CPSC425: Assignment 1 (Python v2.7.15)

Alec Xu
38108130 (p7g9)

January 28, 2020

Part 1

In this question, you will be practicing filtering by hand on the following image. You will enter your final result for each question in the provided empty tables.

1	0	0	0	1
2	3	0	8	0
2	0	0	0	3
0	0	1	0	0

Note that you do not have to fill all the cells. If a cell contains a number that is not an integer, enter it as a fraction.

Question (1a)

Apply the correlation filter to the image with **no padding**.

0	0	1
0	0	0
-1	0	0

Enter your final result here in integer or fraction:

	-2	0	1	
	0	8	-1	

Question (1b)

Apply the convolution filter to the image with **no padding**.

0	0	1
0	0	0
-1	0	0

Enter your final result here in integer or fraction:

	2	0	-1	
	0	-8	1	

Question (1c)

Apply the filter to the image with **zero padding** (zero padding means to pad your image with zeros; it is not the same as "no padding").

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Enter your final result here in integer or fraction:

1/9*

6	6	11	9	9
8	8	11	12	12
7	8	12	12	11
2	3	1	4	3

Part 2

NOTE: Testing scripts included in appendix at end

Question (1)

Output in python shell for box filter

```
n = 3
[[ 0.11111111  0.11111111  0.11111111]
 [ 0.11111111  0.11111111  0.11111111]
 [ 0.11111111  0.11111111  0.11111111]]
n = 4
Dimension "n" must be odd
n = 5
[[ 0.04  0.04  0.04  0.04  0.04]
 [ 0.04  0.04  0.04  0.04  0.04]
 [ 0.04  0.04  0.04  0.04  0.04]
 [ 0.04  0.04  0.04  0.04  0.04]
 [ 0.04  0.04  0.04  0.04  0.04]]
```

Question (2)

Output of 1D Gaussian for different sigmas rounded to 4 sig figs.

```
sigma = 0.3
[ 0.0038  0.9923  0.0038]
sigma = 0.5
[ 0.1065  0.787   0.1065]
sigma = 1
[ 0.0044  0.054   0.242   0.3991  0.242   0.054   0.0044]
sigma = 2
[ 0.0022  0.0088  0.027   0.0648  0.1211  0.1762  0.1997  0.1762  0.1211
  0.0648  0.027   0.0088  0.0022]
```

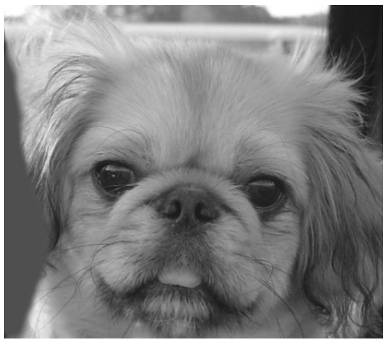
Question (3)

Output of 2D Gaussian for different sigmas rounded to 4 sig figs.

```
sigma = 0.5
[[ 0.0113  0.0838  0.0113]
 [ 0.0838  0.6193  0.0838]
 [ 0.0113  0.0838  0.0113]]
sigma = 1
[[ 0.      0.0002  0.0011  0.0018  0.0011  0.0002  0.    ]
 [ 0.0002  0.0029  0.0131  0.0216  0.0131  0.0029  0.0002]
 [ 0.0011  0.0131  0.0586  0.0966  0.0586  0.0131  0.0011]
 [ 0.0018  0.0216  0.0966  0.1592  0.0966  0.0216  0.0018]
 [ 0.0011  0.0131  0.0586  0.0966  0.0586  0.0131  0.0011]
 [ 0.0002  0.0029  0.0131  0.0216  0.0131  0.0029  0.0002]
 [ 0.      0.0002  0.0011  0.0018  0.0011  0.0002  0.    ]]
```

Question (4)

Self-implemented Gaussian blur filter applied to image of a dog.


