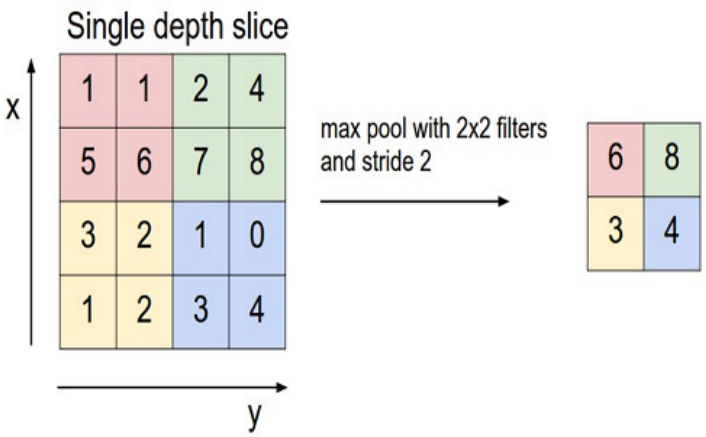
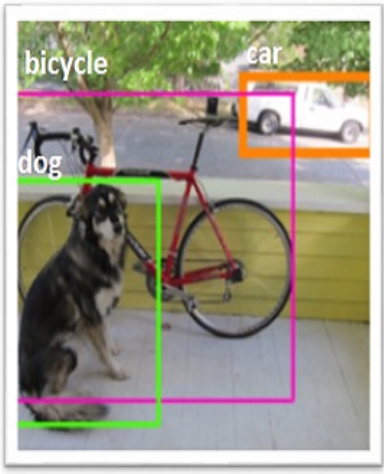
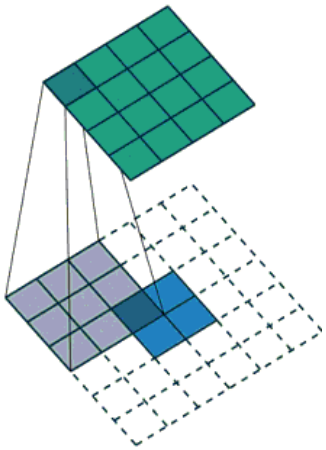
Adit Deshpande (/adeshpande3.github.io/)
CS Undergrad at UCLA ('19)

A Beginner's Guide To Understanding Convolutional Neural Networks Part 2







Introduction

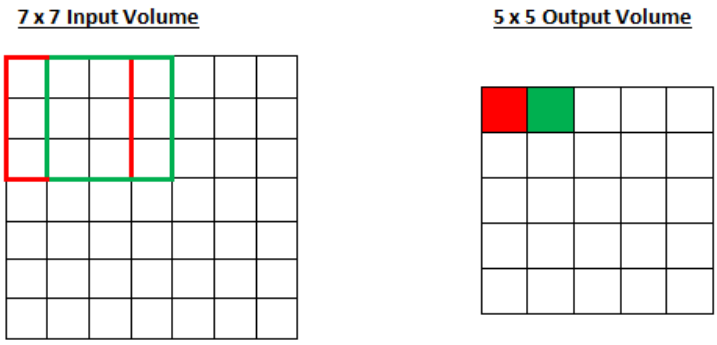
Link to Part 1 (https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/)

In this post, we'll go into a lot more of the specifics of ConvNets. **Disclaimer:** Now, I do realize that some of these topics are quite complex and could be made in whole posts by themselves. In an effort to remain concise yet retain comprehensiveness, I will provide links to research papers where the topic is explained in more detail.

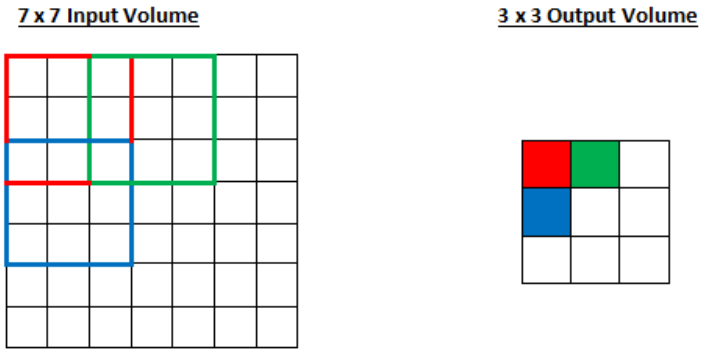
Stride and Padding

Alright, let's look back at our good old conv layers. Remember the filters, the receptive fields, the convolving? Good. Now, there are 2 main parameters that we can change to modify the behavior of each layer. After we choose the filter size, we also have to choose the **stride** and the **padding**.

