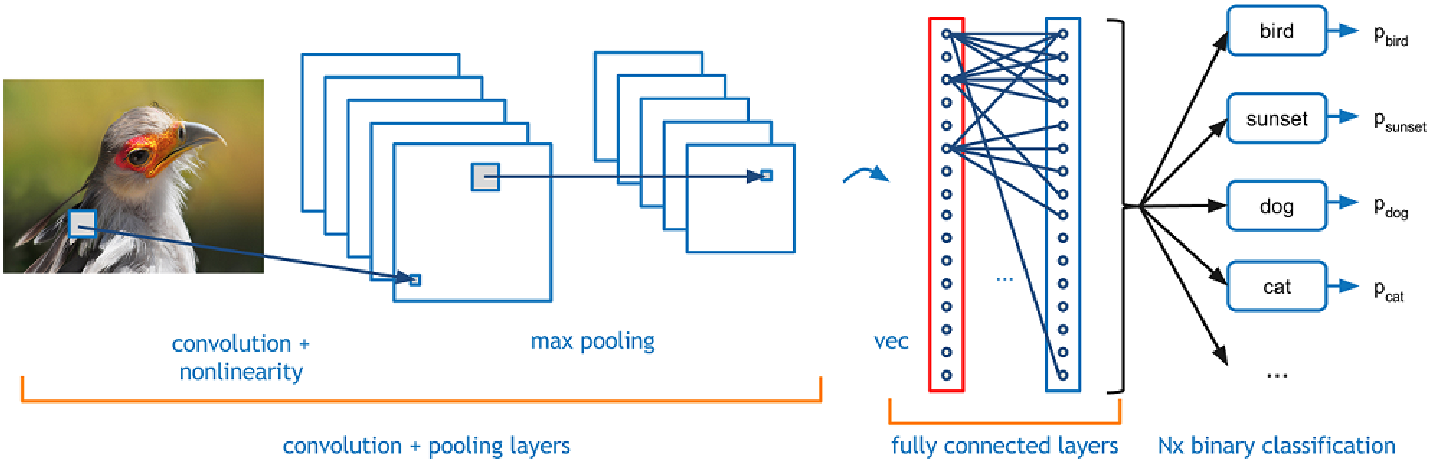
Adit Deshpande (/adeshpande3.github.io/)

CS Undergrad at UCLA ('19)

(/adeshpande3.github.io/)

Blog (/adeshpande3.github.io/) About (/adeshpande3.github.io/about) GitHub (https://github.com/adeshpande3) Projects (/adeshpande3.github.io/projects) Resume (/adeshpande3.github.io/resume.pdf)

A Beginner's Guide To Understanding Convolutional Neural Networks



(http://www.kdnuggets.com/2016/09/top-news-week-0905-0911.html)

Introduction

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time.Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let's take a look at how to use these CNNs for image classification.

The Problem Space

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.

