1x1 Convolution: Demystified

Shedding light on the concept of 1×1 convolution operation which appears in paper, Network in Network by Lin et al. and Google Inception

Anwesh Marwade Feb 19 · 7 min read

Photo by Liam Charmer on Unsplash

Having read the Network in Network (NiN) paper by Lin et al a while ago, I came across a peculiar convolution operation that allowed cascading across channel parameters, enabling learning of complex interactions by pooling cross-channel information. They called it the “cross channel parametric pooling layer” (if I remember correctly), comparing it to an operation involving convolution with a 1x1 convolutional kernel.

Skimming over the details at the time (as I often do with such esoteric terminology), I never thought I would be writing about this operation, let alone providing my own thoughts on its workings. But as it goes, it's usually the terminology that seems formidable and not the so much the concept itself; which is quite useful! Having completed the back-story and a pause for effect, let us demystify this peculiar but multi-purpose, **1x1 convolutional layer.**

1×1 Convolution:

As the name suggests, the 1x1 convolution operation involves convolving the input with filters of size 1x1, usually with zero-padding and stride of 1.

Taking an example, let us suppose we have a (general-purpose) convolutional layer which outputs a tensor of shape ***(B, K, H, W)*** where*,*

- ***B*** represents the batch-size.
- ***K*** is the number of convolutional filters or kernels
- ***H, W*** are the spatial dimensions i.e. Height and Width.

> In addition, we specify a filter size that we want to work with, which is a single number for a square filter i.e. **size=3 implies a 3x3 filter**.

Feeding this tensor into our 1x1 convolution layer with ***F*** filters (zero-padding and stride 1), we will get an output of shape ***(B, F, H, W)*** changing our filter dimension from ***K*** to ***F.*** Sweet!

Now depending on whether ***F*** is *less or greater* than ***K***, we have either *decreased or increased* the dimensionality of our input in the filter space without applying any spatial transformation (you'll see). All this using the 1x1 convolution operation!

But wait, how was this any different from a regular convolution operation? In a regular convolution operation, we usually have a larger filter size like say, a 3x3 or 5x5 (or even 7x7) kernels which then generally entail some kind of padding to the input which in turn transforms it's spatial dimensions of **H x W** to some **H' x W';** capisce? If not, here is the link for my go-to article for **any** (believe me) clarification on Convolutional Nets and its operations.

CS231n Convolutional Neural Networks for Visual Recognition

Table of Contents: Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous...

cs231n.github.io

Benefits?

In CNNs, we often use some kind of pooling operations to spatially down-sample the activation maps of a convolution output in order to keep things from becoming intractable. The danger of intractability is due to the number of generated activation maps which increases or rather blows up dramatically, proportional to the depth of a CNN. That is, the deeper a network, the larger the number of activation maps it generates. The problem is further exacerbated if the convolution operation is using large-sized filters like 5x5 or 7x7 filters, resulting in a significantly high number of parameters.

> ***Understand:*** *A* ***Filter*** *refer to the* ***kernel*** *that is being applied over the input in a sliding window fashion as a part of a convolution operation. An* ***Activation Map*** *on the other hand is the output result of applying a single filter over the input image. A convolution operation with multiple filters usually generates multiple (stacked) activation maps.*

Spatial downsampling (through pooling) is achieved by aggregating information along the spatial dimensions by reducing the height and width of the input at every step. While it maintains important spatial features to some extent, there does exist a trade-off between down-sampling and information loss. Bottom line: we can only apply pooling to a certain extent.

The 1x1 convolution can be used to address this issue by offering filter-wise pooling, acting as a projection layer that pools (or projects) information across channels and enables dimensionality reduction by reducing the number of filters whilst retaining important, feature-related information.

This was heavily used in Google's inception architecture (link in references) where they state the following:

> *One big problem with the above modules, at least in this naive form, is that even a modest number of 5x5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters.*
>
> *This leads to the second idea of the proposed architecture: judiciously applying dimension reductions and projections wherever the computational requirements would increase too much otherwise. This is based on the success of* ***embeddings****:* ***even low dimensional embeddings might contain a lot of information about a relatively large image patch.*** *Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose.*

They introduced the use of 1x1 convolutions to compute reductions before the expensive 3x3 and 5x5 convolutions. Instead of spatial dimensionality reduction using pooling, reduction may be applied in the filter dimension using 1x1 convolutions.