Stride controls how the filter convolves around the input volume. In the example we had in part 1, the filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride. In that case, the stride was implicitly set at 1. Stride is normally set in a way so that the output volume is an integer and not a fraction. Let's look at an example. Let's imagine a 7 x 7 input volume, a 3 x 3 filter (Disregard the 3rd dimension for simplicity), and a stride of 1. This is the case that we're accustomed to.

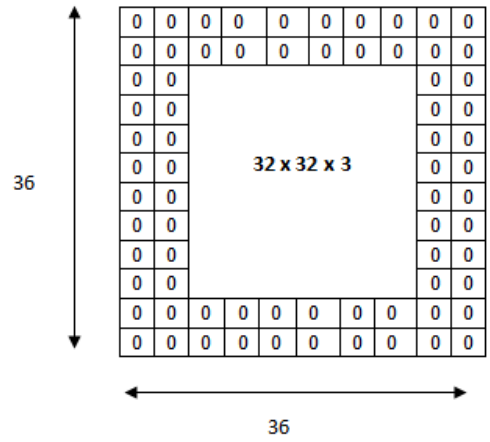


Same old, same old, right? See if you can try to guess what will happen to the output volume as the stride increases to 2.



So, as you can see, the receptive field is shifting by 2 units now and the output volume shrinks as well. Notice that if we tried to set our stride to 3, then we'd have issues with spacing and making sure the receptive fields fit on the input volume. Normally, programmers will increase the stride if they want receptive fields to overlap less and if they want smaller spatial dimensions.

Now, let's take a look at padding. Before getting into that, let's think about a scenario. What happens when you apply three 5 x 5 x 3 filters to a 32 x 32 x 3 input volume? The output volume would be 28 x 28 x 3. Notice that the spatial dimensions decrease. As we keep applying conv layers, the size of the volume will decrease faster than we would like. In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features. Let's say we want to apply the same conv layer but we want the output volume to remain 32 x 32 x 3. To do this, we can apply a zero padding of size 2 to that layer. Zero padding pads the input volume with zeros around the border. If we think about a zero padding of two, then this would result in a 36 x 36 x 3 input volume.



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

If you have a stride of 1 and if you set the size of zero padding to

$$Zero\ Padding = \frac{(K-1)}{2}$$

where K is the filter size, then the input and output volume will always have the same spatial dimensions.

The formula for calculating the output size for any given conv layer is

$$O = \frac{(W-K+2P)}{S} + 1$$

where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.

Choosing Hyperparameters

How do we know how many layers to use, how many conv layers, what are the filter sizes, or the values for stride and padding? These are not trivial questions and there isn't a set standard that is used by all researchers. This is because the network will largely depend on the type of data that you have. Data can vary by size, complexity of the image, type of image processing task, and more. When looking at your dataset, one way to think about how to choose the hyperparameters is to find the right combination that creates abstractions of the image at a proper scale.

