Deep Convolutional Neural Networks

Benjamin Santana Velazquez

May 2023

1 Introduction

Humans can instantly identify objects in images. We can tell not only what object is inside the image, but we can easily segment the image into regions (this part is grass, this other part is water and so on). Machines have always struggled with that. Identifying an object in an image while assigning a category, to then describe the regions of that image, was a daunting task for a computer.

Before deep learning, computer vision techniques consisted on designing rules to classify and localize objects. Needless to say, this was not scalable. There are many variables when it comes to an image, lighting scales, colors, backgrounds just to name a few. It becomes impossible to encompass all possible scenarios in a fixed set of rules.

The problem of identifying and categorizing the objects present in an image is called *image classification* while if, we also want to identify the location of the object inside the image, it is called *object detection*, and if on top of that, you want to draw the boundries of each object in the image, it is called *image segmentation*.

Recently, a new class of algorithms emerged, *Convolutional Neural Networks* (aka *CNN*). These, do not rely on a fixed set of rules, but instead they make the computer learn relevant features from the data. These models are really accurate, sometimes they even beat humans when trying to solve this problem.

In this short paper, I will do my best to explain how they work, their components, and give some examples on the architecture of different convolutional neural networks, like LeNet and VGG.

2 What is a Convolution?

First I will explain what a convolution is, we will need a bit of math for this part, but bare with me, I believe that understanding this, is the key for understanding how this models work.

A convolution is a specialized operation that examines local patterns in an input signal. [1] For example, when we analyze an image, the input is a 2D array of pixels. We can define convolution operation that will help us detect edges and corners in a small neighborhood of pixels. Once we have detected such local properties, we can combine them and detect higher level structures, like a ears and noses. Then we can take the output of that, and classify even higher level structures like a face.

The astute reader, (with some background on neural networks) will see how this naturally lends itself to a multi-layer neural network. The layers closest to the input, detect edges and corners, while the closest to the output, detects higher level structures.

The exact local pattern captured depends on weights of the convolution operator. We do not know for sure what local patterns of the input we need to specify to recognize high level structures; meaning we do not know the specific weights of the convolutions. We want the machine to learn them through the process of training.

Since we do not specify it, we are not sure what local patterns the layers will learn to extract. Although, in practice the initial layers often learn to recognize edges and corners [1].

2.1 One Dimensional Convolution

The explanation above will become clearer when put into practice. I will do a convolution in one dimension since this will be easier to understand. It is left to the reader to investigate the process for more dimensions.

We can think of a 1D convolution as a string with a ruler. The string is the input while the ruler is something we will call *kernel*. The ruler is smaller than the string. We will move the ruler along the string while we calculate some values based on the input and the ruler; to then create a new string with the output of the operations.

I know this is hard to imagine. I will use a diagram from *Math and Architectures of Deep Learning* by Krishnendu Chaudhury (great book) that I found helpful while trying to understand the topic. Please refer to Figure 1

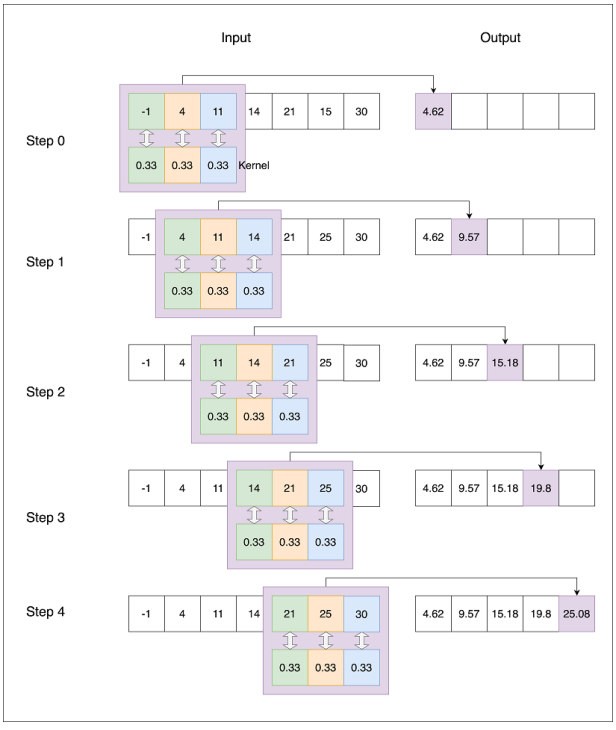

Figure 1: One Dimensional Convolution Graphic Representation taken from *Math and Architectures of Deep Learning* by Krishnendu Chaudhury

The left white boxes are the string. The purple box is the ruler/kernel, and the right is the output. Note that in each step we are moving the ruler one box at a time; we then calculate the output and put it in the output boxes.

