

Dictionary learning for applications in image processing

by

Pradyot Prakash

Roll No: 130050008

under the guidance

of

Prof. Suyash Awate

Bachelors' of Technology Thesis (I)



Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 22nd November, 2016

Pradyot Prakash

Place: IIT Bombay, Mumbai

Roll No: 130050008

Acknowledgements

I am thankful to the people who have been instrumental in helping me out throughout this project. First and foremost, I express my sincere gratitude towards my supervisor Prof. Suyash Awate for his guidance. I would also thank Samarth and Ravi for helping me in the initial phases of this project. I am grateful to Sougata Singha for the leaf dataset collected by him. It was helpful for the shape analysis experiments. I am also thankful to my friends, family and teachers who have been always there for me whenever I needed them.

Contents

1	Introduction	2
1.1	Why dictionaries?	3
1.2	Dictionaries	3
2	Approaches to dictionary learning	6
2.1	Non-Negative Sparse Coding (NNSC)[3]	6
2.1.1	Non-increasing updates to W	7
2.1.2	NNSC algorithm	7
2.2	Non-linear Sparse Coding (NLSC)	8
2.2.1	Manifolds	8
2.2.2	Riemannian manifolds	8
2.2.2.1	The notion of distance	9
2.2.2.2	Exponential and the logarithm maps	9
2.2.3	Dictionaries on spheres	10
2.2.4	The algorithm	12
2.2.5	Gradients	12
2.2.6	Some implementation details	14
3	Application of dictionaries for MNIST classification	15
3.1	Experiment	15
3.2	Results	17

4	Application of dictionaries on image patches for image denoising	18
4.1	Reconstruction algorithm	19
4.2	Experiments and results	20
5	Dictionary for shapes	24
5.1	Algorithm for shapes	26
5.2	Experiments and results	27
5.2.1	Ellipses	27
5.2.2	Leaf images	28
5.2.2.1	Anhui barberry	30
5.2.2.2	Trident maple	30
5.2.2.3	Joint learning of shapes	30
6	Conclusion	36
7	Future Work	37

List of Tables

3.1	Accuracy for each digit	17
5.1	Weights for the 10 atoms for leaf in 5.10	32

List of Figures

4.1	Comparison of reconstruction before and after convergence	20
4.2	50 atoms for the Lena image	21
5.1	Two atoms used to construct the ellipses	28
5.2	Reconstructed ellipses using 2 atoms	29
5.3	Reconstruction of ellipse using its atoms	29
5.4	Atoms used to construct the leaves	30
5.5	Reconstructed leaves using 5 atoms	31
5.6	Atoms used to construct the leaves	32
5.7	Reconstructed leaves using 10 atoms	33
5.8	Atoms used to construct the leaves	34
5.9	Reconstructed leaves using 10 atoms	35
5.10	A leaf from Anhui barberry reconstructed from the mixed dictionary	35

Chapter 1

Introduction

Researchers working in the field of image processing have tried to find succinct ways of representing images. The usual notation containing intensity values of each pixel is not interesting and does not exploit the structure within the image. The application of Fourier transform through frequency based analysis has been present for a long time. The Fourier transform and its neighbor, discrete cosine transform are universal bases used for image representation. However, they miss out on identifying the locality within the images. For instance, the Fourier transform of a box-function gives us a sinc function but we can't pinpoint its exact coordinates. To alleviate this inefficiency, other kinds of bases such as contourlets and wavelets have been used. We see the application of dictionaries as another such method.

This project explores the ways in which dictionaries have been used in image processing for tasks such as classification, denoising and shape analysis. A lot of work has been done in these areas and we explore the possibility of extending the same idea using hyperspheres and borrowing concepts from manifolds.

1.1 Why dictionaries?

The transforms mentioned above are global bases of a Hilbert space. Because of such nature, they are not adaptive to data and cannot adapt. Take the Fourier transform for instance. We cannot represent edges using that. The wavelet transform is spatially local and can represent edges using a small number of basis functions. But all these methods are general and are analytically defined. We want something which is data dependent and can be learned accordingly. Hence, we look towards dictionaries to pave the way.

1.2 Dictionaries

Dictionaries[5] can be understood as a generalization of the idea of bases in linear algebra. The notion of dictionaries comes while trying to come up with a solution for the following problem.

Suppose that we have a set of training data points, labeled as $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ in some d dimensional space. Define,

$$\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2 \dots \mathbf{X}_n]$$

to be a $d \times n$ matrix. Our goal is to find a set called **dictionary**,

$$\mathbf{A} = [\mathbf{A}_1 \mathbf{A}_2 \dots \mathbf{A}_m]$$

containing m vectors in the d dimensional space such that \mathbf{X} can be written as a linear combination of $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_m$, i.e.,

$$\mathbf{X}_i = w_{1i}\mathbf{A}_1 + w_{2i}\mathbf{A}_2 + \dots + w_{mi}\mathbf{A}_m$$

which is equivalent to

$$\mathbf{X}_i = \mathbf{A} \mathbf{W}_i$$

where

$$\mathbf{W}_i = [w_{1i} w_{2i} \dots w_{mi}]^T$$

For the sake of brevity, the entire composition can be written as,

$$\mathbf{X} = \mathbf{A}\mathbf{W}$$

where

$$\mathbf{W} = [\mathbf{W}_1 \mathbf{W}_2 \dots \mathbf{W}_n]$$

Note that the components of the set A are called **atoms**. Also, $\mathbf{X} \in \mathbb{R}^{d \times n}$, $\mathbf{A} \in \mathbb{R}^{d \times m}$ and $\mathbf{W} \in \mathbb{R}^{m \times n}$. Our task at hand is to estimate both \mathbf{A} and \mathbf{W} given \mathbf{X} . This can be approximated by trying to minimize the error, $\|\mathbf{X} - \mathbf{A}\mathbf{W}\|_F^2$ where $\|\cdot\|_F$ is the Frobenius norm.

This is done by modeling this as the following optimization problem,

$$\arg \min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \|\mathbf{X}_i - \mathbf{A}\mathbf{W}_i\|_F^2$$

One may wonder as to how does a dictionary differ from the basis set. One can simply choose \mathbf{A} to be the d dimensional basis set and the coefficients can be accordingly computed. Dictionaries differ in the sense that m may or may not be equal to d . The cardinality of the basis set is what we call the dimension of the data, in our case, d . This basis set can be used to represent all the vectors in the d dimensional space. However, for many problems, we do not need to cover the entire wide space, but only a small subset of it. For such cases, we consider a reduced basis of sorts, commonly called the dictionary. For some cases, a small number of atoms, $m < d$ can suffice and for other scenarios, we may need more than d atoms.

Furthermore, the notion of orthogonality is often missing in dictionaries. Even in the case of $m = d$, orthogonality may be missing. The number m is a hyper-parameter of the model and needs to be guessed. The dictionary \mathbf{A} is called **undercomplete** if $m < d$ or **overcomplete** in case $m > d$. The first case is similar to representing the data using a smaller number of dimensions. The PCA algorithm tries to do something similar. The second case motivates the sparse dictionary learning problem. The usual notion of dictionaries is extended by to include sparsity in the \mathbf{W} matrix. We wish to be able to represent

\mathbf{X}_i using only a subset of \mathbf{A} . To account for this, the optimization problem is tweaked slightly to look like,

$$\arg \min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \|\mathbf{X}_i - \mathbf{A} \mathbf{W}_i\|_F^2 + \lambda f(\mathbf{W})$$

Here, $f(\cdot)$ is some function defining the sparsity for \mathbf{W} . The l_0 , l_1 norms are popular choices for $f(\cdot)$.

Chapter 2

Approaches to dictionary learning

2.1 Non-Negative Sparse Coding (NNSC)[3]

The problem of non-negative matrix factorization can be represented succinctly as the minimization of,

$$C(\mathbf{A}, \mathbf{W}) = \frac{1}{2} \|\mathbf{X} - \mathbf{AW}\|_F^2$$

where $C(., .)$ is the cost function. Here, the non-negativity constraints are on \mathbf{A} and \mathbf{W} , i.e. $\forall ij : A_{ij} \geq 0, W_{ij} \geq 0$.

The problem of non-negative sparse coding adds a specific sparsity function to this function along with some other constraints. Sparsity is added through the addition of

$$f(\mathbf{W}) = \sum_{i=1}^m \sum_{j=1}^n W_{ij}$$

giving us the resultant optimization function

$$C(\mathbf{A}, \mathbf{W}) = \frac{1}{2} \|\mathbf{X} - \mathbf{AW}\|_F^2 + \lambda \sum_{i,j} W_{ij}$$

subject to the constraints $\forall ij : A_{ij} \geq 0, W_{ij} \geq 0$ and $\forall i : \|\mathbf{A}_i\| = 1$. This is identical to the l_1 norm for each W_{ij} as $W_{ij} \geq 0$. The hyper-parameter λ is assumed to be non-negative.

