Image Processing HW_4

Student -1- : Mostufa Jbareen (212955587) מוסטפא גבארין

Student -2- : Mohammed Egbaria (318710761) מוחמד אגבאריה

Problem 1:

Section a):

In most cases, the first method using geometric operations to scale pixel coordinates with interpolation is preferred over changing the Fourier transform of the image and then inverse transforming it back to get the scaled image. There are several reasons for this:

Robustness: Geometric operations with interpolation are more versatile and handle various image types and scaling factors better than Fourier transform methods.

Artifact Introduction: Interpolation methods reduce the likelihood of introducing artifacts compared to Fourier transform methods, which may cause image degradation.

Accuracy and Preservation of Details: Geometric operations with interpolation yield more accurate results and preserve more image details.

Computational Complexity: Geometric operations with interpolation are less computationally intensive than Fourier transform methods, which involve multiple transformations.

Section b):

The Fourier transform provides a unique representation of the frequency content of an image or function. This uniqueness is derived from the properties of the Fourier transform:

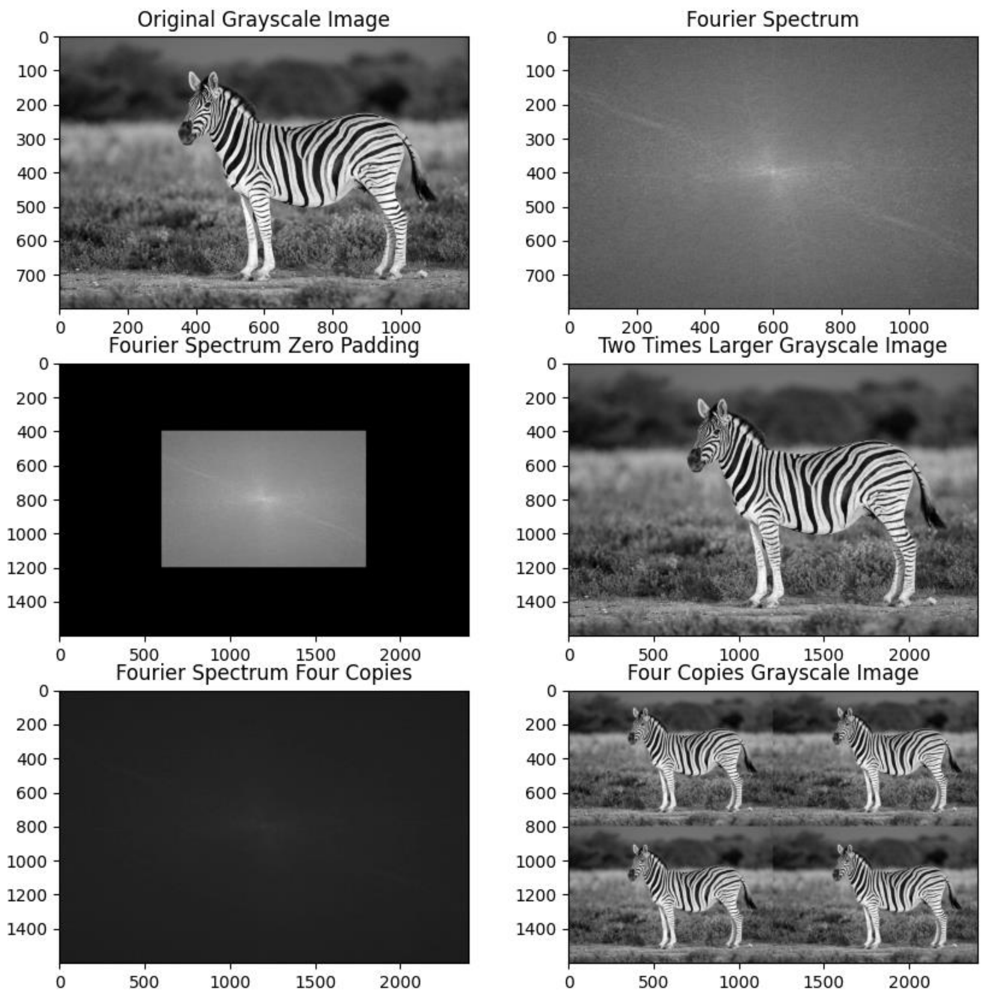
Orthogonality of Basis Functions: The Fourier transform decomposes a function into a sum of sinusoids of different frequencies, tthese sinusoids form an orthogonal basis in the frequency domain. Since different functions have different frequency components, they will have different coefficients in this basis, resulting in different Fourier transforms.

