VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VISIPEDIA:
EMBEDDING-DRIVEN VISUAL FEATURE EXTRACTION AND LEARNING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN JAKEŠ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VISIPEDIA:
EMBEDDING-DRIVEN VISUAL FEATURE EXTRACTION AND LEARNING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAN JAKEŠ
AUTHOR

VEDOUCÍ PRÁCE — Prof. Dr. Ing. PAVEL ZEMČÍK
SUPERVISOR

BRNO 2014

Abstrakt

Multidimenzionální indexování je účinným nástrojem pro zachycení podobností mezi objekty bez nutnosti jejich explicitní kategorizace. V posledních letech byla tato metoda hojně využívána pro anotaci objektů a tvořila významnou část publikací spojených s projektem Visipedia. Tato práce analyzuje možnosti strojového učení z multidimenzionálně indexovaných obrázků na základě jejich obrazových příznaků a přestavuje metody predikce multidimenzionálních souřadnic pro předem neznámé obrázky. Práce studuje příslušené algoritmy pro extrakci příznaků, analyzuje relevantní metody strojového účení a popisuje celý proces vývoje takového systému. Výsledný systém je pak otestován na dvou různých datasetech a provedené experimenty prezentují první výsledky pro úlohu svého druhu.

Abstract

Multidimensional embedding is a powerful method of representing similarity measures among objects without the need for their explicit categorization. It has been increasingly used in recent years to annotate objects making an important part of the Visipedia project and its related work. This work explores the possibilities of learning from embedding-annotated images using their visual attributes and develops methods of predicting embedding coordinates for previously unseen images. It studies the relevant feature extraction and learning algorithms and describes the whole process of design and development of such a system using common machine learning approaches. The system is tested and evaluated with two different datasets and the performed experiments present the first results for a task of its kind.

Klíčová slova

multidimenzionální indexování, obrazové příznaky, extrakce příznaků, počítačové vidění, strojové učení, regrese, Visipedia

Keywords

multidimensional embedding, visual features, feature extraction, computer vision, machine learning, regression, Visipedia

Citace

Jan Jakeš: Visipedia: Embedding-driven Visual Feature Extraction and Learning, diplomová práce, Brno, FIT VUT v Brně, 2014

Visipedia: Embedding-driven Visual Feature Extraction and Learning

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Prof. Dr. Ing. Pavla Zemčíka. Další imformace mi poskytl Prof. Serge Belongie. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . .
Jan Jakeš
28. května 2014

Poděkování

Rád bych zde poděkoval profesoru Pavlu Zemčíkovi za vedení této práce, cenné rady a koordinaci mého pobytu v USA. Speciální dík patří profesoru Sergi Belongiemu za jeho odborné vedení a podporu při mém pobytu v San Diegu. Dále bych chtěl poděkovat členům vědecké skupiny SO(3) a výzkumníkům z týmu projektu Visipedia. V neposlední řadě bych chtěl poděkovat rodičům za jejich podporu při studiu.

Acknowledgments

I would like to express my sincere thanks to professor Pavel Zemčík for leading this work, his precious advices and the coordination of my stay in the USA. I would especially like to thank professor Serge Belongie for his technical lead and support during my stay in San Diego. I would also like to thank to all the members of research group SO(3) and all the researches from the Visipedia project team. Finally, I would like to thank to my parents for their support during my studies.

Contents

Chapter 1

Introduction

In the last two decades machines have rapidly gained many new abilities of learning and image understanding. Some tasks such as face detection have already achieved a great performance and one can encounter these even in everyday life. Another very common and successful task is *classification.* Having a sufficient amount of images divided into categories, classification algorithms aim to learn to recognize those categories for new images. Each category thus groups images that are similar according to some criteria.

Although such classification systems can be very powerful, in some cases the categories may not be a convenient similarity representation. Sometimes it can be hard to decide how detailed the categorization should be, in some cases there might be too many categories while in others the categories may be hard to find, and sometimes they might not even exist. One may also be interested in more information than just a category – sometimes we want to know how much two images are similar or dissimilar according to a given criteria.

To challenge a larger understanding of image similarity without the need of their explicit categorization a different similarity representation has been increasingly used in recent years. Instead of assigning class labels to the images this representation "places" them in a multidimensional space where their mutual similarities are represented by the distances among them. More similar images thus lie closer to each other while the dissimilar ones are further away. This method is called *multidimensional embedding* since it embeds the images in a multidimensional space.

Multidimensional embedding has been widely used in a project called Visipedia that was created to challenge fine-grained categorization and better similarity understanding. A lot of work has been done in exploring the ways of getting multidimensional embeddings and searching through their space. Both of the tasks can be done completely without image processing or understanding, only by asking humans to answer simple questions. That can, however, get very expensive and time-consuming.

This work focuses on applying computer vision and machine learning approaches on images that are annotated with a multidimensional embedding. Its aim is to learn from images that are already annotated with such an embedding and explore possibilities of creating a system that would place new images to the embedding automatically. Such a system is important not only for adding new images to the embedding but also for searching the multidimensional space automatically using image queries. The algorithms can also be combined with human interactivity to get better results.

This work studies the relevant methods to create such a system, describes its development and analyses the performed experiments. It also presents new datasets and sets initial performance on them since no comparable work has been done on embedding-annotated images previously.

My choice of this topic was motivated not only by the fact I had the chance to join the Visipedia team, but especially due to my interest in computer vision and machine learning techniques in order to create universal learning systems. This topic was particularly challenging since this task was not explored before.

The Visipedia project and the role of this work are introduced in Chapter 2. Chapter 3 describes the concept of multidimensional embedding, presents its types, and outlines the possible ways of creating it. The extraction of visual features from images is studied in Chapter 4 as well as their fundamental properties and usage. Chapter 5 then analyses relevant machine learning approaches and places the embedding-driven learning in their context. The relevant learning methods are then studied in Chapter 6.

Chapter 8 describes the development of a library that implements the described functionality and presents a simple demo application. Chapter 9 introduces two different dataset and explains how they were obtained. Finally, Chapter 10 and Chapter 11 describe the performed experiments and analyze their results.

Chapter 2

The Visipedia Project

This work makes part of a large computer vision and machine learning oriented project called *Vispedia.* This chapter will briefly introduce the Visipedia project, describe its current state and its main objectives, and explain the role of this work for the project.

2.1 The Vision

The idea of Visipedia was first introduced in 2009 by professor Dr. Pietro Perona at California Institute of Technology (Caltech) [19] who has developed the concept with the help of his colleagues and a research group of professor Serge Belongie at University of California, San Diego (UCSD). Dr. Perona originally described his vision of Visipedia with the following words:

> “In order to see how one could improve the web and make pictures first-class citizens of the web, I explore the idea of Visipedia, a visual interface for Wikipedia that is able to answer visual queries and enables experts to contribute and organize visual knowledge.”

He claims that although images and videos make the vast majority of data on the web they are not well indexed, hyperlinked, and organized and thus form a “digital dark matter” that we cannot find easily. Also, experts do not contribute their Visual knowledge online since the process of image annotation and hyperlinking is too difficult and time-consuming.

Dr. Perona presented an idea to make pictures and videos “first class citizens of the web”. He proposed a system that would understand their content and that would help organizing the visual knowledge. Such system would let users browse the information starting with an image and using “visual queries”. This would be very useful in cases when we see an object and we want to find out what is it, what is its name, what does it serve for, etc. Dr. Perona demonstrated how hard is it to find such information using only textual queries and hyperlinks and claims that the current indexing of information on the web is not perfect and could be improved by higher understanding and indexing of the visual information.

Visipedia, as originally presented, would work as a visual layer over Wikipedia and it would generalize it in many ways. Textual hyperlinks would be accompanied with links from pictures to pictures, from pictures to text, and from text to pictures. Particular regions of images would thus become clickable hyperlinks. All of this would not be reached by manual image annotation but the people would collaborate with sophisticated computer vision and machine learning algorithms to make the task as quick and easy as possible.

2.2 The System

As already mentioned in Section 2.1, Visipedia was meant to be developed as a layer on top of Wikipedia that would augment the search capabilities with visual queries and generalize hyperlinks to work on both text and images. The content would be still editable by the users but also a high level of automation would be integrated. Dr. Perona described his concept of the system with the following words:

> "In order to make Visipedia a reality, we will need to develop software that can 'understand' automatically what is there in pictures, and link related 'visual concepts'. Humans would, of course, need to be 'there' and provide some guidance and information."

The idea of Visipedia went beyond a simple user-system interaction – Dr. Perona proposed to incorporate *humans-in-the-loop* techniques where human interaction would become a part of an algorithm [4]. This way he wanted to promote the system to a higher level of quality and he suggested five groups of humans to interact with state-of-the-art computer vision and machine learning algorithms:

1. *Users* would be using the system to find some information with textual and visual queries and browse the hyperlinked text and images.

2. *Experts* would provide and share their detailed knowledge with everybody, similarly as nowadays they are willing to share their knowledge in a textual form on Wikipedia.

3. *Editors* would be keeping the content clean, correcting errors and inconsistencies and keeping the form of the data standardized.

4. *Annotators* would provide the image annotations such as part segmentation, naming and identification. They would be likely paid to do so and they could be hired on some already existing crowdsourcing service such as Amazon Mechanical Turk.

5. *Automation providers* would be computer scientist working on machine learning and computer vision automation tools as well as on the system architecture itself.

When uploading an image to the system or making a visual query, an automatic segmentation and object detection would be done. Based on this relevant hyperlinks would be created automatically. However, since it is not yet possible to have this task completely automatized, it would be achieved with the help of humans and the machines would take the job over gradually.

The system would require the experts to annotate some prototype images manually and it would facilitate their work by posing them the most relevant and informative questions. Then the acquired knowledge would be generalized to other similar images that are already out on the web and the whole system would thus learn from the experts. The experts would later be asked to solve only some special and difficult cases while the easy ones would be done automatically.

Visipedia would also try to diagnose and improve the content automatically by detecting problems and uncertainties and choosing which questions to ask which groups of humans and which data should be analyzed by some of the algorithms. The human answers and the results of the algorithms would be evaluated in terms of their reliability and their certitude would be ensured by posing same questions multiple times.

In the beginning a lot of work would be done by humans using crowdsourcing services such as Amazon Mechanical Turk and the cost of the job and their speed would probably limit the size of the system. Later, more and more tasks would be transferred to machines and the system would start to grow faster. The development of automating algorithms would also be opened as the content itself and any computer scientist would be able to participate and supply new software.

The described architecture of the Visipedia system is depicted in Figure 2.2.

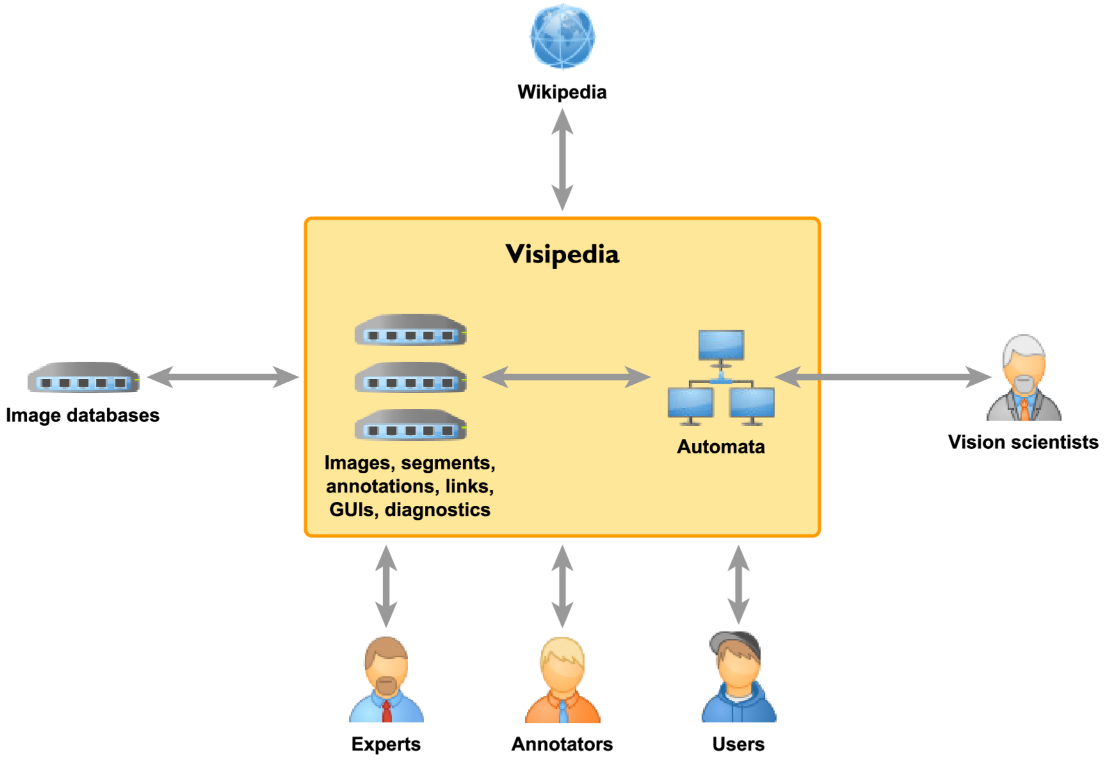


Figure 2.1: Architecture of the Visipedia system as proposed by Dr. Perona.

2.3 Visipedia-related Work

The Visipedia project started to come alive in two computer vision groups – Dr. Perona's group at Caltech and a group of professor Serge Belongie at UCSD. In collaboration with experts from Cornell Lab of Ornithology at Cornell University the two research groups started to work on Visipedia's first image taxon – *bird species*.

In the beginning the two research groups created a challenging image dataset of 200 bird species [31]. Unlike for other datasets where a human can achieve almost perfect classification accuracy the categories given by distinct bird species are often visually very similar and the visual borders among the categories are very thin. Most of the humans would probably perform badly in classifying bird species without using a field guide.

The new dataset have challenged a task of classifying objects with *fine-grained* categories. Since such task is very difficult also for a machine the researchers have developed algorithms with humans in the loop [4, 29], and used crowdsourcing techniques [30].

2.4 This Work and Visipedia

I first got in touch with Visipedia when I joined professor Belongie's group at UCSD and started to work on this thesis and other Visipedia-related projects. At the time my colleague and I arrived to San Diego, professor Belongie presented us his idea to work on a new taxon that would differ from birds in some ways.

While the visual boundary between two different bird species might be extremely thin, as said in Section 2.3, the species identification is still a classification problem with a limited amount of well-defined classes. Professor Belongie proposed to work on a new taxon where no explicit ground-truth categories would exist, such as food, fashion, furniture, faces, etc. Although one may argue these taxa can actually be categorized, we never can get the complete ground-truth classification of all the taxon.

The, non-existence of classes however does not mean that implicit similarities among the objects do not exist. Instead of being categorizable the similarities are continuous and instead of being labeled with classes their similarity can be captured with a *mutlidimensional embedding* that maps the object to a multidimensional space where their Euclidean distances among objects represent the similarity measure. The concept of multidimensional embedding is described in Chapter 3.

While some researchers from professor Belongie's group work on efficient ways of getting similarity measures with the means of crowdsourcing, the goal of this work is to explore how we can use machine learning algorithms (see Chapter 5) to discover the underlying relation between an embedding of images (see Chapter 3) and their visual features (see Chapter 4) and predict embedding coordinates for images out of the original embedding.

Chapter 3

Multidimensional Embedding

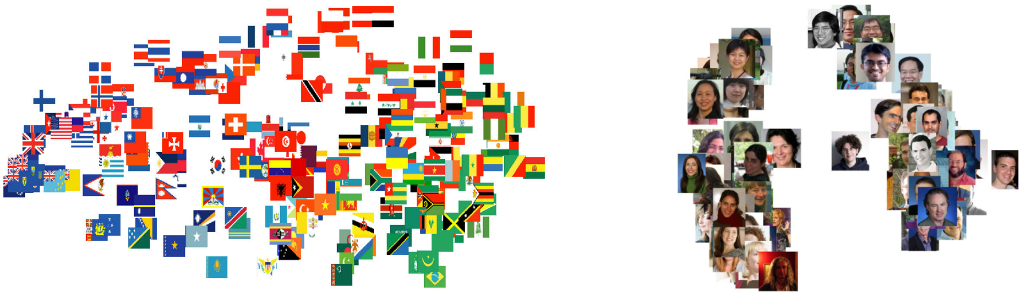
This chapter explains the concept of multidimensional embedding, presents its formal definition, and briefly describes some ways of creating it using a set of similarity comparisons among objects. Such objects can generally represent almost anything but for the purposes of this work images will be used. This text does not intend to provide an exhaustive study of the embedding problematic since its main concern is learning from already embedded objects. Instead, it presents basic terms and methods relevant in context of this work.

3.1 Motivation

As mentioned in Chapter 2, one of the Visipedia’s main goals is to work with fine-grained taxa that do not necessarily have a well-defined categorization. In other words, we still want to “order” or “group” images according to some criteria but we can’t label them with classes. The thing we’re looking for is a *similarity* measure – we’re interested in the information which objects are similar or dissimilar according to a given similarity criteria.

Such measure among a set of objects can be expressed with a *multidimensional embedding* which places the objects in a multidimensional space in a way that distances among them reflect the similarity measure. Figure 3.1 shows how such embeddings can look like in two dimensions, grouping more similar images closer together and pushing the dissimilar ones further apart.

Detailed study of multidimensional embeddings goes beyond the scope of this work since its main objective is to learn from an already computed embedding. However, it is necessary to define some embedding-related terms and show how the embeddings work.



(a) Flags

(b) Faces

Figure 3.1: Examples of multidimensional embeddings. Both (a) and (b) were projected on the top two principal dimensions from a higher-dimensional embedding. Taken from [24].

3.2 Multidimensional Embedding

Given a dissimilarity (distance) matrix $\mathbf{\Delta}$ and a set of objects X, a d-dimensional embedding is a map

$$\begin{aligned} f:\quad & X \to \mathbb{R}^d, \\ \text{such that}\quad & \forall x_i, x_j \in X:\ \|f(x_i) - f(x_j)\|_2 \approx \delta(x_i, x_j), \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} X &= \{x_1, x_2, \ldots, x_n\}, \\ \mathbf{\Delta} &= \begin{pmatrix} \delta(x_1, x_1) & \delta(x_1, x_2) & \ldots & \delta(x_1, x_n) \\ \delta(x_2, x_1) & \delta(x_2, x_2) & \ldots & \delta(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \delta(x_n, x_1) & \delta(x_n, x_2) & \ldots & \delta(x_n, x_n) \end{pmatrix}, \end{aligned} \tag{3.2}$$

and $\|\cdot\|_2$ denotes the Euclidean distance. The dissimilarity matrix $\mathbf{\Delta}$ is a square zero-diagonal symmetric matrix that defines pairwise distances for all pairs of objects from X. The distance between objects x_1 and x_2 is denoted by $\delta(x_1, x_2)$.

In other words, an embedding of objects X according to given similarity measures $\mathbf{\Delta}$ is a map of these objects to points in a multi-dimensional space, where the inner Euclidean distances among the points reflect their similarities. For instance, given all world's capital cities as objects and their air distance as a similarity measure their 2-dimensional embedding would be simply a map of the world.

3.3 Non-metric Multidimensional Embedding

Multidimensional embedding as defined in Section 3.2 requires having a measure of distance between every two objects. However, collecting such data might be a very difficult and expensive task. In some cases, such as representing the human perception of similarity, the collection of data with a metric of similarity might produce extremely noisy results and thus become almost impossible. Thinking of food as the given objects and its taste as perceived by humans as the similarity measure, it would be very hard to find a numeric value representing the taste distance between two meals.

The difficulty of collecting similarity metrics does not necessarily imply the non-existence of such data. There may be some ground-truth dissimilarity matrix but in this case it is hidden to us. Going back to the example with food it would be probably easier for humans to provide an information in form "a is more similar to b than to c" than guessing the exact taste metric between two meals. Having such information we can generalize the embedding definition into a *non-metric* embedding.

Given a set of triplets $\mathcal{T}$ and a set of objects X, a d-dimensional non-metric embedding is a map

$$\begin{aligned} f:\quad & X \to \mathbb{R}^d, \\ \text{such that}\quad & \forall (x_i, x_j, x_l) \in \mathcal{T}:\ \|f(x_i) - f(x_j)\|_2 < \|f(x_i) - f(x_l)\|_2, \end{aligned} \tag{3.3}$$

where

$$\begin{aligned} X &= \{x_1, x_2, \ldots, x_n\}, \\ \mathcal{T} &= \{(a, b, c) \in X^3 \,|\, a \text{ is more similar to } b \text{ than to } c,\ a \neq b \neq c\}, \end{aligned} \tag{3.4}$$

and $\|\cdot\|_2$ denotes the Euclidean distance.

In other words, an embedding of objects X according to given set of similarity triplets $\mathcal{T}$ is a map of these objects to points in a multi-dimensional space, where the inner Euclidean distances preserve the triplet inequalities. Going back to the example with World's capital cities and supposing we have a sufficient amount of triplets in form "city A is closer to city B than to city C", the 2-dimensional embedding built from such triplets would be a map of the World where the distances among the cities would be only relative – they would not have a real metric like miles or kilometers. Although such map might be deformed by rotation, scaling, and other transformations, a lot of important information would be preserved. We would still be able to say whether any two cities are closer together or further apart than any other city pair.

An important thing to note about the non-metric variant of the multi-dimensional embedding is that we do not need to have all the possible triplet comparisons on a given set of objects. Even with just a small fraction of all the possible triplets we can still build an embedding that would satisfy all the triplet inequality constraints. The number of triplet comparisons have of course an impact on the quality of the embedding, but generally speaking, it is not impossible to build a good-quality embedding without having all possible triplet comparisons.

