DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
ECE 211A: DIGITAL IMAGE PROCESSING I

INSTRUCTOR: Prof. Achuta Kadambi
TA: Yunhao Ba

NAME: Qiong Hu
UID: 405065032

HOMEWORK 2: SPARSE CODING AND DICTIONARY LEARNING

PROBLEM	TOPIC	MAX. POINTS	GRADED POINTS	REMARKS
2.1	Training and Testing Images	1.0		
3.1	Image Patches	1.0		
3.2	Random Dictionary	1.0		
3.3	Reconstruction Function	1.0		
3.4	Reconstruction Result	1.0		
4.1	K-SVD Implementation	2.0		
4.2	K-SVD Dictionary	1.5		
4.3	K-SVD Reconstruction	1.5		
	Total	10.0		

1 Problem Setup

In sparse coding, the goal is to find a sparse representation $\mathbf{x}$ of the input data $\mathbf{y}$ with a dictionary $\mathbf{A}$. The whole process can be characterized by the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2 \\ &s.t. \|\mathbf{x}\|_0 \leq S. \end{aligned} \tag{1}$$

Orthogonal Matching Pursuit (OMP) is one of the common methods to solve the above sparse coding problem, and the dictionary $\mathbf{A}$ can be learned using the K-SVD algorithm. In this homework, we will use these two algorithms for an image inpainting task, where our goal is to remove the extra text content on an image. Similarly, we will use PSNR and SSIM as the metrics to evaluate the reconstruction performance in this homework.

2 Image Preparation

First, we need to prepare some images for the inpainting task. Here are the procedures:

1. Pick two images from the BSDS500 dataset[1], and convert them to gray scale. Crop a 256×256 region from each of the selected images. Denote these two image as $y_{train}(n_1, n_2)$ and $y_{test}(n_1, n_2)$.
2. Put some text on $y_{test}(n_1, n_2)$. Denote the obtained image as $y_{test\_missing}(n_1, n_2)$.
3. Based on $y_{test\_missing}(n_1, n_2)$, generate a mask map to indicate whether a pixel in $y_{test\_missing}(n_1, n_2)$ is missed or not. Denote this mask as $m_{test\_missing}(n_1, n_2)$.

[1] `https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html`

2.1 Training and Testing Images (1.0 points)

Plot $y_{train}(n_1,n_2)$, $y_{test}(n_1,n_2)$, $y_{test\_missing}(n_1,n_2)$, and $m_{test\_missing}(n_1,n_2)$. Report PSNR($y_{test}$, $y_{test\_missing}$) and SSIM(y_{test}, $y_{test\_missing}$).

