Jesse Dubbs
Curtis Hiscock

What the code does and how:

This software consists of two programs, embed.py and decode.py. Embed is used to embed an 8-bit, grayscale image within another 8-bit grayscale image (the vessel). Decode is used to extract the embedded image from the vessel.

Embed functions by:
Opens both the -i file (referred to as vessel henceforth) and -s file (referred to as secret henceforth) using Pillow and creates NumPy uint8 matrices of height x width to represent them.
Next the program slices both matrices' uint8s into bitplane matrices of height x width x 8.
Secret's bitplane is entirely sliced into 8x8s and placed in a queue and moves on to embedding and error checks to ensure that secret was large enough to embed

Program iterates through every 9x9 square in vessel until queue is empty moving left to right, top to bottom, least significant to most significant. At each 9x9 checks the complexity of the top-left corner 8x8, if the complexity is high enough, overwrite the 8x8 with metadata (# of elements in queue, secret height, secret width), if on first run, or the next 8x8 from queue and the last bit [8,8] with a 0 to indicate the data has not been checkerboarded yet.
After writing, the program checks the newly written 8x8 complexity and, if it is not above the complexity threshold, the program xors the 8x8 with a checkerboard to ensure complexity and marks the last bit [8,8] equal to 1 to signify to decode.py that this square needs to be un-checkerboarded.

After this the program ensures that the queue was completely emptied, and if not prints that the embedding failed because the vessel does not contain enough noisy sectors to fit all of the secret image and the program exits.

Otherwise converts the vessel bitplane now holding embedded data back to a uint8 bit matrix, saveArr.
Then uses Pillow to convert the saveArr into an Image and saves as embedded.bmp

Decode functions by:
Opens the -i file (referred to as vessel henceforth) using Pillow and creates a numpy uint8 matrix of height x width to represent it.
Next the program slices each of the matrix's uint8s into bitplane matrices of height x width x 8.
All of the vessel is iterated through by 9x9s checking the complexity of 8x8 top-left corners from left to right, top to bottom, least significant to most significant placing any 9x9 whose 8x8 complexity was above .45 into a queue.

The program then decodes the metadata stored in the first element in the queue from embed.py to create bitPlaneArr's dimensions and a counter for elements to be placed in bitPlaneArr.

Then the program iterates through every position 8x8 in bitPlaneArr from left to right, top to bottom, least significant to most significant getting the next 9x9 from queue, checking if the last bit[8,8] is 1 and un-checkerboarding and then writing the top-left corner 8x8 into bitPlaneArr. This iterates until the number of 8x8s placed is equal to the number of 8x8's that belong in the image, as learned from the metadata in the first element.
Finally, the program converts the bitPlaneArr into a uint8 matrix, saveArr, and uses Pillow to convert the saveArr into an Image and saves as decoded.bmp

How to configure and use the software:

To configure and run this software on Window 10:

1. Install Python 2.7.15, making sure to allow Python to edit path variables
2. Update Pip, using Command Prompt command "pip install --upgrade pip"
    2.1. This software used with Pip version 18.1
3. Using Pip in Command Prompt, install NumPy "pip install numpy"
    3.1. This software used NumPy version 1.15.4
4. Using Pip in Command Prompt, install Pillow "pip install pillow"
    4.1. This software used Pillow version 5.3.0

5. Next run the python file embed.py to embed img2 into img1
    5.1. Command "python embed.py -i 'img1'.bmp -s 'img2'.bmp"
    5.2. Output file saved as "embedded.bmp"
    5.3. Will print failure statement and exit if img1 does not have enough noisy areas to contain all of img2 (I.e. img2 is too large or img1 is very plain and lacks complexity)
6. To extract an image embedded by embed.py
    6.1. Command "python decode.py -i 'img1'.bmp"
    6.2. Output file saved as "decoded.bmp"

- Note: To visualize the bitplanes of any img run PrintSlices.py
    - Command "python PrintSlices.py -i 'img'.bmp"
    - Bitplanes will be displayed in most to least significant layers

Usage Restrictions:

All files used must be .bmps or .pngs renamed as .bmps.
Embed

-i file must have a height and width of greater than or equal to 9x9
-s file must have a height and width of greater than or equal to 9x9
To successfully run -i must have enough noisy space to fit all of -s file