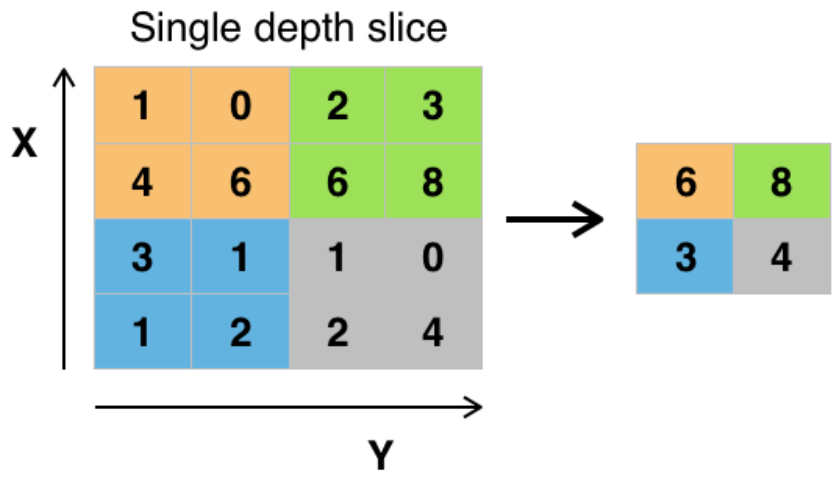
ReLU (Rectified Linear Units) Layers

After each conv layer, it is convention to apply a nonlinear layer (or **activation layer**) immediately afterward.The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that **ReLU layers** work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers (Explaining this might be out of the scope of this post, but see here (https://en.wikipedia.org/wiki/Vanishing_gradient_problem) and here (https://www.quora.com/What-is-the-vanishing-gradient-problem) for good descriptions). The ReLU layer applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.

Paper (http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf) by the great Geoffrey Hinton (aka the father of deep learning).

Pooling Layers

After some ReLU layers, programmers may choose to apply a **pooling layer**. It is also referred to as a downsampling layer. In this category, there are also several layer options, with maxpooling being the most popular. This basically takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every subregion that the filter convolves around.



Example of Maxpool with a 2x2 filter and a stride of 2

Other options for pooling layers are average pooling and L2-norm pooling. The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. As you can imagine, this layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. The first is that the amount of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it will control **overfitting**. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of overfitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.

Dropout Layers

Now, **dropout layers** have a very specific function in neural networks. In the last section, we discussed the problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. The idea of dropout is simplistic in nature. This layer "drops out" a random set of activations in that layer by setting them to zero. Simple as that. Now, what are the benefits of such a simple and seemingly unnecessary and counterintuitive process? Well, in a way, it forces the network to be redundant. By that I mean the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out. It makes sure that the network isn't getting too "fitted" to the training data and thus helps alleviate the overfitting problem. An important note is that this layer is only used during training, and not during test time.

Paper (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf) by Geoffrey Hinton.

Network in Network Layers

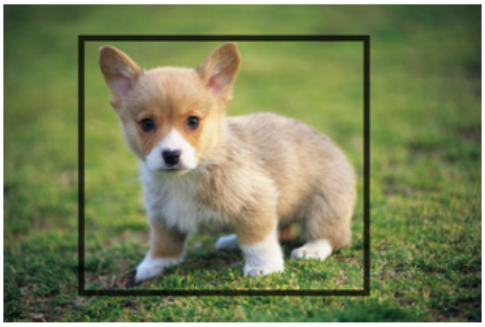
A **network in network** layer refers to a conv layer where a 1 x 1 size filter is used. Now, at first look, you might wonder why this type of layer would even be helpful since receptive fields are normally larger than the space they map to. However, we must remember that these 1x1 convolutions span a certain depth, so we can think of it as a 1 x 1 x N convolution where N is the number of filters applied in the layer. Effectively, this layer is performing a N-D element-wise multiplication where N is the depth of the input volume into the layer.

Paper (https://arxiv.org/pdf/1312.4400v3.pdf) by Min Lin.

Classification, Localization, Detection, Segmentation

In the example we used in Part 1 of this series, we looked at the task of **image classification**. This is the process of taking an input image and outputting a class number out of a set of categories. However, when we take a task like **object localization**, our job is not only to produce a class label but also a bounding box that describes where the object is in the picture.

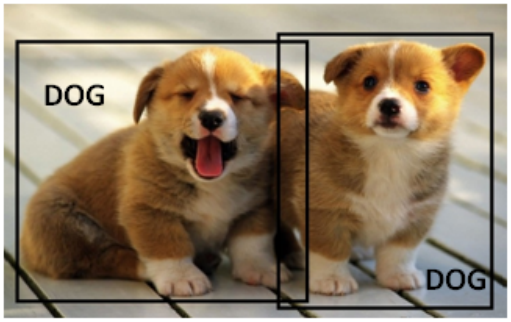
Object Classification is the task of identifying that picture is a dog



Object Localization involves the class label as well as a bounding box to show where the object is located.

