המעבדה לעיבוד גיאומטרי של תמונות
Geometric Image Processing Laboratory

Faster and Lighter Online Sparse Dictionary Learning

Project report

By: Shay Ben-Assayag, Omer Dahary

Supervisor: Jeremias Sulam

Table Of Contents

1 Abstract

Sparse representation has shown to be a very powerful model for real world signals, and has enabled the development of applications with notable performance. Combined with the ability to learn a dictionary from signal examples, sparsity-inspired algorithms are often achieving state-of-the-art results in a wide variety of tasks. However, most existing methods are restricted to small dimensions, mainly due to the computational constraints that the dictionary learning problem entails. In the context of image processing, this implies handling small image patches instead of the entire image. A novel work which has circumvented this problem is the recently proposed Trainlets framework, where the authors proposed the Online Sparse Dictionary Learning (OSDL) algorithm that is able to efficiently handle bigger dimensions. This approach is based on a double sparsity model which uses a new cropped Wavelet decomposition as the base dictionary, and an adaptive dictionary learned from examples by employing Stochastic Gradient Descent (SGD) ideas.

In continuation to this work, which has shown that dictionary learning can be up-scaled to tackle a new level of signal dimensions, our project is focused on studying and improving OSDL. In this report, we present several modifications to the algorithm which are aimed at dealing with its limitations, results of experiments conducted on high dimensional large datasets, our conclusions and suggestions for future work.

2 Background

2.1 Sparse Coding

A central achievement of classical linear algebra was a thorough examination of the problem of solving systems of linear equations. Surprisingly, within this well-understood arena there is an elementary problem which only recently has been explored in depth: obtaining the sparsest solution to redundant systems, as a way of selecing only one of the possibly infinite solutions.

Consider a full-rank matrix $\mathbf{D} \in \mathbb{R}^{n \times m}$ with $n < m$, and define the underdetermined linear system of equations $\mathbf{Dx} = \mathbf{y}$. This system has infinitely many solutions, but the choice can be narrowed down to a well-defined solution which contains the smallest number of nonzero entries. In other words, a new optimization problem is introduced by:

$$(P_0): \ \min_{\mathbf{x}} \|\mathbf{x}\|_0 \ \text{ s.t. } \mathbf{Dx} = \mathbf{y}$$

where $\|\mathbf{x}\|_0 \coloneqq \#\{i \mid \mathbf{x}_i \neq 0\}$ is the ℓ_0 norm.

The reader may notice that a similiar problem, which revolves around minimizng the ℓ_2 norm, is known as the least squares problem and has a closed solution given by $\mathbf{x} = \mathbf{D}^{\dagger}\mathbf{y}$. This solution is based on the fact that the ℓ_2 norm is a convex and differentiable function, a major difference from the discrete and

discontinuous nature of the ℓ_0 norm. Indeed, it has been proven that (P_0) is, in general, NP-hard.

Today many pure and applied mathematicians are pursuing results concerning sparse solutions. In parallel with this development, another insight has been developing in signal and image processing, where it has been found that many media types (still imagery, video, acoustic) can be sparsely represented using transform-domain methods. In fact many imporant tasks dealing with such media can be fruitfully viewed as finding sparse solutions to underdetrmined system of linear equations. These tasks include famous applications such as noise removal, image deblurring, and signal compression.

The basic assumption of this model is that natural signals can be expressed as a sparse linear combination of atoms, chosen from a collection called a dictionary. Formally, for a signal $\mathbf{y} \in \mathbb{R}^n$, this can be described by $\mathbf{y} = \mathbf{D}\mathbf{x}$, where $\mathbf{D} \in \mathbb{R}^{n \times m}$ is a redundant dictionary that contains the atoms as its columns, and $\mathbf{x} \in \mathbb{R}^m$ is the represenation vector.

The introudction of computationally tractable algorithms for pursuing a sparse solution, such as OMP and SP, suggests that sparsity-deriven signal processing is a potentially useful parctical tool. These algorithms utilize a greedy strategy where a solution is iteratively constructed by performing locally optimal updates on $\mathbf{x}$. We shall explain OMP in greater detail in the next sections.

2.2 Dictionary Learning

A fundamental element in sparse coding, is the choice of the dictionary $\mathbf{D}$. It has been shown that a better representation technique - one that leads to more sparsity - can be the basis for a practically better solution to signal processing problems. Therefore, in order to gain better results, one would like to wisely choose $\mathbf{D}$ to perform well on the signals. While some analytically-defined dictionaries were used originally, learning the dictionary from signal examples for a specific task has shown to perform significantly better.

Assuming a training database $\{\mathbf{y}_i\}_{i=1}^M$ is given, one would like to solve the following optimization problem:

$$\min_{\mathbf{D},\{\mathbf{x}_i\}_{i=1}^M} \sum_{i=1}^{M} \|\mathbf{y}_i - \mathbf{D}\mathbf{x}_i\|_2^2 \text{ s.t. } \|\mathbf{x}_i\|_0 \leq k_0,\ 1 \leq i \leq m$$

Clearly, there is no general practical algorithm for solving this problem, for the same reasons that there is no general practical algorithm for solving (P_0). However, algorithms such as MOD and K-SVD have been able to work around this issue by utilizing a strategy of alternating minimization, switching between the optimization over $\mathbf{D}$ and the optimization over $\{\mathbf{x}_i\}_{i=1}^M$ in each iteraion. Unfortunately, these iterative methods have been restricted to relatively small signals, due to the computational complexity of this problem.

2.3 OSDL

Our project revolves around researching a newly proposed algorithm aimed at tackling the previously menotined issues, called Online Sparse Dictionary Learning (OSDL). Its purpose is training a sparse matrix $\mathbf{A}$, such that the desired dictionary is given by $\mathbf{D} = \mathbf{\Phi A}$, where $\mathbf{\Phi}$ is a 2-D separable Wavelet transform. This double sparsity model cancels the need to store the entire dictionary in memory, and the separability of the transform enables an efficient computation of $\mathbf{Dx}$ by $\mathbf{\Phi}_1 \mathbf{A} \mathbf{x} \mathbf{\Phi}_1^T$, where $\mathbf{\Phi}_1$ contains the basis elements of the 1-D DWT arranged column-wise. Every atom in the effective dictionary $\mathbf{D}$ is therefore a linear combination of few and arbitrary atoms from the base dictionary $\mathbf{\Phi}$. Formally, this means that the training procedure requires solving the following problem:

$$\min_{\mathbf{A},\mathbf{X}} \frac{1}{2} \|\mathbf{Y} - \mathbf{\Phi A X}\|_F^2 \text{ s.t. } \begin{cases} \|\mathbf{x}_i\|_0 \leq p & \forall i \\ \|\mathbf{a}_j\|_0 = k & \forall j \end{cases}$$

While tradional dictionary learning algorithms require many sweeps of the whole training corpus, an infeasible method when learning from a large number of examples, OSDL adopts online learning ideas taken from Stochastic Gradient Decent (SGD). This approach allows to handle very large training sets while using high dimensional singals, achieving faster convergence than the batch alternative and providing a better treatment of local minima, which are abundant in non-convex dictionary learning problems.