What We See

What Computers See

Inputs and Outputs

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

What We Want the Computer to Do

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

Biological Connection

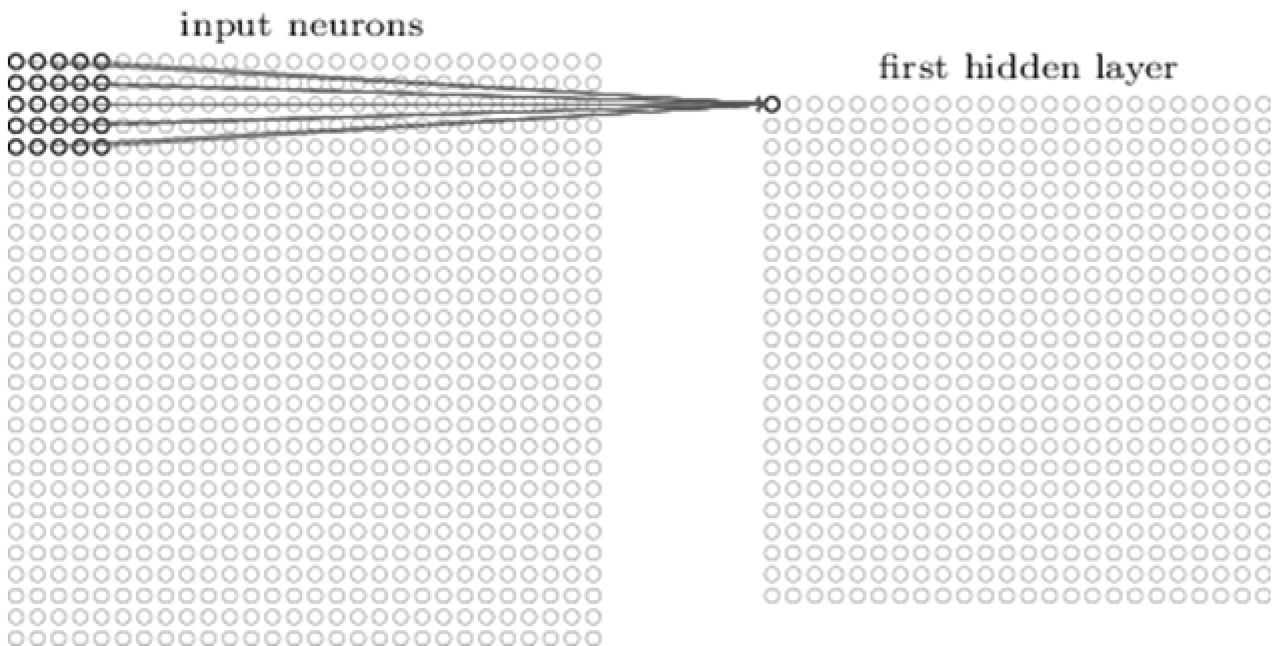
But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 (Video (https://www.youtube.com/watch?v=Cw5PKV9Rj3o)) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs.

Structure

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part is understanding what each of these layers do. So let's get into the most important one.

First Layer – Math Part

The first layer in a CNN is always a **Convolutional Layer**. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a 32 x 32 x 3 array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5 x 5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a **filter**(or sometimes referred to as a **neuron** or a **kernel**) and the region that it is shining over is called the **receptive field**. Now this filter is also an array of numbers (the numbers are called **weights** or **parameters**). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimensions of this filter is 5 x 5 x 3. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or **convolving**, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing **element wise multiplications**). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a 28 x 28 x 1 array of numbers, which we call an **activation map** or **feature map**. The reason you get a 28 x 28 array is that there are 784 different locations that a 5 x 5 filter can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

(Quick Note: Some of the images, including the one above, I used came from this terrific book, "Neural Networks and Deep Learning" (http://neuralnetworksanddeeplearning.com/) by Michael Nielsen. Strongly recommend.)

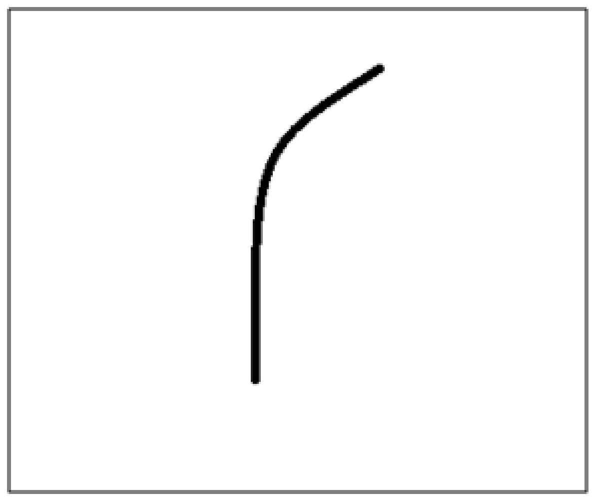
Let's say now we use two 5 x 5 x 3 filters instead of one. Then our output volume would be 28 x 28 x 2. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

First Layer – High Level Perspective

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as **feature identifiers**. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is 7 x 7 x 3 and is going to be a curve detector. (In this section, let's ignore the fact that the filter is 3 units deep and only consider the top depth slice of the filter and the image, for simplicity.)As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

