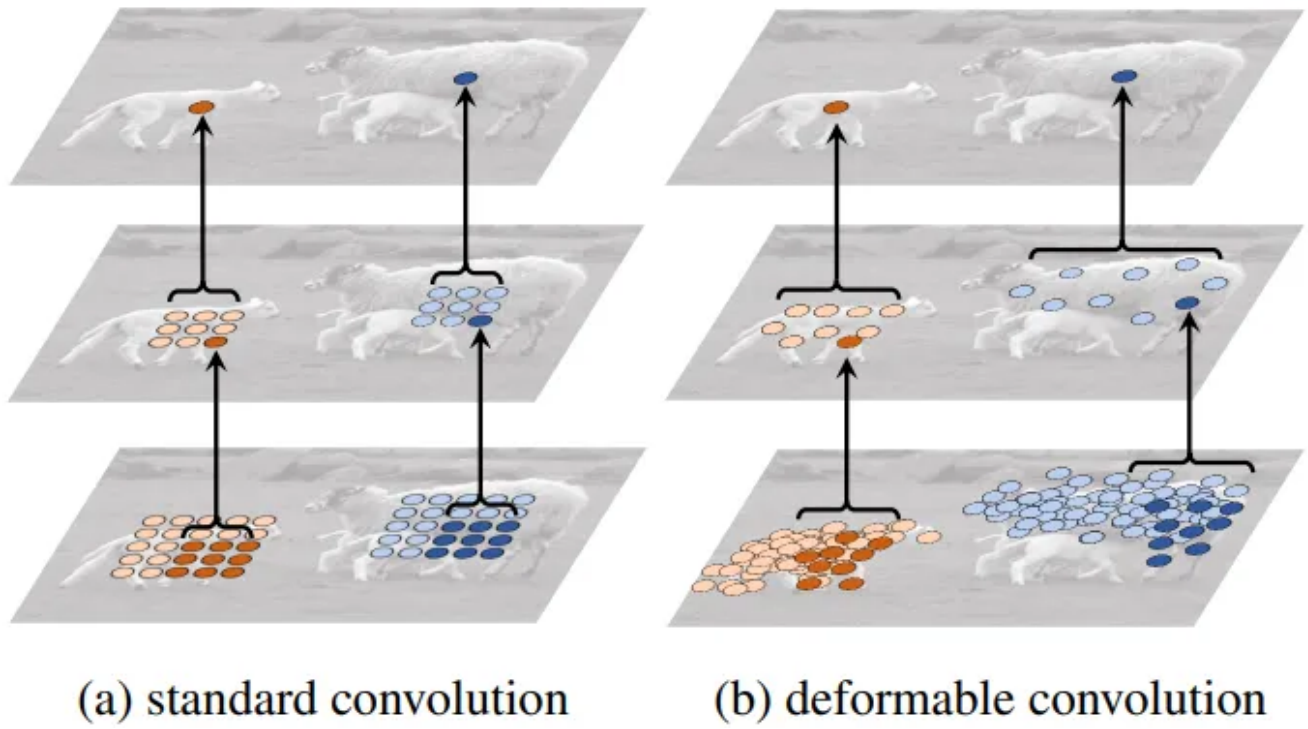
Deformable Convolutional Networks (DCNs) : A Complete Guide

Alejandro Ito Aramendia · Follow

8 min read · Feb 1, 2024



An image comparing standard and deformable convolutions. It is clear how the deformable convolution can encapsulate objects of different shapes, allowing for more accurate and reliable feature extraction.

Table of Contents

Introduction

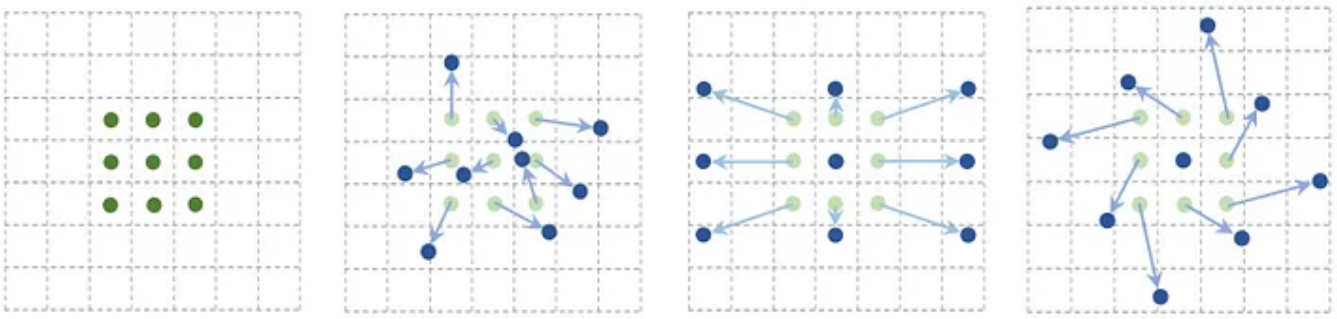
A key issue with traditional convolutional neural networks (CNNs) is their inability to **adapt** to geometric variations and transformations. Using **fixed** geometric structures such as: fixed filters, fixed pooling and fixed RoI spatial bins, limits any flexibility within the network.