Linearity: The Fourier transform is a linear operation. If two functions are different, their linear combinations will also be different, and thus their Fourier transforms will also be different.

These properties ensure that if two functions F(x, y) and G(x, y), have the same Fourier transform, it implies that they have the same frequency content, so they must be identical f(x, y) = g(x, y).

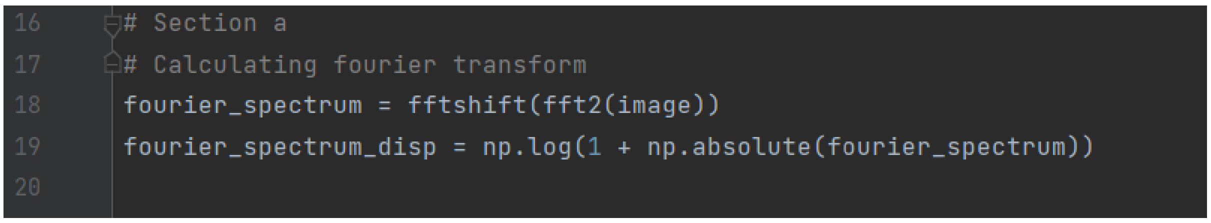
Problem 2:

The results (zebra_scaled):



Section a):

Simply Calculating Fourier Transform and preparing the log(1+cofficients) to display, you can see the display of the Fourier transform in the image in the first row second image.

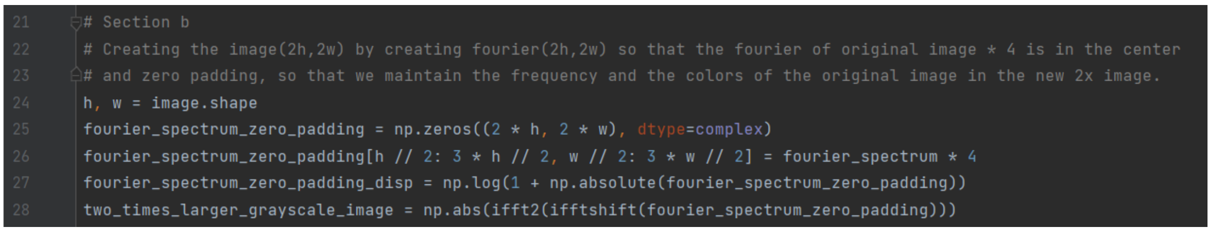


```
# Section a
# Calculating fourier transform
fourier_spectrum = fftshift(fft2(image))
fourier_spectrum_disp = np.log(1 + np.absolute(fourier_spectrum))

```

Section b):

As we learned in Tutorial 6, in order to upscale the image by factor 4, we use zerro padding to the new shape of the upscaled image (2h,2w) (h and w of the original image), and then centering the orignal transform * 4 in the center, when we upscale an image using Fourier transform by a factor of 4, we multiply the Fourier transform by 4 because increasing the spatial dimensions corresponds to increasing the spatial frequencies. This ensures consistency between the spatial and frequency domains, leading to the correct scaling in the spatial domain when inverse transformed (Getting the upscaled image in the same details and brightness as the original image).

* Note: You can see both Fourier and the image of the upscaled image in row 2 above in zebra_scaled

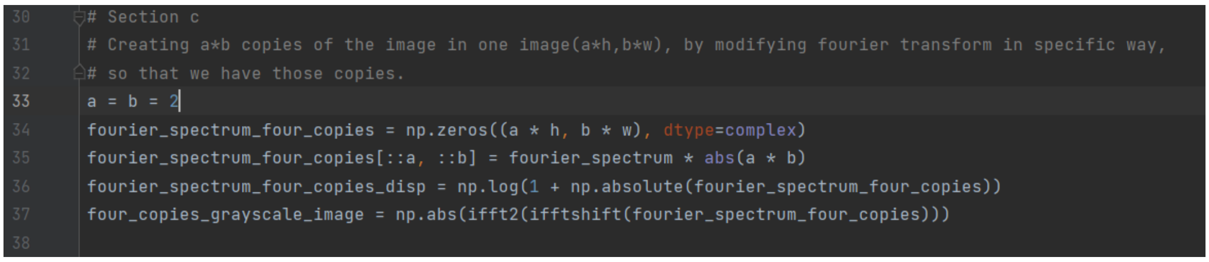


```python
# Section b
# Creating the image(2h,2w) by creating fourier(2h,2w) so that the fourier of original image * 4 is in the center
# and zero padding, so that we maintain the frequency and the colors of the original image in the new 2x image.
h, w = image.shape
fourier_spectrum_zero_padding = np.zeros((2 * h, 2 * w), dtype=complex)
fourier_spectrum_zero_padding[h // 2: 3 * h // 2, w // 2: 3 * w // 2] = fourier_spectrum * 4
fourier_spectrum_zero_padding_disp = np.log(1 + np.absolute(fourier_spectrum_zero_padding))
two_times_larger_grayscale_image = np.abs(ifft2(ifftshift(fourier_spectrum_zero_padding)))
```

