In [1]:
```
from PIL import Image
import numpy as np
import math
from scipy import signal
import cv2
import matplotlib.pyplot as plt
import time
```

Part 2: Gaussian Filtering

Question 2.1

In [2]:
```
def boxfilter(n):
    try:
        assert n % 2 == 1
        array = np.ones((n,n))
        array = array / (n*n)
        return array
    except AssertionError as error:
        print("AssertionError: Dimension must be odd, dimension given is {}".f
ormat(n))

print(boxfilter(3))
boxfilter_4 = boxfilter(4)
print(boxfilter(5))
```

```
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
AssertionError: Dimension must be odd, dimension given is 4
[[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 2.2

In [3]:
```
def gauss1d(sigma):
    # filter length is sigma*6 rounded up to the next odd int
    filter_len = round(sigma * 6)
    if filter_len % 2 == 0:
        filter_len += 1

    # create 1D array, where x is distance away from center
    filter = np.arange(start=-np.floor(filter_len/2), stop=np.ceil(filter_len/
2))

    # pass array through gaussian density function
    filter = np.exp(-filter**2 / (2*sigma**2))

    # normalize and return
    filter = filter/np.sum(filter)
    return filter

print(gauss1d(0.3))
print(gauss1d(0.5))
print(gauss1d(1))
print(gauss1d(2))
```

```
[0.00383626 0.99232748 0.00383626]
[0.10650698 0.78698604 0.10650698]
[0.00443305 0.05400558 0.24203623 0.39905028 0.24203623 0.05400558
 0.00443305]
[0.0022182  0.00877313 0.02702316 0.06482519 0.12110939 0.17621312
 0.19967563 0.17621312 0.12110939 0.06482519 0.02702316 0.00877313
 0.0022182 ]
```

Question 2.3

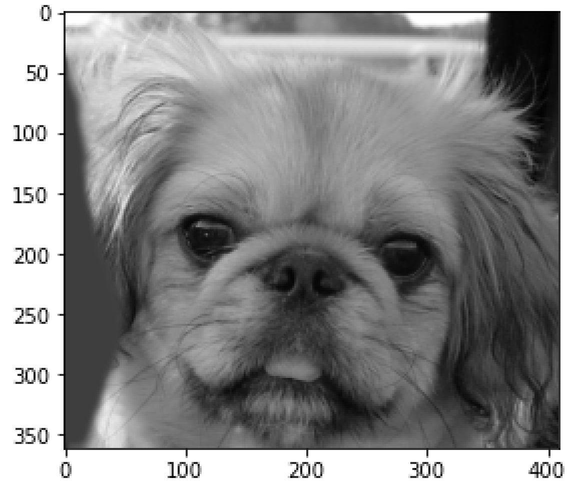
In [4]:
```
def gauss2d(sigma):
    # create 1D gaussian
    filter1d = gauss1d(sigma)[:,np.newaxis]
    # create 2D gaussian by convolution of 1D gaussian w/ its transpose
    filter2d = signal.convolve2d(filter1d, filter1d.T)
    return filter2d

print(gauss2d(0.5))
print(gauss2d(1))
```

```
[[0.01134374 0.08381951 0.01134374]
 [0.08381951 0.61934703 0.08381951]
 [0.01134374 0.08381951 0.01134374]]
[[1.96519161e-05 2.39409349e-04 1.07295826e-03 1.76900911e-03
  1.07295826e-03 2.39409349e-04 1.96519161e-05]
 [2.39409349e-04 2.91660295e-03 1.30713076e-02 2.15509428e-02
  1.30713076e-02 2.91660295e-03 2.39409349e-04]
 [1.07295826e-03 1.30713076e-02 5.85815363e-02 9.65846250e-02
  5.85815363e-02 1.30713076e-02 1.07295826e-03]
 [1.76900911e-03 2.15509428e-02 9.65846250e-02 1.59241126e-01
  9.65846250e-02 2.15509428e-02 1.76900911e-03]
 [1.07295826e-03 1.30713076e-02 5.85815363e-02 9.65846250e-02
  5.85815363e-02 1.30713076e-02 1.07295826e-03]
 [2.39409349e-04 2.91660295e-03 1.30713076e-02 2.15509428e-02
  1.30713076e-02 2.91660295e-03 2.39409349e-04]
 [1.96519161e-05 2.39409349e-04 1.07295826e-03 1.76900911e-03
  1.07295826e-03 2.39409349e-04 1.96519161e-05]]
```

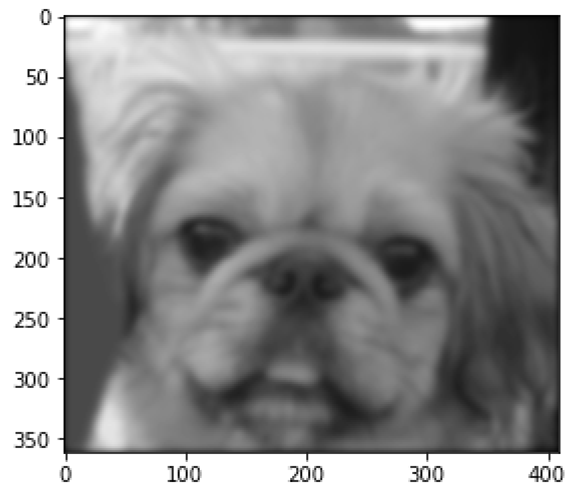
Question 2.4

In [5]:
```
def convolve2d_manual(array, filter):
    # Output image
    image_out = np.zeros_like(array)

    # Add zero padding to the input image
    padding = int(np.floor(filter.shape[0]/2))
    image_padded = np.zeros((array.shape[0] + 2*padding, array.shape[1] + 2*pa
dding))
    image_padded[padding:-padding, padding:-padding] = array