The problem boils down to minimizing this error function. Due to the lack of a closed

form solution to the solution, an iterative approach is taken. Usually, an algorithm identical to gradient descent is used to iteratively reduce the value of the function to a minima.

However, the authors of the paper show that under the above choice of $C(., .)$, the iteration can be done in a specific way which has certain good properties.

2.1.1 Non-increasing updates to \mathbf{W}

The objective function is non-increasing under the following update rule:

$$\mathbf{W}^{t+1} = \mathbf{W}^t .* (\mathbf{A}^T \mathbf{X}) ./ (\mathbf{A}^T \mathbf{A} \mathbf{W}^t + \lambda)$$

where $.*$ and $./$ notations are MATLAB compatible. The addition of λ is to all the elements of the matrix. Note that T represents the transpose of a matrix and t represents time.

2.1.2 NNSC algorithm

Algorithm 1 NNSC

- 1: $t \leftarrow 0$
 - 2: Initialize \mathbf{A}^0 to strictly positive values
 - 3: $\mathbf{A}^0 \leftarrow \text{normc}(\mathbf{A}^0)$
 - 4: Initialize \mathbf{W}^0 to strictly positive values
 - 5: **repeat**
 - 6: $\mathbf{A1} \leftarrow \mathbf{A}^t - \mu(\mathbf{A}^t \mathbf{W}^t - X)(\mathbf{W}^t)^T$
 - 7: $\mathbf{A2} \leftarrow \max(0, \mathbf{A1})$
 - 8: $\mathbf{A}^{t+1} \leftarrow \text{normc}(\mathbf{A2})$
 - 9: $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t .* ((\mathbf{A}^{t+1})^T \mathbf{X}) ./ ((\mathbf{A}^{t+1})^T (\mathbf{A}^{t+1}) \mathbf{W}^t + \lambda)$
 - 10: $t \leftarrow t + 1$
 - 11: **until** convergence
-

2.2 Non-linear Sparse Coding (NLSC)

In this section, we look at another approach to learn dictionaries. The previous approach described dictionaries in the Euclidean space. However, an equivalent coordinate system, the spherical coordinate system, has also been explored for such tasks. This section discusses one such approach[6] and delineates the operations done using them.

Representing points on a sphere requires special treatment. This is motivated by the fact that the notion of distance on spheres no longer follows the one in the Euclidean space. Distances on a sphere are calculated along the great circle joining the two points in question and log maps come into the picture here. The corresponding operations are also modified to account for this different metric. Manifolds are topological spaces which help us deal with operations on spheres. From the following sections on manifolds, we will see that the distance metrics are defined differently for each point. This task makes working with manifolds more complex and needs to be handled carefully.

2.2.1 Manifolds

A manifold \mathcal{M} of dimension d is a topological space such that each point $x \in \mathcal{M}$ has a neighborhood which can be continuously transformed to a Euclidean space of the same dimension. More formally, the neighborhood is homeomorphic to the Euclidean space, \mathbb{R}^d . With the additional property of being able to perform differential calculus on the manifold, it becomes a differential manifold.

2.2.2 Riemannian manifolds

Every differential manifold has a tangent space associated with it. The tangent space, $T_x\mathcal{M}$ defined at $x \in \mathcal{M}$ is the vector space containing all the tangent vectors to \mathcal{M} at the point x . With the additional constraint of having an inner product, on the tangent space, $T_x\mathcal{M}$, we get the Riemannian manifold.

2.2.2.1 The notion of distance

Let $v \in T_x\mathcal{M}$ be a tangent vector to \mathcal{M} at x . There exists a unique smooth curve, a geodesic,

$$\gamma_v : [0, 1] \rightarrow \mathcal{M}$$

satisfying $\gamma_v(0) = x$ with initial tangent vector in the direction of v , $\gamma'_v(0) = v$.

Suppose x and y are two points on \mathcal{M} . The distance function $d : \mathcal{M} \times \mathcal{M} \rightarrow [0, \infty)$ is defined as,

$$d(x, y) = \inf_{\gamma} \{x, y \in \gamma\}$$

In simpler terms, the distance between them is the minimum length of all the curves that start at x and end at y .

2.2.2.2 Exponential and the logarithm maps

We can simply add or subtract two vectors to get a third one. However, if we apply the same notion over here, we may get a point which does not lie on the manifold. Hence, it becomes important to define the operations of addition and subtraction carefully. There are two important functions which do this — \exp and \log .

The exponential map $\exp_x : T_x\mathcal{M} \rightarrow \mathcal{M}$ is defined as,

$$\exp_x(v) = \gamma_v(1)$$

This signifies that point on \mathcal{M} which lies in the direction of v and is one unit distance away from x . Let's give this an informal interpretation. The constant e is defined to be the limit $\lim_{x \rightarrow 0} (1 + x)^{\frac{1}{x}}$ from calculus. This can be understood as adding some small quantity to 1 and see what it is, followed by adding a small quantity to this new entity and repeating the operating infinite number of times. The exponential map is analogous. Since directly adding two points on a manifold may push it away from it, we add small increments to a point several times until we have reached a point which is 1 unit away from it in a particular direction. These small increments help ensure that we are still on the manifold and the

operations are meaningful. The inverse of the exponential map is the logarithm map,

$$\log_x : \mathcal{M} \rightarrow T_x \mathcal{M}$$

Under some assumptions of the global existence of the exp and the log maps, we get the following two important results:

1. $d(x, y) = \|\log_x(y)\|_x$ where $\|v\|_x$ is the length of the vector $v \in T_x \mathcal{M}$
2. $d^2(x, \cdot)$ is a smooth function for all $x \in \mathcal{M}$

2.2.3 Dictionaries on spheres

The usual definition of a dictionary searches for a set A , whose linear combination gives us the best set of points X . However, this linear combination makes no sense on a sphere. Hence, we want to look for a generalization of $\mathbf{X}_i = \mathbf{A}\mathbf{W}_i$ since our spherical manifold does not support a global vector space structure. Riemannian manifolds help us in this regard by providing a tangent space, $T_x \mathcal{M}$, at point $x \in M$ (Riemannian manifold) which allows us to extract global information using the exponential and logarithm maps.

Unlike the Euclidean case where we could exploit the vector structure we do not have that freedom here. The notion of “origin” is not present here since the tangent space is locally defined. The point x can be interpreted as the origin for $T_x \mathcal{M}$. Since we want to model the linear reconstruction nature of dictionaries for \mathcal{M} , we impose an affine constraint on the coefficients. So if,

$$\mathbf{X}_i = w_{1i} \mathbf{A}_1 + w_{2i} \mathbf{A}_2 + \cdots + w_{mi} \mathbf{A}_m$$

then by setting

$$w_{1i} + w_{2i} + \cdots + w_{mi} = 1$$

for each i gives us an origin independent environment.

[6] shows that

$$\min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \left\| \sum_{j=1}^m W_{ij} \log_{x_i}(\mathbf{A}_j) \right\|_{x_i}^2 + \lambda \|\mathbf{W}\|_1$$

$$\sum_{j=1}^m W_{ij} = 1, \forall i = 1, 2, \dots, n$$

is a good optimization problem for learning the dictionaries on \mathcal{M} .

For unit spheres, on putting the appropriate log function, this boils down to,

$$\min_{\mathbf{A}, \mathbf{W}} \sum_{i=1}^n \left\| \sum_{j=1}^m W_{ij} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle) \frac{\mathbf{u}_{ij}}{\|\mathbf{u}_{ij}\|} \right\|_{x_i}^2 + \lambda \|\mathbf{W}\|_1$$

$$\mathbf{u}_{ij} = \mathbf{A}_j - \langle \mathbf{X}_i, \mathbf{A}_j \rangle \mathbf{X}_i$$

$$\sum_{j=1}^m W_{ij} = 1, \forall i = 1, 2, \dots, n$$

where $\langle \cdot, \cdot \rangle$ is the vector dot product.

Unfortunately, there is no closed form iterative algorithm to get convergence as unlike the previous method. We resort to our old friend, gradient descent, to reach a convergence condition. Here, one needs to be careful because we are working under the assumption of a unit sphere. So for the log maps to be meaningful we need to assert that $\|\mathbf{A}_j\|_2 = 1$ and $\|\mathbf{X}_i\|_2 = 1$. This requires the use of projected gradient descent. Furthermore, the affine constraint also needs to be maintained. The procedure to do this can be summarized as follows.

