CS 4442b Assignment 1

Danilo Vladicic - Dvladici@uwo.ca - 250861339

March 19, 2020

Question 1

Consider the simplest 3x3 Gaussian kernel. Where x, y represent the row/col offset respectively.

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\begin{bmatrix} \exp(-\frac{(1+1)}{2\sigma^2}) & \exp(-\frac{(1+0)}{2\sigma^2}) & \exp(-\frac{(1+1)}{2\sigma^2}) \\ \exp(-\frac{(0+1)}{2\sigma^2}) & \exp(-\frac{(0+0)}{2\sigma^2}) & \exp(-\frac{(0+1)}{2\sigma^2}) \\ \exp(-\frac{(1+1)}{2\sigma^2}) & \exp(-\frac{(1+0)}{2\sigma^2}) & \exp(-\frac{(1+1)}{2\sigma^2}) \end{bmatrix}$$

This matrix can be seperated into two identical vectors (one row vector, and one column vector)

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma}\begin{bmatrix} \exp(-\frac{1}{2\sigma^2}) \\ \exp(-\frac{0}{2\sigma^2}) \\ \exp(-\frac{1}{2\sigma^2}) \end{bmatrix} \quad G_\sigma(y) = \frac{1}{\sqrt{2\pi}\sigma}\begin{bmatrix} \exp(-\frac{1}{2\sigma^2}) & \exp(-\frac{0}{2\sigma^2}) & \exp(-\frac{1}{2\sigma^2}) \end{bmatrix}$$

Then it can be seen that (EQN 1)

$$G_\sigma(x,y) = G_\sigma(x)G_\sigma(y)$$

So if a convolution is done on matrix $O(i,j) = A \star G_\sigma(x,y)$ then element 1 will be

$$\begin{aligned} O_{11} = {} & A_{1,1} * G_\sigma(1,1) + A_{1,2} * G_\sigma(1,2) + A_{1,3} * G_\sigma(1,3) \\ & + A_{2,1} * G_\sigma(2,1) + A_{2,2} * G_\sigma(2,2) + A_{2,3} * G_\sigma(2,3) \\ & + A_{3,1} * G_\sigma(3,1) + A_{3,2} * G_\sigma(3,2) + A_{3,3} * G_\sigma(3,3) \end{aligned}$$

If the convolution is done row-wise and column wise then

$$\begin{aligned} O_{11} = {} & (A_{1,1} * G_\sigma(1) + A_{1,2} * G_\sigma(2) + A_{1,3} * G_\sigma(3)) * G_\sigma(1) \\ & + (A_{2,1} * G_\sigma(1) + A_{2,2} * G_\sigma(2) + A_{2,3} * G_\sigma(3)) * G_\sigma(2) \\ & + (A_{3,1} * G_\sigma(1) + A_{3,2} * G_\sigma(2) + A_{3,3} * G_\sigma(3)) * G_\sigma(3) \end{aligned}$$

Then if this is expanded and EQN 1 is used to substitue the multiplication then it is exactly equivalent to the convolution done using 2D matrix.

Sobel Kernel

The Sobel kernel is one of the most well known spatially separable matrix. It seperates into two vectors

$$a = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} b = \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

Both vecotrs can be transposed and used to find vertical vs horizontal edges!

Why Separable Kernels?

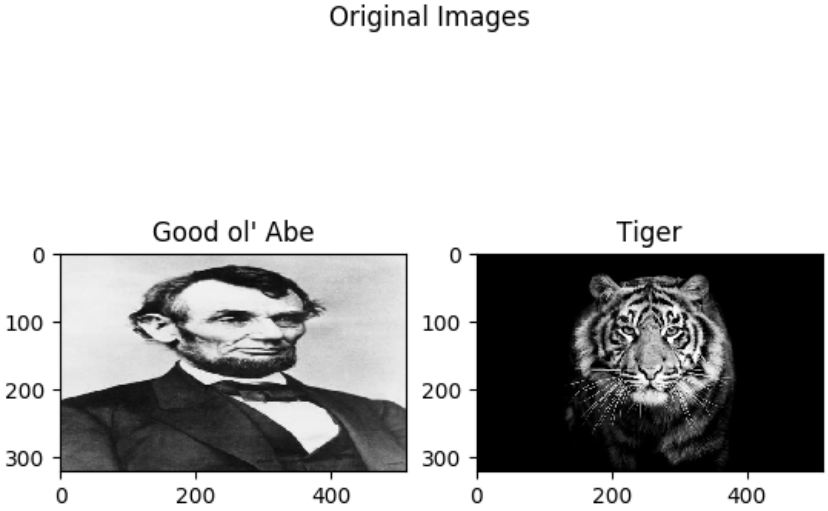
Consider from above the two equations for calculating the elements of the convolution for a 5x5 matrix A and a 3x3 kernel K. For each application of the kernel there is (kernel width*kernel height) multiplications. The kernel can slide horizontally (number of columns of A - number of columns of K + 1) and the kernel can slide vertically (number of rows A - number of rows K + 1). Thus the total number of multiplications is (kernel width * kernel height * horiz slides * vert slides). For our 5x5 matrix and 3x3 kernel that results in a total = 3*3*(5-3 + 1)*(5-3 + 1) = 81 multiplications Now for a separable kernel there are 2 1D kernels of size 3x1 and 1x3. For the rows there are kernel width multiplications and the kernel can slide (number of columns of A - number of columns of K + 1 = 3) and there are a total of 5 rows. Thus total row ops = 3*3*5 = 45. The new A' is 5x3. For cols = 3*3*3 = 27 thus. total ops = 27 + 45 = 72, therefore reducing the overall time complexity of the operation. There is also a small decrease in the space complexity, ie storing 2 1D kernels is 6 elements vs a 2D kernel of 9 elements.

Question 2

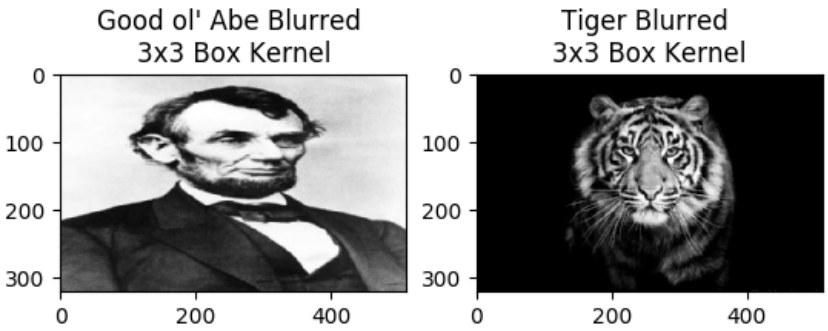
Important Code: question2.py

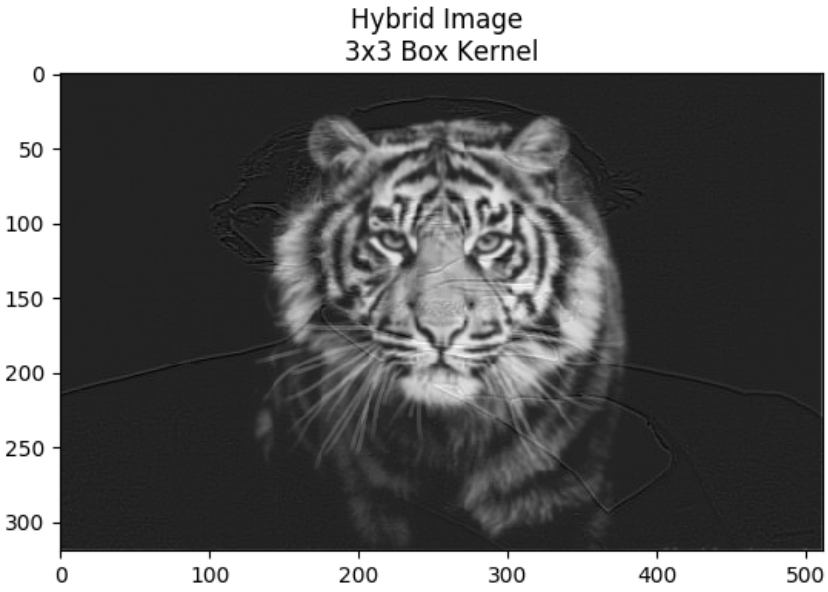
Images: tiger.jpg and abe.jpg

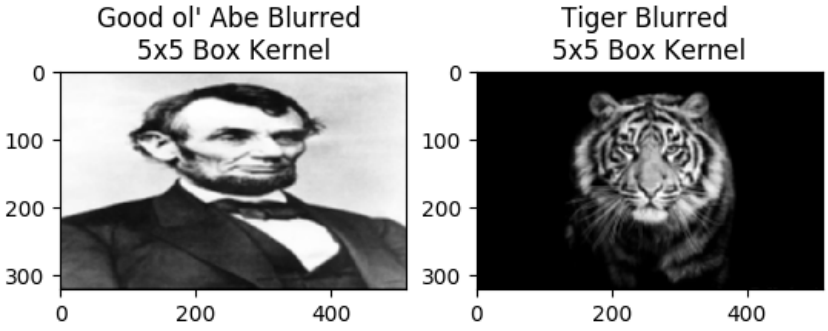
Both images are 512x320.