3.4 Creating Embedding from Similarity Comparisons

Once we have collected a sufficient amount of similarity comparisons of the triplet form "a is more similar to b than to c" we can create an embedding representing the similarity distances among objects. An embedding that satisfies a given set of triplet inequalities can be defined in many different ways since only the relative distance inequalities have to be preserved and we need no exact measures. Various different methods of building such embedding thus exist and produce different results. Each of the methods have different properties and might be suitable for different use cases.

Generalized Non-metric Multidimensional Scaling (GNMDS). GNMDS aims to find an embedding where the pairwise distances between embedded objects satisfy the similarity triplets [1]. This is achieved by solving an optimization problem where each of the triplets becomes a constraint. GNMDS is suitable for data with low percentage of errors and with a large the number of correct triplets it can produce very good results. However, its performance can drop significantly while working with noisy data.

Crowd Kernel Learning (CKL). CKL defines probabilities that measure how well each of the triplets is modeled [24]. It was designed to learn an embedding from triplets provided by humans and it aims to choose the triplet questions adaptively to optimize the number of necessary answers. Therefore, it can be suitable in situations when we want to minimize the number of necessary triplets to lower the cost of data collection.

The main problem of CKL is that it is mainly focused on correcting large constraint violations in the embedding that can be often caused by noise and uncommon responses to the triplet question and can thus damage the resulting embedding.

Stochastic Triplet Embedding (STE). STE works much more locally than the previous methods to address the problem noisy and uncommon triplets in the data. It assigns nearly a constant penalty for largely violated triplets and nearly a constant rewards to triplets that are well satisfied and thus ignores the uncommon triplets that could be produced by noise. The main problem of STE is that it is hard to fix errors that were done in early steps of the computation.

T-Distributed Stochastic Triplet Embedding (t-STE). The problem of early made errors during the STE embedding creation was solved by using a Student-t kernel function to measure local similarities between data points. The resulting method is called t-Distributed Stochastic Triplet Embedding (t-STE) [26]. The t-STE method not only lowers the sensibility to noise in the data but also improves the embedding even when a triplet constraint is already satisfied. It thus pushes points together when there are no triplets to keep them apart and pulls them apart when there are no triplets to keep them close. Both these properties make t-STE particularly suitable for human provided triplets based on their perception of similarity that can contain noise and incompatibilities.

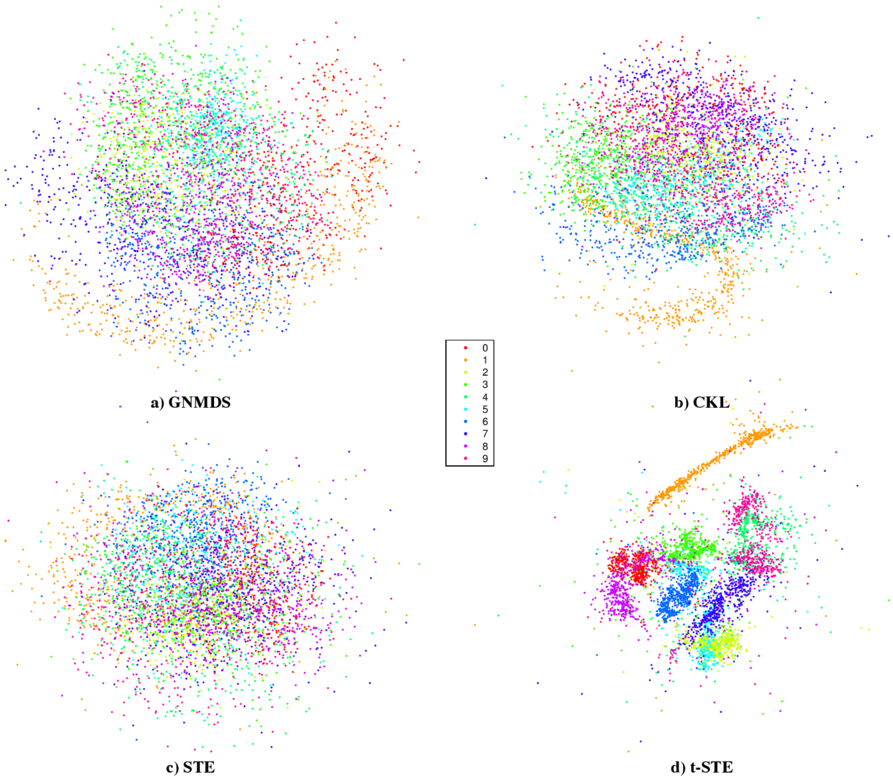


Figure 3.2: Embeddings of the MNIST digit dataset produced by four different methods from automatically generated triplets based on the differences in pixel values among images. The advantages of t-STE are nicely visible – pushing similar points close together and pulling dissimilar ones further apart even when the constraints are satisfied results in a better separation of the digit classes. Taken from [26].

Embeddings computed using all the described methods are depicted in Figure 3.2. Since the embeddings used in the Visipidea related projects were usually based on human provided triples according to their perception of some similarity criteria, t-STE appears to be the most suitable method for the majority of such tasks. However, in particular cases some of the other methods might be useful as well and the choice depends on the type of the task.

Chapter 4

Visual Features

Visual feature extraction is a process of getting useful information from images and thus makes an essential part of computer vision and image-oriented machine learning systems. Different visual features extract different information and should be selected depending on the addressed task. In order to create a robust and universal system multiple visual features can be used to get more information from a single image.

This chapter describes some of the most common and powerful feature extraction methods covering different image properties such as color, shape, texture and stable regions. The aim is not to list all existing feature extraction methods but to analyze the most popular and efficient ones for each of the named image properties.

4.1 Color Histogram

A color histogram is one of the fundamental image features used in the computer vision field. Its role is to describe the color or intensity distribution in an image which makes color histograms particularly useful for any visual task where color or intensity matters.

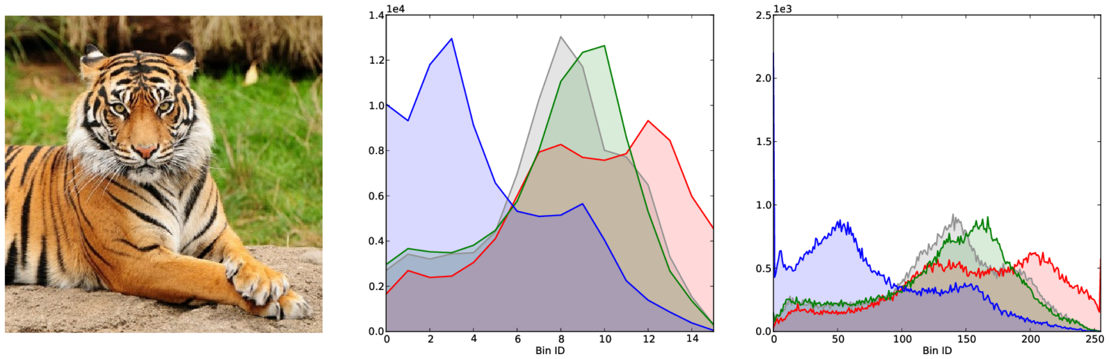
In its simplest form color histogram characterizes the amount of pixels that belong to each of previously defined color or intensity ranges that are usually called *bins*. A color histogram computed for a grayscale image will extract information about the intensity distribution of its pixel values. For color images it is usually computed for each color component given by the used color model.

Formally, given a set of pixel color values – an image I – and a set of disjoint intervals $\mathcal{B} = \{B_1, B_2, ..., B_n\}$, where $\bigcup_{B_i \in \mathcal{B}} = [\min I, \max I]$, a color histogram is a map

$$\begin{aligned} H:&\quad \mathcal{B} \to \mathbb{N} \\ \text{such that}&\quad H(B_i) = |\{p \,|\, p \in B_i,\, p \in I\}|, \quad \sum_{B_i \in \mathcal{B}} H(B_i) = |I|. \end{aligned} \tag{4.1}$$

Such color histogram is a purely *global* feature since it extracts information about the color range quantities in whole image. Sometimes it is desirable to extract the color distribution along with its approximate position in the image. In that case we can simply divide the image in smaller blocks and compute the histogram for each of them. Then we speak of *local histograms*. The choice of the number and size of the bins have an impact on the information that we get from the image. Lower amount of larger bins describes the general tone of an image while big amount of smaller bins can capture the details.

Examples of global color and intensity histograms with different numbers and sizes of bins are depicted in Figure 4.1.



(a) Original image (b) 16 bins (c) 256 bins

Figure 4.1: Color and intensity histograms with different numbers of bins.[1]

4.2 Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients (HOG) is a feature descriptor that captures localized shapes in an image by computing the distribution of intensity gradients in smaller image regions [6]. HOG is a powerful descriptor especially for object detection and with cautious parameter fine tuning and local contrast normalization it outperforms most of the previous similarly focused descriptors.

The first step of the HOG feature extraction algorithm is normalization of color and gamma values in the image. However, this step can usually be omitted since the normalizations bring only a modest improvement, probably due to later descriptor normalization.

The second step is the gradient computation which is done by applying simple 1-D derivative masks in both horizontal ($\mathbf{f}_h$) and vertical ($\mathbf{f}_v$) directions. In particular, filters

$$\mathbf{f}_h = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{f}_v = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{4.2}$$

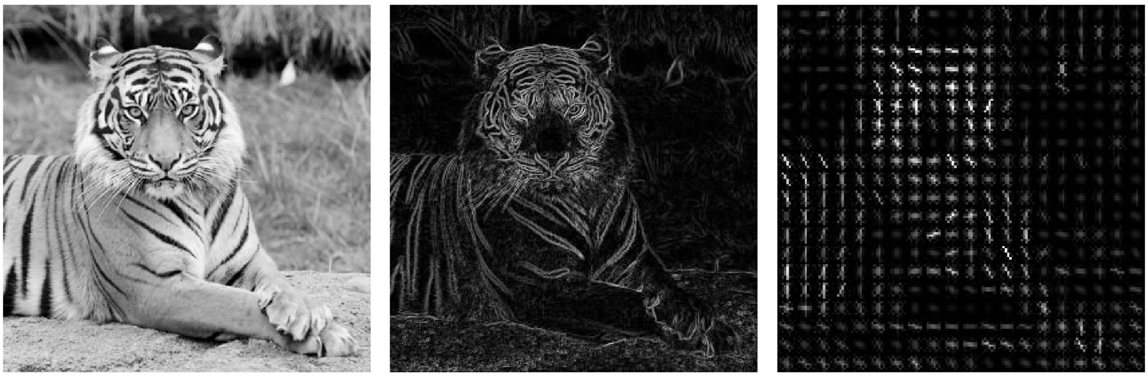
are frequently used since they are known to have a very good performance. For a colored image the filters are applied on each of the color components, whereas for a grayscale image the intensity is used.

The oriented gradients are then represented by two values – magnitude ρ and orientation θ – that can be obtained from the two filtered images by applying Cartesian to polar coordinate conversion. For pixel p_h from the gradient image produced by filter $\mathbf{f}_h$ and its corresponding pixel p_v from the gradient image obtained by filtering with $\mathbf{f}_v$ the values of ρ and θ can be computed as

$$\rho = \sqrt{p_h^2 + p_v^2} \quad \text{and} \quad \theta = \arctan \frac{p_v}{p_h}. \tag{4.3}$$

In the third step the image is divided in a uniform grid of small local regions called *cells*. Based on the computed gradient values each pixel in a cell performs a weighted vote for an edge orientation histogram that accumulates the votes for each cell in *orientation bins*. The best results are achieved by simply taking the gradient magnitude as a vote and using 9 gradient orientation bins in range from 0° to 180° while ignoring the gradient sign. The gradients magnitudes and orientation histograms are visualized in Figure 4.2.

[1] Tiger image taken from `http://www.desktopwallpapers4.me/`, used also further in this chapter.



(a) Input image (b) Gradient magnitude (c) Orientation histograms

Figure 4.2: HOG visualisation. Figure (a) shows the input grayscale image, (b) illustrates the magnitude of the gradients (the higher the magnitude, the lighter the image), and (c) visualizes the orientation histograms in the cells (histogram value for each bin is shown as a light line rotated according to the bin).

The last step of HOG extraction normalizes the gradients over larger image regions called *blocks*. A block is a group of several cells and adjacent blocks are partially overlapped so that each cell contributes multiple times to the final feature vector, each time normalized across different block. This step is crucial for the descriptor quality since local illumination and contrast changes are frequently present in the input images.

The blocks can be either rectangular (usually square, referred to as R-HOG) or circular (referred to as C-HOG). To improve performance the rectangular blocks can downweight the pixels near the block edges by applying a Gaussian spatial window. Although both block shapes perform quite similarly, the best choice depends on the particular data. The HOG's authors state that one of the best settings is a square block containing 2×2 or 3×3 cells of 8×8 pixels, where the blocks overlap by $^3/_4$.

The HOG features are particularly useful in capturing shapes of objects in images and are therefore often used for object detection. The use of localized cells and normalization using larger overlapped blocks makes them invariant to illumination and contrast changes and some geometric transformations.

4.3 Local Binary Patterns (LBP)

Local Binary Patterns (LBP) is a feature descriptor designed for texture and pattern characterization. The main idea of LBP is to describe the texture by its local spatial structure in a grayscale image and compute a histogram of these local structure descriptors. This way LBP combines structural and statistical approaches of texture characterization and proves to be a powerful method of their description.

The LPB is computed by thresholding the neighborhood of each pixel by its value and encoding the result as bits of a single unsigned integer. A histogram of these integers is then used as the feature descriptor. Originally, a neighborhood of 8 pixels was used producing histogram of $8^2 = 256$ values [14, 15]. The algorithm was later extended with rotation invariance, improved with so called uniform patterns, and generalized for any size of the neighborhood [16, 17, 18]. Some of the typical texture patterns captured by LBP are depicted in Figure 4.3.

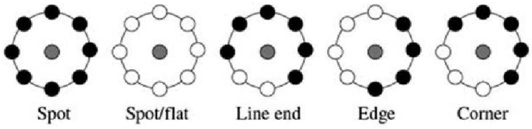


Figure 4.3: Some of the basic patterns that can be captured with LBP.[2]

4.3.1 Generalized LBP

Given a pixel p and a set $N = \{n_0, n_1, \ldots, n_{P-1}\}$ of its P neighboring points lying on a radius R from p, the generalized LBP is computed as

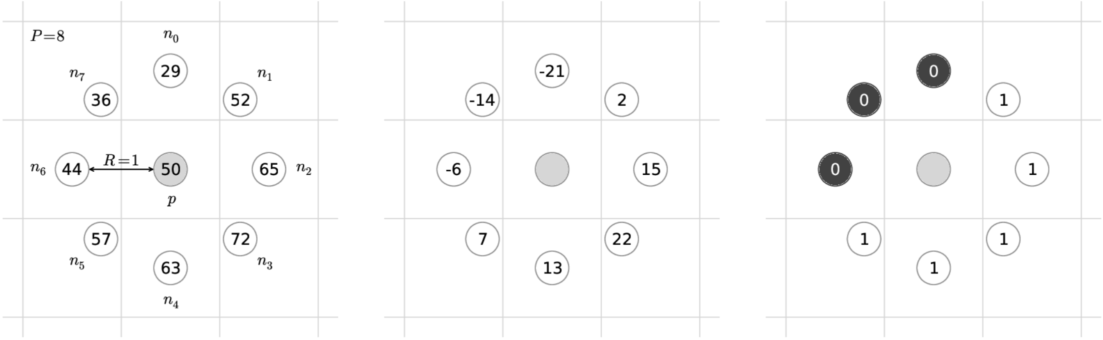
$$LBP_{P,R} = \sum_{i=0}^{P-1} s(n_i - p)\, 2^i, \tag{4.4}$$

where

$$s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \tag{4.5}$$

In other words, the pixel p is compared to each of its neighboring points n_i and if $n_i \geq p$, number 1 is assigned to the comparison, 0 otherwise. The values of points that are not on integer pixel positions are bilinearly interpolated. To express the results for all the points in a neighborhood with a single number, the comparison for neighbor n_i is multiplied by 2^i and the values for all the points are summed. This is equivalent to expressing the sum as a binary number of P digits and setting the value of $(i+1)$-th digit from right to 1 or 0, depending on the result of the comparison. This process is depicted in Figure 4.4.

Using the thresholded local differences $n_i - p$ between the central and neighboring pixels, the LBP is by definition invariant to any monotonic transformation of the gray scale. This means that any intensity change where the order of gray values is preserved does not affect the feature descriptor. This makes LBP very robust to illumination changes.



(a) Original pixel values (b) Differences $s_i - p$ (c) Thresholded values

Figure 4.4: LBP visualization for $P = 8$ and $R = 1$. Figure (a) shows the original values of pixel p and its neighbors n_i, (b) demonstrates the state after computing $n_i - p$, and (c) contains the results of $s(n_i - p)$. The next step is multiplication of the thresholded values by 2^i and summing the results. In this case it would be $LBP_{8,1} = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 0 \cdot 2^7 = 62$ or simply $00111110_2 = 62_{10}$.

[2] Image taken from http://what-when-how.com/face-recognition/.

4.3.2 Rotational Invariance

When an image is rotated, the neighboring pixels of the central point p get different values of i and thus are multiplied by a different value 2^i. In other words, when rotating the input image, the bits of the LBP expressed as a binary number rotate as well. Therefore, to achieve rotation invariance, the bits of the binary LBP are rotated to all possible locations and the smallest of the resulting values is chosen. Formally, a rotation invariant LBP is defined as:

$$LBP_{P,R}^{ri} = \min\{ROR(LBP_{P,R}, i) \,|\, i = 0, 1, \ldots, P-1\}, \tag{4.6}$$

where $ROR(x, i)$ is a circular bitwise right shift of a binary number x in amount of i bits. The $LBP_{P,R}^{ri}$ descriptor is also smaller since for P neighbors it makes $P-1$ times less possible combinations than $LBP_{P,R}$.

4.3.3 Uniform Patterns

Uniform patterns is an extension improving the $LBP_{P,R}^{ri}$ operator. The idea of the uniform patterns is based on the fact that some of all the possible rotation invariant LBP patterns occur much more often than others. Such patterns are called *uniform* and sometimes form more than 90 percent of all the patterns in a texture. The property they have in common is a uniform circular structure that contains very few spatial transitions. They describe local structures such as dark and bright spots, flat areas and edges.

The uniform patterns, denoted by $LBP_{P,R}^{riu}$, are defined by the number of bitwise transitions (0 to 1 and 1 to 0) that occur in the binary LBP value when traversed circularly. All the patterns that have at most 2 bitwise transitions are called uniform. For a neighborhood of P pixels there is exactly $P+1$ possible uniform patterns. All the possible rotation invariant and uniform patterns for $R = 1$ and $P = 8$ are shown in Figure 4.5.

Since the uniform patterns form the vast majority of the texture but there is only $P+1$ different shapes of them, the descriptor size is reduced by labeling all the non-uniform patterns as a single pattern shape. This makes only $P+2$ different bins in the resulting histogram, $P+1$ for the uniform patterns and one for the rest. Such histogram is not only smaller but also discriminates the textures better avoiding very small values for each of the non-uniform patterns that could be affected by noise.

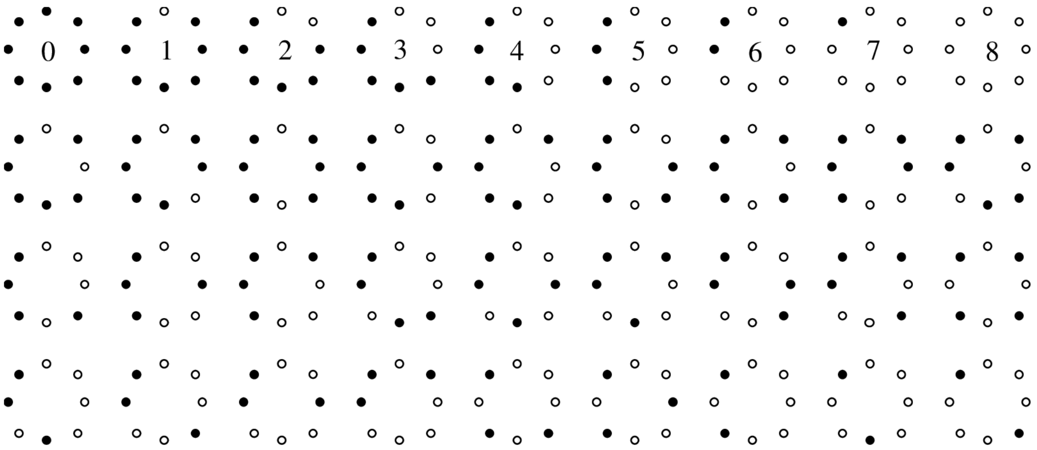


Figure 4.5: All the existing rotation invariant LBP patterns for $R = 1$ and $P = 8$. The patterns $0-8$ in the first row are uniform. Taken from [16].

4.4 Scale-invariant Feature Transform (SIFT)

Scale-invariant Feature Transform (SIFT) is a robust local feature detector and descriptor designed especially for object recognition and related tasks [12, 13]. It extracts stable point-based features that are invariant to scaling, translation and rotation, and partially robust to illumination changes and affine or 3D projection. The algorithm is inspired by the behavior of neurons that participate in primate vision and the scale invariance, high performance and ability to detect partially occluded objects make it very efficient.

4.4.1 SIFT Key Point Detection

The first step of the algorithm is the identification of key locations in the image that can be found in all image scales. The locations are found by selecting minima and maxima of a difference of Gaussian function (DoG) computed in so called scale space [10] with a Gaussian kernel. The scale space consists of multiple levels of blur applied on the input image. To achieve high performance the smoothing is done incrementally and the blurred images are stacked in octaves of a scale pyramid as shown in Figure 4.6. To proceed to the next level of the scale pyramid, the most blurred image is resampled to a smaller size using bilinear interpolation. SIFT does fast blurring by convolving the image in both horizontal and vertical directions with a 1D Gaussian function

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \tag{4.7}$$

where σ is fixed to $\sigma = \sqrt{2}$ and higher smoothing is done by applying the same filter on an already blurred image. The DoG is a band-pass filter that is calculated by subtracting a blurred version of an image from a less blurred version of the same image. For image I the DoG is then computed as $DoG = A - B$, where $A = I * g$, $B = A * g$ and $*$ denotes convolution.

