To all who were there and to all who will come.

For ones who know.
ACKNOWLEDGMENTS

To all teachers; ones that adorned the form of humans and ones that came in shape of moments. To gods and devils, to friends and foes. To unrequited loves and countless crushes, ones who destroyed me and ones who made me better. To untamed desires and unlimited dreams (which one is better?); that kept urging to climb when thrown in the valleys of failures, ones that never let you be in peace, never let you be happy for what you have. To Allah, the most mercy-full. In the last to mere mortals, like me.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>7</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>8</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>12</td>
</tr>
<tr>
<td>1.1 Deconstructive Learning And Computer Vision</td>
<td>16</td>
</tr>
<tr>
<td>1.2 Related Work</td>
<td>21</td>
</tr>
<tr>
<td>1.3 Road Map</td>
<td>22</td>
</tr>
<tr>
<td>1.4 Etymology</td>
<td>23</td>
</tr>
<tr>
<td>2 DECONSTRUCTING KERNEL MACHINES</td>
<td>24</td>
</tr>
<tr>
<td>2.1 Problem Statement</td>
<td>25</td>
</tr>
<tr>
<td>2.2 Preliminaries</td>
<td>28</td>
</tr>
<tr>
<td>2.3 Deconstruction Method</td>
<td>30</td>
</tr>
<tr>
<td>2.3.1 Bracketing</td>
<td>31</td>
</tr>
<tr>
<td>2.3.2 Estimating Normal Vectors</td>
<td>33</td>
</tr>
<tr>
<td>2.3.3 Determine Kernel Subspace $\Sigma$</td>
<td>33</td>
</tr>
<tr>
<td>2.3.4 Determine Kernel Type</td>
<td>36</td>
</tr>
<tr>
<td>2.3.5 Complexity Analysis and Exact Recovery of $\Sigma$</td>
<td>37</td>
</tr>
<tr>
<td>2.4 Experiments</td>
<td>39</td>
</tr>
<tr>
<td>2.4.1 Evaluation of Deconstruction Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>2.4.2 Kernel Machine Upgrade Without Source Code</td>
<td>42</td>
</tr>
<tr>
<td>3 DECONSTRUCTING CASCADE OF LINEAR CLASSIFIERS</td>
<td>47</td>
</tr>
<tr>
<td>3.1 Problem Introduction</td>
<td>48</td>
</tr>
<tr>
<td>3.2 Preliminaries</td>
<td>50</td>
</tr>
<tr>
<td>3.3 Deconstruction Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>3.4 Experiments</td>
<td>59</td>
</tr>
<tr>
<td>3.4.1 Soft Cascade</td>
<td>59</td>
</tr>
<tr>
<td>3.4.2 Viola-Jones Haar Cascade</td>
<td>60</td>
</tr>
<tr>
<td>3.5 Concluding Remarks</td>
<td>62</td>
</tr>
<tr>
<td>4 TWO STAGE CLASSIFICATION SYSTEMS</td>
<td>63</td>
</tr>
<tr>
<td>4.1 Problem Introduction</td>
<td>63</td>
</tr>
<tr>
<td>4.2 Deconstruction Process and Method</td>
<td>67</td>
</tr>
<tr>
<td>4.2.1 Feature Identifiers</td>
<td>69</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.2.2 Deconstructor Methods</td>
<td>70</td>
</tr>
<tr>
<td>4.2.3 Geometric Compatibility Between Features and Deconstructors</td>
<td>70</td>
</tr>
<tr>
<td>4.3 Experiments</td>
<td>71</td>
</tr>
<tr>
<td>4.3.1 Distinguish between HOG and SIFT</td>
<td>72</td>
</tr>
<tr>
<td>4.3.2 Deconstructing OpenCV HOG-based Pedestrian Detector</td>
<td>73</td>
</tr>
<tr>
<td>4.3.3 Deconstructing Airplane Detector</td>
<td>76</td>
</tr>
<tr>
<td>4.3.4 Deconstructing when Dimensionality Reduction is Being Employed</td>
<td>76</td>
</tr>
<tr>
<td>5 CONCLUSIONS AND FUTURE WORK</td>
<td>80</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>84</td>
</tr>
<tr>
<td>BIOGRAPHICAL SKETCH</td>
<td>88</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Confusion matrices for the original kernel machine and the kernel machine</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>defined by the recovered quasi-support vectors</td>
<td></td>
</tr>
<tr>
<td>2-2</td>
<td>Comparisons of classification results for the original kernel machine and</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>the upgraded kernel machine</td>
<td></td>
</tr>
<tr>
<td>4-1</td>
<td>Effects of different image transforms on classification results. HOG-based</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>features as expected produce unstable results under rotation and scale change.</td>
<td></td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Schematic illustration of deconstructive learning</td>
<td>14</td>
</tr>
<tr>
<td>1-2</td>
<td>Proposed deconstructing algorithm for classifier with two component design.</td>
<td>21</td>
</tr>
<tr>
<td>2-1</td>
<td>Bracket process and the PN-pairs.</td>
<td>32</td>
</tr>
<tr>
<td>2-2</td>
<td>Calculating normals using PN-Pairs.</td>
<td>34</td>
</tr>
<tr>
<td>2-3</td>
<td>Intersections of $\mathbf{\Sigma}$ and two-dimensional affine subspaces</td>
<td>35</td>
</tr>
<tr>
<td>2-4</td>
<td>Singular values of the normal matrix $\mathbf{N}$ for different choices of $\delta$</td>
<td>42</td>
</tr>
<tr>
<td>2-5</td>
<td>Means and variances of the cosines of the twelve principal angles between $\mathbf{S}_Y$ and $\mathbf{S}_Y$.</td>
<td>43</td>
</tr>
<tr>
<td>2-6</td>
<td>Means and variances of the the twelve principal angles between $\mathbf{S}_Y$ and $\mathbf{S}_Y$.</td>
<td>43</td>
</tr>
<tr>
<td>2-7</td>
<td>Distribution of Votes on Kernel Type.</td>
<td>44</td>
</tr>
<tr>
<td>2-8</td>
<td>Cosines of the principal angles between the recovered kernel subspace and the ground-truth kernel subspace.</td>
<td>44</td>
</tr>
<tr>
<td>2-9</td>
<td>Singular values of the matrix $\mathbf{N}$.</td>
<td>45</td>
</tr>
<tr>
<td>3-1</td>
<td>Algorithms for deconstruction of classifiers.</td>
<td>50</td>
</tr>
<tr>
<td>3-2</td>
<td>Filters recovered by probing the decision boundary using positive and negative pairs</td>
<td>51</td>
</tr>
<tr>
<td>3-3</td>
<td>Multiple Haar-like features present in same retrieved filter.</td>
<td>53</td>
</tr>
<tr>
<td>3-4</td>
<td>Average running time for the 4400 features rejected by the classifier $\mathbf{C}$.</td>
<td>55</td>
</tr>
<tr>
<td>3-5</td>
<td>Average time taken by features rejected at a given level.</td>
<td>56</td>
</tr>
<tr>
<td>3-6</td>
<td>Dictionary elements ordered according to the average running time taken by negative features in the pairs.</td>
<td>58</td>
</tr>
<tr>
<td>3-7</td>
<td>Recovered soft-cascade.</td>
<td>60</td>
</tr>
<tr>
<td>3-8</td>
<td>Percentage of the OpenCV Haar cascade filters recovered at different correlation thresholds.</td>
<td>60</td>
</tr>
<tr>
<td>4-1</td>
<td>Schematic illustration of deconstructive learning and proposed algorithm.</td>
<td>66</td>
</tr>
<tr>
<td>4-2</td>
<td>PN-pairs obtained by bracketing process, during deconstructing OpenCV person detector.</td>
<td>74</td>
</tr>
<tr>
<td>4-3</td>
<td>PN-pairs obtained by bracketing process, during deconstructing Motorbike detector.</td>
<td>75</td>
</tr>
</tbody>
</table>
4-4  Classification errors exemplifying concept of geometric compatibility between features and deconstructors. ....................................................... 76

4-5  2D projections of the labeled PN-pairs in three different feature spaces, exemplifying concept of geometric compatibility between features and deconstructors. . . . 77

4-6  Deconstruction of classification-system with HOG features and Linear SVM as classifier. ................................................................. 78

4-7  Singular Values of the normal matrix. .............................................. 79

4-8  Identifying Kernel Type. ............................................................... 79
This dissertation introduces the novel notion of deconstructive learning and it proposes a practical computational framework for deconstructing a broad class of binary classifiers commonly used in computer vision applications. While the ultimate objective of most learning problems is the determination of classifiers from labeled training data, for deconstructive learning, the objects of study are the classifiers themselves. As its name suggests, the goal of deconstructive learning is to deconstruct a given classifier by determining and characterizing (as much as possible) the full extent of its capability, revealing all of its powers, subtleties and limitations. In particular, this work is motivated by the seemingly innocuous question that given an image-based binary classifier \( C \) as a black-box oracle, how much can we learn of its internal working by simply querying it? To formulate and answer this question computationally, I will first describe a general two-component design model employed by many current computer vision binary classifiers, a clear demonstration of the division of labor between practitioners in computer vision and machine learning. In this model, an input image is first transformed via a (nonlinear) feature transform to a feature space and a classifier is applied to the transformed feature to produce the classification output. The deconstruction of such a classifier therefore aims to identify the specific feature transform and the feature-space classifier used in the model.
Accordingly, the process of deconstructing a classifier C will be formulated as the identification problem for its two internal components from a finite list \( \mathcal{F} \) of candidate features and a finite list \( \mathcal{C} \) of candidate classifiers. As the main technical components of the deconstruction algorithm, I will introduce three novel ideas: feature identifiers, classifier deconstructors and the geometric feature-classifier compatibility. Specifically, feature identifiers are a set of image-based operations that can be applied to the input images, and the different degree of sensitivity and stability of the features in the feature list \( \mathcal{F} \) under these operations would allow us to exclude elements in \( \mathcal{F} \). The classifier deconstructors, on the other hand, are algorithms that can deconstruct classifiers in the candidate list \( \mathcal{C} \) using a (relatively) small number of features by recognizing certain geometric characteristics of the classifier’s decision boundary such as its parametric form. In this dissertation, I will present deconstruction algorithms for two popular families of classifiers in computer vision: cascades of linear classifiers and support vector machines. The interaction between elements in the feature and classifier lists during the deconstruction process is based on the notion of geometric feature-classifier compatibility that provides a principled and effective criterion for selecting the correct pair of feature and classifier as the output of the deconstruction process.

The bulk of this work will be devoted to realize the deconstruction framework in concrete and practical terms. In particular, I will present a variety of experimental results that validate the proposed deconstruction methods and demonstrate the viability of deconstructing computer vision algorithms. Interesting highlights of the experimental results include the deconstruction of the popular OpenCV pedestrian and face detectors and the demonstration of a kernel machine update/upgrade without using its source code. To the best of my knowledge, no similar results have been reported in the literature previously. Finally, in the last part of this dissertation, I will briefly speculate on the future application potential of deconstuctive learning.
CHAPTER 1
INTRODUCTION

Significant progress has been made in computer vision and machine learning in the past decade and increasingly, this advance has manifested itself in the form of artificial intelligence-based products (applications) that are starting to permeate our daily life, providing the coveted convenience and expediency that makes them becoming more indispensable each passing day. Such examples are readily available and they range from the more typical applications such as the new generation of hand-held cameras that use \textit{face detection} to re-focus and cameras used in smart surveillance systems that incorporate \textit{human detection} and \textit{tracking}, to the more esoteric display of futuristic vision such as the recent announcement by Facebook of its \textit{face verification} system \cite{46}, Google’s development of self-driving cars \cite{2} and the advanced robots that are replacing hoarding animals in difficult terrains under battlefield condition \cite{1}. It is transparently clear that with the current pace of technology development, our life in a not-so-distant future will be largely shared with and perhaps even dominated by these AI-based applications in such a way that the majority of the decision-making processes in our life will be delegated to machines and the algorithms powering them.

Decision makers and their decision-making processes, however insignificant in appearance (butterfly effect), make history. Precisely because of this, the study of decision makers (hitherto humans) and their decision-making process has occupied a central place in the intellectual heritage of men everywhere. Since time immemorial, philosophers have pondered and argued over the nature of men and their decisions, and historians have expounded and occasionally rationalized the decision makers and their processes that shaped the history. With the rise of scientific methods, modern social scientists such as psychologists and economists use ingenious experiments or construct sophisticated mathematical models to analyze and explain human behavior. Almost until recently, all these efforts have been indirect and external in their methodology in
that conclusions are drawn based on the responses of the decision makers to external events and the consequences of the decisions. Notable examples include the ink-blob test administered by the psychologists and the various market surveys and opinion polls conducted by the economists and political scientists\(^1\). Historically, the motivation for undertaking such intellectual pursuit are various, and they can be broadly classified as constructive or adversarial. In particular, out of their insatiable intellectual curiosity and perhaps also some boredom, ancient philosophers embarked on this profound intellectual journey that is still far from complete after several millennia. However, with the increasing understanding of ourselves and our decision-making process, we gain new insights and ideas on how to improve ourselves individually and collectively as a society. In an ideal world, this knowledge can be used for the betterment of mankind that optimizes the world for its harmony and productivity. In a less altruistic way, this knowledge also confers advantage in a competitive world. For example, individuals are often abstractly considered as autonomous agents that aim to maximize their rewards in many economic models, and in this constantly involving and competitive environment, understanding the opponent’s decision-making process provides a critical advantage, as articulated by Sun Tzu more than two millenniums ago\(^2\).

With the ascendency of machines and their ever-increasing roles as decision makers, I would argue that a parallel intellectual pursuit should commence, this time, for the machines. While humans have tried to understand humans in human terms in the past few millennia, the rise of machines entails the machines to understand themselves in their own terms. This is the gist of the new type of learning,

---

\(^1\) While powerful medical imaging devices such as functional MRI can now look into the brain and perhaps offer a physiological perspective on human decision-making process, many details are still elusive and obscure at this time.

\(^2\) It is said that if you know your enemies and know yourself, you will not be imperiled in a hundred battles (\textit{The Art of War, Chapter Three}).
Figure 1-1. Schematic illustration of deconstructive learning. Deconstructive Learning aims to understand the inner working of an AI-system without access to its source code. Here, we assume a given classification system is a black-box oracle and our objective is to figure out how it works by simply querying it.

The deconstructive learning, proposed and investigated in this dissertation. While the ultimate objective of most learning problems in machine learning and computer vision is the determination of classifiers from labeled training data, for deconstructive learning, the objects of study are the classifiers themselves. As its name suggests, the goal of deconstructive learning is to deconstruct a given classifier by determining and characterizing (as much as possible) the full extent of its capability, revealing all of its powers, subtleties and limitations. This is an uncharted territory that has not been studied previously in the literature, and in this dissertation, I will focus my attention on a (broad) class of object detectors (binary classifiers) in computer vision. My starting point is the seemingly-innocuous question that given an image-based binary classifier $C$ as a black-box oracle, how much can we learn of its internal working by simply querying it (Figure 1-1)? The main contribution of this dissertation is my answer to this question, in the form of a novel computational framework that aims to capture the spirit of deconstructive learning outlined above. Specifically, I will propose an algorithm
that, by simply querying a given detector with images, can identify its internal feature and classifier. In this novel context of understanding the inner working of a (smart) machine, several parallels with the example of understanding human decision making described above are immediately noticeable. First, our approach is also indirect and external in methodology in that our algorithm has access only the responses of the machine (detector), and its similarity with the often conjured image of a psychologist psychoanalyzing his patient by issuing probing questions is quite apparent. Second, there are several practical motivations for pursuing deconstructive learning and there are both constructive and adversarial components. On the practical side, I believe that deconstructive learning can provide a greater flexibility to the users of AI/machine learning products because it allows the users to determine the full extent of an AI/ML system, and therefore, create his/her own adaptation, modification or improvement of the given system for specific and specialized tasks. For example, once a kernel machine has been deconstructed, it can be subject to various kinds of improvement and upgrade in terms of its application scope, runtime efficiency and others. Imagine a kernel machine that was originally trained to recognize humans in images. By deconstructing the kernel machine and knowing its kernel type and possibly its support vectors, we can improve and upgrade it to a kernel machine that also recognizes other objects such as vehicles, scenes and other animals. On the other hand, in the context of adversarial learning, deconstructive learning allows a given system to be defeated and deficiency revealed. Finally, perhaps the most compelling reason for studying deconstructive learning is inscribed by the famous motto uttered by David Hilbert more than eighty years ago: *we must know and we will know!* Indeed, when presented with a black-box classifier (especially the one with great repute), I have found the problem of determining

---

3 Support Vector Machines (SVM) using kernel functions
the secret of its inner working by simply querying it with images both fascinating and challenging, a problem with its peculiar elegance and charm.

