



Adit Deshpande (/adeshpande3.github.io/)

CS Undergrad at UCLA ('19)

(/adeshpande3.github.io/)

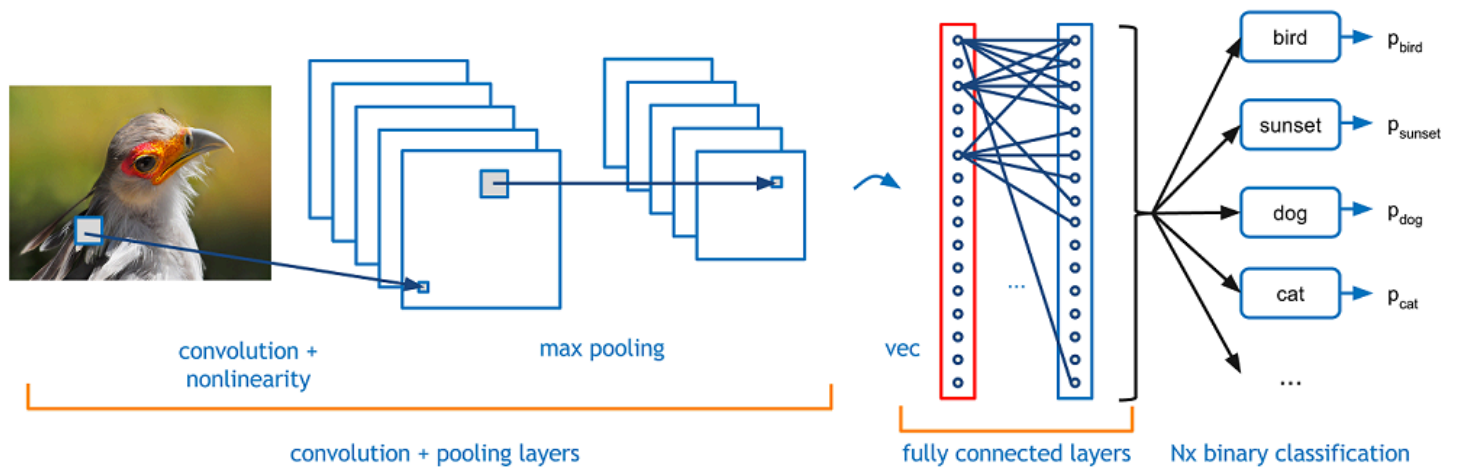
Blog (/adeshpande3.github.io/) About (/adeshpande3.github.io/about)

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

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 84 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

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 $32 \times 32 \times 3$ 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 480×480 . The representative array will be $480 \times 480 \times 3$. 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.

