



**Adit Deshpande** (/adeshpande3.github.io/)

Engineering at Forward | UCLA CS '19

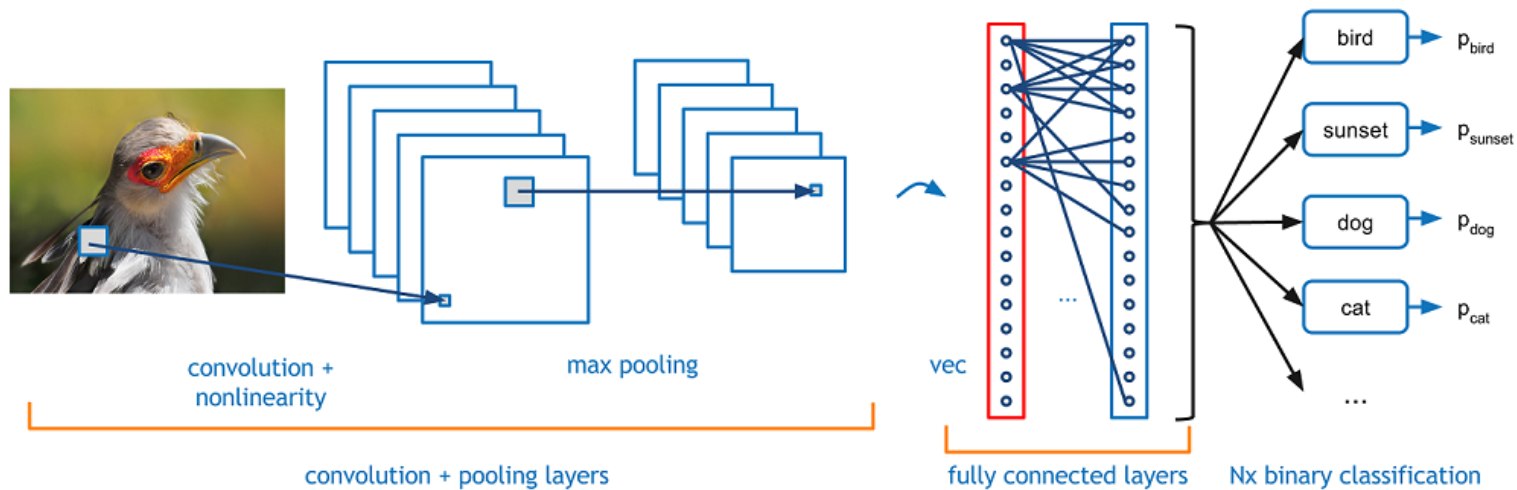
(/adeshpande3.github.io/)

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



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

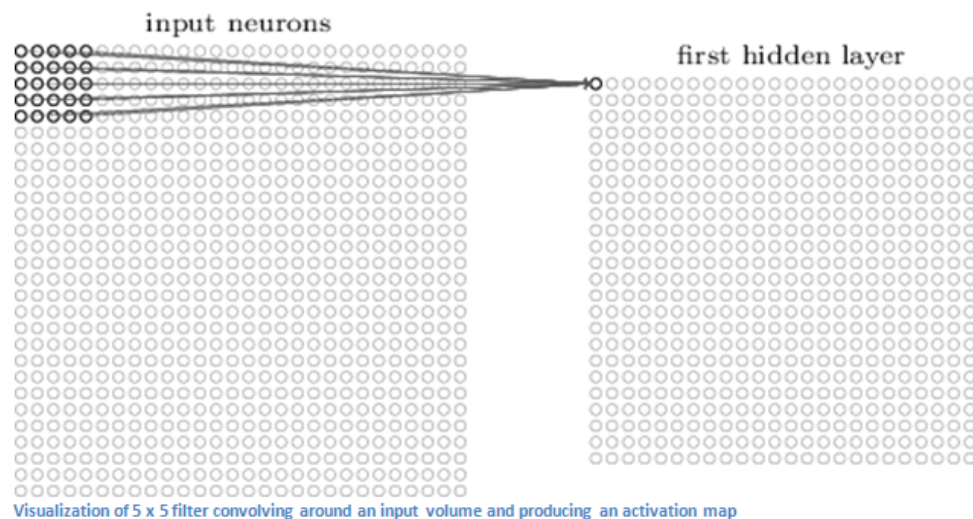
## Structure

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  $5 \times 5 \times 3$ . 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  $28 \times 28$  array is that there are 784 different locations that a  $5 \times 5$  filter can fit on a  $32 \times 32$  input image. These 784 numbers are mapped to a  $28 \times 28$  array.