In accordance with MOD and K-SVD, OSDL alternates between a sparse coding stage, thorughout which the dictionary is held constant, and a dictionary update stage, in which the sparse coefficients are kept fixed. We shall focus on the second stage, as the first remains unchanged, essentially applying sparse coding to a group of examples by using OMP. The problem to consider in the dictionary update stage is the following:

$$\min_A \underbrace{\frac{1}{2} \left\| \mathbf{E}_j - \mathbf{\Phi} \mathbf{a}_j \mathbf{x}_j^T \right\|_F^2}_{f(\mathbf{a}_j)} \text{ s.t. } \|\mathbf{a}_j\|_0 = k \ \forall j$$

where $\mathbf{\Phi}$ is the base dictionary of size $n \times L$, A is a matrix of size $L \times m$ with k non-zeros per column, $\mathbf{E}_j$ is the error given by $\mathbf{Y} - \sum_{i \neq j} \mathbf{\Phi} \mathbf{a}_i \mathbf{x}_i^T$, $\mathbf{a}_j$ is the j-th column in $\mathbf{A}$, and $\mathbf{x}_i^T$ denotes the i-th row of $\mathbf{X}$. Note that we could rewrite the cost function as $f(\mathbf{a}_j) = \frac{1}{2} \|\mathbf{E}_j - \mathcal{H}_j \mathbf{a}_j\|_F^2$ for an approriate operator $\mathcal{H}_j$. In this way, we can perform gradient descent by iterating:

$$\mathbf{a}_j^{t+1} = \mathcal{P}_k \left[\mathbf{a}_j^t - \eta_j^t \mathcal{H}_j^* \left(\mathbf{E}_j - \mathcal{H}_j \mathbf{a}_j^t \right) \right]$$

where $\mathcal{H}_j^*$ is the adjoint of $\mathcal{H}_j$, $\mathcal{P}_k$ is a hard-thresholding operator that keeps the k largest non-zeros (in absolute value), and η_j^t is an appropriate step-size.

It is not compulsory to accumulate all the examples to perform an update in the gradient direction. Instead, OSDL uses a SGD approach by globally

updating the atoms of the dictionary based on an estimation of the gradient over a random mini-batch of examples $\mathbf{Y}_t$:

$$\mathbf{A}^{t+1} = \mathcal{P}_k \left[\mathbf{A}^t_{\mathcal{S}} - \eta^t \nabla f \left(\mathbf{A}^{\mathbf{t}}_{\mathcal{S}} \right) \right]$$

where the thresholding operator now operates in each column of its argument, and:

$$\eta^t = \frac{\left\| \nabla f \left(\mathbf{A}^t_{\mathcal{S}} \right) \right\|_F}{\left\| \mathbf{\Phi} \nabla f \left(\mathbf{A}_{\mathcal{S}} \right) \mathbf{X} \right\|_F}$$

Using the mini-batch approach enables us to keep an up-to-date instance of the Gram matrix of the dicionary $\mathbf{G}_t = \mathbf{D}_t^T \mathbf{D}_\mathbf{t}$ in order to reduce the computational cost of OMP. In a regular online learning scheme, this would be infeasible due to the need to recompute this matrix for each example. In OSDL, however, the matrix needs to be updated only once per mini-batch, and only a few atoms get updated each time. Moreover, this update can be done efficiently due to the sparsity of $\mathbf{A}$.

Stochastic algorithms often introduce different strategies to regularize the learning process and try to avoid local minimum traps. To do so, OSDL incorporate a momentum term $\mathbf{U}^t$ controlled by a parameter $\gamma \in [0, 1]$, which helps to attenuate oscillations and can speed up the convergence by using information from the previous gradients. In addition, many dictionary learning algorithms include the replacement of (almost) unused atoms and the pruning of similar atoms. These strategies are incorporated as well by checking for such cases once every few iterations.

Data: Training samples $\{\mathbf{y}_i\}$, base-dictionary $\mathbf{\Phi}$,
initial sparse matrix $\mathbf{A}^0$
Initialization: $\mathbf{G}_\mathbf{\Phi} = \mathbf{\Phi}^T \mathbf{\Phi}$; $\mathbf{U} = \mathbf{0}$;
for $t = 1, \ldots, T$ **do**
  Draw a mini-batch $\mathbf{Y}_t$ at random;
  $\mathbf{X}_t \leftarrow$ Sparse Code $(\mathbf{Y}_t, \mathbf{\Phi}, \mathbf{A}^t, \mathbf{G}^t)$;
  $\eta^t = ||\nabla f(\mathbf{A}^t_\mathcal{S})||_F / \|\mathbf{\Phi} \nabla f(\mathbf{A}^t_\mathcal{S}) \mathbf{X}^\mathcal{S}_t\|_F$;
  $\mathbf{U}^{t+1}_\mathcal{S} = \gamma \mathbf{U}^t_\mathcal{S} + \eta^t \ \nabla f(\mathbf{A}^t_\mathcal{S})$;
  $\mathbf{A}^{t+1}_\mathcal{S} = \mathcal{P}_k \left[\mathbf{A}^t_\mathcal{S} - \mathbf{U}^{t+1}_\mathcal{S} \right]$;
  Update columns and rows of $\mathbf{G}$ by
  $\left(\mathbf{A}^{t+1} \right)^T \mathbf{G}_\mathbf{\Phi} \mathbf{A}^{t+1}_\mathcal{S}$
end
Result: Sparse Dictionary $\mathbf{A}$

Figure 1: OSDL

3 Implementation Changes

We concentrated our work upon expanding and improving the implementation of OSDL provided in Matlab. The main features we dealt with include:

- **Providing an alternative pursuit algorithm for the sparse coding stage:** OSDL uses OMP for its sparse coding stage. This algorithm requires as many iterations as the number of non-zeros and has high memory constraints, properties which become prohibitive when managing very large dimensions. In order to cope with these issues, we chose to integrate the implementation with an alternative pursuit algorithm, called SP.

- **Providing a global alternative to the hard-thresholding operator:** In order to make $\mathbf{A}$ sparse, OSDL multiplies each atom by a hard-thresholding operator, resulting in each atom being as sparse as the others. We chose to experiment with another approach, in which the global constraint on the number of non-zero entries in $\mathbf{A}$ is kept, but each atom is free to have a different ℓ_0 norm.

- **Reimplemting the dictionary clearing stage:** OSDL replaces atoms that are used by a low number of training examples, as well as prunes similiar atoms. These cases are checked and dealt with by calling the *cleardict* function every few iterations. We noticed a few problems with the provided implementation of the function and chose to reimplement it.

- **Fixing bugs:** We noticed a few minor bugs in the provided implementation. Those were, of course, fixed.

The above implemented modifications were eventually added as optional features in the OSDL implementaion, easily configurable by the respective parameters, which are documented in the code. These main changes will be explained in detail in the next sections.

3.1 Alternative Sparse Coding Algorithm

As mentioned before, OSDL utilizes Orthogonal Matching Pursuit (OMP) for its sparse coding stage. This algorithm is based on a greedy strategy of performing a series of locally optimal single-term updates. Starting from $\mathbf{x}^0 = 0$ it iteratively constructs a k-term approximation $\mathbf{x}^k$ by maintaining a set of active columns - initially empty - and, at each stage, expanding that set by one additional column. The column chosen at each stage maximally reduces the residual ℓ_2 error in approximating $\mathbf{y}$ from the currently active columns. After constructing an approximant including the new column, the residual ℓ_2 error is evaluated; if it now falls below a specified threshold, or if a target cardinality has been reached, the algorithm terminates.

The computational complexity of OMP depends on the number of iterations needed for exact reconstruction - standard OMP always runs through k iterations, where $k = \|\mathbf{x}\|_0$ is the signal sparsity. Moreover, this algorithm does not have provable reconstruction quality at level of other methods, such as of Linear Programming (LP), where the problem of ℓ_0 optimization is relaxed into a ℓ_1 optimization. However, the complexity of LP techniques is still highly impractical for many applications. Another drawback of using OMP is the requirement for

storing the Gram matrix in memory, in order to reduce the computational cost. Doing so is infeasible when operating on high dimensions, as both the dimension of each atom and their total amount increase. Therefore, it is reasonable to consider other sparse coding algorithms.

Task: Approximate the solution of (P_0): $\min_{\mathbf{x}} \|\mathbf{x}\|_0$ subject to $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Parameters: We are given the matrix $\mathbf{A}$, the vector $\mathbf{b}$, and the error threshold ϵ_0.

Initialization: Initialize $k = 0$, and set

- The initial solution $\mathbf{x}^0 = 0$.
- The initial residual $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0 = \mathbf{b}$.
- The initial solution support $\mathcal{S}^0 = Support\{\mathbf{x}^0\} = \emptyset$.