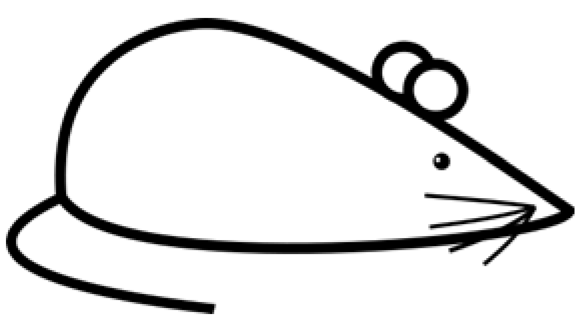
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

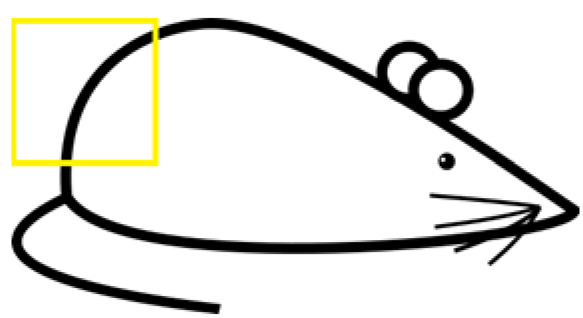


Visualization of a curve detector filter

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.

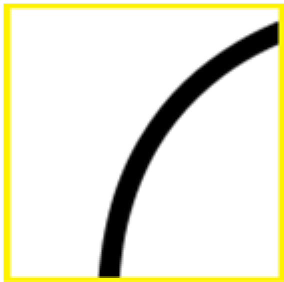


Original image



Visualization of the filter on the image

Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

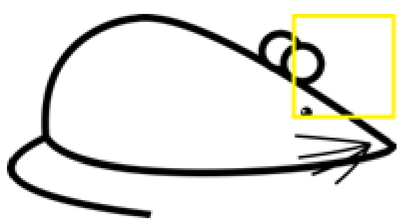
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

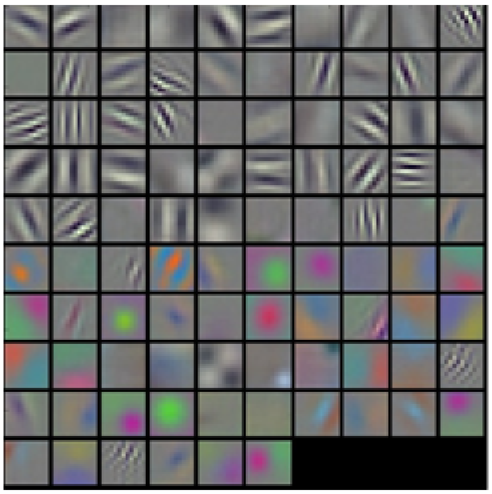
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there at mostly likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate (or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

Disclaimer: The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature it is looking for is in the input volume.



Visualizations of filters

(Quick Note: The above image came from Stanford's CS 231N course (http://cs231n.stanford.edu/) taught by Andrej Karpathy and Justin Johnson. Recommend for anyone looking for a deeper understanding of CNNs.)

Going Deeper Through the Network

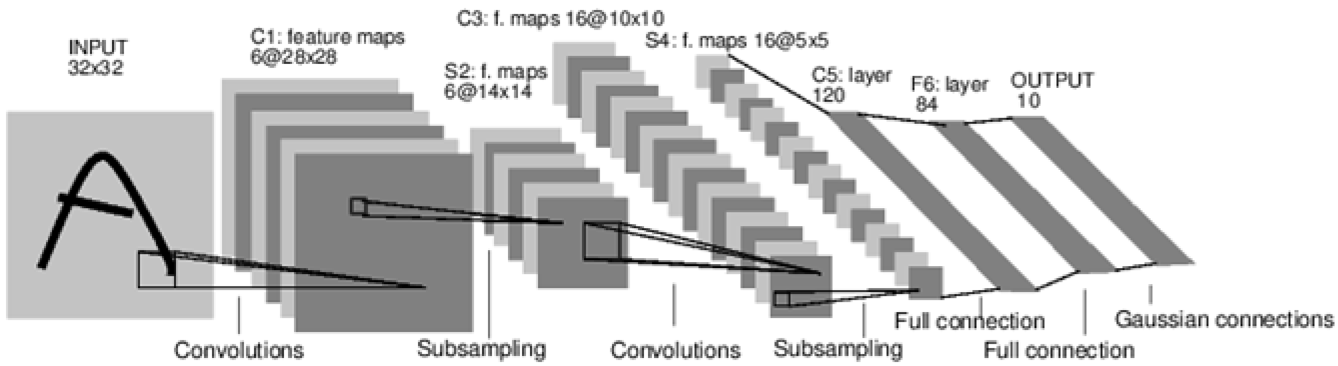
Now in a traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control overfitting. A classic CNN architecture would look like this.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order to predict whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a 28 x 28 x 3 volume (assuming we use three 5 x 5 x 3 filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. If you want more information about visualizing filters in ConvNets, Matt Zeiler and Rob Fergus had an excellent research paper (http://www.matthewzeiler.com/pubs/arxive2013/arxive2013.pdf) discussing the topic. Jason Yosinski also has a video (https://www.youtube.com/watch?v=AgkfIQ4IGaM) on YouTube that provides a great visual representation. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