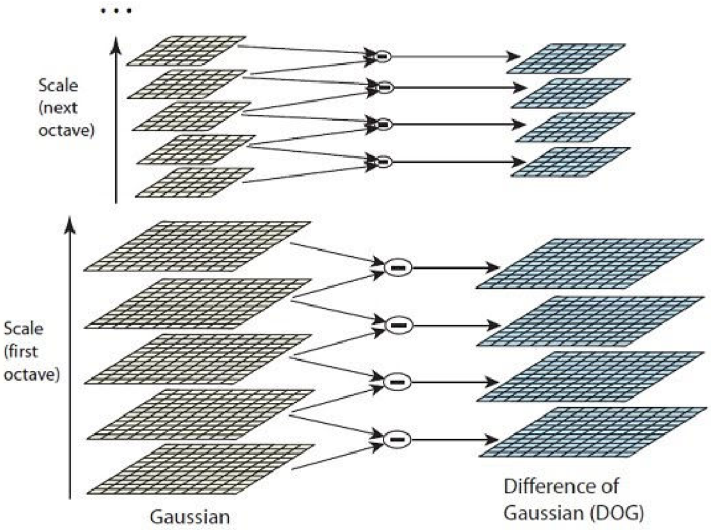


Figure 4.6: DoG function in a scale space. The scale space function is computed by incremental application of Gaussian blur and image resampling in octaves (on the left). The DoG function is then applied on adjacent scale steps in each octave (on the right).[3]

[3] Image taken from http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/.

The minima and maxima of the DoG are then selected by comparing pixels through the space pyramid. Each pixel is initially compared to its surrounding pixels at the same level of the pyramid. If it is a minima or maxima, the pixel is compared to its closest neighbor on the next level of the pyramid. If even then the pixel remains the minima or maxima, the process is repeated on the next pyramid level. When the chosen pixel is not a minima or maxima of the points we are comparing with, the process is stopped, making the search very efficient since most of the pixels are eliminated quickly.

4.4.2 SIFT Key Point Description

When the key points are identified, it is desirable to describe them in a way that would be invariant to geometric transformations and local changes in the image. Since the key point location and scale are already identified, the translation and rotation robustness can be satisfied.

To describe the image at each local region, oriented gradients are extracted from the first smoothed image on each level of the pyramid. The oriented gradients consist of magnitude ρ and orientation θ and for an image I the gradient at pixel $I_{i,j}$ is computed as follows:

$$\begin{aligned} \rho_{i,j} &= \sqrt{(I_{i,j} - I_{i+1,j})^2 + (I_{i,j} - I_{i,j+1})^2} \\ \theta_{i,j} &= \mathrm{atan2}(I_{i,j} - I_{i+1,j} + I_{i,j} - I_{i,j+1}) \end{aligned} \tag{4.8}$$

The gradients are then used to compute local histograms of gradient orientations and the peak in each histogram is selected as a canonical orientation for every key point. Having a canonical orientation for each key point the rotation invariance can be accomplished.

To achieve the partial invariance to illumination changes, the gradient magnitudes are thresholded at a given level relative to their maximal value. This gives more importance to the gradient orientations which are less likely to be influenced by an illumination change.

The robustness against distortions caused by affine and 3D projections is implemented by so called orientation planes – multiple images created for each local region, each of them containing only gradients corresponding to one orientation. To achieve the invariance to local distortions, the orientation planes are then blurred and resampled.

SIFT is a powerful key point detector and descriptor that can be used for object detection, key point matching between two images, classification and regression tasks, etc. Its matching abilities can be seen in Figure 4.7.

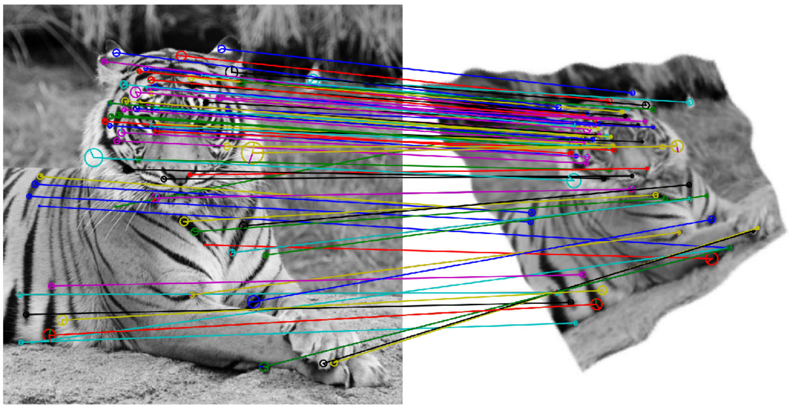


Figure 4.7: SIFT key point matches between an image and its copy that was damaged by scaling, rotation, projection, deformation, change of brightness, noise, and Gaussian blur.

4.5 Bag of Visual Words (BoVW)

The regression learning and predicting algorithms such as SVR (see Section 6.4) require their inputs to be feature vectors of the same length for all input images. However, some of the feature extraction algorithms such as SIFT (see Section 4.4.2) produce feature vectors of different sizes since the number of extracted features depends on the amount of detected information in the image. SIFT detects and describes key points in images but different images can have different number of them which means that we are not able to construct feature vectors of constant length that would be ordered alike.

Some other feature extraction algorithms such as HOG (see Section 4.2) produce feature vectors of constant length but only if all the images have the same size. One approach of facing this problem is to resize all the images to the same dimensions but this could damage them when their proportions vary significantly.

To solve the above mentioned problems we can use a technique called *bag of visual words* (BoVW) [7]. The BoVW method is analogous to *bag of words* method that is widely used in learning from written text. The main idea of BoW is to create a "bag" of different words that appear in all the documents and then compute their occurrences in each of the documents, i.e. for each document we compute a histogram of its words. The BoVW applies the same principle with the difference that instead of words we have encoded features. However, the number of different features can be very high which means that we need to define larger bins for the histogram o visual words. BoVW usually determines these bins by clustering the encoded features with k-means algorithm.

4.5.1 *k*-means

The k-means clustering algorithm aims to partition data into a given number of clusters where each sample lies in the cluster that has the nearest mean. Formally, for a set of n data vectors $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the k-means algorithm divides the data into $k \leq n$ sets $C = \{C_1, C_2, \ldots, C_k\}$ called *clusters* by minimizing the within-cluster sum of squares. This is done by solving minimization problem of form

$$C = \arg\min_{C} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \overline{C_i}\|_2^2, \tag{4.9}$$

where $\overline{C_i}$ denotes the mean of C_i. The k-means clustering algorithm has a very high computational complexity and some of its implementations thus use various heuristic algorithms.

Chapter 5

Machine Learning

Machine learning is a large field in computer science, a subdomain of artificial intelligence that is concerned with design and development of systems capable of learning from data. Its importance lies mainly in solving problems where an analytical solution is unknown, extremely hard to find or unfeasible.

This chapter presents the machine learning approaches that are essential for the design of an embedding-driven learning system, explains the fundamental related terms, and places the task of embedding-driven learning in their context.

5.1 Definition

Probably the first definition of machine learning was given by Arthur Samuel in 1959. Samuel was an American scientist who is considered to be the first to develop a self-learning program. He is believed to have defined machine learning as a

> "Field of study that gives computers the ability to learn without being explicitly programmed."

Although such definition does not appear in either of the usually referred articles [20, 21], it is without doubt that Samuel was a pioneer of the machine learning field. It worths noting that computers usually have to be programmed in order to learn something, but – what this definition probably intends to say – they do not have to be explicitly programmed to solve a specific problem because the method of solving the task will be learned instead.

Tom Michel, an American computer scientist and a professor of machine learning, provided another definition in his book Machine Learning published in 1997 [25]:

> "The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience."

In the same book he also defined machine learning in a more formal way:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Michel accompanied this definition with an example of a program that is learning to play checkers – the performance measure P would be its ability to win, the set of tasks T would be represented by playing games and the experience E would be the knowledge obtained when playing games against itself.

5.2 Generalization, Underfitting and Overfitting

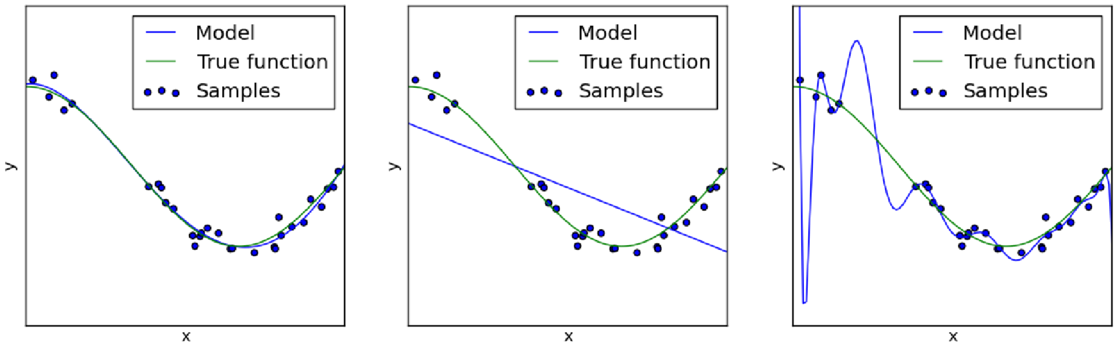
The main objective of machine learning algorithms is to learn underlying hidden patterns in the input data and to produce accurate predictions for unseen inputs based on the gained knowledge. This key concept of machine learning is called *generalization*. According to authors of Machine Learning: An Artificial Intelligence Approach [3], generalization can be defined as

> "Extending the scope of a concept description to include more instances."

Generalization is a crucial capability of a well-operating learning system. Failure to generalize means that we can have no confidence in the predictions performed on unseen data.

One type of generalization failure may occur when we do not have enough training data or the learning algorithm does not train enough. Such phenomenon is called *underfitting*. The underfitting problem may be solved by using more data or improving the learning algorithm, often with the cost of increased computational complexity.

Another problem may occur when the system is too complex or the algorithm trains too much. The system is then loosing the ability to generalize in favor of specializing to a particular training set and noise in its data. The prediction accuracy might significantly decrease. This problem is called *overfitting* and its elimination usually requires extending the learning algorithm with a mechanism of overfitting prevention. Different learning methods have different sensibility to the overfitting problem and thus the way of its prevention depends on the used learning method.



(a) Generalization (b) Underfitting (c) Overfitting

Figure 5.1: Generalization, underfitting and overfitting. The aim is to model the true function from a limited amount of noisy samples. While (a) learned the function almost perfectly, both (b) and (c) can produce large errors for unseen samples.[1]

The differences between generalization, underfitting and overfitting are visualized in Figure 5.1. Both underfitted and overfitted systems can produce large errors and fail to give reasonable predictions for unseen samples.

A universal and reliable way of avoiding overfitting and choosing the best learning parameters to face unnecessary underfitting is to split the input data to training and testing sets and evaluate the errors as described in Section 5.3 together with cross validation explained in Section 5.4.

[1] Images taken from `http://scikit-learn.org/`.

5.3 Training and Testing

Training (also called *fitting*) is a process of acquiring knowledge from a previously prepared set of data called *training set*. The aim of training is to find an underlying structure or pattern in the data and generalize the the knowledge for items out of the training set. In other words, we are trying to find a general *model* of the data. The quality and size of the training set can significantly influence the quality of the model which makes the training set an important part of machine learning systems.

Testing (also called *predicting*) is a process of predicting outputs for data that were not used during training. For evaluation and verification of the quality of the model a dataset with known ground truth outputs is usually used and the results of the prediction are compared to the known ground truth values. Such dataset is called *testing set*. However, the term “testing” is not used only to test the model and evaluate its quality but also when using the trained model in production on data with unknown ground truth outputs. The “production phase” can be thus equally called “testing phase”.

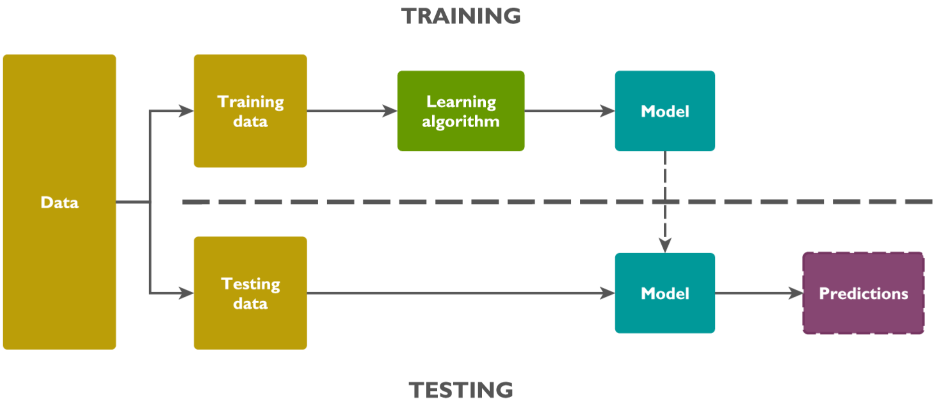


Figure 5.2: Training, testing, and a model. The data is first split to training and testing sets, an algorithm then learns a model from the training data which is then used to compute predictions for the testing data. Those can be used to evaluate the quality of the model.

While evaluating a learning method and tuning its parameters, training and testing is usually combined with so called cross validation techniques that aim to avoid bias that could be caused by the specificity of a single training and testing set. Cross validation is useful also for avoiding overfitting and tuning the parameters of the learning algorithm and its principles are described in Section 5.4.

The resulting product of the training step is a general model of the hidden function we were looking for estimated from the training data. A well-fitted model should be able to give reasonably good predictions for previously unseen data samples. The basic training and testing pipeline is depicted in Figure 5.2.

Most of the current machine learning algorithms require large sets of data to be able to produce a well-fitted model. Depending on the the task and the used method at least hundreds or thousands of items are usually necessary. Insufficient amount of training data may result in a bad underfitted model. This makes the data collection also and important part of the machine learning process.

5.4 Cross Validation

Cross validation [11] is a method of evaluating the performance of a machine learning system. It can be also used to tune up the parameters of a learning algorithm and avoid the problem of overfitting (see Section 5.2).

The main idea of cross validation is to split the data in training and testing sets (see Section 5.3) multiple times and evaluate the results on each iteration. In each of the iterations the training and testing sets contain different samples from the original data. The system is thus evaluated multiple times and the results are averaged. This way cross validation avoids bias in the quality measures that would be caused by the selection of a single training and testing data.

The most common method of cross validation used in machine learning is called *k-fold cross validation*. The main idea of k-fold cross validation is to split the original data in k even parts and perform k training and testing iterations. For i-th iteration the i-th last part of the data is used as a testing set while the resulting parts are used for training. An example of k-fold cross validation is shown in Figure 5.3.

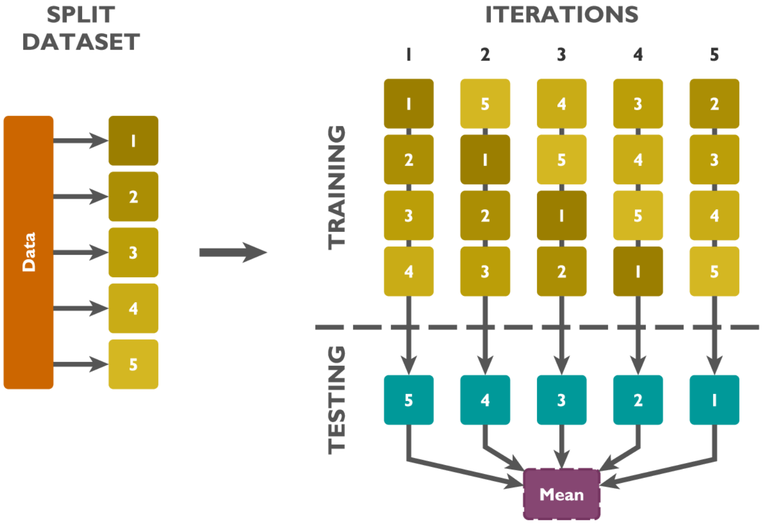


Figure 5.3: An example of 5-fold cross validation. The data is split to 5 parts and 5 iterations of training and testing are performed while averaging the results. For i-th iteration the i-th last part of data is used for testing while the rest serves for training.

When the value of k in the k-fold cross validation equals to the number of samples in the input data, we speak of *leave-one-out cross-validation*. Leave-one-out cross-validation thus uses only a single data sample in the testing set in each iteration and the number of iterations is equal to the number of data samples.

Another method of cross validation splits the data to training and testing sets randomly and then performs the learning and evaluation. Such approach is called *repeated random subsampling* and its main advantage is that the number of iterations is completely arbitrary and does not depend on the number and size of data parts. On the other hand some samples may never be selected to the validation set.

5.5 Unsupervised Learning

Unsupervised learning algorithms learn from *unlabeled* data [2]. This means that during training we have pure data with no additional information. The aim of unsupervised learning algorithms is to discover hidden structures and patterns in the data itself and create a model that describes them in a generalized way.

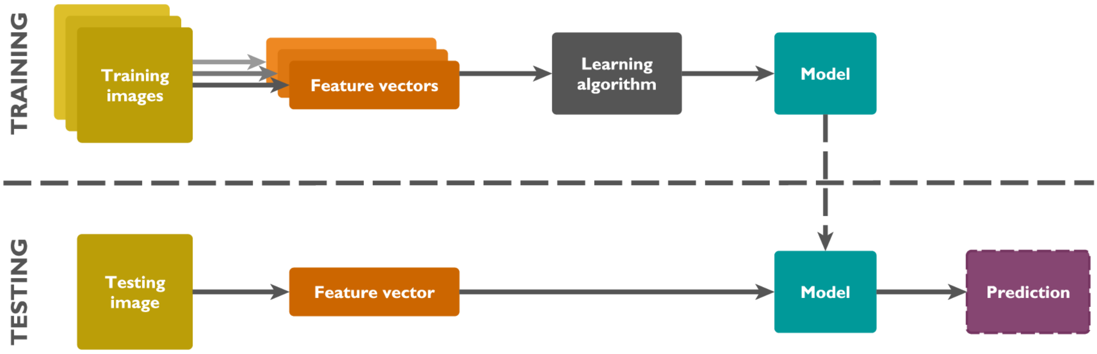


Figure 5.4: Unsupervised learning from images. The system learns directly from the feature vectors with no additional data. A prediction can be for instance a cluster identifier.

Unsupervised learning methods include clustering algorithms, hidden Markov models, blind signal separation and other approaches. Since the data samples are unlabeled we have no error values to evaluate the quality of the predictions. Unsupervised learning is used to discover hidden groups of similar examples in the input data.

Figure 5.4 displays a typical pipeline of training and testing phases of unsupervised learning from images.

5.6 Supervised Learning

Supervised learning algorithms learn from data with *labels* [2]. A label represents the desired output of the prediction for a given example. The aim of supervised learning algorithms is to find a function that would map data to their labels and that would be able to give correct predictions for previously unseen inputs.

Supervised learning algorithms can be easily evaluated since we know the values of the desired outputs. A typical pipeline of training and testing phases of supervised learning from images is displayed in Figure 5.5. Depending on the character of the labels supervised learning can be further divided in two categories – *classification* and *regression*.

5.6.1 Classification

When the data labels have limited amount of discrete values, the task of mapping data to labels is called *classification*. All the possible discrete values of labels are then called *classes* and the algorithms to solve classification problems are often called *classifiers*. A classifier is a function that predicts a class for any given data including unseen inputs. Formally, for a set of n d-dimensional data vectors $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n\}$ and a set of classes C, a classifier is a function

$$f : \mathcal{X} \to C. \tag{5.1}$$

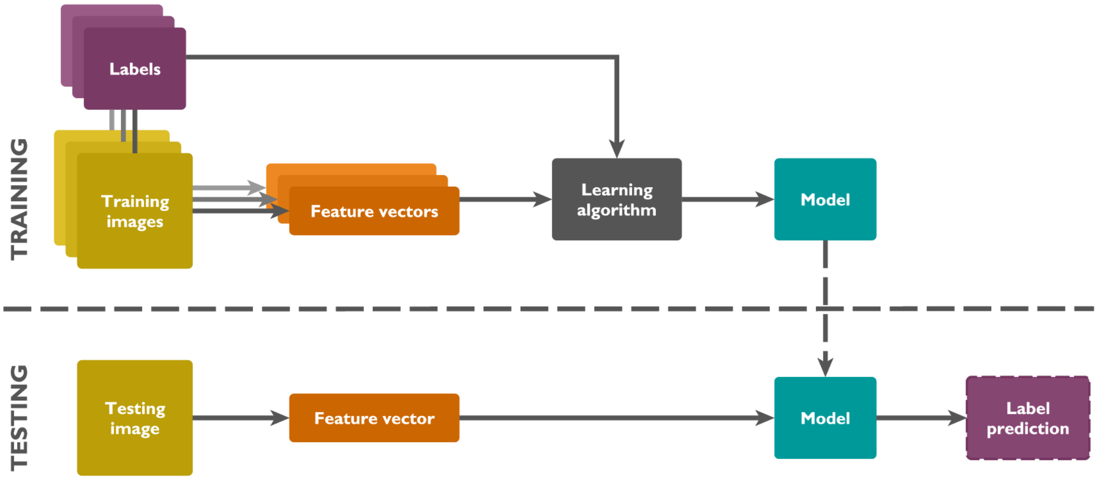


Figure 5.5: Supervised learning from images. The system aims to learn the relation between feature vectors and data labels. A prediction is an estimated label value.

A classifier with only two classes is called *binary*. Binary classifiers are important for predicting whether an objects belongs to a given class or not. This is typical for *detection* tasks when we want to know if some object is present in the data or not. An example can be a decision whether an image frame contains a human face or not.

