Lab5_Yunting

October 10, 2021

1 Lab05 - Sparsity Aware Learning

Author: Yunting Chiu adapted from Dr. Zois Boukouvalas

2 Install the required packages

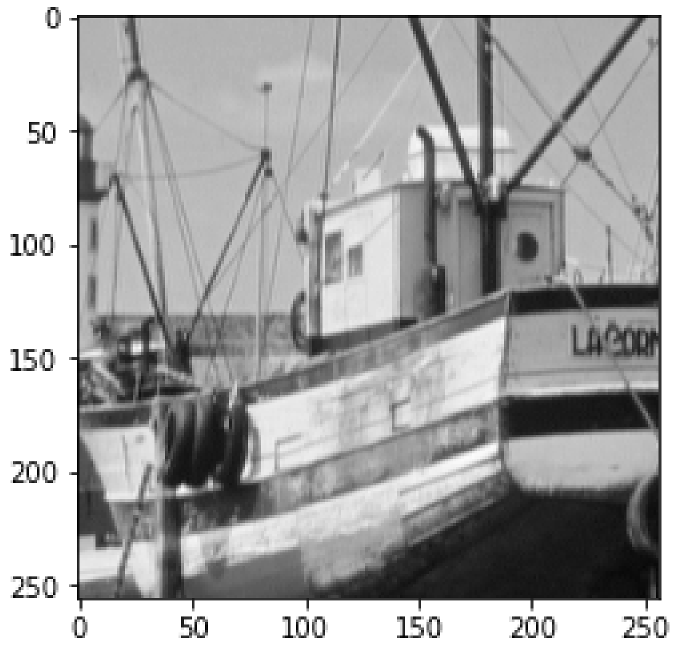
[76]:
```
import imageio
import cv2
import matplotlib.pyplot as plt
import math
from scipy.io import loadmat
from google.colab.patches import cv2_imshow
import numpy as np
import os
import sys
from sklearn.linear_model import Lasso
from skimage.util import view_as_windows as viewW
```

3 Read the image

Before we start, we read the image to see what does it looks like. Because it is a grayscale image, so the image only has 2 dimensions.

[77]:
```
# read the image
path = "/content/drive/MyDrive/American_University/2021_Fall/
↪DATA-642-001_Advanced Machine Learning/GitHub/Labs/05/boats.mat"
y_exact = loadmat(path, variable_names = 'boats').get('boats')
y_exact = np.array(y_exact, dtype=np.float32)
plt.imshow(y_exact, cmap='gray')
print(y_exact.dtype, y_exact.shape)
```

```
float32 (256, 256)
```



4 Define the functions

- `im2col` is one of MATLAB functions that rearranges image blocks into columns. There is no existing function supporting in Python, so we need to define a new one. For more detail: https://www.mathworks.com/help/images/ref/im2col.html
- `waitbar` is used for tracking the process when we run the code.

[78]:
```
# Note: BSZ is designed as a 2D array
def im2col(A, BSZ, stepsize=1):
  return viewW(A, (BSZ[0], BSZ[1])).reshape(-1, BSZ[0]*BSZ[1]).T[:, ::stepsize]
```

[79]:
```
def waitbar(count, total, suffix=''):
  bar_len = 40
  filled_len = int(round(bar_len * count / float(total)))
  percents = round(100.0 * count / float(total), 1)
  bar = '=' * filled_len + '-' * (bar_len - filled_len)
  sys.stdout.write('\t[%s] %s%s\t%s\r' % (bar, 100*percents, '%', suffix))
  sys.stdout.flush()
  #print('\t[%s] %s%s\t%s\r' % (bar, 100*percents, '%', suffix))
```

