Dimensionality-Reduction Algorithms for Progressive Visual Analytics

Pezzotti, Nicola

DOI
10.4233/uuid:df6c0760-89ba-4db0-9621-19c512eb1955

Publication date
2019

Document Version
Final published version

Citation (APA)
Pezzotti, N. (2019). *Dimensionality-Reduction Algorithms for Progressive Visual Analytics*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:df6c0760-89ba-4db0-9621-19c512eb1955

Dimensionality-Reduction Algorithms for Progressive Visual Analytics

Dimensionality-Reduction Algorithms for Progressive Visual Analytics

Proefschrift

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on the 8th of April 2019 at 10:00 AM

by

Nicola Pezzotti

Master of Science in Computer Science and Engineering,
Università degli Studi di Brescia, Italy

born in Iseo, Italy

This work was out in the TU Delft graudate school.

This work was out in the ASCI graudate school.
ASCI dissertation series number 405.

This research is supported by the Dutch Technology Foundation STW, which is part pf the Netherlands Organisation for Scientific Research (NWO) and partly funded by the Ministry of Economic Affairs (project number 12720: "Visual Analysis in Population Imaging Research")

Printed by: Proefschriftmaken

Front & Back: Simona Bonafini Illustrations.

ISBN 978-94-6380-274-1

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/` and
`https://nicola17.github.io/dissertation`.

In a dark and confusing world,
you burn brightly.
I never feel lost.

Contents

Summary

Visual analysis of high dimensional data is a challenging process. Direct visualizations work well for a few dimensions but do not scale to the hundreds or thousands of dimensions that have become increasingly common in current data analytics problems. Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces, and it has been proven as an effective tool for high-dimensional data analysis. In visual analytics systems, several visualizations are jointly analyzed in order to discover patterns in the data.

One of the fundamental tools that has been integrated in visual analytics, is non-linear dimensionality-reduction; a tool for the indirect visualization aimed at the discovery and analysis of non-linear patterns in the high-dimensional data. However, the computational complexity of non-linear dimensionality-reduction techniques does not allow direct employment in interactive systems. This limitation makes the analytic process a time-consuming task that can take hours, days or even weeks to be performed.

In this thesis, we present novel algorithmic solutions that enable integration of non-linear dimensionality-reduction techniques in visual analytics systems. Our proposed algorithms are, not only much faster than existing solutions, but provide richer insights into the data at hand. This result, is achieved by introducing new data processing and optimization techniques and by embracing the recently introduced concept of Progressive Visual Analytics; a computational paradigm that enables the interactivity of complex analytics techniques by means of visualization as well as interaction with intermediate results.

Moreover, we present several applications that are designed to provide unprecedented analytical capabilities in several domains. These applications are powered by the algorithms introduced in this dissertation and led to several discoveries in areas ranging from the biomedical research field, to social-network data analysis and machine-learning models interpretability.

Samenvatting

Visuele analyse van hoog-dimensionale gegevens is een uitdagend proces. Directe visualisaties werken goed voor een klein aantal dimensies, maar schalen niet naar honderden of duizenden dimensies zoals steeds vaker het geval in huidige data-analyse problemen. Visuele analyse is de wetenschap van het analytisch redeneren gefaciliteerd door interactieve visuele interfaces, en het is bewezen als een effectief hulpmiddel voor hoog-dimensionale data-analyse. In visuele analysesystemen worden verschillende visualisaties gezamenlijk geanalyseerd om patronen in de data te ontdekken.

Een van de fundamentele tools die in visuele analyse is geïntegreerd, is niet-lineaire dimensionaliteitsreductie; een tool voor de indirecte visualisatie gericht op het ontdekken en analyseren van niet-lineaire patronen in de hoog-dimensionale data. De computationele complexiteit van niet-lineaire dimensie-reductie technieken laat echter geen directe tewerkstelling in interactieve systemen toe. Deze beperking maakt het analyseproces tot een tijdrovende taak die uren, dagen of zelfs weken in beslag kan nemen.

In dit proefschrift presenteren we nieuwe algoritmische oplossingen die het mogelijk maken om niet-lineaire dimensie-reductie technieken te integreren in visueel-analytische systemen. Onze voorgestelde algoritmes zijn niet alleen veel sneller dan bestaande oplossingen, maar geven ook een rijker inzicht in de data. Dit resultaat wordt bereikt door de introductie van nieuwe technieken voor gegevensverwerking en -optimalisatie en door het recent geïntroduceerde concept van Progressive Visual Analytics; een computationeel paradigma dat de interactiviteit van complexe analysetechnieken mogelijk maakt door middel van visualisatie en interactie met tussenresultaten.

Bovendien presenteren we verschillende toepassingen die ontworpen zijn om ongekende analytische mogelijkheden te bieden in verscheidene vakgebieden. Deze toepassingen worden aangedreven door de algoritmes die in dit proefschrift worden geïntroduceerd en hebben geleid tot meerdere ontdekkingen in gebieden variërend van biomedisch onderzoek tot sociale netwerkdata-analyse en interpretatie van machine-learningmodellen.

1 Introduction

And yet it moves!

Galileo Galilei

1.1 Motivation

In the 17th century, mainly thanks to the work of Galileo Galilei, what was known as natural philosophy became Science as we know today. This revolution was driven by the widespread adoption of the Scientific Method. The Scientific Method consists in a body of techniques that allowed humankind to understand the laws governing our world and, consequently, to manipulate it to our advantage. At its core, the Scientific Method is a tool for data-driven hypothesis generation and validation. In order to understand a natural phenomenon, scientists carefully design experiments and collect numerical data. Hypothesis on the laws governing the phenomenon are then formulated and are tested through a new set of experiments. The process is iterated until a law is found that is not disproved by new experiments.

Since the early days of Science, data visualization, i.e., the discipline focused on visual representation of data, played a crucial role in understanding the natural phenomena. A good example of this can be found in the early work of Galileo, more specifically, on his observation of the sun thanks to the then recently introduced telescope. Galileo observed and recorded the position of the "Sunspots", dark regions on the sun surface over a period of several days. By observing the evolution of the position of the Sunspots over time, an example of which is presented in Figure 1.1, Galileo observed that their movement could be partially explained by making the hypothesis that the sun is an imperfect and rotating sphere; an observation that went against the Aristotelian tradition that thought the Sun as unflawed and unmoving. Another seminal example of data visualization for hypothesis generation is the work of Dr. John Snow in the identification of the cause of Cholera outbreaks. In the 19th century it was thought that Cholera was caused by pollution and "bad air", generally identified with the term *Miasma*. Dr. Snow was skeptic of the Miasma theory and, therefore, performed a methodical data collection of Cholera cases during the outbreak in London of 1854. By plotting the location and the number of Cholera cases on the map presented in Figure 1.2, Dr. Snow hypothesized that the source of the disease was a water pump at the center of the map. This hypothesis, which originated from the visual analysis of the data, had to be empirically verified. Since no chemical nor microscopic examination of the water was able to confirm the hypothesis, the pump was made not functional by removing the rod that was activating it. Following this action, the Cholera outbreak ended, reinforcing the hypothesis that will be proven 30 years later by direct microscopical analysis.

These two results are just examples of the many successes of a data visualization approach in an Exploratory Data Analysis setting. Exploratory Data Analysis was formally introduced by Tukey in 1961 as a set of *"[P]rocedures for analyzing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data"* [172]. More specifically, data visualization helps in forming hypothesis of the underlying phenomenon that is currently investigated. After the data is gathered,

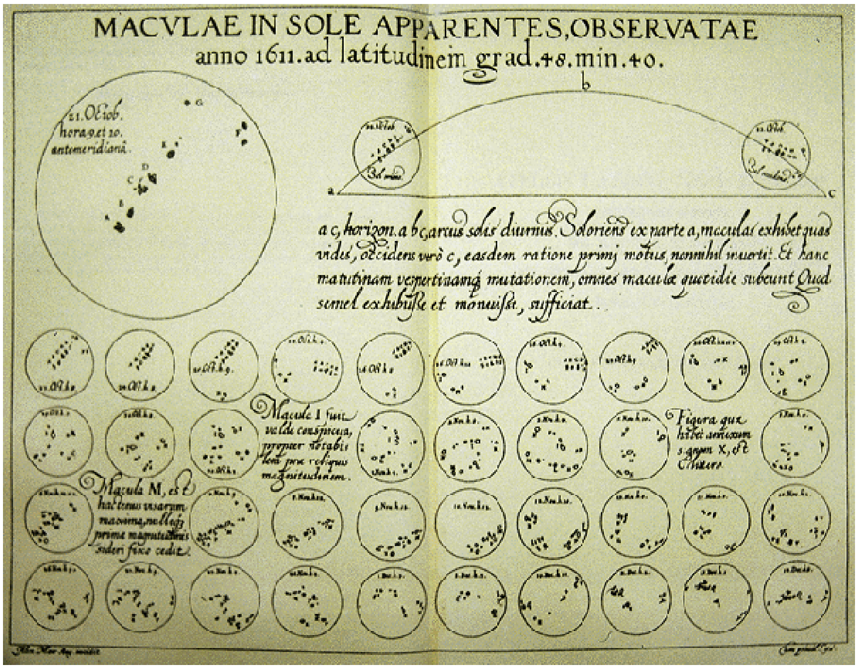

MACVLAE IN SOLE APPARENTES, OBSERVATAE
anno 1611. ad latitudinem grad. 48. min. 40.

Figure 1.1: **Maculae in Sole Apparentes** is the first example of visual analytics for hypothesis generation. By observing the position of the sunspots, i.e., dark points area on the surface of the sun for several days, Galileo Galilei inferred that the sun must be a rotating sphere, a notion against the Aristotelian tradition that thought the Sun as unflawed and unmoving.

the scientist creates visual representations that aim at discovering important patterns that would have been impossible to extract by a direct analysis of the data. These visual representations are then used to assist the scientist in phrasing a hypothesis on the phenomenon under analysis, and consequently, in supporting the design of experiments that can confirm or disprove the developed model.

Visual Analytics [85] is the research field that integrates human and machine analysis to provide solutions to problems whose size and complexity would make them otherwise intractable. Interactive interfaces and visualizations are complemented, in a visual analytics system, with computational tools to support the extraction of knowledge from the data. However, despite the successful application of visual analytics to support the hypothesis generation, recent years are characterized by new challenges that limit their application. New data acquisition techniques in the digital era, allow to collect and store data beyond any previously imaginable level. Scientists are not only faced with the problem of effectively analyzing millions of acquired data points, but also to deal with the inherent complexity of the acquired data due to the number of readings, i.e., dimensions, associated to each single data point.

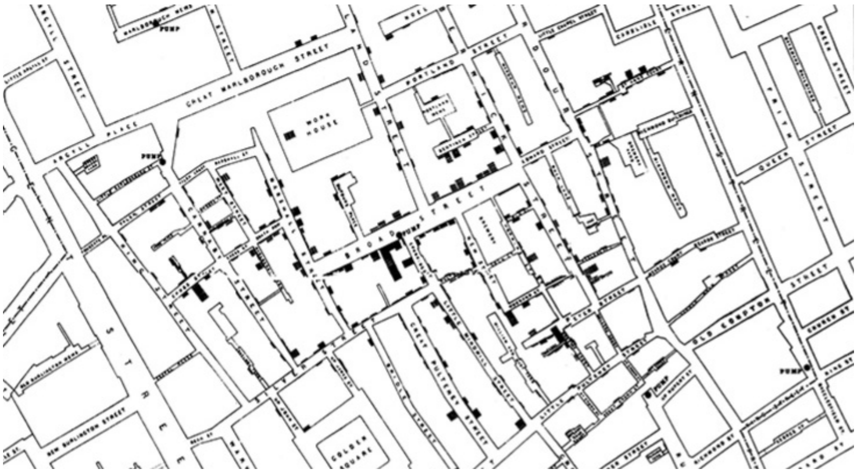


Figure 1.2: **Dr. John Snow's map** of Cholera cases during the outbreaks of 1854 in London. A visual analysis of the data allowed for the identification of the source of the outbreak as a water pump located in Broad Street, epicenter of the reported cases.

High-dimensional data are, in particular, inherently challenging to visualize and analyze. As humans, we learn to understand the world surrounding us trough our sensory input. Hence, we are naturally designed to be able to navigate in a 3-dimensional world and to interpret other sensory input such as sound and smell. It is extremely difficult for us to make sense of a higher number of dimensions, a setting that seldom has to do with our day-to-day experience. This problem is further aggravated by the size of the datasets that are often analyzed in an exploratory data analysis settings. These datasets are not just high-dimensional, but may also contain millions of data points that ought to be analyzed. This work is motivated by the need for the development of scalable algorithmic solutions that enables the analysis of extremely large and high-dimensional data. We explore the intersection of visualization and machine learning techniques, while providing new algorithms and applications that are specifically designed to empower users during the analytical process.

1.2 Contribution and Outline

In this dissertation, we focus on a body of techniques for analyzing high-dimensional data that rely on dimensionality reduction. Dimensionality reduction techniques aim at reducing high-dimensional data in a low-dimensional space, i.e. two or three dimensional, that is easily visualized with traditional visualization techniques such as scatterplots. While the dimensionality is reduced and the information the data contains, some characteristic of the high-dimensional data are preserved. Depending on the characteristic that is preserved, different insights on the data are obtained. For example, linear-dimensionality reduction techniques preserve large pairwise distances between data points and give an intuition on the major trends in the data.

Recent years have seen the widespread adoption of new types of dimensionality reduction that have been proven to be beneficial in several analytical tasks [151]. Non-linear dimensionality reduction, also known as manifold learning techniques, aim at the discovery, preservation and visualization of non-linear structures of points. The development of these techniques is motivated by the “Manifold Assumption”, i.e., the idea that redundancy exists among the dimensions and the data lay on multiple non-linear manifolds that are embedded in the high-dimensional space. The manifold assumption has been empirically verified in many settings and it is at the core of many unsupervised learning algorithms [26].

While non-linear dimensionality reduction techniques allow for the discovery, visualization and analysis of the manifolds, they are usually costly to compute and do not scale well in the number of data points to be analyzed. This dissertation presents several techniques that improve the scalability of non-linear dimensionality-reduction algorithms, allowing to push the analytical capabilities to a whole new level. The proposed techniques power several application, that are also presented in this dissertation, that provide novel insights in several fields such as biomedical data analysis, deep neural network interpretability and social-network analysis.

More specifically, the contributions of this dissertation are as follows:

- In Chapter 4, we demonstrate that approximated computations of a widely used non-linear dimensionality-reduction algorithm, the t-distributed Stochastic Neighbor Embedding (tSNE), allows for a much scalable visual data analysis pipeline with negligible reduction in the quality of the generated embedding. Following this insight, we present the Approximated-tSNE [138] and we describe how it is used in a Progressive Visual Analytics (PVA) computational paradigm. PVA is a recent analytical approach that present the user with partial results of complex algorithms without waiting for their completion.

- In Chapter 5, we present a novel approach to the computation of the gradient descent of the tSNE algorithm [139]. Thanks to a reformulation of the gradient, our technique makes heavy use on the GPU rendering pipeline, speeding up computations by several orders of magnitude while, at the same time, is computed in the client side of a web browser.

- A novel hierarchical approach for the exploration of high-dimensional datasets; the Hierarchical Stochastic Neighbor Embedding (HSNE) [136] is presented in Chapter 6. HSNE creates a hierarchical representation of the data that is interactively explored by the user. During the exploration, clusters at different scales are revealed.
- The algorithm presented in this thesis are used in different applications developed for different fields. We present how our algorithms power several tools that support the exploratory analysis in biomedical research. In particular, in Chapter 7 we present how the HSNE algorithm was used in the Cytosplore application for the analysis of large single-cell datasets for new cell-type discovery [68, 73, 90, 102, 179].
- We introduce the “Who’s Acting On What-Visualization” (WAOW-Vis), a novel technique for the multiscale visual exploration of large bipartite graphs [135]. WAOW-Vis is developed with the specific goal of analyzing datasets of social-network scale, i.e. containing millions of users, and it is introduced in Chapter 8. We show how our technique allows to discover “filter bubbles” on Twitter, i.e., groups of users that follow only polarized source of information.
- In Chapter 9, we present DeepEyes [137], an analytical tool that permits a visual analysis of deep neural networks directly during training. DeepEyes makes use of our non-linear dimensionality-reduction techniques in order to highlight how networks behave with respect to their input. The insights obtained trough DeepEyes allow the user to make informed decisions about the design of the network.

In order to contextualize our work with regard to the existing literature, the next chapter presents the related work, introducing visual analytics techniques for large and high-dimensional data analysis, while Chapter 3 establishes the technical background of this work. The chapters from 4 to 9 present the contributions of the dissertation as stated above. Finally, Chapter 10 concludes the dissertation with an overview of the results achieved and reflections on future work.

2 Related Work

If I have seen further, it is by standing on the shoulders of giants.

Isaac Newton

In this chapter we present an overview of the research field to which this dissertation belongs. We introduce the reader with the concept of Exploratory Data Analysis and high-dimensional data analysis. Then we present visualization techniques for exploring and analyzing high-dimensional, with a focus on dimensionality-reduction algorithms and, finally, we introduce the concept of Progressive Visual Analytics. Other related work, that are more specific to the techniques and applications presented in this dissertation, will be discussed in each one of the following chapters.

2.1 Exploratory Data Analysis

When faced with novel data, the user performing an analysis does not have a clear picture of which model can be fitted on it. Therefore, a first analysis is usually performed to understand the main characteristics of the acquired dataset. This analysis takes the name of *Exploratory Data Analysis* and, among its goals, are the extraction of important variables, the detection of outliers or the identification of underlying non-convex structures [172]. By exploring the data, the user can form hypothesis on the underlying phenomenon that is at the base of the acquired data. This knowledge is then used to devise novel experiments or to define statistical models to fit and automatize the data analysis for a specific task at hand.

Due to its exploratory nature, the data is analyzed by the user without imposing much prior knowledge on the patterns that ought to be found in the data. For this reason, Exploratory Data Analysis heavily relies on a number of visualization techniques that are used to support the understanding of the data for a hypothesis-generation process. A simple, yet powerful, example of why it is important to perform an Exploratory Data Analysis of the data is given by the "Anscombe's Quartet" [8] which is presented in Figure 2.1. The quartet consists of four 2-dimensional datasets that have nearly identical descriptive statistics. The four datasets have similar mean and standard deviation on the x and y axis and they also have a similar correlation between the two variables, identified by the linear regression line drawn on the plot. However it is clear that, after visual inspection of the data, the descriptive statistics are not enough to reveal important trends captured by the data.

The two datasets in the top row do not contain outliers. However, while the dataset on the left has a noisy but linear relationships between the values in x_1 and y_1, the dataset on the right is characterized by an exact parabolic relationships between x_2 and y_2, a trend that would be unnoticed without a direct visual inspection of the data. Other interesting observations can be made on the remaining two datasets. More specifically, these two examples highlight how the presence of outliers, i.e., data points that are distant from the other observations, can derail the statistical analysis. While for the dataset at the bottom left the regression line describing the data is only marginally modified by the outlier, for the dataset on the right, a single data point can completely ruin the line fitted to the data.

The Anscombe's quartet is a great example that motivates the need for a qualitative understanding of the data in order to form hypothesis. These hypothesis are then validate through proper quantitative analysis with statistical techniques. However, data seldom comes in the simple form of a 2-dimensional dataset. In order to describe complex phenomena, a higher number of dimensions are required and, to this end, more advanced Exploratory Analysis techniques and visualizations are needed. In the remainder of this chapter, we present related work in the visual data analysis for hypothesis generation for high-dimensional data and how this dissertation provides novel techniques for the visual exploration of high-dimensional data.

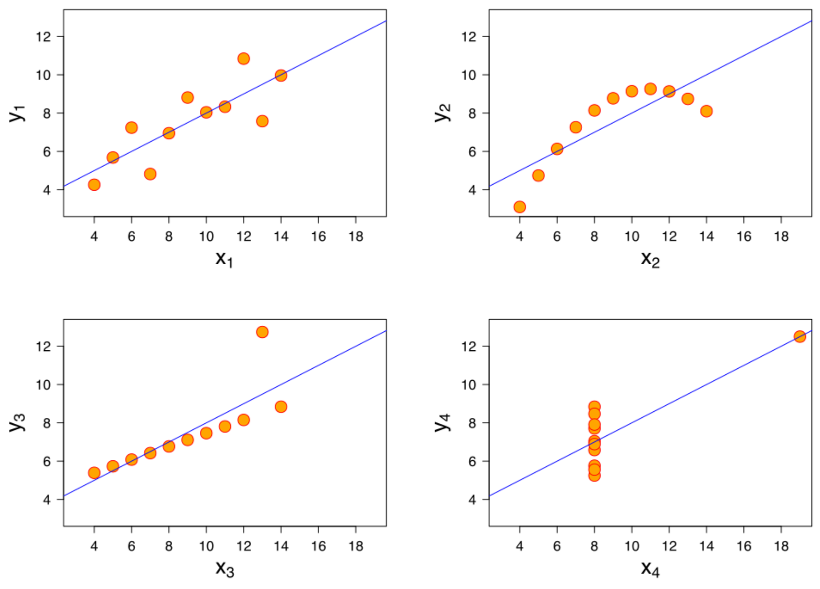


Figure 2.1: **The Anscombe's Quartet** consists of four 2-dimensional datasets with nearly identical descriptive statistics. However, upon visualization important characteristics and differences are revealed, i.e., the presence of outliers or non-linear relationships between the dimensions. The Anscombe's Quartet is the simplest and yet a clear example of the benefits of adopting an Exploratory Data Analysis approach to the understanding of data.

2.2 Visualization of High-Dimensional Data

In the previous section, we presented the motivation to adopt a visual inspection approach for data analysis. However, in a real-world setting data is described by many variables, i.e., dimensions, and a scatterplot visualization as presented in Figure 2.1 is not enough to reveal patterns in the data. Therefore, visualization techniques specifically designed for high-dimensional data analysis have been developed with the goal of analyzing a number of dimensions that is higher than 2- or 3-dimensions. In the remainder of this Section we introduce the most commonly used visualizations of high-dimensional data.

A familiar setting for displaying high-dimensional data is to organize it in a tabular form, where each reading, or data point is a row in the table. Each column of the table correspond to a dimension in the high-dimensional data. Microsoft's Excel or LibreOffice's Calc are just two examples of possible software that save the data in this form. However, without enriching the table with some visual feedback, it is in general impossible, if not for limited test cases, to find interesting insights by looking directly at the numbers in the table. A possible improvement is to enrich the table by a heatmap visualization. Here, the cells in the table are colored according to

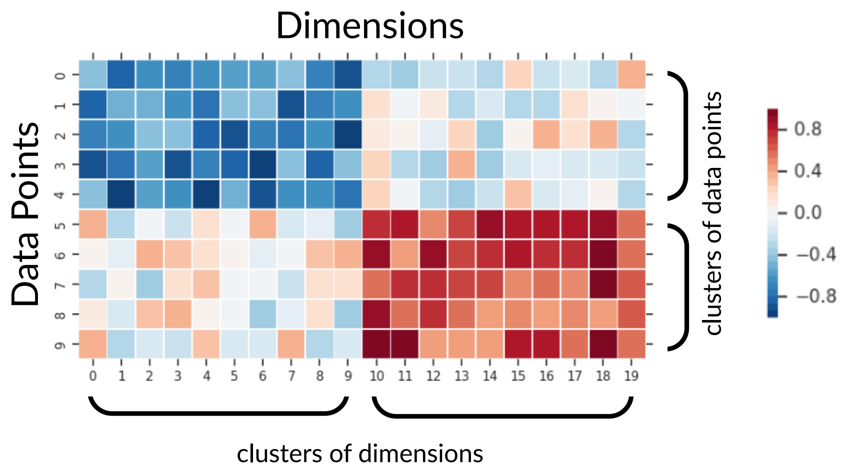
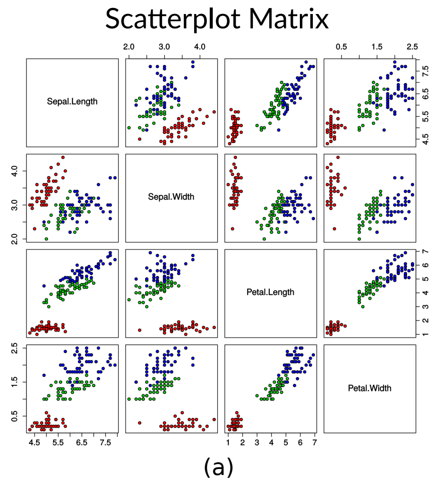


Figure 2.2: **Heatmap visualization** of high-dimensional data. Reordering of the data points and dimensions is used to show clusters of similar entities. Two clusters of data points, i.e., rows, are visible as they share low values for the first group of dimensions. At the same time, groups of similar dimensions, i.e., columns, are identified as share similar values in the dataset.

the value they contain allowing for a better identification of similar rows. However, in order to identify patterns of similar data points, the order of the table is of major importance [7,13,49,133,184]. It is indeed much easier to identify groups of similar points if those points are close together. Figure 2.2 shows an example of heatmap visualization for high-dimensional data with rearranged columns and rows. Two clusters of data points, i.e., rows, are visible as they share similar values in almost all dimensions. At the same time, groups of similar dimensions, i.e., columns, are identified. A heatmap scalability is, however, limited by the resolution of the screen. Furthermore, not all the relationships become easily identifiable in this encoding.

A scatterplot matrix, or SPloM, is an alternative visualization for high dimensional data [24,173]. Scatterplot matrices consists of all pairwise scatterplots organized in a matrix layout, where each scatterplot shows the relationships between a pair of dimensions. Figure 2.3a shows a scatterplot matrix for the Iris dataset, which is a 4-dimensional dataset containing three different classes of objects. While a SPloM scales better than a heatmap visualization in the number of data points visualized, it does not scale as well to a larger number of dimensions. As a matter of fact, by increasing the number of dimensions, the occupied visual space grows quadratically. Therefore, SPloMs are adequate for datasets containing less than, approximately, 30 dimensions [123]. Moreover, SPloMs require also a significant cognitive load from the user when relations beyond two values are of interest. This can be improved by making use of brushing and linked selections, for example by highlighting the selection of one scatterplot in all the others in the matrix.

Another widely used visualization for high-dimensional data analysis is the par-



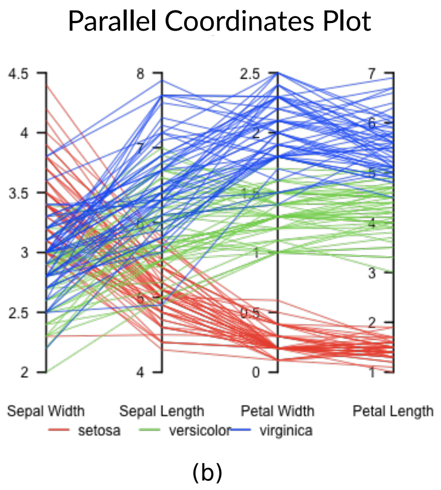


Figure 2.3: **Scatterplot Matrix and Parallel Coordinates Plot** of a 4-dimensional dataset. In a SPloM (a), 2-dimensional scatterplots are arranged in a grid. Each scatterplot shows the correlation between two dimensions. In the parallel coordinates plot (b) each dimension is represented by a vertical axis. Data point are polylines intersecting the vertical axes according to their values in each dimension. These visual representations do not scale well with the number of dimensions.

allal coordinates plot (PCP) [76]. In a parallel coordinates plot, each dimension is represented by a vertical axis. A data point is visualized by a polyline intersecting the axes according to the values of the point in the corresponding axis. PCPs allow for the detection of patterns in the data, where similar data points create similar line bundles. Obtaining these insights is facilitated by interactions [62,159] such as brushing [59,143] and reordering of the axis [7,133]. An example of a parallel coordinates plot for the Iris dataset is presented in Figure 2.3b. Parallel coordinates plots also become ineffective when the number of dimensions increases. The visual space needed to visualize more than a dozen of dimensions makes the generation of the plot infeasible on a computer screen. Furthermore, the trends can be identified just between neighboring elements, i.e., dimensions.

The visualizations presented in this section are indeed powerful tools for an exploratory data analysis task. However, when the dimensionality of the data is high, e.g., hundreds or thousands of dimensions, direct visual representations fail to highlight complex patterns in the data. In the remainder of this chapter, we present algorithmic solutions that can be adopted to extract these complex patterns from the data for a visualization purpose. These algorithmic solutions take the name of dimensionality-reduction techniques. They aim at reducing the dimensionality of the data to a number of dimensions that can be easily visualized, e.g., in a 2-dimensional scatterplot. Despite the inevitable loss of information due to the dimensionality reduction, these algorithms preserve some characteristic of the original data that is of user interest, hence enabling an effective data exploration.

2.3 Dimensionality-Reduction for Visualization

The visualizations presented in the previous overload the user with too much information if used to analyze very high-dimensional data. This information overload leads to two problems during the exploratory analysis. First, it translates to a cognitive overload for the user exploring the visualization. The more information is presented to the user, the more difficult is to effectively discover patterns in the data and finding the underlying rules governing the phenomenon [123]. Second, the visualization of all the dimensions for every data point may be infeasible due to technical limitations, e.g., due to the limited number of pixels on screen or to the amount of clutter in the resulting visualization.

Dimensionality-reduction techniques adopt a different approach for the analysis of high-dimensional data. Instead of the direct visualization of the dataset, they aim at finding a low-dimensional representation that preserves some important characteristic of the data. This low-dimensional representation, also called embedding, is then visualized and analyzed by the user. We define a high-dimensional data as $X = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$, N being the number of data points $\mathbf{x}_i \in \mathbb{R}^h$ residing in a h-dimensional space. Dimensionality-reduction techniques find a mapping function $DR : \mathbb{R}^h \Rightarrow \mathbb{R}^l$ that embeds the high-dimensional points in an l-dimensional space, where l in visualization is usually chosen to be 2 or 3. By applying the mapping function to the original dataset:

$$\forall x_i \in X : \mathbf{y}_i = DR(\mathbf{x}_i) \tag{2.1}$$

The mapped points are collected in a derived dataset $Y = \{\mathbf{y}_1 \dots \mathbf{y}_N\}$ which is usually referred as embedding. Since $\mathbf{y}_i \in \mathbb{R}^2$ or $\mathbb{R}^3$, known visualization techniques such as scatterplots are used to effectively analyze the embedding Y. How the mapping function DR is defined is crucial for the correct understanding of the data during the exploratory phase. As a matter of fact, since the user explores the dataset in, for example, a 2D scatterplot, the understanding of the phenomenon is mediated by the the mapping generated by DR. Moreover, the creation of the mapping must not only be informative, but also computationally feasible due to the size and dimensionality of the data at hand.

A dimensionality reduction technique that is extensively used is the Principal Component Analysis (PCA) [81]. PCA aims at finding a orthogonal linear transformation of the data such that the greatest variance in the data is explained by the first coordinates in the transformed space. PCA defines a square transformation matrix W that, when multiplied to an element in X, expresses this point in a new orthogonal basis, whose axes are ordered by decreasing variance with respect to the original dataset. By taking only the first columns of W, we create a dimensionality-reduction transformation W' that, when multiplied to the data point $\mathbf{x}_i$

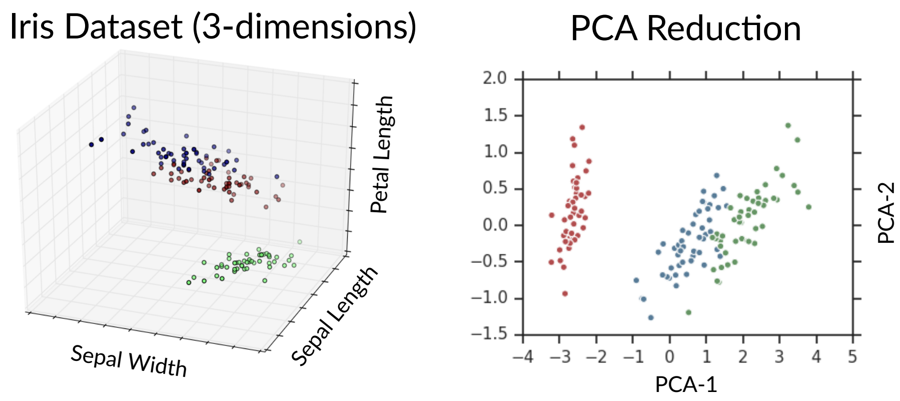


Figure 2.4: **Principal Component Analysis** of the Iris dataset presented in Figure 2.3. On the left, 3 dimensions of the dataset are used to create a 3D scatterplot. On the right, the first two principal components are used to visualize the data on a 2D scatterplot. On the first dimension, PCA-1, the dataset presents the highest variance.

$$\forall x_i \in X : \mathbf{y}_i = \mathbf{x}_i W', \tag{2.2}$$

where $\mathbf{y}_i$ is the low dimensional representation of $\mathbf{x}_i$. The dimensionality of the resulting dataset Y correspond to the number of columns chosen to be in W'. For visualization purposes, the dimensionality of Y is usually 2 or 3, hence enabling the visualization of the data in a scatterplot. An example of a PCA transformation for a 3D dataset to a 2D representation is shown in Figure 2.4.

The principal components are obtained by a Single Value Decomposition (SVD) of the covariance matrix of the dataset [81]. This leads to a computational complexity of the dimensionality reduction of $O(h^2N + h^3)$, where h is the number of dimensions and N is the number of data points and it has a memory complexity of $O(N^2)$ due to the need of storing the covariance matrix. While the approach scales linearly in the number of data points, increasing the dimensionality of the dataset makes the exploratory analysis of the data prohibitive, or even impossible, due to the computation time.

A different approach that, instead of focusing on the variance of the data is focused on the distances between points is the Classic, or Metric Multidimensional Scaling (MDS) [18]. In the multidimensional scaling the mapping function DR is chosen to preserve in least-square sense the pairwise distances between the data points. Therefore, a pairwise distance matrix D is computed from the points in the dataset and a linear transformation of the dataset is computed such as the distances between the points in Y reflects as closely as possible the distances in D. Note that, if the distance metric chosen to populate D is L^2, then MDS is equivalent to a PCA reduction.

The computational and memory complexity of metric MDS is $O(N^3)$ since involves a singular value decomposition of a matrix derived from D. Since the complexity of the technique is prohibitive for large datasets, several techniques have been developed in the past in order to be able to deal with larger datasets. Silva et al. [33] introduced the Landmark-MDS, where the dimensionality reduction is performed only on a subset of points that are called landmarks. The location of the rest of the points in the embedding is then obtained through a linear interpolation of the position of the landmarks in the embedding. Ingram et al. [75] propose Glimmer, a multilevel MDS approach that uses multiple level of landmark to guide the creation of the embedding.

In this section, we presented the generic framework for visualizing high dimensional data through dimensionality reduction. The introduced techniques, namely the Principal Component Analysis and the Metric Multidimensional Scaling are characterized by a linear mapping function DR. This characteristic impose a transformation a global transformation on that is shared by each point in X. In the next section we introduce a different set of techniques known as non-linear dimensionality-reduction, or manifold learning, where DR is a non-linear mapping and the transformation is local in nature.

2.4 Non-Linear Dimensionality-Reduction for Data Visualization

In recent years, a better understanding of high-dimensional data obtained from real world phenomena, lead to the formulation of the so-called "Manifold Hypothesis". The manifold hypothesis states that high-dimensional data often lay in low dimensional manifolds embedded in the high-dimensional space at hand. In this context, it is more interesting to understand the local characteristics of the manifolds than achieving a global mapping of the data introduced by a linear transformation as presented in the previous section. Examples of the insights that we aim at obtaining are the number of disconnected manifolds in the data, their interrelationships and their local dimensionality, also known as intrinsic dimensionality.

Non-linear dimensionality-reduction techniques, also known as manifold learning, have been developed in recent years with the focus on the unsupervised discovery and analysis of manifolds in high-dimensional data. The mapping function DR obtained from these techniques is characterized by a non-linear mapping with local properties. More specifically, the mapping DR behaves differently in different regions of the high dimensional space. Whereas linear-techniques imposes a global transformation to the data, non-linear techniques often adopt a bottom-up approach, where the mapping is defined by fitting a local model on each data point in X.

The Sammon-Mapping techique [152], introduced in 1969, is the first example of non-linear mapping, i.e., where the resulting dimensions of the embedding have no relationships with the original dimensions of the dataset X. The Sammon-Mapping minimizes the Sammon's stress function, a measure of the mismatch between the localized distances between pairs of points in X and in the embedding Y. The mapping is created by gradient descent minimization or other iterative minimization techniques. Despite the approach chosen for the minimization, the embedding is created by randomly placing the points in the embedding Y which are then moved in the low dimensional space to minimize the chosen cost function.

Isomap [169] treats the high-dimensional data as a graph. Data points are considered vertices in the graph which are connected to only a subset of neighbors, where the edges are weighted by the euclidean distances between the corresponding points. The shortest pairwise distances between all the points are then computed using the Floyd-Warshall algorithm [154]. The resulting distance matrix encodes the geodesic length between the points, i.e., the distance that are traversed from one point to another while remaining on the data manifold. The points are then embedded by preserving the distances with a Multidimensional Scaling approach, where the distances are the geodesic length.

The Locally-linear embedding (LLE) [150] also relies on the search of a set of neighboring points for each point in X. Each point is expressed as a linear combination of its neighbors in the high-dimensional space. LLE then applies an eigenvector-based optimization technique that aims at creating a low-dimensional embedding where the linear-combinations are also preserved. LLE has the advantage over Isomap of creating a sparse problem, not requiring to compute a full distance matrix.

Stochastic Neighbor Embedding (SNE) [66] is a non-linear dimensionality reduction technique that encodes local similarities between points in a stochastic fashion. For each point a Gaussian kernel is found in such a way that only a small number of neighbors are covered. The Gaussian kernel encodes, for each point in X, the probability that another point is close to it on the manifold. Points are randomly placed in the low-dimensional embedding Y and the same computation of the similarities is performed. Points in the embedding are optimized with a gradient descent technique that minimizes the divergence between the corresponding probability distributions in the high-dimensional space and the embedding.

Several other techniques have been introduced over the years such as Laplacian Eigenmaps [14], Diffusion Maps [91] and non-linear PCA [153]. However, van der Maaten et al. observed in a comparative review [178] that, while non-linear techniques perform well on selected artificial datasets, the good performance does not necessarily extend to real-world data. In particular, the presented techniques suffer from the *crowding-effect*, i.e., the inability to disentangle manifolds that are often intermixed in the resulting embedding. To this end, van der Maaten and Hinton introduced the t-Distributed Stochastic Neighbor Embedding (tSNE), an evolution of the SNE algorithm which overcomes the crowding-effect while, at the same time, it is easier to optimize. tSNE [176,177] has been accepted as the state of the art for non-linear dimensionality reduction applied to visual analysis of high-dimensional space in several application areas, such as life sciences [6,12,73,90,102,107,157] and machine learning model understanding and human-driven supervision [83,116, 137]. This dissertation heavily relies on the tSNE algorithm, for which a detailed description is provided in the next chapter, and provides several new techniques that improve on the scalability and insightfulness of the embeddings. The presented techniques are general and are already used in novel non-linear dimensionality-reduction algorithms such as LargeVis [168], UMAP [112] and TriMap [5].

Finally, it is worth mentioning that the preservation of local and non-linear properties of the mapping does not come without a price. Contrary to linear dimensionality reductions, where the new axes are a linear combination of the original dimensions, in non-linear techniques the axis cannot be interpreted by the user. To improve the interpretation of the embeddings, visual analytics systems have been developed to visualize and validate the resulting embeddings [68, 109, 144]. These systems allow us, due to linked visualizations, to understand which dimensions are responsible for the patterns that are visible in the embedding.

2.5 Progressive Algorithms for Interactive Systems

In the previous sections we introduced dimensionality-reduction techniques for exploratory data analysis. Among dimensionality-reduction techniques, non-linear algorithms are at the core of several discoveries, for example, in life sciences [6, 12, 107, 157]. The main advantage of using this approach for exploratory data analysis is that they make only limited assumptions on the data at hand, e.g., the presence of relatively low-dimensional manifolds. Therefore, the user can explore the data and obtain insights that are then validated by experiments, or are used to create automatic data-processing tools.

However, despite the advantages introduced by this new data analysis approach, dimensionality reduction, and non-linear techniques in particular, are characterized by high computational complexity that limits their application for interactive tools. Depending on the size of the data to be analyzed, it may take hours, or even days, before an embedding is computed and ready to be analyzed by the user. While this waiting time may be acceptable for many applications, it is a major obstacle for introducing dimensionality-reduction techniques in interactive visual analytics tools.

In recent years, a novel computational paradigm has been introduced to improve the interactivity of visual analytics systems that rely on complex and time costly algorithms. This paradigm, which takes the name of Progressive Visual Analytics (PVA), aim at the visualization and analysis of incrementally better partial results. The term Progressive Visual Analytics was introduced by Stolper et al. [165] together with a list of requirements. More specifically, visual analytics systems should be designed to:

- Provide increasingly meaningful partial results during the execution of the algorithms.
- Allow the user to focus the computations on a subspace of interest [122].
- Allow users to ignore irrelevant subspaces.

Moreover, visualizations used within these systems must be designed with the following properties in mind:

- Minimize user distraction with abrupt changes.
- Guide the user by providing cues on the subspace of the data that contains new insights.
- Support an on-demand refresh of the visualizations.
- Provide interfaces to specify on which subspaces the algorithm must focus.

An early examples of the application of PVA in visual analytics systems is *sampleAction* presented by Fisher et al. [42]. SampleAction performs simple database queries on extremely large databases that are refined over time. Mühlbacher [119] provided a list of more advanced data mining algorithms that support the Progressive Visual Analytics paradigm, while advocating for a more strict collaboration between algorithm and visualization researchers. Finally, Fekete and Primet [41] formalize the concept of progressive computations and present *ProgressiVis*, a toolkit that enables the implementation of algorithms in a natively progressive environment.

In this dissertation, we present novel non-linear dimensionality-reduction techniques that fully embrace the Progressive Visual Analytics paradigm. This novel approach enabled the development of analytical systems, such as Cytosplore [68], DeepEyes [137] and WAOW-Vis [135], that make use of dimensionality reduction for the analysis of the data in a fully interactive setting.

3 Background

But in my opinion, all things in nature occur mathematically.

René Descartes

In this chapter, we provide the reader with an in-depth description of the mathematical background needed to understand the contributions presented in the following chapters. More specifically, we introduce the t-Distributed Stochastic Neighbor Embedding and the Barnes-Hut-SNE algorithms. Moreover, we present the MNIST dataset, a widely used benchmark for validating dimensionality-reduction techniques.

3.1 t-distributed Stochastic Neighbor Embedding

As presented in the previous chapter, visual analysis of high dimensional data is a challenging process. Direct visualizations such as parallel coordinates [76] or scatterplot matrices [58] work well for a few dimensions but do not scale to hundreds or thousands of dimensions. Typically indirect visualization is used for these cases. First the dimensionality of the data is reduced, usually to two or three dimensions, then the remaining dimensions are used to lay out the data for visual inspection, for example in a two dimensional scatterplot. A variant of tSNE [177], the Barnes-Hut SNE [176] has been accepted as the state of the art for non-linear dimensionality reduction applied to visual analysis of high-dimensional space in several application areas, such as life sciences [6,12,107,157]. tSNE is a non-linear dimensionality reduction algorithm that aims at the preservation of local neighborhoods during the dimensionality reduction.

In this section, we provide an introduction to tSNE [177], which is at the base of several contributions presented in this dissertations. tSNE interprets the overall distances between data-points in the high-dimensional space as a symmetric joint-probability distribution P. Likewise a joint-probability distribution Q is computed, that describes the similarity in the low-dimensional space. The goal is to achieve a representation, referred to as embedding, in the low-dimensional space where Q faithfully represents P. This is achieved by optimizing the positions in the low-dimensional space to minimize the cost function C given by the Kullback-Leibler (KL) divergence between the joint-probability distributions P and Q:

$$C(P,Q) = KL(P||Q) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} p_{ij} \ln\left(\frac{p_{ij}}{q_{ij}}\right) \tag{3.1}$$

Given two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the dataset $X = \{\mathbf{x}_1 ... \mathbf{x}_N\}$, p_{ij} models the probability of finding the two points in close vicinity in the high-dimensional space. To this extent, for each point a Gaussian kernel, P_i, is chosen whose variance σ_i is defined according to the local density in the high-dimensional space and then p_{ij} is described as follows:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}, \tag{3.2}$$

$$\text{where} \quad p_{j|i} = \frac{\exp(-(||\mathbf{x}_i - \mathbf{x}_j||^2)/(2\sigma_i^2))}{\sum_{k\neq i}^{N} \exp(-(||\mathbf{x}_i - \mathbf{x}_k||^2)/(2\sigma_i^2))} \tag{3.3}$$

$p_{j|i}$ can be seen as a relative measure of similarity based on the local neighborhood of a data-point $\mathbf{x}_i$. Similarly, $p_{i|j}$ is a measure of similarity based on the data point $\mathbf{x}_j$. The perplexity value μ is a user-defined parameter that describes the effective number of neighbors considered for each data-point. The value of σ_i is chosen such that for fixed μ and each i:

$$\mu = 2^{-\sum_j^N p_{j|i} \log_2 p_{j|i}} \tag{3.4}$$

A *Student's t-Distribution* with one degree of freedom is used to compute the joint-probability distribution in the low-dimensional space Q, where the positions of the data-points should be optimized. Given two low-dimensional points $\mathbf{y}_i$ and $\mathbf{y}_j$, the probability q_{ij} that describes their similarity is given by:

$$q_{ij} = \left((1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)Z\right)^{-1} \tag{3.5}$$

$$\text{with} \quad Z = \sum_{k=1}^{N} \sum_{l \neq k}^{N} (1 + ||\mathbf{y}_k - \mathbf{y}_l||^2)^{-1} \tag{3.6}$$

The gradient of the Kullback-Leibler divergence between P and Q is used to minimize C (see Equation 3.1). It indicates the change in position of the low-dimensional points for each step of the gradient descent and is given by:

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4\left(F_i^{\text{attr}} - F_i^{\text{rep}}\right) \tag{3.7}$$

$$= 4\left(\sum_{j \neq i}^{N} p_{ij} q_{ij} Z(\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i}^{N} q_{ij}^2 Z(\mathbf{y}_i - \mathbf{y}_j)\right) \tag{3.8}$$

The gradient descent can be seen as a *N-body simulation* [1], where each data-point exerts an attractive and a repulsive force on all the other points (F_i^{attr} and F_i^{rep}). The computational and memory complexity of the tSNE algorithm is $O(N^2)$, where N is the number of points in the dataset. The algorithms computes, for each point, the forces exerted on it by all the other points in the dataset, hence limiting its application to datasets containing less than a thousand points. In the next section we introduce the Barnes-Hut-SNE algorithm, a technique that is designed to scale the tSNE computation to tens of thousands points.

3.2 Barnes-Hut Stochastic Neighbor Embedding

The Barnes-Hut-SNE (BH-SNE) [176] is an evolution of the tSNE algorithm that introduces two different approximations to reduce the computational complexity to $O(N\log(N))$ and the memory complexity to $O(N)$.