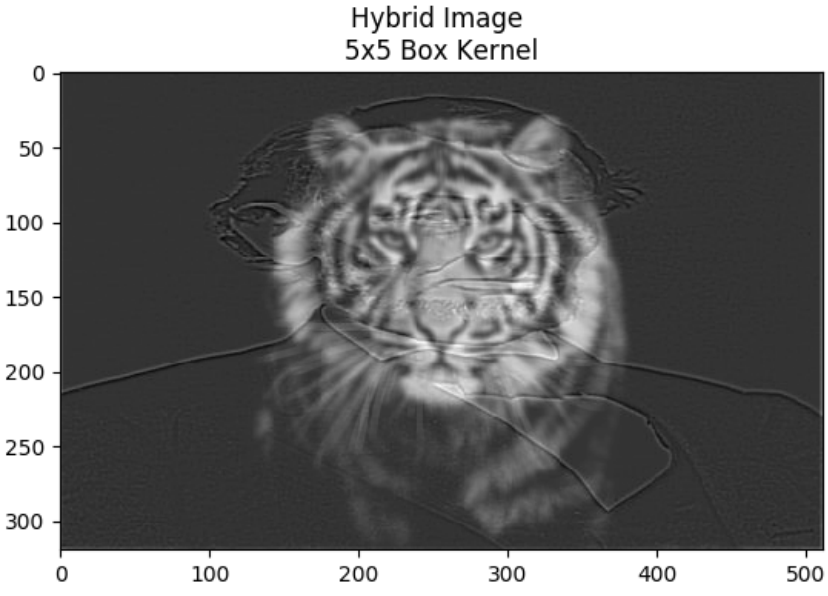


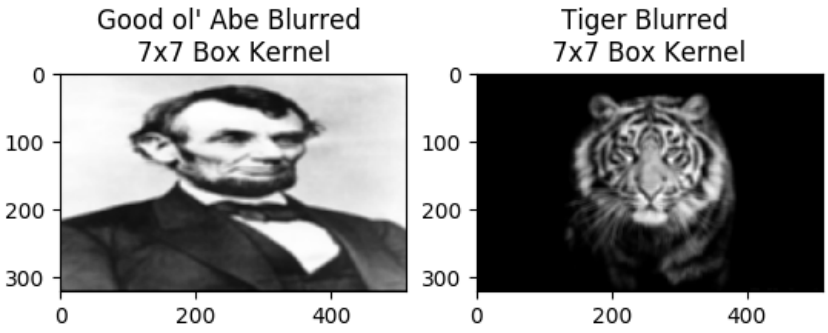
Hybrid images using mean kernel to filter the images

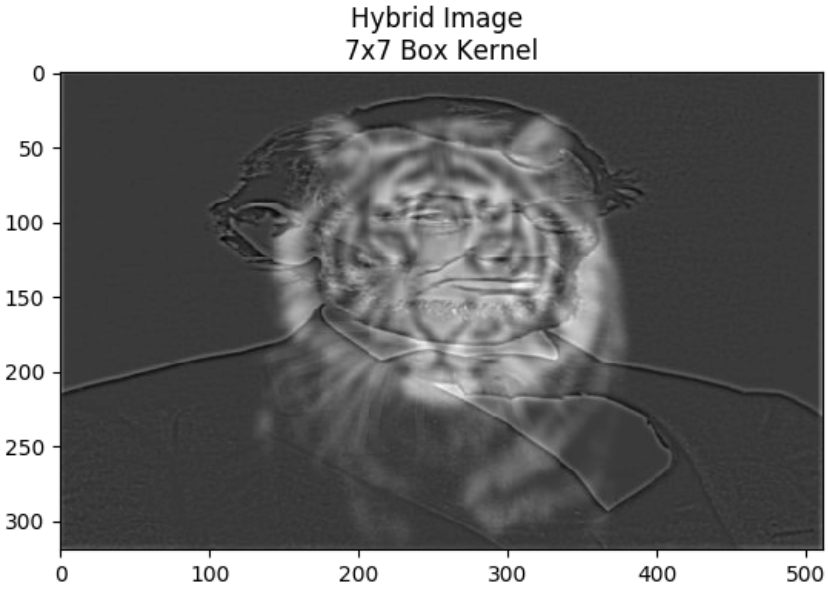


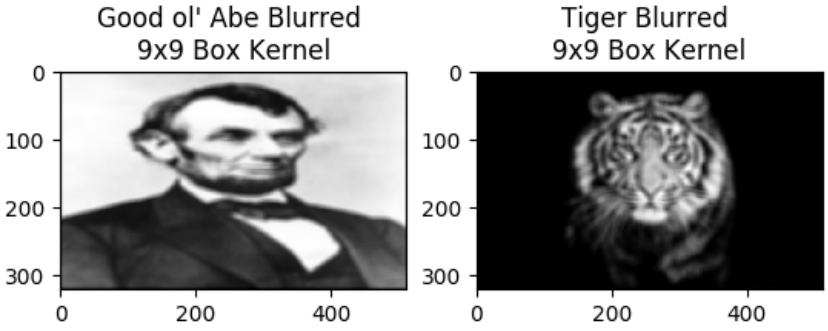


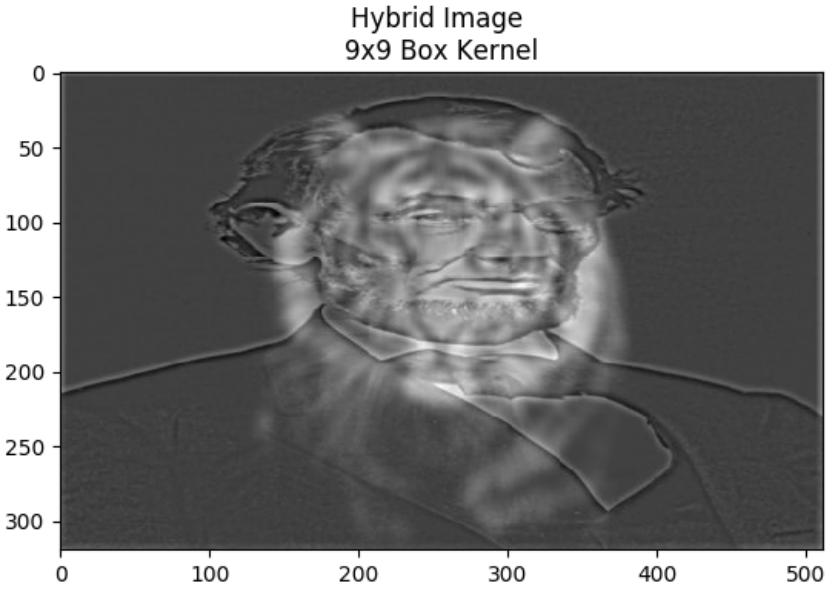

Good ol' Abe Blurred
5x5 Box Kernel
Tiger Blurred
5x5 Box Kernel
0
100
200
300
0
200
400

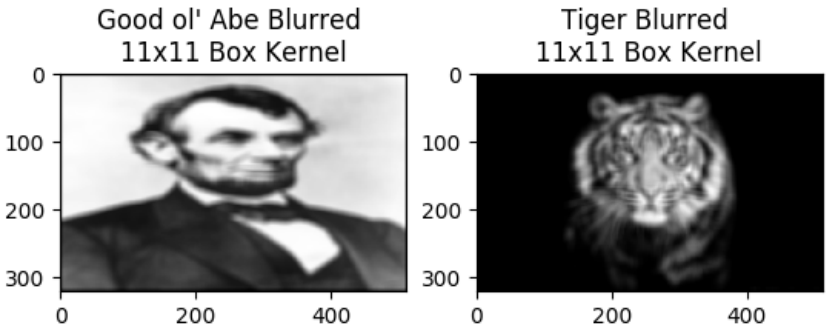

Hybrid Image
5x5 Box Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500

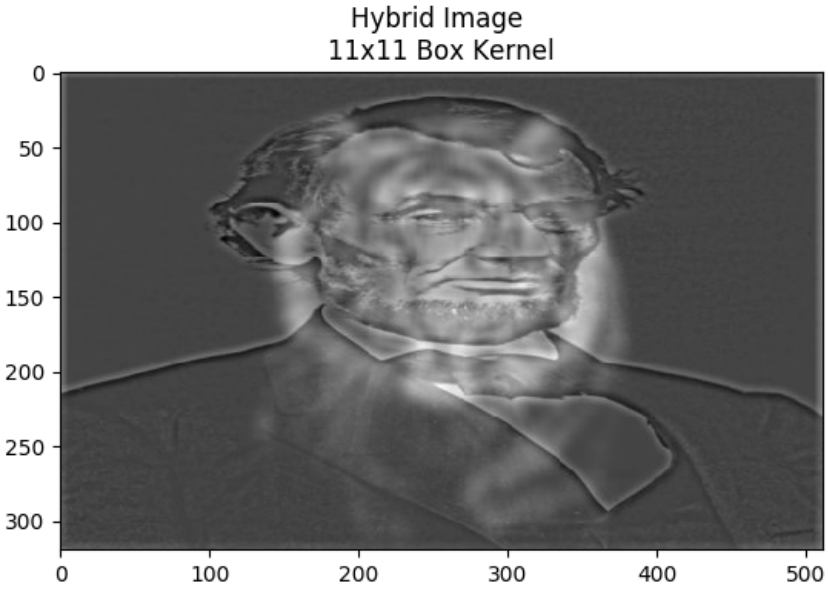

Good ol' Abe Blurred
7x7 Box Kernel
Tiger Blurred
7x7 Box Kernel
0
100
200
300
0
200
400