A way that CNNs attempt to solve this problem is by augmenting existing data samples through various transformations. This, however, requires additional computing power and more complex model parameters. Equally, this type of network will quickly crumble when facing **unknown** transformations due to a lack of generalisation in the model.

As a result, a more feasible way of tackling this problem is through the introduction of deformable convolutional networks, which attempt and succeed in adapting to unprecedented transformations. This article aims to explore two such modules: **deformable convolutions** and **deformable RoI pooling**.

- **Deformable Convolutions**: Allows the filter to dynamically adjust its sampling locations with learnable offsets so that a better spatial relationship of the input data may be modelled.

- **Deformable RoI Pooling:** The RoI is divided into a fixed number of bins with a learnable offset allowing for a more dynamic pooling operation. This lets the model focus on more relevant spatial locations within the RoI.



Four examples of how a regular 3×3 filter may be deformed to accommodate various transformations.

The ability of free form deformation as illustrated in the image above clearly demonstrates the power deformable convolutional networks can have in generalising different transformations.

Now, before we dive into the article, I highly advise that you are first familiar with traditional convolutions and RoI pooling, if not already.

Deformable Convolutions

Before we immediately dive into deformable convolutions, let's quickly revise regular convolutions and build up from there.

A convolution is an operation used to extract and capture features of an image such as edges, shapes and also abstract features undetectable to the human eye.

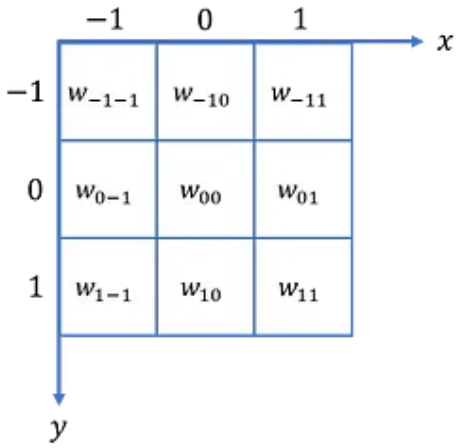
The 2D convolution is an operation that uses a regular grid ***R*** that has weights **w** and is sampled over an input feature map. The summation of all the sampled values equates to the convolution's output result.

In other words, a 2D convolution is the dot product between the filter and corresponding input feature map values.

The grid ***R*** and its corresponding weights **w** are shown below.

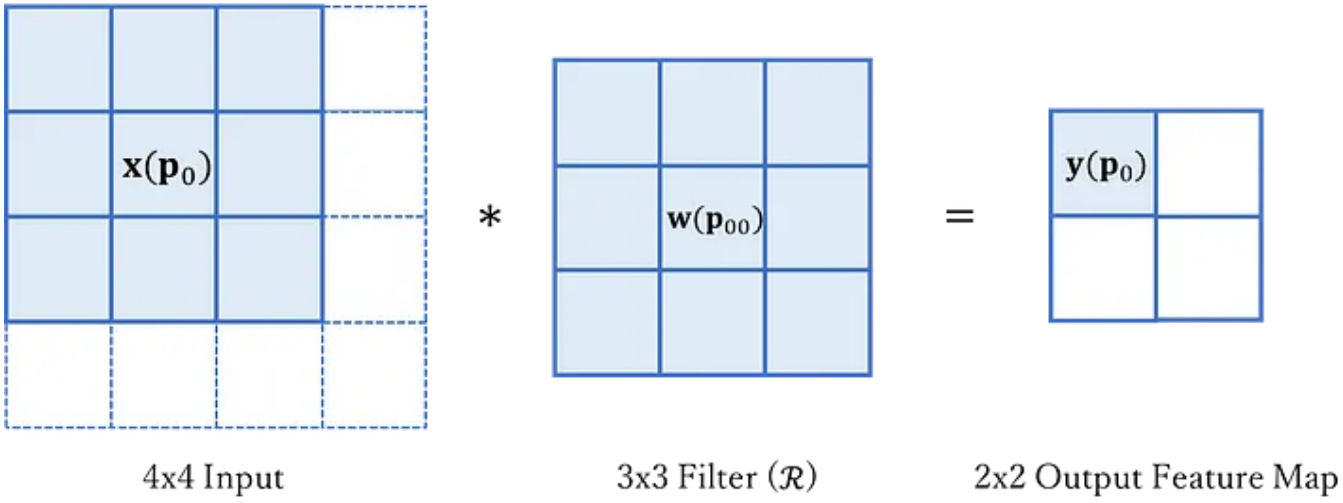
Remember that in image processing, the coordinate axis is often defined from the top left-hand corner of the image. Equally, please note that in the grid, position (**i**, **j**) corresponds to weight $\mathbf{w}_{ji}$ (matrix notation), not $\mathbf{w}_{ij}$.

$$\mathcal{R} = \{(-1,-1),(-1,0),\dots,(0,1),(1,1)\} \quad \Rightarrow$$



Then, for each location $\mathbf{p}_0$ on the output feature map **y**, its value can be defined mathematically like below. In this equation, $\mathbf{p}_n$ enumerates all locations in grid ***R***.