1.1 Deconstructive Learning And Computer Vision

As an example of deconstructive learning, imagine that we are presented with a classifier of great repute, say a pedestrian (human) detector. The detector, as a binary classifier of images, is presented only as a binary executable that takes images as inputs and outputs $\pm 1$ as its decision for each image. The classifier is laconic in the sense that except the predicted label $\pm 1$, it does not divulge any other information such as confidence level or classification margin associated with each decision. However, we are allowed to query the detector (classifier) using images, and the problem studied in this dissertation is to determine the inner working of the classifier using only image queries and the classifier’s laconic responses. For example, can we determine the type of features it uses? What kind of internal classifier does it deploy? Support vector machine (SVM) or cascade of linear classifiers or something else? If it uses SVM internally, what kind of kernel function does it use? How many support vectors are there and what are the support vectors?

Similar to many problems tackled in computer vision, deconstructive learning is an inverse problem; therefore, without an appropriate regularization, the problem is ill-posed and it is impossible to define desired solutions. In particular, since we are allowed to access only the laconic responses of the classifier, the scope seems almost unbounded. The appropriate notion of regularization in this context is to define a tractable domain on which solutions can be sought, and the main contribution of this dissertation is the proposal of a computational framework that would allow us to pose and answer the above questions as computationally tractable problems. Our proposal is based on a specific assumption on the classifier $C$ that its internal structure follows the common two-component design: a feature-transform component that transforms the input image into a feature and a machine-learning component that produces the output by applying
its internal classifier to the feature (Figure 1-2). Many existing binary classifiers in computer vision follow this type of design, a clear demonstration of the division of labor between practitioners in computer vision and machine learning. For example, most of the well-known detectors such as face and pedestrian detectors (e.g., [8, 34, 50]) conform to this particular design, with other lesser-known but equally-important examples in scene classification, object recognition and others (e.g., [10, 27]) adopting the same design. By clearly delineating the vision and learning components, we can formulate a computational framework for deconstructing \( C \) as the identification problem for its two internal components from a finite collection of potential candidates.

More precisely, for a given vision classifier \( C \) (e.g., an object detector), the deconstruction process requires a list of features (and their associated transforms) \( \mathcal{F} \) and a list of (machine learning) classifiers \( \mathcal{C} \). Based on these two lists, the algorithm would either identify the components of \( C \) among the elements in \( \mathcal{F} \) and \( \mathcal{C} \) or return a void to indicate failure in identification. Computationally, the lists define the problem domain, and they constitute the required minimal prior knowledge of \( C \). In practice, the general outline of the feature used in a particular vision algorithm is often known and can be ascertained through various sources such as publications. However, important design parameters such as smoothing values, cell/block/bin sizes etc., are often not available and these parameters can be determined by searching over an expected range of values that made up the elements in \( \mathcal{F} \). Similarly, the type of classifier used can often be narrowed down to a small number of choices (e.g., an SVM or a cascade of linear classifiers). Within this context, we introduce three novel notions, feature identifiers, classifier deconstructors and geometric feature-classifier compatibility, as the main technical components of the deconstruction process. Specifically, feature identifiers are a set of image-based operations such as image rotations and scalings that can be applied to the input images, and the different degree of sensitivity and stability of the features in \( \mathcal{F} \) under these operations would allow us to exclude elements in \( \mathcal{F} \), making
the process more efficient. For example, suppose $\mathcal{F}$ contains both Scale-invariant feature transform (SIFT) and Histogram of Oriented Gradients (HOG)-based features. Since SIFT is in principle rotationally invariant, SIFT-based features are more stable under image rotations than HOG-based features; and therefore, if $\mathbf{C}$ uses a SIFT-based feature internally, it outputs would be expected to be more stable under image rotations. Therefore, by querying $\mathbf{C}$ with rotated images and comparing the results with un-rotated images, we can exclude features in $\mathcal{F}$ that are rotationally sensitive. The classifier deconstructors, on the other hand, are algorithms that can deconstruct classifiers in $\mathcal{C}$ using a (relatively) small number of features by recognizing certain geometric characteristics of the classifier’s decision boundary (e.g., its parametric form). For example, an SVM deconstructor algorithm is able to (given sufficiently many features) determine the number of support vectors and the type of kernel used by a kernel machine by recognizing certain geometric characteristics of its decision boundary. The interaction between elements in $\mathcal{F}$ and $\mathcal{C}$ during the deconstruction process is based on the notion of geometric feature-classifier compatibility: for a pair $(f, c)$ of feature $f$ and classifier $c$, they are compatible if given sufficiently many features defined by $f$, the deconstructor algorithm associated to $c$ can correctly recognize its decision boundary. More specifically, given a vision classifier $\mathbf{C}$ internally represented by a pair $(f, c)$ of feature $f$ and classifier $c$, we can query $\mathbf{C}$ using a set of images $I_1, ..., I_n$, and using the feature (and it associated transform) $f$, we can transform the images into features in the feature space specified by $f$. The deconstructor algorithm associated with $c$ then determines the classifier based on these features. However, for an incorrect hypothetical pair $(\overline{f}, c)$, the difference between the transformed features specified by $\overline{f}$ and $f$ are generally non-linear, and this non-linearity changes the geometric characteristics of the decision boundary in the feature space specified by $\overline{f}$, rendering the deconstructor algorithm $c$ unable to identify the decision boundary (Figure 1-2).
The abstract framework outlined above provides a practical and useful modularization of the deconstruction process so that the individual elements such as the formation of feature and classifier lists, feature identifiers and classifier deconstructors can be subject to independent development and study. In the following chapters, we will realize the abstract framework in concrete terms. Specifically, we introduce two deconstructor algorithms for support vector machine (SVM) and for the cascade of linear classifiers. The former is a popular family of classifiers widely used in vision applications and the latter is often deployed in object detectors, with the face detector of Viola-Jones as perhaps the most well-known example [50].

**Deconstructing Kernel Machines:** An important technical contribution of this work is an algorithm for deconstructing support vector machines. More specifically, given a kernel machine \( C \), we ask the following four questions:

1. Can the kernel function be determined?
2. Can the number, \( m \), of support vectors be determined?
3. Can the (kernel) subspace spanned by the support vectors of \( C \) be determined?
4. Can the support vectors themselves be determined?

My investigation has shown that the first three questions can be answered affirmatively. I will assume that the kernel machine uses one of the five types of kernel functions: polynomial kernels (linear, quadratic, and cubic), hyperbolic tangent kernel and RBF kernel. Let \( \mathbb{R}^l \) denote the feature space in which \( C \) is defined and I will also assume that \( m << l \) and support vectors are linearly independent. The latter assumption implies that the dimension of the kernel subspace, the subspace spanned by the support vectors, is the same as the number of support vectors. Once the kernel subspace is determined, the complexity of characterizing the decision boundary is significantly reduced, and for polynomial kernels, using the recent results on tensor decomposition (e.g., [12]), we are able to compute a set of quasi-support vectors can define the decision boundary exactly. The proposed deconstructive learning algorithm is straightforward and is based
on computing rudimentary differential geometric features of the decision boundary, its normal vectors.

**Deconstructing Cascades of Linear Classifiers:** Recall that a cascade of linear classifiers is structurally a degenerate decision tree (e.g. [50]) where each node is a linear classifier. Unlike the case of kernel machines above, the decision boundary of a linear cascade is generally not smooth (unless the cascade has only one linear classifier) but piecewise linear; therefore, its deconstruction requires a larger number of randomly generated $PN$-pairs to capture locally linear areas. More specifically, the deconstructor algorithm is able to

- Determine the number of levels (depth of the tree) in the cascade.
- Show that for each linear classifier in the cascade, $h(x) = n^T x + b$, the normal vector $n$ and bias $b$ can be determined up to a multiplicative constant.

Compared to the SVM deconstructor above, the deconstructor for cascade of linear classifiers is considerably simpler since only linear classifiers are involved. In particular, the continuity assumption on the feature domain is necessarily for recovering the normal vector $n$ using bracketing. However, the method we have developed for deconstructing linear cascades is more general and it works also for discrete domains (integral lattices and integral images) as in the original setting [50]. For practical evaluation of our method we deconstructed SoftCascade (based on [11]) trained on the Face images.

Deconstructing implementation of Viola-Jones Face Detector [49], present in OpenCV library, is more challenging. In [49], each linear classifier is constructed by AdaBoosting [25] haar-like features. Evaluation of the Haar-like features, used by the Viola-Jones, on any given image could be represented by the dot product between image vector and vector $h$ such that elements of $h$ have values equal to weights of positive and negative regions of the given haar feature; thus they behave like linear classifier. For complete deconstruction we need not only find the linear classifier on each node of decision tree but also haar-like features used to construct those linear classifiers. In both the original
Boosted Cascade by Viola-Jones [49] and SoftCascade [11] ordering and placement of the classifier is very important; using the computational time it required to reject negative samples we were able to recover number of stages in cascades, location of linear classifier and what features make that classifiers.

![Figure 1-2. Proposed deconstructing algorithm for classifier with two component design.](image)

**Left:** Schematic illustration of deconstructive learning. **Center:** Two-component design of a classifier: a feature-transform component provided by computer vision followed by a feature-classification component furnished by machine learning. **Right:** Schematic illustration of the proposed deconstruction algorithm. Internally, the algorithm searches over a set of candidate feature spaces and probes the spaces for decision boundaries. Only in the correct feature space the parametric form of the decision boundary would be recognized by the deconstructor algorithm.

### 1.2 Related Work

To the best of our knowledge, there is no previous work on deconstructive learning (DL) as described above. However, [36] studied the problem of deconstructing linear classifiers in a context that is slightly different from ours. This corresponds to linear kernel machines and consequently, their scope is considerably narrower than ours as (single) linear classifiers are relatively straightforward to deconstruct. Active learning (e.g., [20] [3] [4]) shares certain similarities with deconstructive learning (DL) in that it also has a notion of querying a source. However, the main distinction is their specificities and outlooks: for active learning, it is general and relative while for DL, it is specific and absolute. More precisely, for active learning, the goal is to determine a classifier from a concept class with some prescribed (PAC-like) learning error bound using samples generated from the underlying joint distribution of feature and label. In this model, the learning target is the joint distribution and the optimal learned classifier is relative to the given concept class. On the other hand, in DL, the learning target is a given classifier...
and the classifier defines an absolute partition of the feature space into two disjoint regions of positive and negative features. Furthermore, the classifier is assumed to belong to a specific concept class (e.g., kernel machines with known types of kernel function) such that the goal of DL is to identify the classifier within the concept class using the geometric features of the decision boundary. In this absolute setting, geometry replaces probability as the joint feature-label distribution gives way to the geometric notion of decision boundary as the main target of learning. In particular, bracketing is a fundamentally geometric notion that is generally incompatible with a probabilistic approach, and with it, DL possesses a much more efficient and precise tool for exploring the spatial partition of the feature space, and consequently, it allows for a direct and geometric approach without requiring much probability.

1.3 Road Map

In chapter 2, we will present the deconstructor algorithm for support vector machines. In this chapter, we will also introduce most of the technical concepts used in the subsequent deconstruction processes: bracketing, PN-pairs, finding normals and their subspace, etc. In chapter 3, we will present the deconstructor algorithm for cascades of linear classifiers, and in this section, we will present the details experimental results on the deconstruction of the OpenCV face detector. Using these deconstructor algorithms for two different families of classifiers, we will present the detailed method and algorithm for deconstructing classification systems employing the two-component design described above. In particular, technical constructs such as feature and classifier lists \( \mathcal{F}, \mathcal{C} \) and notions such as geometric feature-classifier compatibility will be discussed in detail with multiple experimental results that successfully deconstruct linear and nonlinear classification systems using common computer vision features such as HOG. Finally, in the last chapter of this dissertation, I will briefly speculate on the future application potential of deconstructive learning.
1.4 Etymology

The word *Deconstruction* in Deconstructive Learning, takes part of its meaning from how it has been used in philosophical and literary analysis. Mainly derived from Jacques Derrida's 1967 work *of Grammatology* [22], *Deconstruction* method tends to look at a philosophical argument or a literary piece by analyzing fundamental conceptual distinctions presented in it, with the aim of showing contradictions and/or possibility of multiple meanings. Further clarification on the meaning of deconstruction has been given by Barbara Johnson [18] who described it as *much closer to the original meaning of the word 'analysis' itself, which etymologically means "to undo" – a virtual synonym for "to de-construct"*. Instead of finding multiple versions by assigning different meanings to parts of an argument, in deconstructive learning, we are trying to find relationship between parts (e.g., support vectors) that can explain the given classifier. In particular, deconstructive learning provides a framework for analyzing classification systems and exposing their functionality and mechanics. For this, deconstructive learning dose a form of exploration in the domain specified by the feature and classifier lists in order to properly explain the given classification system, an aspect shared by deconstructive reading where multiple versions are generated by assigning different meanings to parts of arguments. However, this is where the similarities between intention and objective of our work and *Deconstruction* in literature analysis ends. Our work deals with more concrete settings and explores classification systems with tools that generate measurable and verifiable results.
While the ultimate objective of most learning problems is the determination of classifiers from labeled training data, for deconstructive learning, the objects of study are the classifiers themselves. As its name suggests, the goal of deconstructive learning is to deconstruct a given classifier \( \mathbf{C} \) by determining and characterizing (as much as possible) the full extent of its capability, revealing all of its powers, subtleties and limitations. Since classifiers in machine learning come in a variety of forms, deconstructive learning correspondingly can be formulated and posed in many different ways. This chapter focuses on a family of binary classifiers based on support vector machines [48], and deconstructive learning will be formulated and studied using geometric and algebraic methods without recourse to probability and statistics.

Given an SVM (kernel)-based binary classifier \( \mathbf{C} \) as a black-box oracle, we want to figure out how much can we learn of its internal working by querying it? Specifically, we assume the feature space \( \mathbb{R}^d \) is known and the kernel machine has \( m \) support vectors such that \( d > m \) (or \( d >> m \)), and in addition, the classifier \( \mathbf{C} \) is laconic in the sense that for a feature vector, it only provides a predicted label (\( \pm 1 \)) without divulging other information such as margin or confidence level. We formulate the the problem of understanding the inner working of \( \mathbf{C} \) as characterizing the decision boundary of the classifier, and we introduce the simple notion of bracketing to sample points on the decision boundary within a prescribed accuracy. For the five most common types of kernel function, linear, quadratic and cubic polynomial kernels, hyperbolic tangent kernel and Gaussian kernel, we show that with \( O(dm) \) number of queries, the type of kernel function and the (kernel) subspace spanned by the support vectors can be determined. In particular, for polynomial kernels, additional \( O(m^3) \) queries are sufficient to reconstruct the entire decision boundary, providing a set of quasi-support vectors that can be used to efficiently evaluate the deconstructed classifier. We speculate briefly
on the future application potential of deconstructing kernel machines and we present experimental results validating the proposed method.

