ECE 269-Project: Sparse Dictionary Learning

Gokce Sarar, *A53244992*

I. Objective

Sparse signal representation by using overcomplete dictionaries has been a widely researched area in the recent years. An overcomplete dictionary $\Phi \in C^{MxN}$ would have more columns than its rows (N>M) and the columns of a dictionary are called atoms. The sparse signal representation problem is representing a signal as a linear combination of the atoms of a given dictionary. However having more columns, the dictionary would be column rank deficient, so that if we want to represent y as a linear combination of dictionary atoms, there would be infinite ways to represent it. The problem is same as finding solutions to the system of linear equations:

$$y = \Phi x \tag{1}$$

where there are infinitely many x vectors which satisfies this equation. However, since we are looking for a sparse representation, if we look for the sparsest x then we would eliminate the results with more nonzero elements. Then the problem becomes the following:

$$\min_x ||x||_0 \tag{2}$$
$$\text{subject to } y = \Phi x$$

where the l_0-norm is the count of the number of non-zero elements in x.

There are different kinds of dictionaries which are commonly used such as DFT, DCT and Wavelet dictionaries and each of these perform differently for obtaining a sparse representation, given a class of signals. That's why the question arises if there can be an optimal dictionary for representing a class of signals and if this dictionary can be learned from them so that new signal members of this class can be represented with at most a sparsity level of m with use of this dictionary. This is the goal of this project. First Orthogonal Matching Pursuit algorithm will be implemented for getting the sparse representations of the signals. Then K-SVD algorithm as a dictionary learning algorithm will be implemented and lastly, the performance of this dictionary will be tested on a image denoising task, so that by examining the SNR increase in the image, the performance of the dictionary can be evaluated.

II. Background and motivation

As stated above, finding the sparsest representation of a signal reduces itself to equation (2), so that we can eliminate the solutions of the problem in (1), which have more nonzero elements. Nevertheless, here there is still the question if the sparsest solution is unique, because after eliminating non-sparse solutions, if we still can have infinitely many solutions, the representation of the signal may vary and this causes problems in the applications. The Theorem 1 in [1] answers this question as follows:

The spark of a dictionary $\Phi, \sigma\{\Phi\}$, is the smallest number of columns that form a linearly dependent set and a linear representation over m atoms (i.e., $||x||_0 = m$) is unique if

$$m < \frac{\sigma\{\Phi\}}{2} \tag{3}$$

So by searching for the sparsest representation we can represent y uniquely.

There is another practical problem with equation 2. In practical applications, the signals which we want to represent sparsely may not be in a noise-free form. We should be able to represent noisy signals sparsely, so that we can apply sparse representation problem to real life scenarios. Following [2] the reformulation for noisy signals can be done.

In [2], they approached the sparse representation problem as a basis selection problem by using Sparse Bayesian Learning (SBL). As said we are looking for the sparsest x to represent y, which means we try to represent y with a minimum number of atoms, which is equivalent to represent it with a minimum number of basis vectors. x vector is treated as a weight vector, whose entries are mostly zero. In the light of Bayesian learning, a parameterized prior is introduced on the weights, so that sparse representation is strengthened. They said that the state-of-art basis selection algorithms introduces a regularizing weight prior such as:

$$p(x) \sim exp\left(-\sum_{i=1}^{M} |x_i|^p\right) \tag{4}$$

where $p \in [0, 1]$. It is said that such priors encourage the sparsity due to their heavy tails and sharp peak at zero. Given this prior, maximum a posteriori (MAP) solutions to

$$y = \Phi x + \varepsilon \tag{5}$$

where ε is the noise are formulated as

$$x_{MAP} = arg \max_x p(x|y)$$
$$= arg \min_x -log(p(y|x)) - log(p(x))$$
$$= arg \min_x \nu ||y - \Phi x||^2 + \sum_{i=1}^{M} |x_i|^p \tag{6}$$

where a Gaussian likelihood model is assumed and ν represents a trade-off parameter, which balances sparsity with quality of fit. And they say that in the absence of noise where $\nu \to \infty$ the solutions take the form

$$x = arg \min_x \sum_{i=1}^{M} |x_i|^p \tag{7}$$
$$\text{subject to } y = \Phi x$$

such that the log prior becomes the objective function over the constraint surface given by $y = \Phi x$.

Till now these formulations were very different than the original problem whose solution was (2) in the absence of noise. However when $p \to \infty$, the exponent of the prior (4) becomes an l_0-norm, such that (7) becomes equivalent to (2) when $p \to \infty$.

So as can be seen (6) would be the reformulation of the sparse representation problem when noise is added to the signals if we let $p \to \infty$. So the reformulation is as follows:

$$= arg \min_x \nu||y - \Phi x||^2 + ||x||_0 \tag{8}$$

Notice that this equation can be written as follows

$$= arg \min_x ||\Phi x - y||^2 + \mu||x||_0 \tag{9}$$

where $\mu = \frac{1}{\nu}$. The reason I reformulate like this is the fact that authors Michael Elad and Michal Aharon use this formulation in [7] and derive denoising algorithm using this equation. Since in the next steps of this project , their denoising algorithm is used, I am reformulating equation (8) as (9), so that the transition would be easier, when explaining their method in the following sections.