Let us break down some concepts that apply to this one dimensional convolution operation, but that will help us understand convolutions of higher dimensions.

- **input**: one dimensional array, where n represent its length.
- **output**: one dimensional array, where o represent its length.
- **kernel**: a small array of weights whose size is a parameter of the convolution, we will use k to represent it.
- **stride**: how much are we going to move the kernel, in Figure 1, we just move it one box each step
- **padding**: when the kernel reaches the end or of the array, it may fall outside the input array. The padding refers to how we deal with this situation.

Basically we can calculate a single output value in a 1D convolution with the following Equation 1

$$Y_x = \sum_{j=0}^{k} X_{x+j} W_j \tag{1}$$

Where, Y denotes output, X denotes the input, W denote kernel, and k denotes the size of W (as mentioned earlier)

Now, it is not hard to see that the weights of the kernel will have an effect on the output. Different weights will help us solve different problems. Let us take a look at two examples.

In Figure 1 the vector $\vec{w} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$ represents the kernel's weights. All the values are the same and it is normalized (meaning the sum of all elements is 1); we find that the output this kernel produces is a smoothed (local averaged) version of the input.

It captures the broad trend in the input data while eliminating short term fluctuations.

Earlier we talk about the importance of recognizing edges, and edge is defined as a sharp change in the values of an input array. What we mean by this is that if two successive elements in the vector have a large absolute difference in values we are finding an edge. This would become more apparent if we graph the input vector. We can polot vector values in the y axis against the array indices, when the condition above is satisfied an edge would appear. A vector that may help us identify edges would look something like $\vec{w} = [\frac{1}{2}, -\frac{1}{2}]$. This will create large values in the output vector when a corner is found.

This was a brief introduction to one dimensional convolutions. In summary, a convolutional layer will help capture local patterns in input data because they connect only a small set of adjacent input values to an output value. This will help us identify higher level structures, like a nose and ears, whilst the result of this will help us identify even higher level structures like a face.

Before jumping into specific *CNNs*, let us discuss one last concept important for all this.

3 Pooling

As seen before, in a typical deep neural network, multiple convolution layers are stacked one after the other to recognize complex structures. One issue with this is that the convolutional layers are very sensitive to the location of the features in the input. Small variations in the position of the input features may result on different outputs. This is where *pooling* comes to play.

Pooling is when we perform a down-sampling operation, meaning we will create a lower resolution version of the feature map, this new map will still contain the important features, but at a lower precision. So even if this features are in varying locations in a high resolution map, they would be more or less the same in the low resolution one.

Pooling layers slide a small filter across the entire image; they capture a summary of the local patch (the area where the filter is). Two common algorithms for pooling filters are when we calculate the maximum value for each patch (**Max Pooling**) or when we can calculate the average of the patch (**Average Pooling**).

Please refer to Figure 2 where we find another image that I complete stole from *Math and Architectures of Deep Learning* by Krishnendu Chaudhury that showcases pooling.

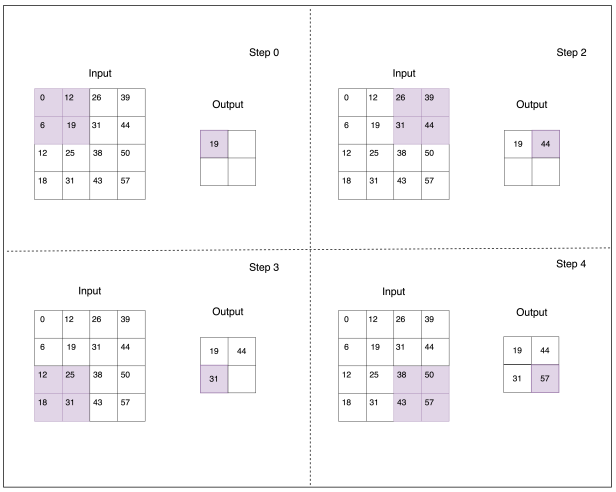

Figure 2: Max pooling using a 2x2 kernel with stride 2. taken from *Math and Architectures of Deep Learning* by Krishnendu Chaudhury

4 Image Classification - LeNet

For the sake of simplicity let us use the most popular example when someone is introduced to image classification. We need to build a classifier that takes a 28x28 image of a handwritten number and emits a label from 0 to 9 based on the digit contained in the image.