An interesting line of thought was provided by **Yann LeCun** where he analogizes fully connected layers in CNNs as simply convolution layers with 1x1 convolution kernels and a full connection table. See this post from 2015:

Log into Facebook

Log into Facebook to start sharing and connecting with your friends, family, and people you know.

www.facebook.com

Implementation

Having talked about the concept, time to see some implementation, which is quite easy to follow with some basic PyTorch experience.

```
class OurConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.projection = None
        # Input dims expected: HxWxC = 36x36x3

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=
```

```python
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding
        # Output dims: HxWxC = 36x36x32

        # softconv is the 1x1 convolution: Filter dimensions go from 32 -> 1 implies Output dims
        self.softconv = nn.Conv2d(in_channels=32, out_channels=1, kernel_size=1, stride=1, paddi

    def forward(self, input_data):
        # Apply convolution
        x = self.conv1(input_data)
        # Apply tanh activation
        x = torch.tanh(x)
        # Apply the 1x1 convolution
        x = self.softconv(x)
        # Apply sigmoid activation
        x = torch.sigmoid(x)
        # Save the result in projection
        self.projection = x
```

1x1 ConvNet.py hosted with ♡ by GitHub view raw

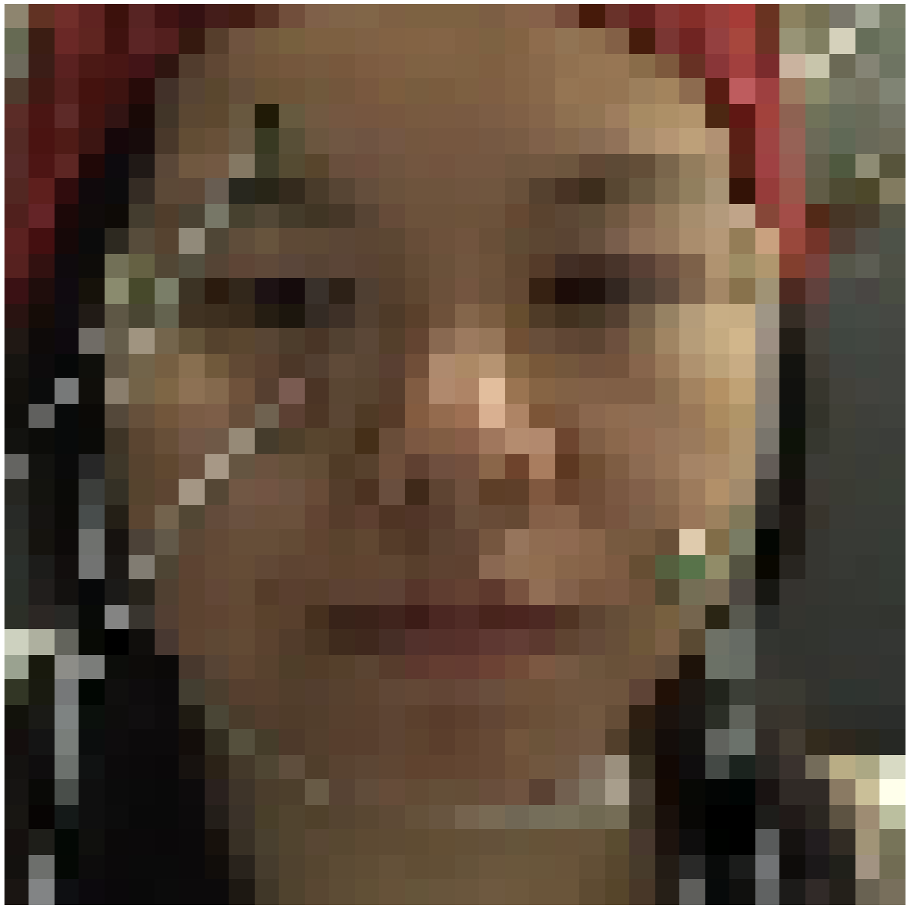
Code from my project on replicating DeepPhys. Code by Author

Here, I've used ***softconv*** to denote the 1x1 convolution. This is a code snippet from a recent project, where the 1x1 convolution was used for projecting the information across the filter dimension (32 in this case) and pooling it into a single dimension. This brought three benefits to my use-case:

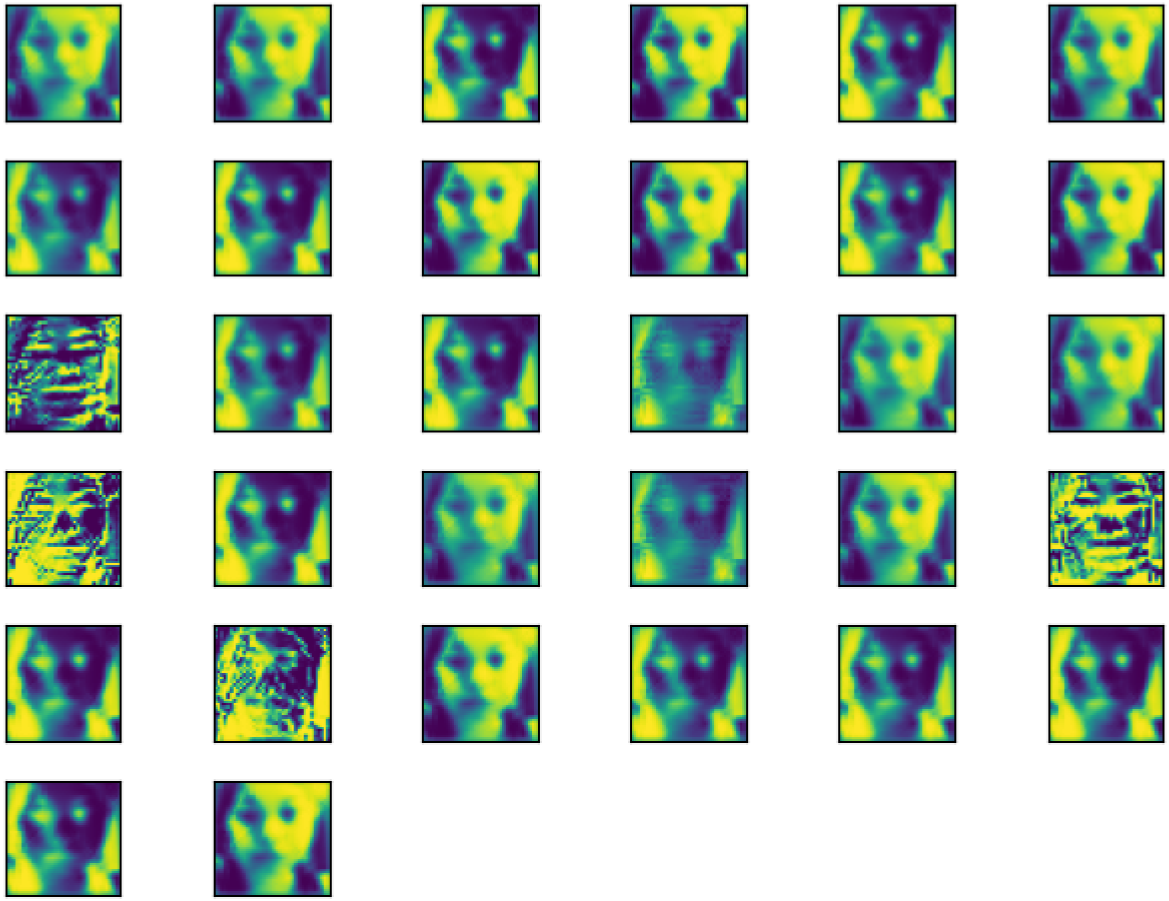
1. Projecting the information from 32 two-dimensional activation maps (which are generated by convolving the filters of the previous convolution layer over the input) into a single activation map (remember, the output from the ***softconv*** layer has filter dim $= 1$). Why did I need such a projection, this is explained along with the images below.
2. Reducing the dimensionality using 1x1 convolutions, allowed for keeping computational requirements in check. This is based on the intuition and success of low-dimensional embeddings which have worked quite well in the Natural Language Processing domain.
3. Offers better tractability in terms of ease of visualising the filter weights which will be learned i.e. being able to answer what the layer is actually doing to a given input and understanding what model is learning.

> Additionally, the tanh and sigmoid activations are implementation details that can be ignored for this post.