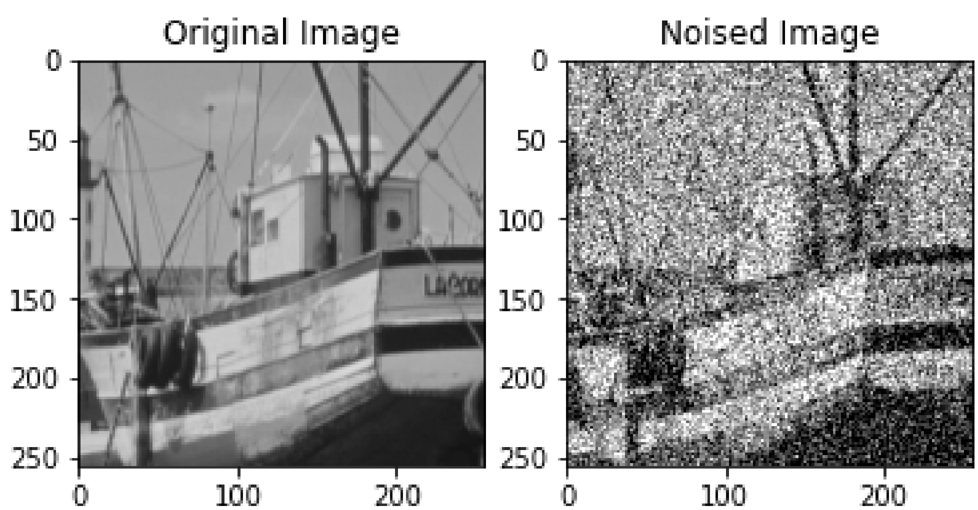
5 Generate some noises into the image

Now, the original image and the noised image look like

[80]:
```
# print(y_exact)
sigma = 100
# Construct a noisy version of the original image, adding the noise in each␣
 ↪pixel
y_noise = y_exact + np.random.randn(y_exact.shape[0], y_exact.shape[1])*sigma
blocksize = 12 # block size
K = 14**2 # number of atoms in the dictionary
Y = im2col(y_noise, BSZ=(blocksize, blocksize))

# Plot
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(y_exact, cmap='gray', vmin=0, vmax=255)
plt.subplot(1, 2, 2)
plt.title('Noised Image')
plt.imshow(y_noise, cmap='gray', vmin=0, vmax=255)
```

[80]:
```
<matplotlib.image.AxesImage at 0x7f85fb1f7250>
```



And the shape that we rearranged the image blocks into columns is 144 x 60025, which is make sense. There are 60,025 patches in total are obtained.

[81]:
```
print(Y.shape)
```

```
(144, 60025)
```

6 Construct a fixed dictionary

[82]:
```
n = np.array(range(0, blocksize))
# make a 12*14 martrix
DCT = np.zeros(shape = (blocksize, int(math.sqrt(K))))

# normalized to unit norm D resembles a redundant DCT matrix.
for k in range(0, int(math.sqrt(K))):
  V = np.cos(n.conj().transpose()*k*np.pi/math.sqrt(K))
  DCT[:, k] = V/np.linalg.norm(V)

# Kronecker product of two arrays
DCT = np.kron(DCT, DCT)
Dict_fixed = DCT

# Now, we have a 2D-DCT
print(len(DCT.shape))

for k in range(0, Dict_fixed.shape[1]):
  Dict_fixed[:, k] = Dict_fixed[:, k]/np.linalg.norm(Dict_fixed[:, k])
print(Dict_fixed.shape)
```

```
2
(144, 196)
```

7 Denoising using Lasso

In order to compute the estimated signal, we use the Lasso formulation in every 100 iterations.

[83]:
```
Y_reconst = np.zeros(shape=Y.shape)
subproblems = Y.shape[1]
print(subproblems/100)

# Denoising each image patch separately using Lasso regression
waitbar(0, 100, 'Please wait...')
for i in range(0, Y.shape[1]):
  if i % 100 == 0:
    waitbar(i/subproblems, 100, 'Please wait...')
  G = Lasso(fit_intercept=False, normalize=False, alpha = 0.4).fit(Dict_fixed,␣
↪Y[:, i])
  Y_reconst[:, i] = np.dot(Dict_fixed, G.coef_)
sys.stdout.flush()
print('')
```

```
600.25
```