As stated in section I, main goal of this project is learning a dictionary given a class of signals and then use this dictionary for denoising purposes. In [3], the authors presented a brief presentation for the state-of-art dictionary learning techniques. I will follow their paper to discuss the three commonly used techniques in dictionary learning in the following.

A. Maximum Likelihood Methods

The equation (5) is used where ε is a Gaussian white residual vector with variance σ^2. The dictionary is tried to be found which maximizes the likelihood function P(Y|Φ), where $Y = \{y_i\}_{i=1}^N$. There are two assumptions made in this methods.

Assumption 1: The measurements $y_i's$ are drawn independently. Assumption 2: There is a hidden variable x such that

$$P(y_i|\Phi) = \int P(y_i, x|\Phi)dx = \int P(y_i|x, \Phi)P(x)dx$$

Due to Gaussian white residual

$$P(y_i|x, \Phi) = \alpha \cdot exp\left\{\frac{1}{2\sigma^2}||\Phi x - y_i||^2\right\}$$

A prior for x is introduced such that the entries of x are zero-mean i.i.d. Olshausen and Field suggested manipulations for solving the integral [4]. So after manipulations [3], the problem of dictionary finding becomes

$$\Phi = arg \min_\Phi \sum_{i=1}^N \min_{x_i} \left\{||\Phi x_i - y_i||^2 + \nu||x_i||_1\right\} \tag{10}$$

This formulation penalizes the entries of x_i but not Φ. So l_2-norm of each atom is constraint such that the output variance of the coefficients is kept a plausible level.

An iterative method was suggested for solving (10) by calculating the coefficients x_i with a simple gradient descent procedure and updating the dictionary using the following formula in each iteration.

$$\Phi^{n+1} = \Phi^n - \eta \sum_{i=1}^N (\Phi^n x_i - y_i)x_i^T \tag{11}$$

This iterative algorithm can be seen as a generalization of K-means algorithm.

B. The MOD Method

MOD is abbreviation for method of optimal directions. It is suggested by K. Engan. The MOD method has two main stages: a sparse coding stage which uses either Orthogonal Matching Pursuit (OMP) or the Focal Underdetermined System Solver (FOCUSS) and a dictionary update stage. I follows K-means outline more closely. The dictionary update is done very simply, which is the main contribution of this method. Errors are defined as $e_i = y_i - \Phi x_i$ so that the mean square error is

$$||E||_F^2 = ||[e_1, e_2......e_N]||_F^2 = ||Y - \Phi X||_F^2 \tag{12}$$

where $||\cdot||_F$ is the Frobenius norm and defined as $||T||_F = \sqrt{\sum_{ij} T_{ij}^2}$.

In the dictionary update stage X is fixed so the Φ is updated to minimize this error. The derivative of 12 is taken with respect to Φ and the following update equation is found

$$\Phi^{n+1} = YX^{(n)^T}\left(X^{(n)}X^{(n)^T}\right)^{-1} \tag{13}$$

The interesting thing with MOD is if the iteration in (11) is done infinite many times and a small enough η is used, the MOD update matrix in (13) is reached as the steady state outcome. So MOD accepts the coefficients as known in each iteration and does the dictionary update, whereas the ML method by Olshausen and Field gets closer to the best current solution and then calculates the coefficients.

C. Maximum Aposteriori Probability Approach

MAP approach is suggested by the researches of MOD in order to combine the efficiency of MOD with preferences in the recovered dictionary. Here instead of maximum likelihood $P(Y|\Phi)$ the posterior $P(\Phi|Y)$ is used. According to Bayes Rule the posterior $P(\Phi|Y)$ is proportional to $P(Y|\Phi)P(\Phi)$ such that the previous likelihood can be used after addition of a dictionary prior.

If there isn't any prior then the update formula is the same as Olshausen and Field in (11). However, the following update formula is created by adding a prior, which constraints Φ to have a unit Frobenius norm:

$$\Phi^{(n+1)} = \Phi^{(n)} + \eta EX^T + \eta \cdot tr\left(XE^T\Phi^{(n)}\right)\Phi^{(n)}$$

The first two term are the same as in (11), whereas the last term serves for the compensation of the deviations caused by the constrained. So different atoms can have different norms, such that atoms with small norms are used less due to the fact that their coefficients would be larger and penalized more.

In order to prevent it another prior is introduced, which makes all atoms have a unit l_2-norm. So the new update formula is as follows

$$\phi_i^{(n+1)} = \phi_i^{(n)} + \eta \left(I - \phi_i^{(n)} \phi_i^{(n)^T} \right) E \cdot x_i^T$$

where x_i^T is the i^{th} column of X^T

It is a slower training algorithm in comparison to MOD. Nevertheless it has offered promising results.

D. Desired Propertied of Dictionary Training Algorithms

In [3], the authors listed the properties, which are desired for dictionary training algorithm.

First of all they made the observation that almost all these algorithm are generalizations of K-means algorithm and they stated that there are four main properties, which good dictionary training algorithms should have: flexibility, simplicity, efficiency and well-defined objective.

Flexibility is important in the sense that they should work with any pursuit algorithm, so that the proper ones for the goal can be chosen. MOD and MAP-based methods have this property.

