Deconstructing Binary Classifiers in Computer Vision

Mohsen Ali, Jeffrey Ho

Information Technology University, Pakistan; University of Florida, US

Abstract. This paper further develops the novel notion of deconstructive learning and proposes a practical model for deconstructing a broad class of binary classifiers commonly used in vision applications. Specifically, the problem studied in this paper is: 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, we propose a novel framework that explicitly identifies and delineates the computer vision and machine learning components, and we propose an effective deconstruction algorithm for deconstructing binary classifiers with the typical two-component design that employ support vector machine or cascade of linear classifiers as their internal feature classifiers. The deconstruction algorithm simultaneously searches over a collection of candidate feature spaces by probing the spaces for the decision boundaries, using the labels provided by the given classifier. In particular, we demonstrate that it is possible to ascertain the type of kernel function used by the classifier and the number of support vectors (and the subspace spanned by the support vectors) using only image queries and ascertain the unknown feature space too. Furthermore, again using only simple image queries, we are able to completely deconstruct OpenCV's pedestrian detector, ascertain the exact feature used, the type of classifier employed and recover the (almost) exact linear classifier.

1 Introduction

This paper further develops on the notion of deconstructive learning [1] and proposes a practical model for deconstructing a broad family of binary classifiers (e.g., object detectors) in computer vision. While the ultimate objective of all types of learning in computer vision is the determination of classifiers from labeled training data, for deconstructive learning, the objects of interest are the classifiers themselves. As its name suggests, the goal of deconstructive learning is to deconstruct a given binary classifier **C** by determining and characterizing (as much as possible) the full extent of its capability, revealing all of its power, subtlety and limitation. As an example, 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 ± 1 as its decision for each image. The classifier is laconic in the sense that except the predicted label ± 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 paper 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 paper 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 (see Figure 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., [2–4]) conform to this particular design, with other lesser-known but equally-important examples in scene classification, object recognition and others (e.g., [5, 6]) 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 $\mathbf{C}$ 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 $\mathbf{I}_1, ..., \mathbf{I}_n$, 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 (see Figure 1). 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 paper, we 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 [4]. In the experimental section, we present three preliminary experimental results demonstrating the viability of the ideas proposed in this paper. 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,

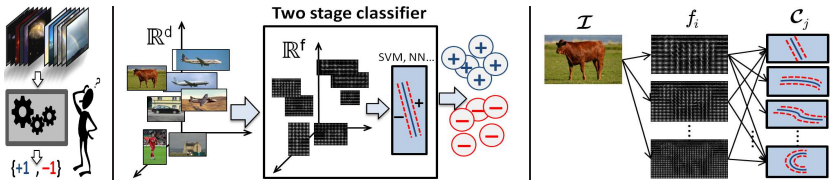

Fig. 1. Left: Schematic illustration of deconstructive learning. **Center:** Two-component design of a classifier: a feature-transform component provided by computer vision followed by a feature-classification component furnished by machine learning. **Right:** Schematic illustration of the proposed 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.

allow for a more efficient deconstruction process. In the second experiment, we show, using MNIST dataset, how the type of kernel functions used in a support vector machine and the number of support vectors can be determined using only image queries. In the third and final 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°). 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.

Related Work To the best of our knowledge, there is not a previous work on deconstructive learning comparable to the one outlined above. However, [7] studied the problem of deconstructing linear classifiers in a context that is quite different from ours. Since only single linear classifiers are studied and there is no explicit considerations of the interaction between feature transforms and classifiers, their scope is considerably narrower than ours. Active learning (e.g., [8] [9] [10]) shares certain similarities with deconstructive learning (**DL**) in that it also has a notion of querying a source (classifier). 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. 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.

2 Deconstruction Process and Method

Let $\mathcal{F}, \mathcal{C}$ denote the feature and classifier lists. Given a classifier $\mathbf{C}$ with the two-component design as described above, the deconstruction algorithm attempts to identify the feature-transform and feature-classification components of $\mathbf{C}$ with the elements in $\mathcal{F}$ and $\mathcal{C}$. Specifically, we assume that