Fully Connected Layer

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



A Full Convolutional Neural Network (LeNet)

Training (AKA:What Makes this Stuff Work)

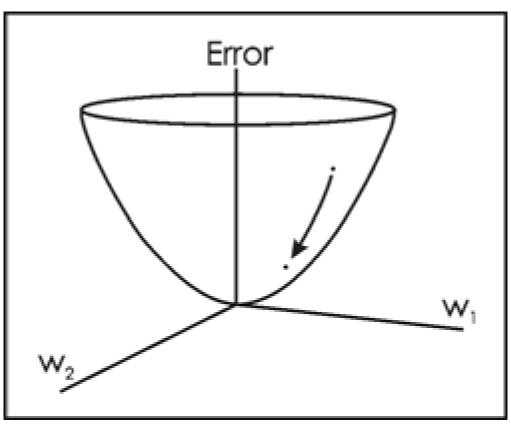
Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being given an image and a label is the training process that CNNs go through. Before getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the **forward pass**, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the **loss function** part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is ½ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right).In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in our case) most directly contributed to the loss (or error) of the network.



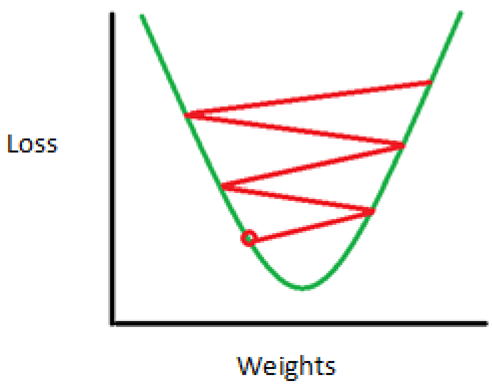
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

This is the mathematical equivalent of a **dL/dW** where W are the weights at a particular layer. Now, what we want to do is perform a **backward pass** through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the **weight update**. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

The **learning rate** is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.



Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

Testing

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

How Companies Use CNNs

Data, data, data. The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

Disclaimer

While this post should be a good start to understanding CNNs, it is by no means a comprehensive overview. Things not discussed in this post include the nonlinear and pooling layers as well as hyperparameters of the network such as filter sizes, stride, and padding. Topics like network architecture, batch normalization, vanishing gradients, dropout, initialization techniques, non-convex optimization,biases, choices of loss functions, data augmentation,regularization methods, computational considerations, modifications of backpropagation, and more were also not discussed (yet).

Link to Part 2 (https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/)

Dueces.

Sources (/assets/Sources.txt)

Tweet

Written on July 20, 2016

132 Comments **Adit Deshpande** 1 Login

Recommend 116 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Chong Yu • a year ago

Very well written and nice examples!

12 • Reply • Share ›

Brandon Rohrer • a year ago

Very nice tutorial! It's the most concrete description of CNNs that I've seen.

8 • Reply • Share ›

Bharat → Brandon Rohrer • 4 months ago

Hi Adit and Brandon,

Can you please suggest me any link or material where I can easily understand how back propagation obtain filters/kernals.

I know working of backpropation for finding optimum weights but I want to understand how does it obtain that best filters from the data provided.

