Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Master thesis

Semantic Structuring of Shapes and other Visual Data

submitted by

Daniel Mewes

submitted

August 27th 2013

Supervisor

Dr. Michael Wand

Advisor

Dr. Robert Herzog

Reviewers

Prof. Dr. Hans-Peter Seidel
Dr. Michael Wand

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement under Oath

I confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,..... .....

(Datum / Date) (Unterschrift / Signature)

I would like to thank everybody who has personally or professionally supported me while writing this thesis.

I would especially like to thank Michael Wand and Robert Herzog, with whom I had many fruitful and inspiring discussions and who have supported me in developing and implementing my ideas.

Further thanks go to the Saarbrücken Graduate School of Computer Science, the Max-Planck society and especially my parents for creating an appropriate environment and providing funding to make my studies and research possible.

Table of Contents

1 Introduction

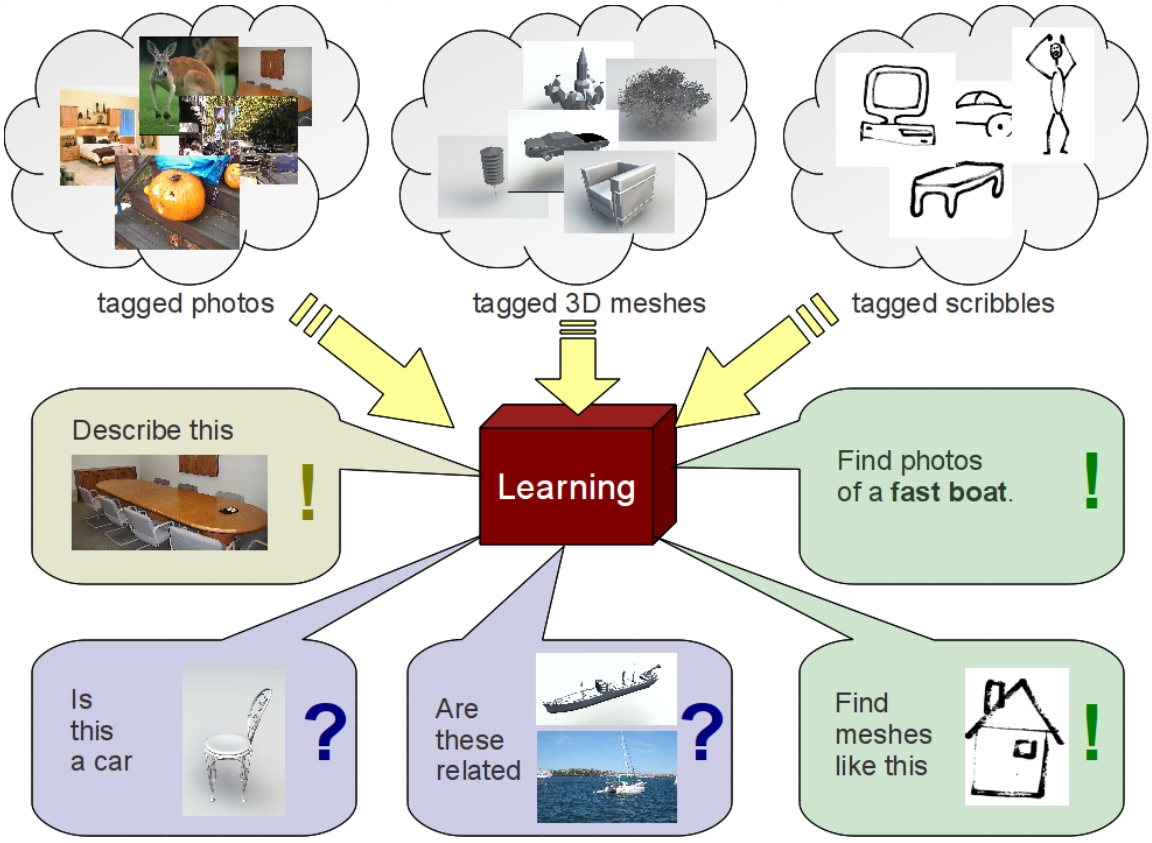


Illustration 1: Our goal: We are given collections of photos, 3D meshes, hand-drawn scribbles or other visual data. Each object in these collections is tagged with a set of text labels. We want to utilize machine learning in order to answer questions about the meaning and relation of objects, or to retrieve specific kinds of objects from a collection.

Digital visual data comes in a variety of forms. There are manually modeled 3D shapes, 3D scans of real-world scenes, two dimensional photographs and hand-drawn sketches. We as humans have an amazing capability at assigning meaning to such data. If we see a photograph, we can immediately tell which objects it is composed of. We can describe the photo by means of words, or by drawing a sketch capturing important aspects of the photo. If we are

given a digital 3D model together with the appropriate visualization software, we can easily understand its structure. We can segment it into symmetric parts, into semantic object parts, and separate interesting features of the model from less salient background regions.

It has been a long-going endeavor to teach these skills which we so often consider trivial to a computer. In the 2010 Iron Man 2 film by Marvel Comics, the super hero Tony Stark hands his computer a scaled-down model of a theme park. After the computer scans and digitalizes the model, Tony asks “How many buildings are there?”. The computer replies: “Shall I include the hot dog stands?”. This scene from a science fiction movie is a prime example of computers assigning semantic meanings and automatically structuring a shape.

In 1969, a group of researches at the university of Edinburgh built a simple robot called “Freddy” which task was to recognize objects it looked at through a video camera. Freddy already utilized machine learning and could be trained to recognize new objects. Its capabilities were limited though, as it could only detect relatively simple objects which were placed in isolation on a special table. The limited computational power available at the time also made the process slow (roughly 10 minutes to classify an object). Details on Freddy can be found in [Barrow et al. 1969]. Detecting individual objects in photographs composed of many different objects, with varying lighting conditions and taken from a number of different perspectives was out of reach at that time. Since then, research in the area of computer vision has made great progress in solving this problem. Other research has focused on locating symmetric (reappearing and/or similar) parts in 3D data, or on the retrieval of 3D shapes from a shape database based on a key word or scribble query.

The recent availability of crowd-sourced data bases such as Flickr for images or Google 3D Warehouse for 3D shapes makes access to vast amounts of visual data easier than ever before. These publicly available data bases consist of pictures and 3D meshes respectively uploaded by a large number of different users. While the images and shapes in these databases come annotated with user-provided semantic labels, those annotations are often noisy and inconsistent (see for example Illustration 11). Organizing these data sets by semantic criteria therefore remains a difficult task.

In this thesis, our goal is to develop a method which – based on a large amount of input data from one or multiple crowd-sourced data bases – can reliably answer questions involving the semantic meaning of the visual data. We look at existing techniques to approach this problem of semantic structuring with a special focus on 3D shapes. We suggest own extensions to these techniques, and evaluate their effects.

In chapter 2 we look at how shapes can be represented in a form which makes their interpretation by a computer easier. We then continue by looking at methods for establishing an association between semantic labels and shapes in chapter 3. Finally, we propose a method which enables the computer to derive an integrated notion of semantic structures across different forms of visual data, and evaluate the method on 3D shapes, 2D photographs and hand-drawn sketches in chapter 4.

Our contributions are as follows: We successfully applied the “WSABIE” method of [Weston et al. 2011] to 3D shapes. Originally, the WSABIE method was developed for and evaluated on photographs only. We quantitatively evaluated different aspects of this method and could demonstrate performance benefits compared to a commonly used baseline method (linear

Support Vector Machines). Furthermore, we compared different shape representations (descriptors) and could significantly improve over an existing standard descriptor by using our own extension of the two-dimensional HOG descriptor to 3D shapes. Finally, our method for the semantic structuring of multi-modal data shows promising results, and we present an application for interactively exploring a semantically structured space of visual data. We believe that our method can be easily expanded to cover even non-visual modalities such as sound or text, making it a very powerful tool for tasks which involve a wide range of multi-modal data.

2 Representing Objects

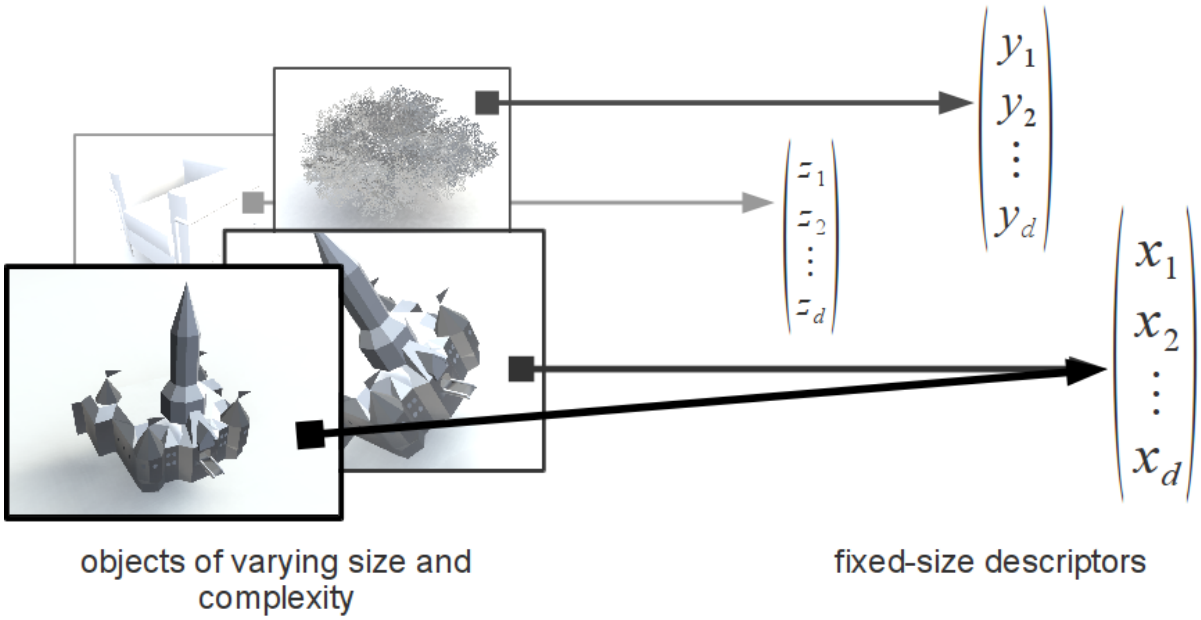


Illustration 2: Objects of varying size and complexity (here: 3D meshes) are converted into a fixed-length real descriptor vector. While some information is usually lost during this transformation, the fixed size representation allows for easier processing and can be designed to exhibit certain invariance properties (e.g. invariance against rotation and scaling). Meshes from Google 3D Warehouse.

Many machine learning algorithms require that observations are represented as vectors from a fixed vector space. Furthermore, the learning problem can be strongly simplified if we can obtain a representation which is invariant under variations that are irrelevant for the semantics of an observation, while still catching its most relevant properties. While it depends on the application which kinds of information are relevant and which should be ignored, there are some heuristics that often apply. For example if the task is to perform object recognition in photographs, it is usually desirable to obtain a representation that is as invariant as possible with respect to varying lighting conditions and different perspectives.

A descriptor is a function that takes input data and converts it into a fixed-length vector. We consider descriptors for 2D RGB images and for 3D shapes. RGB is a way to encode a color $c \in C$ as a three-dimensional vector, specifically $C := [0,1]^3$ where its components correspond

to the intensities of red, green and blue light frequencies respectively. A two dimensional RGB image is a partial function $i : \mathbb{R}^2 \to C$ defined on a rectangular area $[0, w] \times [0, h]$ where w and h correspond to the width and height of the image. A 3D shape is a 2-manifold $\Omega \subset \mathbb{R}^3$. Intuitively, the manifold property means that a 3D shape is a thin surface in the three dimensional Euclidean space.

An *image descriptor* is a function $D_{img} : P(\mathbb{R}^2 \to C) \to \mathbb{R}^d$ where $P(\circ)$ denotes the power set and $d \in \mathbb{N}$ is the dimensionality of the resulting descriptor vector. Analogously a *3D shape descriptor* is a function $D_{shape} : P(\mathbb{R}^3) \to \mathbb{R}^d$. In the following, we use the word *descriptor* to refer to both the descriptor function D_{img} or D_{shape} as well as to the vectors that result from applying this function to a given image or shape.

The general goal of a descriptor is to provide an efficiently computable mean for comparing two pieces of data with respect to the criteria that are relevant for the application at hand. Many descriptors are designed to have a specific component of it encode a specific characteristic of the underlying data. Quantitatively comparing the characteristics of the underlying data can then be accomplished by computing the Euclidean or a similar distance between the corresponding descriptor vectors.

The descriptors we look at describe local properties of geometry and images respectively. This means that in order to compute a descriptor, a center position in the shape has to be specified, and the descriptor will be a different one for different center positions. While local descriptors can be applied to a shape in total (e.g. by using the center of mass as the descriptor center in a 3D shape), we assume that they are applied to a number of local positions all over the shape and we later combine these local descriptors into a global descriptor for the object using the bag of features model (see 2.1.3).

2.1 Related Work

There are many different kinds of descriptors available. Among the most common ones for 2D images are the Scale Invariant Image Transform ("SIFT") [Lowe 1999], 2D spin image descriptors [Lazebnik et al. 2003] and Histogram of Oriented Gradient descriptors ("HOG" descriptor) [Dalal et al. 2005]. [Winder et al. 2007] propose a framework in which descriptors themselves can be learned. Their model is expressive enough to cover both SIFT and spin image descriptors among many others. We look at HOG descriptors in more detail in 2.1.1.

For 3D shapes, some of the 2D image descriptors have more or less direct counterparts. Spin images were originally proposed for 3D shapes by [Johnson et al. 1999]. Harmonic Shape Descriptors [Kazhdan et al. 2003] utilize a transformation to a Spherical Harmonics basis in order to derive a rotation invariant representation of local geometry. [Gatzke et al. 2005] propose a descriptor which is based on curvature values along intrinsic "fans" spreading from a given point. A descriptor that purely depends on intrinsic properties of the geometry

and is therefore invariant under some more complex extrinsic transformations of a mesh was proposed by [Tevs et al. 2011]. We take a closer look at Harmonic Shape Descriptors and also propose a way to adapt 2D HOG descriptors to 3D shapes.

In 2.1.3, we cover the "bag of features" technique which can be used to combine local descriptors into a global description of a shape or image and briefly mention more sophisticated models that can be built out of local descriptors.

2.1.1 HOG Image Descriptor

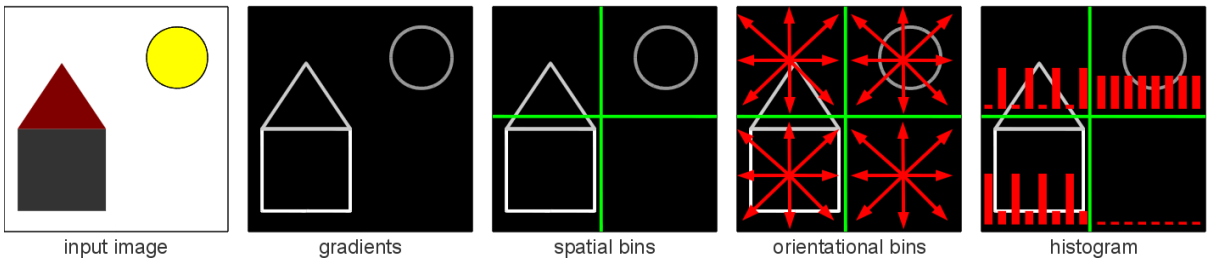


Illustration 3: Histogram of Oriented Gradients (HOG) descriptor: The local gradients of an image are assigned to a fixed number of spatial bins. Each spatial bin is further subdivided into multiple rotational bins, to which the gradients are assigned based on their orientation. The histogram that counts the number of local gradients assigned to a given bin form the descriptor. Note that while we apply the HOG descriptor globally to the whole image in this example, it is typically applied to a local neighborhood around a given center point only.

[Dalal et al. 2005] introduce a descriptor for images which they call the HOG descriptor (Histogram of Oriented Gradients). The idea behind HOG descriptors is shown in Illustration 3. Both the number of spatial and rotational bins can vary depending on the application. Furthermore, the rotational bins can either take the sign of the gradient into consideration and span a full 360 degrees, or ignore the sign, in which case they span just 180 degrees of the circle. [Dalal et al. 2005] achieved the best results in their application of detecting humans by using relatively fine rotational histogram bins (around 9) and relatively coarse positional histogram bins (3x3 in their case). They further added a "contrast-normalization" step to make the descriptor robust under both global and local changes in image contrast. By themselves, gradients are already invariant under additive changes of illumination. Thanks to the use of a histogram, small variations in both rotation and position can be eliminated. Note that HOG descriptors are still sensitive to large rotations. Specifically, changes in image perspective can have a huge impact on the descriptor. For photos, this is sometimes not a huge problem, as the upward direction is usually fixed, and the majority of photos are taken with a camera orientation almost parallel to the ground. From [Dalal et al. 2005]: "The subjects are always upright, but with some partial occlusions and a wide range of variations in pose, appearance, clothing, illumination and background. "

In chapter 4.3, we will apply HOG descriptors not only for representing photographs, but also for hand-drawn scribbles. Thanks to the configurability of HOG descriptors, we can adapt the descriptor to cope well with the specific properties of such drawings. In a work specifically aimed at working with scribbles, [Eitz et al. 2012] use Gabor-filter based descrip-

tors, which – similar to HOG features – decomposes the local image data based on the rotation of lines or edges. They also divide a local image patch into multiple "tiles", which is comparable to the spatial bins of the HOG descriptor. They found that having 4 different orientations (spread over a 180 degrees) and a 4x4 grid of tiles yielded the best overall results in their retrieval application. We use the same parameters for our HOG descriptor when representing scribbles.

2.1.2 Harmonic Shape Descriptor

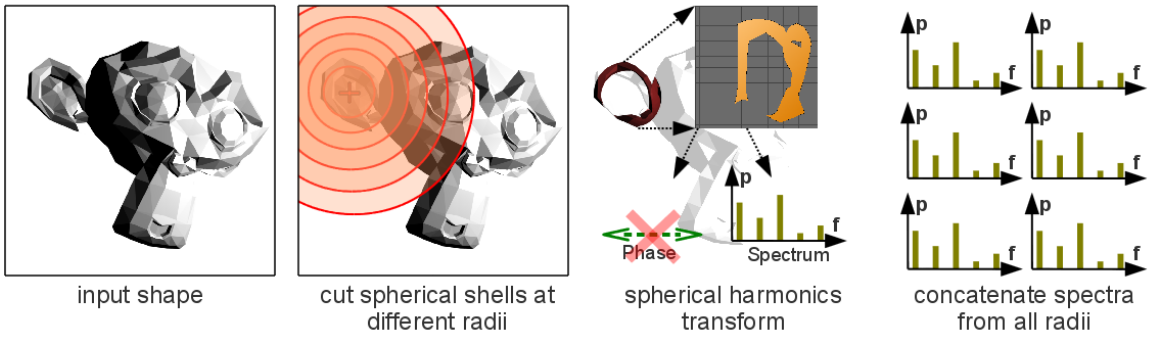


Illustration 4: Harmonic Shape Descriptors: Spherical shells at different radii are intersected with the input shape. The two-dimensional geometry on the surface of the shells is transformed into frequency and phase components by a spherical harmonics transform. The phase is discarded, and the frequency spectra from the different shells constitute the harmonic shape descriptor.

Harmonic Shape Descriptors ([Kazhdan et al. 2003]) represent a local volumetric interpretation of a shape in a rotation invariant manner. This is especially useful for 3D meshes, where the rotation of surface patches can vary widely. The descriptor is computed as follows: First, the mesh is converted into a volumetric shape V. In contrast to our earlier definition of a shape Ω as a thin 2-manifold in $\mathbb{R}^3$, $V \supset \Omega$ is constructed from Ω by adding a small ball around each point $c \in \Omega$. More precisely, $x \in V \Leftrightarrow x \in \Omega \vee \left(\exists c \in \Omega \text{ s.t. } x \in B_\epsilon(c)\right)$ where $B_\epsilon(c)$ is the ball of radius ϵ around the center c. Then, for a given point $x \in V$, a number of spherical shells of different radii around x are intersected with V. The volume intersecting with each shell is approximated as a linear combination of spherical harmonics basis functions. The spherical harmonics basis is similar to the Fourier basis, but represents a function on a three dimensional sphere. Only the energies of the different frequency components are pertained. The concatenated energy spectra from all of the shells yields the Harmonic Shape Descriptor. By removing phase infor-

Illustration 5: A rotation of the inner part of this airplane model yields the same harmonic shape descriptor as the unrotated version. This represents a case where the harmonic shape descriptor loses relevant information. Illustration from [Kazhdan et al. 2003].

mation and keeping only the energy distribution of the frequency components, the descriptor gains invariance against rotation. However, it also loses information about the relative phase between the frequency components. Another disadvantage of the HSD is that the relative orientation of the volumes across the shells of different radii is lost in the descriptor. Each shell can be rotated against the remaining shells without influencing the resulting descriptor. The effect is illustrated in Illustration 5.

2.1.3 Bag of Features Model

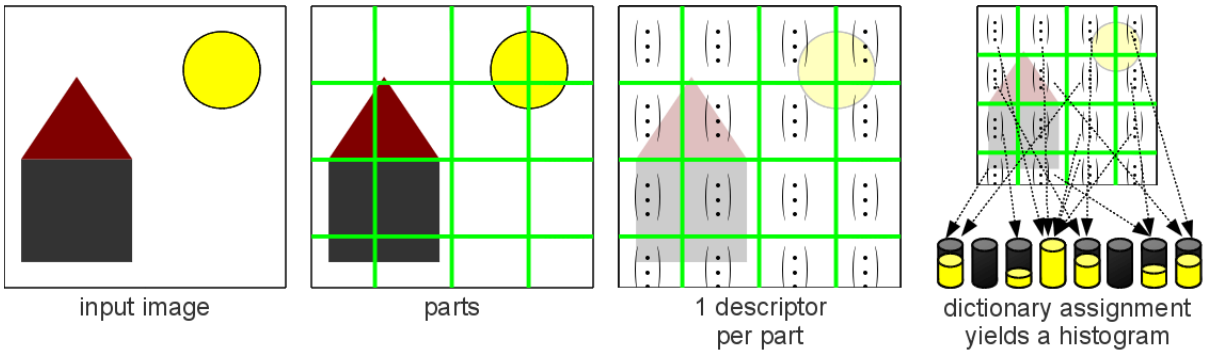


Illustration 6: Bag of features descriptor: The input data is partitioned into parts. A local descriptor is calculated for each part. The descriptors are then assigned to a dictionary bin. The number of assigned parts per bin yields a histogram which constitutes the bag of features descriptor.

We have seen how to calculate descriptors for local patches of images and 3D shapes. In order to obtain a constant-size descriptor vector for a whole observation, we use the following technique, commonly known as the bag of words or bag of features descriptor: First, we calculate local descriptors for all patches for all of the training observations. Then, we cluster the resulting descriptors to build a dictionary of reference descriptors. There are different ways in which the dictionary can be obtained (see [Coates et al. 2011] for an evaluation of different methods), such as k-means clustering or randomly picking reference samples from the set of calculated patch descriptors. In our experiments, we first reduce the descriptor space to 16 dimensions using PCA ([Pearson et al. 1901]). PCA gives us a set of orthogonal eigenvectors together with their eigenvalues. We take the eigenvectors that correspond to the 16 largest eigenvalues and compute their dot products with the descriptors. This provides us with a 16-dimensional vector of coefficients for each descriptor, on which we then perform k-means clustering in order to select k cluster means for our dictionary. The rationale behind performing a dimensionality reduction first is that performing k means in high dimensional spaces quickly becomes unreliably and inefficient, due to the “curse of dimensionality”. When uniformly sampling random points in a Euclidean space, the “curse of dimensionality” describes the effect that pairwise distances between such points become increasingly similar as the dimensionality of the space increases. For high-dimensional spaces, this renders Euclidean distances almost useless for obtaining information on where the points are located (see e.g. [Beyer et al. 1999] for more details). To compute the bag of features histogram for a given observation, we assign each part descriptor to the reference descriptor from the dictionary which is closest to the descriptor in a Euclidean sense (alternative assignment schemes are discussed in [Coates et al. 2011]). The resulting bag of features descriptor is then given

by the histogram over the clusters. For our application, we rescale each bin of the histogram individually to improve the performance of the descriptor under certain conditions further. Specifically, each bin is rescaled by the reciprocal of the total count of parts that are assigned to this bin across all training shapes. The rationale behind this modification is the fact that some parts appear a lot more often than others (e.g. parts of a flat surface). However the less frequently occurring parts are often at least as relevant for the characterization of a shape. Some learning methods such as support vector machines (see chapter 3.1.2) or WSABIE (see 3.1.3) are sensitive to this kind of imbalance, because they rely on symmetric regularizers.[1] Finally, we also normalize the overall descriptor vector in order to achieve invariance against varying patch counts.