- Each feature $f_i \in \mathcal{F}$ defines a feature transform $f_i : \mathbb{R}^d \to \mathbb{R}^{n_i}$ from the image space $\mathbb{R}^d$ to a feature space $\mathbb{R}^{n_i}$ of dimension n_i. For technical reason, the feature transform $f_i : \mathbb{R}^d \to \mathbb{R}^{n_i}$ 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 $v_i \in \mathbb{R}^{n_i}$, an image in $f_i^{-1}(v_i)$ can be computed[1].
- 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 $\mathbb{R}^{n_i}$, 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 $\mathbb{R}^{n_i}$, 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 ($\mathbb{R}^{n_i}$) 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 [12]): 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 $\mathbf{C}$. 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 $\mathbb{R}^{n_i}$. 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 feature space $\mathbb{R}^{n_i}$ specified by the feature $f_i \in \mathcal{F}$. For these sampled PN-pairs in each feature space $\mathbb{R}^{n_i}$, 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 $\mathbb{R}^{n_i}$ to sample additional features near the decision boundary if necessary. Essentially,

[1] For simplicity, we assume that the transform f_i is surjective and as a set (of images), $f_i^{-1}(v_i)$ is nonempty and we can compute an element (image) in $f_i^{-1}(v_i)$(*e.g.*,[11]).

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 $\mathbb{R}^{n_i}$ 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_i either succeeds in detecting a recognizable boundary or fails to do so. If there are no successful pairs (f_i, c_j), the algorithm then fails to deconstruct $\mathbf{C}$. Otherwise, it provides the user with all the successful pairs (f_i, c_j) as potential candidates for further investigation[2]. In this paper, 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.

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 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 $\mathbf{C}$ 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 $\mathbf{C}$ 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 $\mathbf{C}$, 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 $\mathbf{C}$.

2.2 Deconstructor Methods

We summarize below prominent and important features of one the deconstructor algorithms.

[2] This part is beyond the scope of this paper.

Deconstructor for SVM Given a feature space $\mathbb{R}^l$ and sufficiently many PN-pairs near the decision boundary, the deconstructor algorithm is able to

- Determine the kernel type (assuming polynomial, exponential and hyperbolic tangent kernels).
- Determine the (kernel) subspace spanned by the support vectors. If the number m of support vectors is smaller than the feature space dimension l, then the dimension of the kernel subspace also gives the number of support vectors, assuming linear independent.
- For linear SVM (i.e., one single global linear classifier), $h(\mathbf{x}) = \mathbf{n}^\top \mathbf{x} + b$, the normal vector $\mathbf{n}$ and bias b can be determined.
- In theory, the number of sampled features required is in the order of $\mathbf{O}(ml)$.

The detailed analysis of the SVM deconstructor algorithm is presented in [1] and in the remaining section, we present the simplest case of polynomial kernels, demonstrating that the kernel type and the kernel subspace can be determined by simply querying the classifier $\mathbf{C}$. Recall that the general polynomial kernel of degree d has the form: for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^l$,

$$\mathbf{K}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + 1)^d, \tag{1}$$

and the decision function is given as

$$\mathbf{\Psi}(\mathbf{x}) = \omega_1 \mathbf{K}(\mathbf{x}, \mathbf{y}_1) + \cdots \omega_m \mathbf{K}(\mathbf{x}, \mathbf{y}_m), \tag{2}$$

where $\mathbf{y}_i, \omega_i$ are the support vectors and their weights, respectively, and the decision boundary $\mathbf{\Sigma}$ is defined by the equation $\mathbf{\Psi}(\mathbf{x}) = b$ for some $b \geq 0$. The reason that we can determine the kernel type (in this case, the degree d) is that the locus of the intersection of the decision boundary $\mathbf{\Sigma}$ with a two-dimensional affine subspace containing a point close to the decision boundary is (generically) a polynomial curve of degree d.

More specifically, let $\mathbf{x}_+, \mathbf{x}_-$ denote a PN-pair that is sufficiently close to $\mathbf{\Sigma}$. We can randomly generate a two-dimensional subspace containing $\mathbf{x}_+, \mathbf{x}_-$ by, for example, taking the subspace $\mathbf{A}$ containing $\mathbf{x}_+, \mathbf{x}_-$ and the origin in $\mathbb{R}^l$. For a generic two-dimensional subspace $\mathbf{A}$, its intersection with $\mathbf{\Sigma}$ is an one-dimensional curve, and the parametric form of this curve is determined by the (yet unknown) kernel function. This can be easily seem as follows: take $\mathbf{x}_+$ as the origin on $\mathbf{A}$ and choose an (arbitrary) pair of orthonormal vectors $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^l$ such that the triplet $\mathbf{x}_+, \mathbf{U}_1, \mathbf{U}_2$ identifies $\mathbf{A}$ with $\mathbb{R}^2$. Therefore, any point $p \in \mathbf{A}$ can be uniquely identified with a two-dimensional vector $\mathbf{p} = [\mathbf{p}_1, \mathbf{p}_2] \in \mathbb{R}^2$ as

$$p = \mathbf{x}_+ + \mathbf{p}_1 \mathbf{U}_1 + \mathbf{p}_2 \mathbf{U}_2. \tag{3}$$