(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.)

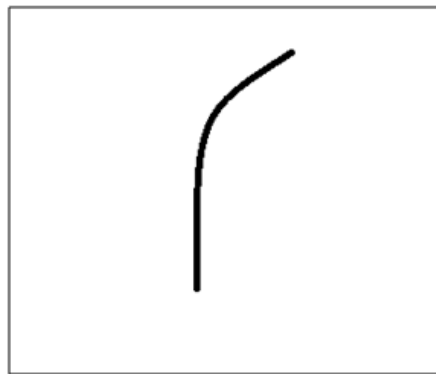
Let's say now we use two  $5 \times 5 \times 3$  filters instead of one. Then our output volume would be  $28 \times 28 \times 2$ . By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

## 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

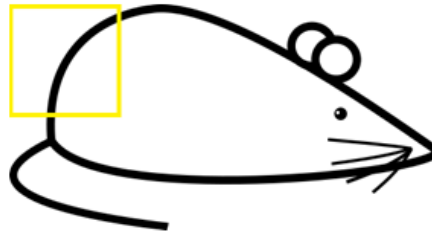


Visualization of a curve detector filter

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

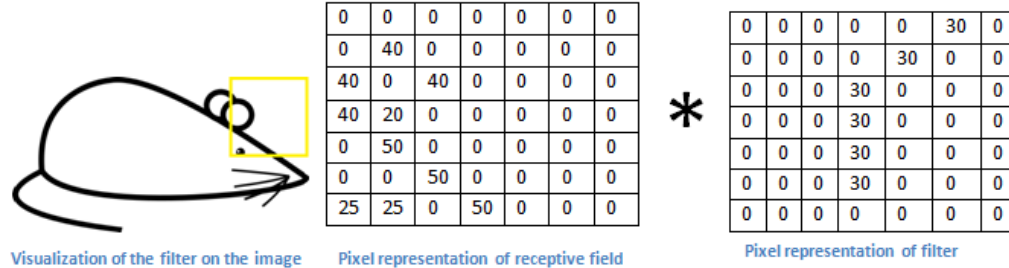
\*

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!)

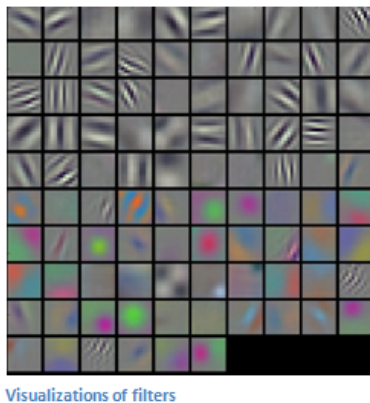
Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.



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 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.



(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

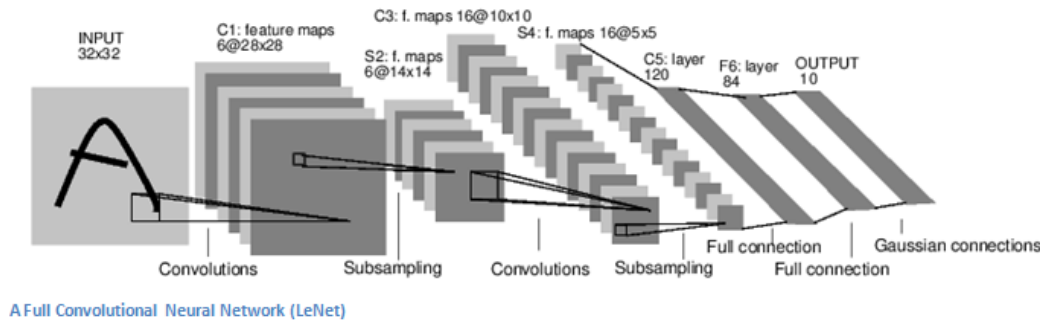
The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would 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 2<sup>nd</sup> 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 2<sup>nd</sup> 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 2<sup>nd</sup> 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 (<https://cs.nyu.edu/~fergus/papers/zeilerECCV2014.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  $[0.1 \ 0.1 \ 0.75 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0.05]$ , 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**.