Main Iteration: Increment k by 1 and perform the following steps:

- **Sweep:** Compute the errors $\epsilon(j) = \min_{z_j} \|\mathbf{a}_j z_j - \mathbf{r}^{k-1}\|_2^2$ for all j using the optimal choice $z_j^* = \mathbf{a}_j^T \mathbf{r}^{k-1} / \|\mathbf{a}_j\|_2^2$.
- **Update Support:** Find a minimizer j_0 of $\epsilon(j)$: $\forall\, j \notin \mathcal{S}^{k-1},\ \epsilon(j_0) \leq \epsilon(j)$, and update $\mathcal{S}^k = \mathcal{S}^{k-1} \cup \{j_0\}$.
- **Update Provisional Solution:** Compute $\mathbf{x}^k$, the minimizer of $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ subject to $Support\{\mathbf{x}\} = \mathcal{S}^k$.
- **Update Residual:** Compute $\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$.
- **Stopping Rule:** If $\|\mathbf{r}^k\|_2 < \epsilon_0$, stop. Otherwise, apply another iteration.

Output: The proposed solution is $\mathbf{x}^k$ obtained after k iterations.

Figure 2: OMP

In our work we chose to incorporate the Subspace Pursuit (SP) algorithm, which has provable reconstruction capability comparable to that of LP methods, and exhibits the low reconstruction complexity of matching pursuit techniques for very sparse signals (specifically with $k \leq \text{const} \cdot \sqrt{m}$). This algorithm employs a search strategy in which a constant number of vectors is pruned from the candidate list, in contrast to OMP who generates a list of candidates sequentially, without backtracing. We will now cover this algorithm with detail.

The major challenge associated with sparse signal reconstruction is to identify in which subspace, generated by not more than k columns of the dictionary $\mathbf{D}$, the measured signal $\mathbf{y}$ lies. Once the correct subspace is determined, the non-zero signal coefficients are calculated by applying the pseudoinversion process. The defining character of the SP algorithm is the method used for finding the k columns that span the correct subspace: SP tests subsets of k columns in a group, for the purpose of refining at each stage an initially chosen estimate for the subspace. More specifically, the algorithm maintains a list of k columns of $\mathbf{D}$, performs a simple test in the spanned space, and then refines the list. If $\mathbf{y}$ does not lie in the current estimate for the correct spanning space, one refines the estimate by retaining reliable candidates, discarding the unreliable ones while adding the same number of new candidates. The "reliability property" is captured in terms of the order statistics of the inner products of the received signal with the columns of $\mathbf{D}$, and the subspace projection coefficients.

Input: K, $\mathbf{\Phi}$, $\mathbf{y}$

Initialization:

1) $T^0 = \{K$ indices corresponding to the largest magnitude entries in the vector $\mathbf{\Phi}^*\mathbf{y}\}$.
2) $\mathbf{y}_r^0 = \text{resid}\left(\mathbf{y}, \mathbf{\Phi}_{\tilde{T}^0}\right)$.

Iteration: At the ℓ^{th} iteration, go through the following steps

1) $\tilde{T}^\ell = T^{\ell-1} \bigcup \{K$ indices corresponding to the largest magnitude entries in the vector $\mathbf{\Phi}^*\mathbf{y}_r^{\ell-1}\}$.
2) Set $\mathbf{x}_p = \mathbf{\Phi}_{\tilde{T}^\ell}^\dagger \mathbf{y}$.
3) $T^\ell = \{K$ indices corresponding to the largest elements of $\mathbf{x}_p\}$.
4) $\mathbf{y}_r^\ell = \text{resid}\left(\mathbf{y}, \mathbf{\Phi}_{T^\ell}\right)$.
5) If $\left\|\mathbf{y}_r^\ell\right\|_2 > \left\|\mathbf{y}_r^{\ell-1}\right\|_2$, let $T^\ell = T^{\ell-1}$ and quit the iteration.

Output:

1) The estimated signal $\hat{\mathbf{x}}$, satisfying $\hat{\mathbf{x}}_{\{1,\cdots,N\}-T^\ell} = \mathbf{0}$ and $\hat{\mathbf{x}}_{T^\ell} = \mathbf{\Phi}_{T^\ell}^\dagger \mathbf{y}$.

Figure 3: SP

In our case, we incoroporated a parallelized version of SP into the current implementation of OSDL. As a starting point, we downloaded an open source implementation of SP in Matlab and adapted it into the OSDL implementation, so as to employ the double sparsity model. As we ran and tested it, we chose to design and implement modifications in order to deal with its bottelnecks and reach improved results. Among these changes were:

- We downloaded an open source C implementation of the maxk Matlab function for finding the k largest elements in a vector. This implementation was integrated into the SP implementation, as this subtask is vastly used throughout the algorithm.

- We downloaded an open source implementation of the Conjugate Gradient Least Squares (CGLS) algorithm, which iteratively solves the least squares problem, and used it in order to prevent the direct computing of $\mathbf{A}^\dagger\mathbf{y}$ repeated throughout SP. This implementation also utilizes our efficient method of computing $\mathbf{Dy}$ due to the separability of $\mathbf{\Phi}$. Moreover, the use of an iterative method refines our control over the computational cost of SP, as we can limit the number of iterations.

- Taking into account the fact that the number of non-zero entries in SP's output does not depend on the number of iterations, we limited its iterations to a number we deemed empirically sufficient by conducting experiments.

- Exploiting OSDL nature of working with mini-batches and the independence of calculating $\mathbf{x}_i$ for each column $\mathbf{y}_i$ of $\mathbf{Y}$, we chose to run the different calls to SP in parallel. In order to cope with the absence of shared memory support in Matlab, the builtin parallel.pool.Constant function

was used to minimize memory redundancy caused by copying $\mathbf{A}$ and $\mathbf{\Phi}$ between workers.

- Because SP does not require the computation of the Gram matrix, significant memory and computations can be saved by avoiding the frequent updates of this matrix. Therefore, all these calculations are avoided in the case of employing SP as the pursuit method.

3.2 Global Hard-Thresholding

While reviewing OSDL, we mentioned that the hard-thresholding operator $\mathcal{P}_k$ keeps the k largest non-zeros (in absolute value) in a vector, while clearing the other entries. This step keeps $\mathbf{A}$ sparse, a property that have both time and memory complexity benefits. However, forcing each atom to have a strict ℓ_0 norm appears to be an unnecessary limiting constraint, giving motivation to consider less restrictive strategies that might reveal themselves as more fruitful. In order for $\mathbf{A}$ to be sparse, the global constraint on the number of non-zeros in the entire matrix must be kept. Therefore, we chose to violate the local constraint on each atom's norm, replacing $\mathcal{P}_k$ with a step that keeps the mk largest entries in the matrix. Formally, we rewrite the training problem with the following one:

$$\min_{\mathbf{A},\mathbf{X}} \frac{1}{2} \left\| \mathbf{Y} - \mathbf{\Phi A X} \right\|_F^2 \text{ s.t. } \begin{cases} \left\| \mathbf{x}_i \right\|_0 \leq p & \forall i \\ \left\| \mathbf{A} \right\|_0 = mk & \forall j \end{cases}$$

Experimenting with this naive approach resulted with a great deal of degenerated atoms, indicating that a local invariant must be maintained. With this insight in mind, we forced each atom to have a minimum norm of k', where $k' \leq k$, giving the algorithm the freedom to choose the rest $m\left(k - k'\right)$ non-zeros in the matrix.

3.3 Reimplementation of *cleardict*

The *cleardict* function, which is called every few iterations, is responsible for clearing the dictionary from atoms that are used by a low number of training examples, as well as pairs of highly similar atoms, a property quantified by their normalized inner product. The discarded atoms are replaced by a sparse vector $\mathbf{x}$, where $\mathbf{\Phi x} = \mathbf{y}$ and $\mathbf{y}$ is a training example. This is reasoned by the fact that our purpose, in some manner, is to eventually have $\mathbf{y} \in \mathrm{Im}\left(\mathbf{D}\right)$ for every image $\mathbf{y}$.