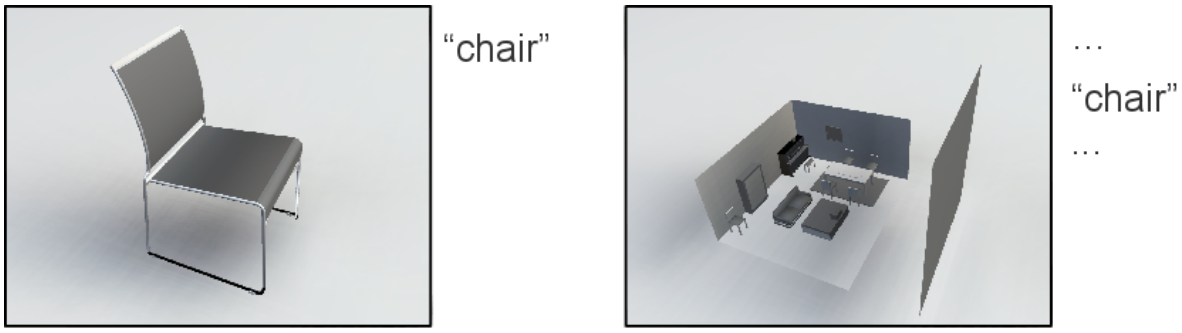


Illustration 7: Both meshes are tagged as "chair" (among other tags in the case of the living room). However their global shapes are vastly different. Both meshes contain local parts which are characteristic for the "chair" attribute though. Meshes from Google 3D Warehouse.

A special property of bag of features descriptors is the fact that they ignore both absolute as well as relative positions of the patches. This can be an advantage, if observation are compositions of objects or characteristic elements, where their (relative) positions are unstable. An example are photographs that show complex scenes combining a variety of different objects (e.g. trees, cars and humans), where the positions of the objects in the scenes is not consistent across pictures. There can however be cases where relevant information is lost due to this invariance. Constellation models (e.g. [Fergus et al. 2005]) are a way to re-establish the relative locality of parts. However they are more complex to use and implement, and we do not cover them here in more detail.

Sampling Local Descriptors

We have seen how a set of locally computed descriptors can be combined into a single per-object descriptor. A question which remains is how many and where within an object the local descriptors should be computed. For images, we compute descriptors along a regularly spaced grid. We remove high image frequencies by filtering the image through a Gaussian filter. HOG descriptors have a limited spatial resolution. We chose the resolution of the HOG descriptors (specifically the size of their spatial bins) to match the Nyquist frequency of the

1 In the context of an optimization problem, a symmetric regularizer is a regularizer which is invariant under rotations of the solution (such as the L_2 norm and in contrast to for example the L_1 norm). Both WSABIE and linear SVM apply their regularizers to the coefficients of a linear classification function. In this setting, the linear classifier will be penalized by a symmetric regularizer if it attempts to scale some components of the descriptor much higher than others.

grid spacing and the Gaussian filter bandwidth. When discretely sampling a continuous signal, the Nyquist frequency is the highest frequency that the signal may contain without introducing loss of information or distortions when reconstructing the continuous signal from its discretely sampled counterpart later. This is known as the Nyquist-Shannon sampling theorem, and the Nyquist frequency can be shown to be half of the sampling resolution (here: the diagonal of the HOG bins). Note that the theorem does not say that encoding any continuous signal by a discretely sampled representation at sampling rate r allows a full reconstruction of all frequencies $f \leq 0.5\,r$. In contrast, the continuous signal must already be guaranteed to contain no frequencies $f > 0.5\,r$ before it is sampled. Otherwise higher frequencies will typically introduce distortions on the lower frequencies during reconstruction. Hence the need for Gaussian filtering in our approach. In practice, we use a grid spacing of 1/16 of the total image width or height, whichever is greater. This results in up to 256 local descriptors per image.

The same principles carry over to 3D shapes. Again we use a regularly spaced grid along which to compute local descriptors. Just as in the case of 2D HOG descriptors, their 3D counterparts that we introduce in chapter 2.2.1 have a well defined spatial resolution. The same holds for Harmonic Shape Descriptors, which resolution is given by the spacing of the spherical shells and the number of spherical harmonics considered. Often, many of the points on the grid are located in empty space. We ignore these regions, and only compute descriptors where they intersect with the shape. We use a grid spacing of $0.01\sqrt{l}$ with l being the maximal diagonal length of the shape.

None of the local descriptors we describe in this thesis are invariant under rescaling of a shape or image. In many data sets, this is a problem because the scale of a given object is not well defined. In photographs, the size of an object depends on its distance to the camera. 3D meshes often do not come with an absolute scale. To counteract this issue, we compute local descriptors at different scales. In addition to the fine grids mentioned so far, we also compute descriptors along grids that have $(1.5)^n$ times the grid spacing of the finest level grid. The Gaussian filtering and resolutions of the descriptors are adapted correspondingly. In practice, we use values for n from 1 to 5, yielding a total of 6 grid levels. Local descriptors from all scales are combined into the same bag of features descriptor.

As a measure to reduce computational costs, we compute 3D descriptors only at locations where the shape has a certain minimum principal curvature. This has the effect of ignoring essentially flat regions of a surface. These regions often make up large parts of a shape, while conveying little specific information about it. On the data sets we have tested, this filtering reduces the number of local descriptors by a factor of 5 to 20 without negatively impacting the performance of the resulting bag of feature descriptors in our applications.

2.2 Our Extensions

The rotation invariant representation obtained by the Harmonic Shape Descriptor makes its application to 3D models where their orientation is unknown and/or inconsistent very convenient. However not only is the relative local rotation of geometry completely lost, but HSD is also relatively fuzzy when it comes to representing sharp edges in a piece of geometry. The HSD does not directly encode changes in the surface normals. Such edges can often convey valuable information on a characteristic shape though.

We propose an adaptation of the 2D HOG descriptor to 3D shapes. We will also see how a consistent orientation of a local piece of geometry can be established in many typical cases.

2.2.1 Extending HOG to 3D

We use an interpretation of HOG features applied to 3D meshes and/or 3D point clouds. The idea is to replace the image gradients used in HOG descriptors by an oriented 3D counterpart. We have experimented with two variants of our descriptor: First, we used the oriented principal curvature along a 3D manifold. Secondly, we used the surface normals to build the histograms. We call these descriptors HOC descriptors for “Histogram of Oriented Curvatures” and HON descriptors for “Histogram of Oriented Normals” respectively.

Illustration 8: Principal curvature of a 3D manifold. Illustration by M. Wand.

While in images we assumed that the upward direction was more or less stable, and it was possible to derive a full canonical coordinate system by taking the direction orthogonal to the upward direction as the second axis, this is not possible in 3D anymore. For many user-generated models there still is a common canonical upward direction. However different orthogonal pairs can be picked for the two remaining axes by rotating them around the upward direction. In order to build a consistently oriented histogram around a given point on the shape surface, we use the average normal around the point as a second direction. We assume that normal directions are given. Where that is not the case, the normals can be estimated in a pre-processing step by performing a PCA analysis ([Pearson et al. 1901]) of the local geometry around each surface point. For building the histogram, we first project all surface points onto the plane defined by the normal, and then use the projection of the upward direction onto this plane to fix the orientation of the histogram on the plane. The sign of the normal fixes the clockwise/anti-clockwise direction. The process is illustrated in Illustration 9.

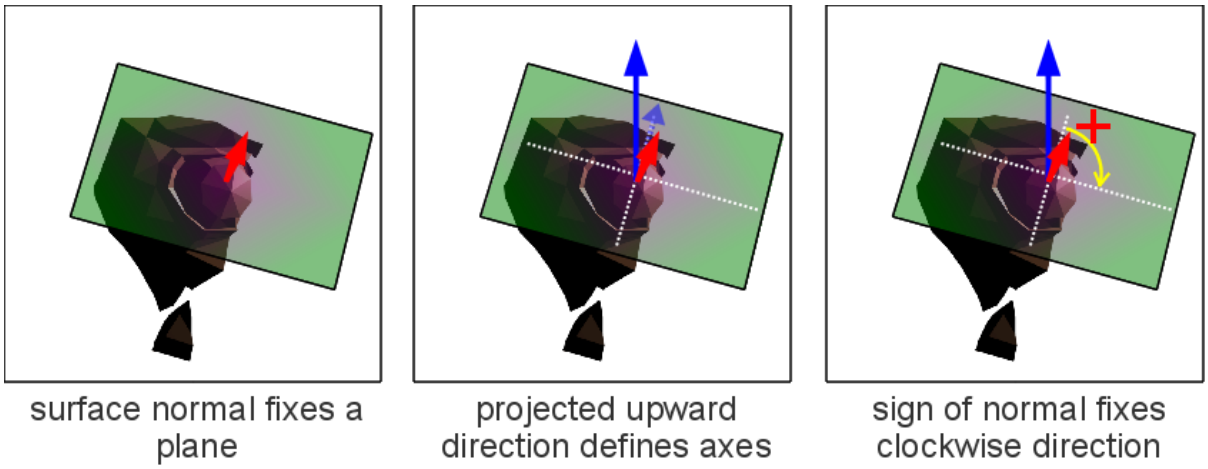


Illustration 9: Fixing a local coordinate frame: The surface normal defines a plane in 3D space. We assume a known upward direction. Its projection onto the plane and the orthogonal direction fix two axes. Finally, the sign of the normal fixes a clockwise direction on the plane.

Thanks to the way we fix the local coordinate frame, our HOC and HON descriptors are invariant under rotations, as long as the upward direction remains intact. Additionally, we normalize the resulting histograms, in order to obtain a) invariance under changes of the sampling density in 3D point clouds and b) invariance under scaling of the curvatures in the case of HOC descriptors. Please note that the latter does not imply invariance under uniform scaling of the whole shape, as the assignment of surface points into the spatial bins will still change.

3 Learning Semantic Attributes

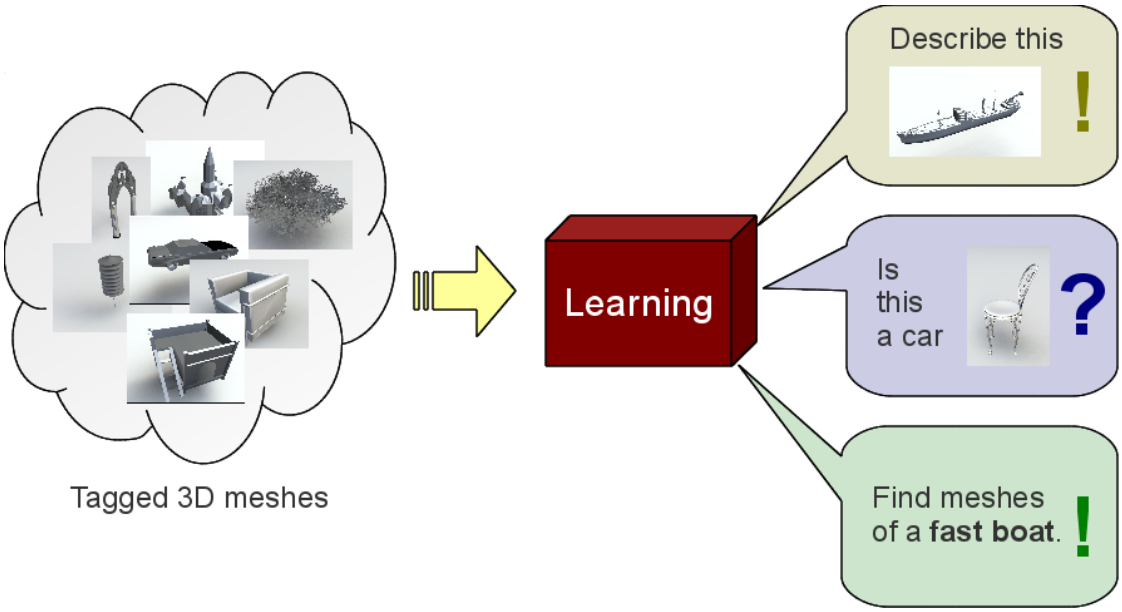


Illustration 10: We are given a set of training meshes, each with an associated set of semantic tags. We use machine learning in order to learn an association between geometry and tags. Our goal is to answer queries regarding the meaning of a shape, or to retrieve shapes from a database by their tags (including shapes without a known tag set).

Assume that we are given a set of input meshes X, each of them represented by its d-dimensional descriptor. Additionally, we assume that each training observation $x \in X$ has a known associated set of text labels $Y_x \subset Y$ that specifies certain aspects of its semantic meaning. By $Y \subset \{0,1\}^{|Y|}$ we denote the set of all labels occurring in the input data. We use the convention that an individual label is represented as a vector which has a one at exactly one coordinate, and zeros everywhere else, that is $\forall\, y \in Y, \|y\| = 1$. Note that the label sets given in the input data can be noisy and/or incomplete. In fact some data sources such as Google 3D Warehouse show label sets which are very noisy and contain a high number of both semantically and geometrically vague or inconclusive labels (such as “awesome” or “greatest”). An example for such a label set is shown in Illustration 11. Based on such training data, our goal in this chapter is to obtain a way to reliably predict the semantic

labels for a previously unseen 3D mesh. Some of the methods we will look at also allow to work in the other direction, and can be used to locate meshes that best adhere to a number of query labels.

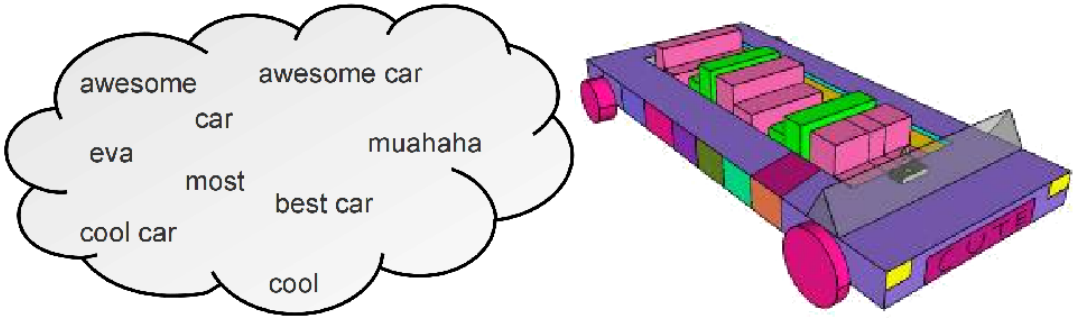


Illustration 11: An example of a noisy label set for a model from Google 3D Warehouse. Model and labels by user 3Dprincess343.

All methods we will look at make use of the following prior: We assume that observations $x, x' \in X$ that have similar descriptors also are semantically similar. For most kinds of descriptors, this translates to "what looks similar, is semantically similar". While this assumption might not always hold, there is also a pragmatic reason for why is makes sense to assume this prior: if the input data contains Gaussian noise or there are any numerical instabilities in the computation of the descriptors, these factors will lead to slight variations in the descriptor values. These slight variations should not have a dramatic impact on the associated labels.

If the space of sets of labels was continuous (or more generally a topological space), this prior could be formalized as follows: We assume that there exists an optimal (in the Bayesian sense) classifier $f : \mathbb{R}^d \to P(Y)$, mapping a given descriptor to a set of labels, which is also a continuous function. However, the power set $P(Y)$ is not by itself a topological space, and a continuous function cannot be defined on it. Completely formalizing the prior for $P(Y)$ is difficult. A weaker, but still useful version of the prior can be formalized as follows: There exists an optimal classifier $f : \mathbb{R}^d \to P(Y)$, such that the set of points $x \in \mathbb{R}^d$ at which f is not constant (more specifically: at which there exists no ball of radius $\epsilon > 0$ with center x over which f is constant), is a thin set.

We look at a number of existing methods for label prediction in chapter 3.1. We then propose a number of adaptations to one of those methods in chapter 3.2, and finally perform an extensive evaluation of the methods and our proposed adaptations on both 3D mesh and 2D photo data sets in chapter 3.3.

3.1 Related Work

A broad variety of shape retrieval methods have been proposed. Shape retrieval allows to retrieve shapes from a database which are similar to a given query shape, and in some cases to retrieve shapes based on a query label as well. Many methods are based on a robustly designed descriptor, on top of which a similarity between shapes can be defined. The SHREC Shape Retrieval Contest is held annually to compare results of current developments in this area. An overview of recent methods and their performance can be found in the results of SHREC 2012 [Li et al. 2012]. We take a look at the K Nearest Neighbor retrieval method, which is based on a distance metric defined on top of a descriptor, in chapter 3.1.1. Other methods employ learning to improve the similarity metric for the specific task given. Based on an annotated set of training shapes, a classifier is trained to reliably separate shapes from different semantic classes. Among others, this has been done by [Li et al. 2007], who use a special kernel over local surface descriptors for training an SVM. [Bronstein et al. 2011] apply a method called similarity-sensitive hashing to improve classification results. We have a closer look at linear support vector machines (SVM) in chapter 3.1.2, which is a very frequently used method for learning classifiers. Finally, we look at the WSABIE method in chapter 3.1.3 which specifically aims at learning image attributes in large scale databases with noisy annotations. The method itself can be applied to other kinds of input data as well, and we will see how it performs for classifying 3D shapes later.

3.1.1 K Nearest Neighbor

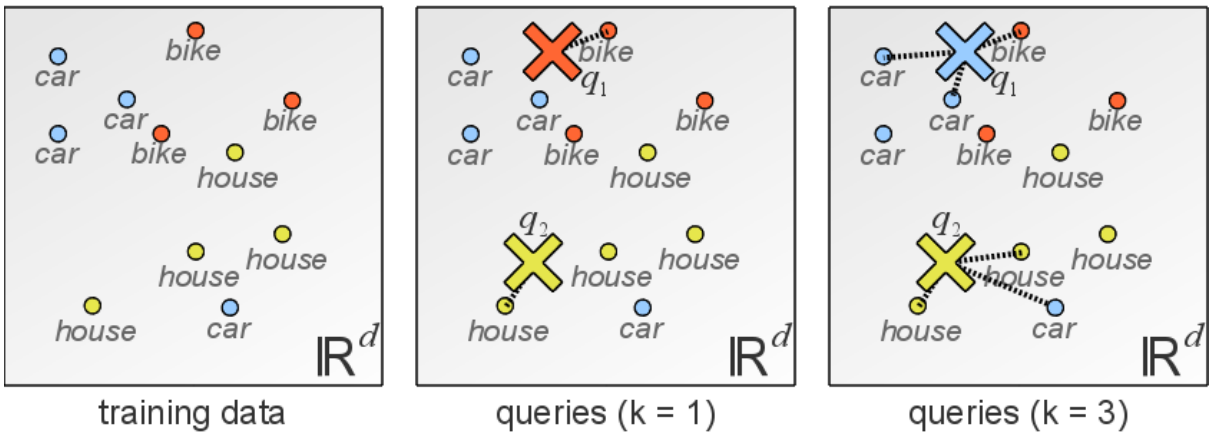


Illustration 12: K Nearest Neighbor querying example. We are given a number of training descriptors in $\mathbb{R}^d$ with associated labels (car, bike, house). We obtain predictions for the queries q_1 and q_2 by considering their k nearest neighbors in the training set. We show hypothetical predictions for k=1 and k=3.

The k nearest neighbor (short: kNN) method is one of the easiest methods for label prediction. We assume a set of training observations S represented by their descriptors $x_1, \ldots, x_{|S|} \in \mathbb{R}^d$. Further, we assume that each training observation x_i has a single associated ground-truth label $y_i \in Y$. Our goal is to predict the label of a non-training observation with descriptor x.

Assume that we have a distance metric $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ on the descriptor space $\mathbb{R}^d$. We define the dissimilarity of two observations through the distance $dist$ between their respective descriptors. For kNN, this distance metric is fixed a priori. In chapter 3.1.3 we describe a method which – while sharing some concepts with kNN – derives a similarity measure specifically tailored to the training data given. Here we make the additional assumption that observations that are similar with respect to this definition, also – at least in tendency – share the same label. From this assumption follows a very easy method for predicting the label of x: We look through the training descriptors $x_1, \ldots, x_{|S|}$, and pick the one that has the smallest distance to x. Assume this descriptor is x_i. Then, we can predict the label for x to be y_i. After all, it looks similar to x_i which also has this label. This is the simplest form of the k nearest neighbor method with $k = 1$. The $k = 1$ simply means that we have considered only one neighbor of x in the descriptor space, here x_i.

We can extend this method to $k > 1$. Instead of just determining the training observation that is closest to x, we determine the k closest ones, x's k nearest neighbors. Assume that we have determined $x_1, \ldots, x_k$ to be those neighbors. We can do so without losing generality, as the indexes of the training observations can be permuted arbitrarily. If $y_1 = \ldots = y_k$, we would obviously predict $y = y_1 = \ldots = y_k$. However it can happen that the neighbors do not agree on a single tag. A frequently used method to solve such conflicts is to predict the label that receives the highest number of votes among the k nearest neighbors. If the result is still ambiguous, higher weights can be assigned to the votes of closer neighbors. This can be done

in a way which guarantees an unambiguous result (assuming no two neighbors have the same distance to x). Alternatively, one could dynamically increase the number of neighbors considered until the ambiguity is resolved

The k nearest neighbor method comes with a number of limitations. First, it usually requires a high number of training samples. It relies on the ability to find a similar looking observation in the training set. In contrast to some other methods, it cannot learn deeper structures in the data. For example assume a case where out of all the components of the descriptors, only one actually determines the label. With symmetric metrics such as the Euclidean one, the prediction results will be heavily influenced by the values of the remaining components. The k-nearest neighbor method cannot detect the underlying structure, and has to rely on a sufficiently dense sampling of the descriptor space through a high number of training observations.

Another problem with k nearest neighbor methods is the so called curse of dimensionality. If the descriptor space is relatively high-dimensional (e.g. $d = 128$), two problems arise. First, many metrics become very unreliable. Specifically, pairwise distances between uniformly distributed points in the high dimensional space become increasingly similar, making them a bad criteria for distinguishing between points. A theoretic and empirical evaluation of this problem can be found in [Beyer et al. 1999]. Secondly, many implementations of k nearest neighbor search become computationally inefficient for high dimensions (e.g. kd-trees). Certain tricks can be used to mitigate these problems. For the latter problem, approximate nearest neighbor (ANN) techniques such as [Arya et al. 1994] can be used. Generally, this will lead to less accurate prediction results. Another method is to reduce the dimensionality of the data. In contrast to ANN algorithms, this can also improve the stability of the distance metric. An often used dimensionality reduction approach is the principal component analysis, or PCA ([Pearson et al. 1901]). PCA first re-centers the data points such that the mean of the data end up at the origin, and then determines orthogonal directions of maximal variation between the data points. The descriptors can then be converted to a lower-dimensional representation through a simple affine transformation. In cases where the relevant components of the data lie in a low-dimensional subspace of the descriptor space, the dimensionality reduction through PCA will lose little information and work well to improve the running time of the algorithm (compare [Beyer et al. 1999]).

3.1.2 Linear Support Vector Machine

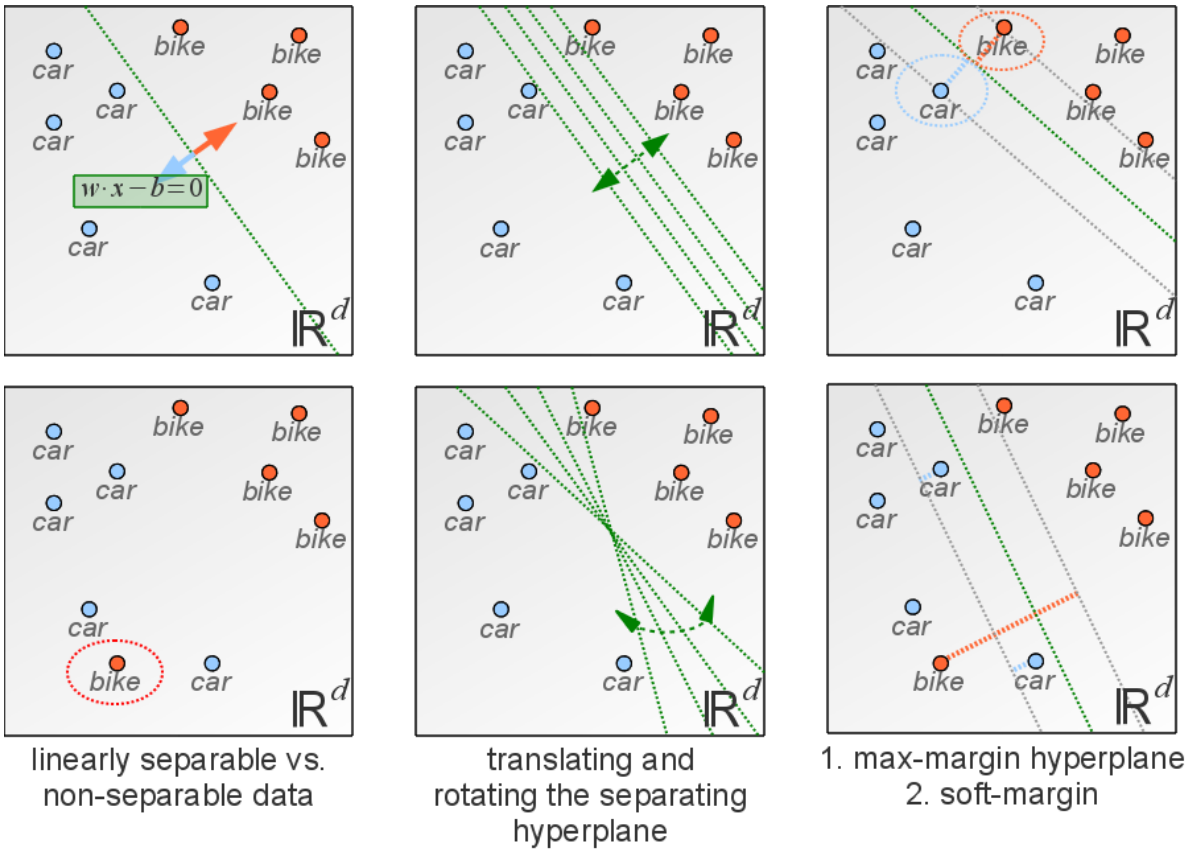


Illustration 13: Linear Support Vector Machines: Linearly separable classes (here: car and bike) can be separated by a linear hyperplane. Usually, the separating hyperplane is not unique and can be translated and/or rotated. The max-margin hyperplane is a canonical separating hyperplane used in Support Vector Machines (in circles: the support vectors). Soft-Margin Support Vector Machines can be used to handle non-linearly separable data sets.