We also have the task of **object detection**, where localization needs to be done on all of the objects in the image. Therefore, you will have multiple bounding boxes and multiple class labels.

Finally, we also have **object segmentation** where the task is to output a class label as well as an outline of every object in the input image.



Object Detection involves localization of multiple objects (doesn't have to be the same class).

Object Segmentation involves the class label as well as an outline of the object in interest.

More detail on how these are implemented to come in Part 3, but for those who can't wait…

Detection/ Localization: RCNN (https://arxiv.org/pdf/1311.2524v5.pdf), Fast RCNN (https://arxiv.org/pdf/1504.08083.pdf), Faster RCNN (http://arxiv.org/pdf/1506.01497v3.pdf), MultiBox (http://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Erhan_Scalable_Object_Detection_2014_CVPR_paper.pdf), Bayesian Optimization (http://web.eecs.umich.edu/~honglak/cvpr15-cnn-detection.pdf), Multi-region (http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Gidaris_Object_Detection_via_ICCV_2015_paper.pdf), RCNN Minus R (http://www.robots.ox.ac.uk/~vedaldi/assets/pubs/lenc15rcnn.pdf), Image Windows (http://calvin.inf.ed.ac.uk/wp-content/uploads/Publications/alexe12pami.pdf)
Segmentation: Semantic Seg (http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf), Unconstrained Video (http://calvin.inf.ed.ac.uk/wp-content/uploads/Publications/papazoglouICCV2013-camera-ready.pdf), Shape Guided (https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/Borenstein06.pdf), Object Regions (http://crcv.ucf.edu/papers/cvpr2013/VideoObjectSegmentation.pdf), Shape Sharing (http://www.cs.utexas.edu/~grauman/papers/shape-sharing-ECCV2012.pdf)

Yeah, there's a lot more.

Transfer Learning

Now, a common misconception in the DL community is that without a Google-esque amount of data, you can't possibly hope to create effective deep learning models. While data is a critical part of creating the network, the idea of transfer learning has helped to lessen the data demands. **Transfer learning** is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and "fine-tuning" the model with your own dataset. The idea is that this pre-trained model will act as a feature extractor. You will remove the last layer of the network and replace it with your own classifier (depending on what your problem space is). You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

Let's investigate why this works. Let's say the pre-trained model that we're talking about was trained on ImageNet (For those that aren't familiar, ImageNet is a dataset that contains 14 million images with over 1,000 classes). When we think about the lower layers of the network, we know that they will detect features like edges and curves. Now, unless you have a very unique problem space and dataset, your network is going to need to detect curves and edges as well. Rather than training the whole network through a random initialization of weights, we can use the weights of the pre-trained model (and freeze them) and focus on the more important layers (ones that are higher up) for training. If your dataset is quite different than something like ImageNet, then you'd want to train more of your layers and freeze only a couple of the low layers.

Paper (https://arxiv.org/pdf/1411.1792v1.pdf) by Yoshua Bengio (another deep learning pioneer).
Paper (http://arxiv.org/pdf/1403.6382.pdf) by Ali Sharif Razavian.
Paper (https://arxiv.org/pdf/1310.1531.pdf) by Jeff Donahue.
Paper (https://arxiv.org/pdf/1705.07706.pdf) and subsequent paper (https://arxiv.org/pdf/1707.09872.pdf) by Dario Garcia-Gasulla.

Data Augmentation Techniques

By now, we're all probably numb to the importance of data in ConvNets, so let's talk about ways that you can make your existing dataset even larger, just with a couple easy transformations. Like we've mentioned before, when a computer takes an image as an input, it will take in an array of pixel values. Let's say that the whole image is shifted left by 1 pixel. To you and me, this change is imperceptible. However, to a computer, this shift can be fairly significant as the classification or label of the image doesn't change, while the array does. Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as **data augmentation** techniques. They are a way to artificially expand your dataset. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more. By applying just a couple of these transformations to your training data, you can easily double or triple the number of training examples.

Link to Part 3 (https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html)

Dueces.