The first approximation aims at scaling the computation of the joint-probability distribution P. It is based on the observation that the probability p_{ij} is infinitesimal if $\mathbf{x}_i$ and $\mathbf{x}_j$ are dissimilar. Therefore, the similarities of a data-point $\mathbf{x}_i$ can be computed taking into account only the points that belong to the set of nearest neighbors $\mathcal{N}_i$ in the high-dimensional space. The cardinality of $\mathcal{N}_i$ can be set to $K = \lfloor 3\mu \rfloor$, where μ is the user-selected perplexity and $\lfloor \cdot \rfloor$ describes a rounding to the next-lower integer. Without compromising the quality of the embedding [176], we can adopt a sparse approximation of the high-dimensional similarities. Equation 3.3 can now be written as follows:

$$p_{j|i} = \begin{cases} \frac{\exp(-(||\mathbf{x}_i - \mathbf{x}_j||^2)/(2\sigma_i^2))}{\sum_{k\in\mathcal{N}_i} \exp(-(||\mathbf{x}_i - \mathbf{x}_k||^2)/(2\sigma_i^2))} & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The computation of the K-Nearest Neighbors is performed using a Vantage-Point Tree (VP-Tree) [190]. A VP-Tree is data structure that computes KNN queries in a high-dimensional metric space, in $O(\log(N))$ time for each data point. Therefore, the complexity of the computation of the joint-probability distribution P becomes $O(N\log(N))$, since a KNN query is computed for each point in the dataset. It is a binary tree that stores, for each non leaf-node, a hyper-sphere centered on a data-point. The left children of each node contains the points that reside inside the hyper-sphere, whereas the right one contains the points outside it.

The second approximation aims at scaling the computation of the optimization of the tSNE cost function, presented in Equation 3.1, and it makes use of the formulation of its gradient as presented in Equation 3.7. As described in the previous section, tSNE can be seen as a N-body simulation, where attractive and repulsive forces are applied on each point based on their high-dimensional similarity. The Barnes-Hut algorithm [10] is used to speed up N-body simulation problems by jointly computing the effects of clusters of distant points. This optimization makes use of a tree structure and reduces the computational complexity of the tSNE optimization to $O(N\log(N))$. For further details, please refer to van der Maaten [176].

3.3 The MNIST Dataset

To validate the embeddings generated by the tSNE algorithm and the novel techniques introduced in this dissertation, a number of datasets are used. Among these datasets, the MNIST dataset is often used as benchmark to validate novel non-linear dimensionality-reduction techniques. In this section we introduce the dataset and we explain why it is considered a good benchmark for non-linear dimensionality reduction techniques. The MNIST dataset is a collection of 70 thousands images of handwritten digits. The images were obtained by scanning documents created at the American Census Bureau and documents obtained from American high school students [95]. The images, for which few examples are presented in Figure 3.1, are saved in a grayscale format and have a resolution of 28x28 pixels. For each image, the corresponding label, i.e., the associated digit, is known. The dataset was widely used, in particular during the first decade of this century, for training and testing machine learning models with the goal of identifying the label associated to an image [95, 140]. In order to train a model, the images are separated in two groups, 60 thousand images form the so called training set, i.e., a collection of images on which machine learning models are trained to perform the classification. The remaining 10 thousand images are used to test the performance of the training model, hence they are part of the so called test set.

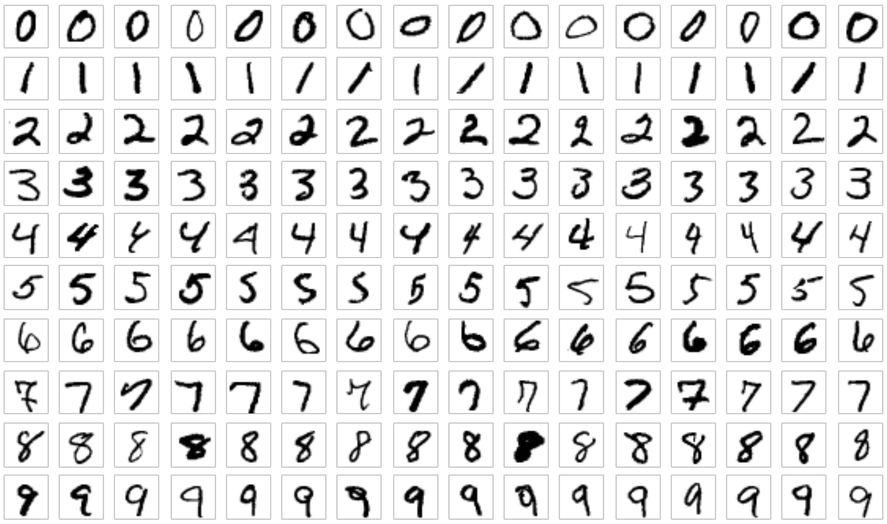


Figure 3.1: **Examples of the MNIST dataset**. The dataset contains images of handwritten digits.

The reader may now wonder what is the relationship between the images presented in Figure 3.1 and high-dimensional data that is at the core of this dissertation. As a matter of fact, we can treat each single image as a high-dimensional point; each dimension correspond to a pixel in the image and the corresponding value is given by the grayscale value in the pixel of interest. The resulting dataset has therefore 728 dimensions and 60 thousand data points for the training set and the 10 thousand images for the test set. The MNIST dataset is particularly well suited to test visual analysis techniques for high-dimensional data due to the large number of dimensions and data points. Moreover, it allows the validation the obtained insights as we have a clear understanding of the phenomenon behind the data. More specifically, we expect to find 10 distinct manifolds, each one corresponding to a different digit. Figure 3.2 presents a tSNE embedding of the MNIST dataset, where each image is drawn in the corresponding location in the embedding.

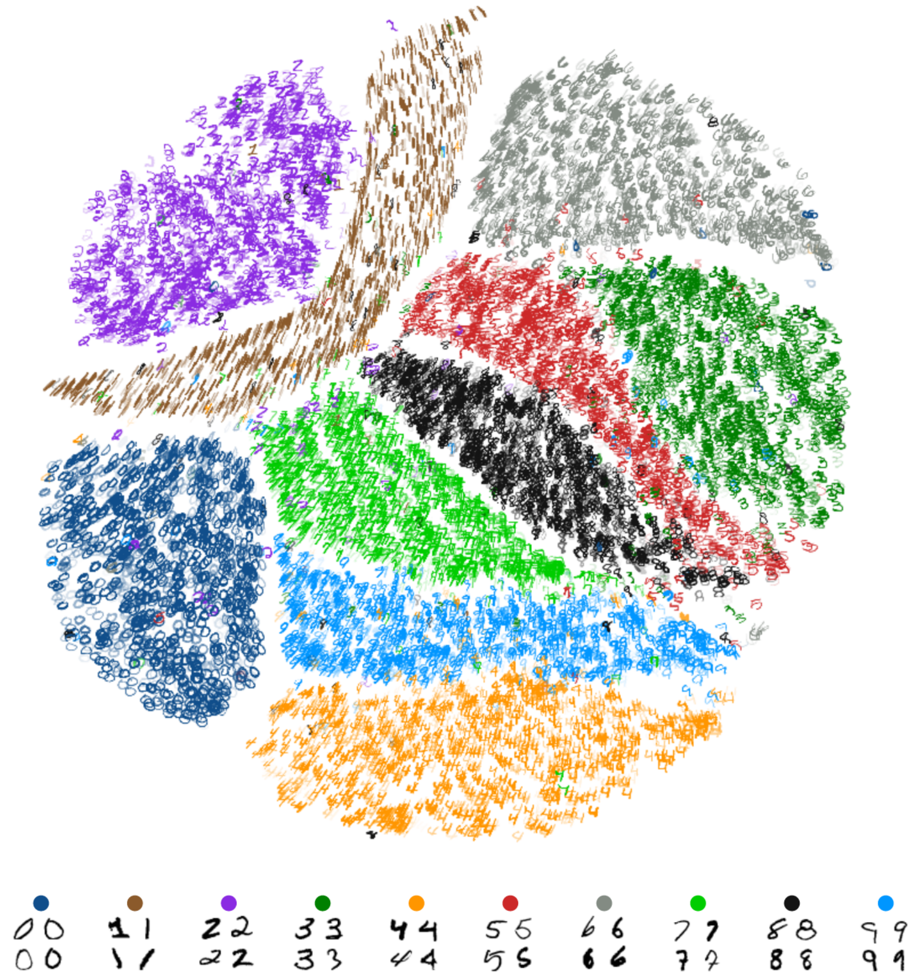


Figure 3.2: **tSNE embedding of the MNIST dataset**. The embedded data points are visualized as the MNIST images colored according to the digit they represent. Ten manifolds, one for each digit, are visible.

4

Approximated and User-Steerable tSNE for Progressive Visual Analytics

Science is the belief in the ignorance of experts.

Richard Feynman

In this chapter we present the Approximated-tSNE, an evolution of the tSNE algorithm that improves the computation time of a tSNE embedding by adopting approximated computations of the k-nearest-neighbor queries. Approximated-tSNE is particularly useful in progressive visual analytics applications, a claim that is validated by two use cases presented in this chapter.

N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. IEEE Transactions on Visualization and Computer Graphics, 23(7):1739-1752, 2017 [138].

4.1 Introduction

In Chapters 2 and 3 we introduced the tSNE algorithm [177] and we explained why it is beneficial for the analysis of high-dimensional dataset. tSNE produces 2D and 3D embeddings that are meant to preserve local structure in the high-dimensional data. The analyst inspects the embeddings with the goal to identify clusters or patterns that are used to generate new hypothesis on the data, however, the computational complexity of this technique does not allow direct employment in interactive systems. This limitation makes the analytic process a time consuming task that can take hours, or even days, to adjust the parameters and generate the right embedding to be analyzed.

In Chapter 2 we also introduced Progressive Visual Analytics. In Progressive Visual Analytics the user is provided with meaningful intermediate results in case computation of the final result is too costly. Based on these intermediate results the user can start the analysis process without waiting for algorithm completion. Mühlbacher et al. [119] provided a set of requirements, which an algorithm needs to fulfill in order to be suitable for Progressive Visual Analytics. Based on these requirements they analyze a series of different algorithms, commonly deployed in visual analytics systems and conclude that, for example, tSNE fulfills all requirements. The reason being that the minimization in tSNE builds up on the iterative gradient descent technique [177] and can therefore be used directly for a per-iteration visualization, as well as interaction with the intermediate results. However, Mühlbacher et al. ignore the fact that the distances in the high-dimensional space need to be precomputed to start the minimization process. In fact this initialization process is dominating the overall performance of tSNE for relatively high-dimensional spaces. Even with a per-iteration visualization of the intermediate results [27, 119, 165] the initialization time will force the user to wait minutes, or even hours, before the first intermediate result can be generated on a state-of-the-art desktop computer. Every modification of the data, for example, the addition of data-points or a change in the high-dimensional space, will force the user to wait for the full reinitialization of the algorithm.

In this chapter, we present A-tSNE, a novel approach to adapt the complete tSNE pipeline, including a distance computation for the Progressive Visual Analytics paradigm. Instead of precomputing precise distances, we propose to approximate the distances using Approximated K-Nearest Neighborhood queries. This allows us to start the computation of the iterative minimization nearly instantly after loading the data. Based on the intermediate results of the tSNE, the user can now start the interpretation process of the data immediately. Further, we modified the gradient descent of tSNE such that it allows for the incorporation of updated data during the iterative process. This change allows us to continuously refine the approximated neighborhoods in the background, triggering updates of the embedding without restarting the optimization. Eventually, this process arrives at the precise solution. Furthermore, we allow the user to steer the level of approximation by selecting points of interest, such as clusters, which appear in the very early stages of the optimization and enable an interactive exploration of the high-dimensional data.

More specifically, the contributions of this chapter are as follows:

1. We present A-tSNE, a twofold evolution of the tSNE algorithm, which
   (a) minimizes initialization time and as such enables immediate inspection of preliminary computation results.
   (b) allows for interactive modification, removal or addition of high-dimensional data, without disrupting the visual analysis process.
2. Using a set of standard benchmark data sets, we show large computational performance improvements of A-tSNE compared to the state of the art while maintaining high precision.
3. We developed an interactive system for the visual analysis of high dimensional data, allowing the user to inspect and steer the level of approximation. Finally, we illustrate the benefits of exploratory possibilities in a real-world research scenario and for the real-time analysis of high-dimensional streams.

4.2 Related work

The tSNE [177] algorithm builds the foundation of this work, which is used for visualization of high-dimensional data in a wide field of applications, from life sciences to the analysis of deep-learning algorithms [6,12,44,53,107,117,157]. tSNE is a non-linear dimensionality-reduction algorithm that aims at preserving local structures in the embedding, whilst showing global information, such as the presence of clusters at several scales. A detailed description of tSNE is presented in Section 3.1. Most of the user tasks associated with the visualization of high-dimensional data embeddings are based on identifying relationships between data points. Typical tasks comprises the identification of visual clusters and their verification based on detail visualization of the high-dimensional data, e.g., using parallel coordinate plots. For a complete description of such tasks we refer to Brehmer et al. [21].

As presented in Chapter 3, tSNE's computational and memory complexity is $O(N^2)$, where N is the number of data-points, which constrains the application of the technique. An evolution of the algorithm, called Barnes-Hut-SNE (BH-SNE) [176], reduces the computational complexity to $O(N\log(N))$ and the memory complexity to $O(N)$. This approach was also developed in parallel by Yang et al. [189]. However, despite the increased performance, it still cannot be used to interactively explore the data in a desktop environment.

Interactive performance is at the center of the latest developments in Visual Analytics. New analytical tools and algorithms, which are able to trade accuracy for speed and offer the possibility to interactively refine results [40, 42], are needed to deal with the scalability issues of existing analytics algorithms like tSNE. Mühlbacher et al. [119] defined different strategies to increase the user involvement in existing algorithms. They provide an in-depth analysis on how the interconnection between the visualization and the analytic modules can be achieved. Stolper et al. [165] defined the term *Progressive Visual Analytics*, describing techniques that allow the analyst to directly interact with the analytics process. Visualization of intermediate results is used to help the user, for example, to find optimal parameter settings or filter the data [165]. Many algorithms are not suited right away for Progressive Visual Analytics since the production of intermediate results is computationally too intensive or they do not generate useful intermediate results at all. tSNE is an example of such an algorithm because of its initialization process.

To overcome this problem, we propose to compute an approximation of tSNE's initialization stage, followed by a user steerable [122] refinement of the level of approximation. To compute the conditional probabilities needed by BH-SNE, a K-Nearest Neighborhood (KNN) search must be evaluated for each point in the high-dimensional space. Under these conditions, a traditional algorithm and data structure, such as a KD-Tree [43], will not perform well. In the BH-SNE [176] algorithm, a Vantage-Point Tree [190] is used for the KNN search, but it is slow to query when the dimensionality of the data is high. In this work, we propose to use an approximated computation of the KNN in the initialization stage to start the analysis as soon as possible. The level of approximation is then refined on the fly during the analytics process.

Other dimensionality-reduction algorithms implement approximation and steerability to increase performance as well. For example MDSteer [186] works on a subset of the data and allows the user to control the insertion of points by selecting areas in the reduced space. Yang et al. [188] present a dimensionality-reduction technique using a dissimilarity matrix as input. By means of a divide-and-conquer approach, the computational complexity of the algorithm is reduced. Other techniques provide steerability by means of guiding the dimensionality reduction via user input. Joja et al. [80] and Paulovich et al. [131] let the user place a small number of control points. In other work, Paulovich et al. [129], propose the use of a non-linear dimensionality-reduction algorithm on a small number of automatically-selected control points. For these techniques the position of the data points is finally obtained by linear-interpolation schemes that make use of the control points. However, they all limit the non-linear dimensionality reduction to a subset of the dataset limiting the insights that can be obtained from the data. In this work, we provide a way to directly use the complete data allowing the analyst to immediately start the analysis on all data points.

Ingram and Munzner's Q-SNE [74] is based on a similar idea as our approach, using Approximated KNN queries for the computation of the high-dimensional similarities. However, they use the APQ algorithm [74] that is designed to exploit the sparse structure of high-dimensional spaces obtained from document collections, limiting its application to such a context. A-tSNE improves Q-SNE in the direction of providing a fast but approximated algorithm for the analysis of traditional dense high-dimensional spaces. For this reason it can be used right away in contexts where BH-SNE is applied and Q-SNE would not be applicable. A further distinction is that A-tSNE incorporates the principles of the Progressive Visual Analytics by means of providing a visualization of the level of approximation, the ability to refine the approximation based on user input, and allowing the manipulation of the high-dimensional data without waiting for the recomputation of the exact similarities.

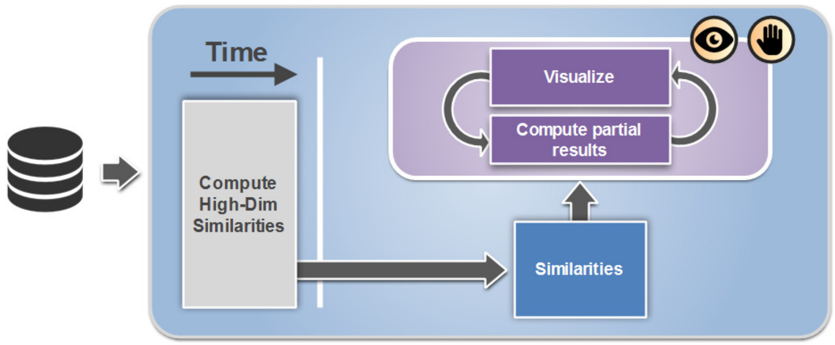
Density-based visualization of the tSNE embedding has been used in several works [6, 157, 176], however, they employ slow-to-compute offline techniques. In our work, we integrate real-time Kernel Density Estimation (KDE) as described by Lampe and Hauser [92]. The interaction with the embedding is important to allow the analyst to explore the high-dimensional data. Selection operations in the embedding and the visualization of the data in a coordinated multiple-view system are necessary to enable this exploration. The iVisClassifier system [28] is an example of such a solution. In our work, we take a similar approach, providing a coordinated multiple-view framework for the visualization of a selection in the embedding.

4.3 Approximated-tSNE in Progressive Visual Analytics

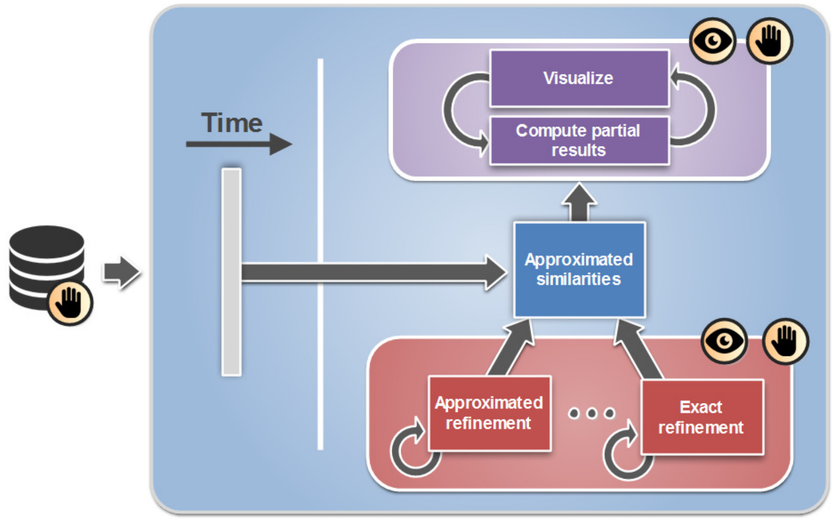
We now introduce Approximated-tSNE (A-tSNE), an evolution of the BH-SNE algorithm, using approximated computations of high-dimensional similarities to generate meaningful intermediate results. The level of approximation can be defined by the user to allow control on the trade off between speed and quality. The level of approximation can be refined by the analyst in interesting regions of the embedding, making A-tSNE a computational steerable algorithm [122]. tSNE is well suited for the application in Progressive Visual Analytics: after the initialization of the algorithm, the intermediate results generated during the iterative optimization process can be interpreted by the analyst while they change over time, as shown in previous work [27,119]. Figure 4.1a shows a typical Progressive Visual Analytics workflow for tSNE.

Algorithms that can be used in a Progressive Visual Analytics system often have a computational module, e.g. the initialization of the technique, that cannot be implemented in an iterative way, creating a *speed bump* [165] in the user analysis. tSNE is a good example for such an algorithm. It consists of two computational modules that are serialized. In the first part of the algorithm, similarities between high-dimensional points are calculated. In the second module, a minimization of the cost function (Equation 3.1) is computed by means of a gradient descent. The first module, depicted in light grey in Figure 4.1a, is slow to compute and does not create any meaningful intermediate results.

We extend the Progressive Visual Analytics paradigm by introducing approximated computation rather than aiming at exact computations, in the modules that are not suited for a per-iteration visualization. Figure 4.1b shows the analytical workflow for A-tSNE. While the generation and the inspection of the intermediate results



(a) Progressive Visual Analytics workflow for tSNE.



(b) Progressive Visual Analytics workflow for A-tSNE.

Figure 4.1: **Comparison between the traditional and our tSNE workflow.** The eye icon marks modules which produce output for visualization, whereas the hand icon marks modules that allow manipulation by the user. The increased performance of the similarity computation allows the user to seamlessly manipulate the input data. The level of approximation can be visualized and the user can steer the refinement process to interesting regions.

is not changed, we introduce a refinement module, depicted in red in Figure 4.1b, which can be used to refine the level of the approximation in the embedding in a concurrent way. Furthermore, the increased performance of the initialization module and the ability to update the high-dimensional similarities during the gradient descent minimization, allows the analyst to manipulate the high-dimensional data without waiting for the reinitialization of the algorithm.

We follow the guideline proposed by Stolper et al. [165], focusing on providing increasingly meaningful partial results during the minimization process (purple modules in Figure 4.1). Furthermore, we impose the following requirements to the modules that compute the approximated similarities (grey and red modules in Figure 4.1):

1. The performance gain due to the approximation must be high enough to enable interaction.
2. The amount of degradation caused by the approximation must be controllable. A small increase of approximation must not lead to large degradation of the results.
3. The approximation quality must be measured and visualized to avoid misleading the user.
4. The approximation can be refined during the evolution. The refinement can be steered by the user.

In the following Sections 4.3.1 to 4.3.4, we describe the A-tSNE algorithm in detail using the MNIST [95] dataset for illustration. The dataset, which we introduced in Section 3.3, consists of 60k labeled gray scale images of handwritten digits. Each image is represented as a 784 dimensional vector, corresponding to the gray values of the pixels in the image.

4.3.1 A-tSNE

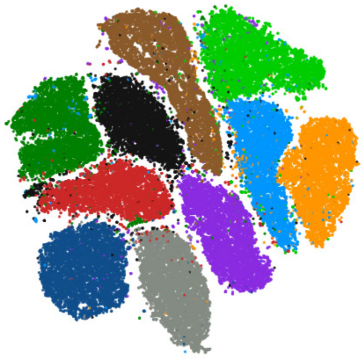
A-tSNE improves the BH-SNE algorithm, by using fast and Approximated KNN computations to build the approximated high-dimensional joint-probability distribution P^A, instead of the exact distribution P. The cost function $C(P^A, Q^A)$ is then minimized in order to obtain the approximated embedding described by Q^A.

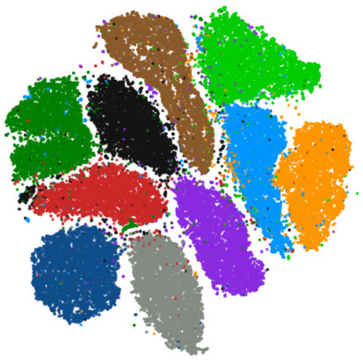
The similarity between points is computed using the set of approximated neighbors $\mathcal{N}_i^A$, instead of the exact neighborhood $\mathcal{N}_i$ (see Equation 3.9). We define the precision of the KNN algorithm as ρ. ρ describes the average percentage of points in the approximated neighborhood $\mathcal{N}_i^A$ that belongs to the exact neighborhood $\mathcal{N}_i$:

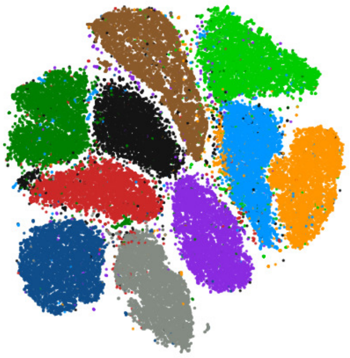
$$\rho = \sum_{i=1}^{N} \frac{\rho_i}{N} \qquad \rho_k = \frac{|\mathcal{N}_k^A \cap \mathcal{N}_k|}{|\mathcal{N}_k|}, \tag{4.1}$$

where $|\cdot|$ indicates the cardinality of the neighborhood. The cardinality of $\mathcal{N}_k$ is indirectly specified by the user as explained in Section 3.2, as three times the value of the perplexity parameter μ. ρ is an input parameter that can be defined by the user. The larger the value of ρ the more similar will P_A be to P and in turn the more similar the approximated embedding will be to the exact one.

To better understand the effect of the approximated queries, it is useful to interpret the BH-SNE algorithm as a force-directed layout algorithm [45], which acts on an undirected graph created by the KNN relationships. A data point $\mathbf{x}_i$ is repelled by


(a) BH-SNE - Time: 3191.8 s


(b) $\rho = 0.34$ - Time: 30.1 s


(c) $\rho = 0.23$ - Time: 20.4 s

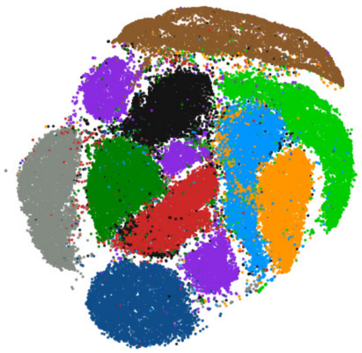

(d) $\rho = 0.07$ - Time: 13.0 s

Figure 4.2: **Embeddings of the MNIST dataset** using different approximation levels. Each point represents an image of a handwritten digit in the MNIST dataset presented in Section 3.3. Points are colored according to the classification of the image. It can be seen that a reasonable approximation as in (b) and (c) produces nearly identical results, compared to the original BH-SNE (a) two orders of magnitude faster. Even very low precision (d) produces clearly distinguishable clusters, even though the embedding visually differs from (a)-(c). Extensive tests on the quality of the results are provided in Section 4.3.4.

all other data-points but to a subset of the data-points given by its neighborhood relationships, where attraction forces are created by a set of springs which connect $\mathbf{x}_i$ with all the points in $\mathcal{N}_i$.

When specifying a lower precision ρ, resulting in a coarser approximation, some springs that connect points, which are close in the high-dimensional space will be missing and instead distant points will be connected. This will result in a false repulsion between the points missing a connecting spring. Using P^A reduces the quality of the embedding but improves its computation time by several orders of magnitude. However, reasonable results can be achieved even with low precision, because each data point is usually connected to a large number of springs and, therefore, the overall structure can be preserved. This observation holds for local as well as global structures. Intuitively, even if two points are no longer connected, they might share a common neighbor, which indirectly connects both.

Figure 4.2 shows the embeddings generated using different precision values ρ for the computation of the high-dimension similarities. We use the whole MNIST dataset as the input and we color each data-point accordingly to the digit it represents for validation purposes. Figure 4.2a shows the embedding generated with the exact neighborhood, whereas Figure 4.2b shows the embedding generated with a precision of $\rho = 0.34$. It can be seen that similar structures are preserved using approximated neighborhoods. Figure 4.2d shows the embedding generated with $\rho = 0.07$. Even though the embedding visually differs from the exact embedding, depicted in Figure 4.2a, the overall clustering of the data is preserved rather well, whilst the time needed for the computation of the similarities is greatly reduced. Where the original algorithm needs 3191 seconds for the initialization using a precision of $\rho = 0.34$ we can achieve a speedup of two orders of magnitude, resulting in a computation time of 30 seconds. By using a precision of $\rho = 0.07$, it is further reduced to 13 seconds.

4.3.2 Approximated KNN

We achieve different levels of precision by means of different parameterizations of an approximated KNN algorithm called *Forest of Randomized Kd-Trees*. In this section, we describe this technique and how its parameters can be mapped to the precision ρ.

When the dimensionality of the data is high, there are no exact KNN algorithms performing better than linear search [121]. Therefore, the development of approximated KNN algorithms is needed to deal with high-dimensional spaces. A survey on existing algorithms, including an extensive set of experiments, can be found in the work of Muja et al. [120]. For our Approximated-tSNE, we use a space partitioning technique called *Forest of Randomized KD-Trees* [160] to compute the approximated neighborhoods. This technique has proven to be fast and effective in querying of high-dimensional spaces [121]. A KD-Tree [43] is a binary tree used to partition a k-dimensional space. Each node in the tree is a $k-1$ dimensional hyper-plane, orthogonal to one of the initial k-dimensions, that splits the space into two half spaces. The recursive splitting creates a hierarchical partition of the k-dimensional space.

In a *Forest of Randomized KD-Trees*, a number $\mathscr{T}$ of KD-Trees are generated. The splitting hyper-planes are selected by splitting along a randomly selected dimension among the $\mathscr{V}$ dimensions characterized by the highest variance. A KNN search is computed on all $\mathscr{T}$ KD-Trees, while a maximum number of leaves $\mathscr{L}$ are visited. A priority-queue, ordered by increasing distances to the closest splitting hyper-plane, is used to decide which nodes must be visited first across the forest.

The process is stopped when the necessary number of leaves have been evaluated. The parameterization of the Forest of Randomized KD-Trees can overburden the typical end user. To hide this complexity, we integrate the work by Muja et al. [121] and expose only the single precision parameter ρ to the user. The parameters $(\mathscr{T}, \mathcal{V}, \mathscr{L})$ used for the creation and querying of the *Forest of Randomized KD-Trees* are heuristically chosen, as described by Muja et al. [121], to generate KNNs with a target precision ρ.

4.3.3 Steerability

A-tSNE is computationally steerable [122], in the sense that the user can define the level of approximation to specific, interesting areas. In this section, we present the changes we made to the BH-SNE algorithm to allow for the refining of the approximation.

The refinement that we propose is done by computing the exact neighborhood for one point at a time. This process leads to a mix of exact and approximated neighborhoods. For each updated neighborhood, a Gaussian distribution P_i is computed and the sparse joint-probability distribution P^A must be updated accordingly. This update, however, is not straightforward. First, the symmetrization of P^A in Equation 3.2 requires to combine Gaussian distributions enforced by different data-points and, second, the sparse nature of the distribution P^A renders fast updates challenging.

We solve these issues by observing that a direct computation of P^A can be avoided and the distribution can be indirectly obtained using the Gaussian distributions enforced by the K-Nearest Neighbors. Equation 3.2 can be split into two components which correspond only to the Gaussian distributions P_i and P_j:

$$p_{ij} = \frac{p_{j|i}}{2N} + \frac{p_{i|j}}{2N}. \tag{4.2}$$

Using this formulation, we only need to store one Gaussian distribution per point. Therefore, points can be handled individually without any performance loss. This allows us to execute the refinement of the high-dimensional similarities in parallel to the gradient descent, and serves as the base for the manipulation of the high-dimensional data. Furthermore, we are not constrained to updating the neighborhood of a data-point just once. The analyst can request different levels of approximation for a given area before starting the computation of the exact high-dimensional similarities. For each data-point we store ρ_i as the requested precision for the neighborhood $\mathcal{N}_i$.

A change in a neighborhood, however, yields a change in the cost function C, see Equation 3.1, which we are minimizing. To avoid the risk of getting stuck in a local minimum during the gradient descent, we introduce an optimization strategy called *Selective Exaggeration with Exponential Decay*. Our strategy is inspired by the optimization strategy called *Early Exaggeration* presented by van der Maaten et al. [177]. The idea of *Early Exaggeration* is that, by exaggerating the attractive forces, see Equation 3.7, by a factor τ during the first I_τ iterations of the gradient descent, local minima can be avoided. Using the *Selective Exaggeration with Exponential Decay*, we apply an exaggeration τ to the attractive forces acting on a data-point x_i when it is refined. The exaggeration is then smoothly removed on a per-point basis using an exponential decay of the exaggeration factor. This can be interpreted as a localized reinitialization of the gradient descent triggered by user interaction with the embedding.

4.3.4 Performance and Accuracy Benchmarking

In this section, we present a detailed performance analysis of A-tSNE compared to BH-SNE using several standard benchmark datsets. All performance measurements were obtained using a DELL Precision T3600 workstation with a 6-core Intel Xeon E5 1650 CPU @ 3.2GHz, 32GB RAM and a NVIDIA GTX 680. We apply the same preprocessing steps as presented by van der Maaten [176], without applying a preliminary dimensionality-reduction by means of a Principal Component Analysis. We use the MNIST dataset [95] (60k data-points, 784 dimensions), the NORB dataset [96] (24300 data-points, 9216 dimensions) and the TIMIT dataset [156] (1M data-points, 39 dimensions). Throughout the experiments we used a parameter setup similar to the one used to benchmark the BH-SNE [176] and a fixed perplexity value of $\mu = 30$. First, we evaluate the performance of A-tSNE in relation to the parameters $(\mathcal{T}, \mathcal{V}, \mathcal{L})$ used in the *Forest of Randomized KD-Trees*, as described in Section 4.3.2, using three different configurations: $\mathcal{T} = 4$ $\mathcal{L} = 1024$, $\mathcal{T} = 2$ $\mathcal{L} = 512$ and $\mathcal{T} = 1$ $\mathcal{L} = 1$. For all configurations we set $\mathcal{V}$ to 5 as suggested by Muja et al. [121].

The left chart in Figure 4.3 shows the comparison of computation times (in logarithmic scale) of the high-dimensional similarities on the MNIST dataset obtained by our technique and by the BH-SNE algorithm. The right chart in Figure 4.3 depicts the precision ρ of the neighborhoods. The precision is given by Equation 4.1 and it is computed using the exact and the approximated neighborhoods. Generally, our approach generates a good embedding very efficiently for any given dataset we tested. Figure 4.2(b-e) show the embeddings generated using the described parameter settings for the MNIST dataset after 1000 iterations. It can be seen that we achieve visually comparable results more than two orders of magnitude faster compared to the BH-SNE implementation.

Figure 4.3 shows how the precision decreases when increasing the data size for a fixed parameter setting. The number of leaves (corresponding to data points) to visit, included in the parameter setting, is fixed independently of the data size. When the data size increases the same number of leaves, corresponding to a smaller fraction of the overall data, is visited, causing the lower precision. In general, we can see that with a small reduction in precision, the computation time can be greatly reduced.

Finally, we analyze the error introduced by the approximation of the similarities in the high-dimensional space using the NORB, MNIST and TIMIT datasets. The

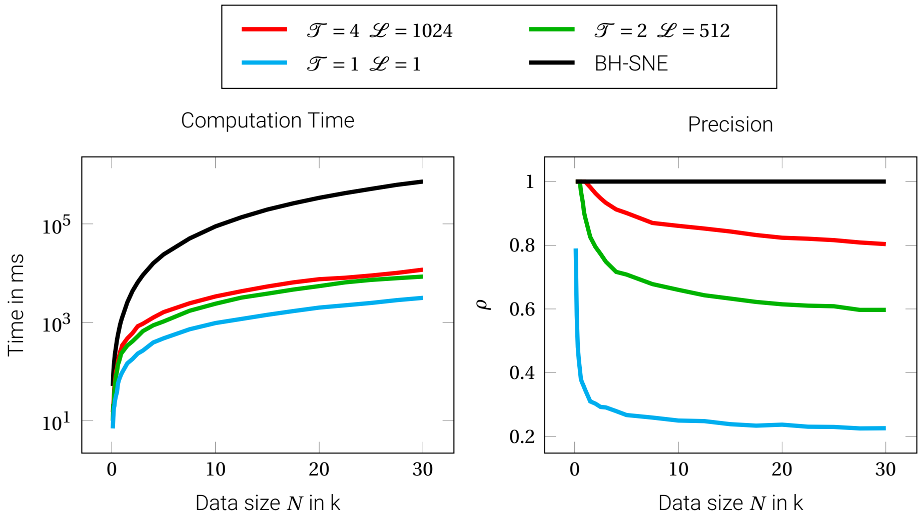


Figure 4.3: **Computation time for the high-dimensional similarities** using the MNIST dataset, with BH-SNE and A-tSNE with different parameters (left) and precision with different parameter settings (right).

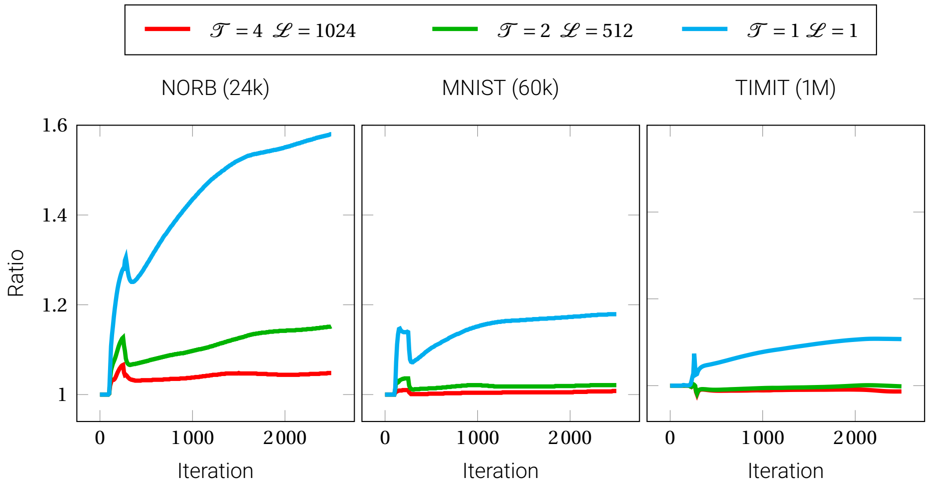


Figure 4.4: **Approximated to exact cost ratio** on different datasets of increasing size. When the size of the data increases, the ratio of the approximated cost divided by the exact cost is reduced given the same set of parameters.

cost function $C(P,Q)$ is the most direct indication of the quality of the embedding and we compare minimizing of the cost function $C(P,Q^A)$ to $C(P,Q)$. Q^A is the joint-probability distribution that describes similarities in the approximated embedding obtained by the minimization of $C(P^A,Q^A)$. Figure 4.4 shows the $C(P,Q^A)/C(P,Q)$

ratio. Smaller values indicate less error, with a value of 1 meaning that no approximation error is present. The *Early Exaggeration* of the attractive forces (see Section 4.3.3) is responsible for the peak in the ratio that is visible during the first 250 iterations. By exaggerating the attractive forces the approximation error is increased. The absolute value of the cost (not depicted in Figure 4.4) decreases with every iteration.

The usage of a Forest of Randomized KD-Trees with $\mathscr{T} = 1$ $\mathscr{L} = 1$ generates an embedding with a large error. This configuration is an upper bound of the error and a lower bound in computation time; by visiting only one leaf during the traversal of the forest composed by just one tree, the approximated KNN algorithm becomes a *greedy algorithm*. We can also note that with increased data sizes the approximation error decreases. For the TIMIT dataset we observe that the approximation errors generated using $\mathscr{T} = 2$ $\mathscr{L} = 512$ and $\mathscr{T} = 4$ $\mathscr{L} = 1024$, are similar or better, than the exact one. By increasing the number of points, the effect of the false repulsive forces (Section 4.3.1) is compensated by the increasing number of attractive forces among data-points. The results clearly show that we can rapidly provide very accurate embeddings allowing immediate interaction, without misleading the user. With a large number of data points we effectively generate tSNE embeddings as demonstrated by the reduced approximation error.

4.4 Interactive Analysis System

Using A-tSNE, the data analysis is started without waiting for the exact computation of the similarities in the high-dimensional space. This operation is the main bottle neck for interactivity, e.g., when data is modified or tSNE parameters are changed by the user. However, the embedding is created based on approximated information. Our system supports three different strategies for the refinement of the approximation, leading to the generation of different and more precise, embeddings.

To steer the refinement, the user must be aware of the error in the embedding. Therefore, we present a visualization that shows the level of approximation (Section 4.4.2). We also take advantage of the steerability of A-tSNE (Section 4.3.3) to allow for direct manipulation of the high-dimensional data, for example, by adding and removing data-points or by changing the dimensions used to represent the data. Finally, we implemented these techniques in a coordinated multiple-views framework that allows for the direct inspection of the data in the embedding.

4.4.1 User Steerable Refinement

The refinement process used to steer the computation of an A-tSNE embedding works on a per-point basis, see Section 4.3.3. A naive strategy to refine the embedding, is to progressively update the neighborhoods of all the points in X, while the gradient descent optimization is computed. However, when computational resources are scarce, it makes sense to steer the refinement process to increase precision ρ in areas of the embedding that the analyst finds interesting, e.g., based on initial visual clusters appearing in the embedding. We propose three different strategies that are used to select the data points to be refined: *user selection*, *breadth-first search* and *density-based refinement*. These strategies are presented in the following sections.

User Selection

The user selects a subset of points for immediate refinement, by brushing in the embedding. This strategy is less effective when just a few points are selected for refinement, as the forces exerted on its neighbors are still approximated, which can lead to an unfaithful description of the high-dimensional data.

Breadth-First Search

If only a few points are selected for refinement, we extend the process to include their neighborhoods. We use a breadth-first visit on the graph created by the KNN relationships to extend the refinement. When a point is refined, its neighbors are queued for refinement. We also implemented this strategy using a priority queue, where, e.g., points can be prioritized by their euclidean distance to already refined points. This allows better control on the expansion of the refined area at the cost of slower computations introduced by the priority queue.

Density-Based Refinement

When the user is more interested in gaining a global overview of the exact embedding, a density-based refinement strategy is used instead of a local refinement. This strategy is based on the observation that points in the less dense areas of the high-dimensional space, are responsible for the creation of the global relationship in a tSNE embedding [177]. The data-points are refined with an order given by the density in the high-dimensional space, where low-density points are refined first. An indication of this density is the variance σ_i of the Gaussian distribution, as explained in Section 3.1. This strategy works within a user-defined selection or on the whole dataset.

4.4.2 Visualization and Interaction

The visualization of the tSNE embedding provides an overview on the high dimensional data and should be combined with the ability to inspect the data on demand. In our system, the user selects data points by brushing in a point- or density-based representation of the embedding, the *overview*. We provide specific visualizations of the high-dimensional space using linked views, adaptive to the data at hand. Additionally, we use a magic lens or a full-view overlay to indicate the approximation level. A detailed description of such solutions is given in the following sections.

Density-Based Visualization

The visualization of the embedding, using simple points, is affected by visual clutter when the number of points increases. Density-based [162] visualizations are commonly used to show a tSNE embedding [6, 12, 157, 176] because of their ability

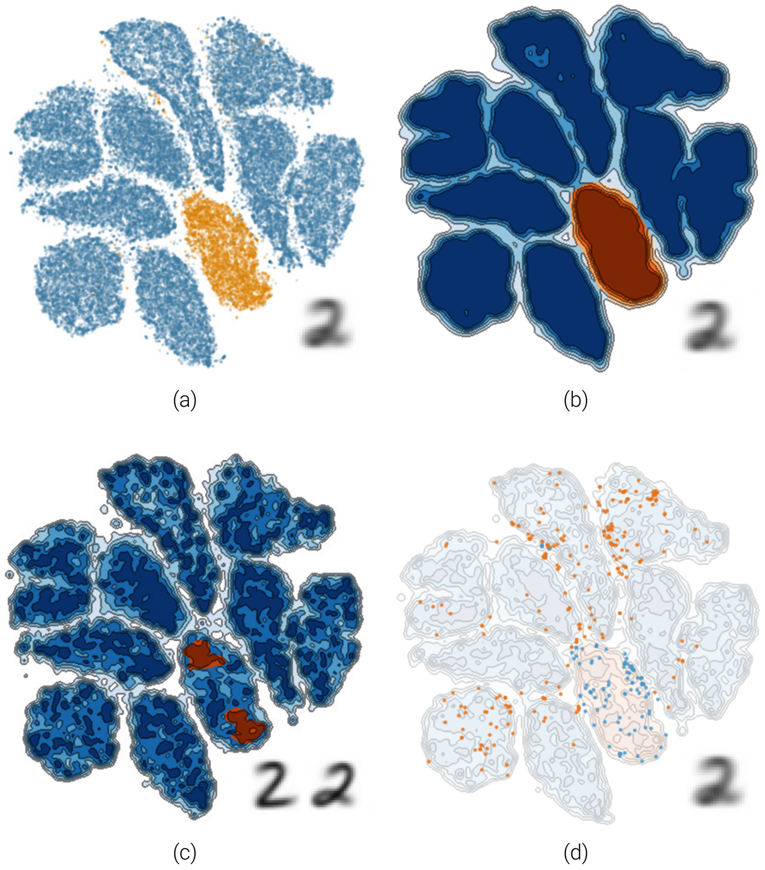


Figure 4.5: **A-tSNE embedding of the MNIST dataset**. (a) uses a point-based visualization with an alpha value of 0.25, the points colored in orange correspond to the digit '2'. (b,c) uses the real-time density-based visualization as described in Section 4.4.2. By changing the bandwidth of the kernel density esitmation, clusters at different scales are visible. (d) shows the outliers in the data-points representing the digit '2' by means of a combination of the density-based and the point-based visualization. All figures show the average image of the selected clusters.

to visualize features at different scales. We apply real-time kernel density estimation (KDE) [92] for the creation of an interactive density-based visualization of the embedding. We use changes in the color hue to visualize selections, for example to highlight data points that are selected to be analyzed in other views of the coordinated multiple-view framework. The KDE is computed by assigning a value for each pixel $\mathbf{p}$ using the *kernel density estimator* $f(\mathbf{p}, h)$ as follows:

$$f(\mathbf{p},h) = \frac{1}{N}\sum_{i=1}^{N} G(||\mathbf{p}-\mathbf{y}_i||,h). \tag{4.3}$$

$G(d,h)$ is a zero mean Gaussian distribution with standard deviation h, which can be interactively chosen by the user in order to reveal clusters at different scales. Additionally, we introduce a transfer function, mapping $f(\mathbf{p},h)$ to a color, in order to highlight user-defined selections. Areas with a large percentage of selected points are visualized with a different transfer function, and selection outliers are shown as points. To achieve this goal, we introduce a new kernel density estimator $s(\mathbf{p},h)$, which illustrates the density of the user selection in a pixel $\mathbf{p}$. Given a set of selected data-points S we use:

$$s(\mathbf{p},h) = \frac{1}{f(\mathbf{p},h)}\frac{1}{|S|}\sum_{\mathbf{y}_i\in S} G(||\mathbf{p}-\mathbf{y}_i||,h) \tag{4.4}$$

If $s(\mathbf{p},h)$ is higher than a threshold S_{thresh}, a transfer function based on a different hue and with a higher luminance is used. We found empirically that a value $S_{thresh} = 0.5$ performs satisfactorily without compromising the quality of the visualization. We also use a point-based visualization of isolated selected data-points and, unselected data-points in selected regions. Finally, the user can adjust the opacity of the points and the density-based visualization to the needs of the analysis.

An example of different visualizations of the embedding is presented in Figure 4.5, using the MNIST dataset. The analyst can change the bandwidth h, the transfer function, and the opacity interactively in order to show clusters at different scales and outliers in the selection. For example, Figure 4.5b shows the selection of a high-level cluster. If a different bandwith is chosen, as in Figure 4.5c, clusters at a different level appear. Finally, if the labels are used to make a selection in the embedding, as in Figure 4.5d, it is possible to see the distribution of the outliers in the density-based visualization.

