Computer Vision HW2 Group 16

0856622 余家宏、309551067 吳子涵、309551122 曹芳驊

Introduction

There are three tasks in this assignment. In task 1, we need to use both high pass filter and low pass filter to generate a hybrid image with input images pairs. In task 2, we need to generate image pyramids for input images. In task 3, we need to align the three separate channels of image and generate one single RGB image.

Task 1: Hybrid Image

Hybrid images combine the low spatial frequencies of one picture with the high spatial frequencies of another picture, producing an image with an interpretation that changes with the viewing distance.

Task 2: Image Pyramid

Image pyramid is a type of multi-scales image representation developed by computer vision and image processing, in which an image is subject to repeated subsampling and upsampling with gaussian smoothing beforehand.

Task 3: Colorizing the Russian Empire

Colorizing the Russian Empire is to produce a color image from the digitized Prokudin-Gorskii glass plate images with as minimal visual artifacts as possible. In order to do this, extract the three color channels from the glass plate, then place and align one above the other so that the combination forms a single RGB color image.

Implementation Procedure

Task 1: Hybrid Image

- Step 1: Image read in
    - Use OpenCV built-in function to read in the image file and check if the size of two images is equal.
    - If not, upscale the smaller image to match the large one with our handcraft upscale function (using nearest-neighbor interpolation).
- Step 2: Calculate the Fourier transform
    - Use NumPy FFT module to calculate Fourier transform and shift the spectrum
- Step 3: Create filter masks for each FT of images
    - Calculate center coordinate of the spectrum as the origin

```
o_x = int(rows/2) if rows%2 == 0 else int(rows/2) + 1
o_y = int(cols/2) if cols%2 == 0 else int(cols/2) + 1
```

    - Calculate the mask coefficient of each pixel (with gaussian or ideal filter)

D_0 : $cutoff\ frequency$

$u,\ v$: $coordinate\ of\ pixel$

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

$$H(u,v) = \begin{cases} 1 & if\ D(u,v) \le D_0 \\ 0 & if\ D(u,v) > D_0 \end{cases}$$

```
def gaussian(x,y):
    coef = math.exp(-1.0 * ((x - o_x)**2+(y - o_y)**2)/(2 * sigma ** 2))
    return coef if islowpass else 1 - coef
def ideal(x,y):
    near_center = True if math.sqrt((x - o_x)**2+(y - o_y)**2) <= sigma else False
    return 1 if (islowpass and near_center) or (not islowpass and not near_center) else 0
```

    - Aggregate as a mask

```
func = gaussian
return np.array([[func(i,j)for j in range(cols)] for i in range(rows)] )
```

- Step 4: Create high-pass and low-pass image and hybrid
    - Multiply FT of images and the corresponding mask then apply inverse FFT
    - Take the real part of results as the filtered image
    - Hybrid high-pass and low-pass images (add up two images)

Task 2: Image Pyramid

- Step 1: Read input image
- Step 2: Make gaussian pyramid (total 5 layers)
    - Append the original image as the lowest layer
    - Apply gaussian blurring and subsample to it with a certain factor
    - Loop through the above process 4 times

```
for i in range(4):
    lowpass = transform(pyramid[-1], islowpass = True, sigma = 10)
    sub = lowpass[::factor,::factor].copy()
    pyramid.append(sub)
```

    - Store the magnitude spectrum for each layer

```
DFT = fftshift(fft2(sub))
spectrums.append(20 * np.log(abs(DFT)))
```

- Step 3: Construct laplacian pyramid with above gaussian pyramid
    - Append the highest layer of the gaussian pyramid to the same layer of the laplacian pyramid.
    - From the highest layer, upsample it with nearest-neighbor interpolation, and apply the same smooth filter when constructing the gaussian pyramid.
    - Obtain the next layer by subtracting the next layer of gaussian's with the above result.
    - In the subsampling process, there might be some odd number dimensions : (100, 75) -> (50, 38). For these conditions, we reduce one dimension of the

upsample result : (100, 76) -> (100, 75) to retain the dimension consistency when upsampling.