• Reply • Share ›

adeshpande3 Mod → Bharat • 4 months ago

http://cs231n.github.io/opt...

http://jefkine.com/general/...

https://mattmazur.com/2015/...

2 ^ | v • Reply • Share ›

Ritvik Raj • 2 months ago

Amazing tutorial. It helped me a lot.

5 ^ | v • Reply • Share ›

Jim Morgenstern • a year ago

Excellent presentation; i last played with neural nets almost 20 years ago and have steered clear of them. so this was very useful.
Also, i would like to mention:
* HSV vs RGB: RGB has a lot of correlation between the color planes, and especially so if derived by a Bayer imager so HSV strikes me as a better choice for image analysis.
* One of the comments here is against using an extensive set of training images but this is what one needs to do with CNN -- inherent in the whole CNN philosophy is the sense that image classification is really not a linear solution so to express all of the non-linearities requires large data sets. One of the consistent complaints about CNN is that when it does not work the only solution is to add more training sets because one has yet to arrive at correct weights in all the layers. Linear solutions, when they do not work at least provide some error [confidence] measures that allow one to predict when they do not work and what parameters to poke around in to improve performance; but CNN [to the best of my admittedly limited knowledge] do not provide confidence estimates or indications of what is not working -- the CNN simply gives the wrong answer but with total confidence.

4 ^ | v • Reply • Share ›

BounceGodzilla ➦ Jim Morgenstern • a year ago

I think HSV would probably be a quite poor choice of color representation for the following two reasons:

1. The hue component jumps discontinuously as the color space includes a branch cut, normally in the color red, so you jump 2 pi radians, or 360 degrees, when the color goes from purple-ish red to yellow-ish red, which would lead the filters to detect edges in the image where there are none. Besides, you would get different behaviors depending on at which hue you chose to put the branch cut, so suddenly you would have to choose carefully where you choose to put it, which would feel a bit awkward.

2. You have a singularity (cause by the branch point associated with the branch cut) in the color grey, as the hue becomes increasingly sensitive to color changes the closer you get to grey (to the point where it becomes undefined in the color grey) which would activate a lot of

feature detectors where there is really nothing of interest if you only introduce a tiny amount of noise.

The fact that the human eye uses cone cells that are sensitive to red, green and blue respectively (hence RGB) indicates that a similar color representation would probably be well suited also for convolutional neural networks.

For similar reasons, in some situations where you specify a point on a sphere, it is actually better to use Cartesian coordinates than spherical coordinates because Cartesian coordinates include no branch points or branch cuts (which spherical coordinates do), even though this means that you need to use one extra coordinate to represent the point.

2 • Reply • Share ›

jake → BounceGodzilla • 5 months ago

What about LAB color space? Isn't that supposed to better mimic the human eye, in that colors with a larger euclidean distiance in LAB will also 'look' further apart to the human eye?

• Reply • Share ›

James Page → BounceGodzilla • 7 months ago

I still don't understand.

• Reply • Share ›

BounceGodzilla → James Page • 7 months ago

What is it you don't understand?

• Reply • Share ›

SpaceKadet → BounceGodzilla • a year ago

Eyes use RGB color primaries because they provide an optimal separation into separate but overlapping signals early in the visual system.
Because these red, green and blue filters overlap, they permit a unique triplet of values across the visible spectrum.
There is no elegant way to directly detect and encode continuous hue biologically.
Even spectrum analyzers use arrays of narrow-band filters acting as cones encoding hundreds of wavelengths with narrow bandwidths.

Color wheels provide a perfect counter-example to the above claim, as there is no discontinuity from purple to orange through the red.
Not perfectly matching the CIE curve and its manifolds does not preclude creating a very close analogy, and we perceive and process vision in HSV for the higher visual perception tasks.

You can tell someone that a color is yellowish-orange or bluish green. You do not think of colors by their RGB coordinates even approximately. Additionally, people who are color-anomalous can match hues along much of the spectrum, but lacking one cone, they will never be able to describe any RGB triad.

Lastly, for reasons of acceptable compression (critical when one's computations involve teraFLOPs) RGB is horribly redundant. An image of mainly gray-scale colors would have almost identical values in each of red, green and blue in every pixel, all requiring processing.