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n)$$



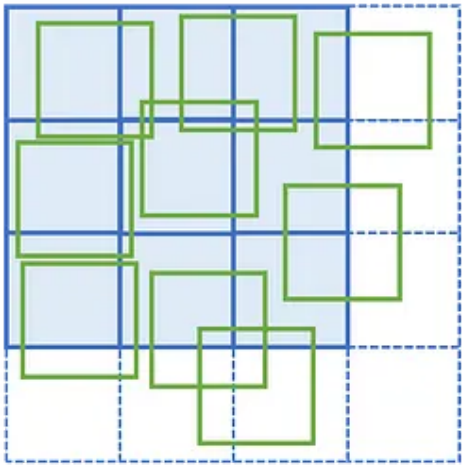
The mechanism of a standard convolution. In the input image $\mathbf{x(p_0 + p_{00}) = x(p_0)}$.

Now, that's it for regular convolutions. Let's now extend this model.

In **deformable convolutions**, we introduce a grid ***R*** that is augmented by offsets $\Delta\mathbf{p}_n$. This essentially means that the grid ***R*** becomes irregular and grid boxes shift ever so slightly.

This can be easy to visualise with the below diagram, along with the corresponding equation for deformable convolutions.

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$$



4x4 input with 3x3 filter
and offsets applied (green).

As you can see in the diagram, a regular convolution will sample locations from the receptive field shown in blue. However, **in deformable convolutions**, sampling occurs on the irregular offset locations $\mathbf{p}_n + \Delta\mathbf{p}_n$ shown in green. In practice, the offset $\Delta\mathbf{p}_n$ is typically fractional.

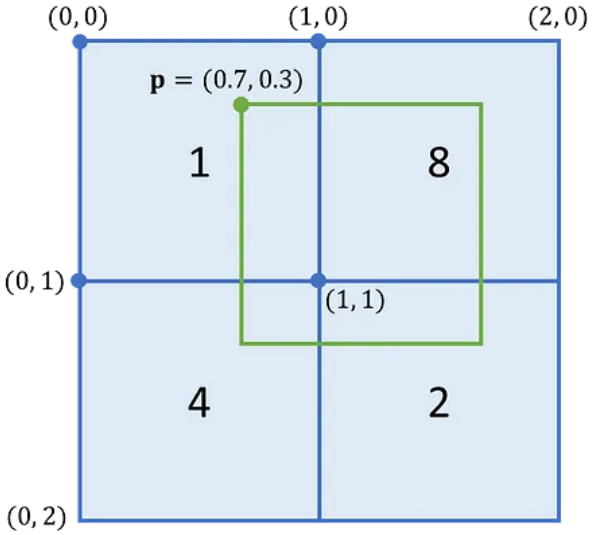
If $\Delta\mathbf{p}_n$ is fractional, then how can we know the value of $\mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)$?

Well, we will never know with absolute certainty, but, we can make very accurate estimates by using **bilinear interpolation.**

> ***Bilinear interpolation*** *is a method used to interpolate functions of two variables. In other words, given a rectangle with only the corner values known, bilinear interpolation allows us to estimate the value of the function at any point within the rectangle.*

In the case of deformable convolutions, this is represented by a bilinear interpolation kernel $G(...)$ where $\mathbf{p}$ denotes an arbitrary location ($\mathbf{p} = \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n$) and $\mathbf{q}$ enumerates all original spatial locations in the feature map $\mathbf{x}$.

The implementation of this can be seen in the below equations. Accompanying this, is a visual example where a pixel has been offset to the location in green. I will provide some further explanation next.



$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q})$$

where $\begin{cases} G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y) \\ g(a, b) = max(0, 1 - |a - b|) \end{cases}$

$$\begin{aligned} \mathbf{x}(\mathbf{p}) = & \, max(0, 1 - |0 - 0.7|) \cdot max(0, 1 - |0 - 0.3|) \cdot \mathbf{x}(\mathbf{0}, \mathbf{0}) \\ & + max(0, 1 - |1 - 0.7|) \cdot max(0, 1 - |0 - 0.3|) \cdot \mathbf{x}(\mathbf{1}, \mathbf{0}) \\ & + max(0, 1 - |0 - 0.7|) \cdot max(0, 1 - |1 - 0.3|) \cdot \mathbf{x}(\mathbf{0}, \mathbf{1}) \\ & + max(0, 1 - |1 - 0.7|) \cdot max(0, 1 - |1 - 0.3|) \cdot \mathbf{x}(\mathbf{1}, \mathbf{1}) \end{aligned}$$