2.2.4 The algorithm

Algorithm 2 Non-linear sparse coding

```
1: Initialize  $\mathbf{A} \in \mathbb{R}^{d \times n}$ 
2:  $\mathbf{A} \leftarrow \text{normc}(\mathbf{A})$ 
3: Initialize  $\mathbf{W} \in \mathbb{R}^{n \times m}$ 
4: repeat
5:   repeat
6:      $\mathbf{A1} \leftarrow \mathbf{A} - \mu_A \nabla_{\mathbf{A}} f(\mathbf{X}, \mathbf{A}, \mathbf{W})$ 
7:      $\mathbf{A} \leftarrow \text{normc}(\mathbf{A1})$ 
8:   until  $\mathbf{A}$  converges
9:   repeat
10:     $\mathbf{W1} \leftarrow \mathbf{W} - \mu_W \nabla_{\mathbf{W}} f(\mathbf{X}, \mathbf{A}, \mathbf{W})$ 
11:     $\mathbf{W} \leftarrow \text{bsxfun}(@\text{rdivide}, \mathbf{W1}, \text{sum}(\mathbf{W1}, 2))$ 
12:   until  $\mathbf{W}$  converges
13: until both  $\mathbf{A}$  and  $\mathbf{W}$  converge
```

In the above algorithm, columns of \mathbf{A} represent the dictionary atoms and the rows of \mathbf{W} represent the coding coefficients.

2.2.5 Gradients

The gradient needs to be explicitly calculated. After a tedious calculation and optimizations, the gradients can be summed up as follows.

Let's just focus on the first term of the optimization function. The second terms if the l_1 norm and is easy to deal with. Furthermore, it has no contribution to $\nabla_{\mathbf{A}} f$.

$$\mathbf{v}_{ij} = W_{ij} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle) \frac{\mathbf{u}_{ij}}{|\mathbf{u}_{ij}|}$$

$$f(\mathbf{X}, \mathbf{A}, \mathbf{W}) = \sum_{i=1}^n \left\| \sum_{j=1}^m \mathbf{v}_{ij} \right\|^2$$

Note that \mathbf{u}_{ij} and \mathbf{v}_{ij} are vectors in \mathbb{R}^d . Let's denote the k^{th} component of any vector \mathbf{x} as $x^{(k)}$ and its transpose as \mathbf{x}^T .

Hence,

$$v_{ij}^{(k)} = W_{ij} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle) \frac{u_{ij}^{(k)}}{|\mathbf{u}_{ij}|}$$

The gradient with respect to W_{pq} is,

$$\frac{\partial f}{\partial W_{pq}} = 2 \frac{\cos^{-1}(\langle \mathbf{X}_p, \mathbf{A}_q \rangle)}{|\mathbf{u}_{pq}|} \mathbf{u}_{pq}^T \left(\sum_{j=1}^m \mathbf{v}_{pj} \right)$$

The gradient with respect to A_{st} is,

$$\frac{\partial f}{\partial A_{st}} = 2 \sum_{i=1}^n \sum_{j=1}^m (F_5(i, j) \langle \mathbf{u}_{ij}, \mathbf{u}_{it} \rangle + F_6(i, j) u_{ij}^s + F_7(i, j) \langle \mathbf{X}_i, \mathbf{v}_{ij} \rangle)$$

$$\mathbf{F}_1(i, \cdot) = -\frac{W_{it} X_{si}}{|\mathbf{u}_{it}|^2}$$

$$\mathbf{F}_2(i, \cdot) = \frac{W_{it} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_t \rangle)}{|\mathbf{u}_{it}|}$$

$$\mathbf{F}_3(i, \cdot) = -\frac{W_{it} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_t \rangle)}{|\mathbf{u}_{it}|} X_{si}$$

$$\mathbf{F}_4(i, \cdot) = \frac{W_{it} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_t \rangle)}{|\mathbf{u}_{it}|^3} X_{si} \langle \mathbf{X}_i, \mathbf{A}_t \rangle$$

$$\mathbf{L}(i, j) = \frac{W_{ij} \cos^{-1}(\langle \mathbf{X}_i, \mathbf{A}_j \rangle)}{|\mathbf{u}_{ij}|}$$

$$\mathbf{F}_5 = (\mathbf{F}_1 + \mathbf{F}_4) \cdot * \mathbf{L}$$

$$\mathbf{F}_6 = \mathbf{F}_2 \cdot * \mathbf{L}$$

$$\mathbf{F}_7 = \mathbf{F}_3 \cdot * \mathbf{L}$$

2.2.6 Some implementation details

The gradient computations, as can be seen, are extremely complicated operations. A lot of optimizations were done to make them memory as well as time efficient. In conjunction with this, two factors can be considered to speed up the convergence of the algorithm.

1. The initialization, if far away from the optimum, can take a long time to converge. This project used K-means for this.
2. The rate parameters μ_A and μ_W determine the speed at which the algorithm converges. We use adaptive gradient descent to tweak the step-size at each step of convergence.
3. The RMSProp optimizer also builds on the idea of adaptive gradient descent but chooses the learning rate dynamically based on the data. The convergence is faster and can often avoid local minimas.

For the purpose of these experiments, an adaptive variant of RMSProp was used where the learning rate was both adaptively chosen from (2) and the one in the RMSProp algorithm.

Chapter 3

Application of dictionaries for MNIST classification

Dictionaries can be used well for classification tasks as well. The coefficient vector is a compressed representation for the image with respect to the dictionary matrix. This experiment was done using the non-negative sparse coding algorithm described in section 2.1.

3.1 Experiment

We conducted the experiments using the above mentioned coding approach on the MNIST dataset of handwritten digits. This data set contains 28 x 28 images the digits 0 to 9 written by people in different styles. There are 60,000 images for training and 10,000 for testing. For the purpose of this experiment, the images were resized to 14 x 14 and the random forest classifier with 100 trees was used.

The approach taken is as follows:

1. We consider images of one class at a time, $label \in 0, 1, \dots, 9$. Label the set of images \mathbf{X}^{label}

2. Set the dictionary of atoms to be learned for each class, say \mathbf{A}^{label}
3. Use NNMF to learn the dictionary \mathbf{A}^{label} along with sparsity regularization. The hyper-parameter λ dictates this. Too high a sparsity, will lead to fewer atoms being used to determine the image reconstruction, and vice-versa
4. Now that we have the learned atoms $\{\mathbf{A}^0, \mathbf{A}^1, \dots, \mathbf{A}^9\}$
5. Following this we combine all the atoms and the data points together, i.e., set

$$\mathbf{A} = [\mathbf{A}^0 \mathbf{A}^1 \dots \mathbf{A}^9]$$

$$\mathbf{X} = [\mathbf{X}^0 \mathbf{X}^1 \dots \mathbf{X}^9]$$

6. Using these \mathbf{A} , we fit \mathbf{X} to it, i.e., we keep \mathbf{A} constant and find the \mathbf{W} which minimizes the cost function. In other words, we sparse code the data. Refer to Algorithm 2 for a detailed description. This step gives us the coefficients that define the recon-

Algorithm 3 Sparse coding the learned dictionary

- 1: $t \leftarrow 0$
 - 2: Initialize \mathbf{W}^0 to strictly positive values
 - 3: **repeat**
 - 4: $\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t \cdot * (\mathbf{A}^T \mathbf{X}) ./ (\mathbf{A}^T \mathbf{A} \mathbf{W}^t + \lambda)$
 - 5: $t \leftarrow t + 1$
 - 6: **until** convergence
-

struction of images from the atoms. Note that it is possible that an image of, say class 7, may have components from different classes. The algorithm does nothing to make this distinction. However, we would expect that the atoms corresponding to class 7 will have a major contribution to the data points from class 7. Our project is trying to validate the same hypothesis, and as it turns out, it does it really well.

7. Now, we fit a random forest classifier with 100 trees. The input to the classifier are the sparse coded coefficients and the labels are their corresponding classes.
8. The validation of the model is done in a similar manner. The testing data is sparse coded using the dictionary matrix, \mathbf{A} , and the corresponding sparse coefficient vectors are fed to the trained classifier. The classifier tells us which class the coefficient vector should belong to.

3.2 Results

The average classification accuracy was 94.610000% for the 10,000 testing samples. The class-wise breakdown is as follows: In this model, 10 atoms were used for each class. The

Digit	Accuracy(%)
0	98.36
1	98.85
2	96.31
3	94.65
4	95.11
5	92.15
6	97.07
7	94.55
8	87.16
9	90.98

Table 3.1: Accuracy for each digit

coefficient λ was set to 0.