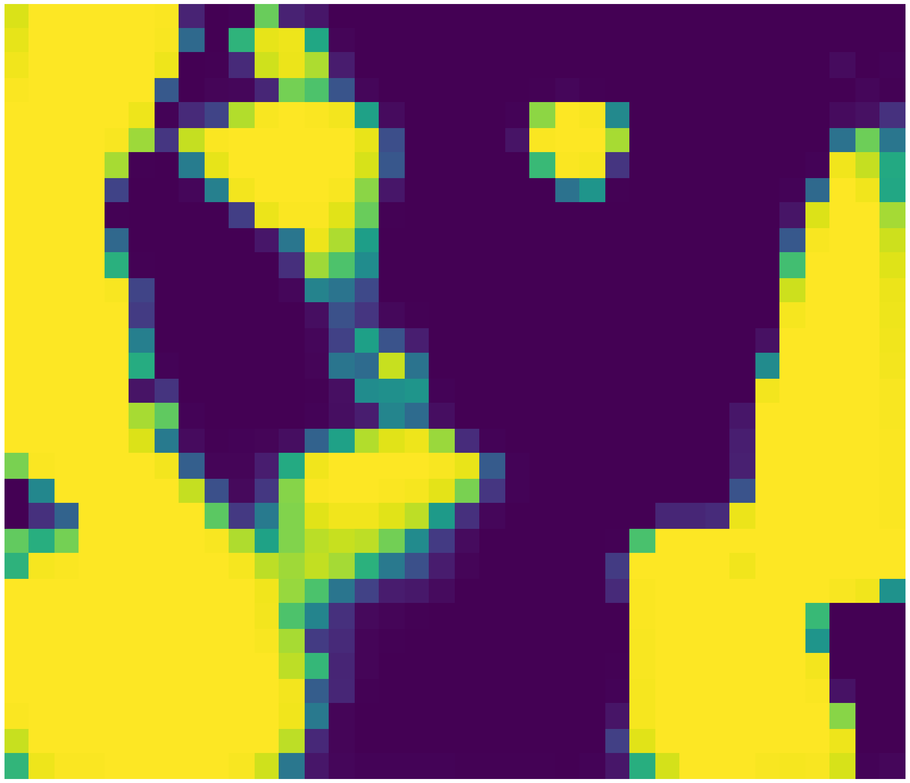
To drive home the idea of possibly using a 1x1 convolution, I am providing an example use-case from a model trained on the DEAP emotion dataset without delving into many details. The model is trained to predict heart rate signals from facial videos (images). Here, the pooling of information from 32 filters (obtained from previous convolutions) using the 1x1 convolutional layer into a single channel allows the model to create a mask whereby it is able to differentiate between skin and non-skin pixels. This is a work in progress but hopefully one can see the point of using a 1x1 convolution here.



A facial image being used as input. Image from DEAP dataset



State of 32 filters after applying the regular convolutional layers. Image by Author.



The output as obtained from the softconv layer where 32 filters have been pooled into a single channel. Image by Author

Key Takeaways:

1. ***1x1 convolution*** can be seen as an operation where a ***1 x 1 x K*** sized filter is applied over the input and then weighted to generate F activation maps.

2. ***F > K*** results in an increase in the filter dimension whereas ***F < K*** would cause an output with reduced filter dimensions. The value of F is quite dependent on the project requirement. Reduced dimensionality might

help in making things more tractable and there may be a similar use-case for increasing the number of filters using **1x1 convolutions**.

3. Theoretically, using this operation, the neural network can 'choose' which input 'filters' to look at using the **1x1 convolution**, instead of force-multiplying everything.

References:

[1] Stats StackExchange: https://stats.stackexchange.com/questions/194142/what-does-1x1-convolution-mean-in-a-neural-network

[2] Google Inception Architecture: Going Deeper with Convolutions by Lin et al.

[3] Article: https://iamaaditya.github.io/2016/03/one-by-one-convolution/

[4] https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural-networks/

[5] "DEAP: A Database for Emotion Analysis using Physiological Signals", S. Koelstra, C. Muehl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, I. Patras, IEEE Transactions on Affective Computing, vol. 3, no. 1, pp. 18–31, 2012

Deep Learning Convolutional Network Machine Learning Data Science