How are 1x1 convolutions the same as a fully connected layer?

I've recently read Yan LeCuns comment on 1x1 convolutions:

> **In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.**
>
> It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input. **In that scenario, the "fully connected layers" really act as 1x1 convolutions.**

I would like to see a simple example for this.

Example

Assume you have a fully connected network. It has only an input layer and and output layer. The input layer has 3 nodes, the output layer has 2 nodes. This network has $3 \cdot 2 = 6$ parameters. To make it even more concrete, lets say you have a ReLU activation function in the output layer and the weight matrix

$$W = \begin{pmatrix} 0 & 1 & 1 \\ 2 & 3 & 5 \end{pmatrix} \in \mathbb{R}^{2\times 3}$$
$$b = \begin{pmatrix} 8 \\ 13 \end{pmatrix} \in \mathbb{R}^2$$

So the network is $f(x) = ReLU(W \cdot x + b)$ with $x \in \mathbb{R}^3$.

How would the convolutional layer have to look like to be the same? What does LeCun mean with "full connection table"?

I guess to get an equivalent CNN it would have to have exactly the same number of parameters. The MLP from above has $2 \cdot 3 + 2 = 8$ parameters.

neural-network convnet

edited Jul 17 '16 at 13:29

asked Jul 17 '16 at 13:23

Martin Thoma
3,792 3 27 86

3 Answers

Your Example

In your example we have 3 input and 2 output units. To apply convolutions, think of those units having shape: `[1,1,3]` and `[1,1,2]`, respectively. In CNN terms, we have `3` input and `2` output feature maps, each having spatial dimensions `1 x 1`.

Applying an `n x n` convolution to a layer with `k` feature maps, requires you to have a kernel of shape `[n,n,k]`. Hence the kernel of you `1x1` convolutions have shape `[1, 1, 3]`. You need `2` of those kernels (or filters) to produce the `2` output feature maps. Please Note: 1×1 convolutions really are $1 \times 1 \times \text{number of channels of the input}$ convolutions. The last one is only rarely mentioned.

Indeed if you choose as kernels and bias:

$$w_1 = (0 \quad 1 \quad 1) \in \mathbb{R}^3$$
$$w_2 = (2 \quad 3 \quad 5) \in \mathbb{R}^3$$
$$b = \begin{pmatrix} 8 \\ 13 \end{pmatrix} \in \mathbb{R}^2$$

The conv-layer will then compute $f(x) = ReLU\left(\begin{pmatrix} w_1 \cdot x \\ w_2 \cdot x \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)$ with $x \in \mathbb{R}^3$.

Transformation in real Code

For a real-life example, also have a look at my vgg-fcn implementation. The Code provided in this file takes the VGG weights, but transforms every fully-connected layer into a convolutional layers. The resulting network yields the same output as `vgg` when applied to input image of shape `[244,244,3]`. (When applying both networks without padding).

The transformed convolutional layers are introduced in the function `_fc_layer` (line 145). They have kernel size `7x7` for FC6 (which is maximal, as `pool5` of VGG outputs a feature map of shape `[7,7, 512]`. Layer `FC7` and `FC8` are implemented as `1x1` convolution.

"Full Connection Table"

I am not 100% sure, but he might refer to a filter/kernel which has the same dimension as the input feature map. In both cases (Code and your Example) the spatial dimensions are maximal in the sense, that the spatial dimension of the filter is the same as the spatial dimension as the input.

edited Jul 19 '16 at 2:35

answered Jul 17 '16 at 21:41

MarvMind
266 2 4

"Hence the kernel of you 1x1 convolutions have shape [1, 1, 3].". What? There seems to be a bigger misunderstanding of convolutions. I thought if a convolution kernel has shape [1, 1, 3], then one would say it is a 1x1x3 convolution? So 1x1 convolution is only about the output, not about the kernel? – Martin Thoma Jul 18 '16 at 3:28

For me `kernel = filter`, do you agree? >> "So 1x1 convolution is only about the output, not about the kernel? Not at all. A `3x3` convolution can have an arbitrary output shape. " Indeed, if padding is used and `stride=1` then the `output shape = input shape`. >> "I thought if a convolution kernel has shape [1, 1, 3], then one would say it is a 1x1x3 convolution?" No, I have never heared someone talking about `3x3x512` convolutions. However all convolution-filters I have seen have a third spatial dimension equal to the number of feature-maps of the input layer. – MarvMind Jul 18 '16 at 5:51

For reference, have a look at the `Convolution Demo` of Karpathies CS321n course: cs231n.github.io/convolutional-networks/#conv. Or at the tensorflow API: tensorflow.org/versions/r0.9/api_docs/python/nn.html#conv2d Filters are supposed to have shape `[filter_height, filter_width, in_channels, out_channels]`. – MarvMind Jul 18 '16 at 5:55

May I add the thing with "1x1 convolutions are 1 x 1 x number of channels of the input" to your answer? This was the source of my confusion and I keep forgetting this. – Martin Thoma Jul 18 '16 at 7:17

Sure, go ahead! – MarvMind Jul 18 '16 at 17:29

I

A fully connected layer (for input size $n * n$ over with i channels, and m output neurons) IS NOT equivalent to a 1x1 convolution layer but rather to an nxn convolution layer (i.e. a big kernel, same size as input- no pad) with number of filters equal to the FC output/hidden layer (i.e. m filters)

As you asked, it has the same number of parameters as the FCN, i.e. $n * n * i * m$ (plus bias):

FCN: $n * n * i$ (weights per input layer= input∗channels) $*m$ (times output/hidden layer width)

CNN: $n * n$ (each kernel) $*i$ (kernel per input channel) $*m$ (number of filters)

(Source)

edited Oct 9 at 10:04

answered Oct 8 at 17:19

Michael Yahalom
31 2

Of course these two things are equivalent in the special case $n = 1$; I think that's where the confusion comes in. – Yibo Yang Oct 15 at 21:20

The equivalent kernel simply has whatever shape the input has, and computes a tensor dot product. (I use the word "shape" as there seems to be some confusion over "size", which often ignores the channel/depth dimension). There's no "sliding the kernel across input" involved, since the kernel is as large as it can be. Quoting Stanford CS 231n course notes:

> any FC layer can be converted to a CONV layer. For example, an FC layer with K=4096 that is looking at some input volume of size 7×7×512 can be equivalently expressed as a CONV layer with F=7,P=0,S=1,K=4096,F=7,P=0,S=1,K=4096. In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will

> simply be 1×1×4096 since only a single depth column “fits” across the input volume, giving identical result as the initial FC layer.

I believe "F=7,P=0,S=1,K=4096,F=7,P=0,S=1,K=4096" here means each conv kernel has shape 7x7x512, and there's 4096 such filters.

The earlier answer mentioned that the last fc of AlexNet (which receives input with shape 1x1x4096 and computes 1000 class scores) is implemented as "1x1 convolution". To be complete, each such conv kernel has **shape** 1x1x4096, and there's 1000 of them.

Le Cunn also explains this in the CNN paper, page 8, description of LeNet5:

> Layer C5 is a convolutional layer with 120 feature maps. Each unit is connected to a 5x5 neighborhood on all 16 of S4's feature maps. Here because the size of S4 is also 5x5, the size of C5's feature maps is 1x1; this amounts to a full connection between S4 and C5.

answered Oct 15 at 21:19

Yibo Yang
111 4