Chapter 4

Application of dictionaries on image patches for image denoising

Dictionary learning has been applied to a good degree for images. Let us first understand how do we represent images as vectors to use the previously developed formalism. To do this, we introduce the notion of a patch within an image. A patch is a rectangular sub-region in an image which essentially is a part of the image. We take a patch and vectorize it to convert it into a vector which can be used in the algorithm.

The interesting question is what does a dictionary for an image mean? Images have a lot of interesting features in them — edges, texture, etc. — and we want to be able to represent them using atoms. If we are able to identify these building blocks of an image, we can use their affine combinations to reconstruct image patches. If we consider overlapping patches in our image, then a single pixel is a part of multiple patches. We can find the intensity of that pixel by an average over all the patches in which the pixel lies. If the original image contains noise, then the averaging will reduce the noise and bring us closer to the actual image. Note that this method of taking affine combinations is not exact as we are working on spheres but it works reasonably well.

One subtle point to note is not all patches help learn the dictionary well. A patch which is of constant intensity does not provide interesting insights into the structural components

of the image and hence we can safely ignore it. Building on this, only those patches which have a high amount of variance should be considered for the training process. For the reconstruction phase, we add a constant intensity patch to the learned dictionary and try to find the coefficients whose affine combination will construct the image with the minimum error.

One must remember that patches have a lot of variations among themselves. One patch may be brighter while other may be dull. Some patches may have different range of intensities. Therefore, we want to get rid of these and give a uniform nature to all the patches.

4.1 Reconstruction algorithm

These algorithms are based on the non-linear sparse coding method described in section 2.2.

Algorithm 4 Learning the dictionary

- 1: Find all high variance patches within the image using a threshold
 - 2: Vectorize all the high variance patches to get the matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$
 - 3: $\mathbf{X} \leftarrow \text{normc}(\mathbf{X})$
 - 4: Initialize $\mathbf{A} \in \mathbb{R}^{d \times n}$ using K-means
 - 5: $\mathbf{A} \leftarrow \text{normc}(\mathbf{A})$
 - 6: Initialize $\mathbf{W} \in \mathbb{R}^{n \times m}$ using K-means
 - 7: **repeat**
 - 8: Converge A keeping W fixed
 - 9: Converge W keeping A fixed
 - 10: **until** both A and W converge
-

Algorithm 5 Reconstructing the image back

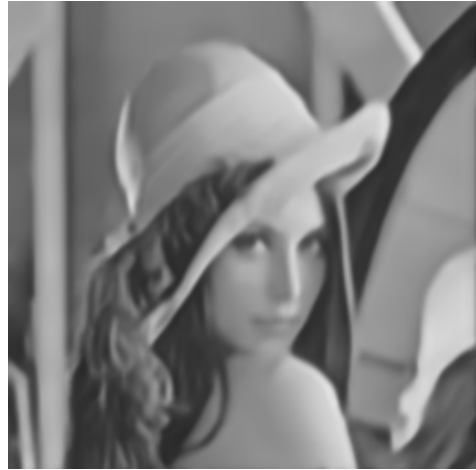
- 1: Vectorize **all** the patches to form the matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$
 - 2: $\mathbf{N} \leftarrow \text{repmat}([\|\mathbf{X}_1\|, \|\mathbf{X}_2\|, \dots, \|\mathbf{X}_n\|], d, 1)$
 - 3: $\mathbf{X} \leftarrow \text{normc}(\mathbf{X})$
 - 4: Get the dictionary matrix $\mathbf{A} \in \mathbb{R}^{d \times n}$ from Algorithm 4
 - 5: $\mathbf{A}^{(1)} \leftarrow [\mathbf{A}, \mathbf{P}]$ where $\mathbf{P} \in \mathbb{R}^d$ is the unit norm constant intensity patch
 - 6: Initialize $\mathbf{W}^{(1)} \in \mathbb{R}^{n \times m}$ using KNN search from the dictionary $\mathbf{A}^{(1)}$
 - 7: Fit $\mathbf{A}^{(1)}$ to \mathbf{X} to get the new coefficient matrix $\mathbf{W}^{(1)}$
 - 8: $\mathbf{X}^{(2)} \leftarrow (\mathbf{A}^{(1)}\mathbf{W}^{(1)}) \cdot * \mathbf{N}$
 - 9: Reshape columns of $\mathbf{X}^{(2)}$ into patches and find average intensity of pixel
-

4.2 Experiments and results

The experiment was done with the standard Lena test image of size 256×256 using the second algorithm. 9×9 sized high variance patches were considered for the experiments. 50 atoms were considered for the experiment.



(a) Lena reconstructed after k-means



(b) Lena reconstructed after iteration

Figure 4.1: Comparison of reconstruction before and after convergence

The denoised image appears more blurred compared to the corresponding image on the left because of possibly two reasons - (1) presence of noise and (2) lesser number of training points used for the second case to save on computational resources. The gradient computation for matrix A takes a lot of time and that is the major bottleneck of the algorithm and increasing the number of training points proportionally blows up the training time both for learning and fitting the atoms.

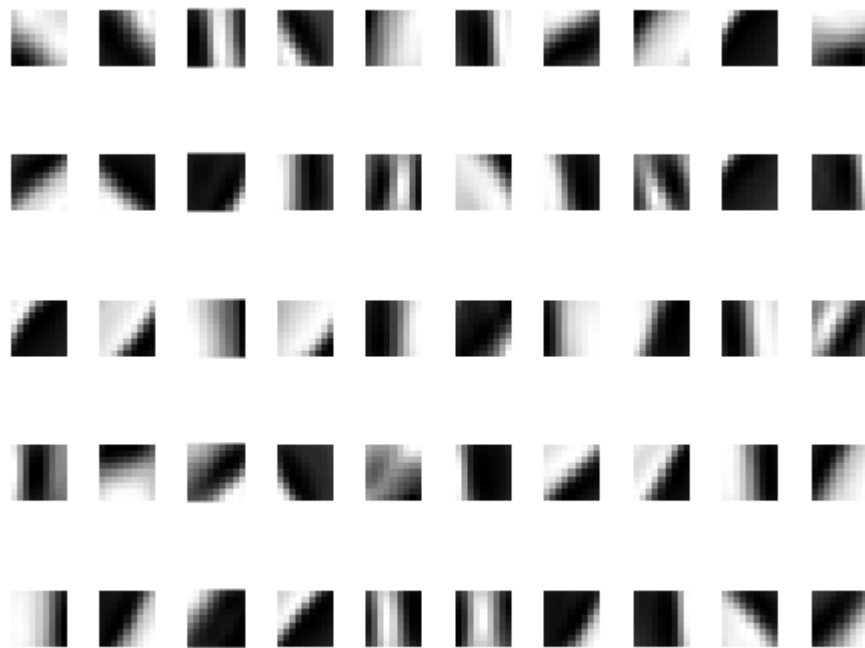




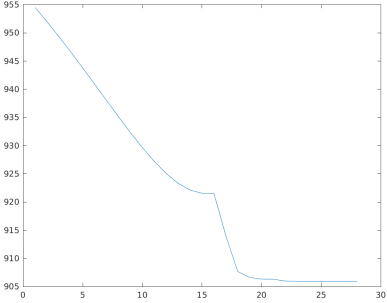
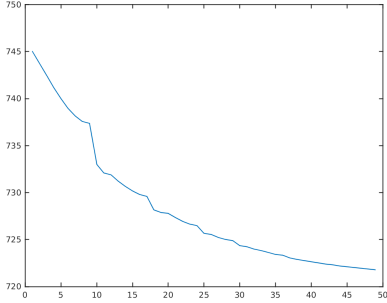
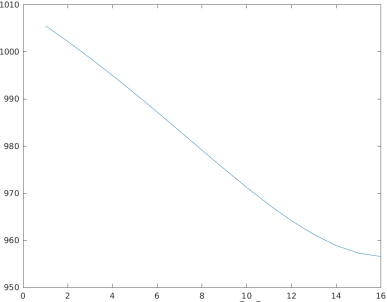
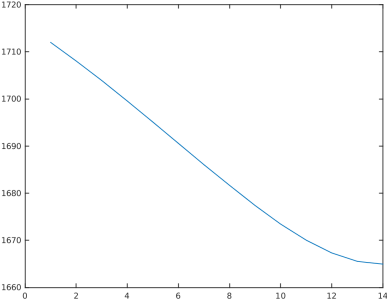


Figure 4.2: 50 atoms for the Lena image

These 50 atoms were learned by the algorithm. The different segments within the image are the various atoms. One can see that the atoms are different and each atom captures the intricate and edge-like features from the image. Since we are not allowing reflections, so different orientations of edges are counted separately. Also edges occurring at different

place, although, in the same orientations account as different atoms.

The algorithm also does better than the basic k-means initialization as can be seen in figure 4.1.