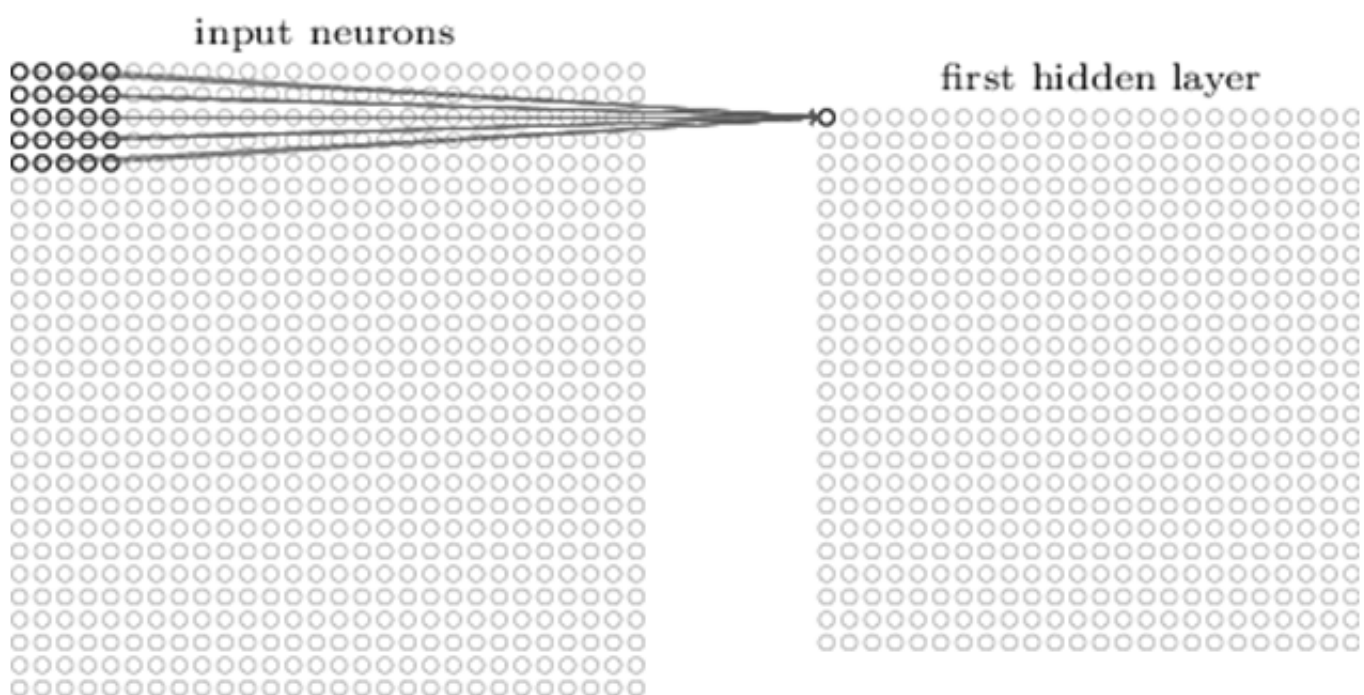
Structure

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 $32 \times 32 \times 3$ 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 5×5 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 $5 \times 5 \times 3$. 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 $28 \times 28 \times 1$ array of numbers, which we call an **activation map** or **feature map**. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter can fit on a 32×32 input image. These 784 numbers are mapped to a 28×28 array.



Visualization of 5×5 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 $5 \times 5 \times 3$ filters instead of one. Then our output volume would be $28 \times 28 \times 2$. 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 $7 \times 7 \times 3$ 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



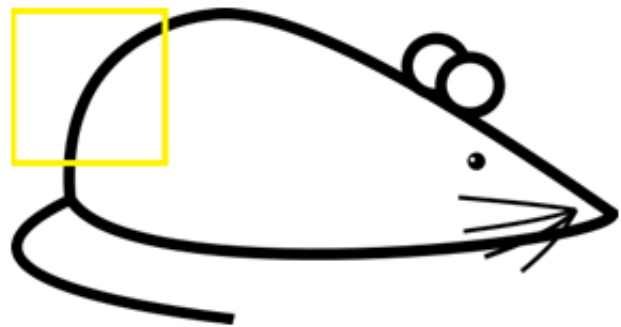
Visualization of a curve detector filter

