Search Medium

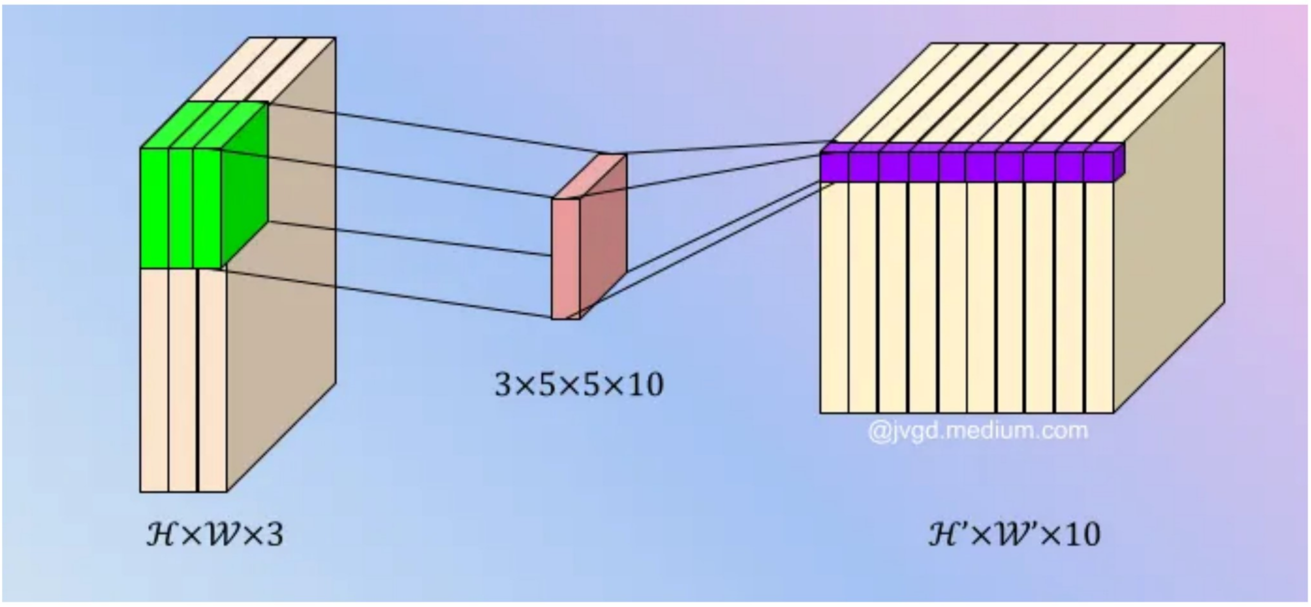
3×5×5×10
@jvgd.medium.com
$\mathcal{H}\times\mathcal{W}\times 3$
$\mathcal{H}'\times\mathcal{W}'\times 10$

Image by Author

# Pytorch Conv2d Weights Explained

Understanding weights dimension, visualization, number of parameters and the infamous size mismatch

Javier · Following
Published in Towards Data Science
5 min read · Nov 26, 2021

Listen Share More

One of the most common problems I have found in my journey with Pytorch is the size mismatch error when uploading weights to my models. As you know, Pytorch does not save the computational graph of your model when you save the model weights (on the contrary to TensorFlow). So when you train multiple models with different configurations (different depths, width, resolution…) it is very common to misspell the weights file and upload the wrong weights for your target model.

This misspell translates into the infamous Pytorch error for the Conv2d weights: the size mismatch. Here you have it:

```
In [11]: conv_layer = nn.Conv2d(3, 15, 5,5)
In [12]: conv_layer.load_state_dict(weights)
---------------------------------------------------------------------------
RuntimeError    Traceback (most recent call last)

[... Traceback ommited for this post... @jvgd]

RuntimeError: Error(s) in loading state_dict for Conv2d:
 size mismatch for weight: copying a param with shape torch.Size([10,
3, 5, 5]) from checkpoint, the shape in current model is
torch.Size([15, 3, 5, 5]).
 size mismatch for bias: copying a param with shape torch.Size([10])
from checkpoint, the shape in current model is torch.Size([15]).
```

Sound familiar? If so, I might have some insights to share with you about how the Pytorch Conv2d weights are and how you can understand them.

## Conv2d

The Conv2d Layer is probably the most used layer in Computer Vision (at least until the transformers arrived) If you have ever instantiated this layer in Pytorch you would probably have coded something like:

```
In [5]: conv_layer = nn.Conv2d(in_channels=3, out_channels=10,
kernel_size=5)
```

You and I would normally use the layer without inspect it too much but since we are here to get our hands dirty let's look under the hood. If you go ahead and inspect the layer weights you can check that the weights dimensions are:

```
In [6]: conv_layer.weight.shape
Out[6]: torch.Size([10, 3, 5, 5])
```

So how do you interpret this?