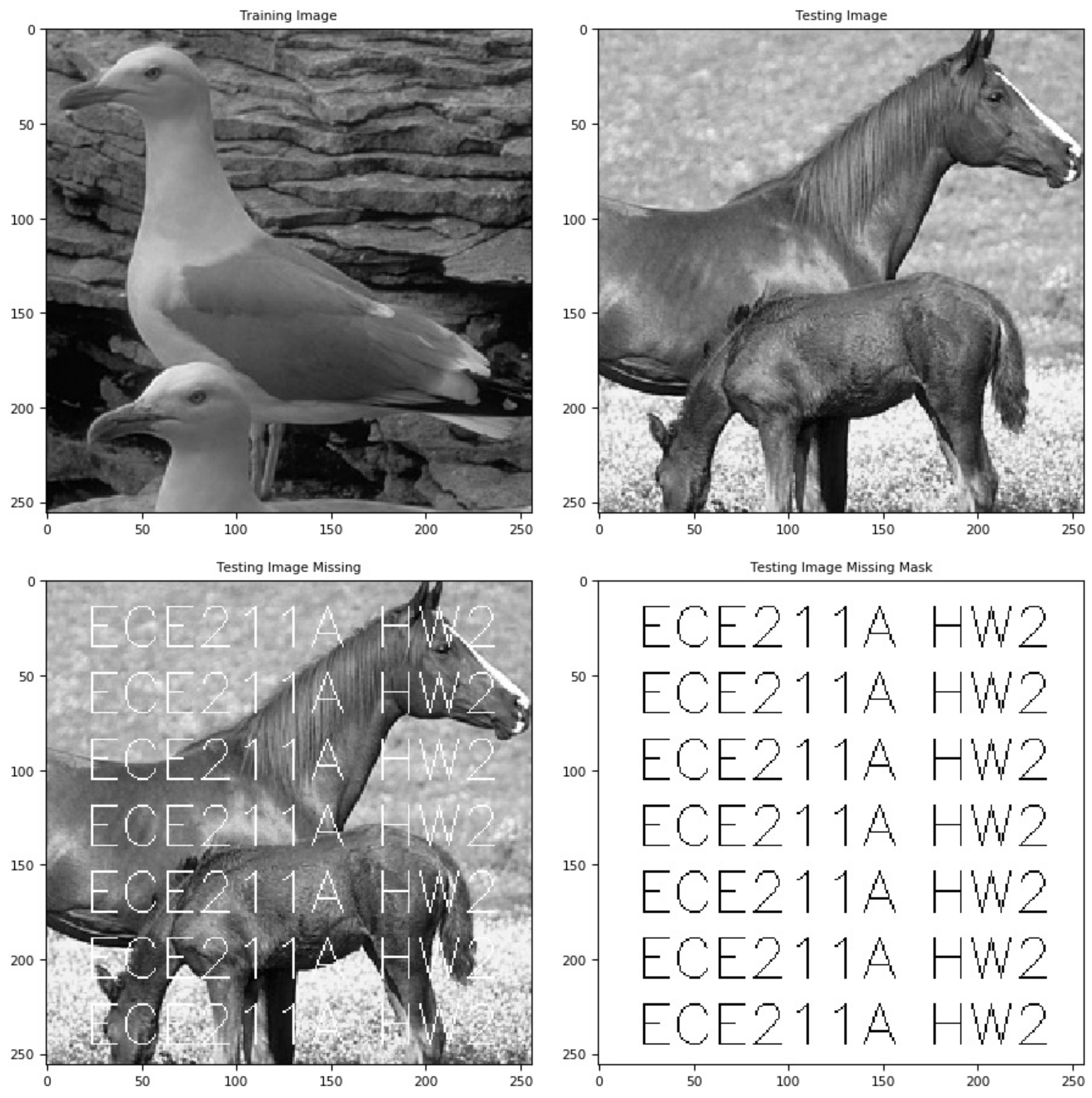

Figure 1: Images of $y_{train}(n_1,n_2)$, $y_{test}(n_1,n_2)$, $y_{test\_missing}(n_1,n_2)$, and $m_{test\_missing}(n_1,n_2)$.

$$\text{PSNR}(y_{test}, y_{test\_missing}) = 20.3874$$
$$\text{SSIM}(y_{test}, y_{test\_missing}) = 0.7089$$

3 Random Dictionary Reconstruction

3.1 Image Patches (1.0 points)

Considering the dimension of images, it is common to split the images into smaller block patches in sparse coding problems. Split $y_{test\_missing}(n_1, n_2)$ into multiple block patches of size 8×8 pixels. How many non-overlapping block patches can you get? Plot any 25 block patches extracted together with their associated masks.