Sources (/assets/Sources2.txt)

Written on July 29, 2016

jeff moskowitz • a year ago

Thanks for doing this series. This is the clearest explanation of CNNs I've encountered thus far so really, fabulous work.

I followed Part 1 quite easily, but I was hoping you could expand on a few things in Part 2 that weren't quite as clear (for me at least).

In your formula for calculating the output size of any given conv layer, what does W represent? You labeled all the other variables but that one. Is it the input size?

For the ReLu layer, could you elaborate a little more on why these are needed? Why is non-linearity good? So that those layers will be able to recognize curving and sinusoid lines also rather than just straight lines? But then how does merely converting all of the negative values to zero help with that?

And I'd also like some clarity on the purpose of the Pooling Layer. Certainly reducing the computation costs makes sense. But how can you know you're not losing something important when simplifying it so drastically? And does it help with overfitting because if you are reducing a section of the input to one number (one feature) then it doesn't matter where in that section the feature you're identifying appeared? That way if you're trying to identify a clown for instance, the red nose does not have to be right in the middle, it can be a little to one side or another and the program will still recognize the clown correctly? Did I understand that right?

Finally, the N-D element wise multiplication you mentioned in the Network in Networks layer? What is the D? Why are we doing this?

I know you gave links to papers under most of these sections that explain these concepts in greater detail and I am reading those as well, but I really appreciate your overviews so any additional light you can shed on these areas would be really helpful. Thanks again!

6 • Reply • Share ›

adeshpande3 Mod → jeff moskowitz • a year ago

Thanks for the input Jeff, appreciate it!

Good catch on the W variable, forgot to label that one. You're correct, W is the input volume height/width (For a 32 x 32 x 3 input volume, W would be 32). I'll make the corresponding change in the article.

As for the ReLU layer, I definitely do understand the confusion. I think that this idea of a nonlinear activation function is something that came from early neural networks and multilayer perceptrons and even biology (not CNNs in particular, so I don't think that statement about curving lines and sinusoid lines would be accurate). From my understanding, this nonlinearity is good because there's a limit to how well you can represent and modify your data with linear functions (convolutions in this case). I think that this Quora answer (http://bit.ly/1spofdg) illustrates it well when they talk about how logistic regression results can improve with nonlinearity because it is better able to "fit" the data and make more accurate classifications. This lecture notes page (http://cs231n.github.io/neu... also has a little more on how that idea relates to CNNs in particular. As for why ReLU in particular helps, (http://bit.ly/2akvL4U) basically talks about benefits being sparsity and reduced risk of a vanishing gradient, which are definitely plausible explanations.

Those definitely are good questions on the Pooling layers. I do agree with you on that risk of losing something important, but I think that because we use maxpool in particular, this will make sure to keep all of the values with the high activations (and remember from Part 1, when we say high activations, we're talking about the presence of certain features in the input volume, aka the "important things") and so because of that I think the risk of losing something is minimized. And yeah, you definitely got the overfitting part right. That's a perfect example, we're able to decrease the input volume during the pool layer while still retaining that information about the clown's nose.

Sorry for the confusion on N-D. I realize I could have worded that better. For example, let's say you have this 1x1 filter (with depth of 3) and you're applying it to a 32x32x3 input volume, What I meant to say is that you have a 3 dimensional (or N dimensional) product where N is the depth. I guess the basic point I'm just trying to get across is that regardless of the fact

that the filter is 1x1, there still are multiplications and summation that is taking place because of the fact that there is a depth component. (I think mentioning the N-D term might have just brought on more confusion)

And yeah, having the research papers as links is helpful, but they definitely take a while to understand, at least for me, so I'm definitely with you there!

3 ^ v • Reply • Share ›

Syed Rafay • 2 months ago

I started training the models without having much knowledge, but now i understand some of the basic concepts and the terms and what is happening in the background.
Thanks, great work!

5 ^ v • Reply • Share ›

thepete5 • 5 months ago

Really great series, thank you for this!

Do you need to keep the weights frozen during transfer learning for the entire training cycle? What happens if you finish training, then unfreeze, and retrain? At some point, after training the custom layer, would you really expect the lower layers to change much? I suppose another way of asking the question is why presuppose no correlation between a pre-trained set and a custom layer?