Linear support vector machines (SVM), initially introduced in [Cortes et al. 1995] as "Support-Vector Networks", solve learning problems where there are two non-empty classes of observations in $\mathbb{R}^d$ which can be separated by a hyperplane through $\mathbb{R}^d$. More specifically, if there are classes y_1, y_2, a support vector machine will learn a classifier $w, b \in \mathbb{R}^d$ such that $w \cdot x_i - b < 0$ for all observations x_i of class y_1 and $w \cdot x_i - b > 0$ for all observations of class y_2. At first, we will assume that such a classifier exists. Data for which this is the case is called "linearly separable". In general, there are multiple such classifiers which perform equally well on the training data. Support vector machines use the concept of a "maximal margin" to pick a canonical one. We look at how this is done and what the "support vectors" in a support vector machine are. Afterwards, we look at "soft margin" support vector machines, which allow to find a canonical classifier also in cases where only a subset of the data is linearly separable. Finally, we will see how multiple support vector machines – each being capable of handling two classes only – can be combined to perform multi-class predictions, as required in our label prediction problem.

Assume that we have found a separating hyperplane $H := \{x \in \mathbb{R}^d \mid w \cdot x - b = 0\}$. This hyperplane is often not a unique solution to the problem posed above. Illustration 13 shows an example for such a case. Still, it seems that intuitively, some of the solutions would generalize better to unknown data than others. Support vector machines use the maximum-margin criteria for picking a unique solution. The margin of a hyperplane H is defined as

$$m_S(H) := \min_{x_i \in S, x \in H} \{\|x_i - x\|\}$$

An SVM picks from all the separating hyperplanes the one that maximizes $m_S(H)$. We call this maximum-margin hyperplane $\hat{H}$. Illustration 13 demonstrates the process. It follows from the definition of the margin that there is a certain set of points which lie exactly on the margin. For the maximum-margin solution $\hat{H}$, there is at least one point per class lying on the margin. If this was not the case, the hyperplane could be slightly shifted towards the class which does not have a data point on the margin, thereby increasing the margin. This contradicts the maximum-margin assumption on $\hat{H}$. The points on the margin are called the "support vectors" of the support vector machine. We can modify the initial definition of a hyperplane which separated the classes y_1, y_2 to incorporate the concept of the margin: without loss of generality, we can replace the criteria $w \cdot x_i - b < 0$ for observations x_i of class y_1 by $w \cdot x_i - b \leq -1$, and for the x_i of class y_2 by $w \cdot x_i - b \geq -1$. This definition is equivalent to the original one, as the magnitude of w can be freely changed without actually changing the resulting hyperplane. All hyperplanes $\{x \in \mathbb{R}^d \mid \alpha w \cdot x - \alpha b = 0\}$ are equivalent for all $\alpha \in \mathbb{R}^{\neq 0}$. In this formulation, the margin of a separating hyperplane H can be simplified to

$$m_S(H) = \min_{x_i \in S, x \in H} \{\|x_i - x\|\} = \frac{1}{\|w\|}$$

As we want to maximize $m_S(H)$, the resulting optimization problem is as follows:

$$\min_{w,b} \|w\| \text{ subject to } \forall x_i \in S, w \cdot x_i - b \begin{cases} \leq -1 & \text{if } x_i \text{ is of class } y_1 \\ \geq 1 & \text{if } x_i \text{ is of class } y_2 \end{cases}$$

To simplify the optimization of the problem, $\min_{w,b} \|w\|$ is often replaced by $\min_{w,b} \left\{ \frac{1}{2} \|w\|^2 \right\}$, which reaches its minimum at the same parameters as the original formulation.

So far we have assumed that the given training data is linearly separable. In reality, data is often noisy, the descriptors used might not always contain enough information (or contains it in a form not suitable for linearly separating the data), or some of the data points might have been mislabeled with the wrong class. We would like to derive an adaptation of the support vector machine optimization problem which still has a solution in such cases. Also, we want the solution to be "as close as possible" to the maximum-margin solution in the case of completely linearly separable training data. Please note that we still maintain the requirement that a majority of the data can be linearly separated. We will look at an alternative for non-linear classification in chapter 3.2.3. [Cortes et al. 1995] introduce the idea of a "soft margin" solution. A soft margin of a hyperplane is a margin which can be penetrated by some of the data points. Formally, we relax our previous hard constraint $w \cdot x_i - b \leq -1$ (for

y_1) to $w \cdot x_i - b \leq -1 + \xi_i$, where $\xi_i \in \mathbb{R}^{\geq 0}$ is called the "slack variable" for x_i . We do the same for y_2 . We can then formulate the optimization objective such that penetrating the margin is discouraged, but permissible when necessary:

$$\min_{w,b,\xi} \left\{ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{|S|} \xi_i \right\} \text{ subject to } \forall\, x_i \in S,\ w \cdot x_i - b \begin{cases} \leq -1 - \xi_i & \text{if } x_i \text{ is of class } y_1 \\ \geq 1 + \xi_i & \text{if } x_i \text{ is of class } y_2 \end{cases}$$

The parameter $C > 0$ determines how strongly a violation of the margin is penalized. The term $\sum_{i=1}^{|S|} \xi_i$ in combination with the side constraints is often referred to as the *hinge loss*. For a general classifier f it can be rewritten as $loss_{\text{hinge}}(x_i) := max\left(0, 1 - f(x_i) t_{x_i}\right)$, where $t_{x_i} = -1$ if x_i is of class y_1 and $t_{x_i} = 1$ otherwise.

We have explained how soft-margin support vector machines can be used to learn a classifier that distinguishes between to classes y_1, y_2 . Our initial goal was to obtain a classifier for predicting labels from an arbitrarily sized label set Y . An observation x_i could have any label $y_i \in Y$ assigned to it. We look at two prominent schemes of how binary learning methods and their resulting classifiers can be combined to multi-class classification methods: The one-versus-one, and the one-versus-rest scheme.

In one-versus-one, we first build all label subsets of size two $\{y_j, y_k\} \subset Y$. We then independently train a binary SVM for each of those pairs. This yields a classifier

$$f_{j,k}(x) := \begin{cases} y_j & \text{if } w_{j,k} \cdot x - b_{j,k} < 0 \\ y_k & \text{otherwise} \end{cases}$$

The two-class classifiers can now be combined into an overall multi-class classifier. Different methods can be applied. A simple one is to count how many of the binary classifier "vote" for a specific label, and classifying a given data point as the label that received the highest number of votes. As an extension, votes can be weighted by how far away the classified data point is from the respective decision hyperplane (more distance → more stable classification). A disadvantage of the one-vs-one schema is that it requires a quadratic number of binary SVM, and that it might become unreliable if some of the classes have only few training samples available.

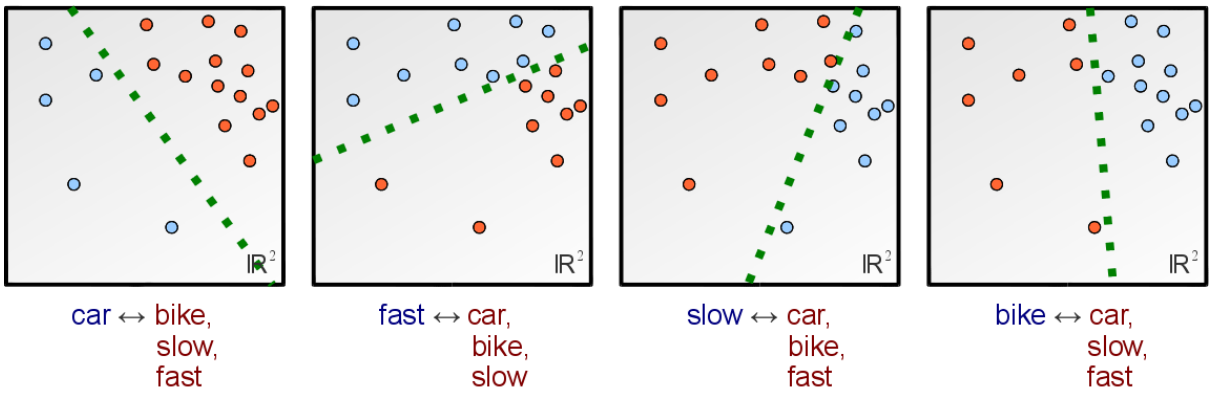


Illustration 14: One-vs-rest scheme for applying SVMs to multi-class problems.

Another common strategy is the one-vs-rest strategy. Here, we segment the training observations into a set S_j containing all samples of class y_j and a second set $S_{\text{rest}} = S - S_j$. We assign a separate class $y' \notin Y$ to the set of observations in S_{rest} and train an SVM, yielding a classifier

$$f_j(x) := \begin{cases} y_j & \text{if } w_j \cdot x - b_j < 0 \\ y' & \text{otherwise} \end{cases}$$

Again, we combine all of these classifiers (we get $|Y|$ of them) into a single multi-class classifier. For example we can define the unified classifier as

$$f(x) := \underset{f_j(x)}{argmin} \left| w_j \cdot x - b_j \right|$$

Attempts have also been made to adapt the SVM learning objective itself in order to support multi-class learning directly, for example [Crammer et al. 2001].

3.1.3 Learning a Latent Semantic Space

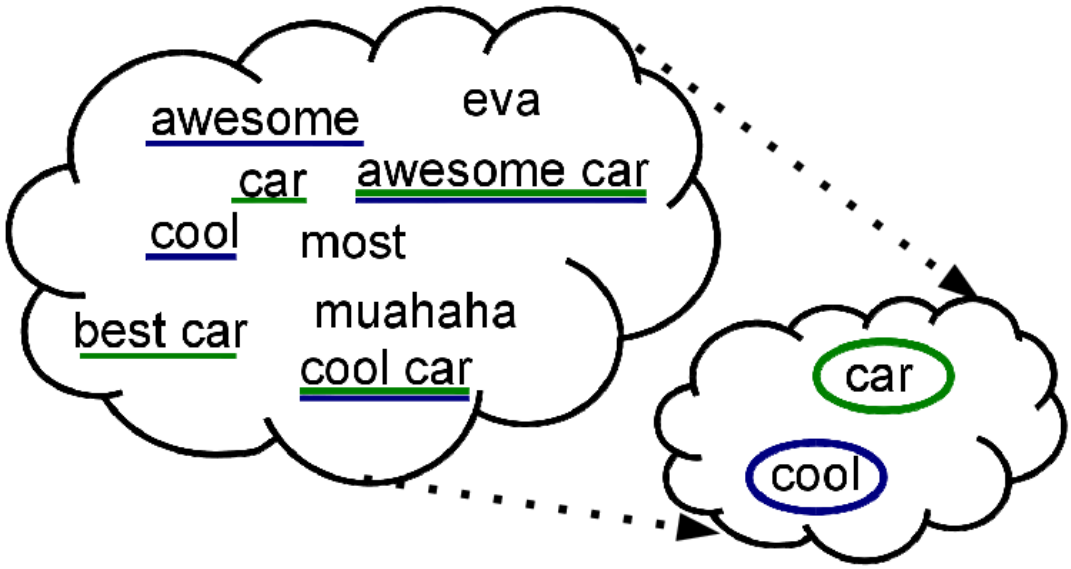


Illustration 15: Complex label sets from user-annotated data can often be reduced to a smaller number of major semantic concepts. We would like to encourage our learning scheme to automatically detect such redundancies in the label sets. Original label set from Google 3D Warehouse.

[Weston et al. 2011] propose a different approach for learning descriptor-label associations. A special aspect of their approach is that it forces the learned model to have a low number of effective dimensions, despite a potentially very high-dimensional label space. The method tries to perform this dimensionality reduction with as little loss of information as possible. As a consequence, labels which are related in one of the following ways are automatically detected: 1. labels which frequently occur together in the ground-truth label sets, or 2. labels for which the associated training objects cannot be discriminated from one another within the boundaries of the model used and the information available in their descriptors. The dimensionality reduction can be used to increase the computational efficiency of the model.

Additionally, it has the effect that strongly related labels are optimized jointly, each taking advantage of the other labels' training data. This concept is similar to what is often attempted in recommendation systems, where the goal is to predict the taste of a user based on only a small number of known ratings or preferences known about them. If a low-dimensional “preference space” can be learned from the sparse set of known preferences of a huge number of users, a small number of known preferences for any given specific user suffices to constrain their position in this space and recommendations can be made based on that information. A prominent example is [Rennie et al. 2005].

In this chapter, we first introduce the loss function that [Weston et al. 2011] use in their training. Next, we look at how they make use of a low-dimensional latent space in order to regularize the resulting model and speed up the training calculations. Finally we see how the loss function can be optimized.

For our application, we assume that as input data, we are given a number of training observations, each represented by an associated fixed-length descriptor $x_i \in \mathbb{R}^d$. Further, we assume for now that there is a set of labels, and each observation x_i is annotated by exactly one associated label y_i. The restriction to a single label per observation will be lifted at the end of this chapter. We refer to the set of all labels by Y, and to the set of all training observations by X. Our goal is to learn an efficiently computable function $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^{|Y|}$ which, given a descriptor, provides us with scores for the different labels. In the following, we assume that Φ is a linear mapping. We want the labels that apply the most to the given descriptor to receive the highest scores, and the ones that do not apply as much lower scores. We will see how to formalize this objective in the following section.

Rank Loss and WARP Loss

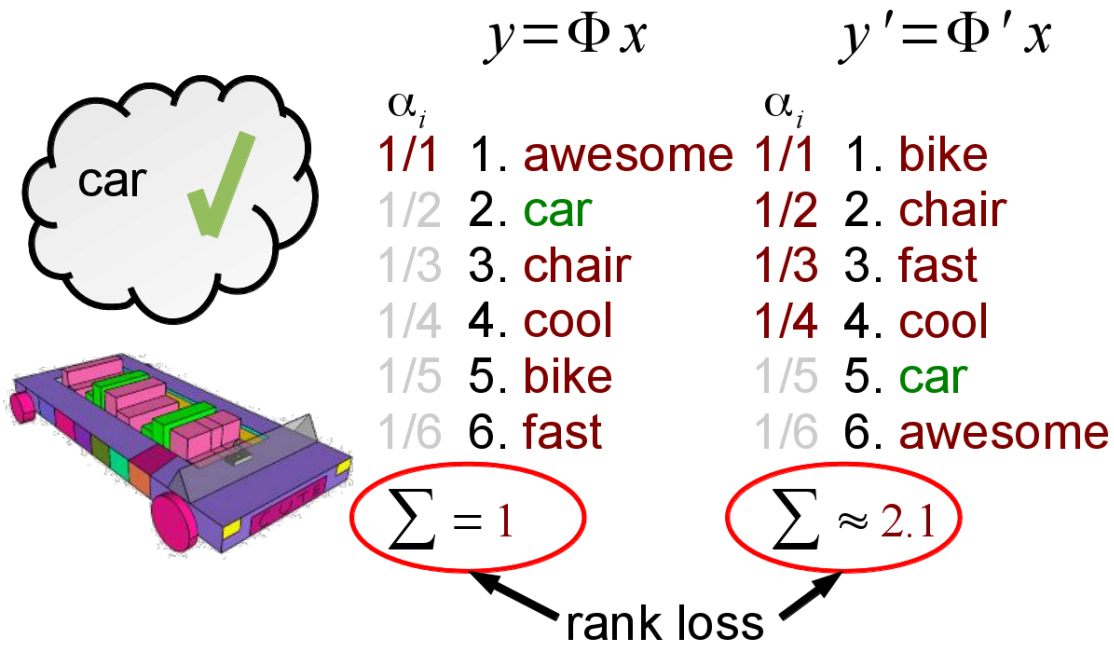


Illustration 16: We want to compare two classifiers Φ and Φ' with respect to their predictions y and y' for a given shape with descriptor x. The rank of the correct label ("car") is determined and all violating labels (incorrect labels ranked above the correct label) are assigned a penalty α_i based on their rank. Summing over these penalties yields the rank loss of the respective classifiers for x. Mesh from Google 3D Warehouse.

In order to derive a suitable mapping Φ, we first have to define a criteria to rate how "good" a given candidate Φ is, given the available the training data. Given a prediction $y_{pred} = \Phi(x_i)$ for one of the training observations, [Weston et al. 2011] use ideas from [Usunier et al. 2009] to derive a loss to this prediction. First, all labels are ranked based on y_{pred}. The rank of a label y is given by

$$rank_y(x_i) := \sum_{y' \in Y - \{y\}} I(\Phi_y(x_i) \leq \Phi_{y'}(x_i))$$

where $\Phi_y(x_i)$ is the component of y_{pred} that corresponds to the label y, and

$$I(x) := \begin{cases} 1 & \text{if } x = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

is the indicator function.

In the simplest case, we can define a rank-based loss for x_i as follows:

$$L_{lin}(x_i) := \frac{1}{|Y| - 1} rank_{y_i}(x_i)$$

where y_i is the ground-truth label for x_i. Note that $0 \leq L_{lin}(x_i) \leq 1$.

The overall risk for Φ then follows as

$$risk(\Phi) := \frac{1}{|X|} \sum_{x_i} L_{lin}(x_i)$$

[Weston et al. 2010] point out that in cases where no perfect fitting of the training data is possible, this risk does not necessarily lead to good predictions on the top rank(s). For example consider two alternative models Φ, Φ' where Φ puts the correct label for observation x_1 at rank 1 and for observation x_2 at rank 100, while Φ' puts them at rank 50 each. Assuming all other predictions are equal, $risk(\Phi) = risk(\Phi')$, even though – if we are interested in deriving good predictions on the top ranks as much as possible – Φ actually provides the better predictions. [Usunier et al. 2009] generalize the notion of rank loss, allowing to incorporate considerations of this kind. The idea behind this notion is to assign a weight α_j to each of the $|Y|-1$ potentially violating ranks. The loss for an observation x_i is the sum of the weights α_j for all the ranks of violating labels $y \neq y_i$ for which $rank_y(x_i)$ is higher than the rank of the correct label $rank_y(x_i)$:

$$L(x_i) := \sum_{j=1}^{rank_{y_i}(x_i)} \alpha_j$$

[Usunier et al. 2009] put the additional constraint on the weights that $\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_{|Y|-1}$ and $\sum_j^{|Y|-1} \alpha_j = 1$. To resemble the earlier introduced linear rank loss L_{lin}, we would set $\alpha_1 = ... = \alpha_{|Y|} = (|Y|-1)^{-1}$. [Weston et al. 2010] use linearly decreasing weights, defined in [Usunier et al. 2009] as

$$\alpha_j = \frac{1}{j} * \left(\sum_{j'=1}^{|Y|-1} \frac{1}{j'} \right)^{-1}$$

[Usunier et al. 2009] also propose a variant that uses constant positive weights for the top p ranks, and zero weights for the remaining ones, as well as a variant which uses exponentially decreasing weights, which were not used in [Weston et al. 2010] though.

Stochastic Gradient Descent Optimization

While we have defined a risk function that tells us how "good" a given model Φ is, we have not talked about how the best model for a given set of training data can actually be found. The high number of degrees in which Φ can be varied ($d * |Y|$), exhaustively testing different Φ for their associated risk is not feasible. Luckily, there are many known methods for efficiently optimizing objective functions of the form $F : \mathbb{R}^n \rightarrow \mathbb{R}$. Our *risk* fits into this definition if we represent the linear function Φ as a vector of its coefficients. Gradient descent is one of the simplest iterative methods for finding the minimum of convex, differentiable objective functions F. Starting with an initial guess $b_0 \in \mathbb{R}^n$, an updated solution is obtained in each iteration:

$$b_{i+1} = b_i - \gamma_i \nabla F(b_i)$$

where γ_i is the step size. The step size can either be constant throughout all iterations, vary in a fixed pattern (often monotonically decreasing), or can be adaptive based on the current state of the optimization. When using a fixed step size, a high value usually leads to faster convergence in the first iterations, but a lot of oscillations at later ones, including the possibility of never actually coming close to the optimum. Small values on the other hand usually approach the optimum on a more direct path, but take more iterations for obtaining the first rough approximation. We will compare different options with respect to their convergence behavior in our application in 3.2.1.

We could apply gradient descent for [Weston et al. 2010] to minimize $risk(\Phi)$. Two problems occur when attempting this though: First, $risk(\Phi)$ is not differentiable at a number of points, as it is based on the indicator function. What's worse, at the points where it is differentiable, its gradient is zero. Thus, gradient descent would never make any progress. In the next paragraph, we will see how [Weston et al. 2010] circumvent this problem by replacing the indicator function with a differentiable hinge loss formulation. Secondly, we will see that even with that formulation, computing $risk(\Phi)$ is expensive for large data sets and/or a high number of potential labels. We will look at the idea of stochastic gradient descent to work around this issue.

Let us first look at the issue of the non-differentiable risk. In [Weston et al. 2010], the loss for a single observation x_i is first rewritten as follows:

$$L(x_i) = L(x_i)\frac{rank_{y_i}(\Phi(x_i))}{rank_{y_i}(\Phi(x_i))} \overset{\text{def. rank}}{=} L(x_i)\sum_{y' \in Y - \{y_i\}} \frac{I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i))}{rank_{y_i}(\Phi(x_i))}$$

with $L(x_i)=0$ if $rank_{y_i}(\Phi(x_i))=0$. In order to obtain a differentiable function to use for the gradient descent optimization, $L(x_i)$ and $rank_{y_i}(\Phi(x_i))$ are held constant in each iteration step, and we concentrate on $I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i))$ to obtain a gradient. While this term is not yet differentiable, this will serve us as a basis for additional modifications in the next paragraph. On first sight, this decision seems very arbitrary. We have multiplied the original loss with a term which is equivalent to 1, and then decided that we would like to base our gradient computation on only a specific part of this 1-term. We could have chosen any other representation of 1, and could doubtlessly end up with vastly different gradients. However basing the gradient on $\sum_{y' \in Y - \{y_i\}} I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i))$ makes sense because it actually behaves very similar to $L(x_i)$. It ignores the weights α_j, but as long as we consider only a single observation x_i in our loss, minimizing $\sum_{y' \in Y - \{y_i\}} I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i)) = rank_{y_i}(\Phi(x_i))$ will also minimize $L(x_i)$. While this does not hold anymore when we later combine multiple such losses into a risk spanning all observations, the assumption will still hold locally in the sense that slightly modifying Φ such that it approaches the optimum of $rank_{y_i}(\Phi(x_i))$, will also bring the overall risk over all $L(x_i)$ closer to its optimum (except for a thin set of points in the parameter space, across which the ranking of a label actually changes for one of the observations). In the following paragraphs we will see how rewriting $L(x_i)$ like this will allow us to apply some further modifications, which will ultimately make the application of a gradient descent based optimization method very efficient.

[Weston et al. 2010] modify the variable term $I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i))$ in two aspects: First, they introduce a margin like in SVMs (compare chapter 3.1.2): $I(\Phi_{y_i}(x_i) \leq \Phi_{y'}(x_i)+1)$. The margin will play an important role in the next chapter, when we introduce regularization. Secondly, to make the loss and thereby the risk differentiable, the indicator function is replaced by the hinge loss similar to a soft-margin SVM:

$$L_{rankHinge}(x_i) := L^1(x_i) \sum_{y' \in Y - \{y_i\}} \frac{|1 - \Phi_{y_i}(x_i) + \Phi_{y'}(x_i)|_+}{rank^1_{y_i}(\Phi(x_i))}$$

where $|x|_+ := max\{x, 0\}$, and margin-penalized variants of the rank

$$rank^1_y(x_i) := \sum_{y' \in Y - \{y\}} I(\Phi_y(x_i) \leq \Phi_{y'}(x_i)+1)$$

and discrete loss

$$L^1(x_i) := \sum_{j=1}^{rank^1_{y_i}(x_i)} \alpha_j$$

are used. The effect of these changes is that even if Φ classifies a label correctly, it still results in a non-zero loss unless the score of the correct label is at least by one higher than the score of the highest-ranked wrong label. While the resulting risk

$$risk_{rankHinge}(\Phi) := \frac{1}{|X|} \sum_{x_i} L_{rankHinge}(x_i)$$

is still not convex over the course of multiple iterations, it is always differentiable and yields non-zero gradients until an optimum is found (remember that Φ is a linear function).