Hybrid Image
7x7 Box Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500

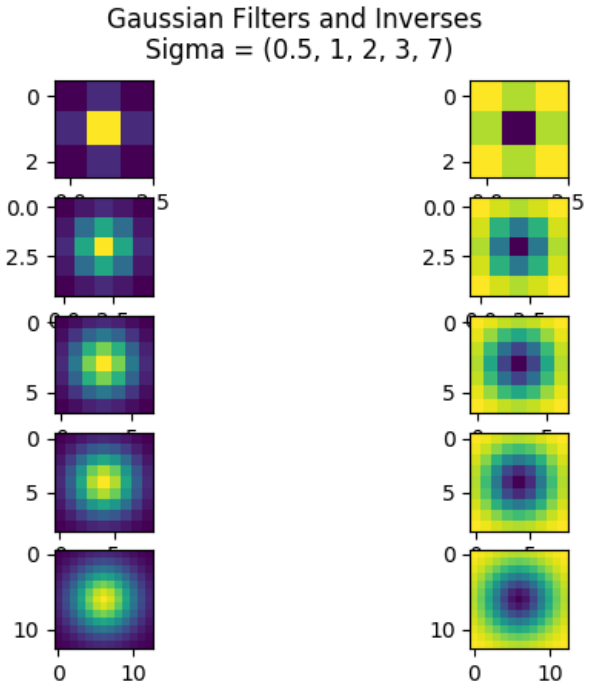

Good ol' Abe Blurred
9x9 Box Kernel
Tiger Blurred
9x9 Box Kernel
0
100
200
300
0
200
400

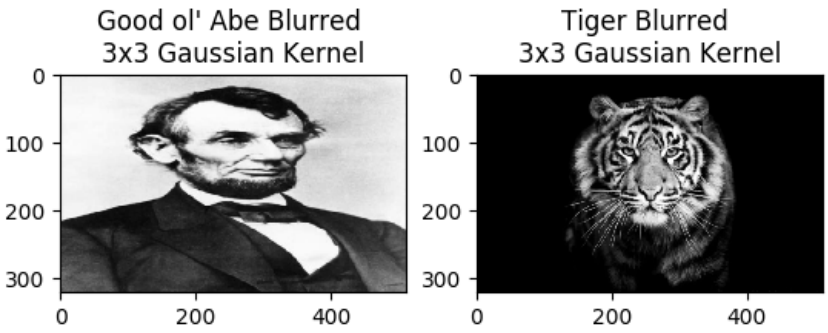

Hybrid Image
9x9 Box Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500

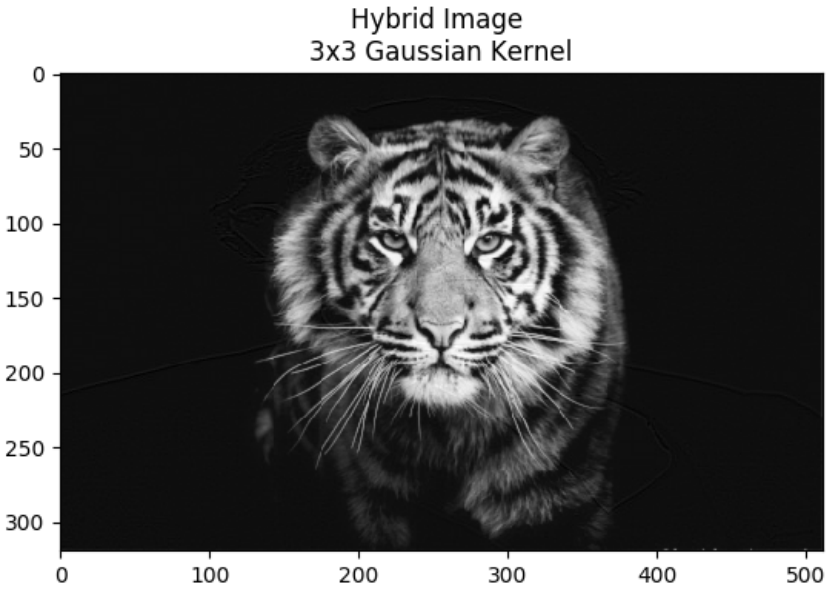


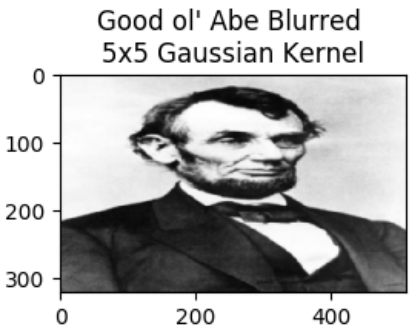


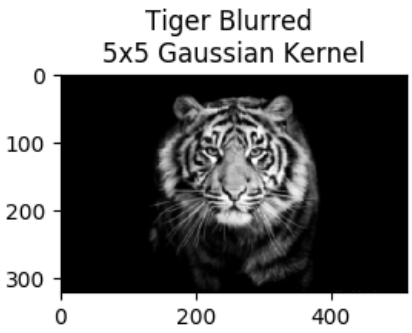
Visualizing the 2D Gaussian Filter and its inverse

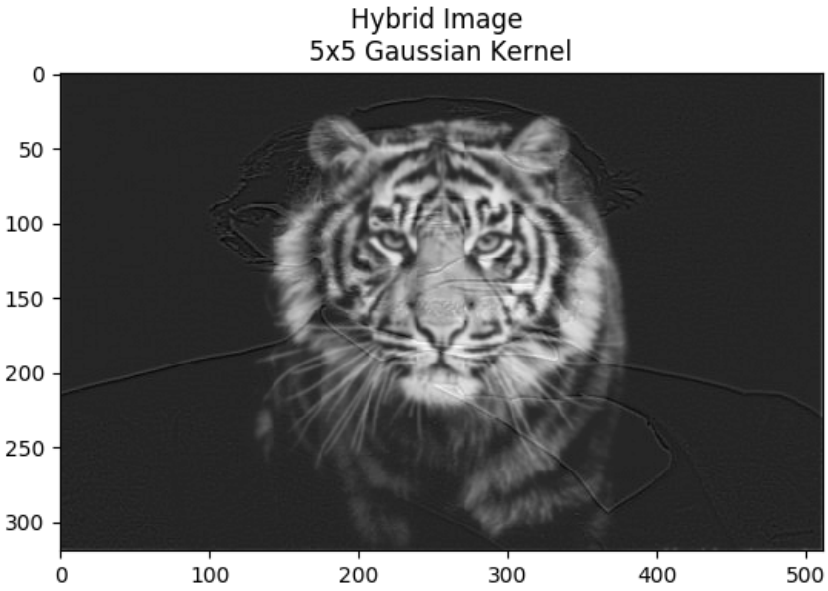

Gaussian Filters and Inverses
Sigma = (0.5, 1, 2, 3, 7)

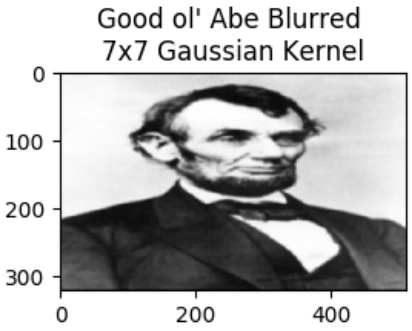

Good ol' Abe Blurred
3x3 Gaussian Kernel
Tiger Blurred
3x3 Gaussian Kernel

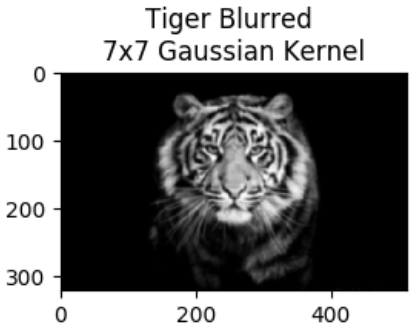

Hybrid Image
3x3 Gaussian Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500