Simplicity is also a very desirable property and the authors explains it being similar to K-means, which is very explainable and implementable. MOD is very successful in that sense.

As very understandable, the algorithm should be efficient and have fast convergence. Nevertheless, all these methods are pretty slow.

And lastly, a good algorithm should have a measure of the solution quality. Even though some algorithms improve mean square errors and sparsity, their global objective measure of quality gives oscillations and a well defined objective would serve their evaluation in that sense.

E. Uniqueness of Dictionary Learning Problem

As explained before, we have the linear system equations $Y = \Phi X$, we try to find the best dictionary for a signal family. The question is if this best solution is unique. So this question can be formulated as follows: Given the matrix Y, can it be factorized into two matrices Φ and X such that Φ has normalized columns and X has at most m non-zero entries in each column.

In [1], the authors studied the uniqueness of this factorization problem ad they proved that given some conditions, the uniqueness is guaranteed except the trivial permutation and sign-changes of the columns of Φ. These conditions are the number of given signals, (i.e columns of Y), the cardinality of the columns in X and the specific properties of the desired dictionary Φ. So the three necessary assumptions for the uniqueness are as follows:

1. Support: The support of all representation vectors, i.e. the columns of X must satisfy equation 3 with $m = L$, which guarantees that for a signal, a linear representation over L atoms is existing and unique. Moreover, they assumed that L is known. 2. Richness: There should be at least L+1 signals in Y for every possible combination of L dictionary elements from Φ. So, there should be at least $(L+1)\binom{N}{L}$. They also say that this can be relaxed, but that they keep it for the simplicity of their proof. 3. Non-degeneracy: L+1 signals which are created by the same L atoms should have a rank of L and not less(The expectation would be a rank of L or less, but for uniqueness it is accepted to be L). And secondly, L+1 signals which are created with different support should have a rank of L+1. With these assumptions there won't be any degeneracies in the construction of the signals, which means that there won't be any degenerate coincidences.

So they give the result based on these assumption as Theorem 3 in [1]:

Under these assumptions the factorization of Y is unique, such that Φ with normalized columns and X with L non-zeros in each column, is unique. Nevertheless, a right-multiplication of Φ by a signed permutation matrix, which doesn't change the desired properties of Φ and X is the exception of this uniqueness.

The related proof can be seen in [1].

III. Methods and Techniques

1) Orthogonal Matching Pursuit: OMP is a greedy algorithm, which is used to find the sparse representations of signals when the dictionary is given. In each iteration one of the atoms of the dictionary is chosen, which is most correlated to the residual. The residual is the part of y, which is remained after representing y with the so far chosen atoms. After choosing the most correlated atom, a new residual is calculated and it continues with new iteration. If the sparsest level of the representation is m, then after m iterations, the algorithm should find the correct set of atoms. In [6], the author investigated under which conditions, OMP succeeds finding the correct atoms and perform exact recovery. But before reviewing these conditions, below the OMP algorithm can be seen. As input to OMP we should give the dictionary $\Phi \in C^{MxN}$, N-dimensional vector y, whose sparse representation is tried to be found and the sparsity level m.

```
Initialize residual $r_0 = y$, index set $\Psi_0 = \{\}$, chosen
atoms $\Phi_0 = \{\}$, counter $t = 1$;
**while** $t <=$ *sparsity level* $m$ **do**
    Find index $\psi_t$;
    $\psi_t = argmax_{j=1:N} |\langle r_{t-1}, \phi_j \rangle|$;
    Update index set and chosen atoms;
    $\Psi_t = \Psi_{t-1} \cup \{\psi_t\}$;
    $\Phi_t = [\Phi_{t-1} \;\; \phi_{\psi_t}]$;
    Get new signal estimate;
    $z_t = arg \min_z ||y - \Phi_t z||_2$;
    Calculate new residual;
    $r_t = y - \Phi_t z_t$;
    $t = t + 1$;
**end**
Approximation of data $a_m = \Phi_m z_m$;
Sparse signal representation $\hat{x} = zeros(M, 1)$;
$\hat{x}(\Psi_m) = z_m$;
```

Algorithm 1: OMP algorithm

It should be emphasized that the residual is always orthogonal to the chosen atoms [5], due to projection properties and that's why OMP never chooses the same atom twice.

As said before in [6], the author gave the sufficient condition for OMP to recover the exact sparsest solution.

Say the exact recovered columns would be Φ_m. So, if for each of the remaining atoms in the original dictionary Φ, the following condition holds, OMP recover the sparsest representation of the input signal:

$$||(\Phi_m^H \Phi_m)^{-1} \Phi_m^H \phi_j||_1 < 1 \tag{14}$$

where $\phi_j \notin \Phi_m$. So if this condition holds for every signal with a sparsity level of m, the OMP will recover the representation exactly in m steps, which means that in none of the iterations an atom is chosen which isn't the part of the original signal. The proof can be seen in [6].

