How to calculate the number of parameters for convolutional neural network?

Asked 2 years, 4 months ago Active 5 months ago Viewed 32k times

33

40

I'm using Lasagne to create a CNN for the MNIST dataset. I'm following closely to this example: Convolutional Neural Networks and Feature Extraction with Python.

The CNN architecture I have at the moment, which doesn't include any dropout layers, is:

```
NeuralNet(
    layers=[('input', layers.InputLayer),        # Input Layer
            ('conv2d1', layers.Conv2DLayer),     # Convolutional Layer
            ('maxpool1', layers.MaxPool2DLayer), # 2D Max Pooling Layer
            ('conv2d2', layers.Conv2DLayer),     # Convolutional Layer
            ('maxpool2', layers.MaxPool2DLayer), # 2D Max Pooling Layer
            ('dense', layers.DenseLayer),        # Fully connected layer
            ('output', layers.DenseLayer),       # Output Layer
            ],
    # input layer
    input_shape=(None, 1, 28, 28),

    # layer conv2d1
    conv2d1_num_filters=32,
    conv2d1_filter_size=(5, 5),
    conv2d1_nonlinearity=lasagne.nonlinearities.rectify,

    # layer maxpool1
    maxpool1_pool_size=(2, 2),

    # layer conv2d2
    conv2d2_num_filters=32,
    conv2d2_filter_size=(3, 3),
    conv2d2_nonlinearity=lasagne.nonlinearities.rectify,

    # layer maxpool2
    maxpool2_pool_size=(2, 2),


    # Fully Connected Layer
    dense_num_units=256,
    dense_nonlinearity=lasagne.nonlinearities.rectify,

   # output Layer
    output_nonlinearity=lasagne.nonlinearities.softmax,
    output_num_units=10,

    # optimization method params
    update= momentum,
    update_learning_rate=0.01,
    update_momentum=0.9,
    max_epochs=10,
    verbose=1,
    )
```

```
  #  name      size
---  --------  --------
  0  input     1x28x28
  1  conv2d1   32x24x24
  2  maxpool1  32x12x12
  3  conv2d2   32x10x10
  4  maxpool2  32x5x5
  5  dense     256
  6  output    10
```

and outputs the number of learnable parameters as **217,706**

I'm wondering how this number is calculated? I've read a number of resources, including this StackOverflow's question, but none clearly generalizes the calculation.

If possible, *can the calculation of the learnable parameters per layer be generalised?*

For example, convolutional layer: number of filters x filter width x filter height.

neural-network deep-learning conv-neural-network lasagne nolearn

edited Nov 5 '17 at 16:54

nbro
6,130 10 55 104

asked Mar 14 '17 at 12:59

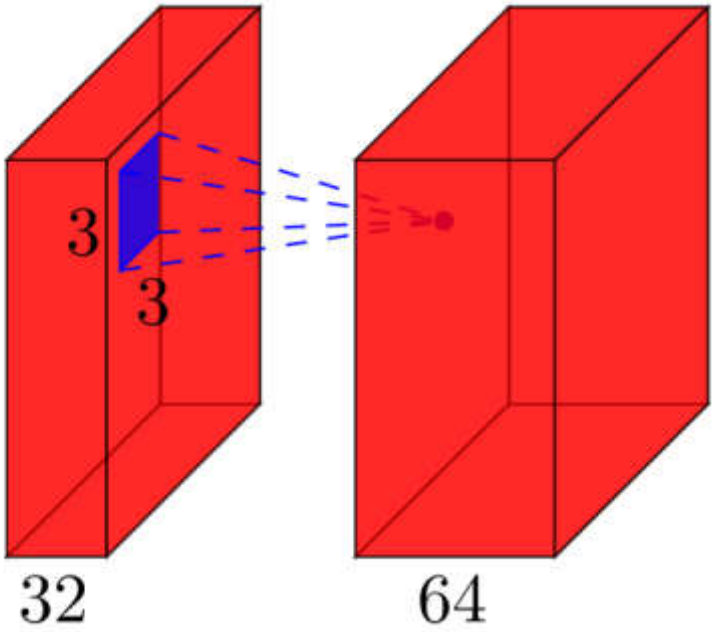
Waddas
468 1 10 21

2 Answers

76

Let's first look at how the number of learnable parameters is calculated for each individual type of layer you have, and then calculate the number of parameters in your example.

