MP #1: Peer Feedback #1 (Pre-Feedback Work to Date) ⇒ Post-Feedback with annotation alongside.

Libo Zhang (lz200)

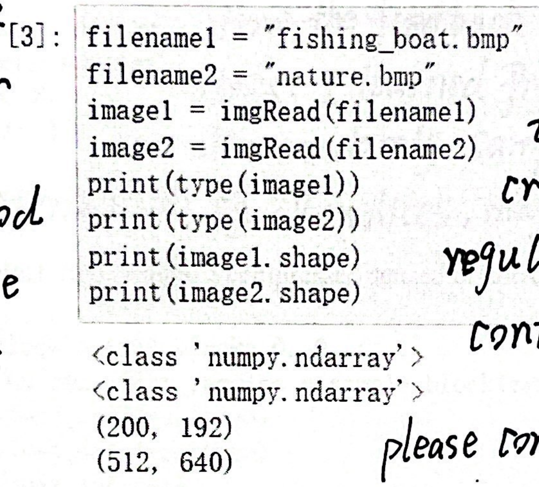
The structure of my work to date format will follow the recommended project milestones.

Note: Only the code written for **testing** my algorithm will be displayed to help peer review/feedback.

Week 1:

(1) Load/read the images

Overall peer feedback I received from other group members: Currently making good progress based on the Recommended Project Milestones.



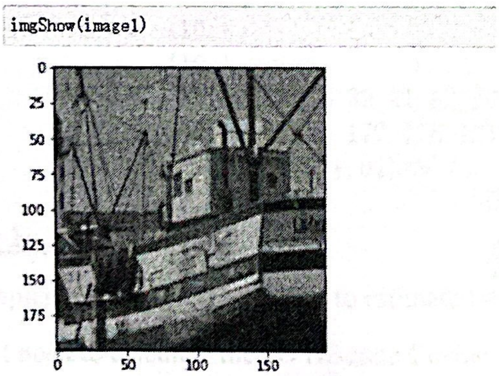
```
[3]: filename1 = "fishing_boat.bmp"
     filename2 = "nature.bmp"
     image1 = imgRead(filename1)
     image2 = imgRead(filename2)
     print(type(image1))
     print(type(image2))
     print(image1.shape)
     print(image2.shape)

     <class 'numpy.ndarray'>
     <class 'numpy.ndarray'>
     (200, 192)
     (512, 640)
```

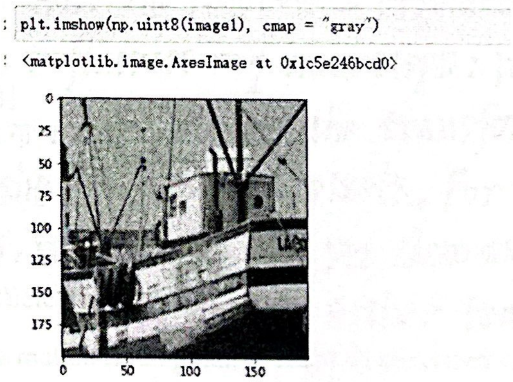
Notes to myself - what I will do next: I will follow the Project milestones and try to implement random subset cross-validation to choose the regularization parameter λ, and continue to update my slide doc.

please continue to the next page for the post-feedback reflection.