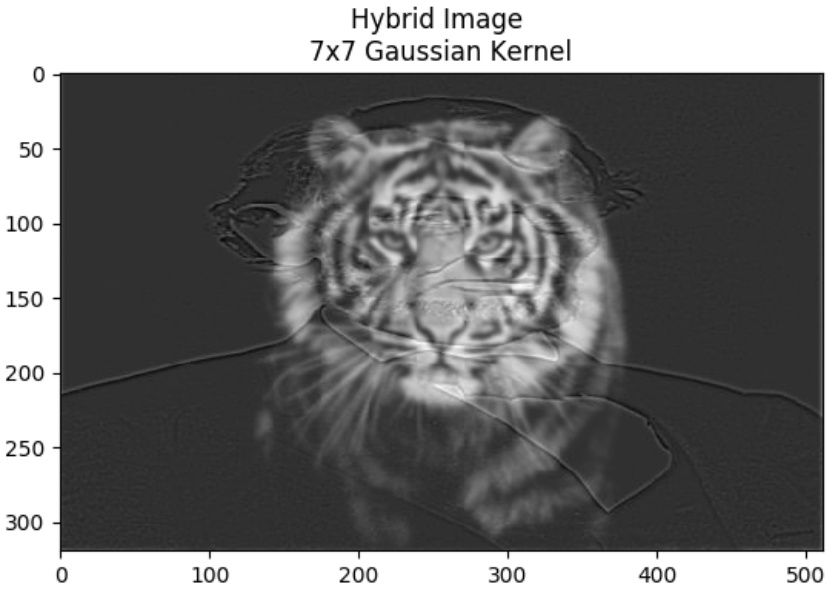

Good ol' Abe Blurred
5x5 Gaussian Kernel
0
100
200
300
0
200
400

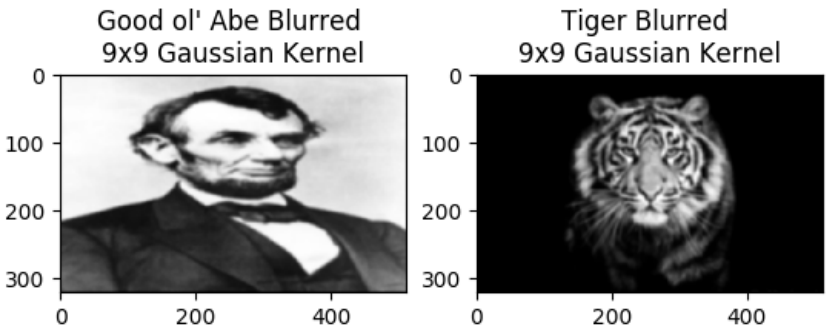

Tiger Blurred
5x5 Gaussian Kernel
0
100
200
300
0
200
400


Hybrid Image
5x5 Gaussian Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500

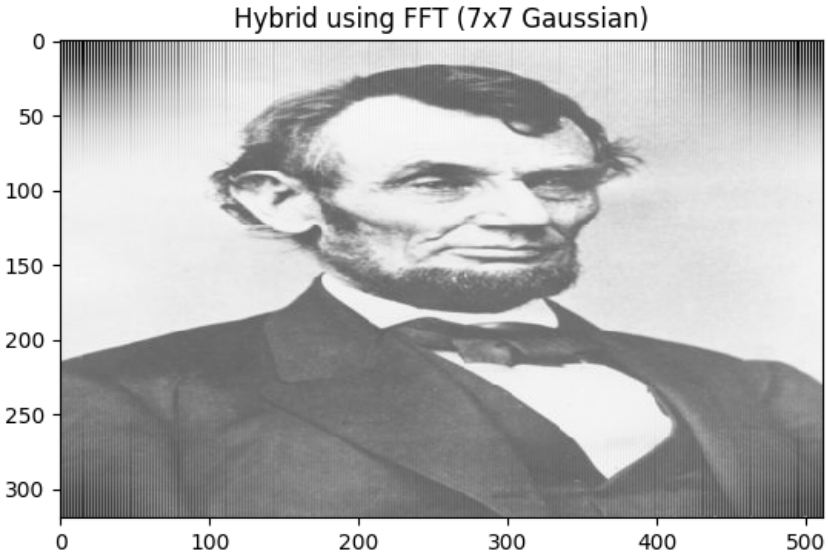

Good ol' Abe Blurred
7x7 Gaussian Kernel
0
100
200
300
0
200
400

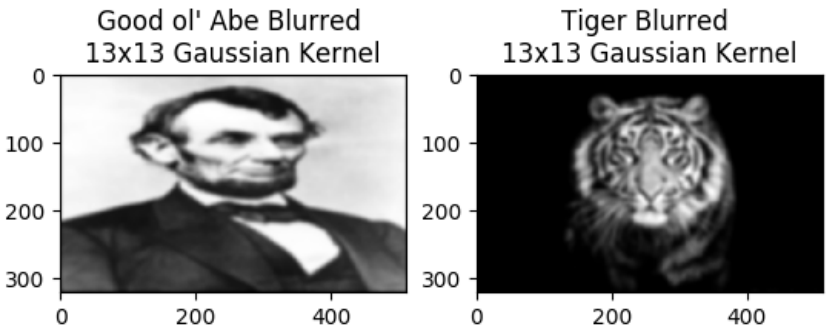

Tiger Blurred
7x7 Gaussian Kernel
0
100
200
300
0
200
400

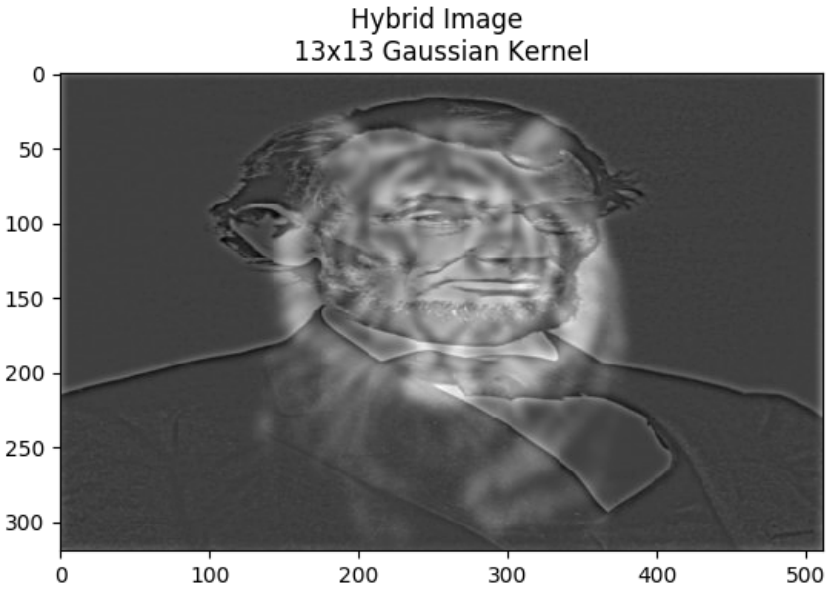

Hybrid Image
7x7 Gaussian Kernel
0
50
100
150
200
250
300
0
100
200
300
400
500


Good ol' Abe Blurred
9x9 Gaussian Kernel
0
100
200
300
0
200
400

Tiger Blurred
9x9 Gaussian Kernel
0
100
200
300
0
200
400

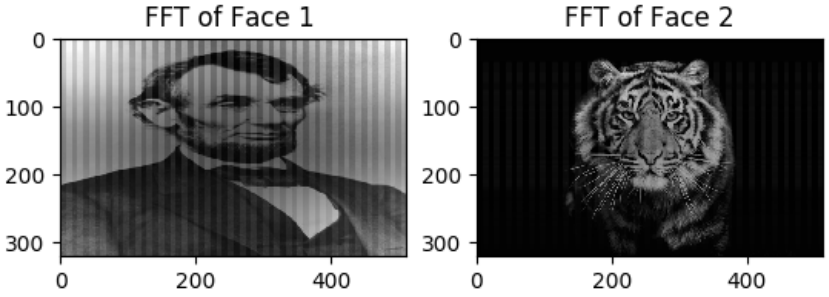

Hybrid using FFT (7x7 Gaussian)
0
50
100
150
200
250
300
0
100
200
300
400
500

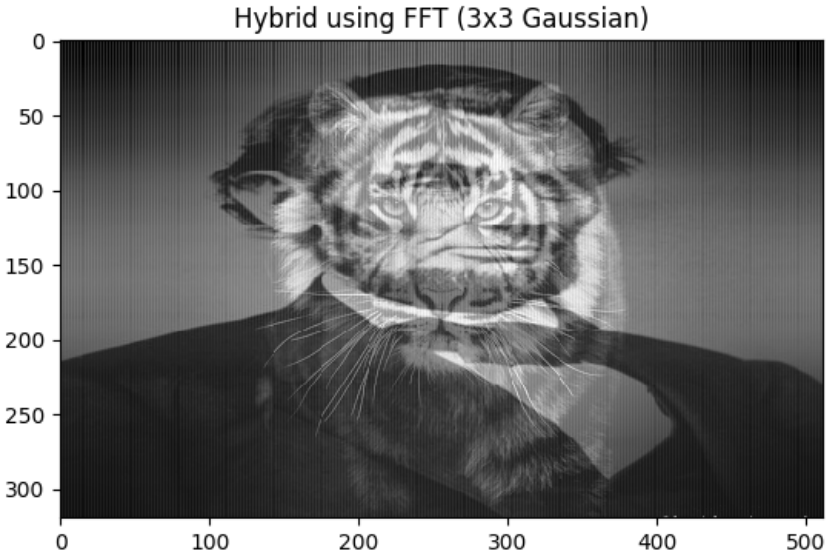

Good ol' Abe Blurred
13x13 Gaussian Kernel
Tiger Blurred
13x13 Gaussian Kernel
0
100
200
300
0
200
400
0
100
200
300
0
200
400