Section c):

As we learned in Tutorial 6, in order to get and image that contains a * b copies of the original image, we need to separate Fourier coefficients by adding an a zeros vertically and b zeros horizontally, and that is result of following the given scale formula and then fixing it by multiplying the Fourier transform by a * b for the same reason in the upscaled image (to get the copies of the image in the same details and brightness as the original image)

* Note: You can see both Fourier and the image of the copies image in row 3 above in zebra_scaled

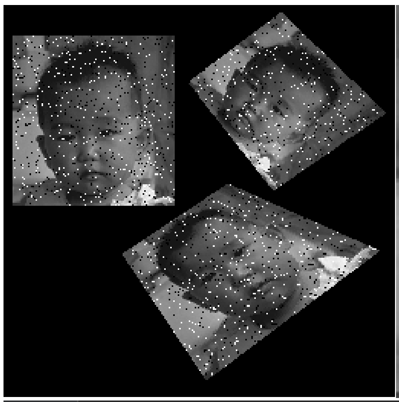


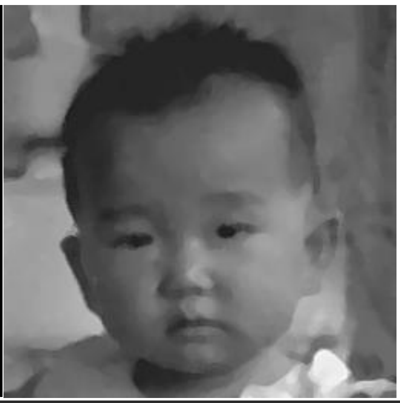
```python
# Section c
# Creating a*b copies of the image in one image(a*h,b*w), by modifying fourier transform in specific way,
# so that we have those copies.
a = b = 2
fourier_spectrum_four_copies = np.zeros((a * h, b * w), dtype=complex)
fourier_spectrum_four_copies[::a, ::b] = fourier_spectrum * abs(a * b)
fourier_spectrum_four_copies_disp = np.log(1 + np.absolute(fourier_spectrum_four_copies))
four_copies_grayscale_image = np.abs(ifft2(ifftshift(fourier_spectrum_four_copies)))
```

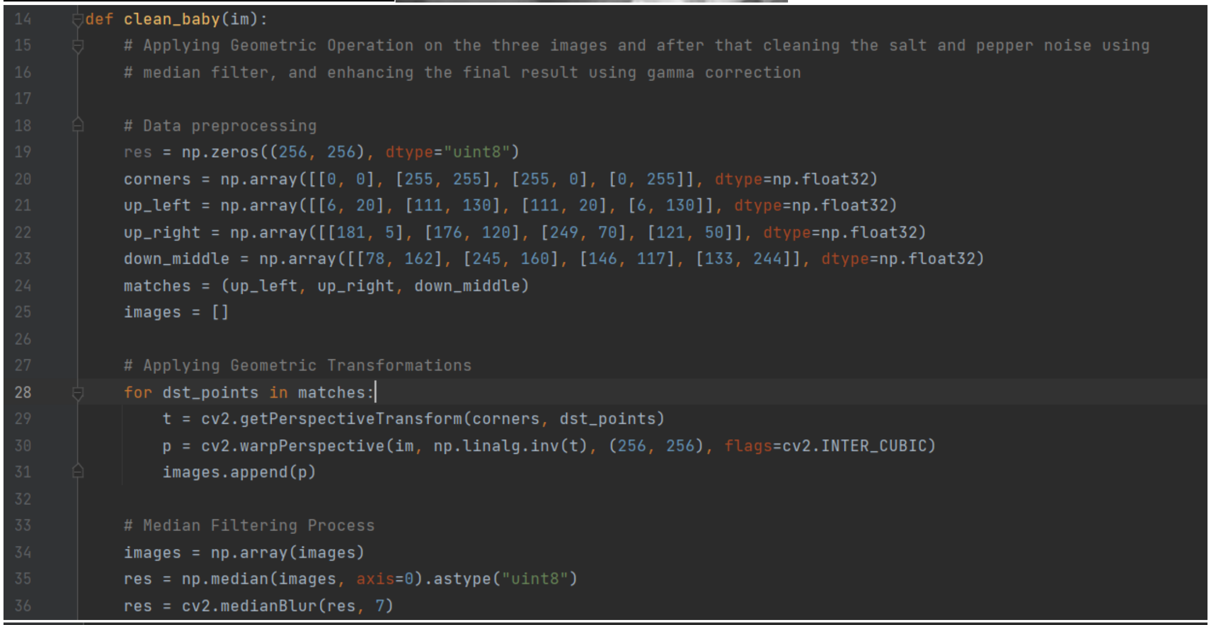
Problem 3:

Cleaning Baby:

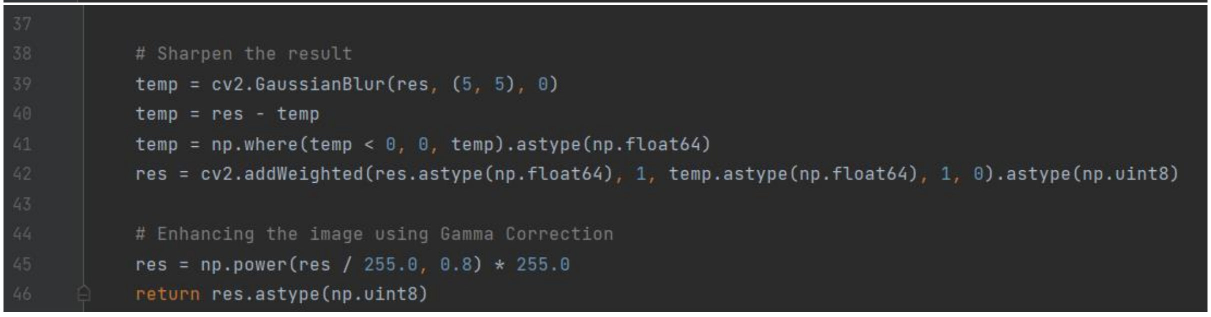
This problem is combination between hw1, hw2, and hw3, we first applied Geometric Transformations (Projective Transformation type) on the 3 images after finding matches points manually, then we filtered the images from salt and pepper noise using median filter, and finally since the image is dark we used gamma correction to enhance the image colours and get an better image. (we did not use max bright and contrast because the image already has high contrast)







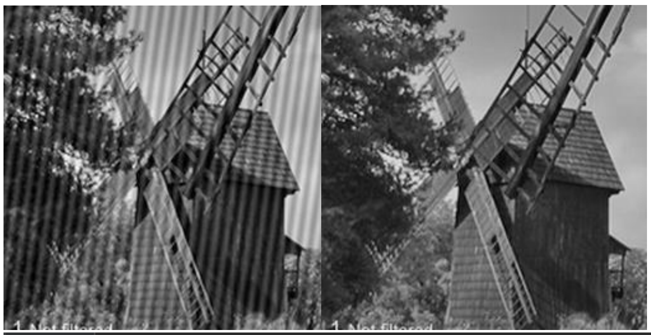
```python
def clean_baby(im):
    # Applying Geometric Operation on the three images and after that cleaning the salt and pepper noise using
    # median filter, and enhancing the final result using gamma correction

    # Data preprocessing
    res = np.zeros((256, 256), dtype="uint8")
    corners = np.array([[0, 0], [255, 255], [255, 0], [0, 255]], dtype=np.float32)
    up_left = np.array([[6, 20], [111, 130], [111, 20], [6, 130]], dtype=np.float32)
    up_right = np.array([[181, 5], [176, 120], [249, 70], [121, 50]], dtype=np.float32)
    down_middle = np.array([[78, 162], [245, 160], [146, 117], [133, 244]], dtype=np.float32)
    matches = (up_left, up_right, down_middle)
    images = []

    # Applying Geometric Transformations
    for dst_points in matches:
        t = cv2.getPerspectiveTransform(corners, dst_points)
        p = cv2.warpPerspective(im, np.linalg.inv(t), (256, 256), flags=cv2.INTER_CUBIC)
        images.append(p)

    # Median Filtering Process
    images = np.array(images)
    res = np.median(images, axis=0).astype("uint8")
    res = cv2.medianBlur(res, 7)
```