    # Loop through each neighbourhood and calculate new pixel value
    filtersz = filter.shape[0]
    for i in range(array.shape[1]):
        for j in range(array.shape[0]):
            image_out[j, i] = np.sum(filter * image_padded[j:j+filtersz, i:i+f
iltersz])
    return image_out

def gaussconvolve2d_manual(array,sigma):
    # Create 2D gaussian filter and apply convolution
    filter = gauss2d(sigma)
    filtered_image = convolve2d_manual(array, filter)
    return filtered_image

# Load image and convert to grayscale
coloured_image = cv2.imread("images/dog.jpg")
grey_image = cv2.cvtColor(coloured_image, cv2.COLOR_BGR2GRAY)
filtered_image = gaussconvolve2d_manual(grey_image, 3)

print("Original image:")
plt.imshow(grey_image, cmap="gray")
plt.show()
print("Filtered image:")
plt.imshow(filtered_image, cmap="gray")
plt.show()
```

```
Original image:
```




```
Filtered image:
```

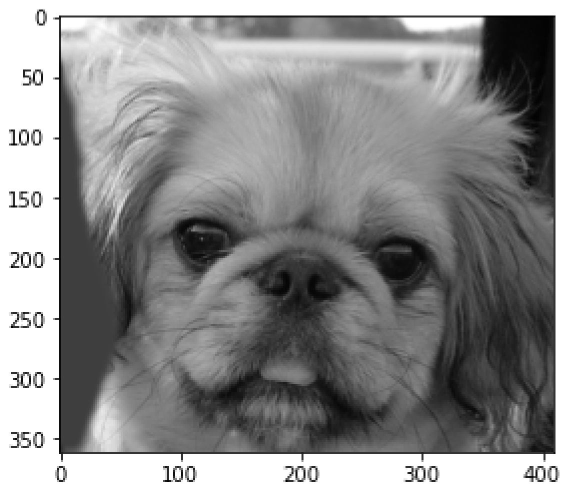


Question 2.5

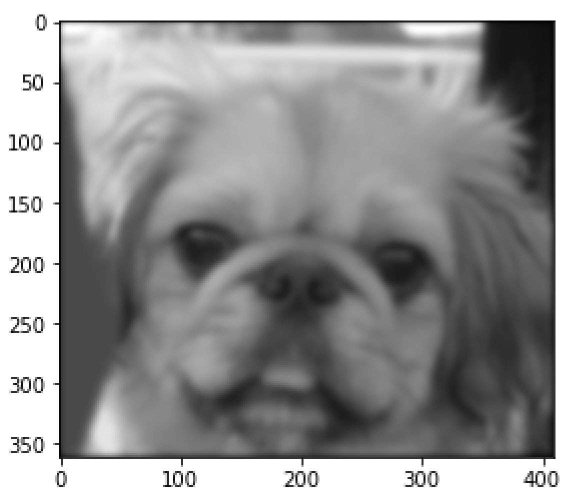
In this case, correlate2d and convolve2d return the same results because our 2D Gaussian filter is symmetric both horizontally and vertically. If the filter were not symmetric, the two functions would not return the same results.

In [6]:
```
def gaussconvolve2d_scipy(array,sigma):
    filter = gauss2d(sigma)
    filtered_img = signal.convolve2d(array,filter,'same')
    return filtered_img

# Load image and convert to grayscale
coloured_image = cv2.imread("images/dog.jpg")
grey_image = cv2.cvtColor(coloured_image, cv2.COLOR_BGR2GRAY)
filtered_image = gaussconvolve2d_scipy(grey_image, 3)

print("Original image:")
plt.imshow(grey_image, cmap="gray")
plt.show()
print("Filtered image:")
plt.imshow(filtered_image, cmap="gray")
plt.show()
```

```
Original image:
```




```
Filtered image:
```



Question 2.6

The SciPy implementation is clearly faster when sigma=10. It may be possible that the scipy implementation uses various convolution speed-up techniques such as taking the logarithm so that multiplications become additions, or using Fourier transforms to reduce convolutions to complex multiplication. My manual implementation does not implement any of these techniques, and hence is likely why it runs slower.

In [7]:
```python
# Load image and convert to grayscale
coloured_image = cv2.imread("images/dog.jpg")
grey_image = cv2.cvtColor(coloured_image, cv2.COLOR_BGR2GRAY)

start_time = time.time() # start timestamp
filtered_image = gaussconvolve2d_manual(grey_image, 10)
duration_manual = time.time() - start_time # duration in seconds

start_time = time.time() # start timestamp
filtered_image = gaussconvolve2d_scipy(grey_image, 10)
duration_scipy = time.time() - start_time # duration in seconds

print("Manual implementation runtime: {} seconds".format(duration_manual))
print("SciPy implementation runtime: {} seconds".format(duration_scipy))
```