2) K-SVD: K-SVD is created by Michal Aharon, Michael Elad and Alfred Bruckstein [3]. It is an algorithm for training dictionaries based on the given signals $Y = \{y_i\}_{i=1}^R$. It is assumed that there is a dictionary Φ, whose sparse combinations create Y. The optimum dictionary Φ is seeked which is used in (2). They formulate their objective function as the following:

$$\min_{\Phi, X} \left\{ ||Y - \Phi X||_F^2 \right\}$$

$$\text{subject to } \forall i, ||x_i||_0 \leq m$$

There are mainly two steps in the iterative algorithm. In the first step given a fixed dictionary the sparse representation matrix X is found and then int the next step a better dictionary is searched. Each columns of the dictionary is updated individually and following each updated column the new coefficients are calculated, which makes the convergence faster. Due to this sequentially update, the authors denote this method as a generalization of K-means algorithm[3].

For the sparse coding stage the $||Y - \Phi X||_F^2$ can be written as $\sum_{i=1}^R ||y_i - \Phi x_i||_2^2$ so that for the first part we have R distinct problem of

$$\min_{x_i} \left\{ ||y_i - \Phi x_i||_2^2 \right\}$$

$$\text{subject to } \forall i, ||x_i||_0 \leq m \text{ for } i = 1, 2, ..., R$$

which can be solved by any pursuit algorithms.

For the second stage, both Φ and X are treated fixed except one column of Φ, say ϕ_k and the coefficients corresponding to it x_T^k, which is the k^{th} row in X. $||Y - \Phi X||_F^2$ can be written as

$$\left\| Y - \sum_{j=1}^N \phi_j x_T^j \right\|_F^2$$

$$\left\| \left(Y - \sum_{j \neq k} \phi_j x_T^j \right) - \phi_k x_T^k \right\|_F^2$$

$$||E_k - \phi_k x_T^k||_F^2 \tag{15}$$

E_k is the error when the k^{th} atom is removed.

By using SVD E_k can be find directly, which minimizes the error. However, by doing that we wouldn't put a sparsity constraint on the x_T^k so that it would be probably filled. In order to overcome this problem, authors suggested the following approach. They defined a vector w_k as a group of indices where $x_T^k(i)$ is non-zero, such that

$w_k = \left\{ i | 1 \leq i \leq N, x_T^k(i) \neq 0 \right\}$

Then they define a matrix $\Omega_k \in C^{Rx|w_k|}$ which has zeros everywhere except on the $(w_k(i), i)$th entries, where there are ones. If x_T^k is multiplied by Ω_k, then we receive a shrinked row vector $x_S^k = x_T^k \Omega_k$, where we discard the zero entries of x_T^k. And multiplication $Y \Omega_k = Y_k^S$ gives the subset of signals which use ϕ_k atom. Same is true for $E_k \Omega_k = E_k^S$. So the minimization problem in (15) becomes equivalent to

$$||E_k^S - \phi_k x_S^k||_F^2$$

where we take the sparsity into account. By applying SVD on the shrinked matrix $E_k^S = U \Delta V^T$, the solution for $\widetilde{\phi_k}$ is defined as the first column of U and the coefficient vector x_S^k as the first column of V multiplied by $\Delta(1,1)$. Then this procedure is applied to all columns of the dictionary Φ by keeping the already updated atoms and coefficients.

It should be noted that in this procedure all atoms remain normalized and the support of all representations either stay the same or got smaller. Given the fact that the pursuit algorithms can recover the sparsest representation, the authors show that the convergence to a local minimum is guaranteed [3]. That's why the performance of K-SVD depends on the performance of pursuit algorithms tightly.

3) Denoising: The authors of [7], suggests that we can find the sparse representations of the noisy images so that we can get rid of the noise components by approximating the images with sparse vectors, which have more zero components. Eventually, this leads to denoising of a noisy image. Firstly, they answer why an image of big size can be treated with small dictionaries, which leads to treat the big image as patches. They suggest two reasons for this: firstly a small dictionary means locality of the algorithms so that the overall image treatment is simplifies and secondly, if we learn a dictionary for sparse representation inf the training small dictionaries are created, which is a benefit computationally. Although there might be artifacts on the patch boundaries, if overlapping patches are used and their results are averaged, this artifacts can be beaten.

So in order to process an image as small patches, they reformulate the equation 9 as the following:

$$\left\{ \hat{x}_{i,j}, \hat{B} \right\} = arg \min_{x_{i,j}, B} \lambda ||B - Y||_2^2$$

$$+ \textstyle\sum_{i,j} \mu_{i,j} ||x_{i,j}||_0 + \sum_{i,j} ||\Phi x_{i,j} - R_{i,j} B||$$

the first term is for the similarity between the measured image Y and its denoised version B, which we don't know. The second and the third terms are the image priors, which provides that every patch $b_{ij} = R_{i,j} B$ has a sparse representation with a bounded error. The matrix R_{ij} extracts the ij^{th} block of the image. μ_{ij} is relative to all patches, since it deals with the constraints $||\Phi x_{ij} - b_{ij}||_2^2 \leq T$, where T is the error bound.

If the dictionary is known, this denoising formula can be handled in two easier problems. First,

$$\hat{x}_{ij} = arg \min_{x} \mu_{ij}||x||_0 + ||\Phi x - b_{ij}||_2^2$$

can be solved for each image patch with OMP by stopping when $||\Phi x - b_{ij}||_2^2$ goes below an error bound of T so that μ_{ij} is considered implicitly.

