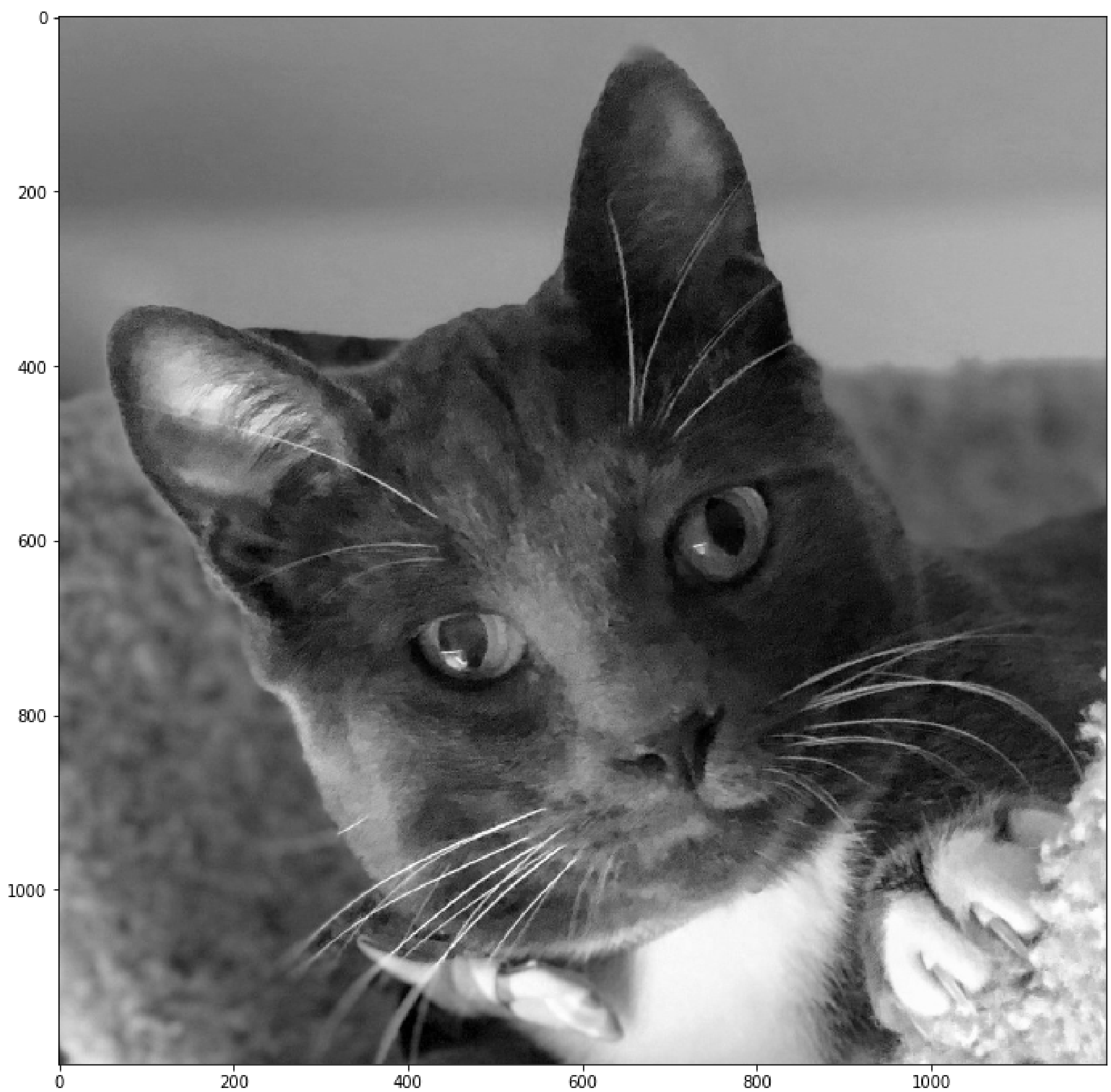
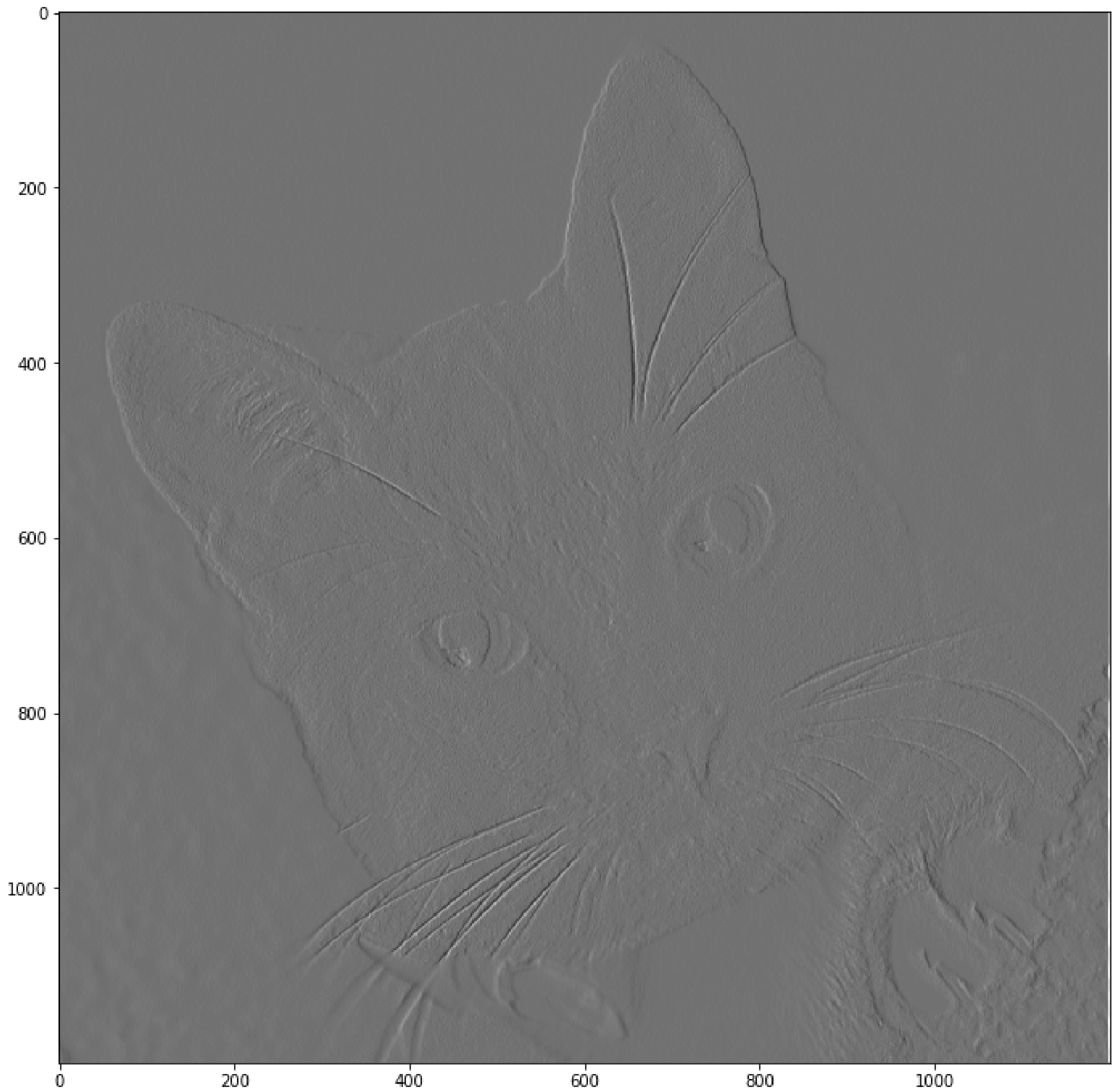
Feb_21