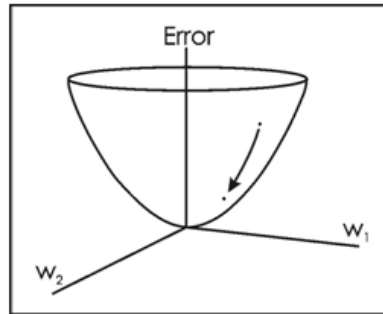
Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values 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.



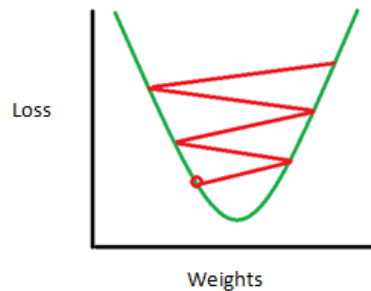
One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves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

Sponsored Links

### Jennifer Aniston Doesn't Leave Much to the Imagination According to Her Maid

Monagiza

### This Tiny Girl Dropped The Reins While Leading A Horse, What Happened Next Was Surprising

EternalLifeStyle

### CPAP Makers Scrambling After New Snoring Fix Unveiled

Purch Expert

### The One WD40 Trick Everyone Should Know About

Oceandraw

### Some Rheumatoid Arthritis Signs Might Be Quite Surprising. Research Rheumatoid Arthritis Symptoms

Rheumatoid Arthritis Management | Search Ads

### G4 By Golpa - Permanent Teeth in 24 Hours

G4 By Golpa

61 Comments

Adit Deshpande

 Login ▾

 Recommend 72

 Tweet

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH



OR SIGN UP WITH DISQUS 

Name



**halite19** • 2 years ago

In first layer-math part section, I don't think I understand this part "Let's say now we use two  $5 \times 5 \times 3$  filters instead of one. Then our output volume would be  $28 \times 28 \times 2$ ". Why would it not be  $28 \times 28 \times 3$ ?

5  |  • Reply • Share ▾



**adeshpande3** Mod ➔ **halite19** • 2 years ago

Because the depth of the output volume ( $28 \times 28 \times 2$ ) will always equal the number of the filters (2) that were applied in the previous convolutional layer. You may be getting confused because of 3 since that is the depth of the filter itself. Just remember:

- Filter depth always needs to match depth of input volume
- Output volume depth is equal to the number of filters in the layer.

4 ^ | v • Reply • Share ›



**halite19** → adeshpande3 • 2 years ago

ok! now it all makes so much sense. Very well written. Thanks.

1 ^ | v • Reply • Share ›



**Rahul Dagade** • 8 months ago • edited

Hi,

In LeNet figure above having C3 layer, how you have got 16 activation maps?

My understanding is as below:

Input: 32x32x1 (Grayscale)

Filter: 6 filters of size 5x5

S2: Pooling layer of size 2x2

Now in C3 how many filters of 5x5 size are used?

1 ^ | v • Reply • Share ›



**Rahul Dagade** → Rahul Dagade • 8 months ago

Got the answer here

<https://stackoverflow.com/q...>

1 ^ | v • Reply • Share ›



**Saurabh Jain** • 2 years ago

I have two queries:

- 1) At training time how filter come to know, whether it has to train for detecting curve edges or straight edges or face etc. ?
- 2) In illustration given in this post depth was ignored to make understanding simpler. Let's consider the depth and suppose we have two convolution layer L1 and L2. Output of L1 is input to L2. Let's say output size of L1 is 20\*20\*10 i.e there are 10 filters in layer L1. Now a filter in L2 will extract feature by reading area 5\*5 (lets say) in one dimension i.e along depth in total it will read 5\*5\*10 area. Am I right ?

Thanks for the post. It has very precise details.

1 ^ | v • Reply • Share ›



**adeshpande3** Mod → Saurabh Jain • 2 years ago

We don't handcode the values for the features. The example I gave of the curve detector is just for illustration purposes. The filters are initialized at random, and are adjusted through gradient descent. And yeah, the filter's depth in a particular convolutional layer will always equal the depth of the input volume to that layer.

^ | v • Reply • Share ›



**Aamir Khan** → adeshpande3 • a year ago

do you mean the filter depth will be equal to the depth of previous activation maps?