	<p># of training points: ~33K # of atoms: 50</p>	<p># of training points: ~12K # of atoms: 50 Noise: sigma 5% of intensity range</p>
Original image		
Reconstructed image		
Objective function for learning the dictionary		
Objective function for fitting the patches to the dictionary		

Chapter 5

Dictionary for shapes

This section refers to method developed in [1] [4] and [2]. Shapes play an important role in medical image processing. Most body parts have a fixed shape and structure. Defects and diseases can often be identified by analysis of the structural changes in them. This motivates one string factor for the study of shape analysis.

Dealing with shapes requires a different way to represent them. A shape is a continuous curve and we choose some k points on it to get a discrete sampling. If the shape is in a d dimensional space, then we essentially get $k \times d$ dimensional shape space. Since we are working with images, $d = 2$ or $d = 3$. This part of the thesis deals with image for which $d = 2$ or 2D images. Let's denote a shape, \mathbf{S} using some sampled points as $\mathbf{S} = \left[\begin{pmatrix} x_1^{(1)} \\ x_1^{(2)} \end{pmatrix}, \begin{pmatrix} x_2^{(1)} \\ x_2^{(2)} \end{pmatrix} \dots \begin{pmatrix} x_k^{(1)} \\ x_k^{(2)} \end{pmatrix} \right]$. Note that these are all points on the same curve and not points on different curves.

To apply the previously developed formalism, we convert these set of points to a vector by concatenating the vectors. The vector representing this curve takes the form

$$\mathbf{X}_i = [x_1^{(1)}, x_1^{(2)}, x_2^{(1)}, x_2^{(2)}, \dots, x_k^{(1)}, x_k^{(2)}]^T$$

One issue resolved, there are more. The difficulty in coming up with a general notion of shape is difficult because of 4 factors:

1. Translation: Moving the image does not alter its shape
2. Scaling: Changing the scale by the same factor along all the axes also maintains the shape
3. Rotation: By changing the orientation of the image, the shape remains unaltered
4. Reflection: Reflection about a place or axis also keeps the shape same. We won't be dealing with reflections as part of this thesis

Hence, before moving on to analyzing shapes and their similarities, we need to remove the first three factors mentioned above. If this were not so then the notion of distance between two shapes won't be meaningful. We need to get them to a common coordinate frame and that is done step by step as follows:

1. Translation: To get away with this, we shift the centroid of the shape to the origin. This gives us a consistent notion of placement of a shape in the coordinate axis — at the origin. The centroid, μ of a shape S is

$$\mu = \frac{1}{k} \sum_{i=1}^k \begin{pmatrix} x_i^{(1)} \\ x_i^{(2)} \end{pmatrix}$$

To remove translation, $\bar{x}_i = x_i - \mu$. This is then further converted to the combined vector representation \bar{X}_i .

2. Scaling: We set the variance of \bar{X}_i to 1 by dividing it by its norm to consistently get a unit norm shape
3. Rotation: As discussed above, we get the same shape on rotation. So if we have two shapes, S_1 and S_2 , the distance between them would have been defined between them as,

$$d(S_1, S_2) = \|S_1 - S_2\|_F^2$$

This is essentially measuring how far away the component points of the shapes are far away from each other. However, note that the we can possibly find a smaller distance

between them by rotating the shapes and aligning them to each other using some distance metric. This idea is captured by what is called the Procrustes distance. We find an orthogonal rotation matrix which finds the nearest shape for a given shape, Ω such that

$$\begin{aligned}\Omega^{\text{opt}} &= \arg \min_{\Omega \in \mathbb{R}^{d \times d}} \|\Omega \mathbf{S}_1 - \mathbf{S}_2\|_F \\ \Omega^T \Omega &= \mathbb{I}, \det(\Omega) = 1\end{aligned}$$

This finds the “nearest” shape in the squared error sense. We will use this to align one shape with other. This above problem has a closed form solution given by $\Omega^{\text{opt}} = \mathbf{U}\mathbf{V}^T$ where \mathbf{U} and \mathbf{V} are the left and right singular matrices of $\mathbf{M} = \mathbf{S}_2\mathbf{S}_1^T$. If $\det(\Omega^{\text{opt}}) = -1$ then we flip the sign of one of the left singular values. MATLAB does this internally through the function `procrustes()`.

Having addressed these issues, we are now ready to develop the methodology for learning shapes using spheres. The algorithm needs to be modified at three places to accommodate shapes. First, while computing the value of the optimization function, $f(\mathbf{X}, \mathbf{A}, \mathbf{W})$, we need to align each \mathbf{A}_j with each \mathbf{X}_i so that the log map operates on the closest shape. Secondly, while computing $\nabla_{\mathbf{A}_j} f$, we need to align each \mathbf{X}_i with \mathbf{A}_j . This ensures that all the data are aligned with the atom and the log map gives us a consistent notion of distance. If this were not done, then the distance might be with respect to distance projections. Finally, while computing $\nabla_{\mathbf{W}_i} f$, we align each \mathbf{A}_j with \mathbf{X}_i . Similar to above, this is also done to get a uniform distance metric.

5.1 Algorithm for shapes

The overall algorithm is a costly one. This can be attributed to that fact the operations involve aligning one shape with another. The alignment requires computing the SVD of a particular matrix which has been described in the previous section and it adds to the computational time. This algorithm is based on the non-linear sparse coding method described in section 2.2.

Algorithm 6

- 1: Input: shapes S_1, S_2, \dots, S_n each defined by k points
 - 2: Preprocess the shapes to remove translation and scale factor
 - 3: Vectorize the processed shapes as described earlier to form the matrix $\mathbf{X} \in \mathbb{R}^{dk \times n}$
 - 4: Initialize $\mathbf{A} \in \mathbb{R}^{dk \times n}$ using K-means
 - 5: $\mathbf{A} \leftarrow \text{normc}(\mathbf{A})$
 - 6: Initialize $\mathbf{W} \in \mathbb{R}^{n \times m}$ using K-means
 - 7: **repeat**
 - 8: **repeat**for each \mathbf{A}_j
 - 9: Align each \mathbf{X}_i with \mathbf{A}_j using the Procrustes distance metric to get $\bar{\mathbf{X}}$
 - 10: $\mathbf{A}\mathbf{1}_j \leftarrow \mathbf{A}_j - \mu_A \nabla_{\mathbf{A}_j} f(\bar{\mathbf{X}}, \mathbf{A}_j, \mathbf{W})$
 - 11: $\mathbf{A}_j \leftarrow \text{normc}(\mathbf{A}\mathbf{1}_j)$
 - 12: **until** \mathbf{A}_j converges
 - 13: **repeat**for each row vector \mathbf{W}_i
 - 14: Align each \mathbf{A}_j with \mathbf{X}_i using the Procrustes distance metric to get $\bar{\mathbf{A}}$
 - 15: $\mathbf{W}\mathbf{1}_i \leftarrow \mathbf{W}_i - \mu_W \nabla_{\mathbf{W}_i} f(\mathbf{X}, \bar{\mathbf{A}}, \mathbf{W}_i)$
 - 16: $\mathbf{W}_i \leftarrow \text{bsxfun}(@\text{rdivide}, \mathbf{W}\mathbf{1}_i, \text{sum}(\mathbf{W}\mathbf{1}_i, 2))$
 - 17: **until** \mathbf{W}_i converges
 - 18: **until** both \mathbf{A} and \mathbf{W} converge
-

5.2 Experiments and results

The aim of the experiments for this section is to find a sparse dictionary that can represent all the images well enough that we have in our dataset. The experiment for this section were performed in 2 stages in varying order of complexities.

5.2.1 Ellipses

This experiment was conducted using a very simple manually constructed dataset. We used a set of ellipses with their axes aligned along the x and y-directions. The major axis

was along the x-direction, with the semi-major axis length = 0.5 and the semi-minor axis length varied from 0.1 to 0.4 with step size of 0.01 units giving us 31 ellipses to learn the model from. Each ellipse had 36 points describing its shape. The points were separated by 10 degree angles. The algorithm does excellent for this simple model. Figure 5.2 shows the reconstructed as well the original ellipses, just that the original set of ellipses has been masked by the reconstructed ellipses almost exactly. The RMS error over all the ellipses was just 0.082037 which explains the close similarity. The figure 5.1 shows the two atoms used to reconstruct the ellipses of figure 5.2. Figure 5.3 shows how a particular ellipse is constructed by taking the linear combination of the two atoms. The linear addition is done by first aligning each atoms with the data points and then reconstructing it.