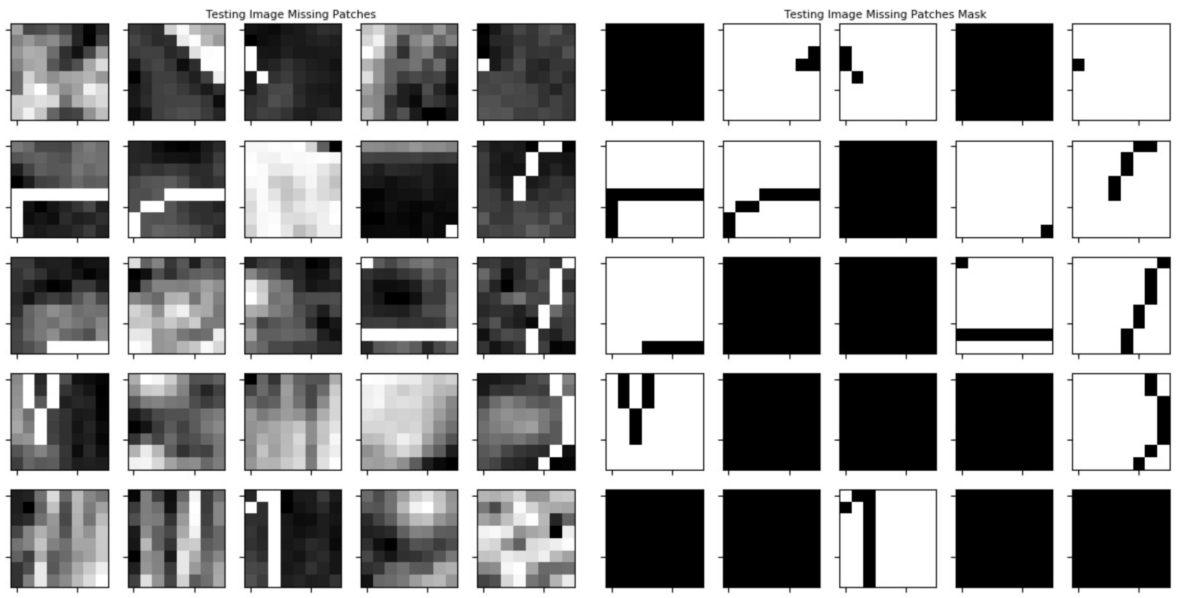

Figure 2: The plotted patches and their masks.

$\because 256/8 = 32,$

$\therefore$ I can get $32 * 32 = 1024$ non-overlapping block patches.

3.2 Random Dictionary (1.0 points)

To recover the missing pixels using OMP, we need a dictionary $\mathbf{A}_{random}$. In this question, you need to create a random dictionary with 512 atoms. What is the size of this dictionary? Plot any 25 atoms of $\mathbf{A}_{random}$.

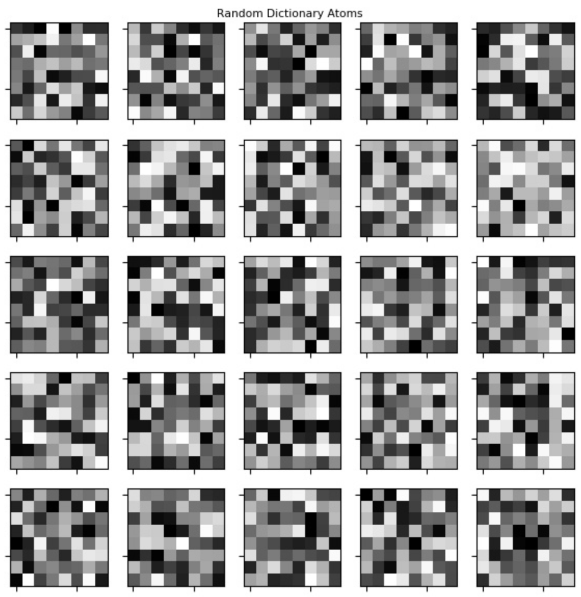

Figure 3: Image of 25 different atoms in the random dictionary.

The size of the dictionary $\mathbf{A}_{random}$ is 64*512.

3.3 Reconstruction Function (1.0 points)

Implement a function to reconstruct a block patch using OMP. You can use the OMP function in scikit-learn package[2]. Make sure all the atoms in the dictionary have unit norm before using this function, and the maximal number of non-zero elements S is 20. Place your code in the box below.

```
from sklearn.preprocessing import normalize
from sklearn.linear_model import orthogonal_mp as OMP
import numpy as np

def reconstruct(dictionary, patch, mask_patch, n_nonzero = 20):
    # Given:
    #       dictionary: size = 64 * 512
    #       patch: size = 8 * 8
    #       mask_patch: size = 8 * 8; value=0 at masking text pixels, =1 otherwise
    # Return:
    #       patch_recons: size = 8 * 8

    # Normalize columns of dictionary
    dictionary = normalize(dictionary, axis = 0)

    # Remove pixels with texts
    patch = patch.flat
    mask_patch = mask_patch.flat
    index = np.nonzero(mask_patch)[0]

    # Normalize non-zero patch elements
    patch_mean = np.mean(patch[index])
    patch_norm = np.linalg.norm(patch[index] - patch_mean)
    patch_normalized = (patch[index] - patch_mean)/patch_norm

    # OMP and reconstruct
    sparse_code = OMP(dictionary[index, :], patch_normalized, n_nonzero_coefs = n_nonzero)
    patch_recons = dictionary.dot(sparse_code)
    patch_recons = patch_recons * patch_norm + patch_mean
    patch_recons = patch_recons.reshape(8, 8)

    return patch_recons
```