February 21, 2020

1 Goal

Our goal for today is to understand how we can do edge detection in grayscale images using "convolutions" (Actually we are using cross-correlations, not convolutions - but the neural network community uses the term convolution, so we'll use that term too. Convolutions are like backwards cross-correlations.):




0
200
400
600
800
1000
0
200
400
600
800
1000

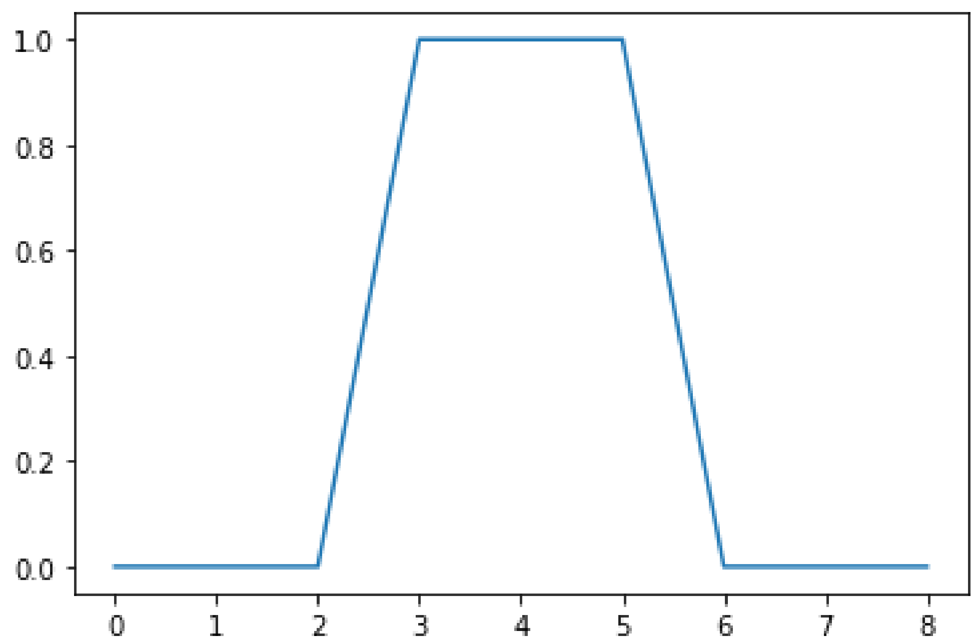
2 Simplified setting: 1 dimensional input