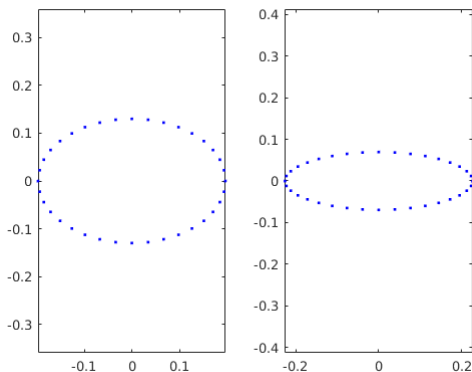


Figure 5.1: Two atoms used to construct the ellipses

5.2.2 Leaf images

This experiment was done on the leaf dataset collected by Sougata Singha. This dataset has images of several leaves of different tree species. The boundary points have been computed for each leaf which help analyze the shapes of the leaves. Each leaf has been marked with 100 points along its boundary with a good amount of correctness. We learn shapes using these points for this leaf dataset. The experiments below are in two parts. The first section

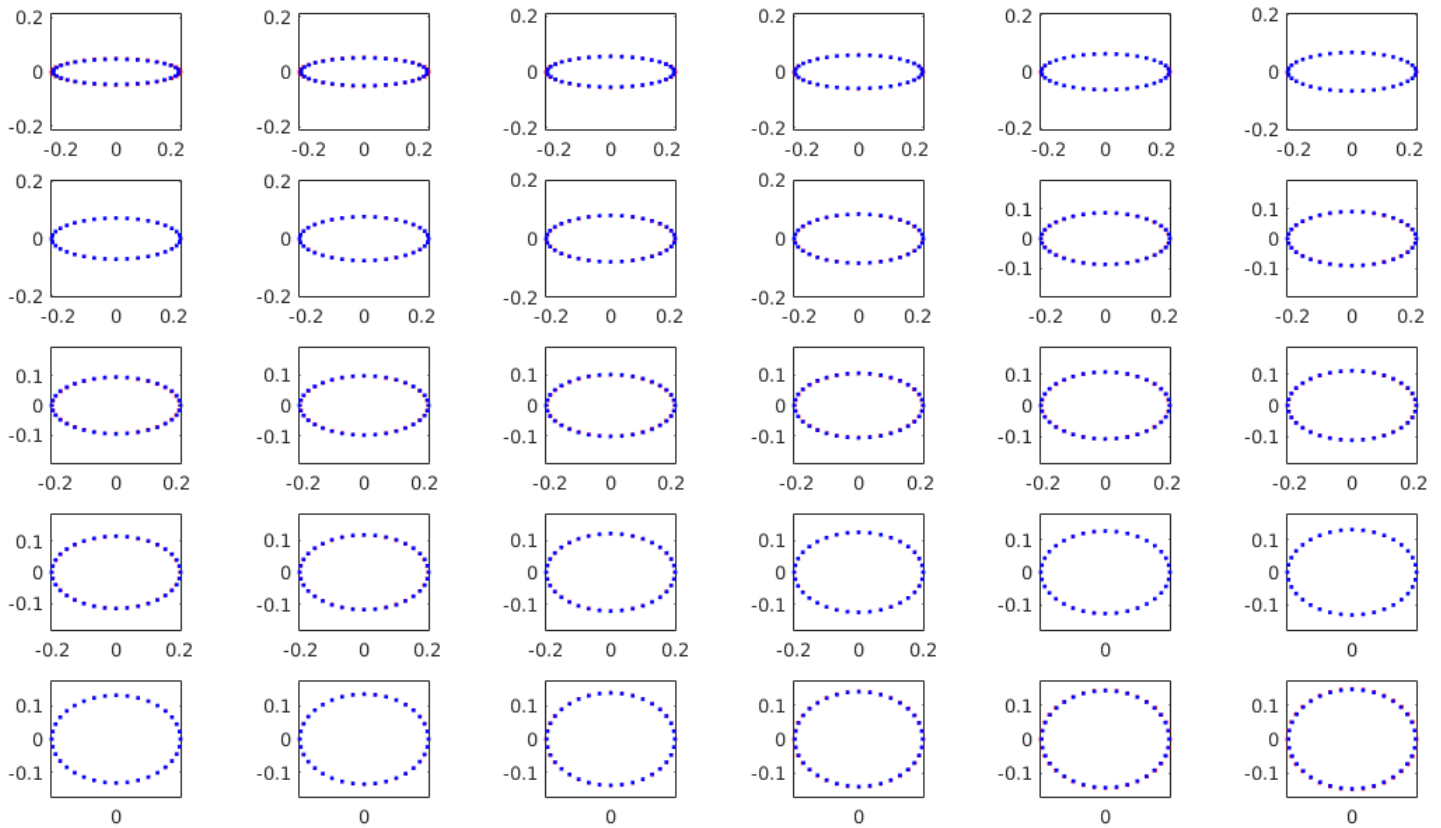


Figure 5.2: Reconstructed ellipses using 2 atoms

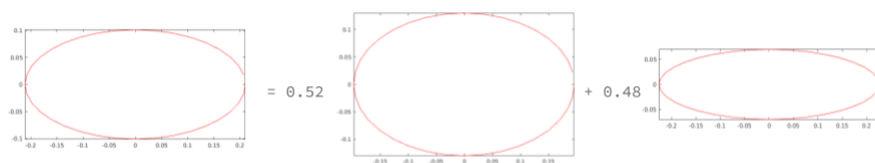


Figure 5.3: Reconstruction of ellipse using its atoms

does shape analysis for just one leaf type from the tree *Anhui barberry* and the second experiment analyses the joint model for *Anhui barberry* and *Trident maple*.

5.2.2.1 Anhui barberry

The dictionary was learned using 5 atoms. The results have been depicted in figure 5.5. The leaf atoms shown in figure 5.4 capture the different shapes of leaves as are there in the dataset. The average reconstruction RMS error was 0.18. We observe that the average reconstruction accuracies do not vary much as the number of atoms are varied in the dataset. Even with 2 atoms, the error is 0.1836 which matches that for $m = 5$ atoms. With 10 atoms, the error is 0.1491. Smaller number of atoms suffice here because the shape structures are similar as the leaves are from the same tree. With more number of leaf types to learn, we incrementally need more atoms as is shown in the next section.

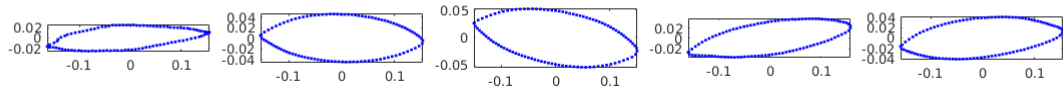


Figure 5.4: Atoms used to construct the leaves

5.2.2.2 Trident maple

In this case, there was more variation when the number of atoms were varied. For $m = 5$ atoms, the average reconstruction error was 0.32 and with $m = 10$, the error was 0.24. The atoms in this are more varied and capture more orientations compared to the previous leaf. This is because this leaf has a more directed and varying structure and smaller number of atoms are not able to capture all of them changes. Figures 5.6 and 5.7 are the corresponding images for this leaf.

5.2.2.3 Joint learning of shapes

A model was learned on the combined leaf shapes of the two trees mentioned above. The number of atoms were set of 10 and the reconstruction error was 0.24.