As mentioned before, computing $risk^1(\Phi)$ is an expensive operation, requiring computation time in $O(|X||Y|)$ (assuming constant costs for evaluating $\Phi(x)$). [Weston et al. 2010] use stochastic gradient descent to mitigate this issue. In each iteration, stochastic gradient descent picks one training observation randomly. Say that we pick observation x_i with a probability of $P(x_i)$, then the stochastic risk estimation is given by

$$srisk_{rankHinge}(\Phi) := \frac{1}{P(x_i)} L_{rankHinge}(x_i)$$

Computing $srisk_{rankHinge}(\Phi)$ can be performed in $O(|Y|)$. While this makes each iteration a lot faster, we will also need more iterations to obtain good training results. However overall convergence can still be faster if there is redundancy in the training data. More details on the convergence behavior of stochastic gradient descent can be found in [Bottou 2003].

[Weston et al. 2010] additionally introduce a stochastic estimator for $L_{rankHinge}$. While the worst-case complexity of their estimate is still in $O(|Y|)$, their stochastic estimator is claimed to be faster in practice. Assume an observation x_i has been sampled in a particular iteration of the stochastic gradient descent. The estimation of $L_{rankHinge}$ works by repeatedly sampling a random label $\hat{y} \in Y - \{y_i\}$ and checking if it is better ranked than the ground-truth label y_i

(or violating its margin): $1+\Phi_{\hat{y}}(x_i)>\Phi_{y_i}(x_i)$. This sampling is repeated until such a violating label has been found, up to a maximum of $|Y|-1$ times. Let N be the number of trials before a violating label has been found. Then an estimate for $rank^1_y(x)$ is given by

$$rank^1_{y_i}(x_i)\approx floor\left(\frac{|Y|-1}{N}\right)$$

and an overall stochastic estimator for $L_{rankHinge}$ is given by

$$sL_{rankHinge}(x_i,\bar{y}_i,\hat{y}):=L^1(x_i)\left|1-\Phi_{y_i}(x_i)+\Phi_{\hat{y}}(x_i)\right|_+$$

where $L^1(x_i)$ can be computed based on the approximate estimate for $rank^1_{y_i}(x_i)$.

Clearly, the sooner a violating label is found, the more computation time is saved by this approach. The main benefit of using the estimate can therefore be expected at the beginning of the optimization and for very complex datasets, where perfect label prediction results cannot be obtained.

Handling Multi-Label Data

[Weston et al. 2010] define the losses and optimization steps under the assumption that for a given observation x_i, there is exactly one ground-truth label $y_i\in Y$. However many complex data sets such as Google 3D Warehouse or Flickr can have multiple labels for any given object. We denote the set of ground-truth labels for observation x_i by $Y_{x_i}\subset Y$. We

adapt the single-label learning scheme as follows: In each iteration, we randomly sample one of the ground-truth labels $\bar{y}_i\in Y_{x_i}$ for the previously sampled observation x_i. We redefine the (margin-penalized) rank of $\bar{y}_i$ such that it ignores how $\bar{y}_i$ gets ranked compared to the other ground-truth labels of the same object:

$$rank^1_{\bar{y}_i}(x_i):=\sum_{y'\in Y-Y_{x_i}} I(\Phi_y(x_i)\le\Phi_{y'}(x_i)+1)$$

For stochastically estimating $rank^1_{\bar{y}_i}(x)$ (see previous paragraph), we now sample only labels $\hat{y}\in Y-Y_{x_i}$ in the sampling loop, and the estimated rank is then given by

$$rank^1_{\bar{y}_i}(x_i)\approx floor\left(\frac{|Y|-|y_{x_i}|}{N}\right)$$

To sum up, a single iteration in the WARP optimization for multi-label training data works as follows:

1. Sample an observation $x_i \in X$
2. Sample a ground-truth label $\bar{y}_i \in Y_{x_i}$ from the ground-truth label set of x_i
3. For $N = 1 \cdots (|Y| - |Y_{x_i}|)$
    1. Sample a random label $\hat{y} \in Y - Y_{x_i}$
    2. Check whether $1 + \Phi_{\hat{y}}(x_i) > \Phi_{\bar{y}_i}(x_i)$. If yes: continue in step 4
4. Compute the stochastic estimate for the loss $sL_{rankHinge}$
5. Compute gradients and update Φ accordingly (see next section)

Low-Dimensional Latent Space Learning

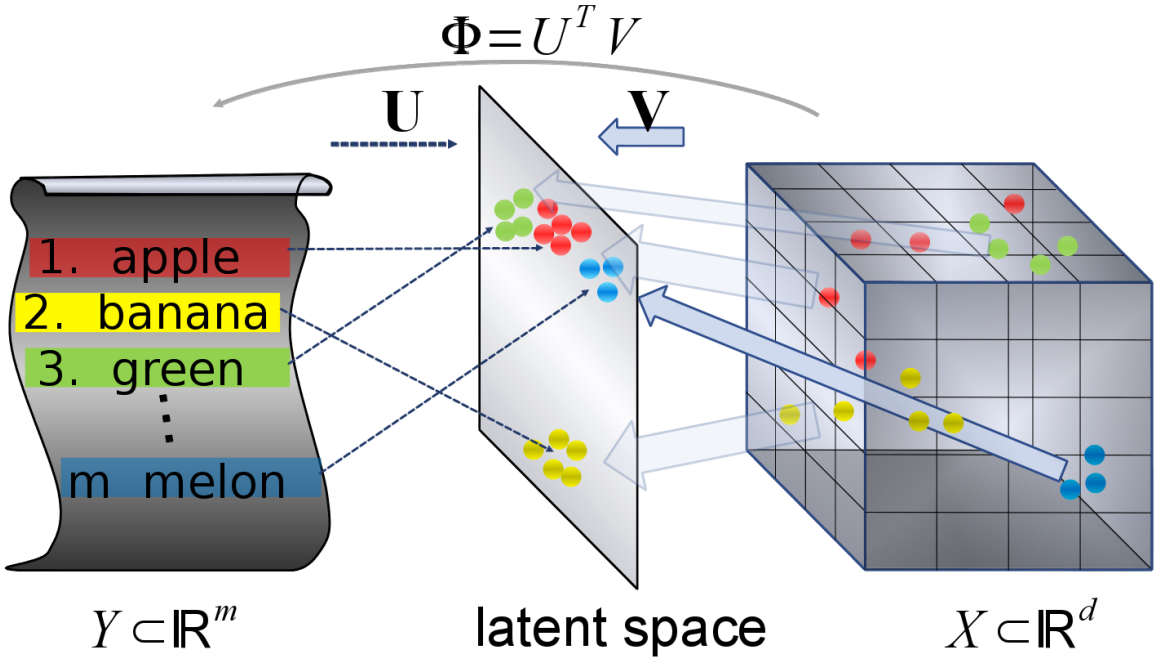


Illustration 17: Using a low-dimensional latent space to express semantic similarity: Both labels and descriptors are mapped into a common low-dimensional latent space through linear mappings U and V respectively. The dot product of their embeddings in the latent space corresponds to the semantic similarity between descriptors and/or labels. Illustration by M. Wand and R. Herzog.

Building on the rank-based risk function introduced in [Weston et al. 2010], [Weston et al. 2011] shows how to learn a low-rank classifier for assigning labels to input descriptors in what they call the "WSABIE" method.

[Weston et al. 2011] use a linear classifier $\Phi \in \mathbb{R}^{|Y| \times d}$ with two constraints on Φ . First, Φ must be of rank no larger than D , i.e. it can be written as $\Phi = U^T V$ with matrices $U \in \mathbb{R}^{D \times |Y|}, V \in \mathbb{R}^{D \times d}$. Second, the norm of the columns of these matrices is limited:

$\forall 1 \leq i \leq |Y|, \|U_i\|_2 \leq C$ and $\forall 1 \leq i \leq d, \|V_i\|_2 \leq C$ for a constant $C \in \mathbb{R}^+$. We first explain the rationale behind the rank limiting and give an intuitive interpretation of its effect. We will then see how limiting the column-norm provides an additional regularization to counteract over-fitting.

The idea behind limiting the rank of the classifier is to force it to utilize semantic or visual overlap between labels. This is most useful in settings where the label set Y is huge and contains semantic redundancies. For example crowd-annotated data sets often contain synonyms, or words from different languages. Without the rank limit, all of these would be learned and modeled completely independently of each other. By introducing the rank limit, training samples from different but synonymous labels will tend to be "coupled together" during the training. Finally, learning a low-rank linear classifier reduces the number of variables, making the training and later classification more computationally efficient.

The factorization $\Phi = U^T V$ provides an intuitive interpretation of the low-rank scheme. Assume a D-dimensional Hilbert space with the following semantic interpretation: The inner product between points in the space corresponds to semantic similarity. We can now formulate the learning objective as follows: We want to find mappings $U \in \mathbb{R}^{D \times |Y|}$ and $V \in \mathbb{R}^{D \times d}$ that map labels and images respectively into the common Hilbert space. The resulting positions of the training observations should be consistent with our semantic interpretation of the space. Specifically the inner product of semantically similar labels and images in this space should be high (compared to less similar pairs). We define the similarity between two points $s_1, s_2 \in \mathbb{R}^D$ in the semantic space as the usual inner product $\langle s_1, s_2 \rangle$. In order to classify a given image represented by its descriptor $x \in \mathbb{R}^d$, we calculate its position in the semantic space Vx and then perform a nearest-neighbor search to find the nearest label $y \in Y$ with respect to the similarity $\langle Vx, Uy \rangle$. Remember that we represent a label $y \in Y$ as a unique $|Y|$-dimensional vector consisting of a one in one component and zeros elsewhere. Then performing the closest-neighbor search in the semantic space is equivalent to computing $\Phi x = U^T V x$, ranking the components of the resulting vector and interpreting them as prediction ranks of the corresponding labels.

To counteract over-fitting, the column norms of the involved matrices are limited by a hard threshold, specifically $\forall 1 \leq i \leq |Y|, \|U_i\|_2 \leq C$ and $\forall 1 \leq i \leq d, \|V_i\|_2 \leq C$. [Weston et al. 2011] claim that this "acts as a regularizer in the same way as is used in lasso" (compare [Tibshirani 1996]). Unfortunately [Weston et al. 2011] do not provide additional information on how exactly the properties of Lasso, which is defined for least-squares optimization problems, carries over to the given rank-based optimization problem. In fact, on the first sight, this kind of regularization might appear odd. Assume that we have an optimal (potentially over-fitted) solution with respect to the training data $U_{\text{opt}}, V_{\text{opt}}$ that does not adhere to this regularization. As our classifier is rank-based, we can easily derive a solution U', V' which is equivalent with respect to its predictions, and fulfills the regularization criteria. For example, such a solution can be obtained as $U' := c^{-1} U_{\text{opt}}, V' := c^{-1} V_{\text{opt}}$, where $c = max\{\|U_1\|_2, \ldots, \|U_{|Y|}\|_2, \|V_1\|_2, \ldots, \|V_d\|_2\}$. However it turns out that while this solution is equivalent to $U_{\text{opt}}, V_{\text{opt}}$ with respect to the classification predictions they make, it is not generally equivalent with respect to their losses. The reason for this is the fixed-size margin that we had added to the loss L' earlier, and which would not work without the regularizer.

The matrices U and V are optimized directly in the WSABIE method, rather than optimizing Φ and deriving a factorization $\Phi = U^T V$ afterwards, which would be more computationally expensive and would make enforcing the rank constraint difficult. At the beginning, U and V are initialized randomly. In each iteration of the stochastic gradient descent, the loss $L^1(x_i)$ for a sample observation x_i is computed as explained earlier, based on a sampled ground-truth label $\bar{y}_i \in Y_{x_i}$ and a violating label $\hat{y} \in Y - Y_{x_i}$. The gradients for the update steps are then given by the following equations:

$$\frac{\partial L_{rankHinge}}{\partial U}(x_i, \bar{y}_i, \hat{y}) = \begin{cases} (\hat{y} - \bar{y}_i) \cdot (V x_i)^T & \text{if } sL_{rankHinge}(x_i, \bar{y}_i, \hat{y}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial L_{rankHinge}}{\partial V}(x_i, \bar{y}_i, \hat{y}) = \begin{cases} U(\hat{y} - \bar{y}_i) \cdot x_i^T & \text{if } sL_{rankHinge}(x_i, \bar{y}_i, \hat{y}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The results from [Weston et al. 2011] show that the WSABIE method yields prediction performance well ahead of the baseline methods (one-vs-rest SVM, kNN) and has good optimization efficiency for very huge datasets. Specifically, their evaluation was performed on two image data sets, consisting of 2.5 and 9.9 million images respectively. The images were annotated with tags from a set of 15,952 and 109,444 different tags respectively, demonstrating the successful dimensionality-reduction of very high-dimensional and noisy input label spaces. We perform our own evaluation of the WSABIE method in chapter 3.3.

3.2 Our Adaptations

So far, we have looked at a few base line methods (chapters 3.1.1 and 3.1.2) and have introduced the learning approach of [Weston et al. 2011], which is based on optimizing a low-dimensional latent semantic space using stochastic gradient descent (chapter 3.1.3). We now consider a number of optimizations and extensions to the method of [Weston et al. 2011]. First, we see how the stochastic gradient descent optimization itself can be optimized. We then look at the option of utilizing correlations in the ground-truth label sets in order to make the algorithm more robust for crowd-annotated data sets. Finally, we introduce the idea of "label aliasing", which allows us to learn non-linear classifiers without sacrificing the option of interpreting the resulting classifier with respect to the structure of the input data. The suggested adaptations are later evaluated empirically in chapter 3.3.

3.2.1 Optimizations of the Stochastic Gradient Descent

A critical parameter in gradient descent is the step size. For convex optimization problems, certain choices for the step size can be shown to guarantees that the global optimum is reached eventually. However such guarantees can in general not be made for non-convex problems. Yet, if the step size is too large, the algorithm may oscillate around a local optimum without ever reaching it. If on the other hand it is too small, it will require a lot of iterations to get close to any optimum in the first place. For non-convex problems like the one in the WSABIE method, small step-sizes pose the additional problem that they increase the

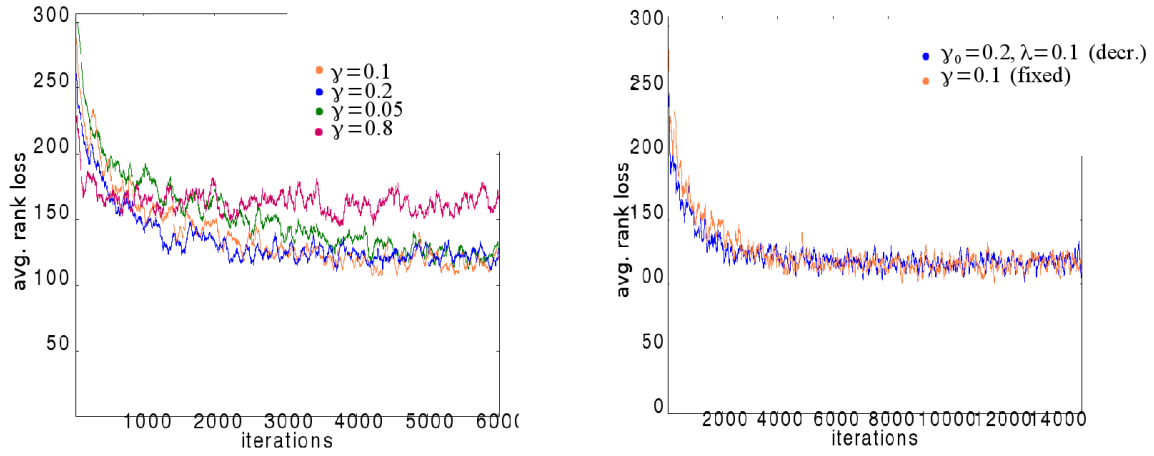


Illustration 18: Comparison of step sizes: Left: Different fixed step sizes. Right: Fixed vs. decreasing step size. See Table 1 for details. Computed on the Google 3D Warehouse dataset (compare 3.3.2).

chances of getting stuck in local optima. A monotonically decreasing step size such as $\gamma_i = \gamma_0 i^{-\lambda}$ for some $\lambda > 0$ in iteration i can be used as a compromise between the two. In Illustration 18, we compare the convergence behavior in the WSABIE problem with different choices for the step size. We found that a fixed step size of 0.1 yields relatively fast convergence, without sacrificing much on the optimality of the solution. A decreasing step size yielded only marginally faster convergence in our application.

	500 iter.	1000 iter.	2000 iter.	4000 iter.	8000 iter.
Fixed $\gamma = 0.05$	200	190	170	150	125
Fixed $\gamma = 0.1$	180	170	140	110	110
Fixed $\gamma = 0.2$	165	145	125	120	115
Fixed $\gamma = 0.4$	160	140	130	130	125
Fixed $\gamma = 0.8$	170	155	150	145	145
Decreasing $\gamma_0 = 0.05$, $\lambda = 0.1$	200	180	160	135	115
Decreasing $\gamma_0 = 0.1$, $\lambda = 0.1$	185	160	140	120	110
Decreasing $\gamma_0 = 0.2$, $\lambda = 0.1$	170	145	130	120	120
Decreasing $\gamma_0 = 0.4$, $\lambda = 0.1$	160	145	140	140	140
Decreasing $\gamma_0 = 0.8$, $\lambda = 0.1$	170	165	160	160	160

Table 1: Rank loss after i iterations with different step sizes. Computed on the Google 3D Warehouse dataset (compare 3.3.2).

As an extension to the stochastic gradient descent, which picks one training observation per iteration, we use a batched stochastic gradient descent as described in [Bottou 2003]. The idea is to use a compromise between the expensive but more stable non-stochastic gradient descent, and the often faster but rather unstable stochastic gradient descent. In each iteration, we pick a fixed number of $b \in \mathbb{N}$ training observations $x_1, \ldots, x_b$ randomly. We compute the corresponding gradients individually, and then use their mean to actually update the classifier. In a limited quantitative evaluation on the Google 3D Warehouse dataset, we found $b = 32$ to yield good speed and robust convergence (compare Table 2). Larger batch sizes converged faster initially, but remained unstable and did not fully converge during this experiment. Note that in this experiment, we adapted the step size proportionally with the batch size, in order to get a benefit from the increased stability of larger batches. Individually tuning the step size for a given batch size would likely have yielded better results, and the decreasing convergence stability for $b = 128$ indicates that proportionally increasing the step size with the batch size might be too much. Our goal here is simply to find a reasonable value for achieving acceptable convergence speeds, and not to perform an in-depth analysis of the effects of different batch sizes.

	30 sec	1 min	2 min	4 min	8 min
$b=1$	185	170	150	125	115
$b=8$	155	135	120	115	110
$b=32$	145	120	120	115	115
$b=128$	140	125	125	125	125

Table 2: Comparison of batch sizes: The table shows the average rank loss (smaller = better) after a given amount of time for different batch sizes. The step size γ was increased proportionally with the batch size b, with $\gamma=0.003125$ at $b=1$ ($\gamma=0.1$ at $b=32$). Times were taken on an Intel Xeon X5650 system. Computed on the Google 3D Warehouse dataset (compare 3.3.2).

3.2.2 Utilizing Ground-Truth Label Correlation (Soft Ranking)

When using crowd-annotated data sets for training a latent semantic space, there is a problem of having many different, but semantically equivalent labels. For example, different objects might be labeled in different languages, such as “tree” and “baum” (German for tree). So far, our training algorithm is unaware of such cases. While it would – given a sufficient number of training samples – be forced to map “tree” and “baum” to similar places in the latent space due to them being indistinguishable from each other in the descriptor space, it would still assign an error to an object labeled with “tree” if it was classified as “baum” by the current classifier. In effect, the algorithm would try to separate “tree” from “baum” just as much as it would try to separate “tree” from “airplane”. This is a problem especially in cases where the number of training samples for a specific label (e.g. for “baum”) is small, and the training algorithm could therefore actually succeed in separating “baum” from “tree” through over-fitting. To remedy this issue, we can utilize the correlation between labels in the ground-truth data set. As some of the people contributing to databases such as Google 3D Warehouse want to maximize the visibility of their contributions in search results, they tend to tag their models with a number of synonymous labels. We can make use of this information, by calculating the correlation between labels in the ground-truth label set. Specifically, if $Y_x \subset Y$ are the ground-truth label sets for the training samples $x \in S$, we calculate for each pair of labels $y, y' \in Y$ the following statistical values:

$$mean_y := \frac{1}{|X|} \sum_{x \in S} I(y \in Y_x) \quad \text{where } I \text{ is the indicator function}$$

$$cov_{y,y'} := \frac{1}{|X|-1} \sum_{x \in S} \left(I(y \in Y_x) - mean_y \right) \left(I(y' \in Y_x) - mean_{y'} \right)$$

$$corr_{y,y'} := \frac{cov_{y,y'}}{\sqrt{cov_{y,y} cov_{y',y'}}}$$

We then integrate the correlation values into the loss function. For this, we add an additional weight term w to the loss:

$$L^1_w(x_i) := w(x_i, \bar{y}_i) \sum_{j=1}^{rank^1_{y_i}(x_i)} \alpha_j$$

We tried two variants for adapting the weight based on label correlation:

1. $w_{max}(x_i, y) := 1 - \max\left\{0, \max_{y' \in Y_{x_i}} corr_{y,y'}\right\}$
2. $w_{mean}(x_i, y) := 1 - \frac{1}{|y_{x_i}|} \sum_{y' \in Y_{x_i}} corr_{y,y'}$

Note that w_{mean} can also reach values >1 if the average correlation is negative, while w_{max} will result in weights in $[0,1]$ and ignore negative correlations.

We evaluate this "soft ranking" approach in chapter 3.3.3.

3.2.3 Aliasing Labels to Enable Non-Linear Decision Boundaries

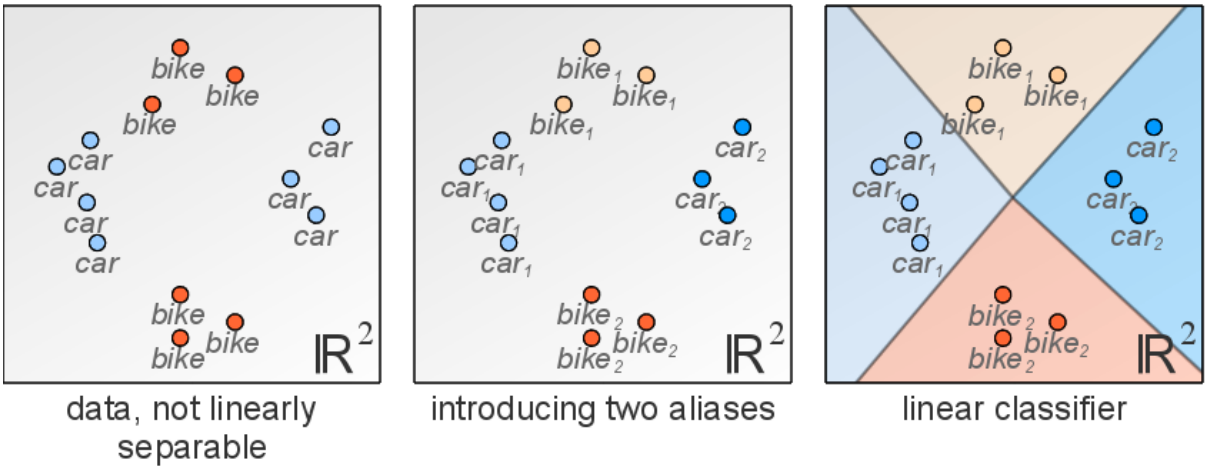


Illustration 19: Label aliasing allows to learn a linear classifier for non-linearly separable classes of data.

So far we have based the classification of an object on the ranking of the linear mapping $y^T U^T \cdot V x$, which allows us to distinguish two labels only if they are linearly separable in the object's descriptor x. Furthermore, we are also interested in keeping the dimensionality of the latent space representation low, not only for computational reasons, but also to exploit the semantic overlap between labels by means of sharing training observations between different but semantically related labels. This kind of sharing leads to a better generalization performance of our classifier (see our evaluation in chapter 3.3.3). On the other hand, some less frequent or geometrically complex labels might not be well predictable in a low-dimensional latent space, as all available dimensions are "used up" by the more frequent dominant labels. While we could increase the dimensionality of the latent space to support many more labels, we would lose generalization performance of the already well-predicted labels. Here, we propose a simple yet powerful solution for implicitly adapting the dimensionality per label, which we call *label aliasing*. Label aliasing is based on the idea of splitting "difficult" labels into piecewise linear classification sub-tasks. At the same time we keep the advantage of having a scalable and interpretable linear classifier.