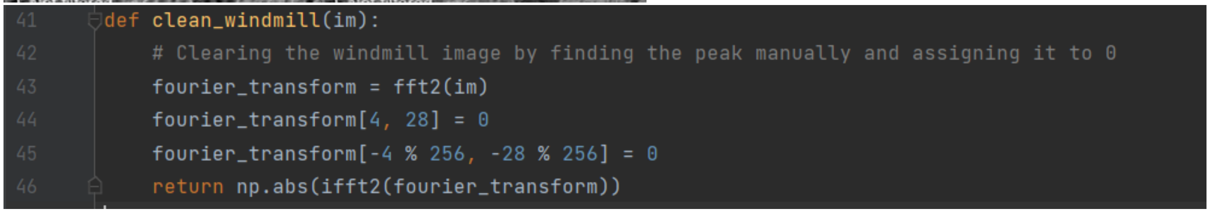


```python

    # Sharpen the result
    temp = cv2.GaussianBlur(res, (5, 5), 0)
    temp = res - temp
    temp = np.where(temp < 0, 0, temp).astype(np.float64)
    res = cv2.addWeighted(res.astype(np.float64), 1, temp.astype(np.float64), 1, 0).astype(np.uint8)

    # Enhancing the image using Gamma Correction
    res = np.power(res / 255.0, 0.8) * 255.0
    return res.astype(np.uint8)
```

Cleaning Windmill:

We can see there is a frequency noise in the image, so in order to remove the image as we learn we need the find the peak in the Fourier transform and decrease it (or make it 0), and by doing that we successfully denoised the image

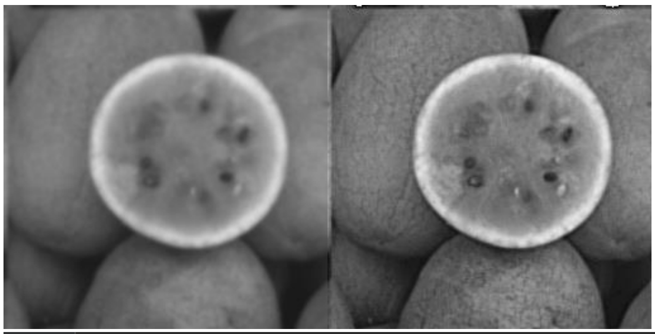


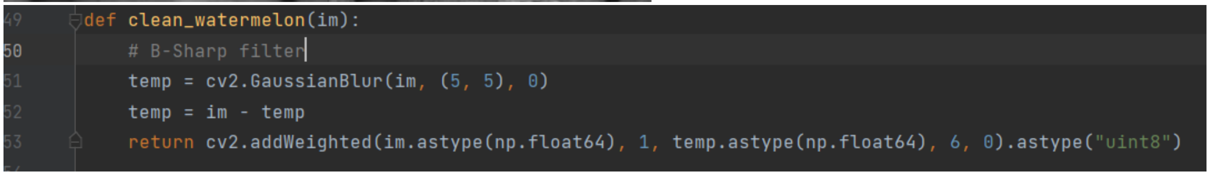


```
def clean_windmill(im):
    # Clearing the windmill image by finding the peak manually and assigning it to 0
    fourier_transform = fft2(im)
    fourier_transform[4, 28] = 0
    fourier_transform[-4 % 256, -28 % 256] = 0
    return np.abs(ifft2(fourier_transform))
```

Cleaning Watermelon:

We have an blurred image so in order do deblur it we need the sharpen the image, we did that by applying B-sharp filter(high pass filter) on the image to sharpen the edges and the details.