Classification with more than two classes is called *multiclass classification*. This is often used for *recognition* problems when we want to know which object is present in the data. An example could be a decision who's face is in an image frame.

5.6.2 Regression

The task of mapping data to labels with continuous values is called *regression*. Regression is a statistical method that aims to find a continuous function that would capture the relation between data and labels and that would be generalized for unseen inputs. Such function that maps data vectors to scalar label values is usually called *regressor*. Formally, for a set of n d-dimensional data vectors $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$, a regressor is a function

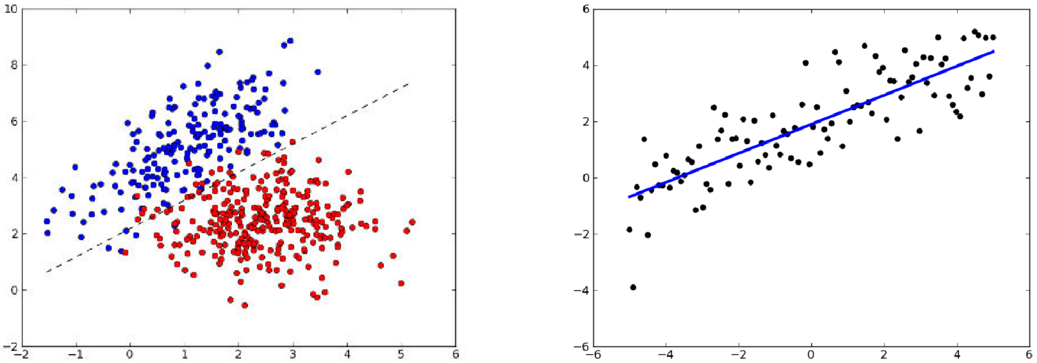
$$f : \mathcal{X} \to \mathbb{R}. \tag{5.2}$$

When we expect the relation between data and labels to be linear, the task is to *fit a line* in the data-label space. The line parameters are usually computed by minimizing the square distances between the line and training data. More sophisticated line fitting methods then add robustness to various kinds of possible errors.

However, we cannot expect the real-world computer vision data to always have a linear relation between the data inputs and their labels. In such case of possibly nonlinear data the objective is to *fit a curve* in the data-labels space that would represent their relation. Solving nonlinear regression problems is, however, significantly harder than the linear case and requires the use of sophisticated regression methods.

Linear and nonlinear regression problems are described in Chapter 6 as well as relevant methods of regression learning and their evaluation.

The difference between classification and regression is depicted in Figure 5.6.



(a) Classification[2]

(b) Regression[3]

Figure 5.6: Classification and regression. While (a) tries to separate the data in two classes according to their labels (red and blue), (b) aims to fit a line to the data-label space.

5.7 Embedding-driven Learning

This work is focused on learning from images that are embedded in a multidimensional space. In other words, the input data are images, their multidimensional coordinates represent their labels and the aim is to predict the embedding coordinates for an unseen image. Since we have labels, embedding-driven learning is a form of *supervised learning*.

In general, the coordinates of points in an embedding can be given by vectors of arbitrary real values as defined in Section 3.2. Hence, learning from embedded images is a *regression* problem. Regression problems and the methods of solving them and their evaluation are analyzed in Chapter 6.

An important thing to note is that regression, as defined in Section 5.6.2, predicts *scalar* continuous values. However, the coordinates in a d-dimensional space are represented by *vectors* of d elements. The regression learning system thus requires to be extended to handle arbitrary number of dimensions. The methods of creating such system together with the use of multiple different visual features are proposed in Chapter 7.

[2] Image taken from http://mlpy.sourceforge.net/docs/3.5/lin_class.html.

[3] Image taken from http://gaelvaroquaux.github.io/scikit-learn-tutorial/.

Chapter 6

Regression Analysis

As said in Section 5.7, embedding-driven learning is a *regression* problem since the embedding coordinates are represented by continuous values. This chapter describes the main principles of regression learning, studies the relevant methods of estimating regression problems, and presents possibilities of their evaluation.

6.1 Regression

Regression is a statistical method for estimating the relationships among variables [8]. The importance of regression for the machine learning field is its ability to work with continuous variables and thus complement a common discrete classification. The difference between classification and regression is described in Chapter 5 and visualized in Figure 5.6.

Regression is an approach of modeling the relationship between *data* (one or more variables denoted by $\mathbf{x}$) and *observations* (also *labels*, a scalar variable denoted by y). The data variables are often called *independent variables* (also *regressors*) and the observation variable is usually called *dependent variable* (also *regressand*). The relationship between data and observations is modeled with an *error term* ε (random variable, also called *noise*).

Depending on the properties of the function used for modeling the relation between data and observations we speak of *linear* and *nonlinear* regression. The aim of linear regression methods is to fit a line to the data while nonlinear approaches intend to fit a curve.

6.1.1 Linear Regression

Linear regression assumes that the relationship between data and observations is modeled by a linear function. For n observations and p data variables the linear regression can be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{6.1}$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} \mathbf{x}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{x}_n^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix} \quad \text{and} \quad \mathbf{x}_n = \begin{pmatrix} \mathbf{x}_{i1} \\ \vdots \\ \mathbf{x}_{ip} \end{pmatrix}. \tag{6.2}$$

The $\mathbf{y}$, $\mathbf{X}$ and $\boldsymbol{\varepsilon}$ are vector variants of y, $\mathbf{x}$ and ε described in Section 6.1 for n observations. The relation between data and observations is represented by $\boldsymbol{\beta}$, a vector of *regression coefficients* (also called *parameter vector*).

When solving a regression problem, the main interest is to correctly estimate the $\boldsymbol{\beta}$ vector of regression coefficients. Predicting outputs for testing data is then done by a dot product of $\mathbf{X}$ and $\boldsymbol{\beta}$.

In other words the main objective of solving linear regression is to fit a line into a data-label space by correctly estimating its parameters. Different methods of linear regression have different capabilities of robustness to noise and other problems and some of the most important ones are described in Section 6.2.

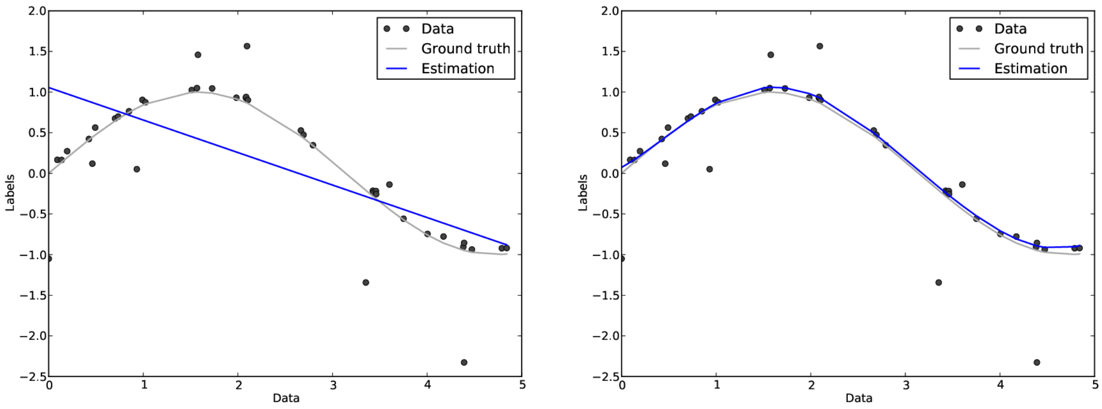
6.1.2 Nonlinear Regression

Nonlinear regression is used when the relationship between data and observations is not linear, i.e. when the observations cannot be expressed as a linear combination of the data. Instead of fitting a line to the data-label space nonlinear regression techniques intend to represent the hidden function by an arbitrary curve.

Solving nonlinear regression problems is generally a much more difficult task than estimating parameters of linear functions. The techniques of learning nonlinear regression are based on solving linear cases in a very high-dimensional space by the use of so called *kernel functions*. The principles of using kernel functions are described in Section 6.3 and a very robust method of solving both linear and nonlinear regression problems is presented in Section 6.4.

Nonlinear regression is a very powerful way of function estimation especially while working with complex data such as human-annotated relative similarities captured by multidimensional embedding. While linear classification may sometimes do a good job in separating classes of complex shapes, linear regression on nonlinear data will always result in a loss of prediction accuracy making the task of solving nonlinear regression problems essential for some kinds of data.

Figure 6.1 depicts the difference between linear and nonlinear regression and shows that for nonlinear data the linear estimation can produce large errors while the nonlinear methods can work much better.



(a) Linear regression

(b) Nonlinear regression

Figure 6.1: Linear and nonlinear regression. The function estimations were computed using SVR with a linear (a) and nonlinear RBF (b) kernels (see Section 6.4) using noisy data samples as displayed in the figures.

6.2 Basic Methods of Estimating Regression Problems

This section presents some basic methods of solving linear regression problems and outlines the techniques of dealing with noise and other defects in the input data. The main objective is to describe some essential principles of solving regression problems that are necessary to understand more complex and robust methods presented in sections 6.3 and 6.4.

6.2.1 Ordinary Least Squares Regression

Ordinary least squares (OLS) is the simplest and most straightforward method of solving regression problems with linear relationship between data and observations. The idea is to fit a line to the observed data that would minimize the residual sum of square distances between itself and the observations. This is done by finding an estimate $\hat{\boldsymbol{\beta}}$ of the parameter vector $\boldsymbol{\beta}$. Mathematically, OLS solves a minimization problem of form

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2, \tag{6.3}$$

where $\|\cdot\|_2$ denotes the Euclidean distance.

This method is, however, very sensible to the character of the data and under some circumstances can produce large errors. The first problem are *outliers* – points in the training set that are very far from the rest of the data. The problem is that OLS tries to minimize the sum of the squared distances from the data but such points will have a strong influence on that sum.

The OLS also assumes that there is no *multicollinearity* in the input data. This means that there must be no linear dependence between the columns of the data matrix $\mathbf{X}$, otherwise the method can become highly sensitive to random errors.

6.2.2 Ridge Regression

Ridge regression (also known as Tikhonov regularization) faces the multicollinearity problem by introducing a penalty on the size of the coefficients. This technique is called *regularization*, in statistics also known as *shrinkage*. The penalty is defined as a square of the Euclidean norm of the coefficients controlled by a small constant $\alpha \geq 0$ called *complexity parameter*. The minimization problem from Equation 6.3 then takes the following form:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_2^2. \tag{6.4}$$

The robustness to multicollinearity depends on the value of the complexity parameter α. The higher its value the more the the parameters are shrunk, resulting in higher robustness to multicollinearity.

6.2.3 Other Modifications of Least Squares Regression

Many other methods improve the robustness of the least squares regression by introducing different parameters and normalizers. One of them, called LASSO (standing for Least Absolute Shrinkage and Selection Operator), replaces the Euclidean penalty in Equation 6.4 with a ℓ_1 norm. The main advantage is its capability of producing zero parameters and thus generating sparse solutions making LASSO also a parameter *selection* method.

Another modification, called *Elastic Net*, combines the penalties of Ridge and LASSO regression using both the ℓ_1 and ℓ_2 norms [32]. This way it is capable of learning sparse solutions like LASSO with the regularization properties of ridge regression.

6.3 Kernel Ridge Regression

Ridge regression is probably the simplest algorithm that can be extended to solve nonlinear regression problems. The idea of its nonlinear extension was first developed in 1998 [22] and then was extended and named Kernel Ridge Regression (KRR) [5, 28].

KRR maps the input data to a high-dimensional feature space and performs linear regression on those features. This results in constructing nonlinear regression function on the original data. However, the feature space representation can result in a very large increase of dimensionality and thus produce long feature vectors with large number of parameters. This can cause the problem to be extremely hard or impossible to compute.

To solve the problem of a very high dimensionality of the feature vectors KRR implements *kernel functions*. Kernel functions represent dot products in the feature space which allows to perform computations in that space without having to actually build it. In other words, we do not have to compute all the feature vectors but we can still compute their dot products by applying the kernel functions.

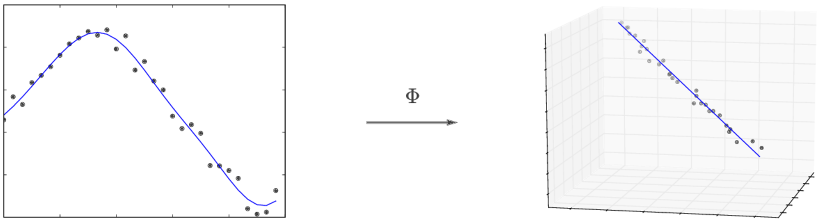
To construct linear regression in the feature space we need to choose a mapping Φ from the original observations space $\mathcal{X}$ to the high-dimensional feature space $\mathcal{F}$:

$$\Phi : \mathcal{X} \to \mathcal{F}. \tag{6.5}$$

However, instead of choosing the exact Φ we can choose only a kernel function $\mathcal{K}$ as long as we know that

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \qquad \forall \mathbf{x}_{i,j} \in \mathcal{X}, \tag{6.6}$$

where $\cdot$ denotes dot product. In other words, we must choose a kernel function that, taking two observations as inputs, performs dot product of their images in the feature space. The concept of kernel function is depicted in Figure 6.2.



(a) Input space (b) Feature space

Figure 6.2: The principle of kernel regression. The inputs are mapped to a high-dimensional feature space where a linear regression is evaluated resulting in non-linear function estimation in the input space. The figure is only schematic and does not show real data.

The kernel functions can be defined in many ways and more information on the techniques of their derivation as well as further technical details of the KRR method can be found in [22, 28]. It worths noting that KRR can be also defined as a special case of the SVR method, that is described in Section 6.4, and ridge regression, as defined in Section 6.2.2, is a special case of KRR when choosing dot product as a kernel function. Moreover, the OLS method, as defined in Section 6.2.1, is a special case of ridge regression (when setting $\alpha = 0$ in definition 6.4), which allows KRR to compute also a kernel version of OLS.

The main disadvantage of the KRR method is that it does not produce sparse parameter vectors, i.e. parameter vectors with many zeros, which means all of the parameters are always used resulting in poor computational performance.

6.4 Support Vector Regression

Support vector regression (SVR) is a complex and robust regression method based on the support vector machines (SVM) algorithm that is widely used also for classification. The SVM implement a learning algorithm for recognizing patterns in complex datasets and generalizing the knowledge to unseen data. It was largely developed in the 1990s at AT&T Bell Laboratories as a nonlinear generalization of so called *generalized portrait algorithm* that was introduced in the 1960s [23].

Similarly to the case of KRR (see Section 6.3) the SVM algorithm maps the observed data to a high-dimensional feature space (see Equation 6.5) and instead of costly computing the space itself it only performs a dot product of the feature vectors by the use of *kernel functions* directly on the input data (see Equation 6.6).

Depending on the used kernel function the SVM can construct a nonlinear function in the input data by constructing a linear function in the feature space. In case of classification a linear separation of the feature space can produce a nonlinear separation of the input data, whereas for regression a linear function fitted to the feature space can result in a nonlinear function fitted to the data.

6.4.1 ε-SVR

The SVR algorithm was first introduced by Vapnik in 1995 [27]. His method is usually denoted by ε-SVR because its goal is to find a function that has at most ε_{SVR} deviation from all the training data and at the same time is as *flat* as possible. This means the method does not care about errors smaller than ε_{SVR} but will not accept any error that is larger. The value of ε_{SVR} thus bounds an area around the fitted curve inside which most of the data samples should ideally lie. Such area is called ε-*tube*.

Taking a linear case at first and going back to the Equation 6.1 that defines linear regression as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, the requirement of flat solution means that we are trying to find small values of $\boldsymbol{\beta}$. This can be achieved by minimizing its norm and Vapnik formulated the problem as follows:

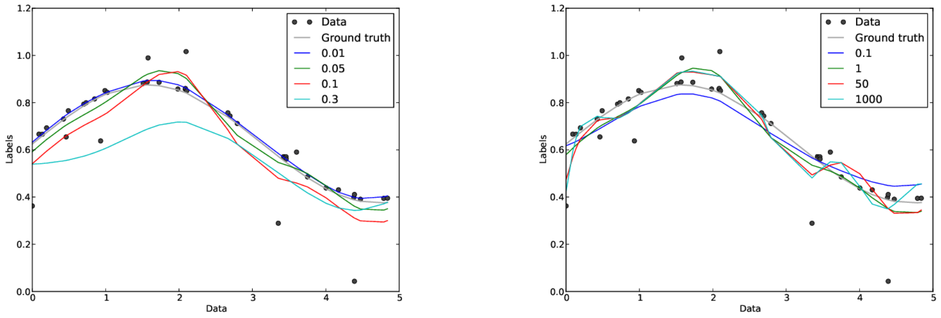
$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & y_i - \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} - \varepsilon_i \leq \varepsilon_{SVR} \\ & -y_i + \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_i \leq \varepsilon_{SVR} \end{aligned} \tag{6.7}$$

In other words, we are trying to find small values of $\boldsymbol{\beta}$ that would make all the data samples lie inside the ε-tube. However, such formulation works only if the minimization problem is feasible and the data samples are not damaged. To allow for some errors and noise the formulation was extended with slack variables s_i^+ and s_i^- and their sum is also a subject to the minimization process:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + C\sum_{i=1}^{n}(s_i^+ + s_i^-) \\ \text{subject to} \quad & y_i - \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} - \varepsilon_i \leq \varepsilon_{SVR} + s_i^+ \\ & -y_i + \mathbf{x}_i^{\mathrm{T}}\boldsymbol{\beta} + \varepsilon_i \leq \varepsilon_{SVR} + s_i^- \\ & s_i^+, s_i^- \geq 0 \end{aligned} \tag{6.8}$$

The constant $C > 0$ represents a trade-off parameter between the flatness of the estimated function and the tolerance of errors larger than ε_{SVR}.

The choice of values of both ε_{SVR} and C parameters can have a significant impact on the quality of the resulting model. Higher values of ε_{SVR} produce flatter solutions and may cause underfitting if set to very high values while very low values can become sensible to errors and overfit. Similarly, low values of C allow for larger errors and may tend to underfit while high values may become error-sensible and cause overfitting. The effects different values of both of the parameters are shown in Figure 6.3.



(a) ε_{SVR}

(b) C

Figure 6.3: Different values of parameters ε_{SVR} and C. Parameter ε_{SVR} controls the flatness of the resulting function whereas C affects the sensibility to errors. Both tend to underfit while using low values and overfit when set to high ones. In (a) the value of C was fixed to 1.0 while in (b) the ε_{SVR} was set to 0.06.

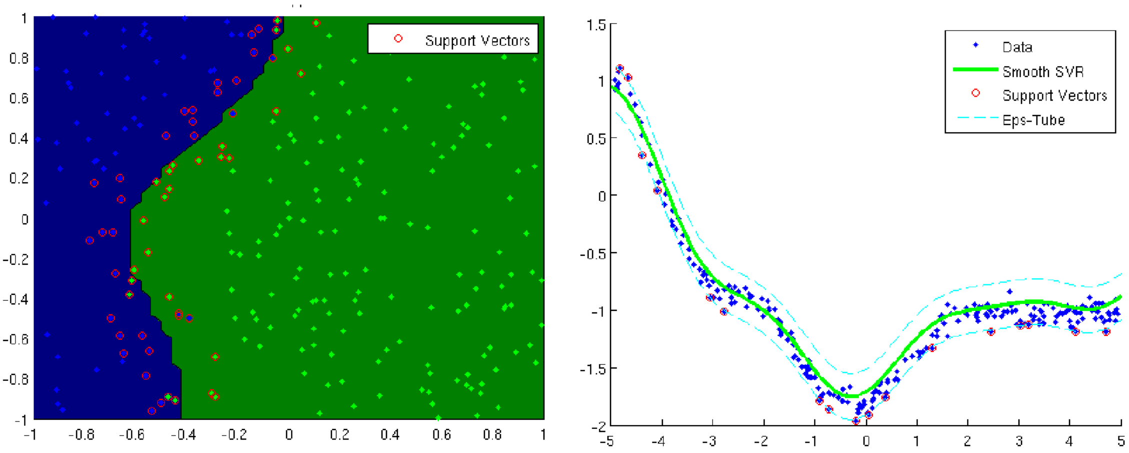
Choosing the right values of ε_{SVR} and C and avoiding problems of overfitting and underfitting is not an easy task. One way to approach the parameter tuning is to evaluate the performance for different parameter values using the technique of cross validation as described in Section 5.4

6.4.2 Support Vectors

In case of classification the SVM algorithm separates two classes of training data in the feature space by a hyperplane with a maximum margin, i.e. in a way that the area without data points around the hyperplane is the largest possible. However, only some of the training points that are close to the hyperplane are necessary to define its position. In other words, only some points will change the parameters of the hyperplane if we remove them or change their position.

When computing regression with the ε-SVR algorithm we aim to fit a line to the data in a high-dimensional feature space. All data points that lie inside the ε-tube of such line are ignored (see Section 6.4.1). This means that removing points lying inside the ε-tube or changing their position in the tube does not influence the parameters of the fitted line. Only the points that lie outside the ε-tube are thus important for the computed parameters.

Such points from the training set that are important for the shape of the hyperplane (SVC) or line (SVR) in the feature space are called *support vectors* and their positions in the input data are visualized in Figure 6.4. The main advantage of support vectors is that they produce sparse resulting parameter vector, i.e. many parameters have zero values, and thus significantly reduce the computational complexity because it is not necessary to compute dot products with all of the training points.



(a) SVC support vectors

(b) SVR support vectors

Figure 6.4: Support vectors displayed in the input data for SVC and SVR. While the SVC support vectors lie close to the border between two classes (a), the SVR support vectors are the points outside the ε-tube (b).[1]

6.4.3 Kernel Functions

As stated in Equation 6.6, a kernel function can be defined in any way, as long as it corresponds to a dot product in the feature space. Hence, various kernel functions with different properties can be used with SVR. This section briefly lists the most common kernel functions used with SVMs and describes their most important properties [9].