Decode

-i file must be the embedded file returned from embed.py

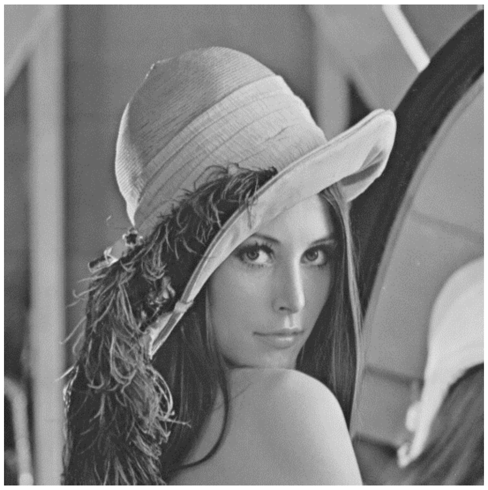
Known Issues:

During imbedding at most 7 pixels of height and 7 pixels of width of secret image are truncated so that secret image's dimensions are a multiple of 8 height pixels and a multiple of 8 width pixels and can therefore be cleanly written in 8x8s.

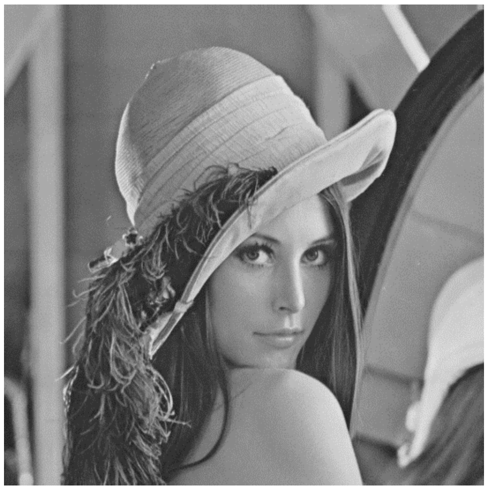
Test Cases:

Representative greyscale images were tested for a variety of lengths, widths, and complexities.