Visualization of the Approximation

The complexity of high-dimensional structures, also known as *intrinsic dimensionality*, usually does not allow for an exact representation of the data in 2D. For this reason, it is of crucial importance to integrate the visualization of the embedding with tools that allow to assess its quality. Such an assessment is challenging and several interactive techniques have been developed in recent years [109]. In this work, we are not concerned with the quality of the embedding itself, but rather with the level of approximation introduced by A-tSNE. This information is provided to the user to focus the attention on specific areas of the embedding for a quality analysis, performed with a separate tool.

We enhance our density-based visualization to show the precision ρ_i. Note that ρ_i is different for every data-point and changes during the refinement process, as

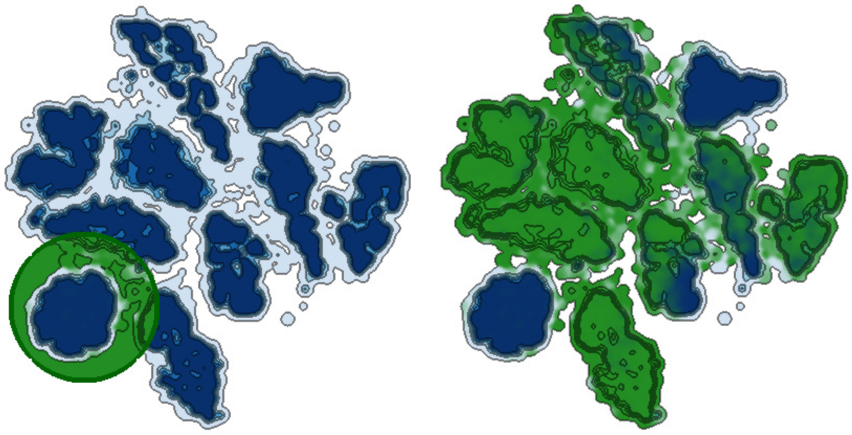


Figure 4.6: **Visualization of the approximation** in the embedding by means of a magic lens (a) and the full view mode (b).

described in Section 4.3.3. For each pixel $\mathbf{p}$ we assign a value given by the function $a(\mathbf{p},h)$ that represents the approximation value given the bandwidth h:

$$a(\mathbf{p},h) = \frac{1}{f(\mathbf{p},h)} \frac{1}{\sum_{i=1}^{N} \rho_i} \sum_{i=1}^{N} \rho_i G(||\mathbf{p}-\mathbf{y}_i||,h)$$

$a(\mathbf{p},h)$ is the precision ρ_i weighted kernel-density divided by the kernel-density estimator $f(\mathbf{p},h)$. The value $a(\mathbf{p},h)$ is between zero and one and is used directly for encoding of the approximation in the visualization.

The value of the function $a(\mathbf{p},h)$ is visualized in two different ways. First, we introduce a Magic Lens [171] that shows the approximation with a minimal conceal of the data. We use a circular lens that can be overlayed on the density-based visualization and $a(\mathbf{p},h)$ is used to define the transparency α of every pixel in the lens. To better highlight the refined areas, we use $\alpha = 1 - a(\mathbf{p},h)^k$, where k is a user selected parameter, to compute α. We provide a default value of $k = 2$.

Figure 4.6a shows the lens over a cluster that is already refined and, therefore, is visible through the lens. The green tone indicates the area where similarities are still approximated. Contours in approximated areas are preserved to indicate the structure of the embedding. We color the areas without points in green to put more emphasis on refined areas. In addition to the Magic Lens, we provide the possibility to map approximation to the complete view.

This view is especially useful when one of the global refinement strategies is selected as it shows an overview on the refinement process. However it also diminishes the ability to distinguish high-density areas.

Figure 4.6b shows the approximation in the embedding using this approach. It is possible to see that two clusters are already refined, relying on exact neighborhood relationships. The user selected a *Breadth-first search* refinement strategy, therefore, the refinement is spreading through the embedding, leading to some areas in the top-right corner having the original color. However the perception of clusters is reduced by removing the color information inside the contours.

4.4.3 Data Manipulation

In Section 4.3.3, we show that we are able to update high-dimensional similarities between data-points during the gradient-descent minimization. In this section, we take advantage of this possibility, introducing different operations that are used to manipulate the original data-points in their high-dimensional feature space. The embedding does not need to be recomputed but evolves dynamically as the data changes. At the center of an interactive exploration of data is the ability to add or remove data on demand, use different representations of the same dataset or adapt to any changes in the data [40]. For example, the addition and the removal of data points are two fundamental operations that enable us to monitor a high-dimensional stream in real-time.

Inserting Points

For a point $\mathbf{x}_a$, which we want to add to the embedding, its neighborhood $\mathcal{N}_a$ needs to be computed. We compute the neighborhood with the approximated KNN algorithm, as described in Sec 4.3.2. Finally, we check whether $\mathbf{x}_a$ belongs to the KNN of each point in X. We define d_i^{Max} as the maximum distance between a point $\mathbf{x}_i$ and the points in its neighborhood $\mathcal{N}_i$. The update of the neighborhoods is written as follows:

$$\begin{aligned} &\forall \mathbf{x}_i \in X \text{ if } ||\mathbf{x}_a - \mathbf{x}_i|| < d_i^{\text{Max}} \\ &\quad \text{then } \mathbf{x}_a \in \mathcal{N}_i \text{ and } \mathbf{x}_j \notin \mathcal{N}_i : ||\mathbf{x}_i - \mathbf{x}_j|| = d_i^{\text{Max}} \end{aligned} \tag{4.5}$$

We cache d_i^{Max}, leading to a complexity for this update of $O(N)$. A priority queue is used to efficiently update d_i^{Max} after the insertion of $\mathbf{x}_a$ in a given neighborhood $\mathcal{N}_i$. It is important to observe that the insertion of $\mathbf{x}_a$ in $\mathcal{N}_i$ will not reduce the estimated precision ρ_i. The initial position in the embedding $\mathbf{y}_a$ is given by the average position of its neighbors $\mathcal{N}_a$ weighted by their similarity $p_{j|i} : \mathbf{x}_j \in \mathcal{N}_i$. The new point $\mathbf{x}_a$ is then added in the *Forest of Randomized KD-Trees*. This operation is performed in $O(\log(N))$.

Deleting Points

Removing a point $\mathbf{x}_r \in X$ is performed by deleting $\mathbf{x}_r$ from the KNN of every point $\mathbf{x}_i \in X$. This operation has a computational complexity of $O(N)$. By removing $\mathbf{x}_r$ from a neighborhood $\mathcal{N}_i$ we reduce the number of $\mathbf{x}_i$ neighbors to $K-1$ and a new neighbor must be found to maintain the precision level. However, the new point in the neighborhood is the most dissimilar of the points in $\mathcal{N}_i$ thus its attractive force is rather small and we propose to ignore the contribution of the missing point, decreasing the estimated precision ρ_i by $1/K$. To avoid degeneracies, when the size of the neighborhood $\mathcal{N}_i$ goes below a given threshold, e.g., $K/2$, the neighborhood is updated using approximated computations. The *Forest of Randomized KD-Trees* is updated in $O(\log(N))$.

Data Modification

The insertion and deletion of data points enables a new way of analyzing data changes, for example, changes in time. New data points are added to the embedding when ready and old ones are removed in real-time. However, data that are already present in the embedding can change over time and must be updated accordingly. We handle changes in the value of a single high-dimensional data-point by a combination of removal and addition operations. A different modification of the data is performed not by changing the values of single data points, but by changing the dimensions of the data itself. Examples of this operation are the addition or the removal of dimensions to inspect the influence of a given dimension in the generation of visual clusters. With such a modification, all the data points in X change their position in the high-dimensional space. Therefore, all the neighborhoods must be reconsidered and it is more convenient to compute a new approximated joint-probability distribution P^A. When the distribution P^A is changed, the function that is to be minimized by the gradient descent also change, see Equation 3.1. To avoid local minima, we apply the *Selective Exaggeration with Exponential Decay*, see Section 4.3.3, to all the data points. After such an operation, the user expects to see major changes in the embedding, where the extent of such modifications gives information about the differences of the new representation to the old one.

4.4.4 Visual Analysis Tool

We implemented A-tSNE as a module in an integrated, interactive, multi-view system for the analysis of high-dimensional data. Figure 4.7 shows a screenshot of the system and its different views. The interface is divided into two main areas. At the top, three different views are used to show the intermediate embeddings (7a), the data (7b) and the state of refinement processes (7c), respectively. Controls are at the bottom of the interface: (7d) for the generation of intermediate embeddings, (7e) visualization of the embedding, (7f) data manipulation and (7g) refinement.

The data subject to the analysis are visualized in the *Data View* (7b). Selections in the embeddings are reflected in the *Data View* with strategies that depend on the data type. We implemented multiple widgets that are used to support the analysis process of different data types. These widgets include a heatmap view, a 3D volume view (7b bottom) and an image view (7b top row). If necessary multiple and different views are combined for the analysis.

The *Refinement-Status View* (7c) is used to give an overview of the progress of the refinements started by the user. The user can steer the evolution of the embedding by refining areas with strategies as described in Section 4.4.1. A refinement

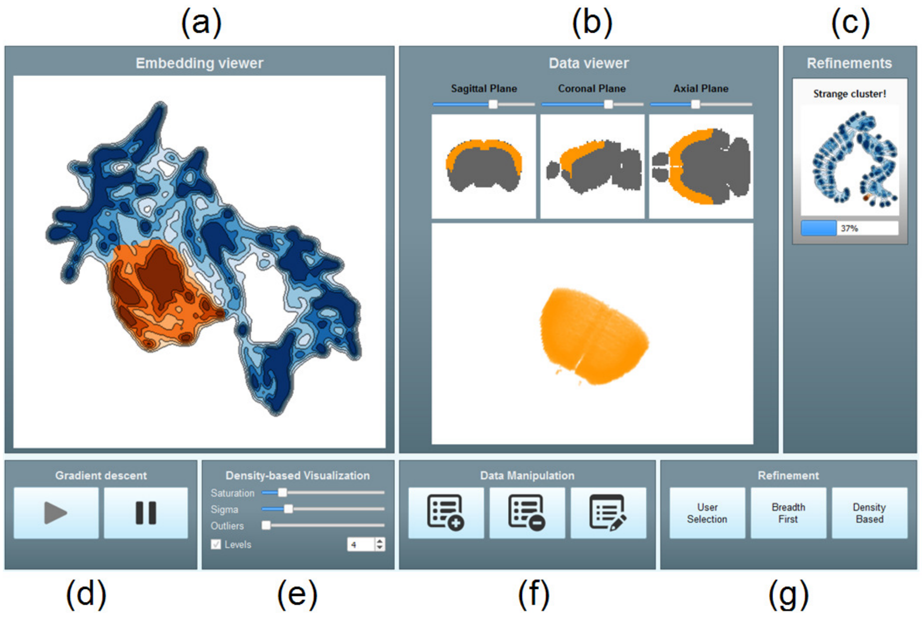


Figure 4.7: **Screenshot of our integrated system** using multiple linked views for interaction. The system comprises an embedding viewer (a), a data viewer (b) and a refinement viewer (c). Controls on the gradient descent (d), the density-based visualization (e), the data-manipulation (f) and the refinements (g) are at the bottom of the interface.

process is identified by the snapshot of the embedding when the user started the refinement, a user-defined description, and a progress bar that shows the percentage of the refined data-points over the selected ones.

4.4.5 Implementation

We implemented the system using a combination of C++ and Qt, as well as OpenGL with custom shaders in GLSL for the visualization of the embedding. Where possible, we used parallel computations with OpenMP. The approximated neighborhoods are computed using the FLANN library [121], which implements KNN algorithms. The density-based visualization is computed on the GPU using OpenGL and GLSL shaders. A precomputed floating-point texture is generated using a Gaussian kernel. A geometry shader is used to generate a quad for each point that is colored using the precomputed texture, the KDE is obtained by drawing into a Frame Buffer Object using an additive blending [92].

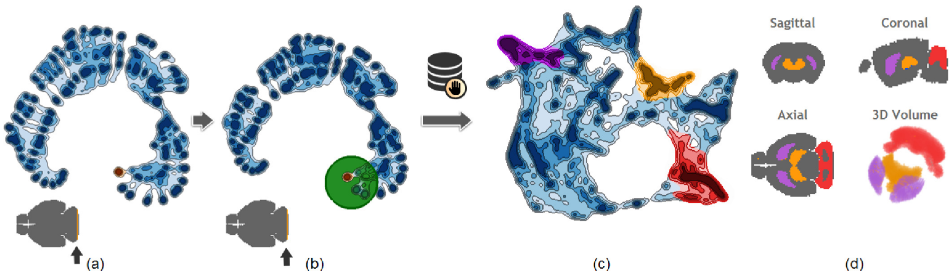


Figure 4.8: **Analysis of the gene expression in the mouse brain using A-tSNE**. The first embedding (a) is generated in ≈ 51 seconds while 3 hours and 50 minutes are required by BH-SNE. The analyst inspects a cluster and finds that it corresponds to a slice in the data. The cluster does not disappear after the neighborhoods are refined, as shown by the lens in (b). A change in the high-dimensional data reveals that genetic information can be used to differentiate anatomical regions. (c) shows the final embedding based on a small number of Principal Components where three clusters are highlighted and (d) shows the corresponding regions in the brain.

4.5 Case Study I: Exploratory Analysis of Gene Expression in the Mouse Brain

In this section, we demonstrate the advantages of using A-tSNE in our visual analysis tool for the visual analysis of high-dimensional data. To this extent, we present a case study, based on the work by Mahfouz et al. [107], who use tSNE to explore the Allen Mouse Brain dataset [99]. The dataset is composed by 61164 voxels obtained by slicing the mouse brain in 68 slices. Each voxel is a 4345-dimensional vector, containing the genetic expression at the corresponding spatial position. tSNE is computed using the voxels as data-points and the expression of the genes as high-dimensional space. Mahfouz et al. discuss the hypothesis that genetic information can be used to differentiate anatomical structures in the brain. Some regions in the brain, e.g. the Cerebellum, are known to have a highly different genetic footprint compared to the rest of the brain. They demonstrate that tSNE is effective in separating different anatomical structures, e.g. white and grey matter, only based on the genetic footprint.

Figure 4.8 depicts the typical analytic workflow using our visual analysis tool. The first goal during the analysis is to validate the input data. The acquisition process may not be perfect, data can be incomplete or noisy, therefore, it must be re-acquired or preprocessed before interesting results can be generated. Driven by the need to validate the data as soon as possible, the user selects a reasonably low value for the desired precision, e.g. $\rho = 0.2$, that will be used to estimate the parameters of the KNN algorithm. With such a parameterization, A-tSNE computes the high-dimensional similarities in ≈ 51 seconds while 3 hours and 50 minutes are required by BH-SNE.

The user then analyzes the intermediate embeddings, produced by A-tSNE, in order to validate the input data. After ≈ 170 seconds several clusters become visible in the embedding as depicted in Figure 4.8a. The clusters are stable for several iterations indicating that they are not an artifact of the minimization process. The user can validate this by selecting the clusters in the embedding and can inspect them in more detail, for example, by highlighting their spatial positions in the feature view, see Fig 4.8a. Points or clusters are selected by brushing in the embedding. During a brushing operation the generation of intermediate embeddings is stopped to make sure the user does not accidentally brush areas as they change. Selected points are then highlighted by a change of hue, in this case from blue to orange. Further inspection using the *Data View* in our interactive system, shows that each cluster corresponds to a slice in the dataset. Figure 4.8a shows a cluster, highlighted in orange, and the corresponding slice in the volume.

To make sure the clusters are not an artifact introduced by the approximated similarities, the user refines the selected data-points while the embedding evolves. Figure 4.8b shows the embedding after the refinement is complete. Note that the global structure of the embedding does not change during the refinement. Changes are constrained to the selected cluster, giving to the user a sense of stability in the information provided as requested by the Progressive Visual Analytics paradigm. The user can inspect the degree of approximation in the embedding using the interactive lens. The lens is less transparent over approximated areas of the embedding and transparent on the areas that contain no approximation. After the refinement of the high-dimensional similarities of the selected data points, the clusters do not disappear, which indicates that clustering is indeed driven by the data, rather than by the approximation.

Therefore, the user stops the computation of the fully refined embedding. Further analysis performed by domain experts on the raw data reveals that missing values in the input data cause the formation of small clusters in the embedding. Mahfouz et al. removed this effect by using the first 10 components, extracted by a Principal Component Analysis of the raw data, as the high-dimensional space. In the traditional analytical workflow, after the high-dimensional data are changed, a new tSNE embedding is computed from scratch. However, in our system the user directly changes the high-dimensional space and the current embedding evolves accordingly. Given that the gradient descent is minimizing a different function, the user expects structural changes that can be considerably large, see Section 4.4.3. The extent of these changes provides information about the modification in the high-dimensional space. If the embedding is stable, the new high-dimensional representation preserves relationships between data points, while an abrupt change means that new relationships are encoded in the data. In the traditional workflow without A-tSNE, any continuity and the encoded information are lost.

Approximately 200 seconds after the change in the high-dimensional data, a stable embedding is obtained. Figure 4.8c shows the final embedding, where three different clusters are highlighted. Figure 4.8d depicts the selected voxels in the brain, note how the anatomical structures are now revealed. It is possible to see how the clusters that were present in the first intermediate results disappear, showing that the cluster fragmentation is removed.

Voxels that belong to the same anatomical structure are close together in the embedding. A-tSNE is able to separate anatomical structures based on the gene expression of the 4345 genes. In their work, Mahfouz et al. [107] present embeddings created using 2, 3, 5, 10, and 20 principal components as the high-dimensional space. Identifying the right number of components is a time consuming task and the adoption of our analytic workflow helps the user in finding a good compromise by interactively analyzing the resulting embedding generated changing the number of components.

4.6 Case Study II: Real-time monitoring of high-dimensional streams

Improved computation time and the ability to modify data are the key for applying tSNE in new application scenarios, such as the real-time monitoring of high-dimensional data streams. The original tSNE algorithm fails in providing a solution for such applications. The computation of a tSNE map imposes a time constraint that cannot be ignored, when the rate in which new data is generated is higher than the time required for the computation of a tSNE map.

As proof of concept, we selected a dataset for physical activity monitoring [147] that comprises readings of three Inertial Measurement Units (IMU) and a heart rate monitor applied to 9 different subjects. Every IMU generates 17 readings every 10 ms, while the heart rate monitor generates one reading every 100 ms. Taking all sensors into account, we have a stream of data consisting of 52 readings, where a new data point is generated every 100 ms for each subject. Every subject also has a device to label the physical activity. We use the labeling of every reading to validate the insights obtained by the analysis of the embeddings.

We analyze the stream of a subject by keeping the readings of the previous $\mathscr{M}$ minutes in the embedding with a fixed approximation level. When a new reading is generated, we add it to the embedding using the technique described in Section 4.4.3. Similarly, when a reading is older then $\mathscr{M}$ minutes, we remove it from the embedding. In the test presented in this section, $\mathscr{M} = 10$ is set leading to an embedding composed, in average, by 6000 data-points that is updated every 100 ms. We add a point-based visualization to our density-based visualization, which shows the last points inserted in the embedding. The new points are colored according to the classification of the activity made by the subject and they will fade out in $\mathscr{F}$ seconds. By showing the new data-points the analyst can identify where new points are added, providing at the same time an overview of the embedding in the last $\mathscr{M}$ minutes and the trend of the last $\mathscr{F}$ seconds.

Figure 4.9a shows an embedding obtained from *subject 105*, where the color of the data-points, green in this specific case, indicates that the subject is lying down. The embedding is composed of a single big cluster that represent the *lying down activity*. The cluster is divided in four different sub-clusters that identify different readings of the sensors. The readings of the last 30 seconds belong to a single

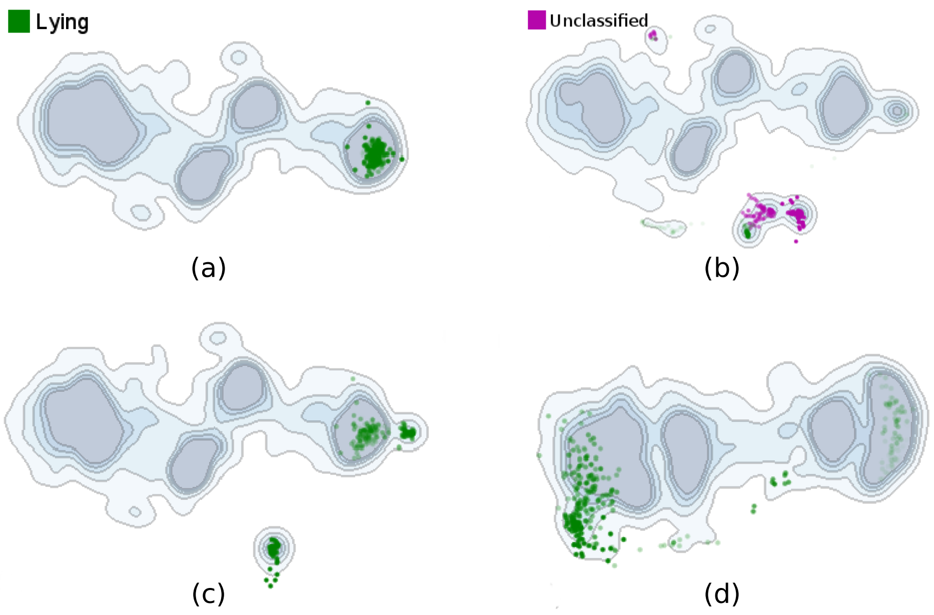


Figure 4.9: **A-tSNE used for the real-time analysis of high-dimensional streams**. The embeddings are generated using the readings of the last 10 minutes. As new readings arrive they are inserted in the embedding and they are highlighted using a point-based visualization. (a) shows the initial embedding, the color of the data-points indicates that the subject is lying down. The embedding evolves as in (b), a new cluster indicates readings of a different activity. This insight is confirmed by a change in the color of the data-points that indicates a new type of label activity. (c) shows an evolution of the embedding presented in (a) where new readings are generated from a miscalibrated sensor and, therefore, are clustered together. By removing the features corresponding to the miscalibrated sensor the embedding evolves as in (d). The cluster that identifies miscalibrated readings is removed.

sub-cluster and can be seen as points on the right side of the embedding. The embedding evolves based on new readings from the sensors, after few seconds the new data-points start to be placed further away from the original cluster, leading to the creation of a new cluster, as depicted in Figure 4.9b. After a few seconds the subject changes the classification of his activity from lying down to an *unclassified activity*, whose corresponding data-points are colored in purple. It is interesting to note that, simply by looking at the embedding, it is possible to predict a change in the labeled activity before the subject is able to record the change on his labeling device. It can be seen by the fact that few data-points labeled as a *lying down activity*, hence colored in green, are in the same cluster as the ones identified as *unclassified activity*. In this particular case, we can guess that the subject sat up before changing the labeled activity.

Finally, we simulated a miscalibration in an inertial measurement unit. Differently from a faulty sensor (not generating any readings), a miscalibrated one generates readings affected by a constant offset that is different for every dimension. We simulate the miscalibration by enforcing a random offset to the readings generated by one of the IMUs. A miscalibrated sensor generates readings that are different from the normal one and, therefore, they should be clustered together as faulty readings. Figure 4.9c shows the evolution of the embedding presented in Figure 4.9a where the miscalibrated readings are grouped by A-tSNE. After the inspection of the readings generated from the IMUs, the analyst can identify that something is wrong with one of the sensors. At this point the sensor may be replaced or, in case this is not possible, the readings from the miscalibrated sensor can be excluded by removing the corresponding dimensions from the high-dimensional space, as presented in Section 4.4.3. Such an update requires a few seconds in which the embedding is updated in order to encode the new relationship in the high-dimensional space. Figure 4.9d shows how the previous embedding evolves when the readings generated by the miscalibrated sensor are removed from the high-dimensional space. It is possible to see that the readings affected by the miscalibration are now close to the cluster that represents the *lying down activity*. However, differently from the test case presented in Section 4.5, the global structure of the embedding is preserved, still showing four different clusters.

Liu et al. [104] demonstrate that, when dealing with real-time data, the response time of the algorithm is of great importance to the user. In the presented case study, we reach real-time performance for a limited data size for the sliding window of 6000 points. However, it should be noted that when the sampling rate or the window size of the stream is much larger, A-tSNE also will not be able to handle the data in real-time in all cases.

4.7 Discussion and Conclusions

Motivated by the need of interactivity in Visual Analytics, we developed the A-tSNE technique. A-tSNE enables the rapid generation of approximate tSNE embeddings by adopting a fast and approximated computation of the high-dimensional similarities. Our algorithm is designed to be used within the Progressive Visual Analytics context, allowing the user to have a quick preview of the data. Insight obtained using approximated embeddings can be validated by refining the approximation in interesting areas with different strategies. Therefore, we present different visualization techniques for the level of approximation, which are used to guide the user during the refinement process in Section 4.3.3. It should be noted, that the level of approximation is only an indicator for how well the approximated embedding represents the exact embedding. It cannot, however, be used to judge the quality of the embedding itself, as even an exact embedding might not represent the original data perfectly. The quality of the embedding itself can be analyzed, e.g., by inspecting the preservation of k-nearest-neighborhoods [109]. The full precision of BH-SNE can always be reached by setting the precision parameter accordingly, or refining the data. Therefore, A-tSNE can effectively replace BH-SNE for the analysis of dense high-dimensional data. However, A-tSNE cannot outperform algorithms such as Q-SNE in the analysis of sparse high-dimensional data.

The refinement of the approximation itself is a stable process. As demonstrated in Section 4.3.4 and Figure 4.2, P^A is close to P if a reasonable parameterization is chosen. As a result gradually refining P^A will lead to small changes in the embedding, only. In addition, we present three different operations for the direct manipulation of the high-dimensional data. *Addition* and *removal* of data-points are mainly aimed at the inspection of high-dimensional streams. *Data modification* is used to visualize different models of the same data. Different from the refinement process, changing the model might lead to drastic changes in P^A (as it would in P) and as such might also create a very different embedding. We chose to start the optimization with the embedding created before changing the model. As a result points in the embedding might move drastically during the optimization process. While this might be confusing and less adequate for Progressive Visual Analytics, the amount of movement is related directly to the strength of the changes and as such is a very good indicator of the influence of the parts of the data that were modified on the whole embedding.

We presented two case studies to show the effectiveness of A-tSNE. *Case Study I* shows a typical analysis of a static dataset. In such a setting it is crucial to allow an interactive feedback loop, between modeling the data (i.e., finding the right number of dimensions for the PCA before embedding) and visualizing the data. Even though, we do not achieve real-time performance, we are able to drastically cut computation times, i.e., from four hours to less than a minute, allowing such interactive exploration of the data. *Case Study II* shows an example for the monitoring and analysis of streaming data. Here it is crucial to achieve real-time performance. We use efficient addition and removal of data points (see Section 4.4.3) to visualize a temporal sliding window of the data. As discussed in Section 4.6 even the large increase in performance provided by A-tSNE does not allow real-time analysis of large data. We believe that this example illustrates as well, that real-time feedback can be important for data analysis. While in this chapter we focused on improving the computation of the similarities between data points, in the next chapter we focus on the gradient descent computation as presented in Chapter 3. More specifically, we present how the kernel density estimation presented in Section 4.4.2 is adapted to speed-up the computation of the gradient of tSNE's objective function.

5 Linear tSNE Optimization

Hours instead of days! Now we have minutes instead of hours!

James Tiberius Kirk

In this chapter we present a novel approach to the computation of the gradient of the tSNE's objective function that takes a fraction of the time requested by the Barnes-Hut-SNE algorithm. Our technique, which makes use of the rendering pipeline to compute the gradient as a derivation of three scalar fields, is implemented on the GPU and run in the client side of a web browser.

N. Pezzotti, A. Mordvintsev, T. Höllt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Linear tSNE optimization for the Web. arXiv preprint arXiv:1805.10817, 2018 [139].

5.1 Introduction

Given the popularity of the tSNE algorithm [177], research efforts have been spent on improving its $O(N^2)$ computational and memory complexity. While many works focused on the improvement of the similarity computation [112,136,168,176], including our A-tSNE algorithm presented in the previous chapter [138], only limited effort has been spent in improving the minimization algorithm employed for the creation of the embedding [87, 112, 176]. The most notable of these improvements is the Barnes-Hut-SNE (BH-SNE) that is presented in Chapter 3. BH-SNE makes use of an N-body simulation approach [1] to approximate the repulsive forces between the data points. Despite the improvements, the minimization requires many minutes using a highly-optimized C++ implementation.

In this chapter we present a novel approach that focus on the minimization of the objective function for the creation of the embedding. We observe that the heavy tail of the t-Student distribution used by tSNE makes the application of the N-body simulation not particularly effective. To address this problem we propose a novel minimization approach that embraces this characteristic and we reformulate the gradient of the objective function as a function of scalar fields and tensor operations. Our technique has linear computational and memory complexity and, more importantly, is implemented in a GPGPU fashion. The latter allowed us to implement a version for the browser that minimizes the objective function for standard datasets in a matter of seconds.

The contribution of the technique presented in this chapter is twofold:

- A linear complexity minimization of the tSNE objective function adopts GPGPU computations. Specifically, we
  - approximate the repulsive forces between data points by drawing low-resolution textures and
  - we adopt a tensor-based computation of the objective function's gradient.
- An efficient implementation of our algorithm using WebGL and is released as part of Google's TensorFlow.js library

The rest of the chapter is structured as follows. In the next section, we present the related work, while in Section 5.3 we describe our approach for the minimization of the objective function. In Section 5.4, we provide the details regarding our implementation, released within Google's TensorFlow.js library.

5.2 Related Work

We now present the work that has been done to improve the sacalability of the tSNE algorithm, which was introduced in detail in Section 3.1. In Chapter 3 we introduced the Barnes-Hut-SNE (BH-SNE) [176], which reduces the complexity of the algorithm to $O(N\log(N))$ for both the similarity computations and the objective function minimization. More specifically, in the BH-SNE approach the similarity computations are seen as a k-nearest neighborhood graph computation problem, which is obtained using a Vantage-Point Tree [190]. The minimization of the objective function is then seen as an N-body simulation, which is solved by applying the Barnes-Hut algorithm [10].

In the previous chapter of this thesis we observed that the computation of the k-nearest neighborhood graph for high-dimensional spaces using the Vantage-Point Tree is affected by the curse of dimensionality, limiting the efficiency of the computation. To overcome this limitation, we proposed the Approximated-tSNE (A-tSNE) algorithm [138], where approximated k-nearest neighborhood graphs are computed using a forest of randomized KD-trees [120]. A similar observation was later made by Tang et al. that led to the development of the LargeVis technique [168]. LargeVis uses random projection trees [32] followed by a kNN descent procedure [34] for the computation of the similarities and a different objective function that is minimized using a Stochastic Gradient Descent approach [86]. Despite the improvements, both the A-tSNE and LargeVis tools require 15 to 20 minutes to optimize the cost function on the MNIST dataset [95], a 784-dimensional dataset of 60k images of handwritten digits that we introduced in Chapter 3. Better performance is achieved by the UMAP algorithm [112], which provides a different formulation of the dimensionality-reduction problem as a cross-entropy minimization between topological representations. Computationally, UMAP follows very closely LargeVis and adopts a kNN descent procedure [34] and Stochastic Gradient Descent minimization of the objective function.

The techniques presented so far do not take advantage of the target domain in which the data is embedded. As a matter of fact, tSNE is mostly used for data visualization in 2-dimensional scatterplots, while the previously introduced techniques are general and can be used for higher dimensional spaces. Based on this observation, Kim et al. introduced the PixelSNE technique [87] that employs a N-body simulation approach similar to the BH-SNE, but quantizes the embedding space to the pixels used for visualizing the embedding. However, PixelSNE requires to scale the number of used pixels with respect to the size of the dataset in order to achieve a good embedding quality due to the quantization of the embedding space.

In this chapter, we take advantage of the 2-dimensional domain in which the embedding resides and we propose a more efficient way to minimize the tSNE objective function. Contrary to PixelSNE we observe that, by quantizing only the 2-dimensional space for the computation of the repulsive forces presented in Equation 3.8, embeddings that are hardly distinguishable from those generated by the BH-SNE implementation are computed in a fraction of the time.

5.3 Linear Complexity tSNE Minimization

In this section, we present our approach to minimize the objective function, presented in Equation 3.1, by rewriting its gradient, presented in Equation 3.7. The computation of the gradient relies on a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$ that are computed in linear time on the GPU.

5.3.1 Gradient of the Objective Function

The gradient of the objective function has the same form as the one introduced in Section 3.1:

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4(\hat{F}_i^{\text{attr}} - \hat{F}_i^{\text{rep}}), \tag{5.1}$$

with attractive and repulsive forces acting on every point $\mathbf{x}_i \in X$. We denote the forces with a $\wedge$ to distinguish them from their original counterparts. Our main contribution is to rewrite the computation of the gradient as a form of a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$ in the embedding space. We define S and V as

$$\mathcal{S}(\mathbf{p}) = \sum_i^N \left(1 + ||\mathbf{y}_i - \mathbf{p}||^2\right)^{-1}, \mathcal{S} : \mathbb{R}^2 \Rightarrow \mathbb{R}, \text{ and} \tag{5.2}$$

$$\mathcal{V}(\mathbf{p}) = \sum_i^N \left(1 + ||\mathbf{y}_i - \mathbf{p}||^2\right)^{-2} (\mathbf{y}_i - \mathbf{p}), \mathcal{V} : \mathbb{R}^2 \Rightarrow \mathbb{R}^2, \tag{5.3}$$

where $\mathbf{p}$ is a location in the embedding space and $\mathbf{y}_i$ is one of the N points in the dataset. Intuitively, $\mathcal{S}$ represents the density of the points in the embedding space, according to the t-Student distribution, and it is used to compute the normalization of the joint probability distribution Q, as presented in Section 3.1. An example of the field $\mathcal{S}$ is shown in Figure 5.1b. The vector field $\mathcal{V}$ represents the directional repulsive force applied to the entire embedding space. An example of $\mathcal{V}$ is presented in Figure 5.1c-d, where the horizontal and vertical components are visualized separately. In the next section, we will present how both $\mathcal{S}$ and $\mathcal{V}$ are computed with a complexity of $O(N)$ and sampled in constant time. For now, we assume these fields given and we present how the gradient of the objective function are derived from these two fields, accelerating hereby their calculation drastically.

For the attractive forces, we adopt the restricted neighborhood contribution as presented in the Barnes-Hut-SNE technique [176]. The rationale of this approach is that, by imposing a fixed perplexity to the Gaussian kernel, only a limited number of neighbors effectively apply an attractive force on any given point (see Equation 3.3 and 3.4). Therefore we limit the number of contributing points to a multiple of the value of perplexity, equal to three times the value of the chosen perplexity, effectively reducing the computational and memory complexity to $O(N)$, since $k \ll N$ where k is the size of the neighborhood.

$$\hat{F}_i^{\text{attr}} = \hat{Z} \sum_{l \in \text{kNN}(i)} p_{il} q_{il} (\mathbf{y}_i - \mathbf{y}_l) \tag{5.4}$$

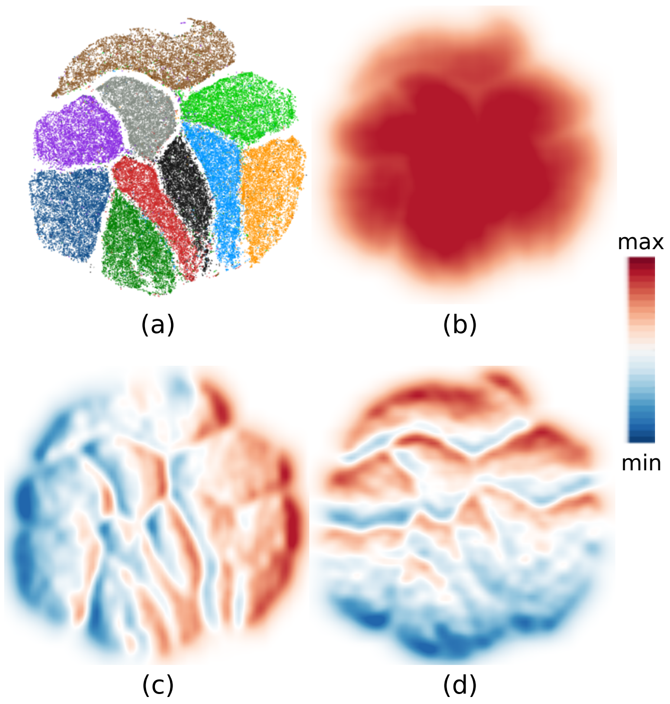


Figure 5.1: **Overview** of our approach. The MNIST dataset contains images of handwritten digits and is embeedded in a 2-dimensional space (a). The minimization of the objective function is computed in linear time by making use of a scalar field (b) and a 2-dimensional vector field (c-d). The fields are computed on the GPU by splatting properly designed kernels using the additive blending function of the modern rendering pipeline. The rest of the minimization is treated as a tensor computation pipeline that is computed on the GPU using TensorFlow.js

The normalization factor Z, as it was presented in Equation 3.6, has complexity $O(N^2)$. In our approach we compute $\hat{Z}$ in linear time by sampling the scalar field $\mathscr{S}$.

$$\hat{Z} = \sum_{l=1}^{N} \left(\mathscr{S}(\mathbf{y}_l) - 1\right) \tag{5.5}$$

Note that Z and $\hat{Z}$ formulation is identical but, since we assume that $\mathscr{S}$ is computed in linear time while the sampling is done in constant time, computing $\hat{Z}$ has linear complexity. Moreover, since $\hat{Z}$ does not depend on the point $\mathbf{y}_i$, for which we are computing the gradient, it needs to be computed only once for all the points.

The repulsive force assumes even a simpler form

$$\hat{F}_i^{\text{rep}} = \mathscr{V}(\mathbf{y}_i)/\hat{Z}, \tag{5.6}$$

being the value of the vector field $\mathcal{V}$ in the location identified by the coordinates $\mathbf{y}_i$ normalized by $\hat{Z}$. Similarly as for $\hat{Z}$, $\hat{F}^{\text{rep}}$ has an equivalent formulation as F^{rep} but with computational and memory complexity equal to $O(N)$. So far, we assumed that $\mathcal{S}$ and $\mathcal{V}$ are computed in linear time and queried in constant time. In the next section we present how we achieve this result by using the WebGL rendering pipeline to compute an approximation of these fields.

5.3.2 Computation of the Fields

In the previous section, we formulated the gradient of the objective function as dependent from a scalar field $\mathcal{S}$ and a vector field $\mathcal{V}$. If the fields are evaluated independently, the complexity of the approach is $O(N^2)$ due to the summation in Equations 5.2 and 5.3. We achieve a linear complexity by precomputing and approximating the fields on the GPU using textures of appropriate resolution. An example of the fields for the MNIST dataset [95] is given in Figure 5.1b-d.

A similar approach is used for Kernel Density Estimation [149] that has applications in visualization [92] and non-parametric clustering [68]. In this setting, given a number of points, the goal is to estimate a 2-dimensional probability density function, from which the points were sampled. This is usually achieved by overlaying a Gaussian kernel, whose σ has to be estimated, on top of every data point. This approach is at the base of the density-based visualization of the embeddings that is presented in Section 4.4.2 in the previous chapter.

Lampe et al. [92] were the first to propose a computation of the kernel density on the GPU for a visualization purpose. They observed that the Gaussian kernel used for estimating the density has a limited support, i.e., having value almost equal to zero if they are sufficiently far away from the origin. A good approximation of the density function is then achieved by drawing, instead of the points, little quads that are textured with a precomputed Gaussian kernel. By using additive blending available in OpenGL, i.e., by summing the values in every pixel, the resulting drawing corresponds to the desired density function.

If we analyze Equations 5.2 and 5.3, we can observe that every element in the summations for both $\mathcal{S}$ and $\mathcal{V}$ have a limited support, making it indeed very similar to the Kernel Density Estimation case discussed before. The drawn functions, however, are different and Figure 5.2 shows them for $\mathcal{S}$ and $\mathcal{V}$. Therefore, we can compute the fields by drawing over a texture with a single additive drawing operation. Each point is drawn as a quad and colored with a floating-point RGB texture where each channel encodes one of the functions shown in Figure 5.2.

Contrary to the Kernel Density Estimation case, where the size of the quads changes according to the σ chosen for the Gaussian kernel, our functions have a fixed support in the embedding space. Therefore, given a certain embedding Y, the resolution of the texture influences the quality of the approximation but not the overall shape of the fields. To achieve linear complexity, we define the resolution of the target texture according to the size of the embedding. In this way, every data point updates the value of a constant number of pixels in the target texture,

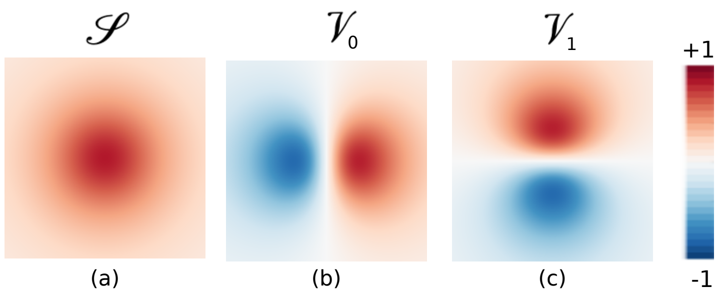


Figure 5.2: **Functions** drawn over each embedding point to approximate the scalar field $\mathscr{S}$ and the 2-dimensional vector field $\mathscr{V}$.

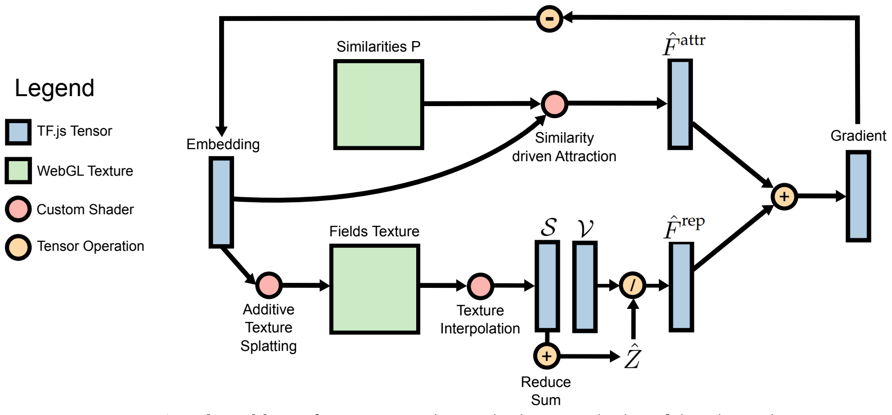


Figure 5.3: **Computational worklow** of our approach. On the lower side the of the chart, the computation of the repulsive forces is presented. The fields texture is generated by the additive texture splatting presented in Section 5.3.2. The value of $\mathscr{S}$ and $\mathscr{V}$ are obtained through a texture interpolation and are used to compute the repulsive forces. The attractive forces are computed in a custom WebGL shader that takes as input the similarities P and the embedding. The gradient of the objective function is then computed and used to update the embedding.

effectively leading to $O(N)$ complexity for the computation of the fields.

Computing the value of $\mathscr{S}$ and $\mathscr{V}$ for a point $\mathbf{y}_i$ corresponds to extracting the interpolated value in the textures that represents the fields. This operation is extremely fast on the GPU, as WebGL natively supports the bilinear interpolation of texture values. In the next section, we provide a more detailed overview of the computational pipeline as a number of tensor operations and custom drawing operations.

5.4 Implementation

In this section, we present how the ideas presented in the previous section are concretely implemented in a JavaScript library that can be used to execute an efficient tSNE computation directly in the user's browser. Figure 5.3 shows an overview of the overall approach. We rely on TensorFlow.js, a WebGL accelerated, browser-based JavaScript library for training and deploying machine-learning models. TensorFlow.js has extensive support for tensor operations that we integrate with custom shader computations to derive the tSNE embeddings.

The randomly initialized tSNE embedding is stored in a 2-dimensional tensor. We then proceed to compute the repulsive forces $\hat{F}^{\text{rep}}$ and attractive forces $\hat{F}^{\text{attr}}$, shown respectively in the lower and upper side of Figure 5.3. The attractive forces $\hat{F}^{\text{attr}}$ are computed in a custom shader that measures the sum of the contribution of every neighboring point in the high-dimensional space. The neighborhoods are encoded in the joint probability distribution P that is stored in a WebGL texture. P can be computed server-side, for example using an approximated k-nearest-neighborhood algorithm [32,34,120] as presented in the previous chapter. However, we provide a WebGL implementation of the kNN-Descent algorithm [34] and the computation of P directly in the browser to enable a client-side only computational workflow.

The repulsive forces $\hat{F}^{\text{rep}}$ are computed using the approach presented in previous sections. In a custom shader, we draw for each point, whose location is defined by the value in the embedding tensor, a quad that is textured with the functions presented in Figure 5.2. The resulting 3-channel texture, an example of which is presented in Figure 5.1b-d, represents the scalar field $\mathscr{S}$ and the vector field $\mathscr{V}$. For each embedding point $\mathbf{y}i$, the values of $\mathscr{S}(\mathbf{y}i)$ and $\mathscr{V}(\mathbf{y}i)$ are stored in tensors and are computed by a custom WebGL shader that interpolates the value of the texture in the corresponding channel. The normalization factor $\hat{Z}$ is then obtained by summing all the elements in the tensor with the interpolated values of $\mathscr{S}$, an operation that is efficiently performed on the GPU by TensorFlow.js.

The remaining computational steps are computed as tensor operations. $\hat{F}^{\text{rep}}$ is obtained by dividing the interpolated values of $\mathscr{V}$ by $\hat{Z}$, and, by adding the attractive forces $\hat{F}^{\text{attr}}$, the gradient of the objective function is obtained. The gradient is then added to the embedding, hence, modifying the position of the points according to their similarities. Our work is released as part of the TensorFlow.js library and can be found on GitHub at the following address: `https://github.com/tensorflow/tfjs-tsne`

5.5 Conclusion

In this chapter, we presented a novel approach for the optimization of the objective function of the tSNE algorithm that scales to large datasets in the client side of the browser. Our approach relies on modern graphics hardware to efficiently compute the gradient of the objective function from a scalar field that represents the point density and the directional repulsive forces in the embedding space. The implementation of the technique is based on the TensorFlow.js library and can be found on GitHub at the following address: `https://github.com/tensorflow/tfjs-tsne`. Examples that validate our approach can also be found on GitHub `https://github.com/tensorflow/tfjs-tsne-examples`.

In this chapter and in Chapter 4 we presented two techniques that improve on the scalability of the computation of the tSNE algorithm. These improvements are particularly needed in a Progressive Visual Analytics context, where the user cannot wait hours, or even days, before the 2-dimensional embedding is computed. However, after extensive use of the Approximated-tSNE technique, we came to realize that the user is confronted with a different kind of scalabity issue. When millions of data points are embedded and visualized on a computer screen, it becomes increasingly difficult to identify sub-clusters within tSNE embeddings. This limitation hinders the ability of the user of effectively explore and collect insights on the data at hand. In the next chapter we introduce the Hierarchical Stochastic Neighbor Embedding [136] technique, a multiscale approach for dimensionality reduction that is designed to address this problem.