Changes in HSV have an additional huge advantage: Being orthogonal, with "value" approximating brightness, shadows can be removed from images, allowing processing of real image elements, and light sources can be identified by shadows using only the value channel.

The amount of information loss due to a smoothed version of HSV is minimal compared to either the speed gain or the losses due to neural approximations in processing.

• Reply • Share ›

BounceGodzilla ➦ SpaceKadet • a year ago

If you wouldn't have a discontinuity in the hue channel, you wouldn't get back to the same hue value again when going one lap around the color wheel (which you should, since you should end up on the same color). Open ms paint for example and look at what the hue approaches as the color approaches red when starting in the color purple, and then look at what the hue approaches as the color approaches red when starting in the color orange. You do have a discontinuity there, as the hue approaches two different values when the color approeaches red, depending on what color you start in.

Otherwise, as the discontinuity might be located in another color than red (depending on where the branch cut has been made), consider what color you have when the hue is at its minimum, and what color you have when the hue is at its maximum, and you realize that those are the same color. And in that color, there will be a discontinuity in the hue.

HSV may possibly bring some benefits that you don't have when using the RGB color space, but I have never seen it being used in a research paper about convolutional neural networks.

• Reply • Share ›

SpaceKadet ➦ BounceGodzilla • a year ago

There are stable point to point reversible transforms in the literature between these two color spaces. The 1976 CIE curve is still a simplification of human vision, based upon the original sample of young, healthy caucasian undergrads. The only requirement upon any system is consistent mapping and a formula permitting the loop to land one back at the same place in RGB space (of which there are now a dozen popular conflicting models!) The system fails in several modes in all spaces. McAdam ellipses have discontinuities, and the complete XYZ manifold implies at least two additional spatially sparse cone channels.

At the end of the day, we think in hues, saturations and brightnesses to distinguish shadows, foreground/background, human health, food quality, color temperature in metal working, etc. RGB fails dismally in subtle hue shifts and demands much more processing for these applications.

^ | v • Reply • Share ›

piikey • 2 months ago

Its a great tutorial. However, there is one missing point (as long as I did not miss it). Its basically the explanation, why NNs really can learn. I have never found this explanation so far in any book or article I read, and since I am not a student anymore who can ask a professor and its the first time I really try to implement a NN (by self teaching), I just try to ask you about my understanding.

The problem is in the fully connected networks at the end. As we know, when the weights are initialized to 0, then nothing is learnt at all. When they are for example initialized all to 0.5 then the network cannot break its symmetry, so for every node in the hidden layer in the fully connected network you will get a copy of all the weights connected to this node.

Thats the moment, when every article says, you have to initialize it randomly (or perhaps with some normal distribution, I am not yet sure about this), but there is nowhere a real interpretation, what this means. Now, in my opinion thats the real basis of the whole idea in NNs. Note, its just my understanding, perhaps I am totally wrong, thats why I am open here for a discussion. So my interpretation is, the NN is prepared with kind of randomized filters (represented by the hidden nodes).These filters then are adapted during the learning process. However, you can only learn as much as you have filters. So, feature extraction and therefore learning is only possible because we have these random filters / hidden nodes.

So the whole learning as far as I understand is like having a lot of placeholders / hidden nodes / features which you hope are so different that all of them will finde something special in their connections. Thats the learning!

What do you think about this?

1 ^ | v • Reply • Share ›

tarek • 10 months ago

is it possible to do a supervised classification with multi kernel learning using CNN ??

1 ^ | v • Reply • Share ›

adeshpande3 Mod → tarek • 6 months ago

Sorry for the late reply. What exactly do you mean by "multi kernel learning"?

^ | v • Reply • Share ›

Rajesh Govindan • a year ago

Thanks for the short tutorial - great effort! One quick comment - neural networks, in my understanding, do not generally have convex error functions (like you would with a regression problem). Hence, I think the images used in the explanation are not fully representative. But, having said that, you have distilled the concept very well. Thank you.

1 ^ | v • Reply • Share ›

adeshpande3 Mod ➦ Rajesh Govindan • a year ago

Thanks for the input Rajesh! And good catch, I was actually under the impression that convex optimization was used to minimize loss (I was thinking about a softmax regression method), but I poked around and found out that NNs do generally use non-convex approaches because it doesn't really matter what local minima you end up with while doing learning/optimization. Here are some good links for anyone interested.

http://stats.stackexchange....
https://www.quora.com/Why-i...