$$\begin{aligned} \mathbf{x}(\mathbf{p}) = & \, 0.3 \cdot 0.7 \cdot \mathbf{1} \\ & + 0.7 \cdot 0.7 \cdot \mathbf{8} \\ & + 0.3 \cdot 0.3 \cdot \mathbf{4} \\ & + 0.7 \cdot 0.3 \cdot \mathbf{2} \end{aligned}$$

$$\mathbf{x}(\mathbf{p}) = 4.91 \approx 5$$

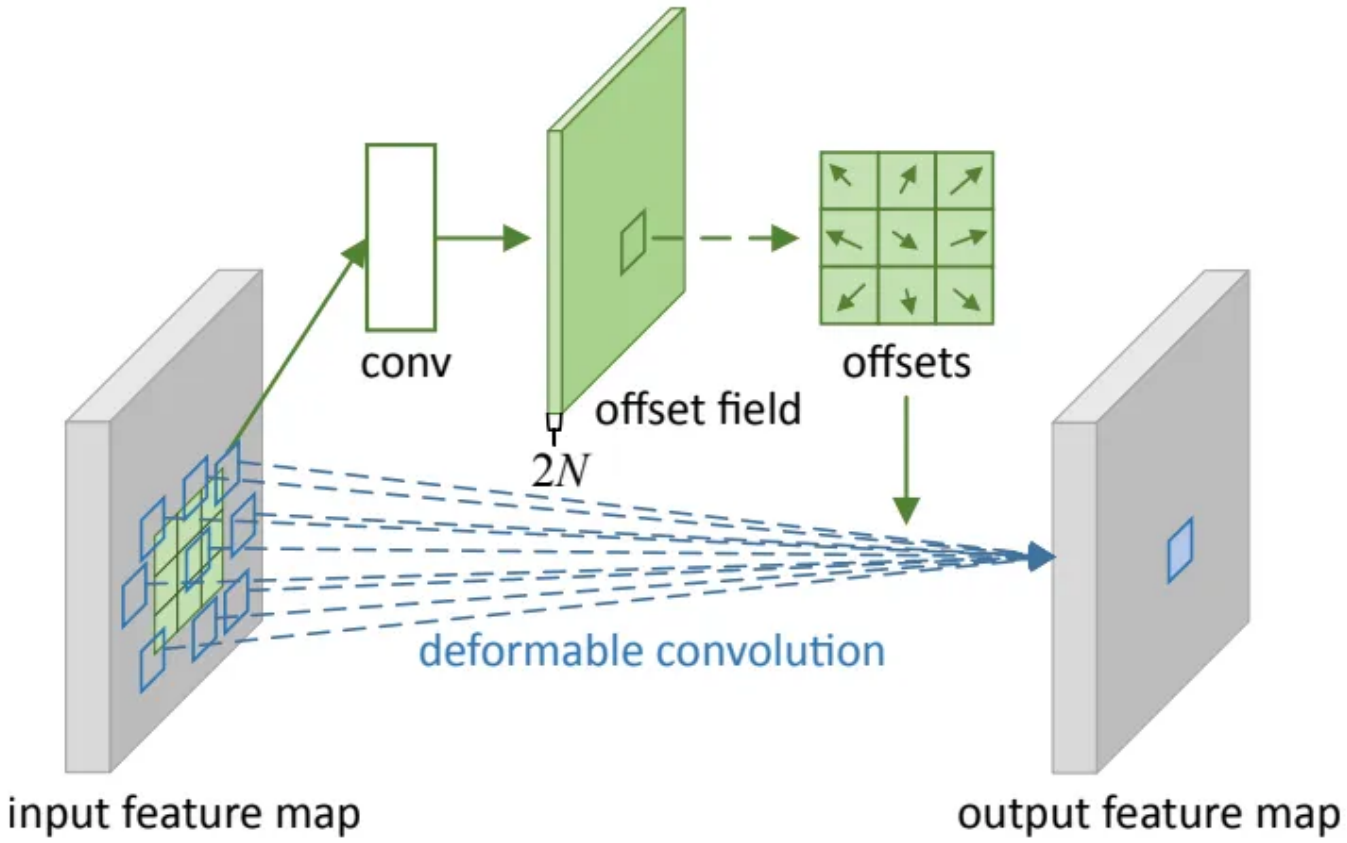
Brief Explanation about the Example

- In the image above, there are four pixels with their values shown.
- In green, an offset pixel **p** with a fractional location is shown. Note that **p**'s original pixel location is **not important** as bilinear interpolation depends only on surrounding pixels.
- We must use the **same reference point** for each pixel when measuring the **offset distance** between pixels. In this example, I chose the top left-

hand corner of each pixel (other points such as top right, bottom left or centre, could have also been chosen).