^ | v • Reply • Share ›




**adeshpande3** Mod → Aamir Khan • a year ago


Well yes, but be careful with the wording. It's equal to the depth of the maps stacked on top of each other. For example, if there were 5 activation maps created as the output of one layer, then the depth would be 5, and thus the filters that are part of the next layer will have depth 5


^ | v • Reply • Share ›

 **Aamir Khan** → adeshpande3 • a year ago  
Thank you so much Adit Deshpande.  
^ | v • Reply • Share ›


 **Aamir Khan** → adeshpande3 • a year ago  
Hi Adit! excellent and very helpful article on cnn.

I' am new to cnn. I have data set .png format. Do i need to convert it to gray scale ? if yes then how can i convert it to gray scale?  
^ | v • Reply • Share ›


 **acrid56** → Aamir Khan • 22 days ago  
You may do so using any desaturation process.  
If you can yourself easily recognize the contents of the image, then so should the network.  
^ | v • Reply • Share ›

 **Rajat Mehta** → adeshpande3 • a year ago • edited  
How do we get to know how many features needs to be learned ?  
Is it true that a particular feature gets activated for inputs belonging to only a particular class ?  
Or a particular feature can be activated for multiple classes as well?

Suppose we have a classification problem with 10 classes, Dow we need to have 1 filter for each class ? or 5 filters in total? How do we decide on the number of filters to use?  
^ | v • Reply • Share ›

 **adeshpande3** <sup>Mod</sup> → Rajat Mehta • a year ago  
<https://datascience.stackex...>

The short answer is that the # of filters is highly dependent on the depth of your model, the number of classes in your classification problem, the number of examples in your training set, your compute power, etc so there's not a single straightforward answer.  
^ | v • Reply • Share ›


 **Rajat Mehta** → adeshpande3 • a year ago • edited  
Hi Adit,Thanks for your reply.

I read that post, but it did not answer my second question. It would be really helpful for me if you can answer it.

Suppose we have mnist classification problem. In our first hidden layer we have 5 filters. How are those 5 filters trained on our 10 classes ?

Is it like, each one of the five filters is trained for identifying features of all 10 class labels or a single filter is trained only for an image of a particular class label(lets say an image of a '2') ?

Like in your explanation, you took a filter which looks like a curved line, does that filter gets high activations only when it sees an image of a rat(with that curve present in it)? If yes, then what will the activation for that particular filter look like if you give it a non-rat(image other than that of a rat) image?  
^ | v • Reply • Share ›

 **adeshpande3** <sup>Mod</sup> → Rajat Mehta • a year ago  
I think you should frame the questions a little differently. Think back to how our weights/filters are getting trained. We're starting off with completely random values and then slowly adjusting them with gradient descent in such a way that we

minimize our loss. Therefore, before training, we can't necessarily say how our 5 filters will get trained and we don't exactly know what values they will take on a priori. Therefore, there can't really be an answer to your question of "How are those 5 filters trained on our 10 classes". It depends on the training examples you have and the stochastic nature of batch gradient descent. The general understanding is that the X number of filters will "learn" (adjust their weights) such that we get a more accurate model. I don't think it's accurate to say that one of the filters will exactly correspond to one class, it's more of an aggregated effect that comes with all the filters + fully connected weights at the end. As for your last question, it would get high activations wherever the curve is present (not necessarily a rat). The filters at the lower levels are detecting a particular shape/feature rather than a class, so yes even if you give it a non-rat image, you still may see a high activation if there are curves in the image.

^ | v • Reply • Share ›



**Saurabh Jain** → adeshpande3 • 2 years ago

Thanks for replying adit. Still I did not get answer to my first query. Suppose we have two filters F1 and F2. F1 is being trained for detecting curve edges and F2 is being trained for detecting horizontal straight lines. Now at training time weights will be updated for filters F1 and F2( suppose we are using Gradientdescent for updating weights). Now my question is that how Gradientdescent will come to know that it has to update weights for F1 to detect curve edges not horizontal lines and updated weights for F2 to detect horizontal lines not curve edges.

^ | v • Reply • Share ›



**adeshpande3** Mod → Saurabh Jain • 2 years ago

F1 and F2 aren't predetermined like that. Gradient descent adjusts weights in the direction of minimizing the loss, and as a result of that, certain filters will get adjusted to detect some low level feature (like curves or whatever) and other filters will get adjusted to detect other features. It's not predetermined that F1 is looking for curves or that F2 is looking for horizontal lines.

1 ^ | v • Reply • Share ›



**Saurabh Jain** → adeshpande3 • 2 years ago

Thanks Adit.