2 ^ v • Reply • Share ›

adeshpande3 Mod ➚ thepete5 • 5 months ago

Thanks for the great question. Definitely had me thinking. To be honest, not sure if I have a good answer for that one. I think the idea of having the weights frozen, then training, then unfreezing, and then retraining is interesting, because I'm not sure how much the weights would end up changing (from after the first training to after the second training). Would the initially frozen weights start to change? If so, would those also cause an effect on the weights later on in the network? If not, maybe there isn't much of a difference between having them frozen or not in the first place. I think maybe the most important thing we should take away from transfer learning is that that we're using weights from a pretrained network, instead of starting with random values, so that's 100% bound to help us. Maybe the question of whether or not to freeze the weights of the lower layers is up for debate. Let me know if you find anything else on this topic!

^ v • Reply • Share ›

Orian Beltrame da Silva • a year ago

Amazing Post! Thanks!

2 ^ v • Reply • Share ›

Nam Nguyen • 6 months ago

Hi, I want to apply my filter to a special sub-region of my input space, for instance, say my input is a 10 * 10 array, I only want to apply my filter to the first 3 columns of that array. I have a look at the documentation for tensorflow, but in every example I found, the filter is applied to the entire input. This makes me wonder if there is any way I can achieve what I want in tensorflow.

1 ^ v • Reply • Share ›

adeshpande3 Mod ➚ Nam Nguyen • 6 months ago

I guess you can zero out all of the input values that aren't part of the first 3 columns right? And then when you go through a convolution layer, the resulting activation map will be a function of just those 3 columns

1 ^ v • Reply • Share ›

vishal gupta • 6 months ago

Thank you for your help , It really helped me a lot, Can you please tell one thing that how you decide which layer to put first and then next int the network, I mean when should you add max pooling layer in the network or when should you add convolution layer. please tell me what factor decide this .

1 ^ v • Reply • Share ›

adeshpande3 Mod ➚ vishal gupta • 6 months ago

Designing network architecture is more of an art than a science. I just generally look at what the state of the art models do and then base my networks on those. Check out part 3 of this series for more on those networks. The general theme of Conv - ReLu - Pool seems to be effective. Whether or not you want to pool also largely depends on how big of an image you're dealing with as well, since you don't want to have a lot of pooling operations for a small image.

• Reply • Share ›

Abraham Hoffman • a year ago
baalin!
1 • Reply • Share ›

Miki Nacucchi • a year ago
Great post! looking forward to read part 3 :-)
1 • Reply • Share ›

Aurélien Werenne • a month ago
Great explanations! Thanks, It helps a lot!
• Reply • Share ›

Hao Liu • 2 months ago
Thanks a lot for the great post. This is the first article to let me really understand how DL works.
• Reply • Share ›

Jayakumar H • 5 months ago
Thanks for your effort, ONLY after reading this article I understood this: http://cs231n.github.io/con...
• Reply • Share ›

Yasar Hayat • 7 months ago
Thanks a lot for making this stuff so clear
• Reply • Share ›

Jingwei Zhu • 9 months ago
Thanks a lot Adit for making all of these so clear!
• Reply • Share ›

The_MightyDee • 10 months ago
Good read
• Reply • Share ›

Moondra • a year ago
Hi. I'm a little confused about how color is retained between layers as well as the usage of RGB values when using matrix multiplication of input values and filters in conv layers. In part 1, you used an example with an image of a mouse and a filter which had a small curve/edge. When they were multiplied at the first receptive field, you got a whopping value of 6000, which hints at a positive correlation. Now if we use RGB values, aren't they limited to 255 (I'm honestly not sure). Wouldn't a lot of multiplications result in numbers over 255, thus making that value void(?) or even if under 255, could result in different color schemes due to multiplications with filters, making it difficult to retain original colors. You had mentioned that the activation map of the first layer with then be the input layer of the second layer, so the first activation map has to be valid pixels values (right?). It seems it would be difficult to retain colors going from one layer to another. Thanks.
• Reply • Share ›

adeshpande3 Mod → Moondra • a year ago
You're correct in that values could be greater than 255 or less than 0. However, the only part of the network that needs to have valid pixel values is the input image into the network. After you apply filters and get your first activation map, these values don't necessarily need to be in that range. I think that idea of retaining color is not something that CNNs really do. Their job is to just calculate activation values, and figure out which of these values most influences the classification decision (ex: a high curve filter activation = a wheel or circular object). Try not to think of the subsequent activation maps as actual images, just try to understand what they represent about the original input image.
• Reply • Share ›

Moondra → adeshpande3 • a year ago
Thanks for the clarification. =)
• Reply • Share ›

Joo-Kyung Kim • a year ago

During the test time, dropout layers are used to multiply (1-drop_rate) to the input.