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

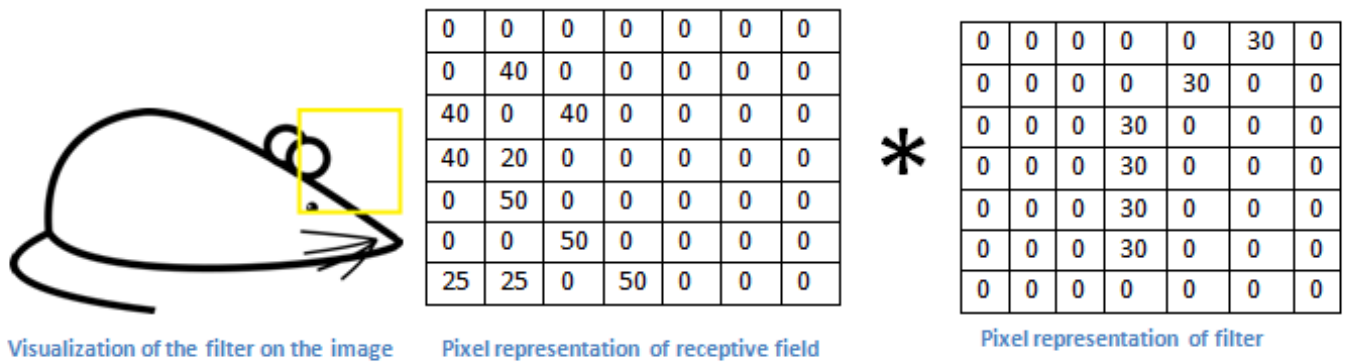
*

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!)

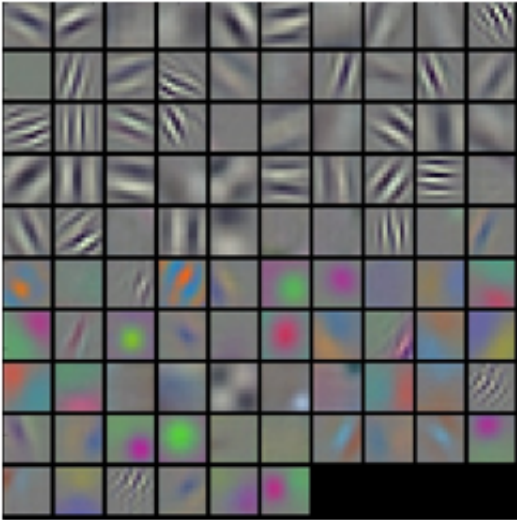
Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.



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 are mostly likely to be curves in the picture. In this example, the top left value of our 28 x 28 x 1 activation map 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

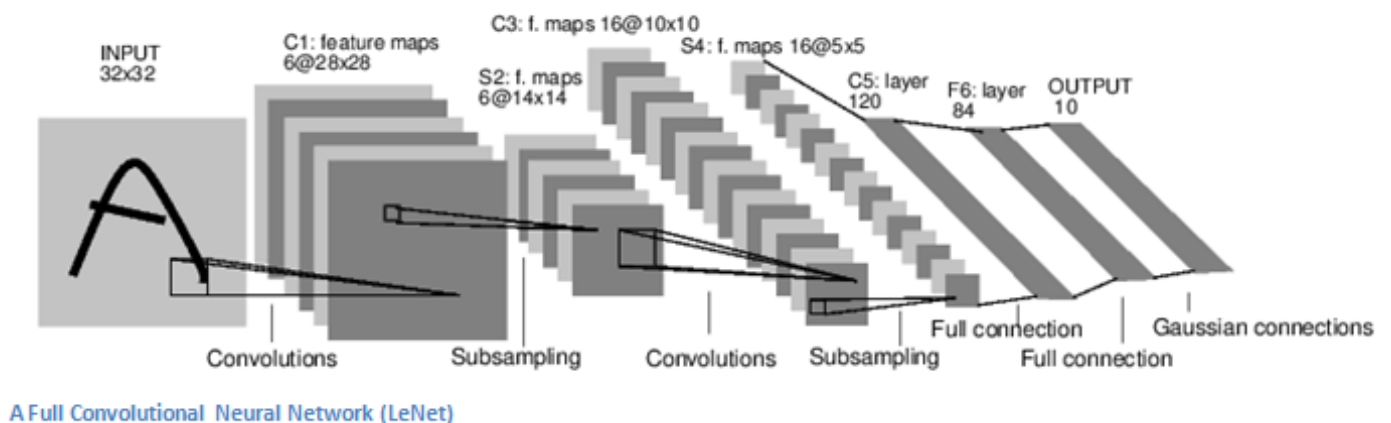
The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would 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 $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ 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=AgkflQ4lGaM>) 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.



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**.