We dedicated a great deal of our work to reimplementing *cleardict*, with some changes stemming directly from our previous work, and some being fixes to what was grasped by us as unwanted behaviour. These include:

- In order to compute the similarity between atoms, *cleardict* uses the Gram matrix of $\mathbf{A}$. This matrix is maintained in order to accelerate the computation time of OMP, an unnecessary step when using SP at the sparse

coding stage. Moreover, keeping the Gram matrix is infeasible when operating on high dimensions, motivating us to enable the choice of OMP as the pursuit algorithm, even when computing this matrix is not an option. Therefore, we implemented a direct calculation of the similarity when the Gram is not available in memory.

- While working on the previous feature, we have noticed that *cleardict* directly computes the normalized Gram matrix with regard to outdated atoms. We fixed this issue by passing the most recent normalized Gram to *cleardict*. Moreover, we canceled the redundant updating of the Gram outside of *cleardict*, in the case it was already updated by it.

- Relying on the Gram matrix for estimating similarity causes unwanted behaviour in the case a highly used atom resembles a rarely used one: the first will be pruned because of the similarity, and the second will be replaced because of its unusefulness. Furthermore, the previous implementation replaced all similar atoms, instead of keeping a single representative for each pair. To cope with this issue, we separated the pruning process into two sequential parts, where the first searches and replaces rarely used atoms, and the second does the same for similar ones. Additionally, atoms that are replaced are not taken into account when checking for similarity, and a representative is kept for each pair.

- Each atom is replaced by $\mathbf{x}$ where $\mathbf{\Phi x} = \mathbf{y}$, and $\mathbf{y}$ is chosen from the current mini-batch $\mathbf{Y}_t$ such that the representation error when sparse coding $\mathbf{DX}_t = \mathbf{Y}_t$ with a small number of non-zeros will be minimized. Therefore, when *cleardict* replaces a great deal of atoms, it will eventually become indifferent to the choice of $\mathbf{y}$. This causes the previous implementation to keep selecting the same $\mathbf{y}$, flooding $\mathbf{A}$ with the same atom. We changed the implementation to react to this case by selecting $\mathbf{y}$ randomly.

- Lastly, in order to keep *cleardict*'s running time as short as possible, the pruning process was changed to perform only on a randomly chosen subset of the atoms.

3.4 Bug Fixes

Going through the implementation of OSDL, we have noticed and fixed the following bugs:

- The code provides two options for passing the training set, either by a matrix variable, or as external data obtained through a mat file, as the latter avoids storing all images in memory. When the first option is selected, the mean of each example is subtracted from its original entries. With the purpose of establishing unifrom behaviour, the code was changed so this will be performed on externally provided examples as well.

- In order to identify which atoms are used when representing a mini-batch, the sum of each coordinate over all representations is calculated and compared to zero. This approach will give wrong results in case the calculation is composed of positive and negative numbers whose sum is zero. Therefore, the calculation was changed to consider the absolute values of each coordinate.

- The time of each iteration is saved in order to analyze the algorithm performance. We refined this data by excluding the time of saving the dictionary, and fixing a bug that caused a portion of the time dedicated for calculating the test error to be taken into account.

- When keeping track of the train error, the algorithm would override previous error values, in case the error was ordered to be sampled more than once per iteration. This was fixed by changing the size of the array the train error was kept in, and updating the correspondig index when writing to it.

- The number of times each atom is used when representing a training example is kept for each iteration. This information is used by *cleardict* in order to identify rarely used atoms. A faulty arrangement of the code caused *cleardict* to receive an out-of-date version of this information. This was fixed by rearranging the code.

- Similarly, the arrangement of the code caused the training error and elapsed time data for each iteration to be saved only at the next iteration. This was changed by updating the data before saving it.

4 Experiments

In this section we present a number of experiments to illustrate the effect of our work on the behaviour of OSDL. All the experiments were run on a 64-bit operating system with an Intel Core i7 microprocessor, with 16 GB of RAM.

4.1 The Dataset

Our experiments were conducted on a collection of roughly a million face images. It was assembled from the following publicly available datasets:

- MegaFace, a face recognition dataset with 4.7M faces and 672K identities obtained from Flickr.

- WIKI-crop and IMDB-crop, datasets which are composed of celebrity images taken from the corresponding websites. Together they contain about 500K images.

These datasets offer images shot from a wide variety of angles and positions, with the corresponding faces belonging to a diversity of ages and ethnicities. This

stands in contrast to the closed dataset the algorithm was previously tested with. Since most provided images contain a large background, and some even more than a single face, we implemented a Python script to identify faces in the images and crop them accordingly. The results were then gray-scaled and resized to 128×128 pixels. This process eventually produced 960,728 images. 20 randomly sampled images from the dataset are presented below, in both their original and cropped form.

Figure 4: Random Images from the Dataset and Their Corresponding Cropped Versions

4.2 Side Features

To test the effect of the changes to the implementation, we have conducted experiments using OMP as the chosen sparse coding algorithm. We ran the old version of OSDL (indicated by the letter (A) in the graph), and compared it to the new one with varying versions:

- (B) Using atom-wise hard-thresholding of k non-zeros with *cleardict* inspecting all atoms.
- (C) Using atom-wise hard-thresholding of k non-zeros with *cleardict* inspecting only a subset of randomly chosen atoms.
- (D) Using global hard-thresholding with a minimum constraint of $0.01k$ non-zeros in each atom and with *cleardict* inspecting all atoms.
- (E) Using global hard-thresholding with a minimum constraint of $0.01k$ non-zeros in each atom and with *cleardict* inspecting only a subset of randomly chosen atoms.
- (F) Using global hard-thresholding with a minimum constraint of $0.08k$ non-zeros in each atom and with *cleardict* inspecting all atoms.
- (G) Using global hard-thresholding with a minimum constraint of $0.08k$ non-zeros in each atom and with *cleardict* inspecting only a subset of randomly chosen atoms.

In this experiment, the training was performed using 5,150 examples randomly picked from the dataset, and was tested using 270 other images randomly picked from the same source. This was repeated twice: once for $k = 98$ and once for $k = 327$. Other essential parameters are presented below:

name	A	B	C	D	E	F	G
Number of atoms in dictionary	1500	1500	1500	1500	1500	1500	1500
Non zeros in each dictionary atom	98	98	98	98	98	98	98
Number of data sweeps	5	5	5	5	5	5	5
Batch size	128	128	128	128	128	128	128
Non zeros in train sample	163	163	163	163	163	163	163
Pursuit method	omp	omp	omp	omp	omp	omp	omp
Atomwise threshlding/ Global thresholding	~	Atomwise	Atomwise	Global	Global	Global	Global
Number of clear dictionary operations per one data sweep	1	1	1	1	1	1	1
pursuit method for testing	~	omp	omp	omp	omp	omp	omp
Size of subset tested in cleardict	~	1500 (all atoms)	300 (random subset)	1500 (all atoms)	300 (random subset)	1500 (all atoms)	300 (random subset)
Minimum number of non zeros in atom	~	~	~	10	10	79	79

name	A	B	C	D	E	F	G
Number of atoms in dictionary	1500	1500	1500	1500	1500	1500	1500
Non zeros in each dictionary atom	327	327	327	327	327	327	327
Number of data sweeps	5	5	5	5	5	5	5
Batch size	128	128	128	128	128	128	128
Non zeros in train sample	163	163	163	163	163	163	163
Pursuit method	omp	omp	omp	omp	omp	omp	omp
Atomwise threshlding/ Global thresholding	~	Atomwise	Atomwise	Global	Global	Global	Global
Number of clear dictionary operations per one data sweep	1	1	1	1	1	1	1
pursuit method for testing	~	omp	omp	omp	omp	omp	omp
Size of subset tested in cleardict	~	1500 (all atoms)	300 (random subset)	1500 (all atoms)	300 (random subset)	1500 (all atoms)	300 (random subset)
Minimum number of non zeros in atom	~	~	~	33	33	262	262

Figure 5: Experiment #1 Parameters

The following graphs show the mean train and test errors as a function of time, with each star in the curve marking a full iteration of OSDL. As we have expected, the new implementation gains substantially better results than the original one. However, quite surprisingly, the global hard-thersholding approach falls behind the original one, albeit the freedom it provides to the algorithm in picking the non-zeros. This is further affirmed by the better results obtained when giving the global hard-thersholding a more restrictive constraint. A possible reason is the considerable amount of time it takes to choose the non-zeros across the whole dictionary, which is a slower procedure then setting a unifrom threshold for all atoms. Another, less surprising observation, is the fact that the version of *cleardict* that inspects all of the atoms performs better than the one that inspects only a subset of the atoms. This was expected, since OMP already stores the Gram matrix in memory, canceling the need for directly calculating the similarity between the atoms. Checking the entire set of atoms is much more costly when using SP.