Once we have $\hat{x}_{ij}$, by fixing them B can be updated by solving

$$\hat{B} = arg \min_{B} = \lambda||B - Y||_2^2 + \sum_{ij} ||\Phi \hat{x}_{ij} - R_{ij}B||$$

which has a closed form solution

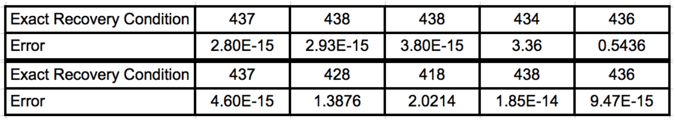
$$\hat{B} = \left(\lambda I + \sum_{ij} R_{ij}^T R_{ij}\right)^{-1} \left(\lambda Y + \sum_{ij} R_{ij}^T \Phi \hat{x}_{ij}\right) \quad (16)$$

This would be the main formula for the following denoising implementation. As dataset I used the Faces 1999(Front) database of Caltech Computational Vision Group.

IV. Results

A. Orthogonal Matching Pursuit

In order to implement OMP and test my implementation, I used a DCT dictionary of size 64x441 and chose three atoms randomly to create a data signal. Than I gave this data signal, DCT dictionary and the sparsity level of 3 to my OMP implementation and measured the l_1-norm of the difference between the data signal and the approximated signal by OMP, where the approximated data is the multiplication of the dictionary with the sparse representation of the data signal. In Fig. 1, the results of my implementation can be seen.



Exact Recovery Condition	437	438	438	434	436
Error	2.80E-15	2.93E-15	3.80E-15	3.36	0.5436
Exact Recovery Condition	437	428	418	438	436
Error	4.60E-15	1.3876	2.0214	1.85E-14	9.47E-15

Fig. 1. Results of OMP implementation

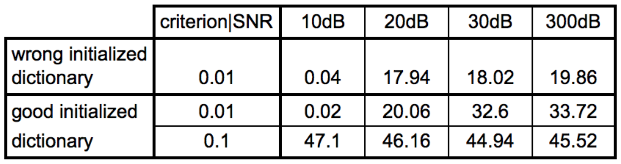
As can be seen in 6 out of 10 trial for recovering the sparse representation of the signal, the error is almost 0, so that OMP recovered the signal exactly. However in other 4 trial, the error is bigger than 0, which I wouldn't expect, since I give the exact dictionary to the algorithm, with which I created the signals. However, as I explained above, OMP doesn't always recover the sparse representation and I also stated the sufficient condition in equation (14). So in all these tests I also checked the sufficient condition. Since I generate the data signals, I know the generating atoms, so that I applied the sufficient condition check to the matrix of generating atoms with each other atom of the original dictionary. Since the total number of the atoms is 441 and since I have 3 generating atoms, the exact recovery condition should be smaller than one for 438 remaining atoms. The numbers corresponding to exact recovery condition in Fig. 1, are the number of atoms, for which the exact recovery holds. As can be seen if the number is 438, I definitely got an almost zero error. For 437 I also got an almost zero error, which may happen, because the exact recovery condition guarantees the exact recovery but the lack of condition doesn't prevent necessarily the exact recovery. And with a number 436, I got both exact recovery and a bad recovery with an error of 0.5436. And in every other bad recovery I got, the ER condition was holding less than 438 atoms. So as a result, I can conclude that when ER holds, I get exact recovery and when there isn't ER I may or may not get an exact recovery.

B. K-SVD

In order to implement K-SVD, I created a random matrix Φ as the generating matrix with i.i.d uniformly distributed entries, whose size is 20x50 and whose columns are normalized to l_2-norm as described in [3]. I followed the method of the authors by creating 1500 data signals, which are linear combinations of three different generating atoms, with uniformly distributed i.i.d. coefficients in random and independent locations. I added white Gaussian noise with different SNR to these data signal and initialized the dictionary with the data signals for the K-SVD algorithm again as described in [3] and iterate the K-SVD algorithm 80 times. Lastly, I performed 50 trials to imitate the results of the authors. I checked the success of the generated dictionary with measuring the distance by $1 - |\phi_i^T \tilde{\phi}_i|$, where $|\phi_i^T$ is the generating dictionary atom and $\tilde{\phi}_i$ is the corresponding atom in the recovered dictionary. It is found by sweeping through the atoms of the recovered dictionary and finding the closest atom to the relevant generating atom. The authors considered a distance less than 0.01 as a success.

At first my random dictionary was generated between -1 and 1by using the code line "2.*rand([20 50])-1" and my recovery rate for 30 dB was 18.02 out of 50 atoms on average of 50 trials. When I checked changing the success criterion from 0.01 to 0.1, I saw that the result was not changing because some of the atoms had a distance between 0.5 to 0.8, which apparently showed that my code was not working. After trying different fixing options I generated the random dictionary with the following code line "rand([20 50])" and my recovery rate became 32.6 out of 50 on average which was almost double of my first trial but still not as nearly good as the authors. Nevertheless, when I checked changing success criterion, I saw that this time non of the atoms had big distance above 0.2, which indicates definitely that my code was working and changing the dictionary improved the results. However, I couldn't solve the underlying reason for this difference. In the first code line I was trying to have 0 mean atoms and I was normalizing all of the atoms as I do for my second experiment too. That's why I couldn't find out the reason, but this result can be regenerated with my attached code.