- Loop through the above processes 4 times

```
for Gi, Gi_1 in zip(g_Pyr[-2::-1], g_Pyr[-1:0:-1]):
    Gi_1_up = upsample(Gi_1,factor)
    if Gi_1_up.shape[0] > Gi.shape[0]:
        Gi_1_up = Gi_1_up[:-1]
    if Gi_1_up.shape[1] > Gi.shape[1]:
        Gi_1_up =  Gi_1_up[:,:-1]
    up = Gi - transform(Gi_1_up, islowpass=True, sigma = 10)
    pyramid.append(up)
```

```
def upsample(img, factor):
    rows, cols = img.shape
    resize_img = np.zeros((rows*factor, cols*factor))
    for i in range(resize_img.shape[0]-1):
        for j in range(resize_img.shape[1]-1):
            org_x = round(i * (1/factor))
            org_y = round(j * (1/factor))
            resize_img[i,j] = img[org_x, org_y]
    return resize_img
```

Task 3: Colorizing the Russian Empire

- Step1: Feed-in image
  - Read the JPG file (roughly 170KB) and TIFF file (roughly 70MB) sequentially.
- Step2: Crop the input image
  - We crop 3% of the image from top and bottom, and 6% from left and right to remove border artifacts that would interface with alignment matching.

original image

cropped image

- Step3: Split image
    - We split the whole grayscale image into three equal pieces, which represent blue, green, red channels respectively.

blue	green	red
		

```python
def crop_img(img, top=0.03, down=0.03, left=0.06, right=0.06):
    h, w = img.shape
    img = img[int(h * top) : int(h - h * down), int(w * left) : int(w - w * right)]
    return img

def split_img(img):
    h, w = img.shape
    blue = img[0 : int(h/3), :]
    green = img[int(h/3): int(h/3) * 2, :]
    red = img[int(h/3) * 2: int(h/3) * 3, :]

    return blue, green, red
```

- Step4: Get the edges of image using sobel filter

blue	green	red
		

```python
def sobel(img):
    sobel_x = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    sobel_y = sobel_x.T

    img_x = conv(img, sobel_x)
    img_y = conv(img, sobel_y)
    img_x = np.uint8(np.absolute(img_x))
    img_y = np.uint8(np.absolute(img_y))
    filter_img = cv2.bitwise_or(img_x, img_y)

    return filter_img

def conv(A, B):
    return np.real(ifft2(fft2(A)*fft2(B, s=A.shape)))
```

- Step5: Image Pyramid
    - In order to speed up the alignment process, we perform offset searching on the image pyramid.
    - The offset search can start from the coarsest scale and gradually work down to the largest scale to reduce the necessary search space.
    - We use a larger search window to search the displacement in the small image, and a smaller search window to search in the large image.
    - This procedure can reduce execution time because we don't need to use a large search window on the highest resolution.

```python
### pyramid ###
crop_height = blue.shape[0]
crop_width = blue.shape[1]
ratio = init_ratio(crop_width)

ratio_between_layer = 2
offset = int((height // ratio) // 5)

total_offset_g2b = np.zeros([2])
total_offset_r2b = np.zeros([2])

filter_b_ = conv(filter_b, gaussian_kernel(1))
filter_g_ = conv(filter_g, gaussian_kernel(1))
filter_r_ = conv(filter_r, gaussian_kernel(1))

while(ratio >= 1):    ### small to large ###
    ### down sampling ###
    down_b = filter_b_[::ratio, ::ratio]
    down_g = filter_g_[::ratio, ::ratio]
    down_r = filter_r_[::ratio, ::ratio]

    ### align ###
    offset_g2b = alignment(down_b, down_g, offset, zncc=0)
    offset_r2b = alignment(down_b, down_r, offset, zncc=0)

    ### rolling ###
    offset_g2b = [element * ratio for element in offset_g2b]
    offset_r2b = [element * ratio for element in offset_r2b]

    filter_g_ = np.roll(filter_g_, offset_g2b, axis=(0,1))
    filter_r_ = np.roll(filter_r_, offset_r2b, axis=(0,1))

    total_offset_g2b = [a + b for a, b in zip(total_offset_g2b, offset_g2b)]
    total_offset_r2b = [a + b for a, b in zip(total_offset_r2b, offset_r2b)]

    ratio = int(ratio / ratio_between_layer)
    offset = int(offset / ratio_between_layer)

### final rolling ###
total_offset_g2b = [int(a) for a in total_offset_g2b]
total_offset_r2b = [int(a) for a in total_offset_r2b]
```