- $G(\mathbf{q}, \mathbf{p}) = 0$ for all pixels not in the offset pixel's immediate surrounding. This is because $g(a, b) = max(0, 1 - |\mathbf{a} - \mathbf{b}|) = 0$ as $1 - |\mathbf{a} - \mathbf{b}| < 0$ for all pixels **q** further that one pixel length from the offset pixel **p**. This is why we **only use four pixels** for bilinear interpolation.
- Substituting all the values gives us an approximate pixel value of 5, which makes sense if we look at it visually.

I hope you have understood everything up until this point. Now, I will briefly discuss its **implementation** within the convolutional layer with reference to the below image.



The architecture of a deformable convolution.

In deformable convolutions, a regular filter is applied over the input feature map, producing a standard output feature map. Simultaneously, an **offset field** is generated, representing **2D offsets for N filters** (N channels for both the x and y direction). These offsets serve as predictions for the adjustments needed in the next forward pass during training and **do not represent the current filter offsets**. During training, the convolutional filter and offsets are learned simultaneously, with backpropagation applied on the bilinear operations for offset learning. The gradient for these bilinear operations can be seen below.

$$\frac{\partial \mathbf{y}(\mathbf{p}_0)}{\partial \Delta \mathbf{p}_n} = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \frac{\partial \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n)}{\partial \Delta \mathbf{p}_n} = \sum_{\mathbf{p}_n \in \mathcal{R}} \left[\mathbf{w}(\mathbf{p}_n) \cdot \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta \mathbf{p}_n)}{\partial \Delta \mathbf{p}_n} \mathbf{x}(\boldsymbol{q}) \right]$$

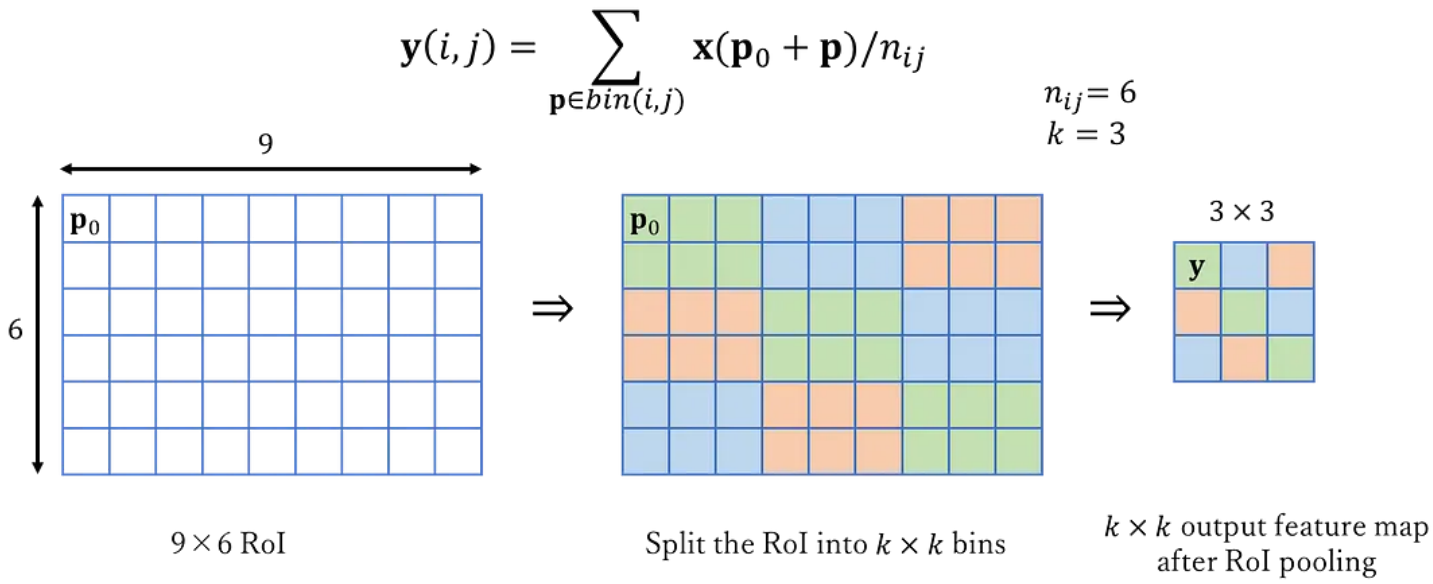
Great! Now you have learnt deformable convolutions. Let's now move onto the second deformable module, deformable RoI pooling.

Deformable RoI Pooling

Region of Interest (RoI) pooling is an important concept that is used in all region proposal based **object detection** methods. It's purpose is to **convert** variable-sized regions of a feature map into **fixed-sized** representations. This allows the network to downsample each RoI into a same sized output while displaying only its most important features.