1 ^ | ˅ • Reply • Share ›

Manaswi ➦ adeshpande3 • a year ago

I would like to point out one more observation, In the 3D image visualization of the error function error won't be minimum when both the weights are zero. Although I understand you just took a standard image for least squared error minimization as analogy in explaining the concept. I feel that if this point is mentioned it will be a good help for beginners understanding cnn concept.

^ | ˅ • Reply • Share ›

adeshpande3 Mod ➦ Manaswi • a year ago

Yes, that's a good point. I'll be sure to add a little note to that image. Thanks!

^ | ˅ • Reply • Share ›

Rohan Varma • a year ago

Excellent discussion! I especially like your mathematical description of filters and how they tend to "activate" when convoluting over certain regions of input. The discussion on how additional convolution steps produce more "high level" filters is also excellent - this is a pretty confusing step when trying to understand the idea that detecting complex entities such as a paw is pretty much the detection of a bunch of lower level features put together.

A few quick comments -

1) You seem to state that the RGB matrix is the only way an image can be represented to a computer, at least for neural network training purposes ("These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer"). While one could certainly argue that standard RGB input works best for image classification, many studies have shown that using the HSV color space may yield better results. HSV (Hue-Saturation-Value) represents pixel intensities in cylindrical coordinates, and better mimics how humans perceive color than standard RGB. As you state, image thresholding is an essential step in any image processing algorithm, and HSV seems to better retain pixel information by providing values for intensity and shade differences. Obviously there's two sides to this story but it's something to consider - http://www.eletrica.ufpr.br... is a study that yields positive results for HSV, while

see more

1 ^ | ˅ • Reply • Share ›

adeshpande3 Mod → Rohan Varma • a year ago

Thanks for response! I totally agree with you on the higher level features being confusing to explain. As for the data representation, I actually didn't know about the HSV color space. It'd be interesting to see what kind of filters get created when you have that instead of the RGB values. As you said, they provide values for intensity and shade differences, so yeah I could definitely see that representing the data better.

From what I understand, the difference between batch and stochastic lies in the number of samples that you are computing the gradient over. Stochastic is popular because it is computationally a lot faster, which would be beneficial to networks with large datasets. http://stats.stackexchange....

Yes, you're right in that the convolutional part of CNNs makes the network somewhat spatially invariant. As a side note, there's also some interesting developments with spatial transformer networks (STNs) which also help the spatial invariance, to my knowledge. https://arxiv.org/pdf/1506....

I've actually been thinking about your last point as well. I think you put it well in that an accurate statement about NNs would be that without a "sufficient" (And I guess this word is also subjective) amount of data, you can't get the CNN (especially when you start talking about architectures like 10, 15+ layers) to learn and tune the weights of the features it is looking at. But yes I definitely do agree with some aspects of your statement. I think that deep learning has grown over the past 4-5 years mainly because of compute power and accessibility to data, and so I think at some point, like you said, we're going to have to create new algorithmic approaches to deal with those "inherent inefficiencies of neural networks" instead of just throwing more data at it.

1 • Reply • Share ›

James Hahn • 4 months ago

Can you explain softmax please? I think I understand every step, but every time I google how to do softmax, nobody can explain it well. From what I understand, you have your input 32x32x3 -> conv layer goes down to 28x28 -> ReLU normalizes -> conv layer goes down to 24x24 -> ReLU -> conv layer goes down to 19x19 -> ReLU -> conv layer goes down to 15x15 -> fully connected with probability distribution of classes . Is this correct? Does the conv layer decrease in size for every iteration? Is the fully connected layer the exact same thing as softmax? Is that where you find your probabilities? If so, why do some architectures that I see have several fully connected layers? Doesn't quite make sense to me.

Also, when you do the back pass, the dL/dW in your equation is the exact same value as the Loss function (error) value, right?

Lastly, when I hear the term 'neuron' being thrown around in other tutorials, is that referring to every convolutional layer? Or is it every filter?