```
Manual implementation runtime: 2.521883487701416 seconds
SciPy implementation runtime: 1.7840995788574219 seconds
```

Question 2.7

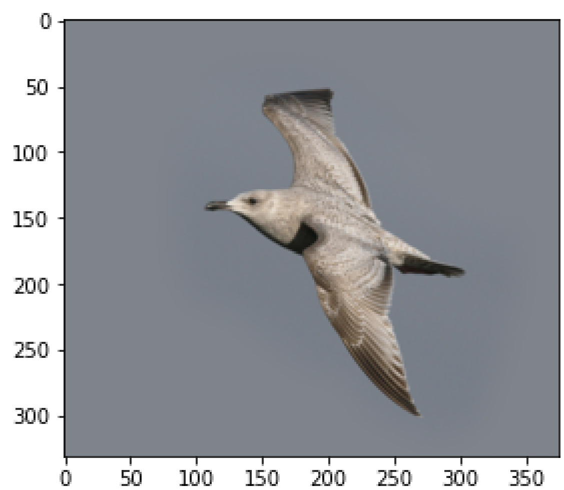
An implementation of the 2D Gaussian filter using convolution would be more efficient, because we can use 1D convolutions instead of 2D convolutions. We can first convolve each row with a 1D filter, and then convolve each column with a 1D filter. This works because the 2D Gaussian filter can be expressed as an outer product of two 1D filters (one as a function of x, the other as a function of y).

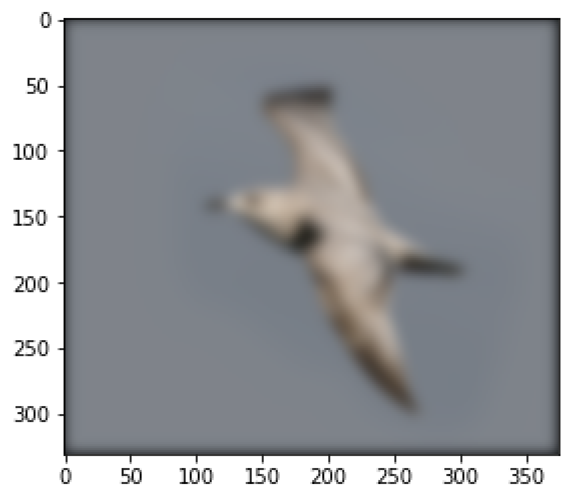
The naive implementation would need m^2 *n^2 multiplications, where m is the size of the window and n is the size of the image. A separable implementation only needs 2m* n^2 multiplications.

Part 3: Hybrid Images

Question 3.1

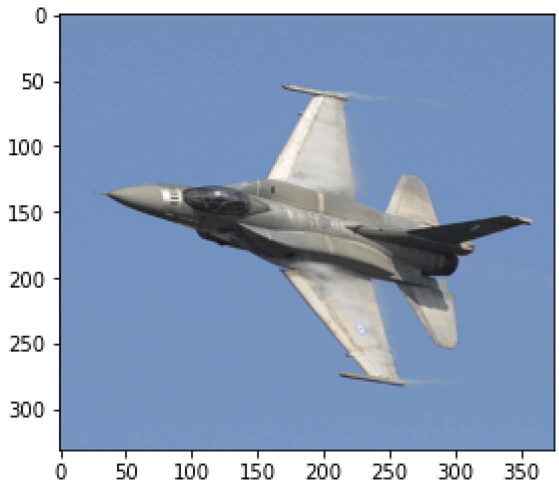
In [8]:
```
im_a = cv2.imread("images/4a_bird.bmp")
im_a = cv2.cvtColor(im_a, cv2.COLOR_BGR2RGB)
plt.imshow(im_a)
plt.show()

def blur_color_img(image, sigma):
    filtered_channels = []
    # Split image into RGB channels
    red, green, blue = cv2.split(image)
    # Apply Gaussian blur filter to each channel separately
    for channel in [red, green, blue]:
        filtered_channel = gaussconvolve2d_scipy(channel, sigma)
        filtered_channels.append(filtered_channel.astype(int))
    # Merge channels back together and return filtered image
    filtered_color_img = cv2.merge(filtered_channels)
    return filtered_color_img

low_freq_im_a = blur_color_img(im_a, sigma=5)
plt.imshow(low_freq_im_a)
plt.show()
```

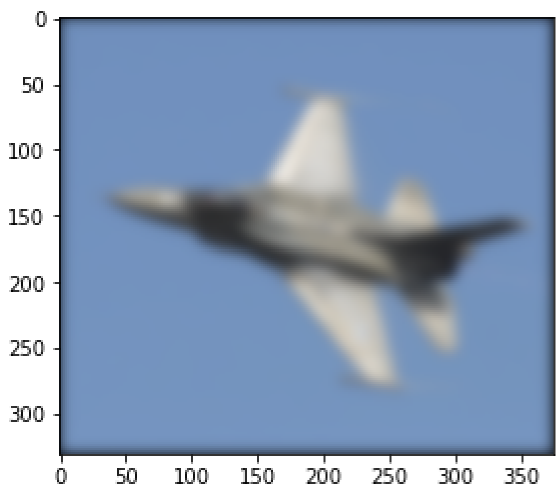


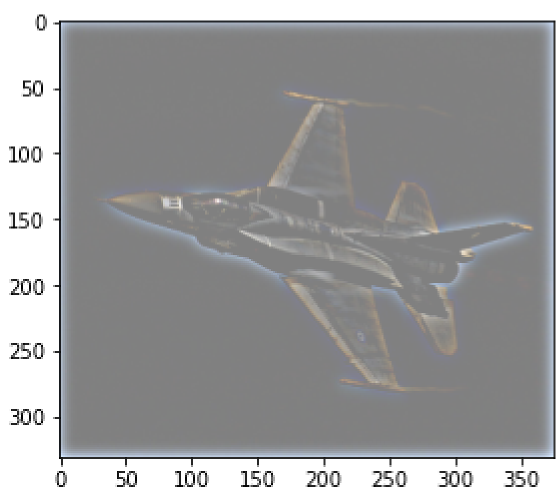


Question 3.2

In [9]:
```
im_b = cv2.imread("images/4b_plane.bmp")
im_b = cv2.cvtColor(im_b, cv2.COLOR_BGR2RGB)
plt.imshow(im_b)
plt.show()

low_freq_im_b = blur_color_img(im_b, sigma=5)
plt.imshow(low_freq_im_b)
plt.show()

