Electronics and Computer Science
Faculty of Physical and Applied Sciences
University of Southampton

Brian Formento

2 November 2018

COMP6223: Image Filtering and Hybrid images

Image Processing

Image processing In computer science is the field that uses algorithms to performs operations on an image.

Convolution

Convolution is the mathematical means to perform such operations. The most basic being brute force convolution. This operation involves convolving a kernel of size $i * j$ on an image of size $M * N$. Each pixel will be the weighted sum combination of closer pixels.

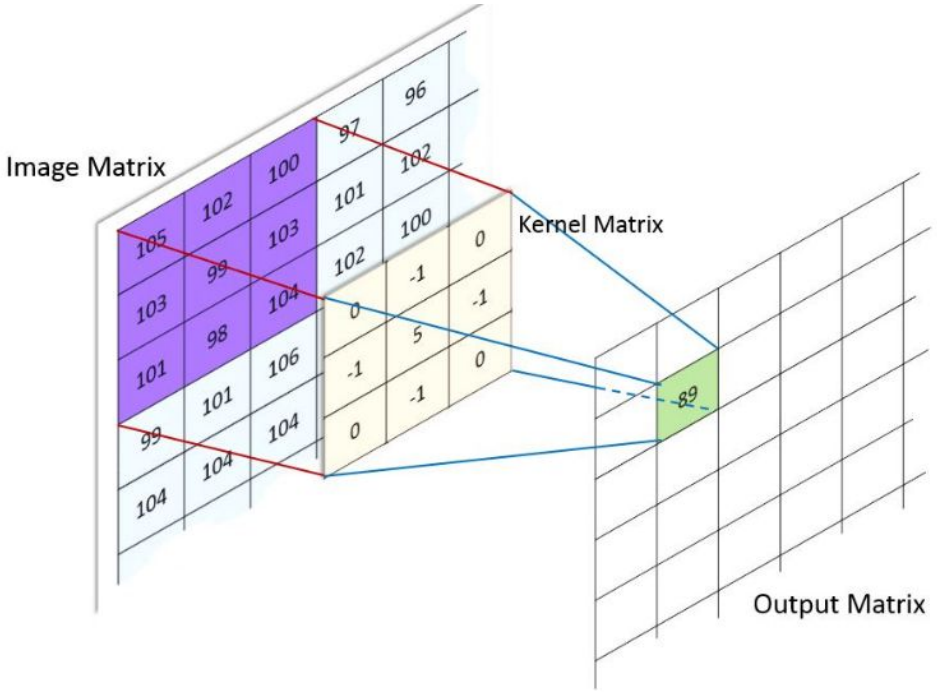

Fig1: The convolution operation, note how the result's element at 0,0 has been removed. This is because the convolution operation shrinks the image. [1]

The complexity of convolution is $O(MNmn)$, making it computationally expensive for large images. Because of this, and because of the shrinking effect this project makes use of FFT convolution instead.

Convolution FFT

This uses the Frequency domain of an image to perform image processing operations. By turning both the kernel and image in the frequency domain using the fast Fourier transform algorithm it is possible to just multiplicate both matrices together. This is due to the theorem:

"Convolution in the spatial domain is equivalent to multiplication in the frequency domain" [1]

$$g * h = F^{-1}[F[g]F[h]] \text{ (eq 1)}$$

Because of this, it has a lower complexity of $O(Nlog(N))$. Where N is the number of pixel width and height in a square image

```python
# Return the Component using FFT convolution
def Convolution_FFT(Img, Gaussian):
    # Create an empty image that will later be used to store the result
    Filtered = np.zeros(Img.shape)

    # Take the image, apply the fast Fourier transform,
    def FFT(Component, Gaussian):
        """applies the FFT to one of the 3 RGB channels, placing us in the
        frequency domain and shifts the 0 frequency component back in the middle"""
        C_fft = np.fft.fftshift(np.fft.fft2(Component))
        # Matrix multiplication
        Result = C_fft*Gaussian
        # inversing everithing allows to go back to the spatial.
        ifft = np.fft.ifft2(np.fft.ifftshift(Result))
        return ifft
    # check to see if the shape has more than one channel, if not its a grayscale.
    if len(Img.shape) == 2:
        Filtered[:,:] = FFT(Img[:,:], Gaussian)
    else:
        """Do the FFT function and convolution for every RGB channel and fill each
        channel in Filtered"""
        for i in range(Img.shape[-1]):
            Filtered[:,:,i] = FFT(Img[:,:,i], Gaussian)
    return Filtered

# Add the low and high frequency component together producing the final hybrid image
def Merge(low, high):
    low = low + high
    return low
```