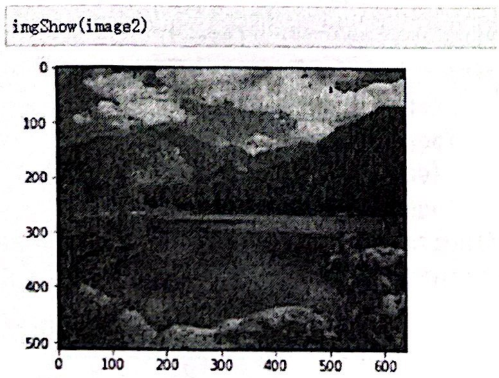
(2) Plot ("imshow") the images



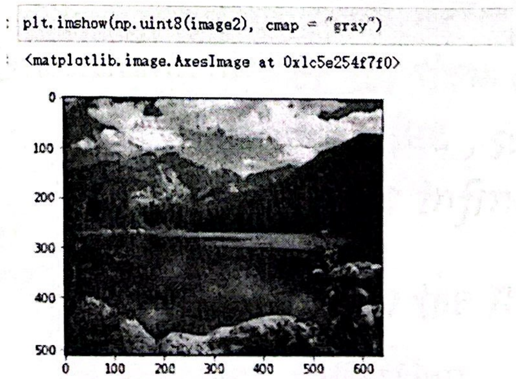
```
imgShow(image1)
```



```
: plt.imshow(np.uint8(image1), cmap = "gray")
: <matplotlib.image.AxesImage at 0x1c5e246bcd0>
```

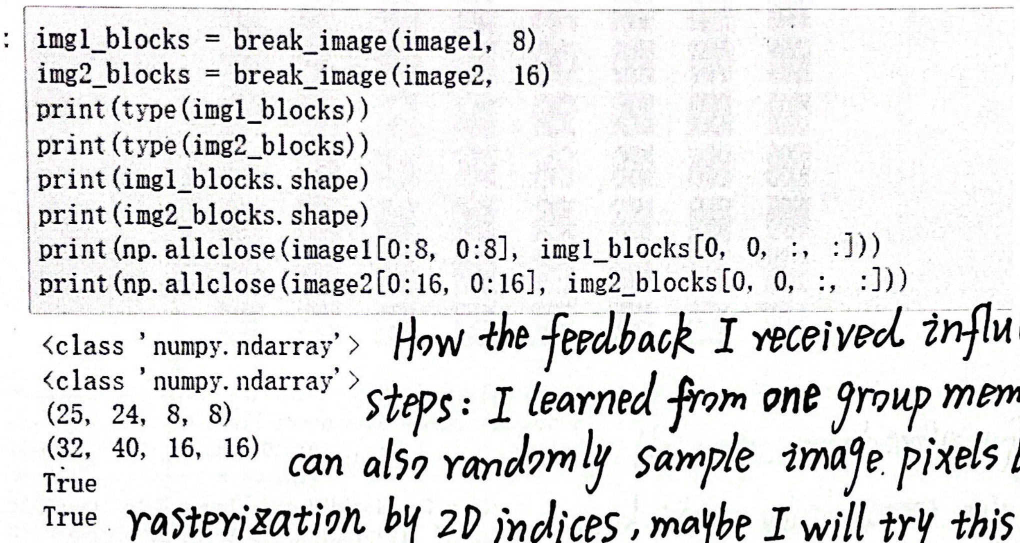


```
imgShow(image2)
```



```
: plt.imshow(np.uint8(image2), cmap = "gray")
: <matplotlib.image.AxesImage at 0x1c5e254f7f0>
```