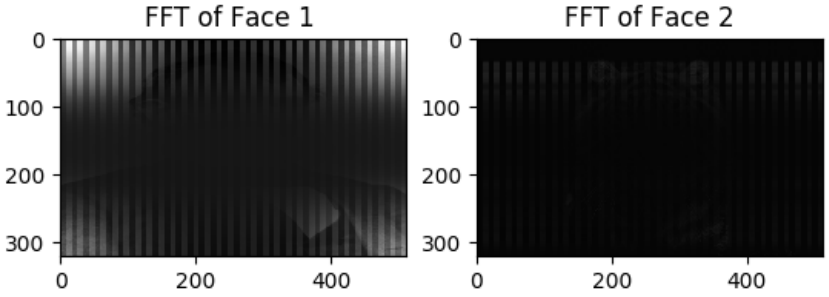
These are done using the same 2D Gaussian Kernel on the images

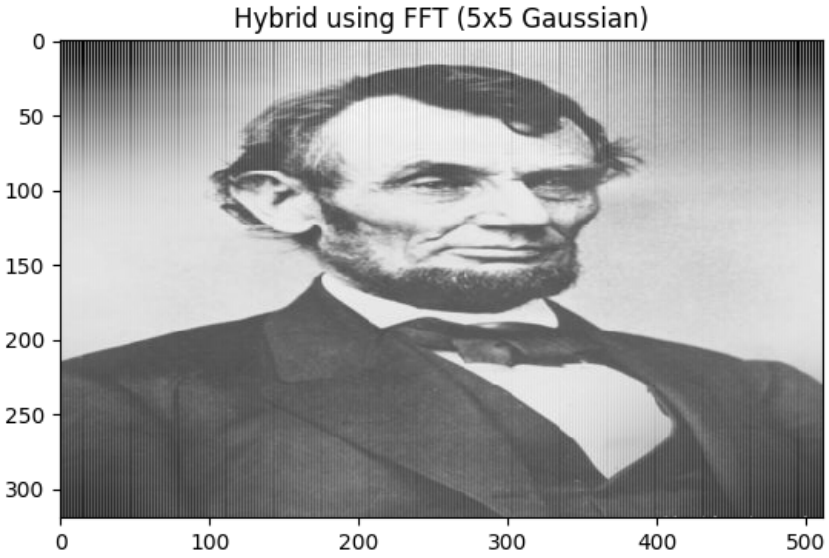
3x3 Gaussian Blur



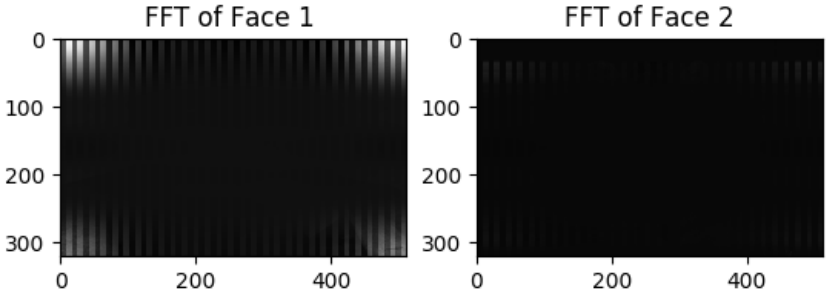


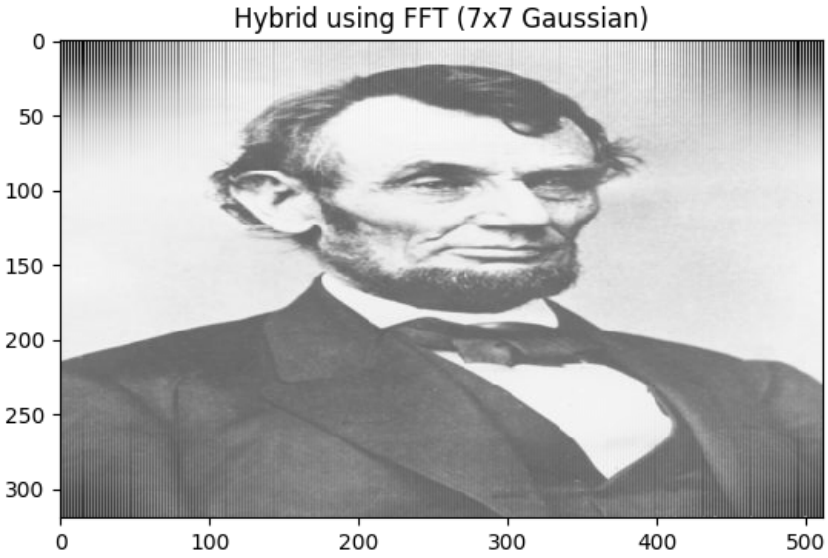
5x5 Gaussian Blur



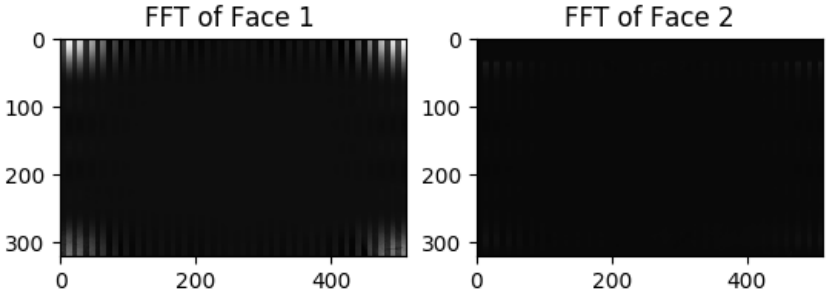


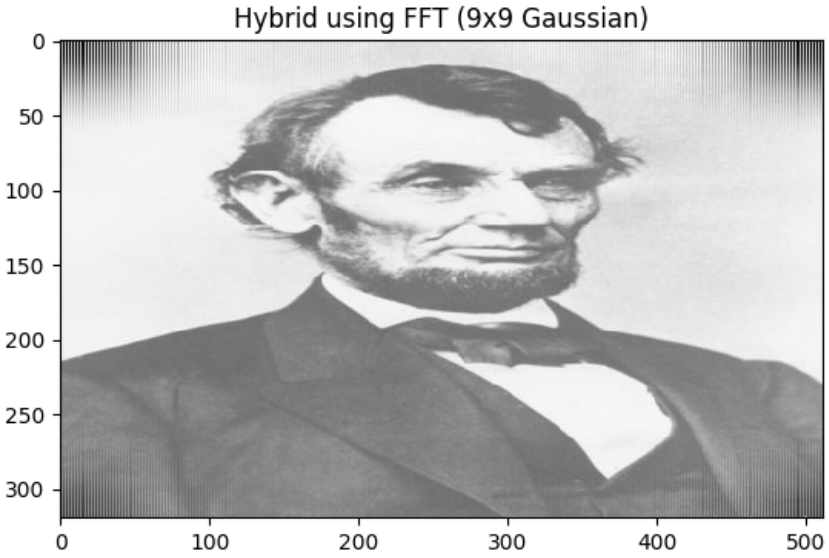
7x7 Gaussian Blur