If $\mathbf{p} \in \mathbf{A}$ is a point in the intersection of $\mathbf{A}$ with the decision boundary $\mathbf{\Psi}(\mathbf{p}) = \mathbf{b}$, we have

$$\sum_{i=1}^{l} w_i((\mathbf{x}_+^\top \mathbf{Y}_i + \mathbf{p}_1 \mathbf{U}_1^\top \mathbf{Y}_i + \mathbf{p}_2 \mathbf{U}_2^\top \mathbf{Y}_i) + 1)^d = \mathbf{b}, \tag{4}$$

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 9 points on the intersection of the Σ and $\mathbf{A}$.
- Fit a bivariate polynomial of degree d to the points. If the fitting error is sufficiently small, this gives a good indication that the polynomial kernel is indeed of degree d.

On the other hand, the reason that the kernel subspace $\mathbf{S_Y}$ spanned by the support vectors $\mathbf{y}_i$ can be recovered is the following well-known formula for the normal vector of the decision boundary at a point $\mathbf{x} \in \mathbf{\Sigma}$:

$$\mathbf{n}(\mathbf{x}) = \nabla \mathbf{\Psi}(\mathbf{x}) = \sum_{i=1}^{m} \alpha\, d\, \omega_i\, (\mathbf{x}^\top \mathbf{y}_i + 1)^{d-1}\, \mathbf{y}_i. \tag{5}$$

We remark that even though the support vectors $\mathbf{y}_i$ and weights ω_i are unknown, the above formula shows that the normal vectors of the decision boundary $\mathbf{\Sigma}$ span the kernel subspace $\mathbf{S_Y}$. Therefore, if sufficiently many points on $\mathbf{\Sigma}$ and their normal vectors can be determined, the kernel subspace can also be determined. For non-polynomial kernels, the corresponding formula for the normal vectors is more complicated and the algorithm to extract the kernel subspace from the sampled normal vectors requires a more elaborated convex optimization.

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:

$$\begin{array}{ccc} \mathbb{R}^d & \xrightarrow{f_2} & \mathbb{R}^{n_2} \\ \cong\downarrow & & \uparrow\pi \\ \mathbb{R}^d & \xrightarrow{f_1} & \mathbb{R}^{n_1} \end{array}$$

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 $\mathbf{C}$ 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 π 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 π 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 π 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.

3 Experiments

In this section, we present 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, we show that with real image data how a classifier employing polynomial kernel can be deconstructed. In the third experiment, we detail the experimental result of deconstructing the pedestrian (human) detector in the OpenCV library. More experimental results are presented in the supplemental material.

	SIFT + BoW	Dense-SIFT + Spatial Pyramid	HOG(4)	HOG(8)
Rotation(180°)	0.8000	0.4300	0.6800	0.7450
Zoom-in	0.9950	0.9600	0.3550	0.3850
Translation (8,8)	0.9550	0.8500	0.9550	0.9350

Table 1. Effects of different image transforms on classification results. HOG-based features as expected produce unstable results under rotation and scale change.

3.1 Distinguish between HOG and SIFT

Many vision algorithms use features derived from the well-known gradient-based features such as HOG [13] or SIFT[14]. 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 the 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 (see Table 1 and supplementary materials for further details) by constructing aeroplane SVM classifiers using the images from Caltech 101 dataset [15]. 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° 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 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

3.2 Deconstructing A Cubic Kernel Machine

In this experiment we trained a simple SVM classifier with cubic kernel using images from the MNIST dataset [2]. These images were not subjected to feature transform and the SVM was trained directly on the vectorized images. Using the method described earlier, we perform bracketing to locate pairs of positive and negative features (PN-pairs) that are close to the decision boundary. Given a PN-pair $\mathbf{x}_+, \mathbf{x}_-$, we randomly generate a two-dimensional subspace $\mathbf{A}$ containing these two points and compute the intersection of the decision boundary Σ with the subspace. We remark that this intersection can be easily computed by randomly generating points on the subspace $\mathbf{A}$ and applying bracketing on $\mathbf{A}$. Two typical intersections are shown in Fig. 2 and both examples display the "two humps" characteristic of cubic curves. Small fitting error, when approximating each curve with bivariate cubic polynomial, is a good indication that the underlying polynomial kernel is of degree three.