So far we have had a set of labels $Y \subset \{0,1\}^{|Y|}$. We now allow each individual label $y \in Y$ to be represented by a number of label aliases $y^1, \ldots, y^n$. The number of aliases can be different for different labels $y \in Y$ (but not smaller than one), and most labels will still be represented by only a single alias. We denote the set of all aliases by $Y^A \subset \{0,1\}^{|Y^A|}$, and we represent aliases just like labels as vectors with a one in exactly one component and zeros everywhere else. While the classifier $\Phi = U^T V$ has so far been a mapping from descriptors into $\mathbb{R}^{|Y|}$, we now extend it to map into the higher-dimensional space $\mathbb{R}^{|Y_A|}$. While the aliases $y^1, \ldots, y^n$ are just different ways of representing the semantic label y (we will later slightly extent this notion, interpreting aliases as subclasses of a common semantic concept), each alias can be mapped to a completely different position in the latent space. We have two contradicting goals which we now want to solve: 1. We want to improve the classification of the training data by splitting "difficult" labels into multiple aliases. However, 2. we also want to avoid introducing more aliases for any given label than necessary, because this can easily lead to over-fitting. In the extreme case, we could split each label into as many aliases as there are training samples that have this label in their ground-truth label set. This would yield perfect training performance, but the resulting classifier would not generalize well to previously unseen data (essentially the WSABIE classifier would degenerate into a kNN classifier with $k=1$).

This problem casts two sub-problems:

1. We need to identify those "difficult" labels that require splitting. An example could be labels such as "old" or "sporty", which can have vastly different geometric features depending on the semantic context they appear in. As a consequence, such labels are typically not linearly separable from their negations ("young", "non-sporty").
2. We need to decide about the number and initialization of their aliases.

To address the question of which labels should be split, we use a score inspired by the "recall" metric often used to assess information retrieval performance. We call this score the *coverage* of a label. For any given label $y \in Y$, the coverage c_y is the number of all objects x_i classified by the currently trained classifier as having label y, divided by the total number of such objects that have y as a label in their ground-truth labeling Y_{x_i}. Intuitively, c_y tells us how many of the objects actually labeled with y are "covered" by the classifier's current interpretation of y. Note that we have not yet defined what it means for an object x_i to be classified as having label y. In order to do so, we have to convert the ranking of all labels produced by applying $U^T V$ to x_i into a binary classification, that is a set of labels predicted by the classifier. The following gives the detailed steps of how c_y is computed:

1. for each object x_i for which $y \in Y_{x_i}$, calculate $scores_{x_i} = U^T V x_i$
2. rank the scores and select the k top-ranked labels, where $k = |Y_{x_i}|$
3. if y shows up in the k top-ranked labels for x_i, we say that y *covers* x_i
4. the overall coverage value c_y is the number of all objects covered by y divided by the number of all objects x_i for which $y \in Y_{x_i}$

As the space of possible classifiers is significantly increased by allowing for labels to be aliased, over-fitting becomes an important concern. To reduce the risk of over-fitting, we partition the training data into two disjoint cross-validation sets. We use the first subset to train the classifier, and the second one for computing coverage values.

A low coverage c_y indicates that only a subset of the objects labeled with y are reliably predicted. We identify labels with low coverage and introduce a configurable number of aliases $y^1, \ldots, y^n$ for each of these labels y. The goal of this approach is to allow a single label y to be mapped not only to a single, but to multiple positions in the latent space. These aliases are now included in the general optimization scheme just as if they were distinct labels by extending the label embedding matrix U with new randomly initialized rows. Once aliases have been introduced, we continue training the classifier in an online fashion using a slight modification of the previously described stochastic gradient descent algorithm.

The modification is as follows: In each iteration, we pick a training sample x_i as before (compare chapter 3.1.3). We then check for the sample x_i if it has any label in its ground truth set Y_{x_i} which has previously been aliased. We now assume the following model: We assume that the aliases $y^1, \ldots, y^n$ of a label y are actually a number of distinct sub-categories of the original label (for example if the original label is "sporty", we imagine that there are sub categories such as "sporty shoes", "sporty car", "sporty human" etc.). We do not know which of the sub-categories x_i actually belongs to, because the ground-truth label set Y_{x_i} only contains the original label y. We could assume that x_i belongs to all of the sub-categories, but that would make it difficult for the aliases to differentiate themselves in the latent space (they would all converge to the same location). Also, this would effectively increase the weight that the aliased label has during the optimization, as it would be sampled n times as often as non-aliased labels, where n is the number of aliases. A slight adaptation of this would be to assume that x_i belongs to any single one of the sub-categories with equal chance. While this would maintain the weighting of the aliased label, it would still not allow the different aliases to converge to different locations. Additionally, none of the aliases would be consistently correct for a given object over a number of iterations. As a consequence, the optimization would try to avoid all aliases, and likely embed them into the latent space very close to the zero position, at which point they become useless. Instead of these simple schemes, we use a probability $P(y^j \in Y_{x_i})$ which increases with the similarity $V x_i \cdot U y^j$ of y^j to x_i in the latent space. The idea is that once an alias has been converged to be close to a group of objects labeled with the corresponding label, those objects have a high chance of actually being in the sub-category represented by this alias. Starting with randomly initialized alias positions in the latent space, this has the effect that aliases "snap in" to clusters of objects, and will be drawn more and more towards them as they get closer. Different aliases will be assigned to different such clusters. For this to work, we assume that objects for a certain sub-category of the original label are already close together in the latent space, and therefore will attract their very own alias. So how exactly should we define $P(y^j \in Y_{x_i})$? In the simplest case, we could perform a hard "maximum likelihood" assignment. For a given x_i , we would locate the alias $y^{\max}$ with the highest similarity in the latent space. We would then define $P(y^{\max} \in Y_{x_i}) = 1$ and $P(y^j \in Y_{x_i}) = 0 \; \forall \; j \neq \max$. However, this approach leads to artifacts due to its discontinuities with respect to varying latent space embeddings. For

example, aliases can get stuck at the discontinuities of this function, oscillating between being and not being the highest-similarity alias for a certain object. In general, we would like to have $P(y^j \in y_{x_i})$ such that

1. $P(y^j \in Y_{x_i}) < P(y^k \in Y_{x_i})$ whenever $V x_i \cdot U y^j < V x_i \cdot U y^k$
2. $P(y^j \in Y_{x_i}) \geq 0 \; \forall \, y^j, \sum_{y^j} P(y^j \in Y_{x_i}) = 1$
3. $P(y^j \in Y_{x_i})$ is independent of the global linear scaling of all similarities / distances in the latent space. This is a useful property because it eliminates the need for an additional parameter that would have to be adjusted depending on the data set and latent space regularization used. Specifically, if we replace the similarities $V x_i \cdot U y^j$ by a rescaled variant $\alpha V x_i \cdot \beta U y^j = \alpha \beta \left(V x_i \cdot U y^j \right)$ for scaling constants $\alpha, \beta > 0$, all $P(y^j \in Y_{x_i})$ should remain the same.

Criteria 1 can be easily fulfilled by using a monotonically increasing mapping of the similarity. Criteria 2 can usually be fulfilled through renormalization. This becomes especially easy if we first convert the similarities into distances (such that they are always positive). For the similarity $s(x_i, y^j) V x_i \cdot U y^j$ a related distance is given by

$$d(x_i, y^j) := \sqrt{s(x_i, x_i) + s(y^j, y^j) - 2\, s(x_i, y^j)}$$

(construction from [Wikipedia MDS 2013]). Criteria 3 is slightly more challenging to fulfill. For example, using normalized Gaussians of the distances would not fulfill this property. Instead, we use the following function:

$$P(y^j \in Y_{x_i}) := \frac{1}{\sum_{y^k} \left(d(x_i, y^k)^m + \epsilon \right)^{-1}} * \left(d(x_i, y^j)^m + \epsilon \right)^{-1}$$

with the blue part serving as a normalization factor, and $m > 0$ determining how steeply the probability decreases (we found $m = 2$ to work well for our purposes). For $\epsilon = 0$, criteria 3 is met. The proof goes as follows:

Assume that we have distances $\alpha d_1, \ldots, \alpha d_n \geq 0$ with $\alpha > 0$. Then we have to show that

$$\frac{1}{\sum_{1 \leq k \leq n} \frac{1}{(\alpha d_k)^m}} * \frac{1}{(\alpha d_i)^m}$$

being the definition of $P(y^j \in Y_{x_i})$ with the distances $\alpha d_1, \ldots, \alpha d_n$ and $\epsilon = 0$ substituted, is the same for all choices of $\alpha > 0$. We simplify:

$$\begin{aligned}
&\frac{1}{\sum_{1 \leq k \leq n} \frac{1}{(\alpha d_k)^m}} * \frac{1}{(\alpha d_i)^m} \\
&= \frac{1}{\alpha^{-m} \sum_{1 \leq k \leq n} d_k^{-m}} * \frac{1}{\alpha^m d_i^m} \\
&= \frac{\alpha^m}{\sum_{1 \leq k \leq n} d_k^{-m}} * \frac{1}{\alpha^m d_i^m} \\
&= \frac{1}{\sum_{1 \leq k \leq n} d_k^{-m}} * \frac{1}{d_i^m}
\end{aligned}$$

which does not depend on α anymore.

Unfortunately with $\epsilon = 0$, $P(y^j \in Y_{x_i})$ is undefined if any of the distances becomes zero, and becomes numerically instable if very small distances are involved. Using a small $\epsilon > 0$ avoids these issues, and the resulting probabilities still *almost* fulfills criteria 3, as long as ϵ remains much smaller than most of the distances.

Now, having a way to stochastically estimate which alias of a given label $y \in Y_{x_i}$ represents the actual sub-category that x_i belongs to, we derive a new ground-truth label set Y_{x_i}' which contains exactly one alias for each label in the original ground-truth label set Y_{x_i}. The aliases are sampled randomly based on the just defined probability distribution $P(y^j \in Y_{x_i})$. Y_{x_i}' is then used instead of Y_{x_i} in the remaining parts of the current stochastic gradient descent iteration.

In order to adaptively tune the number of aliases for each attribute throughout the optimization, we use the following approach: First, we train the latent space with just one alias per attribute for a certain number of iterations. Then, we calculate the coverage values of all attributes. We pick one that has a low coverage, and introduce a fixed number of aliases (e.g. 16). The new aliases are initially embedded at random locations in the latent space. We then run a smaller number of iterations to optimize the embedding of new alises and then re-calculate the coverage values. If the newly introduced aliases contribute to the coverage of the corresponding attribute, we keep them. If they do not contribute significantly, they get removed. Additionally, we remove aliases that cover mostly the same set of observations as other aliases. This usually happens when multiple aliases of an attribute get attracted to the same cluster of observations and converge to the same position in the latent space. We then

repeat the last two steps – introducing and cleaning up aliases in alternating steps – with a fixed number of gradient descent iterations between each, until all low-coverage attributes have had a chance to introduce new aliases.

Label aliasing indirectly allows us to learn non-linear classifiers. Illustration 19 shows an example of training data which can not be classified well by any linear classifier, but can be fit perfectly after introducing two aliases for the labels “car” and “bike” respectively. In this respect our method is similar to kernel methods. Kernel methods are based on the following idea: Instead of training a classifier $f : \mathbb{R}^d \to \{-1, 1\}$ on the descriptor space $\mathbb{R}^d$, the descriptors are mapped into a different space $\mathbb{R}^D$ through a mapping $\varphi : \mathbb{R}^d \to \mathbb{R}^D$, where D can be different from d, and even be infinite. φ does not necessarily have to be linear, and a linear classifier $f' : \mathbb{R}^D \to \{-1, 1\}$ on $\mathbb{R}^D$ can in effect be non-linear when applied to $\mathbb{R}^d$ as $f'(\varphi(x))$. The so called “kernel trick” is to avoid the mapping into $\mathbb{R}^D$ in the first place. Instead, using the inner product $\langle \circ, \circ \rangle$ on $\mathbb{R}^D$, a kernel $k := \langle \varphi(\circ), \varphi(\circ) \rangle$ on $\mathbb{R}^d$ is derived. In some cases, k can be compactly expressed and efficiently computed, even if φ itself is very complex or $\mathbb{R}^D$ is very high dimensional. A frequently used kernel is the Gaussian kernel

$$k_{Gauss}(x, y) := e^{\frac{-\|x-y\|^2}{2\sigma^2}}$$

which – despite being efficiently computable – can be shown to correspond to a transformation into an infinite dimensional space $\mathbb{R}^D$. Some learning schemes can be expressed in a way which depends solely on calculations of inner products of the data points. By replacing the inner product by the corresponding kernel, the potentially expensive transformation φ can be avoided. A variant of SVMs called Kernel SVM is frequently used. Unfortunately a kernel SVM still requires the kernel function to be applied to all pairs of training data samples, leading to quadratic learning costs. Our method does not suffer from this performance hit. Our classifier remains purely linear, allowing for efficient predictions. While the dimensionality of the prediction space is increased through the introduction of aliases, which does effect the performance of the classifier, this increase is adaptive and only occurs for labels of otherwise low coverage. Finally, our method also provides a nice interpretation of the data. Label aliases are designed to correspond to geometrically distinct groups of the corresponding label. For example, the subcategories “sporty shoe” and “sporty car” of the label “sporty” can be detected and will be explicitly represented by our method. Kernel methods generally do not provide an interpretable result in the same way.

3.3 Evaluation

We would like to know how well a classifier performs in predicting labels for previously unknown observations. We will first look at available evaluation metrics, which give us a quantitative way of comparing classifier performances. We then evaluate different configurations and variations of the WSABIE method (see chapter 3.1.3) with respect to these metrics on 3D mesh and photo data sets.

3.3.1 Evaluation Metrics

We would like to numerically compare the quality of different classifiers. Assume that we have a set of test observations $S \subset \mathbb{R}^d$ together with a ground-truth label sets for each of those observations $Y_x \subset Y \; \forall \, x \in S$. The (trained) multi-class classifiers take the form $f \in \mathbb{R}^d \to \mathbb{R}^{|Y|}$. In general, we want to define a metric $m_S : (\mathbb{R}^d \to \mathbb{R}^{|Y|}) \to \mathbb{R}$, which – given a classifier f – tells us how well the predictions made by f on S match the ground truth labels Y_x. Which metric makes most sense highly depends on the requirements that we have for a specific application. We will start with a motivational example and then introduce a few standard metrics commonly used to assess information-retrieval methods.

A very simple metric could count how often the highest-ranked result $\underset{1 \le i \le |Y|}{max} f_i(x)$ for a sample $x \in S$ is also in the associated ground-truth set Y_x:

$$m_{\text{top},S,t}(f) := \frac{1}{|S|} \sum_{x \in S} I\left(\left(\underset{1 \le i \le |Y|}{max} f_i(x) \right) \in Y_x \right)$$

where I is the indicator function and we write $\left(\underset{1 \le i \le |Y|}{max} f_i(x) \right) \in Y_x$ to mean that the label y_j associated with the highest-ranked position in the prediction vector $j := \underset{1 \le i \le |Y|}{argmax} f_i(x)$ is an element of Y_x. Unfortunately this metric discards a lot of information. Clearly, a classifier which puts the first correct label at rank two should be preferred over one which puts it at rank 50, all other predictions being equal. For data sets with many detailed labels (e.g. adjectives rather than just labels denoting disjoint object categories), different labels in the label set of an object can describe different orthogonal attributes of the object. Only looking at the highest-ranked result would ignore such orthogonal elements of information in the evaluation of the classifier.

An obvious candidate for an evaluation metric given the WSABIE algorithm from [Weston et al. 2011] is the rank loss with weights

$$\alpha_j = \frac{1}{j} * \left(\sum_{j'=1}^{|Y|-1} \frac{1}{j'} \right)^{-1}$$

for the j-th rank of the prediction as defined in chapter 3.1.3. However this evaluation metric is not commonly used, and would likely give the WSABIE method an unfair advantage over other methods. [Weston et al. 2011] themselves evaluate their method by looking at the precision metric, which we introduce next. Later, we also look at the discounted cumulative gain (DCG) metric, which is similar to a rank loss, but uses a different counting and weighting scheme.

Precision-recall plots are commonly used in information retrieval settings. The idea is to explicitly express a trade-off between finding as many true results as possible on the one hand, and finding as few false results as possible on the other hand. In an extreme case, an information retrieval method might simply return all objects from the database. All true objects will certainly be among those. However there will be a lot of false-positives as well. The former aspect is measured by the *recall*, while the latter is captured by the *precision*. In

the other extreme, a method might return no object at all. None of the desired objects would be in the set, but the method would also never return a false object by mistake. Formally, the recall is defined as

$$recall := \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

and precision as

$$precision := \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Similarly, when we want to assess the quality of label predictions, we can consider the predicted labels as retrieved objects, and the initial descriptor as a query given to a retrieval method. Given the classifier f as defined earlier, we can rank all labels in the label set Y based on the scores computed by $f(x)$. We can then select a subset $r_k \subset Y$ of "retrieved" objects by taking just the k highest ranked labels. The set of "relevant" objects on the other hand is given by the associated ground-truth label set Y_x. This leads to the following metrics:

$$recall_{k,x} := \frac{|r_k \cap Y_x|}{|Y_x|}$$

and

$$precision_{k,x} := \frac{|r_k \cap Y_x|}{|r_k|}$$

with

$$r_k := \{ y \in Y \mid rank_y(f(x)) < k \}$$

These metrics apply to a specific test sample $x \in S$. To obtain a value for the whole test set S, we take the average recall and precision values:

$$recall_{k,S} := \frac{1}{|S|} \sum_{x \in S} recall_{k,x}$$

$$precision_{k,S} := \frac{1}{|S|} \sum_{x \in S} recall_{k,x}$$

Note that for $k = 1$, the precision is equivalent to our earlier $m_{\text{top},S,t}$ metric. In many settings, we know approximately how many results are important, and we can take the precision for this specific k to compare different classifiers. If we do not want to fix k, we can vary k and obtain a recall-precision curve. The recall will increase (non-strictly) monotonically with increasing k, while the precision will decrease (non-strictly) monotonically. The area under this curve can then be calculated to obtain a single real-valued quality measure for a multi--label classifier.

An alternative way to write $recall_{k,x}$ can be obtained by assigning a gain $G_x[i]=1$ whenever the prediction at rank i is correct and $G_x[i]=0$ otherwise. With these specific gains, we get

$$recall_{k,x}=\sum_{i=1}^{k} G_x[i]$$

For general gains $G_x[i]\in\mathbb{R}^+$, the value $\sum_{i=1}^{k} G_x[i]$ is known as the *cumulated gain* $CG_x[k]$. Originally, [Järvelin et al. 2000] defined the cumulated gain recursively. However we think that the explicit definition given here is more intuitive to read. The specific values $G_x[i]$ can be chosen to represent the relevance of a retrieved document or predicted label respectively in a quantitative way. In our setting, we could for example define certain labels to be more relevant for our specific problem than others. Or we could make use of the ground-truth label correlation (compare chapter 3.2.2) to assign a larger-than-zero gain also to labels which are not in Y_x, but have correlated labels present in Y_x. For example, if a mesh x has $Y_x=\{\text{car},\text{fast}\}$, a prediction of the label vehicle would still provide a certain "gain" with respect to defining the semantics of the object. For simplicity reasons and to make our numbers easier to interpret, we do not use anything of this sort for our evaluation.

[Järvelin et al. 2000] introduce a refinement of the cumulated gain which has the property that – similar to the rank loss used in [Weston et al. 2010] – it assigns more weight to higher-ranked predictions. The *discounted cumulative gain* is defined as

$$DCG_x[i]:=G_x[1]+\sum_{i=2}^{k}\frac{G_x[i]}{\log_b i}$$

The basis of the logarithm b can be used to adjust the weight distribution more or less strongly towards high-ranked gain. Note that if $b>2$, $G[2]$ will actually receive a higher weight than $G[1]$, often making such settings undesirable. We use $b=1$ in our evaluations. The DCG is especially useful because – similar to the area under a precision-recall curve – it can provide a single number to assess the prediction quality without having to specify k in advance. $DCG_x[|Y|]$ assigns higher values to classifiers which place correct labels at higher positions in the ranking. It is also possible to normalize the DCG through division by the highest achievable DCG given the data. The value can be obtained by computing $DCG_x[|Y|]$ for a ranking which is ordered in decreasing order of the gains for the corresponding labels, e.g.

$$DCG_{max,x}[i]=1+\sum_{j=2}^{|Y|}\frac{1}{\log_b(j)}$$

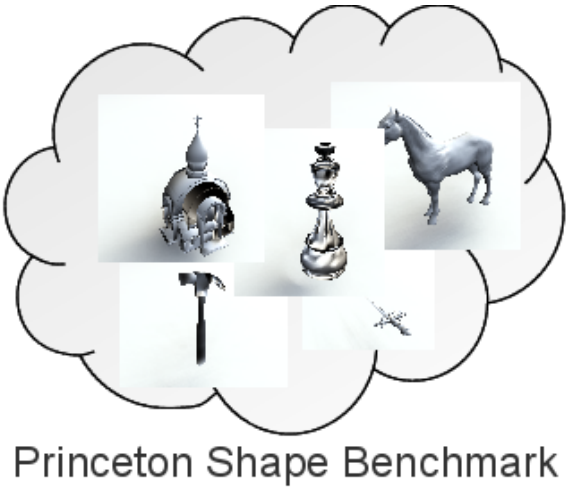
for the setting where $G_x[i]=1$ whenever the prediction at rank i is correct and zero otherwise.

Again, we can use an average DCG to assess the whole test set S in a single number:

$$DCG_S:=\frac{1}{|S|}\sum_{x\in S} DCG_x[|Y|]$$

3.3.2 Data Sets

We use two different 3D mesh datasets and one photo dataset for our evaluation.



freq.	label
0.18	aircraft
0.17	animal
0.11	airplane
0.10	furniture
0.09	human
0.09	biped
0.07	plant
0.06	vehicle
0.06	body_part
0.06	fighter_yet

Illustration 20: The Princeton Shape Benchmark data set. Example shapes and top labels.

Our first data set is the Princeton Shape Benchmark, described in [Shilane et al. 2004]. The Princeton Shape Benchmark is a widely used data set for evaluating mesh retrieval and classification methods. It contains a total of 1,814 individual meshes from different areas, split into a test and a training set of 907 meshes each. Each mesh is tagged by a single label. The labels itself are organized in a hierarchy, from more coarse (“aircraft”, “building” etc.) to very specific (“biplane airplane”, “commercial airplane”, “multi_fuselage airplane” etc.). We make use of this hierarchy and enrich the single-label tagging of each object with its more generic parents to obtain sets of multiple ground-truth labels. Furthermore, we filter out labels which have fewer than five instances in the training set (this affects some of the most specific labels). After the filtering, 117 labels remain, with an average number of 2.4 labels assigned to each object. The ten most frequent labels are listed in Illustration 20. We build a 512 dimensional bag of features descriptor (compare 2.1.3), based on HON descriptors with 4x4 spatial bins and 8 rotational bins (compare 2.2.1).

freq.	label
0.10	sport
0.10	house
0.10	car
0.09	modern
0.08	computer
0.06	old
0.06	table
0.05	human
0.05	boat
0.05	skateboard

Illustration 21: The Google 3D Warehouse data set. Example shapes and top labels.

Our second data set of 3D meshes was acquired by querying the Google 3D Warehouse database for a number of search terms one at a time and downloading the resulting meshes together with their tag sets. Google 3D Warehouse is a crowd-generated, public database of 3D meshes. Users can upload their own work, and assign text labels. In our experience, the quality of both the meshes and the assigned label sets varies widely (see Illustration 11 for an example of a bad mesh and label set). This makes learning label semantics on this data set especially challenging. Furthermore, many of the meshes contain compositions of multiple objects (compare Illustration 7). We downloaded a data set of 1,939 meshes, which we split randomly into a training set of 1,155 meshes and a test set consisting of 784 meshes.

animal, balcony, bicycle, boat, bottle, cathedral, car, castle, cat, chair, computer, dog, gothic, guitar, helicopter, house, human, insect, laptop, living room, modern, motorcycle, old, plane, shoe, skateboard, spider, sport, sporty, streetlamp, table, traditional, tree, window

Text 1: Search terms used to build our Google 3D Warehouse dataset.

The search terms that we used to crawl the data set are listed in Text 1. As with the Princeton Shape Benchmark, we filtered out low-frequency labels which occurred less than 10 times in the training set. 582 labels remained, with an average of 7.6 labels per mesh. The ten most frequent labels are listed in Illustration 21. We use the same descriptor as for the Princeton Shape Benchmark, building a 512 dimensional Bag of Word descriptor from 4x4x8 HON descriptors. Note that while many of the meshes are textured, we do not currently make use of the textures and consider the geometry only.

freq.	label
0.19	sky
0.17	building
0.15	car
0.14	road
0.14	tree
0.14	window
0.12	person
0.10	sidewalk
0.08	sign
0.07	trees

Illustration 22: The LabelMe data set. Example pictures and top labels.

For photos, we use the LabelMe dataset ([Russell et al. 2008]). Like Google 3D Warehouse, LabelMe consists of pictures collected from a wide variety of sources. The data set as we downloaded it consists of 37,038 images, split into a training set of 18,538 and a test set of 18,500 pictures. In LabelMe, users can select regions of a photo and assign labels to these regions. We discard the region information, but use the set of all labels assigned to any region within a given picture as the label set for that picture. As before, we remove labels with a frequency of less than 10 in the training set. After filtering, we are left with 1155 labels, of which an average number of 5.0 are assigned to each image. The ten most frequent labels are listed in Illustration 22. As a descriptor, we first compute HOG descriptors with 6x6 spatial and 16 rotational bins (compare 2.1.1). We then build a dictionary and compute a 512 dimensional bag of features descriptor (compare 2.1.3) for each image. Finally, we append a 5x5 downsampled RGB-encoded version of the image to the Bag of Words descriptor to encode the rough color composition of the images. The final descriptor has a dimensionality of 587.