2.1 Problem Statement

Let us assume that the (continuous) feature space in which the classifier $C$ is defined is assumed to be a $d$-dimensional vector space $\mathbb{R}^d$, and the classifier $C$ is given as a binary-valued function $C : \mathbb{R}^d \rightarrow \{-1, +1\}$, indicating the class assignment of each feature $x \in \mathbb{R}^d$. As a kernel machine, $C$ is specified by a set of $m$ support vectors $y_1, \ldots, y_m \in \mathbb{R}^d$ and a kernel function $K(x, y)$ such that the decision function $\Psi(x)$ is given as the sum

$$\Psi(x) = \omega_1 K(x, y_1) + \cdots + \omega_m K(x, y_m),$$

(2–1)

where $\omega_1, \ldots, \omega_m$ are the weights. With the bias $b$,

$$C(x) = \begin{cases} +1 & \text{if } \Psi(x) \leq b, \\ -1 & \text{if } \Psi(x) > b. \end{cases}$$

(2–2)

The classifier $C$ is also assumed to be laconic in the sense that except for the binary label, it does not divulge any other potentially useful information such as margin or confidence level. With these assumptions, we formulate the problem of deconstructing $C$ through the following list of four questions (ordered in increasing difficulty)

1. Can the kernel function $K(x, y)$ be determined?
2. Can the subspace $S_Y$ spanned by the support vectors be determined?
3. Can the number $m$ of support vectors be determined?
4. Can the support vectors themselves be determined?

Without loss of generality, we will henceforth assume $b = 1$. Therefore, if the support vectors and the kernel function are known, the weights $\omega_i$ can be determined completely given enough points $x$ on the decision boundary

$$\Sigma = \{ x \parallel x \in \mathbb{R}^d, \ \Psi(x) = b \}.$$  

(2–3)
That is, a kernel machine \( \mathbf{C} \) can be completely deconstructed if its support vectors and kernel function are known.

The four questions above are impossible to answer without further quantification on the type of kernel function and the number of support vectors. In this chapter, we assume 1) the unknown kernel function belongs to one of the following five types: polynomial kernels of degree one, two and three (linear, quadratic and cubic kernels), hyperbolic tangent kernel and RBF kernel, and 2) the number of support vectors is less than the feature dimension, \( d > m \) (or \( d >> m \)) and they are linearly independent. For most applications of kernel machines, these two assumptions are not particularly restrictive since the five types of kernel are arguably among the most popular ones. Furthermore, as the feature dimensions are often very high and the support vectors are often thought to be a small number of the original training features that are critical for the given classification problem, it is generally observed that \( d > m \). With these two assumptions, the method proposed in this chapter shows that the first three questions can be answered affirmatively. While the last question cannot be answered for transcendental kernels, we show that using recent results on tensor decomposition (e.g., [12]), a set of quasi-support vectors can be computed for a polynomial kernel that recover the decision boundary exactly.

Given the laconic nature of \( \mathbf{C} \), it seems that the only effective approach is to probe the feature space by locating points on the decision boundary \( \Sigma \) and to answer the above questions using local geometric features computed from these sampled points. More precisely, the proposed algorithm takes the classifier \( \mathbf{C} \) and a small number of positive features in \( \mathbb{R}^d \) as the only inputs. Starting with these small number of positive features, the algorithm proceeds to explore the feature space by generating new features and utilizing these new features and their class labels provided by \( \mathbf{C} \) to produce points on the decision boundary. The challenge is therefore to use only a comparably small number of sampled features (i.e., queries to \( \mathbf{C} \)) to learn enough about \( \Sigma \) in order to
answer the questions, and our main contribution is an algorithm that has complexity (to be defined later) linear in the dimension $d$ of the ambient space.

Sampling points on $\Sigma$ can be accomplished easily using bracketing, the same idea used in finding the root of a function (e.g., [30]). Given a pair of positive and negative features (PN-pair), the intersection of $\Sigma$ and the line segment joining the two features cannot be empty, and bracketing allows at least one such point on $\Sigma$ to be determined up to any prescribed precision. Using bracketing as the main tool, the first two questions can be answered by exploring the geometry of $\Sigma$ in two different ways. First, the decision boundary $\Sigma$ is given as the implicit surface of the multi-variate function, $\Psi(x) = b$. With high-dimensional features, it is difficult to work directly with $\Sigma$ or $\Psi(x)$; instead, the idea is to examine the intersection of $\Sigma$ with a two-dimensional subspace formed by a PN-pair. The locus of such intersection is in fact determined by the kernel function, and by computing such intersection, we can ascertain the kernel function on this two-dimensional subspace. For the second question, the answer is to be found in the normal vectors of the hypersurface $\Sigma$. Using bracketing, the normal vector at a given point on $\Sigma$ can be determined, again in principle, up to prescribed precision. From the parametric forms of the kernel functions, it readily follows that the normal vectors of $\Sigma$ are generally quite well-behaved in the sense that they either belong to the kernel subspace $S_\gamma$ spanned by the support vectors or they are affine-translations of the kernel subspace $S_\gamma$. For the former, a quick application of singular value decomposition immediately yields the kernel subspace $S_\gamma$, and for the latter, the kernel subspace $S_\gamma$ can be computed via a rank-minimization problem that can be solved (in many cases) as a convex optimization problem with the nuclear norm. If we define the complexity of the algorithm as the required number of sampled points in the feature space, it will be shown that the complexity of the proposed method is essentially $O(dm)$ as it requires $O(m)$ normal vectors to determine the $m$-dimensional kernel subspace and $O(d)$ points to determine the normal vector at a point in $\mathbb{R}^d$. The constant depends on the number of
steps used for bracketing, and if the features are assumed to be drawn from a bounded subset in $\mathbb{R}^d$, this constant is then independent of the dimension $d$.

We note that for a polynomial kernel of degree $D$, its decision function $\Psi(x)$ is a degree-$D$ polynomial with $d$ variables. Therefore, in principle, $C$ can be deconstructed by fitting a polynomial of degree $D$ in $\mathbb{R}^d$ given enough sampled points on $\Sigma$. However, this solution is in general not useful because it does not extend readily to transcendental kernels. Furthermore, the number of required points is in the order of $d^D$, and correspondingly, a direct polynomial fitting would require the inversion of a large dense (Vandermonde) matrix that is in the order of $d^D \times d^D$. With a moderate dimension of $d = 100$ and $D = 3$, this would require $10^6$ points and the inversion of a $10^6 \times 10^6$ dense matrix. Our method, on the other hand, encompasses both the transcendental and polynomial kernels and at the same time, it avoids the direct polynomial fitting in $\mathbb{R}^d$ and has the overall complexity that is linear in $d$, making it a truly practical algorithm.

2.2 Preliminaries

Let $\mathbb{R}^d$ denote the feature space equipped with its standard Euclidean inner product, and for $x, y \in \mathbb{R}^d$, $\|x - y\|^2 = (x - y)^\top(x - y)$. For the kernel machines studied in this chapter, we assume their kernel functions are of the following five types:

- **Linear Kernel**
  $$K(x, y) = x^\top y,$$

- **Quadratic Kernel**
  $$K(x, y) = (x^\top y + 1)^2,$$

- **Cubic Kernel**
  $$K(x, y) = (x^\top y + 1)^3,$$

- **Hyperbolic Tangent Kernel**
  $$K(x, y) = \tanh(\alpha x^\top y + \beta),$$

- **Gaussian Kernel**
  $$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right),$$

for some constants $\alpha, \beta, \sigma$. We will further refer to the three polynomial kernels and the hyperbolic tangent kernel as the Type-A kernels and the Gaussian kernel as the Type-B kernel. This particular taxonomy is based on their forms that can be generically written
as
\[ K(x, y) = f(x^T y), \quad K(x, y) = g(\|x - y\|^2), \]
for some smooth univariate function \( f, g, \mathbb{R} \to \mathbb{R} \).

Given the forms of the kernel function, an important consequence is that the decision boundary \( \Sigma \) is determined in large part by its intersection with the kernel subspace \( S_y \) spanned by the support vectors. More precisely, for \( x \in \mathbb{R}^d \), let \( \bar{x} \) denote the projection of \( x \) on \( S_y \):
\[
\bar{x} = \arg\min_{y \in S_y} \|x - y\|^2.
\]
For Type-A kernel \( K(x, y) = f(x^T y) \), we have \( K(x, y_i) = K(\bar{x}, y_i) \) for every support vector \( y_i \). In particular, \( \bar{x} \) is on the decision boundary if and only \( x \) is. For Type-B kernels, we have (using Pythagorean theorem with \( q^2 = \|x\|^2 - \|\bar{x}\|^2 \))
\[
K(x, y_i) = g(\|x - y_i\|^2) = g(\|x - y_i\|^2 + q^2),
\]
and with the Gaussian kernel \( g \), we have \( g \). It then follows that for any \( x \in \Sigma \), its projection \( \bar{x} \) on \( S_y \) must satisfy
\[
\Psi(\bar{x}) = e^{\frac{q^2}{2\sigma^2}} \Psi(x) = e^{\frac{q^2}{2\sigma^2}} b.
\]
In other words, the decision boundary \( \Sigma \) is essentially determined by the level-sets of \( \Psi(x) \) on the kernel subspace \( S_y \).

Since decision boundary \( \Sigma \) is given as the implicit surface \( \Psi(x) = b \), a normal vector \( n(x) \) at a point \( x \in \Sigma \) can be given as the gradient of \( \Psi(x) \):
\[
n(x) = \nabla \Psi(x) = \sum_{i=1}^{m} \omega_i \nabla_x K(x, y_i).
\] (2-4)
For the two types of kernels we are interested in, their gradient vectors assume the following forms:

\[
\nabla_x K(x, y) = f'(x^\top y) y, \\
\nabla_x K(x, y) = 2g'\left(\|x - y\|^2\right) (x - y). 
\]

(2–5) (2–6)

Using the above formulas, it is clear that for Type-A kernels, the normal vector \( n(x) \) depends on \( x \) only through the coefficients in the linear combination of the support vectors, while for Type-B kernels, \( x \) actually contributes to the vectorial component of \( n(x) \). It will follow that an important element in the deconstruction method introduced below is to exploit this difference in how the normal vectors are computed for the two types of kernels. For example, for a polynomial kernel of degree \( D \), a normal vector at a point \( x \in \Sigma \) is

\[
n(x) = \sum_{i=1}^{m} D \omega_i (x^\top y_i + 1)^{D-1} y_i. 
\]

(2–7)

As a special case, for linear kernel \( D = 1 \), we have

\[
n(x) = \sum_{i=1}^{m} \omega_i y_i, 
\]

that is independent of \( x \). For the Gaussian kernel, we have

\[
n(x) = \sum_{i=1}^{m} -\frac{\omega_i}{\sigma^2} \exp\left(-\frac{\|x - y_i\|^2}{2\sigma^2}\right) (x - y_i). 
\]

(2–8)

### 2.3 Deconstruction Method

The deconstruction algorithm requires two inputs: 1) an SVM-based binary classifier \( \Psi(x) \) that uses one of the five kernel types indicated above, and 2) a small number of positive and negative features. The algorithm uses the small number of input features to generate other pairs of positive and negative features. For a pair \( p, n \) of positive and negative features (a PN-pair), we can be certain that the line segment joining \( p, n \) must
intersect the decision boundary in at least one point. Using bracketing, we can locate
one such point \( x \) on the decision boundary within any given accuracy, i.e., we can use
bracketing to obtain a PN-pair \( p, n \) such that \( \|p - n\| < \epsilon \) for some prescribed \( \epsilon > 0 \).
With a small enough \( \epsilon \), the midpoint between \( p, n \) can be considered approximately as
a sampled point \( x \) on \( \Sigma \) and its normal vector can then be estimated. The algorithm
proceeds to sample a collection of points and their normals on the decision boundary \( \Sigma \),
and using this information, the algorithm first computes the kernel subspace \( S_Y \) and this
step separates the Type-A kernels from the Type-B kernels (Gaussian kernel). The four
Type-A kernels can further be identified by computing the intersection of \( \Sigma \) with a few
randomly chosen two-dimensional subspaces. These two steps provide the affirmative
answers to the first three questions in the introduction. For polynomial kernels, we can
determine a set of quasi-support vectors that provide the exact recovery of the decision
boundary \( \Sigma \). However, no such results for the two transcendental kernels are known at
present and we leave its resolution to future research.

2.3.1 Bracketing

Given a PN-pair, \( p, n \), the decision boundary must intersect the line segment joining
the two features. Therefore, we can use bracketing, the well-known root-finding method
(e.g., [30]), to locate the point on \( \Sigma \). Note that bracketing does not require the function
value, only its sign. This is compatible with our classifier \( C \) that only gives binary values
\( \pm 1 \). In particular, if we bisect the interval in each step of bracketing, the length of the
interval is halved at each iteration, and for a given precision requirement \( \epsilon > 0 \), the
number of steps required to reach it is in the order of \( \log \epsilon \). If we further assume that
the features are generated from a bounded subset of \( \mathbb{R}^d \) (which is often the case) with
diameter less than \( K \), then for any PN-pair \( p, n \), bracketing terminates after at most

\[
\log_2 K - \log_2 \epsilon + 1
\]  

(2-9)
steps, a number that is independent of the ambient dimension $d$. Figure 2-1 demonstrates bracketing procedure. Figure on bottom-right indicates resultant information we get when we perform bracketing many time over many random pairs of positive and negative samples, we have pegged down the decision boundary. It also shows how the complexity of decision boundary will effect how many PN-pairs will be required. If decision boundary is linear we can do with very few samples (atleast $d$), quadratic or cubic we will require more; however linearly-piecewise decision boundaries are most complex and will require large number of PN-pairs.

Figure 2-1. Bracket process and the PN-pairs. Bottom-Right: Bracketing process is repeated using different positive-negative samples to obtain multiple PN-pairs These PN-pairs allow us to obtain geometrical information of the decision boundary of the classifier $C$. 
2.3.2 Estimating Normal Vectors

Given the pair \( p, n \), let \( \mathbf{p}, \mathbf{n} \) denote the two points near \( \Sigma \) after the bracketing step and \( x \) denote their midpoint. To estimate the normal vector at \( x \), we use the fact that the (unknown) kernel function is assumed to be smooth and \( \Sigma \) is a level-surface of the decision function \( \Psi(x) \) that is a linear combination of smooth functions. Consequently, a randomly chosen point on \( \Sigma \) is almost surely non-singular \([32]\) in that it has a small neighborhood in \( \Sigma \) that can be well-approximated using a linear hyperplane (its tangent space) in \( \mathbb{R}^d \). Accordingly, we will estimate the normal vector at \( x \) by linearly fitting a set of points on \( \Sigma \) that belong to a small neighborhood of \( x \) (Figure 2.2). More specifically, we chose a small \( \delta > \epsilon > 0 \) and generate PN-pairs on the sphere centered at \( x \) with radius \( \delta \). Using bracketing and the convexity of the ball enclosed by the sphere, we obtain PN-pairs that are near \( \Sigma \) and no more than \( \delta \) away from \( x \).

Taking the midpoint of these PN-pairs, we obtain a set of randomly generated \( O(d') \) points on \( \Sigma \). We linearly fit a \((d - 1)\)-dimensional hyperplane to these points and the normal vector is then computed as the eigenvector associated to the smallest eigenvalues of the normalized covariance matrix. The result can be further sharpened by repeating the step over multiple \( \delta \) and taking the (spherical) average of the unit normal vectors. However, in practice, we have observed that good normal estimates can be consistently obtained using one small \( \delta \approx 10^{-3} \) (with \( \epsilon = 10^{-6} \)) and \( 2d \) sampled points\(^1\).

2.3.3 Determine Kernel Subspace \( S_Y \)

To determine the kernel subspace \( S_Y \), we will use the formulas for the normal vectors given in Equations 2–5 and 2–6. Assume that we have sampled \( s > m \) points on \( \Sigma \) and

---

\( ^1 \) We note that for sufficiently small \( \delta \), the angular error of the estimated normal is approximately in the order of \( \tan^{-1}(\frac{\epsilon}{s \delta}) \).
Figure 2-2. Calculating normals using PN-Pairs. (A) PN-pair due to bracketing, (B) sampling points around PN-pair, (C) using classifier system \( C \) to label the points, (D) recovering normal on the decision boundary.