$\theta_i \in R^{196}$

[84]:
```
print(Dict_fixed.shape)
```

```
(144, 196)
```

This is a Python implementation of the code contained in `Chapter_15_CoreInpaining1.m` of Elad's book "*Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*", Springer, 2010

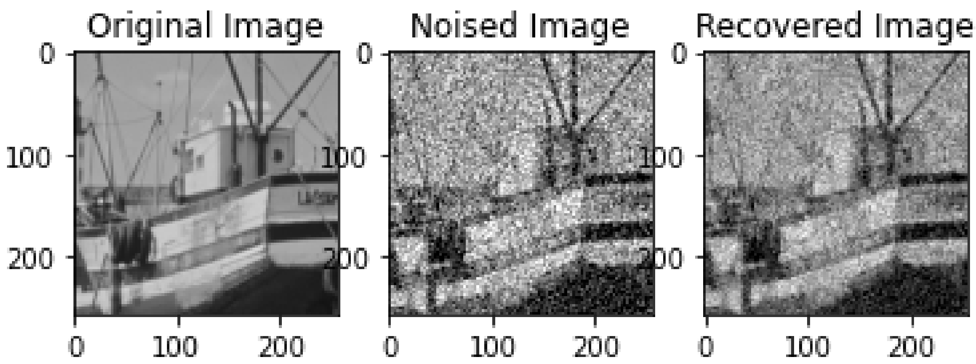
[85]:
```
# Average the values of the overlapping patches in order to form the full␣
↪denoised image
N = y_exact.shape[0]
n = blocksize
yout = np.zeros(shape=(N, N))
Weight = np.zeros(shape=(N, N))
i = 0
j = 0
for k in range(0, (N-n+1)**2):
  patch = np.reshape(Y_reconst[:, k], newshape=(n, n))
  yout[i:(i+n), j:(j+n)] += patch
  Weight[i:(i+n), j:(j+n)] += 1
  if j < (N-n):
    j += 1
  else:
    j = 0
    i += 1
recovered_boat_dct = np.divide(yout, Weight)
print(20*np.log10(255 * np.sqrt(y_exact.shape[0]*y_exact.shape[1]) / np.linalg.
↪norm(y_exact-y_noise)))
print(20*np.log10(255 * np.sqrt(y_exact.shape[0]*y_exact.shape[1]) / np.linalg.
↪norm(y_exact-recovered_boat_dct)))
print(20*np.log10(255 * np.sqrt(y_exact.shape[0]*y_exact.shape[1]) / np.linalg.
↪norm(y_exact-y_exact)))

# Plots
plt.figure(1)
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(y_exact, cmap='gray', vmin=0, vmax=255)
plt.subplot(1, 3, 2)
plt.title('Noised Image')
plt.imshow(y_noise, cmap='gray', vmin=0, vmax=255)
plt.subplot(1, 3, 3)
plt.title('Recovered Image')
plt.imshow(recovered_boat_dct, cmap='gray', vmin=0, vmax=255)
plt.show()
```

```
8.140247343811431
12.92369245381151
```

```
inf
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: RuntimeWarning:
divide by zero encountered in double_scalars
```



[86]:
```
print(np.linalg.norm(y_exact-y_exact))
print(np.linalg.norm(y_exact-y_noise))
print(np.linalg.norm(y_exact-recovered_boat_dct))
```

```
0.0
25572.18152077926
14743.327847552691
```

As we can see from the results above, if we normalize the difference between the original image and the nosied image, the value is 25578, and if we normalize the difference between the original image and the recovered image, the value is 14783, meaning that the smaller value is close to the orginal image.

8 Output

[87]:
```
# should access the Google Drive files before running the chunk
%%capture
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
!jupyter nbconvert --to pdf "/content/drive/MyDrive/American_University/
↪2021_Fall/DATA-642-001_Advanced Machine Learning/GitHub/Labs/05/submit/
↪Lab5_Yunting.ipynb"
```