In regular RoI pooling, we first define an input feature map $\mathbf{x}$ and an RoI of size $\boldsymbol{w} \times \boldsymbol{h}$ with a top left-hand corner $\mathbf{p}_0$. Essentially, RoI pooling divides the RoI into $k \times k$ bins where k is a hyperparameter. The number of pixels in each bin is defined with $\mathbf{n}_{ij}$. The output is then a $k \times k$ feature map $\mathbf{y}$.

The output for each bin can be seen below. Note that this is **average RoI pooling** and other pooling types may also be used.



An example of a 9×6 RoI being divided into 3 × 3 bins during RoI pooling.

In this equation, **p** represents the positions in each bin and for this example, vector **p** holds $\mathbf{n_{ij}}$ = **6 positions**. Note that in this example, the RoI perfectly splits into 3 × 3 regions, however, this is not the case for all RoI sizes, take 10 × 4 for example. This can be solved with methods such as RoI align, which are sadly outside the scope of this article.

Now, similar to what we did with deformable convolutions, we can add offsets $\Delta\mathbf{p_{ij}}$ to the spatial binning positions before they are pooled.

$$\mathbf{y}(i,j) = \sum_{\mathbf{p}\in bin(i,j)} \mathbf{x}(\mathbf{p}_0 + \mathbf{p} + \Delta\mathbf{p}_n)/n_{ij}$$

However, since RoIs can have different sizes, $\Delta \mathbf{p}_{ij}$ may vary in scale.

Due to this, the network could learn **different offsets** for identical images of different scale. This is not what we want.

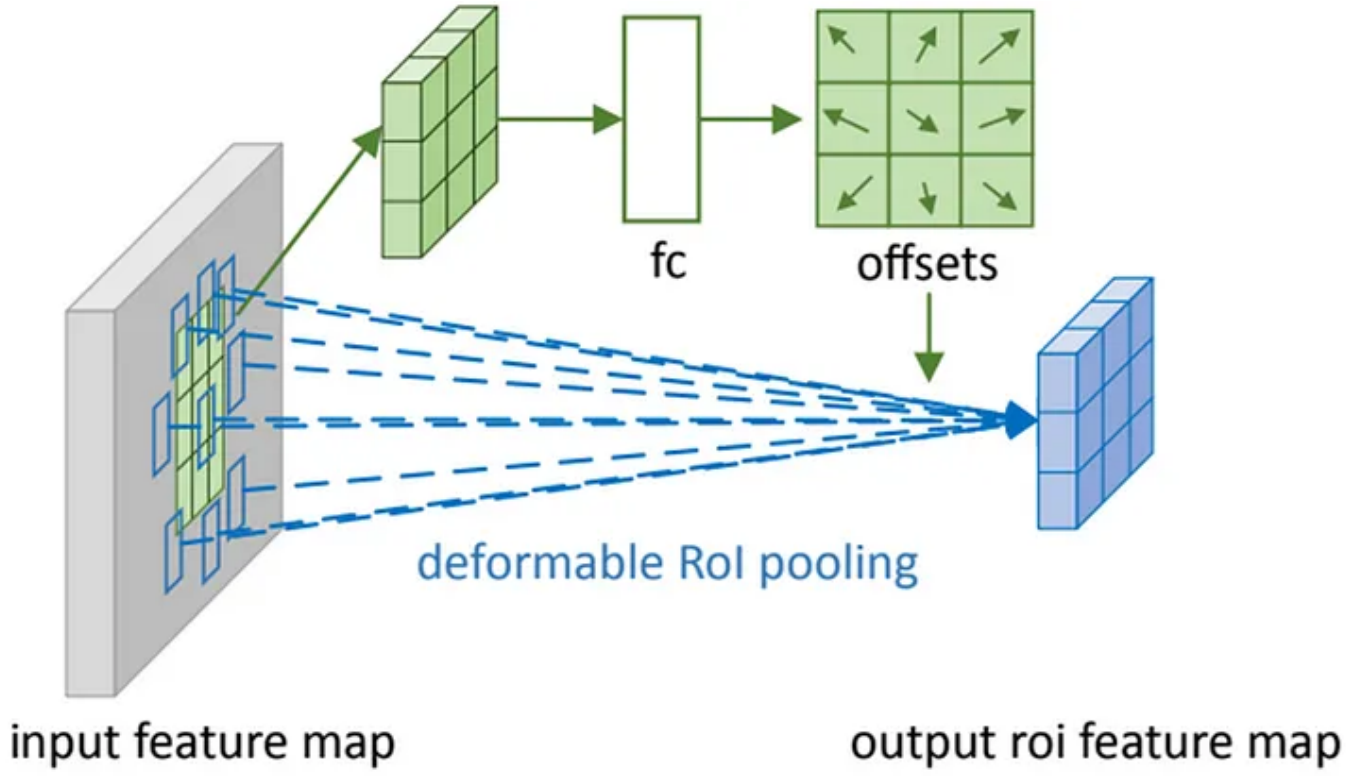
In order to resolve this, an **FC-layer** is implemented.