Fig2: FFT convolution code

Hybrid Algorithm

By taking the high-frequency component of one image and adding it to the low-frequency component of another it is possible to create a hybrid image.

To build such representation firstly the Gaussians have to be generated, as they are needed to extract the high and low-frequency components. As it can be seen from Fig3 and Fig4 the high-frequency Gaussian kernel is the inverse of the low.

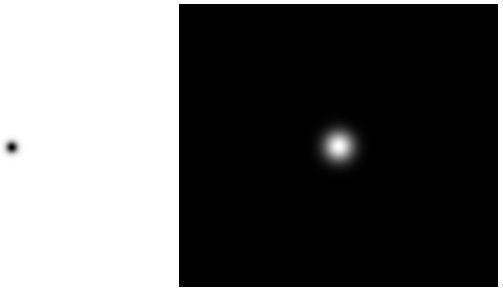

Fig3 left: is the high-frequency component, while Fig4 right: is the low-frequency component.

```python
# Creation of Gaussians
def Make_Gaussian(Img, Low_pass_filter, S):
    # Take the size of the input image
    Rows =  Img.shape[0]
    Col = Img.shape[1]
    # Find the centre of the Image, if it's a decimal increment by one
    I_center = int(Rows/2) + 1 if Rows % 2 == 1 else int(Rows/2)
    J_center = int(Col/2) + 1 if Col % 2 == 1 else int(Col/2)
    # Create an empty 0 image to store the gaussian
    K = np.zeros((Rows, Col))

    """ Iterate through every col and row, at each stage when calculating the Gaussian
     value at I and j take away the value of the centre. This will allow the forming of
     the highest concentration when i = I_center and j = J_center being both 0 the exp
     part will equal to one. This is the highest point for the low-frequency template"""
    for i in range(Rows):
        for j in range(Col):
            c = math.exp(-(((i - I_center)**2+(j - J_center)**2)/(2*S**2)))
            if Low_pass_filter:
                K[i][j] = c
            else:
                # if we are working on the high frequency part we can invert the Gaussian
                K[i][j] = (1-c)
    return K
```

Fig5: Gaussian generation code

All the channels in each image can be transformed in their frequency domain equivalent E.g (Fig6) where the gaussian multiplication can then be performed as shown in the FFT function in (Fig2). This will cut out the high (middle part of the image) or low (edges of the image) frequency components depending on the image.

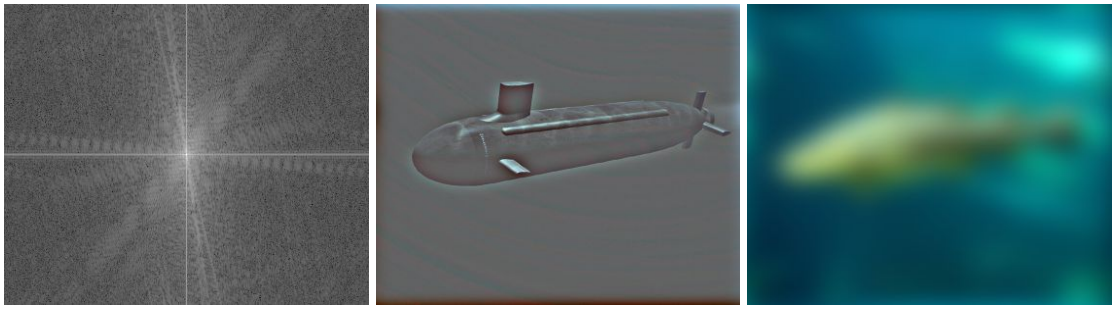

Fig6: The blue component of the submarine in the frequency domain (left). Fig7: Isolating the high frequency highlights the edges (middle). Fig8: For the low frequency (right) a blurry image is to be expected.