Let \( N \) denote the following two matrices

\[
    X = [x_1 \ x_2 \ ... \ x_n], \quad N = [n_1 \ n_2 \ ... \ n_n]
\]

(2–10)

that horizontally stack together the points \( x_i \) and their normal vectors \( n_i \), respectively. If all \( n_i \) are correctly recovered (without noise), we have the following:

- For Type-A kernels, \( n_i \in S_Y \), i.e., \( n_i \) is a linear combination of the support vectors.
- For Type-B kernels, \( n_i \in \gamma x + S_Y \), for some \( \gamma_i \in \mathbb{R} \), i.e., \( n_i - \gamma x \in S_Y \).

Note that \( \gamma_i \) depends on \( x_i \) and the two statements can be readily checked using Equations 2–4 - 2–6. Therefore, the kernel subspace \( S_Y \) can be recovered, for Type-A
Figure 2-3. Intersections of $\Sigma$ and two-dimensional affine subspaces. An SVM using the cubic kernel is trained on MINST dataset. **Top Row:** Midpoints of PN-pairs near the decision boundary $\Sigma$ after bracketing. **Bottom Row:** Sampled polynomial curves given the intersections of the decision boundary with two-dimensional affine subspaces containing the images above.

kernels, using Singular Value Decomposition (SVD). Specifically, let $N = UDV^\top$ denote the singular value decomposition of $N$. There are precisely $m$ nonzero singular values and $S_Y$ is spanned by the first $m$ columns of $U$. For Type-B, a slight complication arises because we must determine $s$ constants $\gamma_1, \cdots, \gamma_s$ such that the span of the following matrix is $S_Y$:

$$N - X\Gamma \equiv [n_1 \ n_2 \ \cdots \ n_s] - [\gamma_1 x_1 \ \gamma_2 x_2 \ \cdots \ \gamma_s x_s],$$

(2–11)

where $\Gamma$ is a diagonal matrix with $\gamma_i$ as its entries. Note that in general, $N, X$ are of full-rank $\min(d, s)$, and we are trying to find a set of $\gamma_i$ such that the above matrix has rank $m < s$. However, for a generically chosen set of $x_1, \cdots, x_s$, the rank of $N - X\Gamma$ is at least $m$ because the support vectors are linearly independent. Therefore, $\gamma_i$ can be determined via the following rank-minimization problem

$$\arg \min_{\gamma} \text{Rank}( [n_1 \ n_2 \ \cdots \ n_s] - [\gamma_1 x_1 \ \gamma_2 x_2 \ \cdots \ \gamma_s x_s]) .$$

(2–12)

As is well-known, a convex relaxation of the above problem uses the nuclear norm $\| \cdot \|_*$ (sum of singular values) as the surrogate

$$\arg \min_{\text{diagonal } \Gamma} \| N - X\Gamma \|_* ,$$

(2–13)
and there are efficient algorithms for solving this type of convex optimization problem [41].

We note that for Type-A kernels, the rank is minimized at $\gamma_1 = \cdots = \gamma_s = 0$. In both cases, the span of $N - X\Gamma$ gives the kernel subspace $S_Y$. As the support vectors are assumed to be linearly independent, the dimension of $S_Y$ then gives the number of support vectors. For noisy recovery, the above method requires the standard modification that uses the significant gap between singular values as the indicator. For Type-A kernels, this is applied to the SVD decomposition of $N$ directly, and for Type-B kernels, this is applied to the SVD decomposition of $N - X\Gamma$ with $\Gamma$ determined by the nuclear norm minimization.

### 2.3.4 Determine Kernel Type

For determining the four Type-A kernels, we will examine the locus of the intersection of the decision boundary with a two-dimensional affine subspace containing a point close to the decision boundary. More specifically, let $x_+, x_-\in \mathbb{R}^d$ denote a PN-pair that is sufficiently close to the decision boundary $\Sigma$. We can randomly generate a two-dimensional subspace containing $x_+, x_-$ by, for example, taking the subspace $A$ formed by $x_+, x_-\in \mathbb{R}^d$ and the origin. For a generic two-dimensional subspace $A$, its intersection with $\Sigma$ is a one-dimensional curve, and the parametric form of this curve is determined by the (yet unknown) kernel function. See Figure 2-3.

Take a polynomial kernel of degree $D$ as an example. By its construction, the intersection of the decision boundary and the affine subspace $A$ is nonempty, and the locus of the intersection formed a curve in $A$ that satisfies a polynomial equation of degree $D$. This can be easily seem as follows: take $x_+$ as the origin on $A$ and choose an (arbitrary) orthonormal vectors $U_1, U_2 \in \mathbb{R}^d$ such that the triplet $x_+, U_1, U_2$ identifies $A$ with $\mathbb{R}^2$. Therefore, any point $p \in A$ can be uniquely identified with a two-dimensional vector $p = [p_1, p_2] \in \mathbb{R}^2$ as

$$p = x_+ + p_1 U_1 + p_2 U_2.$$
If \( \mathbf{p} \in \mathbf{A} \) is a point in the intersection of \( \mathbf{A} \) with the decision boundary \( \Psi(\mathbf{p}) = \mathbf{b} \), we have

\[
\sum_{i=1}^{m} w_i ((\mathbf{x}_i \mathbf{Y}_i^\top + \mathbf{p}_1 \mathbf{U}_1 \mathbf{Y}_i^\top + \mathbf{p}_2 \mathbf{U}_2 \mathbf{Y}_i^\top)^D = \mathbf{b},
\]

which is a polynomial of degree \( D \) in the two variables \( \mathbf{p}_1, \mathbf{p}_2 \). Therefore, to ascertain the degree of the polynomial kernel, we can (assuming \( D < 4 \))

- Sample at least nine points on the intersection of the decision boundary and \( \mathbf{A} \).
- Fit a bivariate polynomial of degree \( D \) to the points. If the fitting error is sufficiently small, this gives an indication that the polynomial kernel is indeed of degree \( D \).

We note that up to a multiplicative constant, a bivariate cubic polynomial in \( \mathbb{R}^2 \) has nine coefficients and this gives the minimum number of points required to fit a cubic polynomial. In addition, since the degree of the polynomial is invariant under any linear transform, this shows that the choice of the two basis vectors is immaterial. The advantage of the reduction from \( \mathbb{R}^d \) to \( \mathbb{R}^2 \) is considerable as it implies that the complexity of this step is essentially independent of the ambient dimension \( d \). For a transcendental kernel such as the hyperbolic tangent kernel, the locus of the intersection is generally not a polynomial curve and this can be detected by the curve-fitting error. Although, in principle, one affine subspace \( \mathbf{A} \) is sufficient to distinguishing the four Type-A kernels (as shown by the above equation), in practice, due to various issues such as possible degeneracy of the polynomial curve and the curve fitting error, we randomly sample several affine subspaces and use a majority voting scheme to determine the kernel type.

### 2.3.5 Complexity Analysis and Exact Recovery of \( \Sigma \)

The steps outlined above essentially aim to ascertain the parametric form of the decision boundary \( \Sigma \) using a (relatively) small number of sampled points on \( \Sigma \). We note that the bracketing error in general can be explicitly controlled, and there are only two steps above that incur uncertainty: the normal estimate and the nuclear norm relaxation of the rank minimization problem. Our approach of using the local linear approximation to estimate the normal vector at a point is the standard one common in computational
geometry and machine learning (e.g., [6, 44] [43]), and the nuclear norm relaxation is the standard convex relaxation for the original NP-hard rank minimization problem [14].

A complete complexity analysis of the proposed algorithm would require detailed probabilistic estimates pertaining to these two steps, and although there are partial and related results scattered in the literature (e.g., [14] [7]), we are unable to provide a definitive result at this point. Instead, we present a simple complexity analysis below under the assumption that these two steps can be determined exactly, i.e., the convex relaxation using the nuclear norm gives the same result as the original rank minimization problem.

The computational complexity can be defined as the number of features (not necessarily only on the decision boundary) in $\mathbb{R}^d$ sampled during the process and this number is the same as the number of queries to the classifier $C$. From the above, it is clear that to determine the $m$-dimensional kernel subspace, it requires at least $O(m)$ sampled normals, i.e., $N$ has at least $m$ columns. Furthermore, to determine each normal vector at a given point $x$, it would require $O(d)$ number of points as the ambient dimension is $d$. Therefore, the total complexity is $O(dm)$. The multiplicative constant here, as can be readily seen, is bounded by the maximum number of steps required for the bracketing, and this number is independent of the dimensions $d, m$, provided the features are drawn from a bounded subset of $\mathbb{R}^d$ (Equation 2–9).

Once the kernel subspace $S_Y$ and the kernel type are determined, this allows us to focus on the intersection $\Sigma \cap S_Y$. In the case $m << d$, this reduction from $\Sigma \subset \mathbb{R}^d$ to $\Sigma \cap S_Y \subset S_Y$ is computationally significant. In particular, for polynomial kernels, we can sample $O(m^D)$ points on $\Sigma \cap S_Y$ to reconstruct the polynomial $\Psi(x)$ on $S_Y$. At this point, $\Psi(x)$ is a degree-$D$ polynomial in $m$ variables, and using recent results on
tensor decomposition (e.g., \cite{12}[5])\(^2\), we can decompose \(\Psi(x)\) (more precisely, its homogenized version)

\[
\Psi(x) = \sum_{i=1}^{r} \ell_i(x)^D, \tag{2–15}
\]

where \(\ell_1, \cdots, \ell_r\) are linear (homogeneous) polynomials. The smallest integer \(r\) for such decomposition gives the rank of the (homogeneous) polynomial (as a symmetric tensor) and in general, such decomposition is also possible for \(r\) greater than the rank. If we write the linear polynomials (after de-homogenization) as \(\ell_i(x) = z_i^\top x + 1\) for some vector \(z_i\), it is tempting to infer \(z_i\) as the support vector \(y_i\) from the above equation. However, because the non-uniqueness of the decomposition, \(z_i \neq y_i\) in general. Nevertheless, \(z_i\) do act as if they are support vectors in the sense that the evaluation of the polynomial \(\Psi(x)\) becomes computationally trivial using the above decomposition. For polynomial kernels, the recovery of these quasi-support vectors \(z_i\) then determines the decision boundary \(\Sigma\) exactly, essentially completing the deconstruction process. Although the general algorithms for tensor decomposition \cite{12}[5] require some mathematical machinery, the special case of quadratic kernels (degree-two polynomials) can be readily solved using eigen-decomposition of a symmetric matrix. For transcendental kernels, no similar results are known at present. Although the reduction from \(\Sigma \subset \mathbb{R}^d\) to \(\Sigma \cap S_Y \subset S_Y\) offers the possibility of reconstructing the decision boundary in \(S_Y\), due to the nature of the transcendental functions, the details are considerably more difficult than the polynomial case, and we leave its resolution to future research.

### 2.4 Experiments

We present two sets of experiments in this section. The first set of experiments evaluates various components of the proposed method and the second set of experiments

\(^2\) Algorithm 5.1 in \url{http://arxiv.org/pdf/0901.3706v2.pdf}, the archived version of \cite{12}.
applies the proposed method to explicitly deconstruct a kernel machine and subsequently improve it using incremental SVM [24].

2.4.1 Evaluation of Deconstruction Algorithm

We present two experiments using kernel machines whose support vectors are randomly generated (first experiment) and support vectors trained using real image data (second experiment). We remark that there is no qualitative differences between deconstructing kernel machines with randomly-generated support vectors and deconstructing kernel machines trained with real data since, in both cases, the kernel function and decision function (Eq 2–1) are the same. Using randomly-generated kernel machines allow us to study the behavior of the deconstruction algorithm over a much wider range of support vector configuration, demonstrating its accuracy and robustness.

In the first set of experiments, we set feature dimension $d = 30$, and we randomly generate 12 support vectors. For determining the kernel type, we sample 25 points close to the decision boundary $\Sigma$ and at each point, we compute the intersection of $\Sigma$ and a two-dimensional subspace. We fit a quadratic and then a cubic polynomial to these points, and the smallest degree giving an error below some threshold value is declared as the degree of the kernel. However, if in both cases the fitting errors are greater than the threshold value, the kernel is declared to be a Gaussian kernel at this location.

This is repeated at 25 sampled locations and a majority vote is used to determine the kernel type. Once the kernel type is determined, we use SVD to determine the dimension of the kernel subspace $S_\gamma$ and the subspace itself. For the Gaussian kernel, the nuclear-norm minimization is performed before using SVD to locate the subspace $S_\gamma$. In this experiment, we sample $s = 100$ points on the decision boundary in order to form the matrices $\mathbb{N}, \mathbb{X}$ and the tolerance in the bracketing step is set at $10^{-6}$. Let $\bar{S}_\gamma$ denote the kernel subspace computed by our method. We use the principal angles [29] between the two subspaces $S_\gamma, \bar{S}_\gamma$ as the metric for quantifying the error.
Summary  The gap between the singular values of $N$ is an important indicator of the dimension of the kernel subspace, and it is affected by the accuracy of the normals. Figure 2-4 shows the effect in terms of the radius $\delta$ used in computing the normals, showing the expected result that the ratio of $\delta/\epsilon$ is directly related to the accuracy of the recovered normals (larger ratios provide more accuracy). For determining the kernel type, the specificity for the polynomial kernels is close to 100% with the specificity of approximately 80% for the Gaussian kernel (and hyperbolic tangent kernel). This can be attributed to the majority voting scheme used in assigning the kernel type, and we leave it as important future work for designing more robust criteria. The accuracy of the recovered kernel subspaces is shown in Figure 2-5 and 2-6. The first figure shows the means and variances of the (cosine of) twelve principal angles, taken over one hundred randomly generated kernel machines using polynomial kernels. Note that $\cos^{-1}(0.99)$ is approximately 8° and this gives a good indication of the accuracy. In the second figure, the twelve principal angles computed before and after the rank-minimization are shown, indicating the correctness and necessity of performing rank-minimization. Finally, each deconstruction makes between 60,000 and 70,000 queries to the classifier, and on a typical 3Ghz machine, it takes no more than a few minutes to complete the deconstruction process. Since the algorithm is readily parallelizable (which would be important for deconstruction in high-dimensional feature spaces), a full parallelized and optimized implementation can be expected to shorten the running time considerably, perhaps in the range of only a few seconds.

In the second experiment, we train a kernel machine with cubic polynomial kernel using 1000 images from MNIST dataset [34]. The positive class consists of images of the digit 2 and the negative class consists of 0, 5, 7, 8. The trained kernel machine has 275 support vectors. Figure 2-3 displays the intersections of the decision boundary with several two-dimensional affine subspaces, noticing the superpositions of the images of 2 with images of other digits. In this experiment, we randomly generate 200
two-dimensional affine subspaces and for each subspace, its vote on the type of kernel is determined as above. Figure 2-7 shows the distribution of votes, clearly indicating the correct result. For this experiment, the gap in the singular values of $\mathbb{N}$ indicates the correct dimension of the kernel subspace (275) and the kernel subspace is also successfully recovered.