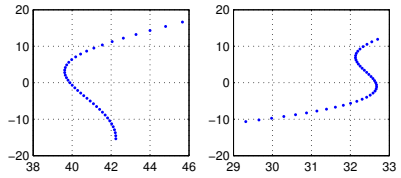

Fig. 2. Examples of the intersection of the decision boundary $\mathbf{\Sigma}$ with a 2−D subspace.

3.3 Deconstructing OpenCV HOG-based Pedestrian Detector

In this experiment, we deconstruct the pedestrian (human) detector provided in the OpenCV library [16]. This implementation is based on the algorithm proposed in [13], 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 (see 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 $\mathcal{C}$ has only one element and the kernel type is assumed to be linear. For the feature list $\mathcal{F}$, we define approximately 30 candidate parameter settings and accordingly, the feature list $\mathcal{F}$ 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 $\mathbf{C}$, 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 (Fig. 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 $\mathbb{R}^{n_i}$ 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 [11].

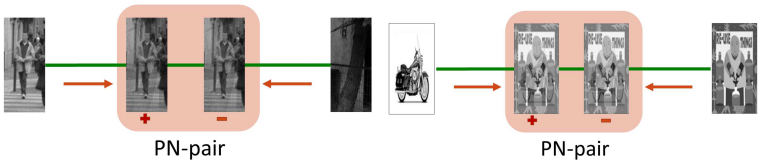

Fig. 3. Example showing PN pair recovered as the process of the bracketing. Notice that even when they appear to be very similar, one is labeled as positive sample by OpenCV person classifier and other as negative. A similar behavior is visible in the right figure where we trained HOG based SVM classifier for Motorcycle class in Caltech101 dataset [15].

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

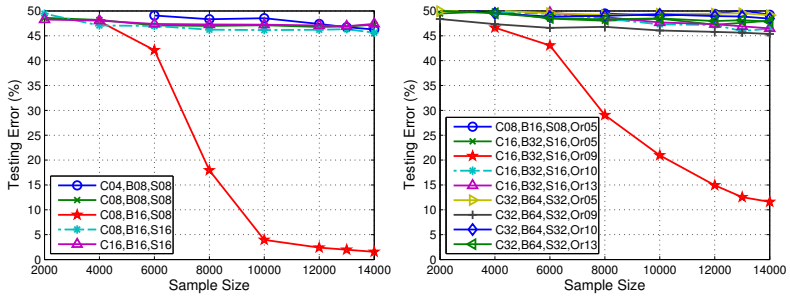

Fig. 4. LEFT: Deconstruction of OpenCV Pedestrian Detector: Classification errors for the FLD trained in five different feature spaces. The feature parameters are given in the legend (cell size, block size) and the visible decrease in classification rates is observed only in the correct feature space. **RIGHT: Deconstruction of airplane Detector**: Number of Orientations (number of bins) is also varied in this experiment, as indicated by the plots correct number of bins quite strongly impact the results

in Figures 4 and 5. In Fig. 4, the classification error of the trained FLD decreases only in the correct feature space. And in Fig. 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 $2°$.

4 The Case and Outlook For Deconstructive Learning

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

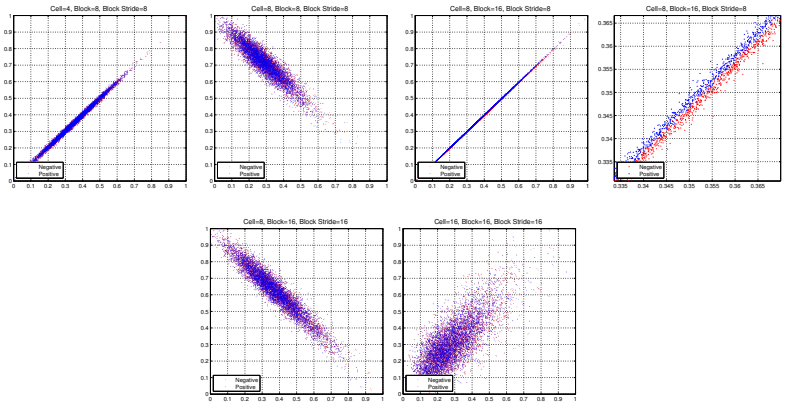

Fig. 5. 2D projections of the labeled PN-pairs in three different feature spaces. The 2-D projected subspace is spanned by the normal vector determined by FLD and the first singular vector for the collection of features. The results from the correct feature space is shown as the second pair, the two plots display the projections in two different scales.