On this task we will use the MNIST data set which is a large collection of handwritten digits (0 through 9). Figure 3 shows an sample of this data set.

For more information on this data set and the algorithm, please refer to *Gradient Based Learning Applied to Document Recognition* [2], this is where LeCun proposed the neural network architecture. The

Figure 3: Sample images from MNIST data set, taken from *Gradient Based Learning Applied to Document Recognition* [2]

paper will make a deeper explanation on the concepts and process.

Please feel free to go back and forward between the explanation and Figure 4 while reading this part, try to understand which component is which, and what is happening on each layer.

The neural network, is formed by 3 convolutional layers, each with a kernel of 5x5 that are convolved with a stride of 1. They are the ones with the letter C in the diagram (C1, C3, C5).

The first convolutional layer produces 6 *feature maps* of size 28x28. A feature map is a 2D array of points, with a fixed size vector associated with every point [1]. A common example of a feature map is an image, the points are the pixels and the vectors are be the colors that compose the specific pixel (RGB).

Then we do the *Pooling* discussed a few paragraphs above. Which is tagged as sub-sampling in Figure 4. The pooling performs a local averaging of the feature map, reducing the resolution and reducing the output's sensitivity to shifts and distortions in the input.

We then do a second convolution layer that produces 16 feature maps of size 10x10. Now we do the pooling again; and the third and final convolution produces 120 feature maps of size 1x1 which conforms a 120 dimensional vector.

Every feature map is followed by a TanH activation layer. This function will make our network non-linear, it makes it more expressive because it can now model the output as a non-linear combination of inputs, opposed just a bunch of lines.

Finally the output feature map is passed through two fully-connected layers which produces a vector with 10 values, each representing the probability of the image belonging to that class.

5 Image Classification - VGG

The VGG family of networks were created by the Visual Geometry Group (hence the name) from the University of Oxford. Their idea was to increase the depth of the neural network by using an architecture with very small kernels (3x3). They demonstrated that by using 3x3 convolutions and networks with 16 to 19 layers, they could outperform different models. [3]

But, what is the advantage of using smaller kernels and making the network deeper? smaller kernels (3x3) allows more non-linearity, if you remember from a few paragraphs above, we applied a non-linear function to allow for more expressiveness, we used $TanH$ but here they used $ReLU$.

The VGG network comes in 5 different configurations, nevertheless all of them follow the same structure.

- They have 5 convolution blocks.
- Each block can have many convolution layers follow by a **MaxPool** layer at the end.
- All the convolution layers use the small kernel 3x3 with a stride of 1.
- Each convolution layer is followed by a non-linear function (ReLU)

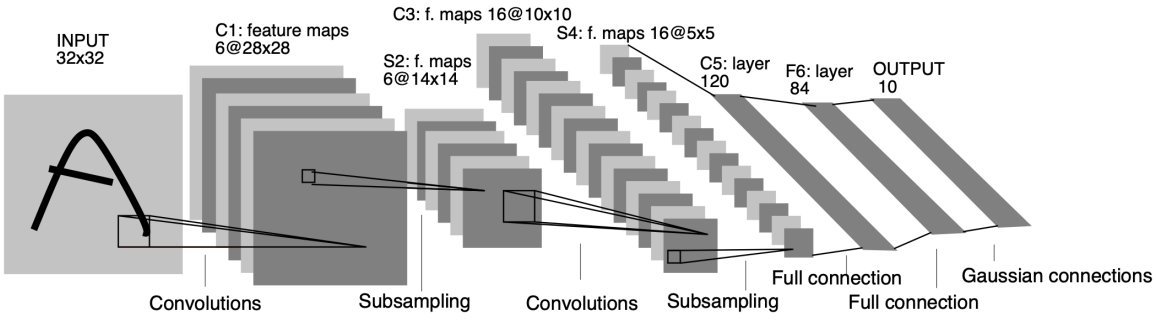

Figure 4: LeNet Architecture, taken from *Gradient Based Learning Applied to Document Recognition* [2]

- All architectures end with 3 Fully Connected Layers. These fully connected layers take the high-level features extracted by the convolutional layers and map them to class probabilities or predictions.

Figure 5 will show how a VGG-11 looks like. Please refer to it for more details.

You might be wondering: Why are you explaining another algorithm for image classification? Well, glad you asked. You will see how the dots connect in the following chapter.

6 Image segmentation

When we look at a picture, it is not only important to tell what the picture is about, but where things inside the image are, divide the image into different regions based on characteristics to identify boundaries and analyze the image more efficiently.