1) einstein.bmp embedded into lena_gray.bmp

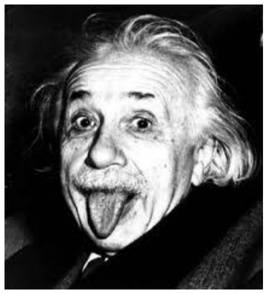


(A) Unembedded vessel image

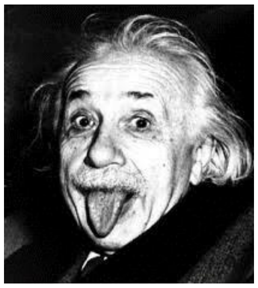


(B) Embedded vessel image

Figure 1: Vessel Image, before and after



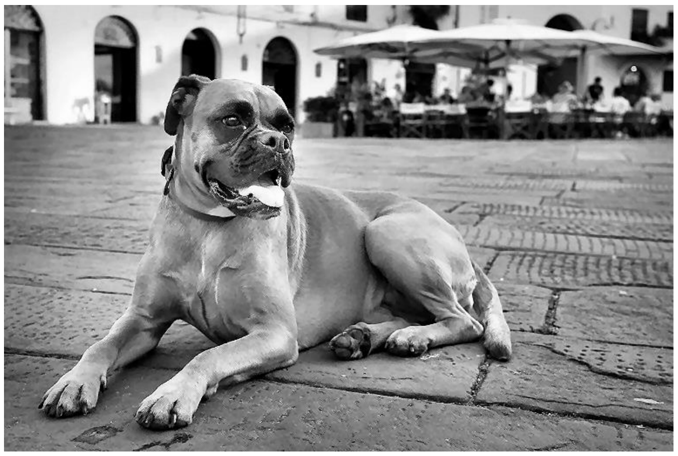
(C)Image to be embedded



(B) Image after extraction

Figure 2: Hidden Image, before and after

2) apple.bmp embedded into dog.bmp


(A) dog.bmp before embedding

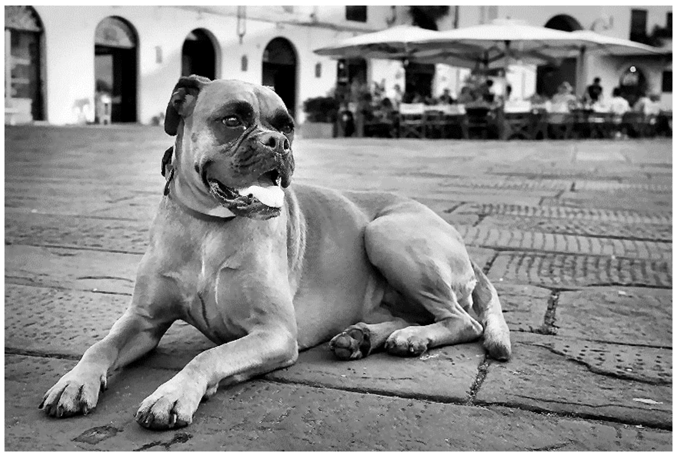
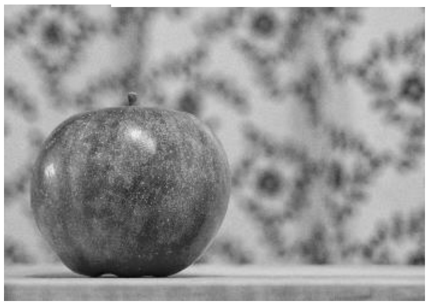

(B) dog.bmp after embedding

Figure 3: dog, before and after embedding


(A) apple.bmp before embedding

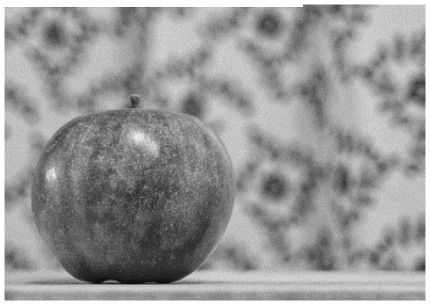

(B) apple.bmp extracted from dog (above)

Figure 4: apple, before and after extraction

3) Embedding lena_gray.bmp into face.bmp. Fails due to capacity.

Note: face.bmp is half the size of lena_gray.bmp

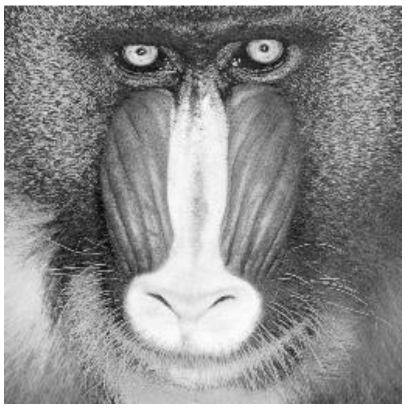
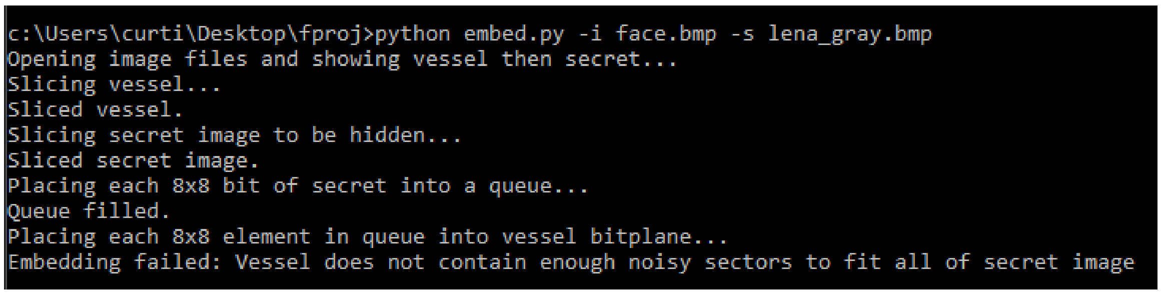

(A) face.bmp

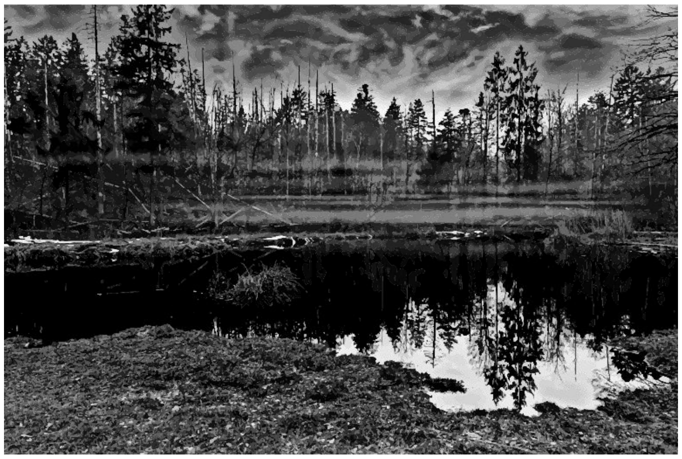
Figure 5: Example of capacity failure



```
c:\Users\curti\Desktop\fproj>python embed.py -i face.bmp -s lena_gray.bmp
Opening image files and showing vessel then secret...
Slicing vessel...
Sliced vessel.
Slicing secret image to be hidden...
Sliced secret image.
Placing each 8x8 bit of secret into a queue...
Queue filled.
Placing each 8x8 element in queue into vessel bitplane...
Embedding failed: Vessel does not contain enough noisy sectors to fit all of secret image
```