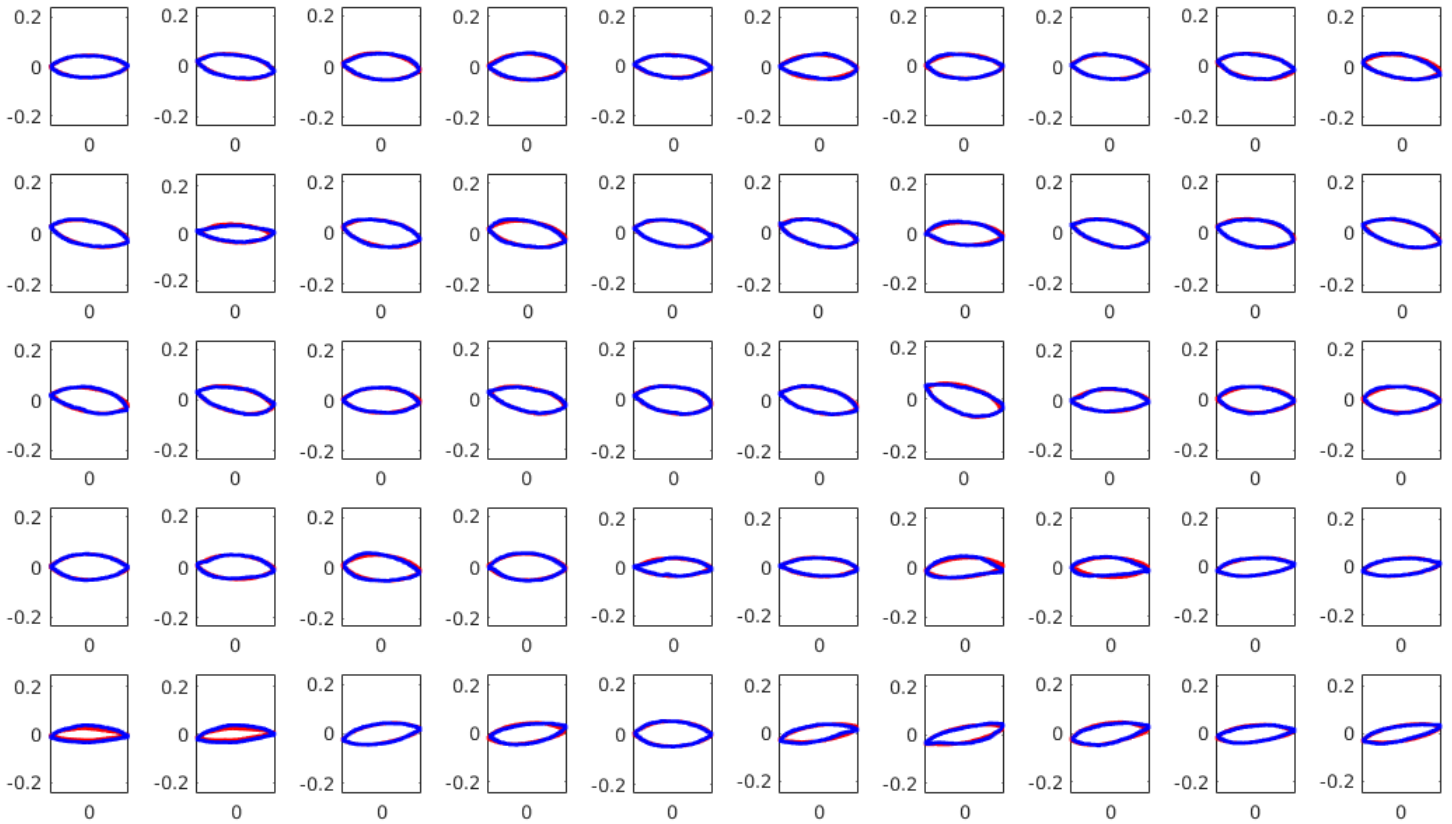


Figure 5.5: Reconstructed leaves using 5 atoms

We would want the weights to be higher for the atoms which are closer to the shape than other atoms. This is what the algorithm does as well. Table 5.1 lists down the weights for the 10 atoms from figure 5.10 which help reconstruct this leaf back. The weights are higher for the atoms which are similar to this leaf. We see the highest weight for the first atom which most closely resembles this leaf. Smaller weights are those which correspond to the atoms from the tree trident maple.

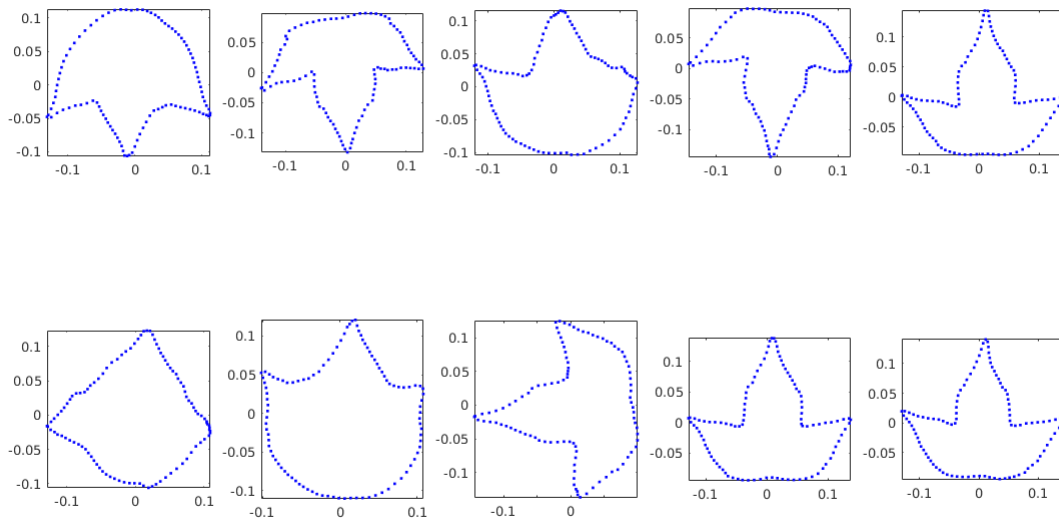


Figure 5.6: Atoms used to construct the leaves

1.2992	-0.1307	0.0868	-0.0298	-0.0370	-0.0426	0.0504	0.0125	-0.0869	-0.1220
--------	---------	--------	---------	---------	---------	--------	--------	---------	---------

Table 5.1: Weights for the 10 atoms for leaf in 5.10

The algorithm adapts to the dictionary and it can be used with good affect for shape analysis. Though the approach is computationally more expensive because of operations on a hypersphere which take up time for normalization operations, the results are promising and can be extended to general real life cases such as medical image analysis.