- Step6: Alignment
    - Searching the best (x, y) displacement vectors within a specified region to overlay the green and red channels over the blue channel.
    - To determine the offset, we use a metric named SSD to measure the similarity between two shifted images and choose the largest response as the best offset.
    - SSD (Sum of Squared Differences):

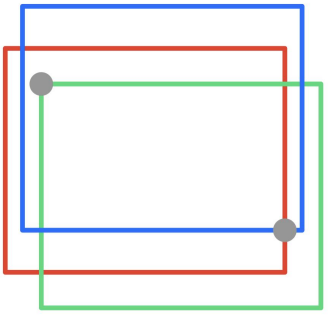
$$ssd(u,v) = \sum_{(x,y)\in N} \left[I(u+x, v+y) - P(x,y) \right]^2$$

original blue	original green	original red
		
shifted blue	shifted green	shifted red
		

```python
def align_SSD(img1, img2, offset):

    max_response = float("inf")
    for i in range(-offset, offset):
        for j in range(-offset, offset):
            ncc_response = SSD(img1, np.roll(img2, [i, j], axis=(0, 1)))
            if ncc_response < max_response:
                max_response = ncc_response
                shift_pos = [i, j]

    return shift_pos

def SSD(img_1, img_2):
    ssd = np.sum((img_1 - img_2) ** 2)
    return ssd
```

- Step7: Combine images of the three channels to form the RGB image.

- Step8: Crop the image to remove the rolling effect
    - Since the aligning process will create the useless borders, it's better to crop the image to remove the rolling effect.
    - We only keep the union area of the three images, that is the area between two gray dots in the below image.



```
### form RGB image ###
rgb_img = (np.dstack((shift_red, shift_green, blue))).astype(np.uint8)

top_left_x = max(total_offset_g2b[0], total_offset_r2b[0], 0)
top_left_y = max(total_offset_g2b[1], total_offset_r2b[1], 0)

bottom_right_x = min(total_offset_g2b[0] + crop_height, total_offset_r2b[0] + crop_height, crop_height)
bottom_right_y = min(total_offset_g2b[1] + crop_width, total_offset_r2b[1] + crop_width, crop_width)
cropped_rgb_img = rgb_img[top_left_x:bottom_right_x, top_left_y:bottom_right_y, :]
rgb_img = Image.fromarray(cropped_rgb_img)
```