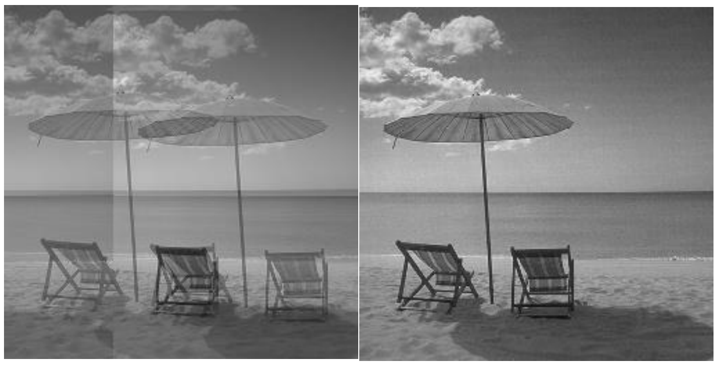


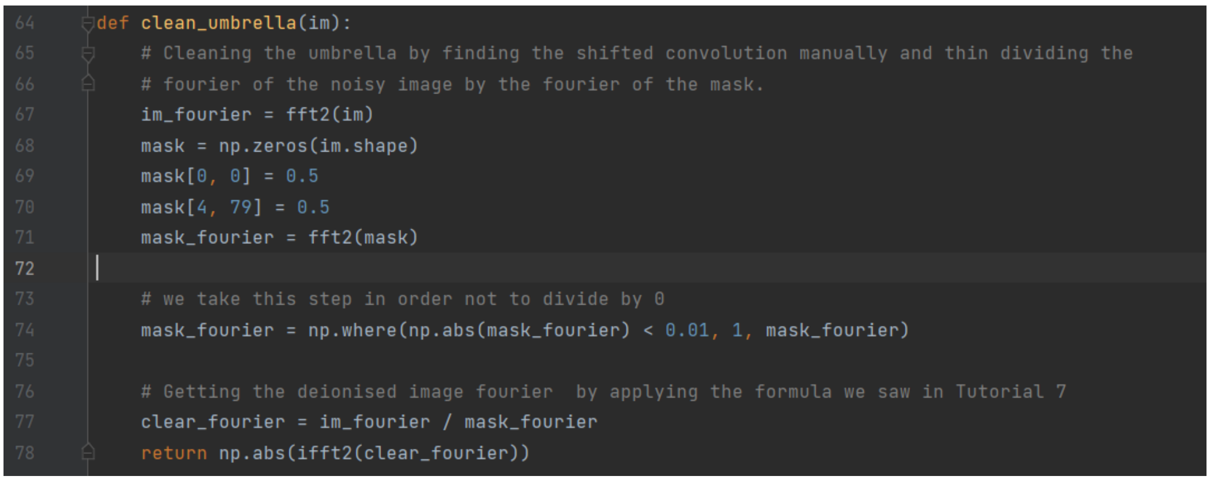


```
def clean_watermelon(im):
    # B-Sharp filter
    temp = cv2.GaussianBlur(im, (5, 5), 0)
    temp = im - temp
    return cv2.addWeighted(im.astype(np.float64), 1, temp.astype(np.float64), 6, 0).astype("uint8")
```

Cleaning Umbrella:

We can see in the image, a shifted noise, a copy from the image shifted and placed on the other one, in order to clear the noise we created the convolution for the noised image, and by dividing the Fourier of the noised image by the Fourier of the mask of the convolution we can get the denoised image as we saw in Tutorial 7.