Linear kernel. Linear kernel is probably the simplest kernel function, defined as the dot product itself:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j, \tag{6.9}$$

where $\cdot$ denotes dot product. Learning methods using linear kernel are usually equivalent to their non-kernel variants and linear kernel thus does not help in solving nonlinear problems. This makes linear kernel useful only for linear data or to compare the performance of learning linear and nonlinear function estimations.

Polynomial kernel. Polynomial kernel represents data samples in the feature space over polynomials of the original variables. In other words, it uses not only the input vectors to measure their similarity but also combinations of them. For polynomials of degree d the kernel function can be defined as:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d, \tag{6.10}$$

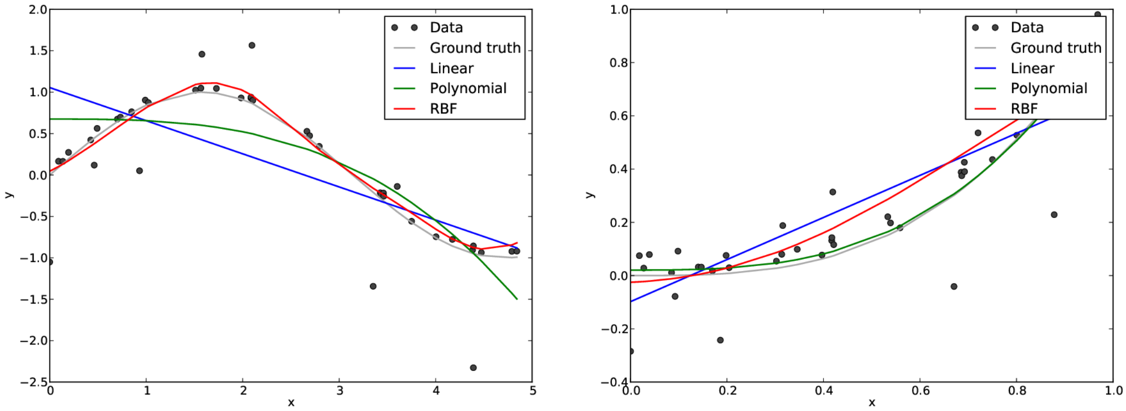
where c is a constant term that impacts the importance of higher and lower orders of the polynomial. The polynomial kernel is often used for some tasks in SVM classification. Its main problems are its tendency to overfit for larger values of d and a numerical instability in some special cases.

[1] Image taken from http://www.di.ens.fr/~mschmidt/Software/minFunc/minFunc.html.

Radial basis function kernel (RBF). A Gaussian radial basis function is probably the most popular kernel in SVM classification and regression. It can be defined as:

$$\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}}, \tag{6.11}$$

where σ is a parameter that impacts its ability of learning from the data samples. Excessively high values of σ may cause underfitting and almost a linear behavior while lower values may tend to overfit. The RBF kernel is suitable for a large range of machine learning tasks and its versatility makes it the default kernel function in some SVM implementations.



(a) $y = \sin x$

(b) $y = x^3$

Figure 6.5: Different kernel functions used with SVR. In (a) RBF gives the best results whereas in (b) it is outperformed by the polynomial kernel as the character of the ground truth function is also polynomial.

Although the RBF kernel might be suitable for most of the computer vision and machine learning tasks, other kernels may perform better in some specific cases. It is therefore possible to include the selection of the kernel function in the parameter tuning step. Examples of using the described kernel functions are depicted in Figure 6.5.

6.5 Evaluating Regression Models

When a regression model is learned we need to evaluate its performance using some numeric measures. This section focuses on error measurements that can be incorporated in a common machine learning pipeline using training and testing data and cross validation techniques (see sections 5.3 and 5.4).

6.5.1 Mean Absolute Error (MAE)

The MAE measure simply computes the mean of absolute difference between all predicted values and their ground truth. For n the MAE measure is defined as:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |f_i - y_i|, \tag{6.12}$$

where y_i denotes i-th predicted output and f_i represents the corresponding value of the true function.

MAE averages the error magnitudes with the same weights across all values of n, i.e. each error participates to the average by its real magnitude. Such error measure is particularly meaningful when we work with continuous variable that can be affected with noise.

6.5.2 Mean Squared Error (MSE)

MSE computes the mean of squared differences between all predicted values and their ground truth:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(f_i - y_i)^2, \tag{6.13}$$

where y_i denotes i-th predicted output and f_i represents the corresponding value of the true function.

This way MSE assigns weights to the errors, accentuating the large values and reducing the small ones. The MSE measure is thus much stricter than MAE and it is suitable in cases where we want to completely avoid large errors.

Chapter 7

System Design

This chapter proposes a design and an architecture of an image-based embedding-driven learning system. It connects the theory studied in chapters 3, 4 and 5 and analyzes a few simple approaches of improving the prediction performance by using the information from multiple visual features at once.

7.1 Basic Architecture

The basic architecture of the system is based on the classical supervised learning pipeline described in Section 5.5. The training data labels are represented by an embedding (see Chapter 3), in particular each image from the training set has its coordinates in an embedding. The learning algorithm used to learn the relations among the testing images and their embedding coordinates is Support Vector Regression (SVR, see Chapter 6).

Since the SVR algorithm outputs a single real number, the embedding is decomposed to its single dimensions and each of them is learned separately. The SVR algorithm is thus extended to a multi-model version that can output vectors of arbitrary length. The basic system architecture works with a single chosen visual feature and it is shown in Figure 7.1.

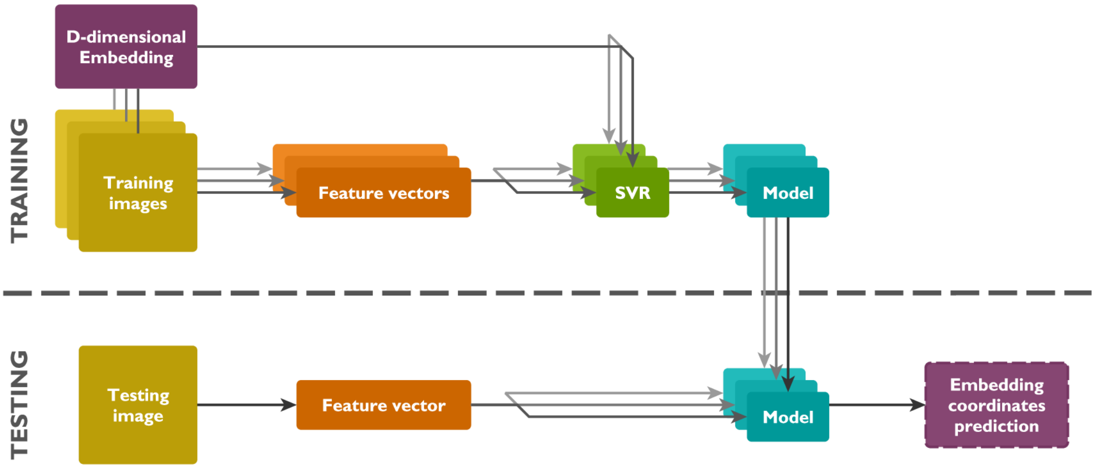


Figure 7.1: The basic system architecture. A single feature vector is extracted from each of the training images and an SVR is used to learn a model for each of the dimensions. In the test time one feature vector is used to predict each of the dimensions of the coordinates.

7.2 Feature Selection

The simplest way of using multiple visual features to improve the prediction accuracy is choosing the best one for each of the dimensions of the embedding.

Multiple different feature vectors are extracted from each of the training images and an SVR learning is computed for each of the visual features and each of the dimensions of the embedding. The best performing visual feature is then chosen for each of the dimensions and the resulting multidimensional model is then composed of multiple simple one-dimensional models while each of them can be fitted on different visual features.

This method can significantly improve the prediction accuracy since different dimensions of an embedding can represent different attributes of the objects and those can be in some cases captured by different visual features. For instance a 2-dimensional embedding can capture color in one of the dimensions and shape in the other one. Although more complex embeddings cannot be decomposed to a simple image attributes like color and shape, different visual features can still show significant differences in performance for estimating the position in a particular dimension.

The design of a per-dimension feature selection pipeline is visualized in Figure 7.2.

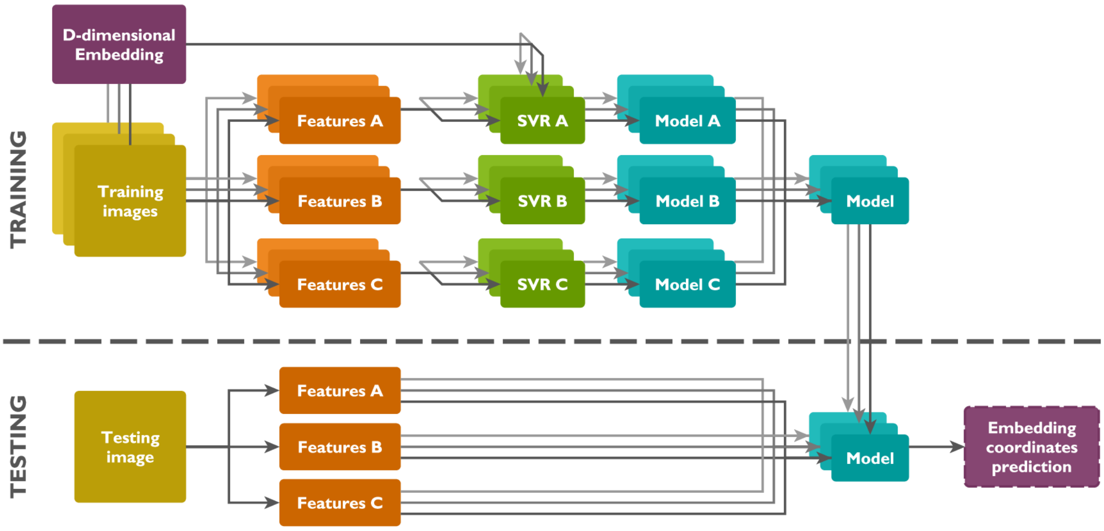


Figure 7.2: Feature selection. Multiple visual features are extracted from each image in the training step and a model is learned and evaluated for each of them. The best model is then selected for each of the dimensions. During testing only the best features need to be extracted and passed to the learned models.

Feature selection is relatively computationally expensive in the training phase since a multidimensional SVR learning has to be computed for each of the extracted visual features. This can make the learning process time-consuming, especially on a larger dataset or while working with images of large proportions.

During the testing phase the computational complexity does not have to be significantly worse than for the single-feature case, since one prediction per dimension has to be done in both cases. However, using the feature selection can make the feature extraction process d-times more complex for a d-dimensional embedding since at most d-times more feature vectors may be extracted.

7.3 Feature Concatenation

Another simple method of using the information from multiple extracted visual features is their concatenation. The idea seems meaningful – since the extraction of every different visual feature produces a feature vector that carries different information, all the feature vectors can be concatenated to a single long feature vector to capture more information.

However, doing so might be risky for various reasons. Several problems may occur depending on the used learning method. At first, different feature vectors may contain completely different absolute values and some learning algorithms might not work well with the resulting concatenated feature vectors. Such problems can, however, be avoided by normalizing the values of feature vectors from the training set and then applying the same scale to the testing features. Usually the training features are normalized to the range from 0 to 1.

Some learning algorithms may also be sensible to the length of the feature vectors. Depending on the number of used visual features the concatenated feature vectors may be extremely long and some learning methods may not work well if the feature vectors are many times longer than the size of the dataset. However, the SVR algorithm should be robust to such situations. The resulting feature vector can also be composed of simple features that vary a lot in length which might be another source of errors in some cases.

The principle of feature concatenation is depicted in Figure 7.3.

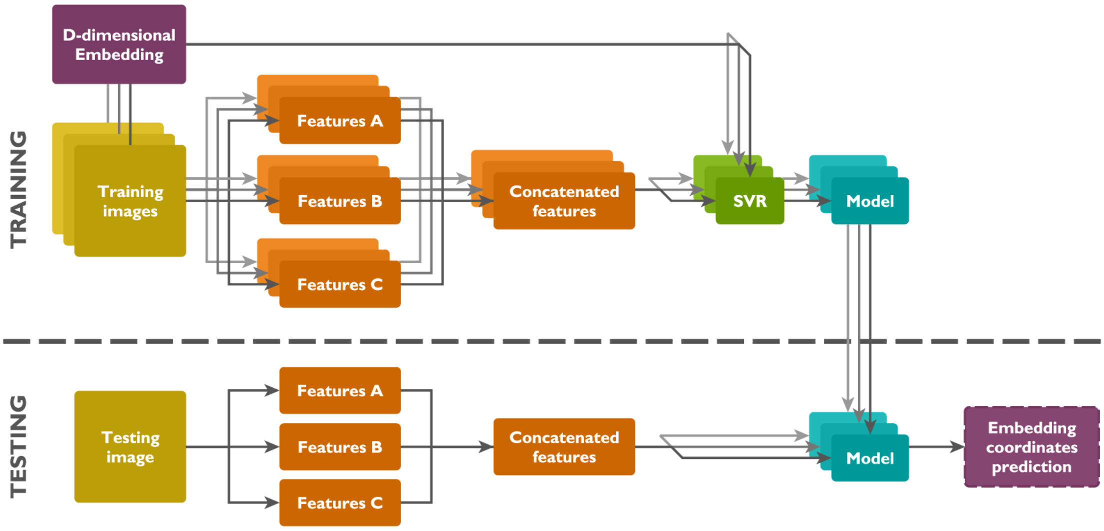


Figure 7.3: Feature concatenation. Multiple feature vectors are extracted from each of the images and concatenated to a single high-dimensional feature vector in both training and testing steps. Careful normalization of the feature values is necessary to avoid errors.

When dealing carefully with all the possible problems, the feature concatenation can give good results with a low computational complexity. In both training and testing phases all the chosen visual features are extracted, concatenated to a single vector and then used the same way as for a single visual feature. The learning process thus runs only once for each of the dimensions and the prediction in testing phase is done using a single feature vector. However, the process may be much slower then using a single visual feature because the concatenated vectors can be much longer.

7.4 Feature Weighting

A more complex way of using multiple visual features is assigning them some kind of weights controlling how much each of them contributes to the result. The weights can be either chosen statically or defined as a function that can be learned and modeled. This section presents a solution that learns a weighting function using the SVR algorithm on top of predictions computed from all the used visual features.

Initially, all the different visual features are extracted from each of the training images. Then a complete multidimensional model is learned for each of the the features, similarly as described in Section 7.2. Then, all the models are used to compute their predictions on the whole training set. For each dimension of the embedding we thus have a set of different predictions and we want to figure out how to combine them to get the best value.

All the predictions for a particular dimension are then used as an input of another SVR learning driven by the embedding. This step thus learns a model of a function that is capable of weighting different predictions of the same value to get the best result. Testing phase requires extracting all the used visual features from the input image, predicting embedding coordinates for each of them, and then estimating the best coordinates from all of the predictions.

The described principles of feature weighting are portrayed in figure 7.4.

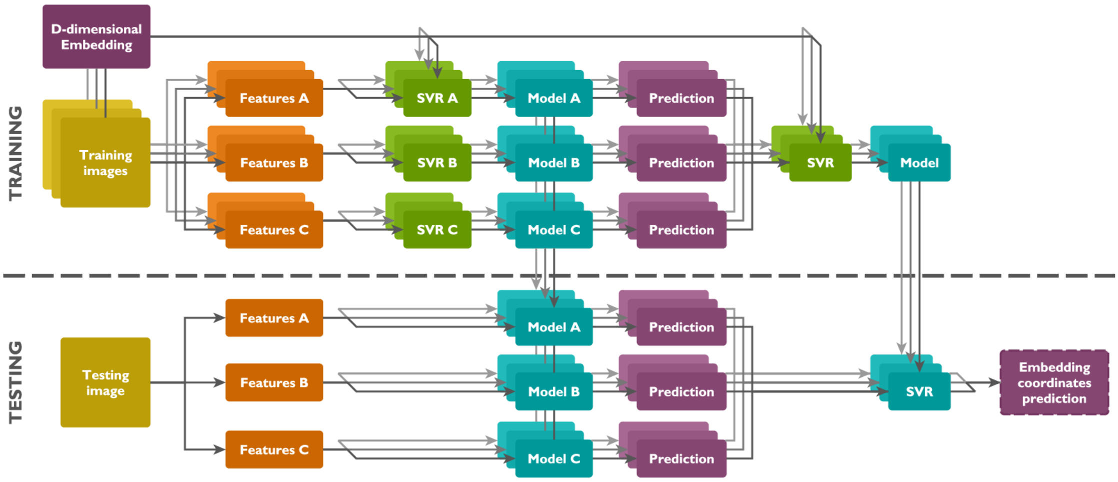


Figure 7.4: Feature weighting. Multiple visual features are extracted from each of the images and a model is learned for each feature type. Predictions on the training set are then computed with all of the models and their values are used to learn another model that would be able of weighting them. During testing all the predictions are computed and weighted afterwards. Although this method is computationally expensive it is a robust solution that should improve the prediction accuracy.

Feature weighting using an SVR is a robust method of combining multiple different visual features but it is also the most computationally expensive one since many models are learned in the training phase and many predictions have to be computed during testing. However, if the prediction accuracy is important, the described method of feature weighting might be a suitable solution.

7.5 Prediction Accuracy Evaluation

Having an image-based embedding-driven learning system we need a way of evaluating its performance. Two measures of regression accuracy were described in Section 6.5 – MAE and MSE. Although these measures can be used to evaluate the model from the training set itself, we are primarily interested in the ability of the regression algorithm to generalize on unseen data samples. Therefore, all error measures will be evaluated in a common machine learning way – i.e. on a testing set that was not used for training.

Mean absolute error (MAE). Since the regression models are learned per-dimension we need a measure to evaluate the performance in each of them. Such measure should tell us how good the predictions are on average. MSE is, however, mainly concerned with penalizing very large errors and a even a small number of large error values will significantly impact its result. Therefore, the simple principle of MAE appears to be much more suitable for an embedding driven learning since it gives us a better idea about the average prediction quality. The definition of MAE is stated in Equation 6.12.

Mean Euclidean error (MEE). The SVR algorithm is sensible to the size of the data samples and thus requires a normalization of its inputs. Each dimension of the embedding is therefore normalized and the predictions are scaled back to the original embedding size in the testing phase. Once the predictions for all of the dimensions are transformed to the original embedding we can compute the Euclidean distance between the prediction and the original point. The mean of such distances can give us an overall prediction quality measure that will be further denoted as mean Euclidean error (MEE). For n embedded objects it can be defined as

$$\text{MEE} = \frac{1}{n}\sum_{i=1}^{n}\|\hat{f}(x_i) - f(x_i)\|_2, \tag{7.1}$$

where $\hat{f}(x_i)$ denotes the predicted position of object x_i and $f(x_i)$ its true coordinates.

The prediction performance of the proposed system thus can be evaluated with the MAE measure on the per-dimension basis and using both MAE and MEE to obtain the overall error values.

When we have no previous results to compare with, it is desirable to evaluate how well the system performs compared to a *random* behavior. In classification such randomness corresponds to a system that for each input picks a predicted class at random. To capture such measures in context of embedding-driven learning, two random models were introduced.

Random Value Model (RVM). RVM is a model that for each input from the testing set predicts the embedding coordinates as random continuous values limited by the minimal and maximal values in each dimension. RVM thus produces completely random predictions that are limited only by the size of the embedding.

Random Neighbor Model (RNM). RNM is a model that for each input used during testing picks a random sample from the training set and uses its embedding coordinates as a prediction. When the training samples are embedded evenly across all the embedding space, RNM will act similarly to RVM. However, if the original embedding contains dense clusters where majority of points is concentrated, the RNM measure will give better results than RVM since the chances of getting close to the true position are much higher. Therefore, RNM depends on the embedding's inner structure while RVM is structure-independent.

Chapter 8

Implementation

This chapter describes a library that was developed to simplify the implementation of an image-based embedding-driven learning system. The library wraps various existing computational tools, brings additional functionality, and provides handy programming interfaces. All the experiments presented in further chapters were performed using the developed tool.

8.1 Choice of Programming Environment

Implementing a tool for image-based embedding-driven learning have vast requirements in terms of data processing and learning algorithms. We need to work with multidimensional embeddings (see Chapter 3), extract various visual features from images (see Chapter 4), transform and normalize them, work with large sets of data, apply regression methods to learn from the data (see Chapter 6) and perform experiments to evaluate the results.

All the mentioned tasks require an extensive use of numeric methods, matrix computations, scientific libraries with regression solvers, image processing functions, and an interface for data visualization.

8.1.1 Matlab

Based on the requirements the first and probably most intuitive choice might be the Matlab[1] environment since it implements a part of the desired functionality out-of-the-box and many additional libraries exist. Various numeric methods, matrix operations, and data visualization algorithms are implemented in its core and complex toolkits for extracting visual features and solving regression problems exist.

However, Matblab has also some disadvantages that play an important role in software development:

- The first significant limitation of Matlab is its *license*. Matlab is not free and its price and closed license may make it harder to obtain. For the same reasons Matlab may not be an ideal choice for future development since all the developed functionality would be dependent on its licensed environment.

- Another problem of Matlab is its *programming language* which might be suitable for fast prototyping and small scripting but proves to be very inflexible for development of a larger system. While some mathematical expressions have a convenient syntax, classical programming constructs are heavy-footed and the more advanced ones do not exist at all.

[1] http://www.mathworks.com/products/matlab/

- Third point against Matlab is that whole its environment is extremely *heavyweight* requiring a few gigabytes of space to install and making the environment slow on less powerful machines. Although some functions can be written in pure C for performance reasons, their compilation and integration to Matlab is relatively complicated and can make the development harder.

8.1.2 Python

Probably the only alternative environment that provides similar computation base as Matlab along with advanced programming paradigms is Python[2] with a set of open-source scientific libraries. In the last years Python has grown a lot in the scientific community, many mathematical libraries were released and some projects try to completely replace the Matlab environment.