high_freq_im = np.clip(im_b - low_freq_im_b, 0, 255)
plt.imshow(np.clip(high_freq_im + 128, 0, 255))
plt.show()
```



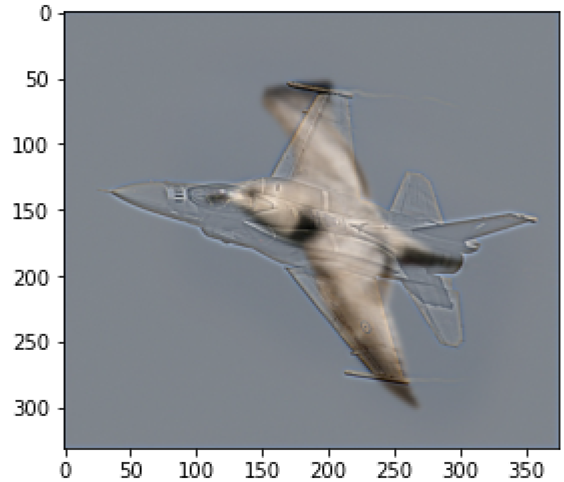




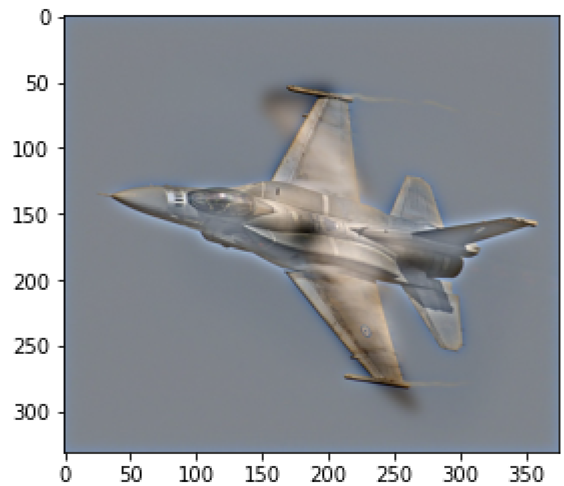
Question 3.3

In [10]:
```
def merge_images(im1_path, im2_path, sigma):
    im_a = cv2.imread(im1_path)
    im_a = cv2.cvtColor(im_a, cv2.COLOR_BGR2RGB)
    im_b = cv2.imread(im2_path)
    im_b = cv2.cvtColor(im_b, cv2.COLOR_BGR2RGB)

    low_freq_im_a = blur_color_img(im_a, sigma)
    low_freq_im_b = blur_color_img(im_b, sigma)
    high_freq_im = im_b - low_freq_im_b
    hybrid_image = np.clip(high_freq_im + low_freq_im_a, 0, 255)
    return hybrid_image

for sigma in [2,5,8]:
    print("Bird and plane at sigma={}".format(sigma))
    hybrid_image = merge_images("images/4a_bird.bmp", "images/4b_plane.bmp", s
igma)
    plt.imshow(hybrid_image)
    plt.show()

for sigma in [2,5,8]:
    print("Cat and dog at sigma={}".format(sigma))
    hybrid_image = merge_images("images/0b_dog.bmp", "images/0a_cat.bmp", sigm
a)
    plt.imshow(hybrid_image)
    plt.show()

for sigma in [2,5,8]:
    print("Bike and motorcycle at sigma={}".format(sigma))
    hybrid_image = merge_images("images/3a_fish.bmp", "images/3b_submarine.bm
p", sigma)
    plt.imshow(hybrid_image)
    plt.show()

for sigma in [2,5,8]:
    print("Einstein and Marilyn at sigma={}".format(sigma))
    hybrid_image = merge_images("images/2a_einstein.bmp", "images/2b_marilyn.b
mp", sigma)
    plt.imshow(hybrid_image)
    plt.show()
```

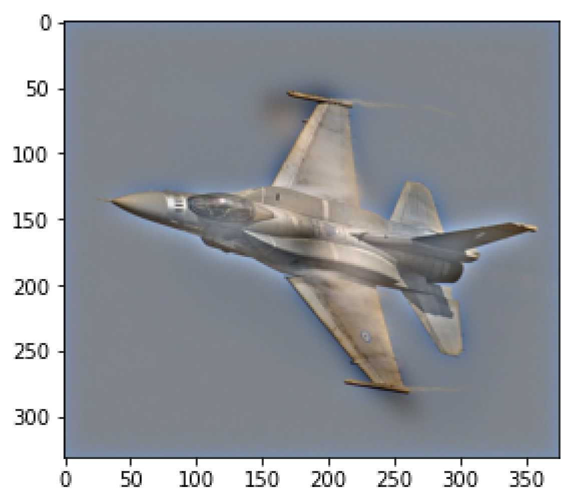
```
Bird and plane at sigma=2
```



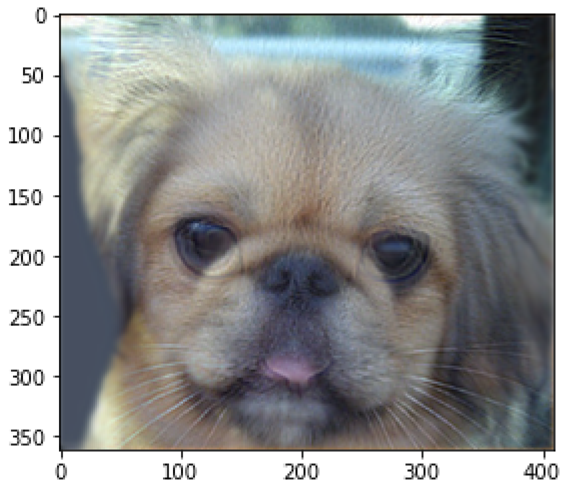