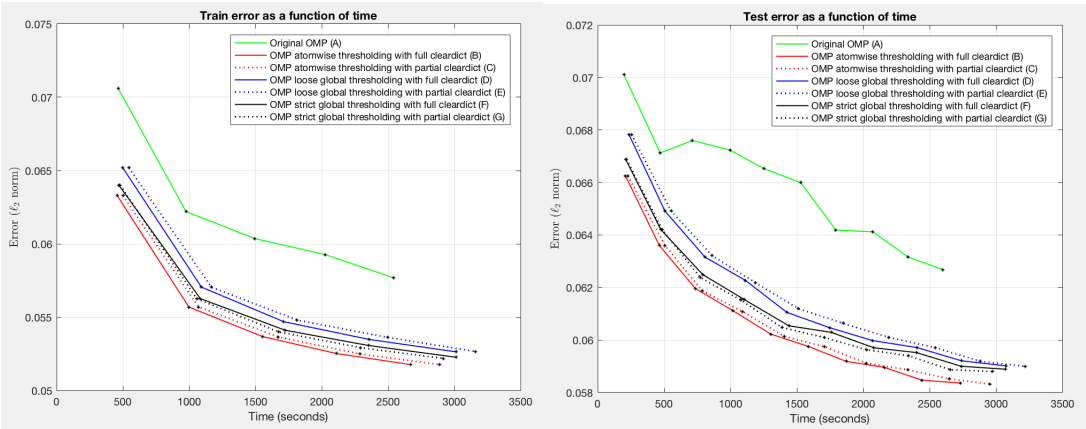

Figure 6: Experiment #1 Train/Test Error with $k = 98$

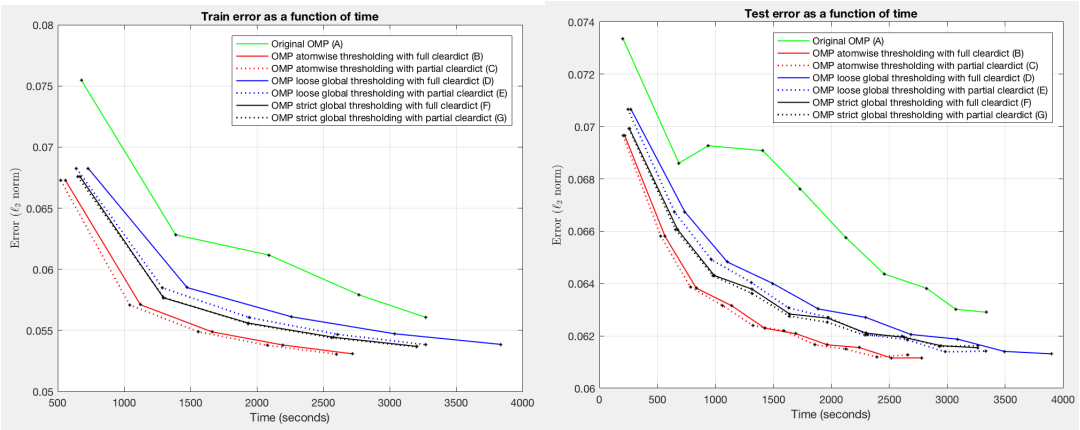

Figure 7: Experiment #1 Train/Test Error with $k = 327$

Next, in order to assess the atoms obtained through the global hard-thresholding approach, we present the following histograms, demonstrating the distribution of non-zeros across the atoms. As we can see, most atoms have the lowest possible amount of non-zero entries, with only a minor portion of them varying in sparsity. This might explain the poor results of the global approach.

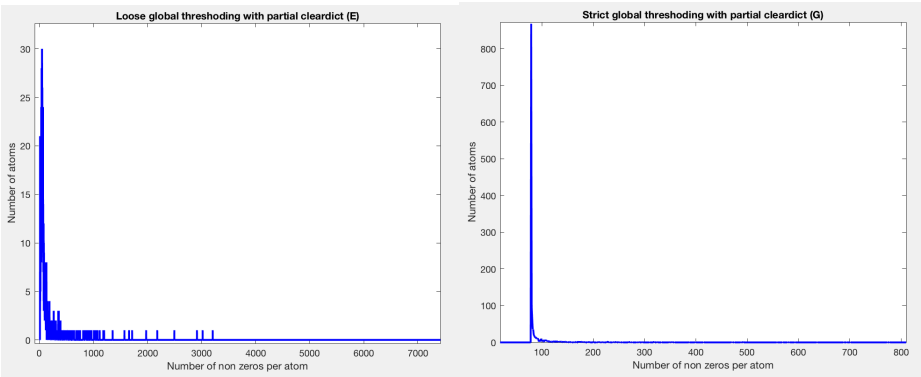

Figure 8: Experiment #1 NNZ Distribution Histograms with $k = 98$

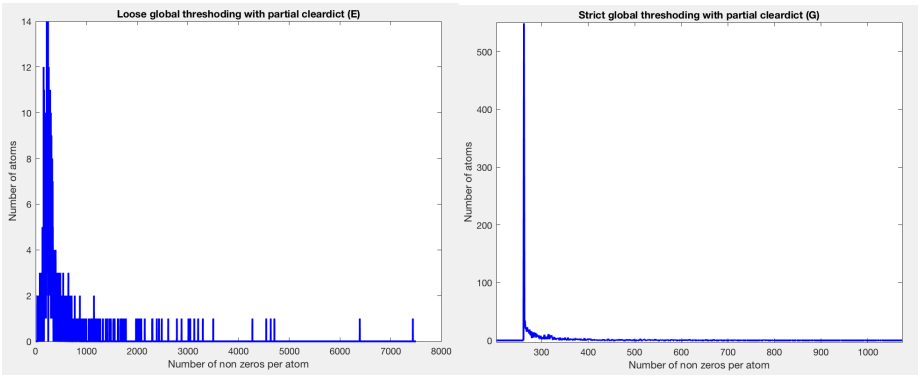

Figure 9: Experiment #1 NNZ Distribution Histograms with $k = 327$

Finally, we would like to test our premissive approach of inspecting only a random subset of the atoms in each call to *cleardict*. In the following tables, we list some statistics about the similarity between atoms, represented by their normalized inner products. We can see that the original implementation is lacking in comparison to the new version. Unsurprisingly, the versions that use randomization fall short behind the strict versions, although the difference between the statistics is not significant.

k=98	A	B	C	D	E	F	G
highest inner product	0.9741	0.9632	0.9557	0.9786	0.9816	0.9592	0.9715
mean of highest 50 values	0.9443	0.8961	0.9006	0.912	0.918	0.8953	0.9002
mean of inner products	0.3893	0.2807	0.2812	0.298	0.2978	0.3085	0.3078

k=327	A	B	C	D	E	F	G
highest inner product	0.9696	0.9693	0.9706	0.9777	0.9897	0.9073	0.9038
mean of highest 50 values	0.9274	0.8658	0.8666	0.8866	0.9037	0.84	0.8471
mean of inner products	0.3703	0.2961	0.2969	0.2948	0.2994	0.2998	0.3002

Figure 10: Experiment #1 Inner Product Statistics

With the purpose of further investigating the differences between the hard-thresholding methods when operating on a larger dataset, we ran the (C) and (G) versions again on 20,556 examples. This process resulted with 4000 atoms containing 327 non-zeros that were tested using 270 other images. The other parameters of the algorithm were not changed. The following graphs present the mean train and test errors as a function of time, and show similar behaviour to the one observed during the previous experiment, where the atom-wise hard-thresholding method outperformed the global one.