The network first performs regular RoI pooling, producing pooled feature maps. Then, these feature maps are flattened and fed into an FC-layer where **normalised offsets** $\Delta \hat{\mathbf{p}}_{ij}$ are then generated. These offsets are then transformed and scaled to $\Delta \mathbf{p}_{ij}$. This transformation is achieved by applying a dot product over the RoI region size.

$$\Delta \mathbf{p}_{ij} = \gamma \cdot \Delta \hat{\mathbf{p}}_{ij} \circ (w, h)$$

In this equation, γ is a predefined scalar that modulates the magnitude of the offsets and is commonly set to $\gamma = 0.1$.

This offset normalisation is crucial as it allows the offset learning and RoI size to be **invariant**. This is important in images where identical objects of different sizes are present.



The architecture of deformable RoI pooling.

Just like in deformable convolution, the offsets are learnt via backpropagation on the bilinear operations. This is shown below.

$$\frac{\partial \mathbf{y}(i,j)}{\partial \Delta \boldsymbol{p}_{ij}} = \frac{1}{n_{ij}} \sum_{\mathbf{p} \in bin(i,j)} \frac{\partial \mathbf{x}\left(\mathbf{p}_0 + \mathbf{p} + \Delta \mathbf{p}_{ij}\right)}{\partial \Delta \mathbf{p}_{ij}} = \frac{1}{n_{ij}} \sum_{\mathbf{p} \in bin(i,j)} \left[\sum_{\mathbf{q}} \frac{\partial G\left(\mathbf{q}, \mathbf{p}_0 + \mathbf{p} + \Delta \mathbf{p}_{ij}\right)}{\partial \Delta \mathbf{p}_{ij}} \mathbf{x}(\mathbf{q}) \right]$$

Great! This is everything you need to know about deformable RoI pooling, I hope you found it resourceful.

Summary

In this article, we have covered deformable convolutional networks and their importance in generalising and learning various transformations.

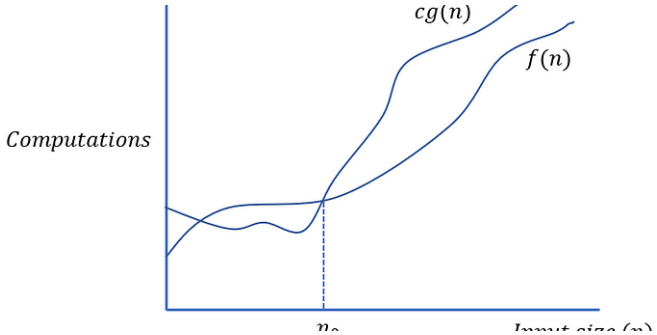
To summarise, deformable convolutions operate by adding additional offset parameters to the network. Instead of a filter being fixed in shape, fractional offsets that are learnt by the network are added. This deforms the filter and allows the same filter to reach different positions of the input feature map. This adjustment is beneficial when dealing with different geometric structures. In addition to this, deformable RoI pooling also experiences the same advantages allowing for more local and accurate object detection.

References

[1] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, Yichen Wei. Deformable Convolutional Networks. arXiv:1703.06211v3.

Data Science | Machine Learning | AI | Technology | Image Processing

More from Alejandro Ito Aramendia



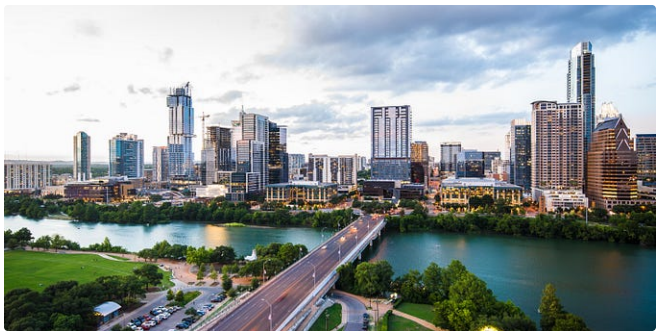
Alejandro Ito Aramendia

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

61 1



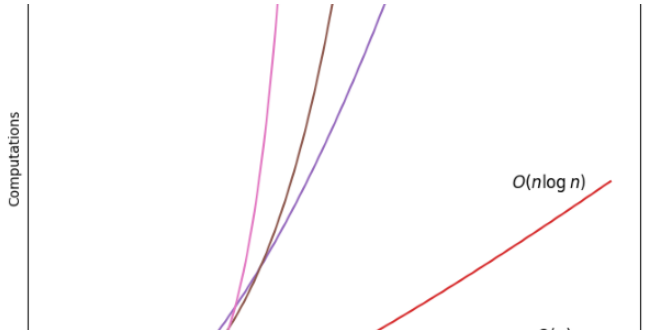
Alejandro Ito Aramendia

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



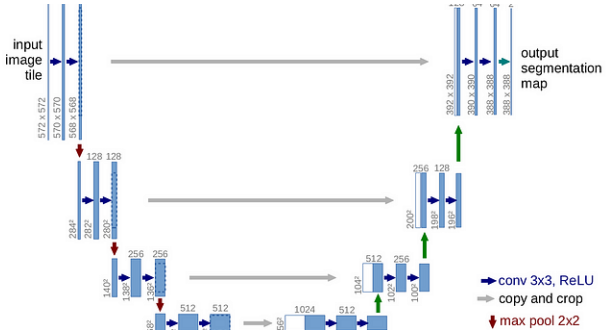
Alejandro Ito Aramendia

What Exactly is Time Complexity?