```
Bird and plane at sigma=5
```



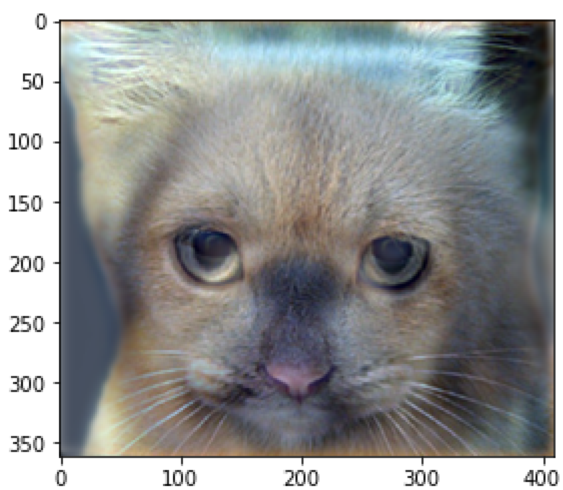

```
Bird and plane at sigma=8
```



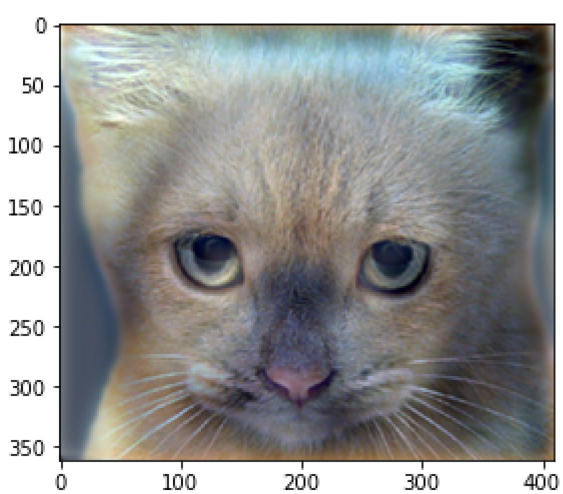

```
Cat and dog at sigma=2
```



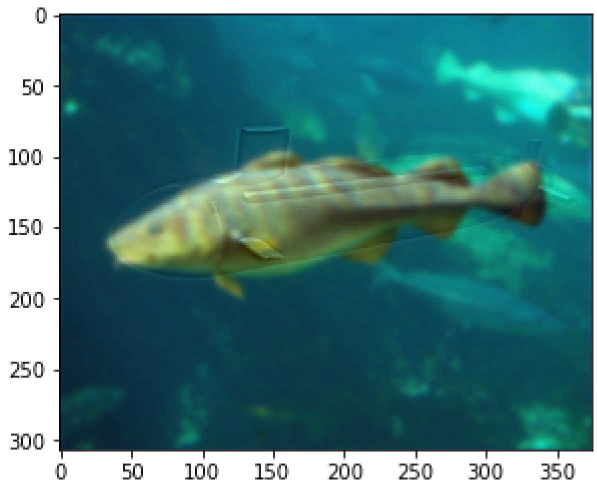
Cat and dog at sigma=5



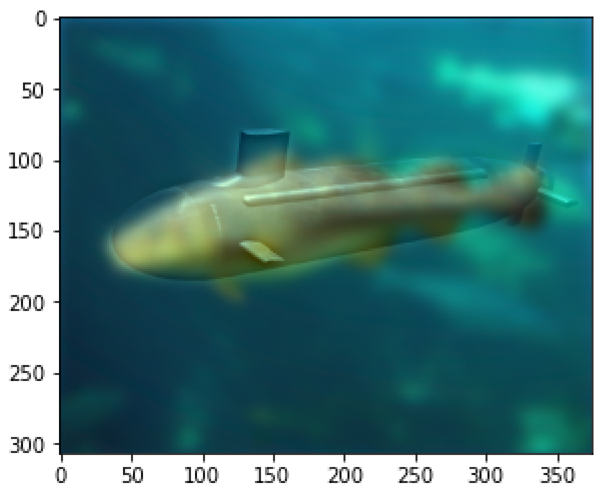
Cat and dog at sigma=8



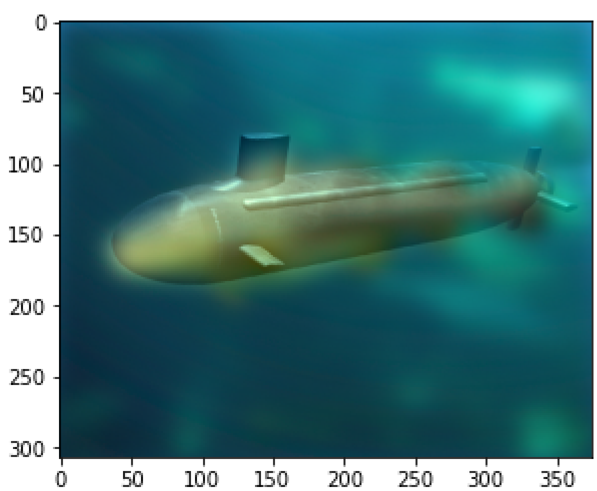
Bike and motorcycle at sigma=2



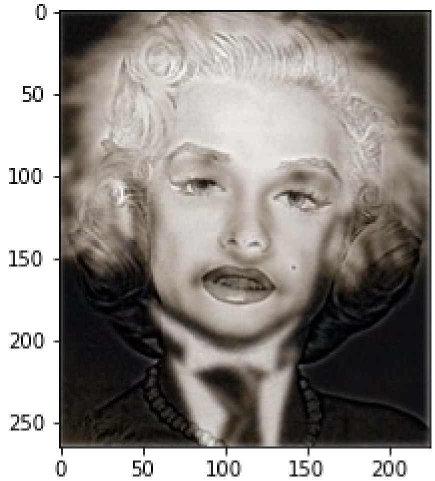
Bike and motorcycle at sigma=5