Say we have an image of a tiger walking, like we do in Figure 6. As a human you can do the two tasks easy, say the image contains a tiger, and drawing the borders that surround the tiger.

Above we have tackled the first problem, how to classify what is inside an image; but how can we make our computers also know where those things are inside the image? where to draw the boundaries? This is where image segmentation comes in. Image segmentation is the process of breaking an image into regions. This, with the intention to locate objects and boundaries (like lines, and curves) to make the image easy to analyze.

6.1 Fully Convolutional Networks (FCN)

In 2015 the paper "Fully Convolutional Networks for Semantic Segmentation" by Jonathan Long, Evan Shelhamer, and Trevor Darrell was published. This paper introduces *fully convolutional networks* (FCN) as an good approach for segmentation tasks.

Remember how in the convolutional networks, we have a fully connected layer at the end? This with the purpose to assign probabilities to classes or something similar. Well, FCNs eliminate the need for fully connected layers, and instead use only convolutional layers, this enables efficient pixel-level predictions.

FCNs removes the fully connected layer to preserve spatial information this becomes important when we care on the spatial context of each pixel (as we do in image segmentation). A quick example of this, Say we have a 3D feature map of size H, W, C, where H, W represents the spatial dimensions and C the number of channels. In a fully connected layer in order to flatten that we would have get the product of HWC. However, in a FCN the 3D feature map is preserved allowing the network to generate predictions with the same spatial dimensions as the input image.

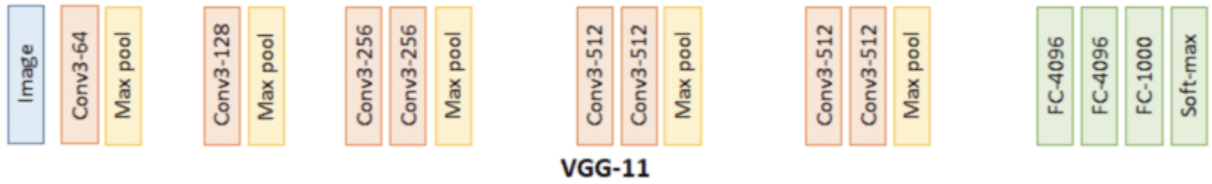

Figure 5: Block diagram of the VGG11 architecture, taken from https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11

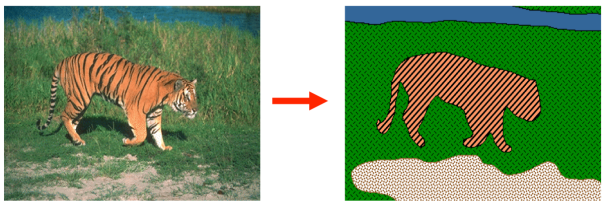

Figure 6: Example of image segmentation, taken from [5]

6.1.1 VGG as a FCN

Let us go back to VGG. The paper suggests some modification we can do to VGG to convert it into a FCN. In the original network, the fully connected layers are responsible for classifying the input image into different classes. Nevertheless, for segmentation, it is necessary to produce dense predictions at the pixel level. They replaced the fully connected layers with 1x1 convolutional layers. These convolutional layers retain the spatial information (like the 3D feature map example) of the input and allow the network to generate dense predictions. By converting the fully connected layers into convolutional layers, the FCN becomes capable of accepting inputs of arbitrary sizes and producing spatially dense output predictions.

6.1.2 Summary

Convolutional Neural Networks have revolutionized the field of computer vision and deep learning. They provide an effective framework for image and pattern recognition tasks. Their architectural components, including convolutional layers, pooling layers, and fully connected layers, work together to make final classification decisions. Examples of convolutional neural networks, such as LeNet and VGG, showcase their impressive performance in various image classification tasks. Additionally, the development of fully convolutional networks (FCNs) enables them to handle inputs of arbitrary sizes and work in segmentation tasks. FCNs have significantly advanced the accuracy and efficiency of image understanding and analysis, making them an indispensable tool in computer vision research and applications.

References

[1] Chaudhury, K. (2022). Math and architectures of Deep Learning. Manning. O'REILLY MEDIA.

[2] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[3] Simonyan, K., Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations (ICLR 2015), 1–14.

[4] Long, J., Shelhamer, E., Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[5] Yeung, S. Y. (n.d.). Personal website. Retrieved May 28, 2023, from https://ai.stanford.edu/syyeung/cvweb/index.html