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. 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 paper, 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 $\mathbf{C}$). 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. On a more practical side, 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

ACCV reviewer 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 paper offers a possible solution by explicitly deconstructing the submitted code. 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), we have found the problem of determining the secret of its inner working by simply querying it with images both fascinating and challenging, a problem with its peculiar elegance and charm.

In this paper, we have demonstrated the viability of deconstructive learning. Although the classifier list $\mathcal{C}$ consists of only two families, both SVM and linear cascades are widely-used classifiers in computer vision applications. Work is currently ongoing to expand the feature and classifier lists $\mathcal{F}, \mathcal{C}$ to include structured SVM [17], dictionary-learning-based classifier [18], bag of words features [19] and others. Recently, neural network-based classifiers such as those emerged from the deep learning community [20, 21] has gained considerable interest and popularity. At the first glance, the direct approach proposed in this paper 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.

5 Conclusions

We have proposed a novel framework for deconstructing binary classifiers that follow the common two-component design: a feature-transform component followed by a feature-classification component. Experimental results have confirmed both the viability and practicality of the proposed deconstruction algorithm. While much work remains for the future, the results in our paper perhaps serve as a small but substantial step towards a better understanding of deconstructive learning, hitherto an uncharted territory that seems ripe for further exploration.

Acknowledgement. We in particular would like to thank one of the reviewers for his/her helpful comment and enthusiasm for our paper.

References

1. Ali, M., Rushdi, M., Ho, J.: Deconstructing kernel machines. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I. (2014) 34–49
2. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86** (1998) 2278–2324
3. Benenson, R., Mathias, M., Timofte, R., Van Gool, L.: Pedestrian detection at 100 frames per second. In: CVPR. (2012)

4. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. Volume 1., IEEE (2001) I–511
5. Fei-Fei, L., Perona, P.: A bayesian hierarchical model for learning natural scene categories. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Volume 2., IEEE (2005) 524–531
6. Bosch, A., Zisserman, A., Munoz, X.: Scene classification via plsa. In: Computer Vision–ECCV 2006. Springer (2006) 517–530
7. Lowd, D., Meek, C.: Adversarial learning. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, ACM (2005) 641–647
8. Dasgupta, S.: Analysis of a greedy active learning strategy. In: In Advances in Neural Information Processing Systems. (2004)
9. Balcan, M., Beygelzimer, A., Langford, J.: Agnostic active learning. In: Proc. Int. Conf. Machine Learning (ICML). (2006)
10. Balcan, M., Broder, A., Zhang, T.: Margin-based active learning. Learning Theory (2007) 35–50
11. Vondrick, C., Khosla, A., Malisiewicz, T., Torralba, A.: Hoggles: Visualizing object detection features. In: Proc. Int. Conf. on Computer Vision. (2013)
12. Heath, M.: Scientific Computing. The McGraw-Hill Companies, Inc (2002)
13. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Volume 1., IEEE (2005) 886–893
14. Lowe, D.G.: Object recognition from local scale-invariant features. In: Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Volume 2., Ieee (1999) 1150–1157
15. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. Pattern Analysis and Machine Intelligence, IEEE Transactions on **28** (2006) 594–611
16. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
17. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: Proceedings of the twenty-first international conference on Machine learning, ACM (2004) 104
18. Mairal, J., Bach, F., Ponce, J., Sapiro, G., Zisserman, A.: Discriminative learned dictionaries for local image analysis. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE (2008) 1–8
19. Csurka, G., Dance, C., Fan, L., Willamowski, J., Bray, C.: Visual categorization with bags of keypoints. In: Workshop on statistical learning in computer vision, ECCV. Volume 1. (2004) 1–2
20. Bengio, Y.: Learning deep architectures for ai. Foundations and trends® in Machine Learning **2** (2009) 1–127
21. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18** (2006) 1527–1554