Numpy Pad: Understanding np.pad()

Posted 2021-01-30 • Last updated 2021-10-21

A common use case for padding tensors is adding zeros around the border of images to convert them to a shape that is amenable to the convolution operation without throwing away any pixel information. You can do this and much more in NumPy with the `np.pad()` function.

Basic usage

This function has a powerful API, but the basics are simple. Let's add one layer of zeros (the default) to a few arrays:

```
import numpy as np

x = np.ones(3)
y = np.pad(x, pad_width=1)
y

# Expected result
# array([0., 1., 1., 1., 0.])
```

Here we have a vector `x` with three 1's. We call `np.pad(x, pad_width=1)` which adds 0's to the beginning and end of the vector. Note: the pad operation works on a copy of the original array `x`.

Let's see how pad works for multiple dimensions. Next, we'll pad a matrix of 1's:

```
x = np.ones((3, 3))
y = np.pad(x, pad_width=1)
y

# Expected result
# array([[0., 0., 0., 0., 0.],
#        [0., 1., 1., 1., 0.],
#        [0., 1., 1., 1., 0.],
#        [0., 1., 1., 1., 0.],
#        [0., 0., 0., 0., 0.]])
```

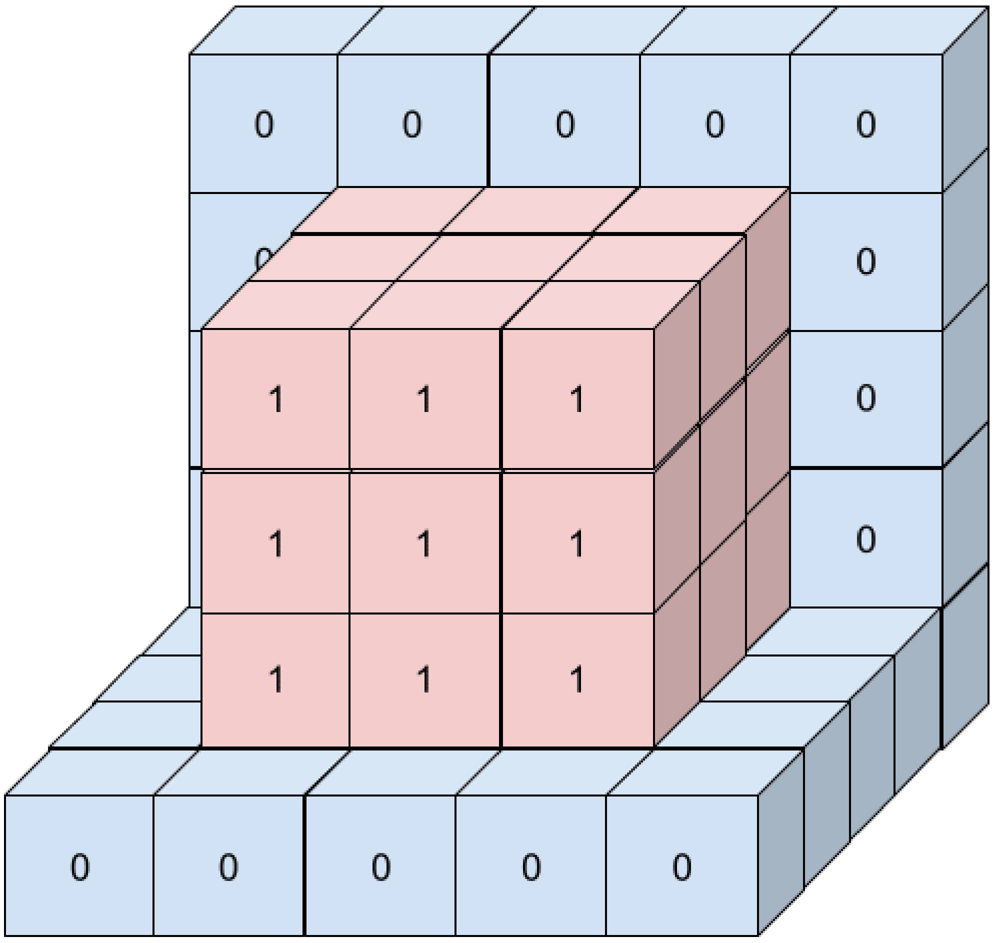
Notice the resulting array has shape `(5, 5)` because `pad_width=1` adds 0's on each side.

Finally, let's do it for a 3-dimensional tensor:

```
x = np.ones((3, 3, 3))
y = np.pad(x, pad_width=1)
y.shape

# Expected result
# (5, 5, 5)
```

As expected, the shape of the 3-dimensional tensor is `(5, 5, 5)`. At this number of dimensions, it's no longer easy to see the pattern of an array by printing it out. Instead, here's a diagram of the 3-dimensional padded array:



This is like ripping off four sides of 0's so you can see the 1's in the middle.

A more interesting padding example

You can do more interesting things with `np.pad()` as well. For one, `pad_width` accepts a sequence with form `((before_1, after_1), (before_2, after_2), ...)`. This allows you to apply different amounts of padding to different axes and sides.

For example, if we had wanted 0's on the top and left of the matrix, we could do that with the following:

```
x = np.ones((3, 3))
y = np.pad(x, ((1, 0), (1, 0)))
y

# Expected result
# array([[0., 0., 0., 0.],
#        [0., 1., 1., 1.],
#        [0., 1., 1., 1.],
#        [0., 1., 1., 1.]])
```

We can also apply different kinds of padding. We could pad with 2's by passing in `constant_values=2`:

```
x = np.ones((3, 3))
y = np.pad(x, ((1, 0), (1, 0)), constant_values=2)
y

# Expected result
# array([[2., 2., 2., 2.],
#        [2., 1., 1., 1.],
#        [2., 1., 1., 1.],
#        [2., 1., 1., 1.]])
```

And we can apply different modes of padding. My personal favorite is `mode='reflect'`. Let's add a reflection of all the pixels in a cat image on the right side:

```
import imageio
import matplotlib.pyplot as plt
```

```
img = imageio.imread("https://sparrow.dev/data/cyrus-chew.jpg")
height, width, channels = img.shape

plt.imshow(np.pad(img, ((0,0), (0, width), (0, 0)), mode='reflect'))
plt.xticks([])
plt.yticks([]);
```