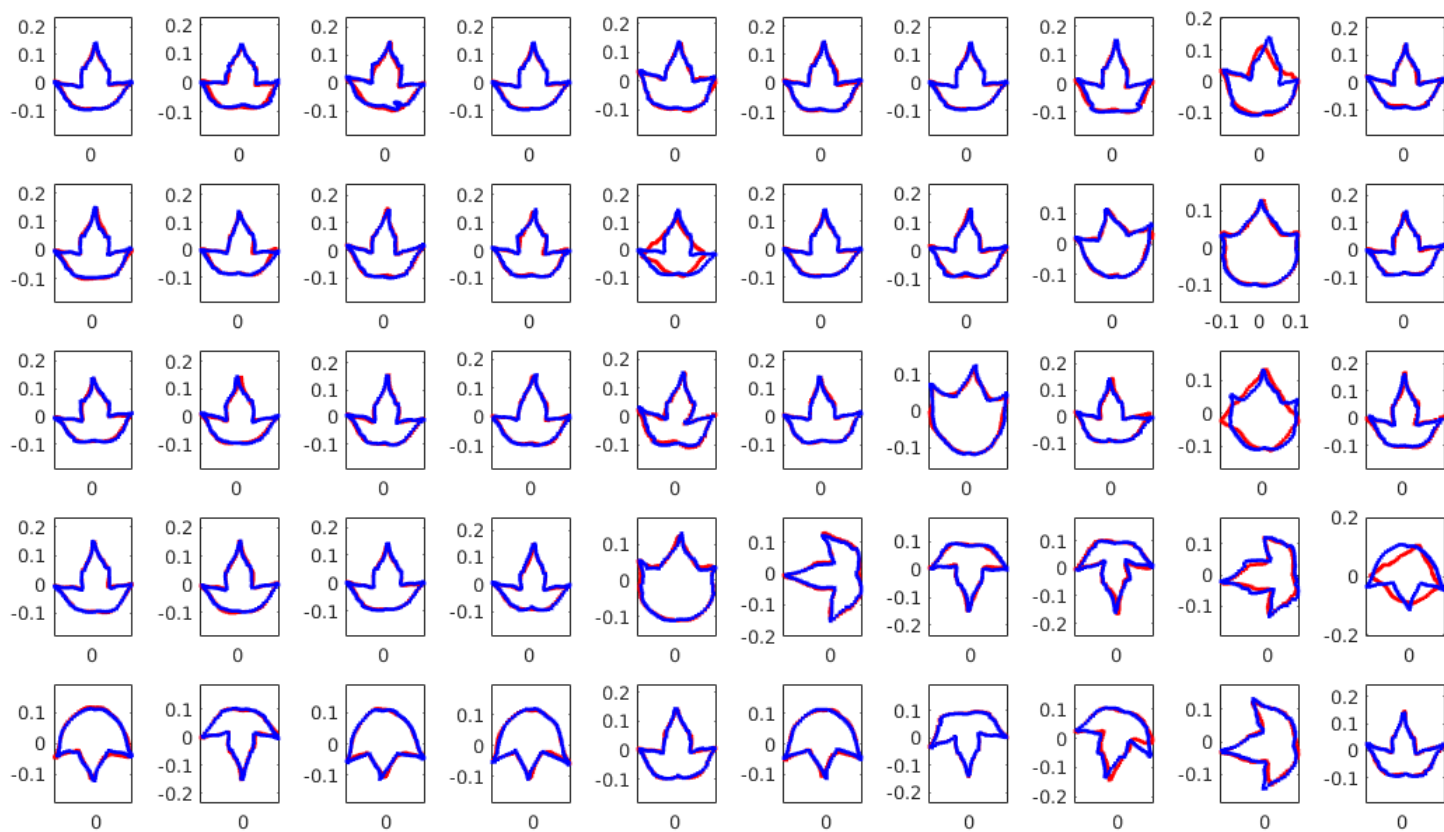


Figure 5.7: Reconstructed leaves using 10 atoms

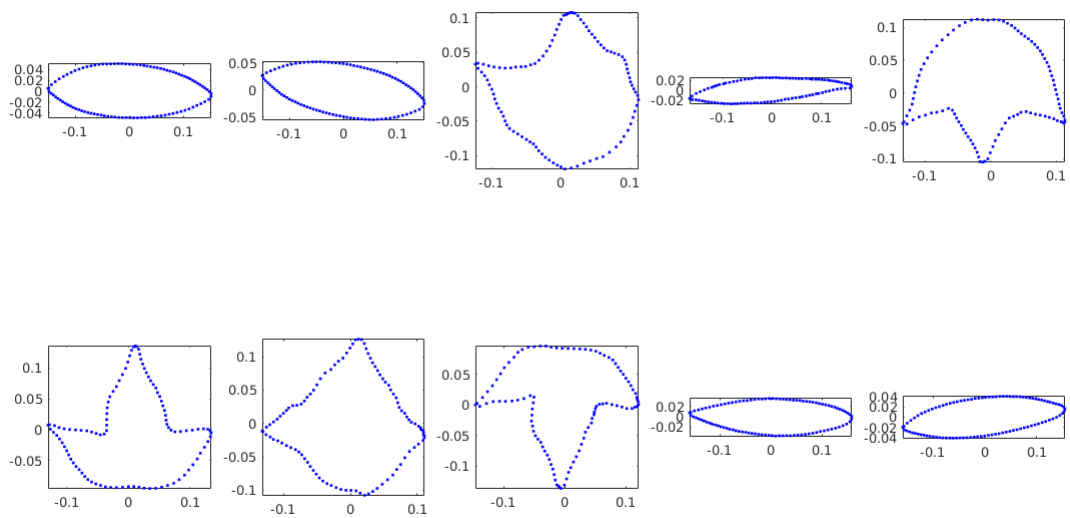


Figure 5.8: Atoms used to construct the leaves

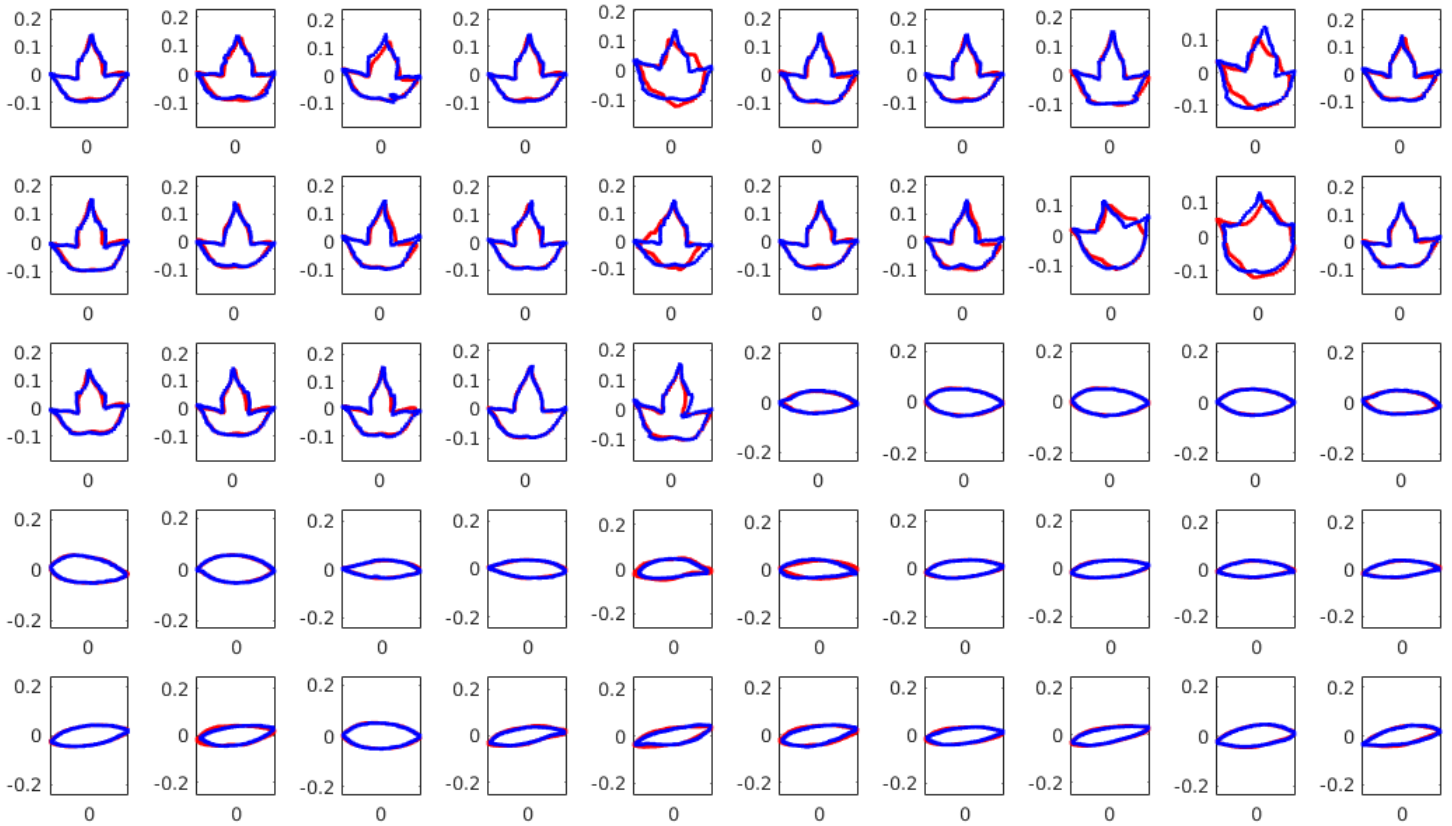


Figure 5.9: Reconstructed leaves using 10 atoms

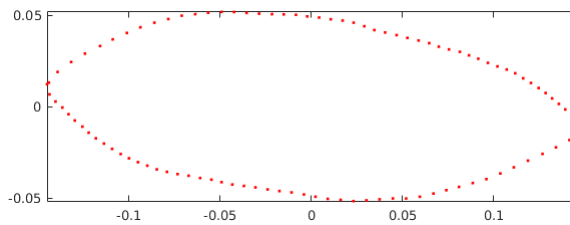


Figure 5.10: A leaf from Anhui barberry reconstructed from the mixed dictionary

Chapter 6

Conclusion

This work revolves around the use of dictionaries for various tasks in image processing. The paper talks about the meaning of dictionaries and discusses two ways of finding them. The first approach, based on non-negative matrix factorization has been used for image classification and the second version based on Riemannian manifolds has been applied to the task of denoising and shape analysis. The first approach works well for image classification achieving accuracies of around 94%. The denoising and shape analysis parts are also quite good. For the denoising task, the algorithm is able to learn the unique features from the image. The algorithm for shape analysis generalizes well for different shapes and learns weights based on the closeness with the atoms in the dictionary. This seems promising avenue of search and more work needs to be done before they can have full fledged applications, as discussed in the next section.

Chapter 7

Future Work

The main application of shape analysis lies in medical image processing. Medical images often are faced with the issue of occlusion. For instance, if an image has number of cells in it, then it is very likely that they won't be having disjoint boundaries and there would be a decent amount of overlap. To alleviate this problem, we would like to learn models which can help detect their outer boundary. We can ascertain that we have a cell in a section of the image if our algorithm is able to match it with a certain degree of accuracy. This is precisely what shape analysis gives us — a template for matching shapes present in images. Therefore, the future part of this project will be to learn shapes for more general types of structures and provide efficient segmentation for such images.

Bibliography

- [1] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59, 1995.
- [2] C. Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 285–339, 1991.
- [3] P. O. Hoyer. Non-negative sparse coding. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*, pages 557–565. IEEE, 2002.
- [4] D. G. Kendall. A survey of the statistical theory of shape. *Statistical Science*, pages 87–99, 1989.
- [5] R. Rubinstein, A. M. Bruckstein, and M. Elad. Dictionaries for sparse representation modeling. *Proceedings of the IEEE*, 98(6):1045–1057, 2010.
- [6] Y. Xie, J. Ho, and B. Vemuri. On a nonlinear generalization of sparse coding and dictionary learning.