(a) Original dog

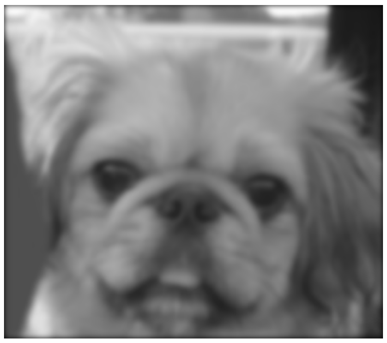
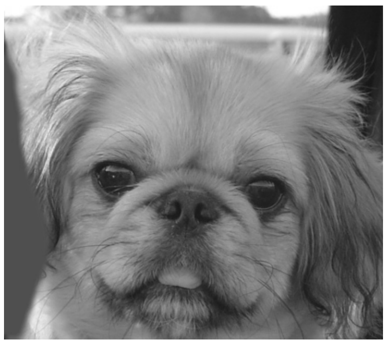

(b) Gaussian filtered dog

Figure 1: Dog image and output of dog image through self-implemented Gaussian filter

Question (5)

Scipy Gaussian blur filter applied to image of a dog.


(a) Original dog

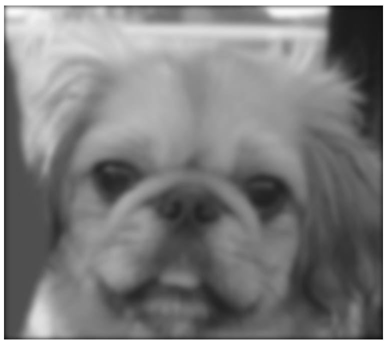

(b) Gaussian filtered dog

Figure 2: Dog image and output of dog image through Scipy Gaussian filter

Question (6)

Manual convolve run time: 16.2030000687
Scipy convolve run time: 0.569000005722

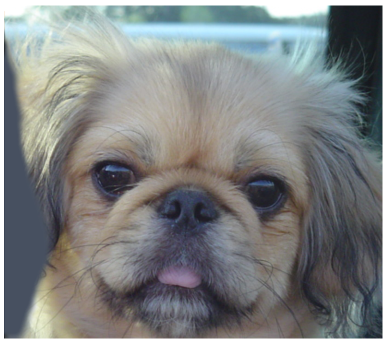
Question (7)

As the gaussian filter is separable, it would be more efficient to separate the filter into X and Y components and pass them separately through the image. This is due to the number of multiplications scaling O($2k$) using two 1D arrays of length k, vs O(k^2) using one 2D array of length k

Part 3

Question (1)

Scipy Gaussian blur filter applied to image of a dog in colour.


(a) Original dog

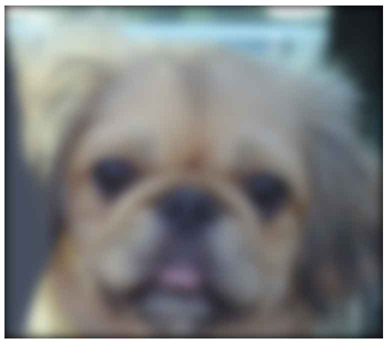
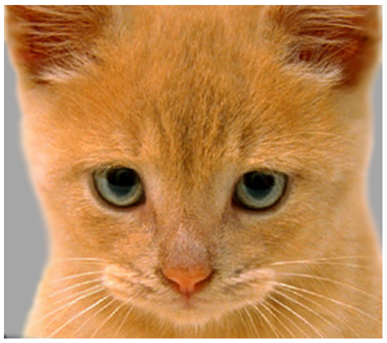

(b) Gaussian filtered dog

Figure 3: Dog image and output of dog image through Scipy Gaussian filter in colour

Question (2)

High frequency image constructed from paired cat image, with 128 added for visualization


(a) Original cat

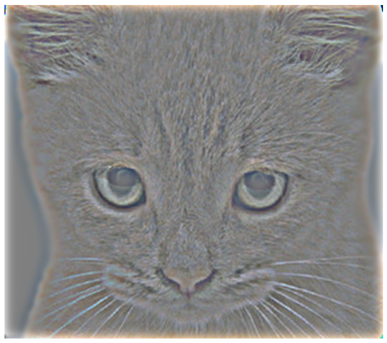

(b) High-pass cat

Figure 4: Cat image and high-pass cat by subtracting gaussian cat and adding 128

Question (3)

Constructed hybrid images.