(B) Example of failure due to lack of capacity

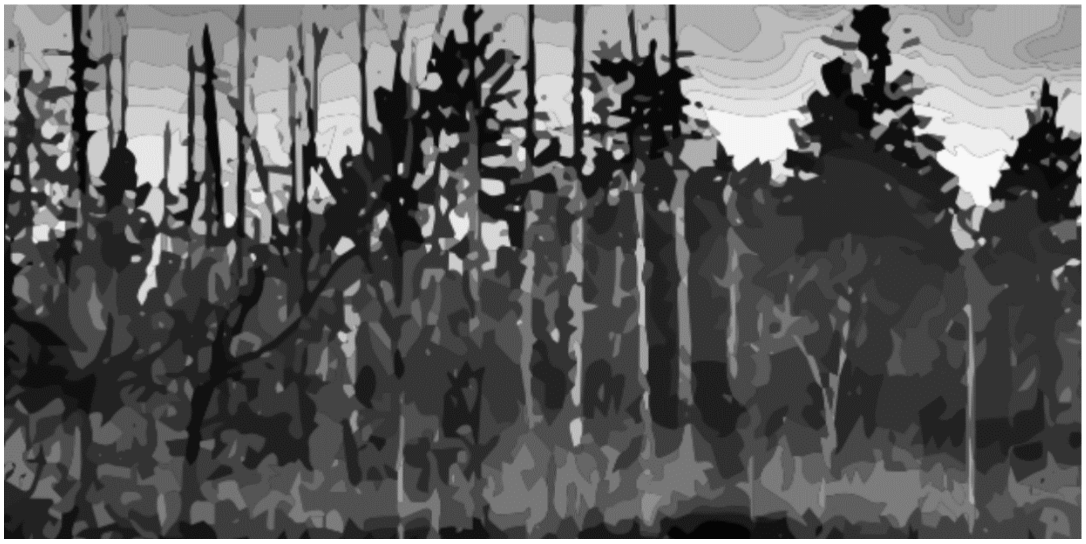
4) Embedding lena_gray.bmp into swamp.bmp. This results in an unexpected failure due to the lack of complexity in the bit plains of swamp.bmp. Here we naively expected the embedding operation to succeed, as swamp.bmp was more than twice the size of lena_gray.bmp.



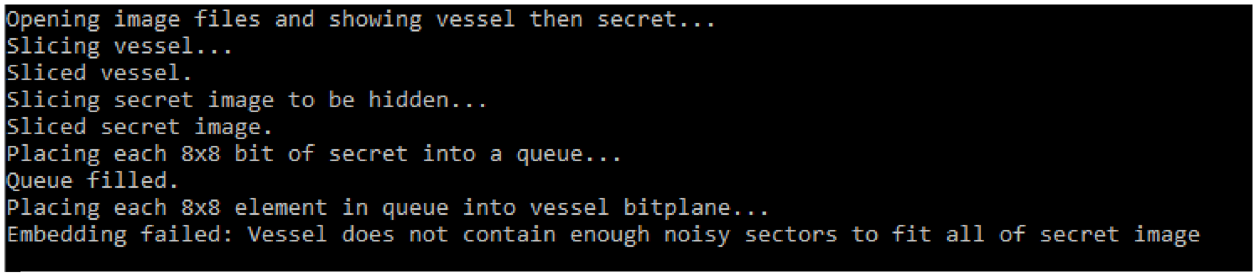
(A) swamp.bmp

(B) lena_gray.bmp

Figure 1: attempt to embed lena_gray.bmp into swap.bmp



(C) Close up of lack of complexity in swamp.bmp. Revealing why embedding fails. Image can be decomposed with PrintSlices.py to reveal the lack of complexity.



```
Opening image files and showing vessel then secret...
Slicing vessel...
Sliced vessel.
Slicing secret image to be hidden...
Sliced secret image.
Placing each 8x8 bit of secret into a queue...
Queue filled.
Placing each 8x8 element in queue into vessel bitplane...
Embedding failed: Vessel does not contain enough noisy sectors to fit all of secret image
```

(C) example of capacity failure due to low vessel complexity