Python is not affected with any of the main Matlab disadvantages:

- Python itself and all of the most important scientific libraries are completely open-source and thus do not limit the area of usage in any way.
- Python itself is a very flexible and dynamic programming language that also provides advanced programming paradigms. Its syntax is very intuitive and mathematical expressions are defined in a similar way to Matlab.
- Python together with some scientific libraries is still a much more lightweight solution when comparing to Matlab and its execution can be in some cases faster. Many complex algorithms are for higher performance also implemented in C.

The Python's main disadvantage is that most of the computer vision and machine learning algorithms were originally developed with Matlab and Python thus have a smaller library base. However, in the last years many algorithms were ported to Python and most of the necessary functionality for image-based embedding-driven learning can be found in the existing libraries.

For the described reasons Python appeared to be the best choice, gaining free lightweight solution with programming flexibility on one hand and the possibly of having to implement some non-existing functionality on the other. The libraries and packages used for the implementation are described in Section 8.2.

8.2 Used Technologies

Python itself does not offer any scientific computational capabilities since it is a universal all-purpose programming language. This section describes the most important libraries and packages that were used to cover the scientific computations and other useful functionality.

Numpy. Numpy[3] is the base of all numeric and scientific computations in Python. It provides fast matrix operations using a convenient Matlab-like syntax, linear algebra computations and other useful functions. Numpy is also the base of most of the scientific libraries in Python.

In this work Numpy was largely used to develop dataset building tools, embedding operations, visual feature extraction and others, including scripts for all the experiments.

[2] https://www.python.org/

[3] http://www.numpy.org/

Scipy. Scipy[4] is a fundamental open-source library for scientific computing in Python built on top of the Numpy package.

Scipy itself was used for computing basic mathematical functions, KD-trees for nearest neighbor search and served as a base for other libraries.

Scikit-learn. Scikit-learn[5] (also called *sklearn*) is a large set of tools for machine learning built on top of Numpy and Scipy and offering many methods of classification, regression, clustering, data normalization, etc.

Scikit-learn was used for data normalization, regression learning, implementation of Bag of Visual Words with K-Means clustering and other tasks.

Scikit-image. Scikit-image[6] (also called *skimage*) is an image processing library built on top of Scipy and Numpy. It implements many image processing algorithms such as transformations, filtering, segmentation, restoration, and others, and also some of the feature extraction algorithms.

Scikit-image served especially for computation of HOG and LBP features and was also used for some data visualizations.

Matplotlib. Matplotlib[7] is a very complex Python plotting library capable of producing a large variety of high quality figures.

Matplotlib was used for all the data visualizations.

VLFeat. VLFeat[8] is a visual feature extraction library written in C. It has official bindings to Matlab and some of the feature extraction algorithms are released as command line tools.

VLFeat was used to extract SIFT features from the images. Although some incomplete unofficial Python bindings exist, they are unstable on some platforms and thus the command line version of SIFT was wrapped with Python code. VLFeat was chosen over OpenCV[9] because its SIFT implementation gives much better results.

8.3 Implemented Functionality

This section describes the main functionality that was developed in order to implement image-based embedding-driven learning. The implemented tool is further in the text referred to as EVFL standing for Embedding-driven Visual Feature Extraction and Learning.

8.3.1 Datasets

The base of the EVFL functionality lies in a module for dataset management and operations. Handling the dataset operations in a convenient is important for the implementation for any larger experiments. The module offers an easy way of dataset creation from a directory with images and its annotating with an embedding. On such dataset we can then perform the following operations:

[4] http://www.scipy.org/

[5] http://scikit-learn.org/

[6] http://scikit-image.org/

[7] http://matplotlib.org/

[8] http://www.vlfeat.org/

[9] http://opencv.org/

- The dataset can be *shuffled*, i.e. the images are randomly reordered preserving their correct annotations and other information. The shuffling function also allows for usage of an explicit seed to get the same shuffled order on its each run. Dataset shuffling is useful for its further splitting to training and testing sets, etc.

- Another important operation is creating a *subset* of the original dataset. An arbitrary number of subsets can be defined by specifying the ranges of items for each of them. All the embedding annotation and other data are correctly preserved. A subset operation returns another instances of dataset.

- Especially for the purposes of k-fold cross-validation the dataset module also offers an operation for *splitting* the dataset to a given number of evenly sized parts. The operation is equal to creating k subsets but all the subset ranges are computed automatically.

- When working with subsets we might want to *join* some of them together to form a larger dataset. A functionality to join an arbitrary number of datasets has been implemented as well as their joining by a common addition which was achieved by overloading of the addition operator.

To lower the memory usage the dataset stores only the paths of the images and loads their content dynamically when required. This functionality lies in the image handling module which is described in the next subsection.

8.3.2 Images

Another important module of the EVFL toolkit provides an interface for working with images. An image can be instanced from its path and instead of being loaded immediately the module stores just the path and loads its content when it is necessary. This way we can avoid storing all the dataset in the memory at the same time which could cause memory errors. The module offers the following functionality:

- An easy *access to any of the color channels* of the image. The image data is loaded lazily when it is necessary and the requested channels are returned in form of Numpy arrays to be easily used in feature extraction algorithms.

- The image module wraps up all the *feature extraction* algorithms that are described in Section 8.3.3 and allows for their automatic caching using the cache system described in Section 8.3.6. The cache of the features automatically invalidates on any parameter or image change to always provide the most actual results.

- The image module also provides methods for a *quick plot* of the image in different color scales.

8.3.3 Feature Extraction

The feature extraction module exposes a unified programming interface for different visual feature extraction algorithms. It also adds some of their missing functionality and implements various ways of their visualization. The module covers extraction of the following features (see Chapter 4):

- Global and local *color histograms* that were implemented with the use of Numpy functions and allow for setting of all the important parameters as well as histogram normalization.

- The extraction of HOG features is built as a wrapper over the implementation from Scickit-image. The function is extended to allow for generating non-flattened raw HOG values that could be used in a Bag of Visual Words.

- The LBP features are also computed with the implementation from Scikit-image which is extended to compute a histogram of the features and automatically estimate its number of bins.

- Extraction of SIFT features is based on a command line version of the VLFeat's SIFT implementation. As explained in Section 8.2, this way of implementation was chosen due to the non-existence of a complete VLFeat Python wrapper and bad quality of the SIFT implementation in OpenCV.

The feature extraction module also provides an implementation of the Bag of Visual Words feature representation and offers various functions for visualization of the extracted features.

8.3.4 Regression

All the important regression learning capabilities are provided by a regression module. The module implements a generic regression learning wrapper that is capable of working with multidimensional embedding and supports models with multiple dimensions. Various error scores can be computed over all the embedding dimensions and evaluated.

The algorithms that implement the regression learning itself are all part of the Scikit-learn package. Along with SVR other simple linear regression methods are included for demonstration purposes. The regression module adds the possibility of storing a learned multidimensional model in a file on a hard drive and then loading it back again.

8.3.5 Caching

The developed EVFL toolset provides also a caching module to speed up some of the computations. The cache is implemented as a key-value storage with a universal interface that can be extended to different caching backends. The default implemented backend uses binary files to store the cached data on a hard drive.

To separate large sets of values in the storage the cache provides a *namespace* functionality that helps to organize the cached data. The cache keys can be represented by almost any basic data type such as strings, numbers, dictionaries or lists and are automatically hashed to a unique string. The module also offers a Python decorator to cache results of whole functions using their parameters as the caching keys.

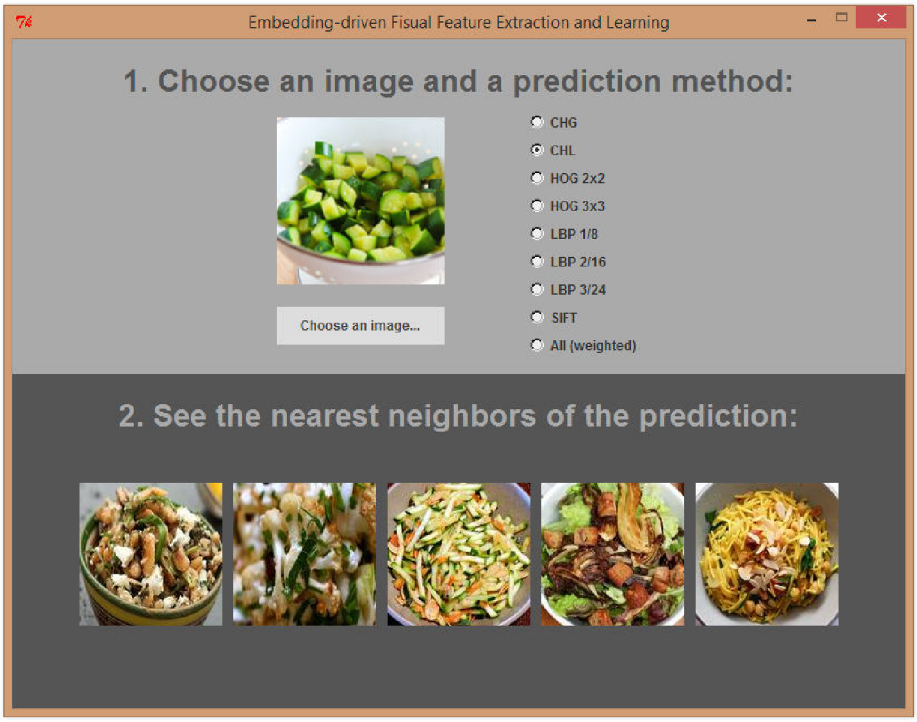
Internally the caching system is used in the feature extraction module to avoid extracting the same visual features on every run of the algorithm.

8.3.6 Tools

The last module of the EVFL package serves as a storage for any other useful algorithms and functions. Currently it implements an embedding scaler that normalizes the embedding size before the training step and resizes it back after prediction.

8.4 Demo Application

In order to demonstrate the functionality of the developed system a simple application with a graphical user interface was created. The application allows the user to insert an image and using a selected method it predicts and displays five of its most similar images from a dataset of food images described in further chapters. The interface of the application can be seen in Figure 8.1.



Embedding-driven Fisual Feature Extraction and Learning

1. Choose an image and a prediction method:

○ CHG
● CHL
○ HOG 2x2
○ HOG 3x3
○ LBP 1/8
○ LBP 2/16
○ LBP 3/24
○ SIFT
○ All (weighted)

2. See the nearest neighbors of the prediction:

Figure 8.1: The interface of the demo application.

Internally, the application contains a set of food images together with an embedding representing their taste similarities and various regression models pre-trained on them using different visual features. Each uploaded image is then resized and some of its visual features are extracted depending on the selected prediction method. A corresponding model is then used to predict an embedding position of the selected image and five nearest neighbors of the predicted position are displayed to the user.

Chapter 9

Testing Data

A well-annotated dataset of images is an essential part of a development and testing of any machine learning system that works with visual features. This chapter describes how the testing data was obtained and what kind of images and annotations it contains.

9.1 Getting the Data

The easiest and cheapest way of getting the testing data is to use an already existing dataset which has sufficient amount of images and sufficiently good quality of desired annotations. Although nowadays there is a large number of open-source image datasets available in the computer vision community, there was no embedding-annotated image dataset at the time of this work. This also means that there are no previous experiments with results that would serve to compare the achieved performance.

Another possibility of getting the data is creating a synthetic embedding automatically based on the dataset annotations. That, however, appeared to be a difficult task since most of the existing image datasets are classification-oriented but embedding-driven learning is a regression problem. Most of the existing regression datasets contain no images at all and those that do have no suitable annotations. An extensive research of existing image datasets thus showed that creating and embedding from the existing annotations would be a very difficult task.

Therefore, it finally appeared that the best option to get a suitable dataset was to create a new one from scratch and collect all the desired annotations. Both the datasets described in this chapter were collected with the help of my colleagues since they were interested in using them as well.

9.2 Chosing an Image Set

While developing and testing machine learning algorithms it is often convenient to use at least two different datasets of different sizes and complexities. A small and simple dataset is useful for algorithm testing, verification, and data visualization, while a larger dataset is better to measure the real performance of the system.

Two different image sets with different requirements were thus chosen, collected, and annotated. A small "toy" dataset is presented in Section 9.3 and a larger dataset is described in Section 9.4.

9.3 A Toy Dataset (Flags-196)

A small "toy" dataset should serve especially for fast algorithm testing, verification of their correct functionality and a comprehensible visualization of the results. The requirements thus were to collect about two or three hundred of small and simple images with an embedding of only two dimensions to allow for its visualization.

After a vast research and consideration these requirements appeared to be fulfilled by a small set of simple and iconic images – the images of *country flags*. The idea of using flags was inspired by [24] since the authors used them to plot an embedding, however, trying to get the data did not bring any results since the dataset turned out to be proprietary.

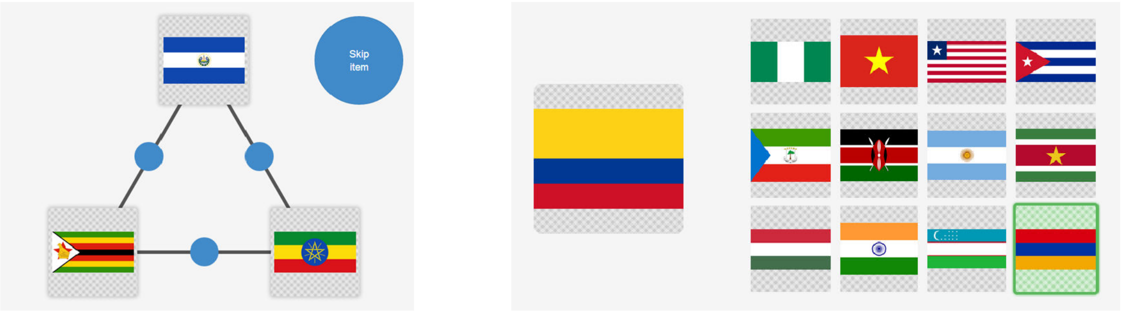
9.3.1 Data Collection

Images of flags of 196 countries[1] were collected from Wikipedia[2]. The images were then presented to different users who were providing answers about the relative visual similarities among the flags. The similarities were collected in two different ways:

1. The users were presented a screen with three flags and they were supposed to choose the two of them that were the most alike. From ten of such screens they were allowed to skip two if the answer seemed to bee to difficult.

2. The users were presented a flag called *probe* and from 12 other flags they were required to choose one that was the most similar to the probe. No answer skipping was allowed.

The answers to both of the above questions can be converted to different number of "triplet" statements of form "Flag A is more similar to B than to C". While an answer to the first questions produces 2 of such triplets, an answer to the second one gives us 11 triplets. The user interfaces for the two different ways of collecting similarities are shown in Figure 9.1.



(a) Choose 2 from 3

(b) Choose 1 from 12

Figure 9.1: Two different methods used to collect relative visual similarities among the flags. While (a) produces 2 similarity triplets, (b) generates 11 of them.

All the similarity annotations were collected by instructed volunteers and could be thus considered reliable. The amount of collected data was the following:

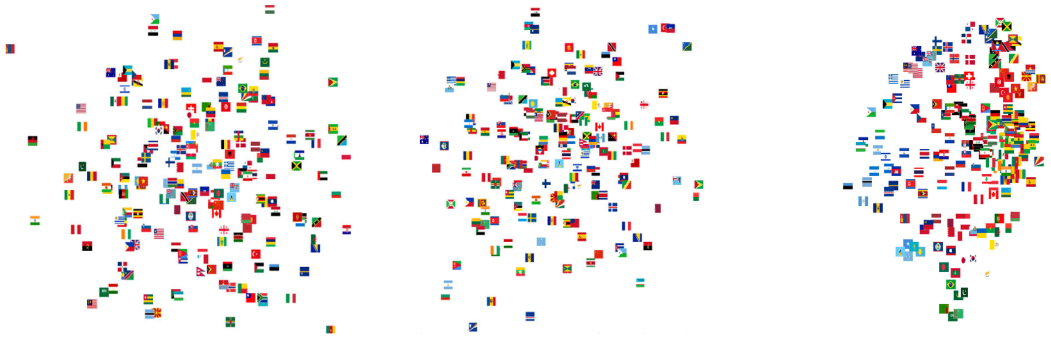
Question	Number of answers	Number of triplets
Choose 1 from 12	706	7766
Choose 2 from 3	2270	4540

[1] All the United Nations members (193), Unated Nations observers (2), and Taiwan.

[2] http://en.wikipedia.org/wiki/List_of_sovereign_states

9.3.2 Computed Embedding

A 2-dimensional embedding was then computed from the collected triplet comparisons using t-STE [26]. However, the results were not satisfactory since the embedding was extremely noisy and did not contain a reasonable structure. The bad quality embeddings computed from the collected data are shown in Figure 9.2 (a) and (b).



(a) Choose 2 from 3 (b) Coose 1 from 12 (c) Manual

Figure 9.2: Different 2-dimensional embeddings of flag images. Embeddings (a) and (b) were computed using t-STE from similarity triplets collected in two different ways. Neither of them seems to have a reasonable structure. Embedding (c) was created manually.

The failure of building a good quality 2-dimensional embedding from the collected similarity comparisons turned out to happen due to a high complexity of the flag dataset. Flag visual similarity appeared to be much more complicated than only a 2-dimensional (i.e. color and shape) property. In reality people perceive the visual similarity among the flags based on many different attributes – different respondents use different attributes to match the most similar flags such as color, stripes and their direction, stars, symbols and emblems, geometric shapes, etc.

The human-perceived visual similarity among country flags thus appeared to be a very high-dimensional, producing good embeddings in about 12 dimensions. That is, however, too complex for a toy dataset and impossible to be visualized in only two dimensions.

9.3.3 Manual Embedding

Since the human-collected similarities among the country flags lead to excessively complex embeddings, the problem of getting a 2-dimensional embedding had to be solved in a different way. For the purposes of development of an embedding-driven learning system it is not important how an embedding was produced and what it actually represents. As long as it has an underlying structure (i.e. the embedding coordinates are not random), the learning algorithms should be able to discover and model it.

Therefore, a much easier approach was chosen to get the embedding with two dimensions – the flags were shown on a screen and manually placed in a 2-dimensional space according to their similarity by a human. The resulting embedding can be seen in Figure 9.2 (c). While the horizontal dimension groups the flags more by color and the vertical one is more shape-oriented, this attribute separation is not strict and the data is on purpose noisy, sorted quickly without any additional corrections or assumptions.

9.4 A Larger Dataset (Food-1000)

A larger dataset should serve to test the performance of the system and show how the algorithms would perform on more realistic data. Such dataset should contain at least a thousand of rather realistic images with an embedding of arbitrary dimensionality. The requirements for such dataset are thus more relaxed and since we are not limited to a specific complexity and dimensionality of the embedding, there should be no problems similar to those that appeared when creating the toy dataset (see Section 9.3).

Since one of the new taxa that the Visipedia project (see Chapter 2) wants to focus on is food and cuisine, the ideal choice of images for a larger dataset was *food*. Images of food have a large variability in colors, shapes, and textures so the data should be complex enough. The property that the embedding of the food images should represent would be its *taste* meaning that meals that taste more similar should lie in the embedding closer to each other while meals of different taste should lie further away.

9.4.1 Data Collection

A set of 1000 random images of food from a recipe portal Yummly[3] was collected along with other meaningful metadata such as ingredients that might be useful for other experiments. The images were then annotated with relative similarity comparisons provided by payed workers[4] on Amazon Mechanical Turk[5].

The similarity comparisons were collected by showing the workers an image as a *probe* and 12 another random images and asking them to select 4 of them that are the most similar to the probe. The required task was formulated as follows:

> “Select 4 foods from the right that **taste** more similar to the food on the left.”

This method is analogous to the “choose 1 from 12” used for the toy dataset (see Section 9.3) and the user interface was similar to that one showed in Figure 9.1 (b). The main difference compared to the method used for the toy dataset is that choosing 4 images at once produces much more information. A single answer to this question creates 32 triplets of form “A is more similar to B than to C” making it much more suitable for a larger dataset.

Since the similarity comparisons were provided by anonymous workers, every 2 from 20 screens were prepared in advance and had clear answers that everybody was expected to submit. Those prepared screens are called *catch trials* and can serve to filter out the data from workers who provide bad responses.

The amount of collected data is summarized in the following table:

Images	1 000
Answers	6 000
Raw triplets	192 000
Catch trials	600
Different workers	65
Cost	$60

[3] http://www.yummly.com/

[4] The fundings were kindly provided by professor Serge Belongie.

[5] https://aws.amazon.com/mturk/

9.4.2 Computed Embedding

Responses from workers who did not pass the catch trials were then filtered out leaving 156 617 different triplets. Those were then used to compute embeddings from 2 to 20 dimensions with t-STE [26], picking the 8-dimensional one since adding more dimension did not show significant improvements in its quality.

The resulting 8-dimensional embedding has a good global structure separating main groups of food, but locally remains quite noisy. Getting a better quality embedding would require collecting more similarity comparisons which could make the dataset much more expensive. A 2-dimensional embedding computed from the same triplet set as the resulting embedding is shown in Figure 9.3.