Cat and dog hybrid image

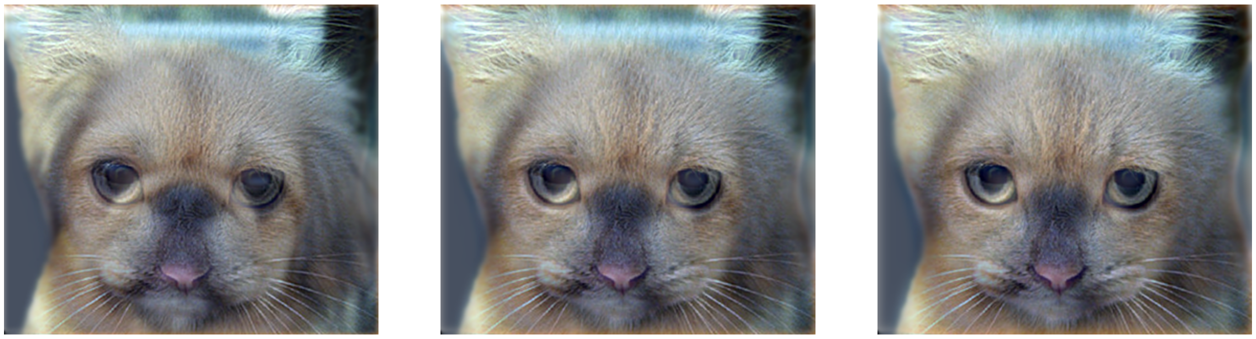


Figure 5: left: $\sigma_1 = 3, \sigma_2 = 3$ middle: $\sigma_1 = 5, \sigma_2 = 5$ right: $\sigma_1 = 7, \sigma_2 = 7$

Fish and Submarine hybrid image

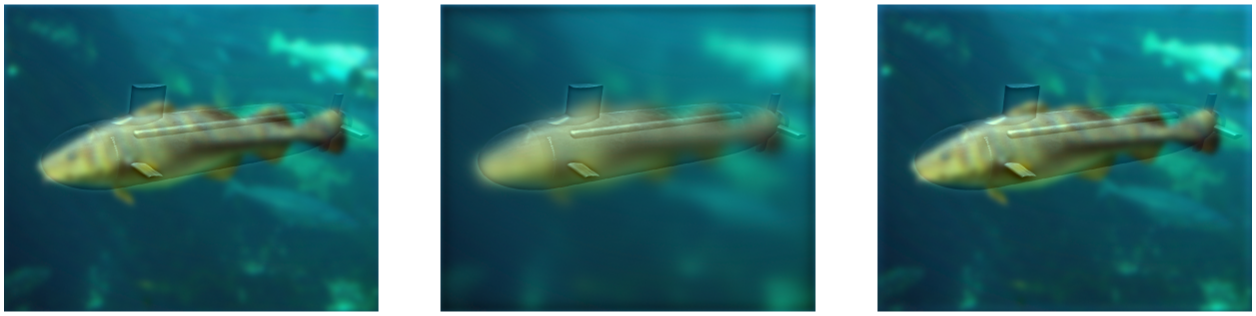


Figure 6: left: $\sigma_1 = 3, \sigma_2 = 3$ middle: $\sigma_1 = 3, \sigma_2 = 7$ right: $\sigma_1 = 5, \sigma_2 = 3$