[2]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.orthogonal_mp.html

3.4 Reconstruction Result (1.0 points)

Use your reconstruction function in Section 3.3 to reconstruct all the block patches of $y_{test\_missing}(n_1, n_2)$. Concatenate these reconstructed patches together to form $\hat{y}_{test\_random}(n_1, n_2)$. Plot $\hat{y}_{test\_random}(n_1, n_2)$, and report PSNR($y_{test}$, $\hat{y}_{test\_random}$) and SSIM(y_{test}, $\hat{y}_{test\_random}$).

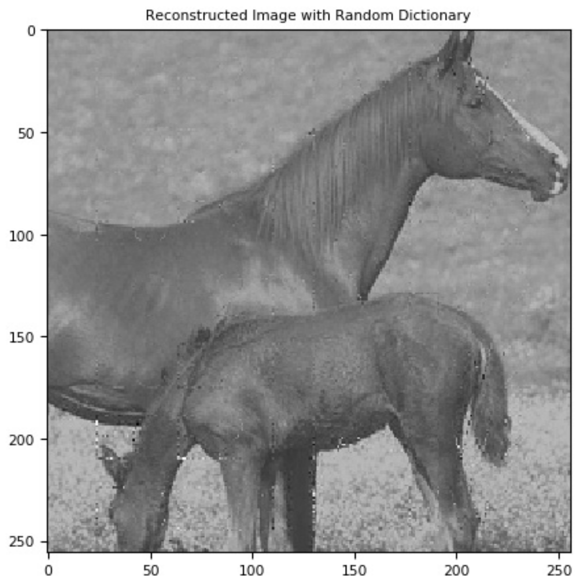

Figure 4: Image of $\hat{y}_{test\_random}(n_1, n_2)$.

$$\text{PSNR}(y_{test}, \hat{y}_{test\_random}) = 16.9079$$
$$\text{SSIM}(y_{test}, \hat{y}_{test\_random}) = 0.8355$$

4 K-SVD

4.1 K-SVD Implementation (2.0 points)

Split $y_{train}(n_1, n_2)$ into multiple block patches of size 8×8 pixels, and implement your own K-SVD algorithm to learn a dictionary $\mathbf{A}_{ksvd}$ from block patches of $y_{train}(n_1, n_2)$. The size of $\mathbf{A}_{ksvd}$ should be the same as the size of $\mathbf{A}_{random}$. In your implementation, you can use the OMP function in scikit-learn and SVD function in NumPy[3]. The maximal number of non-zero elements S is still 20. Place your code in the box below.

```
from sklearn.preprocessing import normalize
from sklearn.linear_model import orthogonal_mp as OMP
import numpy as np

# K-SVD
def dict_update(y, d, x, n_atom = 512):
    # y: training image matrix
    # d: dictionary, unit normalized for each column
    # x: sparse code
    # n_atom: atom number of the dictionary

    for k in range(n_atom):
        index = np.nonzero(x[k,:])[0]        # index of support of residue
        if len(index) == 0:
            continue

        # Update the k-th column of the dictionary
        d[:, k] = 0
        r = (y - np.dot(d, x))[:, index]        # residue
        u, s, v = np.linalg.svd(r, full_matrices = False)
        d[:, k] = u[:, 0]
        x[k, index] = s[0] * v[0, :]
    return d, x

# Main
n_atom = 512

# Initialize dictionary
dictionary = np.random.random((64, n_atom))
dictionary = normalize(dictionary, axis = 0)

# Reshape image (256*256) as matrix (64*1024)
y_train = np.zeros((64, 1024))
```