Why does the weights tensor have 4 dimensions? Should not be a 2d tensor since it is a convolution in a 2d plane? Well, lets try to visualize it first and analyze it after.

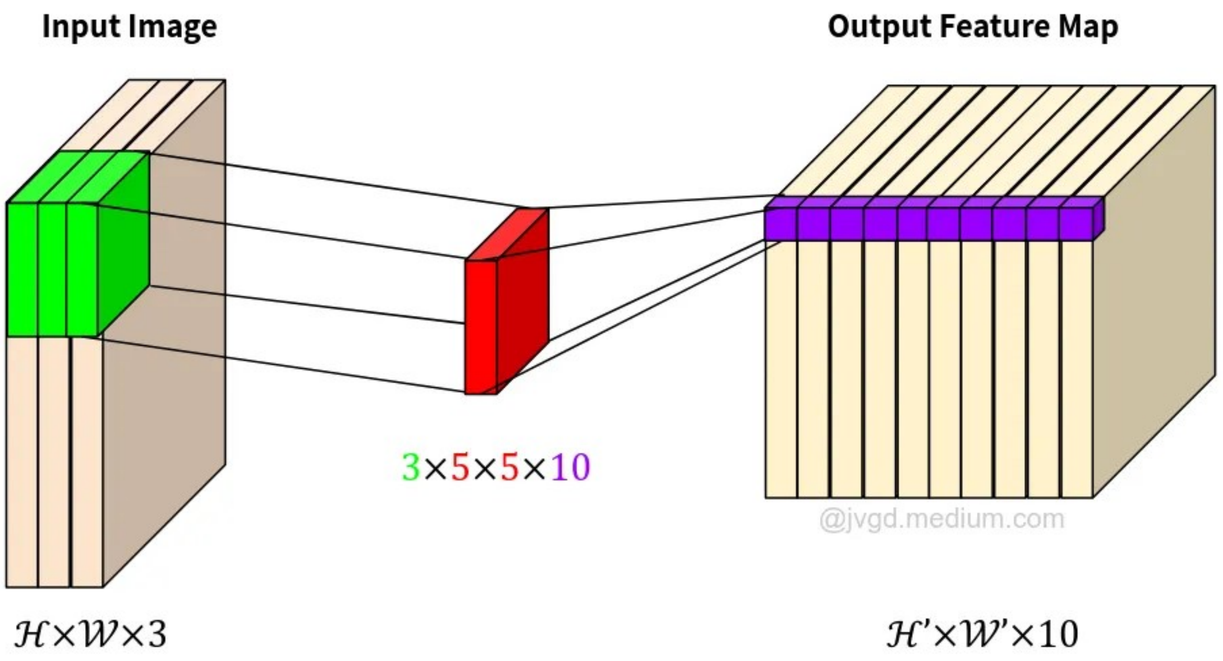
Input Image
Output Feature Map
3×5×5×10
@jvgd.medium.com
$\mathcal{H}\times\mathcal{W}\times3$
$\mathcal{H}'\times\mathcal{W}'\times10$

Image by author

If you feed this convolutional layer with an image of [H,W,3] you would expect an output feature map of [H',W',10]. The weights of the convolutional layer for this operation can be visualized as the figure above.

In the figure it can be seen how the 5x5 kernel is being convolved with all the 3 channels (R,G,B) from the input image. In this sense we would need the 5x5 kernel to have weights for every single input channel. This naturally translates to a tensor of shape [3,5,5].

On the output side, we have set that we want an output feature map of 10 channels. That means that for every convolution step we want an output of [1,1,10] (the purple tensor in the figure). This expansion from input to output channels is supported by additional weights. So the final tensor of our convolutional layer weights is: [3,5,5,10] (read from left to right as we are used to).

Then, why does the weights output shape in the code snippet has a shape of [10,3,5,5]? Fair question. This is due to an implementation issue. You already know that Pytorch works with channel first approach. Meaning that for processing an input image you would need to cast it as a tensor of [C, H, W]. This is not natural for us but helps in the implementation. So when we read the weights shape of a Pytorch convolutional layer we have to think it as:

```
[out_ch, in_ch, k_h, k_w]
```

Where k_h and k_w are the kernel height and width respectively.