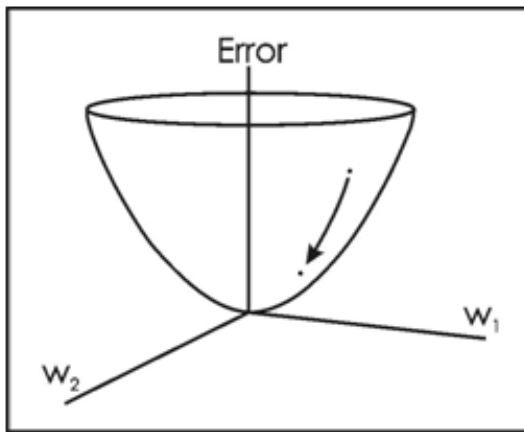
Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values 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]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ 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.



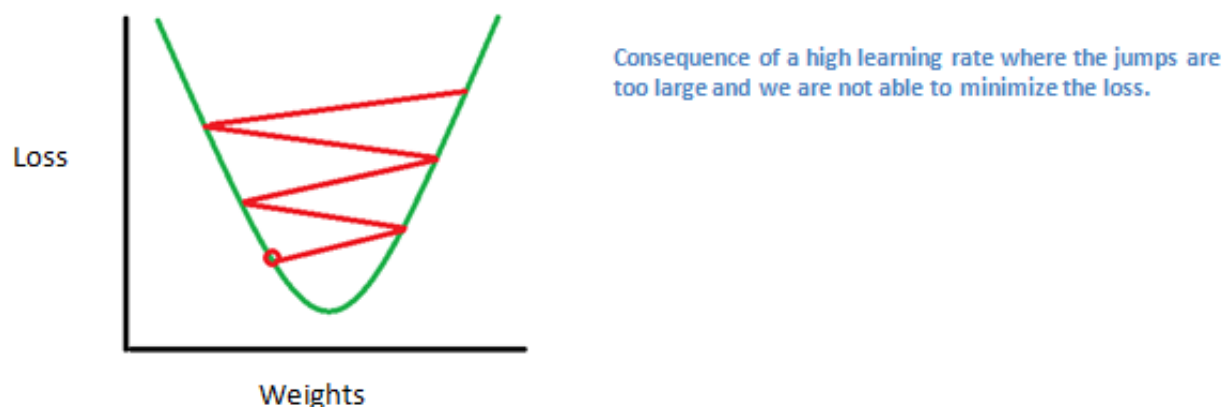
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves 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.



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

83 Comments **Adit Deshpande**

1 Login ▾

♥ Recommend 68

🔗 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!

4 ^ | v • Reply • Share ›



Brandon Rohrer • a year ago

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

4 ^ | v • Reply • Share ›



Jim Meyer • a year ago



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.

2 ^ | 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 2π 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 ^ | v • Reply • Share ›



jake → BounceGodzilla • a month ago

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

^ | v • Reply • Share ›



James Page → BounceGodzilla • 3 months ago

I still don't understand.

^ | v • Reply • Share ›



BounceGodzilla → James Page • 3 months ago

What is it you don't understand?

^ | v • Reply • Share ›



SpaceKadet → BounceGodzilla • 10 months 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

[see more](#)

^ | v • Reply • Share ›



BounceGodzilla → SpaceKadet • 10 months 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 approaches 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.

^ | v • Reply • Share ›



SpaceKadet → BounceGodzilla • 10 months 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 ›



tarek • 5 months ago

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

1 ^ | v • Reply • Share ›



adeshpande3 Mod → tarek • 2 months ago

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

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 ^ | v • 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.

^ | v • 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!

^ | v • 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

[see more](#)

1 ^ | v • 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....>

[see more](#)

1 ^ | v • 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 Kapatthy 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 ^ | v • 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?

^ | v • Reply • Share ›



Randy Welt → Rishabh Venketesh • a year ago

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

^ | v • Reply • Share ›



Kai → Randy Welt • a year ago

do you have examples of a good intro?

^ | v • Reply • Share ›



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.

^ | v • Reply • Share ›



LluviaMaker • 3 days ago

This was an very AWESOME overview. Thank you so much!