Einstein and Marilyn hybrid image

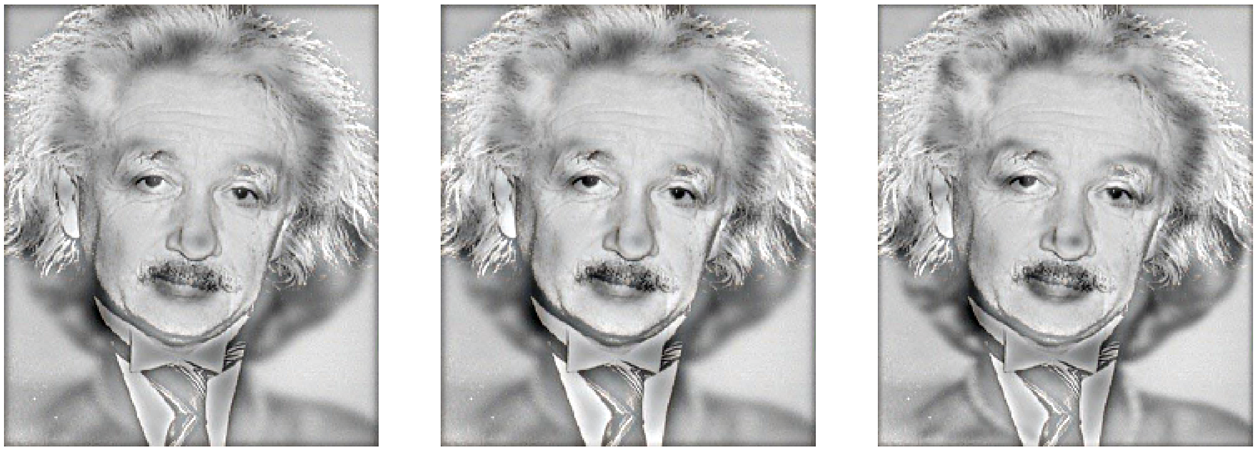


Figure 7: left: $\sigma_1 = 3, \sigma_2 = 3$ middle: $\sigma_1 = 5, \sigma_2 = 3$ right: $\sigma_1 = 3, \sigma_2 = 2$

Part 4

Question 1

Speckle Noise

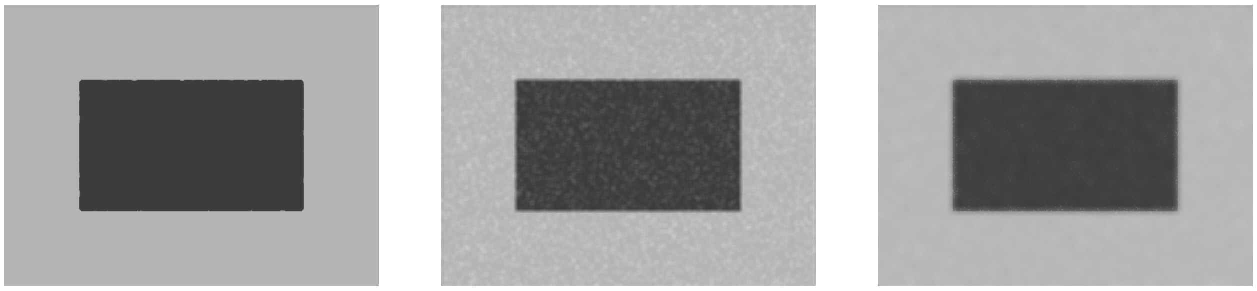


Figure 8: left: median filter, middle: Gaussian filter right: bilateral filter

- Median filter diameter $= 5px$
- Gaussian filter kernel = 5x5, $\sigma_x, \sigma_y = 10$
- Bilateral filter diameter = 15px, $\sigma_{colour} = 110, \sigma_{space} = 30$

Gaussian Noise

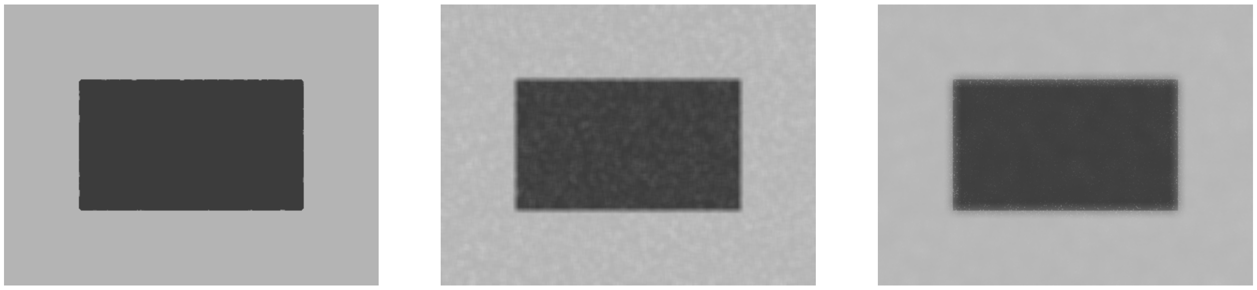


Figure 9: left: median filter, middle: Gaussian filter right: bilateral filter

- Median filter diameter $= 5px$
- Gaussian filter kernel = 7x7, $\sigma_x, \sigma_y = 30$
- Bilateral filter diameter = 20px, $\sigma_{colour} = 100, \sigma_{space} = 80$

Question 2

We apply the given specifications as stated below:

- Median filter diameter $= 7px$
- Gaussian filter kernel = 7x7, $\sigma_x = 50$
- Bilateral filter diameter = 7px, $\sigma_{colour} = 150, \sigma_{space} = 150$