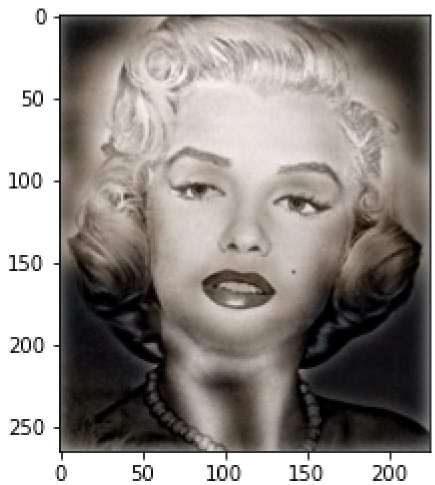
Bike and motorcycle at sigma=8



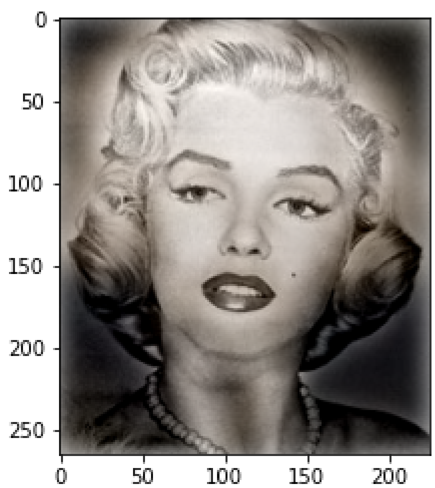
Einstein and Marilyn at sigma=2



Einstein and Marilyn at sigma=5



Einstein and Marilyn at sigma=8

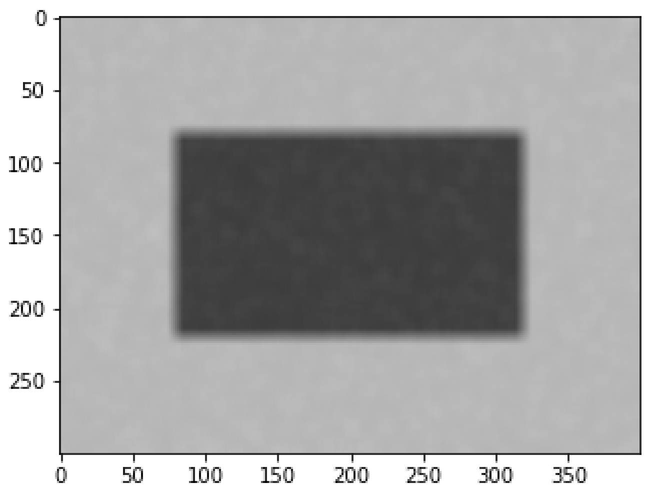


Part 4: Playing with Different Denoising Filters

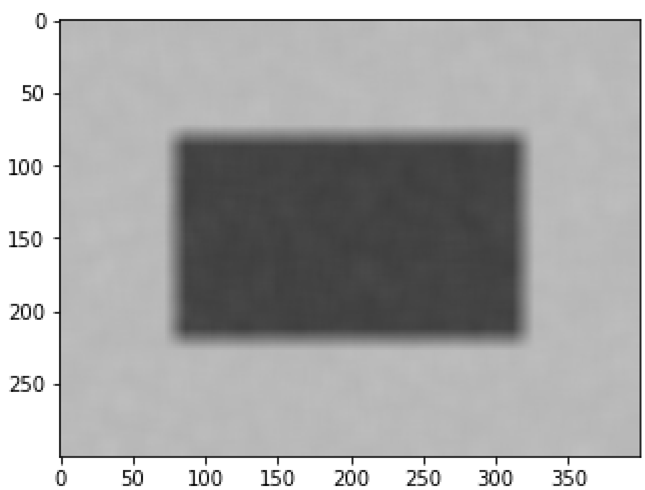
Question 4.1

In [11]:
```
box_gauss = cv2.imread("images/box_gauss.png")
box_speckle = cv2.imread("images/box_speckle.png")