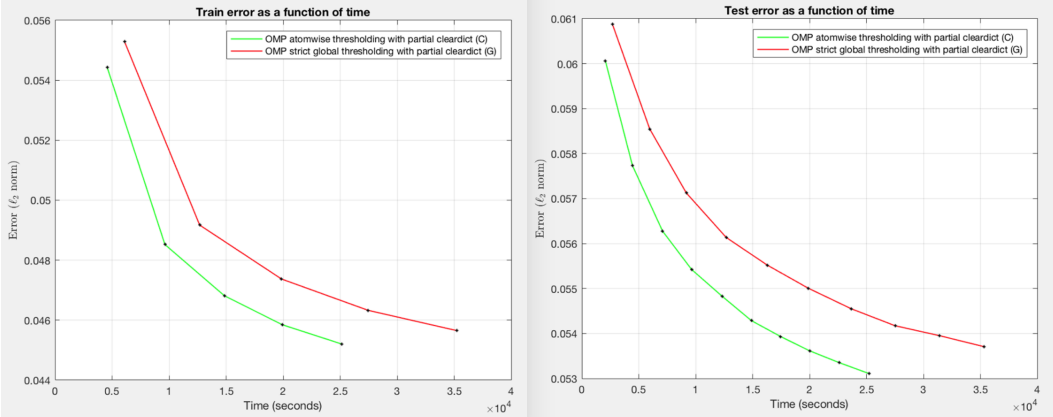

Figure 11: Experiment #1 Train/Test Error with a Larger Train Set

4.3 OMP vs. SP

To test the effect of using SP instead of OMP at the sparse coding step, we ran our new implementation of OSDL both with OMP and with our improved parallel version of SP. Furthermore, in order to investigate the performance of OMP under memory restrictions, we ran OSDL with a version of OMP that doesn't rely on keeping the Gram matrix in memory, thus making the comparison with SP fair in terms of memory consumption. The training was performed using 35,000 examples randomly picked from the dataset, and was tested using 1930

other images randomly picked from the same source as well. The train error was computed using the native sparse coding algorithm, while the test error was computed unifromly using SP. Other essential parameters are presented below:

name	SP	OMP
Number of atoms in dictionary	7000	7000
Non zeros in each dictionary atom	491	491
Number of data sweeps	5	2
Batch size	128	128
Non zeros in train sample	655	655
Pursuit method	sp	omp
Number of testing operations per one iteration	1	1
Number of clear dictionary operations per one data sweep	1	2
Size of subset tested in cleardict	1400	1400
CGLS iterations for SP method	30	~
Maximum number of iterations per each SP sparse coding	100	~

Figure 12: Experiment #2 Parameters

The following graphs indicate the mean train and test errors as a function of time, with each star marking a full iteration of OSDL. As we can see, using SP greatly reduces the computation time of each iteration of the algorithm: OSDL completed 8 iterations in less than 20 hours using SP, and only 2 iterations in more than 23 hours using OMP. Although the slope of the OMP curve is steeper than the SP one, indicating that each iteration using OMP yields better atoms, SP's considerably faster pace gives it the upper hand, presenting better errors in each point of time. Moreover, the fact that the version of OMP employed here is the one using the Gram matrix, emphasizes SP's superiority, as OMP gets an advantage in some sense.

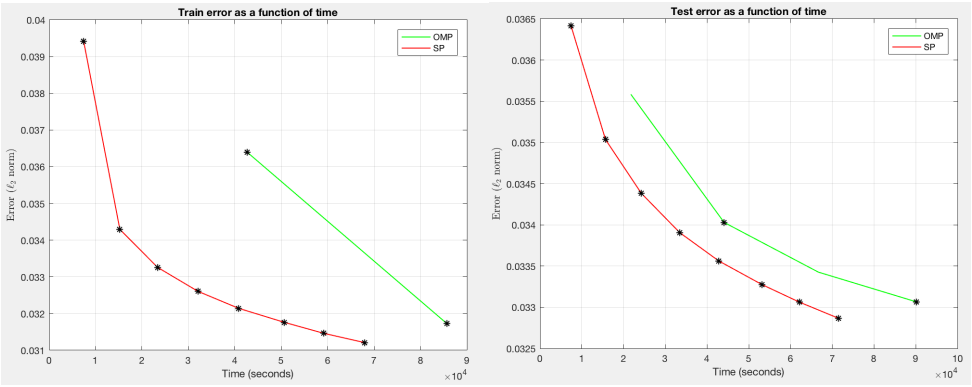

Figure 13: Experiment #2 Train/Test Errors

The results of using OSDL with the version of OMP that doesn't utilize the Gram matrix are not presented, since **after 96 hours of run time, it only managed to complete 25% of the first iteration**. This illustrates SP's fundamental advantage of low memory consumption, making it extremely

practical when working in higher dimensions.

The following graph indicates the error of sparse coding using the final dictionary with SP as a function of the ratio of the non-zero coefficients. As we can see, when sparse coding for only 2-6% non-zeros, the dictionary obtained through SP outperforms the one obtained through OMP, displaying SP's advantage when aiming for high compression ratio. This tendency changes when more non-zeros are allowed, where the dictionary obtained through OMP gives better results.

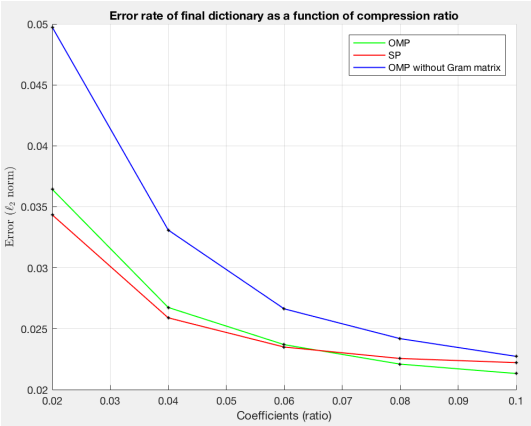

Figure 14: Experiment #2 Compression Error Using SP

In order to have a fair comparison, we also display a similar graph using OMP as the sparse coding algorithm. It is expected that a dictionary that was trained with a certain method would perform better when used for sparse coding with the same method. This seems to be confirmed by the graph, as the results of OMP are better than the results of SP. However, the longer runtime of OSDL with OMP should be taken into account when assessing the graph.

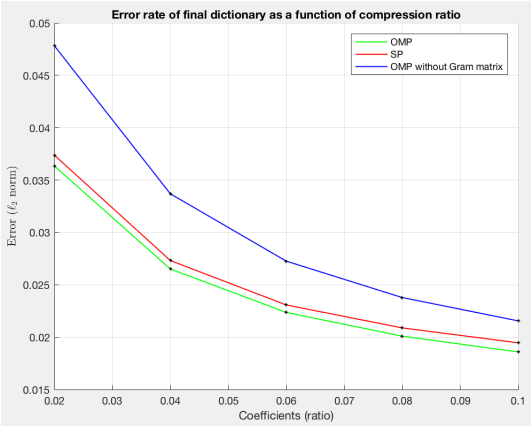

Figure 15: Experiment #2 Compression Error Using OMP

Next, we present the 225 atoms with the highest entropy in each dictionary. It seems that the dictionary obtained through SP contains a great deal of sparse representation of training examples, and therefore we hypothesize that *cleardict* has replaced many of its atoms due to low usability.