6 Hierarchical Stochastic Neighbor Embedding

There is grandeur in this view of life, with its several powers, having been originally breathed into a few forms or into one; and that, whilst this planet has gone cycling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being, evolved.

Charles Darwin

In this chapter we present the Hierarchical Stochastic Neighbor Embedding algorithm (HSNE). HSNE builds a hierarchical representation of the data that is then explored using non-linear dimensionality-reduction embeddings. The exploration follows the overview-first and details-on-demand mantra, enabling the interactive exploration of extremely large datasets. We demonstrate HSNE on several datasets and we show the application potential in the visualization of Deep-Learning architectures and the analysis of hyperspectral images.

N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical Stochastic Neighbor Embedding. In Computer Graphics Forum, volume 35, pages 21–30, 2016 [139].

6.1 Introduction

In *Exploratory Data Analysis*, a number of visualization techniques are used to support the hypothesis-generation process. Among its goals are the extraction of important variables, the detection of outliers or the identification of underlying non-convex structures [172]. As seen in the previous chapters, *Non-linear dimensionality reduction* techniques such as tSNE play a key role in the understanding of high-dimensional data [22, 155]. A simple example is presented in Figure 6.1a, where a non-convex 1D manifold structure is defined in a 2D space. Non-linear dimensionality reduction is used to generate a 1D embedding (Figure 6.1b). Note that a linear transformation cannot project the manifold on such a 1D space.

As presented in Chaper 2, in recent years the application of non-linear dimensionality reduction techniques on real-world data led to new findings as complex real-world phenomena lead to non-convex structures that resides in a high dimensional space [6,12]. Algorithms such as Sammon Mapping [152], LLE [150], ISOMAP [169] or tSNE [177] help during Exploratory Data Analysis by giving a meaningful representation of these high-dimensional spaces. Broadly, two different approaches have been developed by the Machine-Learning and the Visualization community. The Machine-Learning approach tends to focus on accurate but computationally-expensive techniques, whereas the Visualization approach often trades accuracy and non-convex structure preservation for interactivity. Consequently, the first type is often too slow for interactive interfaces, limiting the ability to support the hypothesis generation process. The second type is less accurate and can generate non-existing structures. For example, *hybrid* approaches use a set of *landmarks*, also called *pivots* or *control points*, which are embedded using non-linear dimensionality-reduction techniques. The remaining points are placed by interpolating their positions. Due to the sparse amount of landmarks, this process may not reflect the underlying manifold. An example is given in Figure 6.1c. The landmarks are placed in the wrong order according to the manifold, if the rest of the data is not taken into account. This problem can be partly remedied by letting the user manipulate the landmark positions in the embedding. However, this interaction cannot avoid the creation of non-existing structures and requires prior knowledge of the user about the data, which is usually not available.

In this chapter we present the Hierarchical Stochastic Neighbor Embedding al-

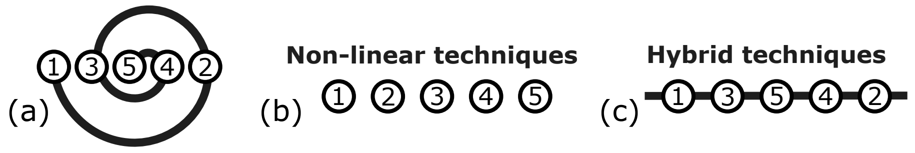


Figure 6.1: **Dimensionality reduction with landmarks**. In non-linear embedding techniques the underlying manifold (a) is respected (b). In *hybrid* approaches, landmarks are placed without considering the underlying manifold (c) and data points are placed by interpolating the landmark positions (grey line in c). The layout quality thus relates to the used number of landmarks.

gorithm (HSNE), a non-linear dimensionality reduction technique that aims at bridging the gap between accuracy and interactivity. It is motivated by the good results that SNE techniques show in user studies [155] and is as fast as the state-of-the-art *hybrid* techniques. While our approach also involves landmarks, it differs significantly from previous work. Our landmarks are enriched by a smooth and non-convex *area of influence* on the data and the landmarks are chosen by analyzing the data points and their k-nearest neighbor graph, while avoiding outliers. Overlaps in the *areas of influence* are used to encode similarities between landmarks. Our process is hierarchical and landmarks at a higher scale are always a subset of the previous scale. This hierarchy allows us to keep the memory footprint small, while enabling a new way of analyzing the data. We follow the *Overview-first*, *Details-on-Demand* paradigm [158] for the analysis of non-linear embeddings. Dominant structures that appear in the *Overview* can be analyzed by generating an embedding of the related landmarks in the subsequent lower scale. In this way, the user can drill down in the data and search for structures at finer scales. It is an approach that scales very well to big datasets and we illustrate its application potential in two different use cases in this chapter. The success of this technique are also shown if Chapter 7, 8 and 9.

The remainder of the chapter is structured as follows. After an overview of the related work, Section 8.4.3 presents the HSNE algorithm with a focus on the construction of the hierarchy, while the hierarchical analysis is presented in Section 6.4. Finally, Section 6.5 contains two use cases showing the potential of our method, while experiments on well known datasets are presented in Section 6.6.

6.2 Related Work

Linear dimensionality-reduction techniques try to preserve global distances between data points in the embedding as in the high-dimensional space. Hierarchical implementations of these techniques have been developed to reduce calculations. Notable examples are Glimmer [75], Steerable MDS [186] and HiPP [128] that linearly separate the space with a top-down approach.

Differently from linear algorithms, non-linear dimensionality reduction techniques try to preserve geodesic distances on manifolds between data points. However, a simple case as in Figure 6.1a is rarely met in practice, and the definition of geodesic distances is a challenging task. In real-world data, data points form manifolds defined by sets of points varying in size, density, shape and intrinsic dimensionality. A class of techniques known as Stochastic Neighbor Embedding (SNE) [66] is accepted as the state of the art for non-linear dimensionality reduction for the exploratory analysis of high-dimensional data. Intuitively, SNE techniques encode small-neighborhood relationships in the high-dimensional space and in the embedding as probability distributions. These techniques aim at preserving neighborhoods of small size for each data point. The embeddings are defined via an iterative minimization of the loss of information when placing the point in the embedding. Besides the discoveries made using algorithms like tSNE [6,12], the ability to reveal interesting structures is demonstrated by extensive user studies on real-world and synthetic data [155]. Unfortunately, the application of SNE techniques to large datasets is problematic, as the computational complexity is usually $O(n^2)$. Using approximations it can be reduced to $O(n\log(n))$ [138, 176]. Furthermore, small-neighborhood preservation might miss structures at different sizes. Our HSNE is an SNE technique, which overcomes the computational complexity and shows structures at different scales by creating a hierarchical representation of the dataset. Differently from other hierarchical techniques [75,128,186], we use a bottom-up approach in the creation of the hierarchy. Our key insight is to use landmarks that represent increasingly large portion of the data.

The usage of landmarks is not new and can be separated in two categories, which we refer to as the *non-linear* and *hybrid* landmark techniques (see Figure 6.1). Both select a set of landmarks from the original dataset. *Non-linear* landmark techniques embed them using metrics that estimate geodesic distances between points [161, 177]. Figure 6.1b shows a simple example, where the neigborhood relationship are extracted using the geodesic distances on the manifold. For example, Landmark-tSNE creates the K-Nearest Neighbor (KNN) Graph between the original data point and computes for each landmark the probability of reaching other landmarks with a random-walk on the KNN-Graph [177]. Non-linear landmark techniques can discover non-convex structures, but their scale is directly related to the number of selected landmarks. Further, the user is limited to the visualization of landmarks and not the complete dataset, limiting the insights that can be extracted from the data. *Hybrid* landmark techniques embed landmarks with non-linear dimensionality reduction techniques based on high-dimensional descriptors of the landmarks derived from the original data. The complete dataset is then embedded using different interpolation schemes [33, 39, 80, 127, 129, 131, 132]. This approach is widely used by the visualization community due to its fast computation, making it ideal for interactive systems. However, non-convex structures are not preserved (unless the sampling is dense enough) because the underlying manifold is ignored. Figure 6.1c illustrates the problem: the selected landmarks are seen as a straight line even by a non-linear technique.

HSNE is a *non-linear* landmark technique, but supports the exploration of non-convex structures at different scales, while sharing the performance of *hybrid* techniques and supporting interaction to gain insights into the data. In particular, our novel hierarchical approach using an *Overview-first*, *Details-on-Demand* paradigm helps in this context.

6.3 Hierarchical Stochastic Neighbor Embedding

Here, we present our HSNE technique with a focus on the creation of the hierarchical data representation. An overview is given in Figure 6.2. Throughout the chapter, calligraphic notations indicate sets, for example, $\mathscr{D}$ is the set of high-dimensional data points. Our representation is composed of different scales, or levels, organized hierarchically. We use superscripts to indicate this scale. Elements in sets

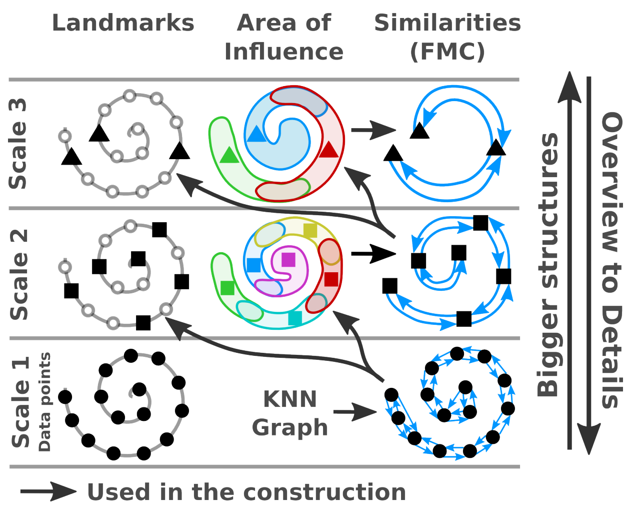


Figure 6.2: **Overview of the hierarchy construction**. A Finite Markov Chain (FMC) is built from the k-nearest neighbor graph. The FMC encodes the similarities between landmarks and it is used for selecting landmarks in the next scale. The FMC is also used to compute the *area of influence* of the selected landmarks on the landmarks in the lower scale. The overlap between the *areas of influence* is used to build a new FMC that encodes similarities in the new scale.

are identified using subscripts. We denote $\mathscr{L}^s$ the set of landmarks representing the dataset at scale s. $\mathscr{L}^1$ represents the first scale, which is the input dataset $\mathscr{D}$. Higher scales are always subsets of previous scales ($\mathscr{L}^s \subset \mathscr{L}^{s-1}$).

Our algorithm works as follows. Starting with $\mathscr{L}^1$, we build a Finite Markov Chain (FMC) from a k-nearest-neighbor graph to encode similarities between landmarks (Section 6.3.1). It is used to guide the selection process of a landmark subset for the next scale (Section 6.3.2) and, then, to compute an *area of influence* for each selected landmark (Section 6.3.3). The overlap between these areas indicates similarity and forms the basis for a new FMC encoding (Section 6.3.4), which is then used to compute the next scale. After preprocessing the different scales, we can perform a multi-scale analysis by computing an embedding of landmarks using their scale-dependent information (Section 6.3.5).

6.3.1 From data points to a Finite Markov Chain

A Finite Markov Chain is a random process that undergoes transitions from one state to another in a state space. Our Finite Markov Chain is used to model the random movement of a hypothetical particle on the manifold, and the states are given by the landmarks in $\mathscr{L}^s$. The transitions are encoded in a square *transition matrix* T^s of size $|\mathscr{L}^s| \times |\mathscr{L}^s|$. $T^s(i,j)$ represents the probability that the landmark $\mathscr{L}^s_j$ belongs to the neighborhood of the landmark $\mathscr{L}^s_i$ in the scale s. It is important to note that HSNE aims at encoding small neighborhoods of fixed size for every landmark.

Therefore T^s is sparse by construction, and its memory complexity grows linearly with the size of the dataset.

For the Finite Markov Chain described by the transition matrix T^1, each data point $\mathscr{D}_i$ is only allowed to transition to a data point $\mathscr{D}_j$, if $\mathscr{D}_j$ belongs to the k-nearest-neighborhood $\mathscr{N}(i)$ of $\mathscr{D}_i$. The probability assigned to the transition is given by the following equation:

$$T^1(i,j) = \frac{\exp(d(i,j)^2/\sigma_i)}{\sum_k \exp(d(i,k)^2/\sigma_i)} \text{ with } j,k \in \mathscr{N}(i), \tag{6.1}$$

where $d(i,j)$ are the Euclidean distances between data points, and σ_i is chosen such that $T^1(i,-)$ has perplexity of $|\mathscr{N}(i)|/3$ [176]. The exponential falloff is used to reduce the problem caused by the presence of outliers, that act as *shortcuts* across manifolds. SNE techniques focus on the preservation of small neighborhoods for each data point. Thus, a small value of K is usually selected, where 100 is a common choice [176, 177]. To improve performance, we adopt the approximated algorithm for the computation of the k-nearest-neighborhoods proposed in the Approximated-tSNE introduced in Chapter 4. Experimentally, we see that such an algorithm does not compromise the quality of the embeddings generated by HSNE while improving the computation time by two orders of magnitude. The computational complexity of this first step is $O(|\mathscr{D}|\log(|\mathscr{D}|))$

6.3.2 Landmark selection and outliers identification

We use the transition matrix to carefully select meaningful landmarks in order to reduce the size of the dataset. This step is of crucial importance, e.g., in order to avoid choosing outliers as landmarks. So far, we have only given the definition of the transition matrix for the lowest scale. We define it for other scales in Section 6.3.4. Nonetheless, the process described here is valid at all scales, which is why we use the superscript s to indicate its generality. Before we explain our sampling solution, we introduce the concept of *equilibrium distribution* of a Finite Markov Chain. A vector π is called *equilibrium distribution* of the Finite Markov Chain, described by the transition matrix T^s, if it represents a probability distribution that is not changed by a transition in the state space:

$$\pi = \pi T^s \text{ and } \sum_i \pi_i = 1 \tag{6.2}$$

Intuitively, the *equilibrium distribution* π represents the probability of being in a state after an infinite number of transitions in the state space. These transitions are often called *random walks* in the state space. Given the transition probabilities defined by Equation 6.1, the *equilibrium distribution* of our Finite Markov Chain assigns higher probability to data points that reside in high-density regions in the original space. Figure 6.3 shows an example, where the landmarks $\mathscr{L}^s$ are color coded according to the *equilibrium distribution* of the Finite Markov Chain that encodes their similarities. Landmarks in dense regions of the space, have high value of π and are

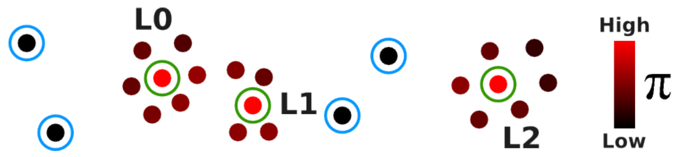


Figure 6.3: **Selection of landmarks and outliers** using the *equilibrium distribution* π of the Finite Markov Chain (see Equation 6.2). Points are color coded from black to red according to their π-value. Selected landmarks are circled in green, while potential outliers are circled in blue.

selected to be in $\mathscr{L}^{s+1}$ (green circles in Figure 6.3). Landmarks with a low value of π are considered outliers in scale $s+1$ (blue circles in Figure 6.3).

Landmarks in $\mathscr{L}^{s+1}$ are selected by sampling the *equilibrium distribution* π, that is computed using a simple Markov Chain Monte Carlo technique [50]. For each landmark in $\mathscr{L}^{s}$, we start β random walks of fixed length θ. Every landmark that is the endpoint of at least $\beta_{\text{treshold}} * \beta$ random walks is selected as a landmark in $\mathscr{L}^{s+1}$, if no random walks reach a given landmark, it is detected as outlier. We experimented with different values of β and θ, finding that $\beta = 100$ and $\theta = 50$ is a good compromise between speed and accuracy for the data we have been analyzing. Notice that the computation of random walks is not costly, and thousands can be performed every millisecond on a state-of-the-art desktop computer. We provide a default value of $\beta_{\text{treshold}} = 1.5$, that we found is conservative enough to create a hierarchical representation for all the dataset that we tested. The computation complexity of this step is $O(|\mathscr{L}^{s}|)$. However, the user can change this value to control the number of landmarks to be selected.

6.3.3 Area of influence

The process of choosing landmarks cannot be simply relaunched, as we would then loose important information from previous scales. In consequence, we will extend the definition of the transition matrix to all scales beyond the first. To this extent, we introduce the *area of influence* for each landmark, which keeps track of a landmark's impact on previous scales. The influence exercised by landmarks in $\mathscr{L}^{s}$ on those in $\mathscr{L}^{s-1}$, is encoded in an *influence matrix* I^s. Matrix I^s has size $|\mathscr{L}^{s-1}| \times |\mathscr{L}^{s}|$, where $I^s(i,j)$ is the probability that the landmark $\mathscr{L}^{s-1}_i$ in the previous scale is well represented by $\mathscr{L}^{s}_j$. Specifically, each row i is a probability distribution that denotes the probability that the landmark $\mathscr{L}^{s-1}_i$ is in the area of influence of landmarks in $\mathscr{L}^{s}$. Consequently, the influence of a scale s on scale r is defined by a chain of sparse matrix multiplications:

$$I^{r \leftarrow s} = \left[\prod_{i=s}^{r} \left(I^i\right)^t\right]^t \quad \text{with } r < s \tag{6.3}$$

It is important to note that the area of influence is localized, implying that I^s is sparse. Therefore, the memory complexity grows only linearly with the set of landmarks. To compute I^s, we start a number of random walks in the Finite Markov

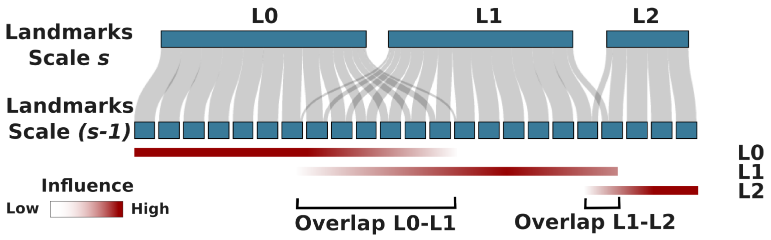


Figure 6.4: **The area of influence** can be seen as flow converging in landmarks of the higher scale. The area of influence of the landmarks selected in Figure 6.3 is shown here. The overlap in the area of influence is used to compute similarities between landmarks (see Equation 6.5).

Chain described by T^{s-1} for each landmark $\mathcal{L}^{s-1}$, leading to a computational complexity of $O(|\mathcal{L}^{s-1}|)$. The random walk stops when a landmark in $\mathcal{L}^s$ is reached. The percentage of random walks reaching every landmarks in $\mathcal{L}^s$ is then used as a row for $I^s(i,-)$. Figure 6.4 shows the area of influence of the selected landmarks in Figure 6.3 as a flow, converging in landmarks of the higher scale. Depending on the data distribution in the high-dimensional space, landmarks can exercise influence on regions of different size. We define the *weight* of a landmark as the size of the region that it represents. The vector W^s encodes the weights of the landmarks at scale s, and it is defined by the following equation:

$$W^s = W^{s-1} * I^s \text{ with } W^1 = \mathbf{1} \tag{6.4}$$

The width of the landmarks in Figure 6.4 represents these weights W^s.

6.3.4 From areas of influence to Finite Markov Chains

Similarities between landmarks in scale s are computed using the overlaps in their *areas of influence* on scale $s-1$. Intuitively, if the areas of influence of two landmarks overlap, it means that they are close on the manifold, therefore their similarity is high. We use the influence matrix I^s to create the FMC, encoding similarities between landmarks in $\mathcal{L}^s$. The transition matrix T^s is given by the following equation:

$$T^s(i,j) = \frac{\sum_{k=1}^{|\mathcal{L}^{s-1}|} I^s(k,i) I^s(k,j) W^{s-1}(k)}{\sum_{k=1}^{|\mathcal{L}^{s-1}|} \sum_{l=1}^{|\mathcal{L}^s|} I^s(k,i) I^s(k,l) W^{s-1}(k)} \tag{6.5}$$

where $I^s(k,i)I^s(k,j)W^{s-1}(k)$ is the overlap of the area of influence of $\mathcal{L}^s_j$ and $\mathcal{L}^s_i$ on landmark $\mathcal{L}^{s-1}_k$. Figure 6.4 depicts overlaps between the *areas of interest* of the landmarks selected in Figure 6.3. The overlap between L0 and L1 is higher than the overlap between L1 and L2, as expected because L1 is more similar to L0 than L2.

6.3.5 Generation of the embedding

SNE methods rely on a probability distribution P, that encodes neighborhood relationships. In practice, we rely on tSNE because of its ability to overcome the so called *crowding problem* [177]. tSNE interprets similarities between data points as a symmetric joint-probability distribution P. Likewise a joint-probability distribution Q is computed, that describes the similarities in the low-dimensional space. The goal is that the position of the data points in the embedding faithfully represent the similarities in the original space. The iterative minimization of the Kullback-Leibler divergence is used to reduce the information loss when Q is used to represent P. An in depth explanation on how tSNE computes Q and minimizes the divergence function is presented in Section 3.1. In our HSNE, P is computed from the transition matrix T^s:

$$P(i,j) = \frac{T^s(i,j) + T^s(j,i)}{2|\mathscr{L}^s|} \text{ where } \sum_{i,j} P(i,j) = 1 \tag{6.6}$$

With this definition, an embedding can be computed even for a subset of the landmarks, the only requirement is that their similarities are encoded in a Finite Markov Chain. This observation is important as it enables the *Overview-First, Details-on-Demand* analysis presented in the next section. However, if the user is interested in generating a complete embedding (as in *hybrid* techniques), it can be achieved by interpolating the position of the landmarks in the top scale o:

$$\mathscr{Y}_i^1 = \sum_j^{\mathscr{L}^o} \mathscr{Y}_j^o I^{1\leftarrow o}(i,j) \tag{6.7}$$

where $I^{1\leftarrow o}(i,j)$ is the influence exercised on the data points, as shown in Equation 6.3.

6.4 Hierarchical Analysis

In this section, we describe how the hierarchical analysis is performed by presenting how the detailed embeddings are generated by *filtering* and *drilling-down* in the data. Before addressing the algorithmic solution, we will motivate the usefulness of such a tool with an example.

6.4.1 Example of a hierarchical analysis

Standard dimensionality reduction techniques are often used to enable a user to visually identify groups of similar data points. This possibility is useful, as it enables tasks, such as *verification*, *naming* or *matching* [22]. Figure 6.5a, shows a simple *naming task* using readings from atmospheric sensors as high-dimensional data. Figure 6.5b shows another example, in the context of traditional analysis. In a naming task, the analysis of the given data might lead to a set of different clusters. A user could inspect these clusters by selecting one and seeing the corresponding region highlighted on the map. Using prior knowledge for a few locations, it becomes possible to attribute conditions, such as sunny, cloudy and rainy weather, on the entire map. Nonetheless, such an analysis assumes that the scale of the clustering was sufficiently precise and not overly precise.

The hierarchical nature of our approach enables a new multi-scale analysis based on the *Overview-First*, and *Details-on-Demand* mantra [158]. An example is given in

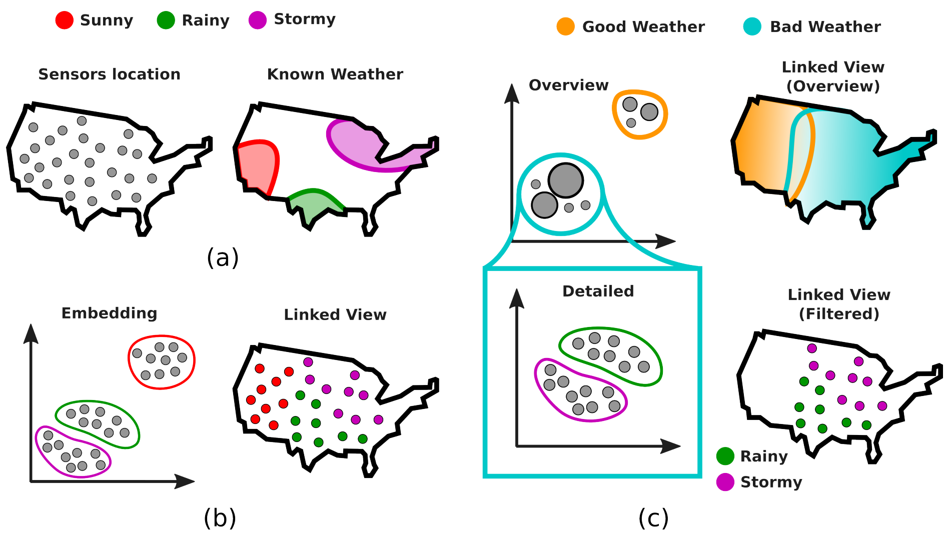


Figure 6.5: **Traditional vs Hierarchical analysis**. (a) High-dimensional readings from sensors located on a map and prior knowledge on the phenomenon of interest are available to the user. In the traditional analysis (b) a single embedding is generated and analyzed. In our hierarchical analysis (c), an overview shows dominant structures in the dataset. Detailed embedding of the structures are created by filtering and drilling into the data.

Figure 6.5c. Instead of showing an embedding of all data points, the analysis starts with the highest-scale landmarks. The resulting clusters will represent very coarse dominant structures, for example, good and bad weather zones. Additionally, the area of influence encoded in the size of the embedded points gives feedback regarding the complexity of the original underlying data. If a user now wishes to explore more detailed information, a cluster can be selected and a lower scale embedding is produced. The heterogeneous data on the lower level then becomes visible, for example, bad weather transforms into cloudy and rainy regions. Our approach is particularly suited for heterogeneity at different scales, which is common in large datasets.

6.4.2 Filtering and drill down

To enable the investigation of details, we start from a selected subset $\mathcal{O}$ of landmarks at scale s: $\mathcal{O} \subset \mathcal{L}^s$. We drill in the data by selecting a subset $\mathcal{G}$ of landmarks at scale $s-1$: $\mathcal{G} \subset \mathcal{L}^{s-1}$, using the influence matrix I^s to connect the two scales. As explained in Section 6.3.3, a row i in I^s represents for $\mathcal{L}^{s-1}_i$ the influence of the landmarks $\mathcal{L}^s$ at scale s. We define F_i as the probability that landmark $\mathcal{L}^{s-1}_i$ is in the area of influence of the landmarks in $\mathcal{O}$:

$$F_i = \sum_{\mathcal{L}^s_j \in \mathcal{O}} I^s(i,j) \tag{6.8}$$

If all the landmarks influencing $\mathcal{L}_i^{s-1}$ are in $\mathcal{O}$, then $F_i = 1$. If no influence from $\mathcal{O}$ is exercised on $\mathcal{L}_i^{s-1}$ then $F_i = 0$. A landmark $\mathcal{L}_i^{s-1}$ is selected to be in $\mathcal{G}$ if $F_i > \gamma$, where γ is a user-defined threshold. However, it should be noted that a low value of γ is not desirable, as it will add landmarks, which are only slightly influenced by $\mathcal{O}$. A high value of γ is also not desirable, leading to the exclusion of regions that are highly influenced by $\mathcal{O}$. While it remains a parameter, we found experimentally that $\gamma = 0.5$ allows for effective drilling in the data. The transition matrix $T_{\mathcal{G}}^{s-1}$, representing the similarities in $\mathcal{G}$, is derived from T^{s-1} by removing the rows and columns of landmarks in L^{s-1}, which are absent from $\mathcal{G}$. Given the transition matrix, the embedding is computed as before (Section 6.3.5).

6.5 Use cases

Here, we show examples for our hierarchical analysis on real-world data, to illustrate our contributions and potential application areas of our HSNE. Besides high-resolution hyperspectral images of the Sun and remote-sensing data, we visualize the training set of a well-known Deep Learning model, showing how it interprets the input images. We demonstrate the HSNE's ability to show dominant structures in the *Overview* and to explore them in detailed embeddings to reveal finer-grained structures. We test our C++ implementation of HSNE on a DELL Precision T3600 workstation with a 6-core Intel Xeon E5 1650 CPU @ 3.2GHz and 32GB RAM.

6.5.1 Hyperspectral images

The visible light spectrum is only a tiny part of the electromagnetic spectrum and some phenomena can only be understood by considering the complete spectrum. Figure 6.6a, shows hyperspectral images of the sun. Different wavelengths of the electromagnetic spectrum reveal different observations, such as solar flares or the corona. The image resolution is 1024×1024, leading to a dataset composed of $\approx 1M$ data points (pixels). Each pixel is described by 12 dimensions corresponding to the intensity readings. We downloaded the data from the Solar Dynamics Observatory[1] on November 13th 2015. In an Exploratory Data Analysis the user needs to analyze all pictures of all wavelengths in parallel. However, with an increasing number of images, the data complexity complicates the generation of a hypothesis or the differentiation of different regions. Here, we show how HSNE supports such analysis.

The hierarchical representation of the data is precomputed in 2’13" minutes and only needs to be processed once. From this representation the overview and detailed embeddings require only a few seconds and can be visualized using a Progressive Visual Analytics approach [138]. Figure 6.6b shows the *Overview* generated by HSNE. The *Overview* is composed of 352 landmarks in two clusters (T0 and T1). Every landmark is drawn using semi-transparent circles, while the size of a landmark encodes its weight as defined in Equation 6.4. The clusters correspond to two dominant structures in the data, the Sun surface (T1) and the Space in the

[1]http://sdo.gsfc.nasa.gov/

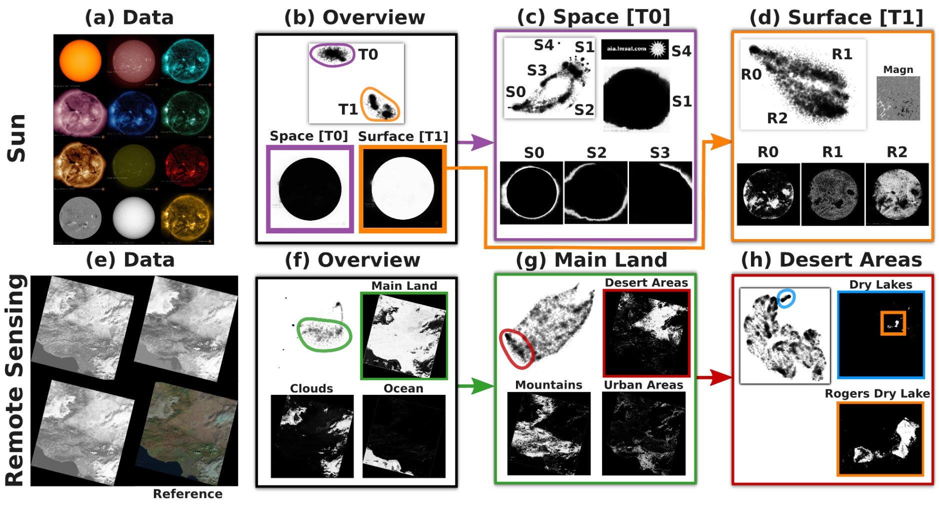


Figure 6.6: **Hierarchical analysis of hyperspectral images**. Hyperspectral images of the Sun and the area surronding the city of Los Angeles are analyzed using HSNE. Dominant structures are revealed at different scales and can further inspected by creating detailed embeddings.

background (T0). Their *areas of influence* is visualized in the linked view. Here, an image of size 1024×1024, where a greyscale colormap is used to represent the probability of a pixel to belong to the *area of influence* of the selection. The user drills in the data by requesting detailed visualizations of the two dominant structures. A detailed embedding of T0 (Figure 6.6c) describes different regions of the Corona. S0 represents the area close to the surface, while S1 represents the background. S2 and S3 represent the external area of the Corona, where S3 is an area with low readings in the AIA 211 channel (pink in Figure 6.6a). S4 is an interesting cluster, representing the overlayed logo, present in all images. S4 is considered an outlier in the overview and, therefore, was not represented as a separate cluster. However, upon refinement, this cluster would appear, as it will be a dominant structure at this scale. A detailed embedding of T1 leads to three clusters (Figure 6.6c). Although not as well separated, they clearly represent different regions on the Sun surface. R0 are hotter regions, or where solar flares are visible, while R1 and R2 represent colder regions separated in one of the input images, namely the *Magn* image (Figure 6.6d).

We performed a similar analysis on hyperspectral images for remote sensing. These data are captured by the LandSat satellites [2], and we present an example of the area surrounding the city of Los Angeles. The data are composed of 11 images, representing different bands of the electromagnetic spectrum. Figure 6.6e shows three of such images, and a reference image. Similarly to the previous ex-

[2]http://landsat.usgs.gov/

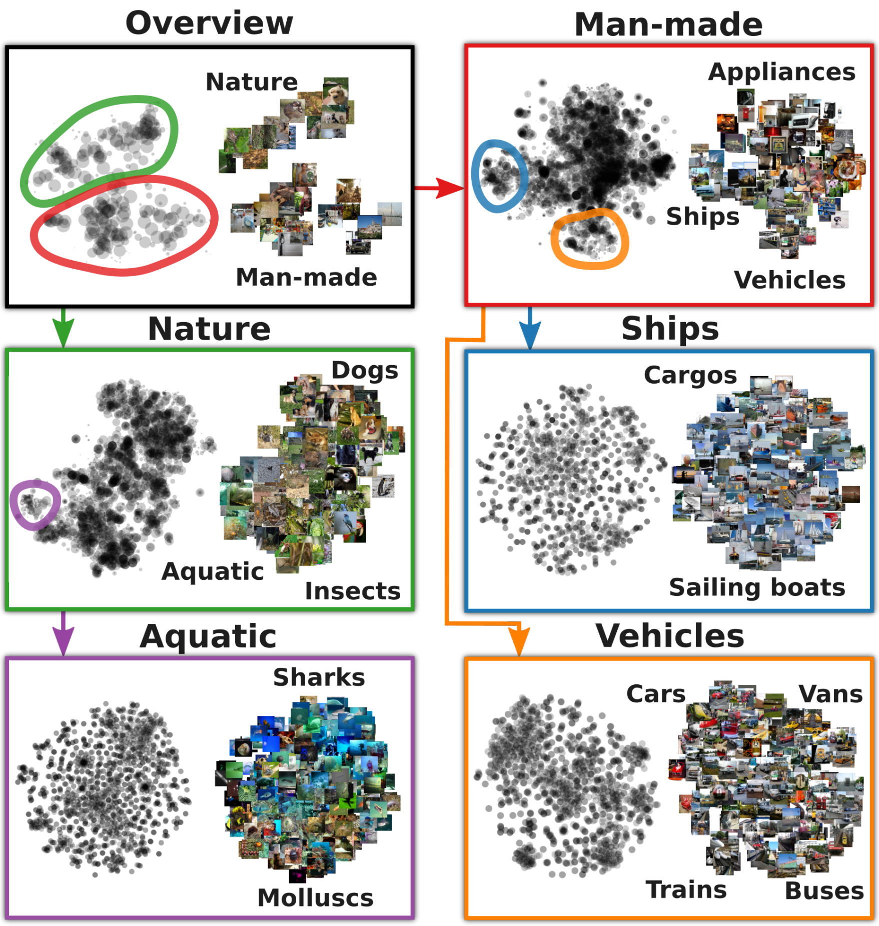


Figure 6.7: **Deep Learning models**. Features are extracted from 100k images using a Deep Neural Network (DNN) [88] and the hierarchical analysis is performed using HSNE. Starting from the overview, dominant structures at different levels are revealed. The user can inspect the embeddings and request detailed visualization. This is achieved through filtering of the landmarks and by drilling down in the hierarchy. A high-resolution version of the figure is provided in the supplemental materials.

ample, we analyzed the images at a resolution of 1024×1024. Figure 6.6f shows the dominant structures in the highest scale, namely *ocean*, *clouds* and the *main land*, that are identified by the user by looking at the reference image and using its prior knowledge on the phenomenon. A detailed embedding representing the *main land* is shown in Figure 6.6g. It is possible to identify different parts of the detailed embeddings related to *mountains*, *urban* and *desert areas*. Drilling in, detailed embeddings are generated, such as the one representing *desert areas*, as depicted in Figure 6.6h. More heterogeneity is revealed at this scale. For instance dry lakes, such as the *Rogers Dry Lake*, are located in the cluster of desert areas.

6.5.2 Visualization of Deep Learning datasets

Deep Learning builds upon neural networks composed of many (hence, the name *deep*) layers. Deep Neural Networks (DNN) achieved impressive results in image recognition, sentiment analysis and language translation. For an overview of the field, we refer to [94]. However, it is difficult to visualize how a DNN works. An approach that was used recently, is to select some potential inputs that are processes by the DNN [82, 116]. For each input, the values computed by the last layer of the network are used as high-dimensional descriptor. Once that the descriptor are assigned to each data point, they are embedded in a 2D space using non-linear dimensionality reduction techniques. tSNE is usually selected for such a task [82,116]. The limitation of this approach is that only small subsets can be visualized at a given time, limiting the ability to evaluate and inspect how the network is trained. We extract features from the test set of a well known DNN [88], leading to a dataset consisting of 100k images and 4096 dimensions. The hierarchical representation of the data is computed in 92 seconds, while every embedding requires only few seconds to be computed. Our approach shows the hierarchical nature of the learning process, as depicted in Figure 6.7. In the overview two clusters are visible. We label them as *Man-made* and *Nature*, based on the inspection of the images represented by the landmarks. Detailed embeddings of the clusters are produced and confirm the previous labeling. In the *Nature* cluster new dominant structures are revealed, such as images of *Aquatic* animals, *Insects* or *Dogs*. Similarly, a detailed visualization of the landmarks labeled as *Man-made* reveal more heterogeneity in the data. The user can drill deeper in the data, for example by requesting detailed visualization of landmarks identified as *Ships*, *Vehicles* and *Underwater* scenes *Acquatic* animals.

6.6 Evaluation

In this section we provide experimental evidence that HSNE outperforms *hybrid* and *non-linear* dimensionality reduction techniques. In our evaluation, we use the MNIST dataset [3] (60k points, 784 dimensions), the CIFAR-10 dataset [4] (50k points, 1024 dimensions) and the TIMIT dataset [5] (1M points, 39 dimensions).

Figure 9.8 shows the embeddings of the MNIST dataset produced with our approach compared to those created by *non-linear* techniques (tSNE and L-SNE [177]) and *hybrid* techniques (LSP [129], Piecewise-LSP [127], created by the Projection Explorer tool [130], and LAMP [80] created by the Projection Analyzer tool [6]). Our HSNE embedding is computed for three scales, resulting in the highest-level embedding

[3]http://yann.lecun.com/exdb/mnist/

[4]https://www.cs.toronto.edu/ kriz/cifar.html

[5]https://catalog.ldc.upenn.edu/LDC93S1

[6]https://code.google.com/archive/p/projection-analyzer/

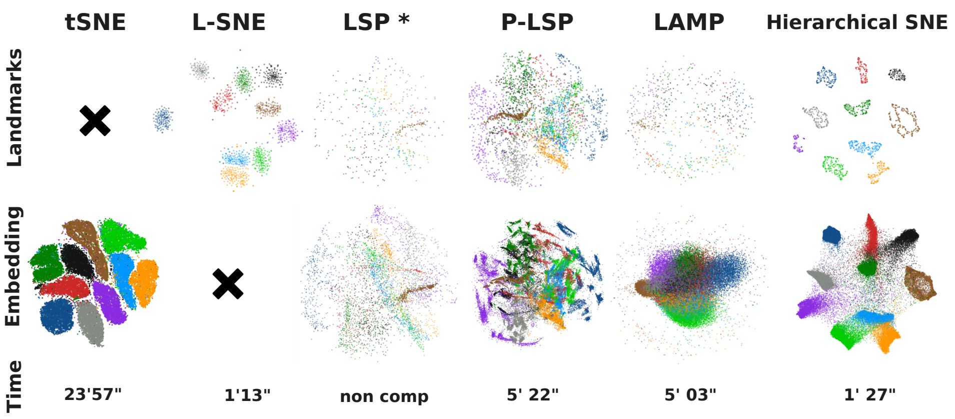


Figure 6.8: **Embeddings of the MNIST dataset** created by non-linear dimensionality reduction techniques (tSNE and Landmark-SNE) and by *hybrid* techniques (LSP, P-LSP and LAMP). Differently from *hybrid* techniques, HSNE preserves the manifold in the landmark embedding, creating compact clusters in the complete embedding.

containing 1431 landmarks. The tSNE embedding is computed using approximated computations [138, 176] to reduce the computational complexity to $O(n\log n)$. For the L-SNE algorithm, we randomly selected 1431 landmarks and we use approximated k-nearest-neighbor computations (see Section 6.3.1), making it comparable to the setting for the HSNE. We were not able to generate a LSP embedding of the MNIST dataset due to its size and present an embedding of 5k randomly selected data points instead. We use the default parameters for the selection of the landmarks, leading to 500 landmarks in LSP, 3714 in P-LSP and 734 in LAMP. For each technique we present, where available, the embedding containing only the landmarks, as well as the complete embedding. Our HSNE is much faster than tSNE and comparable to *hybrid* techniques.

We base our quantitative assessement of the quality of the embedding on the *Nearest-Neighbor Preservation* metric (NNP) as proposed by Venna et al. [181] and implemented by Ingram and Munzner [74]. For each data point, the K-Nearest-Neighborhood (KNN) in the high-dimensional space is compared with the KNN in the embedding. Average precision/recall curves are generated by taking into account high-dimensional neighborhoods of size $K_{high} = 30$ [74]. The precision/recall curves are computed by selecting K_{emb}-neighborhoods in the embedding, iterating K_{emb} from 1 to K_{high} and computing the true positive TP in the K_{emb}-neighborhood. The precision is set as TP/K_{emb} and the recall as TP/K_{high}. The curve is obtained by connecting the points in the precision/recall space for each value of K_{emb} [74]. However, NNP fails to measure the preservation of high-level information, e.g. neighborhood preservation in a geodesic sense and, to the best of our knowledge, no such metric exists. Therefore, we assess the high-level structure preservation both by a visual inspection of the labeled data and by the evaluation of the NNP during the drill-down in the data. Intuitively, if HSNE does not have the ability to preserve high-level structures, during a drill-down part of the data will be left out, leading to gaps in the lowest-level embedding and, consequently, to bad NNP.

Even if a validation of the visual cluster cannot be performed, given its non-convex nature [9], the MNIST dataset contains compact manifolds [177] that represent handwritten digits (see examples in Figure 9.8). Therefore, based on the visual separation of the labeled landmarks, we can conclude that HSNE preserves manifolds similar to non-linear dimensionality-reduction algorithms. Hybrid techniques are incapable of well separating the manifolds in this example. Due to the fact that the underlying manifold is not respected, the landmark positions in the embedding ignores local structures in the data, leading to problems similar to the one depicted in Figure 6.1c.

HSNE separates the manifolds even better than tSNE, see orange cluster in the tSNE embedding compared to orange landmarks in the HSNE embedding. This result is a consequence of tSNE focusing only on the preservation of small neighborhoods. When the size of the data increases, we experimentally found that minimization performed by tSNE will often incur in local minima that disrupt the visual representation of high-level manifolds.

tSNE's ability to preserve small neighborhoods is confirmed by the NNP precision/recall curves presented in Figure 6.9a. For HSNE we compute a precision/recall curve for each scale by linearly interpolating the data points using landmarks in the corresponding scale, as in Equation 6.7. In the highest scale, HSNE outperforms the other *hybrid* techniques but it performs worse than tSNE. This is expected as the information preserved by HSNE at this scale is not measured by NNP. When the lowest scale is considered, the precision/recall curve of HSNE and tSNE are similar. However, HSNE is designed to filter the data during the hierarchical analysis. Figure 6.9b shows the analysis performed by selecting landmarks that belong to the digit '7' (green points in Figure 9.8) and computing the the precision/recall curves using the points selected to be in the lowest scale. HSNE outperforms tSNE in the lowest scale: by reducing the number of data points to embed, HSNE is less influenced by local minima during their placement, leading to a better NNP. This result also confirms that in the higher scales of the hierarchy, manifolds are consistently represented, avoiding the creation of gaps in the lowest level embedding during the analysis. We obtained similar results for different analysis performed on the three datasets.

6.7 Conclusions

We presented Hierachical Stochastic Neighbor Embedding (HSNE). HSNE introduces the well-known mantra *Overview-First*, *Details-on-Demand* in non-linear dimensionality reduction techniques. Our technique preserves non-convex structures, similarly or better than the state-of-the-art methods, but can be employed in interactive software for the Exploratory Analysis of high-dimensional data. Even though complete embeddings (similar to *hybrid* techniques) are possible, a key strength

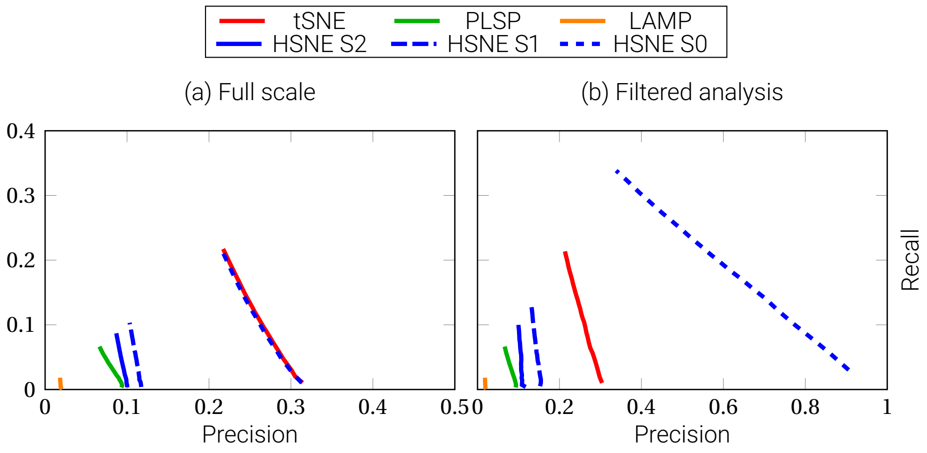


Figure 6.9: **Nearest Neighborhood Preservation (NNP)** on the MNIST dataset. HSNE outperforms hybrid techniques and it is comparable to tSNE on a full scale analysis. When the user filters the data during the drill-in, HSNE outperforms tSNE.

is the interactive hierarchical analysis to reveal dominant structures at different scales, which is useful in various applications, as evidenced by our use cases.

The various results indicate that HSNE is a beneficial replacement for non-linear and hybrid algorithms in Visual Analytics solutions. The use of the area of influence, is an important visualization element and delivers additional information, although new strategies would have to be developed to effectively exploit this information. Nonetheless, this aspect is important when considering systems to assess the quality of embeddings [109]. These mainly focus on visualizing and inspecting missing and false neighborhood relationships between data points. The investigation of neighborhood encoding remain an open problem, in particular for HSNE where these relationships are stochastic by nature. The multi-scale nature of many real-world phenomena leads us to the conclusion that HSNE may give new insights into a number of problem domains. The next chapters are focused on the application and successes of HSNE in several application areas. We start with the application of HSNE for the exploratory analysis of biomedical data. Then, we present the application and extensions of HSNE for analyzing extremely large bipartite graphs and, finally, we present a system the supports the design of deep neural-network architectures.

7 Applications in Life Sciences

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

In this chapter, we present several applications of the A-tSNE [138] and HSNE [136] algorithms with a focus on life sciences. We present Cytosplore, a tool developed for the exploratory analysis of single-cell data. Cytosplore is the enabler of several findings, ranging from the discovery of new immune-system cell types to cell differentiation pathways. Moreover, we introduce a tool for the analysis of 3D hyperspectral data which is based on the HSNE algorithm. This work has been done in a close collaboration with Thomas Höllt and researchers at the Leiden University Medical Center.