### 2.4.2 Kernel Machine Upgrade Without Source Code

In the second experiment, we demonstrate the possibility of upgrading a kernel machine without access to the kernel machine’s source code. As outlined in the introduction, we apply the deconstruction algorithm to deconstruct the kernel machine. This step provides us with the kernel type and quasi-support vectors (for a polynomial kernel machine). For the subsequent upgrade (or update), we use the incremental SVM algorithm [24] to retrain the kernel machine given the new training data. Specifically, we first train a kernel machine using MNIST dataset: images of digit 1 as positive
Figure 2-5. Means and variances of the cosines of the twelve principal angles between $\mathbf{S}_0$ and $\mathbf{S}_Y$. Means and variances are taken over one hundred independent deconstruction results for kernel machines with twelve support vectors using a polynomial kernel (Quadratic kernel on the left and cubic kernel on the right). (*Image best viewed in color*)

Figure 2-6. Means and variances of the the twelve principal angles between $\mathbf{S}_0$ and $\mathbf{S}_Y$. Means and variances are taken over one hundred independent deconstruction results for kernel machines with twelve support vectors using a Gaussian kernel. The principal angles before and after rank minimization are shown. (*Image best viewed in color*)

samples and the negative training samples comprise the remaining digits except 8. Dimensionality reduction is applied to the images using PCA to a feature space of dimension 60. An SVM with quadratic kernel is trained on these training samples, resulting in 97.30% true positive detection rate and 99.17% true negative detection.
Figure 2-7. Distribution of Votes on Kernel Type. For a cubic kernel machine trained on 1000 MNIST images, the distribution of votes on kernel type for 200 randomly sampled two-dimensional affine subspaces. The correct result is clearly indicated.

rate on the test dataset. The initial kernel machine has 48 support vectors. During

Figure 2-8. Cosines of the principal angles between the recovered kernel subspace and the ground-truth kernel subspace.

deconstruction, the kernel subspace is recovered using 800 sampled normal vectors. Let \( \mathbf{N} \) denote the matrix obtained by horizontally stacking together the normal vectors and \( \mathbf{N} = \text{USD} \), its SVD decomposition. The plot of the singular values is shown in Figure 2-9 and the significant gap between the 48th and 49th singular values indicate the
correct dimension (and the number of support vectors). The principle angles between the kernel subspace estimated by the first 48 columns of $U$ and the ground-truth is shown in Figure 2-8. Once the kernel subspace is recovered, we proceed to recover the

![Figure 2-9. Singular values of the matrix $N$. The gap between 48th and 49th singular values is significant as the gaps among the remaining singular values are substantially smaller. The correct dimension of the kernel subspace (and the number of support vectors) is 48.](image)

quasi-support vectors. The kernel machine defined by the quasi-support vectors should be a good approximation of the original kernel machine and this is shown in Table 4.1, where we compare the classification results using the recovered kernel machine and the original one. In this example, the results as expected are quite similar, with the recovered kernel machine actually performing slightly better. Once we have recovered the quasi-support vectors, we next proceed to upgrade the kernel machine. The task is to upgrade a kernel machine that recognizes only digit 1 to a kernel machine that recognizes digits 1 and 8. The classification results for the initial and upgraded kernel machines are tabulated in Table 4.2. As shown in the table, before the upgrade, the
original kernel machine performs poorly on the images of digit 8 and for the upgraded machine, both digits can now be successfully classified.

Table 2-1. Confusion matrices for the original kernel machine and the kernel machine defined by the recovered quasi-support vectors. Both machines are tested on the same test dataset.

<table>
<thead>
<tr>
<th></th>
<th>Quasi-SV Machine</th>
<th>Original Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>outcome</td>
<td>outcome</td>
</tr>
<tr>
<td></td>
<td>+ve</td>
<td>-ve</td>
</tr>
<tr>
<td>Positive</td>
<td>100.00%</td>
<td>00.00%</td>
</tr>
<tr>
<td>Negative</td>
<td>3.73%</td>
<td>96.27%</td>
</tr>
</tbody>
</table>

Table 2-2. Comparisons of classification results for the original kernel machine and the upgraded kernel machine

<table>
<thead>
<tr>
<th></th>
<th>Classification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original Machine</td>
</tr>
<tr>
<td>Digit 1</td>
<td>97.30%</td>
</tr>
<tr>
<td>Digit 8</td>
<td>00.00%</td>
</tr>
<tr>
<td>Negative</td>
<td>99.17%</td>
</tr>
</tbody>
</table>
CHAPTER 3
DECONSTRUCTING CASCADE OF LINEAR CLASSIFIERS

A linear classifier is perhaps the simplest classifier for vectorial features in terms of its computational complexity and the geometric complexity of its decision boundary, a linear hyperplane. However, the geometric simplicity often renders it ineffective for nontrivial classification problems that almost always require nonlinear decision boundaries. The nonlinearity can be achieved using multiple linear classifiers, and a popular method for organizing these linear classifiers into an effective classifier is to structure them using a linear decision tree, a cascade of (binary) classifiers. The combination of linear classifiers and decision trees create a flexible and versatile platform for learning highly nonlinear decision boundaries, with only a modest increase in computational complexity. In particular, the face detector originally proposed by Viola and Jones [49] (and its many variants that have been incorporated into a wide variety of commercially-available devices) is certainly the most well-known example of a successful binary classifier that employ the cascading architecture, and there is a substantial body of literature devoted to methods for learning (linear) cascades. For example, cascade classifiers were used in object detection with rotated Haar-like features [35], contour fragments [45], and deformable part models [28]. Numerous applications of cascade schemes were developed including face detection [52], pedestrian classification [40], biomedical signal analysis [42], and text detection in natural images [16].

As per our discussion in chapter 2, linear classifiers could be deconstructed quite easily by methods listed in that chapter, cascade of linear classifiers however poses a more challenging scenario. The cascade of linear classifiers makes the decision boundary to be piecewise linear, one side guaranteeing success of bracketing with high probability but on the other requiring bracketing to be performed large number of times by pairing random positive and negative samples. Note that, being piece wise
linear decision boundary cannot be represented in parametric form. Order of classifiers in a decision tree is very important, we explore using computational time to figure out size of cascade and ordering of linear classifiers. We present experimental results on deconstructing the face detector implemented in the OpenCV library to confirm the validity and viability of the proposed approach.

### 3.1 Problem Introduction

Let us assume a binary classifier $C$, internally cascade of linear classifiers, is given and the feature space $\mathbb{R}^k$ is known. Examples of the latter include images of a given size ($k$ pixels) for face detection and image features such as SIFT [37] and HOG [19] with known dimensions that are popular for image-based classification problems. The object of interest in deconstructive learning is the classifier $C$, and the goal is to determine as much as possible the internal working of $C$. In the context of deconstructing a linear cascade, the aim is then to recover the constituent linear classifiers in the cascade by probing the feature space and locating the decision boundary. As the simplest linear cascade, a single linear classifier provides an illustrative example. A binary linear classifier $h(v)$ for $v \in \mathbb{R}^k$ is determined by a normal vector $h \in \mathbb{R}^k$ and a threshold value $\omega$ such that a feature $v$ is considered positive ($h(v) = 1$) if $h^\top v \geq \omega$ and negative ($h(v) = -1$), otherwise. Therefore, $C$ is completely deconstructed if $h$ and $\omega$ can be determined, and this can be easily accomplished using the idea of bracketing a pair of positive and negative features to locate sufficiently many points on the decision boundary. In theory, once $k$ points on the decision boundary have been located, both the normal vector $h$ and threshold constant $\omega$ can be determined (up to an unimportant multiplicative constant).

For a general linear cascade, its decision boundary $D$, although highly nonlinear, is readily shown to be piecewise linear. Essentially, each linear piece of $D$ corresponds to a part of the decision boundary for one of its constituent linear classifiers (details below), and geometrically, the local linearity allows the constituent linear classifiers to be
recovered using bracketing, as for single linear classifiers described above. In particular, if \( C \) has \( m \) constituent linear classifiers, recovering these linear classifiers requires bracketing points on \( m \) local linear pieces on the decision boundary \( D \). Unfortunately, there is not a priori way to know these \( m \) local linear pieces on \( D \), and this inevitably requires a randomized approach that initiates bracketing using a large number of pairs of positive negative features. On the other hand, the cascading structure cannot be determined by considering geometry alone; instead, a careful analysis of the CPU running times for a large collection of features will allow us to determine, with reasonably certainty, the location of each linear classifier in the cascade. Once the linear classifiers and their locations in the cascade have been determined, the entire makeup of the classifier \( C \) can be determined in a straightforward manner.

The specific algorithm proposed in this chapter closely follows the outline above, and using bracketing, it samples a large number of points on the decision boundary \( D \), and for each such sampled point, its associated local linear piece is estimated, yielding a normal vector and its location in the cascade. In real and practical experiments, two important complications become apparent. First, due to local linearity of \( D \) and the finite precision of the computation, the algorithm often overestimates the number of linear classifiers by approximately 30\%, and extra steps are required to eliminate spurious linear classifier. Second and more importantly, many image-based applications (e.g., face detection) use integral features, often because the intensity values themselves are quantized integers. In particular, the features belong to the integer lattice \( \mathbb{Z}^k \subset \mathbb{R}^k \), and not surprisingly, the deconstruction problem is considerably easier for real features than integer features (linear programming vs. integer programming) in part because bracketing is no longer effective for integer features. Furthermore, the discrete nature of the integer features also require more constraints to determine the the parameters in full.
### Algorithm 1 Calculating Normal from PN-Pairs

**Input:** $v_+, v_-, \sigma$

**Output:** $n$

$m = (v_+ + v_-)/2$

$S = \text{samples in } 10\sigma$ 

$neighbourhood$ of $m$

$r_S = \text{Classify}(S)$

$n = FB(S, r)$

### Algorithm 2 Bracketing Algorithm for samples in integer lattice

**Input:** $v \in \mathbb{R}^n$

**Output:** $1, -1$

$r \leftarrow 0^n$

$r_v = \text{Classify}(v)$

for $i = 1 \rightarrow n$

$v_1 = v, v_2 = v$

$v_1(i) = v_1(i) + 1$

$v_2(i) = v_2(i) - 1$

$r_1 = \text{Classify}(v_1)$

$r_2 = \text{Classify}(v_2)$

if $r_1 r_2 < 0$

$r_i = 1$

if $r_i = r_v$

$r_i = -1$

end if

end if

end for

### Algorithm 3 Bracketing Algorithm

**Input:** $v_+, v_-, \delta$

while $|v_+ - v_-| < \delta$

$m = (v_+ + v_-)/2$

if $\text{Classify}(d) > 0$

$v_+ = d$

else

$v_- = d$

end if

end while

---

Figure 3-1. Algorithms for Deconstruction of Classifiers: *Left:* Finding normal for the real lattice, FB is boundary by Fisher Discriminant. *Middle:* Bracketing when the integer lattice is being used. *Right:* Bracketing algorithm.

### 3.2 Preliminaries

We will denote by $C$ the classifier to be deconstructed, and it is assumed to be a binary function defined on the $k$-dimensional feature space $\mathbb{R}^k$ taking values (labels) in $\{1, -1\}$: for a feature $v \in \mathbb{R}^k$, the binary classifier $C$ returns $C(v) = \pm 1$, indicating the feature’s label. Other than the labels, the black-box classifier $C$ does not provide any other information (such as confidence level) about the feature $v$. In the following discussion, features will always refer to real-valued features, vectors in $\mathbb{R}^k$. However, on occasions, we will discuss integral features, and they refer to features on the integer lattice $\mathbb{Z}^k \subset \mathbb{R}^k$. A classifier $C$ taking only integral features is then a binary function defined on $\mathbb{Z}^k$. 

50
Figure 3-2. Filters recovered by probing the decision boundary using positive and negative pairs. Black, white and gray regions indicating the negative, positive and null regions of the Haar feature. (a) Recovered Haar features considered as real features, and (b) recovered Haar features considered as integral features. Compared with integer features, the real features are considerably easier to be recovered. For integer features, there are several incorrect and spurious features.

Internally, $C$ is assumed to have employed a cascade of binary linear classifiers. Specifically, the cascade is a linear decision tree with $l$ nodes (or layers), and for each node indexed by the integer $1 \leq i \leq k$, there is an associated binary classifier (node classifier) $L_i$. For a feature $v$, the value $C(v)$ is determined by the values $L_i(v)$ according to

$$C(v) = \begin{cases} 
1, & L_i(v) = 1 \text{ for all } 1 \leq i \leq l, \\
-1, & \text{otherwise}.
\end{cases}$$

Each node classifier $L_i$ in turn is formed by a collection of $n_i$ (weak) binary linear classifiers $h_{i1}, h_{i2}, \ldots, h_{in_i}$, and their associated weights $a_{i1}, a_{i2}, \ldots, a_{in_i}$, and a threshold
value $\Omega_i$ such that the value $L_i(\mathbf{v})$ is given by

$$
L_i(\mathbf{v}) = \begin{cases} 
1, & \sum_{j=1}^n a_{ij} h_{ij}(\mathbf{v}) \geq \Omega_i \\
-1, & \text{otherwise.}
\end{cases}
$$

(3–1)

Finally, weak linear classifier $h_{ij}$ is specified by a (normal) vector $\mathbf{h}_{ij}$ and a threshold value $\omega_{ij}$ such that

$$
h_{ij}(\mathbf{v}) = \begin{cases} 
1, & \mathbf{h}_{ij}^T \mathbf{v} \geq \omega_{ij} \\
-1, & \text{otherwise.}
\end{cases}
$$

For face detectors (e.g.,[1]), each weak classifier $h_{ij}$ is associated with a Haar feature, and in particular, the normal vector $\mathbf{h}_{ij}$, when viewed as an image, is made up of a small number of rectangular regions with constant intensity (a small integer with absolute values typically no greater than four).

In summary, the black-box classifier $\mathbf{C}$ is specified by the following internal parameters: $L, n, \Omega_i, a_{ij}, \mathbf{h}_{ij}, \omega_{ij}$, and the classifier $\mathbf{C}$ has $m = n_1 + n_2 + \ldots + n_i$ constituent linear classifiers $h_{ij}$. The goal of the deconstruction learning is to recover these parameters by probing the classifier $\mathbf{C}$ using features in $\mathbb{R}^k$, predefined or randomly generated. Specifically, the learning algorithm aims to

- determine the number of nodes ($l$) in the cascade,
- determine, for each node $i$, the number of weak linear classifiers ($n_i$) and their associated normal vectors $\mathbf{h}_{ij}$, weights $a_{ij}$ and threshold values $\omega_{ij}$.
- determine, for each node $i$, its threshold $\Omega_i$.

It is clear that the threshold parameters $\Omega_i, \omega_{ij}$ can be normalized to one, in exchange for rescaling the weights $a_{ij}$ and the normal vector $\mathbf{h}_{ij}$, respectively. Therefore, the deconstruction process can be formulated in two related parts:

- Determine the number of nodes, $l$, and the location in the cascade of each weak linear classifier $h_{ij}$ (i.e., $i$ in $h_{ij}$).
- Determine the weak linear classifiers $h_{ij}$ and its associated normal vector $\mathbf{h}_{ij}$. 

52
Figure 3-3. Multiple Haar-like features present in same retrieved filter, indicating the
decision boundary is defined by combination of more than one Haar
classifiers (a) Integer case (b) Real case

The first step can be determined by exploiting the computational consequence of the
cascading architecture, and the second part requires knowing some geometry of the
decision boundary $D$ of $C$.

For computational efficiency, the cascade is often arranged in the order of the node
classifier’s computational complexity, i.e., $0 < n_1 < n_2 < ... < n_l$, and for each feature,
the evaluation starts at the root classifier $L_1$ and proceeds down the tree. The process
terminates whenever $L_i(v) = -1$ for some $i$ or $L_i(v)$ has been evaluated for all $1 \leq i \leq l$.
In the former case, $L_j(v)$ is not evaluated for $j > i$ and $C(v) = -1$, and in the latter,
$C(v) = 1$. In particular, the running time is longer for a feature that is rejected at the
deeper part of the cascade, and positive features always take the longest. The CPU
running time is a useful consequence of the cascading architecture, and careful analysis
of running time will allow us to determine the structure of the cascade.

