

1 Convolution and Backpropagation

In this problem, we will walk through how discrete 2D convolution works and how we can use the backpropagation algorithm to compute derivatives through this operation.

- (a) We have an image I that has three color channels I_r, I_g, I_b each of size $W \times H$ where W is the image width and H is the height. Each color channel represents the intensity of red, green, and blue for each pixel in the image. We also have a filter G with finite support. The filter also has three color channels, G_r, G_g, G_b , and we represent these as a $w \times h$ matrix where w and h are the width and height of the mask. (Note that usually $w \ll W$ and $h \ll H$.) The output $(I * G)[x, y]$ at point (x, y) is

$$(I * G)[x, y] = \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{r, g, b\}} I_c[x + a, y + b] \cdot G_c[a, b]$$

In this case, the size of the output will be $(1 + W - w) \times (1 + H - h)$, and we evaluate the convolution only within the image I . (For this problem we will not concern ourselves with how to compute the convolution along the boundary of the image.) To reduce the dimension of the output, we can do a strided convolution in which we shift the convolutional filter by s positions instead of a single position, along the image. The resulting output will have size $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (H - h)/s \rfloor$.

Write pseudocode to compute the convolution of an image I with a filter G and a stride of s .

- (b) Filters can be used to identify different types of features in an image such as edges or corners. **Design a filter G that outputs a large (in magnitude) value for vertically oriented edges in image I . By “edge,” we mean a vertical line where a black rectangle borders a white rectangle.** (We are not talking about a black line with white on both sides.)

- (c) Although handcrafted filters can produce edge detectors and other useful features, convolutional networks *learn* filters via the backpropagation algorithm. These filters are often specific to the problem that we are solving. Learning these filters is a lot like learning weights in standard backpropagation, but because the same filter (with the same weights) is used in many different places, the chain rule is applied a little differently and we need to adjust the backpropagation algorithm accordingly. In short, during backpropagation each weight w in the filter has a partial derivative $\frac{\partial L}{\partial w}$ that receives contributions from every patch of image where w is applied.

Let L be the loss function or cost function our neural network is trying to minimize. Given the input image I , the convolution filter G , the convolution output $R = I * G$, and the partial derivative of the error with respect to each scalar in the output, $\frac{\partial L}{\partial R[i,j]}$, **write an expression for the partial derivative of the loss with respect to a filter weight, $\frac{\partial L}{\partial G_c[x,y]}$, where $c \in \{r, g, b\}$.**

- (d) Sometimes, the output of a convolution can be large, and we might want to reduce the dimensions of the result. A common method to reduce the dimension of an image is called max pooling. This method works similar to convolution in that we have a filter that moves around the image, but instead of multiplying the filter with a subsection of the image, we take the maximum value in the subimage. Max pooling can also be thought of as downsampling the image but keeping the largest activations for each channel from the original input. To reduce the dimension of the output, we can do a strided max pooling in which we shift the max pooling filter by s positions instead of a single position, along the input. Given a filter size of $w \times h$, and a stride s , the output will be $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (H - h)/s \rfloor$ for an input image of size $W \times H$.

Let the output of a max pooling operation be an array R . **Write a simple expression for element $R[i, j]$ of the output.**

- (e) **Explain how we can use the backpropagation algorithm to compute gradients through the max pooling operation.** (A plain English answer will suffice; equations are optional.)

2 Convolutional Neural Networks

We will work with a Jupyter notebook in this Google Colab. This notebook will touch on many of the same topics discussed in the previous problem, but hopefully it will give a more concrete picture of what is happening in convolutional layers and what filters are being learned.