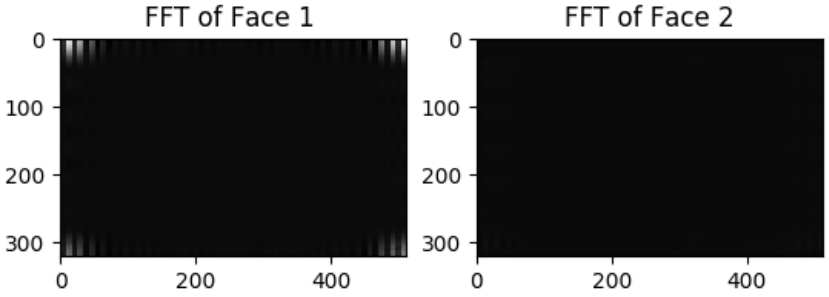


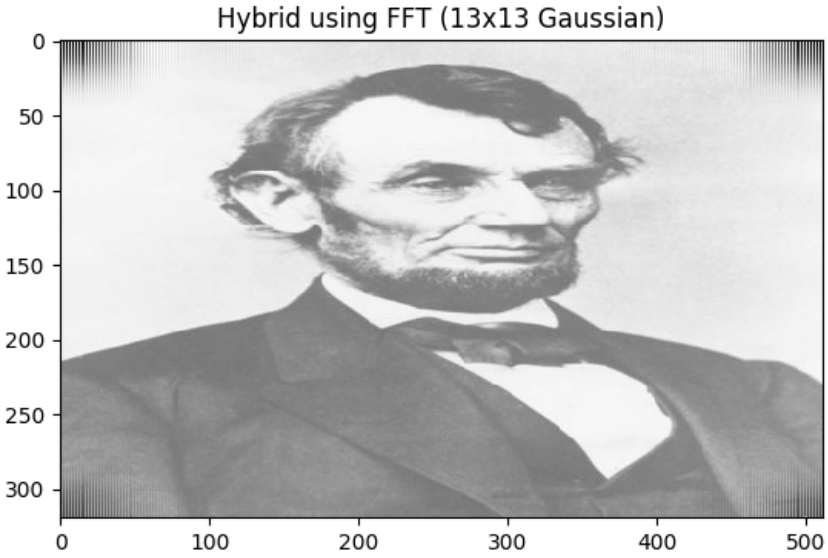
9x9 Gaussian Blur





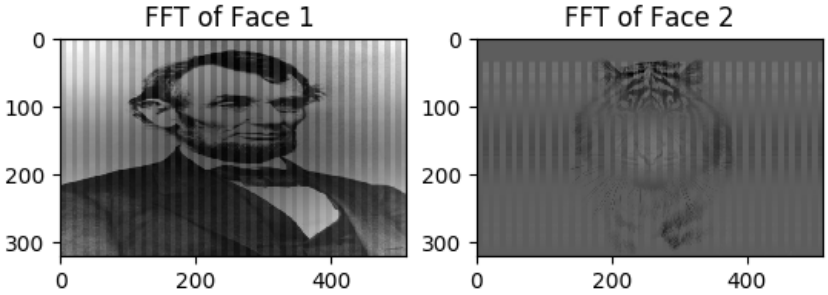
13x13 Gaussian Blur

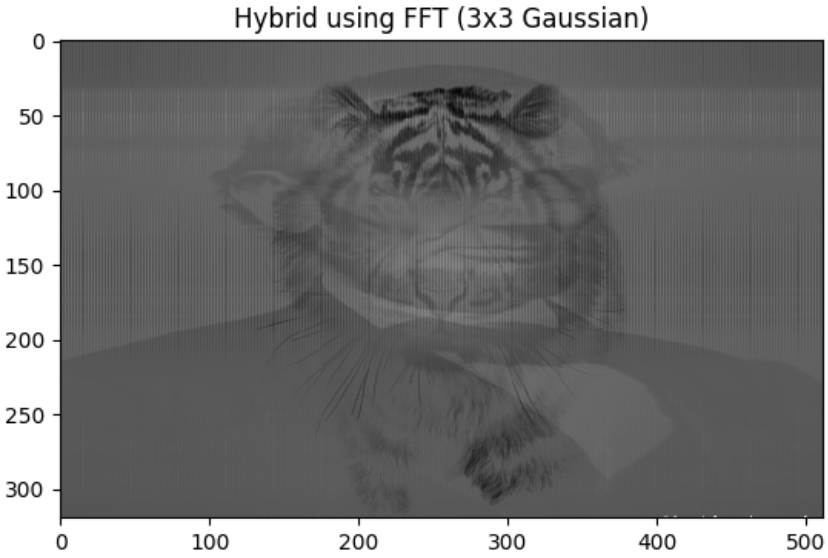




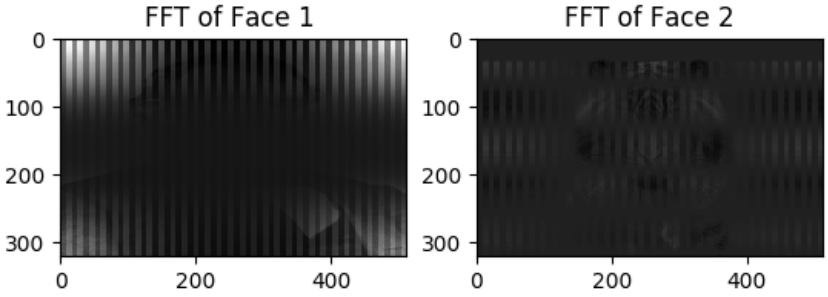
These are done with 2D Gaussian filter and it's inverse kernel applied on the Discrete Frourier Transform (DFT) of the images respectively