- **Input layer**: All the input layer does is read the input image, so there are no parameters you could learn here.
- **Convolutional layers**: Consider a convolutional layer which takes `l` feature maps at the input, and has `k` feature maps as output. The filter size is `n` x `m`. For example, this will look like this:



Here, the input has `l=32` feature maps as input, `k=64` feature maps as output, and the filter size is `n=3` x `m=3`. It is important to understand, that we don't simply have a 3x3 filter, but actually a 3x3x32 filter, as our input has 32 dimensions. And we learn 64 different 3x3x32 filters. Thus, the total number of weights is `n*m*k*l`. Then, there is also a bias term for each feature map, so we have a total number of parameters of `(n*m*l+1)*k`.

- **Pooling layers**: The pooling layers e.g. do the following: "replace a 2x2 neighborhood by its maximum value". So there is no parameter you could learn in a pooling layer.
- **Fully-connected layers**: In a fully-connected layer, all input units have a separate weight to each output unit. For `n` inputs and `m` outputs, the number of weights is `n*m`. Additionally, you have a bias for each output node, so you are at `(n+1)*m` parameters.
- **Output layer**: The output layer is a normal fully-connected layer, so `(n+1)*m` parameters, where `n` is the number of inputs and `m` is the number of outputs.

The final difficulty is the first fully-connected layer: we do not know the dimensionality of the input to that layer, as it is a convolutional layer. To calculate it, we have to start with the size of the input image, and calculate the size of each convolutional layer. In your case, Lasagne already calculates this for you and reports the sizes - which makes it easy for us. If you have to calculate the size of each layer yourself, it's a bit more complicated:

- In the simplest case (like your example), the size of the output of a convolutional layer is `input_size - (filter_size - 1)`, in your case: 28 - 4 = 24. This is due to the nature of the convolution: we use e.g. a 5x5 neighborhood to calculate a point - but the two outermost rows and columns don't have a 5x5 neighborhood, so we can't calculate any output for those points. This is why our output is 2*2=4 rows/columns smaller than the input.
- If one doesn't want the output to be smaller than the input, one can zero-pad the image (with the `pad` parameter of the convolutional layer in Lasagne). E.g. if you add 2 rows/cols of zeros around the image, the output size will be (28+4)-4=28. So in case of padding, the output size is

expression becomes `((input_size + 2*padding - filter_size)/stride) +1` .

In your case, the full calculations are:

```
  #  name                          size                 parameters
---  --------  ------------------------    ----------------------
  0  input                      1x28x28                          0
  1  conv2d1   (28-(5-1))=24 -> 32x24x24    (5*5*1+1)*32    =     832
  2  maxpool1                   32x12x12                         0
  3  conv2d2   (12-(3-1))=10 -> 32x10x10    (3*3*32+1)*32   =   9'248
  4  maxpool2                    32x5x5                          0
  5  dense                          256    (32*5*5+1)*256 = 205'056
  6  output                          10    (256+1)*10      =   2'570
```

So in your network, you have a total of 832 + 9'248 + 205'056 + 2'570 = 217'706 learnable parameters, which is exactly what Lasagne reports.

edited Jun 27 '18 at 12:46

Løiten
2,474 2 15 28

answered Mar 14 '17 at 13:33

hbaderts
12.2k 3 34 43

Great answer, thank you. Only thing's that i'm still confused about is how the convolutional layers size is calculated. I'm unsure where the 24x24 and 10x10 come from. – Waddas Mar 14 '17 at 16:46

I added more details about the size calculation in convolutional layers - please let me know if this helps. – hbaderts Mar 15 '17 at 7:36