^ | v • Reply • Share ›



Rowan G1 • 3 days ago

Hope you write many more articles. Almost feel like I understand this stuff now! Question:
for convolution, in the case of using 2 off $5 * 5 * 3$ filters, to produce a $28 * 28 * 2$ array, is the
"2" the "number of input channels" as discussed in <https://www.tensorflow.org/...>

^ | v • Reply • Share ›



Ankit kumar • 5 days ago

Thanks for the great intro....)

^ | v • Reply • Share ›



Yichen Xu • 13 days ago

Hello.

Your introduction to CNN is so great! i benefit a lot from it!

I am planning to translate it to chinese.

could i have your approval?

^ | v • Reply • Share ›

**adeshpande3** Mod → Yichen Xu • 13 days ago

Yeah, just include the original link to this article when you post it

^ | v • Reply • Share ›

**Yichen Xu** → adeshpande3 • 11 days ago

okay, thank you;)

^ | v • Reply • Share ›

**Sqandaraneh** • 15 days ago

That was an AWESOME* read as my first article about the topic. Thank you very much!

* I rarely use this word.

^ | v • Reply • Share ›

**irving zhang** • 16 days ago

Excellent examples! Thank you Adit!

^ | v • Reply • Share ›

**Vishwam Pandya** • 18 days ago

Beautiful illustration. Can you throw some light on DCNN?

^ | v • Reply • Share ›

**adeshpande3** Mod → Vishwam Pandya • 18 days ago

Deep convolutional neural networks? They are the same as CNNs except they have multiple layers of conv/pooling/relu, which are pretty much there in all CNNs anyway.

^ | v • Reply • Share ›

**Kelly D** • 19 days ago

Great article. Question -- and please excuse my ignorance.

(a) When you showcase 75 multiplications in total, that is PER CEL right? It is the $5 \times 5 \times 3 = 75$ receptive field and when it convolves to Cel #2 -- it does another 75, correct?

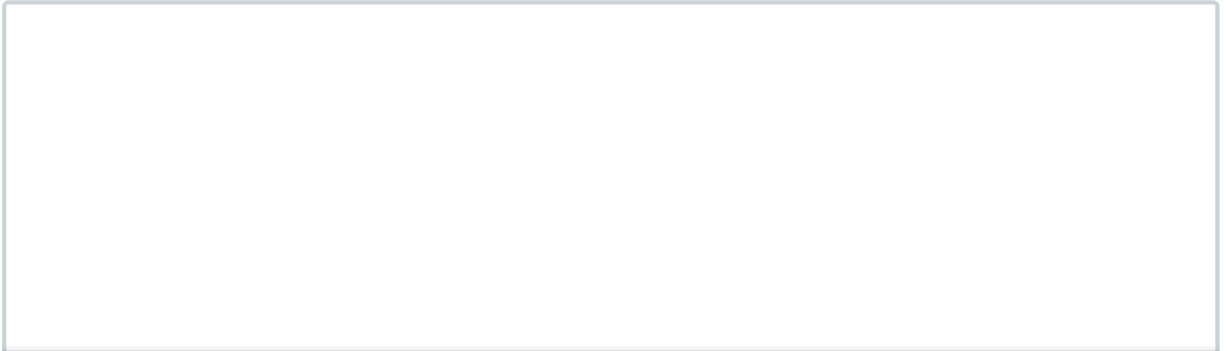
(b) The 28×28 activation map (784 different locations) that I'm left with after convolving over each cel with a 5×5 filter, how did I get to that # ... sorry, stumped...