Speckle Noise

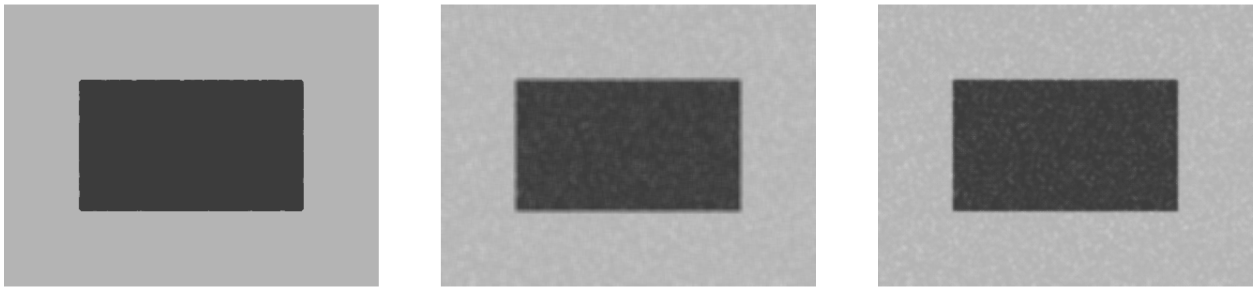


Figure 10: left: median filter, middle: Gaussian filter right: bilateral filter

Gaussian Noise

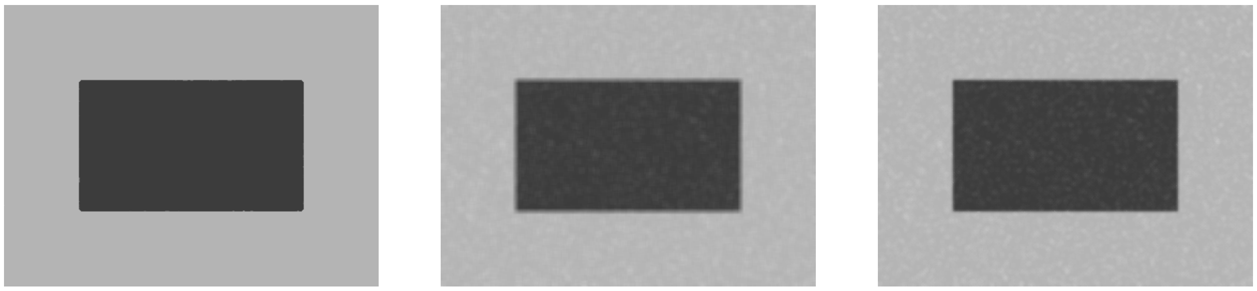


Figure 11: left: median filter, middle: Gaussian filter right: bilateral filter

In both types of noise, the median filter performed the best, although this may be due to the simple geometry. The median filter also introduced jagged edges to the shape in the speck case. The gaussian filter was better at smoothing over the specks, but worse at preserving edges compared to the bilateral filter.

Appendix A: Part 2 Script

```
from hw1Functions import *
import time

print("\nQuestion 1\n")
for i in range(3,6):
    print "n = " + str(i)
    try:
        print boxfilter(i)
    except AssertionError as error:
        print(error)
        continue

print("\nQuestion 2\n")
for i in [0.3, 0.5, 1, 2]:
    print "sigma = " + str(i)
    try:
        print np.round(gauss1d(i),4)
    except AssertionError as error:
        print error
        continue

print("\nQuestion 3\n")
for i in [0.5, 1]:
    print "sigma = " + str(i)
    try:
        print np.round(gauss2d(i),4)
    except AssertionError as error:
        print error
        continue

print("\nQuestion 4\n")

im = Image.open('dog.jpg').convert('L')
imArray = np.asarray(im, dtype=np.float32)

q4Start = time.time()
im2 = Image.fromarray(gaussconvolve2d_manual(imArray,3))
q4Time = time.time() - q4Start
# im.show()
# im2.show()
im.convert('RGB').save('.\\results\\p2q4a.png','PNG')
im2.convert('RGB').save('.\\results\\p2q4b.png','PNG')
raw_input("press any key to continue...")
```

```
print("\nQuestion 5\n")
q5Start = time.time()
im3 = Image.fromarray(gaussconvolve2d_scipy(imArray,3))
q5Time = time.time() - q5Start
# im.show()
# im3.show()
im.convert('RGB').save('.\\results\\p2q5a.png','PNG')
im3.convert('RGB').save('.\\results\\p2q5b.png','PNG')
# The reason why correlation and convolution are the same here
# is because correlation is the same as convolution with the matrix
# flipped. As the gaussian kernel is rotationally invariant
# It makes no difference.

print("\nQuestion 6\n")
print "Manual convolve run time: " + str(q4Time)
print "Scipy convolve run time: " + str(q5Time)

# Q7

# As the gaussian filter is separable, it would be more efficient to
# separate the filter into X and Y components and pass them separately through
# the image. This is due to the number of multiplications scaling O(2k)
# using two 1D arrays of length k, vs O(k^2) using one 2D array of length k
```