3.3.3 Learning Methods

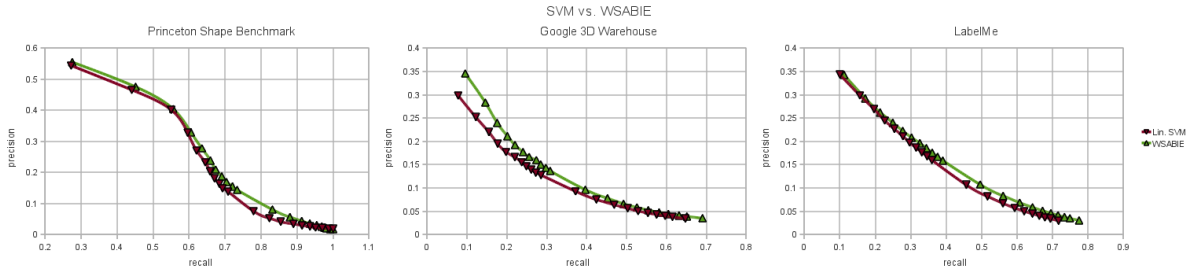


Illustration 23: Linear SVM versus WSABIE: The graphs show the precision-recall values obtained with a one-versus-rest linear SVM classifier and a WSABIE classifier on the Princeton Shape Benchmark, Google 3D Warehouse and LabelMe data sets. See Table 3 for additional details.

We compare the label prediction results of the WSABIE method to linear SVMs trained in a one-versus-rest scheme. For implementing the SVM we use the LIBLINEAR library ([Fan et al. 2008]). We slightly modified the partitioning part of the one-versus-rest scheme, so it can cope with multiple labels per object. To train an SVM to classify between the label $y \in Y$ and the remaining labels $Y - \{y\}$, we compose the set of training observations for the first class by picking all objects that have y in their ground-truth label sets, and for the second ("rest") class by taking the remaining observations in the training set. The soft margin implementation used in LIBLINEAR is tunable through a single C parameter. We optimize C for each data set by searching through a range of candidate values. We use 4-fold cross validation to select the C which generates the highest precision at $k = 3$. The results of the experiment are shown in Illustration 23. WSABIE was roughly equivalent to the linear SVM method on the LabelMe data set, showed slight performance improvements on the Princeton Shape Benchmark, and could demonstrate significant benefits over SVM on the Google 3D Warehouse data set. Objects in the Google 3D Warehouse data set have very noisy, incomplete and inconsistent label sets. The low-dimensional latent space and the weighted rank-based loss function of the WSABIE method make it especially robust under such conditions.

	Princeton Shape B.			**Google 3D Warehouse**			**LabelMe**		
	p@1	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**
Linear SVM	0.54	0.27	1.26	0.30	0.18	1.19	0.34	0.23	1.43
WSABIE	0.56	0.28	1.31	0.35	0.19	1.26	0.34	0.23	1.44

Table 3: Linear SVM versus WSABIE: Precision and DCG values. p@1 / p@5 denote the precision at the top ranked prediction and over the five top ranked predictions respectively. Also compare Illustration 23.

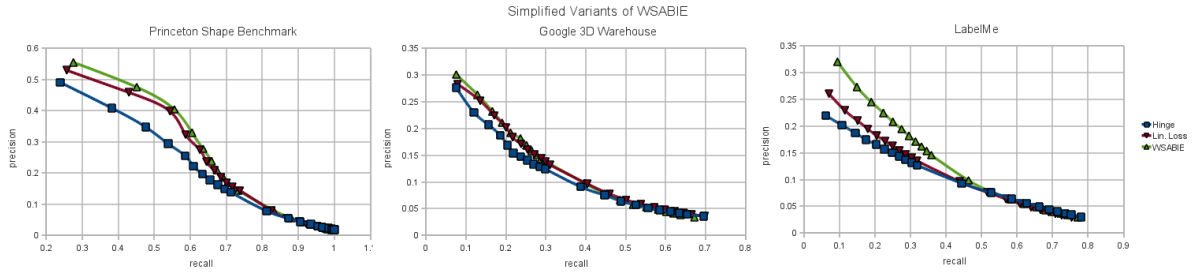


Illustration 24: Simplified variants of WSABIE: We try different simplifications of its loss function. The graphs show the precision-recall plots for those variants on the Princeton Shape Benchmark, Google 3D Warehouse and LabelMe data sets. See Table 4 for additional details.

In order to measure the benefit of different design choices made in the WSABIE method, we evaluate simplified variants of it. Remember the original loss

$$L_{rankHinge}(x_i) := L^1(x_i) \sum_{y' \in Y - \{y_i\}} \frac{\left|1 - \Phi_{y_i}(x_i) + \Phi_{y'}(x_i)\right|_+}{rank^1_{y_i}(\Phi(x_i))}$$

from the WSABIE method (compare chapter 3.1.3).

First, we remove all rank-based parts of the loss, to obtain a regular hinge loss ("Hinge"):

$$L_{hinge}(x_i) := \sum_{y' \in Y - \{y_i\}} \left|1 - \Phi_{y_i}(x_i) + \Phi_{y'}(x_i)\right|_+$$

The resulting learning method is close to the objective of a linear SVM, but still uses a low-dimensional latent space as an intermediate representation. We also evaluate a variant of the rank loss where violations have the same weight independent of the rank that they occur at ("Lin. Loss"). This is equivalent to using L_{lin} as defined earlier in 3.1.3 in the loss function:

$$L_{linHinge}(x_i) := L_{lin}(x_i) \sum_{y' \in Y - \{y_i\}} \frac{\left|1 - \Phi_{y_i}(x_i) + \Phi_{y'}(x_i)\right|_+}{rank^1_{y_i}(\Phi(x_i))}$$

Illustration 24 shows the results of the experiment. The label prediction performance on all tested data sets showed a significant improvement when changing from a the pure hinge loss L_{hinge} to the linear rank based loss $L_{linHinge}$. Results on both the Princeton Shape Benchmark as well as on Google 3D Warehouse improved slightly more with the introduction of the decreasingly weighted ranks used in the WSABIE method ($L_{rankHinge}$). On LabelMe, the performance improvement of this change was even more dramatic. A possible explanation for the increased change on the LabelMe data set might be its large number of labels, which lead to an increased difference between the values of $L_{linHinge}$ and $L_{rankHinge}$. The experiment shows that the design choices made in the design of the WSABIE loss function do indeed provide measurable performance improvements on the tested data sets.

	Princeton Shape B.			Google 3D Warehouse			LabelMe		
	p@1	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**
Hinge Loss	0.49	0.26	1.20	0.28	0.17	1.16	0.22	0.17	1.22
Lin. R. Loss	0.53	0.27	1.29	0.28	0.18	1.23	0.26	0.18	1.26
WSABIE	0.56	0.28	1.31	0.30	0.19	1.25	0.32	0.21	1.35

Table 4: Simplified variants of WSABIE: Precision and DCG values. p@1 / p@5 denote the precision at the top ranked prediction and over the five top ranked predictions respectively. Also compare Illustration 24.

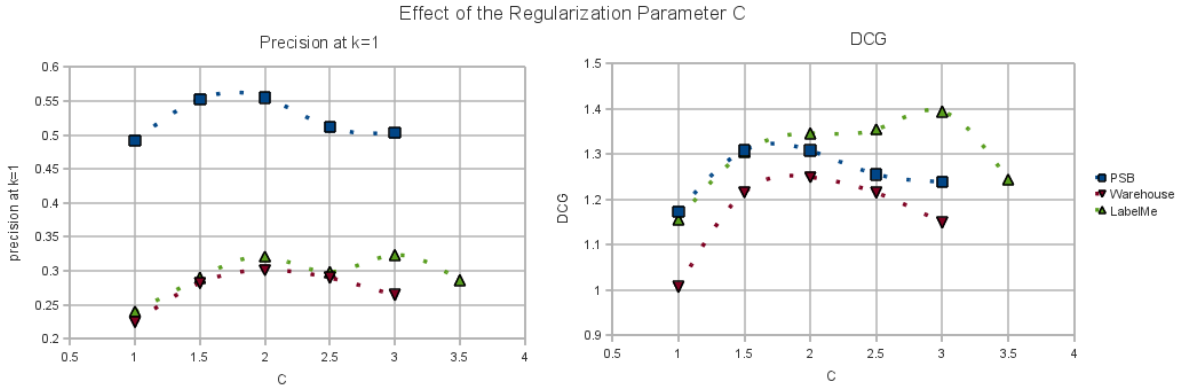


Illustration 25: Effect of the regularization parameter C on the label prediction performance of the WSABIE method.

We evaluate the two major parameters of the WSABIE method. First, there is the C regularization parameter which limits the magnitude of the column vectors of the mappings V and U, and thereby indirectly defines the relative size of the margin of the loss function (see chapter 3.1.3 for details). Illustration 25 shows the results of our experiment. All remaining parameters were kept fixed throughout this experiment. Specifically we used latent space dimensions $D=32$ for the Princeton Shape Benchmark and $D=128$ for the other data sets. We assessed the label prediction performance against the test sets of the Princeton Shape Benchmark, our Google 3D Warehouse data set, and the LabelMe data set. The parameter C was varied in steps of 0.5 in a range between 1 and 3 (3.5 in the case of LabelMe). In general, larger values of C lead to smaller training risks (not shown here). However when using the test sets, it becomes evident that this reduction in the training risk after some point can be contributed to over-fitting of the training data. The peak DCG was obtained at $C=2.0$ for the Princeton Shape Benchmark and the Google 3D Warehouse data set, and at $C=3.0$ for the LabelMe data set.

Similarly, we performed an experiment were the latent space dimension D was varied between values of 16 and 512. Because 512 is also the dimensionality of the Bag of Features descriptors we use, higher values cannot actually increase the rank of the classifier $\Phi=V^T U$ any further. This experiment serves two main purposes: First, we want to test our hypothesis

from chapter 3.1.3 that the label space of our data sets can indeed be reduced to a lower-dimensional structure without losing much information. Second, the choice of the parameter D involves an important compromise between two contradicting goals, and we would like to find out what its ideal value is for the given data: On the one hand, a small dimensionality of the latent space comes with a loss of information when mapping the descriptor of a given test object into it, and we would like to preserve as much information as possible. On the other hand, a high-dimensional latent space increases the degrees of freedom during the optimization, and learning a well-generalizing classifier from the limited amount of available training data becomes increasingly difficult.

The results of the experiment are shown in Illustration 26. All data sets obtained a local maximum in the DCG as well as precision at $k=1$ at a relatively low value of D, either $D=32$ for the Princeton Shape Benchmark and LabelMe or $D=64$ for the Google 3D Warehouse data set. When increasing the dimensionality of the latent space further, the same DCG as at $D=32$ is reached again for $D=512$ in the case of the Princeton Shape Benchmark. This is not the case for the other two data sets, even though for the Google 3D Warehouse data set, the DCG at $D=512$ gets very close to the one obtained at $D=64$. Two things can be concluded from this experiment: First, a relatively low dimensional space is sufficient to obtain good overall label prediction results. Second, enforcing the sharing of axes in the latent space through a low dimensionality can even be beneficial to prediction performance. In our experiment, this was the case for the LabelMe data set and the Google 3D Warehouse data set, both of which have a high number of labels of which many are semantically similar.

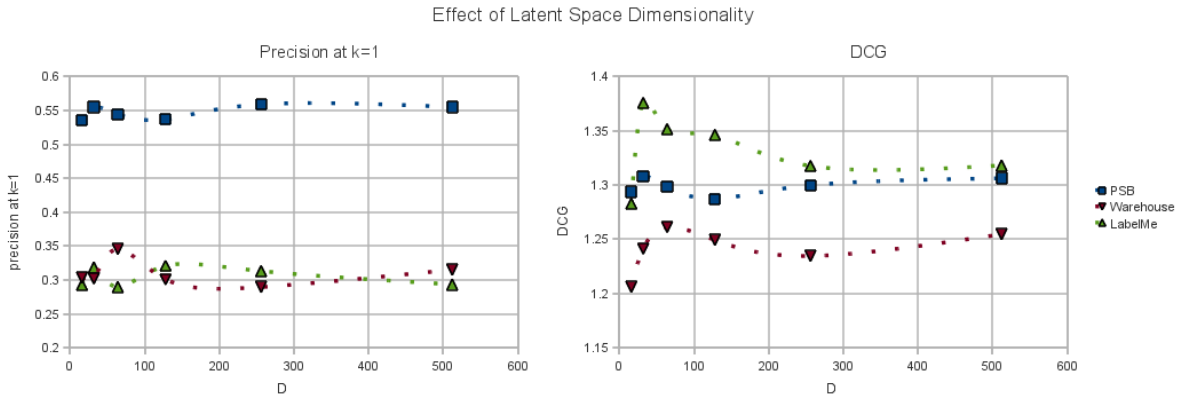


Illustration 26: Effect of the latent space dimensionality D on the label prediction performance of the WSABIE method.

In chapter 3.2.2 we introduced what we called the "soft ranking" modification to the WSABIE learning scheme. We utilized correlations between labels in the ground-truth label sets to fine-tune the loss function. Table 5 shows how the two proposed soft ranking schemes compare to the plain "hard ranking" with constant weights. The overall results were inconclusive. Both soft ranking schemes decrease the label prediction performance on both the LabelMe data set and on the Princeton Shape Benchmark, with the w_{max} scheme being the more detrimental one. Results on Google 3D Warehouse showed an opposite trend, with w_{max} slightly above plain WSABIE with respect to both DCG and precision at $k=1$. w_{mean} improved the precision at $k=1$ for this data set, while decreasing the DCG. In our experi-

ence, Google 3D Warehouse has the most inconsistent label sets of the three data sets, which might suggest that soft ranking is effective in such scenarios, while having a negative impact in cases where the labels are relatively clean and consistent.

	Princeton Shape B.			**Google 3D Warehouse**			**LabelMe**		
	p@1	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**
plain	0.56	0.28	1.31	0.30	0.19	1.25	0.32	0.21	1.35
w_{max}	0.52	0.28	1.29	0.34	0.19	1.26	0.26	0.18	1.22
w_{mean}	0.54	0.28	1.30	0.32	0.18	1.21	0.29	0.20	1.30

Table 5: Evaluation of ranking schemes that utilize ground-truth label correlation, as proposed in chapter 3.2.2.

We could verify that our label aliasing extension (compare chapter 3.2.3) works on an extremely simple artificially created data set not unlike the one shown in Illustration 19. The artificial data set had four labels, with the descriptors of the corresponding training samples designed such that the labels could not be linearly separated from each other. We also measured the effect of the method on the label prediction performance on the Princeton Shape Benchmark, the Google 3D Warehouse and the LabelMe data sets. Unfortunately we could not observe any significant improvements in the prediction performance. Visualizations of the latent space while the optimization was running indicate that freshly introduced aliases could not converge to different clusters of associated training samples. They often collapsed all into the same location, despite the precautions we took to avoid this behavior. In other cases, the aliases could not converge to useful unique locations within a reasonable number of iterations. The complex and non-linear interactions between the embedding of training observations and the different labels makes it difficult to fully understand the effects at work analytically. While it is possible that the failure of the aliasing method in our experiments was merely due to incorrectly adjusted parameters, the increased optimization time required with this method did not allow us to exhaustively probe a wide range of configurations within the scope of this thesis.

Illustration 27: Label predictions on randomly selected samples from the Google 3D Warehouse data set: Blue are the ground-truth labels, red the top three predictions of the WSABIE algorithm in order of their ranking. The a bag of HOC descriptors was used in this experiment. While some predictions are close to perfect (for example 3rd from top, 1st from left), others demonstrate failure cases of the method. For example the chair 4th from top, 2nd from left was classified as "bottle, computer, laptop". We believe that the failure in this case can be attributed to two main factors: 1. our bag of features descriptors do not encode the global shape of the object well. Individual parts of the chair do in fact resemble the screen and keyboard of a laptop computer. 2. Labels which often occur in combination with other objects in the same mesh are difficult to learn and require a very high number of training samples. We found that both "chair" and "bottle" often occur in scenes composed of different kinds of furniture and different variations of bottles (including rectangular ones) respectively in this data set.

3.3.4 Descriptors

We have not yet performed any evaluation of the performance of the descriptors introduced in chapter 2. We omitted an evaluation at that point, because the performance of a given descriptor can only be evaluated in a meaningful way if the objective of its application is known. We have now specified a concrete application for the descriptors (learning and predicting attributes), as well as a metric for evaluating their performance in this application.

While there are many descriptors available for representing 2D image data, we have only introduced and implemented the HOG descriptor. The HOG descriptor in combination with a bag of features dictionary has been used successfully for the given task by others (e.g. in [Weston et al. 2011]). An application of the WSABIE method to 3D shapes has – to the best of our knowledge – not been done before though. Here, we compare only 3D shape descriptors with respect to their performance in the WSABIE learning scheme and under a linear SVM as a reference (again in a one-vs-rest scheme).

We evaluate the HON (→ 2.2.1), HOC (→ 2.2.1) and Harmonic Shape Descriptor (HSD → 2.1.2) with respect to their retrieval performance on the Google 3D Warehouse and the Princeton Shape Benchmark data sets. The descriptors were compute for local patches of the shapes and a 512 dimensional bag of features descriptor was built on top. For each descriptor, we have tried different regularization parameters C of the WSABIE method and show the results for the best one (with respect to the obtained DCG).

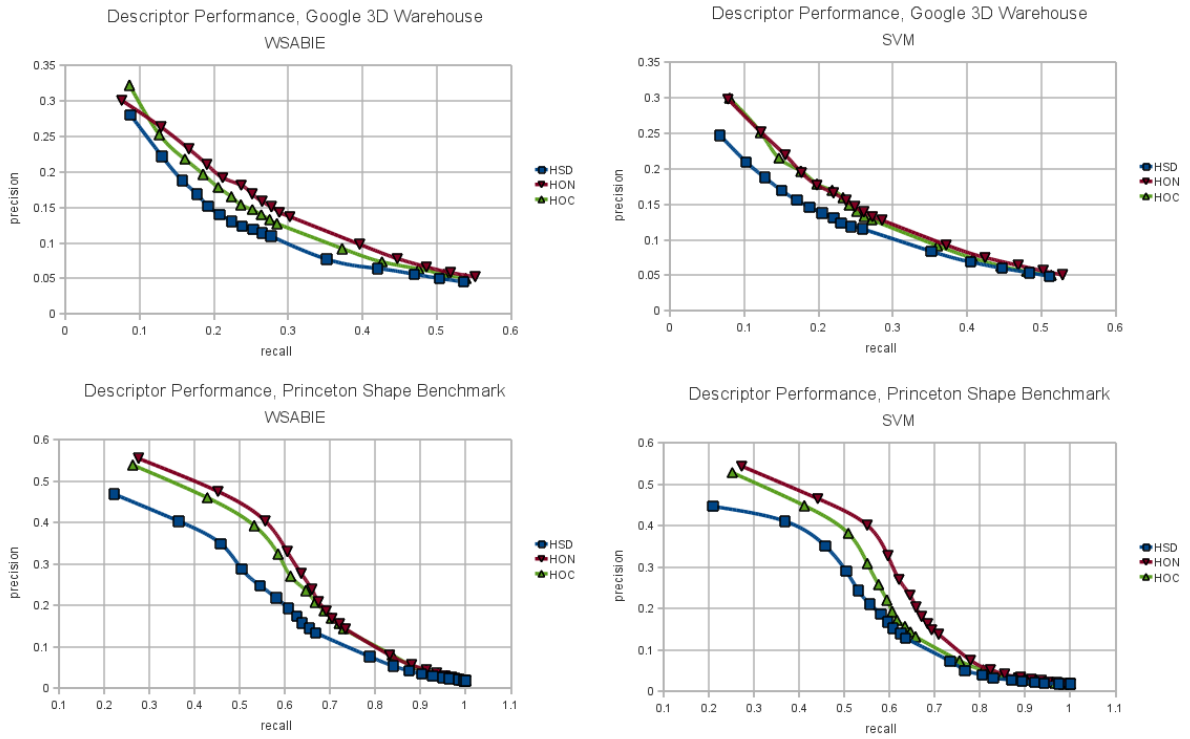


Illustration 28: Descriptor comparison: precision / recall curve (excerpt) on the Google 3D Warehouse and Princeton Shape Benchmark data sets. Also compare Table 6.

In this experiment, we configured WSABIE to use a 32 dimensional latent space in the case of the Princeton Shape Benchmark, and a 128 dimensional latent space for Google 3D Warehouse.

Illustration 28 compares the precision/recall curves achieved with the different descriptors. Both HOC and HON descriptors consistently outperformed the harmonic shape descriptor in our experiment. The normal-based HON descriptor was generally superior to the curvature-based HOC on the Princeton Shape Benchmark data set, while both achieved similar overall performance on Google 3D Warehouse (with a slightly higher precision at 1 for HOC and better precision values at $k>1$ for HON when using the WSABIE method). We believe that the larger difference between HON and HOC descriptors on the Princeton Shape Benchmark compared to their difference on the Google 3D Warehouse data set can be attributed to the different shape composition of those data sets. The HOC descriptor generally expresses sharp edges and corners very efficiently, while we expect the HON descriptor to achieve better results where those are less dominant. The Princeton Shape Benchmark contains many models with smooth and round surfaces. According to the DCG values (compare Table 6), the HON descriptor turns out to have the best overall performance in all tested scenarios.

	Princeton Shape B.			**Google 3D Warehouse**		
WSABIE	**p@1**	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**
HSD	0.47	0.25	1.18	0.28	0.15	1.03
HON	0.56	0.28	1.31	0.30	0.19	1.25
HOC	0.54	0.27	1.29	0.32	0.18	1.18
SVM	**p@1**	**p@5**	**DCG**	**p@1**	**p@5**	**DCG**
HSD	0.45	0.24	1.16	0.25	0.16	1.07
HON	0.54	0.27	1.26	0.30	0.18	1.19
HOC	0.53	0.26	1.22	0.30	0.18	1.18

Table 6: Descriptor comparison: Precision and DCG values on the Google 3D Warehouse and Princeton Shape Benchmark data sets. Also compare Illustration 28.

4 Multi-Modal Semantics

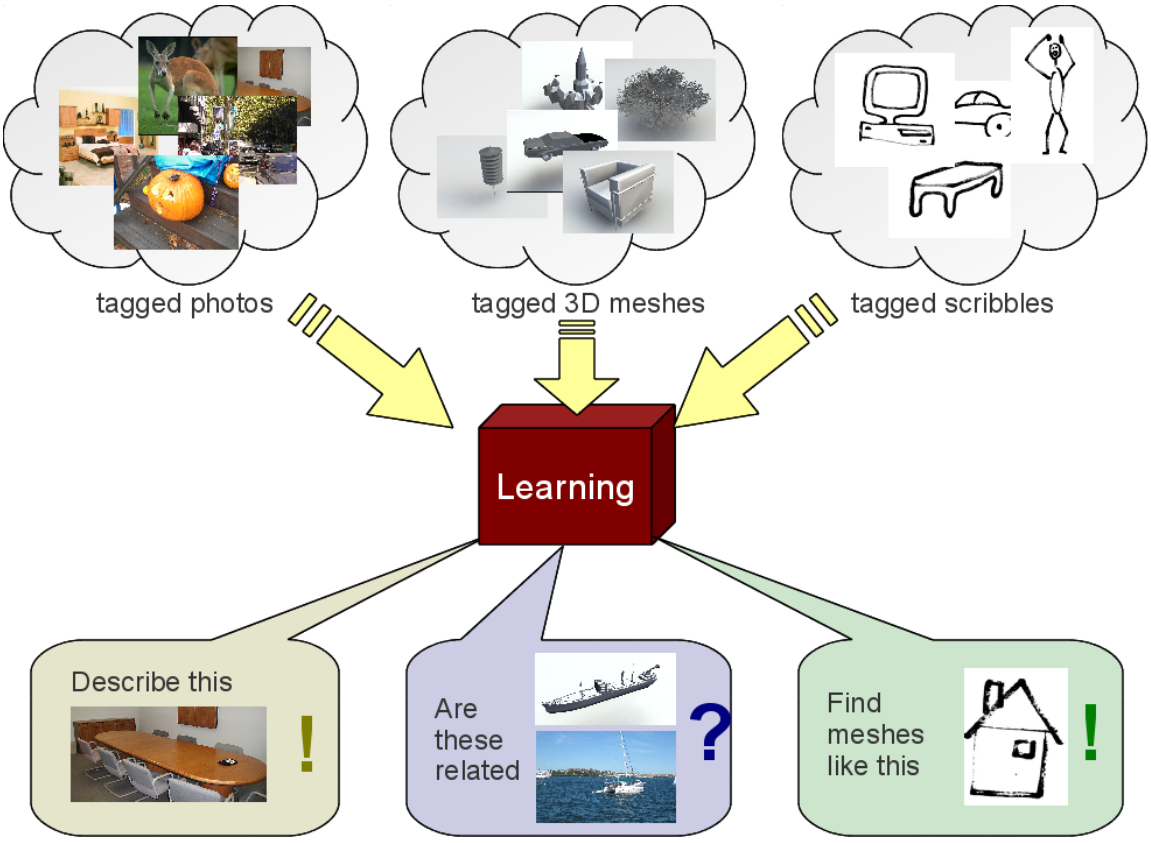


Illustration 29: We are given a set of tagged training objects from different modalities, such as photos, 3D meshes and scribbles. We use machine learning in order to learn a measure of semantic similarity between objects from different modalities. Our goal is to answer queries regarding the semantic relation between objects from different modalities.