- Step9: Final result

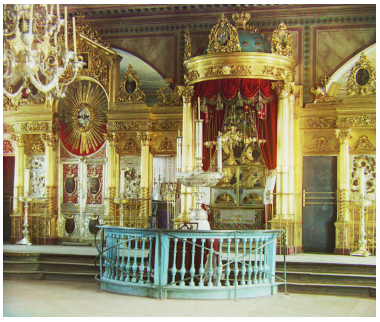
Task 3: Colorizing the Russian Empire







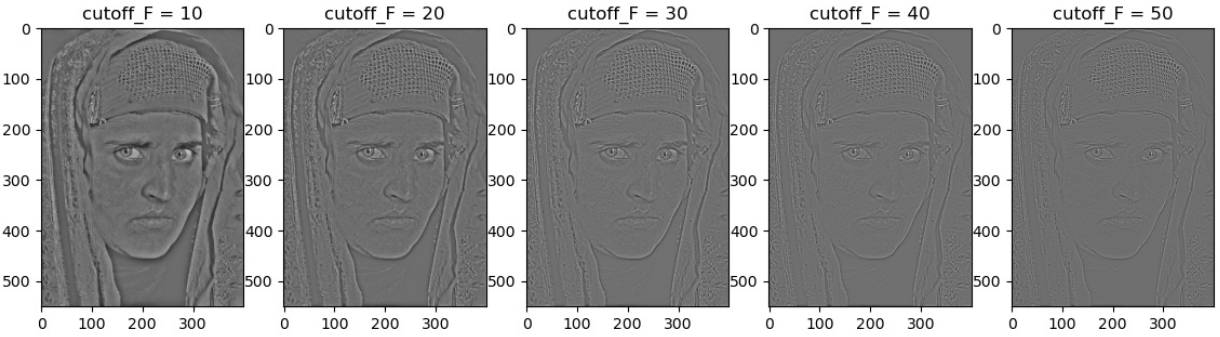


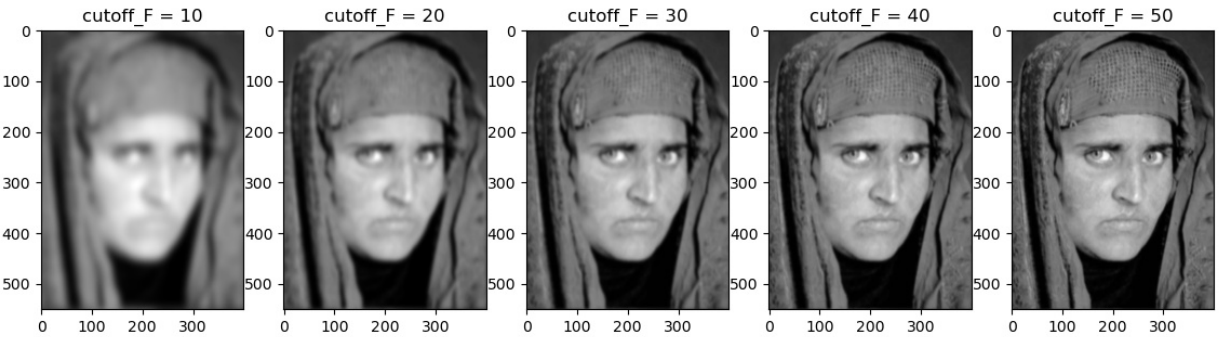



Discussion

Task 1: Hybrid Image

- In the experiment, we discovered the cutoff frequency of the filter seriously influences the result. For the low-pass image, the higher the value, the more detail being retained. And for the high-pass one, the higher value means it retains less information and only keeps the outline of image content.





- In order to hybridize an image of visual illusion, we experiment against many combinations and discover that. We need to use enough high cutoff frequency for high-pass images to only extract the contour of the image and use enough low cutoff frequency for low-pass images to make sure a certain degree of background blurring.

Task 2: Image Pyramid

- For the magnitude spectrum, we first try directly storing the FT result and show it as an image, but the result is non-sense for visual observation.

After some surveys, we learn that it's needed to transform the FT result to the log-scale for observation.

Task 3: Colorizing the Russian Empire

- Using the intensity or the edges of the images to calculate similarity.
    - We first use the intensity of R, G, B images to calculate similarity, and then align them by the best displacement which has the highest similarity score.
    - We find that it can get good displacement value so that it can form a well-aligned colored image when the relative intensity distribution between R, G, B images is similar, such as nativity.jpg and lady.tif.
    - However, the relative intensity distribution between R, G, B of some images is different, such as emir.tif. The intensity of the cloth is very high in the blue channel but very low in the red channel. Thus, while computing the similarity, the similarity score would be very low even if the three channels are exactly aligned by humans.

nativity.jpg	lady.tif	emir.tif
		



- Therefore, we use edge information to calculate similarity alternatively.
- The images below are the result of calculating similarity by edge information using Sobel filter.



- Using SSD or ZNCC as the metric to measure the similarity of two shifted images.
  - SSD (Sum of Squared Differences):

$$ssd(u,v) = \sum_{(x,y)\in N} \left[I(u+x, v+y) - P(x,y)\right]^2$$

  - ZNCC (Zero-mean Normalized Cross-Correlation): simply a dot product between two normalized vectors.

$$ZNCC = \frac{\sum_{i=1}^{MN} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{MN} (x_i - \bar{x})^2 \sum_{i=1}^{MN} (y_i - \bar{y})^2}}$$

  - The [xxx, xxx] behind image name means the displacement vectors along x-direction and y-direction.
  - The second line means the processing time.