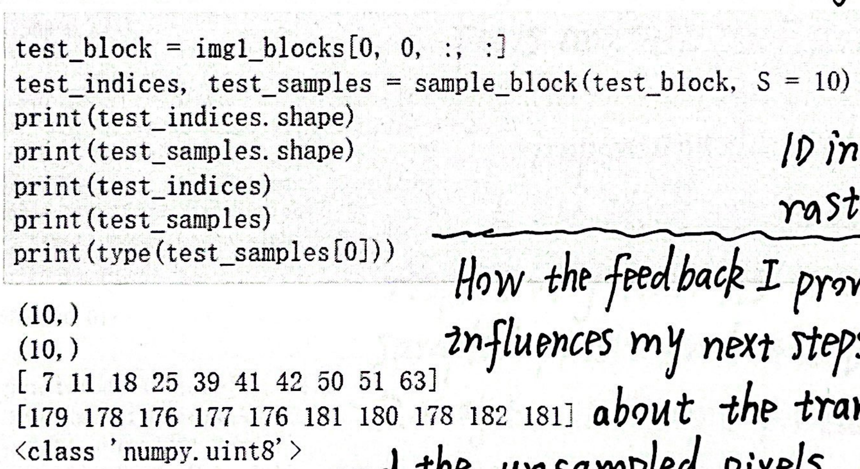
(3) Select a desired K * K block from an image



```
img1_blocks = break_image(image1, 8)
img2_blocks = break_image(image2, 16)
print(type(img1_blocks))
print(type(img2_blocks))
print(img1_blocks.shape)
print(img2_blocks.shape)
print(np.allclose(image1[0:8, 0:8], img1_blocks[0, 0, :, :]))
print(np.allclose(image2[0:16, 0:16], img2_blocks[0, 0, :, :]))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(25, 24, 8, 8)
(32, 40, 16, 16)
True
True
```

(4) Sample pixels from a block (to simulate a compressed sensed or corrupted image)



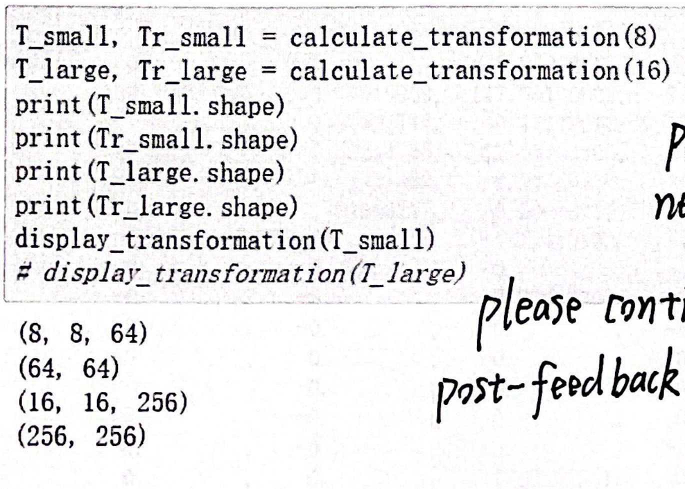
```
test_block = img1_blocks[0, 0, :, :]
test_indices, test_samples = sample_block(test_block, S = 10)
print(test_indices.shape)
print(test_samples.shape)
print(test_indices)
print(test_samples)
print(type(test_samples[0]))
```

```
(10,)
(10,)
[ 7 11 18 25 39 41 42 50 51 63]
[179 178 176 177 176 181 180 178 182 181]
<class 'numpy.uint8'>
```

Week 2:

(1) Implement LASSO regression to estimate DCT coefficients for a single block

First, I need to calculate the 2D Discrete Cosine Transformation (DCT) matrix T and its rasterized version.



```
T_small, Tr_small = calculate_transformation(8)
T_large, Tr_large = calculate_transformation(16)
print(T_small.shape)
print(Tr_small.shape)
print(T_large.shape)
print(Tr_large.shape)
display_transformation(T_small)
# display_transformation(T_large)
```

```
(8, 8, 64)
(64, 64)
(16, 16, 256)
(256, 256)
```

How the feedback I received influences my next steps: I learned from one group member that we can also randomly sample image pixels before rasterization by 2D indices, maybe I will try this later, ~~and~~ maybe not, because I think it is also OK to sample with 1D indices after the rasterizing process.

How the feedback I provided to my peers influences my next steps: we talked a lot about the transformation matrix and the unsampled pixels. For the unsampled pixels, I should not set them as 0, I should either temporarily drop them during regression or set them as invalid pixel value, such as negative infinity.

please continue to the next page for post-feedback reflection.