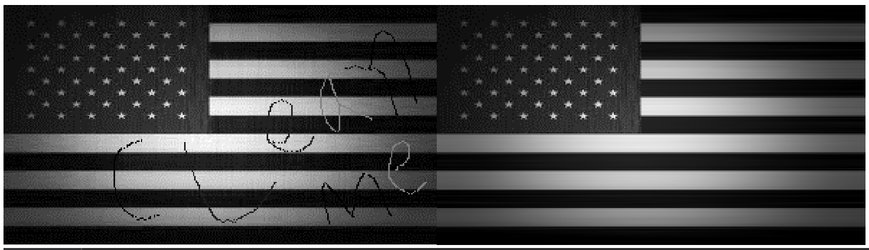


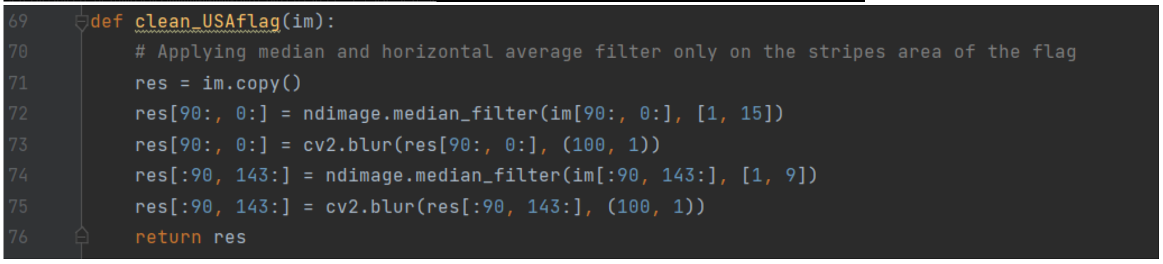


```
def clean_umbrella(im):
    # Cleaning the umbrella by finding the shifted convolution manually and thin dividing the
    # fourier of the noisy image by the fourier of the mask.
    im_fourier = fft2(im)
    mask = np.zeros(im.shape)
    mask[0, 0] = 0.5
    mask[4, 79] = 0.5
    mask_fourier = fft2(mask)

    # we take this step in order not to divide by 0
    mask_fourier = np.where(np.abs(mask_fourier) < 0.01, 1, mask_fourier)

    # Getting the deionised image fourier  by applying the formula we saw in Tutorial 7
    clear_fourier = im_fourier / mask_fourier
    return np.abs(ifft2(clear_fourier))
```

Cleaning USA Flag:

We can see the we have salt and pepper noise in the flag in the stripes area, so in order to clean the noise we applied median filter on those certain areas, and after that to get nice and smooth flag we applied an horizantial average filter so we can get smooth color of each stripe.

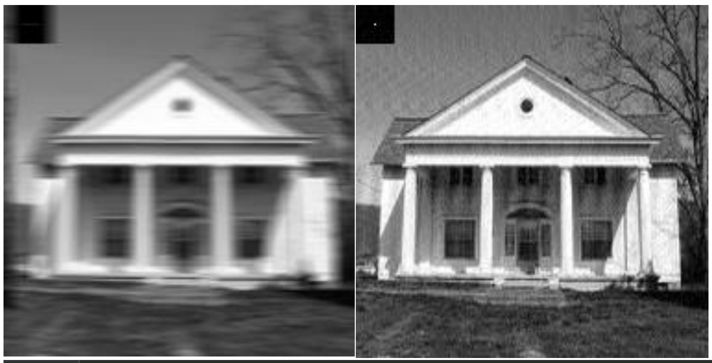


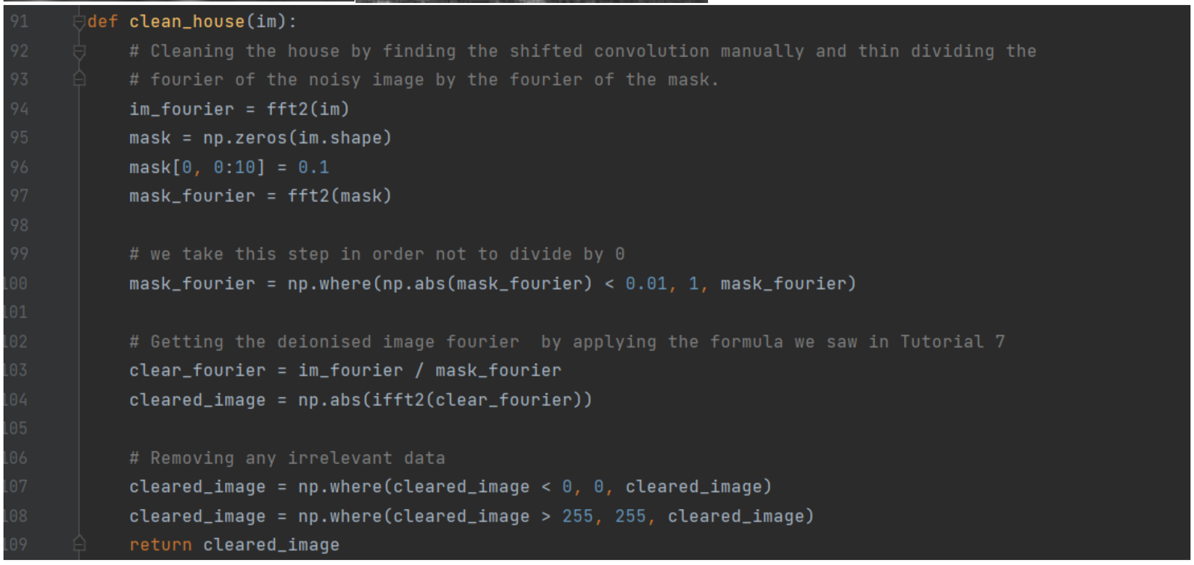


```
def clean_USAflag(im):
    # Applying median and horizontal average filter only on the stripes area of the flag
    res = im.copy()
    res[90:, 0:] = ndimage.median_filter(im[90:, 0:], [1, 15])
    res[90:, 0:] = cv2.blur(res[90:, 0:], (100, 1))
    res[:90, 143:] = ndimage.median_filter(im[:90, 143:], [1, 9])
    res[:90, 143:] = cv2.blur(res[:90, 143:], (100, 1))
    return res
```