To sum up, I continued with this code for the next parts of the project, since it gives reasonable results with the second dictionary. In Fig 2, the results of two experiments can be seen and for the second experiment I have results for both the success criterion as 0.01 and 0.1. And it can be concluded that for the second dictionary, the recovery is very successful with a success criterion of 0.1. The success matrices are also attached.



	criterion|SNR	10dB	20dB	30dB	300dB
wrong initialized dictionary	0.01	0.04	17.94	18.02	19.86
good initialized dictionary	0.01	0.02	20.06	32.6	33.72
	0.1	47.1	46.16	44.94	45.52

Fig. 2. Results K-SVD algorithm on synthetic data

As required, I also learned a dictionary from the image in Fig 3, although I misunderstood the problem statement as learning a dictionary from a clean image and then using it for denoising the image after addition of noise. Due to the fact that learning and denoising algorithms run for a long duration, I couldn't repeat the experiments by learning a dictionary from different images and using this dictionary on a different test image. That's why in the following, all my results related to learned dictionary are given with respect to the dictionary I learned from this image. I initialized K-SVD with a DCT dictionary of size 64x441 and iterated the algorithm for 80 times and in these iterations I didn't change the first atom as the authors suggested, since it is for DC component. I trained on 11000 patches as suggested in the paper. This dictionary can be seen in Fig. 4. As can directly be concluded, This dictionary resembles the DCT dictionary very much, which is an indication that I couldn't generate a very different dictionary than DCT dictionary, which can be seen in Fig. 5. It is probably due to the fact that 80 iterations are not enough for generating a different dictionary in a greater scale. Nevertheless, I can comment on their performances in the next section where I compare their performance on denoising.

Fig. 3. The clean image used for learning dictionary

C. Denoising

In this part, I implemented the denoising algorithm in [7], with my generated dictionary, DCT dictionary and the global dictionary of the authors, which is used in [7]. I downloaded this dictionary with 256 atoms from their web page [8](DC component was missing in the resource, so I added it). It can be seen in Fig. 6.

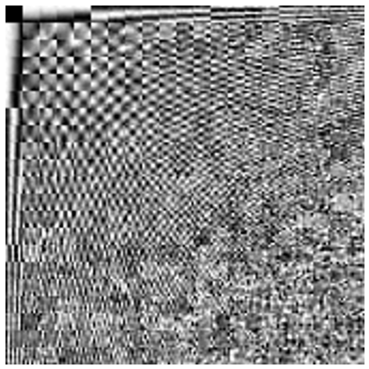


Fig. 4. The learned dictionary

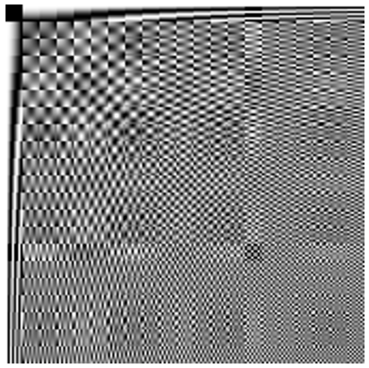


Fig. 5. DCT dictionary

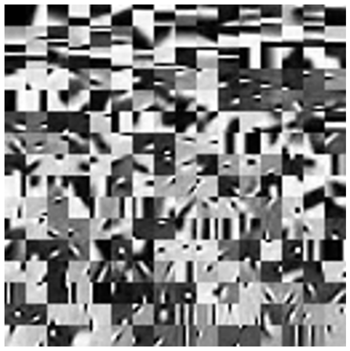


Fig. 6. Global dictionary which is used in [7]

For denoising I used an OMP with an error bound as described in Denoising part of section III. 8x8 patches from noisy images are given to OMP algorithm and instead of using given sparsity level, the error bound is checked and if the approximated patch has an error less than this level, OMP stops for the corresponding patch. As the authors suggested I used a threshold of $\epsilon = 1.15\sigma$. Since the authors did one iteration of iterative denoising, I also implemented that way. Their reasoning was, they wouldn't know the σ value for the next iterations so that a threshold couldn't be set for the next iterations. For the λ value in equation 16, they suggested to use $\lambda = \frac{30}{\sigma}$. As I used this value, I couldn't get a significant improvement, and by trying other values, I recognized that when λ gets bigger up to a level like 50, my SNR values get better, which is an unexpected result and I will discuss it in

Discussion part. In Fig. 7 and 8, the results of my experiments can be seen.