$\lambda = [1\times10^{-4}, 1\times10^{4}]$

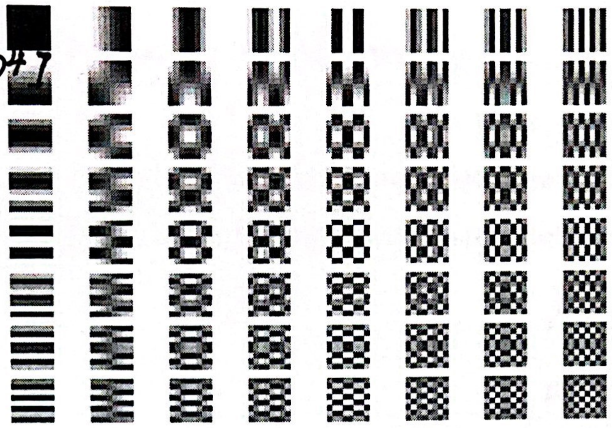
$\uparrow$

$J(x) = 1\cdot \text{error} + \lambda \cdot \text{reg}$

$= (1-\alpha)\,\text{error} + \alpha\,\text{reg}$

$[0 \leq \alpha \leq 1]$

$= \beta \cdot \text{error} + 1 \cdot \text{reg}$



The cost function for sklearn linear model LASSO.

```
# Implement LASSO regression to estimate DCT coefficients for a single block
# First, try the small image with one single block
block1 = img1_blocks[10, 10, :, :]
# samples = B = A * Weights
indices1, B1 = sample_block(block1, S = 50)
A1 = sample_transformation(indices1, Tr_small)
model1 = linear_model.Lasso(alpha = 0.1)
model1.fit(A1, B1)
```

```
Lasso(alpha=0.1)
```

How the exchange of information and ideas with ~~your~~ my peers influence my next steps: After discussion with my peers and the professor, I think it is very important to realize that each $k \cdot k$ block ~~may or should~~ should respectively have its own randomly sampled pixels and the DCT coefficients λ may be different for each block. The key point is that I need to focus on each k by k block, instead of the entire image.

```
# Implement LASSO regression to estimate DCT coefficients for a single block
# Second, try the large image with one single block
block2 = img2_blocks[10, 10, :, :]
indices2, B2 = sample_block(block2, S = 150)
A2 = sample_transformation(indices2, Tr_large)
model2 = linear_model.Lasso(alpha = 0.01)
model2.fit(A2, B2)
```

```
Lasso(alpha=0.01)
```

A good question to be discovered: Can we have a large sparsity while maintaining good regression (image reconstruction) performance?

```
print(model1.score(A1, B1))
print(model2.score(A2, B2))
print(len(model1.coef_))
print(len(model2.coef_))
```

```
0.9853763557346827
0.9972580855515059
64
256
```

```
model1.coef_
```

```
array([   0.        ,   -0.        ,  -76.97093667,   66.09599727,
        130.39442943, -116.78619939, -117.65100802,   77.36717078,
         -7.83392908,    2.57807744,    9.15128278,  -15.63842612,
         13.89369968,  -37.62214755,   27.34703466,   14.66571335,
         -3.77839484,   -8.61521589,   -2.34779236,    4.72752235,
          0.        ,    3.92366117,    6.50030059,  -14.26790545,
         -3.80742547,   -0.        ,   -1.72449202,   -0.        ,
         -0.        ,    0.        ,   -0.        ,   -0.        ,
         -0.        ,   -0.        ,   -3.92660802,    0.        ,
          0.        ,   -0.        ,    0.        ,   -5.72953258,
         -0.        ,    0.        ,   -0.        ,    0.        ,
          0.        ,   -0.        ,    0.        ,    0.        ,
          0.        ,   -0.        ,    0.        ,    0.        ,
          0.        ,   -0.        ,   -0.        ,   -2.03222562,
          0.        ,    0.        ,    1.80992801,   -0.        ,
         -0.        ,    0.        ,   -0.        ,    0.        ])
```