Appendix B: Part 3 Script

```
from hw1Functions import *

print("\nQuestion 1\n")
# Show the gaussian blurred version of the image with sigma = 6
sigma = 6
imPath = '.\\part3images\\0b_dog.bmp'
arrBlur = gaussFilterColour(imPath, sigma)

imBlur = Image.fromarray(np.uint8(arrBlur))
# Image.open(imPath).show()
Image.open(imPath).convert('RGB').save('.\\results\\p3q1a.png','PNG')
# imBlur.show()
imBlur.convert('RGB').save('.\\results\\p3q1b.png','PNG')

raw_input("press any key to continue...")

print("\nQuestion 2\n")
```

```
imPath = '.\\part3images\\0a_cat.bmp'
arrBlur = gaussFilterColour(imPath, sigma)

# Show the high-pass filtered version of the image, adding 128 for visualization
arrImage = np.asarray(Image.open(imPath), dtype=np.float32)
arrHigh =  arrImage - arrBlur + 128
imHigh = Image.fromarray(np.uint8(arrHigh))
# imHigh.show()
imHigh.convert('RGB').save('.\\results\\p3q2.png','PNG')
raw_input("press any key to continue...")

print("\nQuestion 3\n")

# Cat dog
# Show most interesting combinations found for hybrid images
imHybrid01 = gaussHybridize('.\\part3images\\0a_cat.bmp', '.\\part3images\\0b_dog.bmp' ,
# Image.fromarray(imHybrid01).show()
Image.fromarray(imHybrid01).convert('RGB').save('.\\results\\p3q3a.png','PNG')
imHybrid02 = gaussHybridize('.\\part3images\\0a_cat.bmp', '.\\part3images\\0b_dog.bmp' ,
# Image.fromarray(imHybrid02).show()
Image.fromarray(imHybrid02).convert('RGB').save('.\\results\\p3q3b.png','PNG')
imHybrid03 = gaussHybridize('.\\part3images\\0a_cat.bmp', '.\\part3images\\0b_dog.bmp' ,
# Image.fromarray(imHybrid03).show()
Image.fromarray(imHybrid03).convert('RGB').save('.\\results\\p3q3c.png','PNG')

# fish sub
imHybrid11 = gaussHybridize('.\\part3images\\3b_sub.bmp','.\\part3images\\3a_fish.bmp' ,
# Image.fromarray(imHybrid11).show()
Image.fromarray(imHybrid11).convert('RGB').save('.\\results\\p3q3d.png','PNG')
imHybrid12 = gaussHybridize('.\\part3images\\3b_sub.bmp','.\\part3images\\3a_fish.bmp' ,
# Image.fromarray(imHybrid12).show()
Image.fromarray(imHybrid12).convert('RGB').save('.\\results\\p3q3e.png','PNG')
imHybrid13 = gaussHybridize('.\\part3images\\3b_sub.bmp','.\\part3images\\3a_fish.bmp' ,
# Image.fromarray(imHybrid13).show()
Image.fromarray(imHybrid13).convert('RGB').save('.\\results\\p3q3f.png','PNG')


# einstein marilyn
imHybrid21 = gaussHybridize('.\\part3images\\2a_einstein.bmp','.\\part3images\\2b_marily
# Image.fromarray(imHybrid21).show()
Image.fromarray(imHybrid21).convert('RGB').save('.\\results\\p3q3g.png','PNG')
imHybrid22 = gaussHybridize('.\\part3images\\2a_einstein.bmp','.\\part3images\\2b_marily
# Image.fromarray(imHybrid22).show()
Image.fromarray(imHybrid22).convert('RGB').save('.\\results\\p3q3h.png','PNG')
```

```
imHybrid23 = gaussHybridize('.\\part3images\\2a_einstein.bmp','.\\part3images\\2b_marily
# Image.fromarray(imHybrid23).show()
Image.fromarray(imHybrid23).convert('RGB').save('.\\results\\p3q3i.png','PNG')
```