^ | v • Reply • Share ›

**adeshpande3** Mod → Kelly D • 19 days ago

Don't worry man! Asking questions is the best way to understand. For a) Yes you're correct about 75 multiplications for each location on the output map. For b), think about things at a smaller level. Think about a 7×7 image and a 3×3 filter. Start the filter at the top left and move the filter one column at a time until you reach the end. You'll see that you have 5 different locations. Now you move the filter one unit down

you'll see that you have 3 different locations. Now you move the filter one unit down and repeat. So you get a 5x5 output volume. Now, just take that concept and try with a 32x32 image and a 5x5 filter, and you'll see how we get the 28x28 activation map. Take out a piece of paper and draw it out, that always helps. Also, take a look at the GIF about halfway through this page <http://cs231n.github.io/con....> It shows the same example, except with a stride of 2 instead of 1. And check out



[see more](#)

^ | v • [Reply](#) • [Share](#) ›



Kelly D → [adeshpande3](#) • 6 days ago

Thanks so much - I get it now - I did it on paper and then I was like -- man ... how did i not get this the 1st time....lol. I made a slide with animation of it over a picture if you want it ... you can put whatever picture you want behind it. If you want it, google me (Paul Zikopoulos or @BigData_paulz on Twitter and you will find out what I'm about and what I do) and I will send it to you. Great series BTW. You are a very talented communicator...

^ | v • [Reply](#) • [Share](#) ›



Hamid Aksasse • 21 days ago

thank you for this best and nice article about CNNs . its what i was looking for . thank you again

^ | v • [Reply](#) • [Share](#) ›



曹博堯 • 23 days ago

This is the best post I've ever read about CNN. Big THANKS, really. Please keep going.

^ | v • [Reply](#) • [Share](#) ›



vamsi krishna • 24 days ago

I was trying find a good resource to understand CNN. I have gone through a lot of them but was unable to get a good grasp how it works. This one clearly stands out for me. The concepts are very clearly explained in simple terms.

^ | v • [Reply](#) • [Share](#) ›



dilip naga kumar m • 25 days ago

Thank you, Adit Deshpande, for the clear explanation. Very useful information.



Thank you Adit Deshpande, for the clear explanation. very useful information.

Can you please recommend best way (course or book) to learn Neural nets. I am looking for the one to make beginner a pro. My ultimate goal is to able to do image processing for the systems like self driving cars, drones etc

^ | v • Reply • Share ›



adeshpande3 Mod → dilip naga kumar m • 25 days ago

<http://neuralnetworksanddee...>

Or

<https://www.coursera.org/le...>

^ | v • Reply • Share ›



dilip naga kumar m → adeshpande3 • 24 days ago

Thank you Very much Adit

^ | v • Reply • Share ›



vjk1996 • a month ago

Very helpful for beginners. Crystal clear explanation provided!

P.S It would be helpful if you can share any CNN codes related to vision/image recognition.

kams1996@gmail.com

Thank You.

^ | v • Reply • Share ›



adeshpande3 Mod → vjk1996 • a month ago

<https://www.tensorflow.org/...>

^ | v • Reply • Share ›



Dashanka Nadeeshan • a month ago

Thank you. Good explanation and informative for beginners with understandable manner.

^ | v • Reply • Share ›



Dashanka Nadeeshan • 2 months ago

This is great Thanks... (Y)

^ | v • Reply • Share ›



Debdeep Basu • 2 months ago

Amazing article!

^ | v • Reply • Share ›



Yasar Hayat • 3 months ago

Thanks a lot man <3

^ | v • Reply • Share ›

^ | v • Reply • Share ›



Vijeta Kiran Deshpande • 3 months ago

Excellent article Adit.

^ | v • Reply • Share ›



Eddie Forson • 3 months ago

Great introductory article! I am currently learning about CNNs and going through a few resources. I just wanted to point out something that confused me in your post: "This is where we take all the weights of the filters and update them so that they change in the direction of the gradient" - should we not be updating the weights so that they change in the *opposite* direction to the gradient, because otherwise we would be increasing the loss?

^ | v • Reply • Share ›

Load more comments

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy

(1) Weird Trick to Regrow Hair Every Night (Do This Daily)

Your Lost Hair Can Be Regrown in Days - Try This Simple Trick

[Learn More](#)

Sponsored by **FollicleRX**

[Report ad](#)

(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>)