SSD

```
monastery.jpg [-23, 2] [-37, 2]
--- 0.22574782371520996 seconds ---
tobolsk.jpg [-17, 2] [-34, 3]
--- 0.20581316947937012 seconds ---
nativity.jpg [-17, 1] [-32, 0]
--- 0.1696481704711914 seconds ---
cathedral.jpg [-15, 2] [-28, 3]
--- 0.1374361515045166 seconds ---
workshop.tif [-140, -1] [-282, -12]
--- 11.506608724594116 seconds ---
emir.tif [-144, 23] [-279, 40]
--- 9.197450876235962 seconds ---
three_generations.tif [-137, 12] [-273, 9]
--- 11.121381044387817 seconds ---
melons.tif [-114, 10] [-211, 13]
--- 13.196914911270142 seconds ---
onion_church.tif [-141, 25] [-278, 35]
--- 9.75963306427002 seconds ---
train.tif [-153, 0] [-303, 29]
--- 11.553662061691284 seconds ---
icon.tif [-153, 17] [-300, 23]
--- 14.261843204498291 seconds ---
village.tif [-132, 11] [-255, 22]
--- 14.632517099380493 seconds ---
lady.tif [-137, 9] [-266, 13]
--- 12.035430908203125 seconds ---
```

ZNCC

```
monastery.jpg [-23, 2] [-37, 2]
--- 0.3758511543273926 seconds ---
tobolsk.jpg [-17, 2] [-34, 3]
--- 0.4294860363006592 seconds ---
nativity.jpg [-17, 1] [-32, 0]
--- 0.3996551036834717 seconds ---
cathedral.jpg [-15, 2] [-28, 3]
--- 0.3835489749908447 seconds ---
workshop.tif [-140, -1] [-282, -12]
--- 24.37813711166382 seconds ---
emir.tif [-144, 23] [-279, 40]
--- 20.979079246520996 seconds ---
three_generations.tif [-137, 12] [-273, 9]
--- 21.328917026519775 seconds ---
melons.tif [-114, 10] [-211, 13]
--- 22.241860151290894 seconds ---
onion_church.tif [-141, 25] [-278, 35]
--- 21.25407886505127 seconds ---
train.tif [-151, 8] [-303, 29]
--- 22.034119129180908 seconds ---
icon.tif [-153, 17] [-300, 23]
--- 22.508987188339233 seconds ---
village.tif [-132, 11] [-255, 22]
--- 24.172168970108032 seconds ---
lady.tif [-137, 9] [-266, 13]
--- 23.445392847061157 seconds ---
```

- From the above data, we find that using SSD to perform the process is much faster than using ZNCC, especially for large images.
- The displacement vectors calculated by SSD and ZNCC are nearly the same, only a few images have different results, such as train.tif.

SSD	ZNCC
	

- We choose SSD as our metric but not ZNCC because SSD can provide comparable results with less execution time.

- Our test images:

Conclusion

Task 1: Hybrid Image

- In this task, we learned how to implement high-pass and low-pass filters in frequency domain, and how the cutoff frequency influences the visual observation of filtered images. And the hybrid image technique can be apply on many computer vision analysis and image processing application (eg: watermark production)

Task 2: Image Pyramid

- In the second task, we construct the gaussian image pyramid by repeatedly blurring and subsampling the input image and then use the results to construct the laplacian pyramid by inverse upsampling.
  By the experiment result we can observe that even though we use the nearest interpolation to do upsampling (which is not optimal), with the gaussian pyramid, the reconstruction of the laplacian pyramid can still be lossless.

Task 3: Colorizing the Russian Empire

- The key in this task is to well align the R, G, B intensity images efficiently. First, the edge information is more helpful to compute the similarity between two images than the intensity information. Second, it can reduce calculation time that we find the rough displacement from the smaller image first, and then compute the precise offset from the larger image.

Work assignment plan between team members

- Coding: 吳子涵、曹芳驊
- Experiment: **余家宏**、吳子涵、曹芳驊
- Report: **余家宏**、吳子涵、曹芳驊