^ | v • Reply • Share ›



**Ravi** → Saurabh Jain • 2 years ago

Thanks a lot Saurabh for bringing this question and Adit for the reply. A followup qn on the same topic - so since the adjustment happens by the system itself, and we don't have any interference on the weights, it might even happen that that multiple filters might be adjusted to detect the same thing right? I mean MIGHT happen case...

So its all pure luck - basically a very rare scenario that multiple filters are detecting/filtering the same thing. Am i right?

^ | v • Reply • Share ›



**adeshpande3** Mod → Ravi • 2 years ago

Yeah technically that is true. Multiple filters could end up detecting the same thing, but always keep in mind that the weights are adjusting in a way such that the loss decreases. Our dL/dW tells us the impact each weight has on the loss. Thus, I don't think the word "luck" is appropriate. The weights aren't changing in a random way, but they are adjusting in a specific way that encourages loss to decrease. And as a byproduct, yes, some filters could be the same but we're okay with that

^ | v • Reply • Share ›



**Sei Babji Vieranki** → adeshpande3 • a year ago

Thanks for time. I really appreciate.

^ | v • Reply • Share ›



**DC** • 2 months ago

To convolve does not mean to slide. You are just making things more confusing.

^ | v • Reply • Share ›



**Jason Dersu Kim** • 6 months ago • edited

Backproagation is a beauty that many previous CV scholars ignored or did not understand its attraction. I learned it from this article. I wonder our visual cortex does that too while learning.

^ | v • Reply • Share ›



**Joshua Detwiler** • 7 months ago

You've got a broken link: [research paper](<http://www.matthewzeiler.co...>)

^ | v • Reply • Share ›



**adeshpande3** Mod → Joshua Detwiler • 7 months ago

Thanks!

1 ^ | v • Reply • Share ›



**Söül** • 8 months ago

which layer of convolution network layer use update of learning happen?

- a) convolution layer.
- b) pooling layer.
- c) ReLu.
- d) fully-connected layer

^ | v • Reply • Share ›



**adeshpande3** Mod → Söül • 8 months ago

Learning updates happen wherever you have weight matrices that you have to update. So convolutional and FC layers.

^ | v • Reply • Share ›



**Söül** → adeshpande3 • 7 months ago

thank u for replying

^ | v • Reply • Share ›



**mozhdeh** • 9 months ago

It was very helpful. thanks a million for describing this sophisticated subject in such a fluent way :)

^ | v • Reply • Share ›



**Sujal** • a year ago

Great post to incite the interest for beginners like me!

^ | v • Reply • Share ›



**Sannan Ali** • a year ago

Why the input feature image which is  $32 \times 32 \times 1$  changed into  $28 \times 28 \times 1$  after the convolution??

^ | v • Reply • Share ›



**adeshpande3** Mod → Sannan Ali • a year ago

Because the filter covers a  $5 \times 5$  area and produces a single number from the element wise multiplication and addition. That filter is applied at every unique location in the  $32 \times 32$  image. There are 784 ( $28 \times 28$ ) of those locations.





^ | v • Reply • Share ›



**Soumita Chel** → adeshpande3 • 5 months ago

28X28 ---> Is it because we have depth as 3? .

^ | v • Reply • Share ›



**adeshpande3** Mod → Soumita Chel • 5 months ago

Not sure if I understand the question. The 28 comes from the number of locations that the filter can go on. The 1 comes from the number of filters being applied which is 1. The filter and input volume being of depth 3 means that we're doing the element wise multiplications for 75 values instead of 25.

4 ^ | v • Reply • Share ›



**Manish Gupta** • a year ago

Beautifully written article Adit. Thanks a lot!

^ | v • Reply • Share ›



**M. Kumar Raja** • a year ago

could i get any videos for this cnn!!

^ | v • Reply • Share ›



**amr** • a year ago

Can i use already extracted features and pass them to CNN network? i.e i have extracted features for 2d images using GLCM then i want to import say csv with ready features to CNN is that applicable? does it needs special treatment? does it affect the network and give wrong performance?

^ | v • Reply • Share ›



**adeshpande3** Mod → amr • a year ago

From a high level, a CNN itself can be thought of as a feature extractor, so if you've already previously extracted features, then it might be a better idea to train a normal feedforward NN instead a CNN.