The detailed critique of my results are going to be done in the next Discussion part, so that here I will just state which results are superior and show three examples of denoising in Fig 9, 10 and 11, since showing examples for all experiments would take too much space. As can be seen, for Gaussian noise with $\sigma = 10$ and 30, the learned dictionary gives the best results, whereas for $\sigma = 20$ DCT dictionary gives the best results and moreover, in all of them the results of three dictionaries are different only in the decimal points. For speckle noise with $\sigma = 10$ and 20, the learned dictionary gives the best results, whereas for $\sigma = 30$, the global dictionary gives the best results. In Fig. 9, the image with a Gaussian noise of $\sigma = 10$ can be seen and at the first glance, there can't be seen a significant difference. Nevertheless if the chin area of the woman is inspected, denoising becomes clear. For the images in Fig. 10 and 11, the denosing is more clear, although the images are still noisy. Although it was expected to get slightly noisy images after this denoising algorithm, those images have a very high level of noise. The fact that the images remain noisy at this level indicates that my algorithm can be improved, since the authors could achieve a denoising at a much more signigficant level [7]. A probable way of improvement is implementing this algorithm with more iterations, although the authors could achieve their success with only one iteration.

It should also be stated that the same standard deviation values for Gaussian and Speckle noise create different SNR values, so that they can't be compared one-to-one. If I want to compare the effect of the dictionaries for different noise types, I should compare Gaussian Noise with $\sigma = 10$ and Speckle Noise with $\sigma = 20$, for which the noisy images have almost the same SNR and PSNR values, although they are not the same. I can say that for both noises, the learned dictionary performs better than the other dictionaries, but for Gaussian it differs in the second decimal point, where as for Speckle it differs in the first decimal point from the other dictionaries, so that I may say it performs better for the Speckle noise then it performs for the Gaussian Noise.

		psnr	snr	psnr	snr	psnr	snr
gaussian noise	sigma	10	10	20	20	30	30
dictionary	noisy image	28.100	22.560	22.340	16.700	18.970	13.330
learned	lambda=50	***31.121***	***25.481***	25.210	19.540	***21.820***	***16.180***
	lambda=30	30.710	25.069	24.778	19.138	21.398	15.758
	lambda=10	29.443	23.802	23.533	17.892	20.150	14.509
	lambda=30/sigma	28.654	23.013	22.552	16.912	19.099	13.458
dct	lambda=50	31.114	25.473	***25.221***	***19.580***	21.791	16.150
	lambda=30	30.708	25.067	24.812	19.176	21.383	15.742
	lambda=10	29.449	23.808	23.554	17.916	20.129	14.488
	lambda=30/sigma	28.664	23.023	22.566	16.926	19.074	13.433
global	lambda=50	31.113	25.472	25.211	19.570	21.816	16.175
	lambda=30	30.701	25.061	24.809	19.169	21.409	15.768
	lambda=10	29.439	23.799	23.559	17.918	20.155	14.514
	lambda=30/sigma	28.654	23.013	22.571	16.931	19.100	13.459

Fig. 7. Experiments with denoising Gaussian Noise

		psnr	snr	psnr	snr	psnr	snr
speckle noise	sigma	10	10	20	20	30	30
dictionary	noisy image	33.763	28.122	27.775	22.134	24.418	18.608
learned	lambda=50	***36.697***	***31.056***	***30.701***	***25.060***	27.146	21.505
	lambda=30	36.262	30.621	30.260	24.620	26.726	21.085
	lambda=10	34.973	29.332	28.964	23.323	25.461	19.820
	lambda=30/sigma	34.171	28.531	27.968	22.328	24.418	18.777
dct	lambda=50	36.686	31.045	30.680	25.039	27.158	21.518
	lambda=30	36.260	30.619	30.247	24.606	26.726	21.085
	lambda=10	34.970	29.330	28.960	23.319	25.445	19.804
	lambda=30/sigma	34.165	28.524	27.966	22.325	24.391	18.750
global	lambda=50	36.667	31.026	30.675	25.034	***27.181***	***21.541***
	lambda=30	36.237	30.596	30.249	24.608	26.753	21.113
	lambda=10	34.956	29.316	28.974	23.334	25.470	19.829
	lambda=30/sigma	34.159	28.518	27.992	22.351	24.416	18.775

Fig. 8. Experiments with denoising Speckle Noise

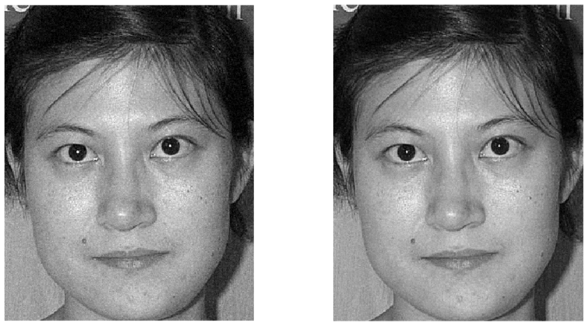


Fig. 9. (Left) Image denoise with Gaussian noise($\sigma = 10$) having a SNR of 22.560 and PSNR of 28.100. (Right) Denoised image having a SNR of 25.481 and PSNR of 31.121. Denoising with Learned Dictionary

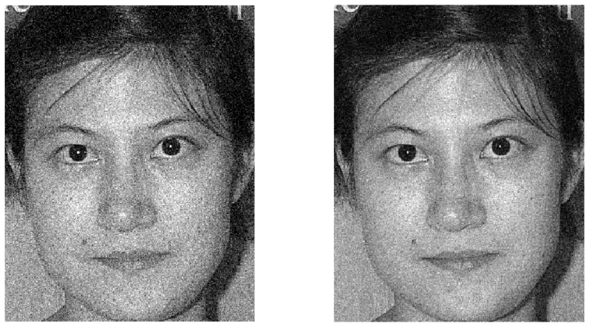


Fig. 10. (Left) Image denoise with Gaussian noise($\sigma = 30$) having a SNR of 13.330 and PSNR of 18.970. (Right) Denoised image having a SNR of 16.180 and PSNR of 21.820. Denoising with Learned Dictionary

