daniyal Masood

Pre-trained CNN architectures designs, performance analysis an...

Pre-trained CNN (Convolutional Neural Network) models are neural networks that...

12 min read · Oct 17, 2023



57



...

H Hrithick Sen

The Math Behind the Machine: A Deep Dive into the Transformer...

The transformer architecture was introduced in the paper "Attention is All You Need" by...

12 min read · Jan 16, 2024



188

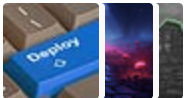


5



...

Lists



Predictive Modeling w/ Python

20 stories · 932 saves



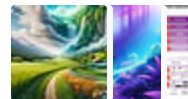
The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 315 saves



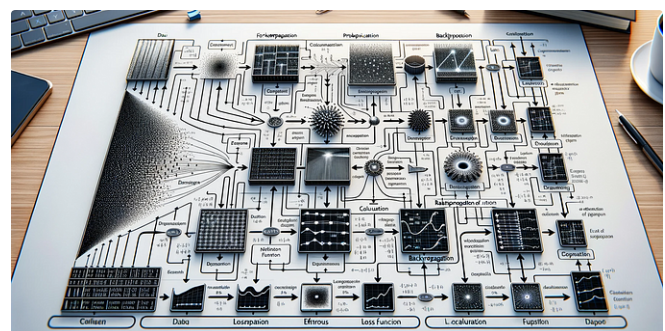
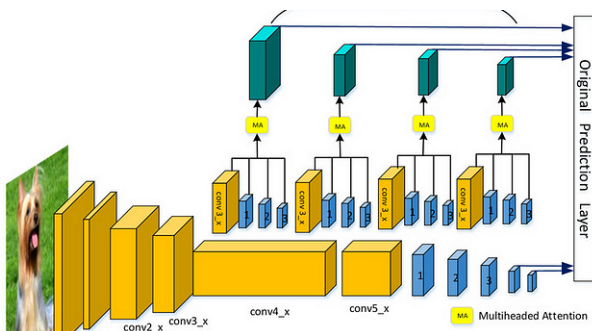
Practical Guides to Machine Learning

10 stories · 1096 saves



Natural Language Processing

1224 stories · 706 saves




 **Everton Gomedé, PhD**

SSD Neural Network: Revolutionizing Object Detection

Introduction

5 min read · Feb 16, 2024

 131  1

 **Jorge Cardete** in Towards AI

Backpropagation

From mystery to mastery: Decoding the engine behind Neural Networks.

8 min read · Nov 2, 2023

 997  7

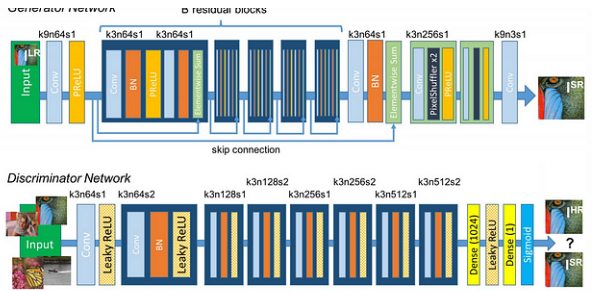


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps

 **Abdulkader Helwan**


How to Implement a Super-Resolution Generative Adversaria...

A Super-Resolution Generative Adversarial Network (SRGAN) is a powerful model in the...

🌟 · 3 min read · 5 days ago

 **Kasun Dissanayake** in Towards Dev

Machine Learning Algorithms(16) —Support Vector Machine(SVM)

This article, delves into the topic of Support Vector Machines(SVM) in Machine Learning,...

12 min read · Feb 3, 2024

 613  3

See more recommendations