Suppose we have the following very fake data:

[53]:
```
x = np.concatenate(
    (np.zeros((3,)), np.ones((3,)), np.zeros((3,))),
    axis = 0
)
print("x = " + str(x))
plt.plot(x)
```

```
x = [0. 0. 0. 1. 1. 1. 0. 0. 0.]
```

[53]:
```
[<matplotlib.lines.Line2D at 0x7f3b106be4a8>]
```



We would like to detect the edges between indices 2 and 3, and between indices 5 and 6.

x = [0. 0. 0. 1. 1. 1. 0. 0. 0.]

2.1 (a) How to do this with a convolutional filter.

2.1.1 i. Filter width $f = 3$: W = [1, 0, -1]

2.1.2 ii. Filter width $f = 5$: W = [1, 1, 0, -1, -1]

2.2 (b) Suppose we have an input of length n and a filter of length f. What is the shape of the output?

2.3 (c) Suppose we *pad* the input by adding p 0's on the left and p zeros on the right.

2.3.1 i. Example calculation with $p = 1$, $f = 5$

x = [0. 0. 0. 1. 1. 1. 0. 0. 0.]

2.3.2 ii. What is the length of the output in terms of n, p, and f?

2.3.3 iii. What value of p should you use to get a "*same*" convolution where the length of the output is the same as the length of the input?

2.4 (d) Suppose we use a *stride* of s (the starting point of each new filter evaluation skips over s inputs).

2.4.1 i. Example calculation with $p = 1$, $f = 3$, $s = 2$

2.4.2 ii. What if our input was of length 8 instead of 9? (Suppose x = [0. 0. 0. 1. 1. 0. 0. 0.]; I deleted a 1 in the middle.)

2.4.3 iii. What is the length of the output in terms of n, p, f, and s?

3 Two-dimensional input

Recall that a greyscale image is represented as a 2-dimensional array of pixel values. Let's denote the shape by (n_H, n_W) (for height and width, corresponding to rows and columns).

Let's pretend that we have the following pixel values (these will be integers between 0 and 255, inclusive):

$$X = \begin{bmatrix} 100 & 100 & 100 & 0 & 0 & 0 \\ 100 & 100 & 100 & 0 & 0 & 0 \\ 100 & 100 & 100 & 0 & 0 & 0 \\ 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 \end{bmatrix}$$

We use the following filter:

$$W = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

3.1 (a) Find the filter output if we use a padding of p = 1 and stride of $s = 1$.

3.2 (b) What does this filter do?

3.3 (c) What is the shape of the output from applying an $f \times f$ filter to an $n_H \times n_W$ image using padding p and stride s?

4 Second Example with 2-dimensional input

Suppose

$$X = \begin{bmatrix} 100 & 100 & 100 & 0 & 0 & 0 \\ 100 & 100 & 100 & 0 & 0 & 0 \\ 100 & 100 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 100 & 100 \\ 0 & 0 & 0 & 100 & 100 & 100 \end{bmatrix}$$

We use the following filter:

$$W = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

4.1 (b) What does this filter do?

4.2 (c) What would the output shape be if instead you used a padding of $p = 1$ and a stride of $s = 2$?

5 Algorithm to compute

We will make some tweaks to this next week to allow for multiple observations (multiple images) and color images with red, green, and blue channels.

Inputs:

- Array A of shape (n_H, n_W) to apply the filter to. (In examples above, we used X – this is what we would use in the first layer of a convolutional network, later layers will use activations A from previous layers)
- Filter of shape (f, f)
- Padding amount p
- Stride amount s

Outputs:

- Filtered inputs of shape $\left(\left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor, \left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor \right)$

Algorithm:

- Pad image with p pixels of 0s on all sides.
- Create output array of shape $\left(\left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor, \left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor \right)$
- For i = 0, $\ldots$, $\left\lfloor \frac{n_H+2p-f}{s} + 1 \right\rfloor$
  - For j = 0, $\ldots$, $\left\lfloor \frac{n_W+2p-f}{s} + 1 \right\rfloor$
    * start_row = i * s, end_row = start_row + f
    * start_col = j * s, end_col = start_col + f
    * output[i, j] = np.sum(W * A[start_row:end_row, start_col:end_col])