Assume that we are given input data from different modalities $m_1, \ldots, m_n$. For example these might be 3D meshes, 2D photographs, sound samples, music, videos or text. We denote the set of input observations from modality m_i by S_{m_i}. Our goal is to establish semantic correspondences between samples from different modalities. We define semantic correspondences through a semantic similarity measure s, mapping from pairs of observations to a positive real number. The higher this number, the more semantically similar the observations in the given pair are. In the uni-modal case, we have already seen how we can establish correspon-

dence between 3D meshes or 2D photos on the one, and text labels on the other side (see chapter 3). This uni-modal case can be thought of as a special case of establishing a semantic similarity measure s, where the set of labels would be interpreted as a second modality. In contrast, in the full multi-modal case, s can not only be applied to a label and an observation from a single given modality, but also to observations from two different modalities. We want to learn s given the training observations $S_{m_1}, \ldots, S_{m_n}$.

In order to approximate the a multi-modal semantic similarity measure s, we need additional information which links the semantics of the different modalities together. In addition – if we also want s to generalize to data samples which are not part of the initial training set – we also must assume a prior on how semantic similarity behaves *within* a given modality.

Previously we have seen how images and 3D meshes can be represented as descriptors (compare chapter 2). However different descriptors have to be used for different modalities, and they will behave very differently. In many cases, it can make sense to use descriptors of different dimensionalities for the different modalities. We say that every observation of modality m_i is represented by a descriptor $x \in \mathbb{R}^{d_i}$, where d_i is the dimensionality of the descriptors used for modality m_i.

As in the uni-modal case, we assume that within a single modality, observations that have similar descriptors also are semantically similar. As for establishing inter-modal correspondences, we have essentially two choices: First, for specific descriptors, we might know that descriptors between different modalities for a specific semantic entity are statistically dependent in a well-defined, sufficiently simple and sufficiently generic way. If we can define this dependence in a stochastic manner, we can use it as a stochastic prior on our multi-modal semantic similarity measure. A simple example would be descriptors for visual modalities (e.g. video, photos and textured 3D meshes), that all contain information about the color of the described object. If we look at a textured 3D mesh on the one and a photo on the other hand, we would find that meshes that have the same kinds of colors as a set of photos are statistically more likely to also be semantically close, simply because they are more likely to depict the same underlying object. Another example would be video with an audio channel, where we could attempt to learn multi-modal semantics by correlating audio and video data. The descriptors can be made to contain a “time” coordinate, and things that are audible in the audio channel at a specific time are more likely to be semantically related with image data in the video channel from the same or a similar time.

Unfortunately, in our setting it is in not always trivial to find such priors, and those that can be found (e.g. the one with color for visual modalities) are often not discriminative enough to allow establishing cross-modal semantic correspondences from these priors alone. We therefore pursue a different approach, which can be thought of as converting the initial unsupervised into a semi-supervised learning problem: We assume that between certain pairs of modalities, semantic correspondences are known. The goal is then to extrapolate, or interpolate respectively, from this knowledge such that a) we can also establish semantic correspondences between modalities where correspondences are not known beforehand, and b) we can apply the semantic similarity measure s to observations for which no ground-truth correspondences are given at all. In practice, we turn to text labels which are widely available for data from different modalities. Many openly accessible databases of visual and/or auditory

data use user-provided labels (“tags”) to enable searching and navigating those databases. While the semantic meaning of an object is generally a subjective concept, we assume that those labels capture relevant aspects of the semantic meaning.

In this chapter, we look at how we can learn a similarity measure which can

1. tell us the similarity between observations within a certain modality, even if the exact observations are not known at the time of training
2. tell us the similarity between observations from different modalities

We first have a brief look at related methods in chapter 4.1, and then propose our own extension of the WSABIE method (compare 3.1.3) to support multiple modalities in chapter 4.2. Finally, we introduce and evaluate an application that makes use of inter-modal similarities in chapter 4.2.2.

4.1 Related Work

Querying 2D image as well as 3D shape databases by hand-drawn scribbles has been addressed in a number of previous papers. [Chan et al. 1997] built on a common descriptor-based representation between images and scribbles in order to allow querying an image database through hand-drawn sketches. Recently, [Eitz et al. 2012] have implemented sketch-based retrieval for 3D shapes. Their approach is based on rendering line-drawings of the shapes in the 3D shape database from a variety of view-points, thereby allowing descriptor-based methods from the area of image retrieval to be applied to 3D data. Since 2012, the SHREC competitions also have a discipline on sketch-based 3D shape retrieval (compare [Li et al. SKETCH 2012]).

More generally, [Yang et al. 2001] proposed a system to query databases of text, images, video and audio data by either keywords or query objects. They use two internal retrieval techniques 1. retrieval by keyword and 2. retrieval through descriptor matching. Descriptor matching can only be performed within one modality, but keywords can be queried across multiple modalities. Both techniques are combined in their system in the sense that initial results of a descriptor-based (that is non-keyword) query are used to initiate related keyword-based sub-queries and vice-versa. The results of the initial query can then be combined with the results of those sub-queries. Thanks to this technique, cross-modal descriptor queries are possible because of the automatically generated keyword-based sub-queries. While no initial learning is performed in their approach, user feedback can be incorporated into future query results.

4.2 Our Approach

To the best of our knowledge, previous methods for multi-modal retrieval were either based on common descriptors (which can not always be established), or on first performing a transfer from the query modality into a textual label (keyword) domain and then back into the target modality. Finer-grained semantic properties can be lost in this transformation, and aspects such as the fact that certain labels frequently occur together and can therefore be regarded as related concepts are usually not taken into account. We propose a method which builds on an embedding of different modalities into a common semantic space, on which we define a similarity measure. Shared labels are utilized to initialize this space, and as in [Weston et al. 2011] we explicitly represent the semantic relationship between different labels by encouraging them to share dimensions in the space.

4.2.1 Multi-Modal WSABIE

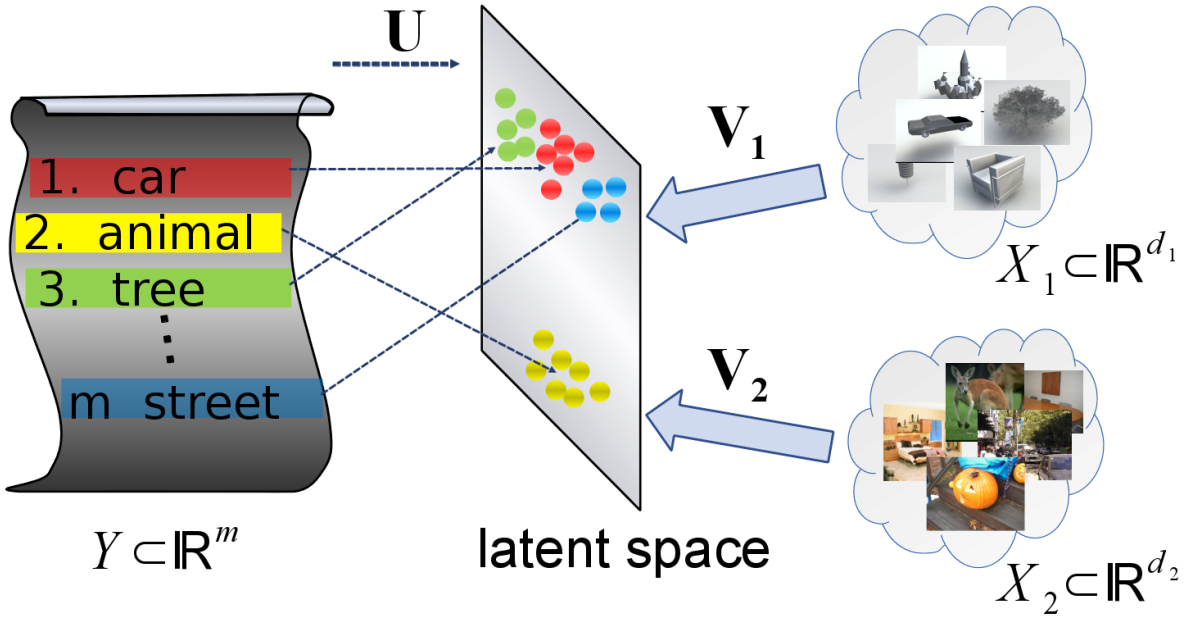


Illustration 30: In contrast to the uni-modal case (compare Illustration 17), we embed multiple modalities, here 3D shapes X_1 and photographs X_2, into the same latent space. While labels are embedded through a single linear mapping U, each modality is embedded by a different mapping $V_1, \ldots, V_n$. Bases on an illustration by M. Wand and R. Herzog.

The method of [Weston et al. 2011] provides a good starting point because it partially decouples semantics – represented by positions in an explicit latent semantic space – from the representation of the observations. As explained in chapter 3.1.3, the mapping from a descriptor $x \in \mathbb{R}^d$ into the D-dimensional semantic space is performed through a linear transformation $V \in \mathbb{R}^{D \times d}$. We extend the method to support observations from a number of different modalities $m_1, \ldots, m_n$, where each modality can use different descriptors. The descriptor for a sample from S_{m_i} is given by $x \in \mathbb{R}^{d_i}$. Note that descriptors from different modalities can have different dimensionalities. To establish a multi-modal semantic model,

we want to map all the different descriptors into one common semantic space $\mathbb{R}^D$. To do this, we train a different linear mapping $V_i \in \mathbb{R}^{D \times d_i}$ for each modality. The label set Y together with its mapping $U \in \mathbb{R}^{|Y| \times D}$ on the other hand is shared across all modalities, to provide a "semantic glue" between the otherwise unrelated descriptors. Once all mappings have been trained, the multi-modal semantic similarity measure $s: \bigcup_{i=1}^{n} R^{d_i} \to \mathbb{R}$ is given by

$$s(x, x') := V_i x \cdot V_j x' \quad \text{if } x \text{ from modality } m_i \text{ and } x' \text{ from modality } m_j$$

We extend the stochastic gradient optimization scheme (compare chapter 3.1.3) as follows: In each iteration, we randomly pick one modality m_i. Then, gradients for V_i and the label mapping U are calculated just as in the uni-modal case and the respective matrices are updated. The probability at which each modality is chosen can be used to prioritize the precision of certain modalities over others in the resulting semantic space. In our experiments, we use uniform probabilities.

4.2.2 Application: A Multi-Modal Semantic Explorer

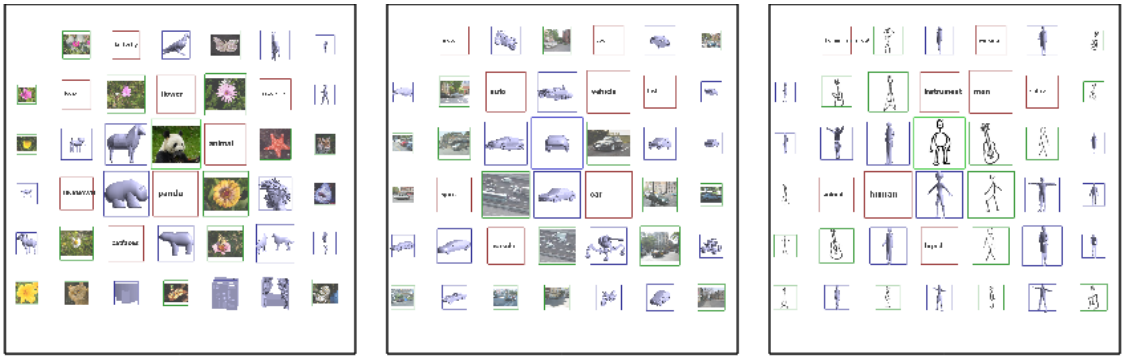


Illustration 31: Screenshots from the multi-modal semantic explorer. The query object is in the center. The nearest neighbors in the semantic space from different modalities are spread around it. The size of the neighbors decreases with increasing distance from the query.

We implement an interactive tool to explore a multi-modal database of objects. The user starts the exploration by providing an initial query. This query can be given either as a label entered through the keyboard, or in the form of an image file. First, the query is converted into the corresponding descriptor. The semantic explorer application then maps the query into a previously learned semantic space, and looks for the k nearest neighbors in that space from each modality with respect to the semantic similarity measure s defined above. The neighbors are visualized around the query and presented to the user. The user can then refine and/or change their query by clicking on one of the results, thereby making the result object the new query object. This step can be repeated.

The option to query the database by an image query allows the user to take a photo with a smart phone or other digital camera and query the database for semantically related objects. For example, a user might look for a 3D mesh related to an object from the real world. Or they might be interested in a semantically similar photograph of higher quality than the query.

Because mapping descriptors into the latent semantic space is very efficient with the linear mappings obtained from the multi-modal WSABIE approach, we do not have to perform any pre-computation other than training those mappings. On our datasets consisting of up to roughly 20,000 objects, simple linear-time sequential filtering without any index structures was sufficient for real-time exploration of the data set without noticeable delays.

4.3 Evaluation

We evaluate the multi-modal extension of WSABIE on two multi-modal data sets:

1. 3D shapes from the Princeton Shape Benchmark together with our own scribbles data set (see below)
2. 3D shapes from the Google 3D Warehouse data set together with photographs from the LabelMe data set

For the Princeton Shape Benchmark and Google 3D Warehouse data set, we use the same descriptors as explained previously in chapter 3.3.2.

Our Scribbles data set contains 374 hand-drawn scribbles, stored as black and white images. Each scribble appears twice in the data set, once in its original variant, and once mirrored along the x axis. The data set was built in-group specifically for the purpose of evaluating multi-modal learning. We assigned an average of 1.8 labels out of a label set of 24 labels to each scribble. The labels themselves are inspired by the labels used in the Princeton Shape Benchmark, and are of medium granularity (such as “human”, “car”, “guitar”). While we use the same HOG descriptor as for LabelMe to avoid having to implement an additional descriptor, we optimize parameters of the descriptor based on the information given in [Eitz et al. 2012]. Because lines in scribbles are often not completely straight nor exact, we use a reduced number of 4x4 spatial and 8 rotational bins for the HOG descriptor. Based on the HOG descriptor, we build a single 256 dimensional Bag of Words descriptor for each scribble.

As a first experiment, we test how our multi-modal semantic explorer works for the use case of exploring a given multi-modal data set. For this, we train a latent semantic space using the multi-modal WSABIE extension on the data sets introduced above. We query by meshes, photographs, scribbles and labels. The nearest neighbors in the semantic space are visualized and can be evaluated subjectively. The results on the Princeton Shape Benchmark / scribbles data set are shown in Illustrations 32, 33 and 34. We found the overall results to be convincing. Failure cases are marked in the result illustrations.

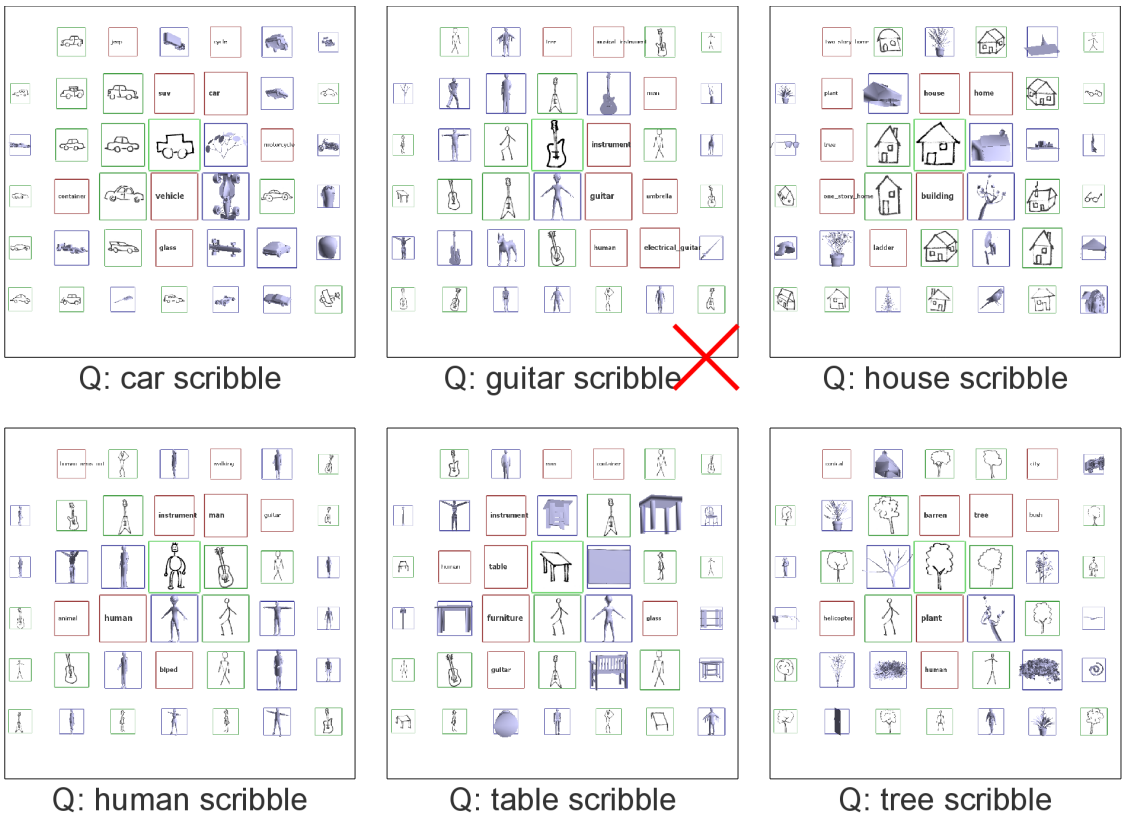


Illustration 32: Multi-modal query results on the Princeton Shape Benchmark and scribble data sets. A representative selection is shown. Query by scribble. The query object is in the center. The nearest neighbors in the semantic space from different modalities are spread around it. The size of the neighbors decreases with increasing distance from the query. In the case of the guitar scribble, our method retrieves both meshes of guitars (correctly) as well as humans (incorrectly). We believe that this is due to the geometric similarities between those classes and the limited power of our descriptor to successfully discriminate between both.

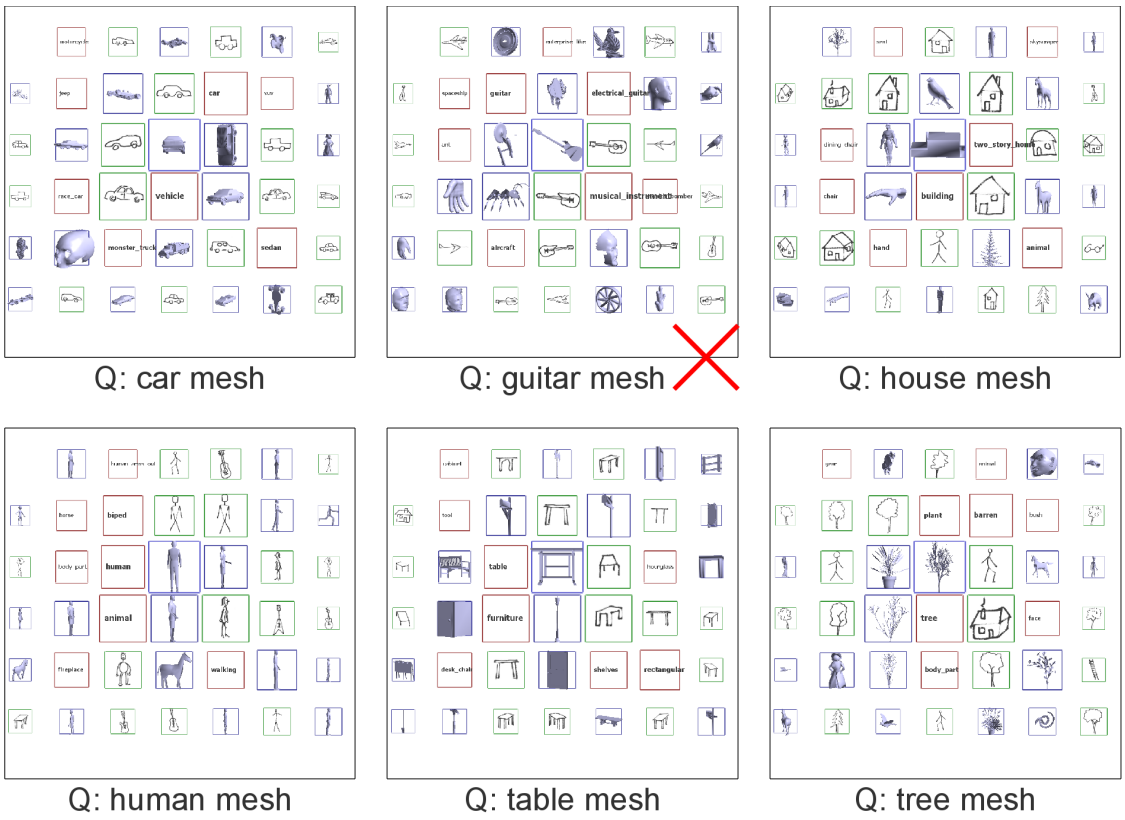


Illustration 33: Multi-modal query results on the Princeton Shape Benchmark and scribble data sets. A representative selection is shown. Query by mesh. As in the case of a scribble query, the results for the guitar mesh query contain many false positives.

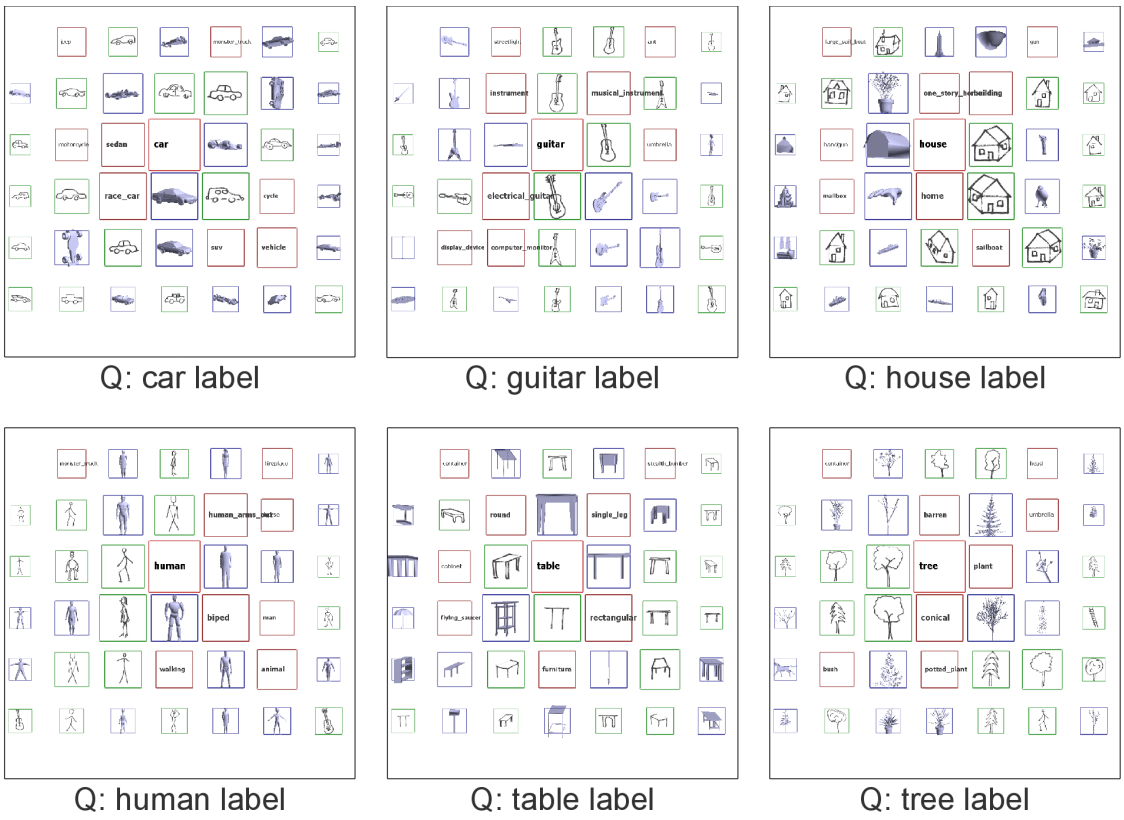


Illustration 34: Multi-modal query results on the Princeton Shape Benchmark and scribble data sets. A representative selection is shown. Query by label.