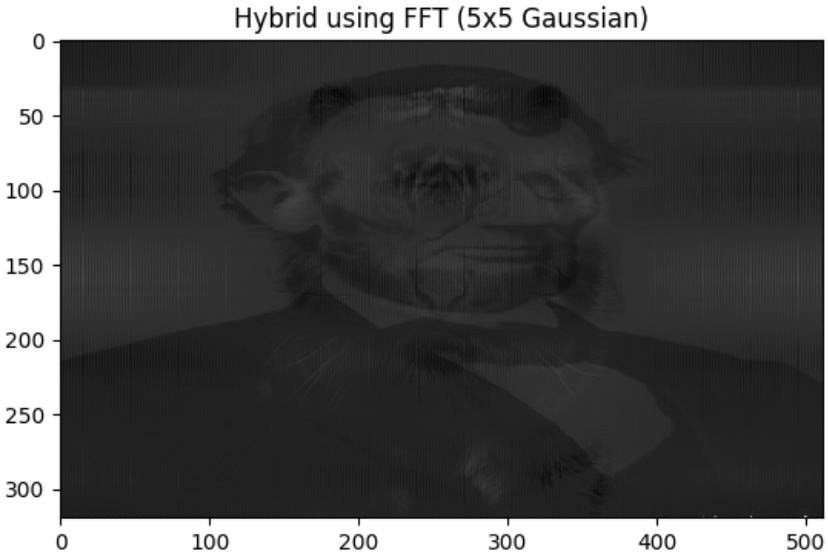
3x3 Gaussian Blur



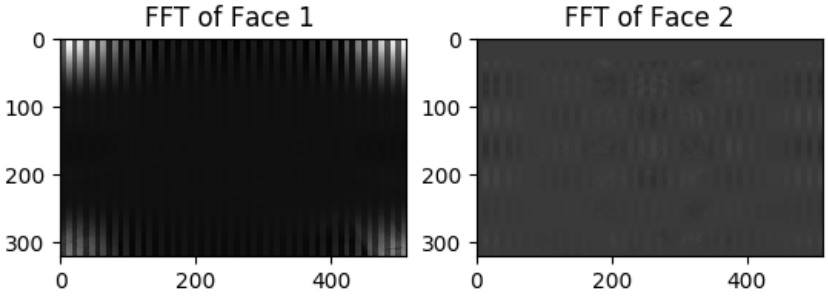


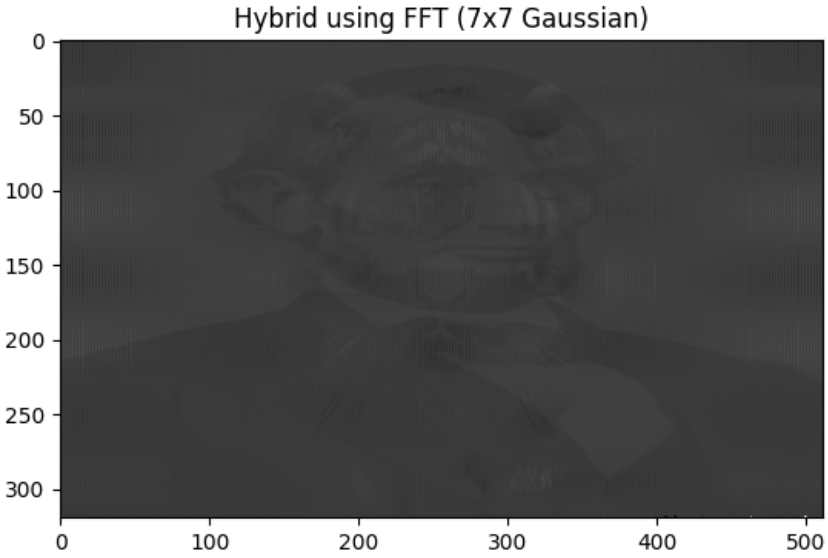
5x5 Gaussian Blur



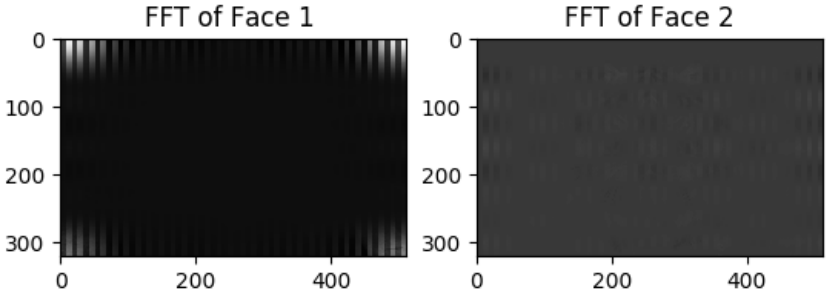


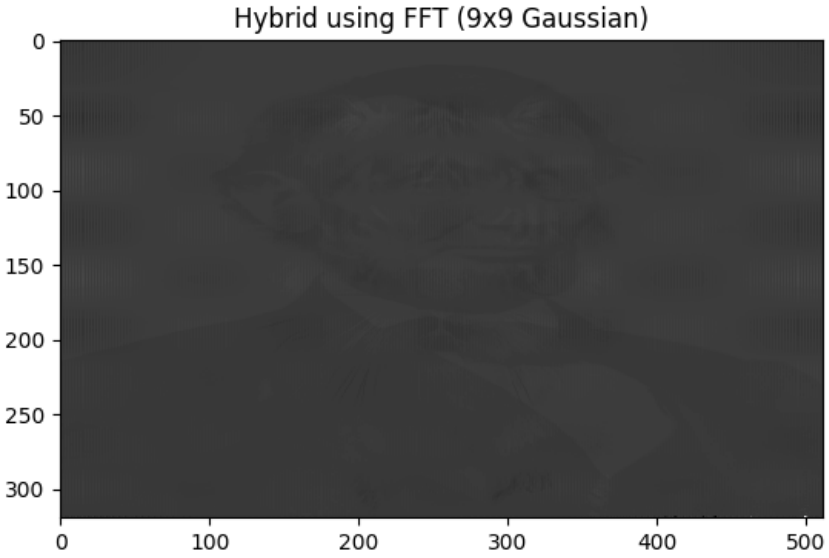
7x7 Gaussian Blur





9x9 Gaussian Blur



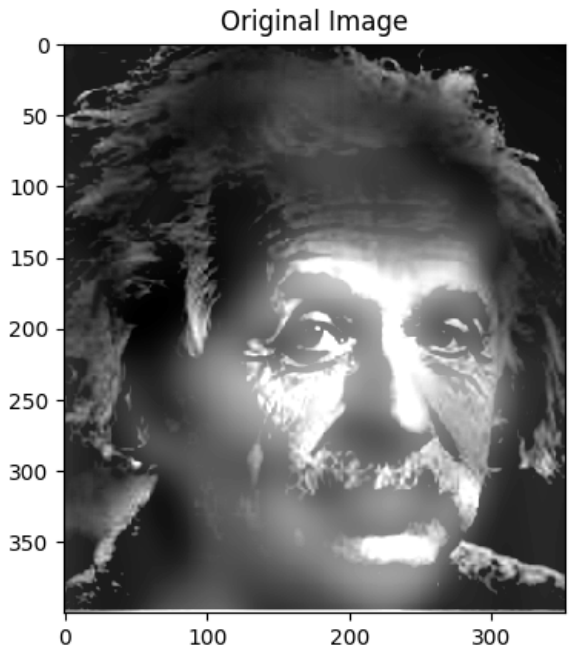

Hybrid using FFT (9x9 Gaussian)
0
50
100
150
200
250
300
0
100
200
300
400
500

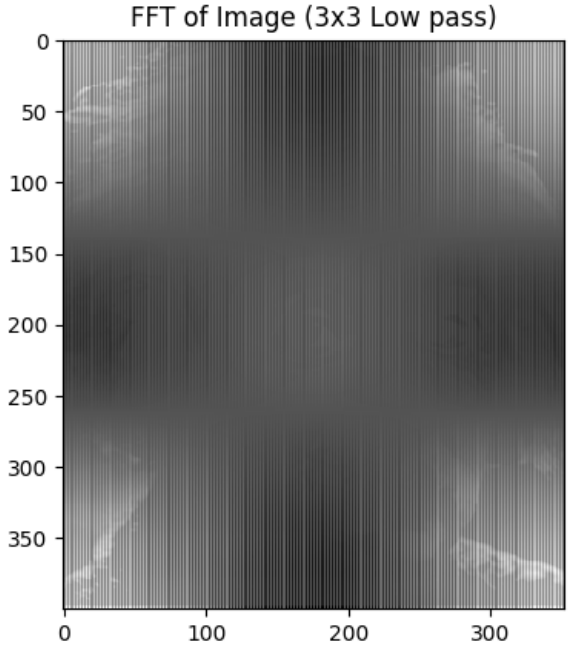
Question 3

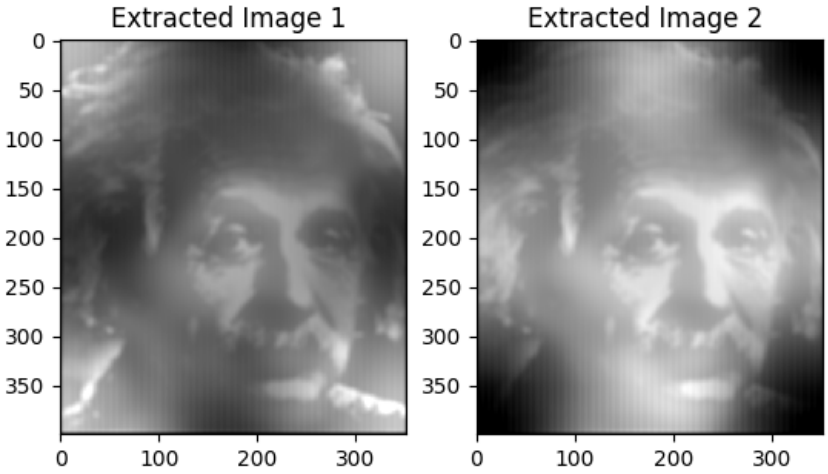
Important Code: question3.py

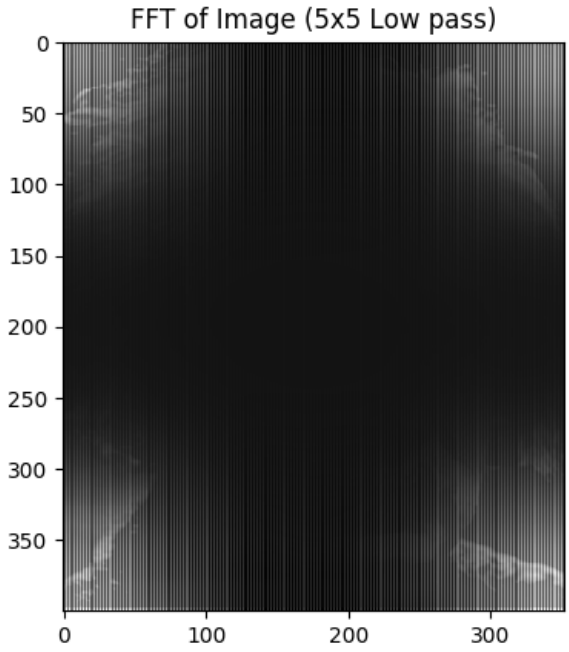
The kernels used in this section are all 2D Gaussian kernels of varying window sizes (3, 5, 7, 9, 13). First the DFT of the original image is computed, then that data is convolved with the 2D Gaussian Kernel, then the inverse DFT is calculated. For the first image the mask is added to the image and then convolved again with a Gaussian kernel. For the second image, the mask is subtracted from the image, and then it is convolved with the inverse gaussian kernel. Additionally both the extracted images are blurred with a 5x5 (or 3x3) mean filter to remove some overlapping features and smooth out the data.