^ | v • Reply • Share ›





**amr** → adeshpande3 • a year ago


Thanks Ade, I want to benifit from the CNN to extract the output of last conv layer and then can match these features with other images features using some sort of metric to retrieve the related images , how can i do that with ready extracted features?


^ | v • Reply • Share ›


 **adeshpande3** Mod → amr • a year ago  
Sorry, I just don't really understand why you would need to pass the features through a CNN if you already have features. Why would you need to compute a 2nd set of features?  
^ | v • Reply • Share ›


 **amr** → adeshpande3 • a year ago  
as i understand the 2D ultrasound images features better to be extracted using tools such GLCM as they have special features, what do you suggest ? extracting the features using CNN itself then continue?  
another question regarding this , do i need a similar sizes images to be used in the CNN? i.e i have deferent images with deferent dimensions, also do i have to do any preprocessing for the images?  
^ | v • Reply • Share ›


 **adeshpande3** Mod → amr • a year ago  
Yeah they all have to have the same dimensionality  
1 ^ | v • Reply • Share ›


 **amr** → adeshpande3 • a year ago  
could you please answer this : as i understand the 2D ultrasound images features better to be extracted using tools such GLCM as they have special features, what do you suggest ? extracting the features using CNN itself then continue?  
  
regarding the size, is there a way to make the input images with same dimensions? any code in CNN,etc  
^ | v • Reply • Share ›


 **Edvin** • a year ago  
Hi, can you explain what is dL and what is dW ?  
^ | v • Reply • Share ›

 **adeshpande3** Mod → Edvin • a year ago  
Will be easier to understand if you have a background in calculus. The two terms are part of a concept called a partial derivative. This might help <https://www.khanacademy.org....> dL represents derivative of the loss with respect to dW which is the derivative of the weight.  
^ | v • Reply • Share ›

 **Tomas Plankis** • a year ago  
<https://en.wikipedia.org/wi...>  
  
Can you explain why using 1/2?  
^ | v • Reply • Share ›

 **Kenan Husayn** → Tomas Plankis • a year ago  
The original formula is with one over "N" ( $1/N$ ) - total number of elements, sometimes you see an additional one over two ( $1/2$ ) for making determinant calculations easy for the next step. Sometimes people don't even include " $1/N$ " when talking about square error alone but I think since he said MSE, "M" for mean (square error) the formula should include it as well.  
^ | v • Reply • Share ›


 **Iram Shahzadi** • a year ago  
I have a question that how to train a network when no labels are available.  
^ | v • Reply • Share ›

 **adeshpande3** Mod → Iram Shahzadi • a year ago  
Not really possible for CNNs. Machine learning that doesn't use labels falls into the category of unsupervised learning. I believe there is

Not really possible for CNNs: machine learning that doesn't use labels falls into the category of unsupervised learning. I believe there is some work in weakly supervised learning for CNNs though.

^ | v • Reply • Share ›

[Load more comments](#)

 [Subscribe](#)  [Add Disqus to your site](#)[Add Disqus](#)[Add](#)  [Disqus' Privacy Policy](#)[Privacy Policy](#)[Privacy Policy](#)

[Sponsored Links](#)

### 3 Ways Your Cat Asks For Help

Dr. Marty Nature's Feast

### Rarely Seen Historical Photos That Will Leave You Speechless

SoGoodly

### This Tiny Girl Dropped The Reins While Leading A Horse, What Happened Next Was Surprising

EternalLifeStyle

### 3 Dangerous Foods People Feed Their Dogs (Without Realizing It)

Dr. Marty ProPower Plus Supplement

### CPAP Makers Scrambling After New Snoring Fix Unveiled

Purch Expert

### Here Are Everyday Roads That Give Drivers Panic Attacks

Livestly

[\(mailto:aditdeshpande3@gmail.com\)](mailto:aditdeshpande3@gmail.com)

[\(https://www.facebook.com/aditdeshpande3\)](https://www.facebook.com/aditdeshpande3)

[\(https://github.com/adeshpande3\)](https://github.com/adeshpande3)

[\(https://instagram.com/aditdeshpande3\)](https://instagram.com/aditdeshpande3)

[\(https://www.linkedin.com/in/aditdeshpande3\)](https://www.linkedin.com/in/aditdeshpande3)

[\(/adeshpande3.github.io/feed.xml\)](https://adeshpande3.github.io/feed.xml)

[\(https://www.twitter.com/aditdeshpande3\)](https://www.twitter.com/aditdeshpande3)