print("GaussianBlur, box_gauss: ksize=11, sigma=10")
plt.imshow(cv2.GaussianBlur(box_gauss, ksize=(11,11), sigmaX=10, sigmaY=10))
plt.show()
print("GaussianBlur, box_speckle: ksize=15, sigma=10")
plt.imshow(cv2.GaussianBlur(box_speckle, ksize=(15,15), sigmaX=10, sigmaY=10))
plt.show()
print("BilateralFilter, box_gauss: d=25, sigma=225")
plt.imshow(cv2.bilateralFilter(box_gauss, d=25, sigmaColor=225, sigmaSpace=225
))
plt.show()
print("BilateralFilter, box_speckle: d=25, sigma=300")
plt.imshow(cv2.bilateralFilter(box_speckle, d=25, sigmaColor=300, sigmaSpace=3
00))
plt.show()
print("MedianBlur, box_gauss: ksize=5")
plt.imshow(cv2.medianBlur(box_gauss, ksize=5))
plt.show()
print("MedianBlur, box_speckle: ksize=7")
plt.imshow(cv2.medianBlur(box_speckle, ksize=7))
plt.show()
```

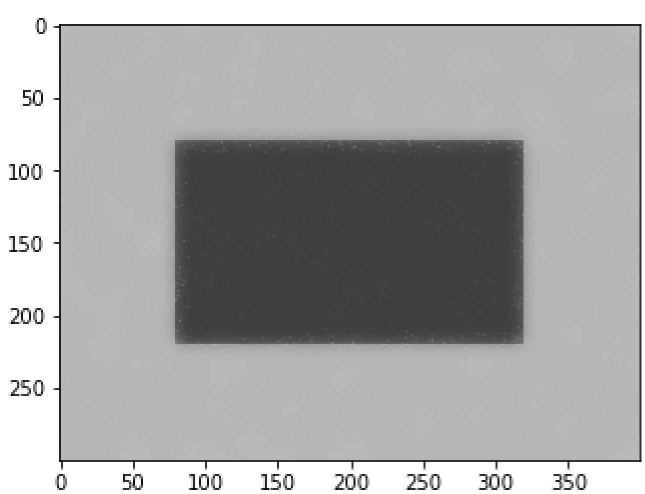
GaussianBlur, box_gauss: ksize=11, sigma=10



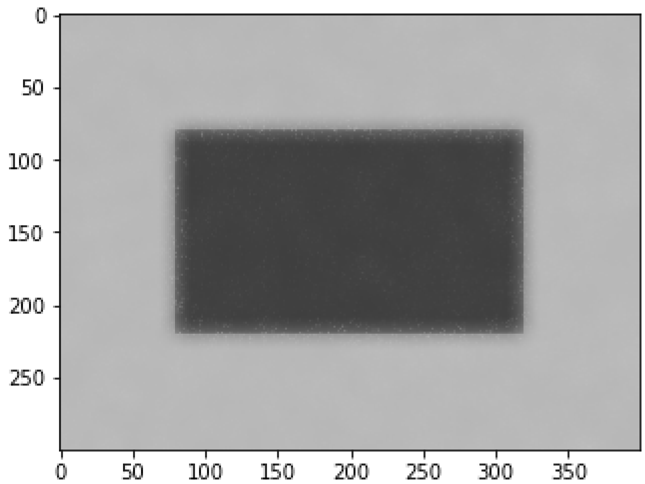
GaussianBlur, box_speckle: ksize=15, sigma=10



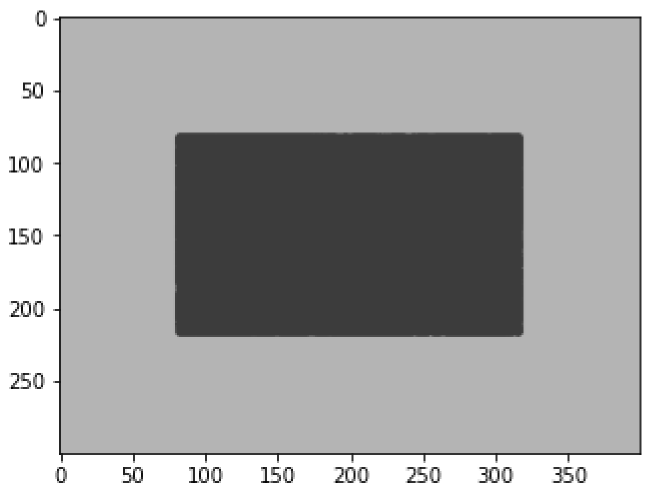
BilateralFilter, box_gauss: d=25, sigma=225



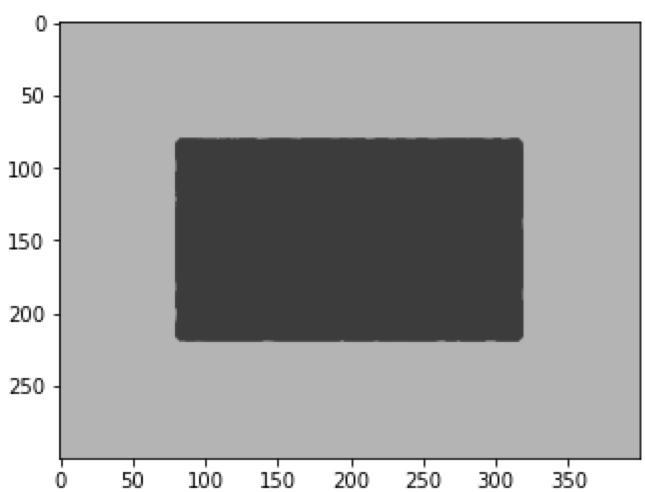
BilateralFilter, box_speckle: d=25, sigma=300



MedianBlur, box_gauss: ksize=5



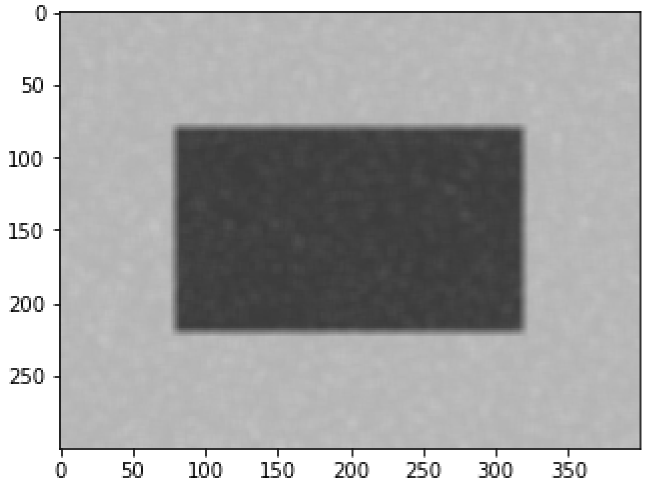
MedianBlur, box_speckle: ksize=7



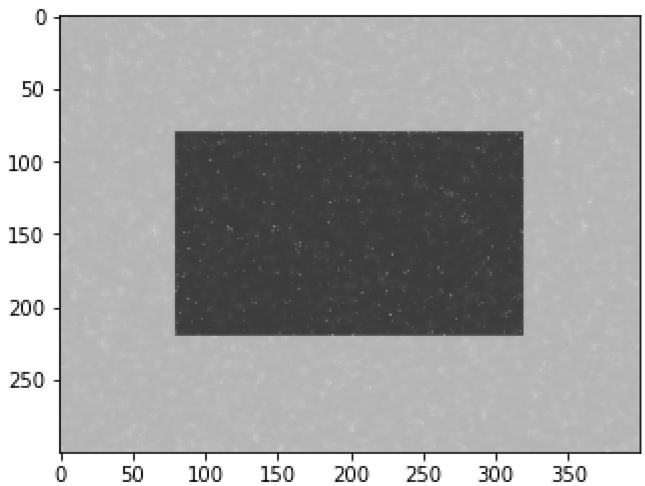
Question 4.2

In [12]:
```
print("box_gauss.png w/ gaussian blur:")
plt.imshow(cv2.GaussianBlur(box_gauss, ksize=(7, 7), sigmaX=50))
plt.show()
print("box_gauss.png w/ bilateral filter:")
plt.imshow(cv2.bilateralFilter(box_gauss, 7, sigmaColor=150, sigmaSpace=150))
plt.show()
print("box_gauss.png w/ median blur:")
plt.imshow(cv2.medianBlur(box_gauss, 7))
plt.show()