7.1 Single Cell Data Analysis

In recent years, novel data acquisition technologies, e.g., mass cytometry [126] (CyTOF), allowed to determine the properties of single-cells with unprecedented detail. This amount of detail allows for much finer cell differentiation, but also comes at the cost of more complex analysis. As a matter of fact, datasets comprise millions of cells, each represented as a high-dimensional data point, where each dimension represents the quantity, or expression, of a given protein per cell. The analysis aims at identifying groups of similar cells with respect to their protein expression, i.e., with a similar high-dimensional profile. However, most of the time, the user has limited prior knowledge on the cell types to be found in the data and the use of an exploratory analysis tool for high-dimensional data would be beneficial and, as presented in this chapter, will enable the discovery of previously unknown cell-types.

In this section, we present how the application of the A-tSNE [138] and HSNE [136] algorithms empowered the analysis of single-cell data and led to novel findings [90, 102,179]. We first provide the motivation of our work, and the biological background for single-cell data analysis. Then, we introduce the Cytosplore [68] application and its CyteGuide extension [69]. Finally, we present two immunology findings enabled by Cytosplore in combination with A-tSNE and HSNE.

7.1.1 Motivation

The immune system primarily protects our body against bacterial, viral and parasitic infections. However, it may also respond to harmless antigens, leading to autoimmune diseases, e.g., type-1 diabetes or rheumatoid arthritis. Detailed knowledge of the immune system's functioning is required to understand the cause of immune-mediated disease, which is an important step towards preventive or therapeutic measures. The cellular immune compartment consists of a variety of cellular subsets, each with a distinct function and associated phenotype. The phenotype describes "the observable physical or biochemical characteristics of an organism, as determined by both genetic makeup and environmental influences" [113]. In the last decades a large number of phenotypically and functionally distinct subsets have been defined and, for some, a major role in disease processes has been found. For immune cells, the functionality mostly relates to a set of proteins expressed on the cells surface.

Recently introduced mass cytometry [126], at the moment allows the observation of around 50 of these proteins at the same time, four times as many as the clinical standard. However, this number is still orders of magnitude smaller than the estimated 10,000 immune-system-wide available proteins, providing phenotypic information. Hence, specific panels of markers, corresponding to proteins of interest, need to be designed for each study. The composition of these panels is often unique to a study and it is not known beforehand, which combinations of proteins can be expected. Therefore, the identification of different phenotypes largely needs to be carried out in a data-driven fashion by studying data heterogeneity rather than applying prior knowledge.

7.1.2 Biological Background

To analyze heterogeneity of immune cell subsets, multiparameter analysis of immune cells at single-cell level is required. Flow cytometry has been the method of choice for this purpose, however, suffers from a limitation; it is restricted by the number of cellular markers that can be simultaneously analyzed, usually 10 to 12. Therefore studies employing flow cytometry are usually focused on very specific, known cell types. This limitation has been overcome by the introduction of mass cytometry. Mass cytometry is a novel, mass spectrometry-based, technique for characterizing protein expression on cells (cytometry) at single-cell resolution. In short, antibodies, selected to bind to specific proteins of interest on the cell membrane, are conjugated with heavy metal reporters. After staining, the cells are vaporized, atomized and ionized one by one and the remaining metals in the ion cloud can be measured in a mass spectrometer to quantify the selected proteins on a per-cell basis. Mass cytometry currently allows the simultaneous analysis of around 50 markers, a number which is expected to rise to 100 in the near future. This allows much broader studies, for example to compare different diseases. Furthermore it allows the inclusion of markers that usually would not be expected in a certain group, possibly allowing the discovery of unknown cell types.

7.1.3 Cytosplore

Cytosplore is an interactive visual analysis system for understanding how the immune system works. The goal of the analysis framework is to provide a clear picture of the immune systems cellular composition and the cells' corresponding properties and functionality. Cytosplore is targeted at the analysis of mass cytometry (CyTOF) data. It provides multiple linked views, showing different levels of detail and enabling the rapid definition of known and unknown cell types. Thanks to A-tSNE and HSNE, the tool handles millions of cells, each represented as a high-dimensional data point, facilitates hypothesis generation and confirmation, and provides a significant speed up of the analytical workow.

Figure 7.1 presents an example of the application, where a HSNE analysis is preformed. On the left side of the user interface, see Figure 7.1a, a control panel is available. On this panel, the user selects the dimensions of interests, i.e., the proteins that will be used to differentiate the cell types. Once that the dimensions are chosen, the user starts analyzing the data by creating A-tSNE or HSNE embeddings. Cytosplore supports progressive computations [41] and the intermediate results are immediately visualized in linked views. In the example, a HSNE analysis is performed. Figure 7.1b shows the overview embedding, where the landmarks are color coded according to the value of one of the chosen dimensions, showing that the corresponding protein is partially responsible for the separations of some of the clusters of landmarks.

The user can manually select a cluster in the overview, either manually or with

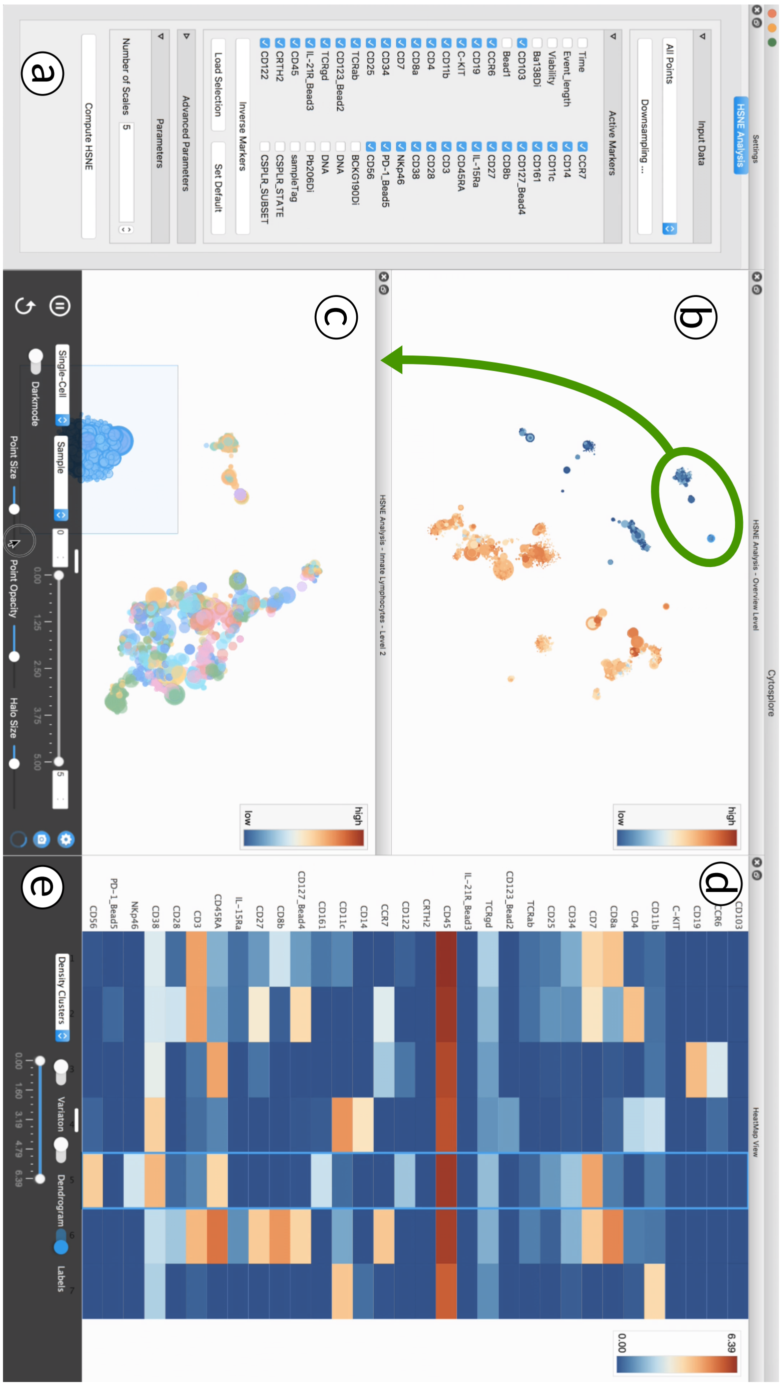


Figure 7.1: **Cytosplore** [68] is an interactive visual analysis system for understanding how the immune system works. Each cell is represented as a high-dimensional point, where the dimensions represents the frequency of protein interactions. Cytosplore offers several linked views to enable the definition of cells differentiation. The user can interactively define which dimensions ought to be considered in the analysis (a). Embeddings of the dataset are created by A-tSNE [138] or HSNE [136]. In this example a HSNE analysis is presented, with an overview embedding (b) and a detailed view (c). In the detailed embedding the user selected a cluster. The corresponding high-dimensional profile is visualized in a heatmap visualization, enabling the verification of which dimensions are responsible of the presented cluster.

the help of a non-parametric clustering algorithm [46]. The selection is then used to and create a detailed view by drilling in the HSNE hierarchy, en example of which is presented in Figure 7.1b. In the detailed view, heterogeneity within the clusters is revealed. The user can select a cluster of points in the embedding and analyze the corresponding profile in a linked heatmap view, see Figure 7.1c.

Cytosplore has been developed in collaboration with immunologists at the Leiden University Medical Center, with the specific goal of empowering their analytical capabilities. Before the introduction of Cytosplore, the analysis of a single dataset could take several months, often relying on printouts of tSNE [177] embeddings on subsets of the data. With Cytosplore, the same analyses are performed in only few hours, while avoiding any subsampling of the data thanks to the adoption of our HSNE algorithm. In particular, Cytosplore and HSNE allowed our research partners to discover new immune-system cell-types as we will present in Section 7.1.4.

CyteGuide

To facilitate the exploration of the HSNE hierarchy, we introduced the CyteGuide extension to cytosplore. In the basic HSNE implementation in Cytosplore, the user would start with a high level embedding, showing only the most representative landmarks, as presented in Chapter 6. Each of these landmarks represents a group of cells of the next, more detailed level. This approach tackles scalability issues of techniques like tSNE in terms of data size and computational performance, however, at the cost of increased user interaction. By looking just at the unordered embeddings, the user can easily lose the overview of the state of the exploration, see the example in Figure 7.2a.

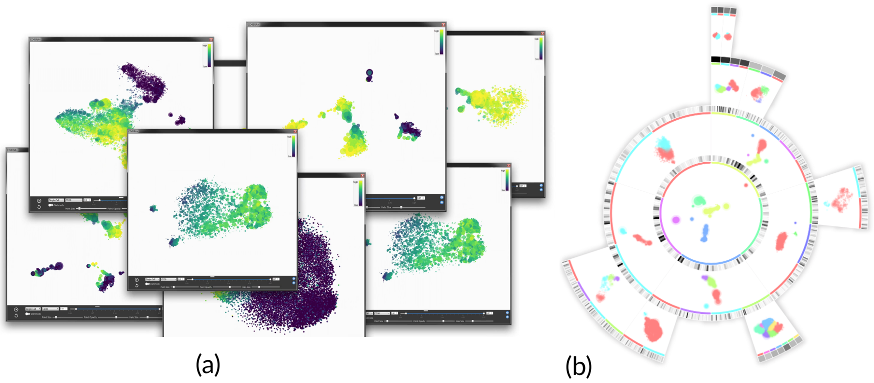
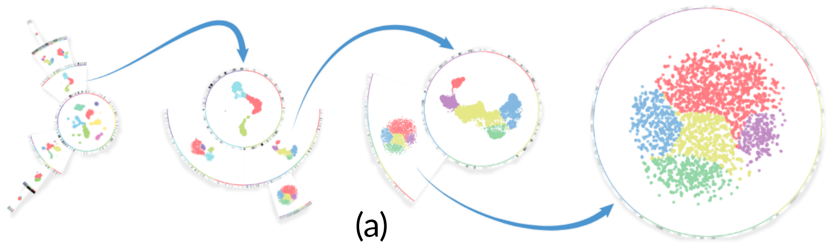


Figure 7.2: **Cyteguide** facilitates the exploration of the HSNE hierarchy. Embeddings are organized in a sunburst diagram, with the overview embedding at the center. Each embedding is automatically clustered using the Mean Shift algorithm [46]. Different clustes are characterized by a different color and, upon user request, a more detailed embedding of a corresponding cluster is created and arranged in the sunburst diagram. For each embedding, a heatmap is used to show the variability of each feature in the cluster. Clusters with higher variance are more interesting to be explored by the user.



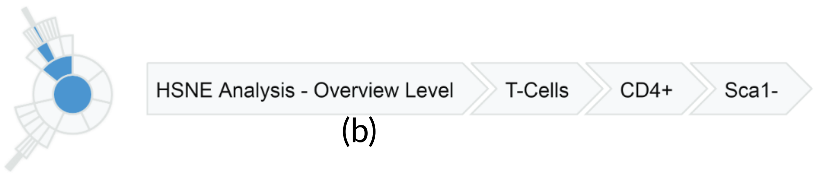


Figure 7.3: **Adaptive layout of CyteGuide**. Detailed embeddings can be placed at the center to reduce clutter (a). A thumbnail and breadcrumb visualization is used to keep track of the interaction (b).

Furthermore, HSNE does not provide any guidance for exploration, or overview of the state of the exploration beyond a set of unordered embeddings. Computing the embeddings is costly and can be unnecessary, if a higher level embedding already shows all features of interest. Therefore, guiding the user to regions of interest can save computation time. While our partners were able to create meaningful insight with HSNE, especially due to its scalability, the exploration of the hierarchy became a challenging task. The goal of Cyteguide is to ease the process of exploring the hierarchy by providing an overview of the current state of the exploration, but also by pointing the user to unexplored parts that could provide deeper insight in lower levels of the hierarchy.

Figure 7.2 presents an example of Cyteguide for the analysis of a single-cells dataset. The embedding representing the overview, i.e., containing landmarks from the highest HSNE scale, is presented at the center of the Cyteguide visualization in Figure 7.2b. The landmarks are colored according to the result of a Mean Shift [46] clustering that is directly controlled by the user. In the example of Figure 7.2b, 5 clusters are identified. The circle surrounding the embedding is then divided in 5 equal-sized sectors, one for each cluster. These sectors are identified by using a line with the same color of the corresponding cluster. On top of that, a heatmap visualization is used to guide the user towards clusters that are interesting to be investigated further. In the heatmap, each feature is colored according to the standard deviation of the feature in the cluster. Intuitively, if a cluster presents high variance in one of the features, it may reveal higher heterogeneity at a lower level in the HSNE hierarchy. The user decides to create more detailed embeddings for all the five clusters. The embeddings are created in a progressive fashion, while the CyteGuide visualization is extended by placing the corresponding embeddings in the sunburst diagram. The process can then be iterated, and the CyteGuide visualization is adapted according to the user interaction.

CyteGuide adopts several design choices to deal with the ever decreasing visual space available for more detailed embeddings. Among the others, the heatmap visualization shows only the feature with the highest standard deviation in a cluster if not enough space is available. Moreover, detailed embeddings can be placed at the center of the visualization in order to reduce clutter, see Figure 7.3a. A thumbnail and breadcrumb visualization is used to keep track of the interaction, see Figure 7.3b.

7.1.4 Discovery of Unknown Cell-Types

In a previous study performed by our partners [180], a mass cytometry data set on 5.2 million cells derived from intestinal biopsies and paired blood samples was analyzed using a pipeline consisting of several algorithms. The goal of the study was the identification of rare cell-types, possibly related to autoimmune diseases. These cell-types are identified by different type of interaction with proteins, resulting in a different high-dimensional profile as presented in Section 7.1.2.

In their study, Van Unen et al. [180] proposed a computational pipeline that havily relies on downsampling. The data was subsampled to 2.2 million cells, hence discarding almost 57% of the entire dataset. This choice is motivated by the limited scalability of the algorithms comprising the pipeline, hence the subsapling of the data is paramount for the feasibility of the computation.

The downsampled dataset was then processed with SPADE [142], a technique that creates a tree of cell clusters. Subsets of the SPADE tree, i.e. clusters of cells, were then used to create tSNE embeddings that are then processed by the ACCENSE algorithm [157]. ACCENSE performs a density-based subsampling and clustering in the tSNE space, allowing for the extraction of high-dimensional profiles associated to the clusters visible in the embedding. After this final subsampling, only 1.1 million cells are analyzed by the experts, effectively discarding approximately 80% of the complete dataset. This pipeline is fully implemented in Cytosplore, and we refer the interested reader to the work of Van Unen et al. [180] for more detail.

Despite the insight achieved by adopting the computational pipeline, it has several limitations. First, it heavily relies on subsampling of the data to make the computation feasible. By discarding almost 80% of the data, rare cell subsets, e.g., containing less than 0.5% of the available cells, are not found during the analysis. Finally, despite the subsampling, the presented pipeline requires a lot of computation time for each step of the pipeline. Several weeks of analysis are required to process single-cell datasets containing millions of cells [68, 180].

We proposed to substitute the complete pipeline with our Hierarchical Stochastic Neighbor Embedding algorithm, as presented in Chapter 6. This approach has several advantages. First, it does not require a combination of several algorithms, i.e. SPADE-tSNE-ACCENSE, but provide a common framework for the analysis. Second, it does not require to subsample the data. HSNE easily scales to several mil-

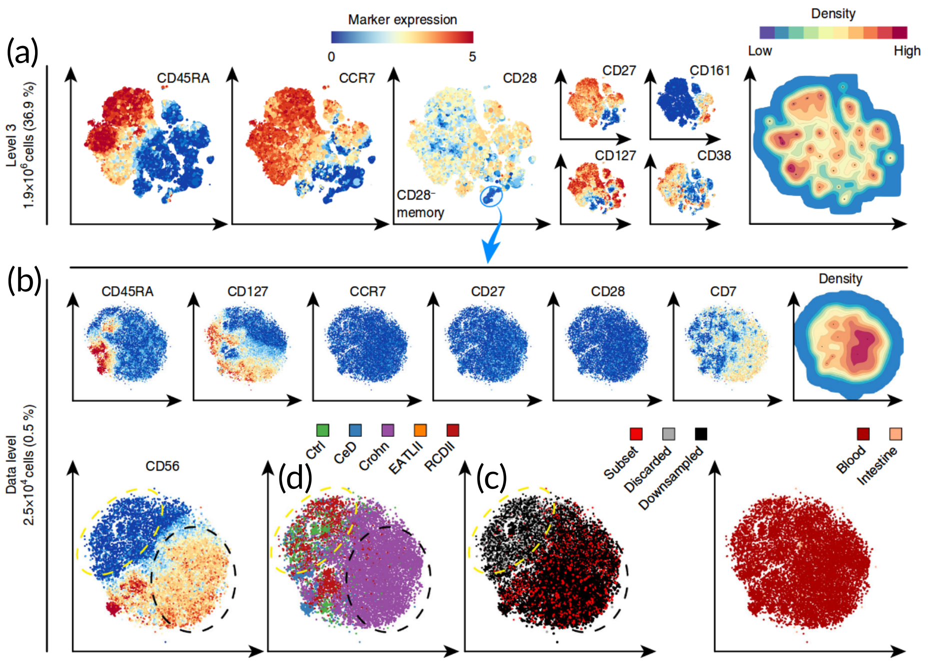


Figure 7.4: **Rare cell-type identification** out of 5.2 million cells. At the top of the figure, a detailed HSNE embedding is shown (a). The embedding contains 1.9 million cells, corresponding to the 36.9% of the entire dataset. The embeddings are color-coded with the expressions of different markers, e.g. CD45RA, CCR7 or CD28. A small cluster is then selected and analyzed in a more detailed embdedding (b). Only 0.5% of the entire dataset are present in this embedding. A cluster of cells, here at the top left of the embedding, was not known before due to the subsampling required by other computational tools (c). The cells show an association with the RCDII disease (d).

lion data points, hence it allows for the analysis of all the 5.2 million cells without requiring to subsample the data. Finally, despite no data being discarded during the analysis, HSNE is much faster than the previously proposed pipeline, requiring only few hours to process the data. Figure 7.4 shows an example of the analysis performed. At the top of the figure, a detailed HSNE embedding is shown. The embedding contains 1.9 million cells, which correspond to 36.9% of number of cells available in the dataset. The embeddings are color-coded with the expressions of different markers, e.g. CD45RA, CCR7 or CD28. A small cluster, having low value for the marker CD28, is selected and analyzed in a more detailed embdedding, as presented in Figure 7.4b. The new detailed embedding, contains only 0.5% of the entire dataset. A cluster of cells, here at the top left of the embedding, was not known before due to the subsampling required by other computational tools. As a matter of fact, all the points in black and gray in Figure 7.4c, were not analyzed in previous study due to the subsampling. Figure 7.4d shows that the subsampled cells show an association with the RCDII disease.

7.1.5 Discovery of Cell-Differentiation Pathways

A different discovery was made thanks to the adoption of the Progressive Visual Analytics paradigm for single-cell analysis enabled by A-tSNE. As we presented in Chapter 4, A-tSNE fully support progressive computations, hence the intermediate results of the optimization are presented to the user that decides on the convergence of the algorithm. Our research partners at the Leiden University Medical Center, were early adopters of the A-tSNE as implemented in Cytosplore. While observing the evolution of a single-cell data embedding, as presented in Figure 7.5a, they observed that in the early stage of the optimization, cells were creating a linear structure of points [102]. This observation was particularly interesting, since the dataset they were analyzing potentially contains *precursor* cells, colored in red

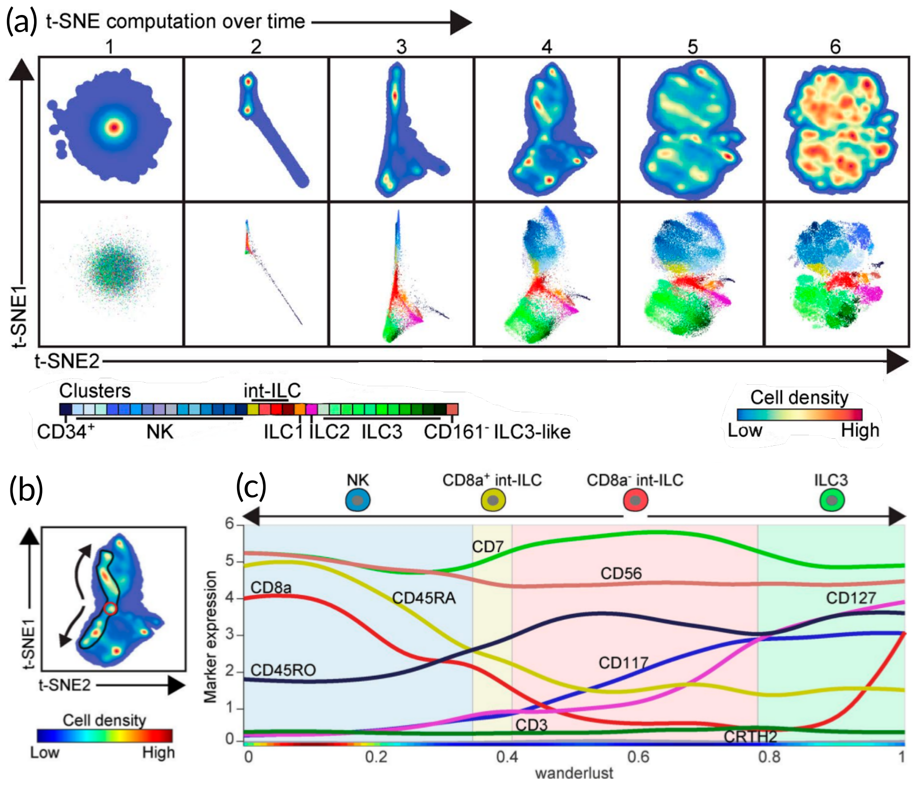


Figure 7.5: **Cell pathways identitification**. A-tSNE is used to visualize the intermediate results of the tSNE optimization (a). Our research partners observed that the embedding presents a dominating linear structure during the early stages of its evolution (a3). They hypothesized that this structure represents the differentiation pathways from precursor cells, colored in red, to two different cell types (b). The high-dimensional profile of the cells on the pathway is then extracted (c) and used to verify the hypothesis with experiments in the laboratory [102].

in Figure 7.5a, and these cells were visible at the center of the structure. Precursor cells play a crucial role in several biological phenomena since they behave as "building blocks" for other immune-system cells, as they can differentiate into multiple types of more specialized cell types. The transformation of a precursor cell to a specialized cell type takes the name of differentiation pathway, and remain largely unclear [102].

Our partners hypothesized that the linear structure was showing two differentiation pathways from the precursor cells, to two different types of specialized cell-types, see Figure 7.5b. Cells along the structure represents the different stages of the differentiation, from which a high-dimensional evolution profile is extracted and presented in Figure 7.5c. This profile, represents the different protein interactions during cell differentiation. This information was used by our partners to validate their hypothesis. In laboratory experiments, they managed to induce the differentiation of the precursor cells by exposing them with the proper combination of proteins [102], hence validating the original insight.

The reader may now wonder, how reliable is an insight obtained by looking at the optimization of the embedding. This is a legitimate consideration, as a matter of fact, the tSNE cost function is very far from being optimized during the early stage of the embedding evolution. However, if we consider carefully how the optimization is performed, we may observe that there is a reason why the linear structure appears. As introduced in Section 4.3.3, an optimization strategy called *early exaggeration* is used. To avoid local minima, during the early stage of the embedding computation, attractive forces between points are exaggerated, hence the name of early exaggeration. This result in better representation of major structures in the high-dimensional data during the first iterations, since the repulsive forces are less effective in spreading the points in the embedding. We believe that this is an interesting example in which, progressive computations may provide additional insight not only on the data that is studied, but also on the adopted algorithm.

7.1.6 Cytosplore Viewer

In the previous sections, we presented applications of the techniques developed in this thesis for single-cell analysis. The applications are aimed at the analysis on mass cytometry data, with a specific focus on the understanding of how the immune system works. Mass cytometry is just one of the acquisition techniques that, in recent years, enabled the acquisition of large single cell datasets. Another notable example, is given by the Allen Institute for Brain Science[1]. In their recent work [67], the Allen Institute acquired data from 15 thousand cells, i.e., neurons, located in a human brain region called the "Middle Temporal Gyrus". For each cell, the amount of activity of more than 20 thousand genes was measured with a technique called RNA-sequencing. The data was then used to define a hierarchy of different cortical cell types that are found in the human brain. Among the findings, the scientist discovered dramatic differences between homologous cells in the human

[1]http://www.brain-map.org/

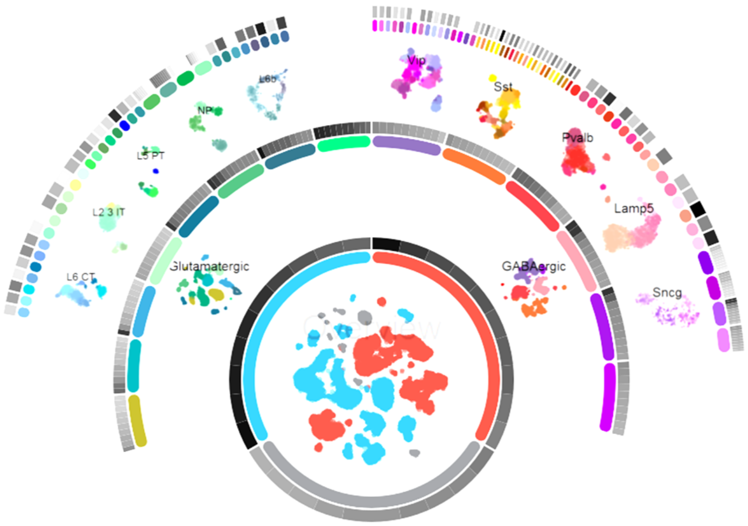


Figure 7.6: **Cytosplore Viewer** is the accompanying visualization tool to the "Middle Temporal Gyrus" dataset. The tools is an evolution of Cytosplore that is specifically designed to facilitate the exploration of the hierarchy of neuron types found by the Allen Institute for Brain Science. In this figure, a Cyteguide layout is used to organize tSNE mapping of the cells. Note that, differently from the original Cyteguide implementation, in Cytosplore Viewer the hierarchy is pre-defined by the scientists at the Allen Institute.

and mouse, hence revealing the importance of studying directly the human brain directly [67].

The Middle Temporal Gyrus dataset is openly available to anyone interested in studying it[2]. In order to facilitate the exploration of the data, together with the Allen Brain Institute we developed a visualization tool called "Cytosplore Viewer"[3]. Cytosplore Viewer is based on the techniques and applications presented in this dissertation. It is an evolution of Cytosplore [68] and is developed as a collaboration between the Allen Institute for Brain Science, the Leiden University Medical Center and the Delft University of Technology. Figure 7.6 shows how the hierarchy of cortical cell types is visualized in Cytosplore Viewer by adopting the Cyteguide [69] visualization. Moreover, the tool allows for the recomputation of the tSNE maps that are provided by the Allen Institute by selecting a reduced set of genes, i.e., dimensions. The embeddings are computed in a matter of seconds by using the A-tSNE [138] algorithm presented in Chapter 5.

[2]http://celltypes.brain-map.org/rnaseq
[3]https://viewer.cytosplore.org/

7.2 Hyperspectral Volume Data

In Chapter 6, we introduced the Hierarchical Stochastic Neighbor Embedding algorithm (HSNE). As a test case for HSNE, in Section 6.5.1 we presented the analysis of hyperspectral images of the Sun surface and the area surrounding the city of Los Angeles. HSNE is well suited for the analysis of this type of data. While each pixel in the image represents a high-dimensional vector encoding the intensity of different wavelengths, making the application of non-linear dimensionality-reduction compelling, the image high resolution poses challenges in the scalability of the chosen approach. The same analytical pipeline can be adopted for biomedical-imaging data. Examples of possible acquisition techniques that generates high-dimensional images are mass-spectrometry imaging [111], functional MRI [17] and perfusion imaging [93].

In collaboration with the Leiden University Medical Center, we demonstrated the effectiveness of HSNE applied to hyperspectral-volume data [3]. Thanks to the scalability of HSNE, we were able to analyze stacks of hyperspectral images representing a volume. Figure 7.7 shows an example of a mass-spectrometry volume containing a tumor. The overview embedding is used to quickly identify the region of interest, i.e., the tissue foreground. Detailed embeddings are then generated and reveal the tumor and the connective tissue. In our study, we analyzed several datasets, ranging from 10 thousand voxels, i.e., cells in the volume, to 1.3 million. We demonstrated that HSNE reveals, relatively fast and in an interactive data-driven manner, multiscale molecular structures that might hold biological interest. These structures are otherwise very computationally difficult to identify using alternative pipelines.

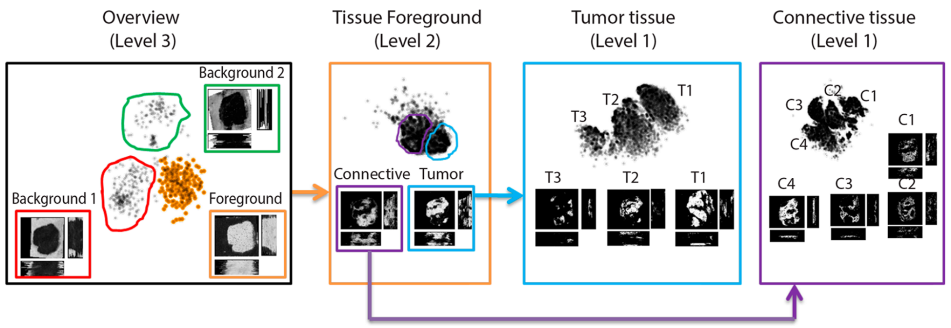


Figure 7.7: **HSNE for hyperspectral volume data**. A mass-spectroscopy volume consisting of 150 thousand voxel, i.e., cells, each described by 8 thousand dimensions is analyzed. By exploring the hierarchy, tumor and connective tissue are easily identified [3].

8 Multiscale Visualization and Exploration of Large Bipartite Graphs

The web is more a social creation than a technical one.

Tim Berners-Lee

In this chapter, we present a novel approach for the visualization of large bipartite graph. Our analytics tool, which takes the name of WAOW-Vis, relies on the HSNE algorithm to create two hierarchical representation of the sets of nodes in the bipartite graph. We enable a multiscale exploration of communities created by connected nodes and we demonstrate the application potential in the exploration of social-network data. In particular, we show how WAOW-Vis highlights communities of Twitter users that follows only polarized sources of information.

N. Pezzotti, J.-D. Fekete, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Multiscale visualization and exploration of large bipartite graphs. In Computer Graphics Forum, volume 37, page 12, 2018 [135].

8.1 Introduction

The previous chapters focused mainly on the analysis of high-dimensional data using non-linear dimensionality-reduction techniques. In this chapter we explore and present extensions to our Hierarchical Stochastic Neighbor Embedding algorithm to enable the analysis of very large bipartite graph. Our results demonstrate that dimensionality-reduction can be used to solve analytical problems beyond the analysis of high-dimensional data.

The bipartite graph is an important abstraction in computer science; vertices in the graph are divided into two disjoint and independent collections of items $\mathcal{U}$ and $\mathcal{V}$. The edges in the graph represent the relationships between the elements in the two collections and, therefore, only connect elements in $\mathcal{U}$ to elements in $\mathcal{V}$. Several problems can be modeled as a bipartite graph analysis, for example, the two collections may be software developers and source files, gene mutations and patients in a cohort study or social media users and the news outlets they follow on social media. Previous work [65,185] has identified the following analytical tasks for the exploratory analysis of bipartite graphs.

- **(T1)** Identifying clusters of similar elements in $\mathcal{U}$ with regard to their connections to elements in $\mathcal{V}$ and vice versa.
- **(T2)** Understanding the interrelationships between the clusters in the two collections $\mathcal{U}$ and $\mathcal{V}$.

A widely used approach for performing these tasks is to visualize the bipartite graph using a node-link visualization. A commonly used approach separates the two collections of items on screen. Vertices are displayed as points arranged along two parallel axes, i.e., corresponding to $\mathcal{U}$ and $\mathcal{V}$, and the edges are visualized as lines connecting the vertices [65,185], see Figure 8.2a. By using matrix reordering algorithms on the adjacency matrix of the graph [13,185], vertices that share a similar connection pattern with respect to the other collection can be placed close together along the axis, allowing for identification of vertices with similar connections **(T1)** and their mutual relationships **(T2)**. However, node-link visualizations do not scale for the analysis of bipartite graphs containing more than a few hundreds vertices due to the resulting visual clutter [52]. To overcome this limitation, algorithmic graph preprocessing techniques are often used to reduce the complexity of the graph to be drawn [185]. For bipartite graphs, biclustering algorithms, also known as co-clustering techniques, become the standard for the identification of sub-clusters in $\mathcal{U}$ and $\mathcal{V}$ that share a similar connection pattern to the other collection [61,105,125,141]. Clusters are then visualized as aggregated vertices in the node-link diagram. Nonetheless, such an approach assumes that there is no variability within a cluster, which is problematic when the data is large and contains a hierarchy of sub-clusters. For example, communities of social media users may share similar connections to a set of news feeds, but they may also contain sub communities where the connections are slightly different.

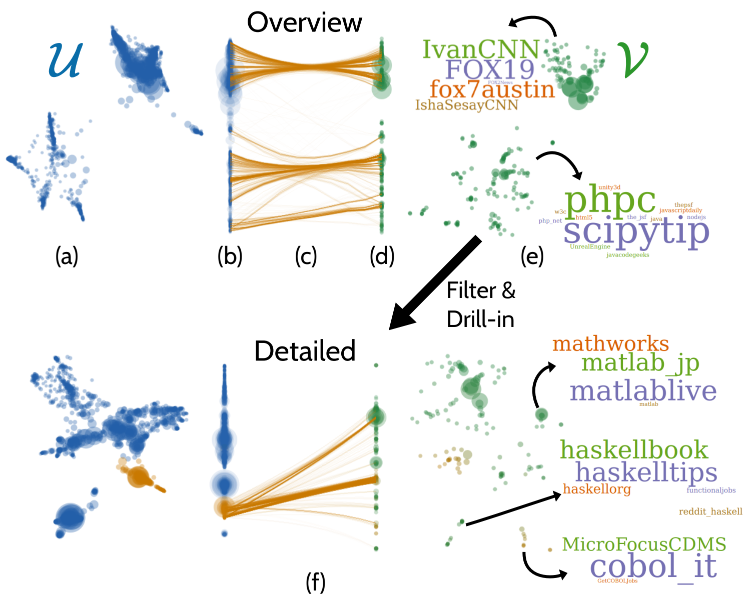


Figure 8.1: **Example of WAOW-Vis** for a bipartite graph containing a collection $\mathcal{U}$ of 9.4M Twitter users is linked to a collection of 228 $\mathcal{V}$ of Twitter feeds associated to programming languages and US' news outlets. The bipartite graph is preprocessed by the HSNE algorithm that extracts a hierarchy of landmarks, i.e., sets of vertices. In the overview, landmarks in the highest scale for the two collections are placed in (b,d) 1- and (a,e) 2-dimensional embeddings that reveal major clusters of similarly connected vertices (c). The hierarchy is explored with an *overview-first* and *details-on-demand* approach that reveals hierarchies of sub-clusters (f).

To address this problem we present Who's-Active-On-What-Visualization (WAOW-Vis), a technique designed to reveal hierarchies of clusters in bipartite graphs. To this end, we adapt our multiscale dimensionality-reduction algorithm, the "Hierarchical Stochastic Neighbor Embedding" (HSNE) [136] that is presented in Chapter 6, for extracting and visualizing clusters of similarly connected vertices. Figure 8.1b-d shows the resulting layout of WAOW-Vis, where two 1-dimensional embeddings are used to visualize the vertices in a layout that mimic node-link visualizations for bipartite graphs **(T1)**. Moreover, we show that, by adding 2-dimensional embeddings of the same vertices, we obtain more detailed insight on the interrelationships between the clusters **(T2)**. As an illustration of our results and goal, in Figure 8.1a we can identify clusters and see the internal structure much easier than in Figure 8.1b. However, Figure 8.1a-e, shows only one scale of the hierarchy computed by HSNE. As we presented in Chapter 6, HSNE does not embed the dataset in its entirety but it selects representative vertices at different scales. i.e., landmarks. The visualization of the landmarks in the highest scale shows an *overview* of the main clusters of connected vertices in the graph. The user can then select these clusters and ask for a *detailed visualization* that reveals more sub-clusters for the landmarks in the lower scale, as shown in Figure 8.1f.

Several challenges need to be overcome to achieve our goals using HSNE. Dimensionality reduction algorithms rely on a dense representation of the data for the computation of the similarities between points. This fact constitutes a problem if the dimensionality-reduction is applied to a large adjacency matrix that is saved in a dense format, as it will not fit in memory. Consider for example a dataset that we mined from Twitter, which we will present in Section 8.7. It comprises of 19M users linked to 329 user interests. The dense representation of the biadjacency matrix would occupy 23 GB in memory if organized as required by dimensionality-reduction algorithms such as HSNE. Then we adopt the Jaccard similarities, a compelling metric that measure the amount of shared links, to compute similarities between vertices. Figure 8.2 shows an example of how the similarities for the collection $\mathscr{U}$ are computed. The vertices in $\mathscr{U}$ are seen as sets of elements in $\mathscr{V}$. The Jaccard similarities are computed as the number of shared elements between sets divided by the size of their union. A similar computation is done for the vertices in $\mathscr{V}$. Scalable computations and memory usage are enabled by the use of compressed bitmaps [100] to represent the sets and by a novel tree-based data structure, the Sets Intersection Tree (SIT), to efficiently compute the Jaccard similarities. Moreover, we show that by combining the k-nearest neighborhood graph, computed as input for HSNE, the resulting embeddings reveal clusters of vertices at different scales **(T1)** and their interrelationships **(T2)**. Finally, since the user can be confused by the fact that same vertices may appear in different locations in the 1- and 2-dimensional embeddings, we present a technique that enforces similar positions on the vertical axis for the same vertices to facilitate the creation of a mental map.

The contribution of this chapter is the WAOW-Vis framework, that allows for analyzing bipartite graphs on a social-network scale at different levels of detail. The development of WAOW-Vis is made possible by a set of additional contributions described in the chapter:

- The usage of compressed bitmaps as high-dimensional data representation for bipartite-graphs.
- A data structure for the efficient computations of similarities between compressed bitmaps in the Jaccard space; the Sets Intersection Tree.
- A modification of the tSNE [177] algorithm for the creation of consistent and interrelated embeddings.

The rest of the chapter is organized as follows. In the next section we present the related work. An overview of WAOW-Vis is given in Section 8.3. Section 8.4 highlights the generation of the hierarchical representation of the data. In Section 8.5 we present the interactive exploration using WAOW-Vis. Finally, in Section 8.7 we validate WAOW-Vis via several test cases based on three datasets that we obtained by mining Twitter. We demonstrate how WAOW-Vis is particularly effective in highlighting communities of users that shares similar interests.

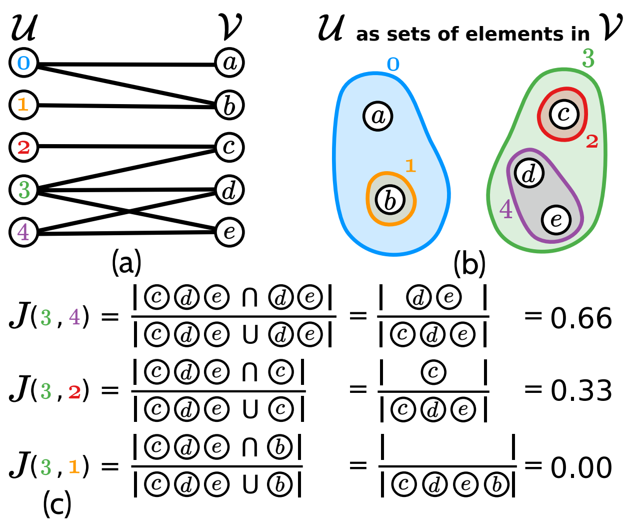


Figure 8.2: **Computation of the similarities** between the vertices in $\mathcal{U}$. Vertices in $\mathcal{U}$ are seen as sets of elements in $\mathcal{V}$ (b). The Jaccard similarities are computed as the cardinality of the intersection divided by the cardinality of the union (c).

8.2 Related Work

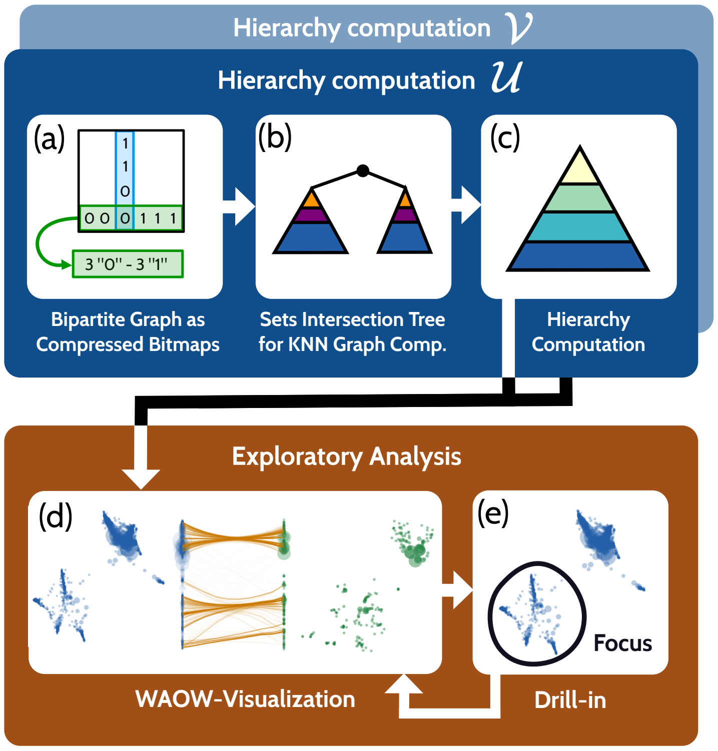
A **Bipartite graph** $(\mathcal{U}, \mathcal{V}, E)$ is a particular type of multifaceted graph [55] where vertices form two disjoint sets $\mathcal{U}$ and $\mathcal{V}$, and edges $E = (u, v) \in \mathcal{U} \times \mathcal{V}$ connect elements from each sets. Bipartite graphs can be displayed using traditional visual encodings for graphs such as **node-link diagrams** and **adjacency matrices** [185]. In a node-link diagram vertices are placed in a 2-dimensional space using a layout algorithm [167]. Adjacency-matrix visualizations rely on matrix reordering techniques for highlighting connectivity patterns, e.g., cliques of strongly connected nodes [13, 185]. Empirical studies show that node-link visualizations are usually more intuitive for understanding the graph but, for dense graphs, adjacency matrix visualizations outperform node-link visualizations due to the reduced visual clutter [52]. Node-link diagrams relying on dimensionality reduction techniques have also been presented [108], but they do not scale beyond few thousand vertices. To combine the advantages of both worlds, **hybrid techniques** had been developed. The MatrixExplorer [63] and the NodeTrix [64] visualization systems are just two examples that combine node-links and adjacency matrix visualization in order to provide greater insights. WAOW-Vis is a hybrid technique that scales to bipartite-graphs with several millions vertices and edges. Similarly to matrix reordering techniques, it allows for identifying clusters of similarly connected vertices as clusters of points in the dimensionality reduction layouts **(T1)**, i.e., embeddings (see Figure 8.1a). Then, thanks to a visual design similar to node-link diagrams, WAOW-Vis provides a better interpretability of the results by showing 1-dimensional embeddings and their interrelationships **(T2)** (see Figure 8.1b-d).

In order to enforce a distinction between the collections $\mathcal{U}$ and $\mathcal{V}$, different approaches allocate **separated visual spaces** to the two collections, which can be parallel axes, interleaved axes [16], or concentric circles [36]. Edges connecting the vertices in the two collections are then visualized as links. A subset of the links may be drawn based on the user's current focus or, to give a complete picture of the data, all the links can be drawn. Lines may be bundled in order to reduce the resulting visual clutter [195]. This approach is used in several visual analytics systems such as VisLink [29], PivotPath [35], PNLBs [51] and Jigsaw [164].

In this chapter we focus on the analysis of large graphs, i.e., containing tens of millions of vertices and edges. Gaining insight from a direct visualization of the data with one of the previously described techniques is not possible due to the resulting visual clutter. Algorithmic graph preprocessing [185] is therefore of major importance in order to create meaningful visualizations. **Graph filtering** algorithms are used for reducing the number of the visualized vertices [20, 101]. Jia et al. [78], for example, remove vertices that are not considered important according to a notion of graph centrality. In **graph aggregation** techniques, the vertices are not simply removed, but multiple vertices are merged into a single one, hence reducing the size of the resulting graph. Vertices can be merged according to different criteria, e.g., by treating cliques of strongly connected vertices as a single node in the visualization [11, 37]. For bipartite graphs, aggregation is usually performed using biclustering algorithms [105] which create clusters in the two collections $\mathcal{U}$ and $\mathcal{V}$ based on their mutual relationships. Biclustering algorithms are often used in bioinformatic [61, 125, 141] and for the analysis of deep neural networks [115]. If the graph simplification is repeated multiple times, a hierarchical graph, also called **compound graph**, is created. The hierarchical graph is then analyzed with visualizations that allow for the exploration of the data at different scales [63, 64, 70, 187]. Existing techniques, however, are limited in the analysis of large graphs. The preprocessing represents a bottleneck that requires many hours, or even days, to complete, hence limiting the interactive analysis of the data. Our goal is to tackle bipartite graphs with tens of millions of vertices and edges at interactive speed on regular hardware.

