Computer Vision

Ideas are cross-field and can be helpful on other areas. Some of the computer vision problems are:

- Image Classification
- Object Detection
- Neural Style Transfer
- Image Similarity

The inputs can be really big as each pixel of each color layer can be seen as an input. This super large inputs in a fully connected neural network make the weights matrix pretty large as well. With this many parameters is hard to get enough data or even avoid the data to overfit.

Edge Detection Example

How do you detect edges in a given image? Having an input matrix representing the input image we can use the convolution operation with a much smaller filter, sometimes also called kernel, to convolve over the input matrix. The result with the correct given filter can detect vertical edges, horizontal edges, etc. The resulting matrix has smaller dimensions and its values can be interpreted as whereas an edge exists in that specific given space or not.

More Edge Detection

From the literature the Sobel filter and the Scharf filter were hand-designed to detect edges giving more weight to the central pixel. Although wouldn’t it be good if we’ve let the algorithm learn the best filters to achieve the best results in our application as well? The filters are much smaller than the input images and this makes them much easier to learn than a direct mapping from the input.

Padding

If you have an $n \times n$ image and a $f \times f$ filter the output of your filter is given by $n - f + 1 \times n - f + 1$. But sometimes you don’t want to reduce the information you’re giving the next layers or you want to give the edges the same amount of importance as an internal pixel. To fix this padding add extra information around the original picture and the output now is given by $n + 2p - f + 1 \times n + 2p - f + 1$.

In terms of how much to pad there are two available solutions: **valid** and **same**.

The **valid** ouptut size is $n - f + 1$ and the **same** output size is $n + 2p - f + 1$ given that $p = \frac{f-1}{2}$ outputs the same size as the input, $n \times n$

f is usually odd for two main reasons:

1. It only allows same padding with odd values of f
2. Odd has a center pixel position which helps understanding which pixel the filter convolves around

Strided convolutions

Stride is the step the kernel jumps every time it goes to the next row or line. If you have an $n \times n$ image and a $f \times f$ filter with padding p and stride s the output of your filter is given by $\lfloor \frac{n+2p-f}{s} + 1 \rfloor \times \lfloor \frac{n+2p-f}{s} + 1 \rfloor$

Convolutions Over Volume

If you have an $n \times n \times n_c$ image (where n_c is the number of channels, also referred as depth, in your image) and a $f \times f \times n_c$ filter with padding p and stride s the output of your filter is given by $\lfloor \frac{n+2p-f}{s} + 1 \rfloor \times \lfloor \frac{n+2p-f}{s} + 1 \rfloor \times n_c^{'}$ where the $n_c^{'}$ is given by the number of filters you used in the last layer.

One Layer of a Convolutional Network

If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?

Well, each filter has 27 parameters +1 (bias) and as we have 10 filter we have 280 parameters. Comparing this size to any given input n we can expect this to be much faster to train and much less prone to overfit the data.

If layer l is a convolution layer:

- $f^{[l]}$ is the filter size
- $p^{[l]}$ is the padding
- $s^{[l]}$ is stride
- Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$
- Output: $n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- $n^{[l]} = \lfloor \frac{n^{[l-1]}+2p^{[l]}-f^{[l]}}{s^{[l]}} + 1 \rfloor$
- Each filter is $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
- Activations: $a^{[l]} = n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$
- Weights: $a^{[l]} = f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- bias: $n_c^{[l]} = 1 \times 1 \times 1 \times n_c^{[l]}$

Simple Convolutional Network Example

Types of layer in a convolution network:

- Convolution
- Pooling
- Fully Connected

Pooling Layers

It has filter size and stride as hyperparameters as well. But instead of operating the element-wise operation it takes a value (max, min, avg). As the hyperparameters are the same the output layers sizes are the same as a regular convolution. This layer has no learnable parameters, thought. This means that no gradients calculations are performed on this type of layer and it's still able to shrink the data passed from the input of the network. Padding is usually 0.

CNN Example

When counting layers in the network one of the most accepted terminologies is to count only the layers that effectively have learnable parameters. As the pooling layer has no parameter it's usually counted as one with its respective convolution pair.

It's common to see the n_H, n_W decreases as deeper you get and the n_C increases

Why Convolutions?

1. Parameter sharing
   - A feature detector that's useful in one part of the image is probably useful in another part of the image as well
2. Sparsity of connections
   - In each layer, each output value depends only on a small number of inputs

Convolutional neural networks are very good at capturing translation invariance since the observation of cat's picture shifted a couple of pixels to the right, is still pretty clearly a cat.

Quiz

1.What do you think applying this filter to a grayscale image will do?

- ☐ Detect horizontal edges
- ☑ Detect vertical edges
- ☐ Detect image contrast
- ☐ Detect 45 degree edges

2.Suppose your input is a 300 by 300 color (RGB) image, and you are not using a convolutional network. If the first hidden layer has 100 neurons, each one fully connected to the input, how many parameters does this hidden layer have (including the bias parameters)?

- ☐ 9,000,001
- ☐ 9,000,100
- ☐ 27,000,001
- ☑ 27,000,100

3.Suppose your input is a 300 by 300 color (RGB) image, and you use a convolutional layer with 100 filters that are each 5x5. How many parameters does this hidden layer have (including the bias parameters)?

- ☐ 2501
- ☐ 2600
- ☐ 7500
- ☑ 7600

4.You have an input volume that is 63x63x16, and convolve it with 32 filters that are each 7x7, using a stride of 2 and no padding. What is the output volume?

- ☐ 16x16x32
- ☐ 16x16x16
- ☐ 29x29x16
- ☑ 29x29x32

5.You have an input volume that is 15x15x8, and pad it using “pad=2.” What is the dimension of the resulting volume (after padding)?

- ☐ 19x19x12
- ☐ 17x17x8
- ☑ 19x19x8
- ☐ 17x17x10

6.You have an input volume that is 63x63x16, and convolve it with 32 filters that are each 7x7, and stride of 1. You want to use a “same” convolution. What is the padding?

- ☐ 1
- ☐ 2
- ☑ 3
- ☐ 7

7.You have an input volume that is 32x32x16, and apply max pooling with a stride of 2 and a filter size of 2. What is the output volume?

- ☐ 15x15x16
- ☐ 16x16x8
- ☑ 16x16x16
- ☐ 32x32x8

8.Because pooling layers do not have parameters, they do not affect the backpropagation (derivatives) calculation.

- ☐ True
- ☑ False

9.In lecture we talked about “parameter sharing” as a benefit of using convolutional networks. Which of the following statements about parameter sharing in ConvNets are true? (Check all that apply.)

- ☐ It allows gradient descent to set many of the parameters to zero, thus making the connections sparse.
- ☑ It allows a feature detector to be used in multiple locations throughout the whole input image/input volume.
- ☑ It reduces the total number of parameters, thus reducing overfitting.
- ☐ It allows parameters learned for one task to be shared even for a different task (transfer learning).

10.In lecture we talked about “sparsity of connections” as a benefit of using convolutional layers. What does this mean?

- ☑ Each activation in the next layer depends on only a small number of activations from the previous layer.
- ☐ Each layer in a convolutional network is connected only to two other layers
- ☐ Each filter is connected to every channel in the previous layer.
- ☐ Regularization causes gradient descent to set many of the parameters to zero.