Illustration 35: Multi-modal query results on the Google 3D Warehouse and LabelMe data sets. A representative selection is shown. Query by photo. The query object is in the center. The nearest neighbors in the semantic space from different modalities are spread around it. The size of the neighbors decreases with increasing distance from the query. Note the failure case of the face query. The beach photo query illustrates how semantically related concepts are mapped close to each other in the latent space, showing meshes of boats. Similar effects can be observed in for the road and motorcycle photo queries.

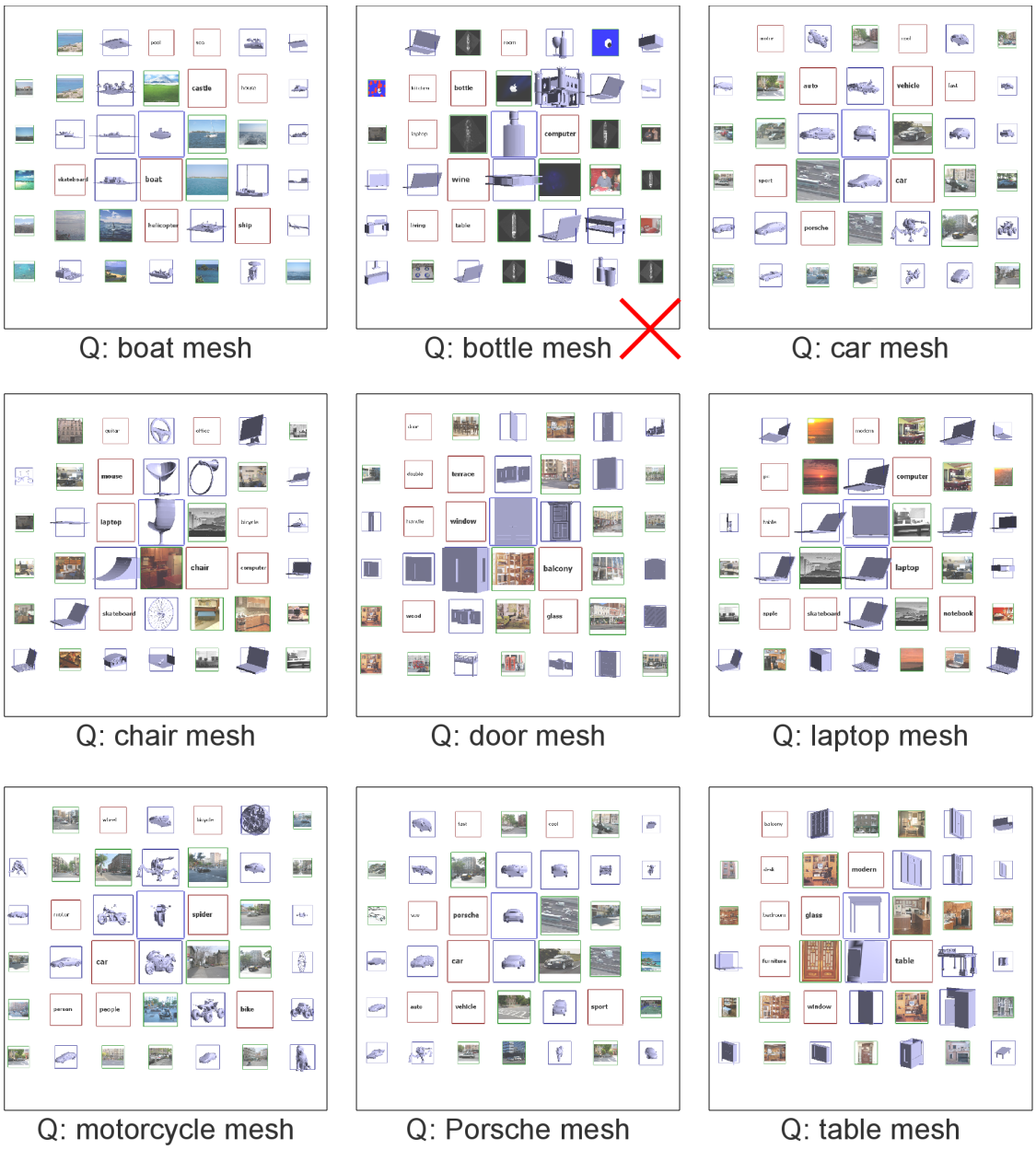


Illustration 36: Multi-modal query results on the Google 3D Warehouse and LabelMe data sets. A representative selection is shown. Query by mesh. The failure case of the bottle query shows the limits of the bag of features descriptor. The descriptor does not capture the global shape of the bottle well, but is based on local pieces of it that have little discriminative power.

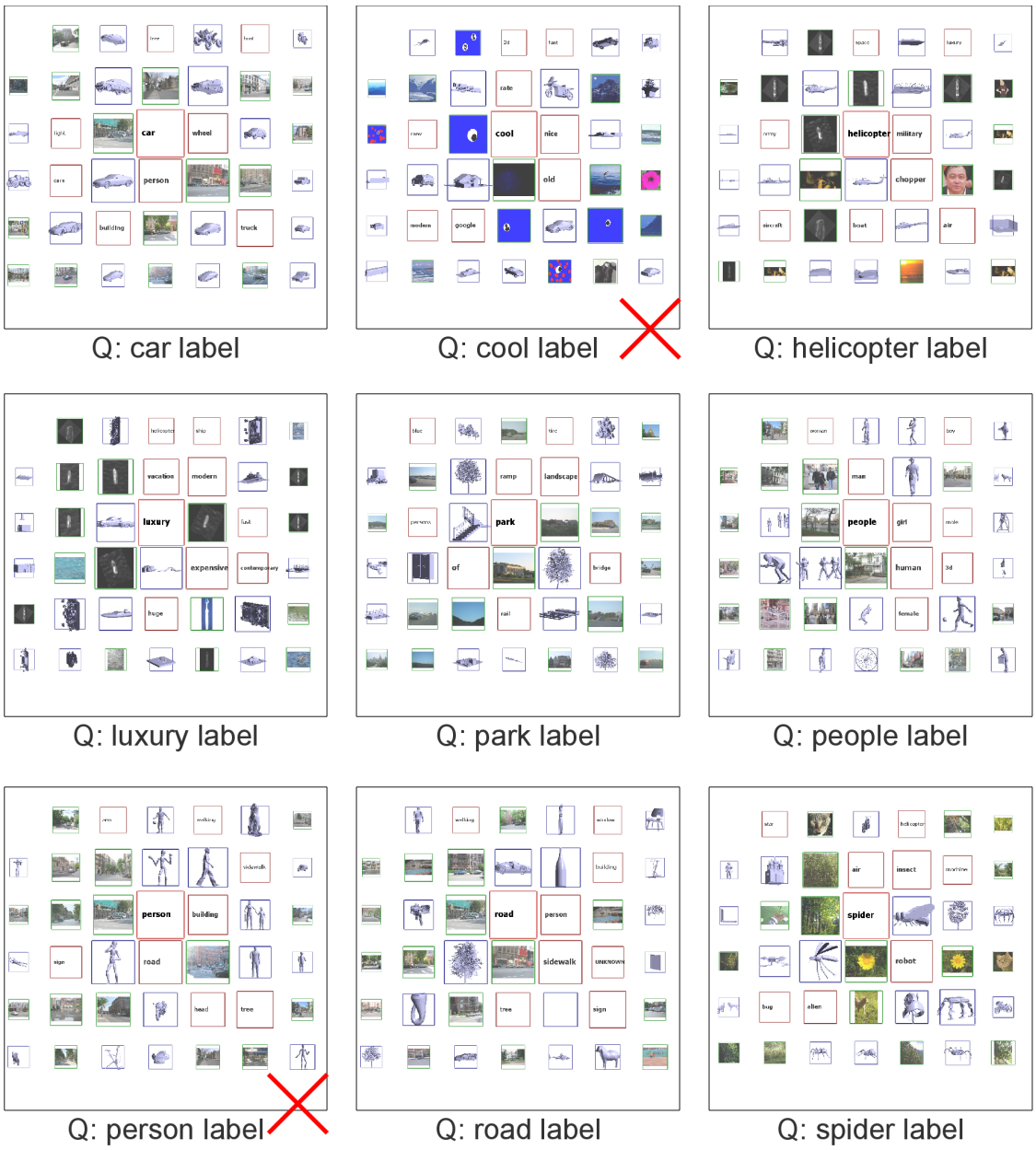


Illustration 37: Multi-modal query results on the Google 3D Warehouse and LabelMe data sets. A representative selection is shown. Query by label. Our method has difficulties in covering abstract attributes such as "cool", as they cannot easily be associated with characteristic visual features. While the query by label "person" resulted in many shapes of human beings, the retrieved photographs often do not contain any people.

We also tested multi-modal querying with photos taken with a digital camera that were not part of the training data set. Some results are shown in Illustration 38. We found that the retrieval worked well as long as there was enough training data similar to the query. The method frequently failed for query photos which had no or few globally close matches in the training data, despite containing known objects in some of their parts.

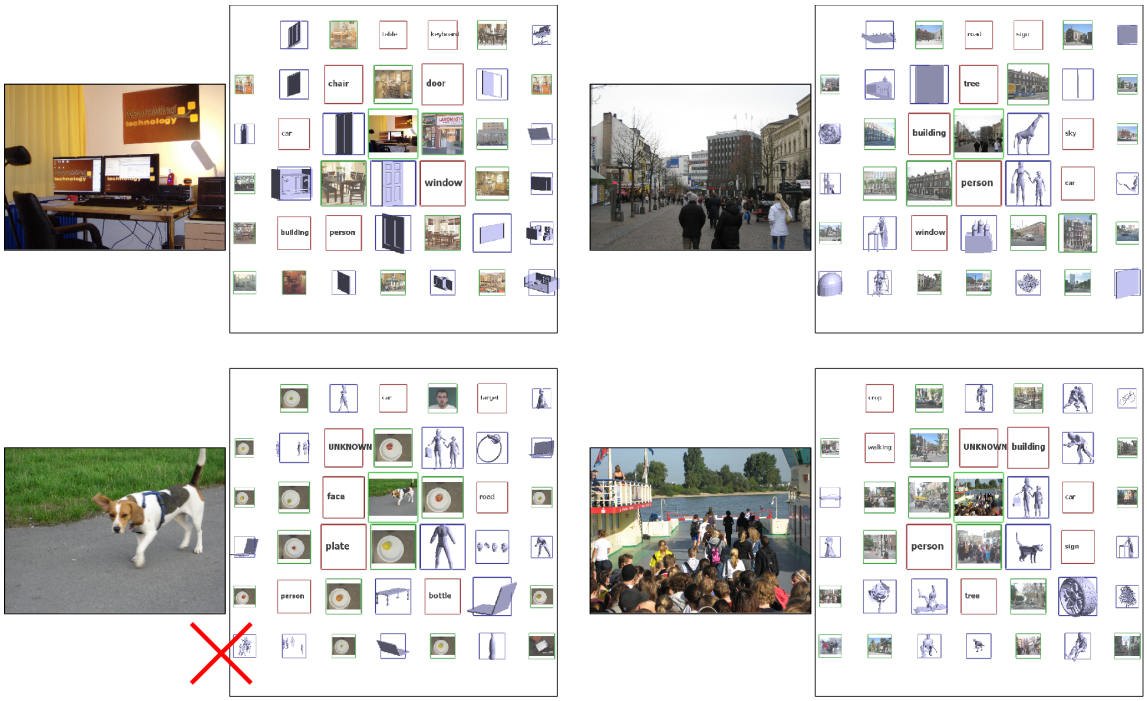


Illustration 38: Multi-modal query results on the Google 3D Warehouse and LabelMe data sets. Examples of querying by photos not in the data set. Note the following (partial) failure cases: Our method failed to recognize the fact that the picture on the bottom right was taken on a boat rather than a building. In the marked failure case (left bottom), the query results had little in common with the query. We believe that this is due to the fact that while the training data did contain some photos of animals, those were shot from different perspectives and in different surroundings than our query image.

Evaluating the quality of multi-modal query results quantitatively is difficult. Because semantic similarity is a subjective concept, the correctness of returned results cannot be easily assessed. There is the option of comparing a query object to the result objects by comparing the respective associated ground-truth label sets. This metric only works if the data sets from different modalities are labeled in a very similar fashion though. While our approach to multi-modal learning is itself based on the assumption that labels are shared across modalities, it is designed to still work in cases where the set of overlapping labels is relatively small. For the special application of scribble-based image retrieval, [Eitz et al. 2011] have introduced a benchmark data set. Unfortunately their data set is aimed at descriptor-based methods. As a consequence, many labels do not come with the amount of training data which is required by a learning-based approach such as ours. The results would be difficult to compare to descriptor based methods which do not require training.

5 Conclusion and Future Work

We have seen different methods for approaching the problem of semantically structuring visual data. We derived a novel method for structuring multi-modal visual data based on the WSABIE method of [Weston et al. 2011] and demonstrated its performance in both uni-modal as well as the multi-modal scenarios.

Our method is based on the assumption that semantic labels are assigned globally to a given shape or image. In many cases, interesting information is contained in the local composition of a shape or image though. One approach to introduce such aspects into our method is an extension of our descriptors. For example [Xiong et al. 2011] introduce a mechanism in which descriptors are enriched with application specific information about their neighborhood. Another interesting approach is to explicitly extract local parts of a shape which are characteristic for a given semantic label, similar to [Shilane et al. 2007] and [Doersch et al. 2012]. Once we know which parts are characteristic for a label, we can use these parts to efficiently localize instances of a semantic class in a huge scene, or even in a large-scale scanned 3D point cloud.

More work can be spent on our label aliasing idea. We see two aspects for future research: First, it is likely that the model can be reformulated to be applicable to more general machine learning problems. Its fundamental properties in a general setting could then be explored. Second, we could not observe significant improvements in the label prediction performance when utilizing label aliasing. More research is necessary to find out the exact reasons for its failure in our specific setting.

Our multi-modal learning approach could be applied to a larger number of modalities. More applications can be developed on top of it.

An interesting challenge could be to refine large-scale 3D scans (for example of a whole town) by replacing individual objects in the scene (such as cars, houses, people etc.) with more detailed, semantically equivalent 3D meshes. This problem poses three sub-problems: First, we must be able to locate instances of a given object class in the 3D scan. A solution to the previously mentioned problem of learning characteristic parts for a label could prove helpful here. Secondly, once we have located an object, we must find an appropriate replacement from a library of 3D meshes. Our multi-modal learning approach can be utilized for this problem, as it allows to establish a joint semantic similarity measure across both 3D point cloud and 3D mesh data. Finally, the 3D mesh from the database must be rescaled and

aligned properly to the corresponding object instance in the scanned 3D point cloud. A constellation or part-based model over objects of the given class could provide a good basis for this, and could potentially be integrated into our learning scheme.

6 References

[Arya et al. 1994] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, Angela Y. Wu:
An Optimal Algorithm for Approximate Nearest Neighbor Searching
In: ACM-SIAM Symposium on Discrete Algorithms – SODA, pp. 573-582, 1994

[Barrow et al. 1969] H.G. Barrow, S. H. Salter:
Design of low-cost equipment for cognitive robot research
In: Machine Intelligence 5, Edinburgh University Press, pp. 555-566, 1969

[Beyer et al. 1999] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, Uri Shaft:
When Is "Nearest Neighbor" Meaningful?
In: International Conference on Database Theory – ICDT, pp. 217-235, 1999

[Bokeloh et al. 2008] Martin Bokeloh, Alexander Berner, Michael Wand, Andreas Schilling, Hans-Peter Seidel:
Slippage Features (Technical Report)
In: WSI-2008-03, University of Tübingen, 2008

[Bottou 2003] Léon Bottou:
Stochastic Learning
In: Advanced Courses – AC, pp. 146-168, 2003

[Bronstein et al. 2011] Alexander M. Bronstein, Michael M. Bronstein, Leonidas J. Guibas, Maks Ovsjanikov:
Shape Google: Geometric Words and Expressions for Invariant Shape Retrieval
In: ACM Transactions on Graphics – TOG, vol. 30, no. 1, pp. 1-20, 2011

[Chans et al. 1997] Yin Chans, Zhibin Lei, Daniel P. Lopresti, Sun-Yuan Kung:
Feature-Based Approach for Image Retrieval by Sketch
In: Storage and Retrieval for Image and Video Databases, vol. 3229, pp. 220-231, 1997

[Chua et al. 1997] Chin-Seng Chua, Ray Jarvis:
Point Signatures: A New Representation for 3D Object Recognition
In: International Journal of Computer Vision – IJCV, vol. 25, no. 1, pp. 63-85, 1997

[Coates et al. 2011] Adam Coates, Andrew Y. Ng:
The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization
In: Proceedings of the 28th International Conference of Machine Learning, 2011

[Cortes et al. 1995] Corinna Cortes, Vladimir Vapnik:
Support-Vector Networks
In: Machine Learning – ML, vol. 20, no. 3, pp. 273-297, 1995

[Crammer et al. 2001] Koby Crammer, Yoram Singer:
On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines
In: Journal of Machine Learning Research – JMLR, vol. 2, pp. 265-292, 2001

[Dalal et al. 2005] Navneet Dalal, Bill Triggs:
Histograms of Oriented Gradients for Human Detection
In: Computer Vision and Pattern Recognition - CVPR, vol. 1, pp. 886-893, 2005

[Doersch et al. 2012] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, Alexei A. Efros:
What Makes Paris Look Like Paris?
In: ACM Transactions on Graphics - TOG, vol. 31, issue 3, 2012

[Eitz et al. 2011] Mathias Eitz, Kristian Hildebrand, Tamy Boubekeur, Marc Alexa:
Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors
In: IEEE Transactions on Visualization and Computer Graphics – TVGG, vol. 17, no. 11, pp. 1624-1636, 2011

[Eitz et al. 2012] Mathias Eitz, Ronald Richter, Tamy Boubekeur, Kristian Hildebrand, Marc Alexa:
Sketch-Based Shape Retrieval
In: ACM Transactions on Graphics - TOG, vol. 31, issue 4, 2012

[Fan et al. 2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, Chih-Jen Lin:
LIBLINEAR: A Library for Large Linear Classification
In: Journal of Machine Learning Research – JMLR, vol. 9, pp. 1871-1874, 2008

[Fergus et al. 2005] Robert Fergus, Pietro Perona, Andrew Zisserman:
A Sparse Object Category Model for Efficient Learning and Exhaustive Recognition
In: Computer Vision and Pattern Recognition – CVPR, vol. 1, pp. 380-387, 2005

[Gal et al. 2006] Ran Gal, Daniel Cohen-Or:
Salient Geometric Features for Partial Shape Matching and Similarity
In: ACM Transactions on Graphics – TOG, vol. 25, no. 1, pp. 130-150, 2006

[Gatzke et al. 2005] Timothy Gatzke, Cindy Grimm, Michael Garland, Steve Zelinka:
Curvature Maps for Local Shape Comparison
In: Shape Modeling International, pp. 246-255, 2005

[Järvelin et al. 2000] Kalervo Järvelin, Jaana Kekäläinen:
IR Evaluation Methods for Retrieving Highly Relevant Documents
In: Research and Development in Information Retrieval – SIGIR, pp. 41-48, 2000

[Johnson et al. 1999] Andrew Edie Johnson, Martial Herbert:
Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes
In: IEEE Transactions on Pattern Analysis and Machine Intelligence – PAMI, vol. 21, no. 5, pp. 433-449, 1999

[Kazhdan et al. 2003] Michael M. Kazhdan, Thomas A. Funkhouser, Szymon Rusinkiewicz:
Rotation Invatiant Spherical Harmonic Representation of 3D Shape Descriptors
In: ACM International Conference Proceeding Series – AICPS, pp. 156-165, 2003

[Lazebnik et al. 2003] Svetlana Lazebnik, Cordelia Schmid, Jean Ponce:
Sparse Texture Representation Using Affine-Invariant Neighborhoods
In: Computer Vision and Pattern Recognition – CVPR, 2003

[Li et al. 2007] Xinju Li, Igor Guskov:
3D Object Recognition from Range Images Using Pyramid Matching
In: International Conference on Computer Vision – ICCV, pp. 1-6, 2007

[Li et al. 2012] B. Li, A. Godil, M. Aono, X. Bai, T. Furuya, L. Li, R. López-Sastre, H. Johan, R. Ohbuchi, C.Redondo-Cabrera, A. Tatsuma, T. Yanagimachi, S. Zhang:
SHREC'12 Track: Generic 3D Shape Retrieval
In: Proceedings of the 5th Eurographics Conference on 3D Object Retrieval – EG 3DOR, pp. 119-126, 2012

[Li et al. SKETCH 2012] B. Li, A. Godil, M. Alexa, T. Boubekeur, B. Bustos, J. Chen, M. Eitz, T. Furuya, K. Hildebrand, S. Huang, H. Johan, A. Kuijper, R. Ohbuchi, R. Richter, J. M. Saavedra, M. Scherer, T. Yanagimachi, G. J. Yoon, S. M. Yoon:
SHREC'12 Track: Sketch-Based 3D Shape Retrieval
In: Proceedings of the 5th Eurographics Conference on 3D Object Retrieval – EG 3DOR, 2012

[Lowe 1999] David G. Lowe:
Object Recognition from Local Scale-Invariant Features
In: International Conference on Computer Vision – ICCV, vol. 2, pp. 1150-1157, 1999

[Niyogi et al. 1999] Partha Niyogi, Chris Burges, Padma Ramesh:
Distinctive Feature Detection Using Support Vector Machines
In: International Conference on Acoustics, Speech, and Signal Processing – ICASSP, vol. 1, pp. 425-428, 1999

[Pearson et al. 1901] Karl Pearson:
On Lines and Planes of Closest Fit to Systems of Points in Space
In: Philosophical Magazine Series 6, vol. 2, no. 11, pp. 559-572, 1901

[Rennie et al. 2005] Jasson D. M. Rennie, Nathan Srebro:
Fast Maximum Margin Matrix Factorization for Collaborative Prediction
In: International Conference on Machine Learning – ICML, pp. 713-719, 2005

[Russell et al. 2008] Bryan C. Russell, Antonio B. Torralba, Kevin P. Murphy, William T. Freeman:
LabelMe: A Database and Web-Based Tool for Image Annotation
In: International Journal of Computer Vision – IJCV, vol. 77, no. 1-3, pp. 157-173, 2008

[Shilane et al. 2004] Philip Shilane, Patrick Min, Michael M. Kazhdan, Thomas A. Funkhouser:
The Princeton Shape Benchmark
In: Shape Modeling International, pp. 167-178, 2004

[Shilane et al. 2007] Philip Shilane, Thomas A. Funkhouser:
Distinctive Regions of 3D Surfaces
In: ACM Transactions on Graphics - TOG, vol. 26, no. 2, pp. 7-es, 2007

[Tevs et al. 2011] Art Tevs, Alexander Berner, Michael Wand, Ivo Ihrke, Hans-Peter Seidel:
Intrinsic Shape Matching by Planned Landmark Sampling
In: Computer Graphics Forum – CGF, vol. 30, no. 2, pp. 543-552, 2011

[Tibshirani 1996] Robert Tibshirani:
Regression Shrinkage and Selection Via the Lasso
In: Journal of the Royal Statistical Society, Series B (Methodological), 58(1):267-288, 1996

[Usunier et al. 2009] Nicolas Usunier, David Buffoni, Patrick Gallinari:
Ranking with Ordered Weighted Pairwise Classification
In: International Conference on Machine Learning – ICML, pp. 133-1064, 2009

[Weston et al. 2000] Jason Weston, Sayan Mukherjee, Olivier Chapelle, Massimiliano Pontil, Tomaso Poggio, Vladimir Vapnik:
Feature Seelection for SVMs
In: Neural Information Processing Systems – NIPS, pp. 668-674, 2000

[Weston et al. 2010] Jason Weston, Samy Bengio, Nicolas Usunier:
Large Scale Image Annotation: Learning to Rank with Joint Wort-Image Embeddings
In: Machine Learning – ML, vol. 81, no. 1, pp. 21-35, 2010

[Weston et al. 2011] Jason Weston, Samy Bengio, Nicolas Usunier:
WSABIE: Scaling Up to Large Vocabulary Image Annotation
In: Proceedings of the International Joint Conference on Artificial Intelligence – IJCAI, 2011

[Winder et al. 2007] Simon A. J. Winder, Matthew Brown:
Learning Local Image Descriptors
In: Computer Vision and Pattern Recognition – CVPR, 2007

[Wikipedia MDS 2013] Wikipedia:
Multidimensionale Skalierung
In: Wikipedia, Die freie Enzyklopädie. Bearbeitungsstand: 25. Juli 2013, 08:36 UTC.
de.wikipedia.org/w/index.php?title=Multidimensionale_Skalierung&oldid=120873146

[Xiong et al. 2011] Xuehan Xiong, Daniel Munoz, J. Andrew Bagnell, Martial Hebert:
3-D Scene Analysis via Sequenced Predictions over Points and Regions
In: International Conference on Robotics and Automation – ICRA, pp. 2609-2616, 2011

[Yang et al. 2001] Jun Yang, Yueting Zhuang, Qing Li:
Multi-Modal Retrieval for Multimedia Digital Libraries: Issues, Architecture and Mechanisms
In: Workshop on Multimedia Information Systems – MIS, pp. 81-88, 2001