8.3 WAOW-Vis overview

Figure 8.3 shows the overview of the generation of WAOW-Vis and its interactive exploration. WAOW-Vis uses separated visual spaces for visualizing the two collections, and creates a layout similar to the traditional and easy to interpret node-link visualization for bipartite graphs, as shown in Figure 8.1b-d. However, we propose to enrich the visualization with 2-dimensional embeddings, see Figure 8.1a and e. This solution reveals more clusters and richer information about their interrelationships, as vertices have more visual space to be layed in **(T2)**. We considered adding links between the 2D and 1D embeddings to make the identification of the same vertices easier. However, we quickly realized it would introduce excessive clutter



Hierarchy computation $\mathcal{V}$
Hierarchy computation $\mathcal{U}$
(a)
1
1
0
0 0 0 1 1 1
3 "0" - 3 "1"
(b)
(c)
Bipartite Graph as Compressed Bitmaps
Sets Intersection Tree for KNN Graph Comp.
Hierarchy Computation
Exploratory Analysis
(d)
(e)
Focus
WAOW-Visualization
Drill-in

Figure 8.3: **Overview** of WAOW-Vis. Two hierarchical representations of the collections $\mathscr{U}$ and $\mathscr{V}$ are computed. The elements in the collections are encoded via compressed bitmaps (a). The k-nearest neighborhood graph are computed for the non-duplicated bitmaps (b) and the hierarchy is computed by HSNE (c). In WAOW-Vis, we first present an overview of the data (d). The user may focus on specific regions of interest and generate more detailed visualizations (e).

and, instead, we developed a novel embedding computation technique keeps corresponding vertices roughly aligned (Section 8.5.2).

The main contribution of WAOW-Vis is it scalability to graphs that, to the best of our knowledge, cannot be handled by existing techniques. This result is achieved thanks to a novel algorithmic graph preprocessing that take advantage of the recent advancements in the field of large high-dimensional data analysis [136, 168]. Our approach can be separated in two steps, the **hierarchy computation** and the **exploratory analysis** of the so computed hierarchies of vertices.

In the **hierarchy computation** step, two identical computational pipelines are applied to the collections $\mathscr{U}$ and $\mathscr{V}$ separately (Figure 8.3a-c). First, the biadjacency matrices of the bipartite graph are transformed into two collections of compressed bitmaps. A bitmap associated with a vertex in $u \in \mathscr{U}$ contains the set of vertices in $\mathscr{V}$ that are connected to u and vice versa (Section 8.4.1). Then, the compressed bitmaps are organized in a novel data structure, the Set Intersection Tree (SIT), presented in Section 8.4.2. The SIT allows for the efficient computation of Jaccard similarities between vertices, leading to a scalable approach for the creation of the k-nearest neighborhoods graph. The k-nearest neighborhoods graph is then used as the input for the Hierarchical Stochastic Neighbor Embedding (HSNE) [136] algorithm that generates a hierarchical representation of the collections $\mathcal{U}$ and $\mathcal{V}$ (Section 8.4.3). Intuitively, the resulting hierarchies contain a number of scales. Each scale contains a number of **landmarks** that can be seen as a collection of elements in the two collection of vertices $\mathcal{U}$ and $\mathcal{V}$.

The **exploratory analysis** starts from an overview visualization, an example of which is presented in Figure 8.1a-e. The visualization contains embeddings of the landmarks at the highest scale of the HSNE hierarchy for each collection $\mathcal{U}$ and $\mathcal{V}$ that reveal similarly connected clusters of vertices (Sections 8.5.1 and 8.5.2). Furthermore, the user can request more detailed visualizations by filtering uninteresting clusters of vertices and by drilling into the hierarchies (Section 8.5.3), hence generating novel layouts containing data points from a more detailed HSNE scale (Figure 8.1f).

With this approach, the user can reveal more heterogeneity within one of the previously identified cluster of vertices **(T1)**. WAOW-Vis is designed to explore the structure of bipartite graphs and the relations between the structures of the two sets of vertices. From the perspective of the task taxonomy of graph visualization by Lee et al. [98], WAOW-Vissupports group-level and cluster-level tasks, but not path-level tasks. The node-level and link-level tasks can be supported with varying levels or precision depending on the visualized scale.

8.4 Hierarchy computation

In this section we present how the bipartite graph is transformed in the two hierarchical data representations that are used for the creation of WAOW-Vis. In each subsection we present a single module of computational pipeline that is introduced in Figure 8.3a-c.

8.4.1 Compressed bitmaps as high-dimensional data

Figure 8.4 shows an example of how the data is transformed in a collection of compressed bitmaps that are used for the efficient computation of the similarities, both in terms of memory and computations. The corresponding adjacency matrix for this data is presented in Figure 8.4b. The matrix can be seen as a composition of two disjoint and symmetric regions called biadjacency matrices, one of which is shown in Figure 8.4c. Biadjacency matrices encode the relationships between the set $\mathcal{U}$ and $\mathcal{V}$ and vice versa.

We propose to treat the biadjacency matrices as high-dimensional datasets and to measure similarities between the vertices using the Jaccard similarities as introduced in Figure 8.2. Given two indices i and j, the Jaccard similarity of the corre-

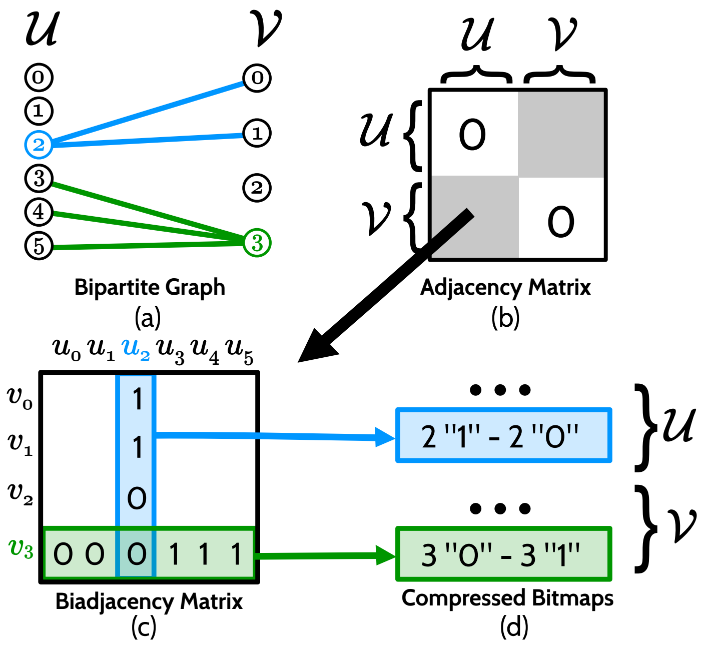


Figure 8.4: **Bipartite graphs as high-dimensional data**. A bipartite graph connects a collection of users $\mathscr{U}$ with a collection of Twitter feeds $\mathscr{V}$. To avoid clutter, here we show only the links connecting u_2 and v_3 (a). The bipartite graph is represented by an adjacency matrix (b). The adjacency matrix contains two symmetric biadjacency matrices (c). The elements in $\mathscr{U}$ and $\mathscr{V}$ are seen as two high-dimensional datasets. To reduce the memory occupation and speed up the similarities computation, the data points are saved as compressed bitmaps using, for example, a Run-Length Encoding [148](d).

sponding vertices is defined as follows:

$$J(\mathbf{v}_i, \mathbf{v}_j) = \frac{\sum_b (\mathbf{v}_i[b] \wedge \mathbf{v}_j[b])}{\sum_b (\mathbf{v}_i[b] \vee \mathbf{v}_j[b])}, \tag{8.1}$$

where $\mathbf{v}_i[b]$ is the b-th element in the i-th row of the biadjacency matrix where $\mathbf{v}_i, \mathbf{v}_j \in \mathscr{U}$. The numerator computes the number of shared elements in $\mathscr{U}$ for $\mathbf{v}_i$ and $\mathbf{v}_j$, while the denominator counts the number of elements in the union of the two.

Dimensionality-reduction techniques requires the data to be in the form of dense matrices, i.e., losing the advantage of a sparse representation of the biadjacency matrices. Moreover, the resulting data matrix will not fit in memory for large graphs, e.g., containing tens of millions of vertices. To overcome this limitation, we propose to treat the bipartite graph as a collection of compressed bitmaps, also called bitsets, where every row in the biadjacency matrices is saved as a compressed bitmap. A bitmap is a data representation in which every element in the set is represented by a bit. Bitmaps permit the extremely fast computation of the Jaccard similarity, see Equation 8.1. The numerator is computed with a bitwise-AND between the two bitmaps and the denominator with a bitwise-OR. However, the memory occupation of the bitmap corresponds to the maximum number of elements in the set, i.e., the number of columns in our setting, making it identical to a dense representation. Compression techniques are used to address this problem and to dramatically reduce the memory occupation of data. A straightforward approach is to use run-length encoding (RLE) [148], in which repetitions of consecutive bits are stored as a single value as well as the number of times it occurs. In this way, the biadjacency matrices are transformed in two collections of compressed bitmaps representing the vertices, see Figure 8.4d. In WAOW-Vis we use the Roaring Bitmaps [100], which are a hybrid data structure that combines different compression schemes on chunks of the bitmap based on their characteristics, e.g. their sparsity. Roaring Bitmaps are up to two orders of magnitude faster than traditional set implementations and are used by several Big-Data processing engines such as Apache-Lucene [110] and -Spark [193]. In the next section we present how the k-nearest neighborhood graph, which is needed for the computation of the HSNE hierarchies, is built from a large collection of compressed bitmaps.

8.4.2 Sets Intersection Tree

Once the biadjacency matrices are converted into two collections of bitmaps, $\mathcal{U}$ and $\mathcal{V}$, we compute the k-nearest neighborhood graph for each collection. As the procedure is the same for both, we will concentrate on the case of $\mathcal{U}$. To the best of our knowledge, no data structure exists to efficiently compute the k-nearest neighbors among compressed bitmaps. To address this problem we propose a novel tree-based data structure; the Sets Intersection Tree (SIT). Each node in this tree represents an element in $\mathcal{U}$ and the SIT will support an efficient algorithm to calculate the k-nearest neighbors of a given query element q, represented by its bitmap, by using a special traversal algorithm. The efficiency of this traversal results from the possibility for an early termination, which enables us to skip testing many elements in $\mathcal{U}$. The early termination will be enabled by two criteria. First, each node contains a union of all bitmaps of its subtree, enabling a quick test to determine if q shares any common element with any node in the subtree by using a bitwise AND operation. Second, the special construction of the tree will enable us to evaluate a bound on the Jaccard similarities of all elements in a subtree using only the bitmap of this subtree's root node, the *pivot*. Before describing the traversal algorithm, we first detail the construction, as it will facilitate deriving the bound on the Jaccard similarity.

The actual **construction** of the tree works as follows. We select the element u with the lowest cardinality to be the root node of our binary tree. Its bitmap will be used as a pivot, hence the name, to partition the remaining elements in $\mathcal{U}$, into a set $\mathcal{U}_1$, which contains all elements intersecting u (an AND operation between the bitmaps will not result in a zero), and the rest $\mathcal{U}_2$. The set $\mathcal{U}_1$ will form the left, $\mathcal{U}_2$ the right subtree of u. The subtrees are build up recursively in the same manner, choosing a pivot of lowest cardinality and building the subtrees. A special case

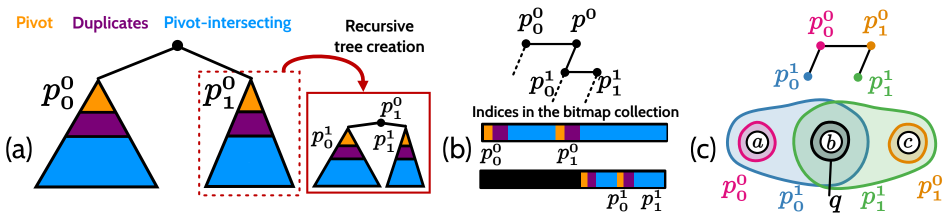


Figure 8.5: **Sets Intersection Tree** Bitmaps are organized in a number of subtrees. (a) Every tree contains bitmaps that are intersecting with the pivot, p_0^0 and p_1^0 in this case. Every sub tree is recursively divided in subtrees. (b) The SIT is implemented with a left-child right-sibling binary-tree. Bitmaps are not actually inserted in the tree but every node references to a linear array of indices. (c) The query for a set q starts from the root, i.e., p_0^0. All the siblings of a visited node are traversed, i.e., p_1^0. Children of a node are visited if the union of the sets in the subtree are intersecting q. Both p_0^1 and p_0^0 are visited in the example.

are identical elements, which do not appear multiple times in the tree, instead each node will contain the indices of the corresponding elements in $\mathcal{U}$. Additionally, we compute the union of the bitmaps in each subtree, which will be used for the early termination. We use a bottom-up method by performing an OR operation between the bitmaps of the children of each node.

The **querying** of k-nearest neighborhoods in the SIT works as follows. Given $q \in \mathcal{U}$ for which we want to find the k-nearest neighbors, we start a recursive visit from the root of the SIT, p_0^0 in Figure 8.5. During the traversal, we maintain a min-heap data structure of size k that keeps track of the closest neighbors found so far; each visited node is compared against the minimal element in the heap and replaces it if its Jaccard similarity (Equation 8.1; the intersection divided by the union) is larger. At the end of the traversal, the heap will contain the k-nearest neighbors.

For each node, we test q's bitmap against the precomputed union of the sets in its subtree M_0^0.If $J(q, M_0^0) \neq 0$, then the traversal continues with the children, otherwise, they have no overlap and cannot be similar (Jaccard similarity is zero). It is insufficient to test only against the bitmap of a node, as illustrated in Figure 8.5c; p_0^0 is not intersecting q (Jaccard similarity is 0), however, q intersects p_0^1 in the subtree, which has $J(q, p_0^1) = 0.5$.

An additional **early termination** criterion stems from the way that the SIT is constructed. By selecting the smallest set to be the pivot of a subtree, we are inherently introducing an ordering for the sets, i.e., the deeper a pivot, the larger it is. Because the denominator of the Jaccard similarity contains the union of the two sets, we can compute an upper bound for the similarities that we may find in a given sub tree. If the upper bound is lower than minimal element in the heap, we can avoid visiting the subtree.

Finally, for high efficiency, we propose a few **optimizations**, which we detail here. First, our tree does not actually store the bitmaps in the nodes, as this would lead to many copy operations of the data during the construction process. Instead, each node contains pointers to a large linear array of indices that contains all bitmaps in sequential order. The partitioning is then only affecting the indices, but not the bitmaps. Second, the construction of a subtree is stopped when only a few elements (typically 20) are left, as then the traversal cost actually exceeds the cost for testing all elements individually. This strategy follows bucket KD-Trees [120], where the final elements are stored in a list. This solution, together with the efficient computation of bitwise-AND and -OR granted by the RoaringBitmaps, enables us to compute the k-nearest neighborhood graphs containing several millions of nodes.

8.4.3 Hierarchical representation

The hierarchical representation of the biparite graph is generated by computing the Hierarchical Stochastic Neighbor Embedding (HSNE) [136]. We differ from the original HSNE algorithm, which is openly available as parte of the HDI library [134], as we compute the hierarchies starting from the k-nearest neighborhoods graph computed using the Jaccard similarities (Equation 8.1). This HSNE result allows us to create visual clusters of vertices in $\mathcal{U}$ that share connections to the other collection $\mathcal{V}$ in a multiscale approach **(T1)**. Furthermore, by combining compressed bitmaps, the SIT tree, and the HSNE algorithm, we are able to scale the computation to extremely large biadjacency matrices, making it possible to analyze dataset of a social-network scale.

More specifically, HSNE organizes the high-dimensional points or, in our case, the vertices, in a number of scales that are organized hierarchically. Each scale contains a number of landmarks that represent the complete data at the level of detail identified by the scale. Intuitively, a landmark is a collection of similarly connected vertices, where the degree of similarity is given by the position in the hierarchy. For lower scales, only vertices that shares very similar connections belong to the same landmark, while this constraint is relaxed the higher the scale in the hierarchy.

We denote the set of landmarks extracted from $\mathcal{U}$ at scale s as the collection $\mathcal{U}^s$. $\mathcal{U}^1$ represents the first scale, which is the input dataset $\mathcal{U}$. Higher scales are always subsets of previous scales, hence $\mathcal{U}^s \subset \mathcal{U}^{s-1}$. Inside a scale, the similarity between the landmarks is encoded by a transition matrix $T^s_{\mathcal{U}}$. For the first scale, $T^1_{\mathcal{U}}$ is given by the k-nearest neighborhood graph that is weighted by the similarities. Landmarks in the next scale are selected among those that have higher centrality in the graph. The centrality is computed by using the transition matrix $T^1_{\mathcal{U}}$ as a Markov Chain and by computing its stationary distribution with a Monte Carlo approach [50]. Vertices with value in the stationary distribution higher than a given threshold are selected to be landmarks in the higher scale $\mathcal{U}^2$.

A link between landmarks $\mathcal{U}^2$ to the landmarks in the lower scale $\mathcal{U}^1$ is then computed. More generally, it is defined as *area of influence* of $\mathcal{U}^s$ over $\mathcal{U}^{s-1}$ and is encoded in the matrix $I^s_{\mathcal{U}}$. $I^s_{\mathcal{U}}$ has size $|\mathcal{U}^{s-1}| \times |\mathcal{U}^s|$, where $I^s(i,j)_{\mathcal{U}}$ is the probability that the landmark $\mathcal{U}^{s-1}_i$ in the previous scale is well represented, i.e., close in the k-nearest neighborhood graph, by $\mathcal{U}^s_j$. The similarity matrix $T^s_{\mathcal{U}}$ for landmarks in the new scale s encodes the overlap of the area of influence of the landmarks $\mathcal{U}^s$. The process is iterated until only a limited number of landmarks, i.e., less then a thousand, remain in the highest scale. A similar hierarchical representation is derived for the collection $\mathcal{V}$. In the next section we present how the hierarchy is visualized and explored.

8.5 Exploratory analysis

In this section we present the design of the visualizations used to interactively explore the hierarchical data representation of the graph. First we present the layout of a single visualization (Section 8.5.1). Then we present how the dimensionality reduction is performed for every collection (Section 8.5.2). Finally, we explain how more detailed visualizations are generated from a subset of user-selected landmarks (Section 8.5.3).

8.5.1 Visual design

Figure 8.1 shows an instance of WAOW-Vis. The visualization consist of four embeddings. Every point in the embeddings represent a landmark in the corresponding HSNE scale. Landmarks are placed close together if they are similar according to their Jaccard similarities at the given scale. This allows us to identify groups of elements in the collection $\mathcal{U}$ that have similar connections to $\mathcal{V}$, and vice versa. Moreover, a landmark corresponds to a set of vertices in the original collection, as presented in Section 8.4.3. The size of points in the visualization encodes the number of vertices represented by the corresponding landmark [136].

At the center of the visualization two 1-dimensional embeddings, one for $\mathcal{U}$ and one for $\mathcal{V}$, are used to create a layout that is similar to a traditional node-link diagram for bipartite graphs. We adopt this layout because it is reported that node-link diagrams are more intuitive for understanding the graph [52]. By brushing on one of the embeddings, the vertices are selected and the links to the other collection are visualized as lines. These lines are then bundled with a real-time implementation of the force-directed bundling algorithm proposed by Holten and van Wijk [71]. Each collection $\mathcal{U}$ and $\mathcal{V}$ is also shown in a 2-dimensional embedding, as shown in Figure 8.1a and e. The rationale behind this choice is that, the more visual space is available for the landmarks, the more interrelationships between the clusters become apparent as it can be seen by comparing Figure 8.1a to Figure 8.1b **(T2)**.

Finally, we found that, if attributes are available for the element in the collection, it is useful for the understanding of the interrelationships between the clusters to add this information using, e.g., a word-cloud visualization. This feature leads to the identification and labeling of clusters of landmarks. In the WAOW-Vis presented in Figure 8.1, the two clusters in the collection $\mathcal{V}$ consists of Twitter feeds associated to two different domains, i.e., computer science and United States' news outlets.

8.5.2 Embedding computation and alignment

Without any additional constraint, the same clusters in the 1- and 2-dimensional embeddings might be placed in different positions along the vertical axis, which makes

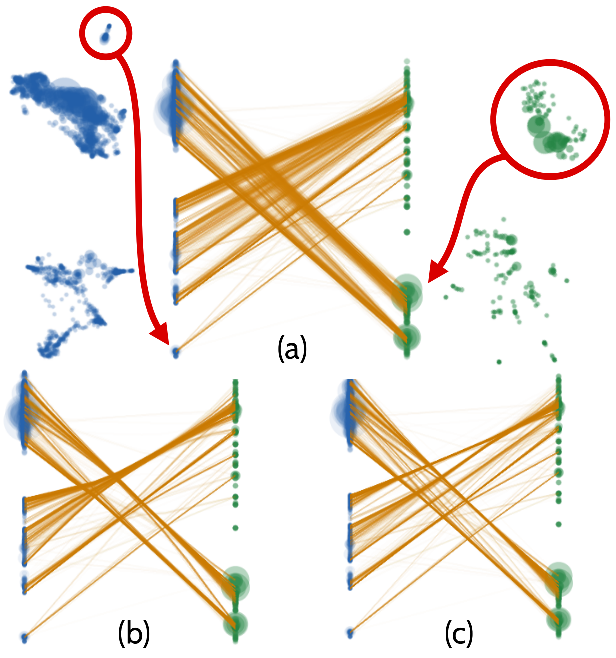


Figure 8.6: **Possible problems** that arise if the embeddings are generated independently from each other. Here, the same data presented in Figure 8.1a-e is embedded with a tSNE minimization [177] instead of our approach as presented in Section 8.5.2. Elements in $\mathcal{U}$ and $\mathcal{V}$ are badly aligned, hence cluttering the visualization of the links between the two collections. Moreover, the same cluster of landmarks may appear in different positions along the vertical axis (a). Bundling the lines reduces clutter but does not produce a neat layout as in Figure 8.1a-e (b,c).

associations between the related embeddings a difficult task. Example of possible problems are shown in Figure 8.6 for the same data presented in Figure 8.1a-e. The two clusters highlighted in red are in different positions along the vertical axis. Furthermore, the two collections are not properly aligned, hence creating a cluttered visualization of the links. By bundling the lines, as shown in Figure 8.6b-c, the problem is mitigated but it is not removed. To address this issue, we implemented a modified version of the tSNE algorithm [177] enforces similar positions on the vertical axis for all landmarks of the same collection and for similarly connected landmarks in $\mathcal{U}$ and $\mathcal{V}$.

A single embedding is computed by randomly placing the landmarks in a 1- or 2-dimensional space. With an iterative gradient-descent minimization, landmarks are then moved in the embeddings in such a way that, after a number of iterations, they are close to similar landmarks according to the transition matrix $T^s_{\mathcal{U}}$. In this way, clusters of landmarks in the embedding represent groups of similar elements. More specifically, we minimize the original tSNE's cost function C_e^{tSNE} to generate the embedding e which is defined as follows:

$$C_e^{\text{tSNE}} = KL(T^s_{\mathcal{U}}||Q_e), \tag{8.2}$$

where $KL(T^s_{\mathcal{U}}||Q_e)$ is the Kullback-Leibler divergence between the joint-probability distributions defined by the transition matrix $T^s_{\mathcal{U}}$ and Q_e. Q_e is a joint-probability distribution that is obtained by weighting the distances between the landmarks in the embedding e with the Student's t-distribution [177]. The points are then iteratively moved in the embedding along the negative gradient of the cost function C_e^{tSNE}, until their positions reflect the similarities in $T^s_{\mathcal{U}}$. We refer the interested reader to Chapter 3 for the detail on how Q_e is computed and how the gradient-descent parameters for C_e^{tSNE} are chosen.

In order to take into consideration the position of landmarks in a set of embeddings F, hence enforcing the alignment between the same landmarks, we modify the cost function C_e as follows:

$$\begin{aligned} C_e &= (1-\alpha)\, C_e^{\text{tSNE}} && +\alpha C_e^{\text{align}} \\ &= (1-\alpha)\, KL(T^s_{\mathcal{U}}||Q_e) && +\alpha \sum_{f\in F} \sum_{i\in\mathcal{U}^s} ||y_i^e - y_i^f||^2, \end{aligned} \tag{8.3}$$

where y_i^e is the vertical position of the landmark i in the embedding e that is iteratively optimized. For an embedding $f \in F$ containing landmarks from the same collection $\mathcal{U}^s$, y_i^f is the vertical position of i in f. For inter-collection embeddings, i.e., optimization of $\mathcal{U}^s$ from $\mathcal{V}^s$, y_i^f is computed as the mean position of the landmarks in f that are connected by an edge to the landmark i in e. C_e is the composition of two different costs, the C_e^{tSNE}, as presented in Equation 8.2, and C_e^{align}, which minimizes the squared distances between the position of a landmarks in the embedding e and in the embedding f. The parameter α controls the weight that is given to the two terms. For $\alpha = 0$ the cost function is the same as a traditional tSNE minimization, while for $\alpha = 1$, only the squared distances along the vertical axes are minimized.

As before, the embeddings are generated by moving the points in the opposite direction of the gradient of C_e. We found that good results are obtained if we optimize all embeddings for $\mathcal{U}$ and $\mathcal{V}$ simultaneously, letting each one influence the other during the minimization. Regarding the parameter α, we found that for our test cases it works well to start with a relatively high value, e.g., $\alpha = 0.5$. In this way, the landmarks are iteratively placed in similar positions along the vertical axes right from the start. However, we believe that the preservation of the similarities between similar vertices as computed by C_e^{tSNE} is of greater importance as it is the main insight that the user aims at achieving **(T1)**. For this reason we linearly reduce the value of α down to 0 after a number of iterations. Empirically, we found that a linear reduction of α to 0 in 500 iterations is a good strategy for all test cases.

8.5.3 Hierarchy exploration

The data exploration is implemented with a *filter* and *drill-in* strategy that starts from the highest scale in the HSNE hierarchy for both collections $\mathcal{U}$ and $\mathcal{V}$. Landmarks at this level of abstraction represent the main clusters. WAOW-Vis provides a multiscale exploration of the clusters, which is performed by letting the user select a set of landmarks in either of the two collections with a brushing interaction. The selection leads to a refined visualization that contains the influenced landmarks in the lower scale for the corresponding collection.

We now provide the details on the creation of embeddings that contain landmarks in lower scales of the hierarchy starting from a selection in a higher scale for one of the collection, e.g., $\mathcal{U}$. A similar approach is performed if the selection is within $\mathcal{V}$. Given landmarks $\mathcal{U}^s$ at scale s and a set of indices of selected landmarks A, the new visualization contains a subset of landmarks in $\mathcal{U}^{s-1}$, which are under the area of influence of the selection in s. As defined in Section 8.4.3, the area of influence of the landmarks associated to a scale s is defined by the matrix $I^s(i,j)_{\mathcal{U}}$. $I^s_{\mathcal{U}}$ has size $|\mathcal{U}^{s-1}| \times |\mathcal{U}^s|$, where $I^s(i,j)_{\mathcal{U}}$ is the probability that the landmark $\mathcal{U}^{s-1}_i$ in the previous scale is well represented by $\mathcal{U}^s_j$. The new embedding contains all landmarks $\mathcal{U}^{s-1}_k$ at scale $s-1$ for which the following is true:

$$\sum_{a\in A} I^s(k,a) > \theta, \tag{8.4}$$

where $0 < \theta \le 1$ is a user defined threshold. Intuitively, $\sum_{a\in A} I^s(k,a)$ represents the probability for the landmark $\mathcal{U}^{s-1}_k$ to be influence by the selection A of landmarks in $\mathcal{U}^s$. We experimentally found that a default value of $\theta = 0.5$, allows for the effective exploration of the clusters of vertices. Figure 8.1 shows an example. Once a cluster is selected, its detailed information at the lower scale can be visualized upon the user's request. In the next section, we provide further examples of how the hierarchical exploration of the data give richer insight on the hierarchy of clusters in the graph.

8.6 Implementation

WAOW-Vis is implemented in C++ for performance reasons and, when possible, heavily uses OpenMP [31] to parallelize computations. It fully supports the Progressive Visual Analytics paradigm [41, 119], allowing for the visualization of the evolutions of the embeddings, while the embeddings are iteratively generated. Therefore, the user does not have to wait a fixed number of iterations and can autonomously decide on the convergence of the embedding by evaluating their visual stability [138]. The compressed bitmaps are implemented using the C++ version of the Roaring Bitmaps library [100]. The modified version of the HSNE algorithm [136], presented in this chapter, is derived from the original C++ implementation available in the High-Dimensional-Inspector library [134]. The embeddings are implemented in OpenGL. The word-clouds are implemented in Javascript using D3 and are integrated in the

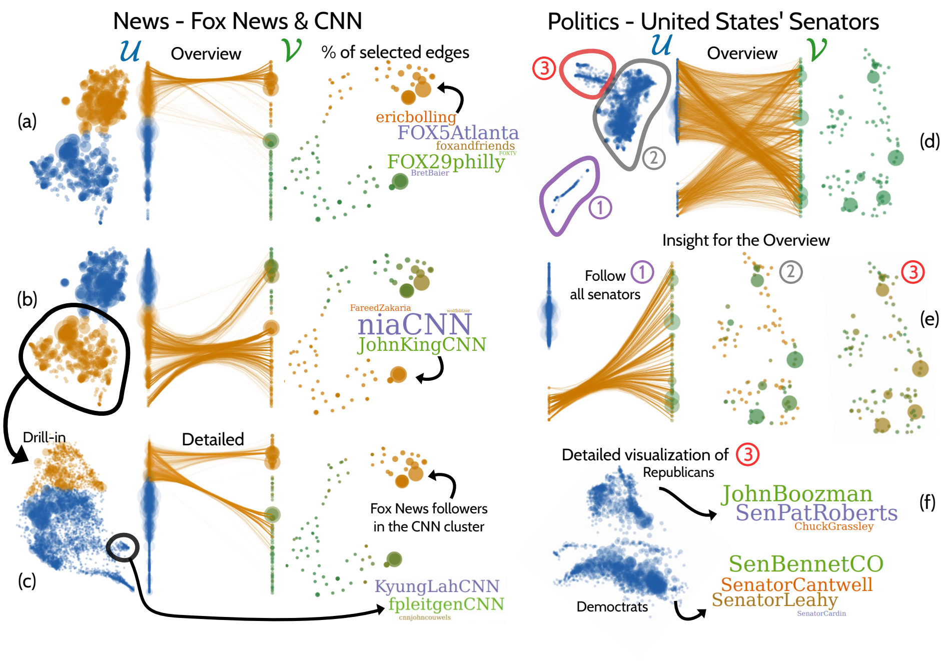


Figure 8.7: **Analysis of the United States' politics and news datasets**. The News dataset comprises a collection $\mathcal{V}$ of Twitter accounts of journalists from two of the United States' major news outlets, CNN and Fox News. Users that follows the feeds in $\mathcal{V}$ are in the collection $\mathcal{U}$. Two echo chambers [47] are identified in the dataset (a,b). Users in the top cluster are following only journalists from Fox News (a), while the ones at the bottom are following only CNN's journalists (b). In the two visualizations (a,b) only the edges linked to the selection are shown. In the right embedding, landmarks are visualized with a green-to-orange color scale that shows the percentage of incoming edges in the current selection. The clusters of orange-colored landmarks in $\mathcal{V}$ confirms the strong association for the selection in $\mathcal{U}$. By drilling into the CNN cluster, a sub community that follows Fox News accounts is identified (c) The analysis of the politics datasets does not show strong evidence of a polarized audience (d). A cluster containing users that follow all the senators is highlighted in purple (d,e). The cluster in gray contains users following the senators with the largest audience, while users in the red cluster follow senators with not so many followers (d,e). A detailed visualization of the red cluster reveal that a polarized audience exists for these senators (f).

C++ application using the QtWebKit Bridge. Finally, WAOW-Vis is released as part of the High-Dimensional-Inspector library [134].

8.7 Test cases

To evaluate WAOW-Vis we present real-world examples of the analysis of bipartite graphs of social-network scale. We identified three domains to analyze: computer science, news, and politics. For each one of these domains, we chose as elements of the collection $\mathcal{V}$ a number of Twitter-feeds, i.e., Twitter users that post mainly in the chosen domain. The collection $\mathcal{U}$ contains all the followers for the element in $\mathcal{V}$.

Therefore, the resulting bipartite graph encodes the *follower* relationships between the users and the Twitter-streams in the specific domain. In WAOW-Vis, clusters of similar users are communities that share similar interests in the specific domain, while clusters of similar feeds, share similar groups of followers.

We decided to focus on the analysis on the relationship between users and feeds in Twitter as we are interested in exploring the presence of echo-chambers, or filter bubble, in social networks. An echo chamber is a community of users that receives only polarized information concerning a specific domain, e.g., politics. The presence of echo chambers in social networks is deemed responsible for the polarization of the public discourse in recent years [47]. Contrary to other social networks like Facebook, follower relationships on Twitter are openly available through the Twitter-API, allowing us to test WAOW-Vis on real-world data.

Table 8.1 presents the overview on the datasets that we collected and analyzed. Every column corresponds to a bipartite graph associated with a specific domain. The computer science dataset contains Twitter-feeds associated with several programming languages, e.g., Java, C++, PHP and OpenGL. The news dataset contains journalists and presenters of two of the major United States' news outlets, i.e., CNN and Fox News. The politics datasets contains the Twitter accounts of every United States' senator. The last column presents an additional dataset which is the union of the previously introduced ones. The first three rows present the number of vertices in the collections and the edges connecting them. Note that, for the presented datasets, $|\mathscr{U}| \gg |\mathscr{V}|$ due to scalability issues of the Twitter mining. Twitter imposes a limitation in the number of user-followers relationships, i.e., edges in our graph, that can be obtained per minute. This limit is of 5000 links per minute, hence it required approximately 9 effective days of mining for gathering the datasets.

In Table 8.1, we present the computation time in seconds for the three processing steps presented in Section 8.4, together with the maximum memory occupation of WAOW-Vis for each dataset. The results are generated on a workstation with an 3.40GHz Intel i7-2600 CPU and 20 GB of memory. The computation of the hierarchies needed for the analysis of the largest dataset, which contains 19.7M users, takes less than one hour. To give a perspective on size, Twitter has an estimated number of 340M active daily users. We performed a comparison with the traditional HSNE hierarchy computation, which relies on the FLANN library [120] for the similarity computation, using the dense representation of the graph. However, the computation of the HSNE hierarchy is impossible to perform for all the datasets due to the heavy memory requirements, hence demonstrating the need for a novel approach as presented in this work.

8.7.1 News dataset

Figure 8.7a-b shows the exploratory analysis of the news dataset performed using WAOW-Vis. Two separated clusters of Twitter feeds are visible in the right embedding, i.e., $\mathscr{V}$. By visualizing the owners of the Twitter feeds in the word cloud, we realize that they belong to Fox News journalists for the cluster at the top and to CNN

Table 8.1: **Datasets** information are presented in the columns. The first three rows present the size of the biparite graph, where (M) means millions of elements. Computation time in seconds for the SIT creation, kNN and HSNE computation are presented. Finally, peak memory occupation of WAOW-Vis is given.

	Computer S.	News	Politics	Combined
$\lvert\mathcal{U}\rvert$	1.97M	7.67M	12.42M	19.7M
$\lvert\mathcal{V}\rvert$	145	83	100	329
#Edges	4.4M	10.8M	24.5M	38.9M
SIT (s)	13	34	63	283
kNN (s)	50	59	430	1960
HSNE (s)	13	11	60	202
Mem. (GB)	1.6	2.1	3.7	10.6

journalists for the one at the bottom. This insight is confirmed by the visualization of the edges connecting the two collections. Contrary to the visualizations that we presented so far, in Figure 8.7a-b we show the edges related to a user-defined selection of landmarks. Selected landmarks in the left embedding are rendered with a shade of orange, and only edges connected to these landmarks are shown. Most of these edges are connected to the top cluster in the embedding on the right. Here, landmark colors encode the percentage of incoming edges that are currently selected by the user with a green-to-orange color scale. The top cluster in the right embedding of Figure 8.7a has the same shade of orange of the selection in the left embedding. This means that the current user selection among $\mathcal{U}$ is mainly connected to the top cluster in $\mathcal{V}$. The same observation can be done for the cluster at the bottom of the visualization, as shown in Figure 8.7b.

For both selections, only a small number of edges are connected to the opposite cluster. This insight leads to the conclusion that two echo chambers [47] exist for the two news outlets. However, in Figure 8.7b, we can observe that a more consistent stream of edges is connecting CNN followers to the Fox News feeds. In order to reveal more sub-communities within the cluster, a detailed visualization is generated by drilling into the hierarchy. Figure 8.7c shows the resulting embedding. The selection in the embedding contains all the landmarks (i.e., group of users) that follow Fox News' accounts. Finally, by selecting the small cluster which is encircled in Figure 8.7, followers of international CNN reporters, such as Kyung Lah and Frederik Pleitgen, are identified.

8.7.2 Politics dataset

Figure 8.7d-f shows the exploratory analysis of the politics dataset. In this test case, we expected to see an echo chamber for users connected to the Republican senators and one for those connected to the Democratic senators. However, a clear separation for the collection $\mathcal{V}$ is not visible in the visualization shown in Figure 8.7d. To better understand the interrelationships between the two collections, the user interacts with WAOW-Vis in order to get a more detailed insight on the visible clusters. The cluster ① in Figure 8.7d is the most distinct one. To understand the connections of the clusters ① to the collection $\mathcal{V}$, the edges linked to it are visualized. Figure 8.7e shows that the selected landmarks are connected to the accounts of every senators. We conclude that this clusters contain political enthusiasts or, more likely, software-controlled Twitter accounts. These accounts, also known as Twitter-bots, work as tweet aggregators for a specific domain, for example, by automatically reposting the senators' tweets.

The cluster ② is then selected by the user. A visualization of the percentage of the incoming edges in the right embedding is visualized in Figure 8.7e. Only the large points in the embedding are colored with a shade of orange. These points correspond to the senators with the largest user base, such a senator Elisabeth Warren and Marco Rubio. This result shows that the cluster ② identifies the community of users who are following mainly the most famous politicians. Finally, the cluster ③ in Figure 8.7d corresponds to users following senators with a much smaller user base, as can be seen by the result of the selection in Figure 8.7e. In the overview, cluster ③ already shows a separation in two sub clusters. A detailed visualization, which is generated by drilling in the hierarchical representation, shows a better separation of these clusters. The resulting embedding, which is presented in Figure 8.7f, shows that a polarization of the users exists for Republicans and Democrats in this sub community.

8.8 Conclusions and Future Work

In this chapter we have presented Who's-Active-On-What-Visualization (WAOW-Vis), a visual analytics system for the exploratory analysis of large bipartite graphs. We presented a novel graph preprocessing pipeline that is inspired by the recent developments in the analysis of large high-dimensional data. The scalability of WAOW-Vis is enabled by three main contributions. The adoption of compressed bitmaps for representing the graph and the novel Sets Intersection Tree (SIT) for efficiently computing Jaccard similarities between the bitmaps. The similarities are then used to generate a hierarchical representation of the graph with a modified version of the Hierarchical Stochastic Neighbor Embedding (HSNE). Moreover, we presented several insights obtained by the exploratory analysis of large datasets that we mined from Twitter.

WAOW-Vis, however, does not come without limitations. First, while WAOW-Vis can handle very large bipartite graphs, it can only handle undirected and unweighted graphs. Extending our technique to handle weighted and directed graphs is an interesting future work. Furthermore, in the exploratory analysis of the data, the several visualizations that are generated make it difficult to keep a mental mapping of the exploration process. Höllt et al. recently proposed CyteGuide [69] for guiding the user in the exploration of a single HSNE hierarchy. An interesting future work is the development of a similar approach for guiding the data exploration in WAOW-Vis.

Moreover, the visualizations of the links between the two collections is limited to the 1-dimensional embeddings. This may be a limitation as most of the insights are obtained through the analysis of the 2-dimensional embeddings. Finally, in the test cases we analyzed datasets where the two collections are very different in size, due to the query limit imposed by Twitter. Because we are not limited to this kind of datasets, it would be interesting to experiment with more balanced bipartite graphs. Finally, bipartite graphs are widely used in biomedical research [61] and for the visualization of deep neural networks [115]. An interesting research direction is the application of WAOW-Vis in the biomedical research field as presented in Chapter 7.

9 DeepEyes: Progressive Visual Analytics for Designing Deep Neural Networks

A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

Asimov's first law of robotics

In this chapter we present DeepEyes, a Progressive Visual Analytics system that supports the design of neural networks during training. We present novel visualizations, supporting the identification of layers that learned a stable set of patterns and, therefore, are of interest for a detailed analysis. The system facilitates the identification of problems, such as superfluous filters or layers, and information that is not being captured by the network. We demonstrate the effectiveness of our system through multiple use cases, showing how a trained network can be compressed, reshaped and adapted to different problems.

N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. IEEE transactions on visualization and computer graphics, 24(1):98–108, 2018 [137].

9.1 Introduction

In this chapter we present how high-dimensional data analysis, and in particular the HSNE algorithm presented in Chapter 6, is beneficial for the analysis of Deep Neural Networks (DNNs), a class of algorithms that have shown outstanding performance in various problems, like image and speech recognition [94]. DNNs consist of various interconnected **layers**. In each layer, a number of **filters** detect increasingly complex **patterns**. For example, in networks trained to recognize objects in an image, the first layer generally contains filters that are trained to detect colors and edges. This information is aggregated by other layers to detect complex patterns, e.g., grids or stripes. By using hundreds or thousands of filters in each layer, DNNs allow for more complex patterns to be learned. Only recently the training of large DNNs was made possible by the development of fast parallel hardware, i.e., GPUs, and the creation of large training sets [88].

While the results that DNNs can achieve are impressive, they essentially remain a black box. An increasing research effort is spent on making the visualization and the analysis of these models feasible. While both, the machine learning and the visualization community, invested considerable effort in understanding how a trained network behaves [103, 145, 194], e.g., by showing the patterns learned by the filters, little effort has been spent on the creation of tools that support design decisions given the pattern recognition problem at hand. Even though basic design guidelines

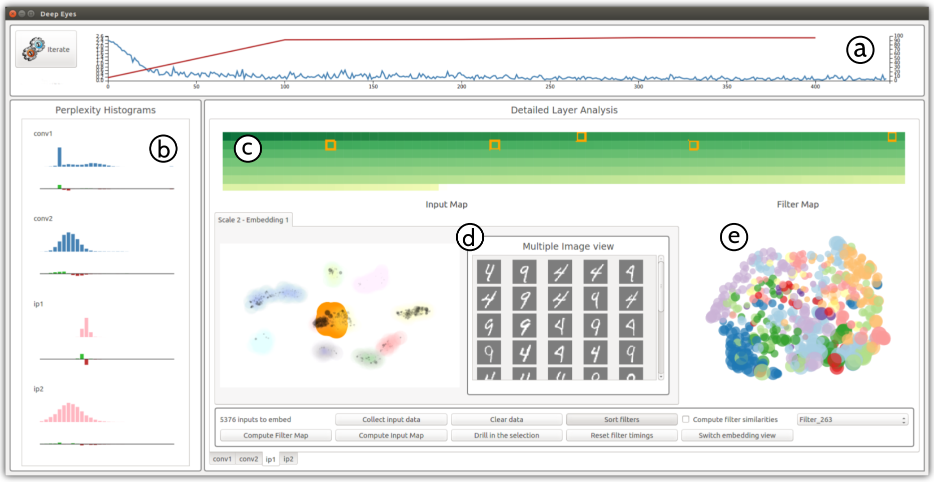


Figure 9.1: **DeepEyes** is a Progressive Visual Analytics system for the analysis of deep neural networks during training. The overview on the training is given by the commonly used loss- and accuracy-curves (a) and the Perplexity Histograms (b) a novel visualization that allows the detection of stable layers. A detailed analysis per layer is performed in three tightly linked visualizations. Degenerated filters are detected in the Activation Heatmap (c), and filter activations are visualized on the Input Map (d). Finally, in the Filter Map (e), relationships among the filters in a layer are visualized.

exist, the process of designing a neural network is an iterative trial-and-error process [4]. For example, experts can change the number of layers or filters per layer but the effect of a modification only becomes obvious after hours, days or weeks, as the network needs to be retrained, a lengthy task given the size of the datasets involved. A visual analytics approach for the analysis of a deep network therefore seems necessary [85]. As mentioned in Chapter 2, a recent paradigm called Progressive Visual Analytics, aims at improving the interaction with complex machine learning algorithms [41, 119, 138, 165]. This interaction is achieved by providing the user with visualizations of the intermediate results while the algorithm evolves, the training of the network in this setting. However, the size of DNNs makes the application of the Progressive Visual Analytics paradigm challenging, requiring the development of visualizations that heavily rely on data aggregation at interactive rates [42, 136, 138, 174].

In this chapter, we present DeepEyes, a Progressive Visual Analytics system that supports the design of DNNs directly during training. After discussing with machine learning experts that collaborated in the design of DeepEyes, we came to realize that the existing work provides limited feedback on how a DNN can be improved by the designer. To overcome this limitation, we identified the following analytical tasks as critical to make informed design-decisions while the network is trained:

(T1) **Identification of stable layers** which can be analyzed in more detail, effectively facilitating the detailed analysis while the network is trained

(T2) **Identification of degenerated filters** that do not contribute to the solution of the problem at hand and, therefore, can be eliminated

(T3) **Identification of patterns undetected** by the network, which may indicate that more filters or layers are needed

(T4) **Identification of oversized layers** that contain unused filters and, therefore, can be reduced in size

(T5) **Identification of unnecessary layers or the need of additional layers**, allowing for the identification of an efficient architecture for the problem at hand

The main contribution presented in this chapter is the DeepEyes framework itself. For the first time, DeepEyes integrates mechanisms to tackle all presented tasks to analyze DNNs during training into a single, progressive visual analytics framework. The development of DeepEyes is enabled by a set of further contributions presented in this chapter:

- a new, data-driven analysis model, based on the sampling of sub-regions of the input space, that enables progressive analysis of the DNN during training
- Perplexity Histograms, a novel overview-visualization that allows the identification of stable layers of the DNN for further exploration

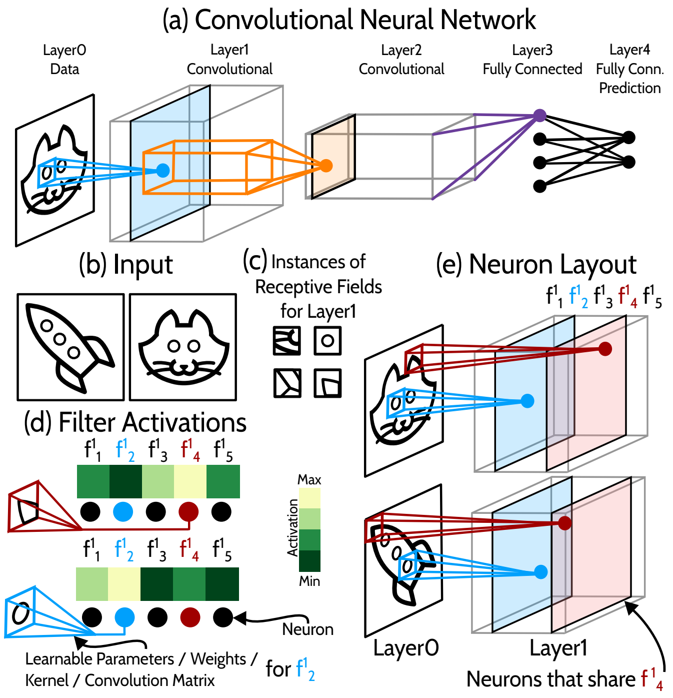


Figure 9.2: **Overview of a DNN** (a). Filter functions are computed by neurons in convolutional layers by applying a kernel or convolution matrix on a subsets of the input (b), called **Receptive Field**, whose instances are image patches (c). Filter functions are trained to detect different receptive field instances (d) and they are organized in a 3D grid (e) according to the spatial relationships of the receptive fields they compute.