I love the rest of the article, it's very helpful and provides a visually appealing approach to a visual problem without obfuscating the entire algorithm with complicated symbols and equations.

1 • Reply • Share ›

adeshpande3 Mod → James Hahn • 4 months ago

Just think of the softmax as being an operation that takes in a series of numbers and outputs the values in a normalized form where the numbers now range from 0 to 1, and add up to 1 (which is why we can think of them as "probabilities"). Check out the Softmax section here https://github.com/Kulbear/....

The outputs for each conv layer will always decrease the spatial size of the input volume, given that you don't have padding (Read Part 2 if you're not sure how padding works).

A fully connected layer is not really the same thing as a softmax. A FC layer is like any other layer in a normal neural network (as opposed to a CNN). It's just a layer that has some input nodes, some output nodes, and linear operations and activation functions in between.

Once you pass the outputs of your final FC layer through a softmax layer, that's when you get probabilities.

Several FC functions just mean that the output volumes from the convolutional layers go through several different layers before getting turned into probabilities. Just adds to the network expressivity of the model.

see more

1 • Reply • Share ›

Randy Welt • a year ago

ah, nothing new and not well written. looking for similiarity is correlation not convolution. how to connect input feature maps to output feature maps? even Kapathy or any other blog like this one here does not say anything about it. in the forward pass using softmax probs. ,but explaining MSE error in the backward pass.. ...if you are new to CNN , this blog is just confusing

1 • Reply • Share ›

Rishabh Venketesh → Randy Welt • a year ago

What is the difference between convolution and correlation?
From what I understand Convolution is correlation with a rotated filter mask.
Am I right?

7 • Reply • Share ›

Randy Welt → Rishabh Venketesh • a year ago

yes correct. so in case of a symmetrical filter cross correlation and convolution are identical.

• Reply • Share ›

Kai → Randy Welt • a year ago

do you have examples of a good intro?

• Reply • Share ›

Randy Welt → Kai • a year ago

Randy Welt ➦ Kai • a year ago

read Kapathy's blog , watch his MOOC + UFLDL Tutorial, read LeNet5 paper ... . Unfortunately I could not find any good compact 'all at once ' tutorial yet.

• Reply • Share ›

Liverio • 2 days ago

clear explanation. good work!

• Reply • Share ›

Nael Hailemariam • 3 days ago

Thanks for your well-written article, Adit! Keep staying awesome!

• Reply • Share ›

Sameer Mahajan • 21 days ago

Very well written **@adeshpande3** Keep writing!

• Reply • Share ›

Amit Prasad • a month ago

Well written for starters!

• Reply • Share ›

tuhh • a month ago

Hi, Thank you very much for the article. There's one thing I'm not quite clear. You said "These multiplications are all summed up (mathematically speaking, this
would be 75 multiplications in total). So now you have a single number." But after summing up the multiplications we need to put the sum into a activation function, and then we get a number. Is it correct?

• Reply • Share ›

tuhh ➦ tuhh • a month ago

I got it! Thank you very much!

• Reply • Share ›

adeshpande3 Mod ➦ tuhh • a month ago

Yes, its just that I'm separating the steps a little. The convolutional layer just does the element wise multiplications and summations and then the nonlinear activation layer that always follows it will do that operation. Just a case of semantics

• Reply • Share ›

Shubham Aggarwal • a month ago

@adeshpande3 awesome explanation and can you please what's the significance of depth of filters ?

• Reply • Share ›

adeshpande3 Mod ➦ Shubham Aggarwal • a month ago

The depth of each filter will always equal the depth of the input volume. If you're asking more of how many filters to use, then this might help https://stats.stackexchange... TLDR basically it depends on the complexity of your image processing task and then same of your network.

2 ^ | v • Reply • Share ›

Hamid Aksasse • a month ago

thank you very much. i begin understanding how a CNN works by reading you post.

(mailto:adeshpande3@g.ucla.edu)

(https://www.facebook.com/adit.deshpande.5) (https://github.com/adeshpande3)

(https://instagram.com/thejugglinguy)

(https://www.linkedin.com/in/aditdeshpande) (/adeshpande3.github.io/feed.xml)
(https://www.twitter.com/aditdeshpande3)