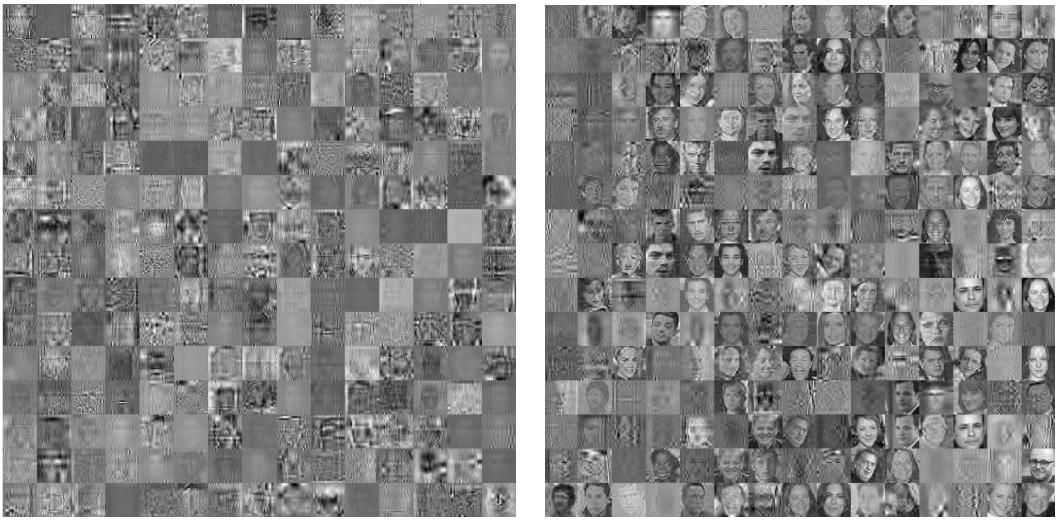

Figure 16: Experiment #2 Atoms (OMP to the left and SP to the right)

This can be verified by looking at the following histograms, presenting the number of times each atom is used when sparse coding the given test set. It is immediately noticeable that the histogram in the SP case is generally more

spiky, showing that the dictionary contains a deal of unused atoms, as well as highly useable ones. This stands in contrast to the even distribution of the atoms in the OMP case, which has a well balanced histogram.

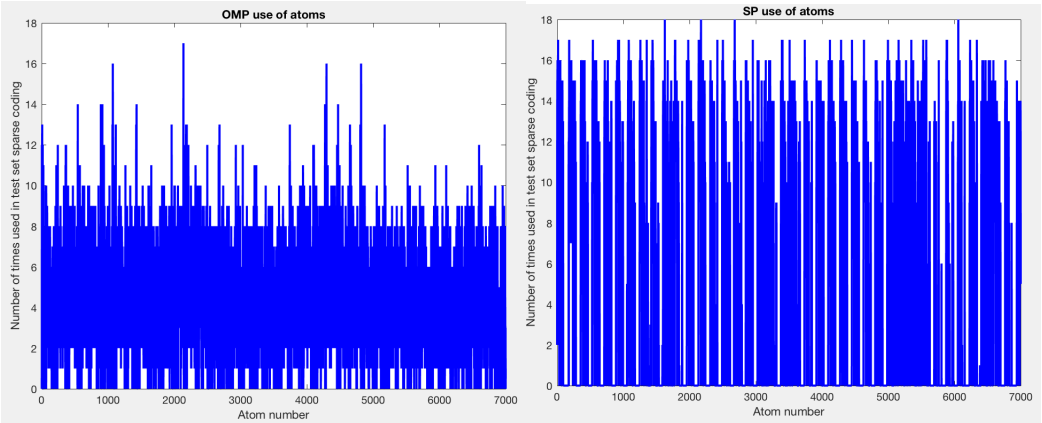

Figure 17: Experiment #2 Use of Atoms Histograms

In order to assess a larger set of atoms that might not be affected by *cleardict*, we also present the next 225 atoms with next highest entropy in the dictionary obained through SP.

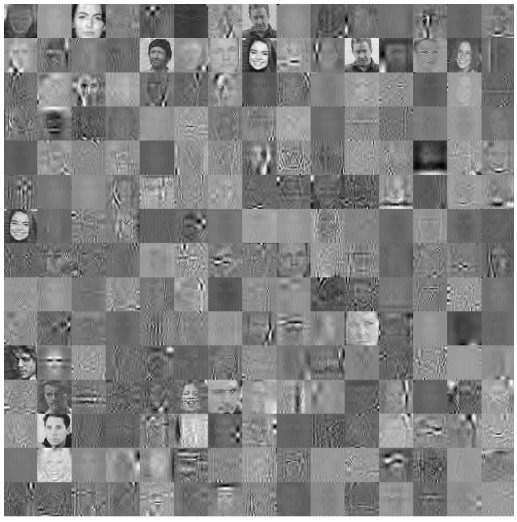

Figure 18: Experiment #2 SP Next Atoms

4.4 Final Experiment

To conclude our work, we ran the new implementation of OSDL on the entire dataset, which contains about million images, once with OMP as its sparse coding algorithm, and once with SP. The OMP version ran for 48 hours with only completing 40% of its first iteration, while the SP one managed to complete two iterations during 36 hours. Since all of the dataset was used during training, we tested the dictionary using images taken from the Chicago Face Database (CFD). The following graph indicates the error of sparse coding using the final dictionary with SP as a function of the ratio of the non-zero coefficients. Interestingly, the SP version provided superiour results across all ratios.

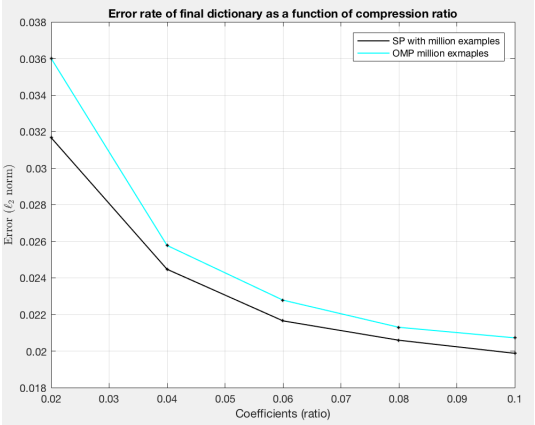

Figure 19: Experiment #3 Compression Error Tested with SP

These results are compared to the ones from the pevious experiments, with the dictionaries obtained by learning from million examples performing better:

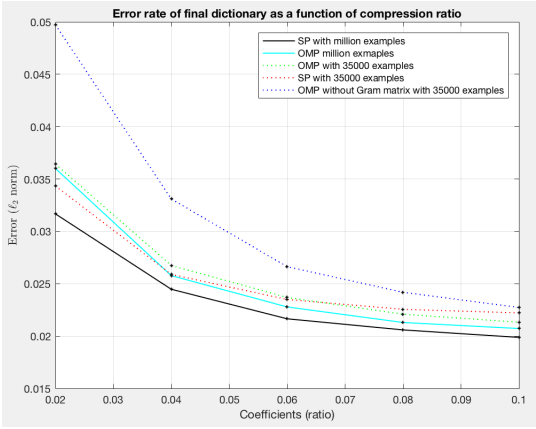

Figure 20: Overall Compression Error

As we did with the previous experiment, we present a similar graph tested with OMP as the sparse coding algorithm. This time, however, the SP version gives better results for some coefficient ratios, even though the test method gives advantage to the OMP version, which was trained 12 hours longer.

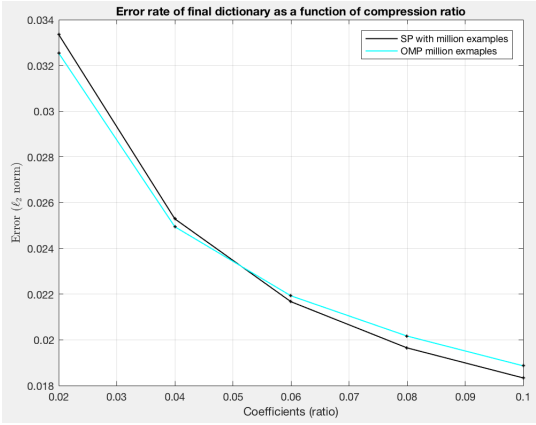

Figure 21: Experiment #3 Compression Error Tested with OMP

We show the 225 atoms with the highest entropy in each dictionary. In contrast to the previous experiment, the dictionary obtained using SP contains less atoms that look like original training examples, making the impression that fewer atoms were replaced by *cleardict*. Moreover, when comparing the atoms between the two dictionaries, it is easy to see that almost each atom obtained through SP emphasizes a different area of a face: with some clearly stressing

eyes, noses, chins and etc.