**Geometry of the Decision Boundary $D$** While the decision boundary $D$ for $C$ can be
highly nonlinear, it is nonetheless piecewise linear. In order to analyze $D$, we will start
with the decision boundary of each node classifier $L_i$. Let $R_i^+$ denote the positive region
for $L_i$ in $\mathbb{R}^k$: $v \in R_i^+$ iff $L_i(v) = 1$. It is clear from Equation (3–1) that $R_i^+$ is the union of a
collection of cells in $\mathbb{R}^k$, where the cells are unions of $n_i$ half spaces in $\mathbb{R}^k$ defined by the
values of the weak classifiers $h_{ij}$. For example, if $L_i$ has three weak classifiers $h_{i1}, h_{i2}, h_{i3},$
$\mathbb{R}^k$ is partitioned into eight cells according to the eight different values of $(h_{i1}, h_{i2}, h_{i3})$, and the set $R_i^+$ is the union of the cells for which $a_{i1}h_{i1} + a_{i2}h_{i2} + a_{i3}h_{i3} \geq \Omega_i$. In particular, the boundary of $R_i^+$ must be piecewise linear. Consequently, let $R^+$ denote the positive
region for \( C \) in \( \mathbb{R}^k \), and it is immediate that

\[ R^+ = R_1^+ \cap R_2^+ \cap \ldots \cap R_i^+, \]

and this also implies that the boundary of \( R^+ \), which is the decision boundary \( D \) of \( C \), is also piecewise linear and a polytope (possibly unbounded). More specifically, \( D \) has a subset of measure zero containing points belonging to the intersection of the decision boundaries of two or more weak classifiers. Outside of this subset, \( D \) is locally linear (piecewise linear) in the sense that each point is contained a (local) linear facet that is a connected component of the (open) interior of the intersection of \( D \) with the decision boundary of one unique weak linear classifier. We remark that the normal vector to a linear facet is also the normal vector for its corresponding linear classifier.

### 3.3 Deconstruction Algorithm

The deconstruction algorithm has two sequential steps. In the first step, the algorithm locates a large number of points on the decision boundary \( D \). The deconstruction algorithm starts with a collection of features, and the main idea is to use these features, if necessary, to generate more features in \( \mathbb{R}^k \), and initiate bracketing on pairs of positive and negative samples to approximately locate points on the decision boundary \( D \). The sampled points on \( D \) will have a large probability\(^1\) of belonging to a linear facet of \( D \), and its associated normal vector can then be estimated. Furthermore, for each sampled boundary point, its location in the cascade is also determined by monitoring its running times. In the second step, these sampled boundary points and their associated normal vectors and cascade locations are used to determine the weights \( a_i \), and hence the entire classifier \( C \).

\(^1\) Although theoretically the probability is one, due to finite precision, in practice, the probability is strictly less than one.
Figure 3-4. Average running time for the 4400 features rejected by the classifier C. (a) For each plotted point \((t, i)\), \(t\) is the average running time (vertical axis) taken by feature \(i\) (horizontal axis). (b) Average running time sorted in increasing order showing that the features can be divided into different groups according to their running times and hence their locations in the cascades. We perform mean-shift clustering using processing times taken by features. Horizontal lines indicate boundaries of clusters. Clusters are visible in color print.

**Bracketing** To locate a point on or near the decision boundary we perform bracketing (section 2.3.1). Specifically, starting with a pair of positive and negative features, \(v_+, v_-\), respectively, we can successively halve the interval between \(v_+, v_-\) to produce a sequence of positive-negative pairs, all of which lie on the line segment joining \(v_+, v_-\)

\[
(v_+, v_-) \quad (v^1_+, v^1_-) \quad (v^2_+, v^2_-) \quad (v^3_+, v^3_-) \ldots
\]

such that \(|v_+^{i+1} - v_-^{i+1}| = \frac{1}{2}|v_+^i - v_-^i|\), as in root finding, using the label provided by \(C(v)\) as the sign of the function value. For real features, this process can be repeated indefinitely to locate the point on the decision boundary within required precision, and the pseudo-code for the bracketing function is given in Figure 3.1. For integral features, the process stops once \(|v_+^i - v_-^i| < 1\).

**Estimating Normal Vectors** For a given sampled boundary point, given as a pair of positive and negative samples \((v_+, v_-)\) such that \(s = |v_+ - v_-| << 1\), the algorithm
proceeds to estimate the normal vector associated with its local linear facet. For real features, the idea is to randomly generate a number of positive and negative samples in a small neighborhood (of radius $r$) of $v_+, v_-$ and compute a linear classifier that separates these features. Note that for a sufficiently small neighborhood, these randomly generated samples are linearly separable, and the normal vector of the computed linear classifier is used as an approximation for the desired normal vector. We remark that it is possible to show that this approximation becomes exact asymptotically as $r \to 0$, $\frac{s}{r} \to 0$ (for detail section 2.3.2).

For the integral features, the above approach is not viable since the interval cannot be decreased indefinitely. However, this can be circumvented by individually perturbing each component of the feature vector $v$ by $\pm 1$. Given the pair $(v_+, v_-)$, the behaviors of the positive and negative regions of the vector $h_j$ are different under $\pm 1$ perturbation for $v_+, v_-$. For instance, suppose for a given component of $h_j$, an increase by one pushes $v_+$ into the negative region. It then follows that this component change has no effect on $v_-$, i.e., $v_-$ will still be classified as a negative sample using the perturbed
$h_{ij}$. However, if the given component is negative, it is straightforward to see that the component change of $-1$ will affect the label of $v_-$ but not $v_+$. Therefore, by observing this qualitative difference between each component of $h_{ij}$, the positive and negative (and also zero) regions of $h_{ij}$ can be completely determined. The pseudocodes for estimating the normal vectors for real and integral features are given in Figure 3.1. We note that for $h_{ij}$ modelled after Haar features, the above process is particularly easy to implement because the positive and negative regions of $h_{ij}$ are rectangular and components typically have integral values with small absolute values.

**Determining Cascade Locations** For a sampled boundary point represented as a pair $v^+, v^-$, let $h_{ij}$ denote the weak linear classifier on whose local linear facet the point resides. Since $v^-$ lies on the negative side of the decision boundary for $h_{ij}$, the negative sample $v^-$ is rejected by $L_i$ (and hence $C$) at node $i$. Therefore, by timing the running time for $C$ to reject $v^-$, the location of $h_{ij}$ in the cascade can be determined. In particular, the negligible difference between the running times of two samples rejected at neighboring nodes can be considerably magnified by running each sample multiple times. Figure 3-4B shows an expected staircase-like plot for the running times.

**Recovering the Weights** The data collected in the first step provides us with the number $l$ of nodes in the cascade and also for each node, the constituent linear classifiers $h_{ij}$ in the node classifier $L_i$. The final step is to determine the weights $a_{ij}$ for node $i$. As mentioned before, we can assume the threshold value $\Omega_i = 1$. Given a sampled boundary point represented as the pair $(v^+, v^-)$, we have

$$
1 + \delta > a_{i1} h_{i1}(v^+) + \ldots + a_{in_i} h_{in_i}(v^+) \geq 1
$$

$$
1 - \delta < a_{i1} h_{i1}(v^-) + \ldots + a_{in_i} h_{in_i}(v^-) < 1
$$

for some small positive constant $\delta$ because $v^+, v^-$ are near the decision boundary. In particular, a good estimate of the weights can be obtained by imposing the linear
Figure 3-6. Dictionary elements ordered according to the average running time taken by negative features in the pairs. The graph follows a familiar staircase pattern, indicating the features' locations in the cascade.

constraint

\[ a_{i1} h_{i1}(v^+) + \ldots + a_{in} h_{in}(v^+) = 1 \]

for pairs \( v^+, v^- \) such that \(|v^+ - v^-|\) is sufficiently small. Therefore, by generating sufficiently many such pairs of positive and negative features, we have enough linear equations to solve for a set of weights\(^2\).

**Practical Working of the Algorithm** In practice, the main qualitative bottleneck is the step that estimates the normal vectors. Because of the finite precision in computation, we cannot compute the limit described above, and in our simplified implementation reported below, the algorithm invariably overestimates the number of weak classifiers,

\(^2\) The error analysis for this approximation will be addressed in a future work.
typically by about 30%. Improvements are certainly possible and it is the focus of an ongoing work. However, in the context of deconstructive learning, this current achievable result is already quite interesting, since presumably the true weak classifiers can be recovered by simply training on this collection of overestimated linear classifiers. Compared with the original training algorithm, which must search for these weak classifiers over a huge set of possible candidates, the search space in our case has been immensely reduced (only 30% more than the actual number of weak classifiers). Recovery of the actual linear classifiers, in principle, should be straightforward given enough training features.

3.4 Experiments

In this section, we present a summary of experiments on deconstructing the Soft-Cascade [11] and Viola-Jones face detector [49] included in OpenCV library. Viola Jones face detector is a linear cascade classifier (as described above) trained on 20 × 20 face images. For the experiments, we use face images that are detected as face images by the detector and we also randomly generate seventy-nine thousand pair of positive and negative samples, i.e. face and non-face image pairs. They are of same size as the training images. The OpenCV face detector only accepts integral features and we modify the implementation so that the detector also accepts real features. Due to limited space, we will present only the results from the first step of the algorithm that determines the weak classifiers and the number of nodes in the cascade.

3.4.1 Soft Cascade

Soft-cascade was trained using 33 Haar like features on face images and mathematically it is of much simpler form. Instead of levels of cascades it has single stage consisting of T weak classifiers. These weak classifiers are ordered according to how soon they can reject a non-face images. Let \( c_t(x) = \alpha_t h_t(x) \) be response of \( t^{th} \) classifier, then at any stage \( t \) if accumulative sum \( H_t(x) = \sum_{i=1}^{t} c_i(x) \) is less than stage threshold \( r_t \) it is rejected. Using only about 2,000 random pairing of positive and negative samples
in the algorithm described above we were able to recover 32 out 33 classifiers. The recovered features are shown in Figure 3-7, where the last classifier is visibly non-Haar in structure.

### 3.4.2 Viola-Jones Haar Cascade

In comparison to self trained soft-cascade, Haar Cascade included in OpenCV has been trained on much larger dataset, it has more than two thousand weak Haar classifiers distributed in its 22 stages. For such a complex classifier, one has to recover the number of stages and assign classifiers to each stage. We were able to recover 98% of the weak classifiers in the OpenCV Haar cascade where recovered classifiers...
had correlation of 0.9 or more with the original classifiers. More detail result is given in Figure 3-8. These results prove that the recovered classifier set does contain classifiers used in the OpenCV cascade. We discuss below how to construct cascade given these recovered classifiers.

**Recovering Weak Classifiers** Figure 3-2 demonstrates the recovered Haar features (weak classifiers) using one hundred pairs of positive and negative features. In this experiment, we show all the recovered Haar features, considering them as real and integral features. The deconstruction process uses bracketing and estimate the local normal vectors as described in the previous section. It is clear from the figure that 1) there are some spurious and incorrect Haar features recovered, particularly for the integer case, and 2) the integer case is substantially more difficult than the real case as there are more spurious recoveries for the integral features.

**Recovering Locations in the Cascade (stages)** We present two results on recovering the locations of weak classifiers in the cascade. In the first experiment, we take 4400 features (images), and each feature is repeated 5000 times for monitoring their running times as described before. The plot is shown in Figure 3-4. For each cascade level, we calculated average running time taken by images rejected at that level and also the standard derivation. This indicates that there is difference between the time taken between two cascade levels, and sufficient samples would help differentiating between the layers.

In the second experiment Figure 3-5, we use 30,000 pairs of positive and negative samples to discover possible classifier and pair of \( v^+, v^- \) making bracket. Since we know that negative images are rejected at different stages, time it took to reject \( v^- \) could be indicative of where the classifier exists (note that \( v^- \) is not just any negative sample but one very near to boundary due to bracketing algorithm. The collection all the distinct weak classifiers generated in the above process is used to learn dictionary \( D \), and associate each dictionary element with the average running time of the pairs with
the given element. The plot is shown in Figure 3-6. To recover the number of stages, we preformed mean-shift clustering on the processing time as shown in Figure 3-4B. This gives us the prediction of 20 stages (original classifier has 22 stages). Few earlier stages are combined in one cluster. Reason being that earlier stages not only have small number of classifiers but also reject negative images quite quickly therefore processing time in earlier stages is undistinguishable. This could be rectified by running negative images showing low processing times more in the loop, thus making times of rejection more distinguishable.

**Recovering Stage Information** After we cluster the recovered features into cascades, it is easy to estimate stage level information (i.e. right and left weights for each classifier, as well stage threshold). Due to bracketing algorithm we have positive and negative pair near boundary associated with each recovered classifier. All such image pairs belonging to classifiers in cluster \( i \) are used to solve for right, left weights and stage threshold. In addition these images, all the boundary pairs associated to classifiers above this stage are taken as positive images.

### 3.5 Concluding Remarks

We conclude this chapter with few pertinent remarks and observations. This chapter exposes fundamental asymmetry between constructive learning (learning classifiers from training samples) and deconstructive learning. As reported in [49], the training time for the Viola-Jones face detector using adaboost takes more than one month. However, for deconstructive learning, the (almost) entire set of Haar features selected by adaboost can be recovered in a matter of a few hours. Deconstructive learning is highly parallelizable and deconstruct a given classifier in few hours. We have shown how to deconstruct a software provided by OpenCV library where we dealt with the issues of software only taking integer images. However we still have to deal with the issues of image processing and normalizations taking place before the image over to classifier. This could be part of our future work.
In previous chapters we laid down much of the foundation of how to deconstruct a classifier. We studied deconstruction of SVM and cascade of linear classifiers; now using these tools we present framework to deconstruct a complete classification system, where classifier is just one part. In following sections we re-introduce classification system and explain why deconstructing such a system is a complex, difficult but at the same time rewarding task. To evaluate effectiveness of our method we present results on deconstruction of OpenCV human/person detector, motorbike detector (trained by us) and deconstruction of few examples on the MNIST dataset.

4.1 Problem Introduction

Like in previous chapters, let us assume a binary classification system $C$. The system, as a binary classifier of images, is presented only as a binary executable that takes images as inputs and outputs $\pm 1$ as its decision for each image. The classifier is laconic in the sense that except the predicted label $\pm 1$, it does not divulge any other information such as confidence level or classification margin associated with each decision. However, we are allowed to query the detector (classifier) using images, and the problem studied in this dissertation is to determine the inner working of the classifier using only image queries and the classifier's laconic responses.

Unlike previous chapters here we are looking for complex information. We want to determine the type of features it uses? Instead of assuming we want to figure out what kind of internal classifier does it deploy? Support vector machine (SVM) or cascade of linear classifiers or something else? And then If it uses SVM internally, what kind of kernel function does it use? How many support vectors are there and what are the support vectors?

Deconstructive learning as presented above is an inverse problem; therefore, without an appropriate regularization, the problem is ill-posed and it is impossible to
define desired solutions. In particular, since we are allowed to access only the laconic responses of the classifier, the scope seems almost unbounded. The appropriate notion of regularization in this context is to define a tractable domain on which solutions can be sought, and the main contribution of this dissertation is the proposal of a computational framework that would allow us to pose and answer the above questions as computationally tractable problems. Our proposal is based on a specific assumption on the classifier $\mathbf{C}$ that its internal structure follows the common two-component design: a feature-transform component that transforms the input image into a feature and a machine-learning component that produces the output by applying its internal classifier to the feature (Figure 4-1). Many existing binary classifiers in computer vision follow this type of design, a clear demonstration of the division of labor between practitioners in computer vision and machine learning. For example, most of the well-known detectors such as face and pedestrian detectors (e.g., [8, 34, 50]) conform to this particular design, with other lesser-known but equally-important examples in scene classification, object recognition and others (e.g., [10, 27]) adopting the same design. By clearly delineating the vision and learning components, we can formulate a computational framework for deconstructing $\mathbf{C}$ as the identification problem for its two internal components from a finite collection of potential candidates.