The hybrid image is constructed by adding the high and low-frequency images together.

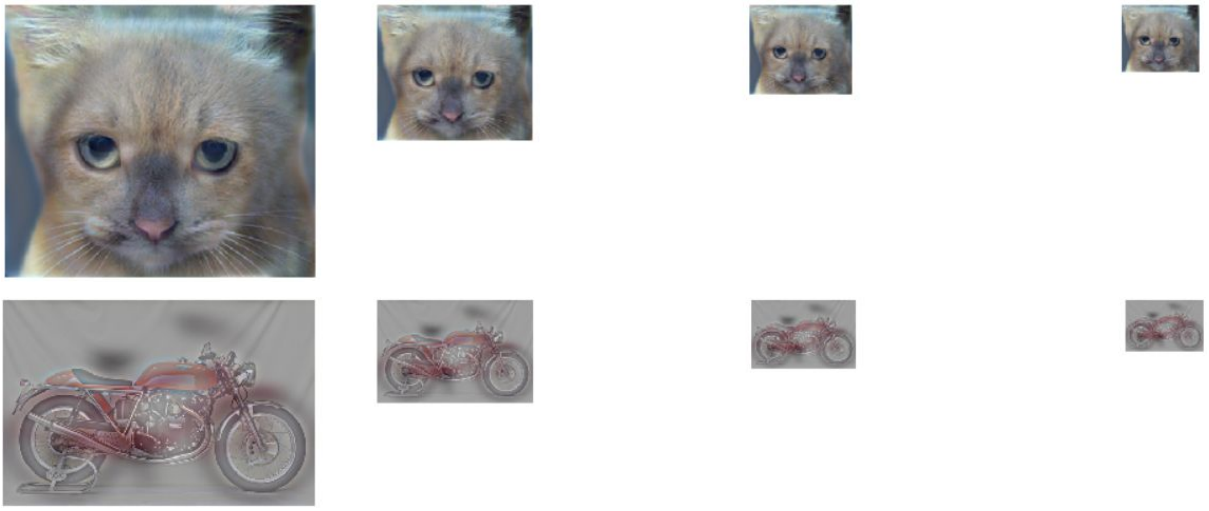

Fig9: The image to the left are clearly a submarine,cat and motorcycle, while the images to the right are a fish, dog and bicycle.

It is also possible to see how changing sigma influences the picture. It is clear that a low sigma blurs the fish image, while a high sigma strengthens the contour of the submarine.

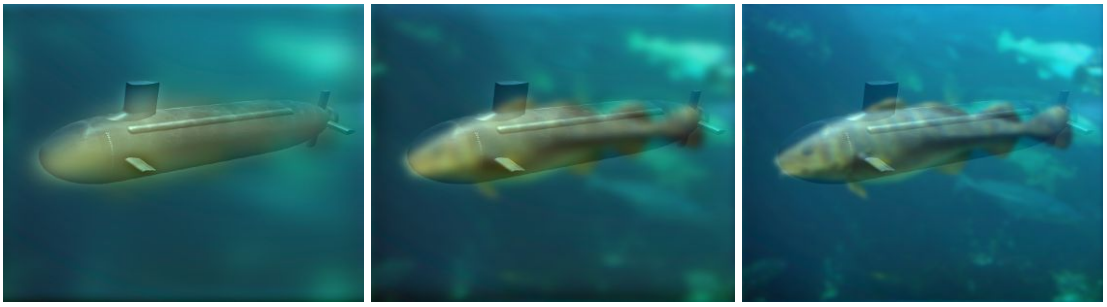

Fig10: For Sigma = S, fish and submarine respectively. Left) S = 4, S = 8. Middle) S = 12, S = 4, Right) S = 25, S = 1.

References

[1] Idar Dyrdal, Unik4690, lecture 2.1 Image processing and image filtering, University of Oslo
[2] Aude Oliva, Antonio Torralba,Philippe. G. Schyns, 2006, Hybrid Images, Siggraph