- a set of existing visualizations have been extended or adapted for our data-driven approach to allow detailed analysis: Activation Heatmap, Input Map, and Filter Map.

In the next section, we provide the reader with a primer on DNNs, with the essential components to understand our contributions and the related work, presented in Section 9.3. In Section 9.4, we present DeepEyes, describing our visualization design based on the insights and support we want to provide to the DNN designer. Furthermore we provide a first example of a DNN for the classification of handwritten digits. Two different use cases are provided in Section 9.5, while implementation details are given in Section 9.6.

9.2 Deep Learning Primer

Deep artificial neural networks are trained on a specific pattern recognition problem, such as image classification. The goal is to predict a class of an unseen sample.

A training set consists of a set of high-dimensional inputs $\mathbf{x} \in \mathbb{R}^n$ together with an associated vector $\mathbf{y} \in \{0,1\}^d$ with $\sum_i \mathbf{y}_i = 1$, where d is the total number of labels. The only non-zero component indicates the associated label. The goal of a DNN is to predict the label $\tilde{\mathbf{y}} \in [0,1]^d$ for an unseen input $\tilde{\mathbf{x}} \in \mathbb{R}^n$. The prediction is usually in the form of a discrete probability distribution over the possible labels, hence $\sum_i \tilde{\mathbf{y}}_i = 1$.

A DNN consists of a set of layers $\mathscr{L}$. An example of a DNN that comprises five layers, more specifically one data layer, two convolutional layers and two fully-connected layers, is presented in Figure 9.2a. Independently from the nature of the layer, every layer $l \in \mathscr{L}$ contains a set of **neurons** that computes **filter functions** $f_i^l \in \mathscr{F}^l$, or, more concisely, **filters**. However, exactly the same filter can be computed by many neurons in the same layer. In the example in Figure 9.2b-e the input consist of images, where each pixel is a dimension in our input space $\mathbb{R}^n$. Filter functions in Layer1 do not take the full dimensionality $\mathbb{R}^n$ as input, but rather a subsets $\mathbb{R}^{k^l} \subset \mathbb{R}^n$, the **receptive fields** where k^l represents the size for layer l. For images, these subsets are patches and a few instance of these patches are presented in Figure 9.2c. Mathematically, a specific receptive field $\delta_r^l \in \Delta^l$ for layer l is a set of indices $\delta_r^l := \{i_j\}_{j=0}^{k^l} \subset \{0 \dots n\}$ that defines a corresponding projection function $\pi(\delta_r^l): \mathbb{R}^n \to \mathbb{R}^{k^l}, (x_0, \dots x_n) \to (x_{i_0}, \dots, x_{i_{k^l}})$. We now focus on the relationship between filters and neurons given an **instance of a receptive field**, i.e., a specific patch for a specific input image $\mathbf{x}$ identified by the projection function $\pi(\delta_r^l)(\mathbf{x})$. In Figure 9.2d a heatmap is shown to illustrate the output of filter functions $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$, also called **filter activations**, given specific instances of receptive fields $\pi(\delta_r^l)(\mathbf{x})$. In the first layer, the filter function is usually a weighted sum of the pixel values on the receptive field. These **weights** are the **learnable parameters** that are trained to detect specific patterns in the data. Further, the weights define the filter function and are the same for all neurons computing this filter. In the example, f_4^1 detects , having high filter activation, while f_2^1 detects .

Given a single instance of a receptive field, as or in Figure 9.2d, a **1-to-1 correspondence** exists between filters and neurons (represented as points in Figure 9.2). However, when the full input is considered, neurons that share the same filter function but process a different location in the image, i.e. receptive field, are organized in a grid-like layout that mimic the input shape. The layout for Layer1 is illustrated in Figure 9.2e, where neurons that compute the same filter function are placed on planes. By stacking these planes, the resulting layout is a 3D grid of neurons. Filter functions give better information on the detected patterns than single neurons, as they are pattern detectors learned by the layer independently of the position in the input. Note how, in Figure 9.2e, is detected by the filter f_2^1 which is computed by different neurons, i.e., where eyes and portholes are located.

The same description holds for any layer, as shown in Figure 9.3a. Here, the receptive fields are larger in deeper layers and the filter functions are trained to detect more complex patterns. For example, is detected by the filter f_1^2 as it has a high activation. The main difference from Layer1 is that, the filter functions are

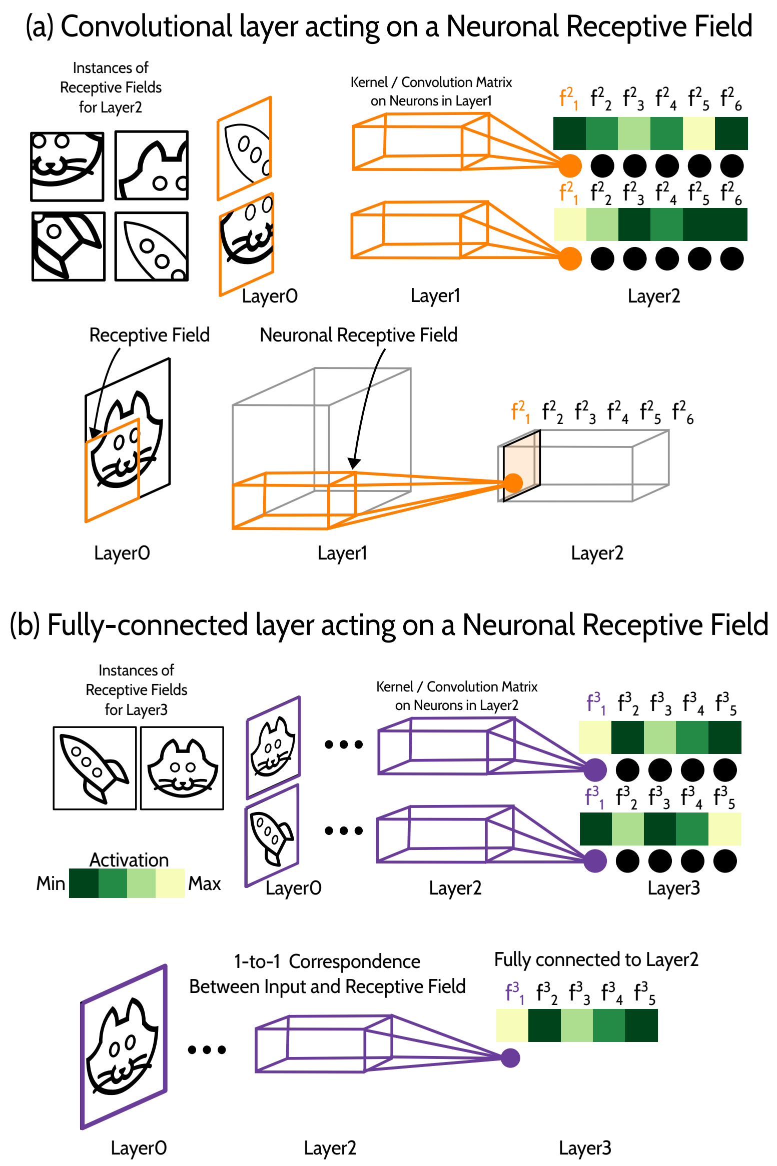

(a) Convolutional layer acting on a Neuronal Receptive Field
Instances of Receptive Fields for Layer2
Kernel / Convolution Matrix on Neurons in Layer1
f^2_1 f^2_2 f^2_3 f^2_4 f^2_5 f^2_6
Layer0
Layer1
Layer2
Receptive Field
Neuronal Receptive Field
(b) Fully-connected layer acting on a Neuronal Receptive Field
Instances of Receptive Fields for Layer3
Kernel / Convolution Matrix on Neurons in Layer2
f^3_1 f^3_2 f^3_3 f^3_4 f^3_5
Activation
Min
Max
Layer3
1-to-1 Correspondence Between Input and Receptive Field
Fully connected to Layer2

Figure 9.3: **In deeper layers**, filter functions are trained to detect more complex patterns in larger receptive fields. In convolutional layers a subset of the neurons in the previous layer, the neuronal receptive field, is the input to the filter functions rather than the receptive field instance (a). The same description holds for a fully-connected layer, however, it differs from convolutional layers as the receptive field of a neuron corresponds to the complete input and the neuronal receptive field contains all the neurons in the previous layer (b).

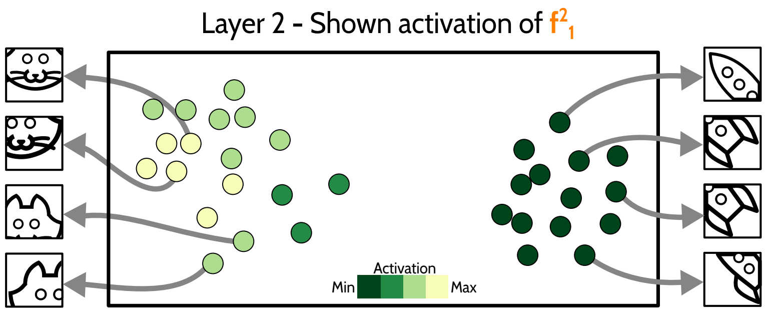


Figure 9.4: **DeepEyes approach** to filter analysis. Instances of receptive fields are sampled and embedded in a 2-dimensional space based on the similarity of the activation in the **neuronal receptive-field**. Activation of filters is highlighted in the resulting scatterplot and the instances of the receptive fields are visualized in a linked view.

not a direct expression of the dimensions in the receptive field in Layer2. In this layer, the filter functions consider as input a subset of the neurons in the previous layer, whose receptive fields are fully contained in the receptive field for Layer3. We define the region in the 3D grid of neurons in the previous layer as the **neuronal receptive field** of the neuron in the considered layer. The filter activation is obtained by weighting the activation of the neurons in the neuronal receptive field. The neurons in Layer2 are also organized in a 3D grid according to the relationships between the receptive fields and the filters. In Figure 9.3b, the computation for the fully-connected Layer3 is presented. Similarly to Layer2, Layer3 takes the neuronal receptive field in the previous layer as input. The receptive fields of filters in fully-connected layers correspond to the complete input, hence there is no need for a 3D grid of neurons. For this reason, a 1-to-1 correspondence between filters and neurons exists, meaning that a filter function is computed by just one neuron.

In this section, we provided an overview of the relationships between relevant elements of the DNN. We only briefly introduced the learnable parameters, or weights, involved in the convolutional or fully-connected layers. These parameters are learned by optimization given the training set. In modern architectures many different layers are used to define the filter functions, e.g., max-pooling and normalization layers. The concepts introduced so far hold, as filters are defined as a composition of the operations performed by different types of layers. For the intereseted reader we refer to LeCun et al. [94] for a more broad overview.

In DeepEyes we rely on the idea that, independently from the chosen layers, input data or receptive field instances are usually interpretable by humans, while abstract weights and relationships between neurons are not [106, 194]. Figure 9.4 provides an intuition of the central approach that we take in DeepEyes for analyzing what patterns a layer is trained to detect. The user creates a 2-dimensional representation of the instances of receptive fields used in the training. Instances that are perceived as similar by the layer, i.e. have similar activation in the neuronal receptive field, are close in the 2-dimensional visualization. Specific filter activation is then highlighted on demand, allowing for the understanding of the response of the filter to the input. For example, in Figure 9.4 we see that a separation of the receptive fields according to the input label, i.e., cat and rocket which are visualized in linked views, is available and the visualized activation of filter f_1^2, is strongly correlated with the cat label. Note that, despite the focus on the analysis of DNNs for image classification, the proposed approach is general as it focuses on filter activations and can be extended to different types of data, e.g., text or video, if appropriate linked views are used [84].

9.3 Related Work

Existing visualization techniques for DNNs can be divided in **weight-centric**, **dataset-centric** and **filter-centric** techniques.

Weight-centric techniques aim at visualizing the relationships between filters in different layers through the visualization of the learnable parameters, or weights, introduced in Section 9.2. A straightforward visualization for the weights are node-link diagrams [146], similar to the one presented in Figure 9.2a for the connection of Layer3 to Layer4. Here weights can be encoded in the edges, e.g., as line thickness. However, this approach does not scale to state-of-the-art networks that comprise millions of connections, limiting the application of weight-centric techniques mainly to didactic purposes [56]. To reduce the clutter generated on such networks, Liu et al. recently proposed a biclustering-based edge bundling approach [103] that aggregates neurons and bundles edges. Neurons are aggregated if they are activated by data that share the same label, while edges are bundled if they have similar and large absolute weights. However, in DNNs, neurons are trained to separate labels only in the last layers, therefore this clustering is not informative in early layers. For example, in Figure 9.2e the filter f_2^1 activates both on 🐱 and 🚀, an information that does not reveal the pattern ⊡ that the filter is trained to detect. Moreover, while the system allows a real-time exploration of the network, the creation of the visualizations requires hours of preprocessing, making the analysis of the network during training unfeasible. DeepEyes does not provide a weight-based visualization. After discussing with the machine learning experts involved in the development, we realized that it is more important to focus on the analysis of filters as pattern detectors, rather than on individual neurons and their connections [194].

The goal of **dataset-centric** techniques is to provide a holistic view on how the input data are processed by the network rather than providing a solution to the previously introduced tasks **(T1,T2,T3,T4,T5)**. The training- or the test-set is processed by the DNN and the activations of neurons in the last fully-connected layer are collected as high-dimensional feature vectors. Using non-linear dimensionality-reduction techniques, the dimensionality of the feature vectors is reduced to two dimensions and visualized in a scatterplot [2,82,116]. Two data points that are close in the 2-dimensional space are also close in the feature space, meaning that the network perceives them as similar. Recently, Rauber et al. [145] showed the evolution of this representation during training, in Chapter 6 we showed that hierarchical information is learnt by DNNs even though this information is not encoded in the training set. While these techniques provide insight on how the network reacts as a whole, they are limited to the analysis of the last fully-connected layer of the network. The only work in the analysis of hidden layers, i.e., not the input- or last-layer, is from Rauber et al. [145] where 2D embeddings are generated for hidden and fully-connected layers. This approach suffers from a severe limitation, being restricted to the analysis of layers where a **1-to-1 correspondence** between neurons and filter functions exists, i.e., fully-connected layers. We extend their work such that it can be used for convolutional layers which are the most widely used layers in modern day architectures [60,88,94,163,166].

Filter-centric techniques aim at giving an intuition on the pattern that a filter f_i^l is trained to detect. A straightforward approach presented by Girshick et al. [54] identifies for each filter f_i^l the instance of a receptive field $\pi(\delta_r^l)(\mathbf{x})$ with the highest activation $f_i^l(\pi(\delta_r^l)(\mathbf{x}))$. The instance of a receptive field $\pi(\delta_r^l)(\mathbf{x})$ is then presented to the user, e.g., as an image patch. A more complex approach aims at inverting the filter function f_i^l by defining $(f_i^l)^{-1}$, allowing for the reconstruction of the receptive field $\pi(\delta_r^l)(\mathbf{x})$ that produces the highest activation for f_i^l [38,106,124,192,194]. However, the explicit definition of $(f_i^l)^{-1}$ is not possible and it is approximated using deconvolutional neural networks [194]. This approach generates images that can give the intuition of the patterns detected by the filters, as demonstrated by Google's Deep Dream [118], and can be further extended for different tasks, such as style transfer [48]. However, according to the feedback provided by machine learning experts, the reconstructed receptive fields can be difficult to interpret for complex patterns, i.e., for late-layers in the network, and do not allow for a reasoning on architectural decisions **(T4,T5)**. Moreover, the reconstruction of the receptive field is a minimization process itself that is time consuming, requires complex regularization techniques and may produce misleading results [106,192] Filter-centric techniques are powerful tools but are generally limited to the analysis of a single and well-behaving filter, making their application for the analysis of a neural network during training difficult. DeepEyes includes novel filter-centric techniques for the identification of badly trained filters **(T2)** and provides a holistic view on filter activations given instances of receptive fields.

Finally, a recently proposed filter technique visualizes relationships between filters, i.e., how similarly they activate on the input and which label they are most strongly associated with [145]. Filters are represented as points and placed in a scatterplot by a multi-dimensional scaling algorithm [19]. Filter-to-label association is then highlighted by coloring every point with the color of the most correlated label. While this filter-centric technique allows for newer insights **(T3)**, it has two limitations that we overcome with a novel approach. First it requires the analysis of the complete dataset and, second, it cannot be applied to convolutional layers.

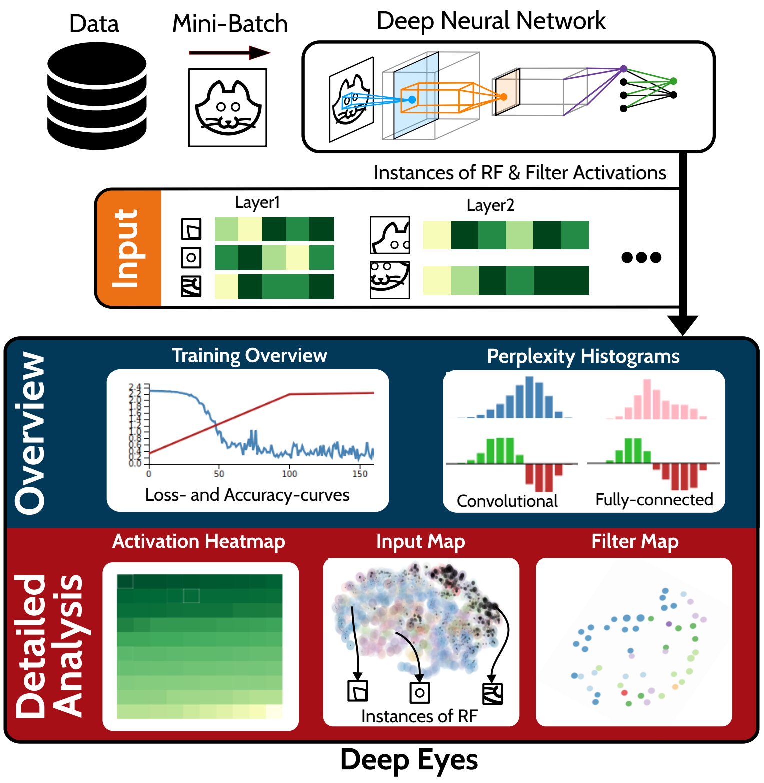


Figure 9.5: **Overview** of the DeepEyes system. The network training overview provided by the loss- and accuracy-curves is integrated with the Perplexity Histograms that allow for the identification of stable layers in the network (blue background). The user focuses on stable layers that are analyzed in detail with three tightly linked visualizations, namely the Activation Heatmap, the Input Map and the Filter Map (red background).

9.4 Deep Eyes

In this section, we introduce DeepEyes, a Progressive Visual Analytics system for the analysis of DNNs during training that combines novel data- and filter-centric visualization techniques. We start with an overview of DeepEyes in relation to these tasks in Section 9.4.1. A detailed description is provided in Sections 9.4.2 to 9.4.5. As a running example throughout this section we use the MNIST dataset [95] which consists of a training set of 60K images and 10K validation images. We train with the Stochastic Gradient Descent [97] the MNIST-Network that is provided in Caffe [79], a commonly used deep learning library which provides the deep-learning framework for DeepEyes. The network comprises two convolutional layers, with 20 and 50 filters respectively, and two fully connected layers with 500 and 10 filters respectively. Note that we use the MNIST-Network as proof of concept of our implementation and, for the sake of reproducibility, we use the architecture and training parameters provided by Caffe even if they do not achieve state-of-the-art results in classification performance.

9.4.1 Overview

Figure 9.5 shows an overview of our system. A DNN is trained by computing the filter activations on subsets of the training set, called *mini-batches*. The loss function, which measures how close the prediction matches the ground truth, is computed and the error is back propagated through the network. The learnable parameters of the network are then updated in the opposite direction of the gradient of the loss function [94, 97]. DeepEyes builds on the notion that the understanding of the relationships between instances of receptive fields $\pi(\delta^l_r)(\mathbf{x})$, which can be visualized and understood by humans, and the activation of filter functions $f^l_i(\pi(\delta^l_r)(\mathbf{x}))$ is crucial for understanding the patterns detected by the network in every layer.

For every mini-batch that is used to train the network, we sample instances of the receptive fields for every layer and the corresponding filter activations. Unless the user specifies otherwise, we sample a number of instances that grants a coverage of at least 50% of each input. This information is used to create a continuously-updated dashboard that provides insights into which patterns are detected by the layers in the DNN. In the Training Overview, loss and accuracy over time are presented. We complement this standard visualization, with a novel visualization, the **Perplexity Histograms** (Section 9.4.2), which allows for identifying when a layer learned to detect a stable set of patterns **(T1)**. The detailed analysis of stable layers is performed using three tightly-connected visualizations, highlighted in red in Figure 9.5. The **Activation Heatmap** (Section 9.4.3) allows for the identification of degenerated filters **(T2)**, while the **Input Map** (Sec 9.4.4) shows the relation of filter activations on instances of receptive fields for a given layer **(T3)**. Finally, the **Filter Map** shows how similar the filters activate on the input. Interaction with the Input- and Filter-Map support the identification of oversized and unnecessary layers **(T4,T5)**.

9.4.2 Perplexity histograms as layer overview

The evolution of the loss- and accuracy-curve presented in the Training Overview, is the de-facto standard way to visualize the evolution of the network during training. However, this visualization only provides information about the global trend of the training and fails to give a per-layer visualization of the changes. Given the size of the network, it is important to guide the user [23] towards layers that can be analyzed in detail while the training progresses, i.e., layers that learned a stable set of patterns **(T1)**. Our solution is based on the notion that every filter in a layer is trained to identify a certain pattern for a specific receptive-field size [194]. Therefore, we propose to treat every layer as a classifier designed to detect patterns, which are unknown at this moment, and we analyze its performance over time. More specifically, we want to know if the classifiers' ability to detect patterns

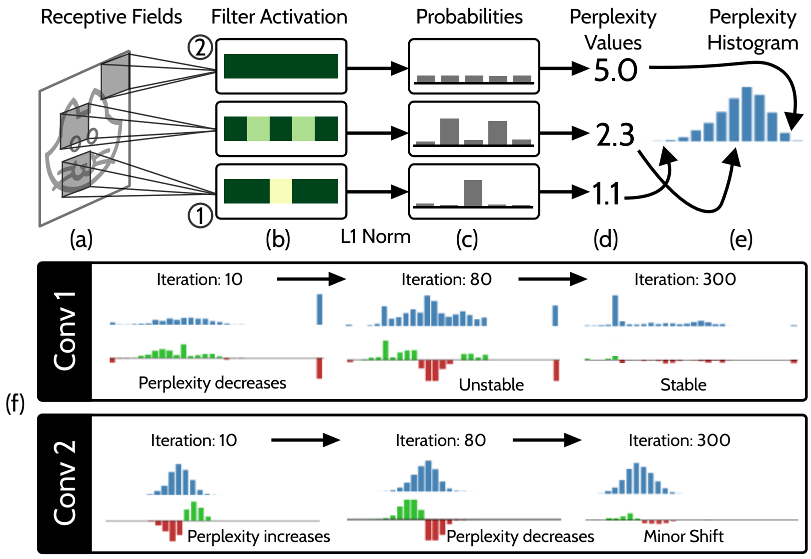


Figure 9.6: **Perplexity Histograms** and their creation. Receptive fields are sampled for every input data (a). The activation of the neurons that correspond to the receptive fields are collected, i.e., the receptive field's depth column (b). The depth columns are transformed in probability vectors (c) whose perplexity is computed (d) and used to populate the perplexity histogram (e). (f) shows the evolution of the perplexity histograms for the layer *Conv1* and *Conv2* in the MNIST-Network. Changes in the histogram over time are presented in a second histogram, highlighting the changes with red and green bars, for decreasing and increasing numbers, respectively.

is stable, increasing, or decreasing during training. If it is stable, it means that the layer learned what it was able to learn. If it decreases, the knowledge that this layer provides to the network is decreasing, and inversely when increasing.

We encode the layer stability as follows. For every input in a mini-batch, we randomly sample a number of instances of receptive fields (Figure 9.6a) and the corresponding filter activations (Figure 9.6b). We transform the activations in a probability vector $\mathbf{p} \in \mathbb{R}^{|\mathcal{F}^l|}$, where $|\mathcal{F}^l|$ is the number of filters in the layer l, by applying a L1-normalization (Figure 9.6c). Then, we compute for every receptive field instance the value of **perplexity** of the corresponding probability vector **p** (Figure 9.6d). The perplexity, a concept from information theory [89] that, in this setting, measures how well a pattern is detected by the layer under consideration.

The perplexity of the distribution $\mathbf{p}$ is equal to 1 if only one filter is activated by the instance of the receptive field. An example is given by the activations marked with ① in Figure 9.6a. On the contrary, the perplexity of $\mathbf{p}$ is equal to the number of filters $|\mathcal{F}^l|$, if the activations of every filter are equal, as shown for the activations marked with ② in Figure 9.6a. The Perplexity Histogram accumulates the sampled input based on the corresponding perplexity value in the range $[1, |\mathcal{F}^l|]$ for every layer l (Figure 9.6e). Changes in the histograms during training are visualized in a second histogram. Here, green bars represent an increase in the corresponding bin, while red bars represent a decrease (Figure 9.6f). A shift to the left in the histogram, i.e., to lower values of perplexity, means that the ability to detect patterns for this layer is increasing and vice-versa. Note that, because the computed perplexity assumes continuous value, the number of bins in the histogram has no link with the number of filters in the layer. We provide a default of 30 bins, that we empirically found to be visually pleasing and does not hamper the ability to detect shifts in the histograms.

Figure 9.6f shows the evolution of the perplexity histograms of the convolutional layers for the MNIST-Network, i.e., *Conv1* and *Conv2*. After 10 iterations a shift to low values of perplexity in the first layer is visible. The peak in the histogram for *Conv1* corresponds to patches that are not detected by any filter **(T3)**. While the histogram of the first layer is shifting to the left, i.e, decreasing the perplexity, the histogram of the second layer is shifting to the right. This behavior shows that the second layer is responding to a change in the filter functions computed in the first layer by becoming less specific, i.e., increasing the resulting perplexity. The histograms are updated at every iteration and the user monitors the stability of the layers. Figure 9.6f shows how the histograms evolved after 80 iterations. Compared to iteration 10, the first layer is still unstable and the second layer is now more specific. After 300 iterations, the first layer is stable, while the second layer shows a shift to lower values of perplexity. This shift is limited, showing that the layer is currently affected by minor changes, allowing the user to start its detailed analysis.

9.4.3 Activation Heatmap

Guided by the Perplexity Histograms, the user focuses on the detailed analysis of a stable layer starting from the Activation Heatmap, where every filter is visualized as a cell in a heatmap visualization (Figure 9.7a). The Activation Heatmap is designed for the quick identification of degenerated filters **(T2)**. We aim at the identification of *dead* filters, i.e., filters that are not activating to any instance of a receptive field, and filters that are activating to all instances. In both cases these filters are not providing any additional information to the network. These filters are detected in a heatmap visualization that shows the maximum- and the frequency-of-activation.

For creating the heatmaps, we randomly sample instances of receptive fields and we compute the maximum activation μ_i^l for every filter f_i^l in layer l

$$\mu_i^l = \max(f_i^l(\pi(\delta_r^l)(\mathbf{x}))),$$

where $\pi(\delta_r^l)(\mathbf{x})$ is the sampled instance of the receptive field. For each filter f_i^l, the corresponding μ_i^l is visualized in the heatmap in the range $[0, \max(\mu_i^l, \forall i)]$. We use a similar approach for the identification of filters that have high activation on every input. For every filter, we keep track of how frequently they activate on the sampled data, and we display these frequencies in the heatmap. We consider a filter to be active on a given patch if its activation is greater than a percentage β of the maximum activation $\max(\mu_i^l, \forall i)$, where a default value of $\beta = 0.5$ is used.

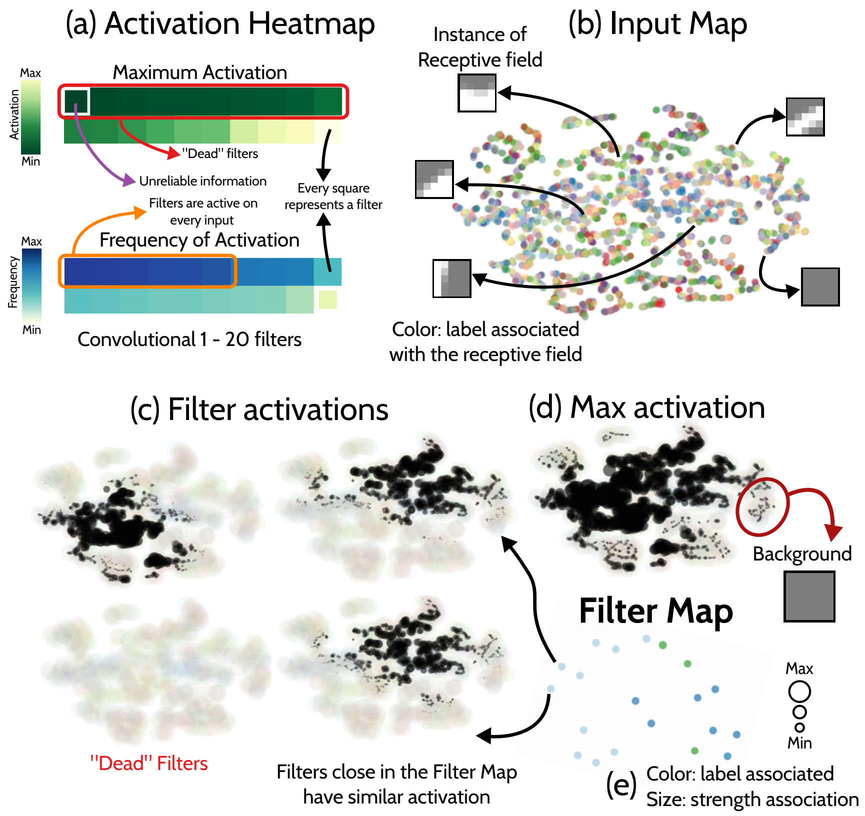

(a) Activation Heatmap
Max
Activation
Min
Maximum Activation
"Dead" filters
Unreliable information
Filters are active on every input
Every square represents a filter
Frequency of Activation
Max
Frequency
Min
Convolutional 1 - 20 filters
Instance of Receptive field
(b) Input Map
Color: label associated with the receptive field
(c) Filter activations
(d) Max activation
Background
Filter Map
Max
Min
"Dead" Filters
Filters close in the Filter Map have similar activation
(e) Color: label associated
Size: strength association

Figure 9.7: **Detailed analysis** performed in DeepEyes. Degenerated filters are detected in the Activation Heatmap (a). The Input Map (b) shows a representation of the input space of a specific layer. By brushing on the Input Map receptive fields are visualized in linked views (insets in (b)). Specific filter activations (c) or the maximum activation of every filter (d) are visualized on the Input Map. The Filter Map (e) allows for the understanding of the relationships between filters that are further investigated in the Input Map. Specific filters are selected by clicking on the activation heatmap or by brushing on the Filter Map.

The user can choose if the maximum- or the frequency-of-activation is visualized in the heatmap and we distinguish between the two by using two different color scales. A green-to-yellow color scale is used for the maximum activation, while a yellow-to-blue color scale is used for the frequency of activation [57]. At this level of detail, we are interested in giving an intuition of the response of the layer as a whole, hence we provide the option to keep the filters sorted according to the currently visualized information. At this level of detail, we are interested in giving an intuition of the response of the layer as a whole, making the identification of uninformative filters easier. Therefore, we provide the option to keep the filters sorted according to the information that is currently visualized. Because the learnable parameters are changing during training, visualizing the maximum activation for a filter may be misleading. For example, a filter that was active in the early phase of training can *"die"* in later steps [94]. Therefore, we compute a measure for the reliability of the information contained in the heatmap. We keep track of the last iteration where a filter f_i^l reached an activation higher than a percentage θ of its maximum activation μ_i^l, where $\theta = 0.8$ by default. We visually encode the distance between the current iteration and the last one that reached the maximum activation threshold θ as the size of the cell that we draw in the heatmap [68] and we allow the reinitialization of the computed maximum in a layer.

An example of the proposed visualization is presented in Figure 9.7a. The maximum activation of the filters in the first convolutional layer of the MNIST-Network after 100 iterations is presented. Ten filters, highlighted in red, out of 20 have a very low activation and do not provide additional information **(T2)**. The smaller size of the cell in the heatmap for the filter identified by a purple arrow means that the maximum activation visualized is not reached in several iterations, leading to the conclusion that at the current iteration its activation is even lower. By visualizing the frequency of activation the user identifies several filters, here highlighted in orange, that have high activation on every input **(T2)**. These insights lead to the conclusion the layer is oversized given the problem at hand **(T4)** and can be removed by the user before continuing the training, making it faster and the final network smaller. Our visual encoding is scalable in the number of visualized filters. One of the layers with most filters in state-of-the-art architectures is the last fully-connected layer in the AlexNet network [88], consisting of 4096 filters. If every filter is encoded, using a 5x5 rectangle, the heatmap results in an image of 320x320 pixels, that easily fits into our user interface.

9.4.4 Input Map

The Input Map is a cornerstone of DeepEyes. It provides the tools to solve several analytical tasks **(T2,T3,T4,T5)** and is based on the idea presented in Figure 9.4. The map is generated upon user's request when a stable layer is identified. An example is given in Figure 9.7b where the first convolutional layer of the MNIST-Network is analyzed in detail. Instances of receptive fields are visualized as points in a scatterplot and colored according to the label of the input they are obtained from. Two instances are close in the scatterplot if they have similar activation for the neurons within the neuronal receptive field and, therefore, are similar input for the current layer (see Section 9.2). The layout is obtained by reducing the dimensionality of the activation of neurons in the neuronal receptive field to 2 dimensions, while preserving neighborhood relationships [136]. By brushing on the scatterplot, the user selects instances of receptive fields of interest that are visualized in a linked view, here abstracted as arrows pointing to image patches. The mix of colors corresponding to the input labels indicates that a separation between the classes is not possible at this level **(T5)**, also showing that a clustering of the neurons based on labels as proposed by Liu et al. [103] is not meaningful for early-layers.

The activation of a user-selected filter is visualized on top of the Input Map, as shown in Figure 9.7c where four filter activations are shown. We keep the Input

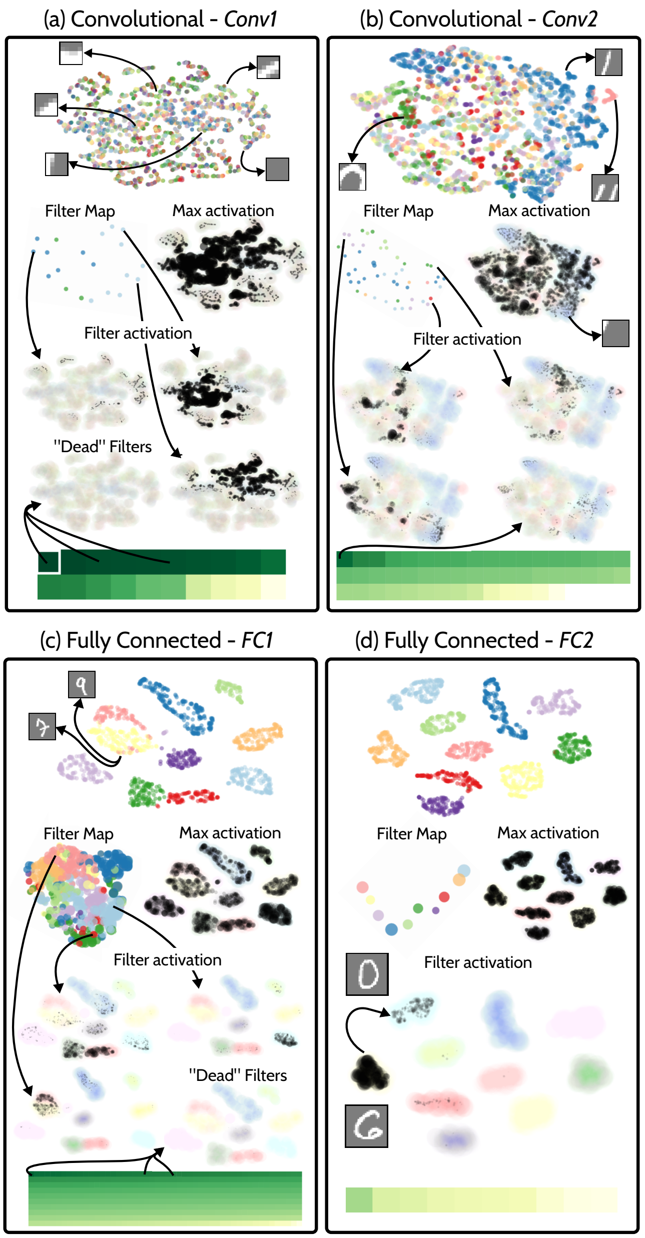


Figure 9.8: **Analysis of the MNIST network**. For each layer the Input- and Filter-Maps are presented alongside their corresponding Activation Heatmaps. We highlight activations for different filters in the different layers. A detailed description of the conclusions, drawn from these visualizations is presented in Section 9.4.6.

Map in the background as a reference, drawing the data points as larger and semi-transparent circles. On top, we draw a new set of semi-transparent black circles, whose size is encoding the intensity of the filter activation on the corresponding input. The user can switch between the two visualization modes, allowing to reason on where the activations are localized in the Input Map, therefore giving a detailed understanding of which input is detected by a filter. For example, we can validate the insights previously obtained through the Activation Heatmap. By clicking on a cell in the heatmap, the corresponding filter activation is visualized in the Input Map, showing that the dead filters are not activating on any input **(T2)**. Moreover, single filters are activating on large portions of the input. Together with the presence of many dead filters, this signals that the current layer contains more filters than needed **(T4)**. By visualizing the maximum activation of the filters on each data point, as presented in Figure 9.7d, we allow for the identification of data that are scarcely or not at all detected by the network. In the example, the outer region of the Input Map contains points that do not produce a strong activation **(T3)**. The inspection of the instances of the corresponding receptive fields reveals that they correspond to background patches and, therefore, are not informative for the problem at hand.

The Input Map is a dataset-centric technique (see Section 9.3), whose improvements over the state-of-the-art are twofold. First, it is built by sampling instances of receptive fields, allowing for the creation of a dataset-centric visualization even if a 1-to-1 correspondence between filters and neurons does not exist, such as for convolutional layers. Second, differently from existing techniques that focus on the activation of the filters in the current layer, the Input Map reduces the dimensionality based on the activations of the filters in the neuronal receptive field rather than the activation of filters in the layer under analysis. This feature allows for the analysis of the relationship between input and output of a layer, an approach that was not possible before. While these two features allow for new insights, they pose computational challenges in the creation of the 2-dimensional layout in the interactive system. Tens of thousands of receptive field instances are sampled during training and ought to be placed in the Input Map. Further, the dimensionality of the feature vector considered is higher than in existing techniques as we do not just consider the activations in the current layer but the whole neuronal receptive field. We considered several dimensionality-reduction techniques for the generation of the scatterplot [178]. The t-distributed Stochastic Neighbor Embedding (tSNE) algorithm is often used [177] in dataset-centric techniques. However, as reported by Rauber et al. [145] for their proposed approach, several dozens of minutes are required for the creation of embeddings containing 70K points described by 50 dimensions, limiting its application in a Progressive Visual Analytics system like DeepEyes. Therefore we use the recently-developed Hierarchical Stochastic Neighbor Embedding (HSNE) [136], as it creates visual representations of tens of thousands of data points, described by several thousand dimensions, in seconds. HSNE enables the analysis of such large data in an interactive system by building a hierarchical representation of the data and by generating Input Maps with only a few hundreds data points sampled from the input data. The exploration of the complete dataset is then performed with a *filter* and *drill-in* paradigm as presented in Chapter 6.

9.4.5 Filter Map

The Filter Map provides a view on how similarly filters in a layer respond to the input as a whole. of oversized layers or the need for more layers **(T4,T5)**. We visualize the filters as points in a scatterplot. Filters with a similar activation pattern are placed closer in the scatterplot (Figure 9.7e). If many filters activate in the same way on the input it is an indications that the layer contains too many filters **(T4)**. Here, we are interested in visualizing the relationships between filters and labels **y**. Hence, points are colored according to the training label that activates a filter the most, while the size of the point shows how strongly the filter is correlated to that label. We choose this encoding for the sake of simplicity, but different visual encodings can be used, e.g., by encoding the correlation with color brightness or saturation [30, 145]. The presence of a cluster composed by large and similarly colored points in the Filter Map is an indication that a classification can be performed at this stage **(T5)**. To the best of our knowledge, the only existing work in this direction is from Rauber et al. [145]. In their work, the Pearson correlation between filter activations is computed and the filters are visualized using a multi-dimensional scaling algorithm. This approach requires the receptive field of the analyzed filters to cover the complete input and it cannot be used for the analysis of convolutional layers, a severe limitation if state-of-the-art architectures ought to be analyzed (see Section 9.2).

We propose to overcome this limitations by computing similarities in a progressive way, using instances of receptive fields instead of the complete input. The similarity between two filters is computed as a weighted Jaccard similarity [77]. This gives a measure of common amount of activation divided by the maximum activation of both filters. If the filters activate equally for the same instances of receptive fields the value will be 1. The more they differ the smaller the similarity will be. For two filters i and j on layer l, their similarity $\phi_{i,j}$ is computed as:

$$\phi_{i,j} = \frac{\sum_{r,\mathbf{x}} \min(f_i^l(\pi(\delta_r^l)(\mathbf{x})), f_j^l(\pi(\delta_r^l)(\mathbf{x})))}{\sum_{r,\mathbf{x}} \max(f_i^l(\pi(\delta_r^l)(\mathbf{x})), f_j^l(\pi(\delta_r^l)(\mathbf{x})))}, \tag{9.1}$$

where $f_i^l(\pi(\delta_z^l)(\mathbf{x}))$ is the activation of the filter f_i^l, given the sampled receptive field for input $\mathbf{x}$. The similarities are updated for every training iteration and, when requested by the user, the filters are embedded in a 2D space with tSNE [177]. In Figure 9.7e, the Filter Map for the first layer of the MNIST-Network is presented. By brushing on the scatterplot the user selects filters whose activation is then visualized in the Input Map. In the example of Figure 9.7, it can be seen that two filters that are close in the Filter Map (e) also have a similar activation pattern on the input (c). We also keep track of which label is most associated with a filter. For each filter

f_i^l, we compute the vector $\mathbf{t}_i^l \in \mathbb{R}^d$, where d is the number of labels in the dataset. It contains the cumulative activation f_i^l on the sampled receptive fields of instances of objects belonging to the same label:

$$\mathbf{t}_i^l(\text{argmax}(\mathbf{y})) = \sum_{r,\mathbf{x}} f_i^l(\pi(\delta_r^l)(\mathbf{x})), \qquad (9.2)$$

where $\mathbf{x}$ is an input with associated label vector $\mathbf{y}$. For every filter f_i^l, the corresponding point in the Filter Map is drawn with the color associated with the label argmax($\mathbf{t}_i^l$). The point size in the Filter Map encodes the strength of the association with a label. This association is computed as the perplexity of the probabilities, obtained by normalizing the vector $\mathbf{t}_i^l$ with L1-norm (see Section 9.4.2). The points size encodes the inverse value of the perplexity, where a low value of perplexity means a strong association with the label. Filters in Figure 9.7e are small in size, showing a low association with the corresponding label, i.e. a large value of perplexity. Also, not all the label colors present in Figure 9.7b are represented in the Filter Map, showing that filters in this layer are not specialized to perform a proper classification.

9.4.6 From insights to network design

Here, we illustrate how insights obtained in DeepEyes support network design decisions. Figure 9.8 shows the analysis of the MNIST-Network introduced in Section 9.4. Driven by the stability of the perplexity histograms, the user is guided to the detailed analysis of layers whose filters are stable. *Conv1* is analyzed first, then *Conv2*, *FC1* and finally *FC2*. In the Input Map of *Conv1*, a separation of the labels with respect to the input is not visible, since all label colors are mixed in the scatterplot (Figure 9.8a). Further, filters are active on large regions of the Input Map, see filter activations in Figure 9.8a for the selected filter in the filter map. Many dead filters are identified **(T2)** by selecting filters with low maximum activation in the Activation Heatmap (Figure 9.8a). The layer is oversized **(T4)** as overly-redundant or non-existent patterns are learnt by the filters. *Conv2* is analyzed next. Here data points in the Input Map start to cluster according to the labels (Figure 9.8b). Notice that the shown instances of the receptive field are larger than for *Conv1*, as *Conv2* processes a larger region of the input images. Differently from the previous layer, filter activations are localized in the Input Map, leading to the conclusion that more filters are needed in *Conv2* than in *Conv1*. Similarly as for Figure 9.7d, points with low maximum activation in Figure 9.8b correspond to background patches **(T3)**.

In *FC1* (Figure 9.8c), inputs cluster in the Input Map according to the associated label. The visualization of the Maximum Activation in Figure 9.8c shows that every data point is activating at least one filter in the current layer, hence every input is identified by the network at this level **(T3)**. Before we can conclude that a classification is feasible at this stage **(T5)**, the Filter Map is analyzed. In the Filter Map, we see that the filters form visual clusters that are associated with labels. However, there is no visible red cluster, associated with the label “digit-5”. The activation of a "digit-5" associated filter is visualized on the Input Map, showing a strong activation also on points in green, i.e., "digit-3". This insight shows that a perfect separation is not possible in this layer, and that the second fully-connected layer is needed **(T5)**. The presence of duplicated filters and dead filters, as in *FC1*, shows that this layer is oversized and fewer filters can be used **(T4)**.

Finally, in the last layer, which performs the prediction (Figure 9.8d), every filter is colored with colors of different labels, showing that a correlation between filter and label exists and the network is correctly classifying the input. By showing the activation of the filters on the Input Map, the user also gets an intuition of which labels are confused by the network, e.g., points that correspond to the "digit-0" and "digit-6", as shown in the filter activation in Figure 9.8d. Based on the insights obtained from DeepEyes, we modified the network reducing the first convolutional layer from 20 to 10 filters, and the first fully-connected layer from 500 to 100. This reduction allows for a smaller network which is faster to be trained and makes predictions without any visible loss in the accuracy of the classification that is stable for both architectures at 98.2% after 2000 iterations. Note that for the sake of reproducibility we used the parameters defined by Caffe in the "lenet_train_test.prototxt" training protocol.