[3]https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html

```
for i in range(32):
    for j in range(32):
        y_patch = im_train[i*8:i*8+8, j*8:j*8+8].flat
        y_train[:, i*32+j] = y_patch[:]

# Train the dictionary using training image
max_iter = 20
for i in range(max_iter):
    x = OMP(dictionary, y_train, n_nonzero_coefs = 20)
    e = np.linalg.norm(y_train - np.dot(dictionary, x))
    dictionary, x = dict_update(y_train, dictionary, x, n_atom)
    print("Loop " + str(i) + ": error = " + str(e))

# Concatenate reconstructed patches together
im_recons = np.zeros((256, 256))
for i in range(32):
    for j in range(32):
        im_recons[i*8:i*8+8, j*8:j*8+8] = reconstruct(dictionary, im_test_missing[i*8:i*8+8,
                                          j*8:j*8+8], im_mask[i*8:i*8+8, j*8:j*8+8])

# Scale the pixel value of the reconstructed image to [0, 1]
im_recons = (im_recons - np.min(im_recons)) / (np.max(im_recons) - np.min(im_recons))
```

4.2 K-SVD Dictionary (1.5 points)

Sort the atoms of $\mathbf{A}_{ksvd}$ based on their standard deviation in ascending order. Plot the first 25 atoms in the box below.

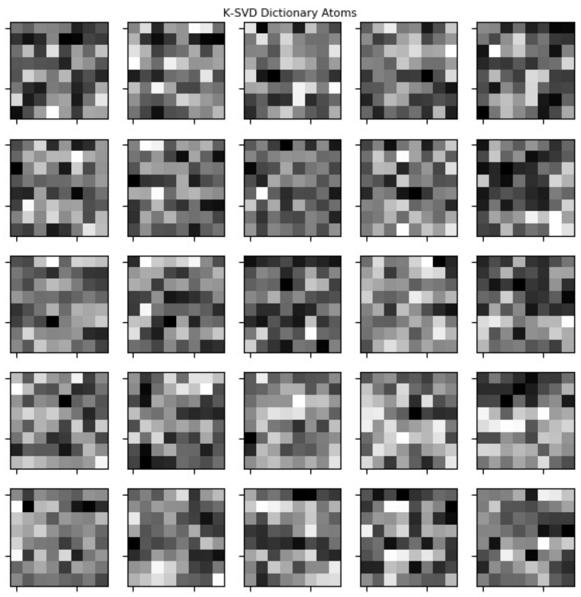

Figure 5: Images of atoms with 25 smallest standard deviation.

4.3 K-SVD Reconstruction (1.5 points)

Use $\mathbf{A}_{ksvd}$ to reconstruct all the block patches of $y_{test\_missing}(n_1, n_2)$. Concatenate these reconstructed patches together to get $\hat{y}_{test\_ksvd}(n_1, n_2)$. Plot $\hat{y}_{test\_ksvd}(n_1, n_2)$, and report PSNR($y_{test}$, $\hat{y}_{test\_ksvd}$) and SSIM(y_{test}, $\hat{y}_{test\_ksvd}$).

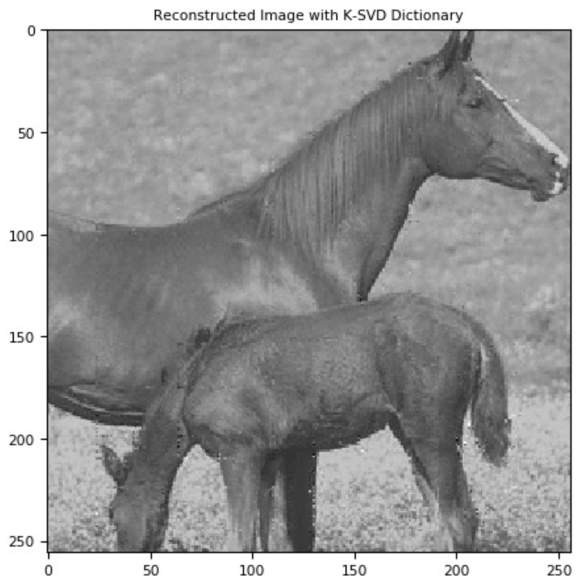

Figure 6: Image of $\hat{y}_{test\_ksvd}(n_1, n_2)$.

$$\mathrm{PSNR}(y_{test}, \hat{y}_{test\_ksvd}) = 18.4620$$
$$\mathrm{SSIM}(y_{test}, \hat{y}_{test\_ksvd}) = 0.8906$$