V. Discussion

I have already stated why I believe that my OMP and K-SVD algorithm work in a good way in the Results part, where I showed that I got reasonable results for their performance. Since they are the key elements in the denoising algorithm and I found them reasonably functioning, I expected my denoising results would be satisfactory, which are not(I will explain in the following). The probable reason for that may be the fact that my K-SVD algorithm was giving good results when the success criterion was 0.1. So it may be the fact that it can't learn a dictionary as perfect as it should be so that I couldn't

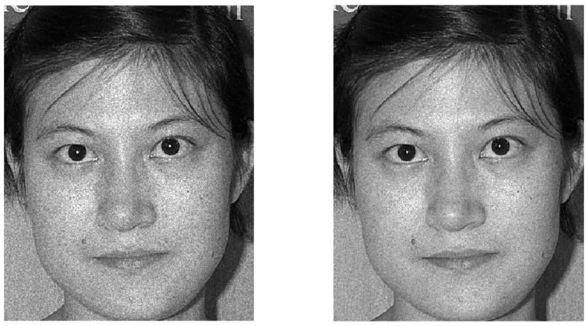


Fig. 11. (Left) Image denoise with Speckle noise($\sigma = 30$) having a SNR of 18.608 and PSNR of 24.418. (Right) Denoised image having a SNR of 21.541 and PSNR of 27.181. Denoising with Global Dictionary

get better results in denoising. As a second possibility, I only ran K-SVD for 80 iterations, due to computational reasons and that probably caused the dictionary not to be learned as good as it should be. And lastly, running the denoising algortihm for more iterations could be another improvement, as I stated in the previous part.

I denoted my results as unsatisfactory, because although I learned the dictionary from the same image, its results are very similar to the results of DCT Dictionary and the Global Dictionary and moreover the resultant images are still noisy at a serious level. So due to the reasons I stated in the previous paragraph, I may have failed to create a very good learned dictionary.

Secondly, as I stated the authors use a λ value of $\frac{30}{\sigma}$, which is for example 1 for $\sigma = 30$, whereas I get my best results for $\lambda = 50$. This is unexpected, because λ is used as the weight of the noisy image, where I average my denoised approximation with the original image in order to have some relaxation. However, when I use $\lambda = 50$, I am almost using the original noisy image as it is and the effect of the approximation which is learned by OMP, is almost not used. This fact actually contradicts to the whole idea. That's why it would be better to experiment with different threshold values of OMP to see if I could get better representations. Nevertheless, I could get denoised images at the end and increased the PSNR and SNR values.

Lastly, as stated before I couldn't apply learning a dictionary from different set of images and denoising a novel image, since both learning and denoising parts take a long time. For denoising, it is partly my fault that I am denoising overlapping patches where overlaps go pixel by pixel. I could have taken overlapping patches by bigger steps. For the future work, I may experiment with such patches and a dictionary which is learned with more iterations.

Nevertheless, I achieved denoising images, although it could have been better. I could observe the effect of the sparse representation of signals in denoising, which was the main purpose of this project. That's why as a conclusion, I may say that I achieved what I planned for this project, although it could have been much better.

REFERENCES

[1] M. Aharon, M. Elad, and A. M. Bruckstein, ”On the uniqueness of overcomplete dictionaries, and a practical way to retrieve them,” Linear Algebra and its Applications, vol. 416, no. 1, pp. 48 - 67, 2006.

[2] D. Wipf and B. Rao, ”Sparse Bayesian Learning for Basis Selection,” IEEE Transactions on Signal Processing, vol. 52, no. 8, pp. 2153 - 2164, 2004.

[3] M. Aharon, M. Elad, and A. Bruckstein, ”K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation” IEEE Transactions on Signal Processing, vol. 54, no. 11, pp. 4311 - 4322, 2006.

[4] B. Olshausen and D. Field, ”Natural image statistics and efficient coding,” Network: Computation in Neural Systems, vol. 7, no. 2, pp. 333 - 339, 1996.

[5] J. A. Tropp, and A. C. Gilbert, ”Signal recovery from random measurements via orthogonal matching pursuit.” IEEE Transactions on information theory, vol. 53, no. 12, pp. 4655-4666,2007

[6] J. Tropp, ”Greed is Good: Algorithmic Results for Sparse Approximation,” IEEE Transactions on Information Theory, vol. 50, no. 10, pp. 2231 - 2242, 2004

[7] M. Elad, M. Aharon. "Image denoising via sparse and redundant representations over learned dictionaries." IEEE Transactions on Image processing, vol. 15, no. 12, pp. 3736-45, 2006.

[8] "Software," Software - Michael Elad's Personal Page. [Online]. Available: http://www.cs.technion.ac.il/ elad/software/. [Accessed: 06 - Dec - 2017].