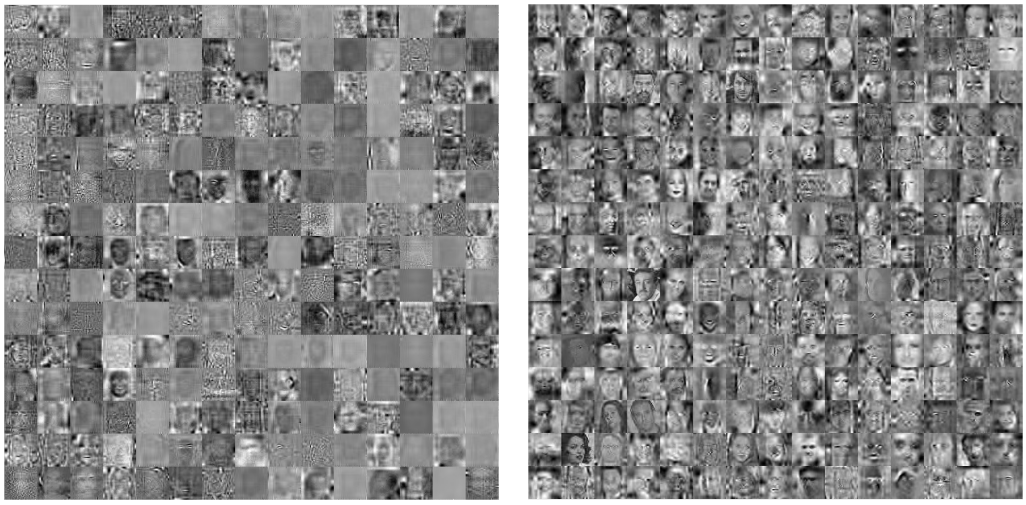

Figure 22: Experiment #3 Atoms (OMP to the left and SP to the right)

Lastly, we present 9 images reconstructed with different ratios of non-zeros using the two dictionaries. As we can see, the SP dictionary constructed smoother, more natural looking images, avoiding the often grainy appearance of the ones constructed by the OMP one. The difference is even more perceptible as the number of coefficients diminishes.

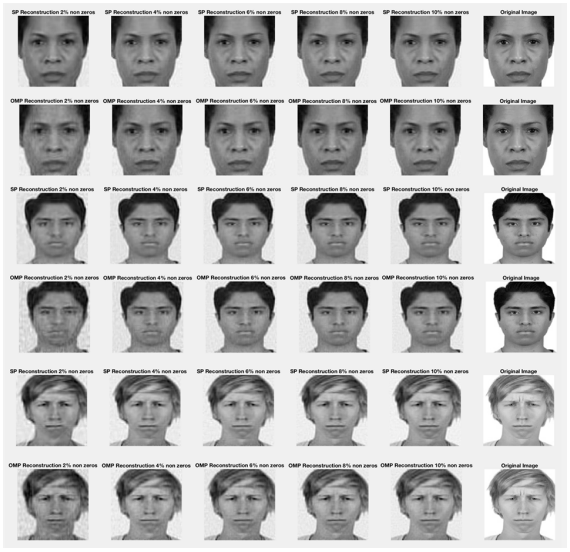

Figure 23: Constructed Images Using the SP and OMP Dictionaries

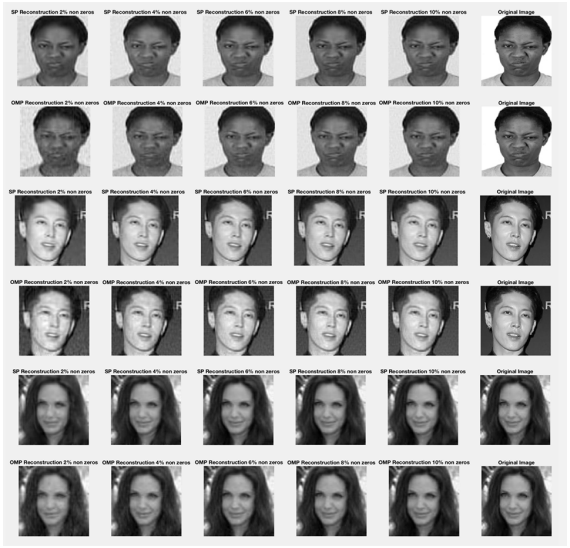

Figure 24: Constructed Images Using the SP and OMP Dictionaries

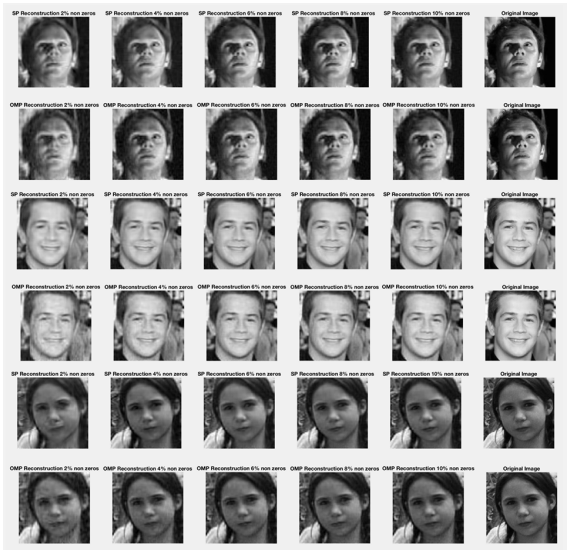

Figure 25: Constructed Images Using the SP and OMP Dictionaries

5 Summary and Future Work

This work concentrated on improving an existing dictionary learning algorithm implementation for high dimensions, and testing it with a massive dataset. Dealing with its high memory consumption constraints rooted at using OMP at its sparse coding stage, we integrated it with SP, which proved itself to be an efficient alternative, acheiving similar results in much less time. Furthemore, by reimplementing the atom clearing process and fixing code flaws, we have been able to outperform the previous implementation cosiderably, even when using OMP. Another attempt was aimed at improving the hard-theresholding step by giving the algorithm more freedom at choosing non-zero entries. Unfortunately, this approach was less successful as shown by our experiments.

While SP proved sufficient over OMP, a more fairfull comparison might be conducted by trying a parallel version of the latter. In addition, it might be beneficial to further experiment with the concept of global thersholding, testing it with different scenarios and considering modifications such that more meaningful atoms will be produced. Although the atom clearing process is an essential part of the algorithm, a large amount of atoms are being replaced by it, suggesting that less strict conditions for pruning should be considered. Furthemore, understanding quantitatively how different parameters affect the learned dictionaries will provide a better understanding of our model. These questions, among others, are part of ongoing work.

References

[1] Alfred M Bruckstein, David L Donoho, and Michael Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM review*, 51(1):34–81, 2009.

[2] Wei Dai and Olgica Milenkovic. Subspace pursuit for compressive sensing signal reconstruction. *IEEE Transactions on Information Theory*, 55(5):2230–2249, 2009.

[3] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4873–4882, 2016.

[4] Debbie S Ma, Joshua Correll, and Bernd Wittenbrink. The chicago face database: A free stimulus set of faces and norming data. *Behavior research methods*, 47(4):1122–1135, 2015.

[5] Stéphane Mallat and Gabriel Peyré. A review of bandlet methods for geometrical image representation. *Numerical Algorithms*, 44(3):205–234, 2007.

[6] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 10–15, 2015.

[7] Jeremias Sulam and Michael Elad. Large inpainting of face images with trainlets. *IEEE Signal Processing Letters*, 23(12):1839–1843, 2016.

[8] Jeremias Sulam, Boaz Ophir, Michael Zibulevsky, and Michael Elad. Trainlets: Dictionary learning in high dimensions. *IEEE Transactions on Signal Processing*, 64(12):3180–3193, 2016.