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

5 min read · Oct 1, 2023

1



Alejandro Ito Aramendia

The U-Net : A Complete Guide

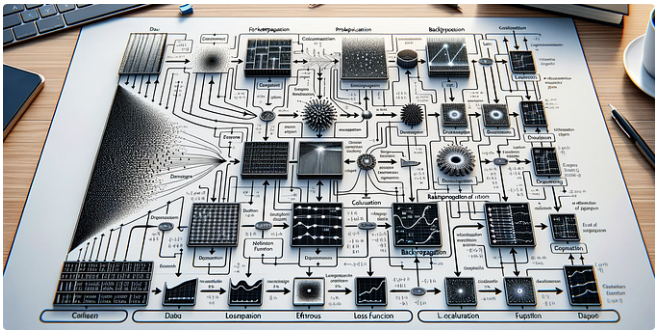
Everything you Need to Know

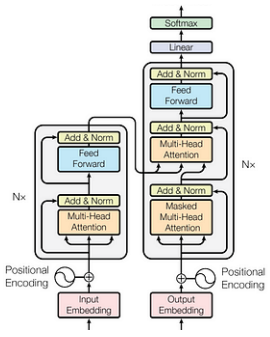
7 min read · Feb 1, 2024

157

See all from Alejandro Ito Aramendia

Recommended from Medium





Jorgecardete in Towards AI

Backpropagation

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

8 min read · Nov 2, 2023

997 7

Hrithick Sen

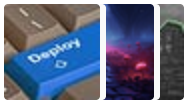
The Math Behind the Machine: A Deep Dive into the Transformer...

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

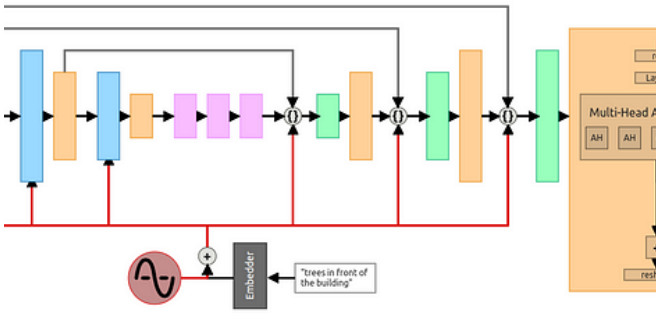
12 stories · 315 saves

ChatGPT prompts

42 stories · 1157 saves

Practical Guides to Machine Learning

10 stories · 1096 saves



Kemal Erdem (burnpiro)

Step by Step visual introduction to Diffusion Models.

How the diffusion models works under the hood? Visual guide to diffusion process and...

15 min read · Nov 10, 2023

Kailash Ahirwar

A Very Short Introduction to Diffusion Models

Artificial Intelligence is constantly evolving to solve hard and complex problems. Image...

4 min read · Sep 26, 2023

206 1 235

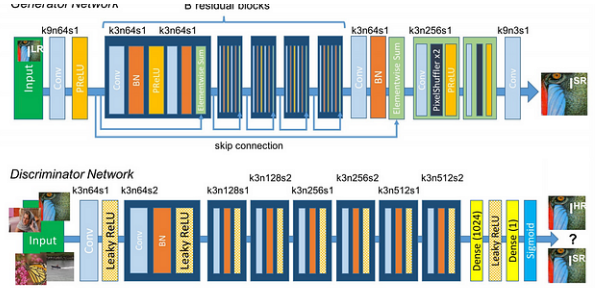
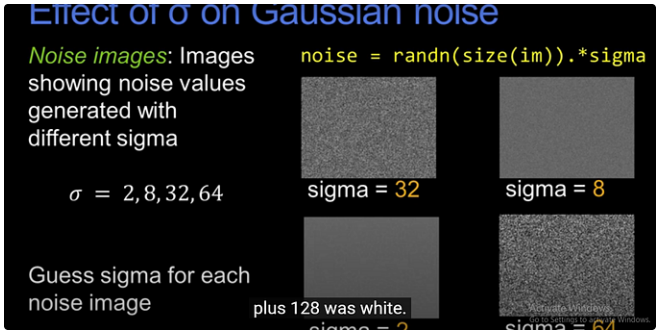


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps



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

Coursesteach

Computer Vision (Part 19)- Displaying and Read Images in...

Chapter: 2-Image As Function

6 min read · Jan 13, 2024

4

See more recommendations