9.5 Test cases

In this section, we provide further examples of analysis performed with DeepEyes. In recent years a great number of different architectures have been presented. For our test cases we decided to focus on widely used architectures derived from AlexNet [88] that are often modified and adapted to solve different problems, a setting in which the insights provided by DeepEyes are greatly needed. AlexNet [88] consists of 5 convolutional layers, with 96-256-384-384-256 filters, and 3 fully-connected layers, with 4096-4096-1000 filters, leading to more than 16 million trainable parameters. Note that AlexNet is among the largest neural networks in terms of computed filter functions, where a trend in reducing the number of filters exists [60, 72]. This analysis demonstrates the scalability of our progressive system in a general setting. In the first test case, we show how DeepEyes allows for a better understanding of the fine-tuning of DNNs, while in the second test case, we show how a better architecture for the medical imaging domain is derived from insights obtained through DeepEyes.

9.5.1 Fine tuning of a deep neural network

Training a large DNN from scratch requires a very large training set, computational power, and time. To overcome this limitation, a common approach is to fine-tune an already trained network for a different pattern recognition problem [15]. The rationale behind this approach is that low-level filters, like color- and edge-detectors, could be reused. To which degree filters can be reused is crucial but not clear a-priori [191]. In this test case, we show how DeepEyes helps in the identification

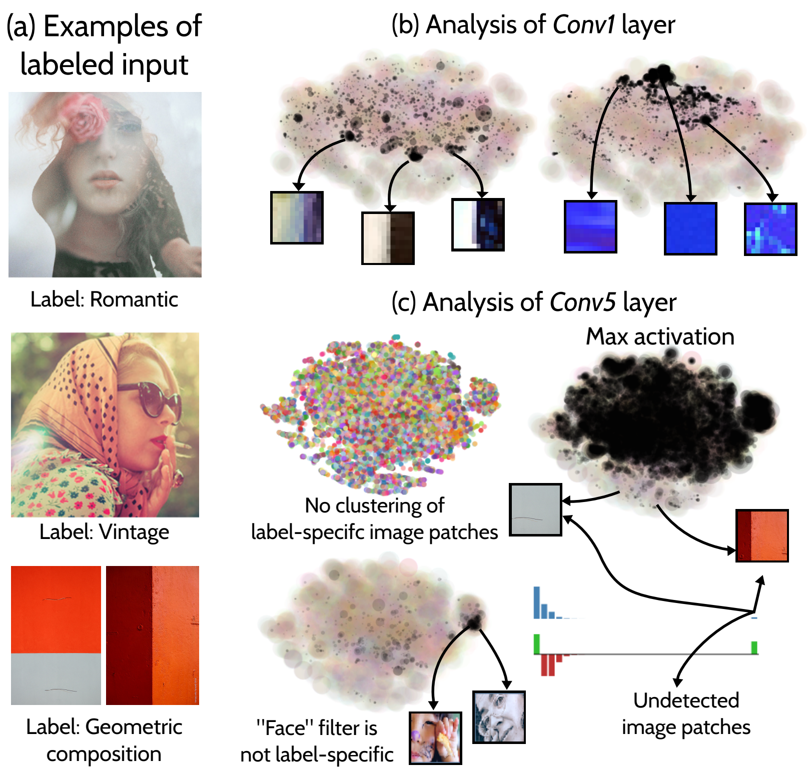

(a) Examples of labeled input
Label: Romantic
Label: Vintage
Label: Geometric composition
(b) Analysis of *Conv1* layer
(c) Analysis of *Conv5* layer
Max activation
No clustering of label-specifc image patches
"Face" filter is not label-specific
Undetected image patches

Figure 9.9: **Fine tuning** of a pretrained neural network. Deep eyes allows for the identification of layers that do not need retraining, e.g. *Conv1*. Unrecognized input data are highlighted in the Perplexity Histograms and in the Maximum Activation visualization of the Input Map, here highlighting data that is labeled as *Geometric Compositions* which are not recognized by the original network. Furthermore, a filter trained to detect faces is not discriminative given the *Romantic* and *Vintage* labels.

of which layers contain useful filters that can be reused and filters that are not needed and must be retrained. We used the fine-tuning example provided in Caffe, where AlexNet, which was trained for image-classification, is retrained for image-style recognition [79]. In this example, the prediction layer of the network is changed from 1000 filters, used to detect 1000 objects, to 20 filters that are retrained to detect 20 styles of images, e.g. "Romantic", "Vintage" or "Geometric Composition" (Figure 9.9a). The network requires 100.000 iterations and more than 7 hours to be retrained with a K40 GPU and achieves an accuracy on the test set of 34.5% [79].

The hypothesis that color and edge detectors are useful filters for the problem at hand is confirmed in the first convolutional layer, i.e., *Conv1* in Figure 9.9b, as they present a localized and consistent activation pattern, e.g., blue- and vertical-edge-detectors are found. While the first layer is stable, the Perplexity Histogram of the fifth convolutional layers, i.e., *Conv5*, shows that an increasingly large number of input patches are not activating any filter, hinting at a problem in the filter functions for this layer. The detailed analysis of *Conv5* shown in Figure 9.9c reveals that data labeled as "Geometric Composition" are in the region of the Input Map that is hardly activating any filters (max activation in Figure 9.9c). Images labeled as "Geometric Composition", i.e., with large and uniform color surface, were not included in the "image-classification" training set, therefore the network has not learnt useful filters for discriminating such images. Another interesting insight is obtained by visualizing the activation of other filters on the Input Map. For example, a filter that detects human faces is found, see Figure 9.9c. While this filter is useful for the image-classification problem, it is not discriminative for style-recognition because human faces are associated with many different styles (Figure 9.9a). This insight shows that the analyzed layer needs to learn new patterns from the input. The fine-tuning of a network or, in general, the reusability of the learned filters, is an active research topic under the name of transfer learning [191]. Insights obtained from DeepEyes can help to improve the fine-tuning of networks by placing the user in the loop.

9.5.2 Mitotic figures detection

We present a different test case from the application of DNNs in the medical imaging domain. In this context, DNNs developed by the machine learning community are applied to different recognition problems. DeepEyes helps in filling the expertise gap, by providing insight on how the network behaves given the problem at hand. The number of nuclei separations in tumor tissue is a measurement for tumor aggressiveness. In radiotherapy treatment planning, histological images of tumor tissue are analyzed by pathologists. Nuclei separations, also known as mitotic figures, are counted. Examples of images with "mitotic figure" label are presented in Figure 9.10a, together with images labeled as "negative". The counting of mitotic figures helps in deciding the dose of radiation used to treat a tumor, leading to a more personalized treatment. However, it is a tedious task and DNNs have been recently proposed to automatize the process. In this test case, we analyze the DNN developed by Veta et al. [182] that is trained on the AMIDA dataset [183] to detect mitotic figures in histological images. The network comprises 4 convolutional layers with 8,16,16 and 32 filters respectively, and 2 fully-connected layers, containing 100 and 2 filters respectively.

After a few training iterations, the first layer stabilizes and is analyzed in detail. Figure 9.10b shows the detailed analysis of the first convolutional layer after 40 iterations. The Input Map shows a cluster of red points, corresponding to instances of the receptive fields sampled from images labeled as mitotic figures. By visualizing the activation of the filters we see that filters are trained to detect dark regions versus bright regions, as they are an important feature at this level. Similar Input Maps are obtained in the other convolutional layers, where the patches processed by the layers are larger.

An interesting observation is made in the first fully-connected layer of the network. The Input Map and the Filter Map for this layer are presented in Figure 9.10c. A separation of the labeled input is visible in the Input Map, showing that the classification is feasible at this level. This is confirmed by the fact that filters are divided in the Filter Map according to the most strongly associated label. Thus, another layer, as is present in the network, is not needed in order to perform a prediction on the problem at hand **(T5)**. Therefore, we change the design by dropping the fully-connected layer and by connecting the prediction layer directly to the last convolutional layer. The analysis of the prediction layer after retraining is provided in Figure 9.10d. The new network reaches an accuracy of 95.9% on the test set, which

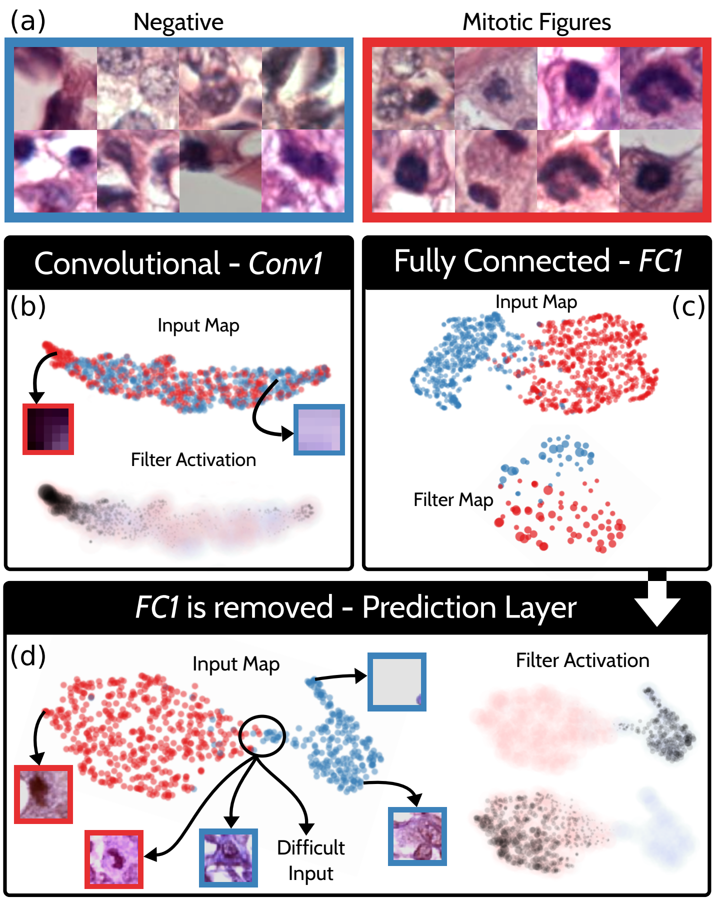


Figure 9.10: **Mitotic Figures** detection. A DNN is trained to detect mitotic figures in histological images (a). Filters in the first convolutional layer *Conv1* are highly associated with mitotic figures (b). Labeled data are separated in the Input Map of the first fully-connected layer *FC1* (c). After removing *FC1* the prediction layer (d) still shows very strong separation, indicating that *FC1* is indeed not needed for classification.

is identical to the accuracy obtained with the previous architecture, while it is much faster to compute a prediction.

We contacted Veta et al. [182], presenting DeepEyes and providing our findings. They informed us that they had come to the same conclusions after several blind modifications of their network, commenting that a system like DeepEyes is beneficial in the definition of networks for a specific medical imaging problem. Furthermore, they showed it particular interest in visualizing the instances of the receptive fields and the corresponding filter activation directly in the system. They also acknowledged that inputs which are difficult to classify are easily identified by the user in the Input Map (Figure 9.10d). Hence, they commented that DeepEyes also gives insights on how the training set can be modified in order to improve the classification as it shows which kind of input must be added to the training set.

9.6 Implementation

DeepEyes is developed to complement Caffe, a widely-used deep-learning library [79]. DeepEyes, requires Caffe files that describe the network and the parameters of the solver as input. DeepEyes trains the network using Caffe, but seamlessly builds the Progressive Visual Analytics system presented in this chapter on top of it.

For optimal performance, we implemented DeepEyes in C++. The interface is implemented with Qt. Perplexity Histograms and the Activation Heatmaps are implemented in JavaScript using D3 and are integrated in the application with QtWebKit Bridge. The Input- and Filter-Maps, are rendered with OpenGL. DeepEyes is implemented using a Model-View-Controller design pattern, allowing for the future extension to different deep-learning libraries, such as Google's TensorFlow [2] or Theano [170].

9.7 Conclusions

In this chapter, we presented DeepEyes, a Progressive Visual Analytics systems that supports the design of DNNs by showing the link between the filters and the patterns they detect directly during training. The user detects stable layers **(T1)** that are analyzed in detail in three tightly-linked visualizations. DeepEyes is the only system we are aware of that supports DNN design decisions during training. Using DeepEyes the user detects degenerated filters **(T2)**, inputs that are not activating any filter in the network **(T3)**, and reasons on the size of a layer **(T4)**. By visualizing the activation of filters and the separation of the input with respect to the labels, the user decides whether more layers are needed given the pattern-recognition problem at hand **(T5)**. We used DeepEyes to analyze three DNNs, demonstrating how the insights obtained from our system help in making decisions about the network design.

A limitation of DeepEyes is that it relies on qualitative color palettes for the visualization of labels in the Input- and Filter-Maps. This solution does not scale when the number of labels is large, therefore we want to address this issue in future work.

Further, the Input- and Filter-Map are created with dimensionality-reduction techniques, which may be affected by projection errors. Hence, adding interactive validation of the projections [109] is an interesting future work. Another interesting future work is the development of linked views that allows for the analysis of different type of data, such as text or video. An interesting development would be to integrate in DeepEyes different deep-learning libraries, such as TensorFlow [2] or Theano [170], and to the analysis of different and other types of network architectures, such as Recurrent Neural Networks [114] and Deep Residual Networks [60]. Finally, it would be interesting to apply DeepEyes for the analysis of DNNs in several application contexts.

10 Conclusions

The known is finite, the unknown infinite;
intellectually we stand on an islet in the midst of an illimitable ocean of inexplicability.
Our business in every generation is to reclaim a little more land.

Thomas Henry Huxley

The effective analysis of high-dimensional data has been proven to be an important driver for new knowledge discovery in many different fields, from life sciencess [6, 12, 102] to artificial intelligence [83, 116, 137]. For a human, direct understanding of high-dimensional data is challenging, since it is far removed from our daily experience. To overcome this limitation, visual analytics techniques and tools combine visualizations, statistical analysis and mining algorithms in order to extract knowledge from data.

In this dissertation we focused on a class of algorithms that, in recent years, achieved impressive results in supporting the hypothesis generation from data in many different fields. Non-linear dimensionality-reduction algorithms, and tSNE [177] in particular, are now widely adopted by biomedical researchers for the exploration of their data, and by machine learning researchers for validation of their models. Among the advantages of these techniques, is the ability to remove redundancy between dimensions by extracting low-dimensional and non-linear structures that are embedded in the high dimensional space.

However, at the time when this work started, non-linear dimensionality techniques were limited in their scalability, limiting the analysis to small datasets. This fact posed a limitation to the applicability of the algorithms to an always increasing dataset size. The work presented in this dissertation aimed specifically at this limitation, proposing new algorithmic solutions that enabled the development of visual analytics applications that are specifically designed for a given domain.

More specifically, in Chapter 4 we introduced the Approximated-tSNE (A-tSNE), which proposes to build dimensionality-reduction embeddings on approximated information. We proposed to adopt approximated k-nearest-neighborhood search algorithms to encode the local similarities between the data points. Thanks to this innovation, a lengthy preprocessing of the data is avoided while, at the same time, high-neighborhood preservation is achieved. Moreover, we provided the user with the ability to locally refine the approximated information while the embedding is computed, hence converging to non-approximated embeddings. Since our work was presented in 2016, almost all the newly introduced non-linear dimensionality-reduction algorithms make use of the approximated computation of the k-nearest-neighborhood graph [25, 87, 112, 168]. This work was further improved by adopting a GPU-based computation of the embedding optimization which is presented in Chapter 5. This technique, which makes use of the rendering pipeline to compute the gradient as a derivation of three scalar fields, is implemented on the GPU and runs in the client side of a web browser and it is openly available in the Google's TensorFlow.js library.

Despite the improved interactivity of systems based on the Approximated-tSNE, we observed that the optimization process becomes increasingly difficult to optimize as the size of the dataset grows. To overcome this limitation, we presented a novel algorithm that is the first to introduce a hierarchical exploration of non-linear data; the Hierarchical Stochastic Neighbor Embedding (HSNE). HSNE, which is presented in Chapter 6, extracts a hierarchy of data points, also known as landmarks, that represent the data at different scales. A hierarchical analysis is then performed by the user. First the embedding representing the overview of the data, i.e., the major high-dimensional structures, is generated. The user can select clusters of landmarks that are of interest, a decision made by looking at some linked views depicting the selection in the data-point space. New embeddings are then generated by using the landmarks associated with the selection at a lower level in the hierarchy. We demonstrated the relevance of HSNE and its applications in the analysis of single cell data where we were able to identify rare cell-types in the immune system that were previously unknown. Other insights obtained by applying the techniques presented in this dissertation to biomedical data are presented in Chapter 7.

We also presented a novel system for the analysis of bipartite graph that is built on top of the HSNE algorithm. Thanks to new data structures we are able to compute HSNE hierarchies of containing tens of million data points. This work is motivated by the growing partisanship that social network creates online, a phenomenon known as Filter-Bubbles or Echo-Chambers. Our WAOW-Vis system allows to analyze this phenomenon on a desktop computer in only few minutes of computation time. HSNE is also the cornerstone used for applying visual analytics for the understanding and improvement of Deep Neural Networks. More specifically we present DeepEyes, a Progressive Visual Analytics system that supports the design of neural networks directly during training. We showed how the system facilitates the identification of problematic design choices and information that is not being captured by the network. We demonstrate the effectiveness of our system through multiple use cases, showing how a trained network can be compressed, reshaped and adapted to different problems.

The work presented in this thesis, provides solutions to the scalability issues of non-linear dimensionality reduction techniques for visual data analysis. Despite the results that we have reported and the increased capabilities of our algorithms, several challenges remain to be tackled. First, the computation of the k-nearest neighbors is a critical step of all the recently developed non-linear dimensionality-reduction algorithms. Faster algorithms for kNN computations will be of great benefit, while a change in the computational paradigm, for example by adopting a differentiable programming approach, may open the door to novel and more efficient techniques. Finally, we believe that the development of progressive data analytics techniques will be of major importance in the near future. As Data Science requires time-consuming and iterative manual activities, the extension of Progressive Visual Analytics would be beneficial if applied in different computational modules, e.g., data cleaning, transformation and modeling. Therefore, the development of a Progressive Data Science [175] pipeline will make the data processing more efficient, as any change of parameters and algorithms will be immediately and progressively reflected to the user, allowing for a quick evaluation of the choices.

References

[1] S. J. Aarseth. *Gravitational N-Body Simulations*. Cambridge University Press, 2003. Cambridge Books Online.

[2] M. Abadi, A. Agarwal, P. Barham, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] W. M. Abdelmoula, N. Pezzotti, T. Holt, J. Dijkstra, A. Vilanova, L. A. McDonnell, and B. P. Lelieveldt. Interactive visual exploration of 3d mass spectrometry imaging data using hierarchical stochastic neighbor embedding reveals spatiomolecular structures at full data resolution. *Journal of proteome research*, 17:1054–1064, 2018.

[4] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2262–2270, 2016.

[5] E. Amid and M. K. Warmuth. A more globally accurate dimensionality reduction method using triplets. *arXiv preprint arXiv:1803.00854*, 2018.

[6] E.-a. D. Amir, K. L. Davis, M. D. Tadmor, E. F. Simonds, J. H. Levine, S. C. Bendall, D. K. Shenfeld, S. Krishnaswamy, G. P. Nolan, and D. Pe'er. viSNE enables visualization of high dimensional single-cell data and reveals phenotypic heterogeneity of leukemia. *Nature biotechnology*, 31(6):545–552, 2013.

[7] M. Ankerst, S. Berchtold, and D. A. Keim. Similarity clustering of dimensions for an enhanced visualization of multidimensional data. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pages 52–60. IEEE, 1998.

[8] F. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27:17–21, 1973.

[9] M. Aupetit. Sanity check for class-coloring-based evaluation of dimension reduction techniques. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 134–141. ACM, 2014.

[10] J. Barnes and P. Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324:446, 1986.

[11] V. Batagelj, F. J. Brandenburg, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani. Visual analysis of large graphs using (x, y)-clustering and hybrid visualizations. *IEEE transactions on visualization and computer graphics*, 17(11):1587–1598, 2011.

[12] B. Becher, A. Schlitzer, J. Chen, F. Mair, H. R. Sumatoh, K. W. W. Teng, D. Low, C. Ruedl, P. Riccardi-Castagnoli, and M. Poidinger. High-dimensional analysis of the murine myeloid cell system. *Nature immunology*, 15(12):1181–1189, 2014.

[13] M. Behrisch, B. Bach, N. Henry Riche, T. Schreck, and J.-D. Fekete. Matrix Reordering Methods for Table and Network Visualization. *Computer Graphics Forum*, 35:24, 2016.

[14] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[15] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *ICML Unsupervised and Transfer Learning*, volume 27, pages 17–36, 2012.

[16] A. Bezerianos, P. Dragicevic, J.-D. Fekete, J. Bae, and B. Watson. GeneaQuilts: A System for Exploring Large Genealogies. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1073–1081, Oct. 2010.

[17] B. Biswal, F. Zerrin Yetkin, V. M. Haughton, and J. S. Hyde. Functional connectivity in the motor cortex of resting human brain using echo-planar mri. *Magnetic resonance in medicine*, 34:537–541, 1995.

[18] I. Borg and P. Groenen. Modern multidimensional scaling: theory and applications. *Journal of Educational Measurement*, 40:277–280, 2003.

[19] I. Borg and P. J. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[20] U. Brandes, M. Gaertler, and D. Wagner. *Experiments on graph clustering algorithms*. Springer, 2003.

[21] M. Brehmer, M. Sedlmair, S. Ingram, and T. Munzner. Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 1–8, 2014.

[22] M. Brehmer, M. Sedlmair, S. Ingram, and T. Munzner. Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 1–8. ACM, 2014.

[23] D. Ceneda, T. Gschwandtner, T. May, S. Miksch, H.-J. Schulz, M. Streit, and C. Tominski. Characterizing guidance in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):111–120, 2017.

[24] J. M. Chambers. *Graphical Methods for Data Analysis: 0*. Chapman and Hall/CRC, 1983.

[25] D. M. Chan, R. Rao, F. Huang, and J. F. Canny. t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. *arXiv preprint arXiv:1807.11824*, 2018.

[26] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[27] J. Choo, H. Kim, C. Lee, and H. Park. PIVE: A per-iteration visualization environment for supporting real-time interactions with computational methods. *Visual Analytics Science and Technology (VAST), 2014 IEEE Symposium on*, 2014.

[28] J. Choo, H. Lee, J. Kihm, and H. Park. iVisClassifier: An interactive visual analytics system for classification based on supervised dimension reduction. pages 27–34. IEEE, 2010.

[29] C. Collins and S. Carpendale. Vislink: Revealing relationships amongst visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 13:1192–1199, 2007.

[30] R. da Silva, P. E. Rauber, R. M. Martins, R. Minghim, and A. C. Telea. Attribute-based visual explanation of multidimensional projections. In *Proceedings of EuroVA (2015)*, pages 134–139.

[31] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[32] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546, 2008.

[33] V. de Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford university, 2004.

[34] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586. ACM, 2011.

[35] M. Dörk, N. H. Riche, G. Ramos, and S. Dumais. Pivotpaths: Strolling through faceted information spaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2709–2718, 2012.

[36] M. Dumas, J.-M. Robert, and M. J. McGuffin. Alertwheel: radial bipartite graph visualization applied to intrusion detection system alerts. *IEEE Network*, 26(6), 2012.

[37] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete. Zame: Interactive large-scale graph visualization. In *Pacific Visualization Symposium (PacificVis)*, pages 215–222. IEEE, 2008.

[38] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009.

[39] S. G. Fadel, F. M. Fatore, F. S. Duarte, and F. V. Paulovich. Loch: A neighborhood-based multidimensional projection technique for high-dimensional sparse spaces. *Neurocomputing*, 150:546–556, 2015.

[40] J.-D. Fekete. Visual analytics infrastructures: From data management to exploration. *Computer*, (7):22–29, 2013.

[41] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[42] D. Fisher, I. Popov, S. Drucker, et al. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1673–1682. ACM, 2012.

[43] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[44] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al. DeViSE: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pages 2121–2129, 2013.

[45] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[46] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21:32–40, 1975.

[47] R. K. Garrett. Echo chambers online?: Politically motivated selective exposure among internet news users. *Journal of Computer-Mediated Communication*, 14:265–285, 2009.

[48] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[49] N. Gehlenborg, S. I. O'donoghue, N. S. Baliga, A. Goesmann, M. A. Hibbs, H. Kitano, O. Kohlbacher, H. Neuweger, R. Schneider, D. Tenenbaum, et al. Visualization of omics data for systems biology. *Nature Methods*, 7:S56, 2010.

[50] C. Geyer. Introduction to markov chain monte carlo. *Handbook of Markov Chain Monte Carlo*, pages 3–48, 2011.

[51] S. Ghani, B. C. Kwon, S. Lee, J. S. Yi, and N. Elmqvist. Visual analytics for multimodal social network analysis: A design study with social scientists. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2032–2041, 2013.

[52] M. Ghoniem, J.-D. Fekete, and P. Castagliola. On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. *Information Visualization*, 4(2):114–135, 2005.

[53] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[54] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[55] S. Hadlak, H. Schumann, and H.-J. Schulz. A survey of multi-faceted graph visualization. In *Eurographics Conference on Visualization (EuroVis).*, pages 1–20, 2015.

[56] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing*, pages 867–877, 2015.

[57] M. Harrower and C. A. Brewer. Colorbrewer. org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[58] J. A. Hartigan. Printer graphics for clustering. In *Journal of Statistical Computing and Simulation*, pages 187–213. 1975.

[59] H. Hauser, F. Ledermann, and H. Doleisch. Angular brushing of extended parallel coordinates. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 127–130. IEEE, 2002.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[61] J. Heinrich, R. Seifert, M. Burch, and D. Weiskopf. Bicluster viewer: a visualization tool for analyzing gene expression data. *Advances in Visual Computing*, pages 641–652, 2011.

[62] J. Heinrich and D. Weiskopf. State of the art of parallel coordinates. In *Eurographics (STARs)*, pages 95–116, 2013.

[63] N. Henry and J.-D. Fekete. Matrixexplorer: a dual-representation system to explore social networks. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):677–684, 2006.

[64] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE transactions on visualization and computer graphics*, 13(6):1302–1309, 2007.

[65] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[66] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 833–840, 2002.

[67] R. Hodge et al. Conserved cell types with divergent features between human and mouse cortex. *bioRxiv*, 2018.

[68] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive immune cell phenotyping for large single-cell datasets. In *Computer Graphics Forum*, volume 35, pages 171–180, 2016.

[69] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, B. P. Lelieveldt, and A. Vilanova. Cyteguide: Visual guidance for hierarchical single-cell analysis. *IEEE Transactions on Visualization and Computer Graphics*, 24, 2017.

[70] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[71] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. In *Computer Graphics Forum*, volume 28, pages 983–990, 2009.

[72] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[73] S. M. Huisman, B. van Lew, A. Mahfouz, N. Pezzotti, T. Höllt, L. Michielsen, A. Vilanova, M. J. Reinders, and B. P. Lelieveldt. Brainscope: interactive visual exploration of the spatial and temporal human brain transcriptome. *Nucleic acids research*, 45(10):e83–e83, 2017.

[74] S. Ingram and T. Munzner. Dimensionality reduction for documents with nearest neighbor queries. *Neurocomputing*, 150:557–569, 2015.

[75] S. Ingram, T. Munzner, and M. Olano. Glimmer: Multilevel mds on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 15(2):249–261, 2009.

[76] A. Inselberg and B. Dimsdale. Parallel coordinates. In *Human-Machine Interactive Systems*, pages 199–233. Springer, 1991.

[77] S. Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *2010 IEEE International Conference on Data Mining*, pages 246–255. IEEE, 2010.

[78] Y. Jia, J. Hoberock, M. Garland, and J. Hart. On the visualization of social and other scale-free networks. *IEEE transactions on visualization and computer graphics*, 14(6):1285–1292, 2008.

[79] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[80] P. Joia, F. Paulovich, D. Coimbra, J. Cuminato, and L. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[81] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[82] A. Joulin, L. van der Maaten, A. Jabri, and N. Vasilache. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision*, pages 67–84. Springer, 2016.

[83] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau. A cti v is: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.

[84] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[85] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. In *Information visualization*, pages 154–175. Springer, 2008.

[86] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466, 1952.

[87] M. Kim, M. Choi, S. Lee, J. Tang, H. Park, and J. Choo. Pixelsne: Visualizing fast with just enough precision via pixel-aligned stochastic neighbor embedding. *arXiv preprint arXiv:1611.02568*, 2016.

[88] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[89] S. Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[90] S. Laban, J. S. Suwandi, V. van Unen, J. Pool, J. Wesselius, T. Höllt, N. Pezzotti, A. Vilanova, B. P. Lelieveldt, and B. O. Roep. Heterogeneity of circulating cd8 t-cells specific to islet, neo-antigen and virus in patients with type 1 diabetes mellitus. *PloS one*, 13(8):e0200818, 2018.

[91] S. S. Lafon. *Diffusion maps and geometric harmonics*. PhD thesis, Yale University PhD dissertation, 2004.

[92] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pages 171–178, 2011.

[93] D. Le Bihan, E. Breton, D. Lallemand, P. Grenier, E. Cabanis, and M. Laval-Jeantet. Mr imaging of intravoxel incoherent motions: application to diffusion and perfusion in neurologic disorders. *Radiology*, 161:401–407, 1986.

[94] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[95] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[96] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–97. IEEE, 2004.

[97] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[98] B. Lee, C. Plaisant, C. Sims, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *BELIV '06: Proceedings of the 2006 AVI workshop on BEyond time and errors*, pages 1–5, Venezia, Italy, May 2006. ACM, ACM.

[99] E. S. Lein, M. J. Hawrylycz, N. Ao, M. Ayres, A. Bensinger, A. Bernard, A. F. Boe, M. S. Boguski, K. S. Brockway, E. J. Byrnes, et al. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124):168–176, 2007.

[100] D. Lemire, G. Ssi-Yan-Kai, and O. Kaser. Consistently faster and smaller compressed bitmaps with roaring. *Software: Practice and Experience*, 46(11):1547–1569, 2016.

[101] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2006.

[102] N. Li, V. van Unen, T. Höllt, A. Thompson, J. van Bergen, N. Pezzotti, E. Eisemann, A. Vilanova, S. M. Chuva de Sousa Lopes, B. P. Lelieveldt, and F. Koning. Mass cytometry reveals innate lymphoid cell differentiation pathways in the human fetal intestine. *Journal of Experimental Medicine*, 2018.

[103] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks . *Transactions on Visualization and Computer Graphics*, 2017.

[104] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20:2122–2131, 2014.

[105] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 1(1):24–45, 2004.

[106] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

[107] A. Mahfouz, M. van de Giessen, L. van der Maaten, S. Huisman, M. Reinders, M. J. Hawrylycz, and B. P. Lelieveldt. Visualizing the spatial gene expression organization in the brain through non-linear similarity embeddings. *Methods*, 73:79 – 89, 2015.

[108] R. M. Martins, G. F. Andery, H. Heberle, F. V. Paulovich, A. de Andrade Lopes, H. Pedrini, and R. Minghim. Multidimensional projections for visual analysis of social networks. *Journal of Computer Science and Technology*, 27:791–810, 2012.

[109] R. M. Martins, D. B. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[110] M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[111] L. A. McDonnell and R. M. Heeren. Imaging mass spectrometry. *Mass spectrometry reviews*, 26:606–643, 2007.

[112] L. McInnes and J. Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[113] H. Mifflin. The american heritage dictionary of the english language. *New York*, 2000.

[114] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[115] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu. Understanding hidden memories of recurrent neural networks. *arXiv preprint arXiv:1710.10777*, 2017.

[116] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[117] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[118] A. Mordvintsev and M. Tyka. Deepdream - a code example for visualizing neural networks, 2015.

[119] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1643–1652, 2014.

[120] M. Muja and D. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[121] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application*, pages 331–340, 2009.

[122] J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future generation computer systems*, 15:119–129, 1999.

[123] T. Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.

[124] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pages 3387–3395, 2016.

[125] A. Oghabian, S. Kilpinen, S. Hautaniemi, and E. Czeizler. Biclustering methods: biological relevance and application in gene expression analysis. *PloS one*, 9(3):e90801, 2014.

[126] O. I. Ornatsky, R. Kinach, D. R. Bandura, X. Lou, S. D. Tanner, V. I. Baranov, M. Nitz, and M. A. Winnik. Development of analytical methods for multiplex bio-assay with inductively coupled plasma mass spectrometry. *Journal of analytical atomic spectrometry*, 23:463–469, 2008.

[127] F. V. Paulovich, D. M. Eler, J. Poco, C. P. Botha, R. Minghim, and L. G. Nonato. Piece wise laplacian-based projection for interactive data exploration and organization. In *Computer Graphics Forum*, volume 30, pages 1091–1100. Wiley Online Library, 2011.

[128] F. V. Paulovich and R. Minghim. Hipp: A novel hierarchical point placement strategy and its application to the exploration of document collections. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1229–1236, 2008.

[129] F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):564–575, 2008.

[130] F. V. Paulovich, M. C. F. Oliveira, and R. Minghim. The projection explorer: A flexible tool for projection-based multidimensional visualization. In *Computer Graphics and Image Processing, 2007. SIBGRAPI 2007. XX Brazilian Symposium on*, pages 27–36. IEEE, 2007.

[131] F. V. Paulovich, C. T. Silva, and L. G. Nonato. Two-phase mapping for projecting massive data sets. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1281–1290, 2010.

[132] E. Pekalska, D. de Ridder, R. P. Duin, and M. A. Kraaijveld. A new method of generalizing sammon mapping with application to algorithm speed-up. In *ASCI*, volume 99, pages 221–228, 1999.

[133] W. Peng, M. O. Ward, and E. A. Rundensteiner. Clutter reduction in multi-dimensional data visualization using dimension reordering. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 89–96. IEEE, 2004.

[134] N. Pezzotti. High dimensional inspector, 2017.

[135] N. Pezzotti, J.-D. Fekete, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Multiscale visualization and exploration of large bipartite graphs. In *Computer Graphics Forum*, volume 37, page 12, 2018.

[136] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. In *Computer Graphics Forum*, volume 35, pages 21–30, 2016.

[137] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Deepeyes: Progressive visual analytics for designing deep neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):98–108, 2018.

[138] N. Pezzotti, B. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 23(7):1739–1752, 2017.

[139] N. Pezzotti, A. Mordvintsev, T. Höllt, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. Linear tsne optimization for the web. *arXiv preprint arXiv:1805.10817*, 2018.

[140] J. C. Platt. Using analytic qp and sparseness to speed training of support vector machines. In *Advances in neural information processing systems*, pages 557–563, 1999.

[141] B. Pontes, R. Giráldez, and J. S. Aguilar-Ruiz. Biclustering on expression data: A review. *Journal of biomedical informatics*, 57:163–180, 2015.

[142] P. Qiu, E. F. Simonds, S. C. Bendall, K. D. Gibbs Jr, R. V. Bruggner, M. D. Linderman, K. Sachs, G. P. Nolan, and S. K. Plevritis. Extracting a cellular hierarchy from high-dimensional cytometry data with spade. *Nature biotechnology*, 29:886, 2011.

[143] R. G. Raidou, M. Eisemann, M. Breeuwer, E. Eisemann, and A. Vilanova. Orientation-enhanced parallel coordinate plots. *IEEE transactions on visualization and computer graphics*, 22:589–598, 2016.

[144] R. G. Raidou, U. A. van der Heide, C. V. Dinh, G. Ghobadi, J. F. Kallehauge, M. Breeuwer, and A. Vilanova. Visual analytics for the exploration of tumor tissue characterization. In *Computer Graphics Forum*, volume 34, pages 11–20, 2015.

[145] P. E. Rauber, S. Fadel, A. Falcao, and A. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2017.

[146] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on software Engineering*, 2:223–228, 1981.

[147] A. Reiss and D. Stricker. Creating and benchmarking a new dataset for physical activity monitoring. In *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, page 40. ACM, 2012.

[148] A. H. Robinson and C. Cherry. Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, 55(3):356–364, 1967.

[149] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

[150] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[151] D. Sacha, L. Zhang, M. Sedlmair, J. A. Lee, J. Peltonen, D. Weiskopf, S. C. North, and D. A. Keim. Visual interaction with dimensionality reduction: A structured literature analysis. *IEEE transactions on visualization and computer graphics*, 23(1):241–250, 2017.

[152] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 5:401–409, 1969.

[153] M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig. Non-linear pca: a missing data approach. *Bioinformatics*, 21:3887–3895, 2005.

[154] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

[155] M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2634–2643, 2013.

[156] F. Sha and L. K. Saul. Large margin gaussian mixture modeling for phonetic classification and recognition. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 1, pages I–I. IEEE, 2006.

[157] K. Shekhar, P. Brodin, M. M. Davis, and A. K. Chakraborty. Automatic classification of cellular expression by nonlinear stochastic embedding (ACCENSE). *Proceedings of the National Academy of Sciences*, 111:202–207, 2014.

[158] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, pages 336–343. IEEE, 1996.

[159] H. Siirtola and K.-J. Räihä. Interacting with parallel coordinates. *Interacting with Computers*, 18:1278–1309, 2006.

[160] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[161] V. D. Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in neural information processing systems*, pages 705–712, 2002.

[162] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[163] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[164] J. Stasko, C. Görg, and Z. Liu. Jigsaw: supporting investigative analysis through interactive visualization. *Information visualization*, 7(2):118–132, 2008.

[165] C. Stolper, A. Perer, and D. Gotz. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1653–1662, 2014.

[166] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[167] R. Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007.

[168] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297, 2016.

[169] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[170] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[171] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann. A survey on interactive lenses in visualization. *EuroVis State-of-the-Art Reports*, pages 43–62, 2014.

[172] J. W. Tukey. The future of data analysis. *The Annals of Mathematical Statistics*, pages 1–67, 1962.

[173] P. Tukey and J. Tukey. Graphic display of data sets in 3 or more dimensions. *The collected works of John Tukey*, 5:189–288, 1988.

[174] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):131–140, 2017.

[175] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J.-D. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive data science: Potential and challenges. *arXiv preprint arXiv:1812.08032*, 2018.

[176] L. Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[177] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[178] L. J. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10(1-41):66–71, 2009.

[179] V. van Unen, T. Hollt, N. Pezzotti, N. Li, M. J. T. Reinders, E. Eisemann, A. Vilanova, F. Koning, and B. P. F. Lelieveldt. Interactive visual analysis of mass cytometry data by hierarchical stochastic neighbor embedding reveals rare cell types. *Nature Communications*, 8, 2017.

[180] V. van Unen, N. Li, I. Molendijk, M. Temurhan, T. Höllt, A. E. van der Meulen-de, H. W. Verspaget, M. L. Mearin, C. J. Mulder, J. van Bergen, et al. Mass cytometry of the human mucosal immune system identifies tissue-and disease-associated immune subsets. *Immunity*, 44:1227–1239, 2016.

[181] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *The Journal of Machine Learning Research*, 11:451–490, 2010.

[182] M. Veta, P. J. van Diest, M. Jiwa, S. Al-Janabi, and J. P. Pluim. Mitosis counting in breast cancer: Object-level interobserver agreement and comparison to an automatic method. *PloS one*, 11(8):e0161286, 2016.

[183] M. Veta, P. J. Van Diest, S. M. Willems, H. Wang, A. Madabhushi, A. Cruz-Roa, F. Gonzalez, A. B. Larsen, J. S. Vestergaard, A. B. Dahl, et al. Assessment of algorithms for mitosis detection in breast cancer histopathology images. *Medical image analysis*, 20(1):237–248, 2015.

[184] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer graphics forum*, volume 30, pages 1719–1749. Wiley Online Library, 2011.

[185] T. Von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: state-of-the-art and future research challenges. In *Computer Graphics Forum*, volume 30, pages 1719–1749, 2011.

[186] M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *IEEE Symposium on Information Visualization*, pages 57–64. IEEE, 2004.

[187] P. Xu, N. Cao, H. Qu, and J. Stasko. Interactive visual co-cluster analysis of bipartite graphs. In *Pacific Visualization Symposium (PacificVis)*, pages 32–39. IEEE, 2016.

[188] T. Yang, J. Liu, L. McMillan, and W. Wang. A fast approximation to multidimensional scaling. In *Proceedings of the ECCV Workshop on Computation Intensive Methods for Computer Vision (CIMCV)*, pages 354–359, 2006.

[189] Z. Yang, J. Peltonen, and S. Kaski. Scalable optimization of neighbor embedding for visualization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 127–135, 2013.

[190] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321. Society for Industrial and Applied Mathematics, 1993.

[191] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[192] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[193] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[194] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[195] H. Zhou, P. Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.

Curriculum Vitae

Nicola Pezzotti was born on June 17th, 1986 in Iseo, Italy. Inspired by his father's work, Nicola started programming at the age of 8, and worked as a freelance developer from the age of 13. In 2005 he graduated at ITIS Castelli high school and, in the same year, he won the silver medal in the Italian selection of the *International Olympiad in Informatics* (IOI). Nicola received the Laurea Triennale in Ingegneria dell'Informazione (BSc) from the University of Brescia, Italy, in 2009. For his bachelor thesis titled *"Development of an interprocess communication library between real-time applications working in Linux-Xenomai and other Linux applications"*, he interned with the company G2L srl. He received the Laurea Magistrale in Ingegneria Informatica (MSc) with highest honor from the University of Brescia, Italy, in 2011. For his master thesis titled *"Development and optimization of computing pipeline for the geometric alignment of 3D data structures"*, he interned with the company Open Technologies srl.

In 2011, he joined the University of Brescia as a Research Fellow, where he focused on the development of GPGPU technologies for real-time 3D surface acquisition. In 2011, he also joined the company Open Technologies srl as a Research and Development Engineer, where he developed the high-end real-time scanner *Insight3* and contributed to the development of the company's computational-geometry library. In 2014, Nicola relocated to The Netherlands to pursue a PhD in Visual Analytics in the Computer Graphics & Visualization group at the Delft University of Technology, in close collaboration with the Division on Image Processing at the Leiden University Medical Center.

During his PhD, Nicola focused on the development of Machine Learning techniques, in particular dimensionality reduction, for Visual Analytics. Nicola's work featured in several international conferences and journals, including *Nature Communications*. In 2018, Nicola featured in *"Portraits of Science"*, the Delft University of Technology's showcase of research excellence. In 2017, he visited for three months the AVIZ group at INRIA Saclay (Paris), working on the novel Progressive Visual Analytics computational paradigm. In 2018, Nicola joined for four months Google AI in Zürich as a Research Intern, where he published two research papers on the interpretation and generative capabilities of deep neural networks and developed an open-source library for TensorFlow.js. His work was rewarded among the top research efforts of Google AI in 2018. In 2018, Nicola joined Philips Research in Eindhoven where, as a Research Scientist, he develops human-centered Artificial Intelligence techniques for healthcare.

Acknowledgements

What a journey! Luckily, it wasn’t a lonely one.

It is good, at the end of this jolly adventure that my PhD had been, to stop and thank all the people that walked with me along the road. First and foremost, I would like to thank my supervisor and promotor Anna Vilanova. Thank you for the inspiration, the guidance and the support during these four years. You are an excellent supervisor, not only because of your deep technical knowledge, enthusiasm and cheerful and positive attitude, but also because you truly care about the well being of your PhDs. Thank you for the trust and the freedom that you gave me in pursuing my research goals.

Thanks to my promotors, Boudewijn Lelieveldt and Elmar Eisemann. Boudewijn, your enthusiasm had been a never ending fuel for my research. You immediately realized that I strive for facing difficult problems, and you always provided me with new and interesting challenges. Thank you for the constant effort in sharing and promoting my work, I am deeply grateful for all the doors that you opened for me. Elmar, you are a great researcher and communicator; thank you for being so involved in my research and the great support you provided.

Thanks to all the collaborators of the VAnPIRe project, I has been a pleasure participating in a multidisciplinary team and I firmly believe that this is the true key of our success. Thank you Thomas, you have been more than a supervisor, but a true example of hard work, selflessness and, foremost, integrity. Baldur, thank you for all the technical discussions and the personal support that you gave me during these years. Julian, I am really happy you joined the team, albeit late, I really enjoyed working and discussing together. I would like to express my gratitude also to the institutions and companies that financed my PhD project, STW, Synerscope, Medis, Treparel and to Jorik and Stef, who were very interested and supportive during my research.

Research can thrive only if pursued in a stimulating and positive environment. I am really grateful to the members of the Computer Graphics & Visualization and Pattern Recognition groups at the Delft University of Technology for sharing the journey with me. Thank you Timothy, Jingtang, Changgong, Renata, Noeska, Jean Marc, Bert, Michael, Pablo, Klaus, Rafael, Markus, Christopher, Jerry, Niels, Xuejiao, Ahmad, Victor, Nestor, Leonardo, Peiteng, Baran, Annemieke, Ruud, Bart, Marloes, Helwig, Chaoran, Alessio, Jan and Laurens. I want to dedicate a special thought to

Renata, with your smile and constant positive thinking you helped me overcoming the obstacles on the path. At Leiden University Medical Center, I had the pleasure to work with very talented people, who had the patience of introducing me to biomedical research. Thank you Ahmed, Sjoerd, Thomas, Oleh, Walid, Vincent, Frits, Na and Marcel.

During my PhD I had the opportunity to visit research groups in publicly-funded institutions and private companies. Thank you Jean-Daniel for inviting me in your group at INRIA Saclay, I am still impressed by your profound knowledge and never-ending desire to implement some good code! Thank you to all the other members of the group that welcomed me; Pierre, Tobias, Petra, Lonni, Christoph, Catherine, Paola and Evanthia. Thanks to the amazing people that I had the pleasure to meet at Google AI. In particular, I would like to thank Alexander for collaborating on very interesting projects. Thank you Matt for welcoming in the "Sense" team, to all its members and, in particular, to Marvin for having been an amazing mentor. It has been a pleasure to visit and work with the PAIR team in the Cambridge office; Fernanda, Martin, Yannick, Nikhil and Daniel, and with the Google Brain team in Mountain View; Chris and Ludwig.

At the end of my PhD's journey, a new one has begun for me at Philips Research. Thank you Milan and Richard for giving me the chance to work, together with a talented team, in a very challenging project. It is great to be part of the Data Science team, and I would like to thank all its members for accepting me in this great family. Thanks to all the people that helped and welcomed me during these first months; Dimitrios, Jacek, Maurice, Elwin, Christophe and Joost. I am really looking forward to all the great achievements that we will accomplish!

Thanks to my family.
Dad, you have been a true example. Your work ethic is a compass for me, as it is the passion you put in your hobbies as well as in your work. *"Do what you like and like what you do"* they say; you practice it every day and you taught me to do the same. Mum, how tough was it when you became sick right after I moved to the Netherlands! It had been really difficult, but you went through it and defeated the disease with stubbornness and a smile, as you always do. You taught me humility and that important things in life are the relationships with the loved ones. Luca, I am so happy that you finally found your way; we are all looking forward to what you will achieve. And thanks to all my Italian friends, you are too many to name! It is so great to find you all when I visit, the distance only strengthened the bond, I am so grateful for that.

Finally, thank you Marijke, I am honored that you have chosen me; all this would not have been possible without you.

“There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after.”
J.R.R. Tolkien, The Hobbit or There and Back Again