More precisely, for a given vision classifier $\mathbf{C}$ (e.g., an object detector), the deconstruction process requires a list of features (and their associated transforms) $\mathcal{F}$ and a list of (machine learning) classifiers $\mathcal{C}$. Based on these two lists, the algorithm would either identify the components of $\mathbf{C}$ among the elements in $\mathcal{F}$ and $\mathcal{C}$ or return a void to indicate failure in identification. Computationally, the lists define the problem domain, and they constitute the required minimal prior knowledge of $\mathbf{C}$. In practice, the general outline of the feature used in a particular vision algorithm is often known and can be ascertained through various sources such as publications. However, important design parameters such as smoothing values, cell/block/bin sizes etc.,
are often not available and these parameters can be determined by searching over
an expected range of values that made up the elements in \( \mathcal{F} \). Similarly, the type of
classifier used can often be narrowed down to a small number of choices (e.g., an
SVM or a cascade of linear classifiers). Within this context, we introduce three novel
notions, feature identifiers, classifier deconstructors and geometric feature-classifier
compatibility, as the main technical components of the deconstruction process.
Specifically, feature identifiers are a set of image-based operations such as image
rotations and scalings that can be applied to the input images, and the different degree
of sensitivity and stability of the features in \( \mathcal{F} \) under these operations would allow us
to exclude elements in \( \mathcal{F} \), making the process more efficient. For example, suppose
\( \mathcal{F} \) contains both SIFT and HOG-based features. Since SIFT is in principle rotationally
invariant, SIFT-based features are more stable under image rotations than HOG-based
features; and therefore, if \( \mathcal{C} \) uses a SIFT-based feature internally, it outputs would be
expected to be more stable under image rotations. Therefore, by querying \( \mathcal{C} \) with rotated
images and comparing the results with un-rotated images, we can exclude features
in \( \mathcal{F} \) that are rotationally sensitive. The classifier deconstructors, on the other hand,
are algorithms that can deconstruct classifiers in \( \mathcal{C} \) using a (relatively) small number
of features by recognizing certain geometric characteristics of the classifier's decision
boundary (e.g., its parametric form). For example, an SVM deconstructor algorithm
is able to (given sufficiently many features) determine the number of support vectors
and the type of kernel used by a kernel machine by recognizing certain geometric
characteristics of its decision boundary. The interaction between elements in \( \mathcal{F} \) and \( \mathcal{C} \)
during the deconstruction process is based on the notion of geometric feature-classifier
compatibility: for a pair \((f, c)\) of feature \(f\) and classifier \(c\), they are compatible if given
sufficiently many features defined by \(f\), the deconstructor algorithm associated to \(c\)
can correctly recognize its decision boundary. More specifically, given a vision classifier
\( \mathcal{C} \) internally represented by a pair \((f, c)\) of feature \(f\) and classifier \(c\), we can query \( \mathcal{C} \)
using a set of images \( I_1, \ldots, I_n \), and using the feature (and it associated transform) \( f \),
we can transform the images into features in the feature space specified by \( f \). The
deconstructor algorithm associated with \( c \) then determines the classifier based on these
features. However, for an incorrect hypothetical pair \( (\overline{f}, c) \),
the difference between the
transformed features specified by \( \overline{f} \) and \( f \) are generally non-linear,
and this non-linearity changes the geometric characteristics of the decision boundary
in the feature space specified by \( \overline{f} \), rendering the deconstructor algorithm \( c \) unable to identify
the decision boundary (Figure 4-1).

Figure 4-1. Schematic illustration of deconstructive learning and proposed algorithm.
Left: Two-component design of a classifier: a feature-transform component provided by computer vision
followed by a feature-classification component furnished by machine learning. Right: Schematic
illustration of the proposed deconstruction algorithm. Internally, the algorithm searches over a set of
candidate feature spaces and probes the spaces for decision boundaries. Only in the correct feature space
the parametric form of the decision boundary would be recognized by the deconstructor algorithm.

The abstract framework outlined above provides a practical and useful modularization
of the deconstruction process so that the individual elements such as the formation of
feature and classifier lists, feature identifiers and classifier deconstructors can be subject
to independent development and study. In this dissertation, we realize the abstract
framework in concrete terms. Specifically, we introduced two deconstructor algorithms
for support vector machine (SVM) and for the cascade of linear classifiers. The former
is a popular family of classifiers widely used in vision applications and the latter is often
deployed in object detectors, with the face detector of Viola-Jones as perhaps the most
In the experimental section, we present four preliminary experimental results demonstrating the viability of the ideas proposed in this chapter.

- In the first experiment, we show the application of a few simple heuristics can substantially reduce the size of feature list $\mathcal{F}$ and therefore, allow for a more efficient deconstruction process.

- In the second experiment, we present the result of a complete deconstruction of OpenCV’s HOG-based pedestrian detector. The entire deconstruction process searches over one hundred potential features to correctly identify the linear classifier used in the detector. The normal vector of the linear classifier recovered by our algorithm has the normalized correlation of more than 0.99 with the ground truth (i.e., with an angular difference smaller than 2°).

- In the third experiment, we repeat deconstruct classifier trained to detect aeroplanes.

- In fourth and last experiment, we recover the linear subspace used to perform dimension reduction for feature extraction purposes, and then recover the quadratic kernel based SVM classifier.

The MATLAB implementation of the deconstruction algorithm is only around 100 lines of code and it takes no longer than an hour to correctly identify the feature, the classifier type (linear SVM) and the linear classifier itself.

Let us reiterate that to the best of our knowledge, there is not a previous work on deconstructive learning comparable to the joint recovery of feature transform and classifier as outlined above. Where most of the learning problems deal with the distribution and probabilities in our setting geometry replaces probability as the joint feature-label distribution gives way to the geometric notion of decision boundary as the main target of learning.

### 4.2 Deconstruction Process and Method

Let $\mathcal{F}$, $\mathcal{C}$ denote the feature and classifier lists. Given a classifier $\mathcal{C}$ with the two-component design as described above, the deconstruction algorithm attempts to identify the feature-transform and feature-classification components of $\mathcal{C}$ with the elements in $\mathcal{F}$ and $\mathcal{C}$. Specifically, we assume that
• Each feature \( f_i \in \mathcal{F} \) defines a feature transform \( f_i : \mathbb{R}^d \to \mathbb{R}^n \) from the image space \( \mathbb{R}^d \) to a feature space \( \mathbb{R}^n \) of dimension \( n_i \). For technical reason, the feature transform \( f_i : \mathbb{R}^d \to \mathbb{R}^n \) is assumed to be Lipschitz continuous in that \( \| f_i(\mathbf{I}_a) - f_i(\mathbf{I}_b) \|_2 < L_i \| \mathbf{I}_a - \mathbf{I}_b \|_2 \) for some positive constant \( L_i > 0 \) and \( \mathbf{I}_a, \mathbf{I}_b \in \mathbb{R}^d \). Furthermore, we assume that an inverse of the feature transform \( f_i \) can also be computed: for \( \nu_i \in \mathbb{R}^n \), an image in \( f_i^{-1}(\nu_i) \) can be computed\(^\dagger\).

• Each element \( c_i \in \mathcal{C} \) represents a known family of classifiers (e.g., SVM and cascade of linear classifiers as two different families) and has its associated deconstructor algorithm (also denoted as \( c_i \)). For each feature space \( \mathbb{R}^n \), with sufficiently many (feature) points located on a hypothetical decision boundary, \( c_i \) can determine if such decision boundary is the result of one of its member classifiers and provide other more detailed information about the specific classifier. For example, for the deconstructor associated with SVM, with enough feature points located on a hypothetical decision boundary in \( \mathbb{R}^n \), it can determine if the decision boundary is the result of an SVM classifier and if so, it will return the type of kernel and the number of support vectors, etc. The number of required points on the decision boundary depends on each deconstructor algorithm.

We have assumed that the feature spaces are all continuous (\( \mathbb{R}^n \)) and the feature transforms \( f_i \) are surjective maps. Technically, working in continuous domains is simpler because useful differential-geometric features such as the normal vectors of the classifier's decision boundary are available. Furthermore, continuous domains allow us to locate the decision boundary within any prescribed accuracy using the simple idea of bracketing (as in root-finding [30]): given a pair of positive and negative images (PN-pair), we can produce a PN-pair of images near the decision boundary in the image space by successively halving the interval between a pair of positive and negative images, using the labels provided by \( \mathcal{C} \). By Lipschitz continuity, a PN-pair (sufficiently) near the decision boundary in the image space can be transformed by a feature \( f_i \in \mathcal{F} \) into a PN-pair near the decision boundary in the feature space \( \mathbb{R}^n \). By sampling enough PN-pairs that are near the decision boundary in the image space, we obtain the corresponding PN-pairs near the decision boundary in each

---

\(^\dagger\) For simplicity, we assume that the transform \( f_i \) is surjective and as a set (of images), \( f_i^{-1}(\nu_i) \) is nonempty and we can compute an element (image) in \( f_i^{-1}(\nu_i) \)(e.g., [51]).
feature space $\mathbb{R}^n$, specified by the feature $f_i \in \mathcal{F}$. For these sampled PN-pairs in each feature space $\mathbb{R}^n$, we apply the deconstructor algorithm $c$ to see if it recognizes the decision boundary from these samples. Furthermore, the inverse feature transform $f_i^{-1}$ permits the deconstructor algorithm that operates in the (opaque) feature space $\mathbb{R}^n$ to sample additional features near the decision boundary if necessary. Essentially, each deconstructor algorithm is designed to probe a given feature space for the decision boundary and it recognizes the parametric form of the decision boundary arising from a classifier in its associated family. In particular, starting with a small number of positive features, the deconstructor algorithm proceeds to explore each feature space $\mathbb{R}^n$ by generating points near the decision boundary.

Given the two lists $\mathcal{F}, \mathcal{C}$, the deconstruction process proceeds in a direct manner: run all deconstructor algorithms in $\mathcal{C}$ in parallel over all the candidate features in $\mathcal{F}$, and for each pair $(f_i, c_j)$ of feature (space) and deconstructor, $c_j$ either succeeds in detecting a recognizable boundary or fails to do so. If there is no successful pairs $(f_i, c_j)$, the algorithm then fails to deconstruct $\mathcal{C}$. Otherwise, it provides the user with all the successful pairs $(f_i, c_j)$ as potential candidates for further investigation\(^2\). In this dissertation, the classifier list $\mathcal{C}$ contains two elements: the family of support vector machines (SVM) and the family of cascades of linear classifiers, and in the following three subsections, we provide the remaining details in the proposed deconstruction process.

### 4.2.1 Feature Identifiers

An important first step in the deconstruction process is to properly define the lists $\mathcal{F}$ and $\mathcal{C}$, with the aim of making them as short as possible. Various heuristics based on exploiting the differences between various types of features can be developed to accomplish this goal. For example, it is possible to exclude simple HOG-based

---

\(^2\) This part is beyond the scope of this dissertation.
features from \( \mathcal{F} \) by perturbing the images slightly. In particular, because HOG (unlike SIFT) is in general not rotationally invariant, and hence, if the images are slightly rotated, we can expect less-than-stable results from the classifier \( \mathcal{C} \) if it uses HOG as the main feature. In the experimental section, we study several such feature identifiers in the form of simple image operations such as rotations and scalings, and demonstrate their usefulness in excluding features in \( \mathcal{F} \). On the other hand, the difference between the classifiers in \( \mathcal{C} \) can also be exploited to shorten the list \( \mathcal{C} \). In our case, it is straightforward to determine if the given \( \mathcal{C} \) uses SVM or a cascade of linear classifiers as its internal classifier. Recall that a cascade of linear classifiers is a decision tree with a linear classifier associated with each tree node. Consequently, positive features always take longer to process than negative features and among the negative features, the running time can vary considerably depending on the depth of the tree. However, for an SVM-based \( \mathcal{C} \), all features are expected to have the same or similar running times. Therefore, by checking the distribution of the running time among positive and negative features, we can determine with great certainty the type of classifier used internally by \( \mathcal{C} \).

### 4.2.2 Deconstructor Methods

Given a feature space \( \mathbb{R}^d \), we can find transform our samples from image space to the feature space and perform bracketing. Generating many PN-pairs near the decision boundary we can use methods detailed in chapters chapter 2 and chapter 3 to recover different classifiers. Repeating this process for all the feature spaces available in \( \mathcal{F} \) we build our classifier list \( \mathcal{C} \).

### 4.2.3 Geometric Compatibility Between Features and Deconstructors

In the deconstruction process, the interaction between the features in \( \mathcal{F} \) and deconstructors in \( \mathcal{C} \) is based on the notion of geometry compatibility, and the deconstruction algorithm selects the pair \((f, c)\) as a solution if the deconstructor algorithm \( c \) recognizes the decision boundary from a collection of sampled points in the feature space \( \mathbb{R}^n \).
specified by $f$. The geometric picture is neatly captured by the following diagram:

\[
\begin{array}{cc}
\mathbb{R}^d & \mathbb{R}^n_2 \\
\downarrow^{\cong} & \uparrow^{\pi} \\
\mathbb{R}^d & \mathbb{R}^n_1
\end{array}
\]

Suppose $f_1, f_2$ are two features in $\mathcal{F}$ with their respective feature spaces $\mathbb{R}^n_1, \mathbb{R}^n_2$, and the vision classifier $C$ internally employs the pair $(f_1, c)$. Therefore, if we reconstruct the decision boundary in the correct feature space $\mathbb{R}^n_1$, the deconstructor algorithm would be able to recognize the decision boundary and hence the pair $(f_1, c)$ would be selected by the deconstruction algorithm. However, for the incorrect feature $f_2$, the decision boundary reconstructed in $\mathbb{R}^n_2$ is related to the decision boundary reconstructed in $\mathbb{R}^n_1$ via a map $\pi$ that arises from the fact that we use the same set of images in the image space $\mathbb{R}^d$ to reconstruct the decision boundary in both $\mathbb{R}^n_1$ and $\mathbb{R}^n_2$. The important observation (or assumption) is that for different features $f_1, f_2$, the map $\pi$ is generally nonlinear and it would map the decision boundary in $\mathbb{R}^n_1$ to a decision boundary $\mathbb{R}^n_2$ with an unknown parametric form. For example (as will be shown later), for a linear decision boundary in $\mathbb{R}^n_1$, the corresponding decision boundary in $\mathbb{R}^n_2$ would generally be nonlinear. Therefore, if $c$ is a deconstructor for linear classifier, it would fail to recognize the decision boundary in $\mathbb{R}^n_2$. For SVM deconstructor, the map $\pi$ essentially maps a decision boundary of a known parametric form to a boundary with unknown parametric form, i.e., the decision boundary in $\mathbb{R}^n_2$ is not compatible with the deconstructor $c$.

### 4.3 Experiments

In this section, we four three experimental results. In the first experiment, we demonstrate the idea of using simple heuristics (image operations) to shorten the feature list $\mathcal{F}$. In the second experiment and third one, we detail the experimental result of deconstructing the pedestrian (human) detector in the OpenCV library and aioplane detector (trained by us). In fourth and last experiment we show how to discover the
feature space when linear dimensionality reduction is used as feature transform and then we show results on deconstruction of quadratic kernel based SVM.

4.3.1 Distinguish between HOG and SIFT

Many vision algorithms use features derived from the well-known gradient-based features such as HOG [19] or SIFT[38]. To shorten the search list $\mathcal{F}$, we use various invariance properties of these features. For examples, with non-dense SIFT used in the bag of words model, it is generally invariant under scale, rotation and even shifting transformation. On the other hand, the dense SIFT, when used in a pyramid scheme, is generally invariant under reasonable amount of scale change, image flipping and small amount of translations. It is generally not invariant under rotation or flipping (unless object is symmetric). HOG as mentioned previously in not invariant under rotation, although it is invariant under small amount of translation when it is smaller than size of its cells. In this experiment, we experimentally demonstrate the above general impression on the invariance property of the HOG and SIFT under various image transforms. We compared these properties of the four different type features (Table 4-1) by constructing aeroplane SVM classifiers using the images from Caltech 101 dataset [26]. We randomly selected 100 aeroplane images as positive samples and 100 images from other categories as negative samples. We used four type features extracted from these samples to train linear SVMs. In the test phase, three simple image transforms are applied to the 200 randomly selected test images. The transformations are $180^\circ$ rotation, a translation of eight pixels in both $x$ and $y$ directions, and a simple zoom in (achieved by scaling the by a factor of 1.2 and cropping the boundaries). The classification rates of the SVMs constructed using the four different types of features and under the three transforms are demonstrated in Table 4-1. As shown in the table, rotational invariance of SIFT make it relatively stable under rotation and scale transforms. We use these invariance results to decrease out feature list during our experiments.
Table 4-1. Effects of different image transforms on classification results. HOG-based features as expected produce unstable results under rotation and scale change.