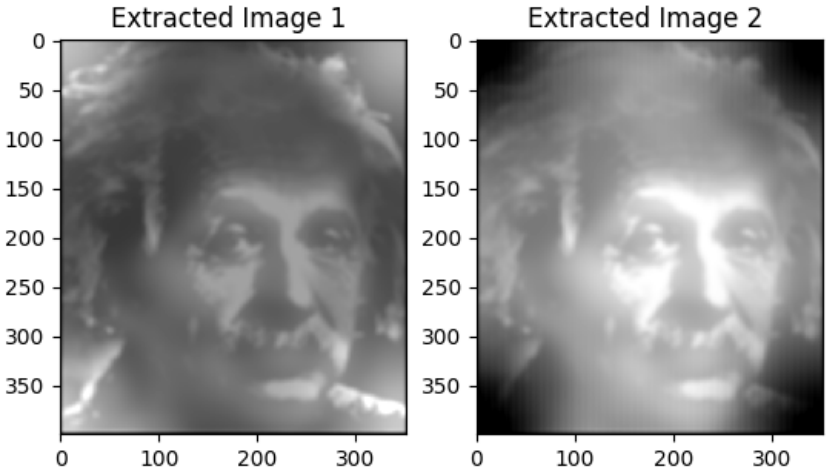
Original image:

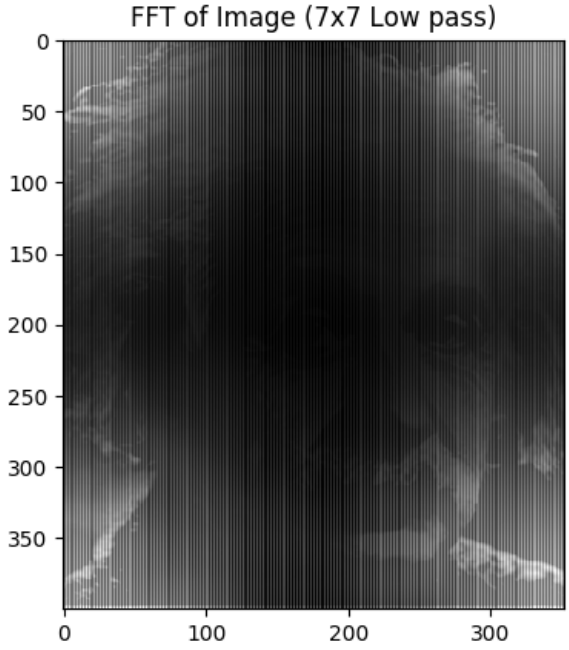


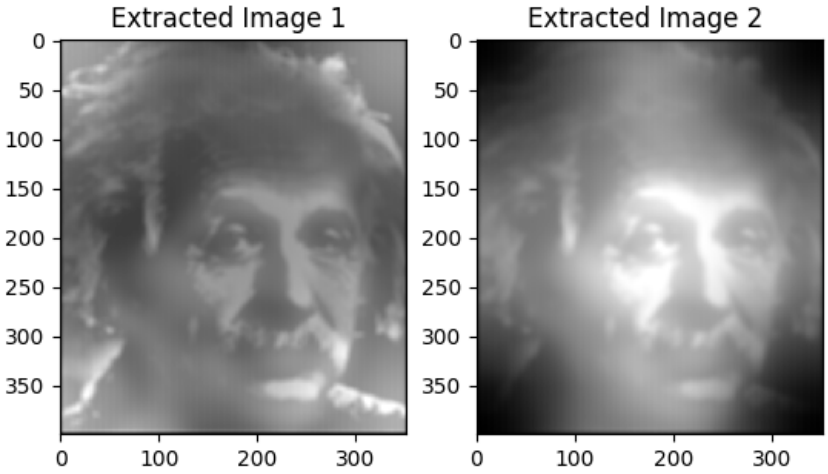

FFT of Image (3x3 Low pass)
0
50
100
150
200
250
300
350
0
100
200
300

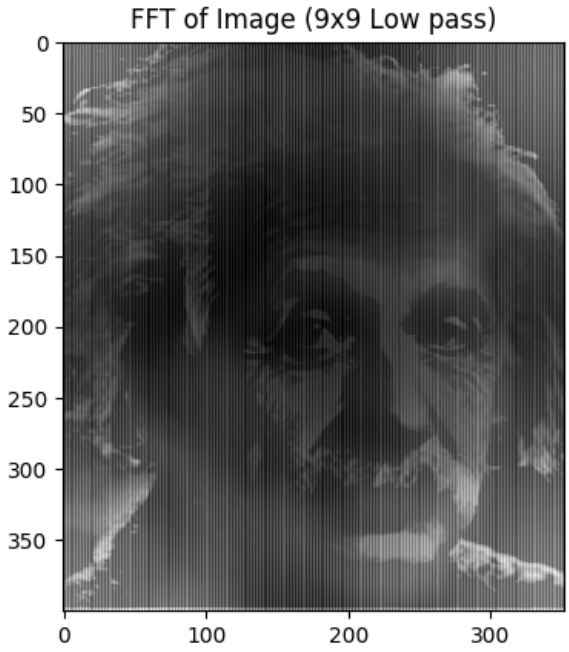

Extracted Image 1
Extracted Image 2
0
50
100
150
200
250
300
350
0
100
200
300

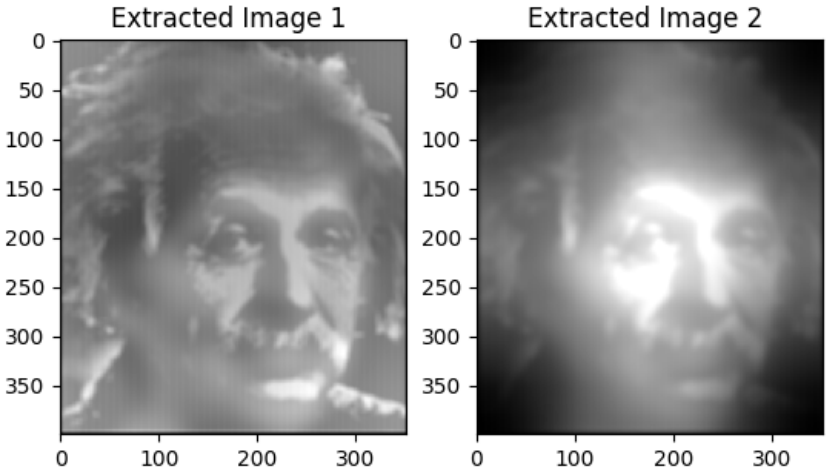

FFT of Image (5x5 Low pass)
0
50
100
150
200
250
300
350
0
100
200
300

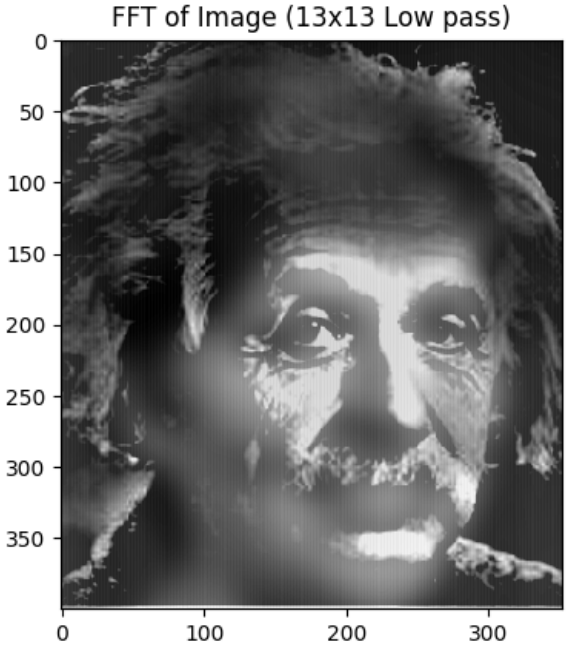

Extracted Image 1
Extracted Image 2
0
50
100
150
200
250
300
350
0
100
200
300

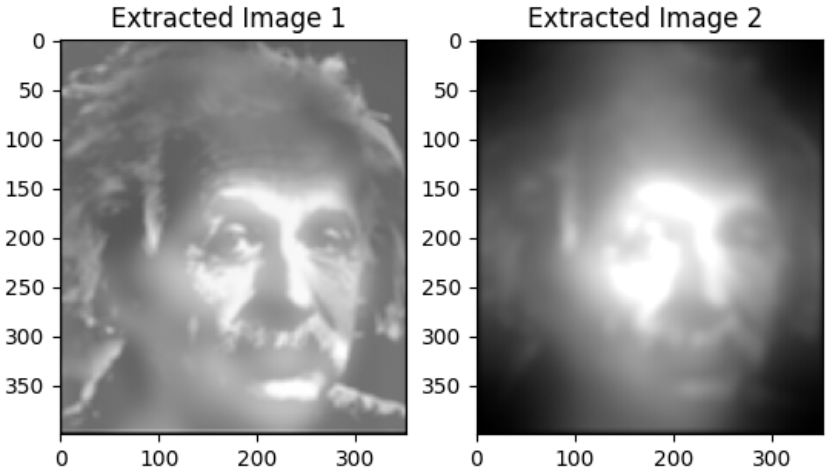

FFT of Image (7x7 Low pass)
0
50
100
150
200
250
300
350
0
100
200
300


Extracted Image 1
Extracted Image 2
0
50
100
150
200
250
300
350
0
100
200
300


FFT of Image (9x9 Low pass)
0
50
100
150
200
250
300
350
0
100
200
300


Extracted Image 1
Extracted Image 2
0
50
100
150
200
250
300
350
0
100
200
300


FFT of Image (13x13 Low pass)
0
50
100
150
200
250
300
350
0
100
200
300


Extracted Image 1
Extracted Image 2
0
50
100
150
200
250
300
350
0
100
200
300