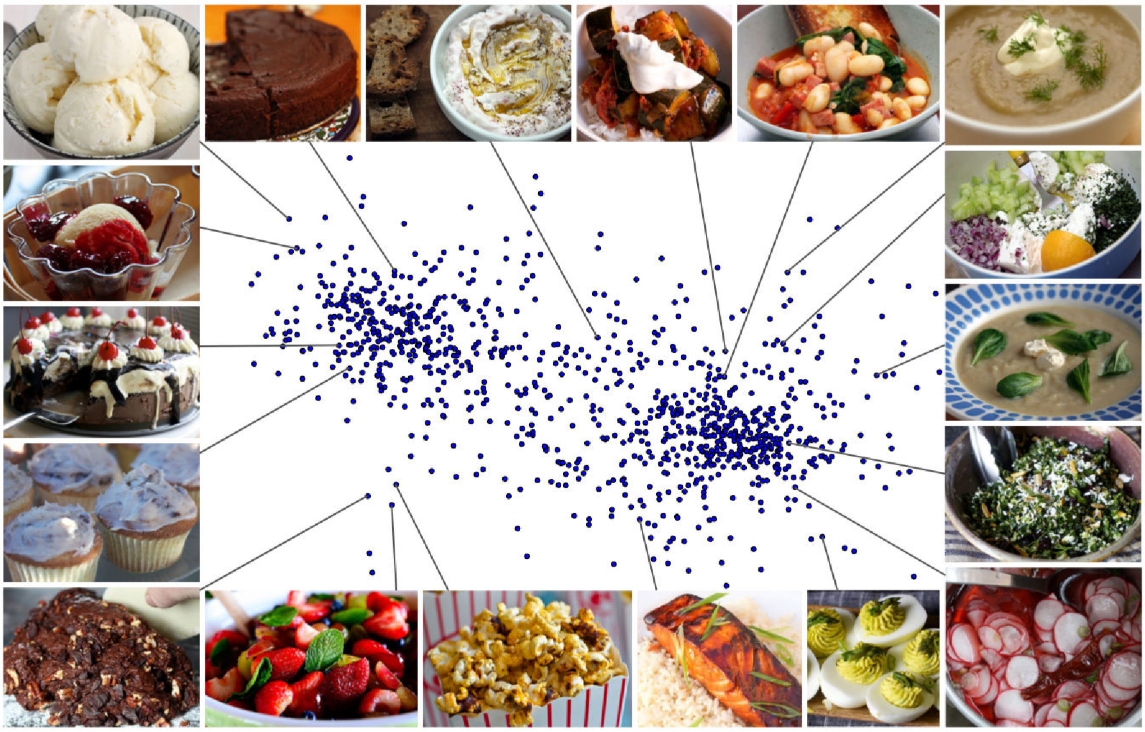


Figure 9.3: A 2-dimensional embedding of the food images computed from the collected similarity comparisons. The embedding contains two main clusters that represent sweet and non-sweet meals. While the global structure is well captured, locally the embedding is still quite noisy. However, the resulting 8-dimensional embedding has a better quality since it satisfies more of the collected triplet comparisons.

Chapter 10

Toy Experiments

This chapters describes experiments that were done using the toy dataset of flags described in Section 9.3. Their main purpose of these experiments was to verify the correct functionality of all the implemented algorithms, visualize the results in a comprehensible way and prepare a set of tools for experiments with larger amount of data.

10.1 Parameter Tuning

A very important and uneasy part of the image-based embedding-driven learning is to choose the best parameters for both the feature extraction algorithm and the regression learning function. Bad choice of visual feature parameters may result in extracting useless information and wrong SVR parameters might cause underfitting or overfiting.

The most accurate choice of parameters can be found by a cross-validation process trying to evaluate the prediction error for every possible parameter combination and then picking the best one. However, the number of all the possible parameter combinations may be extremely large and make such process computationally very difficult or even impossible. For instance trying 10 different values for only 2 parameters means computing 100 times the whole learning pipeline.

To lower the complexity of the parameter tuning process the parameters were at first set to only a few distant values and then evaluated more precisely around the best values. Also, instead of computing all the possible combinations some parameters were fixed to reasonable values and only one was evaluated in detail. This way the parameters were gradually set to their best values and in some cases they were revalidated when another parameters changed significantly.

This section describes the most important parameters for each of the used methods and finally sums up the best found values.

Kernel function. As mentioned in Section 6.4.3 the SVR algorithm can work with various different kernel functions. To choose the best one a significant part of the parameter tuning experiments was tested with three different kernel functions – linear, polynomial and RBF.

In all the test cases the linear function performed significantly worse than the other two kernels. The linear kernel appeared to be very unstable – while in some parameter configurations it could give quite reasonable results, in other cases it showed a very bad performance and it was never able to outperform the RBF and polynomial kernels.

As said in Section 6.4.3 while linear kernels may sometimes work to separate non-linear data for classification, in case of regression the process of fitting a line to non-linear data is much less likely to work.

The best and most stable results were reached with the RBF kernel. While the polynomial kernel was able to produce similar results to RBF, it was always slightly worse or comparable to RBF. Therefore, the RBF function appeared to be the most suitable and universal kernel and there was no reason to choose any other for the experiments. All of the further described experiments were thus computed using the RBF kernel.

SVR. As described in Section 6.4, the SVR algorithm is mainly driven by two parameters – ε_{SVR} and C. The value of ε_{SVR} controls the flatness of the estimated function by setting the size of the ε-tube whereas C affects its sensibility to noise in the data.

Both of the parameters may cause overfitting when set to excessively high values and underfitting when their values are too low. The correct setting of these values is crucial for success of the learning process.

Color histograms. A global color histogram (further denoted by CHG) have one important parameter – the number of bins that specifies the amount of different shades that the histogram will capture for each color channel of the image. Large number of bins can carry more color details but might be more sensitive to noise while lower numbers can describe a more general color structure. The number of bins is further denoted by n_b.

When computing local color histograms (denoted by CHL) we need to set another important parameter – the size of blocks for which the histograms are computed. Smaller blocks are more precisely localized and are thus useful for object that we expect to be aligned at the same position in all images while larger blocks represent a more general color structure. The block size parameter represents the height and width of each block in pixels and is further denoted by b_s.

HOG. The HOG descriptors were extracted in its two most typical configurations. Both of them use square cells of 8×8 pixels but they differ in the number of cells in a block. The first configuration uses blocks of 2×2 cells while the second one has blocks of size 3×3.

Since different block sizes can extract quite a different information from the image, both the configurations were used as two different feature vectors and are further in the text referred to as HOG 2×2 and HOG 3×3.

LBP. For all the experiments the *uniform* variant of LBP patterns was used since it is known to perform the best and it produces smaller feature descriptors (see Section 4.3). The most important parameters of the LBP features influence the size of the neighborhood in which LBP values are computed and the number of such values. The first parameter is the radius of the neighborhood R and the second one represents the number of neighboring points P, as defined in Section 4.3.

Similarly as for HOG, the LBP descriptors have a few most common variants that perform well and extract different kind of information. Values of R from 1 to 3 are the most typical setting the P eight times larger. All the three variants of LBP were extracted and used as three different feature vectors denoted by LBP 1/8, LBP 2/16, and LBP 3/24.

SIFT. The SIFT feature extraction is mostly driven by two parameters that influence the number of detected SIFT key points. By tuning those parameters we can achieve getting a larger number of detected key points of possibly lower stability or a smaller amount of high quality ones.

The first parameter is called *peak threshold* and in this text it will be denoted by t_p. The peak threshold filters out the peaks of the DoG function that have too small absolute values because such peaks could be caused by noise. Higher values of t_p thus produce smaller number of "better-quality" SIFT key points while lower numbers produce larger numbers of key points that do not necessarily have to be so distinctive.

The second parameter, called *edge threshold* and denoted by t_e in this text, is used to eliminate the peaks of the DoG function that are not sharp enough. Such peaks can be produced by edges in places where no stable key points exist. Contrary to peak threshold, higher values of t_e preserve more SIFT key points, while lower values filter them more aggressively.

Since transferring SIFT descriptors to feature vectors for computing regression is done by the BoVW model (see Section 4.5), another important parameter is the size of the BoVW dictionary, i.e. the number of clusters used for the K-Means clustering. This parameter will be denoted by n_c in this text.

10.1.1 Selected Parameters

The best parameters for the visual features and the SVR algorithm selected by the described parameter tuning process are summarized in the following table:

Feature	C	ε	Feature parameters
CHG	20	0.02	$n_b = 3$
CHL	4	0.01	$n_b = 4$, $b_s = 10$
HOG 2x2	2	0.01	
HOG 3x3	2	0.01	
LBP 1/8	30	0.05	
LBP 2/16	2	0.1	
LBP 3/24	1	0.08	
SIFT	0.5	0.05	$n_c = 10$, $t_e = 1000$, $t_p = 0$

For feature combination methods the following values worked the best:

Method	C	ε
Concatenation	3	0.02
Regression	0.5	0.06

As we can see, the best setting for global color histograms uses only 3 color bins for each RGB channel resulting in 27 different color shades. Hovewer, given the fact that most flags contain very little amount of different colors, using 27 different colors seems to be meaningful. The 10-pixel block size used for the local color histograms means that the images are divided into large blocks and thus only approximate color localization matters.

Since most of the country flags are a very simple geometric objects, often composed only from a couple of stripes, there are very few key points that could be detected by SIFT. Therefore, SIFT works the best when setting high values of the edge threshold and low values of the peak threshold and thus increasing the number of detected key points.

10.2 Experiment Setting

At first an SVR model was fitted for each of the visual features used the scheme proposed in Section 7.1 and the prediction accuracy was evaluated on a testing set. This was done on a randomly shuffled dataset using 5-fold cross-validation as described in Section 5.4. Then all the three multiple feature combination methods presented in sections 7.2, 7.3 and 7.4 were applied the same way.

The results were evaluated on a per-dimension basis using the MAE measure and then an overall error evaluation was computed using both MAE and MEE. To get a better idea of the performance of the system, all the mentioned methods were compared against the two proposed random models RVM and RNM. For further details about these error measures and the used random models see Section 6.5.

Finally, all predictions were visualized in context of the training set.

10.3 Per-dimension Prediction Evaluation

The per-dimension results of the prediction quality evaluation for all of the tested methods are depicted in Figure 10.1. The exact numeric results then can be found in Table B.1.

The results show what one would probably expect. In dimension 1 of the flag embedding (horizontal in later figures) which is more color-oriented the global and local color histograms perform much better than the other more structure-based visual features. The best predictions for dimension 2 (vertical in later figures) are then achieved with HOG and LBP since the dimension is more structure-oriented.

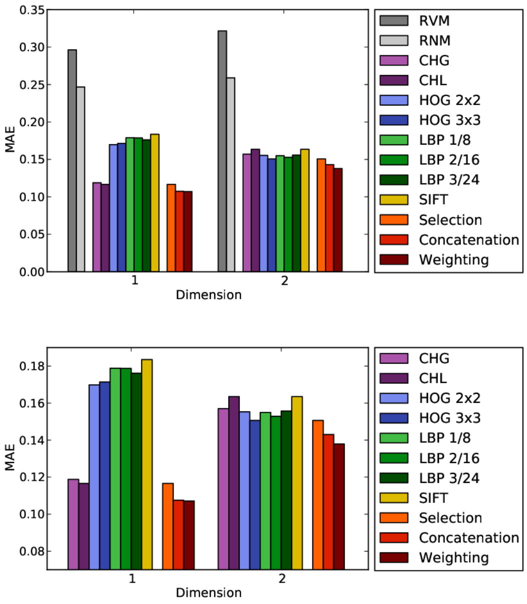


Figure 10.1: Per-dimension MAE for the Flags-196 dataset. The top figure shows the results for each dimension in context of the RVM and RNM predictions while to bottom one zooms in to capture the differences among the different proposed methods. The best single-feature results for dimensions 1 and 2 are achieved with CHL and HOG 3×3 and the total best values belong to feature weighting in both cases.

The best-performing feature in dimension 1 was CHL reaching errors 2.54× smaller than RVM and 2.12× smaller than RNM. In dimension 2 the best feature was HOG 3×3 achieving errors 2.14× smaller than RVM and 1.72× smaller than RNM.

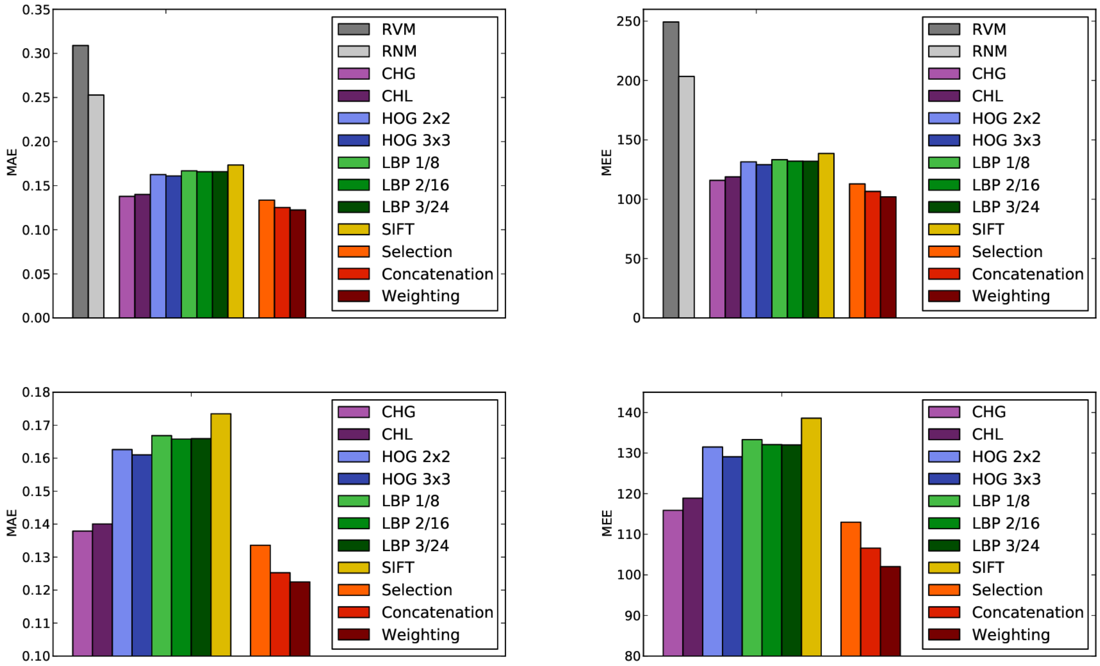
The results show that the combination of multiple visual features has brought further improvements in the prediction accuracy. Naturally, selecting the local color histograms for dimension 1 and HOG 3×3 for dimension 2 performs better than using just a single feature for both of the dimensions.

Concatenation of all of the extracted visual features brings further improvements in both of the dimensions when compared with feature selection which means that the learning algorithm was able to extract more information from the concatenated feature vectors.

The best results in each of the dimensions were achieved with feature weighting with errors 2.77× lower than RVM and 2.30× lower than RNM in dimension 1 and 2.33× lower than RVM and 1.88× lower than RNM in dimension 2.

10.4 Overall Prediction Evaluation

The values of the MAE and MEE measures evaluated over the whole embedding are visualized in Figure 10.2 and the numeric values can be found in Table B.2. From single visual features color histograms performed the best while SIFT gave the worse results. All the three feature combination methods outperformed all of the single-feature predictions.



(a) MAE

(b) MEE

Figure 10.2: Overall values of MAE and MEE on the Flags-196 dataset. The top row shows both the values in context of RVM and RNM while the bottom line zooms in to highlight the differences among the evaluated methods. The best single-feature method was CHG while the best overall results were reached with feature weighting.

The best overall performance using only a single visual feature was reached with CHG showing that color gives the best distinction of the coordinates in the embedding. Using the overall MEE measure CHG was 2.15× better than RVM and 1.75× better than RNM.

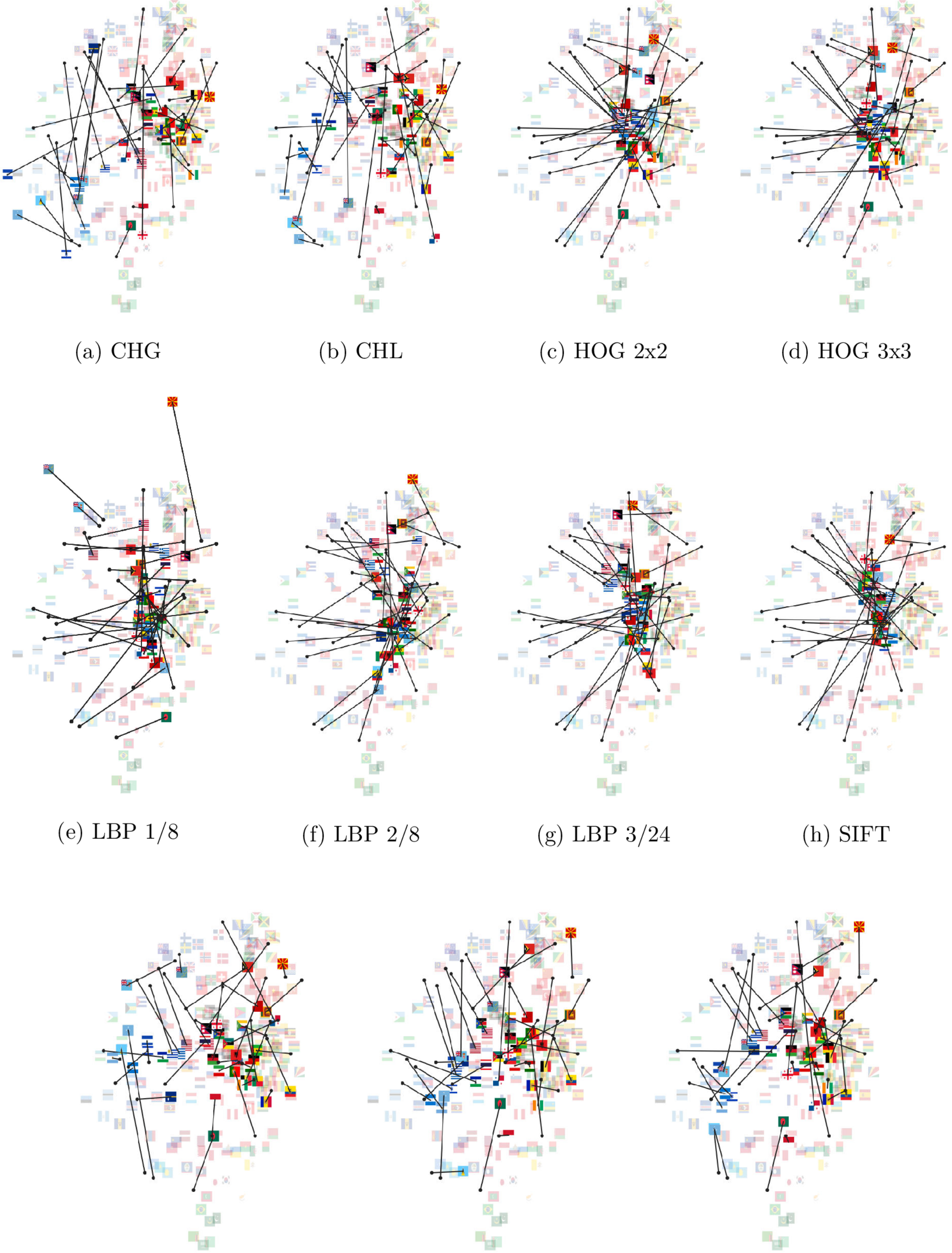
The absolute best overall results were obtained with feature weighting proving that multiple visual features can be combined to get more information. Measured with MEE the feature weighting method produced errors 2.44× lower than RVM and 1.99× better than RNM setting thus the best results that were achieved on the Flags-196 dataset.

The predictions for each of the visual features together with the three methods of their combination are shown in Figure 10.3.

10.5 Summary

The toy dataset served well to implement and debug the embedding-driven learning system and helped to verify that all the algorithms work correctly. The experiments gave expected results – a more color-oriented dimension was predicted the best by color histograms while a more structure oriented one was estimated the best using HOG features. Moreover, combining visual features together have brought further improvements.

The best-performing method produced almost 2 times smaller errors than picking random positions from the training set and on average was off by 12.25 % in each dimension relative to the distance between its extreme values. The remaining errors might be caused by noise in the embedding, small set of training data, visual complexity of some of the images, and imperfection of some of the used methods.



(a) CHG

(b) CHL

(c) HOG 2x2

(d) HOG 3x3

(e) LBP 1/8

(f) LBP 2/8

(g) LBP 3/24

(h) SIFT

(i) Selection

(j) Concatenation

(k) Regression

Figure 10.3: Visualization of the predictions on the Flags-196 dataset. Pale flags represent the training set while the dark ones are testing flags placed on their predicted positions and connected with their true positions with lines. Figures (a) – (h) show predictions for single-feature methods whereas (i) – (k) depict predicted positions using feature combination. Methods (a) and (b) work well in the color-based horizontal dimensions while (c) – (h) fail since they are more structure-oriented. The best overall result is achieved with (k).

Chapter 11

Larger Experiments

This chapter describes the experiments that were done using the Food-1000 dataset that was presented in Section 9.4. The main purpose of these experiments is to evaluate the quality of the developed system using a larger and more complex data with noise.

11.1 Parameter Tuning

The feature extraction and regression learning parameters were tuned applying the same approach that was used for the toy dataset as described in Section 10.1. The RBF kernel was used for all of the experiments and the values of the parameters for each of the methods are the following:

Feature	C	ε	Feature parameters
CHG	30	0.07	$n_b = 4$
CHL	15	0.05	$n_b = 4,\ b_s = 64$
HOG 2x2	2	0.01	
HOG 3x3	2	0.01	
LBP 1/8	30	0.08	
LBP 2/16	10	0.07	
LBP 3/24	5	0.05	
SIFT	5	0.05	$n_c = 10,\ t_e = 10,\ t_p = 10$

The SVR learning parameters for the methods combining multiple visual features worked the best using the following values:

Method	C	ε
Concatenation	3	0.05
Regression	0.2	0.1

All the images were resized to 128×128 pixels providing a good balance between prediction speed and accuracy. Both global and local color histograms worked the best using 4 bins per color giving the resolution of 256 different color shades. For SIFT the best results were achieved when slightly reducing the number of detected key points by increasing the peak threshold from 0 to 10 and using only 10 clusters for the BoVW.

11.2 Per-dimension Prediction Evaluation

All the single feature based and multiple feature combining methods were initially evaluated computing the MAE for each of the 8 dimensions using 5-fold cross-validation. The results are visualized in Figure 11.1 and the exact values can be found in Table C.1.