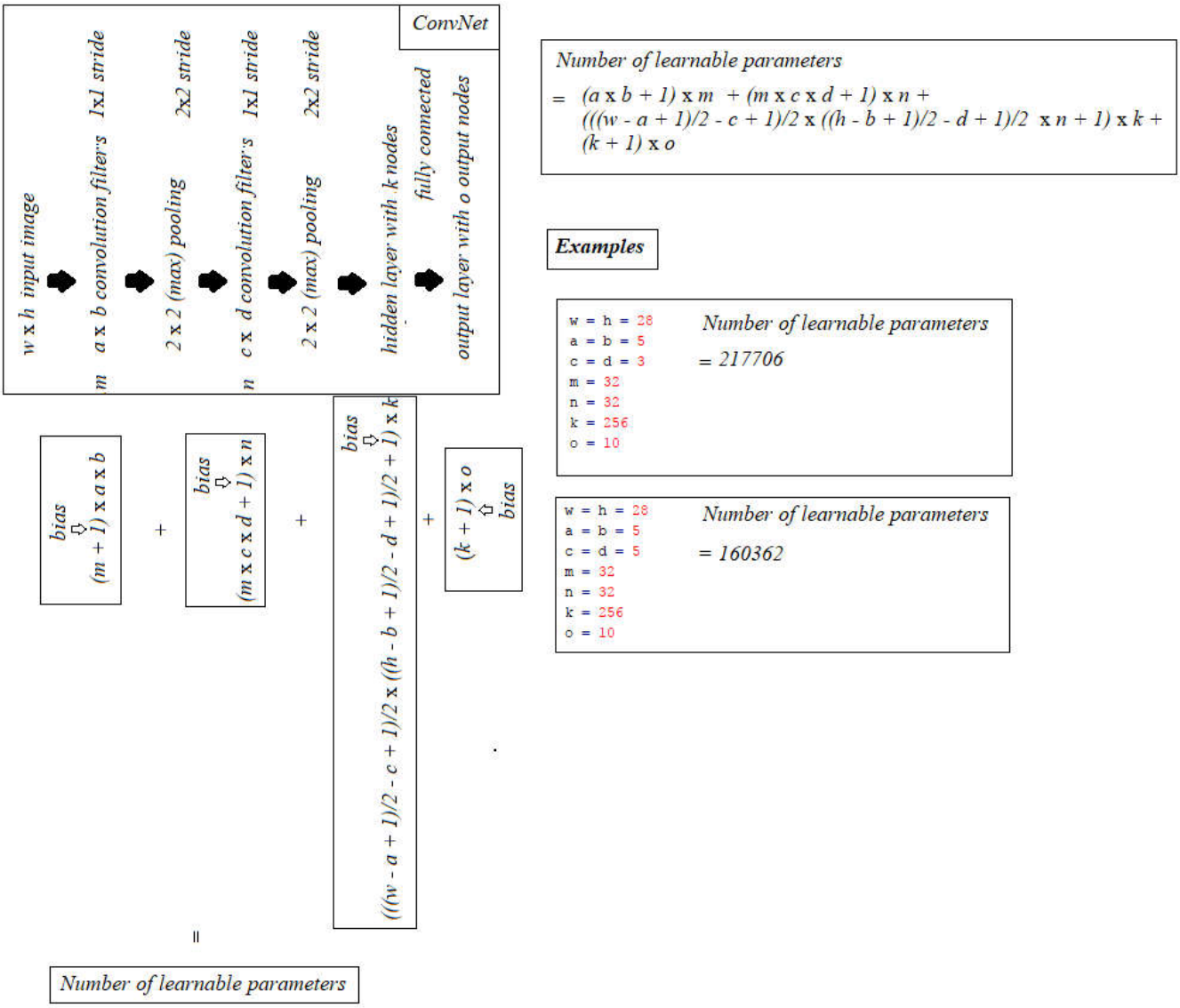
Hi @hbaderts , I had another question. Based on this table that you guys have here, the model size refers to the sum of all individual sizes here, correct? For a CNN, is it sensible to understand that the model size is inversely proportional to the number of learnable params? Please would you take a look at stackoverflow.com/questions/43443342/… ? – Raaj Apr 17 '17 at 14:10

@hbaderts, your explanation is very helpful, but I am confused why you deal with bias a 1 in ((n*m*l+1)*k), if I have 16 output features, so the bias will also be 16, isn't it? so we have to add 16 to above formula? – Hunar A.Ahmed Nov 28 '18 at 8:27

