Intro to Image Understanding

February 19, 2020

1. Write your own code for computing convolution of the 2d (grayscale) image and a 2D filter. Make the output matrix be the same size as the input image.[1]

[1] Be cereful to correctly deal with th border of the image- the easiest way to do this is to "zero-pad" the image prior to convolution.

```
def conv_fast(image, kernel):
  """
  Function for 2D convolution using numpy
  """
    Hi, Wi = image.shape
    Hk, Wk = kernel.shape
    out = np.zeros((Hi, Wi))

    #flipping the kerne in x axis
    kernel = np.flip(kernel,axis=0);

    #padding the image
    padded= zero_pad(image, Hk//2, Wk//2)

    #loop on element
    for i in range(Hi):
        for j in range(Wi):
            out[i,j]=np.sum(padded[i:i+Hk,j:j +Wk]* kernel)

    return out
```

- Extend this code to handle *RGB* images. [2]

[2] To avoid touching the 1 channel convolution, we will write a wrapper to handle both situations.

```
def conv_fast(image, kernel):

def conv(Img, kernel):
  """
  Generalized version to compute convolution
  with Img and kernel. The iage could be 3 channel
  """

  #getting the number of dimensions
  if Img.ndim == 2:
    #gray scale image
    return conv_fast(image, kernel)
  else:
    #filtering in each channel
    F = np.zeros_like(Img):

    #loop over each channel
    for c in range(2):
      F[:,:,c] = conv_fast(Img[:,:,c])
    return F
```

- You convolve your image I with a $2d$ kernel $K_1 = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, then take the output and convolve it with $K_2 = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$.

  It is possible to get the same result with a single convolution? Yes theoretically, convolution product is **associative**. Hence

$$(I * K_1) * K_2 = I * (K_1 * K_2) \tag{1}$$

The kernel $K_3 = K_1 * K_2$ has a dimension $(3,3)$ and its given by[3]

[3] Convolution of the images (n_1, m_2) and (n_2, m_2) has a size of: $(n_1 + n_2 - 1, m_1 + m_2 - 1)$.

$$K_3 = \begin{pmatrix} dc & ac + db & ad \\ bc + de & ac + bd + ec + fd & bd + df \\ be & ae + fd & cf \end{pmatrix} \tag{2}$$

- Write your own function that create an isotropic Gaussian filter with σ and an input parameter.

  First we create a function to create the **Gaussian kernel** with standard deviation σ.

```
def Gaussian_kernel_2d(sigma=1):
    """
    Gaussian kernel in the 2d case
    """
    #number of points after 3 stds
    N = int(3*sigma)

    #distances
    d = np.r_[ np.arange(-N,0), np.arange(0,N+1)]
    n = len(d)

    #diff in X
    X = np.tile(d, (n,1))
    Y = X.transpose()

    cst = 1/(2 * np.pi * sigma**2)

    Z = cst * np.exp(-(X**2 + Y**2)/(2*sigma**2))

    return Z/ np.sum(Z)
```

And now, we simply use the convolution function with the Gaussian kernel

```
def Gaussian_filter(Img, sigma=1):
    """
    Gaussian filter with given sigma
    """

    #getting the kernel
    kernel = Gaussian_kernel_2d(sigma)


    #computing the convolution
    return conv_fast(Img, kernel)
```

- Use a Gaussian filter and produce the results of convolving a Gaussian filter with the *Waldo* image.

Figure 1: Results of the convolution with $\sigma = 1$

Is the vertical derivative, $\dfrac{\partial G(x,y)}{\partial y}$ of a Gaussian filter a separable filter?

Yes, as the original filter is seprable.

$$
\begin{aligned}
G(x,y) &= \frac{1}{2\pi\sigma^2}\, e^{-\dfrac{x^2+y^2}{2\sigma^2}} && (3)\\
&= \Big(\frac{1}{\sqrt{2\pi\sigma}}\, e^{\dfrac{-x^2}{2\sigma^2}}\Big)\Big(\frac{1}{\sqrt{2\pi\sigma}}\, e^{\dfrac{-y^2}{2\sigma^2}}\Big) && (4)\\
&= G_x G_y && (5)\\
\frac{\partial G(x,y)}{\partial y} &= G_x \partial_y (G_y) && (6)
\end{aligned}
$$

- What is the number of operations for performing a 2D convolution?

- What is the number of operations in the case of *separable* filter?

The number of operation[4], the number of quadratic in term of the images size and kernel size

[4] assuming a simple convolution not convolution by fourier transform

$$C(\text{Conv}) = \mathcal{O}(NMkl) \tag{7}$$

For a **separable filter** the complexity is :

$$C(\text{Conv}_{sep} = \mathcal{O}(NMk) + \mathcal{O}(NMl) \tag{8}$$

Figure 2: Waldo Edge Strenght map

Figure 3: Template edge strenght map

- Compute the **gradient magnitude** for the `waldo` image and the `template`?
- Write a function to localize the template (`template.png`) using the Edge Strenght Map?

As a first step we write the *Normalized Cross Corelation* routine to perform the correlation with the normalized patchs.

```
"""
Function to performe the corss corelation to find waldo
"""

#IMG
Img = imread("./waldo_ESM.png")
Img = rescale(Img, 0.5)
#template
patch = imread("./template_ESM.png")

#position
x, y = np.unravel(np.argmax(scors), scores.shape)

scores = np.abs(correlate(Img, patch))
# plt.imshow(scores, cmap = plt.cm.gray)
imsave("./matching.png", scores)
```