Ok, but does not the convolutional layer also have the bias parameter as weights? Yes, you are right, let's check it:

```
In [7]: conv_layer.bias.shape
Out[7]: torch.Size([10])
```

The bias term it is just a single vector of same dimensions of output channels that is added to the output of the convolution. Visually:

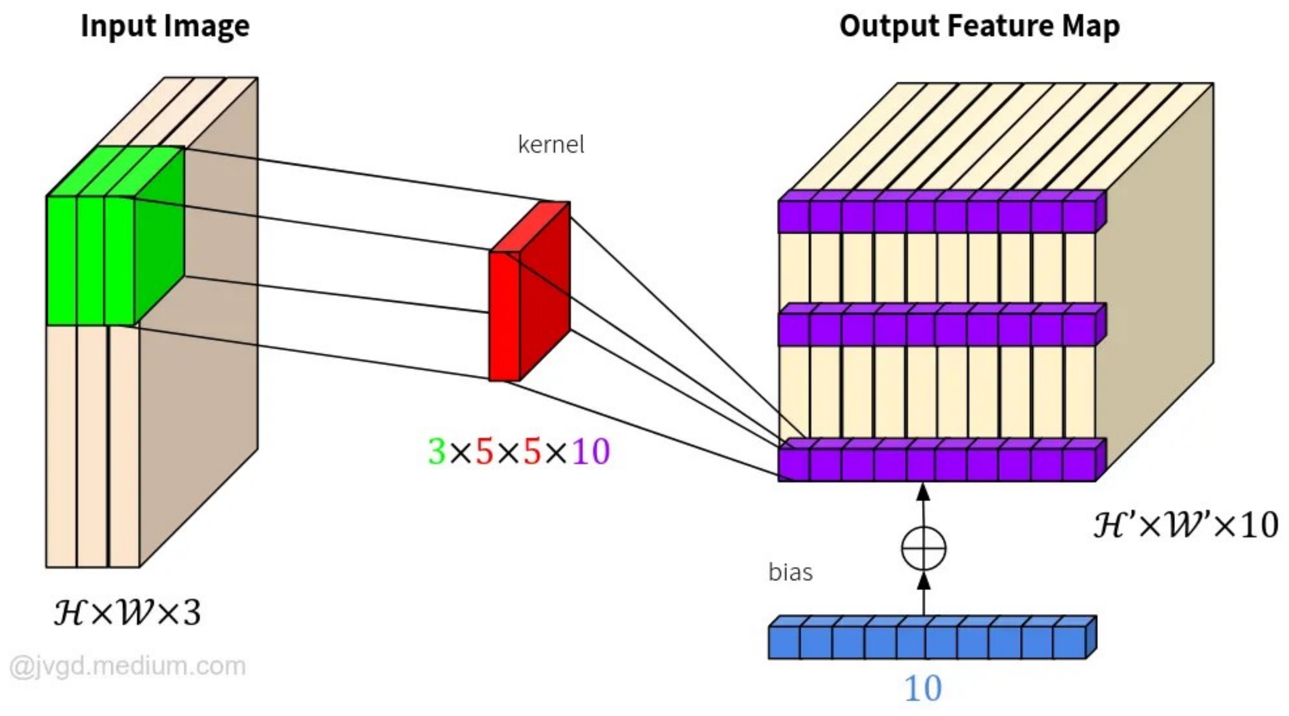

Image by author

So in the end, all things considered we have:

- An input tensor of [H,W,3]
- Convolved by a tensor of [3,5,5,10]

- Giving an output feature map of [H',W',10]
- To which the layer bias [10] is added to every channel component

That is all for the weights. Then, the next question is: how many parameters am I adding to my model with this layer? The answer to that is very straight forward. First let's compute it numerically:

```
In [8]: sum(param.numel() for param in conv_layer.parameters())
Out[8]: 760
```

As you may have guessed that is just the result of the weights dimensions plus the bias:

$$[3 \cdot 5 \cdot 5 \cdot 10] + [10] = 760$$

Image by author

The natural generalization of this to compute the number of parameters per convolutional layer is:

$$Params = C_{in} \cdot k_h \cdot k_w \cdot C_{out} + C_{out}$$

Image by author

After seen all this, we can return to the error that introduced the post:

```
RuntimeError: Error(s) in loading state_dict for Conv2d:
 size mismatch for weight: copying a param with shape torch.Size([10,
3, 5, 5]) from checkpoint, the shape in current model is
torch.Size([15, 3, 5, 5]).
 size mismatch for bias: copying a param with shape torch.Size([10])
from checkpoint, the shape in current model is torch.Size([15]).
```

With this knowledge about the Pytorch Conv2d layer we already know what the error is telling us:

- We are trying to load the Weights of a convolutional layer that expected an input image of 3 channels and would return a feature map of 10 with a kernel of [5,5]

- However the current model has a convolutional layer that, similar to the previous one, expects an input image of 3 channels with a [5,5] kernel but that would return a feature map of 15 instead for 10

So in the end, this is almost always caused by a misspelling, however if we are to debug our models we need to really dig into how they are built inside.

Today we have seen the Conv2d layer, in the future, if I am able to free some time slot for it, I will be writing another post of other famous Pytorch layers.

AI Deep Learning Pytorch Convolutional Network

Following

**Written by Javier**

134 Followers · Writer for Towards Data Science

AI Research Engineer in Deep Learning. Living between the math and the code. A philosophic seeker interested in the meaning of everything from quarks to AI.

**More from Javier and Towards Data Science**