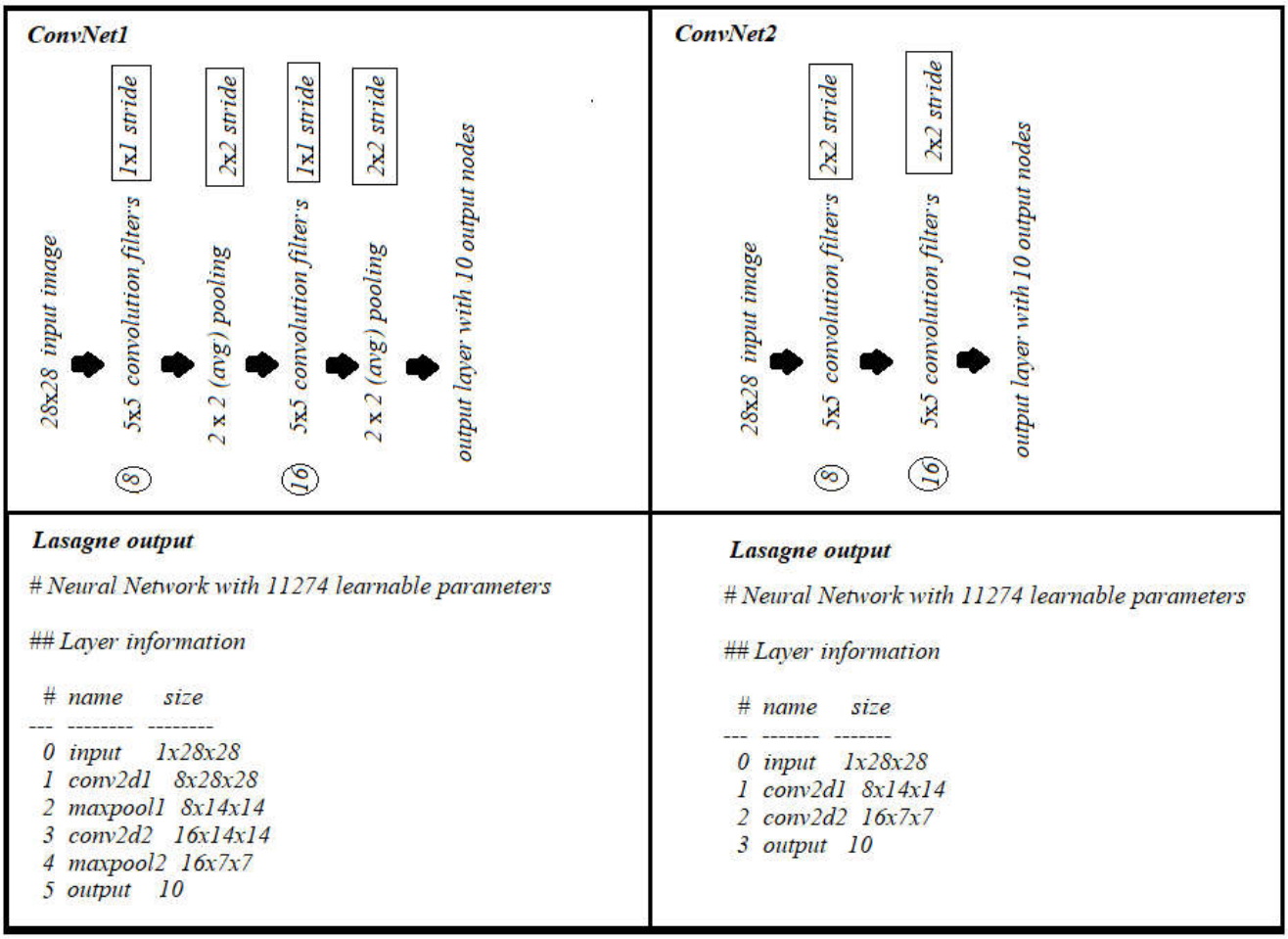
1 @honar.cs if you have 16 output features, then `k=16` . The equation is `(n*m*l+1)*k` , the `+1` is inside the parentheses. So the `+1` is multiplied by 16 too, giving `n*m*l*16 + 16` for your example. Does this help? – hbaderts Nov 30 '18 at 12:14

7

building on top of @hbaderts's excellent reply, just came up with some formula for a I-C-P-C-P-H-O network (since i was working on a similar problem), sharing it in the figure below, may be helpful.



Also, (1) convolution layer with 2x2 stride and (2) convolution layer 1x1 stride + (max/avg) pooling with 2x2 stride, each contributes same numbers of parameters with 'same' padding, as can be seen below:



edited Aug 11 '17 at 7:13

answered Aug 10 '17 at 18:27

Sandipan Dey

14.4k 2 19 34