Appendix C: Part 4 Script

```
from PIL import Image
import numpy as np
import math
from scipy import signal
import cv2

print("\nQuestion 1\n")

arrSpeck = np.asarray(Image.open('.\\part4images\\box_speckle.png'), dtype=np.float32)
arrGauss = np.asarray(Image.open('.\\part4images\\box_gauss.png'), dtype=np.float32)

print("\nSpeck Noise filters\n")
speckFilteredMedian = cv2.medianBlur(arrSpeck, 5)
# Image.fromarray(speckFilteredMedian).show()
Image.fromarray(speckFilteredMedian).convert('RGB').save('.\\results\\p4q1a.png','PNG')

speckFilteredGauss = cv2.GaussianBlur(arrSpeck, ksize=(5,5), sigmaX=10, sigmaY=10)
# Image.fromarray(speckFilteredGauss).show()
Image.fromarray(speckFilteredGauss).convert('RGB').save('.\\results\\p4q1b.png','PNG')

speckFilteredBi = cv2.bilateralFilter(arrSpeck,15,110,30)
# Image.fromarray(speckFilteredBi).show()
Image.fromarray(speckFilteredBi).convert('RGB').save('.\\results\\p4q1c.png','PNG')

print("\nGaussian Noise filters\n")
GaussFilteredGauss = cv2.GaussianBlur(arrSpeck, ksize=(7,7), sigmaX= 30, sigmaY = 30)
# Image.fromarray(GaussFilteredGauss).show()
Image.fromarray(GaussFilteredGauss).convert('RGB').save('.\\results\\p4q1d.png','PNG')

GaussFilteredMedian = cv2.medianBlur(arrSpeck, 5)
# Image.fromarray(GaussFilteredMedian).show()
Image.fromarray(GaussFilteredMedian).convert('RGB').save('.\\results\\p4q1e.png','PNG')

GaussFilteredBi = cv2.bilateralFilter(arrSpeck,20,100,80)
# Image.fromarray(GaussFilteredBi).show()
Image.fromarray(GaussFilteredBi).convert('RGB').save('.\\results\\p4q1f.png','PNG')

raw_input("press any key to continue...")
```

```
print("\nQuestion 2\n")

print("\nSpeck Noise filters\n")
SpeckFilteredGauss = cv2.GaussianBlur(arrSpeck, ksize=(7, 7), sigmaX=50)
# Image.fromarray(SpeckFilteredGauss).show()
Image.fromarray(SpeckFilteredGauss).convert('RGB').save('.\\results\\p4q2a.png','PNG')

SpeckFilteredMedian = cv2.medianBlur(arrSpeck,5)
# Image.fromarray(SpeckFilteredMedian).show()
Image.fromarray(SpeckFilteredMedian).convert('RGB').save('.\\results\\p4q2b.png','PNG')

SpeckFilteredBi = cv2.bilateralFilter(arrSpeck, 7, sigmaColor=150, sigmaSpace=150)
# Image.fromarray(SpeckFilteredBi).show()
Image.fromarray(SpeckFilteredBi).convert('RGB').save('.\\results\\p4q2c.png','PNG')

print("\nGaussian Noise Filters\n")
GaussFilteredGauss = cv2.GaussianBlur(arrGauss, ksize=(7, 7), sigmaX=50)
# Image.fromarray(GaussFilteredGauss).show()
Image.fromarray(GaussFilteredGauss).convert('RGB').save('.\\results\\p4q2d.png','PNG')

GaussFilteredMedian = cv2.medianBlur(arrGauss,5)
# Image.fromarray(GaussFilteredMedian).show()
Image.fromarray(GaussFilteredMedian).convert('RGB').save('.\\results\\p4q2e.png','PNG')

GaussFilteredBi = cv2.bilateralFilter(arrGauss, 7, sigmaColor=150, sigmaSpace=150)
# Image.fromarray(GaussFilteredBi).show()
Image.fromarray(GaussFilteredBi).convert('RGB').save('.\\results\\p4q2f.png','PNG')

# In both types of noise, the median filter performed the best, although this may be due
# to the simple geometry. The median filter also introduced jagged edges to the shape.
# The gaussian filter was better at smoothing over the specks, but worse at preserving
# edges compared to the bilateral filter.
```