Reply • Share ›

adeshpande3 Mod → Joo-Kyung Kim • a year ago

Ah, I see. Thanks for the note.

Reply • Share ›

Gilles Daquin • a year ago

Thank you Adit for sharing the result of your own research. It is very helpful and I could find new papers I was not aware of.
Big THANK YOU.

Reply • Share ›

Chinnu Man • a year ago

heyy... What do you mean by negative activation in the RELU section ? In the activation map we will have only the positive values right ? As, multiplying and adding the positive values gives positive values. Please clarify my doubt.

Thanks in advance.

Reply • Share ›

adeshpande3 Mod → Chinnu Man • a year ago

The values in the activation map could possibly be negative actually. This is because when we are doing the element wise multiplications between the pixel values and the weights, the weights can actually be negative numbers in some cases. Remember the curve detector filter I mentioned in Part 1? Well, that was a relatively simple filter (simple for the purpose of explaining the math behind it), and there are other filters that do utilize negative values. See this link for details of possible kernel/filter matrices which do have negative values. (https://en.wikipedia.org/wi...

Reply • Share ›

Mainak kUNDU • a year ago

when the Part-3 will release

Reply • Share ›

adeshpande3 Mod → Mainak kUNDU • a year ago

By the end of this week most likely

Reply • Share ›

Chris Rigano • a year ago

This is the first time I can say I am gaining an basic understanding of CONVET NETs ... I am waiing for the on line book, series ... like Steven H. bringing Neural Nets to the masses ... so where is post 3!!!! Chris :-)

Reply • Share ›

WZ • a year ago

Hi nice post! Would be nice if you can explain more on detection or localization. I am planning to work on localization. But I am uncertain of how we can apply regression to get the bounding box coordinates.

Reply • Share ›

adeshpande3 Mod → WZ • a year ago

Thanks! I'll try to put a little of that in Part 3, but in the meantime, there are a couple good links I can direct you to.

see more

1 • Reply • Share ›

WZ ➚ adeshpande3 • a year ago

Thanks for sharing. Yup, I watched the lecture videos by Stanford they are very good. However, I still find visualizing the bounding box regression a little difficult. For simplicity lets consider only the localization problem, usually we apply a convnet on the input image, then at the end of the FC (fully connected layer) we connect it to a regression head (say linear regression).
During convnet, we apply convolution and pooling which will reduce the size of the image and each feature map will only be activated by the pixels which "correlates" with the image.
i) At the FC layer usually we are left with 1x1xdepth. So depth here should be 4 corresponding to information for the 4 output describing bounding box?
ii) We apply regression to the FC layer
-> if the depth is 4 the input to the regression head will be only a [1x1x4] matrix which means only 4 numbers. Does it mean the regression model (after training) will have a function which can input the 4 numbers and output 4 numbers relative to the coordinate of the bounding boxes?
--> will we get relative coordinates for the bounding boxes from the 4 output numbers of the regression head (some ratio with respect to the original dimension of the input image) since each conv-pool layer we reduce the size of the image (or feature map)?

1 • Reply • Share ›

adeshpande3 Mod ➚ WZ • a year ago

i) I believe your answer to this would depend on where you place your regression head and where you split up the network. If you pause the video at 9:34, I believe the final output of the FC layer on the regression head would be 1x1x4 like you said. But the final output of the FC layer on the classification head would be 1x1xN where N is the number of classes you want to choose from.
ii) So from what I understand, when you have a trained regression head already, and when you feed an image into the ConvNet, the output will be one class score and 4 bounding box coordinates. I don't think that

(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)