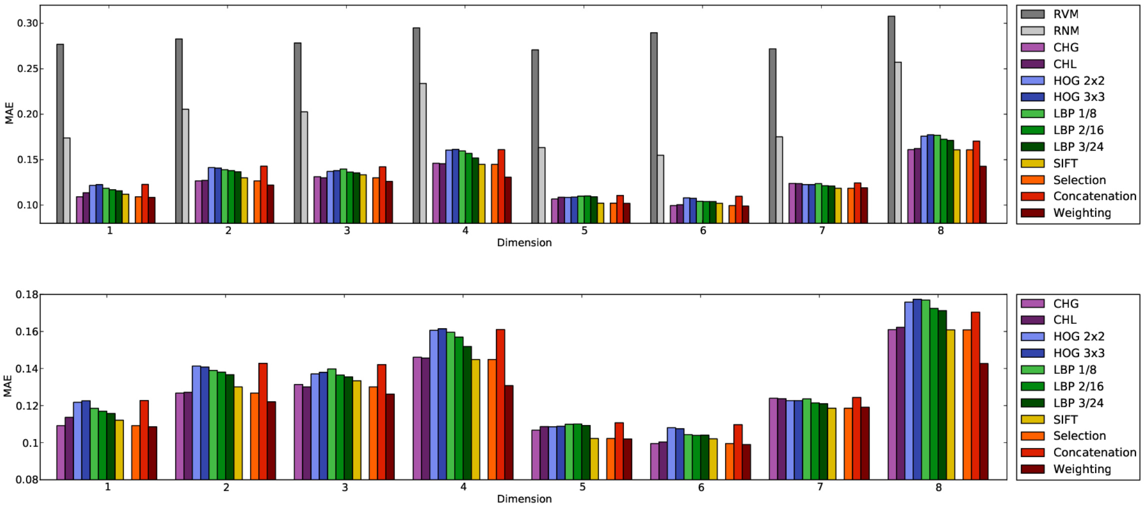


Figure 11.1: Per-dimension MAE for the Food-1000 dataset. The top image shows the error values in context of the RVM and RNM predictions while the bottom one zooms to the differences among the evaluated methods. The best single-feature methods were CHG, CHL and SIFT, the total best values were obtained with feature weighting while feature concatenation failed to work.

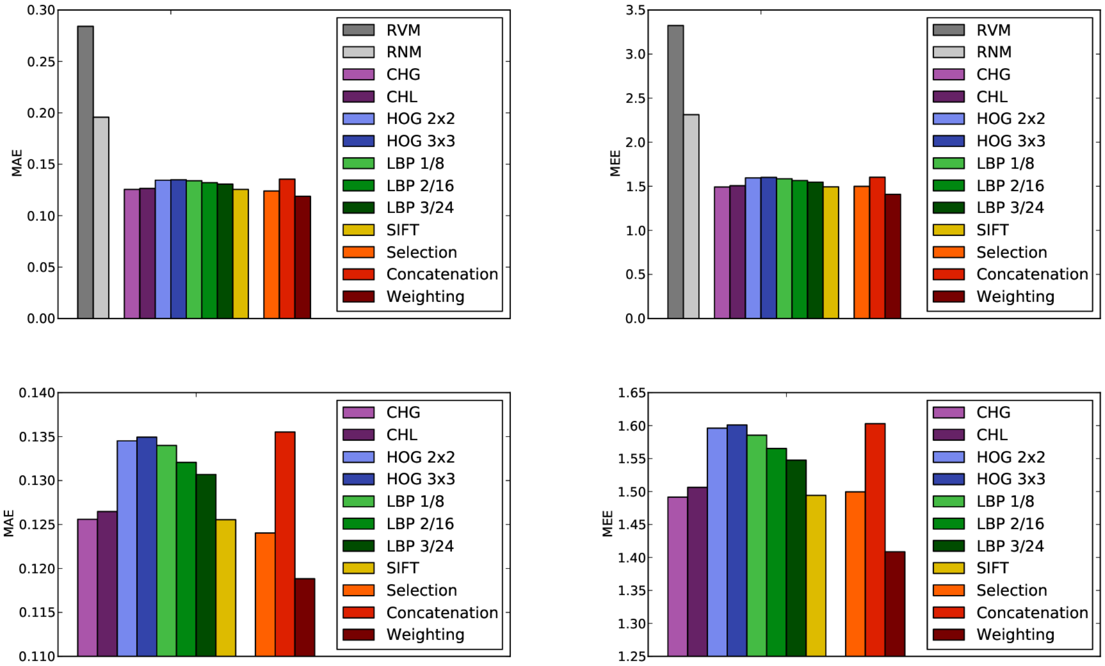
Using single-feature methods the best results were achieved with color histograms and SIFT in all of the dimensions. In some dimensions LBP was reaching comparable results while in others it performed slightly worse. Both configurations of HOG gave slightly worse results than the other methods and using different sized input images or the BoVW approach did not bring any improvements.

Contrary to the toy experiments described in Chapter 10 concatenating all the extracted visual features did not improve the results. In most of the dimensions the results were even worse than using any single visual feature. Such performance drop is probably caused by high multi-modality of the resulting feature vector where the largest part is composed of HOG while some other feature descriptors are much shorter. Even though all of the single feature vectors were normalized to the same range, the dominant parts of different feature vectors can still differ significantly in value and cause other sources of errors. The risks of using feature concatenation were already discussed in Section 7.3. Using principal component analysis to lower the dimensionality of the resulting concatenated vector did not bring significant improvements.

The best results for most of the dimensions were achieved with the regression-based feature weighting as described in Section 7.4. Hence, although the feature concatenation failed to work on the Food-1000 dataset, a more robust feature weighting was still able to improve the results.

11.3 Overall Prediction Evaluation

The overall prediction accuracy was evaluated using both MAE and MEE measures (see Section 7.5) for all of the proposed feature extraction and combination methods. The obtained results are visualized in Figure 11.2 and all the corresponding numeric values are to be found in Table C.1.



(a) MAE

(b) MEE

Figure 11.2: Overall values of MAE and MEE on the Food-1000 dataset. The top row shows both values in context of RVM and RNM while the bottom one zooms in to highlight the differences among the evaluated methods. The best single-feature methods were SIFT and CHG while the best overall results were obtained with feature weighting.

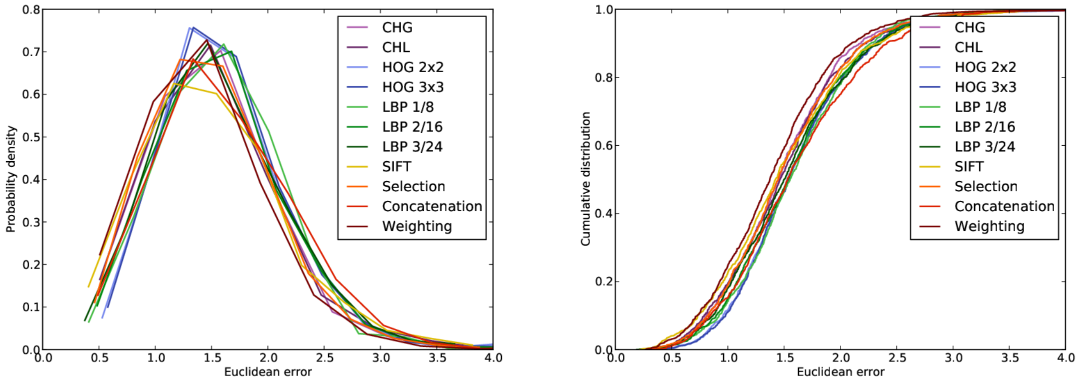
The overall results reflect the per-dimension computed errors described in Section 11.2. Among the single-feature based predictions color histograms and SIFT work the best while HOG and LBP give slightly worse results. Color thus proves to be a very important property for food image description and finding stable key points shows better distinction than using intensity gradients or texture patterns. Hence, shapes seem to be less important for food distinction than stable image areas.

Using per-dimension feature selection gives only a slightly better MAE value than the color histograms and SIFT since the selected features are composed mostly from these two. Computing the MEE measure showed that feature selection gives no improvement over using color histograms or SIFT only. This can be caused by the fact that different dimensions have different error variance and also a different impact on the final predicted coordinates since for learning they have to be normalized but their original scales can vary a lot. While mapping the normalized values for each dimension back to their original scale some errors can be accentuated more than others.

As already discussed in Section 11.2 the feature concatenation failed to work on a larger and more complex data and attempting to reduce the dimensionality of the concatenated vector did no bring any improvements.

The best overall results were achieved using the regression-based feature weighting showing significant improvements in both MAE and MEE measure values. The proposed method of feature weighting thus proved the ability of combining information from predictions based on different visual features even on a larger and more complex data. Measured by MEE the feature weighting method produced errors 2.36× lower than RVM and 1.65× lower than RNM.

In order to get a better picture of the quality of the predictions the cumulative distribution function (CDF) of the Euclidean error was computed for all of the predicted values and each of the learning methods. An approximation of the probability density function (PDF) was also calculated using a sparse histogram of the error values. The shape of the CDF and PDF functions is visualized in Figure 11.3.



(a) PDF

(b) CDF

Figure 11.3: The probability density function (PDF) and cumulative distribution function (CDF) computed for the MEE of all the predicted values. All the methods appear to produce the same shapes of the functions differing only in horizontal shifts.

11.4 Summary

All the developed methods were evaluated using a larger dataset of complex images with a noisy embedding computed from answers produced by humans. The experiments showed that the system works also with such data and can give an idea of the usefulness of the developed system in a real-world applications.

The average error values are comparable to the results of the toy experiments described in Chapter 10. Although the Food-1000 dataset offers a larger set of objects for the learning phase, the images are more complicated and their embedding is probably more noisy. Higher difficulty of the task is thus compensated by a larger amount of data resulting in similar numeric results.

The best overall results were achieved with feature weighting producing 1.65× smaller errors than taking positions of random items in the training set. On average the predictions in each dimension were off by 11.88 % relative to the distance between its extremes.

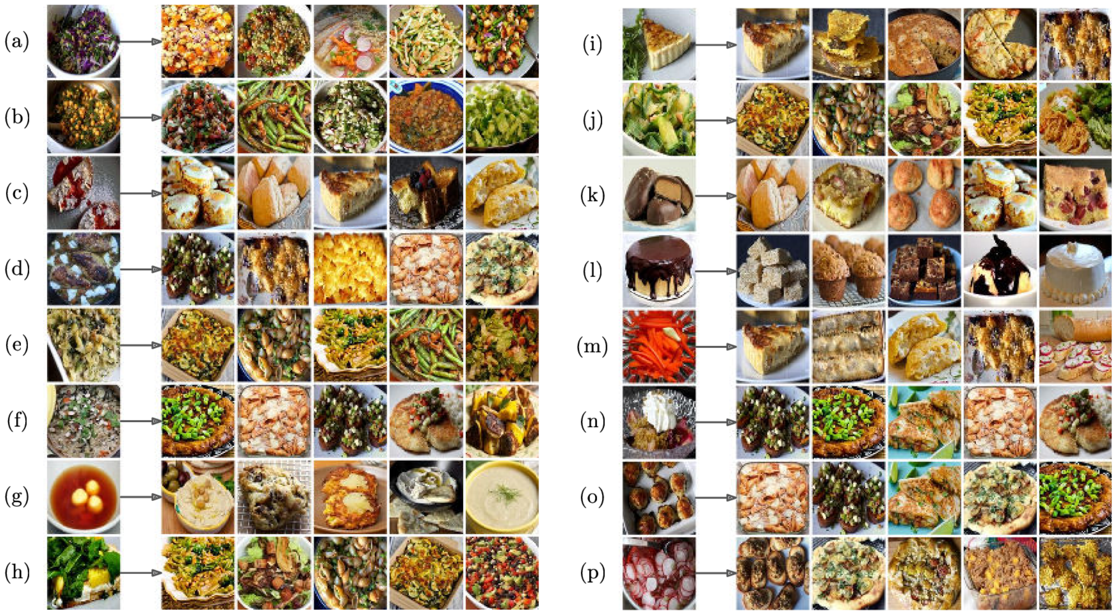


Figure 11.4: Some of the images from the testing set and 5 nearest neighbors of the predicted position computed using feature weighting. Predictions (b), (e), (f) and (h) look particularly good, while (g) was probably biased by non-food objects and some others like (m), (n), and (p) were predicted incorrectly.

To give an intuition about the prediction quality and the scopes of possible usage of the developed system, 5 nearest neighbors from the training embedding are displayed for some of the predicted positions in Figure 11.4 using the regression based feature weighting which gave the best overall results.

While many predictions seem to be quite accurate, others are influenced by confusing data, noise, and background objects leaving many challenges for future improvements. The experiments also provide the first results on the new Food-1000 dataset which can be later compared to results achieved using different learning and feature extraction methods.

Chapter 12

Conclusions

The aim of this work was to study the ways of learning from embedding-annotated images in order to develop a system for predicting embedding coordinates of previously unseen images and evaluate its performance on adequate data. This goal was achieved.

The concept of image annotation using multidimensional embedding was analyzed together with the relevant learning methods and various different visual feature extraction algorithms were studied to capture different visual attributes in an image. An embedding-driven learning system was proposed and implemented including several techniques of combining multiple visual attributes and a simple application demonstrating its capabilities was developed. The performance of the system was evaluated using two different datasets.

The performed experiments have brought promising results. Average distances of predicted embedding positions from their ground truth were, compared to choice of a random item from the training set, 2× smaller on Flags-196 and 1.65× smaller on Food-1000. On average the per-dimension error relative to the distance between its extremes reached 12.25 % on Flags-196 and 11.88 % on Food-1000. Those results also represent the first achieved measures for a task of its kind that can be challenged by future methods.

While recent research has shown that multidimensional embedding can be a very powerful method of capturing similarities among images, this work demonstrates that the underlying similarity metric may also be learned from their visual features. Embedding-driven learning systems thus could be used to add new images to an existing embedding, speed up visual search through the similarity space, and perform fine-grained categorization. Since learning methods are not perfect they could be initially combined with human interaction resulting in applications such as search by an image query accompanied with user's activity to precise the results by clicking on a few images.

This work has approached the embedding-driven learning by using common machine learning approaches to give an initial idea of the possible performance of such systems. Future work could focus on adding more visual features to the learning pipeline and evaluating their performance. An interesting approach would be to extract the visual features with a deep convolutional neural network pre-trained on a large dataset since this technique has recently shown significant improvements in various tasks. Other regression algorithms might also be explored and instead of decomposing the embedding to single dimensions the learning could be optimized to learn all of them at once. Since the task of embedding-driven learning is in its early stage, wide range of opportunities of future research arise.

Although this work has brought some uneasy challenges, the results motivate to explore the problematic more and attempt to improve the achieved performance. I would be happy to participate in promoting the system onto its next level and continue to collaborate with the Visipedia team.

Bibliography

[1] S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman, and S. Belongie. Generalized Non-metric Multidimensional Scaling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 11–18, 2007.

[2] E. Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2004.

[3] J. Anderson, R. Michalski, J. Carbonell, and T. Mitchell. *Machine Learning: An Artificial Intelligence Approach.* Machine Learning. M. Kaufmann, 1983.

[4] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual Recognition with Humans in the Loop. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 438–451, 2010.

[5] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[6] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893, 2005.

[7] C. Dance, J. Willamowski, L. Fan, C. Bray, and G. Csurka. Visual Categorization With Bags of Keypoints. In *ECCV International Workshop on Statistical Learning in Computer Vision*, pages 1–22, 2004.

[8] N. Draper and H. Smith. Applied Regression Analysis (3rd ed.). John Wiley & Sons, Inc., 1998.

[9] C. W. Hsu, C. C. Chang, and C. J. Lin. A Practical Guide to Support Vector Classification, 2010.

[10] J. J. Koenderink. The Structure of Images. *Biological Cybernetics*, 50:363–396, 1984.

[11] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1145, 1995.

[12] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, 1999.

[13] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[14] T. Ojala, M. Pietikäinen, and D. Harwood. Performance Evaluation of Texture Measures With Classification Based on Kullback Discrimination of Distributions. In *Proceedings of the IAPR International Conference on Pattern Recognition (ICPR)*, volume 1, pages 582–585, 1994.

[15] T. Ojala, M. Pietikäinen, and D. Harwood. A Comparative Study of Texture Measures with Classification Based on Featured Distributions. *Pattern Recognition*, 29:51–59, 1996.

[16] T. Ojala, M. Pietikäinen, and T. Mäenpää. Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 1842, pages 404–420, 2000.

[17] T. Ojala, M. Pietikäinen, and T. Mäenpää. A Generalized Local Binary Pattern Operator for Multiresolution Gray Scale and Rotation Invariant Texture Classification. In *Proceedings of International Conference on Advances in Pattern Recognition (ICAPR)*, pages 397–406, 2001.

[18] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:971–987, 2002.

[19] P. Perona. Visions of a Visipedia. In *Proceedings of the IEEE August 2010*, volume 98, pages 1526–1534, 2010.

[20] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[21] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers II: Recent Progress. *IBM Journal of Research and Development*, 11:610–617, 1967.

[22] C. Saunders, A. Gammerman, and V. Vovk. Ridge Regression Learning Algorithm in Dual Variables. In *In Proceedings of the International Conference on Machine Learning (ICML)*, pages 515–521, 1998.

[23] A. J. Smola and B. Schölkopf. A Tutorial on Support Vector Regression. *Statistics and Computing*, 14(3):199–222, 2004.

[24] O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. T. Kalai. Adaptively Learning the Crowd Kernel. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 11–18, 2011.

[25] Tom M. Michel. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[26] L. van der Maaten and K. Weinberger. Stochastic Triplet Embedding. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2012.

[27] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[28] V. Vovk. *Kernel Ridge Regression*, chapter 11, pages 105–116. Springer-Verlag Berlin Heidelberg, 2013.

[29] C. Wah, S. Branson, P. Perona, and S. Belongie. Multiclass Recognition and Part Localization with Humans in the Loop. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2524–2531, 2011.

[30] P. Welinder, S. Branson, S. Belongie, and P. Perona. The Multidimensional Wisdom of Crowds. In *Proceedings of the Neural Information Processing Systems Conference (NIPS)*, pages 2424–2432, 2010.

[31] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-201, Caltech, 2010.

[32] H. Zou and T. Hastie. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B*, 67:301–320, 2005.

Appendix A

CD Contents

The attached CD contains the following directories:

- `evfl` – Implementation of the EVFL[1] toolkit.
- `evfl-doc` – Documentation for the EVFL toolkit.
- `data` – Datasets used for experiments.
- `demo` – Demo application.
- `thesis-pdf` – This text in PDF format.
- `thesis-src` – Source codes of this text in LaTeX.

[1] Embedding-driven Visual Feature Extraction and Learning

Appendix B

Toy Experiment Results

Method	D1	D2
RVM	0.2963	0.3216
RNM	0.2467	0.2588
CHG	0.1188	0.1570
CHL	**0.1166**	0.1635
HOG 2x2	0.1698	0.1553
HOG 3x3	0.1714	**0.1506**
LBP 1/8	0.1788	0.1549
LBP 2/16	0.1787	0.1529
LBP 3/24	0.1761	0.1557
SIFT	0.1835	0.1635
Selection	0.1166	0.1506
Concat.	0.1075	0.1430
Weighting	**0.1071**	**0.1379**

Table B.1: Per-dimension evaluation results.

Method	MAE	MEE
RVM	0.3089	249.3
RNM	0.2528	203.4
CHG	**0.1379**	**115.9**
CHL	0.1401	118.9
HOG 2x2	0.1626	131.5
HOG 3x3	0.1610	129.1
LBP 1/8	0.1668	133.3
LBP 2/16	0.1658	132.1
LBP 3/24	0.1659	132.0
SIFT	0.1735	138.6
Selection	0.1336	113.0
Concat.	0.1253	106.6
Weighting	**0.1225**	**102.0**

Table B.2: Overall evaluation results.

Appendix C

Larger Experiment Results

Method	D1	D2	D3	D4	D5	D6	D7	D8
RVM	0.2770	0.2828	0.2785	0.2950	0.2707	0.2897	0.2719	0.3078
RNM	0.1739	0.2054	0.2026	0.2339	0.1633	0.1549	0.1751	0.2573
CHG	**0.1092**	**0.1268**	0.1314	0.1461	0.1068	**0.0996**	0.1240	0.1610
CHL	0.1137	0.1272	**0.1301**	0.1456	0.1087	0.1004	0.1238	0.1623
HOG 2x2	0.1218	0.1414	0.1371	0.1606	0.1086	0.1081	0.1227	0.1758
HOG 3x3	0.1226	0.1408	0.1380	0.1615	0.1089	0.1076	0.1227	0.1774
LBP 1/8	0.1186	0.1391	0.1398	0.1596	0.1100	0.1043	0.1237	0.1770
LBP 2/16	0.1170	0.1381	0.1365	0.1570	0.1101	0.1040	0.1215	0.1725
LBP 3/24	0.1158	0.1367	0.1355	0.1519	0.1093	0.1041	0.1210	0.1713
SIFT	0.1121	0.1301	0.1334	**0.1449**	**0.1023**	0.1021	**0.1186**	**0.1609**
Selection	0.1092	0.1268	0.1301	0.1449	0.1023	0.0996	**0.1186**	0.1609
Concat.	0.1228	0.1429	0.1421	0.1611	0.1108	0.1097	0.1244	0.1704
Weighting	**0.1086**	**0.1221**	**0.1262**	**0.1308**	**0.1020**	**0.0990**	0.1191	**0.1427**

Table C.1: Per-dimension evaluation results.

Method	MAE	MEE
RVM	0.2842	3.3240
RNM	0.1958	2.3128
CHG	0.1256	**1.4916**
CHL	0.1265	1.5064
HOG 2x2	0.1345	1.5962
HOG 3x3	0.1349	1.6010
LBP 1/8	0.1340	1.5855
LBP 2/16	0.1321	1.5654
LBP 3/24	0.1307	1.5477
SIFT	**0.1255**	1.4942
Selection	0.1240	1.4996
Concat.	0.1355	1.6029
Weighting	**0.1188**	**1.4086**

Table C.2: Overall evaluation results.