Cleaning House:

We did the same thing for the umbrella image, We can see in the image, a shifted noise, a 10 copies from the image shifted and placed on each other, in order to clear the noise we created the convolution for the noised image, and by dividing the Fourier of the noised image by the Fourier of the mask of the convolution we can get the denoised image as we saw in Tutorial 7.

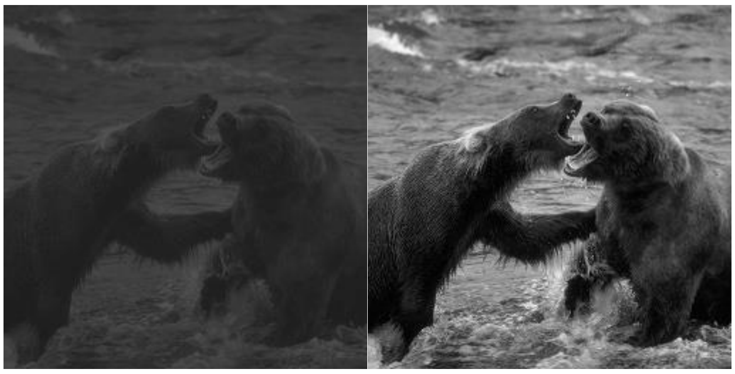


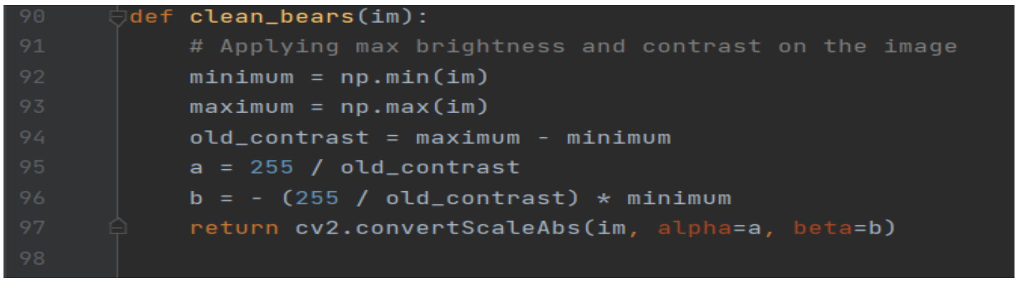


```python
def clean_house(im):
    # Cleaning the house by finding the shifted convolution manually and thin dividing the
    # fourier of the noisy image by the fourier of the mask.
    im_fourier = fft2(im)
    mask = np.zeros(im.shape)
    mask[0, 0:10] = 0.1
    mask_fourier = fft2(mask)

    # we take this step in order not to divide by 0
    mask_fourier = np.where(np.abs(mask_fourier) < 0.01, 1, mask_fourier)

    # Getting the deionised image fourier  by applying the formula we saw in Tutorial 7
    clear_fourier = im_fourier / mask_fourier
    cleared_image = np.abs(ifft2(clear_fourier))

    # Removing any irrelevant data
    cleared_image = np.where(cleared_image < 0, 0, cleared_image)
    cleared_image = np.where(cleared_image > 255, 255, cleared_image)
    return cleared_image
```

Cleaning Bears:

We can see the image in dark, so we need to find a filter to brighten the image and also shows the difference between the objects in the image, and we do that by applying max contrast and brightness filter we can achieve a brighter image, and maximizing the contrast in order to show the details in the image.





```
def clean_bears(im):
    # Applying max brightness and contrast on the image
    minimum = np.min(im)
    maximum = np.max(im)
    old_contrast = maximum - minimum
    a = 255 / old_contrast
    b = - (255 / old_contrast) * minimum
    return cv2.convertScaleAbs(im, alpha=a, beta=b)

```