<table>
<thead>
<tr>
<th>Transform</th>
<th>SIFT + BoW</th>
<th>Dense-SIFT + Spatial Pyramid</th>
<th>HOG(4)</th>
<th>HOG(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation(180°)</td>
<td>0.8000</td>
<td>0.4300</td>
<td>0.6800</td>
<td>0.7450</td>
</tr>
<tr>
<td>Zoom-in</td>
<td>0.9950</td>
<td>0.9600</td>
<td>0.3550</td>
<td>0.3850</td>
</tr>
<tr>
<td>Translation (8,8)</td>
<td>0.9550</td>
<td>0.8500</td>
<td>0.9550</td>
<td>0.9350</td>
</tr>
</tbody>
</table>

4.3.2 Deconstructing OpenCV HOG-based Pedestrian Detector

In this experiment, we deconstruct the pedestrian (human) detector provided in the OpenCV library [13]. This implementation is based on the algorithm proposed in [19], where histogram of oriented gradients (HOG) is used as feature with linear (SVM) classifier as internal feature classifier. The goal of this deconstruction experiment is to recover (a) The three important design parameters for the HOG feature: cell size, block size and block stride. (b) The parameters for the linear (SVM) classifier: its weight (normal) vector and bias.

For this experiment, a quick check on the running times of a few positive images immediately rules out the cascade of linear classifiers as a viable candidate (Section 3.1) or more precisely, it shows that the cascade must be a very shallow tree and we simply interpret the result as ruling out the cascade as a candidate. Furthermore, because the high-dimensionality of the feature space (usually in the thousands), the SVM-based classifier is almost certainly linear, since other types of nonlinear kernels are often computationally demanding for high-dimensional features. In particular, by checking the normal vectors of the decision boundary at forty different places, it essentially provides only one normal vector, indicating the underlying linear classifier. Therefore, the classifier list \( C \) has only one element and the kernel type is assumed to be linear. For the feature list \( F \), we define approximately 30 candidate parameter settings and accordingly, the feature list \( F \) has length 100. More specifically, the cell size can take the three integral values \( \{4, 8, 16\} \), and for each cell size, the block size can equal to cell size, double the cell size or triple the cell size. Similarly, the block stride
is set either to half of the block size, the full block size or twice the block size. We note that different parameters give different HOG features that typically reside in different feature spaces (in particular, with different dimension). Note that since we randomly pair positive and negative images, PN-pair set can have size equal to product of size of positive image set and negative image set. This restricts which feature dimensions we can take in consideration. We use the classifier as provided in the OpenCV without any modification and positive and negative images are obtained by running the classifier over a set of images. We remark that the positive and negative images are according to the outputs of the classifier $C$, not visually which class they should belong to. We randomly pair these positive and negative images and run the bracketing algorithm to locate PN-pairs (Figure 4-2, 4-3) close to the decision boundary in the (fixed) image space, and for each such PN-pair close to the decision boundary in the image space, we compute its corresponding PN-pair in each of the feature space $\mathbb{R}^{n_f}$ for $f_i \in \mathcal{F}$. In this experiment, we do not use the inverse feature transform to determine the feature labels in the feature space, although inverse transform for HOG-based features has been proposed in [51].

Figure 4-2. PN-pairs obtained by bracketing process, during deconstructing OpenCV person detector. Notice that even when they appear to be very similar, one is labeled as positive sample by classifier and other as negative.

An important experimental result is that linear decision boundary is observed only for the correct parameter setting, while for incorrect parameter setting, the decision boundary is generally nonlinear. To efficiently and accurate detect the linear boundary
in these high-dimensional feature spaces, we use Fisher linear discriminant (FLD). Specifically, for the labeled PN-pairs in each candidate feature space, we train a Fisher linear discriminant. If the labeled features are indeed linearly separable, the Fisher linear discriminant would detect it by correctly classifying large portion of the label features. Furthermore, in the right feature space, as we generate more PN-pairs that are close to true decision boundary, the linear classifier provided by FLD will move closer to the true linear classifier. This latter statement is easily visualized and its rigorous justification seems straightforward. In particular, the linear classifier determined by FLD provides good approximations to the weight (normal) vector and the bias of the true linear classifier. On the other hand, in the wrong feature space, the decision boundary would be nonlinear and the trained FLD is not expected to correctly classify large portion of the labeled features, regardless the number of PN-pairs generated. Experimental confirmations of these observations are shown in Figures 4-4 and 4-5. In Figure 4-4, the classification error of the trained FLD decreases only in the correct feature space. And in Figure 4-5, 2D projections of the labeled PN-pairs in three different feature spaces are displayed and linear separability is clearly shown only for features from the correct feature space. The weight (normal) vector recovered using FLD with 10,000 PN-pairs has the normalization correlation of 0.99 with the ground truth, i.e., with an angular difference of roughly 2°.
Figure 4-4. Classification errors exemplifying concept of geometric compatibility between features and deconstrcutors. *LEFT: Deconstruction of OpenCv Pedestrian Detector:* Classification errors for the FLD trained in five different feature spaces. The feature parameters are given in the legend (cell size, block size) and the visible decrease in classification rates is observed only in the correct feature space. *RIGHT: Deconstruction of airplane Detector:* Similar results obtained in the deconstruction of an airplane detector (HOG features + linear SVM.).

4.3.3 Deconstructing Airplane Detector

To further test algorithm we constructed classifier for labelling airplane (Caltech 101 [26]). HOG features were calculated (with cell-size 16, block-size 2 × 2 cell-size, stride equal to cell size) for training images and linear SVM was trained on these features as classifier. This two-stage classification system was provided as black box to our deconstruction algorithm. We follow all the steps described in experiment of deconstructing OpenCV pedestrian detector, however this time we also vary number of orientations (number of bins used) to construct HOG. As indicated in Figure 4-4 separation between $PN - pairs$ increases (hence error decreases) for correct parameters as sample size is increased. In Figure 4-6 we show recovered linear classifier and the original linear classifier.

4.3.4 Deconstructing when Dimensionality Reduction is Being Employed

Computer vision problems are marked by the high dimensionality of data. Dimensionality reduction is commonly used in machine learning tasks (including
Figure 4-5. 2D projections of the labeled PN-pairs in three different feature spaces, exemplifying concept of geometric compatibility between features and deconstructors. The 2D projected subspace is spanned by the normal vector determined by FLD and the first singular vector for the collection of features. The results from the correct feature space are shown in last row, the two plots display the projections in two different scales (for further details refer to the section 4.3.2)

computer vision problems) to not only reduce complexity of data and make processing computationally less expansive, but also as a feature extraction. In following experiment we explore how to recover subspace being used for dimensionality reduction in a given classifier.

We train SVM based classifier for the MNIST dataset having following two stages. First stage consists of linear data transformation (principal component analysis) used to
extract features (we use 10 eigenvectors corresponding to 10 largest eigen values calculated on of covariance matrix of training data). In second stage, SVM with quadratic kernel is used to classify the extracted features. As in previous cases this classifier appears as a black-box to us and we can only get classification result once we input the image.

First step in this experiment is to recover the subspace on which the SVM classifier works. Using bracketing procedure we extract PN-pairs and normals on these locations. Since these normals (let their set be called $N$) are calculated on the decision boundary and our decision boundary should exist in the subspace; these normals should provide us information about subspace. We perform Singular Value Decomposition (SVD) to decompose $N = U D V^T$). Let $s$ be diagonal elements of $D$, then the difference between the elements of $s$ (singular values) should give us estimate of the dimensionality $k$ of subspace. Figure 4-7 shows singular values approaching zero and becoming very similar to each other after the $k$.

Once the $k$ is determined we collect first $k$ columns of $U$ to represent subspace $S$. Once the subspace is recovered we can now proceed to recover the kernel type of the classifier. We follow the procedure, as detailed in chapter 2, with a small variation. In present scenario the decision boundary is to be checked in the feature space where as the classifier-black-box only works in the image space. Fortunately due to the linear nature of transformation used for dimensionality reduction we can transform points...
Figure 4-7. Singular Values of the normal matrix. The singular values are sorted in decreasing order and the figure displays the singular values starting from the ninth singular value. The plot highlights the substantial gap between the ninth and the tenth singular values and the absence of visible gaps for the remaining singular values.

Figure 4-8. Identifying Kernel Type: Histogram shows voting done by fitting curves on the intersections of 2D plane with the decision boundary. The quadratic polynomial kernel is clearly most voted and correct result.

on the 2D plane from feature space to the image space and calculate their labels. Change of the label, as we move from positive element to negative element, indicates intersection of 2D planes with the decision boundary. We sample many such curves and vote for the type of curve it could be. Figure 4-8 shows our result. Once the correct type is found we continue to recover the Quasi-Support Vectors as indicated in the chapter 2. NOTE: use of linear SVM in such setup results into a degenerate case and correct subspace cannot be recovered.
CHAPTER 5
CONCLUSIONS AND FUTURE WORK

We have proposed a new type of learning problem, termed Deconstructive Learning, whose objective is to learn a given (binary) classifier by characterizing as much as possible the full extent of its capability. We have introduced and describe a two-component design model for a broad class of classifiers used in computer vision. In this two-component design model, the classifier first transforms the input image into a feature in the feature space and the classification result is obtained by applying a feature-space classifier to the transformed feature. We have proposed a deconstruction algorithm that is tailored to classifiers with such two-component design, with the deconstruction algorithm aiming to identify, for a given classifier, its internal feature transform and feature-space classifier from a list of potential candidates. We have introduced the notion of geometric feature-classifier compatibility as the criterion for selecting the deconstruction output and we have also presented algorithms for deconstructing two broad classes of classifiers: cascades of linear classifiers and support vector machines. In particular, for SVM-based classifiers, our method can determine the kernel type and the dimension of kernel subspace (spanned by the support vectors) and recover the subspace itself. Although we were not able to recover exact support vectors for the general SVMs, we can recover quasi-support vectors for SVMs using polynomial kernels. For a cascade of linear classifiers, we are able to determine the (weak) classifiers themselves and their locations in the cascade. Multiple experiments have demonstrated the validity and viability of the proposed approach, and successful and practical applications of our method include the deconstruction of pedestrian and face detectors in the OpenCV library.

Work is currently ongoing to expand the feature and classifier lists $F$, $C$ to include structured SVM [47], dictionary-learning-based classifier [39], bag of words features [17] and others. Recently, neural network-based classifiers such as those emerged from
the deep learning community [9, 31] has gained considerable interest and popularity. At the first glance, the direct approach proposed in this dissertation seems insufficient and inadequate for deconstructing this type of classifiers. However, we believe that more indirect approaches and formulations are possible, with the aim of identifying crucial information that are necessary for the deconstruction process. Additionally, to further improve the applicability and practicality of our method, it is necessary to develop the concept of Feature Class Signature. The signature refers to a set of image operations which, if performed on input images, can help determining feature-space classifier. One possible way to define such a signature is to build a decision tree that can generate the “signature” for the given classifier, and much like the signatures generated by the virus-detection software, this signature list can be kept in the database and to be compared with whenever a new classification system is provided.

**The Case and Outlook For Deconstructive Learning:** I conclude this dissertation with a brief discussion on the potential usefulness of deconstructive learning, providing several examples that illustrate its significance in terms of its future prospects for theoretical development as well as practical applications. The geometric approach taken in this dissertation shares some visible similarities with low-dimensional reconstruction problems studied in computational geometry [21], and in fact, it is partially inspired by various 3D surface reconstruction algorithms studied in computational geometry (and computer vision) [23] [33]. However, due to the high dimensionality of the feature space, deconstructive learning offers a brand new setting that is qualitatively different from those low-dimensional spaces studied in computational geometry and various branches of geometry in mathematics. High dimensionality of the feature space has been a hallmark of machine learning, a realm that has not be actively explored by geometers, mainly for the lack of interesting examples and motivation. Perhaps deconstructive learning’s emphasis on the geometry of the decision boundary in high dimensional space and its connection with machine learning could provide stimulating examples or
even counterexamples unbeknown to the geometers, and therefore, provide the needed motivation for the development of new type of high-dimensional geometry [15].

On a more practical side, image-based classifications such as face, pedestrian and various object detections and scene recognition are important computer vision applications that have begun to have visible and noticeable impact in our daily life. Indeed, with the current trend in technology development, it is not difficult to envision a not-so-distant future in which the world is partially powered by such applications. In the backdrop of such futuristic vision, deconstructive learning points to an interesting and uncharted territory, perhaps a promising new direction with potential for generating important impacts. Several potential consequences of this new capability are interesting to ponder. Our belief is supported by the increasing presence and influence of AI/ML systems in daily life and by the fundamental asymmetry between constructive learning (learning classifiers from training samples) and deconstructive learning. Computationally, the former is essentially serial while the latter is parallel in nature. More specifically, deconstruction is a highly parallelizable process, and in our algorithm, the processes of sampling points on the decision boundary are completely independent and parallelizable. However, for constructive learning, parallelization is often ineffective because the immensity of the search space, and often, the only viable solutions are local greedy approaches, which are invariably serial. A case in point is the OpenCV’s HOG-based pedestrian detector. To develop such an application, the designers must have spent weeks if not months of effort in, among other things, gathering useful training images, managing other often time-consuming logistic matters and tuning both the feature parameters (cell/block/bin sizes) and the learning algorithm in order to obtain the best (linear) classifier. However, as demonstrated in this dissertation, the detector can be completely deconstructed in a few hours and the user of the deconstruction algorithm only requires to collect a few positive images to start the deconstruction process, since the negative images can be obtained randomly (with labels provided by the classifier
The result is certainly not surprising since it basically mirrors the well-known fact that finding a solution is always more time-consuming than checking the solution. However, its implications are multiple and perhaps profound. For example, the result of months or even years of hard work can be deconstructed in a matter of a few hours, even when it is hidden under the seemingly impenetrable binary codes. Additionally, we believe that deconstructive learning could provide greater flexibility to the users of AI/machine learning products in the future because it allows the users to determine the full extent of an AI/ML program/system, and therefore, create his/her own adaptation or modification of the given system for specific and specialized tasks. For example, how would an reviewer for any machine learning journal know that a submitted binary code of a paper really does implement the algorithm proposed in the paper, not some clever implementation of some known algorithm? Deconstructive learning proposed in this dissertation offers a possible solution by explicitly deconstructing the submitted code. It provides the opportunity to update (e.g. updating existing SVM with new training data of new instances or even new classes), speed up (e.g. moving filter operations from CPU to GPU) and learning what parameters work in the given classification system so that the same parameters could be used in other systems.
REFERENCES


   URL http://www.google.com/about/careers/lifeatgoogle/
      self-driving-car-test-steve-mahan.html


    A geometric framework for learning from labeled and unlabeled examples.” The

    100 frames per second.” CVPR. 2012.


[10] Bosch, Anna, Zisserman, Andrew, and Munoz, Xavier. “Scene classification via


BIOGRAPHICAL SKETCH

Mohsen Ali received his Ph.D. from University of Florida in spring 2014. He was awarded Fulbright scholarship by International Institute of Education for his PhD studies. Prior to that he completed his pre-doctoral graduate studies at Lahore University of Management Sciences and Punjab University (Lahore), in computer science. His research interests include computer vision and machine learning.