print("box_speckle.png w/ gaussian blur:")
plt.imshow(cv2.GaussianBlur(box_speckle, ksize=(7, 7), sigmaX=50))
plt.show()
print("box_speckle.png w/ bilateral filter:")
plt.imshow(cv2.bilateralFilter(box_speckle, 7, sigmaColor=150, sigmaSpace=150
))
plt.show()
print("box_speckle.png w/ median blur:")
plt.imshow(cv2.medianBlur(box_speckle, 7))
plt.show()
```

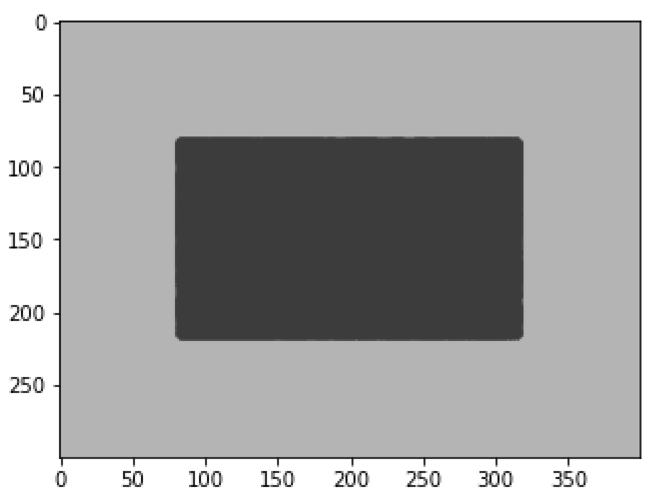
box_gauss.png w/ gaussian blur:



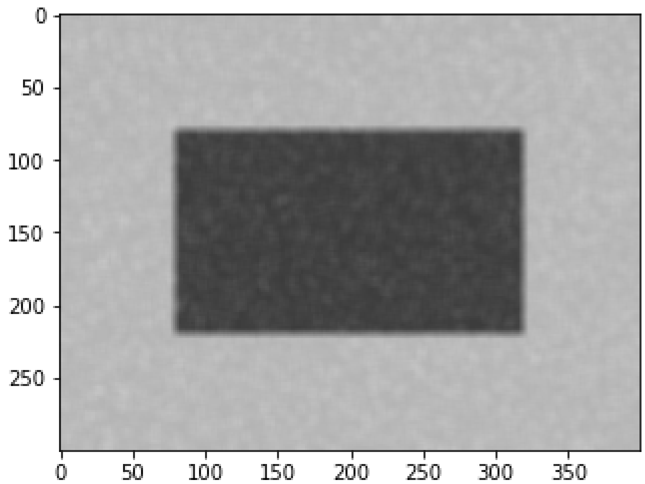
box_gauss.png w/ bilateral filter:



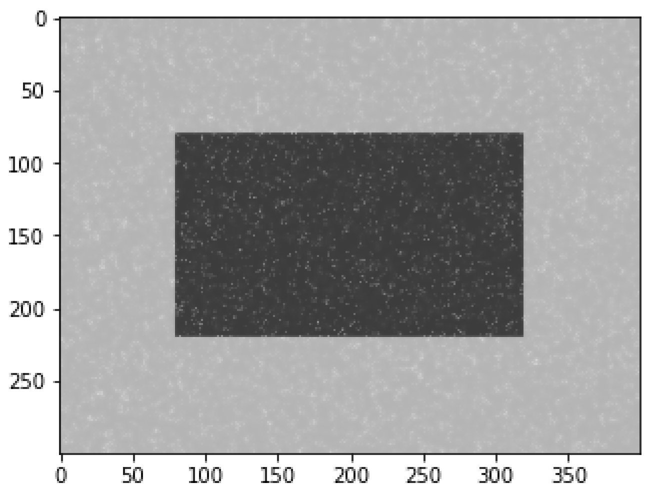
box_gauss.png w/ median blur:



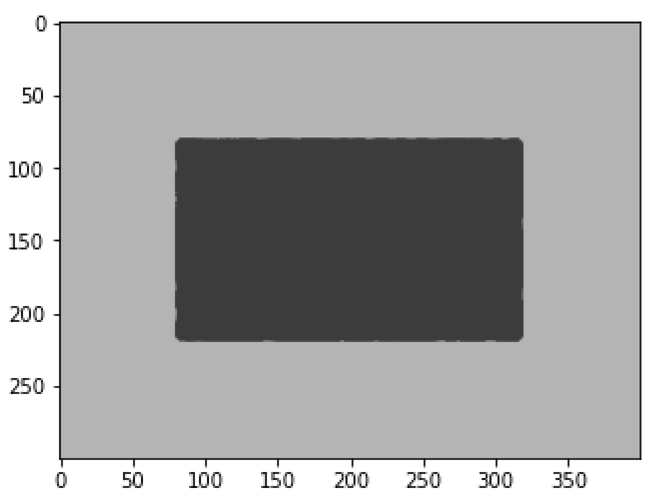
box_speckle.png w/ gaussian blur:



box_speckle.png w/ bilateral filter:



box_speckle.png w/ median blur:



Gaussian blur just tries to remove noise by blurring the image and the results are not impressive at all. In both images, the white artifacts arguably become more noticeable, since they cover larger areas. Additionally, the distinction between the outer and inner boxes are blurred as well.

With the bilateral filter, the line between the outer and inner box remains clear and distinct. This filter does a decent job with Gaussian noise. However, there is no noticeable improvement with speckle noise.

Evidently, for both images, the median blur filter works the best at removing noise. There are some remaining artifacts at the edge between the outer and inner box, but the rest of the image is much clearer than before. One con may be that the line between the outer and inner box does not remain straight.